UNIVERSITA' DEGLI STUDI DI PISA

Facoltà di Scienze Matematiche, Fisiche e Naturali

CORSO DI LAUREA IN INFORMATICA

UNA STRATEGIA DI TEST PER LA DERIVAZIONE SELETTIVA DI ISTANZE A PARTIRE DA XML SCHEMA

Candidati:

Romina Paradisi

Maurizio Santoni

Tutore Aziendale:

Dott. *Eda Marchetti*

Tutore Accademico:

Prof. *Chiara Bodei*

Anno Accademico 2005-2006

Alla mia famiglia,
per essere stata un saldo punto di riferimento
in questo cammino scolastico e nella mia vita.

INDICE

INDICE DELLE FIGURE

INTRODUZIONE

Il lavoro che sarà presentato è stato svolto presso il Consiglio Nazionale delle Ricerche di Pisa e si inserisce all'interno di un progetto a livello europeo basato su e-learning chiamato TELCERT (Technology Enhanced Learning Conformance - European Requirements and Testing) [49]. Il compito di TELCERT è quello di sviluppare test di conformità su applicazioni in ambito e-learning mediante l'uso di nuove tecnologie di sviluppo come Java, UML e XML, mentre tra i suoi obiettivi vi è la ricerca di metodologie volte a garantire l'interoperabilità tra sistemi diversi che si scambiano dati conformi a specifici XML Schema, al fine di facilitarne la comunicazione e l'interazione [1]. Una delle idee emergenti di TELCERT è stata, in particolare, l'utilizzo di istanze XML, formattate secondo le regole dello schema XML, per convalidare e testare il corretto funzionamento delle applicazioni. Il tool TAXI (Testing by Automatically generated XML Instances) è il tool che è in via di sviluppo e che tenta di coprire questo obiettivo di TELCERT.

In letteratura esistono molti strumenti software per la manipolazione di schemi XML, tra i quali XSV (XML Schema Validator) [9], XMLSpy [10], che spesso hanno la sola funzione di parser per il controllo della sintassi e della struttura di un documento XML. Altri tool invece [50], [51] forniscono la possibilità di generare istanze a partire da XML Schema in altre notazioni, come Java o C++; sono tuttavia molto rari quelli che consentono anche la derivazione automatica di istanze. Infatti, di solito, questi generatori non producono in output istanze secondo un criterio rigoroso, ma ricorrono a tecniche random.

Lo scopo dello stage svolto è stato quello di proseguire lo sviluppo del tool TAXI, progettandone l'interfaccia grafica e fornendo l'implementazione di una strategia per la derivazione selettiva di istanze a partire da XML Schema, per il testing *balck-box* di applicazioni.

Ciò è stato possibile consentendo all'utente di assegnare un valore, tramite l'interfaccia, a dei particolari nodi appartenenti allo Schema. Tale valore, detto *peso*, denota l'importanza che determinati nodi devono avere all'interno del processo di generazione delle istanze. Grazie a ciò sarà possibile pilotare la derivazione automatizzata, forzandola a derivare i nodi aventi peso maggiore e quindi maggior importanza ai fini del processo, in relazione alle esigenze dell'utente.

La possibilità di generare istanze automaticamente secondo un criterio, in modo coerente e completo, anziché in maniera casuale, renderà questo tool per il testing sistematico di tutti gli aspetti descritti nello Schema, più flessibile rispetto ai tool già esistenti.

Struttura della relazione

- Nel capitolo 1 è stata fatta una panoramica concernente il funzionamento globale del tool TAXI descrivendo in generale le sue componenti.
- Nel capitolo 2 è illustrata in maniera molto dettagliata la componente del tool da noi sviluppata.
- Nel capitolo 3 si propone una panoramica generale sull'importanza che il nostro lavoro assume nel funzionamento generale del tool e le modifiche, gli aggiornamenti e le estensioni apportati a componenti già esistenti.
- Nel capitolo 4 sono descritte le fasi precedenti allo sviluppo del software, che prevedono la scelta del linguaggio di programmazione e dell'ambiente di sviluppo, le librerie utilizzate etc.
- Nel capitolo 5 descriviamo in dettaglio ed in termini tecnici la fase riguardante la costruzione dell'interfaccia.
- Nel capitolo 6 viene descritto il procedimento di sviluppo della componente Test Strategy Selector del tool TAXI.
- Nel capitolo 7 descriviamo la costruzione delle interfacce per la componente Values Provider e del notepad per la visualizzazione delle istanze finali.
- Nel capitolo 8 vengono esposte le problematiche relative alla fase di testing e di integrazione del lavoro prodotto col software preesistente.
- In appendice è riportata la stampa completa del codice prodotto.

Capitolo 1

IL TOOL TAXI

Il tool TAXI (Testing by Automatically generated XML Instances) [1] è uno strumento che consente la generazione automatica di istanze XML valide a partire da schemi XML. Queste sono usate come casi di test per verificare il corretto funzionamento e l'interoperabilità tra sistemi che si scambiano dati XML conformi ad un preciso XML Schema. Questo tool nasce nell'ambito del progetto europeo Technology Enhanced Learning: Conformance – European Requirements & Testing (TELCERT) il cui scopo è quello di riuscire ad introdurre nuove tecnologie all'interno di tools e sistemi che costituiscono la base di programmi di conformità e certificazione per standards e specifiche [2].

I sistemi *e-learning* sono sviluppati seguendo un'architettura modulare, nella quale le varie componenti o moduli vengono implementate separatamente, attuando quindi una vera e propria *decomponentizzazione*. Per garantire ciò TAXI [1] è stato sviluppato tenendo in considerazione il linguaggio XML. Quest' ultimo si è, infatti, affermato in questi ultimi anni, come il formato standard per lo scambio di dati e documenti tra applicazioni digitali e web [12]. In particolare XML Schema descrive formalmente quale struttura devono avere i documenti XML per essere considerati validi ed utilizzabili nel contesto di una specifica applicazione [48].

TAXI sfrutta al suo interno il grande potenziale di XML (in particolare XML Schema) per valutare la corretta comunicazione dei sistemi interagenti. L'idea originale di questo tool in particolare è stata quella di applicare agli XML Schema il metodo Partition Testing [1] che è una tecnica largamente studiata e utilizzata nell'ambito del testing per derivare sistematicamente un insieme di istanze XML. Il Partition Testing si adatta molto bene alla struttura degli Schemi XML poiché forniscono un'accurata rappresentazione del dominio di input in un formato adatto al processing automatico. La suddivisione del dominio dell'input in sottodomini (unità funzionali), può essere fatta in maniera automatizzata analizzando gli elementi di un XML Schema ed in particolare il costrutto `<choice>`. A partire dai vari sottodomini identificati, Partition Testing porta a termine la derivazione sistematica di un insieme di istanze XML.

Partition Testing basato su XML [19] usa le tecniche classiche relative al testing effettuato tramite partizioni [4], per identificare le configurazioni relative dei dati di input e permette la realizzazione di una strategia automatizzata e controllabile per la generazione di casi di test basati su XML, fornendo i criteri per selezionare i valori degli input associati a ogni classe. In generale l'input è un file XML che viene opportunamente manipolato in maniera tale da fornire in output delle istanze XML valide (casi di test) [3].

Il tool TAXI, grazie a queste sue caratteristiche innovative nel campo del testing automatico, rimane unico nel suo genere.

In generale il tema di testing basato su XML è un argomento molto discusso e trattato in letteratura, ma con fini molto diversi rispetto a quelli del tool TAXI.

Gli approcci esistenti possono, infatti, essere semplicemente classificati in tre gruppi [5]:

- Verifica delle istanze XML stesse;
- Verifica degli XML Schema stessi;
- Verifica di correttezza delle istanze XML rispetto ad uno Schema XML di riferimento.

Per quanto riguarda il primo gruppo, la verifica di un'istanza XML è generalmente chiamato well-formedness (buona forma). Questa è mirata in altre parole a controllare che la struttura del file e i suoi elementi possiedano le caratteristiche necessarie affinché l'istanza sia classificata come un file XML [5]. Questi insiemi di test suites, come [6] e [7], sono stati implementati, facendo uso di determinate metriche, per determinare la conformità del file XML alle regole del World Wide Web Consortium (W3C), il cui scopo è quello di creare standard Web per condurlo al suo massimo potenziale, mediante lo sviluppo di tecnologie (specifiche, linee guida, software e tools) che possano creare un forum per informazioni, commercio, ispirazioni, pensiero indipendente e comprensione collettiva [53].

Ci sono anche diversi tool il cui scopo è quello di validare l'adeguatezza di un documento relativamente ad un insieme prestabilito di regole, come [13], cosa che facilita il testing di componenti e servizi web.

In riferimento al secondo gruppo per la verifica degli XML Schema, esistono diversi validatori che si occupano di analizzare la sintassi e la struttura di un documento XML Schema [5], come per esempio SQC (Schema Quality Checker) [6], XSV (XML Schema Validator) [9], XMLSpy [10], [14] e [15]. Importante anche ricordare [16] in cui viene presentato un approccio che individua errori semantici all'interno di XML Schema tramite un'*analisi delle mutazioni*.

TAXI in questo contesto si occupa della generazione automatica di istanze a partire da uno schema XML ed appartiene quindi al terzo gruppo, a cui appartengono anche Toxgen [17] e SunXMLInstanceGenerator [18]. In generale sono pissibili diversi tipi di approccio, tra cui la generazione manuale. In questo caso però questa operazione può diventare un lavoro improponibile soprattutto quando lo schema di partenza è particolarmente esteso. Gli strumenti software ad oggi disponibili per la generazione di istanze XML implementano soltanto metodologie ad hoc o casuali, e in generale, le istanze derivate sono concepite in modo da coprire solo alcuni degli aspetti importanti dello schema. TAXI con il suo approccio cerca di superare questa limitazione fornendo una soluzione più completa ed innovativa, che consente una generazione non casuale e che segue le esigenze dell'utente [1].

1.1 LA STRATEGIA XPT

Lo scopo del tool TAXI, come sopra descritto, è quello di implementare una particolare strategia detta XML-based Partition Testing (XPT) [11], illustrata in Figura 1, che si basa sul metodo Category Partition (CP) [4] per la derivazione automatica di istanze a partire da un XML Schema.

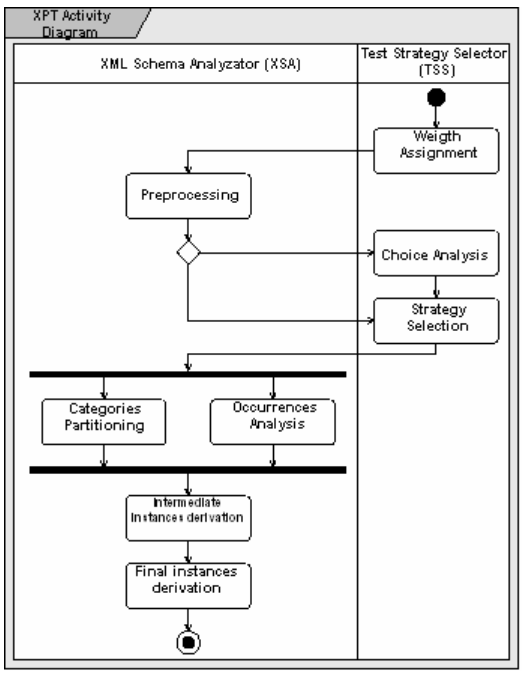


Figura 1. XPT main activities [1]

XPT prevede per la sua realizzazione due passi principali: XML Schema Analyzator (XSA), che si occupa di analizzare e manipolare lo schema XML, e il Test Strategy Selector (TSS), che si occupa di selezionare la strategia di test più opportuna a seconda delle esigenze dell'utente. In Figura 1 sono mostrate le attività di XPT e il modo in cui esse interagiscono.

Nel dettaglio, XSA implementa una metodologia per l'analisi dei costrutti dello schema XML e genera istanze automaticamente, mentre TSS implementa principalmente le varie strategie di test, selezionando le parti dello schema XML che devono essere testate. Queste due componenti lavorano in simbiosi per realizzare l'applicazione di una tecnica di Partition Testing, ovvero Category Partition, che rappresenta il pilastro su cui poggia l'intero approccio XPT.

Il metodo CP, introdotto negli anni ottanta e oggi largamente utilizzato, fornisce un metodo sistematico e semi-automatizzato per testare i dati derivati, cominciando dall'analisi delle specifiche, percorrendo una precisa e ben definita serie di passi [1]:

i. Analisi delle specifiche e identificazione delle unità funzionali;

ii. Suddivisione delle specifiche funzionali di un' unità in categorie;

iii. Suddivisione delle categorie in "Scelte" (Choices[1]);

iv. Determinazione dei vincoli e restrizioni tra le "Scelte" per prevenire la costruzione di combinazioni ridondanti, non significative o contraddittorie;

v. Derivazione delle specifiche del test. Categories, "Scelte" e restrizioni formano una specifica di test (Test Specification), adatta per il preprocessing automatizzato.

vi. Derivazione e valutazione della struttura del test (Test Frames).

vii. Generazione dei casi di test (test scripts).

Fornendo un'accurata rappresentazione del dominio dell'input, XML Schema si presta molto bene all'applicazione di CP. La suddivisione del dominio dell'input in unità funzionali e l'identificazione di categorie può essere fatta sfruttando la rappresentazione formale degli XML Schema. Per di più questa chiara rappresentazione dei sottodomini permette una derivazione di un insieme di istanze XML in maniera sistematica.

[1] È un caso che venga utilizzato il nome Choices come negli schemi XML.

In relazione a quanto detto, possiamo affermare che TAXI, che si basa su XPT, è in grado di fornire all'utente una metodologia completa, innovativa e automatica capace di adattarsi alle più diverse esigenze [11].

1.2 LE COMPONENTI XSA E TSS

Introduciamo ora la funzionalità di cui è responsabile l'analizzatore di schemi XML, cioè XSA, che collabora con la funzionalità che realizza un selezionatore della strategia di test, ovvero TSS [1], come si vede in Figura 1. XSA prende come input la versione pesata (vedremo fra poco cosa significa) di uno XML Schema originale e prevede un'attività di Preprocessor in cui i costrutti di XML Schema, come `all` e `simpleType`, e gli elementi come `Group`, `attributeGroup`, `ref`, `Type` vengono analizzati e manipolati. Per esempio considerando gli elementi `all`, viene derivata una combinazione casuale dei loro figli e usata per generare istanze. Queste operazioni di Preprocessing semplificano le successive derivazioni automatizzate [11]. Gli elementi `choice` sono gli unici esclusi dall'attività di Preprocessor perché analizzati dalla componente TSS.

Le successive due attività sono a carico di TSS e hanno rispettivamente le seguenti funzioni: estrazione delle unità funzionali (es. una lista di sottoschemi) dallo schema XML originale, per mezzo dell'analisi di elementi `choice`, e selezione della strategia di test che deve essere implementata. XSA invece procede con l'implementazione della metodologia CP ed esegue la suddivisione delle categorie e l'analisi delle ricorrenze (Occurrence Analisys) che dividono le categorie in `choice` e determinano le restrizioni su di esse.

In particolare il primo prende come input un insieme di sottoschemi selezionati dalla strategia di test e li analizza uno ad uno. In sostanza per ogni sottoschema viene estratta l'informazione richiesta per la generazione delle strutture intermedie.

La seconda attività analizza le occorrenze dichiarate per ogni elemento nel sottoschema e applica la strategia detta "boundary condition" [1], per derivare i valori limite (`minOccurrences` e `maxOccurences`) che devono essere considerati per la generazione delle istanze finali. I risultati di queste due attività vengono poi combinati insieme durante l'ultima attività.

Durante questi passi viene definito un insieme intermedio di strutture d'istanza, ognuna derivata dalle combinazioni degli elementi associati con le ricorrenze. Infine, in accordo con la

strategia di test selezionata e dando valori agli elementi elencati in una struttura d'istanza intermedia, l'attività di *derivazione finale di istanze* produce l'insieme finale di istanze [11].

Come già detto, gli schemi XML rappresentano la struttura complessiva del dominio dell'input. Grazie alla rappresentazione visiva degli schemi XML, è possibile implementare una strategia pratica e automatizzata per pianificare la derivazione di un certo insieme di istanze. La parte di XPT che ha il compito di fare questa operazione è il TSS. TSS completa l'implementazione di Category Partition e permette la selezione di tre specifiche strategie di test che possono essere applicate. TSS include quindi tre attività:

- assegnamento dei pesi, Weigths Assignment (WA);
- analisi delle `choice`, Choice Analysis (CA).
- scelta della strategie, Strategy Selection (SS).

La prima attività ha il compito di assegnare i pesi ai figli dei nodi `choice` tramite un'interfaccia grafica che consente all'utente di interagire col sistema. La seconda attività deriva un insieme di sottostrutture dallo schema XML originale tramite l'analisi delle `choice`, gestendo opportunamente i pesi. La terza seleziona la strategia di test da seguire per la generazione delle istanze finali. Naturalmente se lo schema non include nessuna `choice`, le prime due attività non vengono eseguite.

Capitolo 2

IMPLEMENTAZIONE DELLE STRATEGIE DI TEST

La parte software da noi sviluppata è interamente relativa a TSS. Di seguito verranno descritte in dettaglio le componenti che costituiscono TSS e il nostro contributo al loro sviluppo. La realizzazione delle componenti ha previsto anche lo sviluppo di un'interfaccia grafica per il tool tramite la quale si può accedere alle varie funzionalità previste. In Figura 2 vengono evidenziate le componenti al cui sviluppo abbiamo preso parte.

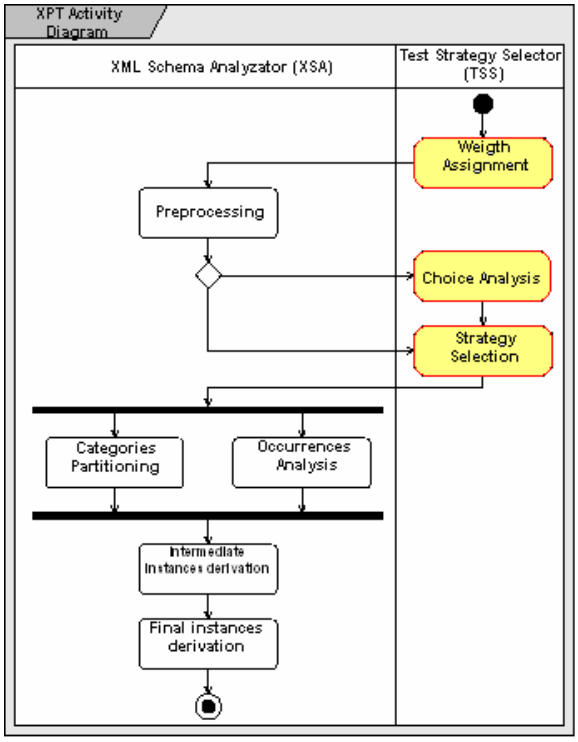


Figura 2. Le componenti relative al nostro lavoro

2.1 L'ASSEGNAMENTO PESI

L'idea sottesa all'attività di assegnamento pesi è che i figli di una stessa `choice` potrebbero non avere la stessa importanza ai fini della derivazione finale delle istanze. Infatti alcuni potrebbero essere poco utilizzati, e quindi di scarsa rilevanza, mentre altri potrebbero avere un impatto importante nella fase di derivazione. Poiché secondo la definizione di `choice` solo un figlio alla volta può apparire nell'insieme delle istanze finali, dal punto di vista dell'utente, la possibilità di scegliere quelle più importanti risulterà molto allettante. L'utente può pilotare la derivazione automatizzata delle istanze forzandola a derivare i figli di `choice` aventi peso maggiore [1].

Lo schema XML non fornisce la possibilità di dichiarare esplicitamente l'importanza dei vari figli di `choice`, ma è spesso lasciata implicitamente alla competenza delle persone che vogliono derivare le istanze. Per questo, TAXI fornisce una strategia sistematica che utilizza i figli di `choice` più importanti per pianificare la generazione delle istanze. In particolare XPT richiede esplicitamente di annotare con un valore ogni figlio di `choice`. Questo valore, appartenente all'intervallo (0,1) e chiamato *peso*, deve essere assegnato in maniera tale che la somma dei pesi associati a tutti i figli della stessa `choice`, sia uguale a 1 perché si tratta di una distribuzione discreta di probabilità [1]. Un peso può essere anche visto come un fattore di rischio, che indica quanto può essere rischiosa la scelta di un determinato nodo ai fini del buon funzionamento dell'applicazione. Inoltre, maggiore è il peso che l'utente vuole assegnare al nodo, maggiore è la probabilità che questo venga scelto per concorrere alla derivazione automatica delle istanze. Si possono essere adottati diversi criteri per assegnare il fattore d'importanza e ovviamente questo aspetto rimane a carico dell'utente.

Con l'assegnamento di pesi si riesce a ridurre in maniera consistente il numero delle istanze XML finali derivate, concentrando il numero di casi di test (istanze) sulle strutture aventi maggior impatto nella valutazione finale delle applicazioni software.

Figura 3. Esempio di inserimento pesi

Come si può osservare in Figura 3 l'utente può assegnare un peso ai nodi (tramite l'interfaccia grafica) compreso, come già detto, nell'intervallo (0,1) e con al massimo 4 cifre decimali, ai

nodi tramite l'interfaccia grafica. Non si accettano i pesi che non sono conformi a tali specifiche. I pesi "limite" 0 e 1 non sono ammessi perché assegnare uno 0, in questo caso, significa apportare una modifica radicale alla struttura dello schema stesso, che va contro quello che è stato definito precedentemente da chi lo ha scritto. Utilizzando uno schema modificato può verificarsi che le istanze derivate non siano più conformi allo schema e questo è un errore poiché il tool deve generare istanze valide per costruzione.

Se l'utente sta lavorando con un file che non è ancora stato pesato, nell'interfaccia, per ogni nodo cui si può assegnare un peso, viene visualizzato un valore di default calcolato secondo la formula

***peso default* = 1 / n° figli nodo padre** (2.1)

ottenendo quindi, per default, una distribuzione uniforme dei pesi dei nodi.

Nel caso in cui, invece, l'utente stia modificando un file precedentemente pesato, vengono caricati nell'interfaccia i valori preventivamente inseriti e memorizzati nello Schema.

Per illustrare una tipica computazione utilizziamo un esempio "giocattolo" supponendo di dover lavorare con il semplice XML Schema mostrato in Figura 4.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <xsd:schema>
  <xsd:choice>
    <xsd:element name="shipAndBill"/>
    <xsd:element name="singleUSAddress" type="USAddress"/>
  </xsd:choice>
 </xsd:schema>
```

Figura 4. Semplice esempio di XML Schema

Secondo la nostra scelta implementativa, una volta che l'utente ha terminato l'inserimento dei pesi ed effettua il salvataggio del file *pesato*, viene generato un nuovo file XML, non ben formato, in cui i pesi di ogni nodo pesato vengono inseriti all'interno di un attributo *peso* di un suo figlio speciale, chiamato `<ISTI_et_CNR_pesi>`. Il nuovo XML Schema prodotto non può essere considerato ben formato in quanto il tipo dei "contenitori di pesi" (`<ISTI_et_CNR_pesi>`) non è riconosciuto dallo schema. Questo non è importante ai fini del funzionamento del programma in quanto a questo livello di elaborazione non vi è la necessità di manipolare documenti XML ben formati. Quindi la versione pesata del file in Figura 4 è quella riportata in Figura 5 dove vediamo appunto l'utilizzo dei "contenitori di peso".

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <xsd:schema>
  <xsd:choice>
   <xsd:element name="shipAndBill">
    <ISTI_et_CNR_pesi peso="0.6" modificato="true"/>
   </xsd:element>
   <xsd:element type="USAddress" name="singleUSAddress">
    <ISTI_et_CNR_pesi peso="0.4" modificato="false"/>
   </xsd:element>
  </xsd:choice>
 </xsd:schema>
```

Figura 5. Esempio di XML Schema pesato

In Figura 5 possiamo notare che i figli di tipo `<ISTI_et_CNR_pesi>` oltre all'attributo *peso* possiedono anche un attributo chiamato *modificato,* la cui importanza è relativa alle politiche di aggiornamento dei pesi adottate.

In relazione alla politica di aggiornamento pesi, è importante conoscere in ogni istante quali sono i nodi già modificati poiché oltre a verificare che un peso inserito appartenga all'insieme consentito di valori, è necessario accertarsi che, sommato ai pesi degli altri nodi modificati, non dia un valore maggiore di 1. Quindi, per effettuare un aggiornamento corretto dei valori, per prima cosa si devono reperire i pesi dei fratelli del nodo appena pesato, verificare quali di essi sono già stati modificati in base al loro stato (modificato / non modificato), memorizzato all'interno di una mappa di flag, ed effettuare la somma dei loro pesi (a questa somma partecipa anche il valore del nodo appena inserito). Una volta fatto ciò si calcola il numero di nodi non ancora pesati dall'utente, che continuano a mantenere un valore di default. Se il numero di questi nodi è inferiore al numero totale dei fratelli si verifica se la somma calcolata sia minore di 1. Se lo è l'aggiornamento va a buon fine, quindi i pesi dei nodi che continuano a mantenere un valore di default verranno aggiornati, assegnando loro un peso calcolato secondo la formula:

$$\textbf{(1 - } \sum \textbf{ valori nodi pesati) / (n° nodi non pesati)} \qquad (2.2)$$

La somma può risultare uguale a 1 solo nel caso in cui il numero di nodi non modificati sia pari a 0, perché in caso contrario ad uno o più nodi dovrebbe essere assegnato un peso uguale a 0, ma ciò non è ammissibile per definizione.

Alla luce di quanto detto si deduce che è importante salvare all'interno del file XML Schema pesato anche lo stato di ogni singolo nodo per il ripristino della sessione di modifica dei pesi. Infatti la fase di "pesatura" non deve necessariamente avvenire con un solo accesso, ma un file può essere aperto e rivisto più volte dall'utente prima di passare alla fase successiva e affinché questa

politica funzioni correttamente è necessario caricare dal file anche lo stato del nodo al momento dell'ultimo salvataggio.

Un peso non è esclusivamente assegnabile ad un figlio di una `choice`. Infatti c'è un'altra categoria di nodi, non presenti nello Schema di origine e introdotti per specifiche necessità, che sono detti *nodi di primo livello* e sono memorizzati nello Schema come nodi di tipo `<ISTI_et_CNR_first_LEVEL_element>`. Infatti nel caso in cui uno Schema avesse più di un figlio di tipo `Element`, questi devono essere raggruppati sotto un unico padre, figlio dello Schema, rappresentato da un nodo di primo livello. Un nodo ti questo tipo deve essere trattato proprio come una `choice` ed è quindi possibile assegnare un peso a tutti i suoi figli. Ciò è importante perché tramite questa operazione possiamo stabilire quali sono gli elementi dello Schema che vogliamo considerare nella derivazione delle istanze e quali no. Quindi in generale, al momento del caricamento del XML Schema, occorre effettuare una manipolazione preventiva dell'albero DOM [21] che prevede una visita *BFS* che si ferma al suo primo livello. Tramite questa operazione possiamo individuare i nodi tipo `Element` ed "appenderli" come figli ad un nodo creato da noi che a sua volta verrà "appeso" allo Schema iniziale come figlio.

I pesi che possiamo assegnare ai figli di un nodo `<ISTI_et_CNR_first_LEVEL_element>` sono compresi nell'intervallo [0, 1]. Ciò significa che anche i valori limite 0 e 1 possono essere assegnati come pesi ad un nodo. È fondamentale poter fare ciò, affinché si possa decidere quali elementi includere nella derivazione delle istanze, con il loro relativo valore di importanza (peso), e quali escludere. A questo livello dare come peso 0 significa l'esclusione di un elemento dalla derivazione finale e assegnare come peso 1 significa escludere tutti gli altri fratelli. Mentre nel caso dei figli dei nodi `choice` assegnare uno 0 comporta una modifica radicale alla struttura dello schema, adesso è un'opzione che si da all'utente per scegliere quali elementi far concorrere per la derivazione finale.

Supponiamo di avere un semplice file XML Schema come quello in Figura 6, ottenuto aggiungendo due figli fittizi di tipo `Element` al file di Figura 4.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <xsd:schema>
  <xsd:element name="AAA"/>
  <xsd:element name="BBB"/>
  <xsd:choice>
   <xsd:element name="shipAndBill"/>
   <xsd:element name="singleUSAddress" type="USAddress"/>
  </xsd:choice>
 </xsd:schema>
```

Figura 6. Esempio di XML Schema con più figli di tipo `Element`

Il file di Figura 6 può essere rappresentato tramite la struttura ad albero di Figura 7 dove i figli di tipo `Element` dello Schema sono evidenziati in celeste.

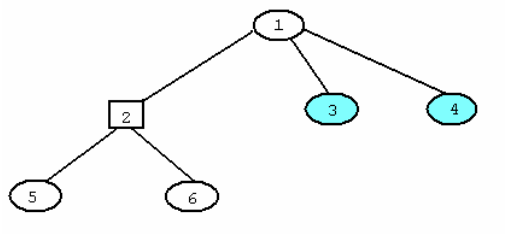


Figura 7. Struttura ad albero di XML Schema con più figli di tipo `Element`

A seguito della manipolazione iniziale otteniamo un albero DOM come quello illustrato in Figura 8, con aggiunto un figlio di primo livello evidenziato in azzurro ed i figli dello Schema a lui assegnati evidenziati ancora in celeste.

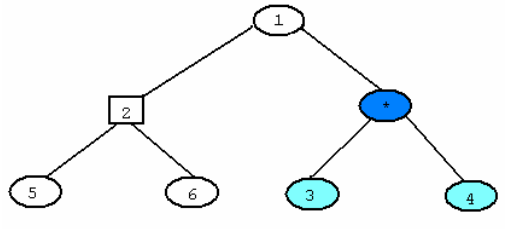


Figura 8. Albero DOM modificato con l'aggiunta di un elemento di primo livello

Per quanto riguarda il salvataggio dei pesi su file, avviene esattamente come nel caso precedente ottenendo uno XML Schema come il seguente.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <xsd:schema>
  <xsd:choice>
   <xsd:element name="shipAndBill">
    <ISTI_et_CNR_pesi peso="0.3" modificato="true"/>
   </xsd:element>
   <xsd:element type="USAddress" name="singleUSAddress">
    <ISTI_et_CNR_pesi peso="0.7" modificato="false"/>
   </xsd:element>
  </xsd:choice>
  <ISTI_et_CNR_first_LEVEL_element>
   <xsd:element name="AAA">
    <ISTI_et_CNR_pesi peso="0.9" modificato="true"/>
   </xsd:element>
   <xsd:element name="BBB">
    <ISTI_et_CNR_pesi peso="0.1" modificato="false"/>
   </xsd:element>
  </ISTI_et_CNR_first_LEVEL_element>
 </xsd:schema>
```

Figura 9. Schema con più figli di tipo `Element` pesato

Una volta che i pesi sono stati assegnati, sia ai figli di nodi `choice` sia a quelli dei nodi di primo livello, vengono usati da XPT per derivare, per ogni figlio di `choice`, il relativo fattore d'importanza, chiamato *peso finale*. Prima di poter calcolare il peso finale dobbiamo attendere che tutte le manipolazioni del documento relative alle `choice` siano terminate. Infatti queste potrebbero provocare un disequilibrio all'interno dei pesi dei figli delle `choice`, per questo dovrà esserne effettuata una normalizzazione secondo determinate politiche. A questo scopo il procedimento del calcolo dei pesi finali verrà introdotto nella sessione successiva.

In base ai pesi finali calcolati per ogni foglia è possibile derivare il peso finale dei sottoalberi derivati dallo schema iniziale.

2.2 L'ANALISI DELLE CHOICE

Dopo l'attività di Preprocessor, XPT prevede l'analisi delle `choice` per derivare un insieme di sottoschemi tramite un algoritmo, che sfrutta una visita *BFS* modificata che si occupa di produrre una serie di sottoalberi, ricavati come combinazioni contenenti ognuna un unico figlio di una determinata choice dell'albero iniziale.

Prima è però necessario effettuare il calcolo dei pesi relativi a ciascun figlio di choice. Il peso del figlio di un nodo `choice` viene calcolato come il prodotto dei pesi di tutti i nodi sul cammino completo dalla radice a quel nodo, includendo il peso del nodo stesso, cioè secondo la formula

$$\textit{peso foglia}^{\,i} = \prod \text{ pesi dei nodi appartenenti al path}^{\,i} \qquad (2.3)$$

I figli dei nodi choice vengono anche detti *foglie* poiché se nei loro sottoalberi non sono contenute altre `choice` tali sottoalberi vengono recisi riducendo il loro padre ad una foglia.

Per rendere più chiaro il procedimento che consente di calcolare il peso finale proviamo a fornire un semplice esempio.

In Figura 10 abbiamo un esempio di albero DOM pesato. Infatti, in rosso sono stati evidenziati i collegamenti dei nodi con i loro figli contenenti i pesi.

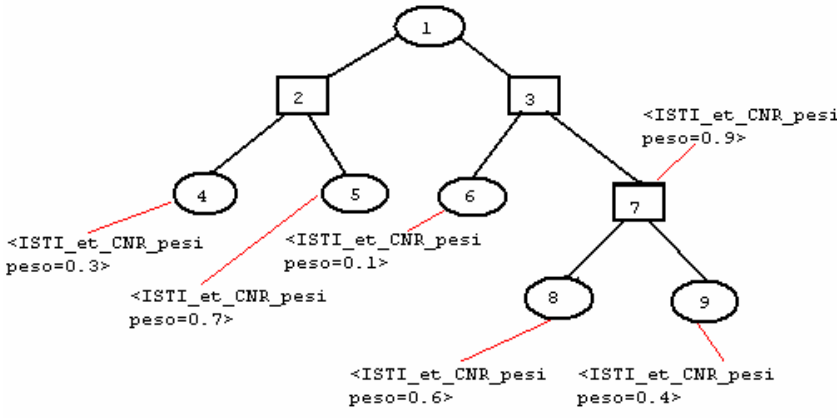


Figura 10. Un esempio di albero DOM pesato

A partire dall'albero posseduto vengono derivati tutti i possibili cammini esistenti che vanno dalla radice ad un figlio di un nodo choice. I cammini relativi all'albero mostrato in Figura 10 vengono messi in evidenza in Figura 11, contrassegnandoli ognuno in un colore diverso.

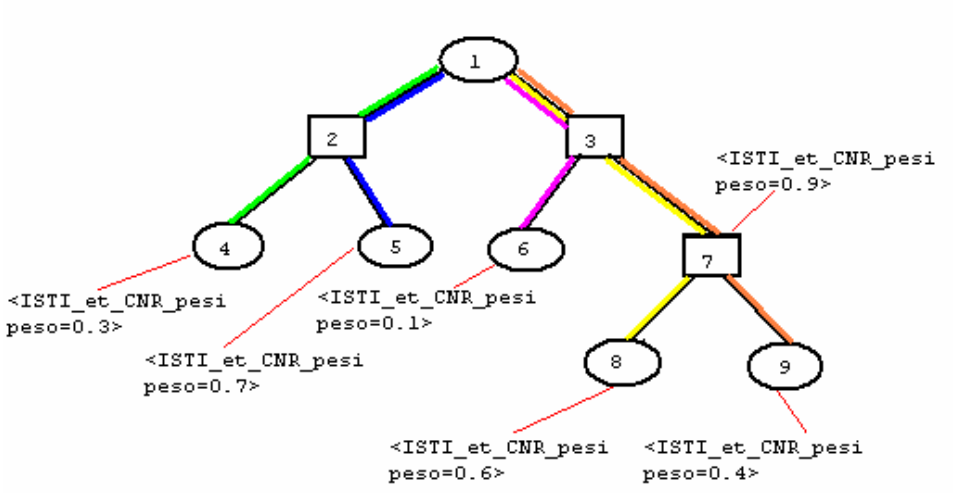


Figura 11. I cammini ricavati dall'albero in Figura 10

Percorrendo uno di questi cammini e moltiplicando tutti i pesi che incontriamo lungo di esso, come specificato in (2.3), otteniamo il peso desiderato. Operando in questo modo con ognuno dei percorsi otteniamo il risultato mostrato in Figura 12.

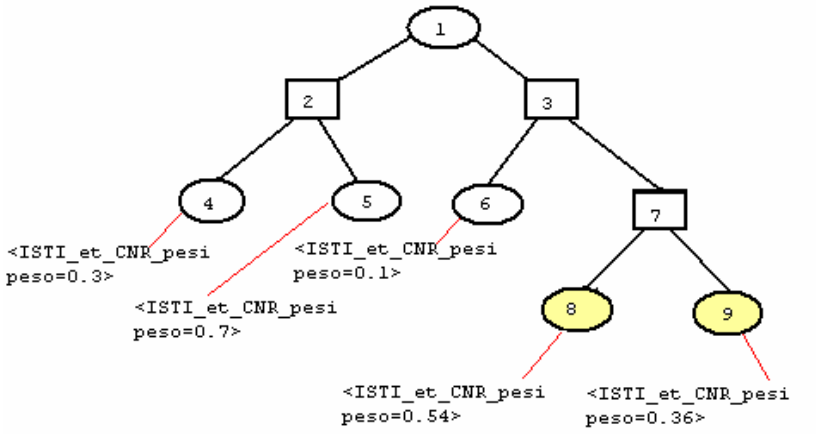


Figura 12. Il calcolo dei pesi alle foglie

Il peso che abbiamo ottenuto rappresenta l'importanza di quel figlio e quanto sforzo dovrebbe essere messo nella derivazione delle istanze che lo contengono [1]. Questo algoritmo viene applicato sull'albero originale pesato. Una volta fatto ciò è possibile procedere con la derivazione delle sottostrutture.

Per quanto riguarda la generazione dei sottoschemi, la `choice` permette ad uno solo dei figli di essere presente dentro un'istanza e ciò significa che per ogni alternativa, dentro un costrutto `choice`, non possono essere derivati più sottoschemi uguali contenenti uno stesso nodo, ma saranno tutti distinti. Estendendo in qualche modo il significato originale di unità funzionale, ogni possibile sottoschema viene messo in corrispondenza con la nozione di *unità funzionale* di CP.

In altri termini le unità funzionali di XPT sono intese come *unità di dominio* e sono quindi assimilate a sottoinsiemi di elementi di schemi XML che possono generare istanze corrette, gestendo un insieme separato di input di dati. Il problema ovviamente è la possibile presenza di più `choice`, anche annidate, all'interno di uno Schema, che porta all'incremento del numero di combinazioni possibili. In questo caso, durante l'attività di analisi delle `choice`, vengono prodotti tanti sottoschemi pari al numero di possibili combinazioni dei figli dei nodi `choice`. Supponiamo di avere una situazione iniziale come quella rappresentata in Figura 13.

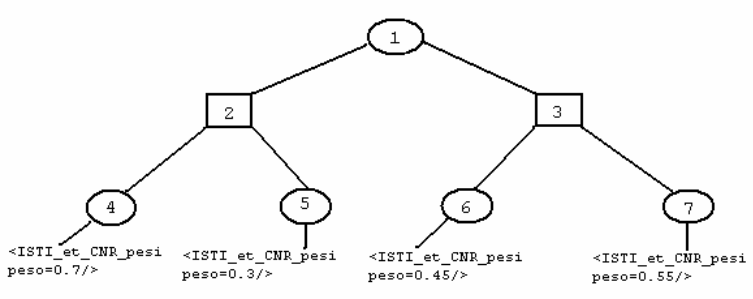


Figura 13. Albero di partenza per il calcolo di combinazioni

Come possiamo notare dalla Figura 14 durante questa operazione, i pesi finali precedentemente calcolati, non vengono modificati e vengono lasciati all'interno delle sottostrutture. Relativamente all'esempio fornito possiamo ricavare le seguenti quattro combinazioni.

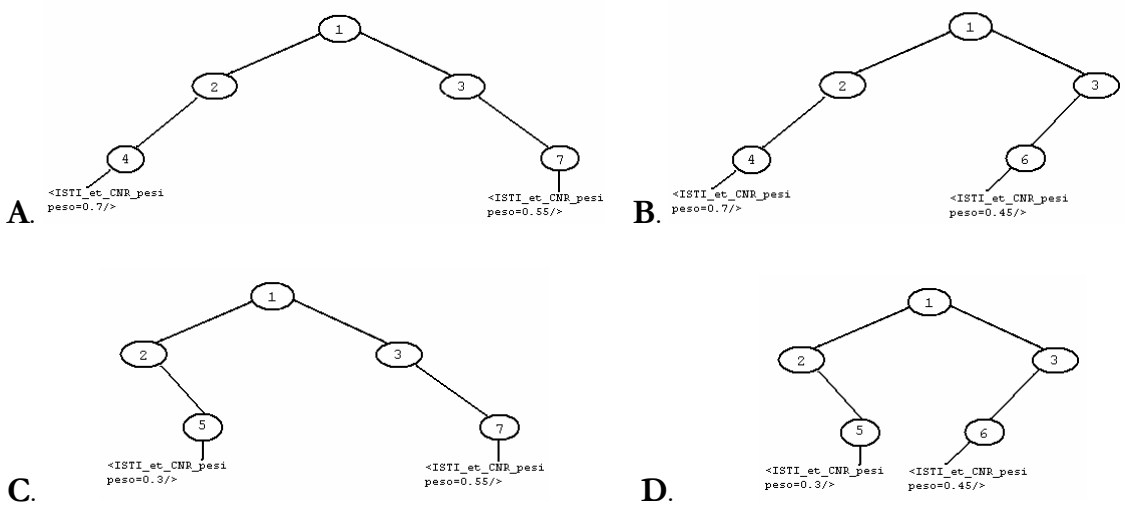


Figura 14. Un esempio di combinazioni ricavate da Figura 13

In ognuna compare uno ed un solo figlio appartenente ad una choice specifica e le `choice` sono state sostituite da elementi di tipo `sequence`.

Una volta completato questo passo è possibile utilizzare i pesi che abbiamo conservato per derivare un unico valore relativo ad ogni "combinazione", chiamato *peso del sottoalbero* (subtree weight), utilizzato poi nella selezione della strategia di test. Chiaramente, considerando ogni sottostruttura, a cominciare dalla sua radice, viene derivato l'insieme dei nodi più profondi aventi un peso finale. Soltanto questi numeri vengono poi sommati insieme per ottenere un peso del sottoalbero parziale, caratterizzante la sottostruttura analizzata. La formula per il calcolo del peso del sottoalbero è data da

$$\textbf{\textit{peso sottoalbero}} = \sum \textbf{peso foglie} \qquad (2.4)$$

Quindi, avendo a disposizione tutte le sottostrutture derivate dallo schema iniziale, Figura 14, adesso è possibile calcolarne il peso. Secondo la (2.4) i pesi dei 4 sottoalberi saranno i seguenti:

A. ***peso sottoalbero*** $_A$ = 0.7 + 0.55 = 1.25 ;

B. ***peso sottoalbero*** $_B$ = 0.7 + 0.45 = 1.15 ;

C. ***peso sottoalbero*** $_C$ = 0.3 + 0.55 = 0.85 ;

D. ***peso sottoalbero*** $_D$ = 0.3 + 0.45 = 0.75 .

Come possiamo notare però la somma dei pesi di tutti i sottoalberi è molto maggiore di 1 poiché pari a 4. Ciò non è ammissibile perchè anche la somma dei pesi delle sottostrutture deve costituire una distribuzione discreta di probabilità. Il peso di ogni sottostruttura infatti indica la probabilità che questa venga scelta per la generazione finale delle istanze. Per risolvere questo inconveniente e per trovare il *peso finale dell'albero* è necessario ancora una volta operare una normalizzazione dei pesi. Il peso finale viene ricavato dividendo il peso attuale di ogni albero per la somma dei pesi parziali di tutti i sottoalberi, così che la somma dei pesi dei sottoalberi dell'intero insieme di sottostrutture sia uguale a 1. Ciò è espresso tramite la proporzione

$$\textbf{1 : somma attuale} = \textbf{\textit{peso finale albero}} \textbf{ : peso attuale} \qquad (2.5)$$

dove **somma attuale** è la variabile che rappresenta la somma totale dei pesi di tutti gli alberi generati e ***peso finale albero*** l'incognita. E da (2.5) si ricava:

$$\textbf{\textit{peso finale albero}} = \textbf{peso attuale / somma attuale} \qquad (2.6)$$

Questa operazione va effettuata su ognuno dei sottoalberi generati in modo da poter lavorare, in seguito, con valori corretti e consistenti. Facendo ancora riferimento al nostro esempio, operando secondo la (2.6) otteniamo i seguenti pesi finali:

A. ***peso finale*** $_A$ = 1.25 / 4 = 0.3135;

B. ***peso finale*** $_B$ = 1.15 / 4 = 0.2875;

C. ***peso finale*** $_C$ = 0.85 / 4 = 0.2125;

D. ***peso finale*** $_D$ = 0.75 / 4 = 0.1875.

Il valore che si ottiene tramite la (2.6) è un valore corretto e consistente ed è quello che determinerà l'eventuale scelta di una sottostruttura ai fini della generazione delle istanze finali in fase di selezione della strategia di test.

2.3 LA SELEZIONE DELLA STRATEGIA DI TEST

Seguendo i passi fin qui descritti, ogni insieme di sottostrutture è stato definito e i pesi di uno specifico sottoalbero assegnati ad ognuno di loro. Adesso è necessario determinare la strategia di test da adottare per la derivazione delle istanze. Per questo si considerano tre differenti situazioni [1]:

i. Nella prima un numero prestabilito di istanze deve essere derivato da uno specifico schema XML. In questo caso XPT permette di distribuire le istanze tra i sottoalberi possibili in maniera accurata;

ii. Nella seconda l'utente vuole coprire solo una percentuale prefissata tra tutti i sottoschemi disponibili. In questo caso XPT seleziona opportunamente quei sottoschemi che risulteranno più adatti ai fini del testing;

iii. Infine nell'ultima viene proposta una strategia di test mista, in cui un numero fissato di istanze vengono distribuite tra uno specifico sottoinsieme di sottoschemi.

Da un punto di vista pratico, le strategie menzionate sopra, sono applicate come decritto nei seguenti paragrafi. Bisogna però spiegare come avviene, in generale, la scelta dei sottoalberi che possono concorrere alla generazione finale delle istanze. A prescindere dalla strategia di test utilizzata, è necessario mantenere tutti i sottoalberi all'interno di una struttura dati in modo tale che questi possano essere ordinati, in base al loro peso, in maniera crescente (o decrescente). Infatti i sottoalberi candidati ad essere scelti sono quelli dal peso maggiore e quindi quelli memorizzati nelle ultime posizioni (o nelle prime se l'ordinamento è avvenuto in maniera decrescente) della struttura. Questo è utile affinché il reperimento dei sottoalberi con peso maggiore possa essere effettuato rapidamente e in maniera più intuitiva.

Affinché l'utente possa scegliere la strategia di test, è stata progettata un'altra interfaccia che è la seguente:

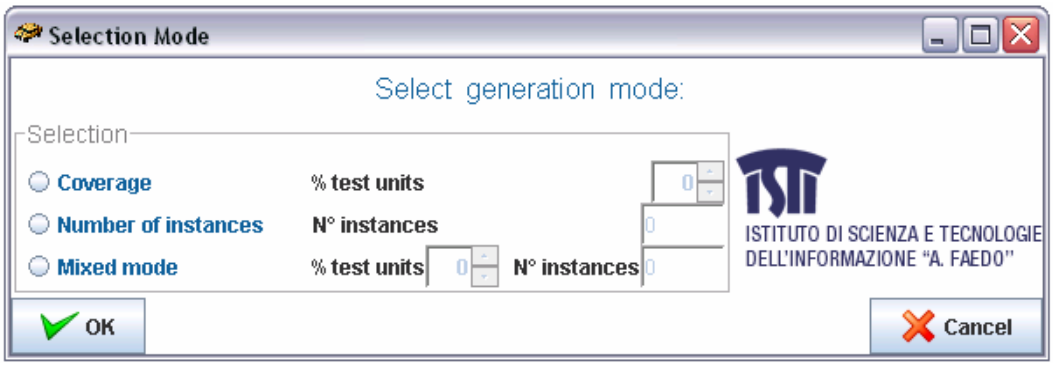


Figura 15. L'interfaccia per la scelta della strategia

Come ultimo passo della scelta della strategia di test, una volta ricavato l'insieme delle sottostrutture, associate o con il peso del loro sottoalbero o con il numero di istanze finali che deve essere derivato, viene passato alla componente XSA per continuare l'esecuzione del CP.

Al fine di comprendere la logica con cui si opera nell'applicazione di ciascuna delle tre metodologie di generazione sopraelencate, è opportuno fornire un esempio pratico che ci consenta di confrontare i risultati ottenuti operando secondo ognuna delle tre strategie. L'esempio su cui lavoreremo viene illustrato nella Figura 16, che costituisce una parte della rappresentazione ad albero di uno schema, contenente due `choice`.

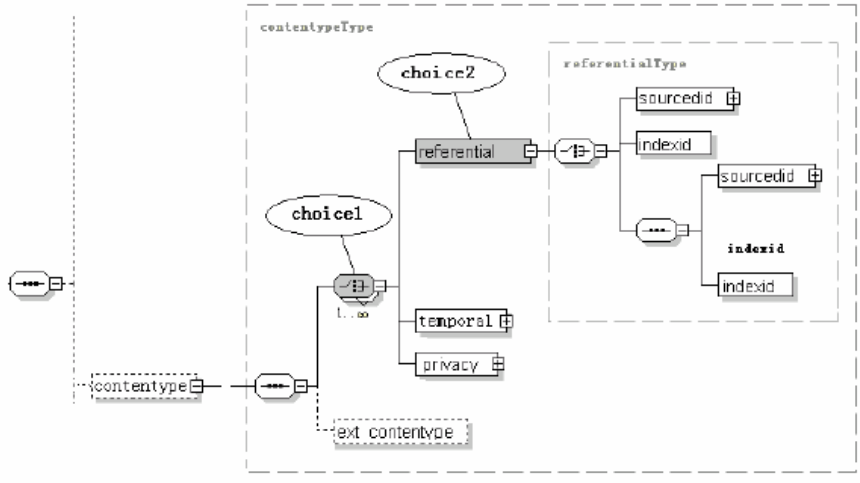


Figura 16. Un esempio di albero per la generazione finale delle istanze

"`choice1`" è figlio di "`contentype`", "`choice2`" è l'elemento "`referential`", figlio di "`choice1`". "`choice1`" ha tre figli e supponiamo che il peso di "`referential`" sia stato impostato a 0.5, quello di "`temporal`" a 0.3 e quello di "`privacy`" a 0.2. L'altra `choice`,

"referential", ha anch'essa tre figli e il peso per "sourceid" è 0.2, per "indexid" 0.3 e per l'ultimo 0.5. Possiamo inoltre notare che la somma dei pesi di ciascuna choice è stato opportunamente normalizzato ad 1 prima di procedere con la generazione automatica.

A partire dallo stralcio di schema fornito in Figura 16, possono essere ricavati 5 sottoschemi i cui pesi sono i seguenti:

1) Il peso del sotto-schema1, che include "sourceid", è 0.15;
2) Il peso del sotto-schema2, che include "indexid", è 0.10;
3) Il peso del sotto-schema3, che include sia "sourceid" che "indexid", è 0.25;
4) Il peso del sotto-schema4, che include "temporal", è 0.3;
5) Il peso del sotto-schema5, che include "privacy", è 0.2.

In base a questi dati possiamo mostrare il risultato ricavato facendo uso delle tre diverse strategie proposte da TAXI.

2.3.1 NUMERO FISSATO DI ISTANZE

Se viene fissato un numero NI di istanze finali da generare, la strategia XPT può essere usata per sviluppare NI istanze finali tra le molte che potevano essere immaginate all'inizio a partire dallo schema XML originale. Usando i pesi dei sottoalberi associati a ogni sottostruttura, il numero di istanze da assegnare ad ogni sottoalbero sarà calcolato secondo la formula:

$$\textbf{\textit{n° istanze}} = \textbf{NI * peso del sottoalbero} \quad [1] \tag{2.9}$$

Facendo riferimento all'esempio riportato in Figura 16, supponiamo di fissare il numero dei casi di test desiderati a "1000". Dopo la generazione automatica, i pesi vengono distribuiti equamente, in base al peso, tra i sotto-schemi prodotti. Così, operando in base alla (2.9), otteniamo la seguente distribuzione di istanze tra i 5 sottoschemi generati:

1) 150 casi di test ottenuti a partire dal sotto-schema1;
2) 100 casi di test ottenuti a partire dal sotto-schema2;
3) 250 istanze ottenute a partire dal sotto-schema3;
4) 300 istanze ottenute a partire dal sotto-schema4;

5) 200 istanze ottenute a partire dal `sotto-schema5`;

2.3.2 PERCENTUALE DI COPERTURA FISSATA

Consideriamo adesso il caso alternativo in cui viene stabilita una certa percentuale di copertura funzionale dei sottoalberi a disposizione (es. 80%), come criterio d'uscita per il testing. In questo caso XPT guida la scelta del caso di test, mettendo in evidenza le sottostrutture più importanti da essere usate per la derivazione automatica delle istanze finali. Chiaramente considerando il valore di copertura C fissato, la scelta delle sottostrutture da usare può essere derivata partendo dai sottoalberi con peso maggiore, moltiplicando per cento i loro pesi e sommandoli tra loro, fino a che non si raggiunge un valore maggiore o uguale a C. Anche in questo caso l'utente, interagendo con l'interfaccia di Figura 15, selezionerà la prima opzione proposta scegliendo la percentuale tramite l'apposito strumento.

In relazione all'esempio di Figura 16, per esempio, possiamo fissare la percentuale di copertura al "50%". Fatto ciò TAXI preleva alcuni tra i sotto-schemi disponibili in base al loro peso. In questo caso vengono considerati solo i sottoschemi 3 e 4 perché, operando secondo quanto detto sopra, la somma dei casi di test ricavabili da questi è maggiore del 50%. In questo caso otterremo 2048 casi dal sottoschema 4 e 128 dal sottoschema 3, per un totale di 2176 casi di test.

2.3.3 PERCENTUALE DI COPERTURA FISSATA E NUMERO DI ISTANZE RELATIVO A TALE PERCENTUALE

In questo caso vengono combinate le due strategie descritte sopra. XPT prima seleziona le sottostrutture appropriate, utili per raggiungere una certa percentuale di copertura funzionale. Poi i pesi delle sottostrutture ricavate vengono usati per la distribuzione tra la sottostruttura selezionata, per il numero fissato di istanze che deve essere derivato automaticamente. Continuando a fare riferimento allo stesso esempio, sappiamo che l'utente dovrà scegliere l'ultima opzione del menù fornito dall'interfaccia di Figura 15, selezionare la percentuale di copertura ed infine scegliere il numero di istanze da derivare.

In relazione all'esempio di Figura 16, supponiamo che la percentuale di copertura sia fissata al "50%" e che il numero di istanze da derivare sia fissato a "1000". In base a questi dati dovremmo ricavare 525 casi di test dal `sotto-schema4` e 475 dal `sotto-schema3`. In realtà il numero massimo di casi che possiamo ricavare dal `sotto-schema3` si limita a 128 quindi, usando questa strategia, non è possibile raggiungere il numero di istanze scelto, cioè 1000, ma ci fermiamo a 653.

Capitolo 3

L'ARCHITETTURA DEL TOOL TAXI

In questa sezione descriveremo l'architettura del tool TAXI, che ha lo scopo di implementare la strategia XPT descritta nella sezione precedente, mostrando le interazioni tra le varie componenti illustrate in Figura 17, ponendo particolare accento su quelle implementate da noi *ex novo* e sulle funzionalità che abbiamo aggiunto al codice preesistente o in fase di sviluppo.

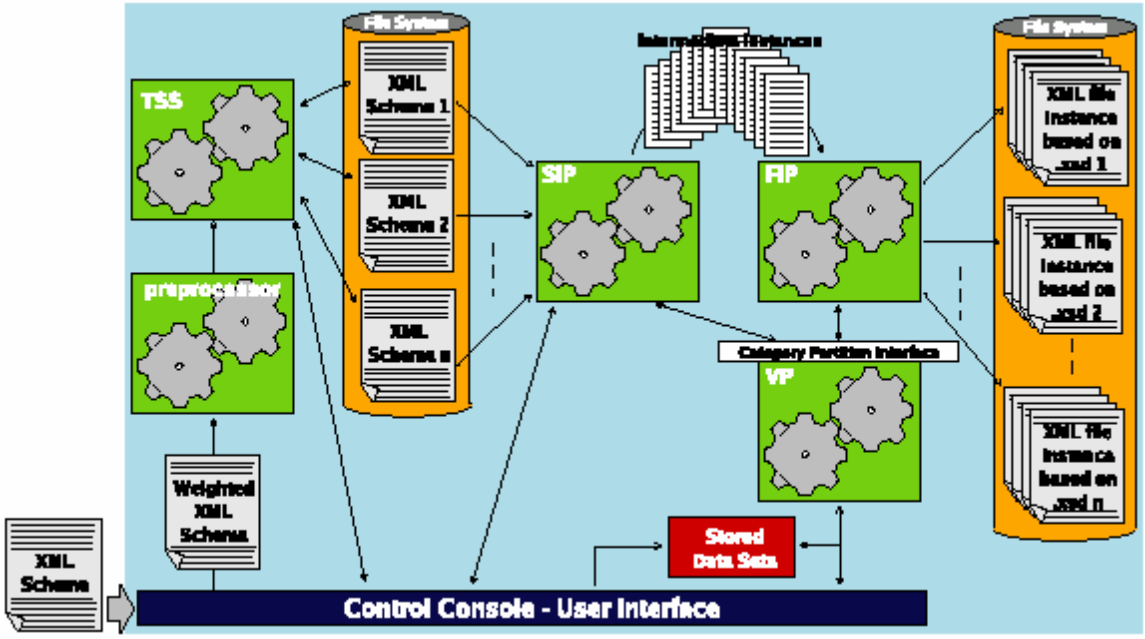


Figura 17. L'architettura del tool TAXI [1]

Come già detto, TAXI prende in input uno XML Schema, come si vede anche in Figura 17, su cui viene fatto il parsing utilizzando W3C Document Object Model (DOM) [21]. Questo è un modo standard per accedere a documenti XML e manipolarli, che permette di memorizzare la struttura dello schema XML in una struttura ad albero che può essere visitata in qualsiasi direzione.

TAXI è diviso principalmente in sei componenti, elencate di seguito, che interagiscono tra di loro come mostrato in Figura 17.

- L'interfaccia utente;
- Preprocessor;
- Test Strategy Selector (TSS);
- Skeleton of Instances Producer (SIP);
- Final Instances Producer (FIP);
- Values Provider (VP).

In particolare in questo tirocinio, in relazione alla Figura 17, sono stati da noi interamente implementati, come si può vedere in Figura 18, la componente TSS e l'interfaccia utente.

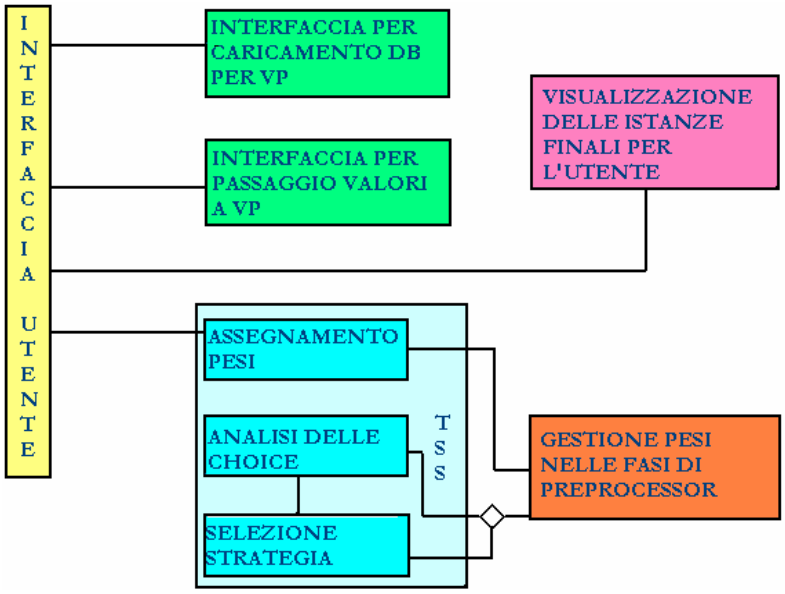


Figura 18. Le componenti implementate

Abbiamo inoltre integrato il codice esistente della componente Preprocessor con alcune funzionalità aggiuntive relative alla normalizzazione dei pesi nei casi in cui, a seguito di manipolazioni dell'XML Schema, venga modificato l'insieme dei figli di una choice; abbiamo realizzato alcune interfacce che verranno utilizzate dalla componente VP, in corso di implementazione, per l'interazione con l'utente e infine abbiamo definito il formato dati con cui FIP mostrerà le istanze prodotte all'utente finale.

L'interfaccia utente è stata la parte più impegnativa sia a livello di quantità di tempo impiegato sia per le difficoltà incontrate, da un lato perché si è dovuta affrontare una consistente fase di documentazione relativamente alle interfacce grafiche, di cui non eravamo esperti, e dall'altro

perché, una volta acquisite le giuste competenze, è stato abbastanza laborioso trovare una soluzione che non fosse estremamente complessa da produrre e, al tempo stesso, accettabile a livello grafico. Il risultato ottenuto è il seguente:

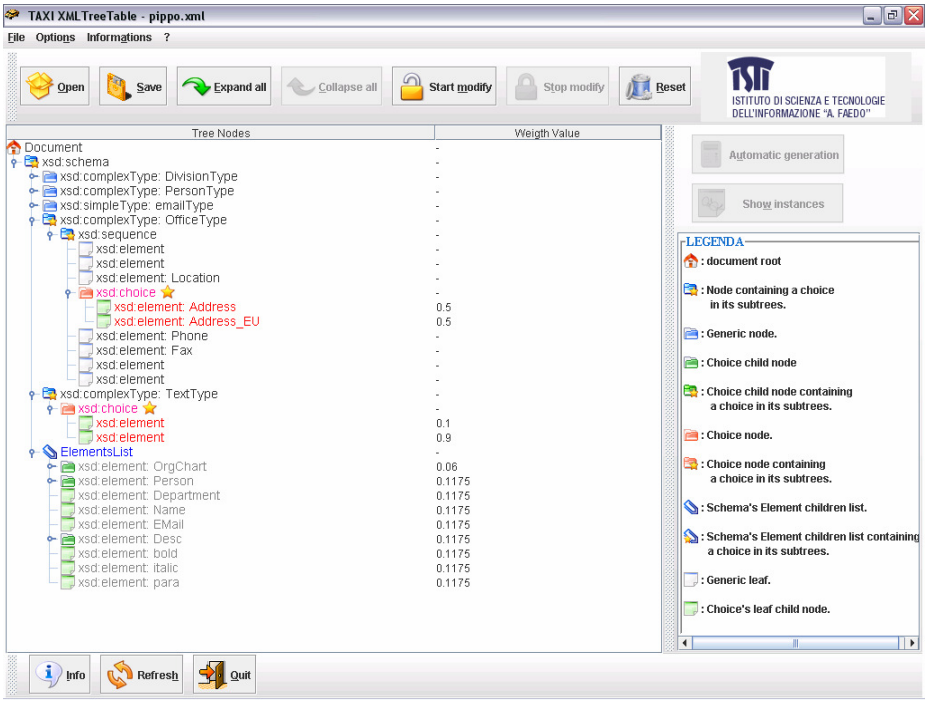


Figura 19. L'interfaccia grafica del tool

L'interfaccia gestisce l'interazione con l'utente, che può influenzare e controllare il processo di generazione delle istanze secondo le proprie esigenze. In questo modo questa componente di TAXI acquisisce l'input per iniziare la generazione dell'insieme dei casi di test. Una delle operazioni richieste all'utente è perciò la scelta del file XML Schema da cui l'utente vuole derivare istanze valide. Da questo punto in avanti prosegue la generazione automatica.

Dopo la scelta dello schema XML, l'utente ha bisogno di poter modificare i pesi dello schema, infatti lo scopo fondamentale dell'interfaccia è quello di consentire all'utente l'assegnamento di opportuni pesi ai figli di nodi `choice` e agli elementi di primo livello, la cui funzione è stata descritta nel capitolo precedente. Quindi in primo piano abbiamo una tabella dove in ogni riga è memorizzato un nodo del file XML con accanto un campo per l'assegnazione del peso.

Inoltre l'interfaccia è indispensabile per scegliere la strategia di test, infatti cliccando sull'apposito pulsante, una volta terminato l'assegnamento pesi, si da il via alla generazione automatica e l'utente potrà scegliere la strategia di test tramite una seconda apposita interfaccia (Figura 15).

Il peso, come descritto nella precedente sezione, viene usato per determinare quali figli di `choice` andranno a contribuire alla generazione delle istanze finali. Usando i pesi assieme alla strategia di test, TAXI può generare la giusta quantità di casi di test da ogni sottoalbero col peso maggiore. Lo schema XML pesato nella prima fase di TSS, viene poi passato alla componente Preprocessor, che implementa l'attività di preprocessing il cui scopo è risolvere i tag: `group`, `attributeGroup`, `ref`, `type`, `restriction`, `extension` e `all`.

La risoluzione di questi tag può però provocare delle situazioni in cui è possibile che ad una `choice` vengano aggiunti figli, con peso e non, oppure che le vengano sottratti. Un tale cambiamento provoca anche un "disequilibrio" all'interno dei pesi poiché aggiungendo o togliendo figli ad una `choice` la somma totale dei loro pesi non sarà certamente più pari ad 1. Lavorando con distribuzioni discrete di probabilità, questo non è accettabile. Nostro compito è stato quindi quello di risolvere ciò applicando delle politiche di normalizzazione dei pesi per risolvere le seguenti situazioni:

i. Somma dei pesi minore di 1, dovuta all'eliminazione di un figlio;
ii. Somma dei pesi maggiore di 1, dovuta all'aggiunta di un figlio con peso;
iii. Somma dei pesi uguale ad 1, dovuta all'aggiunta di un figlio senza peso o ad una non modifica dei figli della `choice`.

Quindi a seguito di determinate elaborazioni dell'albero DOM è necessario effettuare una visita in ampiezza sull'albero e controllare accuratamente le somme dei pesi dei figli di ciascuna `choice` incontrata, risolvendo opportunamente la situazione riscontrata (riquadro arancio di Figura 18).

Nel caso in cui la somma dei pesi sia maggiore (ii) o minore (i) di 1, situazione rappresentata in Figura 20

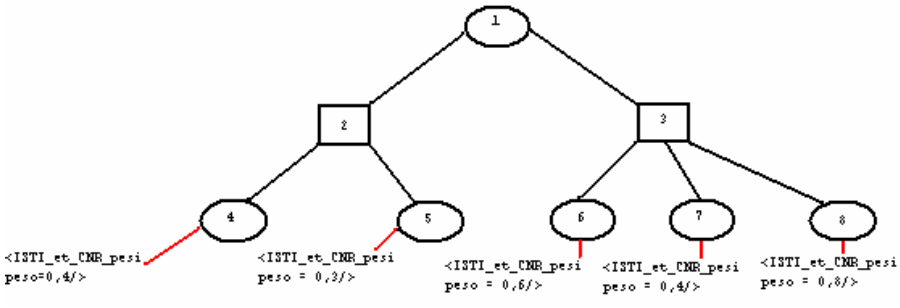


Figura 20. Primi due casi di normalizzazione

la normalizzazione avviene effettuando la proporzione:

1 : somma attuale = *peso normalizzato* : peso attuale (2.3)

dove il termine in corsivo rappresenta l'incognita, **somma attuale** la somma dei pesi dei figli della choice modificata e **peso attuale** il peso del nodo ancora da normalizzare. Quindi, in base alla (2.3), possiamo calcolare il peso normalizzato nel seguente modo:

***peso normalizzato* = peso attuale / somma attuale** 3.2

Una volta trovata l'incognita ***peso normalizzato*** si modifica l'attributo peso di ognuno dei figli `<ISTI_et_CNR_pesi>` inserendo al suo interno il nuovo valore calcolato, come possiamo vedere in Figura 21.

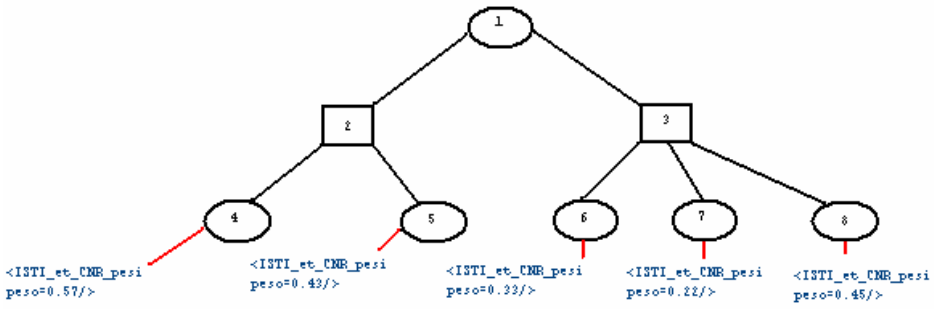


Figura 21. Primi due casi di normalizzazione risolti

Per quanto riguarda il caso della somma uguale ad 1 (iii), si opera in maniera diversa in quanto dobbiamo controllare se esiste almeno un figlio senza peso. Se non esiste significa che la `choice` non è stata modificata, mentre se ne esistono assegniamo loro un peso di default pari ad 1 ottenendo una somma totale dei pesi maggiore di 1. Quindi si opera come per (ii), ottenendo il peso normalizzato di ogni nodo tramite la 3.2. Una volta corrette queste situazioni tutti i pesi sono normalizzati, la distribuzione discreta delle probabilità è ripristinata e si può continuare a lavorare con valori corretti e consistenti.

Dopo la fase di preprocessing il file che era stato dato in input non è più uno schema ben-formato (well-formed), poiché alcuni elementi sono stati duplicati. In questo così detto "schema" sono rimasti soltanto `sequence`, `choice` e `simpleType`.

Successivamente TAXI passa di nuovo questo schema al TSS. Il primo passo di questa componente è risolvere le `choice`, che è quella che produce molteplici sottoschemi. A questo punto la componente SIP recupera e analizza ogni sottoschema, estraendo da ogni elemento soltanto le informazioni necessarie, utili cioè per la costruzione delle istanze finali. Combinando quindi il peso con la strategia, il totale dei casi di test può essere calcolato automaticamente da TAXI, come abbiamo

visto nel capitolo precedente (3° fase TSS Figura 18). In particolare quando è verificata la condizione `minOccurences < maxOccurences`, collaborando con la componente VP, TAXI stabilisce il numero esatto di ricorrenze di ogni elemento.

Usando i dati raccolti, la componente SIP crea un insieme di file scheletro (skeleton file). Questi file sono rappresentazioni modificate di alberi dei vari sottoschemi, in cui vengono introdotte istruzioni e tag speciali per rendere più facile la derivazione dell'istanza finale. Chiaramente il numero di file scheletro che deve essere prodotto è dato da tutte le possibili combinazioni dei valori delle ricorrenze stabilite, assegnati ad ogni elemento. Gli scheletri delle istanze così prodotti sono infine analizzati dalla componente FIP. FIP usa le istruzioni fornite dalla componente SIP nello scheletro, e collabora con VP per ricevere i corretti valori da associare ad ogni elemento. L'inserimento di tali valori avviene tramite un'apposita interfaccia da noi prodotta proprio per questo scopo.

Il risultato finale è un insieme di istanze che sono, per costruzione, conformi allo schema originale e classificate come sottoschemi. La componente VP ha il compito di fornire le ricorrenze stabilite di ogni elemento e i valori che devono essere assegnati ad ogni elemento durante la derivazione delle istanze finali.

Una volta terminata la generazione delle istanze da parte della componente FIP, che vengono salvate all'interno di una cartella prestabilita, l'utente, interagendo con l'interfaccia principale, ha la possibilità di prenderne visione. Infatti oltre all'interfaccia grafica è stato sviluppato un editor di testo, simile a note-pad, tramite il quale tutte le istanze prodotte possono essere aperte ed analizzate.

Capitolo 4

PRIMA DELLO SVILUPPO

Prima di iniziare con la vera propria stesura del codice è stato necessario accordarsi con il committente relativamente al linguaggio di programmazione da utilizzare e l'ambiente di sviluppo su cui lavorare. Oltre a ciò è stata necessaria anche un'abbondante documentazione su XML e XML Schema e sulle librerie Java che avrebbero agevolato e reso possibile il nostro lavoro.

4.1 IL LINGUAGGIO DI PROGRAMMAZIONE E L'AMBIENTE DI SVILUPPO

L'utilizzo di Java come linguaggio di programmazione è stata una richiesta esplicita da parte del committente.

La scelta di Java per lo sviluppo del tool TAXI è dovuta certamente al fatto che Java è un linguaggio multipiattaforma (grazie alla Java Virtual Machine), che si adatta anche alle più moderne esigenze di programmazione [38].

Inoltre, considerando che grazie a Java e alla sua JVM si riesce a scrivere dei programmi abbastanza complessi senza dipendenze da sistema operativo sottostante, questo linguaggio è stato da tempo promosso a piattaforma, ovvero una sorta di sistema operativo parallelo e alternativo [41]. Questo fornisce un ambiente di sviluppo orientato agli oggetti pulito ed efficiente, e mette a disposizione avanzate librerie per la grafica quali `javax.swing` e `java.awt` .

L'utilizzo di altri linguaggi sarebbe stato comunque scoraggiato a causa di una nostra mancanza di competenza, in quanto Java è senza dubbio il linguaggio studiato di più a livello accademico e con cui abbiamo certamente più confidenza.

L'utilizzo di Java, ha sicuramente accelerato i tempi dedicati all'implementazione della componente, benché molto impegnativa, ed ha anche senza dubbio abbreviato la fase di

documentazione e studio antecedente allo sviluppo, che sarebbe stata certamente più onerosa se ci fosse stata la necessità di imparare *ex novo* un linguaggio di programmazione.

L'ambiente di sviluppo fornito e fortemente consigliato è stato Eclipse, ideato dalla Eclipse Foundation, associazione no-profit nata per la creazione e lo sviluppo di tale piattaforma open source e per fornire un insieme di prodotti e servizi complementari per lo sviluppo software [35]. Questo ambiente ci ha agevolato molto durante lo sviluppo della nostra applicazione grazie alle sue innumerevoli funzionalità e agli strumenti per la facilitazione della stesura del codice.

Il sistema operativo su cui abbiamo lavorato è Windows. Questo perché l'applicazione finale girerà su sistema operativo Windows e perché tutte le altre componenti sono state sviluppate sotto questo stesso sistema operativo. Comunque la nostra applicazione gira correttamente sia su sistemi Windows che su sistemi Linux. In realtà Linux è stato ugualmente utilizzato, poiché, in alcune sue versioni, fornisce uno strumento, una sorta di icon-maker, che ci è stato molto utile per produrre alcune delle icone utilizzate nella nostra interfaccia. Inoltre, per la creazione di queste ci siamo fortemente ispirati a quelle abitualmente utilizzate nelle interfacce grafiche di Linux.

4.2 LE LIBRERIE JAVA E IL LINGUAGGIO XML

Il nostro lavoro, come precedentemente spiegato, consisteva nella progettazione e nell'implementazione di un'interfaccia grafica e di gran parte della sua sottostruttura. L'obiettivo di questa interfaccia è quello di visualizzare la struttura di un file XML sotto forma di albero, evidenziando le relazioni di parentela tra nodi (relazione padre – figlio) e i nodi che rivestono un ruolo particolarmente importante all'interno del documento per l'assegnamento dei pesi. Quindi il primo passo necessario è stato un approfondimento sul linguaggio XML, per acquisirne tutte le caratteristiche (requisito fondamentale per riuscire ad implementare correttamente un'applicazione che effettua il parsing di un file XML). Dopo di che abbiamo affrontato un attento studio e una dettagliata ricerca in relazione alle librerie Java che forniscono le potenzialità per la gestione, la creazione e la manipolazione di documenti XML come la libreria `java.org.w3c.*`. Per quanto riguarda invece l'implementazione dell'interfaccia vera ci siamo adeguatamente documentati relativamente alle librerie `javax.swing` e `java.awt` [45].

4.2.1 LE LIBRERIE `javax.swing` e `java.awt`

Le `javax.swing` e le `java.awt` sono librerie dal potenziale enorme, ma anche dalla grande complessità e dalla scarsa maneggevolezza [42], soprattutto per utenti profani quali siamo noi. Il fatto è dovuto principalmente alla loro struttura gerarchica molto ampia. A causa di ciò, inizialmente abbiamo dovuto dedicare non poco tempo alla documentazione e alla ricerca di materiale al fine di ottenere una comprensione sufficiente su queste librerie. Utilissima allo scopo è stata la consultazione di molti siti web che trattano di programmazione Java, come [31] e [22].

Entrambe le librerie mettono a disposizione del programmatore la possibilità di creare molti oggetti grafici quali finestre, frames, toolbar, menu a tendina, popup, il cui uso e la cui utilità verranno illustrati più dettagliatamente nei prossimi capitoli. Durante la stesura del codice, per la creazione delle componenti grafiche dell'applicazione, abbiamo fatto largo uso delle `javax.swing` poiché dotate di maggiore varietà di componenti, più sofisticate rispetto a quelle fornite da `java.awt` [22], che talvolta sono risultate comunque indispensabili. Infatti, nonostante i problemi che le `swing` presentavano in passato, come lentezza, eccessivo consumo di memoria e difetti di visualizzazione, si presentano ora come un'API robusta ed affidabile, benché piuttosto complessa. Per queste loro caratteristiche vengono utilizzate sempre di più e sempre con maggior successo in moltissime applicazioni commerciali [23].

Rispetto alle `awt`, toolkit grafico disponibile fin dalle prime versioni di Java [33], le `swing` possono considerarsi un elemento separato, anche se implementano le classi di base delle `awt`. Le `awt`, fin dai loro primi impieghi per lo sviluppo di interfacce, parvero subito insufficienti per supportare lo sviluppo di GUI più complesse, godendo anche di scarsa portabilità, in quanto si basano fortemente sulla piattaforma software di base [24]. Consistono infatti in un'interfaccia tra componenti grafici nativi del sistema operativo e le componenti GUI di Java [24].

I tre benefici maggiori delle `swing` rispetto alle `awt` sono, innanzitutto, l'indipendenza dal sottostante sistema nativo di windowing, in quanto scritte completamente in Java; l'aggiunta di nuovi componenti grafici ed una modalità di visualizzazione che può essere modificata a runtime; infine offrono una chiara distinzione tra il tipo di dato che l'oggetto visualizza (Model) e l'effettiva visualizzazione di questo (View) [25].

Un package utilizzato per la programmazione è `java.awt.event`, che contiene le classi per la gestione degli eventi necessarie per catturare e gestire "input" dagli utenti.

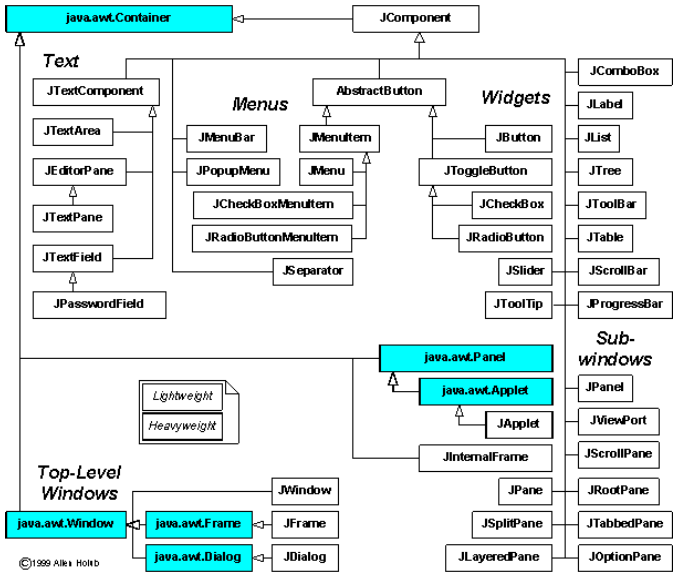


Figura 22. La gerarchia delle swing e delle awt [52]

Le classi delle `java.awt` e delle `javax.swing`, conformemente con l'organizzazione di tutte le altre librerie Java, sono organizzate in gerarchie di ereditarietà secondo il paradigma object-oriented. Possiamo osservare un diagramma rappresentante la gerarchia in Figura 22.

Prima che una componente possa essere visualizzata nell'interfaccia utente, deve essere aggiunta ad un contenitore, sottoclasse di `java.awt.Container,` ovvero una componente che ha la funzione di contenere altre componenti [25]. Tipicamente un contenitore può essere una finestra, `JWindow`, un'applet, `JApplet,` o nel caso di un programma stand-alone (come il nostro) un frame `JFrame`. Questa classe oltre a gestire i metodi di inserimento ed eliminazione di componenti, gestisce anche la disposizione dei componenti tramite metodi detti gestori di layout [23].

`Swing` offre vari tipi di gestori di layout come per esmpio il layout `FlowLayout`, il `BorderLayout`, il `GridLayout` [26].

Il gestore di layout che fornisce maggior libertà strutturale è `FlowLayout` che suddivide il contenitore in un certo numero di righe e colonne e dispone i componenti in determinate celle della griglia; i componenti possono occupare più celle e queste possono avere proporzioni diverse tra loro. Questo gestore di layout è quello leggermente più difficile da utilizzare, rispetto agli altri messi a disposizione da Java, ma anche quello che garantisce il miglior risultato grafico [25].

Per convertire un'interfaccia grafica di Java in un programma, è necessario renderla ricettiva nei confronti di eventi provocati dall'utente, come click effettuati tramite il mouse. Le classi che si occupano della gestione di eventi, sono dette *ascoltatori di eventi* (event listener), i quali creano un

oggetto "ascoltatore" e lo associano al componente dell'interfaccia che deve "ascoltare" (ad esempio un pulsante) [34]. Gli ascoltatori di eventi più importanti in Java sono:

- `ActionListener`: per gli eventi di azione di un utente su un componente;
- `AdjustmentListener`: per gli eventi di modifica, generati da modifiche apportate ai componenti;
- `FocusListener`: eventi di selezione, avvengono quando componenti guadagnano o perdono la selezione;
- `ItemListener`: eventi di elemento, generati dalla modifica di elementi di componenti;
- `KeyListener`: eventi della tastiera, quando un utente preme un tasto;
- `MouseListener`: eventi del mouse, generati dal clic o dallo spostamento del mouse.

Ogni tipo di ascoltatore ha i propri metodi per gestire gli eventi; nel nostro caso gli `ActionListener` e i `MouseListener` sono stati i più utilizzati poiché gli eventi che l'utente può generare sono esclusivamente legati a movimenti e pressioni dei pulsanti del mouse (per esempio l'apertura di un menù popup).

I componenti di maggior importanza che sono stati utilizzati nello sviluppo della nostra GUI, sono:

- pannelli (`JPanel`);
- icone (`ImageIcon`);
- pulsanti (`JButton`);
- etichette (`JLabel`);
- aree di testo (`JTextPane` e `JTextArea`);
- campi di testo (`JTextField`);
- elenchi a discesa (`JComboBox`);
- finestre di dialogo (`JOptionPane`);
- barre di scorrimento (`JScrollBar`);

- riquadri a scorrimento (`JScrollPane`);
- barre degli strumenti (`JToolBar`);
- menù (`JFileMenu`);
- menù popup (`JPopupMenu`);
- alberi (`JTree`);
- tabella (`JTable`).

4.2.2 LA LIBRERIA `org.w3c.dom.*`

Questa libreria fornisce una serie di classi che mettono a disposizione un insieme di comandi standard, una serie di metodi e proprietà per il documento XML ed i suoi discendenti [3].

DOM [27], [28] è stato preferito al modello SAX [29] in quanto più idoneo alle nostre esigenze di utilizzare un'interfaccia object-based e non event-based. Infatti tramite l'approccio object-based il parser mantiene in memoria un albero che contiene tutti gli elementi del documento XML, mentre con l'approccio event-based di SAX un evento viene generato ogni qualvolta il parser incontra un elemento, un attributo, o del testo durante la lettura del documento XML [32]. Inoltre per conformità con le altre applicazioni preesistenti relative al tool TAXI, tutte utilizzanti DOM, l'impiego di SAX sarebbe stato comunque poco conveniente.

Con DOM ogni elemento del documento XML diventa un nodo dell'albero rappresentato dalla classe Node che comprende qualsiasi elemento dell'albero XML: radice, ramo, foglia.

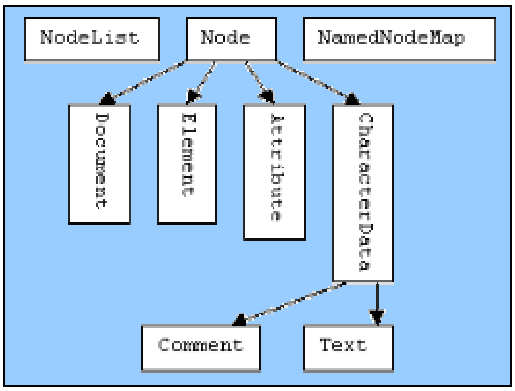


Figura 23. La struttura dell'interfaccia DOM [30]

Allo stesso livello della classe `Node`, ma senza figli, troviamo le classi `NodeList` e `NamedNodeMap`, come possiamo vedere in

Figura 23. Si tratta di oggetti temporanei che vengono generati durante la lettura dell'albero XML, per memorizzare dei gruppi di fratelli (ad esempio tutti i figli o tutti gli attributi di un elemento) [3]. DOM fornisce inoltre metodi per esplorare o modificare l'albero e per conoscere le proprietà di ogni nodo.

L'uso fondamentale che noi facciamo di tale libreria è relativo al parsing di un file XML. Un parser XML è un modulo software che si colloca tra l'applicazione e il documento XML, che permette all'applicazione di accedere al contenuto e alla struttura del documento XML. Esistono due tipi di parser: validanti e non validanti. Per il nostro scopo si è rivelato necessario un parser non-validante, che controllasse quindi solo che il documento fosse ben formato, cioè che ogni elemento fosse racchiuso tra due tag (uno di apertura e uno di chiusura). Non è stato possibile fare uso di un parser validante in quanto dovendo talvolta visualizzare file XML pesati, e quindi contenenti elementi non appartenenti allo schema, saremmo incappati in inevitabili situazioni di errore. Il parser, al termine della computazione, restituisce un oggetto che può essere letto, modificato e utilizzato nel codice del programma.

Capitolo 5

CONTROL CONSOLE

In questo capitolo descriveremo in dettaglio il processo di sviluppo dell'interfaccia grafica, *Control Console*, in ogni sua parte e componente.

Lo sviluppo dell'interfaccia grafica per il tool TAXI si articola su più packages che sono elencati di seguito corredati di una breve descrizione.

- `TAXInterface.file`, contenente le classi per il parsing dei file XML per la loro visualizzazione nell'interfaccia;
- `TAXInterface.graphics`, contenente il main e le classi che determinano la struttura base dell'interfaccia;
- `TAXInterface.menu`, contenente le classi che implementano i vari menù a tendina e le barre degli strumenti;
- `TAXInterface.table`, contenente le classi che implementano la `JTreeTable`, tabella all'interno di cui deve essere inserita la struttura ad albero del file XML parserizzato, e tutte le sue funzioni.

In questo capitolo ci dedicheremo soltanto alla parte grafica del progetto; vedremo più in dettaglio le varie classi che compongono la GUI da noi sviluppata e descriveremo più in dettaglio i package elencati.

5.1 IL PACKAGE `TAXInterface.table`

Questo package è quello contenente tutte le classi di utilità per l'implementazione del cuore dell'interfaccia, cioè una tabella che sia in grado di mantenere su ognuna delle sue righe un nodo di un albero DOM, con a fianco un campo di testo per l'inserimento del peso per quel nodo, ove consentito.

5.1.1 LA CLASSE `JTreeTable.java`

La classe `JTreeTable.java` estende la classe `JTable.java`, utile per la creazione di una tabella dotata di un numero definito di righe e colonne [43].

Nel costruttore viene impostato l'header e il colore dello sfondo della tabella. Nel codice relativo a questa classe viene inserita la costruzione di `JPopup`, menù che compaiono cliccando col tasto destro del mouse, con tutti i suoi menù e sottomenù. Il popup è composto di un menu a tendina con varie opzioni come la open, la save, la expand, la collapse, il refresh e opzioni per cambiare il colore di sfondo della tabella, come si può vedere in Figura 24. Ad ogni opzione del menù popup viene ovviamente associato un `ActionListener` che tramite il metodo `actionPerformed()`, associa un'azione da eseguire in seguito alla pressione di una certa opzione.

Figura 24. Il menù popup

Oltre all'`ActionListener` vi è anche un `MouseListener` che tramite il metodo `mousePressed()` ascolta quando viene premuto il pulsante destro del mouse ed esegue la giusta operazione alla sua pressione. In questo caso viene fatto comparire il popup nell'interfaccia. All'interno del corpo di `MouseListener` vi è anche il metodo `mouseReleased()` che contiene il codice da eseguire quando il tasto del mouse viene rilasciato. Nel nostro caso vogliamo che il menù resti visibile perciò i metodi `mousePressed()` e `mouseReleased()` saranno identici.

Dopo di che viene creato un oggetto di tipo `TreeTableCellRenderer` che è l'oggetto che si occupa di visualizzare l'albero all'interno della `JTreeTable`. Infatti in `JTreeTable` avviene la fusione tra l'albero (`JTree`) e la tabella (`JTable`) tramite il metodo `setSelectionModel()` che forza la condivisione delle righe tra `JTable` e `JTree`.

Poi si imposta l'altezza delle righe in modo che sia uguale sia per `JTree` che per `JTable` e che non ci sia spazio tra ogni riga. Lo sfondo di `JTreeTable` viene impostato in modo che sia di colore uniforme tra l'albero e la table.

Il metodo `getTableCellRendererComponent()` è quello che fornisce il renderer per le celle della `JTable` e qui impostiamo il colore di una cella quando viene selezionata con il mouse.

5.1.1.1 I METODI E LE CLASSI CONTENUTI IN `JTreeTable.java`

- `editingStopped();`
- `getEditingRow();`
- `getScrollableTracksViewportHeight();`
- `class TreeTableCellRenderer;`
  - `setBounds();`
  - `paint();`
  - `getTableCellRendererComponent().`
- `class XmlStringRenderer;`
  - `setValue().`
- `class TreeTableCellEditor;`
  - `getTableCellEditorComponent().`

Delle classi contenute in `JTreeTable` abbiamo scelto di commentare di seguito quella dalla maggiore importanza.

5.1.1.2 LA CLASSE `XmlStringRenderer.java`

La classe `XmlStringRenderer.java` estende la classe `DefaultTableCellRenderer.java` e rappresenta il renderer utilizzato per le stringhe nella tabella, cioè stabilisce che valore visualizzare accanto ad ogni nodo. Nelle celle relative a nodi che non hanno peso, e a cui non ne può essere assegnato uno, viene impostato il carattere '-', altrimenti il peso del nodo. Questa azione avviene tramite il metodo `setValueAt()`.

5.1.2 LA CLASSE `AbstractCellEditor.java`

La classe `AbstractCellEditor` implementa la classe astratta `CellEditor` e fornisce vari metodi per inserire un valore all'interno delle celle.

5.1.2.1 I METODI DELLA CLASSE `AbstractCellEditor.java`

- `getCellEditorValue();`
- `isCellEditable();`
- `shouldSelectCell();`
- `stopCellEditing();`
- `cancelCellEditing();`
- `addCellEditorListener();`
- `removeCellEditorListener();`
- `fireEditingStopped();`
- `fireEditingCanceled();`

Il metodo `isCellEditable()` restituisce un valore booleano per indicare che la cella è editabile. Il metodo `shouldSelectCell()` restituisce false per indicare che la cella non può essere selezionata. Altri metodi abbastanza importanti sono `stopCellEditing()`, che comunica all'editor di terminare l'editing e di accettare qualunque valore inserito dall'utente come valore dell'editor; `cancelCellEditing()`, che comunica all'editor di non accettare alcun valore editato; `addCellEditorListener()`, che aggiunge un listener alla lista dei listener, che consente di capire quando l'editor termina oppure annulla l'inserimento di un valore; `fireEditingStopped()` e `fireEditingCanceled()`, che informano tutti gli ascoltatori interessati a sapere quando si verifica un determinato tipo di evento.

5.1.3 LA CLASSE AbstractTreeTableModel.java

La classe `AbstractTreeTableModel.java` implementa l'interfaccia `TreeTableModel.java` che ha lo scopo di fornire le firme dei metodi che una classe, rappresentante un modello per la creazione di una tabella, deve avere. Infatti `AbstractTreeTableModel` fornisce l'implementazione di tutti i metodi necessari per la gestione di una tabella contenente un albero al suo interno. In realtà l'implementazione corretta ed efficiente dei metodi che risulteranno utili è fornita nella superclasse `ParserXmlModel.java` contenuta nel package `TAXInterface.file`.

5.1.4 LA CLASSE MarkableTreeCellRenderer.java

La classe `MarkableTreeCellRenderer.java` estende la classe `DefaultTreeCellRenderer.java` che si occupa del rendering delle celle dell'albero. È quindi tramite questa classe e i suoi metodi che è possibile modificare i nodi dell'albero aggiungendovi delle icone oppure modificandone il colore dello sfondo e della stampa.

5.1.4.1 I METODI DELLA CLASSE MarkableTreeCellRenderer.java

- `getTreeCellRendererComponent();`
- `getPreferredSize();`
- `paintComponent().`

5.1.4.2 IL METODO `getTreeCellRendererComponent()`

Grazie a questo importante metodo, è possibile modificare tutte le celle dell'albero `JTree` della `JTreeTable`. Dentro il corpo del metodo vengono impostate tutte le icone di ogni singolo nodo, sostituendole a quelle presenti di default. Ciò è piuttosto importante per diversificare i nodi esistenti e per evidenziare quelli dalla maggior importanza. Per cambiare le icone di default abbiamo utilizzato il metodo `setIcon(new ImageIcon("nomeIcona"));`. Oltre a diversificare i nodi tramite l'utilizzo di icone diverse abbiamo evidenziato tramite i colori i nodi più importanti come i figli dei nodi choice, evidenziati in rosso, per dar loro il giusto rilievo all'interno dell'interfaccia, come si vede in Figura 25.

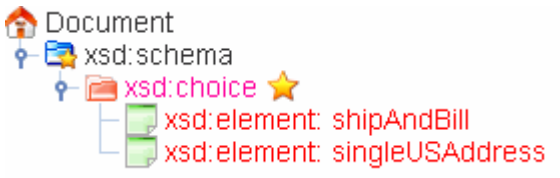


Figura 25. L'evidenziazione dei nodi tramite colori e icone

Tramite il metodo `getPreferredSize()` è possibile impostare una dimensione standard da utilizzare per le icone dell'albero, in modo che abbiano tutte la solita grandezza.

5.1.4.3 IL METODO `paintComponent()`

L'ultimo metodo della classe `MarkableTreeCellRenderer` è `paintComponent()` che crea uno spazio (per le icone), in ogni cella dell'albero, della grandezza esattamente equivalente a quella dell'icona. In più personalizza il font nelle celle dei nodi, il colore del bordo e lo sfondo del rettangolo di ogni nodo quando questo viene selezionato.

5.1.5 LA CLASSE `TreeTableModelAdapter.java`

`TreeTableModelAdapter.java` estende la classe `AbstractTableModel.java` ed ha un costruttore in cui viene creato un oggetto `MarkableTreeCellRenderer` col quale si imposta il renderer per le celle dell'albero e viene aggiunto un listener all'albero per ascoltare quando avviene un evento espansione o collassamento. Il suo scopo è quello di adattare un modello di una semplice tabella in quello di una contenente un albero al suo interno.

5.1.5.1 I METODI DI `TreeTableModelAdapter.java`

- `getColumnCount();`
- `getColumnName();`
- `getColumnClass();`
- `getRowCount();`
- `nodeForRow();`
- `getValueAt();`
- `isCellEditable();`
- `setValueAt().`

5.1.5.2 I METODI `setValueAt()` e `getValueAt()`

Il metodo `getValueAt()`, che prende come parametri due interi, di cui il primo rappresentante una riga ed il secondo una colonna della tabella, è utile per poter recuperare un valore contenuto in una determinata casella. Di maggior importanza è senza dubbio il metodo `setValueAt()` che inserisce l'oggetto passato come parametro nella casella indicata dalle coordinate passate anch'esse come parametro. Questo è ovviamente utile per esempio per impostare un peso nella giusta posizione nella tabella.

5.1.6 L'INTERFACCIA `TreeTableModel.java`

`TreeTableModel` è un'interfaccia che estende `TreeModel.java` e che fornisce i metodi per recuperare informazioni sull'insieme di colonne che ogni nodo in `TreeTableModel` deve avere.

5.2 IL PACKAGE `TAXInterface.file`

Il package `TAXInterface.file` contiene tutte le classi che occorrono per poter effettuare il parsing di un file XML.

5.2.1 LA CLASSE `FileNode.java`

La struttura di un file XML impone che ogni nodo sia separato dall'altro da un terminatore di riga "`\n`" o comunque da un nodo contenente testo qualunque. Tale terminatore, al momento del parsing del file, viene visto come un nodo di testo dal parser, quindi come un nodo da trattare come tutti gli altri. Una richiesta fondamentale, effettuata dal committente, è stata quella di non mostrare, all'interno dell'interfaccia, tali nodi, per non appesantire la visualizzazione per l'utente.

A tale scopo è stata introdotta la classe `FileNode.java`, che incapsula al suo interno un nodo DOM e fornisce i metodi per lavorare con esso, in conformità con le esigenze espresse dal committente. Questa classe ha un ruolo fondamentale all'interno del funzionamento generale del sistema, infatti è questa che stabilisce quali siano i nodi da visualizzare all'interno dell'interfaccia e quelli di rilievo che vanno evidenziati nella struttura complessiva dell'albero.

5.2.1.1 I METODI DELLA CLASSE `FileNode.java`

- `hashCode();`
- `getDom();`

- getNodePrefix();
- getChildren();
- setPeso();
- update();
- getPeso();
- toString();
- stringNodeType();
- stringNodeName();
- StringNodeValue();
- index();
- child();
- getParentNode();
- hasParentNode();
- getPreviousSibling();
- getNextSibling();
- getFirstChild();
- hasFirstChild();
- getLastChild();
- childCount();
- hasPreviousSibling();
- hasNextSibling();
- isFirstLevelElement();
- isChoice();
- equals()
- hasIstiChild();
- getIstiVal();
- hasCommentChild();

- `hasChoiceInPath();`
- `isElementsList()`.

Forniamo una breve descrizione dei metodi della classe che hanno un'importanza particolare.

5.2.1.2 IL METODO `getChildren()`

Il metodo `getChildren()` è il metodo che consente di scegliere i nodi da mostrare a video, che quindi ci consente di non visualizzare all'interno della `JTreeTable` i nodi di tipo `Text` o `Comment`. Per quanto riguarda l'implementazione vera e propria, per prima cosa vengono salvati in una struttura dati di tipo `NodeList` tutti i figli del nodo DOM, variabile istanza di `FileNode`, su cui questo metodo è stato applicato.

A seconda che il metodo sia invocato su un nodo di tipo `Document`, nodo fittizio che rappresenta l'origine del documento, o su un nodo di qualunque altro tipo, vengono seguiti due processi molto diversi. Se il nodo è di tipo `Document` non avrà figli di tipo `Text` (in quanto nodo fittizio usato come root del documento), quindi ci dovremo occupare solo dell'eliminazione dei nodi di tipo `Comment`, che generalmente sono posti all'inizio del file. Così, una volta ottenuta la struttura contenente tutti i figli del nodo DOM incapsulato, calcoliamo quale sarà la dimensione effettiva dell'array di nodi, che il metodo dovrà restituire, contando il numero di nodi di tipo `Comment` che dovremo eliminare e sottraendoli dal numero totale dei figli.

Una volta calcolato questo numero creiamo un array di questa dimensione e vi copiamo all'interno tutti i nodi figli del nodo DOM ad esclusione di quelli di tipo `Comment` e restituiamo l'array. Se invece il nodo non è di tipo `Document` ma di qualunque altro tipo, sapendo che se un nodo con `n` figli ha esattamente anche `n+1` figli di tipo `Text`, e che i figli di tipo `Text` sono posti in alternanza con i veri figli, si esegue un ciclo che legge un nodo ogni due, saltando quelli non utili (cioè quelli di tipo `Text` o `Comment`) e memorizzando gli utili in un vettore.

Nel caso in cui si stia lavorando con un file che è già stato aperto e salvato dall'utente e il nodo incontrato sia figlio di un nodo di tipo `choice`, è possibile che come ultimo figlio non di tipo `Text` abbia un nodo di tipo `ISTI_et_CNR_pesi` ed infine un figlio di tipo `Text`. Essendo i figli di tipo `ISTI_et_CNR_pesi` introdotti dal programmatore, per il salvataggio del peso assegnato dall'utente al nodo, non devono poter essere visualizzati in quanto l'utente non deve essere a

conoscenza della loro esistenza. Quindi, se il nodo ha tali figli, gli ultimi due nodi contenuti nella lista vengono scartati. Nel caso in cui il vettore temporaneo di figli da restituire contenesse anche figli di tipo `Comment` questi verranno eliminati. Infine si copia il contenuto di questo vettore di appoggio in un array e lo si restituisce.

5.2.1.3 I METODI `isChoice()` E `isElementsList()`

Altri metodi piuttosto importanti sono i metodi `isChoice()` che stabilisce se un nodo è di tipo `choice` in base al suo nome e `isElementsList()` che verifica se il nome di un nodo è "`ISTI_et_CNR_first_LEVEL_element`". Benché molto semplici, questi due metodi sono quelli che nella visualizzazione dell'albero all'interno dell'interfaccia fanno si che determinati nodi vengano scelti per poter assegnare loro un peso o per poter dar loro il giusto rilievo, evidenziandoli.

5.2.1.4 IL METODO `setPeso()`

Il metodo `setPeso()` viene richiamato ogni qualvolta l'utente inserisce un nuovo peso. Ogni volta che viene inserito un nuovo peso per un nodo, anche i pesi dei suoi fratelli devono essere aggiornati. Infatti, dovendo la somma di tutti i nodi figli di uno stesso padre essere pari ad 1, ogni volta che un peso di un nodo varia, varieranno anche i pesi dei suoi fratelli, in modo tale da poter mantenere la loro somma pari ad 1. Nel caso di un file che viene caricato e pesato per la prima volta tutti i pesi saranno impostati ad un valore di default, calcolato secondo la (2.1), altrimenti i pesi vengono prelevati direttamente dal file.

Questi valori verranno memorizzati in tabelle apposite per il recupero degli stessi. Ogni volta che un nodo viene modificato si memorizza il suo vecchio peso. Questa operazione è necessaria in quanto, se il nuovo peso inserito non dovesse essere un input corretto, il vecchio valore verrebbe ripristinato. In realtà questo metodo effettua dei controlli per la verifica dell'input, la vera gestione dei pesi e della loro normalizzazione a 1 è delegata al metodo `update()` richiamato all'interno di `setPeso()`. Un esempio di input non corretto è un valore uguale ad 1. In questo caso si stampa un messaggio di errore per l'utente che lo informa della possibilità di inserire solo valori strettamente minori di 1. Inoltre, come richiesto dal committente, i pesi possono avere un massimo di 4 cifre

decimali. Quindi, anche nel caso in cui l'utente inserisse un peso con più cifre decimali del previsto, verrebbe stampato un messaggio di errore e il vecchio valore sarebbe ripristinato.

Messaggi di errore vengono stampati anche nel caso in cui un utente inserisca peso strettamente maggiore di 1 oppure uguale a 0, ad esclusione dei figli di un elemento di primo livello, a cui, come sappiamo, possono essere assegnati anche pesi uguali a 0 e ad 1. Gli unici valori accettati sono tutti i valori compresi tra 0 e 1. Questi valori vengono poi elaborati all'interno del metodo `update()`, che prende come parametri il nodo a cui è stato appena modificato il peso e il peso stesso.

5.2.1.5 IL METODO `update()`

Il metodo `update()` applica una verifica più approfondita sulla validità del peso, nel senso che deve andare a verificare se il nuovo peso inserito, oltre che appartenere al range consentito di valori, sommato ai pesi degli altri nodi modificati non dia un valore strettamente maggiore di 1. Quindi il primo passo eseguito all'interno del metodo è quello di reperire i pesi dei fratelli del nodo in causa, verificare quali di essi sono già stati modificati, in base al loro stato, memorizzato all'interno di una mappa di flag, ed effettuare la somma dei loro pesi. Si lavora con interi per evitare errori di arrotondamento quindi i pesi, avendo al massimo quattro cifre decimali, devono essere moltiplicati tutti per 10000.

Una volta fatto ciò si calcola il numero di nodi non ancora pesati dall'utente, che continuano a mantenere un valore di default. Se il numero di questi nodi è inferiore al numero totale dei fratelli si verifica che la somma calcolata sia minore di 1. Se lo è l'aggiornamento va a buon fine, quindi i pesi dei nodi che continuano a mantenere un valore di default verranno aggiornati, assegnando loro un peso calcolato secondo la (2.2). Se la somma calcolata invece risulta maggiore o uguale a 1 il vecchio valore viene ripristinato e si stampa all'utente un messaggio per informarlo dell'errore.

La somma può risultare uguale a uno solo nel caso in cui il numero di nodi non ancora modificati sia pari a 0, perché in caso contrario ad uno o più nodi dovrebbe essere assegnato un peso uguale a 0, ma ciò non è possibile poiché, come precedentemente descritto, questa situazione è ammissibile solo nel caso in cui il nodo sia figlio di un elemento di primo livello.

5.2.1.6 IL METODO `getPeso()`

Infine un altro metodo di fondamentale importanza è `getPeso()`. Questo è un metodo molto semplice, che restituisce il peso relativo ad un nodo, ma di grande importanza in quanto viene invocato ogni qualvolta un peso deve essere visualizzato nell'interfaccia.

5.2.1.7 IL METODO `toString()`

Questa classe fornisce anche altri metodi di ovvia utilità, come il `toString()`, che stabilisce quali informazioni stampare, relativamente ad un nodo, in ogni cella della tabella. In questo caso è stato deciso di stampare il tipo del nodo ad esempio `Element` o `ComplexType` e, se esiste, il nome associato a quel nodo.

5.2.2 LA CLASSE `ParserXmlModel.java`

Per costruire un oggetto di tipo `JTreeTable` è necessario passare al costruttore una sorta di modello che consenta all'oggetto di capire come visualizzare determinate informazioni all'interno dell'interfaccia. La classe che implementa tale modello è la classe `ParserXmlModel.java` che estende la classe astratta `AbstractTreeTableModel.java`, contenuta nel package `TAXInterface.table`, ed implementa l'interfaccia `TreeTableModel.java`. Nel costruttore di questa classe viene invocato il costruttore della superclasse passando come parametro un nuovo oggetto di tipo `FileNode` costruito a partire dalla radice del documento che verrà visualizzato nell'interfaccia. È fondamentale passare come parametro un `FileNode` perché, in questo modo, vengono sfruttati tutti i metodi che sono implementati al suo interno. Per esempio, all'interno del metodo `getChildren()` di `ParserXmlModel`, che prende come parametro un oggetto di tipo `FileNode`, viene invocato il rispettivo metodo di `FileNode` sul nodo passato come parametro.

5.2.2.1 I METODI DELLA CLASSE `ParserXmlModel.java`

- `getChildCount();`
- `getChild();`
- `getColumnCount();`
- `getColumnName();`
- `getColumnClass();`
- `getValueAt();`
- `isCellEditable();`
- `setValueAt();`
- `getPathToRoot();`
- `isChoiceChild().`

Questa classe fornisce tutti i metodi per la modellazione di una `JTreeTable`. Segue una breve descrizione.

5.2.2.2 I METODI `getColumnCount()`, `getColumnName()`, `getColumnClass()`

Il metodo `getColumnCount()` restituisce il numero di colonne presenti nella tabella (in questo caso 2).

Il metodo `getColumnName()` restituisce il nome della colonna il cu indice è passato come parametro.

Il metodo `getColumnClass()` restituisce il tipo del dato che sarà memorizzato all'interno della colonna di indice passato come parametro. Questo è il metodo più importante tra i tre poiché è quello che consente di non accettare valori inseriti in quella colonna dal formato errato.

5.2.2.3 IL METODO `isChoiceChild()`

Il metodo `isChoiceChild()` prende come parametro un oggetto di tipo `FileNode` ed invoca su di esso il metodo `getParentNode()` per ottenerne il padre, per poi invocare su di esso il metodo `isChoice()`. Se il padre è una `choice` verrà restituito `true`, `false` altrimenti. Questo metodo è molto importante in quanto, nell'interfaccia, solo i nodi figli di un nodo `choice` (o di un elemento di primo livello, che, come già detto, viene trattato come fosse una `choice`) devono avere un campo per l'inserimento dei pesi. Questo viene utilizzato all'interno del metodo `getValueAt()` per sapere quando deve restituire un valore diverso da `null`, e ciò accade solo nel caso in cui si abbia a che fare con figli di `choice` (o di elementi di primo livello).

5.2.2.4 IL METODO `getValueAt()`

Il metodo `getValueAt()` è il metodo che si occupa di invocare il metodo `getPeso()` sul `FileNode` passato come parametro. Questo metodo restituisce un valore diverso da `null`, cioè un oggetto di tipo `Double` rappresentante il peso del nodo, solo nel caso in cui il nodo passato come parametro sia figlio di un nodo `choice`. Il valore restituito da questo metodo è quello che verrà visualizzato nell'interfaccia come peso del relativo nodo passato come parametro.

5.2.2.5 IL METODO `setValueAt()`

Il metodo `setValueAt()` è quello che si occupa di impostare in una determinata posizione all'interno della tabella (corrispondente a un nodo e a un numero di colonna) un valore passato come parametro, quindi è da questo metodo che dipende la corretta visualizzazione nell'interfaccia dei nodi con i loro relativi pesi.

5.2.2.6 IL METODO `isCellEditable()`

Un altro metodo molto importante, benché dall'implementazione piuttosto semplice, è il metodo `isCellEditable()`. Questo restituisce `true` se nella cella della tabella contenuta nella colonna di indice passato come parametro, è possibile inserire un valore oppure se gli eventi del mouse possono essere catturati correttamente. Infatti se la colonna ha indice 0, è quella contenente l'albero, ed è obbligatorio restituire `true`. Se fosse altrimenti, cliccando sui nodi dell'albero, non si verificherebbe nessun evento quindi non potrebbero né essere espansi né essere collassati. Nel caso in cui la colonna sia quella di indice 1, cioè quella contenente i pesi si restituisce una variabile, `CAN_EDIT`, che viene impostata a `true` ogni volta che l'utente desidera aprire una sessione di modifica dei pesi tramite l'apposito pulsante nell'interfaccia e a `false` ogni volta che viene terminata.

5.3 IL PACKAGE `TAXInterface.graphics`

Questo package contiene tutte le classi che si occupano della creazione delle varie componenti grafiche, come menù, barre degli strumenti, messaggi di errore, che possiamo trovare all'interno della nostra interfaccia.

5.3.1 LA CLASSE `Alert.java`

La classe `Alert.java` è stata implementata per la necessità di creare messaggi di errore o avvertimento personalizzati da mostrare a video, con icona e stringa per informare l'utente di un evento avvenuto o di un evento fallito. Ne vediamo un esempio in Figura 26.

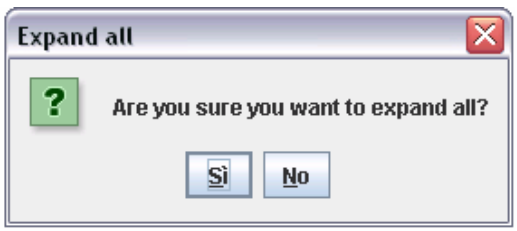


Figura 26. Un esempio di warning

5.3.2 LA CLASSE `InterfaceMain.java`

`InterfaceMain` è la classe che contiene il main dell'interfaccia da noi prodotta. Nel suo corpo inizialmente viene creato uno splashscreen, immagine che solitamente viene visualizzata in attesa dell'avvio di programmi [36], tramite una `JWindow`, cioè una finestra senza bordi, con il titolo TAXI a caratteri dinamici (ruotanti), come possiamo vedere in Figura 27. Questa serve da presentazione per il programma che sta per essere avviato.

Figura 27. Lo splashscreen

Inoltre è qui che viene impostato il look & feel (l'aspetto grafico di default) dell'intera applicazione tramite la classe `UIManager` e il suo metodo `setLookAndFeel()`. Tra i vari look possibili noi abbiamo scelto il `MetalLookAndFeel` di `javax.swing.plaf.metal` che a impatto visivo si presenta piuttosto gradevole. Nel caso in cui il `MetalLookAndFeel` non sia supportato dal sistema su cui sta girando l'applicazione questa verrà avviata con il look and feel di sistema. Una volta impostato il look and feel viene creato un nuovo oggetto di tipo `TreeTableExample()` già opportunamente descritto in precedenza.

Nella classe `InterfaceMain` è contenuto anche un metodo, `centerWin()`, usato per centrare lo splashscreen rispetto allo schermo su cui viene visualizzato.

5.3.3 LA CLASSE `SuperFrame.java`

La classe `SuperFrame.java` estende la classe `JFrame.java` e implementa `Printable.java`. Ciò è stato necessario in quanto il nostro scopo era quello di ottenere un comune

frame (e lo abbiamo ottenuto estendendo `JFrame`) che avesse però la possibilità di essere stampato (ottenuto estendendo `Printable`).

L'unico metodo messo a disposizione dall'interfaccia `Printable` è `print()` che da la possibilità di stampare, tramite una stampante, la pagina all'indice specificato, nello specifico contesto grafico, nel formato specificato (tutti dati passati come parametro).

`SuperFrame` ha un unico costruttore ereditato da `JFrame` che prende come parametro una stringa che sarà il titolo stampato sul bordo della finestra.

5.3.4 LA CLASSE `TreeTableExample.java`

La classe `TreeTableExample.java` rappresenta il nucleo dell'intera interfaccia grafica.

All'interno del costruttore, che è privo di parametri, ci occupiamo di creare il frame in cui dovrà essere inserita la visualizzazione ad albero del documento XML scelto dall'utente, inserendo al suo interno tutte le barre degli strumenti e i menù previsti.

Le implementazioni delle barre degli strumenti a nostra disposizione sono contenute nelle classi `ToolBarNorth.java`, `ToolBarSouth.java` e `ToolBarEast.java`, i cui nomi indicano la loro posizione all'interno dell'interfaccia. Queste vengono create tramite l'apposito costruttore e vengono inserite opportunamente all'interno del frame, così come per la barra dei menù. Inoltre viene creato un `JScrollPane` (un pannello munito di scroll per lo scorrimento) a cui passiamo come parametro una treetable (cioè quella che sarà visualizzata al suo interno) e lo andiamo ad aggiungere nel centro del pannello principale del frame.

All'interno di questo costruttore ci occupiamo anche di implementare una metodologia per richiedere all'utente di scegliere un file XML da file system. All'interno di un ciclo while, che termina quando l'utente ha scelto un file non nullo, viene creata una finestra di dialogo per avvertire l'utente, che per iniziare una nuova sessione è necessario aprire un file. A questo punto viene fornita all'utente un'altra finestra (`FileDialog`) per poter scegliere un file XML da file system. Questo meccanismo consente all'utente di visualizzare soltanto i file con estensione `.xml` per facilitare la scelta del documento e di navigare il file system in ogni sua cartella. Fintanto che non esiste il file scelto dall'utente oppure il path è nullo, vengono stampati degli alert opportuni e si chiede nuovamente la scelta del file. Dopo aver ottenuto il file su cui lavorare ci occupiamo di ricavare da questo un oggetto di tipo `Document` tramite il codice:

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = null;

try {
        builder = factory.newDocumentBuilder();
} catch (ParserConfigurationException e) {
        e.printStackTrace();
}

document = null;

try {
        files = new File(path);
        document = builder.parse(files);
} catch (SAXException e) {
        e.printStackTrace();
} catch (IOException e) {
        e.printStackTrace();
}
```

Figura 28. Come ottenere un albero DOM

A questo punto è necessario analizzare il documento tramite il metodo `normalizeDocument()`, che aggiunge, se necessario, un elemento di primo livello, figlio dello schema, che raggruppa sotto di se tutti i nodi di tipo `Element` figli dello schema. Inoltre vengono inizializzate tutte le mappe necessarie durante la computazione con gli opportuni valori di default o non.

Un altro punto cruciale di questa classe è la creazione del nucleo dell'interfaccia cioè la tabella vera e propria, che viene inserita all'interno del `JScrollPane` al centro del frame.

5.3.4.1 I METODI DELLA CLASSE TreeTablExample.java

- `setCentered();`
- `getActualDimFrame();`
- `getActualDimPanel();`
- `getLoc();`
- `setLoc();`
- `resize();`
- `normalizeDocument();`

- `changeTable()`.

5.3.4.2 IL METODO `normalizeDocument()`

Questo metodo inserisce all'interno dell'oggetto di tipo Document passato come parametro un elemento di primo livello che raggruppa i figli di tipo `Element` (se ne esiste più di uno) dello `schema` e li acquisisce come suoi figli. Questo a sua volta viene aggiunto come figlio di `schema`. Il parametro passato al metodo è di tipo `Node` e rappresenta la radice del documento da modificare. L'operazione svolta da questo metodo è ampiamente descritta nella sez. 2.1 e illustrata nella Figura 7 e nella Figura 8.

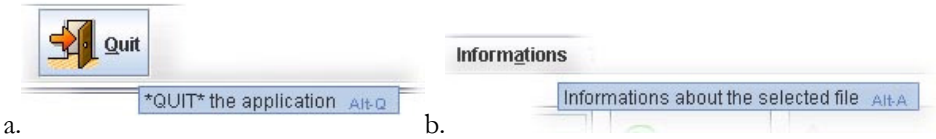
5.3.4.3 IL METODO `changeTable()`

Questo metodo è quello che permette di aggiornare la tabella dove viene visualizzato il documento. Ciò è necessario poiché ogni volta che l'utente sceglie un nuovo file deve essere creata una nuova tabella all'interno della quale sarà possibile inserire il nuovo file.

`changeTable()` ha un solo parametro di tipo String che rappresenta il nome del nuovo file da visualizzare nella `JTreeTable`.

5.4 IL PACKAGE `TAXInterface.menu`

In questi paragrafi illustreremo la costruzione dei menù e delle barre degli strumenti, cioè barre che mettono a disposizione funzionalità per l'utente [39], utilizzate nell'interfaccia sviluppata. Come in ogni interfaccia, ogni pulsante e menù è stato dotato di un tooltip, cioè una stringa che appare sopra una certa componente ogni qualvolta si passi con il puntatore del mouse sopra di essa [37], e di una combinazione di tasti in grado di attivare l'operazione richiesta senza necessariamente dover fare uso del mouse. Possiamo vedere un esempio di quanto detto in figura.



a. b.

Figura 29. Tooltip e combinazioni mnemoniche di tasti

Le stringhe evidenziate in blu rappresentano i cosiddetti tooltip mentre le lettere sottolineate nei titoletti dei pulsanti, Figura 29.a, o dei menù, Figura 29.b, rappresentano le lettere di scelta rapida. All'interno del tooltip compare anche la combinazione dei tasti per l'attivazione di una determinata componenente, ALT-lettera sottolineata.

Nel caso di menù dotati di sottomenù, anche ad ognuno di questi viene associato il proprio tooltip e una combinazione mnemonica di tasti. Ogni sottomenù implementato estende la classe `JMenuBar` [46] di `javax.swing`, mentre ognuna delle barre degli strumenti implementate estende la classe `JToolBar` [44] anch'essa di `javax.swing`.

5.4.1 LA CLASSE `FileMenu.java`

La classe `FileMenu.java` implementa il primo menù a discesa contenuto nella barra dei menù dell'interfaccia. La combinazione di tasti associata a tale opzione è ALT-F.

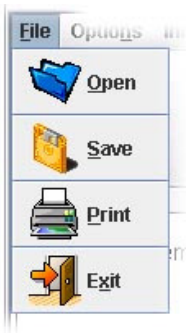


Figura 30. Il menù a discesa *File*

All'interno del costruttore vengono creati tutti i sottomenù di tipo `JMenuItem` da inserire all'interno del menù *File*, che possiamo vedere in Figura 30.

La prima sotto-opzione è l'opzione *Open* per caricare un file da file system. Ogni volta che questa opzione viene scelta viene mostrato un `FileDialog` che consente di navigare all'interno

del file system per la scelta del file desiderato. Se il file scelto esiste ed è diverso da null la tabella viene aggiornata mostrando il nuovo file, altrimenti si continua a visualizzare il file precedente.

La sotto-opzione *Save* viene utilizzata per salvare su di un nuovo file XML il file pesato correntemente aperto. Questa opzione viene lasciata disabilitata, fino a quando non viene aperto un file dall'utente, per evitare situazioni di errore nel caso in cui un utente voglia salvare un file nullo.

La sotto-opzione *Print* viene utilizzata, invece, per inviare alla stampante il documento da stampare. Per fare ciò all'interno dell'`actionPerformed()` relativo a tale pulsante si fa uso del metodo `getPrinterJob()` della classe `java.awt.print.PrinterJob` che restituisce il `PrinterJob` di sistema, cioè la finestra che appare abitualmente quando vogliamo stampare un documento.

L'ultima sotto-opzione, *Exit*, serve per terminare TAXI e si occupa quindi di chiudere il frame e di terminare il programma.

5.4.2 LA CLASSE `HelpMenu.java`

La combinazione dei tasti per attivare questo menù è ALT-0. Le sue sotto-opzioni sono due: la prima è *Help* e la seconda è *Credits* come possiamo vedere in Figura 31.

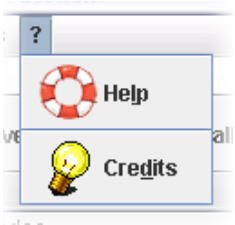


Figura 31. Il menù a discesa *?*

La prima sotto-opzione visualizza un pannello all'utente, cioè una `JOptionPane`, contenente le istruzioni generali e le principali caratteristiche dell'applicazione TAXI.

La seconda invece quando viene scelta visualizza una `JOptionPane` all'utente con le informazioni relative all'applicazione, l'istituto di appartenenza ed altro.

5.4.3 LA CLASSE `InformationsMenu.java`

In questo menù, attivabile tramite la combinazione ALT-A è presente soltanto una opzione: *Info*, tramite la quale l'utente visualizza informazioni sul file su cui sta lavorando.

Figura 32. Il menù a discesa *Informations*

5.4.4 LA CLASSE `OptionMenu.java`

Il menù *Options*, implementato in `OptionMenu.java` e attivabile tramite la combinazione ALT-N, ha due sotto-opzioni: *Reset* e *Background*.

Il primo assegna il peso di default ad ognuno dei nodi con peso dell'albero, annullando così ogni modifica fatta. Da notare, vedere Figura 33, che di default disabilitiamo l'opzione *Reset*, finché non viene aperto un file.

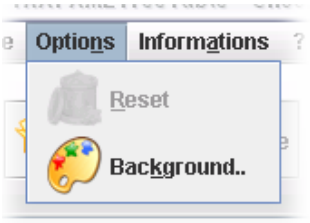


Figura 33. Il menù a discesa *Options*

L'opzione *Background* quando viene selezionata, visualizza una finestra come quella in Figura 34 (creata con la classe `JColorChooser` di Java) tramite cui l'utente può selezionare un colore per lo sfondo della tabella dove viene visualizzato il file.

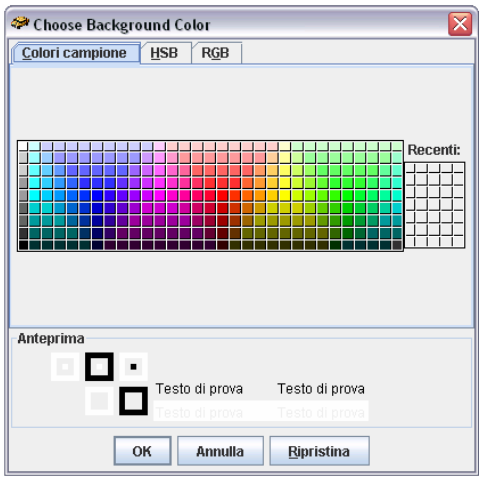


Figura 34. La finestra per la scelta dei colori

5.4.5 LA CLASSE `MenuBar.java`

La classe `MenuBar.java` estende la classe `JMenuBar` di Java, che ci consente la creazione di una barra dotata di un numero variabile, a seconda delle esigenze, di menù e sottomenù [40]. Ha un costruttore unico all'interno del quale viene creato ognuno dei menù sopra descritti e vengono aggiunti alla barra mostrata in Figura 35 che verrà poi inserita nell'interfaccia finale.

Figura 35. La barra dei menù

5.4.6 LA CLASSE `ToolBarNorth.java`

La classe `ToolBarNorth.java` rappresenta la barra degli strumenti situata in alto nell'interfaccia. Nel suo unico costruttore vengono aggiunti ad essa dei pulsanti tramite il metodo `addToolbarButton()`. Vediamo in Figura 36 la toolbar prodotta.

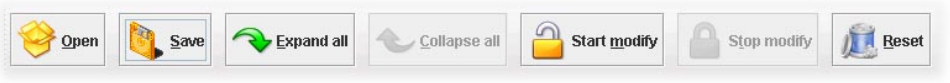


Figura 36. La ToolBarNorth

I pulsanti *Open*, *Save* e *Reset* svolgono le stesse operazioni descritte per le opzioni del menù a tendina *File* e *Options* descritti nelle precedenti sez.

Il pulsante *Expand all* permette all'utente di espandere automaticamente ogni nodo dell'albero visualizzato. Il nucleo di questa applicazione consiste in un ciclo for che scorre tutti i nodi dell'albero, ed applica loro il metodo `expandRow()` che consente di mostrare tutti i figli del nodo su cui è invocato.

Il pulsante *Collapse all* implementa l'azione inversa di *Expand all* restringendo ognuno dei nodi incontrati tramite l'opportuno metodo `collapseRow()`.

Il pulsante *Start modify* è il pulsante tramite cui è possibile aprire una sessione di modifica dei pesi. Infatti la sua pressione rende editabile la colonna contenente i pesi.

Il pulsante *Stop modify* permette all'utente di terminare la modifica dei pesi sui nodi figli di una `choice`. Inoltre quando viene premuto, viene attivato l'aggiornamento automatico dei pesi inseriti. Da notare che se questo pulsante non viene premuto, il file attuale, con i suoi nuovi pesi, non può essere salvato. Infatti in questa condizione il pulsante *Save* risulta disabilitato.

5.4.6.1 IL METODO `addToolbarButton()`

Diamo ora uno sguardo all'implementazione del metodo `addToolbarButton()`. I suoi parametri sono un oggetto di tipo `JToolBar`, che rappresenta la barra degli strumenti a cui vogliamo aggiungere il nuovo pulsante; un valore booleano che indica se deve essere visualizzata un'icona sul pulsante; una stringa che identifica l'etichetta che sarà visualizzata sul pulsante (ad esempio *Open*), un'altra per indicare il path dell'icona da associare a tale pulsante ed infine una stringa che rappresenta il tooltip. Il bottone viene posizionato all'interno della toolbar tramite un layout di tipo `FlowLayout`. Inoltre ad ogni bottone si associa, ovviamente, un `ActionListener` per captare quando esso viene premuto. Nel corpo del metodo ci si occupa di aggiungere alla toolbar stabilita il pulsante, a questo viene assegnata l'etichetta tramite l'opportuno metodo `setText()` e il tooltip con il metodo `setToolTipText()`.

5.4.7 LA CLASSE `ToolBarSouth.java`

`ToolBarSouth.java` è la classe tramite cui abbiamo implementato la toolbar che vediamo in Figura 37 che è situata in basso nell'interfaccia.

Figura 37. La ToolBarSouth

Come possiamo vedere i pulsanti posizionati in questa barra degli strumenti sono tre: *Info*, *Refresh* e *Quit*. Questi vengono aggiunti alla toolbar tramite il metodo `addToolBarButton()` la cui implementazione risulta molto simile a quella contenuta nella classe `ToolBarNorth.java`.

Premendo il pulsante *Info* vengono visualizzate all'interno di un `JTabbedPane` alcune utili informazioni sul file su cui l'utente sta lavorando. Premendolo una seconda volta questo pannello viene nascosto. Queste informazioni mostrano il path completo del file in corso di modifica, il numero di nodi dell'albero visibili al momento della pressione del pulsante e il numero di nodi a cui è possibile assegnare un peso (vedi Figura 38). Il numero dei nodi visibili varia a seconda del numero dei nodi espansi o collassati che ci sono nell'albero in un determinato istante.

Figura 38. Le informazioni visualizzate

Premendo il pulsante *Refresh* aggiorniamo a video la situazione dei pesi. Infatti ogni volta che l'utente inserisce un nuovo peso gli altri, a video, rimangono invariati mentre quelli contenuti all'interno delle strutture dati per la gestione dei pesi sono costantemente aggiornati. Quindi lo scopo del refresh è quello di prelevare i corretti valori dei pesi dalle opportune strutture dati e aggiornare i valori mostrati nell'interfaccia.

L'ultimo pulsante rimasto da descrivere è *Quit* ed ha la ovvia funzione di chiudere l'applicazione mostrando prima una finestra di dialogo di conferma.

5.4.8 LA CLASSE `ToolBarEast.java`

La toolbar che l'utente può vedere alla sua sinistra nell'interfaccia è implementata all'interno della classe `ToolbarEast.java`.

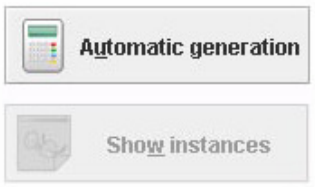


Figura 39. La ToolBarEast

Come possiamo vedere dalla Figura 39 questa toolbar è dotata di due pulsanti, aggiunti alla barra degli strumenti sempre tramite il metodo `addToolbarButton()` che in questo caso non posiziona i pulsanti lungo una fila orizzontale, ma in verticale grazie all'uso di un `BoxLayout` anziché `FlowLayout`.

Il primo pulsante, *Automatic generation,* è quello che si occupa di dare il via alla generazione automatica delle istanze finali. Per abilitarlo l'utente deve prima salvare il file su cui sta lavorando in modo tale da poter iniziare la computazione lavorando su di un file contenente i pesi.

Una volta prodotte le istanze finali è possibile visualizzarle tramite il pulsante *Show instances* che si attiva una volta terminata la generazione automatica. Per prima cosa viene creato un `FileDialog` per visualizzare la finestra dove l'utente potrà scegliere da file system un'istanza tra le tante generate. La cartella di default in cui le istanze vengono salvate è la cartella *Instances* ed è anche la prima cartella il cui contenuto viene mostrato all'interno del `FileDialog`. L'istanza scelta viene mostrata all'interno di un notepad implementato da noi.

5.4.9 LA CLASSE `Legenda.java`

La classe `Legenda.java` nasce dalla necessità di rendere più chiara, specialmente per utenti inesperti, la visualizzazione del file XML. Infatti esistono nodi di molti tipi che devono essere differenziati tra di loro in base alle loro caratteristiche. Per questa ragione ci è sembrato opportuno fornire una piccola guida all'interno della quale sono elencate tutte le icone possibili, che possono essere utilizzate per indicare ed evidenziare un nodo, e il loro significato.

Ogni voce della legenda sarà quindi composta da una `JLabel` contenente un'icona seguita da una breve descrizione del suo utilizzo all'interno dell'albero, come si può vedere in Figura 40.

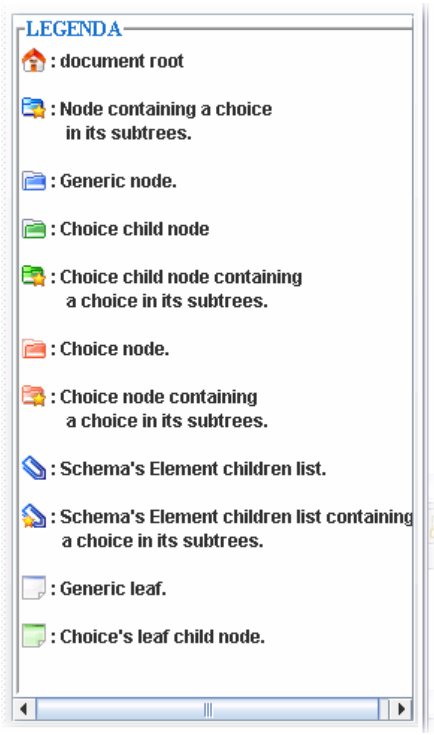


Figura 40. La Legenda

Capitolo 6

TEST STRATEGY SELECTOR

In questa sezione descriveremo in dettaglio l'implementazione della fase relativa alla selezione della strategia di test, TSS, di XPT, spiegando il funzionamento di ciascuna delle componenti che la compongono cioè Weigths Assignment (WA), Choice Analysis (CA) e Strategy Selector (SS), che possiamo osservare in Figura 1.

6.1 WHEIGTHS ASSIGNMENT

Nella fase *Weigths Assignment* di TSS il nostro compito è stato quello di implementare una tecnica che consentisse all'utente di assegnare pesi ai figli dei nodi choice in maniera automatica e di salvarli opportunamente all'interno di un XML Schema, non più ben-formato, in modo tale da poterlo riutilizzare in elaborazioni successive.

Ovviamente, come abbiamo già visto, l'inserimento avviene in maniera semplice ed intuitiva attraverso l'interfaccia fornita. Per comprendere al meglio ciò che accade sotto l'interfaccia quando un utente decide di inserire o modificare un nuovo peso presentiamo, qui di seguito, le classi che sono coinvolte nello sviluppo della fase WA tutte contenute all'interno del package `TSS.WeigthsAssignment`.

- `NodeMap.java;`
- `FlagMap.java;`
- `RowMap.java;`
- `RowCounter.java;`
- `DefaultInitializer.java;`
- `XmlFileBuilder.java.`

L'utilità di tali classi consiste nella memorizzazione temporanea dei pesi prima del salvataggio su file, per evitare che vadano persi, nel mantenere lo stato dei vari nodi durante la computazione e nell'implementazione di tecniche per il calcolo di valori di default e per il salvataggio del file XML pesato su file.

6.1.1 LA CLASSE `NodeMap.java`

La classe `NodeMap.java` è la classe che implementa la struttura dati che tiene memoria dei nodi figli di `choice` (o di un elemento di primo livello) e dei loro relativi pesi. Questa classe incapsula una `HashMap` e fornisce tutti i metodi statici per poter inserire nuove associazioni di tipo `<nodo,peso>` al suo interno, eliminarle, per recuperare il peso di uno specifico nodo, verificare se la mappa contiene già un'associazione per un nodo e per cancellare tutti gli elementi della mappa. L'esigenza di creare questa struttura dati è nata dal fatto che, ogni qualvolta l'utente, durante una sessione di modifica dei pesi, inserisce un nuovo valore, c'è bisogno di salvare il nuovo input da qualche parte, in modo tale che possa essere recuperato ed anche rielaborato. Infatti ogni volta che un peso viene inserito, tutti gli altri devono essere modificati per fare in modo che la loro somma dia 1, come precedentemente descritto, e il loro valore temporaneo viene salvato proprio all'interno della mappa dei pesi. Quindi questa mappa è risultata uno strumento indispensabile per il salvataggio momentaneo dei nuovi pesi, una tappa intermedia, prima del loro effettivo salvataggio su file. Inoltre, contenendo sempre i valori aggiornati, al momento in cui l'utente preme il pulsante *Refresh* nell'interfaccia, vengono prelevati i valori corretti da questa struttura dati e vengono visualizzati accanto al relativo nodo.

6.1.1.1 I METODI DELLA CLASSE `NodeMap.java`

- `put();`
- `containsNode();`
- `set();`
- `reset();`
- `keySet();`

- `size()`.

6.1.2 LA CLASSE `FlagMap.java`

La classe `FlagMap.java` è la classe che fornisce la struttura dati all'interno della quale viene memorizzato lo stato di ogni nodo figlio di `choice` (o di un elemento di primo livello). L'utilizzo di questa classe è piuttosto importante per l'implementazione di una politica per l'aggiornamento dei pesi. Infatti, supponiamo che un utente inserisca un nuovo peso tramite l'interfaccia. I pesi dei nodi fratelli verrebbero aggiornati secondo la formula:

(1 – peso inserito) / (n° nodi - 1) (6.1)

Ma se viene inserito un altro peso, come avviene l'aggiornamento? Non possiamo utilizzare ancora una volta la (6.1) in quanto il primo valore inserito verrebbe soprascritto da quello di default per i nodi non modificati. Per questo motivo occorre tenere memoria dello stato di ogni nodo. Non appena viene modificato un nodo, viene inserita, all'interno della mappa degli stati, un'associazione del tipo `<nodo,true>` dove `true` indica l'avvenuta modifica. Quindi, al momento dell'aggiornamento dei pesi, si controllano quali sono gli stati (modificato / non modificato) dei nodi da aggiornare e, si opera secondo la (2.2). In questo modo nessun peso viene erroneamente soprascritto, poiché si tiene conto dei nodi già modificati, e si hanno sempre valori consistenti all'interno della tabella.

Un altro ruolo importante di questa struttura dati è all'interno del ripristino di una sessione. Infatti, se un utente durante una sessione di modifica dei pesi, decide di salvare quanto fatto fino a quel momento e di riprendere il lavoro in seguito, è necessario conservare anche gli stati dei nodi affinché i valori non vengano soprascritti durante la successiva sessione. Per questo, durante il salvataggio, dobbiamo memorizzare su file anche lo stato di ogni nodo in modo tale che la sessione successiva possa essere ripristinata esattamente al punto di quando è stata abbandonata. Il salvataggio dello stato di ogni nodo avviene all'interno di un attributo chiamato *modificato* del figlio fittizio `ISTI_et_CNR_pesi`.

6.1.2.1 I METODI DELLA CLASSE `FlagMap.java`

- `put();`
- `containsKey();`
- `remove();`
- `reset();`
- `get();`
- `isEmpty().`

6.1.3 LA CLASSE `DefaultInitializer.java`

Affinché si possa lavorare in maniera corretta, le mappe appena descritte devono contenere dei valori che non siano nulli fin dall'inizio dell'esecuzione del tool in quanto la visualizzazione dei pesi iniziali dei nodi nell'interfaccia è il primo passo che viene eseguito. A tale scopo è nata la classe `DefaultInitializer.java` che si occupa, tramite il metodo `initializer()`, che opera ricorsivamente su un documento DOM, di riempire le mappe con valori opportuni. Questo metodo opera distinguendo due casi. Il primo è relativo ad un file che viene elaborato ed aperto per la prima volta. Nel secondo caso si ha invece a che fare con un file che è già stato salvato e che contiene valori consistenti da caricare.

Nel primo caso, per quanto riguarda il peso, è necessario calcolare un valore di default in modo tale che la somma dei pesi dei figli di un nodo `choice` sia pari ad 1. Quindi, una volta verificato che i figli di un nodo `choice` non hanno figli del tipo `ISTI_et_CNR_pesi`, si calcola il peso di default operando secondo la (2.1). Il peso di due o più nodi fratelli sarà quindi uguale e verrà memorizzato all'interno della mappa dei pesi. La mappa dei flag che mantengono lo stato di ogni nodo sarà ovviamente riempita per default con associazioni del tipo `<nodo,false>` in quanto all'inizio ogni nodo è salvato come non modificato dall'utente.

Nel secondo caso invece, ogni nodo figlio di `choice` possiede, all'interno del file, un figlio di tipo `ISTI_et_CNR_pesi` con due attributi, *peso* e *modificato*, in cui sono memorizzati i valori che verranno prelevati e inseriti all'interno delle due mappe.

6.1.4 LA CLASSE `RowMap.java`

Oltre alla necessità di mantenere una struttura dati per memorizzare il peso e lo stato di un nodo si è rivelato indispensabile mantenere anche una struttura dati che tenesse memoria della riga della `JTreeTable`, corrispondente ad ogni nodo figlio di `choice`. Questo perché, per meccanismi interni all'implementazione della classe `JTreeTable`, è possibile aggiornare, nell'interfaccia, un solo peso alla volta, man mano che vengono inseriti dall'utente. Quindi, quando un peso viene modificato, tutti gli altri, benché il loro valore cambi, rimangono invariati all'interno dell'interfaccia, fino a che l'utente non decide di effettuare un'operazione di *Refresh* per aggiornare i valori dei pesi, mostrati a video, a seguito di modifiche da lui effettuate. Per aggiornare il peso di un nodo nella `JTreeTable` però occorre conoscere le coordinate della cella in cui andare ad inserire tale nuovo valore. Una cella è determinata univocamente da un numero di riga e da un numero di colonna. Essendo, in questo caso, l'indice della colonna costante (sempre 1), l'unico dato mancante è quello della riga e lo troviamo memorizzato all'interno della mappa di righe in una associazione del tipo `<nodo,riga>`. Quindi, ogni volta che l'utente richiede un aggiornamento dei valori, si esegue un ciclo per settare agli opportuni nodi il valore corretto prelevandone la riga da questa mappa. La classe `RowMap.java` è, appunto, la classe che fornisce la struttura dati all'interno della quale viene memorizzata la riga di ogni nodo figlio di choice (o di un elemento di primo livello).

6.1.5 LA CLASSE `RowCounter.java`

La mappa contenente la riga relativa ad ogni nodo figlio di un nodo choice deve essere inizializzata non appena il programma viene avviato. Questo avviene grazie al metodo statico `count()` contenuto all'interno della classe `RowCounter.java` che opera ricorsivamente su un documento DOM. Il suo funzionamento è piuttosto semplice infatti questo effettua una visita in profondità sul documento passato come parametro e conta il numero dei nodi che incontra, ed ogni

nodo rappresenta una riga della tabella, escludendo tutti quelli che non devono comparire nell'interfaccia, cioè quelli di tipo `ISTI_et_CNR_pesi`, `Text` e `Comment`, e inserisce le opportune associazioni all'interno della mappa di righe.

6.1.6 LA CLASSE `XmlFileBuilder.java`

La classe `XmlFileBuilder.java` nasce dalla comprensibile necessità dell'utente di voler interrompere il lavoro iniziato, salvandolo su di un file per poterlo riprendere in un secondo momento. Per fare ciò è quindi necessario un metodo che, ogni qualvolta che l'utente decide di interrompere una sessione di modifica dei pesi, produca un nuovo file XML, contenente tutti i dati inseriti dall'utente fino a quel momento, in modo tale da poterlo ricaricare in seguito per riprendere il lavoro precedentemente avviato dal punto in cui è stato abbandonato. La produzione di un file XML contenente i pesi inseriti è comunque necessaria per la prosecuzione del processo di generazione automatica.

6.1.6.1 IL METODO `createFileIstanza()`

Il metodo `createFileIstanza()` è un metodo che opera ricorsivamente partendo dalla radice di un albero DOM implementando una visita di tipo *DFS*. È necessario procedere in profondità poiché su file dobbiamo stampare un nodo e di seguito tutti i suoi figli. Solo dopo questa operazione è possibile procedere con la stampa dei nodi dello stesso livello del nodo preso ad esempio.

Di un nodo si stampa su file XML tutto ciò che era contenuto sul file di origine e si procede applicando la funzione ricorsivamente su ognuno dei suoi figli. Quando si incontra un nodo figlio di una `choice`, e quindi un nodo avente peso, è necessario salvare questo peso su file, affinché il lavoro dell'ultima sessione non vada perduto. Discutendo con gli altri membri del team, operante sul nostro stesso progetto, abbiamo stabilito una convenzione secondo cui il salvataggio del peso e dello stato di un determinato nodo, dovesse avvenire all'interno di un nodo fittizio, quindi non appartenente allo schema, denominato `ISTI_et_CNR_pesi`, figlio del nodo con peso.

Il peso di ogni nodo viene prelevato dall'opportuna struttura dati che tiene memoria dei pesi di ogni nodo e viene salvato all'interno di un attributo di `ISTI_et_CNR_pesi` chiamato *peso*. Lo stato invece viene prelevato da una mappa di flag, in cui viene memorizzato lo stato di ogni nodo (modificato/non modificato), e viene salvato in un attributo chiamato *modificato*. Questo è importante perché, al momento del caricamento, ogni volta che si incontra un nodo con un figlio del tipo `ISTI_et_CNR_pesi` si prelevano i due valori appena descritti e tutte le mappe vengono ricostruite come al momento del salvataggio, in modo tale da ripristinare la sessione al punto esatto in cui è stata interrotta.

6.1.6.2 IL METODO `create()`

Un altro metodo di questa classe è il metodo `create()` che si occupa di stampare le prime righe di intestazione del file XML e di chiamare il metodo `createFileIstanza()` a partire dalla radice di un documento, variabile istanza della classe.

6.1.7 LA CLASSE `ReaderWriter.java`

Durante l'operazione di salvataggio all'interno del file vengono salvati anche i figli di tipo testo che ogni nodo possiede, che si traducono in spazi, linee vuote nel file. Stampare un carattere "`\n`" al termine di ogni linea, e quindi di ogni nodo, è fondamentale per il successivo riutilizzo del file, affinché il sistema possa continuare a funzionare correttamente. Infatti su ogni riga è consentito trovare solo un elemento dello schema.

In alcuni casi però i file aperti precedentemente con qualche particolare editor, possono presentare una indentazione scorretta e non allineata al margine del foglio elettronico, oppure linee troppo distaccate l'una dall'altra. Questo non ha alcun effetto sul funzionamento del programma da noi implementato. Nel momento in cui però il file salvato deve essere preprocessato, per poter giungere alla fase finale di generazione delle istanze, la presenza di questi spazi vuoti si rivela dannosa poiché da origine ad innumerevoli situazioni di errore. Allo scopo di risolvere questo problema, per poter integrare correttamente il nostro codice con quello del resto del team, è nata la classe `ReaderWriter.java`.

Questa classe, all'interno del suo costruttore si occupa di leggere ogni riga del file XML da formattare, il cui path viene passato come parametro. Vengono copiate solo le righe contenenti entrambi i caratteri "<" ">", cioè rappresentanti solo elementi appartenenti allo Schema. Nel caso in cui una riga contenesse tali caratteri vengono anche eliminati eventuali spazi presenti all'inizio della linea in modo tale da rendere il file perfettamente elaborabile. Una volta fatto ciò, se tutto è andato a buon fine, si copia tutto il contenuto del file di appoggio nel file di origine.

Questa classe verrà utilizzata, se si rivelerà necessario, anche da altre componenti del tool TAXI.

6.2 CHOICE ANALYSIS

Nella fase di *Choice Analysis* viene implementata una metodologia che consente la risoluzione del costrutto `<choice>`, che porta alla generazione di una serie di combinazioni contenenti ognuna uno ed un solo figlio di una determinata `choice`. Al termine di tale operazione, cioè quando tutti i sottoalberi saranno prodotti, sarà necessario calcolare il peso di ognuna delle foglie (figli di nodi `choice`) che compaiono nelle combinazioni generate.

L'albero DOM che si riceve in input nella fase CA viene inviato dalla componente preprocessor, che si occupa della risoluzione di alcuni tag. Durante questa fase è possibile che vengano aggiunti (operazione di `extension`) o tolti (operazione di `restriction`) figli ai nodi di tipo `choice`. Questo provoca situazioni inattese in quanto la somma dei pesi dei figli di un nodo `choice` può non essere pari ad uno e questa non è una situazione accettabile in quanto stiamo lavorando con una distribuzione discreta di probabilità.

La ridistribuzione uniforme dei pesi in proporzione a quelli esistenti è un'operazione che dovrebbe essere svolta al termine della fase di preprocessor oppure come la prima fase in assoluto di CA, ed è questo il nostro caso.

Le classi che sono servite per lo sviluppo di CA sono le seguenti e sono tutte contenute all'interno del package `TSS.ChoiceAnalysis`:

- `Redistributor.java;`
- `Choice.java;`
- `Paths.java;`

- `TreeWeigths.java;`

6.2.1 LA CLASSE `Ridistributor.java`

Il primo passo verso la generazione delle istanze finali è quello della normalizzazione a 1 dei pesi dei figli di nodi `choice` contenuti in un dato albero. La classe `Ridistributor.java` contiene i metodi per la risoluzione di questo problema.

Come visto nei capitoli precedenti, possono presentarsi tre diverse situazioni dovute alla modifica dei figli di un nodo `choice`:

i. somma dei pesi dei figli del nodo strettamente minore di 1;
ii. somma dei pesi dei figli del nodo strettamente maggiore di 1;
iii. somma dei pesi dei figli del nodo uguale ad 1.

La somma dei pesi risulta maggiore di 1 quando viene aggiunto ad una `choice` un figlio con peso, minore di 1 quando uno dei suoi figli viene rimosso ed uguale ad 1 quando è stato aggiunto un figlio senza peso o se il numero dei suoi figli non è stato modificato. Il metodo `redistribute()` si occupa di identificare la situazione in cui ci troviamo e di invocare l'opportuno gestore di ridistribuzione dei pesi. Il gestore è lo stesso nel caso in cui la somma sia strettamente maggiore di 1 o strettamente minore di 1. Questo opera ricorsivamente sull'albero, secondo una visita di tipo *DFS*, e ogni qualvolta un nodo di tipo `choice` viene incontrato effettua un controllo sui pesi dei figli memorizzandoli in un vettore. Il metodo `handle()` invece è quello che si occupa della ridistribuzione vera e propria. Questo lavora su un vettore contenente i pesi dei figli e su un oggetto di tipo `NodeList` contenente i figli veri e propri del nodo `choice`. Nel caso in cui la somma dei pesi sia maggiore o minore di 1 si opera secondo la 3.2.

Per quanto riguarda il caso della somma uguale ad 1 si opera in maniera diversa in quanto dobbiamo controllare se esiste almeno un figlio senza peso. Se non esiste significa che la `choice` non è stata modificata, mentre se ne esistono assegniamo loro un peso di default pari ad 1. Infine si opera ancora tramite la 3.2 per ottenere il valore cercato. Una volta corrette queste situazioni tutti i pesi sono normalizzati, si modifica l'attributo *peso* di ognuno dei figli `ISTI_et_CNR_pesi`, la distribuzione discreta delle probabilità è ripristinata e si può continuare a lavorare con valori corretti e consistenti.

6.2.2 LA CLASSE `Choice.java`

La classe `Choice.java` è la classe che si occupa della risoluzione del costrutto `choice`. Il metodo `risolvi_choice()` prende come parametro un documento e restituisce un vettore che contiene tanti documenti quante sono le varie combinazioni derivanti dal metodo `BFS_choice()` che risolve il costrutto `<choice>`. In questo metodo effettuiamo una visita ricorsiva del documento passato in input finchè non abbiamo eliminato dai vari file intermedi creati tutti i costrutti `<choice>`.

Il metodo `BFS_choice()` applica una visita di tipo *BFS*, che termina solo quando viene trovato un nodo del tipo che interessa (in questo caso `<choice>`), cercando l'elemento denominato `<choice>` e ogni volta che lo troviamo gli attributi di tale elemento vengono salvati in un array per poi riutilizzarlo una volta creato il nuovo nodo `<sequence>`. Per creare il nome del tag di questo nodo controlliamo se il nodo `<choice>` aveva il prefisso; in tal caso questo deve essere riportato al nodo `<sequence>`. In seguito prendiamo il fratello a destra e il padre del nodo `<choice>` e inseriamo il nodo appena creato al posto del vecchio nodo `<choice>` e infine per ogni figlio del vecchio nodo `<choice>` creiamo un nuovo documento XML Schema.

6.2.3 LA CLASSE `Paths.java`

Il terzo passo della componente *Choice Analysis* è quello del calcolo del peso delle foglie di un albero DOM. Il peso di una foglia viene calcolato moltiplicando tutti i pesi dei nodi aventi figlio `ISTI_et_CNR_pesi` che si incontrano lungo il cammino dalla radice fino alla foglia stessa. Per i nodi non aventi peso si considera come peso di default 1. Quindi si opera secondo la (2.3) per ottenere poi il peso dell'albero, su cui tale procedimento viene applicato, tramite la (2.4).

Un metodo per poter trovare il peso di una foglia è quello di effettuare una visita dell'albero a livelli e calcolare il peso di ogni nodo di un determinato livello e memorizzarlo, come precedentemente fatto, in un figlio `ISTI_et_CNR_pesi`. In realtà, dovendo operare con alberi piuttosto semplici (generalmente composti da due paths radice – foglia) si è optato per creare una classe, `Paths.java`, che contenesse un metodo, `generate()` ricorsivo, che producesse una lista di liste, ognuna contenente oggetti di tipo `FileNode`, tutti appartenenti ad uno stesso path che conduce dalla radice ad una foglia.

Una volta prodotto questo insieme di path, tramite il metodo `calculate()`, è possibile ottenere un vettore contenente il peso relativo a ciascun path calcolato e quindi all'unica foglia contenuta in quel path. Infatti questo metodo si occupa di scorrere ogni path e di moltiplicare tutti i pesi dei `FileNode` che incontra. Inoltre tramite il metodo `leaves()` è possibile creare un vettore, parallelo a quello dei pesi, che contiene tutte le foglie a cui i pesi corrispondono.

Il vantaggio di questo metodo è che, al termine di questa operazione, tutti i figli di tipo `ISTI_et_CNR_pesi`, che non sono riconosciuti dallo schema, possono essere eliminati dall'albero, che può esser modificato da altre applicazioni senza dover cambiare implementazioni di metodi preesistenti, poiché le foglie e i loro relativi pesi sono già stati salvati in opportune strutture dati grazie ai metodi appena descritti.

6.2.3.1 I METODI DELLA CLASSE `Paths.java`

- `generate();`
- `calculate();`
- `leaves();`
- `validatePaths();`
- `printPaths().`

6.2.4 LA CLASSE `TreeWeights.java`

L'ultimo passo della componente CA è quello del calcolo del peso delle sottostrutture prodotte. La classe `ThreeWeights.java` è la classe che si occupa, tramite gli opportuni metodi, di calcolare il peso di un albero come espresso in (2.4). Il suo costruttore prende come parametri i vettori delle foglie e dei pesi calcolati tramite i metodi della classe `Paths.java`.

6.2.4.1 IL METODO `getWeight()`

Il metodo `getWeight()`, che è quello che si occupa del calcolo vero e proprio del peso di un albero, prende come parametro il documento DOM di cui calcolare il peso totale. Questo lavora ricorsivamente ed effettua una visita in profondità nell'albero per la ricerca delle foglie. Ogni volta che una foglia viene incontrata si controlla se questa è contenuta all'interno del vettore delle foglie, condizione che in realtà è sempre verificata, e se si, si preleva il suo peso dal vettore dei pesi e si somma alla variabile che mantiene il peso totale dell'albero. Il peso di un albero è il fattore che ne determina la partecipazione o non alla generazione finale delle istanze.

6.3 STRATEGY SELECTOR

La componente *Strategy Selector* è l'ultima relativa alla componente TSS. Compito di questa componente sarà quello di scegliere opportunamente i sottoalberi da restituire alla componente successiva di XSA per procedere lungo il processo di generazione delle istanze finali.

La scelta degli alberi avviene, ovviamente, in base al loro peso. Quindi prima di procedere con la richiesta all'utente, tramite opportuna interfaccia, la strategia da scegliere occorre ordinare le sottostrutture generate in base al loro peso, calcolato nella fase precedente, e, se necessario, normalizzare la somma dei pesi dei sottoalberi ad 1, per ottenere il peso finale di ciascuna struttura.

Le classi che sono coinvolte nello sviluppo di questa componente sono le seguenti e sono contenute all'interno del package `TSS.StrategySelection`:

- `Sorter.java;`
- `MultiComponent.java;`
- `CustomTextField.java.`

6.3.1 LA CLASSE `Sorter.java`

La classe `Sorter.java` contiene tutti i metodi necessari per l'applicazione del metodo `mergeSort()` al fine di ordinare un vettore contenente documenti DOM in ordine crescente in base

al loro peso. L'operazione di ordinamento degli alberi e dei loro pesi è una fase molto importante in quanto, l'utente, come precedentemente descritto nel capitolo relativo alle interfacce, una volta generati i sottoalberi derivati dall'albero iniziale, è invitato a scegliere la quantità di sottoalberi che vuole mantenere, secondo diverse strategie, e i sottoalberi restituiti sono ovviamente quelli che hanno il peso maggiore e che quindi hanno gli indici maggiori all'interno dell'array ordinato.

Inoltre occorre ricordare che la somma dei pesi dei sottoalberi, generati a partire da uno stesso documento, deve essere pari ad 1, come descritto nel Capitolo 3. Per questa ragione è necessario effettuare la normalizzazione ad 1 dei loro pesi tramite la (2.6) che determina il *peso finale* di ciascun albero.

Questa operazione va effettuata su ognuno dei sottoalberi generati in modo da poter operare, in seguito, con valori corretti e consistenti. Ai fini dell'ordinamento non è importante se la normalizzazione avviene prima o dopo la chiamata del metodo `mergeSort()`, e noi abbiamo scelto di effettuarla precedentemente.

6.3.1.1 I METODI DELLA CLASSE `Sorter.java`

- `mergeSort();`
- `msort();`
- `merge().`

6.3.1.2 IL METODO `mergeSort()`

Per l'ordinamento di array abbiamo utilizzato il metodo `mergeSort()`, modificato secondo le nostre esigenze. Questo metodo viene utilizzato per ordinare in ordine crescente degli alberi DOM, in base al loro peso. Quindi, essendo i documenti DOM memorizzati in un vettore diverso rispetto ai loro pesi, in questo caso il metodo prende come parametri due array: uno è quello degli alberi e uno è quello dei pesi veri e propri. L'ordinamento viene applicato a quello dei pesi, ma le stesse operazioni effettuate su di esso vanno applicate anche sul vettore dei documenti per far si che ad ogni documento corrisponda sempre la stessa posizione del suo peso di origine nel vettore.

6.3.2 LA CLASSE `MultiComponent.java`

La classe `MultiComponent` è stata implementata per poter creare un'altra piccola interfaccia grafica in cui avviene la scelta della strategia da adottare per la generazione automatica delle istanze finali. Questa deve comparire dopo aver premuto il pulsante *Automatic generation*, una volta che sono state prodotte le opportune sottostrutture.

Il costruttore della classe ha due parametri: il vettore contenente le sottostrutture prodotte e il corrispondente vettore dei pesi. Dato che la somma totale dei pesi delle sottostrutture potrebbe non essere pari ad 1 occorre operare subito una ridistribuzione dei pesi secondo la (2.6), come spiegato nel Capitolo 2, ordinando poi il vettore delle sottostrutture in ordine crescente in base al loro peso. Oltre a queste operazioni preliminari per il reperimento delle sottostrutture ci occupiamo degli opportuni abbellimenti grafici relativi all'interfaccia, aggiungendo loghi e impostando il colore di sfondo e del carattere. Nella parte centrale del frame creiamo invece un pannello contenente tre `RadioButton` (in un `ButtonGroup` che consente la selezione di una sola delle opzioni proposte) per la scelta della strategia con cui generare le istanze.

Nel caso in cui la strategia scelta sia *Coverage* l'utente deve selezionare la percentuale desiderata tramite l'apposito `JSpinner`, ovvero una area di testo dotata di due pulsanti per l'incremento e il decremento del valore da selezionare. I valori ammissibili sono compresi tra 0 e 100 e il valore di default è 0.

Nel caso in cui la strategia scelta sia *N° instances* l'utente deve inserire un valore corrispondente al numero di istanza desiderato all'interno di un `JTextField` che accetta soltanto numeri interi. Per fare si che possano essere accettati solo caratteri numerici si utilizza la classe `CustomTextField` descritta nel paragrafo successivo.

Nel caso in cui la strategia scelta sia *Mixed mode* l'utente deve poter scegliere una percentuale fissata tramite un `JSpinner` ed inserire un numero di istanze da ricavare da tale percentuale all'interno di un `JTextField`.

Ad ognuno dei `RadioButton` descritti dobbiamo associare dei `RadioListener` che determinano quando l'opzione è stata selezionata.

L'interfaccia è dotata di due pulsanti, *Ok* e *Cancel*, ognuno con le rispettive icone e i rispettivi `ActionListener()`. All'interno dell'`actionPerformed()` del pulsante *Ok*, che si attiva quando questo viene premuto, viene chiamato il metodo `ok_actionPerformed()`. Questo metodo si occupa di distinguere quale `RadioButton` è stato selezionato al momento della pressione

del pulsante. Nel caso in cui sia stato selezionato il primo allora, si invoca il metodo `coverage_actionPerformed()` che implementa la strategia descritta nella sez. 2.3.2. Nel secondo caso si esegue un ciclo `do-while` in cui si chiama il metodo `instances_ActionPerformed()`, che implementa la strategia descritta in sez. 0, finché l'utente non da in input dei valori validi. Il terzo ed ultimo caso è del tutto simile al secondo con l'eccezione che dentro al ciclo `do-while` viene invocato un altro metodo chiamato `mixed_actionPerformed()` che implementa la strategia di sez. 2.3.3.

All'interno dell'`actionPerformed()` del pulsante *Cancel*, invece, ci si occupa semplicemente di chiudere la finestra tramite il comando `frame.dispose()`.

6.3.3 LA CLASSE `CustomTextField.java`

Abbiamo implementato questa classe per poter gestire e controllare input forniti dall'utente in modo tale che questi, per esempio, non inserisca lettere in aree di testo dove possono essere inseriti solamente caratteri numerici.

`CustomTextField.java` estende la classe `PlainDocument` (per la gestione del testo). Per prima cosa viene creata una variabile di tipo `StringBuffer` che sarà utilizzata per contenere i caratteri immessi nella `JLabel`. Il costruttore prende come parametro il numero massimo di caratteri da poter inserire nella casella di testo.

Nel corpo del metodo `insertString()` si controlla che la lunghezza della stringa inserita non sia più lunga della lunghezza massima consentita. Il cuore di questo metodo è composto da un ciclo for, che scandisce ogni carattere della stringa in input e controlla che esso sia un carattere numerico. Se lo è viene inserito nella variabile cache, altrimenti si esce dal ciclo. Dopo tale controllo, se la cache contiene almeno un carattere (cioè se la stringa inserita risulta corretta), viene chiamato il metodo `insertString()` della superclasse che si occupa dell'inserimento effettivo della stringa nel giusto campo testuale. Un altro metodo della classe `CustomTextField` è `isNumber()`, che restituisce `true` se il carattere passato come parametro è un numero, `false` altrimenti.

6.3.4 LA CLASSE `RadioListener.java`

La classe `RadioListener`, cui abbiamo fatto alcuni cenni nella sez. 6.3.2, è contenuta all'interno della classe `MultiComponent.java` implementa `ActionListener` ed è quella che si occupa di ascoltare quando un `RadioButton`, per la selezione della strategia, viene cliccato. `RadioListener` contiene il metodo `actionPerformed()` che ha il compito di abilitare esclusivamente il campo per l'inserimento dell'input (`JSpinner` o `JTextField`) associato a quel bottone, disabilitando tutti gli altri.

Capitolo 7

LE INTERFACCE PER VP E LA VISUALIZZAZIONE DELLE ISTANZE

Oltre ad occuparci dello sviluppo dell'interfaccia grafica per il tool e della componente TSS abbiamo fornito una serie di interfacce per VP in via di sviluppo che consentono all'utente l'interazione con la componente. Inoltre abbiamo implementato un visualizzatore di file XML tramite il quale l'utente potrà analizzare tutte le istanze prodotte dal tool.

Per quanto riguarda la costruzione delle interfacce per la componente VP le classi coinvolte sono le seguenti e sono tutte contenute all'interno del package `VPinterfaces`:

- `DBInterface.java;`
- `Interface1.java;`
- `Mode.java;`
- `Restrictions.java.`

Per quanto riguarda invece il formato per la visualizzazione delle istanze le classi utili sono le seguenti, contenute nel package `TAXInstancesVisualizer`:

- `Apertura.java;`
- `Editor.java.`

Abbiamo scelto di presentare insieme questi due lavori così diversi poiché rappresentano entrambi un lavoro esterno a quello che dovevamo implementare in origine e che abbiamo aggiunto al lavoro di base da svolgere per lo stage per completezza.

Illustriamo di seguito le classi e i package elencati.

7.1 IL PACKAGE `VPinterfaces`

Questo package contiene l'implementazione di tutte le interfacce che sono state sviluppate per l'interazione dell'utente con la componente VP.

7.1.1 LA CLASSE `DBInterface.java`

Tramite la classe `DBInterface.java` si crea un frame, all'apertura dell'applicazione, che deve interagire con l'utente per il caricamento del database che sarà utilizzato dalla componente VP.

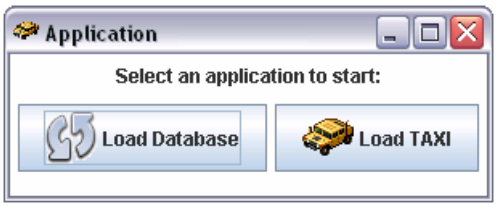


Figura 41. Interfaccia di VP per scegliere l'applicazione da avviare

Nel frame vengono visualizzati due pulsanti, come si può vedere in Figura 41, tramite i quali si può scegliere se si vuole caricare un database esistente, da cui prelevare i valori per i vari elementi per la produzione delle istanze, oppure continuare con l'applicazione TAXI caricando il database in un momento successivo.

7.1.2 LA CLASSE `Mode.java`

Tramite la classe `Mode.java` si crea un frame che deve permettere all'utente di scegliere la modalità di assegnamento dei valori a ciascun elemento. Infatti, come possiamo notare in Figura 42, l'utente ha la possibilità di assegnare manualmente un valore da lui scelto a ciascun elemento e nel caso in cui non voglia farlo esiste anche una assegnazione random di determinati valori.

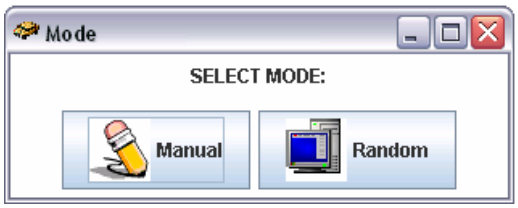


Figura 42. Interfaccia di VP per la selezione della modalità

Nel caso in cui l'utente scelga l'assegnazione manuale si aprirà una seconda interfaccia, progettata a questo scopo, e che verrà descritta nei paragrafi seguenti.

7.1.3 LA CLASSE `Values.java`

Tramite la classe `Values.java` si crea un frame, all'apertura dell'applicazione, che deve permettere all'utente di assegnare manualmente dei valori ad un determinato elemento.

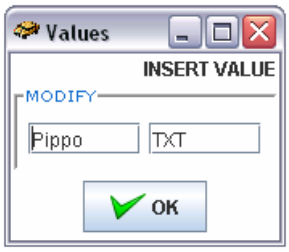


Figura 43. Interfaccia di VP per l'inserzione di valori

Facendo riferimento alla Figura 43, nella casella di testo di sinistra verrà inserito l'elemento a cui deve essere assegnato un valore e in quella di destra il valore che l'utente vuole assegnare.

7.1.4 LA CLASSE `Restriction.java`

L'ultima delle interfacce per VP riguarda le restrizioni ed è implementata tramite la classe `Restriction.java`. Lo scopo di questa interfaccia è quello di consentire all'utente di specificare le restrizioni che reputa di maggior interesse durante la definizione delle istanze finali. Possiamo vederla raffigurata in Figura 44.

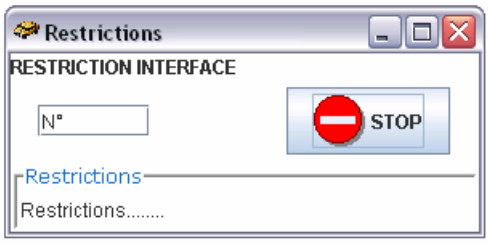


Figura 44. Interfaccia di VP per la gestione delle restrizioni

7.2 IL PACKAGE `TAXInterfaceVisualizer`

Il package `TAXInterfaceVisualizer` contiene le classi che sono servite per l'implementazione di un semplice editor di testo tramite cui, al termine della generazione automatica è possibile visualizzare le istanze finali prodotte. Infatti l'utente tramite questo editor può aprire, una per una, le istanze, prenderne accuratamente visione ed analizzarne la struttura.

7.2.1 LA CLASSE `Editor.java`

`Editor.java` estende la classe `JFrame` ed ha un costruttore che prende come parametro una stringa che rappresenta il path del file XML da mostrare all'interno dell'editor.

7.2.1.1 IL METODO `createPad()`

Questo metodo non ha parametri e rappresenta il vero e proprio cuore del notepad per la visualizzazione delle istanze. Questo notepad è costituito semplicemente da un frame al cui interno viene posizionata un'area di testo e una semplice toolbar con il solo pulsante *Open an instance*. Possiamo osservare il notepad prodotto in Figura 45.

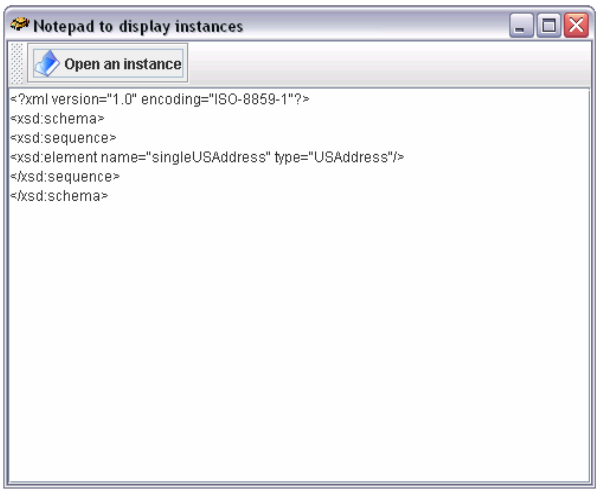


Figura 45. Il notepad per la visualizzazione delle istanze

Quando il pulsante *Open an instance* viene premuto appare all'utente un `FileDialog` che consente la scelta di una istanza dalla cartella predefinita *Instances*. Una volta scelto il file XML da aprire viene invocato il metodo `apri()` della classe `Apertura.java` che vedremo tra breve, il quale si occupa di visualizzare il contenuto del file all'interno dell'area di testo del notepad.

7.2.2 LA CLASSE `Apertura.`

La classe `Apertura.java` mette a disposizione i metodi per rendere completamente operativo il notepad sviluppato. In particolare fornisce il metodo per la visualizzazione delle istanze all'interno dell'editor.

7.2.2.1 IL METODO `apri()`

`apri()` è semplicemente un metodo che ha come parametri un oggetto di tipo `File` e uno di tipo `JTextArea`.

Nel corpo del metodo viene aperto il file passato come parametro e letto carattere per carattere, tramite un ciclo `while` e li appende ad una stringa che verrà poi visualizzata nella nuova area di testo che verrà restituita al termine dell'esecuzione del metodo.

Capitolo 8

TESTING E INTEGRAZIONE

Una volta verificata la correttezza del funzionamento del nostro modulo, l'ultima fase da affrontare è stata quella dell'integrazione del nostro prodotto con i moduli preesistenti, sviluppati da altri e perfettamente funzionanti. Questa era una fase particolarmente temuta in quanto, nonostante i moduli presi singolarmente manifestino un comportamento corretto, è probabile che, nel funzionamento globale del sistema, si verifichino situazioni del tutto inattese. Il problema, infatti, consiste proprio nella fase di aggregazione di più moduli, ossia nell'interfacciamento. Dati possono perdersi nell'attraversare un'interfaccia oppure un modulo può provocare involontariamente effetti indesiderati sul comportamento di un altro modulo e, nel peggiore dei casi, è possibile che nello sviluppo di una componente si sia tenuto conto di particolari convenzioni ignorate completamente in un'altra. Questo ultimo caso non comporta situazioni anomale nel funzionamento di un singolo modulo, ma le comporta nel funzionamento globale del sistema [47].

Una situazione simile è capitata durante il nostro lavoro. Infatti, come precedentemente descritto, durante il salvataggio del file, il peso del figlio di una choice viene inserito all'interno di un suo figlio fittizio di tipo `ISTI_et_CNR_pesi`. Questa è stata una decisione presa insieme agli altri sviluppatori per fare in modo di lavorare tutti nella stessa direzione e con le stesse convenzioni. In realtà però sono sorti dei problemi con una componente antecedente al nostro tirocinio che, non tenendo conto dei figli di tipo `ISTI_et_CNR_pesi`, si comportava come se non esistessero, provocando la perdita di tutti i pesi inseriti. Questo ovviamente impediva il funzionamento del sistema e la corretta generazione delle istanze finali da parte del tool TAXI.

Per ovviare a questo problema è stata necessaria una revisione del codice relativo a quella componente da parte di coloro che ne avevano seguito lo sviluppo. Siamo riusciti ad individuare subito quale fosse la componente che provocava il malfunzionamento grazie alla strategia di integrazione incrementale che abbiamo adottato. Infatti, ogni modulo sviluppato veniva testato singolarmente tramite casi di test pianificati. Per un dato input veniva calcolato l'output atteso, e in base a tali test è stato possibile correggere eventuali errori commessi in fase di stesura. Una volta

verificata la correttezza del modulo relativamente ai casi di test adottati, veniva verificata la bontà del comportamento dello stesso modulo con degli input prodotti direttamente dagli altri moduli.

Quindi la fase di integrazione è avvenuta gradualmente, per evitare di costruire il programma aggregando subito tutti i moduli e di provarlo come un sistema unitario. Il risultato sarebbe stato molto caotico perché al presentarsi degli errori, l'individuazione degli stessi sarebbe stata indubbiamente più complessa. Uno stile di integrazione di questo tipo è comunque destinato al fallimento in quanto raramente un sistema funziona subito correttamente senza presentare il minimo errore, specie se frutto del lavoro di più persone e se le sue componenti vengono sviluppate a grande distanza di tempo. Lo stile di *integrazione incrementale* [47] si è rivelato un ottimo mezzo per la correzione mirata di eventuali errori. Infatti grazie a ciò la correzione dell'errore verificatosi è stata semplice ed è avvenuta in tempi brevi.

Oltre alle tecniche di collaudo appena descritte abbiamo cercato di effettuare anche una specie di *alpha-tes*t svolto in un ambiente controllato, con la nostra supervisione. È stata fatta provare l'interazione con l'interfaccia a dei possibili utenti, coinvolti anch'essi nello sviluppo del tool, per verificare che quanto prodotto fosse conforme alle loro esigenze e ai loro bisogni. Grazie a questi test siamo stati in grado di registrare errori che non eravamo riusciti ad individuare ed eventuali problemi d'uso riscontrati dal tester. Abbiamo registrato consigli relativi a correzioni, che secondo l'utente dovevano essere effettuate, e le abbiamo messe in atto per migliorare la qualità dell'interfaccia e del software proposto.

L'operazione di collaudo ha impiegato non poco del nostro tempo in quanto era indispensabile rilasciare al committente una versione perfettamente funzionante del software da noi prodotto, correggendo il maggior numero di situazioni non desiderabili che si sono presentate durante il funzionamento globale del sistema, in quanto, come tirocinanti, non avremo più la possibilità di correggere eventuali comportamenti inaspettati dovuti a malfunzionamenti provocati dal nostro modulo.

Questa fase si è rivelata molto utile poiché, oltre a garantire il buon funzionamento del sistema, ci ha introdotto alcune delle innumerevoli problematiche che possono verificarsi durante la fase dello sviluppo di un sistema software da parte di un gruppo di sviluppatori, e ci ha orientato verso il giusto modo per trovare ed applicare la soluzione migliore per la loro risoluzione.

CONCLUSIONI

Grazie al lavoro che abbiamo svolto, il tool TAXI adesso è dotato di un'interfaccia grafica grazie a cui l'utente ha la possibilità di influire attivamente nella generazione automatica delle istanze, selezionando i nodi dall'importanza maggiore di un XML Schema iniziale, assegnando loro un peso. Ciò consente anche di ridurre consistentemente l'insieme delle istanze generabili a partire da un XML Schema in input, grazie all'impiego di tecniche combinatorie che operano in base ai valori inseriti dall'utente e alla strategia di generazione da lui scelta.

Formalizzando gli input dell'applicazione da testare all'interno di un XML Schema, questo strumento renderà sistematico e meno oneroso il processo di generazione di casi di test, che solitamente avviene manualmente per mano di tester esperti, che si occupano di analizzare specifiche del dominio in input, scritte in un linguaggio normale o semiformale.

Confrontando le caratteristiche del tool TAXI con quelle di altri tool si può affermare che TAXI cura nuovi aspetti rispetto a quelli presi in considerazione dagli strumenti affini esistenti, tra cui, senza dubbio, la flessibilità introdotta nella derivazione delle istanze, che avviene tramite un processo controllabile direttamente dall'utente. Anche i risultati prodotti possono essere considerati decisamente positivi e soddisfacenti.

Il lavoro da noi prodotto, tramite cui il tool è stato esteso con nuove e indispensabili funzionalità, e i risultati ottenuti tramite l'ausilio di questo tool, vengono utilizzati a scopi di ricerca, e in relazione ad essi sono stati redatti articoli scientifici come [54], a cura del nostro tutore aziendale e del proponente di questo stage.

APPENDICE

TAXInterface.file

FileNode.java

```
package TAXInterface.file;

import java.util.*;


import org.w3c.dom.*;

import TAXInterface.graphics.*;
import TSS.WeigthsAssignment.*;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

public class FileNode {

          /**
           * Questa classe incapsula un nodo DOM ed implementa alcuni metodi
           * utili per la manipolazione di esso, ad esempio un metodo che
           * restituisce i suoi figli, il suo indice, etc...
           *
           */

          // Il nodo DOM
          protected Node domNode;

          // Il peso attuale del nodo
          private Double peso;

          // FLAG utile per sapere se il peso del nodo e'
          // stato modificato oppure no
          public static boolean IS_MODIFIED_FLAG;

          // numero dei fratelli del nodo
          protected static int brothers = 0;

          // contatore che tiene memoria del numero
          // di chiamate della funzione
          //public static int calls = 0;

          // variabile che indica se l'assegnamento del nuovo
          // peso è avvenuta senza errori.
          // (un nuovo peso non viene inserito se sommato con
          // gli altri pesi gia' modificati da un valore maggiore di 1)
          private boolean ok = true;

          // peso del nodo prima dellultima modifica
          private Double vecchio = new Double(0.0);

          // Array di nomi per i tipi di nodo DOM
          // (Indici dell'array = valori possibili restituiti da nodeType())
          static final String[] typeName = { "none", "Element", "Attr", "Text",
                    "CDATA", "EntityRef", "Entity", "ProcInstr", "Comment", "Document",
                    "DocType", "DocFragment", "Notation" };

          /**
           *  Costruisce un FileNode a partire da un nodo DOM.
           *  Il peso iniziale del nodo viene prelevato da una
           *  strittura dati statica (NodeMap) contenente i pesi
           *  di tutti i nodi figli di un nodo choice.
           */
```

```
	public FileNode(Node node) {

		domNode = node;
		this.peso = NodeMap.get(new Integer(domNode.hashCode()));
	}

	/**
	 *  Costruisce un FileNode a partire da un nodo DOM.
	 *  Il peso iniziale del nodo viene assegnato dal chiamante
	 *  passandolo come parametro
	 */
	public FileNode(Node node, Double peso) {

		domNode = node;
		this.peso = peso;
	}

	/**
	 * @method                                      getDom()
	 *
	 * @description                        Restituisce il nodo dom incapsulato dal FileNode
	 *
	 * @return domNode  -         il nodo dom
	 */
	public Node getDom() {
		return domNode;
	}

	/**
	 * @method          hashCode()
	 *
	 * @description                                 Calcola il codice hash del FileNode a partire da
	 *                                                                                  il nodo dom che incapsula
	 *
	 * @return domNode.hashCode() - codice hash della variabile istanza domNode
	 *                                                                                  ( usato per l'inserimento
nelle HashMap )
	 */
	public int hashCode() {

		return domNode.hashCode();
	}

	/**
	 *        getNodePrefix()
	 *
	 *        Individua il prefisso all'interno del nome di un nodo
	 *
	 *        @return String - il prefisso contenuto nel nome di un nodo DOM
	 *                         null   - se non ho prefisso
	 */
	public String getNodePrefix() {

		String n = this.StringNodeName();
		int i;
		char twoPoints = ' ';

		// cerco i due punti all'interno del nome del nodo
		for (i = 0; i < n.length(); i++) {

			twoPoints = n.charAt(i);

			if (twoPoints == ':')
				break;
			else
				twoPoints = ' ';

		}

		// se non li trovo significa che non ho prefisso
		// quindi restituisco null.
		// se li trovo restituisco la sottostringa che
		// rappresenta il prefisso
		if (i == n.length() && twoPoints == ' ')
			return null;
		else
			return n.substring(0, i);
	}

	/**
	 *        getChildren()
	 *
	 *        Restituisce i figli del nodo su cui e invocato
	 *        eliminando i figli di tipo testo e commento
	 *        affinche' non vengano visualizzati
	 *        nell'interfaccia grafica
	 *
	 *        @return Object[] children -  un array di figli del nodo dom esclusi i figli
	 *                                                                                  di tipo text
```

```
 */
public Object[] getChildren() {

          NodeList list = domNode.getChildNodes();
          int count = list.getLength();
          int realCount = count / 2;

          // un figlio con nome  'ISTI_et_CNR_pesi' e'
          // un figlio fittizio introdotto da noi quindi
          // non va calcolato nel numero di figli effettivi
          if (hasIstiChild())
                    realCount = realCount - 1;

          FileNode[] children; // = new FileNode[count];
          // se domNode è di tipo 'Document' i figli sono gli stessi del nodo dom
          if (typeName[domNode.getNodeType()].equals(typeName[9])) {
                    // se a figli di tipo Comment non li aggiungo all'array di figli
                    // affinche non vengano vusualizzati nell'interfaccia
                    if (hasCommentChild()) {
                              int size = count;
                              for (int i = 0; i < count; i++) {
                                        FileNode tmp = new FileNode(list.item(i));
                                        if (!tmp.StringNodeType().equals("Comment")
                                                  && !tmp.StringNodeType().equals("Text"))
                                                  size--;
                              }
                              children = new FileNode[size];
                              int k = 0;
                              for (int i = 0; i < list.getLength(); i++) {
                                        FileNode tmp = new FileNode(list.item(i));
                                        if (!tmp.StringNodeType().equals("Comment")
                                                  && !tmp.StringNodeType().equals("Text")) {
                                                  children[k] = tmp;
                                                  k++;
                                        }
                              }
                              return children;
                    } else {
                              children = new FileNode[count];
                              for (int i = 0; i < count; i++) {
                                        FileNode tmp = new FileNode(list.item(i));
                                        children[i] = tmp;
                              }
                              return children;
                    }
          } else {

                    // ci occupiamo di scartare tutti i nodi di tipo testo, commento
                    // e di tipo 'ISTI_et_CNR_pesi', se esistono,
                    // affinche non compaiano nella visualizzazione
                    // del documento nell'interfaccia grafica

                    Vector child = new Vector();

                    if (!hasIstiChild()) {
                              for (int i = 1; i < count - 1; i += 2)
                                        child.addElement(new FileNode(list.item(i)));
                    } else {
                              for (int i = 1; i < count - 2; i += 2)
                                        child.addElement(new FileNode(list.item(i)));
                    }
                    for (int i = 0; i < child.size(); i++) {
                              if (((FileNode) child.elementAt(i)).StringNodeName().equals(
                              "#comment"))
                                        child.remove(i);
                    }

                    children = new FileNode[child.size()];

                    for (int i = 0; i < child.size(); i++) {
                              children[i] = (FileNode) child.elementAt(i);
                    }
                    return children;
          }

}

/**
 *        setPeso()
 *
 *        Assegna un nuovo peso al nodo rispettando i valori
 *        precedentemente assegnati ai fratelli del nodo
 *
 *        @param pesoNuovo - il nuovo peso da assegnare al nodo
 */
public void setPeso(Double pesoNuovo) {

          // salvataggio del vecchio valore asseganto al nodo
          // nel caso in cui il nuovo peso inserito non fosse
          // legale il vecchio viene ripristinato
```

```
                    this.vecchio = NodeMap.get(new Integer(hashCode()));
                    double oldValue = this.vecchio.doubleValue();

                    // se il peso nuovo e' negativo informiamo l'utente del suo errore
                    // con un messaggio di errore
                    if (peso.doubleValue() < 0) {
                              new Alert("Weigths must be positive!");
                              this.peso = new Double(oldValue);
                    } else {

                              if (ParserXmlModel.is_this && oldValue != pesoNuovo.doubleValue())
                                        FlagMap.put(new Integer(hashCode()), new Boolean(true));

                              String p = pesoNuovo.toString();
                              FileNode parent = this.getParentNode();
                              int n = parent.childCount() / 2;

                              if (n > 0) {
                                        if (n > 1) {
                                                  // i pesi avranno al massimo 4 cifre decimali
                                                  // quindi nel caso in cui l'utente inserisca un
                                                  // numero compreso tra 0 e 1 con troppe cifre decimali
                                                  // viene avvertito dell'errore e viene ripristinato
                                                  // il vecchio valore
                                                  if (p.length() > 6 && pesoNuovo.doubleValue() < 1
                                                            && pesoNuovo.doubleValue() > 0) {
                                                            p = p.substring(0, 6);

                                                            new Alert("WARNING: Too much cyphers!");

                                                            this.peso = vecchio; //Double.valueOf(p);
                                                            update(this, this.peso);
                                                            if (!ok)
                                                                      update(this, this.peso);
                                                  } else if (p.length() <= 6 && pesoNuovo.doubleValue() < 1
                                                            && pesoNuovo.doubleValue() > 0) {
                                                            this.peso = pesoNuovo;
                                                            if (p.contains("E")) {
                                                                      int m = Integer.parseInt(""
                                                                                + p.charAt(p.length() - 1));
                                                                      if (m > 4) {
                                                                                new Alert("WARNING: Too much
cyphers!");
                                                                                this.peso = vecchio;
                                                                      }
                                                            }
                                                            update(this, this.peso);
                                                            if (!ok)
                                                                      update(this, this.peso);
                                                  } else if (pesoNuovo.doubleValue() > 1) {
                                                            // non si accettano pesi maggiori di 1
                                                            // perchè la somma di tutti i pesi deve essere 1
                                                            new Alert("Weights you insert must be smaller than 1!");
                                                            /**if (oldValue == default_value.doubleValue())
                                                             this.peso = default_value;
                                                             else*/
                                                            this.peso = new Double(oldValue);
                                                            update(this, this.peso);
                                                            if (!ok)
                                                                      update(this, this.peso);
                                                  } else if (pesoNuovo.doubleValue() == 1) {

                                                            if (!this.getParentNode().isFirstLevelElement()) {
                                                                      new Alert(
                                                                      "Weights you insert must be smaller than 1!");
                                                                      this.peso = new Double(oldValue);
                                                                      update(this, this.peso);
                                                                      if (!ok)
                                                                                update(this, this.peso);
                                                            } else {
                                                                      this.peso = pesoNuovo;
                                                                      update(this, this.peso);
                                                                      if (!ok)
                                                                                update(this, this.peso);
                                                            }
                                                  } else if (pesoNuovo.doubleValue() == 0) {
                                                            // non sono consentiti pesi uguali a 0
                                                            // se non alle choice di primo livello

                                                            if (!this.getParentNode().isFirstLevelElement()) {
                                                                      new Alert(
                                                                      "Weights you insert must be bigger than 0!");

                                                                      this.peso = new Double(oldValue);
                                                                      update(this, this.peso);
                                                                      if (!ok)
                                                                                update(this, this.peso);
                                                            } else {
                                                                      this.peso = new Double(0.0);
                                                                      update(this, this.peso);
```

```
                                                                        if (!ok)
                                                                                update(this, this.peso);
                                                                }

                                                        }

                                                } else if (n == 1) {
                                                        new Alert(
                                                        "When a choice has only one child its weigth must be 1!");
                                                        this.peso = new Double(1.0);
                                                        update(this, this.peso);
                                                        if (!ok)
                                                                update(this, this.peso);
                                                }
                                        }
                                }
                                // ogni volta che è stato inserito un peso che rispetta i parametri
                                // vengono aggiornati i valori dei pesi contenuti nella mappa di pesi
                                // tramite una chiamata del metodo update()

        }

        /**
         *        update()
         *
         *        Aggiorna i valori dei pesi contenuti nella mappa
         *        ad ogni peso nuovo che viene inserito
         *
         *        @param node                   - il nodo il cui peso e' modificato
         *        @param aValue       - il nuovo peso da assegnare al nodo
         */
        public void update(FileNode node, Double aValue) {

                // i pesi vengono aggiornati secondo la seguente politica:

                NodeMap.put(new Integer(((FileNode) node).hashCode()), (Double) aValue);

                FileNode[] fratelli = (FileNode[]) ((FileNode) node).getParentNode()
                .getChildren();
                Vector fratelli_modif = new Vector();

                int sum = 0;
                /*
                 *  a)    Per ogni fratello del nodo (compreso il nodo stesso)
                 *        si verifica se e' stato modificato verificando il
                 *        valore booleano associato nella mappa contenente
                 *        l'informazione modificato/non modificato (FlagMap).
                 *        Vengono aggiornati solo i pesi che non sono stati assegnati
                 *        direttamente dall'utente, quindi il cui flag e' settato
                 *        a false (non modificato).
                 *        Quindi viene calcolata la somma dei pesi modificati e
                 *        il numero di fratelli modificati fino al momento
                 *        dell'invocazione.
                 *
                 */
                for (int i = 0; i < fratelli.length; i++) {

                        if (FlagMap.containsKey(new Integer(fratelli[i].hashCode()))) {

                                Boolean mod = FlagMap.get(new Integer(fratelli[i].hashCode()));

                                if (mod.booleanValue()) {
                                        // Si lavora con gli interi per evitare errori di arrotondamento
                                        // che si verificano utilizzando i double.
                                        // Dovendo utilizzare al massimo 4 cifre decimali
                                        // per ottenere degli interi dovremo
                                        // moltiplicare tutto per 10000
                                        int to_sum = (int) (((Double) NodeMap.get(new Integer(
                                                        fratelli[i].hashCode()))).doubleValue() * 10000);
                                        sum += to_sum;
                                        fratelli_modif.addElement(fratelli[i]);
                                }
                        }
                }

                /*
                 *
                 *        b)        Una volta calcolato il numero di fratelli modificati
                 *                  e la somma dei pesi loro assegnati si procede facendo
                 *                  controlli sulla somma dei pesi inseriti fino a questo
                 *                  momento per verificare se il peso inserito e' accettabile
                 *
                 */
                int ff = fratelli.length - fratelli_modif.size();

                /*
                 *
                 *                  Se la il numero di fratelli modificati e' minore
                 *                  della somma totale dei fratelli e...
                 *
```

```
                 */
                if (fratelli_modif.size() < fratelli.length) {

                        /*
                         *
                         *                  ...il nodo con cui lavoriamo non è figlio di una
                         *                  choice di primo livello (aggiunta quindi in seguito
                         *                  al documento)...
                         *
                         */
                        if (!node.getParentNode().isFirstLevelElement()) {

                                /*
                                 *
                                 *                  ...se la somma e' minore di 10000 (quindi in realta'
                                 *                  minore di 1) il nuovo peso inserito dall'utente
                                 *                  puo' essere ritenuto valido. Si aggiorna quindi
                                 *                  La mappa dei pesi e la mappa dei flag settando il nodo
                                 *                  corrente a modificato
                                 *
                                 */
                                if (sum < 10000) {

                                        int rm = (10000 - sum) / (ff);

                                        Integer val = new Integer(rm);
                                        String v = val.toString();

                                        if (v.length() > 6) {

                                                v = v.substring(0, 6);
                                                val = Integer.valueOf(v);
                                        }

                                        double xs = val.doubleValue() / 10000;
                                        Double newVal = new Double(xs);
                                        for (int i = 0; i < fratelli.length; i++) {

                                                if (!FlagMap.get(new Integer(fratelli[i].hashCode()))
                                                                .booleanValue()) {
                                                        NodeMap.put(new Integer(fratelli[i].hashCode()),
                                                                        newVal);
                                                }
                                        }
                                } else {

                                        /*
                                         *
                                         *                  ...altrimenti si ripristina il vecchio valore
                                         *                  nella mappa e si informa l'utente di quanto
avvenuto
                                         *                  tramite un opportuno messaggio d'errore
                                         *
                                         */
                                        this.peso = vecchio;//new Double(oldValue);
                                        FlagMap.put(new Integer(hashCode()), new Boolean(false));
                                        this.ok = false;
                                        String s = "The sum of weights you modified\n";
                                        s += "is bigger than 1 and I can't assign\n";
                                        s += "a 0.0 value or a negative value to\n ";
                                        s += "not-modified nodes!\n";
                                        s += "Modify the weigths you inserted!";
                                        new Alert(s);

                                        //aValue = oldValue;
                                        //NodeMap.put(new Integer(((FileNode) node).hashCode()), (Double)
aValue);
                                }
                        }
                        /*
                         *
                         *                  ...altrimenti se il nodo corrente e' figlio di una choice
                         *                  di primo livello si fanno controlli diversi
                         *                  in quanto ai figli di una choice di primo livello
                         *                  e' consentito assegnare anche un peso pari a zero
                         *
                         */
                        else {

                                /*
                                 *                  ...se la somma dei pesi e' minore o uguale a uno
                                 *                  allora si procede come sopra, facendo gli
                                 *                  opportuni aggiornamenti delle mappe, assegnando
                                 *                  automaticamente agli altri fratelli peso 0.
                                 *
                                 */
                                if (sum <= 10000) {

                                        int rm = (10000 - sum) / (ff);
```

```
                                        Integer val = new Integer(rm);
                                        String v = val.toString();

                                        if (v.length() > 6) {

                                                v = v.substring(0, 6);
                                                val = Integer.valueOf(v);
                                        }

                                        double xs = val.doubleValue() / 10000;
                                        Double newVal = new Double(xs);
                                        for (int i = 0; i < fratelli.length; i++) {

                                                if (!FlagMap.get(new Integer(fratelli[i].hashCode()))
                                                                .booleanValue()) {
                                                        NodeMap.put(new Integer(fratelli[i].hashCode()),
                                                                        newVal);
                                                }
                                        }
                                }/* else if(sum == 10000){
                                int rm = (10000 - sum) / (ff);

                                Integer val = new Integer(rm);
                                String v = val.toString();

                                if (v.length() > 6) {


                                v = v.substring(0, 6);
                                val = Integer.valueOf(v);
                                }

                                double xs = val.doubleValue() / 10000;
                                Double newVal = new Double(xs);
                                for (int i = 0; i < fratelli.length; i++) {

                                if (!FlagMap
                                .get(new Integer(fratelli[i].hashCode())).booleanValue()) {
                                NodeMap
                                .put(new Integer(fratelli[i].hashCode()),
                                newVal);
                                }
                                }
                                }*/
                                /*
                                 *
                                 *                  ...altrimenti si comunica all'utente dell'errore
                                 *                  tramite un opportuno messaggio
                                 *
                                 */
                                else {
                                        this.peso = vecchio;//new Double(oldValue);
                                        FlagMap.put(new Integer(hashCode()), new Boolean(false));
                                        this.ok = false;
                                        String s = "The sum of weights you modified\n";
                                        s += "is bigger than 1 and I can't assign\n";
                                        s += "a negative value to not-modified nodes!\n";
                                        s += "Modify the weigths you inserted!";
                                        new Alert(s);
                                }
                        }
                        /*
                         *
                         *                  ...se invece sono stati modificati tutti i fratelli e...
                         *
                         */
                } else if (fratelli_modif.size() == fratelli.length) {

                        /*
                         *
                         *                  ...la somma e' minore di 1 si cominica all'utente
                         *                  di modificare i pesi inseriti in quanto non sono
                         *                  ammessi pesi la cui somma sia minore di 1.
                         *
                         */
                        if (sum < 10000) {
                                this.peso = vecchio;//new Double(oldValue);

                                this.ok = false;
                                String s = "The sum of weights you modified\n";
                                s += "is smaller than 1 and I can't accept\n";
                                s += "weigths whose sum is not equal to 1!\n";
                                s += "Modify the weigths you inserted!";
                                new Alert(s);
                                FlagMap.put(new Integer(hashCode()), new Boolean(false));
                                //aValue = oldValue;
                                //NodeMap.put(new Integer(((FileNode) node).hashCode()), (Double) aValue);
                        }
                        /*
                         *
```

```
                     *                  ...la somma e' maggiore di 1 si cominica all'utente
                     *                  di modificare i pesi inseriti in quanto non sono
                     *                  ammessi pesi la cui somma sia maggiore di 1.
                     *
                     */
                    if (sum > 10000) {
                              this.peso = vecchio;//new Double(oldValue);
                              FlagMap.put(new Integer(hashCode()), new Boolean(false));
                              this.ok = false;
                              String s = "The sum of weights you modified\n";
                              s += "is bigger than 1 and I can't accept\n";
                              s += "weigths whose sum is not equal to 1!\n";
                              s += "Modify the weigths you inserted!";
                              new Alert(s);

                              //aValue = oldValue;
                              //NodeMap.put(new Integer(((FileNode) node).hashCode()), (Double) aValue);
                    }
          }
}

/**
 *        getPeso()
 *
 *        Ritorna il peso del nodo su cui e' invocato
 *
 *        @return peso        - il peso del nodo corrente
 */
public Double getPeso() {
          return this.peso;
}

/**
 *        toString()
 *
 *        Restituisce una stringa che identifica
 *        il nodo nell'albero
 *
 *        @return s - la stringa contenente tipo e nome del nodo
 */
public String toString() {
          // salviamo in s il tipo del nodo

          if(!isElementsList()){
                    //String s = typeName[domNode.getNodeType()];
                    String nodeName = domNode.getNodeName();

                    String s = "";

                    if (!nodeName.startsWith("#")) {
                    //        s += ": " + nodeName;
                              s = nodeName;
                    }
                    if(nodeName.equals("#document"))
                              s = "Document";
                    // salviamo anche il valore del nodo
                    if (domNode.getNodeValue() != null) {

                              if (s.startsWith("ProcInstr"))
                                        s += ", ";
                              else
                                        s += ": ";

                              // trim su t
                              String t = domNode.getNodeValue().trim();
                              int x = t.indexOf("\n");
                              if (x >= 0)
                                        t = t.substring(0, x);
                              s += t;

                    } else {
                              if (s.startsWith("ProcInstr"))
                                        s += ", ";

                              NamedNodeMap map = null;
                              // salviamo il valore dell'attributo nome del nodo
                              if (domNode.hasAttributes()) {

                                        map = domNode.getAttributes();

                                        for (int i = 0; i < map.getLength(); i++) {
                                                  if (map.item(i).getNodeName().equalsIgnoreCase("name")) {
                                                            s += ": ";
                                                            s += map.item(i).getNodeValue();
                                                            break;
                                                  }
                                        }

                              }
                    }
```

```
                    return s;
                    // se abbiamo un elemento di primo livello allora
                    // restituiamo la stringa da visualizzare ElementList
          } else return "ElementsList";
}

/**
 *        StringNodeType()
 *
 *        Restituisce il tipo del nodo
 *
 *        @return nodeType    - il tipo del nodo
 */
public String StringNodeType() {

          // node type
          String nodeType = typeName[domNode.getNodeType()];
          return nodeType;
}

/**
 *        StringNodeName()
 *
 *        Restituisce il nome del nodo
 *
 *        @return nodeName    - il nome del nodo
 */
public String StringNodeName() {
          // node name
          String nodeName = domNode.getNodeName();
          return nodeName;
}

/**
 *        StringNodeValue()
 *
 *        Restituisce il valore del nodo
 *
 *        @return nodeValue   - il valore del nodo
 */
public String StringNodeValue() {
          // node value
          String nodeValue = domNode.getNodeValue();
          return nodeValue;
}

/**
 *        index()
 *
 *        Calcola l'indice del nodo passato come parametro
 *        e viene invocato sul padre di tale nodo
 *
 *        @param child        - il figlio di cui si deve calcolare l'indice
 *        @return l'indice del nodo passato come parametro
 */
public int index(FileNode child) {
          // calcoliamo l'indice del nodo escludento i nodi testo
          int count = childCount();
          for (int i = 0; i < count; i++) {
                    FileNode n = this.child(i);
                    if (child(i).StringNodeName().equals("ISTI_et_CNR_pesi"))
                              return -1;
                    if (child.domNode == n.domNode)
                              return i / 2;
          }
          return -1; // Should never get here.
}

/**
 *        child()
 *
 *        Restituisce il nodo di indice searchIndex
 *
 *        @param searchIndex                    - l'indice del nodo da trovare
 *        @return new FileNode(node)    - il nodo cercato incapsulato in un FileNode
 */
public FileNode child(int searchIndex) {

          org.w3c.dom.Node node = domNode.getChildNodes().item(searchIndex);
          return new FileNode(node);
}

/**
 *        getParentNode()
 *
 *        Restituisce il padre del nodo su cui e' invocato
 *
 *        @return new FileNode(node)    -         il padre del nodo su cui è invocato
 *                                                                                se esiste
 *                            null                                    -         altrimenti
```

```
 */
public FileNode getParentNode() {

          org.w3c.dom.Node node = domNode.getParentNode();
          if (node == null)
                    return null;
          return new FileNode(node);
}

/**
 *        hasParentNode()
 *
 *        Controlla se un nodo ha un nodo padre
 *
 *        @return   true      -         se il nodo su cui e invocato
 *                                                  ha padre
 *                            false     -         altrimenti
 *
 */
public boolean hasParentNode() {

          org.w3c.dom.Node node = domNode.getParentNode();
          if (node == null)
                    return false;
          return true;
}

/**
 *        getPreviousSibling()
 *
 *        Restituisce il fratello precedente se esiste
 *
 *        @return   new FileNode(node)  -         il fratello precedente se esiste
 *                            null                                    -         altrimenti
 */
public FileNode getPreviousSibling() {
          org.w3c.dom.Node node = domNode.getPreviousSibling();
          if (node == null)
                    return null;
          return new FileNode(node);
}

/**
 *        getNextSibling()
 *
 *        Restituisce il fratello successivo se esiste
 *
 *        @return   new FileNode(node)  -         il fratello successivo se esiste
 *                            null                                    -         altrimenti
 */
public FileNode getNextSibling() {

          org.w3c.dom.Node node = domNode.getNextSibling();
          if (node == null)
                    return null;
          return new FileNode(node);
}

/**
 *        getFirstChild()
 *
 *        Restituisce il primo figlio
 *
 *        @return   new FileNode(node)  -         il primo figlio del nodo, se ha figli
 *                            null                                    -         altrimenti
 */
public FileNode getFirstChild() {

          org.w3c.dom.Node node = domNode.getFirstChild();

          if (node == null)
                    return null;

          return new FileNode(node);
}

/**
 *        hasFirstChild()
 *
 *        Controlla se il nodo ha almeno un figlio
 *
 *        @return true        -         se viene individuato almeno un figlio
 *                            false     -         altrimenti
 */
public boolean hasFirstChild() {

          org.w3c.dom.Node node = domNode.getFirstChild();
          // se ha solo figli di tipo text si restituisce false

          if (node == null)
```

```
                return false;
        if (((int) getChildren().length / 2) != 0) {
                return true;
        } else
                return false;
}

/**
 *      getLastChild()
 *
 *      Restituisce l'ultimo figlio
 *
 *      @return   new FileNode(node)  -         l'ultimo figlio del nodo, se ha figli
 *                        null                                  -         altrimenti
 */
public FileNode getLastChild() {

        org.w3c.dom.Node node = domNode.getLastChild();
        if (node == null)
                return null;
        return new FileNode(node);
}

/**
 *      childCount()
 *
 *      Conta il numero dei figli di un nodo
 *
 *      @return   count     -         il numero di figli del nodo
 *                                                  (compresi i nodi di tipo Text)
 */
public int childCount() {

        int count = domNode.getChildNodes().getLength();
        return count;
}

/**
 *      hasPreviousSibling()
 *
 *      Controlla se il nodo ha un fratello che lo precede
 *
 *      @return   true      -         se ha un fratello che lo precede
 *                        false     -         altrimenti
 */
public boolean hasPreviousSibling() {
        Node node = domNode.getPreviousSibling();

        if (node == null)
                return false;
        else {
                Node n = node.getPreviousSibling();
                if (n == null)
                        return false;
        }
        return true;
}

/**
 *      hasNextSibling()
 *
 *      Controlla se il nodo ha un fratello che lo segue
 *
 *      @return   true      -         se ha un fratello che lo segue
 *                        false     -         altrimenti
 */
public boolean hasNextSibling() {
        Node node = domNode.getNextSibling();
        if (node == null)
                return false;
        return true;
}

/**
 *      isChoice()
 *
 *      Verifica se il nodo è di tipo choice. Vengono trattati
 *      come nodi choice anche gli elementi di primo livello
 *      in modo da poter assegnare un peso ai loro figli
 *
 *      @return   true      -         se il nodo e' una choice
 *                        false     -         altrimenti
 */
public boolean isChoice() {

        // se il nome del nodo contiene 'choice'
        // oppure se si tratta di un elemento
        // di primo livello
        // si restituisce true
        String prefix = getNodePrefix();
```

```
          String choice;

          if (prefix == null) {
                    choice = "choice";
          } else {
                    choice = prefix + ":choice";
          }
          if (StringNodeName().equalsIgnoreCase(choice) || isElementsList()) {
                    return true;
          } else
                    return false;
}

/**
 *        isFirstLevelElement()
 *
 *        Verifica se il nodo è una nodo di primo livello
 *        (quindi aggiunto dal programmatore)
 *
 *        @return   true      -         se si tratta di un elemento di primo livello
 *                            false     -         altrimenti
 */
public boolean isFirstLevelElement() {

          boolean yes = false;
          // se una choice ha fra i suoi attributi
          // ISTI_et_CNR_imaginary_ROOT_attribute
          // è una choice di primo livello
          if (isElementsList())
                    yes = true;

          return yes;
}



/**
 * equals()
 *
 * Verifica se il nodo passato come parametro e' uguale
 * a quello su cui e' invocato confrontando i loro
 * codici hash
 *
 * @return          true      -         se i due nodi sono uguali
 *                            false     -         altrimenti
 *
 */

public boolean equals(Object node) {

          // controlliamo se i codici hash sono uguali
          if (domNode.hashCode() == ((FileNode) node).domNode.hashCode())
                    return true;
          else
                    return false;
}

/**
 *        hasIstiChild()
 *
 *        Verifica se ha un figlio con nome ISTI_et_CNR_pesi
 *
 *        @return   true      -         se il nodo ha un figlio di nome ISTI_et_CNR_pesi
 *                            false     -         altrimenti
 */
public boolean hasIstiChild() {

          NodeList list = domNode.getChildNodes();
          boolean yes = false;
          // scorre la lista di figli e controlla se ce n'è uno
          // con nome ISTI_et_CNR_pesi
          for (int i = 0; i < list.getLength(); i++) {
                    if (list.item(i).getNodeName().equals("ISTI_et_CNR_pesi")) {
                              yes = true;
                              break;
                    }
          }
          return yes;
}

/**
 *        getIstiVal()
 *
 *        Restituisce il valore contenuto nel figlio di nome ISTI_et_CNR_pesi
 *
 *        @return   isti      -         il valore contenuto nel figlio di nome ISTI_et_CNR_pesi
 */
public double getIstiVal() {
```

```
          double isti = 0;
          // se ha un figlio con nome ISTI_et_CNR_pesi
          // restituisce il peso inserito nell'attributo peso
          if (hasIstiChild()) {
                    NodeList list = domNode.getChildNodes();

                    for (int i = 0; i < list.getLength(); i++) {
                              if (list.item(i).getNodeName().equals("ISTI_et_CNR_pesi")) {

                                        NamedNodeMap nnp2 = list.item(i).getAttributes();
                                        for (int j = 0; j < nnp2.getLength(); j++) {

                                                  if (nnp2.item(j).getNodeName().equals("peso")) {

                                                            isti = Double.parseDouble(nnp2.item(j)
                                                                      .getNodeValue());
                                                            break;
                                                  }
                                        }
                              }
                    }
          }
          return isti;
}

/**
 *        hasCommentChild()
 *
 *        Verifica se il nodo ha un figlio di tipo Comment
 *
 *        @return   true      -         se il nodo ha figli di tipo comment
 *                            false     -         altrimenti
 */
public boolean hasCommentChild() {

          NodeList l = domNode.getChildNodes();
          boolean ok = false;
          // scorro la lista dei figli e verifico se
          // ne possiede almeno uno di tipo 'Comment'
          for (int i = 0; i < l.getLength(); i++) {

                    System.out.println("OK: " + l.item(i).getNodeName());
                    if (l.item(i).getNodeName().equals("#comment"))
                              ok = true;
          }

          return ok;
}

public String toString2() {

          String s = toString();
          s += "; Peso: " + getPeso();

          return s;
}

/**
 *        hasChoiceInPath()
 *
 *        Verifica, visitando l'albero ricorsivamente se un nodo
 *        contiene nei suoi sottoalberi un elemento di tipo choice
 *
 *        @return   true      -         se il nodo contiene nei suoi sottoalberi
 *                                                  un elemento di tipo choice
 *                            false     -         altrimenti
 */
public boolean hasChoiceInPath() {

          boolean ok = false;

          // visita ricorsivamente per scoprire
          // se un nodo appartiene ad un
          // path contenente un nodo 'choice'
          //if (isChoice())
                    //ok = true;
          //else {
                    // se il nodo ha figli controllo se almeno
                    // uno di essi e' una choice.
                    // In caso contrario inolto la visita ad ognuno
                    // dei figli interrompendo non appena un nodo
                    // di tipo choice viene trovato.

                    FileNode[] l = (FileNode[]) getChildren();

                    if (l != null) {
                              for (int i = 0; i < l.length; i++){
                                        //ok = ok || l[i].hasChoiceInPath();

                                        if(l[i].isChoice()){
```

```
                                                  ok = true;
                                                  break;
                                        } else {

                                                  ok = ok || l[i].hasChoiceInPath();
                                        }
                                }
                        }
                //}
                return ok;
        }

        /**
         *        isElementsList()
         *
         *        Verifica se il nodo contiene all'interno del nome
         *        la stringa ISTI_et_CNR_first_LEVEL_element. In questo
         *        caso si tratta di un elemento di primo livello
         *
         *        @return   true      -         se il nodo e' un elemento di
         *                                                primo livello
         *                  false     -         altrimenti
         */
        public boolean isElementsList() {

                boolean yes = false;
                String prefix = getNodePrefix();
                String elem = "";
                if(prefix != null)
                        elem = prefix + ":ISTI_et_CNR_first_LEVEL_element";
                else
                        elem = "ISTI_et_CNR_first_LEVEL_element";
                // se il nodo ha un nome contenente
                // ISTI_et_CNR_first_LEVEL_element
                // è un elemento di primo livello
                if (this.StringNodeName().equals(elem))
                        yes = true;

                return yes;
        }

}
```

ParserXmlModel.java

```
package TAXInterface.file;


import java.lang.reflect.*;

import TAXInterface.graphics.*;
import TAXInterface.table.*;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

/**
 *
 *        Rappresenta il modello che utilizzeremo per creare
 *        la nostra TreeTable
 *
 */

public class ParserXmlModel extends AbstractTreeTableModel implements
TreeTableModel {

    //public static FileNode parent;

    //public static FileNode modified;

    public static int count = 0;

   // public boolean oneModify = false;
```

```
public static boolean CAN_EDIT = false;

private static String[] methodNames = { "getPeso" };

public static boolean is_this = true;

/**
 * Metodi setter usati per settare valori.  Usati dal metodo
 * setValueAt di TableModel. Un array o una entry nulli
 * indicano che la colonna non e' editabile
 */
private static String[] setterMethodNames = { "setPeso" };

// Nomi delle colonne.
static protected String[] cNames = { "Tree Nodes", "Weigth Value" };

// Tipi di dato contenuti nelle colonne
static protected Class[] cTypes = { TreeTableModel.class, Double.class };


/**
 *
 *    Costruttore. Sovrascrive quello della superclasse
 *
 */
public ParserXmlModel() {
      super(new FileNode(TreeTableExample.document));
}

/**
 * getChildren()
 *
 * Restituisce un array contenente i figli del
 * nodo passato come parametro
 *
 * @param       node                -         il nodo di cui calcolare i figli
 * @return      Object[]  -         un array contenente i figli del nodo
 */

protected Object[] getChildren(Object node) {

    FileNode fileNode = ((FileNode) node);
    return fileNode.getChildren();
}

/*
 * L'interfaccia TreeModel
 */

/**
 * getChildCount()
 *
 * @param       node                -         il nodo di cui calcolare il numero di figli
 * @return      Object[]  -         il numero dei figli del nodo
 */
public int getChildCount(Object node) {

    Object[] children = getChildren(node);
    // 0 oppure la lunghezza dell'array di figli
    return (children == null) ? 0 : children.length;

}

/**
 * getChild()
 *
 * @param       node                -         il nodo padre
 * @param       i                             -         l'indice del figlio che vogliamo
 * @return      Object              -         il figlio di node di indice i
 */
public Object getChild(Object node, int i) {
    return getChildren(node)[i];
}


/**
 * getColumnCount()
 *
 * @return      int  -    il numero di colonne della tabella
 */
public int getColumnCount() {
    return cNames.length;
}

/**
 * getColumnName()
 *
 * @param       column    -         l'indice della colonna che vogliamo
 * @return      String  - il nome della colonna di indice column
 *
```

```
   */
  public String getColumnName(int column) {
      return cNames[column];
  }

  /**
   * getColumnClass()
   *
   * @param       column    -         l'indice della colonna che vogliamo
   * @return      String  - il tipo della colonna di indice column
   *
   */
  public Class getColumnClass(int column) {
      return cTypes[column];
  }


 /**
  * getValueAt()
  *
  * @param        node      -         il nodo du cui vogliamo ottenere il valore
  * @param        column    -         la colonna in cui vogliamo editare
  * @return       Object    -         il peso del nodo
  *
  */

  public Object getValueAt(Object node, int column) {

      try {
          Method method = node.getClass().getMethod(methodNames[0], null);
          if (method != null) {
              if (((FileNode) node).hasParentNode() && column == 1) {
                  // se il nodo e' una choice
                  // si invoca il metodi di getPeso di FileNode
                  // altrimenti si restituisce null
                  if (isChoiceChild(node))
                      return method.invoke(node, null);
                  else
                      return null;
              }
          }
      } catch (Throwable th) {

          System.out.println("exception: " + th);
      }
      return null;
  }


  public boolean isCellEditable(Object node, int column) {

      switch (column) {
      // nella colonna contenente l'albero
      // si deve poter editare in quanto
      // restituendo false non si puo espandere
      // l'albero
      case 0:
          return true;
      // nella prima colonna si puo scrivere
      // una volta che l'utente ha aperto la sessione
      case 1:
          return CAN_EDIT;
      default:
          return false;
      }
  }

 /**
  * setValueAt()
  *
  * @param         aValue    -         il nuovo valore da settare
  * @param         node      -         il nodo a cui settare il nuovo valore
  * @param         column    -         la colonna in cui scrivere
  */
  public void setValueAt(Object aValue, Object node, int column) {
        //ParserXmlModel.modified = (FileNode) node;

        if (count == 0 && ((FileNode) node).hasParentNode()) {
        //        parent = ((FileNode) node).getParentNode();
                  count++;
        }
        // preleviamo i metodi invocabili sul nodo node
        // ne controlliamo i parametri e se ce n'è uno
        // che fa al caso nostro lo invochiamo
        try {
                  Method[] methods = node.getClass().getMethods();

                  for (int counter = methods.length - 1; counter >= 0; counter--) {
                            if (methods[counter].getName().equals(
                                                setterMethodNames[column -1])
```

```
                                        && methods[counter].getParameterTypes() != null
                                        && methods[counter].getParameterTypes().length == 1) {
                                Class param = methods[counter].getParameterTypes()[0];
                                if (!param.isInstance(aValue)) {
                                        if (!(aValue instanceof Double)
                                                        && ((Double) aValue) == null) {
                                                aValue = new Double(0.0001);
                                        } else {
                                                Constructor cs = param
                                                .getConstructor(new Class[] { Double.class });
                                                if (cs != null) {
                                                        aValue = cs
                                                        .newInstance(new Object[] { aValue });
                                                } else {
                                                        aValue = null;
                                                }
                                        }
                                }

                                methods[counter].invoke(node, new Object[] { aValue });

                                break;
                        }
                }
        } catch (Throwable th) {
                System.out.println("exception: " + th);
        }
    }

    /**
     * Costruisce il path del nodo fino alla radice.
     * La lunghezza dell'array restituito rappresenta
     * la profondita del nodo nell'albero
     *
     * @param       parent              -         il nodo di cui calcolare il path
     * @return      FileNode[]          -         un array contenente tutti i nodi nel path
     */
    public FileNode[] getPathToRoot(FileNode parent) {
        return getPathToRoot(parent);
    }

    /**
     * isChoiceChild()
     *
     * Verifica se il padre del nodo passato come parametro
     * e' figlio di un nodo choice
     *
     * @param       node      -         il nodo
     * @return boolean        -         true se il padre e' una choice
     *                                  false altrimenti
     */
    public boolean isChoiceChild(Object node) {

        FileNode parent = ((FileNode) node).getParentNode();

        if (parent.isChoice()) {
            return true;
        } else
            return false;
    }
}
```

TAXInterface.table

AbstractCellEditor.java

```
package TAXInterface.table;

import java.util.*;

import javax.swing.*;
import javax.swing.event.*;

/**
 * @author          Romina Paradisi
```

```
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

public class AbstractCellEditor implements CellEditor {

          protected EventListenerList listenerList = new EventListenerList();

          /**
           * Restituisce il valore dell'editor della cella
           */
          public Object getCellEditorValue() {
                    return null;
          }

          /**
           * Restituisce true se la cella e' editabile, false altrimenti
           */
          public boolean isCellEditable(EventObject e) {
                    return true;
          }

          /**
           * Restituisce true se la cella pou' essere selezionata, false altrimenti
           */
          public boolean shouldSelectCell(EventObject anEvent) {
                    return false;
          }

          /**
           * Comunica all'editor di terminare l'editing e di accittare qualunque
           * valore editato come valore dell'editor
           */
          public boolean stopCellEditing() {
                    return true;
          }

          /**
           * Coumnica all'editor di non accettare alcun valore editato
           */
          public void cancelCellEditing() {
          }

          /**
           * Aggiunge un listener alla lista dei listener che ascolta qundo l'editor
           * ferma o cancella l'editing
           */
          public void addCellEditorListener(CellEditorListener l) {
                    listenerList.add(CellEditorListener.class, l);
          }

          /**
           * Rimuove un listener dalla lista dei listener
           */
          public void removeCellEditorListener(CellEditorListener l) {
                    listenerList.remove(CellEditorListener.class, l);
          }

          /**
           *
           * Informa tutti gli ascoltatori che sono interessati a sapere quando si
           * verifica questo tipo di evento
           *
           */
          protected void fireEditingStopped() {
                    // Garantisce di non restiuire un array nullo
                    Object[] listeners = listenerList.getListenerList();
                    // Processa tutti gli ascoltatori informando
                    // quelli che sono interessati a questo evento
                    for (int i = listeners.length - 2; i >= 0; i -= 2) {
                              if (listeners[i] == CellEditorListener.class) {
                                        ((CellEditorListener) listeners[i + 1])
                                                            .editingStopped(new ChangeEvent(this));
                              }
                    }
          }

          /**
           *
           * Informa tutti gli ascoltatori che sono interessati a sapere quando si
           * verifica questo tipo di evento
           *
           */
          protected void fireEditingCanceled() {
                    // Garantisce di non restiuire un array nullo
                    Object[] listeners = listenerList.getListenerList();
```

```
                    // Processa tutti gli ascoltatori informando
                    // quelli che sono interessati a questo evento
                    for (int i = listeners.length - 2; i >= 0; i -= 2) {
                              if (listeners[i] == CellEditorListener.class) {
                                        ((CellEditorListener) listeners[i + 1])
                                                  .editingCanceled(new ChangeEvent(this));
                              }
                    }
          }
}
```

AbstractTreeTableModel.java

```
package TAXInterface.table;


import javax.swing.event.*;
import javax.swing.tree.*;

import TAXInterface.file.*;
import TAXInterface.table.TreeTableModel;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

public abstract class AbstractTreeTableModel implements TreeTableModel {
    //     operazioni base di TreeModel
    protected static FileNode root;

    protected EventListenerList listenerList = new EventListenerList();

   /**
    * costruttore
    *
    * @param root - la radice del documento
    */
    public AbstractTreeTableModel(FileNode root) {
        AbstractTreeTableModel.root = root;
    }

    /**
     * @return root - la radice del documento
     */
    public Object getRoot() {
        return root;
    }

    /**
     * restituisce true se un nodo non ha figli
     */
    public boolean isLeaf(Object aNode) {
        return getChildCount(aNode) == 0;
    }

    /**
     * I seguenti metodi verranno implementati e
     * commentati nella superclasse ParserXmlModel
     */

    public int getChildCount(Object parent) {
        return 0;
    }

    public Object getChild(Object parent, int index) {
        return null;
    }


    public void valueForPathChanged(TreePath path, Object newValue) {
    }

    /**
     * restituisce l'indice del nodo child figlio di parent
     */
```

```
    public int getIndexOfChild(Object parent, Object child) {
        for (int i = 0; i < getChildCount(parent); i++) {
            if (getChild(parent, i).equals(child)) {
                return i;
            }
        }
        return -1;
    }

    /*
     * Metodi per aggiungere e rimuovere listeners. (Per soddisfare
     * l'interfaccia TreeModel interface, ma non utilizzati.)
     */

    public void addTreeModelListener(TreeModelListener l) {
        listenerList.add(TreeModelListener.class, l);
    }

    public void removeTreeModelListener(TreeModelListener l) {
        listenerList.remove(TreeModelListener.class, l);
    }

    protected void fireTreeNodesChanged(Object source, Object[] path,
            int[] childIndices, Object[] children) {
//          Garantisce di non restiuire un array nullo
        Object[] listeners = listenerList.getListenerList();
        TreeModelEvent e = null;
//      Processa tutti gli ascoltatori informando
                // quelli che sono interessati a questo evento
        for (int i = listeners.length - 2; i >= 0; i -= 2) {
            if (listeners[i] == TreeModelListener.class) {
                // Lazily create the event:
                if (e == null)
                    e = new TreeModelEvent(source, path, childIndices, children);
                ((TreeModelListener) listeners[i + 1]).treeNodesChanged(e);
            }
        }
    }

    protected void fireTreeNodesInserted(Object source, Object[] path,
            int[] childIndices, Object[] children) {
//          Garantisce di non restiuire un array nullo
        Object[] listeners = listenerList.getListenerList();
        TreeModelEvent e = null;
        // Processa tutti gli ascoltatori informando
                // quelli che sono interessati a questo evento
        for (int i = listeners.length - 2; i >= 0; i -= 2) {
            if (listeners[i] == TreeModelListener.class) {
                // Lazily create the event:
                if (e == null)
                    e = new TreeModelEvent(source, path, childIndices, children);
                ((TreeModelListener) listeners[i + 1]).treeNodesInserted(e);
            }
        }
    }

    protected void fireTreeNodesRemoved(Object source, Object[] path,
            int[] childIndices, Object[] children) {
//          Garantisce di non restiuire un array nullo
        Object[] listeners = listenerList.getListenerList();
        TreeModelEvent e = null;
//      Processa tutti gli ascoltatori informando
                // quelli che sono interessati a questo evento
        for (int i = listeners.length - 2; i >= 0; i -= 2) {
            if (listeners[i] == TreeModelListener.class) {
                // Lazily create the event:
                if (e == null)
                    e = new TreeModelEvent(source, path, childIndices, children);
                ((TreeModelListener) listeners[i + 1]).treeNodesRemoved(e);
            }
        }
    }

    protected void fireTreeStructureChanged(Object source, Object[] path,
            int[] childIndices, Object[] children) {
//          Garantisce di non restiuire un array nullo
        Object[] listeners = listenerList.getListenerList();
        TreeModelEvent e = null;
//      Processa tutti gli ascoltatori informando
                // quelli che sono interessati a questo evento
        for (int i = listeners.length - 2; i >= 0; i -= 2) {
            if (listeners[i] == TreeModelListener.class) {

                if (e == null)
                    e = new TreeModelEvent(source, path, childIndices, children);
                ((TreeModelListener) listeners[i + 1]).treeStructureChanged(e);
            }
        }
    }
```

```
    //
    // Implementazione di default dell'interfaccia TreeTableModel.
    //

    public Class getColumnClass(int column) {
        return Object.class;
    }

    /**
     * Per defaulte rende editabile solo la colonna
     * contenente l'albero.
     */
    public boolean isCellEditable(Object node, int column) {
        return getColumnClass(column) == TreeTableModel.class;
    }

    public void setValueAt(Object aValue, Object node, int column) {
    }

    public void setValueAt(Object aValue, int row, int column) {
    }

    // verranno implementati nella sottoclasse ParserXmlModel:

    public int getColumnCount() {
        return 0;
    }

    public String getColumnName(Object node, int column) {
        return null;
    }

    public Object getValueAt(Object node, int column) {
        return null;
    }

}
```

JTreeTable.java

```
package TAXInterface.table;


import java.awt.*;
import java.awt.event.*;
import java.io.*;

import javax.swing.*;
import javax.swing.table.*;
import javax.swing.tree.*;

import TAXInterface.graphics.*;
import TAXInterface.menu.*;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

/**
 * This example shows how to create a simple JTreeTable component, by using a
 * JTree as a renderer (and editor) for the cells in a particular column in the
 * JTable.
 */

public class JTreeTable extends JTable{
          /**
           *
           */
          private static final long serialVersionUID = -7407830804518518704L;

          public static Double value = new Double(0.0);

          protected static TreeTableCellRenderer tree;
```

```
protected int visibleRow;

private static JPopupMenu menu = null;

private static JMenuItem item1;

private static JMenuItem item2;

private static JMenuItem item3;

private static JMenuItem item4;

public static JMenuItem itemStart;

public static JMenuItem itemStop;

private static JMenu submenuFile;

private static JMenuItem refresh;

private static JMenu submenuSession;

private static JMenu submenu;

private static JMenu submenu2;

static JMenuItem subItem11;

public static JMenuItem subItem12;

static JMenuItem subItem20;

static JMenuItem subItem21;

static JMenuItem subItem22;

static JMenuItem subItem23;



Color initialBackground = new Color(255, 255, 255);

public JTreeTable(TreeTableModel treeTableModel) {
          super();
          setTableHeader(tableHeader);
          setBackground(initialBackground);



          /*********************POPUP****************************** */


          menu = new JPopupMenu();
          // Create and add a menu item

          submenuFile = new JMenu("File");
          item1 = new JMenuItem("Open file", new ImageIcon("icons\\open.gif"));
          item2 = new JMenuItem("Save file", new ImageIcon("icons\\save.gif"));
          item3 = new JMenuItem("Print file", new ImageIcon("icons\\printr01.gif"));
          item4 = new JMenuItem("Reset file", new ImageIcon("icons\\reset.png"));



          //item7 = new JMenuItem("Color Background", new ImageIcon("icons\\color.png"));

          submenuFile.add(item1);
          submenuFile.addSeparator();
          submenuFile.add(item2);
          submenuFile.addSeparator();
          submenuFile.add(item3);
          submenuFile.addSeparator();
          submenuFile.add(item4);



          submenuSession = new JMenu("Session modify");
          itemStart = new JMenuItem("Start modify", new ImageIcon("icons\\startModify.png"));
          itemStop = new JMenuItem("Stop modify", new ImageIcon("icons\\stopModify.png"));
          itemStop.setEnabled(false);
          submenuSession.add(itemStart);
          submenuSession.addSeparator();
          submenuSession.add(itemStop);


          submenu = new JMenu("Expand/Collapse");
          subItem11 = new JMenuItem("Expand", new ImageIcon("icons\\expand.png"));
          subItem12 = new JMenuItem("Collapse", new ImageIcon("icons\\collapse.png"));
          submenu.add(subItem11);
```

```
submenu.addSeparator();
submenu.add(subItem12);

submenu2 = new JMenu("Set Background");
subItem20 = new JMenuItem("Select color..", new ImageIcon("icons\\kcoloredit.png"));
subItem21 = new JMenuItem("Gray background", new ImageIcon("icons\\package_graphics.png"));
subItem22 = new JMenuItem("Black background", new ImageIcon("icons\\package_graphics.png"));
subItem23 = new JMenuItem("White background", new ImageIcon("icons\\package_graphics.png"));
submenu2.add(subItem20);
submenu2.addSeparator();
submenu2.add(subItem21);
submenu2.addSeparator();
submenu2.add(subItem22);
submenu2.addSeparator();
submenu2.add(subItem23);

refresh = new JMenuItem("Refresh", new ImageIcon("icons\\u.gif"));

refresh.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
                    try {
                              ToolBarSouth.refresh_actionPerformed(e);
                    } catch (IOException e1) {
                              e1.printStackTrace();
                    }
          }
});

item1.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
                    ToolBarNorth.open_actionPerformed(e);
          }
});
item2.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
                    ToolBarNorth.save_actionPerformed(e);
          }
});
item3.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
                    FileMenu.print_actionPerformed(e);
          }
});
item4.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
                    ToolBarNorth.reset_actionPerformed(e);
          }
});
itemStart.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
                    ToolBarNorth.startModify_actionPerformed(e);
          }
});
itemStop.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
                    ToolBarNorth.stopModify_actionPerformed(e);
          }
});

subItem11.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
                    ToolBarNorth.expand_actionPerformed(e);
          }
});
subItem12.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
                    ToolBarNorth.collapse_actionPerformed(e);
          }
});

subItem20.addActionListener(new ActionListener() {
 public void actionPerformed(ActionEvent e) {
        // Args are parent component, title, initial color
        Color bgColor
          = JColorChooser.showDialog(TreeTableExample.frame,
                                     "Choose Background Color",
                                     getBackground());
        if (bgColor != null)
                    setBackground(bgColor);
      }
});

subItem21.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
                    setBackground(Color.gray);
          }
});
subItem22.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
                    setBackground(Color.black);
```

```
                }
        });
        subItem23.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        setBackground(Color.white);
                }
        });



        //menu.add(item7);
        menu.add(submenuFile);
        menu.addSeparator();
        menu.add(submenuSession);
        menu.addSeparator();
        menu.add(submenu);
        menu.addSeparator();
        menu.add(submenu2);
        menu.addSeparator();
        menu.add(refresh);
        menu.addSeparator();

        // Set the component to show the popup menu
        this.addMouseListener(new MouseAdapter() {
                public void mousePressed(MouseEvent evt) {
                        if (evt.isPopupTrigger()) {
                                menu.show(evt.getComponent(), evt.getX(), evt.getY());
                        }
                }

                public void mouseReleased(MouseEvent evt) {
                        if (evt.isPopupTrigger()) {
                                menu.show(evt.getComponent(), evt.getX(), evt.getY());
                        }
                }
        });

        /** ***************** */



        // Create the tree. It will be used as a renderer and editor.
        tree = new TreeTableCellRenderer(treeTableModel);

        // Install a tableModel representing the visible rows in the tree.
        super.setModel(new TreeTableModelAdapter(treeTableModel, tree));



        // Force the JTable and JTree to share their row selection models.
        tree.setSelectionModel(new DefaultTreeSelectionModel() {
                /**
                 *
                 */
                private static final long serialVersionUID = -8481003390717686853L;

                // Extend the implementation of the constructor, as if:
                /* public this() */{
                        setSelectionModel(listSelectionModel);
                }
        });
        // Make the tree and table row heights the same.
        tree.setRowHeight(getRowHeight());

        // Install the tree editor renderer and editor.

        setDefaultRenderer(TreeTableModel.class, tree);
        setDefaultRenderer(Double.class, new XmlStringRenderer());

        setDefaultEditor(TreeTableModel.class, new TreeTableCellEditor());
        setShowGrid(false);
        setIntercellSpacing(new Dimension(0, 0));
    }

    public void editingStopped2(Object value) {

   setValueAt(value, editingRow, editingColumn);
}

    public int getEditingRow() {
            return (getColumnClass(editingColumn) == TreeTableModel.class) ? -1
                            : editingRow;
    }

    //
    // The renderer used to display the tree nodes, a JTree.

    /*
```

```
	 * MODIFICA:The gray area below the treetable is ugly. This gray area is
	 * actually the scrollpane's viewport.
	 */
	public boolean getScrollableTracksViewportHeight() {
		return getPreferredSize().height < getParent().getHeight();
	}

	public class TreeTableCellRenderer extends JTree implements
			TableCellRenderer {

		/**
		 *
		 */
		private static final long serialVersionUID = 913551938920374007L;

		protected int visibleRow;

		public TreeTableCellRenderer(TreeModel model) {
			super(model);
		}

		public void setBounds(int x, int y, int w, int h) {
			super.setBounds(x, 0, w, JTreeTable.this.getHeight());
		}

		public void paint(Graphics g) {
			g.translate(0, -visibleRow * getRowHeight());
			super.paint(g);
		}

		public Component getTableCellRendererComponent(JTable table,
				Object value, boolean isSelected, boolean hasFocus, int row,
				int column) {




			if (hasFocus) {
				// this cell is the anchor and the table has the focus
			}

			// Configure the component with the specified value

			// Set tool tip if desired
			setToolTipText((String) value);

			if (isSelected)
				setBackground(table.getSelectionBackground());
			else
				setBackground(table.getBackground());

			visibleRow = row;
			// Since the renderer is a component, return itself
			return this;
		}
	}

	/**
	 * The renderer used for String in the TreeTable. The only thing it does, is
	 * to format a null String as '-'.
	 */

	static class XmlStringRenderer extends DefaultTableCellRenderer {

		/**
		 *
		 */
		private static final long serialVersionUID = -6840858913255831152L;

		public XmlStringRenderer() {
			super();
		}

		public void setValue(Object value) {

			JTreeTable.value = (Double) value;
			setText((value == null) ? "-" : JTreeTable.value.toString());
		}
	}

	public class TreeTableCellEditor extends AbstractCellEditor implements
			TableCellEditor {
		public Component getTableCellEditorComponent(JTable table,
				Object value, boolean isSelected, int r, int c) {

			return tree;
		}
	}
}
```

MarkableTreeCellRenderer.java

```
package TAXInterface.table;


import java.awt.*;

import javax.swing.*;
import javax.swing.tree.*;

import TAXInterface.file.*;


/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

          public class MarkableTreeCellRenderer extends DefaultTreeCellRenderer {

          /**
           * Comment for <code>serialVersionUID</code>
           */
          private static final long serialVersionUID = 48149853550581965L;

          private boolean marked;

          private boolean marked2;

          private Image markIcon = new ImageIcon("icons\\keditbookmarks.png")
                              .getImage();

          private int iconWidth = markIcon.getWidth(null);

          private Color txtChoice = new Color(255, 0, 153);

          public MarkableTreeCellRenderer() {

          }

          public Component getTreeCellRendererComponent(JTree tree, Object value,
                              boolean selected, boolean expanded, boolean leaf, int row,
                              boolean hasFocus) {

                    Component comp = super.getTreeCellRendererComponent(tree, value,
                                        selected, expanded, leaf, row, hasFocus);

                    TreePath path = tree.getPathForRow(row);

                    boolean marked3;
                    boolean marked4;
                    boolean marked5;
                    boolean marked6;

                    if (path != null) {
                              marked = ((FileNode) path.getLastPathComponent()).isChoice();
                              marked2 = ((FileNode) path.getLastPathComponent()).isElementsList();
                              marked3 = ((FileNode) path.getLastPathComponent())
                                                  .hasChoiceInPath();
                              marked4 = ((FileNode) path.getLastPathComponent()).hasParentNode();
                              if (marked4)
                                        marked5 = ((FileNode) path.getLastPathComponent())
                                                            .getParentNode().isChoice();
                              else
                                        marked5 = false;

                              marked6 = ((FileNode) path.getLastPathComponent()).getChildren().length > 0;

                              if (!marked6 && marked5)
                                        setIcon(new ImageIcon("icons\\fogliaverde.png"));
                              if (!marked6 && !marked5)
                                        setIcon(new ImageIcon("icons\\unknown.png"));
                              else {
                                        if (marked) {
                                                  if (!marked2) {

                                                            if (marked3)
                                                                      setIcon(new
ImageIcon("icons\\folderRedStar.png"));
```

```
                                                                        else
                                                                                setIcon(new
ImageIcon("icons\\folder_salmone.png"));

                                                                        setForeground(txtChoice);

                                                                } else {
                                                                        if (marked3)
                                                                                setIcon(new ImageIcon("icons\\attachstar.png"));
                                                                        else
                                                                                setIcon(new ImageIcon("icons\\attach.png"));

                                                                        setForeground(Color.BLUE);
                                                                }

                                                        } else {

                                                        /*      if (marked2) {

                                                                        if (marked3)
                                                                                setIcon(new ImageIcon("icons\\attachstar.png"));
                                                                        else
                                                                                setIcon(new ImageIcon("icons\\attach.png"));

                                                                        setForeground(Color.BLUE);
                                                                }*/

                                                                if (marked4) {

                                                                        //if (!marked2) {
                                                                        if (marked5) {

                                                                                if (!marked) {
                                                                                        if (marked3) {
                                                                                                setIcon(new ImageIcon(

                        "icons\\greenstar.png"));                                                               if (((FileNode)
path.getLastPathComponent())

                        .getParentNode().isElementsList())

                        setForeground(Color.GRAY);
                                                                                                else

                        setForeground(Color.RED);
                                                                                        } else {
                                                                                                if (marked6)
                                                                                                        setIcon(new
ImageIcon(

                        "icons\\folder_green.png"));

                                                                                                if (((FileNode)
path.getLastPathComponent())

                        .getParentNode().isElementsList())

                        setForeground(Color.GRAY);
                                                                                                else

                        setForeground(Color.RED);

                                                                                        }
                                                                                }
                                                                        } else {
                                                                                if (!marked3) {
                                                                                        if (marked6)
                                                                                                setIcon(new
ImageIcon("icons\\folder.png"));
                                                                                } else {
                                                                                        setIcon(new ImageIcon(

                        "icons\\folder_favorites.png"));

                                                                                }

                                                                        }
                                                                } else {
                                                                        setIcon(new ImageIcon("icons\\folder_home.png"));
                                                                }
                                                        }
                                        }

                        }
                        return comp;
        }

        public Dimension getPreferredSize() {
                Dimension dim = super.getPreferredSize();
```

```
                dim.width += (iconWidth + 2);
                return dim;
        }

        protected void paintComponent(Graphics g) {

                Color highlightColorMouse = new Color(150, 150, 200);



                super.paintComponent(g);

                Font font = new Font("Arial", Font.PLAIN, 14);
                setFont(font);

                if (marked && !marked2) {

                        setBorderSelectionColor(Color.red);
                        setBackgroundSelectionColor(highlightColorMouse);
                        g.drawImage(markIcon, getWidth() - iconWidth, 0, null);
                }

        }
}
```

TreeTableModel.java

```
package TAXInterface.table;
import javax.swing.tree.TreeModel;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

/**
 * TreeTableModel e' il modello usato da JTreeTable. Estende TreeModel
 * per aggiungere metodi per recuperare informazioni sull'insieme di colonne che ogni
 * nodo in TreeTableModel deve avere. Ogni colonna, come una colonna in
 * una TableModel, ha un nome e un tipo associati. Ogni nodo nel
 * TreeTableModel puo restituire un valore per ognuna delle colonne e
 * e settare quel valore se isCellEditable() restituisce true.
 */
public interface TreeTableModel extends TreeModel {
        /**
         * Restituisce il numero di colonne disponibili.
         */
        public int getColumnCount();

        /**
         * Restituisce il nome della colonna di indice 'column'.
         */
        public String getColumnName(int column);

        /**
         * Restituisce il tipo per il numero di colonna 'column'.
         */
        public Class getColumnClass(int column);

        /**
         * Restituisce il valore visualizzato per il nodo 'node',
         * alla colonna di indice 'column'.
         */
        public Object getValueAt(Object node, int column);

        /**
         * Indica se il valore per il nodo 'node',
         * alla colonna di indice 'column' e' editabile.
         */
        public boolean isCellEditable(Object node, int column);

        /**
         * Setta il valore per il nodo 'node',
         * all'indice di colonna 'column'.
         */
```

```
        public void setValueAt(Object aValue, Object node, int column);
}
```

TreeTableModelAdapter.java

```
package TAXInterface.table;

import javax.swing.*;
import javax.swing.event.*;
import javax.swing.table.*;
import javax.swing.tree.*;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

public class TreeTableModelAdapter extends AbstractTableModel {
    /**
     *
     */
    private static final long serialVersionUID = -7655835275054201144L;

    public static JTree tree;

    public static TreeTableModel treeTableModel;

    /**
     * costruttore
     *
     * @param treeTableModel  -         un TreeTableModel
     * @param tree                              -         un JTree
     */
    public TreeTableModelAdapter(TreeTableModel treeTableModel, JTree tree) {
        TreeTableModelAdapter.tree = tree;

        TreeTableModelAdapter.treeTableModel = treeTableModel;

        MarkableTreeCellRenderer renderer = new MarkableTreeCellRenderer();
        tree.setCellRenderer(renderer);

        tree.addTreeExpansionListener(new TreeExpansionListener() {
            public void treeExpanded(TreeExpansionEvent event) {
                fireTableDataChanged();
            }

            public void treeCollapsed(TreeExpansionEvent event) {
                fireTableDataChanged();
            }
        });
    }

    // Wrappers, implementing TableModel interface.

    /**
           * Restituisce il numero di colonne disponibili.
           */
    public int getColumnCount() {
        return treeTableModel.getColumnCount();
    }

    /**
           * Restituisce il nome della colonna di indice 'column'.
           */
    public String getColumnName(int column) {
        return treeTableModel.getColumnName(column);
    }

    /**
           * Restituisce il tipo per il numero di colonna 'column'.
           */
    public Class getColumnClass(int column) {
        return treeTableModel.getColumnClass(column);
    }
```

```
    /**
     * Restituisce il numero delle righe della tabella
     */
    public int getRowCount() {
        return tree.getRowCount();
    }

    /**
     * Restituisce il nodo alla riga 'row'
     */
    protected Object nodeForRow(int row) {
        TreePath treePath = tree.getPathForRow(row);
        return treePath.getLastPathComponent();
    }

    /**
           * Restituisce il valore visualizzato alla riga 'row',
           * alla colonna di indice 'column'.
           */
    public Object getValueAt(int row, int column) {
        return treeTableModel.getValueAt(nodeForRow(row), column);
    }

    /**
     * Restituisce true se la cella [row, column]
     * e editabile
     */
    public boolean isCellEditable(int row, int column) {
        return treeTableModel.isCellEditable(nodeForRow(row), column);
    }

    /**
           * Setta il valore aValue alla riga 'row',
           * all'indice di colonna 'column'.
           */
    public void setValueAt(Object aValue, int row, int column) {

        treeTableModel.setValueAt(aValue, nodeForRow(row), column);
    }

    /**
           * Setta il valore per il nodo 'node',
           * all'indice di colonna 'column'.
           */
    public void setValueAt(Object aValue, Object node, int column) {

        treeTableModel.setValueAt(aValue, node, column);
    }
}
```

TAXInterface.graphics

Alert.java

```
package TAXInterface.graphics;

import javax.swing.ImageIcon;
import javax.swing.JDialog;
import javax.swing.JOptionPane;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

public class Alert {

    /*
     *  Alert: classe per creare dei warning a video con icona e stringa testuale.
     */
```

```
    private JOptionPane option;

    /**primo costruttore: il più semplice.
    *  prende la stringa da visualizzare nella finestra di dialogo
    */
    public Alert(String s) {

          //creo un joptionpane e setto il tipo a WARNING_MESSAGE con opzioni di default
        option = new JOptionPane(s, JOptionPane.WARNING_MESSAGE,
                JOptionPane.DEFAULT_OPTION);

        /**creo la finestra di dialogo vera e propria
        *dove viene visualizzato il messaggio di warning
        */
        JDialog dialog = option.createDialog(TreeTableExample.frame, "Alert");

        dialog.pack();
        dialog.setVisible(true);
    }

    /**secondo costruttore:
     * prende la stringa da visualizzare nella finestra di dialogo,
     * una stringa per il titolo della finestra di dialogo, un int per
     * il tipo di optionpane e un int per settare il DefaultOption
     */
    public Alert(String s, String title, int type, int opt) {

        option = new JOptionPane(s, type, opt, new ImageIcon("note02.gif"));

        JDialog dialog = option.createDialog(TreeTableExample.frame, title);
        dialog.pack();
        dialog.setVisible(true);
    }

    /** getResult() restituisce un intero che identifica la scelta
     * dell'utente selezionata nella finestra di dialogo
     * @return n: vaolore intero dell'opzione.
     * Ad esempio yes: 0, no:1
     */
    public int getResult() {

        int n = ((Integer) option.getValue()).intValue();
        return n;
    }
}
```

SuperFrame.java

```
package TAXInterface.graphics;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.print.PageFormat;
import java.awt.print.Printable;
import java.awt.print.PrinterException;

import javax.swing.JFrame;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */


/*
 * Classe superframe che estende jframe e implementa Printable. Infatti rispetto
 * a jframe superframe implementa il metodo print di printable. Questo metodo
 * servirà più avanti per fare funzionare l'opzione di stampa del nostro albero.
 *
 */

public class SuperFrame extends JFrame implements Printable {


          private static final long serialVersionUID = -6710504943042116449L;
```

```
	/**
	 * costruttore unico che prende come parametro una stringa.
	 *
	 * @param s: stringa che rappresenta il titolo del frame.
	 */
	public SuperFrame(String s) {

		// eredita il costruttore della superclasse.
		super(s);
	}

	/**
	 * Implementazione di print. Stampa la pagina all'indice
	 * specificato(pageIndex) nello specifico contesto grafico(Graphics) nel
	 * formato specificato(pageFormat)
	 *
	 * @param g - è di tipo Graphics e rappresenta il contesto
	 * @param pageformat - specifica il formato ed è di tipo PageFormat.
	 * @params pageIndex - è l'indice della pagina ed è un int.
	 */
	public int print(Graphics g, PageFormat pageFormat, int pageIndex)
			throws PrinterException {
		Graphics2D g2 = (Graphics2D) g;
		g2.setColor(Color.black);
		int fontHeight = g2.getFontMetrics().getHeight();
		int fontDesent = g2.getFontMetrics().getDescent();
		double pageHeight = pageFormat.getImageableHeight() - fontHeight;
		double pageWidth = pageFormat.getImageableWidth();
		double tableWidth = (double) TreeTableExample.treeTable
				.getColumnModel().getTotalColumnWidth();

		double headerHeightOnPage = TreeTableExample.treeTable
				.getTableHeader().getHeight();
		double tableWidthOnPage = tableWidth;
		double oneRowHeight = (TreeTableExample.treeTable.getRowHeight() + TreeTableExample.treeTable
				.getRowMargin());
		int numRowsOnAPage = (int) ((pageHeight - headerHeightOnPage) / oneRowHeight);
		double pageHeightForTable = oneRowHeight * numRowsOnAPage;
		int totalNumPages = 2;/*
                                                                * (int)Math.ceil((
                                                                * (double)treeTable.getRowCount())/
                                                                * numRowsOnAPage);
                                                                */
		if (pageIndex >= totalNumPages) {
			return NO_SUCH_PAGE;
		}

		g2.translate(pageFormat.getImageableX(), pageFormat.getImageableY());
		file: // bottom center
		g2.drawString("Page: " + (pageIndex + 1), (int) pageWidth / 2 - 35,
				(int) (pageHeight + fontHeight - fontDesent));

		g2.translate(0f, headerHeightOnPage);
		g2.translate(0f, -pageIndex * pageHeightForTable);

		if (pageIndex + 1 == totalNumPages) {
			int lastRowPrinted = numRowsOnAPage * pageIndex;
			int numRowsLeft = TreeTableExample.treeTable.getRowCount()
					- lastRowPrinted;
			g2.setClip(0, (int) (pageHeightForTable * pageIndex), (int) Math
					.ceil(tableWidthOnPage), (int) Math.ceil(oneRowHeight
					* numRowsLeft));
		}
		//file: // else clip to the entire area available.

		g2.setClip(0, (int) (pageHeightForTable * pageIndex), (int) Math
				.ceil(tableWidthOnPage), (int) Math.ceil(pageHeightForTable));


		TreeTableExample.treeTable.paint(g2);

		g2.translate(0f, pageIndex * pageHeightForTable);
		g2.translate(0f, -headerHeightOnPage);
		g2.setClip(0, 0, (int) Math.ceil(tableWidthOnPage), (int) Math
				.ceil(headerHeightOnPage));

		TreeTableExample.treeTable.getTableHeader().paint(g2);

		return Printable.PAGE_EXISTS;
	}

}
```

TreeTableExample.java

```
package TAXInterface.graphics;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;

import javax.swing.*;
import javax.swing.table.*;
import javax.swing.tree.*;
import javax.xml.parsers.*;

import org.w3c.dom.*;
import org.xml.sax.*;

import TAXInterface.file.*;
import TAXInterface.menu.*;
import TAXInterface.menu.MenuBar;
import TAXInterface.table.*;
import TSS.WeigthsAssignment.*;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

public class TreeTableExample {


          //        Creazione variabili d'istanza.
          public static SuperFrame frame;

          public static String path = null;

          private static JScrollPane scroll;

          public static JTreeTable treeTable;

          public static Document document;

          public static boolean active = false;

          public static Action m_action;

          public static JPopupMenu m_popup;

          public static TreePath m_clickedPath;

          /*
           * Primo costruttore senza parametri.
           *
           */
          public TreeTableExample() {

                    //inizializzazione di frame.
                    //prende come parametro il titolo del frame.
                    frame = new SuperFrame("TAXI XMLTreeTable - Choose file");

                    /*se la variabile active è true si entra nel corpo dell'if.
                    *  questa variabile ci serve per l'abilitazione dei pulsanti
                    *  dei puldanti delle toolbar e dei menu. Inizialmente
                    *  alcuni pulsanti devono rimanere diasabilitati e vengono
                    *  resi enabled solo quando active è true.
                    *  Quindi nell'if ci sta l'abilitazione dei pulsanti che
                    *  di default sono not enabled.
                    */

                              //viene creata e aggiunta la toolbar principale
                              //cioè quella che sta in alto sotto i menu.
                              JToolBar myToolBar = new ToolBarNorth();
                              frame.getContentPane().add(myToolBar, BorderLayout.NORTH);

                              //viene creata la toolbar sud cioè quella più in basso
                              JToolBar myToolBar2 = new ToolBarSouth();
                              frame.getContentPane().add(myToolBar2, BorderLayout.SOUTH);

                              JToolBar myToolBar3 = new ToolBarEast();
```

```
          frame.getContentPane().add(myToolBar3, BorderLayout.EAST);


          /*
           * se active non è true, allora si entra
           * nel corpo dell'else. Qui viene creata la toolbar
           * in alto e quella in basso proprio come nell'if,
           * ma con la differenza che alcuni pulsanti devono
           * rimanere disabilitati fintanto che non avviene
           * un certo evento(ad esempio appena aperto un file).
           */
          //JToolBar myToolBar = new ToolBarNorth();
          //frame.getContentPane().add(myToolBar, BorderLayout.NORTH);
          ToolBarNorth.buttonSave.setEnabled(false);
          ToolBarNorth.buttonReset.setEnabled(false);
          ToolBarNorth.buttonExpand.setEnabled(false);
          ToolBarNorth.buttonCollapse.setEnabled(false);
          ToolBarNorth.buttonStartModify.setEnabled(false);
          ToolBarNorth.buttonStopModify.setEnabled(false);

          //JToolBar myToolBar2 = new ToolBarSouth();
          //frame.getContentPane().add(myToolBar2, BorderLayout.SOUTH);
          ToolBarSouth.info.setEnabled(false);
          ToolBarSouth.update.setEnabled(false);
          ToolBarSouth.close.setEnabled(true);


          //JToolBar myToolBar3 = new ToolBarEast();
          //frame.getContentPane().add(myToolBar3, BorderLayout.EAST);

/*
 * Qui viene creata la menubar,
 * occero quei menu a cascata posizionati
 * in cima al frame sotto il titolo.
 */
MenuBar mb = new MenuBar();
frame.setJMenuBar(mb);
/**aggiungiamo al frame un ascoltatore
 * per la finestra. per la chiusura del frame.
 */
frame.addWindowListener(new WindowAdapter() {
          public void windowClosing(WindowEvent we) {
                    //il frame verrà chiuso
                    frame.dispose();
          }
});

//settiamo le misure di default del pannello principale del frame
//passandogli la dimensione 1000(x) 600(y).
((JComponent) frame.getContentPane()).setPreferredSize(new Dimension(
                    900, 730));

//impostiamo il bordo bianco di spessore 3 (larghezza in pixel) al
//pannello principale del frame.
((JComponent) frame.getContentPane()).setBorder(BorderFactory
                    .createLineBorder(Color.white, 3));

//settiamo la posizione iniziale del frame al centro dello schermo
//frame.setLocation(new Point(150, 100));
setCentered(frame);
scroll = new JScrollPane(treeTable);
frame.getContentPane().add(scroll,
                    BorderLayout.CENTER);
frame.pack();
frame.setVisible(true);
/* aggiungiamo al pannello principale
 * del frame un Jscrollpane, posizionato
 *  nel centro (col borderlayout)e a cui viene
 * passato come parametro l'oggetto treetable
 * di tipo JTreetable.
 */
String newPath = null;
while (newPath == null) {

          /**si crea una finestra di dialogo per
           * avvertire l'user che se si vuole cominciare
           * si deve aprire un file da file system.
           */
          JOptionPane option = new JOptionPane(
                              "If you want to start\nyou have to open a\nnew XML file.",
                              JOptionPane.INFORMATION_MESSAGE, JOptionPane.DEFAULT_OPTION);
          JDialog dialog = option.createDialog(TreeTableExample.frame,
          "Choose a XML file");
          dialog.pack();
          dialog.setVisible(true);

          //c prende l'opzione scelta dall'utente(come intero)
          //nella finestra di dialogo iniziale
          Integer c = ((Integer) option.getValue());
```

```
                    //se l'utente chiude la finestra con la x  (cioè c è null)
                    //allora visualizziamo un alert di warning.
                    //altrimenti si entra nel corpo dell'if
                    if (c != null) {
                              int n = ((Integer) option.getValue()).intValue();

                              //se l'utente preme ok si entra nel corpo di questo if
                              if (n == 0) {

                                        String curFile = null;
                                        String dir = "";
                                        FileDialog file = null;
                                        File tester = null;
                                        boolean test = false;

                                        //finchè non esiste il file oppure il path è null cicliamo
                                        //e stampiamo gli Alert.
                                        while (test == false || curFile == null) {

                                                  file = new FileDialog(TreeTableExample.frame,
                                                            "Apri file", FileDialog.LOAD);

                                                  //settiamo il filtro iniziale per i file .xml
                                                  file.setFile("*.xml");

                                                  file.setVisible(true);

                                                  //prendiamo il nome del file (una String) con la getFile()
                                                  curFile = file.getFile();
                                                  if ((curFile) == null) {

                                                            break;
                                                  }

                                                  else {

                                                            dir = file.getDirectory();
                                                            newPath = dir + curFile;
                                                            path=newPath;
                                                            /**creato il path con la directory del file e
                                                            *il nome del file stesso, lo passiamo come
                                                            *parametro all'oggetto tester di tipo File
                                                            */
                                                            tester = new File(path);

                                                            //controlliamo se il file con quel path esiste
                                                            test = tester.exists();

                                                            //se non esiste il file si lancia un alert
                                                            if (!test)
                                                                      new Alert(
                                                                      "WARNING: File not exist!\n Choose an
existing file.");
                                                  }

                                        }
                                        if ((curFile) != null) {

                                                  path = newPath;
                                                  File files = null;

                                                  /* DocumentBuilderFactory.newInstance();
                                                   *  newInstance() Ottiene una istanza di un
documentbuilderFactory.
                                                   *
                                                   *  DocumentBuilderFactory definisce una factory
                                                   *  che permette alle applicazioni
                                                   *  di ottenere un parser che produce gli alberi
                                                   *  DOM dai documenti XML.
                                                   *
                                                   */
                                                  DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
                                                  DocumentBuilder builder = null;

                                                  try {

                                                            /*
                                                             * Crea una nuova istanza di DocumentBuilder
                                                             * usando i parametri configurati.
                                                             */
                                                            builder = factory.newDocumentBuilder();
                                                  } catch (ParserConfigurationException e) {
                                                            e.printStackTrace();
                                                  }
                                                  document = null;
                                                  try {
                                                            files = new File(path);
```

```
                    /* Parsing del documento XML passato
                     * come parametro.
                     * Restituisce un oggetto DOM.
                     * Se il file è null si lancia
                     * una IllegalArgumentException.
                     */
                    document = builder.parse(files);
            } catch (SAXException e) {
                    e.printStackTrace();
            } catch (IOException e) {
                    e.printStackTrace();
            }

            /*getDocumentElement restituisce la radice
            * del documento document.
            */
            Node root = document.getDocumentElement();

            /*mette la choice immaginaria nel
            * documento appendendola allo <schema>.
            * Se vi erano più figli dello schema,
            * allora la choice immaginaria si inserisce
            * come figlia di schema e acquisuce tutti i
            * figli dello schema.
            */
            normalizeDocument(root);


                    /*inizializza tutte le mappe con i
                     *valori di default se i figli dei nodi
                     *choice non sono stati ancora pesati
                     */

DefaultInitializer.initializer(document.getDocumentElement());


            /*
             * contiamo la riga di ogni figlio
             * di choice e la andiamo ad inserire
             * nella nostra mappa di righe RowMap.
             */
            RowCounter.count(0, document.getDocumentElement());

            //RowMap.print();

            /* Grazie a getName ci facciamo restituire
             * il nome del file e lo andiamo a mettere
             * nella barra in alto del frame accanto al titolo.
             */
            //frame = new SuperFrame("TAXI XMLTreeTable - " +
file.getName());
            frame.setTitle("TAXI XMLTreeTable - " + files.getName());
            /*inizializzazione di treetable.
             *prende come parametro un oggetto
             *parserxmlmodel.
             */

            treeTable = new JTreeTable(new ParserXmlModel());

            frame.getContentPane().remove(scroll);

            scroll = new JScrollPane(treeTable);


            frame.getContentPane().add(scroll,
                              BorderLayout.CENTER);

            /*
             * creo un oggetto di tipo MarkableTreeCellRenderer
             * che verrà poi passato al metodo setCellRenderer
             * applicato sull'albero tree(jtree).
             * Questo cell renderer ci serve per manipolare e fare
             * il rendering dei nodi dell'albero jtree(esempio:
             * modificare le icone dei nodi o i colori dei nodi ecc..)
             */
            MarkableTreeCellRenderer renderer = new
MarkableTreeCellRenderer();
            TreeTableModelAdapter.tree.setCellRenderer(renderer);


            //frame.repaint();
            ToolBarNorth.buttonSave.setEnabled(true);
            ToolBarNorth.buttonReset.setEnabled(true);
            ToolBarNorth.buttonExpand.setEnabled(true);
            ToolBarNorth.buttonCollapse.setEnabled(false);
            ToolBarNorth.buttonStartModify.setEnabled(true);
            ToolBarNorth.buttonStopModify.setEnabled(false);

            //JToolBar myToolBar2 = new ToolBarSouth();
```

```
                                                        //frame.getContentPane().add(myToolBar2,
BorderLayout.SOUTH);
                                                        ToolBarSouth.info.setEnabled(true);
                                                        ToolBarSouth.update.setEnabled(true);

                                                        FileMenu.menuItem2.setEnabled(true);
                                                        FileMenu.menuItemPrint.setEnabled(true);

                                                        /*A questo punto visualizziamo anche una
                                                         * finestra di dialogo per chiedrere all'user
                                                         * se vuole caricare immediatamente il
                                                         * database. Altrimenti verrà gestito
                                                         * manualmente più avanti nell'applicazione.
                                                         */
                                                        JOptionPane optionDB = new JOptionPane(
                                                                        "Do you want to load Database?",
                                                                        JOptionPane.INFORMATION_MESSAGE,
JOptionPane.YES_NO_OPTION);
                                                        JDialog dialogDB = optionDB.createDialog(frame,
                                                        "Select an option..");
                                                        dialogDB.pack();
                                                        dialogDB.setVisible(true);

                                                        if(((Integer) optionDB.getValue())!=null){

                                                                  int risposta = ((Integer)
optionDB.getValue()).intValue();

                                                                  if (risposta == 0) {
                                                                        //L'utente ha detto che vuole caricare
il DB
                                                                        //QUI VA IL CODICE PER CARICARE IL
DATABASE ECC ECC...

                                                                  }
                                                                  else {

                                                                        //warning per l'utente
                                                                        JOptionPane optionNO = new JOptionPane(
                                                                                          "Remember to load
Database before\ngenerate instances!",

          JOptionPane.INFORMATION_MESSAGE, JOptionPane.DEFAULT_OPTION);
                                                                        JDialog dialogNO =
optionNO.createDialog(frame,
                                                                        "Suggestion");
                                                                        dialogNO.pack();
                                                                        dialogNO.setVisible(true);

                                                                  }
                                                        }
                                                  }
                                        }
                              } else {

                                        new Alert("WARNING: No Xml file opened");

                                        break;
                              }
                              TableColumn column = null;

                              /* controlliamo che la treetable non sia
                               * null. Se fosse null non impostiamo
                               * la larghezza delle colonne della stessa
                               * treeTable. Se invece non è null
                               * settiamo la larghezza della prima
                               * colonna (la 0) a 300 pixel.
                               */
                              if(treeTable != null){
                                        column = treeTable.getColumnModel().getColumn(0);

                                        column.setPreferredWidth(300);
                              }
}



                    //impostiamo la jmenubar e le passiamo come parametro
                    //mb la jmenubar precedentemente creata.
                    frame.setJMenuBar(mb);

                    /* Questo metodo ci permette di cambiare
                     * l'icona sulla barra del titolo,
                     * sostituendo così la tazza java
                     * di solito impostata di default.
                     * setIconimage prende un imageicon come
                     * parametro che a sua volta prende
                     * il path dell'icona che vogliamo andare
                     * a visualizzare.
```

```
          */
         frame.setIconImage(new ImageIcon("icons\\taxi6.gif")
               .getImage());

   /* rende il frame totalmente visibile.
         *  senza questo metodo vedremo il frame
         *  in maniera parziale, cioè solo
         *  la barra del titolo.
         */
         //TreeTableExample.frame.repaint();

         frame.pack();
         OptionMenu.reset.setEnabled(true);
         //visualizza il frame.
         frame.setVisible(true);

}

/**Metodo che permette di impostare la
 * posizione del componente, passato come
 * parametro, al centro dello schermo.
 *
 * @param Component comp - il componente da
 * centrare nello schermo.
 */
public static void setCentered(Component comp){
         Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
         Dimension f = comp.getPreferredSize();
         comp.setLocation(screenSize.width/2 - (f.width/2),
                          screenSize.height/2 - (f.height/2));


}

/**
 * restituisce le dimensioni (X,Y) del frame principale.
 * @return - restituisce un oggetto di tipo Dimension
 * che rappresenta la larghezza(X in pixel) e
 * l'altezza (Y in pixel) del frame.
 */
public static Dimension getActualDimFrame() {
         return frame.getSize();
}

/**
 * restituisce le dimensioni (X,Y) del pannello principale
 * del frame.
 * @return - restituisce un oggetto di tipo Dimension
 * che rappresenta la larghezza(X in pixel) e
 * l'altezza (Y in pixel) del pannello principale
 * del frame.
 */
public static Dimension getActualDimPanel() {
         return frame.getContentPane().getSize();
}

/**
 * Risetta le dimensioni del frame e del pannello
 * principale con le dimensioni passate come parametro.
 * @param dimf - la dimensione del frame da settare.
 * @param dimp - la dimensione del pannellom principale da modificare.
 */
public static void resize(Dimension dimf, Dimension dimp) {
         frame.setSize(dimf);
         frame.getContentPane().setSize(dimp);
}

/**
 * Restituisce la posizione attuale del frame in cordinate X, Y
 * @return - restituisce un oggetto di tipo Point, che
 * rappresenta le coordinate spaziali rispettivamente sull'asse X
 * e sull'asse Y del frame.
 */
public static Point getLoc() {
         return frame.getLocation();
}

/**
 * Modifica la posizione del frame con le nuove coordinate date da p
 * e quella del pannello principale del frame.
 * @param p - rappresenta l'oggetto con le nuove coordinate spaziali,
 * date da p da assegnare al frame e al suo pannello principale.
 */
public static void setLoc(Point p) {
         frame.setLocation(p);
         frame.getContentPane().setLocation(p);
}

/**
 *        Inserisce un elemento di primo livello
```

```
	 *        all'interno del documento che raggruppa
	 *        i figli di tipo element (se ce n'e' piu' di uno)
	 *        di schema e li acquisisce come suoi figli.
	 *        Questo a sua volta viene aggiunto allo
	 *        schema come figlio.
	 * ù
	 *        @param root - Node che rappresenta la radice
	 * del documento da normalizzare
	 */
	public static void normalizeDocument(Node root){
		NodeList figli = root.getChildNodes();
		String prefix = new FileNode(root).getNodePrefix();

		if(figli.getLength() > 3){
			Vector f = new Vector();

			for(int i = 0; i < figli.getLength(); i++){
				f.addElement(figli.item(i));

			}
			Element first;

			first = document.createElement("ISTI_et_CNR_first_LEVEL_element");
			System.out.println("first: "+first);
			//first.setAttribute("ISTI_et_CNR_imaginary_ROOT_attribute", ""+true);
			int elem_count = 0;

			first.appendChild(document.createTextNode(""));
			for(int i = 0; i < f.size(); i++){

				String element;
				if(prefix != null)
					element = prefix+":element";
				else
					element  = "element";

				if(((Node) f.elementAt(i)).getNodeName().equals(element)){
					Node clone = ((Node) f.elementAt(i)).cloneNode(true);
					document.getDocumentElement().removeChild(((Node)
f.elementAt(i)).getPreviousSibling());
					document.getDocumentElement().removeChild((Node) f.elementAt(i));
					System.out.println("CLONE: "+clone);
					Node txt = document.createTextNode("");
					first.appendChild(clone);
					first.appendChild(txt);
					elem_count++;
				}
			}
			FileNode [] figli2 = (FileNode[]) new FileNode(first).getChildren();
			for(int i = 0; i < figli2.length; i++){
				System.out.println("FIGLIO: "+figli2[i]);
			}
			System.out.println("COUNT : "+elem_count);
			if(elem_count > 1){

		//appende first alla radice del documento(uno <schema>).
				document.getDocumentElement().appendChild(first);
				document.getDocumentElement().appendChild(document.createTextNode(""));

				NodeList ff = document.getDocumentElement().getChildNodes();

				for(int i = 0; i < ff.getLength(); i++){
					System.out.println("FIGLIO_SCHEMA: "+ff.item(i));
				}
				System.out.println("ELEM: "+document.getDocumentElement());
			}
		}
	}

	/**
	 * Metodo che aggiorna la treetable
	 * @param filex - il nuovo file da visualizzare
	 * nella treetable.
	 */
	public static void changeTable(String filex){
		System.out.println("CIAO CIAO CIAO!!");
		File file1 = null;



		/* DocumentBuilderFactory.newInstance();
		 *  newInstance() Ottiene una istanza di un documentbuilderFactory.
		 *
		 *  DocumentBuilderFactory definisce una factory
		 *  che permette alle applicazioni
		 *  di ottenere un parser che produce gli alberi
		 *  DOM dai documenti XML.
		 *
		 */
		DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
```

```
DocumentBuilder builder = null;

try {

          /*
           * Crea una nuova istanza di DocumentBuilder
           * usando i parametri configurati.
           */
          builder = factory.newDocumentBuilder();
} catch (ParserConfigurationException e) {
          e.printStackTrace();
}
TreeTableExample.document = null;
try {
          file1 = new File(filex);

          /* Parsing del documento XML passato
           * come parametro
           * e restituisce un oggetto DOM.
           * Se il file è null si lancia
           * una IllegalArgumentException.
           */
          TreeTableExample.document = builder.parse(filex);
} catch (SAXException e) {
          e.printStackTrace();
} catch (IOException e) {
          e.printStackTrace();
}

/*getDocumentElement restituisce la radice
* del documento document.
*/
Node root = TreeTableExample.document.getDocumentElement();

/*mette la choice immaginaria nel
* documento appendendola allo <schema>.
* Se vi erano più figli dello schema,
* allora la choice immaginaria si inserisce
* come figlia di schema e acquisuce tutti i
* figli dello schema.
*/
normalizeDocument(root);



          /*inizializza tutte le mappe con i
           *valori di default se i figli dei nodi
           *choice non sono stati ancora pesati
           */
          DefaultInitializer.initializer(document.getDocumentElement());

/*
 * contiamo la riga di ogni figlio
 * di choice e la andiamo ad inserire
 * nella nostra mappa di righe RowMap.
 */
RowCounter.count(0, document.getDocumentElement());

RowMap.print();

/*Grazie a getName ci facciamo restituire
 * il nome del file e lo andiamo a mettere
 * nella barra in alto del frame accanto al titolo.
 */
//frame = new SuperFrame("TAXI XMLTreeTable - " + file.getName());
frame.setTitle("TAXI XMLTreeTable - " + file1.getName());
treeTable = new JTreeTable(new ParserXmlModel());

frame.getContentPane().remove(scroll);

scroll = new JScrollPane(treeTable);
frame.getContentPane().add(scroll,
                    BorderLayout.CENTER);


ToolBarNorth.buttonSave.setEnabled(true);
ToolBarNorth.buttonReset.setEnabled(true);
ToolBarNorth.buttonExpand.setEnabled(true);
ToolBarNorth.buttonCollapse.setEnabled(false);
ToolBarNorth.buttonStartModify.setEnabled(true);
ToolBarNorth.buttonStopModify.setEnabled(false);

ToolBarSouth.info.setEnabled(true);
ToolBarSouth.update.setEnabled(true);

FileMenu.menuItem2.setEnabled(true);
FileMenu.menuItemPrint.setEnabled(true);
OptionMenu.reset.setEnabled(true);

TableColumn column = null;
```

```
                    /*controlliamo che la treetable non sia
                     * null. Se fosse null non impostiamo
                     * la larghezza delle colonne della stessa
                     * treeTable. Se invece non è null
                     * settiamo la larghezza della prima
                     * colonna (la 0) a 300 pixel.
                     */
                    if(treeTable != null){
                              column = treeTable.getColumnModel().getColumn(0);
                              System.out.println("grandezza colonne!!");
                              column.setPreferredWidth(300);
                    }

                    //frame.repaint();
                    frame.pack();
          }
}
```

InterfaceMain.java

```
package TAXInterface.graphics;

import java.awt.*;

import javax.swing.*;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

public class InterfaceMain {

          private static String fl;

          public static void main(String[] args) {

                    //creazione di uno splashscreen iniziale

                    //si crea la finestra per lo splashscreen
                    JWindow jwin = new JWindow();



                    //ci creiamo anche un pannelo da posizionare sulla finsetra
                    JPanel panel = new JPanel();

                    /**setting del layout del pannello.
                     * Abbiamo scelto flowlayout che dispone
                     * gli elementi sul pannello da
                     * sinistra versdo destra.
                     */
                    panel.setLayout(new FlowLayout());

                    //inseriamo 4 icone di tipo .gif
                    //la T la A la X e la I.
                    panel.add(new JLabel(new ImageIcon("icons\\TTT.gif")));
                    panel.add(new JLabel(new ImageIcon("icons\\AAA.gif")));
                    panel.add(new JLabel(new ImageIcon("icons\\XXX.gif")));
                    panel.add(new JLabel(new ImageIcon("icons\\III.gif")));

                    //settiamo lo sfondo del pannello color arancio.
                    panel.setBackground(Color.ORANGE);

                    /**creiamo anche una label, cioè un etichetta
                     * con la stringa da visualizzare
                     */
                    JLabel label = new JLabel(
                    "Testing by Automatically generated XML Instances. ISTI C.N.R. Pisa");

                    //setting del font della label.
                    label.setFont(new Font("SanSerif", Font.BOLD, 12));

                    //creiamo un altro pannello.
                    JPanel panel1 = new JPanel();
```

```
        //a questo pannello aggiungiamo la label creata poco fa.
        panel1.add(label);

        //settiamo anche lo sfondo di panel1 color arancio
        //in modo che il colore sia omogeneo.
        panel1.setBackground(Color.ORANGE);

        //creiamo un altro pannello ancora e chiamiamolo panel2.
        JPanel panel2 = new JPanel();

        //creiamo un altra label ancora e chiamiamola label2.
        //a questa label2 aggiungiamo un'icona(.gif) con il logo
        //del CNR reparto ISTI.
        JLabel label2 = new JLabel(new ImageIcon("icons\\logo3.gif"));

        //a panel2 settiamo invece il colore dello sfondo bianco.
        panel2.setBackground(Color.WHITE);

        //a panel2 aggiungiamo label2.
        panel2.add(label2);


        /*settiamo le dimensioni dello splashscreen
         * e gli aggiungiamo, con layout BorderLayout,
         * rispettivamente panel in alto nello splashcreen,
         * panel2 nel centro e panel1 in basso sotto panel2.
         * Inoltre settiamo le cordinate della posizione iniziale
         * dello splashscreen in modo che sia centrato
         * allo schermo.
         */

        //jwin.setBounds(400, 400, 400, 200);
        CenterWin(jwin);
        jwin.setSize(400, 200);
        jwin.getContentPane().add(panel, BorderLayout.NORTH);
        jwin.getContentPane().add(panel2, BorderLayout.CENTER);
        jwin.getContentPane().add(panel1, BorderLayout.SOUTH);

        jwin.getContentPane().setVisible(true);

        //settiamo visibile lo splashscreen.
        jwin.setVisible(true);
        //resta visibile per 5 secondi.
        try {
                  Thread.sleep(5000);
        } catch (InterruptedException e) {
                  e.printStackTrace();
        }

        //dopo cinque secondi chiude lo splaschscreen.
        jwin.dispose();

        /*settaggio del look & feel della nostra applicazione.
         * prima proviamo con il MetalLookAndFeel.
         * Abbastanza gradevole come impatto visivo.
         * se il sistema non supporta tale look,
         * si visualizza una finestra di dialogo
         * che avverte che questo look and feel
         * non è supportato. A questo punto
         * l'applicazione si avvia con il
         * look and feel di sistema.
         */
        try {
                  UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
        } catch (Exception e) {

                  JOptionPane.showMessageDialog(null, "MetalLookAndFeel not supported.");

                  //Se non va con il metal si fa prendere il L&F di sistema.
                  //Occorre un blocco try...catch interno!!!
                  try {
                            UIManager.setLookAndFeel(UIManager
                                                .getSystemLookAndFeelClassName());
                  } catch (Exception e_int) {
                            JOptionPane.showMessageDialog(null, "Look&Feel not supported.");
                  }

        }


        new TreeTableExample();


}

/**
 * Metodo che centra la Jwindow passata come parametro
 * al centro dello schermo.
```

```
          * @param win - la Jwindow che deve essere centrata
          * nello scermo.
          */
         public static void CenterWin(JWindow win){
                  Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
                  Dimension f = win.getPreferredSize();
                  win.setLocation(screenSize.width/3 - (f.width/3),
                                  screenSize.height/3 - (f.height/3));
         }

         /**
          * restituisce il nome del file come stringa
          */
         static String getFile() {
                  return fl;
         }
}
```

TAXInterface.menu

FileMenu.java

```
package TAXInterface.menu;


import java.awt.*;
import java.awt.event.*;
import java.awt.print.*;

import javax.swing.*;

import TAXInterface.graphics.TreeTableExample;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */
public class FileMenu extends JMenu{

          /*si estende la classe JMenu*/
    private static final long serialVersionUID = 12320667671004764 99L;

    public static JMenuItem menuItem2;
    public static JMenuItem menuItemPrint;


    //costruttore senza parametri.
    public FileMenu() {

    /*eredita il costruttore della superclasse JMenu e
     * e viene passata come parametro la stringa "File".
     */
       super("File");

       /*impostiamo il tooltip con la stringa
        * da visualizzare paasata come parametro
        */
       setToolTipText("File");

       /*associamo una combinazione da tastiera a this(filemenu).
        * In questo caso la combinazione di tasti ALT-F.
        */
       setMnemonic(KeyEvent.VK_F);

        //Oggetti del menu a discesa
        /* la prima è la Open */

        JMenuItem menuItem;
        menuItem = new JMenuItem("Open");
```

```
/*settiamo l'icona al opzione Open del menu a tendina,
 * il tooltip e la combinazione da tastiera associata.
 */
menuItem.setIcon(new ImageIcon("icons\\open3.gif"));
menuItem.setToolTipText("Open file");
menuItem.setMnemonic(KeyEvent.VK_O);

/*aggiungiamo un actionListener al menuItem.
 * Questo ci permetterà di ascoltare gli eventi associati
 * all'opzione Open.
 */

menuItem.addActionListener(new ActionListener() {
  /*implementazione del metodo actionPerformed.
   * Nel corpo di questo metodo, viene inserito
   * il codice da eseguire quando si seleziona
   * l'opzione Open dal menu.
   *
   * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
   */
    public void actionPerformed(ActionEvent ae) {

            /*creazione di una finestra di dialogo per
             * permettere l'apertura di un nuovo file.
             * Come parametri prende il frame su cui verrà
             * aperta la finestra, la stringa che rappresenta
             * il titolo della finestra e l'ultimo parametro
             * rigurda il modo d'apertura della finestra: LOAD
             * se stiamo cercando un file sul file system da
             * aprire, SAVE se vogliamo salvare un file.
             */
            FileDialog file = new FileDialog(TreeTableExample.frame,
                "Apri file", FileDialog.LOAD);
            /*Imposta il filtro iniziale per i file.
             * In questo caso si cerca tra i file xml.
             */
            file.setFile("*.xml");
        file.setVisible(true);
        String curFile, filename = "";
        String dir = "";

        //si controlla se il nome del file è diverso da null.
        if ((curFile = file.getFile()) != null) {
            dir = file.getDirectory();
            filename = dir + curFile;

            /*
             * impostiamo le dimensioni di default del frame.
             * Cioè se il frame era 300x400 quando si apre un
             * nuovo file e viene aggiornato il frame, l'utente
             * deve ritrovarsi con le stesse dimensioni della
             * finestra.
             */
            Dimension d = TreeTableExample.frame.getSize();
                     Dimension dim = TreeTableExample.frame.getContentPane().getSize();
                     Point p = TreeTableExample.frame.getLocation();
                     TreeTableExample.changeTable(filename);
                     TreeTableExample.frame.setLocation(p);
                     TreeTableExample.frame.getContentPane().setSize(dim);
                     TreeTableExample.frame.setSize(d);

        }
    }
});

//aggiungiamo menuItem a this ed anche un separatore(uno spazio).
add(menuItem);
addSeparator();


// creazione dell'opzione Save
menuItem2 = new JMenuItem("Save");

/*disabilitiamo inizialmente l'opzione Save.
 * Inoltre gli associamo un'icona, un tooltip e
 * una combinazione da tastiera(ALT-S).
 */
menuItem2.setEnabled(false);
menuItem2.setIcon(new ImageIcon("icons\\save.gif"));
menuItem2.setToolTipText("Save this file");
menuItem2.setMnemonic(KeyEvent.VK_S);

/*aggiungiamo un actionListener a Save e implementiamo
 * l'actionPerformed.
 */
menuItem2.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {

            //codice da eseguire in caso
            //che l'opzione Save sia selezionata
```

```
                /*se la variabile path(directory del file)
                 * di TreetableExample è  diversa da null
                 * allora si esegue il metodo save_actionPerformed(ae)
                 * della classe ToolBarNorth
                 */
                if (TreeTableExample.path != null)
                    System.out.println("Sto salvando....");
                ToolBarNorth.save_actionPerformed(ae);
            }
        });

        //aggiungiamo menuItem2 a this ed anche un separatore(uno spazio).
        add(menuItem2);
        addSeparator();

        /*aggiungo print nel menu a discesa di "File".
         * Stesse procedure delle altre 2 opzioni.
         */
        menuItemPrint = new JMenuItem("Print");
        menuItemPrint.setEnabled(false);
        menuItemPrint.setIcon(new ImageIcon("icons\\printr01.gif"));
        menuItemPrint.setToolTipText("Print file");
        menuItemPrint.setMnemonic(KeyEvent.VK_P);
        menuItemPrint.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (TreeTableExample.path != null)
                    print_actionPerformed(e);
            }
        });

        add(menuItemPrint);
        addSeparator();
        /*aggiungo exit nel menu a discesa di "File".
         * Stesse procedure delle altre 2 opzioni.
         */
        menuItem = new JMenuItem("Exit");
        menuItem.setIcon(new ImageIcon("icons\\exit.gif"));
        menuItem.setToolTipText("Close Application");
        menuItem.setMnemonic(KeyEvent.VK_X);
        menuItem.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                JOptionPane option = new JOptionPane(
                        "Are you sure you want to quit?",
                        JOptionPane.QUESTION_MESSAGE, JOptionPane.YES_NO_OPTION);
                JDialog dialog = option.createDialog(TreeTableExample.frame,
                "Quit");
                dialog.pack();
                dialog.setVisible(true);
                int n = ((Integer) option.getValue()).intValue();
                if (n == 0)
                    TreeTableExample.frame.dispose();
            }
        });
        add(menuItem);


    }
    /**il corpo di questo codice viene eseguito quando si clicca
     * sul pulsante Print nel menu delle opzioni e invia alla
     * stampante la struttura ad albero visualizzata
     * dall'applicazione TAXI.
     *
     * @param ActionEvent event: prende un ActionEvent cioè quando si preme il
     * pulsante Print.
     */
    public static void print_actionPerformed(ActionEvent event) {

        PrinterJob pj = PrinterJob.getPrinterJob();
        pj.setPrintable((Printable) TreeTableExample.frame);
        pj.printDialog();
        try {
            pj.print();
        } catch (PrinterException e) {

            e.printStackTrace();
        }
    }

}
```

HelpMenu.java

```
package TAXInterface.menu;


import java.awt.event.*;

import javax.swing.*;

import TAXInterface.graphics.*;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

/*si estende la classe JMenu*/

public class HelpMenu extends JMenu{


    private static final long serialVersionUID = -6803491065059144571L;

    /*costruttore senza parametri
     * che eredita il costruttore
     * della superclasse e a cui viene
     * passata una stringa "?".
     * questa stringa verrà visualizzata
     * nel menù.
     */
    public HelpMenu() {

        super("?");

        /*impostiamo il tooltip con la stringa
         * da visualizzare paasata come parametro
         */
        setToolTipText("?");

        /*associamo una combinazione da tastiera a this(HelpMenu).
         * In questo caso la combinazione di tasti ALT-0.
         */
        setMnemonic(KeyEvent.VK_0);

        JMenuItem helpItem, authItem;


        //Oggetti del menu a discesa
        /* la prima è la Help */
        helpItem = new JMenuItem("Help");


        /*settiamo l'icona al opzione Help del menu a tendina,
         * il tooltip e la combinazione da tastiera associata.
         */
        helpItem.setIcon(new ImageIcon("icons\\Help_32x32.png"));
        helpItem.setToolTipText("Help");
        helpItem.setMnemonic(KeyEvent.VK_L);
        authItem = new JMenuItem("Credits");
        authItem.setToolTipText("Credits");
        authItem.setMnemonic(KeyEvent.VK_D);
        authItem.setIcon(new ImageIcon("icons\\Miniaturebulb.gif"));

        // azione per help item.
        /* aggiungiamo un actionListener al helpItem.
         * Questo ci permetterà di ascoltare gli eventi associati
         * all'opzione Help.
         */
        helpItem.addActionListener(new ActionListener() {
          /*implementazione del metodo actionPerformed.
           * Nel corpo di questo metodo, viene inserito
           * il codice da eseguire quando si seleziona
           * l'opzione Help dal menu.
           *
           * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
           */
            public void actionPerformed(ActionEvent e) {
                    /*
                     * si invoca il metodo
```

```
                *  menu_help_actionPerformed(e)
                *  perchè prende come parametro
                *  l'ActionEvent che era stato passato
                *  precedentemente ad actioPerformed.
                */
            menu_help_actionPerformed(e);
        }

    });

    //azione per authItem.
    /* aggiungiamo un actionListener all'authItem.
     * Questo ci permetterà di ascoltare gli eventi associati
     * all'opzione Credits.
     */
    authItem.addActionListener(new ActionListener() {

      /*implementazione del metodo actionPerformed.
       * Nel corpo di questo metodo, viene inserito
       * il codice da eseguire quando si seleziona
       * l'opzione Credits dal menu.
       *
       * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
       */
        public void actionPerformed(ActionEvent e) {

                /* si invoca il metodo
                *  auth_help_actionPerformed(e)
                *  perchè prende come parametro
                *  l'ActionEvent che era stato passato
                *  precedentemente ad actioPerformed.
                */
            auth_actionPerformed(e);
        }

    });

    //aggiungiamo helpItem, authitem ed anche un separatore(uno spazio) a this.
    add(helpItem);
    addSeparator();
    add(authItem);
}


/**
 * Questo metodo quando viene invocato
 * visualizza una pannello di dialogo
 * all'utente con le istruzioni generali
 * e le principali caratteristiche
 * dell'applicazione TAXI.
 * Tutto ciò è implementato grazie ad
 * un OptionPane.
 * @param e: L'evento che viene passato come parametro
 * (catturato dall'actionListener associato
 * al helpItem)è l'evento che è stato
 * scatenato dalla pressione
 * del pulsante help.
 */
private static void menu_help_actionPerformed(ActionEvent e) {

      //La stringa da visualizzare nella JoptionPane.
    final String s1="Guida in linea\n\nTAXI application is a tool to generate automatically" +
                "\n" +
                "\nper creare documenti non \nparticolarmente elaborati. " +
                "\n\nQuesto programma può essere " +
                "\nutilizzato per visualizzare o \nmodificare i file di testo " +
                "\n(generalmente con estensione txt)." +
                "\n\nI file creali con questo editor " +
                "\ndi testo sono salvati in formato " +
                "\nUnicode e non ASCII quindi non " +
                "\nvengono formattati in maniera " +
                "\ncorretta da programmi che non posseggono " +
                "\nquesta codifica(può invece aprire " +
                "\nqualunque testo,in formato ASCII o " +
                "\nUNICODE).\n\nPoiché il Notepad supporta solo " +
                "\nelementi di formattazione molto semplici, " +
                "\nnon è possibile salvare inavvertitamente " +
                "\nattributi di formattazione speciali " +
                "\nin documenti che vogliamo " +
                "\nlasciare in formato di solo testo." +
                "\n\nCiò è particolarmente utile quando " +
                "\nsi creano documenti HTML per una " +
                "\npagina Web o programmi in JAVA,in " +
                "\nquanto eventuali caratteri speciali " +
                "\no altri attributi di formattazione " +
                "\npotrebbero non essere visibili.";

    /*Creiamo la JoptionPane e le passiamo
     * la stringa da visualizzare precedentemente
     * creata.
```

```
         */
        JOptionPane.showMessageDialog(TreeTableExample.frame, s1,"Guida in linea",1);

    }


    /**
     * Questo metodo quando viene invocato
     * visualizza una pannello di dialogo
     * all'utente con le informazioni(crediti)
     * sui creatori dell'applicazione e
     * l'istituto di appartenenza.
     * Tutto ciò è implementato grazie ad
     * un OptionPane.
     * @param e: L'evento che viene passato come parametro
     * (catturato dall'actionListener associato
     * al authItem)è l'evento che è stato
     * scatenato dalla pressione
     * del pulsante Credits.
     */
    private static void auth_actionPerformed(ActionEvent e) {

          //Stringa da visualizzare all'utente.
          final String s1="\nJava Notepad\nVersion: 1.0.0" +
                          "\n\nCreated by ISTI CNR PISA" +
                          "\n";

          /*Creiamo la JoptionPane e le passiamo
         * la stringa da visualizzare precedentemente
         * creata.
         */
        JOptionPane.showMessageDialog(null, s1,"Informazioni Sul Prodotto",1);

    }
}
```

OptionMenu.java

```
package TAXInterface.menu;


import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

import TAXInterface.graphics.*;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

/*si estende la classe JMenu*/

public class OptionMenu extends JMenu{


    private static final long serialVersionUID = 1L;

    //creiamo le opzioni da inserire in Options.
    public static JMenuItem reset;
    public static JMenuItem color;

    /*costruttore senza parametri
     * che eredita il costruttore
     * della superclasse e a cui viene
     * passata una stringa "Options".
     * Questa stringa verrà visualizzata
     * nel menubar.
     */
    public OptionMenu() {

          /*Creiamo un altro menu a discesa chiamato options*/
        super("Options");
```

```
/*impostiamo il tooltip con la stringa
 * da visualizzare paasata come parametro
 */
setToolTipText("Options file");

/*associamo una combinazione da tastiera a this(OptionMenu).
 * In questo caso la combinazione di tasti ALT-N.
 */
setMnemonic(KeyEvent.VK_N);

//creiamo l'oggetto del menu a tendina Options.
JMenuItem reset;

//inizializiamo reset.
reset = new JMenuItem("Reset");

/*settiamo l'icona all' opzione Options del menu a tendina,
 * il tooltip e la combinazione da tastiera associata.
 */
reset.setIcon(new ImageIcon("icons\\reset.png"));
reset.setMnemonic(KeyEvent.VK_R);
reset.setEnabled(true);
// azioni per Options.
/* aggiungiamo un actionListener a reset.
 * Questo ci permetterà di ascoltare gli eventi associati
 * all'opzione "Reset".
 */
reset.addActionListener(new ActionListener() {

  /*implementazione del metodo actionPerformed.
   * Nel corpo di questo metodo, viene inserito
   * il codice da eseguire quando si seleziona
   * l'opzione Reset dal menu Options.
   *
   * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
   */
    public void actionPerformed(ActionEvent e) {

            /*se TreeTableExample.path non è null viene richiamato
             * il metodo reset_actionPerformed(e)
             * della classe ToolBarNorth, che
             * riapre il solito file con i pesi resettati.
             */
            if (TreeTableExample.path != null)
            ToolBarNorth.reset_actionPerformed(e);
    }
});

/*di default settiamo a disabilitato
 * il pulsante reset, in quanto se non
 * vi è nessun file aperto non c'è modo
 * di resettare un file che non c'è.
 */
reset.setEnabled(false);

//aggiungiamo reset a this.
add(reset);


//inizializzione di color con la stringa da visualizzare nel menu.
color = new JMenuItem("Background..");

/*settiamo l'icona all' opzione color del menu a tendina,
 * il tooltip e la combinazione da tastiera associata.
 */
color.setIcon(new ImageIcon("icons\\kcoloredit.png"));
color.setToolTipText("Choose Background color..");
color.setMnemonic(KeyEvent.VK_K);

// azioni per color
/* aggiungiamo un actionListener a color.
 * Questo ci permetterà di ascoltare gli eventi associati
 * all'opzione "Background..".
 */
color.addActionListener(new ActionListener() {

  /*implementazione del metodo actionPerformed.
   * Nel corpo di questo metodo, viene inserito
   * il codice da eseguire quando si seleziona
   * l'opzione "Background.." dal menu Options.
   *
   * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
   */
    public void actionPerformed(ActionEvent e) {

            /*Viene creato un JFilechooser cioè una
             * finestra dove l'utente seleziona un colore.
             */
                    Color bgColor
                      = JColorChooser.showDialog(TreeTableExample.frame,
```

```
                                                    "Choose Background Color",
                                                    getBackground());
                            /*se il background selezionato
                             * non è null viene impostato come
                             * sfondo nella treetable.
                             * Altrimenti rimane il colore attuale.
                             */
                            if (bgColor != null)
                                        setBackground(bgColor);
                    }
            });

//      aggiungiamo reset a this.
        add(color);
    }

}
```

InformationsMenu.java

```
package TAXInterface.menu;

import java.awt.event.*;

import javax.swing.*;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

/*si estende la classe JMenu*/

public class InformationsMenu extends JMenu {


    private static final long serialVersionUID = 71925299807159876 50L;

    /*costruttore senza parametri
     * che eredita il costruttore
     * della superclasse e a cui viene
     * passata una stringa "Informations".
     * Questa stringa verrà visualizzata
     * nel menù.
     */
    public InformationsMenu() {
        super("Informations");

        /*associamo una combinazione da tastiera a this(InformationsMenu).
         * In questo caso la combinazione di tasti ALT-A.
         */
        setMnemonic(KeyEvent.VK_A);

        /*impostiamo il tooltip con la stringa
         * da visualizzare paasata come parametro
         */
        setToolTipText("Informations about the selected file");
        //azioni per info

        //creiamo l'oggetto del menu a tendina Informations.
        JMenuItem info = new JMenuItem("Info");

        /*settiamo l'icona all' opzione Informations del menu a tendina,
         * il tooltip e la combinazione da tastiera associata.
         */
        info.setIcon(new ImageIcon("icons\\i.png"));
        info.setToolTipText("Info");
        info.setMnemonic(KeyEvent.VK_I);

        /* aggiungiamo un actionListener a info.
         * Questo ci permetterà di ascoltare gli eventi associati
         * all'opzione Info.
         */
        info.addActionListener(new ActionListener() {

          /*implementazione del metodo actionPerformed.
```

```
           * Nel corpo di questo metodo, viene inserito
           * il codice da eseguire quando si seleziona
           * l'opzione Info dal menu informations.
           *
           * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
           */
          public void actionPerformed(ActionEvent e) {

                    /*viene richiamato il metodo info_actionPerformed(e)
                     * della classe ToolBarSouth, che visualizza informazioni
                     * sul file su cui sta lavorando l'utente.
                     */
                                    ToolBarSouth.info_actionPerformed(e);
             }
        });

        //aggiungiamo info a this.
        add(info);

    }

}
```

MenuBar.java

```
package TAXInterface.menu;

import javax.swing.*;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

/*MenuBar estende la classe JMenuBar
* JmenuBar ci permette di creare barre
* dei menu a tendina.
*/

public class MenuBar extends JMenuBar{


    private static final long serialVersionUID = -6643836112495565949L;

    //costruttore unico senza parametri.
    public MenuBar() {

          //eredita il costruttore dalla superclasse.
        super();

        /*a this(la nostra menubar) aggiungiamo
         * i nostri menu:FileMenu, OptionMenu, InformationsMenu
         * e HelpMenu.
         */
        add(new FileMenu());
        add(new OptionMenu());
        add(new InformationsMenu());
        add(new HelpMenu());
    }
}
```

ToolBarNorth.java

```
package TAXInterface.menu;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
```

```
import javax.swing.*;

import TAXInterface.file.*;
import TAXInterface.graphics.*;
import TAXInterface.table.*;
import TSS.WeigthsAssignment.FlagMap;
import TSS.WeigthsAssignment.NodeMap;
import TSS.WeigthsAssignment.ReaderWriter;
import TSS.WeigthsAssignment.RowMap;
import TSS.WeigthsAssignment.XmlFileBuilder;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

//estende la classe JToolBar.

public class ToolBarNorth extends JToolBar{


    private static final long serialVersionUID = -8604984344242037474L;

          public static JButton buttonOpen;

          public static JButton buttonSave;

          public static JButton buttonReset;

          public static JButton buttonExpand;

          public static JButton buttonCollapse;

          public static JButton buttonStartModify;

          public static JButton buttonStopModify;

          static boolean oneModify = false;

          public static String path;


          //costruttore unico senza parametri.
          public ToolBarNorth() {

                    //viene ereditato il costruttore della superclasse.
                    super();

                    /*chiamiamo il metodo addToolbarButton
                     * che mi restituisce un Jbutton che
                     * viene assegnato a buttonOpen.
                     */
                    buttonOpen = addToolbarButton(this, true, "Open", "open",
                    "*Open* Clicking here you can visualize the tree of an existing XML document");

                     /*associamo una combinazione da tastiera a calculate.
               * In questo caso la combinazione di tasti è ALT-U.
               */
                    buttonOpen.setMnemonic(KeyEvent.VK_O);

                    addSeparator();

                    /*chiamiamo il metodo addToolbarButton
                     * che mi restituisce un Jbutton che
                     * viene assegnato ad ogni bottone
                     * della  toolbar. Inoltre ad ogni
                     * bottone si assegna una combinazione
                     * da tastiera univoca.
                     */
                    buttonSave = addToolbarButton(this, true, "Save", "save",
                    "Save an existing document");
                    buttonSave.setMnemonic(KeyEvent.VK_S);
                    addSeparator();

                    buttonExpand = addToolbarButton(this, true, "Expand all",
                                        "expand", "Expand all");
                    buttonExpand.setMnemonic(KeyEvent.VK_E);
                    addSeparator();

                    buttonCollapse = addToolbarButton(this, true, "Collapse all",
                                        "collapse", "Collapse all");
                    buttonCollapse.setMnemonic(KeyEvent.VK_C);
                    addSeparator();
```

```
            buttonStartModify = addToolbarButton(this, true, "Start modify",
                                "startModify",
            "Start session modifing the node-s interfaces.weights");
            buttonStartModify.setMnemonic(KeyEvent.VK_M);
            addSeparator();

            buttonStopModify = addToolbarButton(this, true, "Stop modify",
                                "stopModify",
            "Stop session modifing the node-s interfaces.weights");
            buttonStopModify.setMnemonic(KeyEvent.VK_T);

            buttonStopModify.setEnabled(false);
            addSeparator();

            buttonReset = addToolbarButton(this, true, "Reset", "reset",
            "Reset an existing document");
            buttonReset.setMnemonic(KeyEvent.VK_R);
            addSeparator();


            /*carichiamo il logo dell'Isti CNR di Pisa.*/
            ImageIcon logo = new ImageIcon(
            "icons\\logo3.gif");

            /*Creiamo una label in cui
             * andiamo a piazzare il nostro
             * logo precedentemente creato.
             */
            JLabel label = new JLabel(logo);

            /*settiamo le dimensioni della
             * label esattamente uguali a
             * quelle del logo. In modo
             * che la label sia sovrascritta
             * dal logo.
             */

            label.setSize(logo.getIconWidth(), logo.getIconHeight());

            //rendiamo visibile la label
            label.setVisible(true);

            //creaiamo un Jpanel
            JPanel panel = new JPanel();

            /*settiamo le dimensioni del
             * panel pari a quelle del logo
             */

            panel.setSize(logo.getIconWidth(), logo.getIconHeight());

            /*Aggiungiamo al panel la
             * label contenente il logo.
             */
            panel.add(label);

            //Rendiamo visibile il panel.
            panel.setVisible(true);

            //aggiungiamo a this il panel creato.
            add(panel);

            addSeparator();
    }

    /**
     * Associa la stringa da visualizzare sul bottone,
     * l'icona e il tooltip. Inoltre il bottone viene
     * posizionato all'interno della toolbar tramite
     * un layout di tipo Borderlayout. Ad ogni bottone
     * si associa un actionListener per captare quando
     * avviene la pressione del pulsante.
     *
     * @param toolBar - la toolbar di appartenenza.
     * @param bUseImage - true se il bottone visualizza
     * anche l'icona.
     * @param sButtonText - La stringa che viene visualizzazata
     * sul bottone.
     * @param sButton - il nome della variabile di tipo Jbutton.
     * @param sToolHelp - il tooltip associato a quel bottone
     * @return b - viene restituito il button settato.
     */
    public static JButton addToolbarButton(JToolBar toolBar, boolean bUseImage,
                      String sButtonText, String sButton, String sToolHelp) {
            JButton b;

            if (bUseImage) {
                      if (sButton.equals("save")) {
                                b = new JButton(new ImageIcon("icons\\" + sButton + ".gif"));
                      } else
```

```
                        b = new JButton(new ImageIcon("icons\\" + sButton + ".png"));
        } else
                b = (JButton) toolBar.add(new JButton());

        // Add the button to the toolbar

        toolBar.add(b);
        // Add optional button text
        if (sButtonText != null)
                b.setText(sButtonText);
        else {
                // Only a graphic, so make the button smaller
                b.setMargin(new Insets(0, 0, 0, 0));
        }

        // Add optional tooltip help
        if (sToolHelp != null)
                b.setToolTipText(sToolHelp);

        // Make sure this button sends a message when the user clicks it
        b.setActionCommand("Toolbar:" + sButton);

        /*Controlla che bottone è, e, a secondo di quale sia viene chiamata la
         * rispettiva actionPerformed
         * Se il controllo è true aggiungiamo un actionListener a b.
         * Questo ci permetterà di ascoltare gli eventi associati
         * ad ogni bottone.
         */
        if (sButtonText.equals("Open"))
                b.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                open_actionPerformed(e);
                        }
                });
        if (sButtonText.equals("Save"))
                b.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                save_actionPerformed(e);
                        }
                });
        if (sButtonText.equals("Reset"))
                b.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                reset_actionPerformed(e);
                        }
                });
        if (sButtonText.equals("Expand all"))
                b.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                expand_actionPerformed(e);
                        }
                });

        if (sButtonText.equals("Collapse all"))
                b.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                collapse_actionPerformed(e);
                        }
                });
        if (sButtonText.equals("Start modify"))
                b.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                startModify_actionPerformed(e);
                        }
                });
        if (sButtonText.equals("Stop modify"))
                b.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                stopModify_actionPerformed(e);
                        }
                });

        return b;
}

/**Questo metodo aprendo una finestra di dialogo
 * permette all'utente di scegliere un file da
 * file system.
 *
 * @param event - L'evento scatenato dalla pressione del
 * bottone Open della toolbar(l'evento che era stato captato
 * dall'actionListener).
 */
public static void open_actionPerformed(ActionEvent event) {

        FileDialog file = new FileDialog(TreeTableExample.frame, "Apri file",
                        FileDialog.LOAD);
        file.setFile("*.xml");// Set initial filename filter
        file.setVisible(true); // Blocks
        String curFile, filename = "";
```

```
          String dir = "";
          File tester;
          if ((curFile = file.getFile()) != null) {
                    dir = file.getDirectory();
                    filename = dir + curFile;
                    tester = new File(filename);

                    while (tester.exists() == false) {
                              new Alert("WARNING: File not exists!");
                              file = new FileDialog(TreeTableExample.frame, "Open file",
                                                  FileDialog.LOAD);
                              file.setFile("*.xml"); // Set initial filename filter
                              file.setVisible(true); // Blocks

                              curFile = file.getFile();

                              dir = file.getDirectory();
                              filename = dir + curFile;
                              tester = new File(filename);

                    }
                    RowMap.reset();
                    NodeMap.reset();
                    FlagMap.reset();
                    //Aggiornamento della treetable con il nuovo file.
                    TreeTableExample.changeTable(filename);
                    ToolBarEast.calculate.setEnabled(false);

          }

}

/**Questo metodo aprendo una finestra di dialogo
 * permette all'utente di salvare un file su
 * file system.
 *
 * @param event - L'evento scatenato dalla pressione del
 * bottone Save della toolbar(l'evento che era stato captato
 * dall'actionListener).
 */
public static void save_actionPerformed(ActionEvent event) {


          FileDialog fc = new FileDialog(TreeTableExample.frame, "Save file",
                              FileDialog.SAVE);
          fc.setFile("*.xml");
          fc.setVisible(true);



          String name = fc.getFile();
          String dir = fc.getDirectory();
          path = dir + name;


          XmlFileBuilder builder = new XmlFileBuilder(TreeTableExample.document, path);

          //TreeTableExample.changeTable(TreeTableExample.path);

          if (name == null)
                    return;
          else {
                    try {
                              builder.create();
                              new ReaderWriter(path+".xml");
                              JOptionPane option = new JOptionPane("Saved file: \n" + name,
                                                  JOptionPane.INFORMATION_MESSAGE,
                                                  JOptionPane.CLOSED_OPTION);
                              JDialog dialog = option.createDialog(TreeTableExample.frame,
                                                  "Saved");

                              dialog.pack();

                              dialog.setVisible(true);
                    } catch (IOException e) {
                              e.printStackTrace();
                    }
          }

          ToolBarEast.calculate.setEnabled(true);


          // MyMap.setChoiceCount();
          // System.out.println("NUMCHOICE: "+ MyMap.choiceCount);


}

/**Questo metodo aprendo una finestra di dialogo
 * permette all'utente di resettare i pesi cancellando
```

```
 * ogni modofica fatta. In pratica viene riaperto il
 * file.
 *
 * @param event - L'evento scatenato dalla pressione del
 * bottone Reset della toolbar.
 */
public static void reset_actionPerformed(ActionEvent event) {

          JOptionPane option = new JOptionPane("Are you sure you want to reset? "
                              + "\nWARNING: Each weights value will reset!!",
                              JOptionPane.QUESTION_MESSAGE, JOptionPane.YES_NO_OPTION);

          JDialog dialog = option.createDialog(TreeTableExample.frame, "Reset!");

          dialog.pack();

          dialog.setVisible(true);

          int n = ((Integer) option.getValue()).intValue();

          if (n == 0) {

                    String p = TreeTableExample.path;

                    NodeMap.reset();
                    FlagMap.reset();
                    RowMap.reset();


                    TreeTableExample.changeTable(p);
          }
}

/**Questo metodo permette all'utente di
 * di espandere automaticamente ogni nodo
 * dell'albero della treetable. In pratica
 * viene espanso totalmente l'albero.
 *
 * @param event - L'evento scatenato dalla pressione del
 * bottone Expand della toolbar.
 */
public static void expand_actionPerformed(ActionEvent event) {

          JOptionPane option = new JOptionPane(
                              "Are you sure you want to expand all?",
                              JOptionPane.QUESTION_MESSAGE, JOptionPane.YES_NO_OPTION);

          JDialog dialog = option.createDialog(TreeTableExample.frame,
                              "Expand all");

          dialog.pack();

          dialog.setVisible(true);

          if ((Integer) option.getValue() != null) {
                    int n = ((Integer) option.getValue()).intValue();

                    if (n == 0) {

                              for (int i = 0; i < TreeTableModelAdapter.tree.getRowCount(); i++)
                                        TreeTableModelAdapter.tree.expandRow(i);
                    }
          }
          buttonExpand.setEnabled(false);
          buttonCollapse.setEnabled(true);

}

/**Questo metodo permette all'utente di
 * di far collassare automaticamente ogni nodo
 * dell'albero della treetable. In pratica
 * viene collassato totalmente l'albero.
 *
 * @param event - L'evento scatenato dalla pressione del
 * bottone Collapse della toolbar.
 */
public static void collapse_actionPerformed(ActionEvent event) {

          JOptionPane option = new JOptionPane(
                              "Are you sure you want to collapse all?",
                              JOptionPane.QUESTION_MESSAGE, JOptionPane.YES_NO_OPTION);

          JDialog dialog = option.createDialog(TreeTableExample.frame,
                              "Collapse all");

          dialog.pack();

          dialog.setVisible(true);

          if ((Integer) option.getValue() != null) {
```

```
                        int n = ((Integer) option.getValue()).intValue();

                        if (n == 0)
                                for (int i = TreeTableModelAdapter.tree.getRowCount(); i > 0 ; i--)
                                        TreeTableModelAdapter.tree.collapseRow(i);
                                /***
                                for (int i = 1; i < TreeTableModelAdapter.tree.getRowCount(); i++) {
                                        TreeTableModelAdapter.tree.collapseRow(i);
                                }*/
                }
                buttonExpand.setEnabled(true);
                buttonCollapse.setEnabled(false);

        }

        /**Questo metodo permette all'utente di iniziare
         * la modifica dei pesi sul nodo dell'albero
         * della treetable. Se non viene premuta la
         * Startmodify non possono essere modificati
         * i pesi sul nodo.
         *
         *
         * @param event - L'evento scatenato dalla pressione del
         * bottone Start modify della toolbar.
         */
        public static void startModify_actionPerformed(ActionEvent event) {

                buttonStartModify.setEnabled(false);
                buttonStopModify.setEnabled(true);
                buttonSave.setEnabled(false);

                JTreeTable.itemStart.setEnabled(false);
                JTreeTable.itemStop.setEnabled(true);

                ParserXmlModel.CAN_EDIT = true;

        }

        /**Questo metodo permette all'utente di terminare
         * la modifica dei pesi sul nodo dell'albero
         * della treetable. Se non viene premuta la
         * Stop modify i pesi sul nodo continuano ad
         * essere modificabili. Inoltre quando viene
         * premuta la Stop modify, automaticamente
         * parte il refresh dei pesi dei fratelli del
         * nodo modificato.
         *
         *
         * @param event - L'evento scatenato dalla pressione del
         * bottone Stop modify della toolbar.
         */
        public static void stopModify_actionPerformed(ActionEvent event) {

                if (!FlagMap.isEmpty()) {
                        try {
                                ToolBarSouth.refresh_actionPerformed(null);
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                }

                buttonStartModify.setEnabled(true);
                buttonStopModify.setEnabled(false);
                buttonSave.setEnabled(true);

                JTreeTable.itemStart.setEnabled(true);
                JTreeTable.itemStop.setEnabled(false);

                new Alert("STORED", "Done", JOptionPane.INFORMATION_MESSAGE,
                                JOptionPane.DEFAULT_OPTION);
                ParserXmlModel.CAN_EDIT = false;
                ParserXmlModel.count = 0;

                //FileNode.calls = 0;
        }

}
```

ToolBarSouth.java

```
package TAXInterface.menu;
```

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;

import javax.swing.*;

import TAXInterface.file.*;
import TAXInterface.graphics.*;
import TAXInterface.table.*;
import TSS.WeigthsAssignment.FlagMap;
import TSS.WeigthsAssignment.NodeMap;
import TSS.WeigthsAssignment.RowMap;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

//estende la classe Jtoolbar.

public class ToolBarSouth extends JToolBar{


    private static final long serialVersionUID = -6684671974410347779L;

    public static JButton info;

    public static JButton update;

    public static JButton close;

     public static JTabbedPane scroll;

    public static JTextArea field;

    static boolean oneModify = false;

    static int count = 1;

    //costruttore unico senza parametri.
    public ToolBarSouth() {

         addSeparator();

         /*chiamiamo il metodo addToolbarButton
                    * che mi restituisce un Jbutton che
                    * viene assegnato a info.
                    */
         info = addToolbarButton(this, true, "Info", "i",
                "*INFO* Clicking here you can visualize informations about the document");


         /*associamo una combinazione da tastiera a info.
          * In questo caso la combinazione di tasti è ALT-I.
          */info.setMnemonic(KeyEvent.VK_I);

         addSeparator();

         /*chiamiamo il metodo addToolbarButton
                    * che mi restituisce un Jbutton che
                    * viene assegnato ad ogni bottone
                    * della  toolbar. Inoltre ad ogni
                    * bottone si assegna una combinazione
                    * da tastiera univoca.
                    */
         update = addToolbarButton(this, true, "Refresh", "reload",
                "*REFRESH* the existing document");
         update.setMnemonic(KeyEvent.VK_H);

         addSeparator();

         close = addToolbarButton(this, true, "Quit",
                "c", "*QUIT* the application");
         close.setMnemonic(KeyEvent.VK_Q);


         //Creiamo un oggetto di tipo Color e uno di tipo Font.
         Color color = new Color(241, 249, 255);
         Font f = new Font("Verdana", Font.ROMAN_BASELINE, 10);

         //A scroll si assegna una jtabbedPane.
         scroll = new JTabbedPane();
```

```
     //Rendiamo visibile scroll.
     scroll.setVisible(false);

     //Creiamo 2 Jtextarea.
     field = new JTextArea();
     JTextArea field2 = new JTextArea();

     /*settiamo alle 2 textfield appena
      * create il nome, il colore dello sfondo
      * e il font. Inoltre le rendiamo visibili.
      */
     field.setName("   Info   ");
     field2.setName("Operations");
     field.setVisible(true);
     field2.setVisible(true);
     field.setBackground(color);
     field2.setBackground(color);
     field.setFont(f);
     field.setEditable(false);
     field2.setEditable(false);

     //aggiungiamo a scroll le 2 textfield
     scroll.add(field);
     scroll.add(field2);

     addSeparator();
     //aggiungiamo scroll a this.
     add(scroll);
}

/**
       * Associa la stringa da visualizzare sul bottone,
       * l'icona e il tooltip. Inoltre il bottone viene
       * posizionato all'interno della toolbar tramite
       * un layout di tipo Borderlayout. Ad ogni bottone
       * si associa un actionListener per captare quando
       * avviene la pressione del pulsante.
       * @param toolBar - la toolbar di appartenenza.
       * @param bUseImage - true se il bottone visualizza
       * anche l'icona.
       * @param sButtonText - La stringa che viene visualizzazata
       * sul bottone.
       * @param sButton - il nome della variabile di tipo Jbutton.
       * @param sToolHelp - il tooltip associato a quel bottone
       * @return b - viene restituito il button settato.
       */
public static JButton addToolbarButton(JToolBar toolBar, boolean bUseImage,
        String sButtonText, String sButton, String sToolHelp) {
    JButton b = null;


        if (sButton.equals("i"))
                b = new JButton(new ImageIcon("icons\\" + sButton + ".png"));

        if (sButton.equals("reload"))
                b = new JButton(new ImageIcon("icons\\" + sButton + ".png"));

        if (sButton.equals("c"))
                b = new JButton(new ImageIcon("icons\\exit.gif"));

    // Add the button to the toolbar

    toolBar.add(b);
    // Add optional button text
    if (sButtonText != null)
        b.setText(sButtonText);
    else {
        // Only a graphic, so make the button smaller
        b.setMargin(new Insets(0, 0, 0, 0));
    }

    // Add optional tooltip help
    if (sToolHelp != null)
        b.setToolTipText(sToolHelp);

    // Make sure this button sends a message when the user clicks it
    b.setActionCommand("Toolbar:" + sButton);

    //Controlla che bottone è e a secondo di quale sia viene chiamata la
    // rispettiva actionPerformed

    if (sButtonText.equals("Info"))
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

                info_actionPerformed(e);
            }
        });
    if (sButtonText.equals("Refresh"))
        b.addActionListener(new ActionListener() {
```

```
                public void actionPerformed(ActionEvent e) {
                  if(!FlagMap.isEmpty()){
                    try {
                                                        refresh_actionPerformed(e);
                                                } catch (IOException e1) {
                                                        e1.printStackTrace();
                                                }
                  }
                }
            });
        if (sButtonText.equals("Quit"))
            b.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                    close_actionPerformed(e);
                }
            });

        return b;
    }

    /**Questo metodo visualizza nella tabbedpane
     * della toolbar alcune utili informzioni
     * sul file su cui sta lavorando l'utente.
          *
          * @param event - L'evento scatenato dalla pressione del
          * bottone Info della toolbar.
          */
    public static void info_actionPerformed(ActionEvent event) {

          if(count==1){
          scroll.setVisible(true);
          field.setText("   Total modifiable nodes:  " + NodeMap.size() +
                  "\n   Path file:  " + TreeTableExample.path+"\n   Total visible nodes:
"+TreeTableModelAdapter.tree.getRowCount());
        count =0;
          }
          else {
                  scroll.setVisible(false);
                  count = 1;
          }
    }

    /**Questo metodo, molto importante è quello
     * che permette di fare il refresh dei pesi.
     * In pratica ridistribuisce i pesi ai fratelli
     * del nodo a cui modificato il peso.
          *
          * @param event - L'evento scatenato dalla pressione del
          * bottone Refresh della toolbar.
          *
          * @exception - Può lanciare una IOException.
          */


    public static void refresh_actionPerformed(ActionEvent event) throws IOException {

          Iterator mapIterator = NodeMap.keySet().iterator();
          int column = 1;


          while(mapIterator.hasNext()){

                  ParserXmlModel.is_this = true;

                  Integer next = (Integer) mapIterator.next();

                          if(!FlagMap.get(next).booleanValue()){

                                  int row = ((Integer) (RowMap.get(next))).intValue();
                                  System.out.println("elementi nella mappa");
                                  Double d = NodeMap.get(next);
                                  System.out.println(d);
                                  ParserXmlModel.is_this = false;
                                  Vector to_collapse = new Vector();

                                  for (int i = 1; i < TreeTableModelAdapter.tree.getRowCount() ; i++) {
                                          if(!TreeTableModelAdapter.tree.isExpanded(i)){
                                                  TreeTableModelAdapter.tree.expandRow(i);
                                                  to_collapse.addElement(new Integer(i));
                                          }
                                  }
                                  TreeTableExample.treeTable.setValueAt(d, row, column);
                                  for (int i = TreeTableModelAdapter.tree.getRowCount(); i > 0 ; i--) {
                                          if(to_collapse.contains(new Integer(i)))
                                                  TreeTableModelAdapter.tree.collapseRow(i);
                                  }
                          }
          }
          //Graphics g = TreeTableExample0.frame.getGraphics();
```

```
          TreeTableExample.frame.repaint();//update(g);
          //TreeTableExample0.frame.setVisible(false);
          //TreeTableExample0.frame.setVisible(true);


        ParserXmlModel.is_this = true;
    }

    /**Questo metodo permette all'utente di terminare
     * l'applicazione (l'interfaccia viene chiusa).
           *
           * @param event - L'evento scatenato dalla pressione del
           * bottone Quit della toolbar.
           */
    public static void close_actionPerformed(ActionEvent event) {

        JOptionPane option = new JOptionPane("Are you sure you want to quit?",
                JOptionPane.QUESTION_MESSAGE, JOptionPane.YES_NO_OPTION);

        JDialog dialog = option.createDialog(TreeTableExample.frame, "Quit");

        dialog.pack();

        dialog.setVisible(true);

        int n = ((Integer) option.getValue()).intValue();

        if (n == 0)
            TreeTableExample.frame.dispose();

    }


}
```

ToolBarEast.java

```
package TAXInterface.menu;

import generation.*;

import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

import TAXInstancesVisualizer.*;
import TAXInterface.graphics.*;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

//ToolbarEast estende Jtoolbar.

public class ToolBarEast extends JToolBar{


          static JButton calculate;
          static JButton show;
          private static final long serialVersionUID = -7001478767933061206L;

          //costruttore senza parametri.
          public ToolBarEast() {

                    //aggiungiamo un separatore a this.
                   addSeparator();

                   /*chiamiamo il metodo addToolbarButton
                    * che mi restituisce un Jbutton che
                    * viene assegnato a calculate.
                    */
                   calculate = addToolbarButton(
                                       this,
                                       true,
```

```
                                        "Automatic generation",
                                        "calculate",
                    "*AUTOMATIC GENERATION* Clicking here you calculate the weigths of the sub-trees obtained from
choice");

                     /*associamo una combinazione da tastiera a calculate.
          * In questo caso la combinazione di tasti è ALT-U.
          */
                    calculate.setMnemonic(KeyEvent.VK_U);

                    //di default impostiamo a disabilitato il button calculate.
                    calculate.setEnabled(false);

                    //aggiungiamo un separatore a this
                    addSeparator();

                    /*chiamiamo il metodo addToolbarButton
                     * che mi restituisce un Jbutton che
                     * viene assegnato a show.
                     */
                    show = addToolbarButton(this, true, "     Show instances     ", "show", "*SHOW INSTANCES* Clicking
here you can visualize an instance in the instances directory");

                    /*associamo una combinazione da tastiera a show.
                     * In questo caso la combinazione di tasti è ALT-W.
                     */
                    show.setMnemonic(KeyEvent.VK_W);

                    //di default impostiamo a disabilitato il button show.
                    show.setEnabled(false);

                    addSeparator();

                    //creiamo un oggetto Legenda.
                    Legenda legenda = new Legenda();

                    //aggiungiamo legenda a this.
                    add(legenda);


          }


          /**
           * Associa la stringa da visualizzare sul bottone,
           * l'icona e il tooltip. Inoltre il bottone viene
           * posizionato all'interno della toolbar tramite
           * un layout di tipo Borderlayout. Ad ogni bottone
           * si associa un actionListener per captare quando
           * avviene la pressione del pulsante.
           * @param toolBar - la toolbar di appartenenza.
           * @param bUseImage - true se il bottone visualizza
           * anche l'icona.
           * @param sButtonText - La stringa che viene visualizzazata
           * sul bottone.
           * @param sButton - il nome della variabile di tipo Jbutton.
           * @param sToolHelp - il tooltip associato a quel bottone
           * @return b - viene restituito il button settato.
           */
          public JButton addToolbarButton(JToolBar toolBar, boolean bUseImage,
                              String sButtonText, String sButton, String sToolHelp) {
                    JButton b = null;

                    JPanel panel = new JPanel();
                    panel.setLayout(new BorderLayout());

                    if (sButton.equals("calculate"))
                              b = new JButton(new ImageIcon("icons\\calc.png"));

                    if(sButton.equals("show"))
                              b = new JButton(new ImageIcon("icons\\txt.png"));
                    // Add the button to the toolbar

                    toolBar.setLayout(new BoxLayout(toolBar, BoxLayout.PAGE_AXIS));

                    toolBar.add(b);


                    //Add optional button text
                    if (sButtonText != null)
                              b.setText(sButtonText);
                    else {
                              //Only a graphic, so make the button smaller
                              b.setMargin(new Insets(0, 0, 0, 0));
                    }

                    // Add optional tooltip help
                    if (sToolHelp != null)
                              b.setToolTipText(sToolHelp);
```

```
        // Make sure this button sends a message when the user clicks it
        b.setActionCommand("Toolbar:" + sButton);

        /*Controlla che bottone è, e, a secondo di quale sia viene chiamata la
         * rispettiva actionPerformed
         * Se il controllo è true aggiungiamo un actionListener a calculate.
         * Questo ci permetterà di ascoltare gli eventi associati
         * al bottone "Automatic generation".
         */
        if (sButtonText.equals("Automatic generation"))
                b.addActionListener(new ActionListener() {

                        /*implementazione del metodo actionPerformed.
         * Nel corpo di questo metodo, viene inserito
         * il codice da eseguire quando si preme
         * il pulsante "Automatic generation" nella toolbar.
         *
         * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
         */
                        public void actionPerformed(ActionEvent e) {

                                //chiamata del metodo calculate_actionPerformed(e);
                                calculate_actionPerformed(e);
                        }
                });



        /* Se il controllo è true aggiungiamo un actionListener a show.
         * Questo ci permetterà di ascoltare gli eventi associati
         * al bottone "     Show instances     ".
         */
        if (sButtonText.equals("     Show instances     "))
                b.addActionListener(new ActionListener() {

                        /*implementazione del metodo actionPerformed.
         * Nel corpo di questo metodo, viene inserito
         * il codice da eseguire quando si preme
         * il pulsante "Show instances" nella toolbar.
         *
         * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
         */
                        public void actionPerformed(ActionEvent e) {
                                show_actionPerformed(e);
                        }
                });


        return b;
}

/**Quando viene invocato si crea un nuovo oggetto PREPROCESSOR.
 * Il metodo non fa altro.
 *
 * @param event - L'evento scatenato dalla pressione del
 * bottone Automatic generation(l'evento che era stato captato
 * dall'actionListener.
 */
public static void calculate_actionPerformed(ActionEvent event) {

        System.out.println("sono in calculate...");
        System.out.println("toolbarnord.path è: "+ToolBarNorth.path);
        new PREPROCESSOR(ToolBarNorth.path);
        show.setEnabled(true);
        calculate.setEnabled(false);
}

/**Quando viene invocato viene visualizzato all'utente
 * una finestra in cui può scegliere le istanze da
 * visualizzare. Le istanze di default sono posizionate
 * in una cartella chiamata "Instances". Le istanze
 * vengono visualizzate in un notepad.
 *
 * @param event - L'evento scatenato dalla pressione del
 * bottone Show instances(l'evento che era stato captato
 * dall'actionListener).
 */
public static void show_actionPerformed(ActionEvent event) {

        FileDialog file = new FileDialog(TreeTableExample.frame, "Apri file istanza",
                        FileDialog.LOAD);
        file.setDirectory("Instances");
        file.setFile("*.xml");// Set initial filename filter
        file.setVisible(true); // Blocks
        String curFile = file.getFile();
        String dir = file.getDirectory();
        String filename = dir + curFile;

        Editor e=new Editor(filename);
        e.createPad();
```

```
        }

}
```

Legenda.java

```
package TAXInterface.menu;

import java.awt.*;

import javax.swing.*;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

//Legenda estende la classe JPanel.

public class Legenda extends JPanel{


        private static final long serialVersionUID = 1L;

        //costruttore unico senza parametri.
        public Legenda(){

                //creiamo un jpanel.
                JPanel mainPanel = new JPanel();

                //creiamo anche un jscrollpane.
                JScrollPane scroll = new JScrollPane();

                /*settiamo il layout di this con
                 * un BorderLayout.
                 */
                this.setLayout(new BorderLayout());

                /*impostiamo il colore(bianco) dello sfondo
                 * e il layout(BoxLayout: dall'alto al basso)
                 *  del pannello mainPanel
                 */
                mainPanel.setBackground(Color.WHITE);
                mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.PAGE_AXIS));

                /*creiamo un font di tipo Times new roman, grassetto
                 * e di grandezza 14.
                 */
                Font font = new Font("Times new roman", Font.BOLD, 14);

                /*settiamo un bordo al mainpanel con un
                 * titleborder(con titolo legenda di color turchino).
                 */
                mainPanel.setBorder(BorderFactory.createTitledBorder(BorderFactory
                                .createLoweredBevelBorder(), "LEGENDA",0, 0, font, new Color(27, 115, 201)));

                /*creaiamo tante label quante sono
                 * le voci della legenda.
                 * Ad ognuna gli settiamo la propria
                 * icona e il testo della label.
                 * Per spaziare tra le voci della legenda,
                 * ad ogni voce creiamo una label senza testo.
                 */
                JLabel l1 = new JLabel(new ImageIcon("icons\\folder_home.png"));
                l1.setText("        : document root");
                JLabel space1 = new JLabel();
                space1.setText("  ");

                JLabel l2 = new JLabel(new ImageIcon("icons\\folder_favorites.png"));
                l2.setText("        : Node containing a choice");
                JLabel l_add2 = new JLabel();
                l_add2.setText("              in its subtrees.");
                JLabel space2 = new JLabel();
                space2.setText("  ");
```

```
JLabel l3 = new JLabel(new ImageIcon("icons\\folder.png"));
l3.setText("        : Generic node.");
JLabel space3 = new JLabel();
space3.setText("  ");

JLabel l4 = new JLabel(new ImageIcon("icons\\folder_green.png"));
l4.setText("        : Choice child node");
JLabel l_add4 = new JLabel();
l_add4.setText("              or Element child of schema.");
JLabel space4 = new JLabel();
space4.setText("  ");

JLabel l5 = new JLabel(new ImageIcon("icons\\greenstar.png"));
l5.setText("        : Choice child node containing");
JLabel l_add5 = new JLabel();
l_add5.setText("              a choice in its subtrees.");
JLabel space5 = new JLabel();
space5.setText("  ");

JLabel l5bis = new JLabel(new ImageIcon("icons\\folder_salmone.png"));
l5bis.setText("     : Choice node.");
JLabel l_add5bis = new JLabel();
JLabel space5bis = new JLabel();
space5bis.setText("  ");

JLabel l5tris = new JLabel(new ImageIcon("icons\\folderRedStar.png"));
l5tris.setText("    : Choice node containing");
JLabel l_add5tris = new JLabel();
l_add5tris.setText("                    a choice in its subtrees.");
JLabel space5tris = new JLabel();
space5tris.setText("  ");

JLabel l6 = new JLabel(new ImageIcon("icons\\attach.png"));
l6.setText("        : Schema's Element children list.");
JLabel space6 = new JLabel();
space6.setText("  ");

JLabel l7 = new JLabel(new ImageIcon("icons\\attachstar.png"));
l7.setText("        : Schema's Element children list containing");
JLabel l_add7 = new JLabel();
l_add7.setText("            a choice in its subtrees.");
JLabel space7 = new JLabel();
space7.setText("  ");

JLabel l8 = new JLabel(new ImageIcon("icons\\unknown.png"));
l8.setText("        : Generic leaf.");
JLabel space8 = new JLabel();
space8.setText("  ");

JLabel l9 = new JLabel(new ImageIcon("icons\\fogliaverde.png"));
l9.setText("        : Choice's leaf child node.");
JLabel space9 = new JLabel();
space9.setText("  ");


//aggiungiamo al mainpanel le label create.
mainPanel.add(l1);
mainPanel.add(space1);

mainPanel.add(l2);
mainPanel.add(l_add2);
mainPanel.add(space2);

mainPanel.add(l3);
mainPanel.add(space3);

mainPanel.add(l4);
mainPanel.add(space4);

mainPanel.add(l5);
mainPanel.add(l_add5);
mainPanel.add(space5);

mainPanel.add(l5bis);
mainPanel.add(l_add5bis);
mainPanel.add(space5bis);

mainPanel.add(l5tris);
mainPanel.add(l_add5tris);
mainPanel.add(space5tris);


mainPanel.add(l6);
mainPanel.add(space6);

mainPanel.add(l7);
mainPanel.add(l_add7);
mainPanel.add(space7);
```

```
                    mainPanel.add(l8);
                    mainPanel.add(space8);

                    mainPanel.add(l9);
                    mainPanel.add(space9);

                    /*rendiamo visibile lo scrollpane
                     * e inseriamoci il mainpanel.
                     */
                    scroll.getViewport().add( mainPanel );

                    /*Impostiamo le dimensioni di this e
                     * aggiungiamogli la nostra scrollpane
                     * al centro(tramite il BorderLayout).
                     */
                    Dimension dimPanel = new Dimension(200, 400);
                    this.setPreferredSize(dimPanel);
                    this.add( scroll, BorderLayout.CENTER );

            }

}
```

TSS.WeigthsAssignment

NodeMap.java

```
package TSS.WeigthsAssignment;

import java.util.*;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */


/**
 * Classe per operare su una struttura dati
 * statica che memorizzi il peso dei nodi
 */
public class NodeMap {

          /**
     *  Struttura dati
     */
    private static HashMap map = new HashMap();

    /**
     * put()
     *
     * Inserisce un nodo col relativo peso all'interno della tabella
     * Nel caso in cui la tabella contenga gia' un'associazione per
     * quel nodo questa verra' sovrascritta con la nuova
     *
     * @param hashCode        -         codice hash del nodo da inserire nella tabella
     * @param peso            -         il peso del nodo
     */
    public static void put(Integer hashCode, Double peso) {

        map.put(hashCode, peso);
    }

    /**
     * containsNode()
     *
     * Controlla se la tabella contiene un associazione per il nodo
     * il cui codice hash viene passato come parametro
     *
     * @param       hashCode  -         il codice hash del nodo da cercare
```

```
     * @return      true                -         se il nodo e' contenuto nella tabella
     *                          false               -         altrimenti
     */
    public static boolean containsNode(Integer hashCode) {

        return map.containsKey(hashCode);
    }

    /**
     * get()
     *
     * Restituisce il valore del nodo avente come codice hash
     * quello passato come parametro
     *
     * @param       hashCode  -         il codice hash del nodo di cui restituire il valore
     * @return      Double              -         il peso del nodo
     *
     */
    public static Double get(Integer hashCode) {

        return (Double) map.get(hashCode);
    }

    /**
     * reset()
     *
     * Cancella tutti gli elementi dalla mappa
     */
    public static void reset(){

        map.clear();
    }

    /**
     * keySet()
     *
     * Restituisce l'insieme di chiavi mappate nella tabella
     *
     * @return      Set       -         l'insieme di chiavi mappate nella tabella
     */
    public static Set keySet(){

        return map.keySet();
    }

    /**
     * size()
     *
     * restituisce il numero di elementi nella mappa
     *
     * @return int - il numero di elementi nella mappa
     */
    public static int size(){

          return map.size();
    }

    public static void print() {

        System.out.println(map.toString());
    }
}
```

FlagMap.java

```
package TSS.WeigthsAssignment;

import java.util.HashMap;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

public class FlagMap {
```

```
    /**
     *
     */
    static HashMap flag_map = new HashMap();

    /**
     * put()
     *
     * Inserisce un nodo col relativo stato (modificato, non modificato)
     * all'interno della tabella. Nel caso in cui la tabella contenga
     * gia' un'associazione per quel nodo questa verra' sovrascritta
     * con la nuova
     *
     * @param hashCode        -         codice hash del nodo da inserire nella tabella
     * @param peso            -         lo stato del nodo
     */

    public static void put(Integer hashNode, Boolean flag) {

        flag_map.put(hashNode, flag);
    }

    public static void remove(Integer hashCode){

          flag_map.remove(hashCode);
    }

    /**
     * containsKey()
     *
     * Controlla se la tabella contiene un associazione per il nodo
     * il cui codice hash viene passato come parametro
     *
     * @param       hashCode  -         il codice hash del nodo da cercare
     * @return      true                -         se il nodo e' contenuto nella tabella
     *                        false               -         altrimenti
     */
    public static boolean containsKey(Integer node) {

        return flag_map.containsKey(node);
    }

    /**
     * get()
     *
     * Restituisce il valore del nodo avente come codice hash
     * quello passato come parametro
     *
     * @param       hashCode  -         il codice hash del nodo di cui restituire il valore
     * @return      Boolean             -         lo stato del nodo
     *
     */
    public static Boolean get(Integer hashCode) {

        return (Boolean) flag_map.get(hashCode);
    }

    /**
     * isEmpty()
     *
     * Controlla se la mappa e' vuota
     *
     * @return      true                -         se la mappa e' vuota
     *                        false               -         altrimenti
     */
    public static boolean isEmpty(){

          return flag_map.isEmpty();
    }

    /**
     * reset()
     *
     * Cancella tutti gli elementi dalla mappa
     */
    public static void reset(){

          flag_map.clear();
    }

    public static void print() {

        System.out.println(flag_map.toString());
    }
}
```

RowMap.java

```
package TSS.WeigthsAssignment;

import java.util.*;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

public class RowMap {

          private static HashMap map = new HashMap();

          /**
           * put()
           *
           * Inserisce un nodo col relativo numero di riga all'interno della tabella
           * Nel caso in cui la tabella contenga gia' un'associazione per
           * quel nodo questa verra' sovrascritta con la nuova
           *
           * @param hashCode  -         codice hash del nodo da inserire nella tabella
           * @param row                -         la riga del nodo
           */
          public static void put(Integer hashCode, int row) {

                    map.put(hashCode, new Integer(row));
          }

          /**
           * contains()
           *
           * Controlla se la tabella contiene un associazione per il nodo
           * il cui codice hash viene passato come parametro
           *
           * @param           hashCode  -         il codice hash del nodo da cercare
           * @return          true                -         se il nodo e' contenuto nella tabella
           *                            false               -         altrimenti
           */
          public boolean contains(Integer hashCode) {

                    return map.containsKey(hashCode);
          }

          /**
           * get()
           *
           * Restituisce la riga del nodo avente come codice hash
           * quello passato come parametro
           *
           * @param           hashCode  -         il codice hash del nodo di cui restituire il valore
           * @return          Integer             -         la riga del nodo
           *
           */
          public static Object get(Integer hashCode) {

                    return map.get(hashCode);
          }

          /**
           * keySet()
           *
           * Restituisce l'insieme di chiavi mappate nella tabella
           *
           * @return          Set       -         l'insieme di chiavi mappate nella tabella
           */
          public static Set keySet() {

                    return map.keySet();
          }

          /**
           * size()
           *
           * restituisce il numero di elementi nella mappa
           *
           * @return int - il numero di elementi nella mappa
           */
```

```
	public static int size() {

		return map.size();
	}

	public static void print() {

		System.out.println(map.toString());
	}

	/**
	 * reset()
	 *
	 * Cancella tutti gli elementi dalla mappa
	 */
	public static void reset() {

		map.clear();
		DefaultInitializer.counter = 0;
	}
}
```

RowCounter.java

```
package TSS.WeigthsAssignment;

import org.w3c.dom.*;

import TAXInterface.file.*;

/**
 * @author		Romina Paradisi
 *			256283
 *			paradisi@cli.di.unipi.it
 *
 * @author		Maurizio Santoni
 *			244237
 *			santon@cli.di.unipi.it
 */

public class RowCounter {

	/**
	 * count()
	 *
	 * Semplice funzione ricorsiva che calcola la riga
	 * dell'albero a cui ciascun nodo corrisponde
	 *
	 * @param count - il contatore di riga
	 * @param node  - la radice del documento su cui lavorare
	 * @return row      - la riga di ciascun nodo
	 */
	public static int count(int count , Node node){

		if(!node.getNodeName().equals("ISTI_et_CNR_pesi"))
			count++;

		int row = count;

		FileNode [] figli = (FileNode[]) new FileNode(node).getChildren();
		// mi interessa salvare le righe dei nodi che dovro' modificare
		// per andare a settare il peso assegnato al nodo
		// alla riga esatta nella tabella
		if(NodeMap.containsNode(new Integer(node.hashCode()))){
			RowMap.put(new Integer(node.hashCode()), row);
			System.out.println("NODE PUT: "+node+", hash Code: "+new Integer(node.hashCode())+",
riga: "+row);
		}
		for(int i = 0; i < figli.length; i++)
			row = count(row, figli[i].getDom());

		return row;
	}
}
```

DefaultInitializer.java

```
package TSS.WeigthsAssignment;

import org.w3c.dom.*;

import TAXInterface.file.*;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

public class DefaultInitializer {

          public static int counter = 0;
          /**
           * initializer()
           *
           * @param           elem      -         inizializza le tabelle che mantengono i pesi dei nodi
                                                          e il loro stato a valori di default se il file e'
           *                                             un file che non contiene pesi ai nodi. In caso
contrario
           *                                             setta nelle mappe i valori precedentemente
assegnati
           *                                             dall'utente
           */
          public static void initializer(Element elem){

                    NodeList nodes = elem.getChildNodes();

                    for (int i = 0; i < nodes.getLength(); i++) {
                              Node figlio = nodes.item(i);

                              if (figlio instanceof Element) {
                                        Element prova = (Element) figlio;
                                        if ((i != nodes.getLength() - 1))
                                        initializer(prova);
                                        FileNode parent = new FileNode(figlio.getParentNode());
                                        Double default_value;
                                        /*
                                         * Se il padre del nodo incontrato e' una choice controlliamo se
                                         * questo ha figli di tipo ISTI_et_CNR_pesi come peso di default
                                         * gli assegnamo il valore memorizzato nel figlio altrimenti
                                         * gli assegnamo un valore di default calcolato dividendo 1
                                         * per il numero dei fratelli del nodo piu il nodo stesso
                                         * in modo tale che la somma dei loro pesi sia 1.
                                         */
                                        if (parent.isChoice()) {
                                                  // Il figlio non ha figlio ISTI_et_CNR_pesi
                                                  // quindi e' la prima volta che l'utente
                                                  // modifica il file
                                                  if (!(new FileNode(figlio).hasIstiChild())) {
                                                            int b = parent.getChildren().length;

                                                            if (b != 0) {

                                                                      default_value = new Double(1.0 / b);

                                                                      String v = default_value.toString();
                                                                      if (v.length() > 6) {
                                                                                v = v.substring(0, 6);
                                                                                default_value = Double.valueOf(v);
                                                                      }
                                                            } else
                                                                      default_value = null;
                                                            // Inserisco i valori calcolati nella mappa
                                                            // che tiene memoria dei pesi
                                                            if (default_value != null) {
                                                                      NodeMap.put(new Integer(new FileNode(figlio)
                                                                                          .hashCode()), default_value);

                                                                      // Essendo la prima volta che apro il file per
                                                                      // inserire i pesi ai nodi questi vengono
                                                                      // settati come NON modificati
                                                                      FlagMap.put(new Integer(new FileNode(figlio)
                                                                                          .hashCode()), new
Boolean(false));
```

```
                                                  } else
                                                            NodeMap.put(new Integer(new FileNode(figlio)
                                                                      .hashCode()), null);
                                        } else {
                                                  NodeList list = figlio.getChildNodes();
                                                  // Se il file invece e' gia' stato modificato
                                                  // carico nelle tabelle i valori
                                                  // precedentemente assegnati dall'utente
                                                  // e lo stato dei nodi (modificato, non modificato)
                                                  // in modo tale da ripristinare la sessione
                                                  // lasciata al momento del salvataggio
                                                  for (int h = 0; h < list.getLength(); h++) {
                                                            // scorro tutti gli attributi fino a trovare
                                                            // l'attributo peso dove e' memorizzato il
                                                            // valore che mi serve
                                                            if (list.item(h).hasAttributes()) {
                                                                      NamedNodeMap map =
list.item(h).getAttributes();
                                                                      Attr[] attributi_choice = new Attr[map

.getLength()];

                                                                      for (int j = 0; j < map.getLength();
j++) {
                                                                                attributi_choice[j] = (Attr)
map.item(j);
                                                                      }
                                                                      for (int k = 0; k <
attributi_choice.length; k++) {
                                                                                if
(attributi_choice[k].getNodeName()
          .equals("peso")) {

                                                                                          NodeMap
                                                                                          .put(
          new Integer(
                              new FileNode(
                                        figlio)
                                        .hashCode()),
                                        new Double(
                                                  Double
                                                  .parseDouble((attributi_choice[k]

.getNodeValue())))));
                                                                                }
                                                                                // se trovo l'attributo
modificato
                                                                                // lo inserisco nell'apposita
mappa di flag
                                                                                if
(attributi_choice[k].getNodeName()
          .equals("modificato")) {
                                                                                          FlagMap.put(new
Integer(new FileNode(
          figlio).hashCode()),
          new Boolean(attributi_choice[k]
                                        .getNodeValue()));
                                                                                }
                                                                      }
                                                            }
                                                  }
                                        }
                              }
                    }
          }
}
```

XmlFileBuilder.java

```
package TSS.WeigthsAssignment;
import java.io.*;

import org.w3c.dom.*;

import TAXInterface.file.FileNode;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

public class XmlFileBuilder {

          // stringhe utili per la stampa su file

    private String inizio = "<";

    private String fine = ">";

    private String sbarra = "/";

    private String virgole = "\"";

    private String uguale = "=";

    //  documento su cui lavorare
    private Document doc;

    // nome del nuovo file da costruire
    private String nomefile;

    private static int counter = 0;

    public XmlFileBuilder(Document doc, String nomefile) {

        this.doc = doc;
        this.nomefile = nomefile;
    }

    public void create() throws IOException {

          FileWriter fileout = new FileWriter(nomefile + ".xml");
          BufferedWriter filebuf = new BufferedWriter(fileout);
          PrintWriter printout = new PrintWriter(filebuf);
          // stampiamo l'inizio del file
          printout.println(inizio + "?xml version=" + virgole + "1.0" + virgole
                           + " encoding=" + virgole + "ISO-8859-1" + virgole + "?" + fine);

          Element elem = doc.getDocumentElement();
          // scorriamo il documento doc e
          // stampiamo tutti gli elementi del documento
          try {
                    createfileIstanza(elem, printout);
          } catch (IOException IOE) {
          }
          printout.close();
    }

    public void createfileIstanza(Element elem, PrintWriter printout)
    throws IOException {

          if (!elem.getNodeName().equals("ISTI_et_CNR_pesi")) {

                    //si stampa il nome del nodo
                    String line = "\n" + inizio + elem.getTagName();
                    // i suoi attributi
                    boolean has_elem = elem.hasAttributes();
                    if (has_elem == true) {

                              NamedNodeMap nnm = elem.getAttributes();
                              for (int i = nnm.getLength()-1; i >= 0; i--) {

                                        line = line + " " + (nnm.item(i)).getNodeName() + uguale
                                        + virgole + (nnm.item(i)).getNodeValue() + virgole;

                              }
```

```
		}
		// si scorrono tutti i figli del nodo
		// prima di stampare il tag di chiusura
		boolean has_child = elem.hasChildNodes();
		if (has_child == true
				|| new FileNode(elem.getParentNode()).isChoice()) {
			line = line + fine;
		} else {
			line = line + sbarra + fine;
		}
		printout.print(line);

		NodeList nodes = elem.getChildNodes();

		for (int i = 0; i < nodes.getLength(); i++) {
			Node figlio = nodes.item(i);

			if (figlio.getNodeName().equals("#text")) {

				printout.print("\n");//figlio.getNodeValue());
			}

			// si chiama ricorsivamente la funzione
			if (figlio instanceof Element) {
				Element prova = (Element) figlio;
				if ((i != nodes.getLength() - 1))
					createfileIstanza(prova, printout);

				FileNode temp = new FileNode(figlio);

				boolean modificato;
				// nel caso in cui un nodo dell'albero
				// abbia un peso assegnato per salvarlo
				// nel file va prelevato dalle strutture
				// dati in cui e' memorizzato. Viene
				// inoltre salvato l'attributo
				// modificato / non modificato in modo tale
				// che quando il file verra' nuovamente
				// caricato sia ripristinata la situazione
				// al momento del salvataggio
				if (temp.hasParentNode()
						&& (temp.getParentNode().isChoice() || temp
								.getParentNode().isElementsList())
								&& !temp.hasIstiChild()) {

					String newTag = "\n" + inizio + "ISTI_et_CNR_pesi ";

					FileNode nodino = new FileNode(figlio);
					newTag += "peso" + uguale + virgole;
					newTag += NodeMap.get(new Integer(nodino.hashCode()))
					.toString()
					+ virgole + " ";
					newTag += "modificato" + uguale + virgole;

					if (FlagMap.get(new Integer(temp.hashCode()))
								.booleanValue())
						modificato = true;
					else
						modificato = false;

					newTag += modificato + virgole + sbarra + fine + "\n";
					newTag += inizio + sbarra + figlio.getNodeName() + fine
					+ "\n";
					printout.print(newTag);
					counter++;
				}

			}
		}
		// stampa del tag di chiusura
		if (!line.endsWith(sbarra + fine)
				&& !(new FileNode(elem.getParentNode()).isChoice()))
			printout.print(inizio + sbarra + elem.getNodeName() + fine
					+ "\n");

		/**if (line.contains("choice")) {
		 printout.print("\n"+inizio + "ISTI_et_CNR_pesi" + sbarra + fine);
		 }*/
	}
	// nel caso in cui avessimo incontrato un nodo del tipo
	// ISTI_et_CNR_pesi per evitare di stampare un nuovo figlio
	// ISTI_et_CNR_pesi ad ogni salvataggio si ripete la stampa
	// come sopra evitando di trattare questo nodo come un nodo
	// comune
	else {
		FileNode padre = new FileNode(elem.getParentNode());
		String newTag = "\n" + inizio + "ISTI_et_CNR_pesi ";
		newTag += "peso" + uguale + virgole;
		newTag += NodeMap.get(new Integer(padre.hashCode())).toString()
		+ virgole + " ";
```

```
                    newTag += "modificato" + uguale + virgole;
                    boolean modificato;
                    if (FlagMap.get(new Integer(padre.hashCode())).booleanValue())
                              modificato = true;
                    else
                              modificato = false;

                    newTag += modificato + virgole + sbarra + fine + "\n";
                    newTag += inizio + sbarra + padre.StringNodeName() + fine /*+ "\n"*/;
                    printout.print(newTag);
          }
     }
}
```

ReaderWriter.java

```
package TSS.WeigthsAssignment;

import java.io.*;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

public class ReaderWriter {

          /**
           * costruttore
           *
           * Verifica che una stringa contenuta in file xml non sia composta di soli
           * spazi. Se si non la stampa nel file di appoggio. Una volta ultimato il
           * file di appoggio viene copiato nel file di origine. Questo e'
           * indispensabile in quanto applicazioni successive che dovranno lavorare su
           * questo file non gestiscono eventuali spaziature inaspettate.
           *
           * @param file -
           *            il file da cui eliminare spazi inutili
           */

          public ReaderWriter(String file) {
                    BufferedReader filebuf = null;
                    FileWriter fileout = null;
                    BufferedWriter filebuf1 = null;
                    PrintWriter printout = null;
                    try {
                              filebuf = new BufferedReader(new FileReader(file));
                              try {
                                        fileout = new FileWriter("appoggio.xml");
                              } catch (IOException e) {
                                        e.printStackTrace();
                              }
                              filebuf1 = new BufferedWriter(fileout);
                              printout = new PrintWriter(filebuf1);
                    } catch (FileNotFoundException e) {
                              e.printStackTrace();
                    }

                    String nextStr = null;
                    try {
                              nextStr = filebuf.readLine();
                              if (nextStr != null && belongsToSchema(nextStr)) {
                                        int i = nextStr.indexOf('<');
                                        nextStr = nextStr.substring(i, nextStr.length());
                                        printout.println(nextStr);
                              }
                    } catch (IOException e) {
                              e.printStackTrace();
                    } // legge una riga del file
                    while (nextStr != null) {
                              System.out.println(nextStr); // visualizza la riga
                              try {
                                        nextStr = filebuf.readLine();
                                        if (nextStr != null && belongsToSchema(nextStr)) {
                                                  int i = nextStr.indexOf('<');
                                                  nextStr = nextStr.substring(i, nextStr.length());
```

```
                                        printout.println(nextStr);
                                        System.out.println("fiffo");
                                }
                        } catch (IOException e) {
                                e.printStackTrace();
                        } // legge la prossima riga
                }
                try {
                        filebuf.close();
                        filebuf1.close();
                } catch (IOException e) {
                        e.printStackTrace();
                } // chiude il file

                try {
                        filebuf = new BufferedReader(new FileReader("appoggio.xml"));
                        try {
                                fileout = new FileWriter(file);
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                        filebuf1 = new BufferedWriter(fileout);
                        printout = new PrintWriter(filebuf1);
                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                }

                nextStr = null;
                try {
                        nextStr = filebuf.readLine();
                        if (nextStr != null && belongsToSchema(nextStr)) {
                                int i = nextStr.indexOf('<');
                                nextStr = nextStr.substring(i, nextStr.length());
                                printout.println(nextStr);
                        }
                } catch (IOException e) {
                        e.printStackTrace();
                } // legge una riga del file
                while (nextStr != null) {
                        System.out.println(nextStr); // visualizza la riga
                        try {
                                nextStr = filebuf.readLine();
                                if (nextStr != null && belongsToSchema(nextStr)) {
                                        int i = nextStr.indexOf('<');
                                        nextStr = nextStr.substring(i, nextStr.length());
                                        printout.println(nextStr);
                                        System.out.println("fiffo");
                                }
                        } catch (IOException e) {
                                e.printStackTrace();
                        } // legge la prossima riga
                }
                try {
                        filebuf.close();
                        filebuf1.close();
                } catch (IOException e) {
                        e.printStackTrace();
                } // chiude il file

        }

        /**
         * belongsToSchema()
         *
         * @param s -
         *            La stringa da analizzare
         * @return true - Se la stringa non e' composta di soli spazi false -
         *         altrimenti
         */
        private static boolean belongsToSchema(String s) {

                if (s.contains("<") && s.contains(">"))
                        return true;
                else
                        return false;
        }
}
```

TSS.ChoiceAnalysis

Redistributor.java

```
package TSS.ChoiceAnalysis;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

import TAXInterface.file.FileNode;

/**
 *        La classe Normalizer fornisce i metodi per la
 *        ridistribuzione dei pesi ai figli dei nodi choice
 *        nel caso in cui venga applicata loro una
 *        'restriction' o una 'extension'
 */
public class Redistributor {

          // documento su cui si lavora
          private Document doc;

          /**
           * Costruttore
           *
           * @param doc - il documento su cui si opera la ridistribuzione
           */
          public Redistributor(Document doc){

                    this.doc = doc;
          }


          /**
           *        redistribute()
           *
           *        Implementa una visita ricorsiva che ci consente
           *        di vedere se ad una choice sono stati tolti o
           *        aggiunti figli.
           *        In questo caso viene chiamato l'opportuno metodo
           *        per attuare la politica la ridistribuzione.
           *
           *        @param root         -         la radice del documento su cui lavorare
           *        @return doc         -         il documento con radice root su cui e'
           *                                      stata attuata la ridistribuzione dei pesi
           *                                      ai figli dei nodi choice
           */
          public Document redistribute(Node root){

                    double somma = 0.0;

                     NodeList figli = root.getChildNodes();
                     double[] pesi = new double[figli.getLength()];
                     if(root != null){
                              if(!root.getNodeName().equals("#text")){
                                        if(root.getNodeName().equals("ISTI_et_CNR_pesi"))
                                                  System.out.println(""+root+new
FileNode(root.getParentNode()).getIstiVal());
                                        else
                                                  System.out.println(""+root);
                              }
                              // se incontro un nodo di tipo choice
                              // controllo tutti i figli e i loro pesi
                              // calcolandone la somma
                               if(new FileNode(root).isChoice()){
```

```
                                        for(int i = 0; i < figli.getLength(); i++){

                                                  FileNode f = new FileNode(figli.item(i));

                                                  if(f.hasIstiChild()){
                                                           somma = somma + f.getIstiVal();
                                                          // System.out.println("SOMMA:"+ somma);

                                                           pesi[i] = f.getIstiVal();
                                                           System.out.println("PESO[i]:"+ pesi[i]);
                                                  }
                                                  // in questo caso ho trovato un figlio di choice senza peso
                                                  // cioe' un figlio certamente aggiunto in seguito e quindi
                                                  // non presente in origine
                                                  else  pesi[i] = 0;
                                        }
                                //        System.out.println("SOMMA:"+ somma);
                                        // Nel caso in cui la somma sia minore di 1
                                        // almeno un figlio e stato eliminato
                                        // Nel caso in cui la somma sia maggiore di 1
                                        // almeno un figlio (con peso) e' stato aggiunto.
                                        // Entrambe le situazioni possono essere risolte
                                        // applicando lo stesso tipo di proporzione
                                        // svolta dal metodo handle()
                                        if(somma > 1 || somma < 1)
                                                  handle(1, figli, pesi, somma);
                                        // Nel caso in cui la somma dei pesi sia uguale ad 1
                                        // o i figli non sono stati modificati oppure
                                        // e stato aggiunto un figlio senza peso
                                        // alla choice. Questo verra verificato
                                        // dal metodo handle()
                                        if(somma == 1)
                                                  handle(2, figli, pesi, somma);


                                 }
                                 for(int i = 0; i < figli.getLength(); i++)
                                          redistribute(figli.item(i));
                        }
                        return doc;
                }

                private void handle(int caso, NodeList figli, double[] pesi,
                                 double somma) {

                        switch (caso) {

                        case 1:
                                 for (int i = 0; i < figli.getLength(); i++) {

                                          FileNode f = new FileNode(figli.item(i));

                                          if (f.hasIstiChild()) {
                                                   NodeList figli2 = figli.item(i).getChildNodes();

                                                   for (int j = 0; j < figli2.getLength(); j++) {
                                                            if (figli2.item(j).getNodeName().equals(
                                                            "ISTI_et_CNR_pesi")) {

                                                                     // si assegna il nuovo peso normalizzato ad 1
                                                                     // ai figli la cui somma dei pesi era
                                                                     // maggiore (minore) di 1
                                                                     // dividendo il loro peso per la somma totale dei
                                                                     // pesi
                                                                     ((Element)
figli2.item(j)).removeAttribute("peso");
                                                                     ((Element) figli2.item(j)).setAttribute("peso",
""
                                                                                       + pesi[i] / somma);
                                                                     System.out.println("NEWWWWW: "+((Element)
figli2.item(j)).getAttribute("peso"));
                                                                     System.out.println("NEW: "+pesi[i] / somma);
                                                                     NamedNodeMap m = figli2.item(j).getAttributes();
                                                                     System.out.println("ATTRIBUTI");
                                                                     for (int k = 0; k < m.getLength(); k++){

        if(m.item(k).getNodeName().equals("peso")){

        System.out.println(m.item(k).getNodeName());

        System.out.println(m.item(k).getNodeValue());
                                                                              }
                                                                     }
                                                            }
                                                   }
                                          }
                                 }

                        case 2:
```

```
                        int count = 0;

                        for (int i = 0; i < figli.getLength(); i++) {

                                FileNode f = new FileNode(figli.item(i));
                                // conto il numero dei figli di choice
                                // che non hanno peso
                                if (!f.StringNodeName().equals("#text") && !f.hasIstiChild())
                                        count++;
                        }
                        // sommo queste unita alla somma dei pesi
                        somma += count;

                        for (int i = 0; i < figli.getLength(); i++) {

                                FileNode f = new FileNode(figli.item(i));

                                if (f.hasIstiChild()) {
                                        NodeList figli2 = figli.item(i).getChildNodes();

                                        for (int j = 0; j < figli2.getLength(); j++) {
                                                if (figli2.item(j).getNodeName().equals(
                                                "ISTI_et_CNR_pesi")) {
                                                        // si assegna un nuovo peso (normalizzato a 1)
                                                        // a ciascun figlio che ha gia peso
                                                        ((Element)
figli2.item(j)).removeAttribute("peso");
                                                        ((Element) figli2.item(j)).setAttribute("peso",
""
                                                                        + pesi[i] / somma);
                                                }
                                        }
                                } else {
                                        if(!f.StringNodeName().equals("#text") && !f.hasIstiChild()){
                                                Node isti = doc.createElement("ISTI_et_CNR_pesi");
                                                // si assegna peso
                                                // a ciascun figlio non avente peso
                                                // supponendo che il vecchio peso fosse 1
                                                ((Element) isti).setAttribute("peso", ""
                                                                + 1.0 / somma);

                                                // si appende il figlio contenente il peso
                                                // sotto il nodo interessato
                                                ((Element) isti).setAttribute("modificato", "false");
                                                figli.item(i).appendChild(isti);
                                        }
                                }
                        }
                }
        }

        public void print(Node root){
                NodeList figli = root.getChildNodes();

                if(root != null){
                        if(!root.getNodeName().equals("#text")){

                                if(root.getNodeName().equals("ISTI_et_CNR_pesi"))
                                        System.out.println("" + root + " " + new
FileNode(root.getParentNode()).getIstiVal());
                                else
                                        System.out.println("" + root);
                        }
                         for(int i = 0; i < figli.getLength(); i++){
                                print(figli.item(i));
                         }
                 }
                }
}
```

Paths.java

```
package TSS.ChoiceAnalysis;

import java.util.*;

import org.w3c.dom.*;

import TAXInterface.file.FileNode;

/**
```

```
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

public class Paths {


        /**
         * generate()
         *
         * Genera ricorsivamente tutti i path radice-foglia possibili
         * all'interno di un albero dom, inserendo i nodi del path
         * all'interno di una lista
         *
         * @param           root                -         la radice dell'albero
         * @return          all_paths -         tutti i paths dell'albero
         */
        public static LinkedList generate(Node root) {

                LinkedList all_paths = new LinkedList();

                // si evita di aggiungere nodi fittizzi
                // aggiunti dal programmatore
                if (!root.getNodeName().equals("ISTI_et_CNR_pesi")) {

                        FileNode tmp = new FileNode(root);
                        FileNode[] figli = (FileNode[]) tmp.getChildren();

                        // Se il nodo e' una foglia:
                        if (figli.length == 0) {

                                LinkedList path = new LinkedList();

                                // se il nodo ha un figlio ISTI_et_CNR_pesi
                                // si aggiunge al path un nuovo FileNode
                                // recuperandone il peso all'interno del suo
                                // figlio ISTI_et_CNR_pesi, altrimenti settiamo
                                // il suo peso a null
                                if (tmp.hasIstiChild())
                                        path.add(new FileNode(root, new Double(tmp.getIstiVal())));
                                else
                                        path.add(new FileNode(root, null));
                                // aggiungo il path calcolato alla lista contenente
                                // tutti i path
                                all_paths.add(path);

                        } else {
                                // Applico la funzione ricorsivamente ai figli
                                // del nodo
                                for (int i = 0; i < figli.length; i++) {
                                        LinkedList partial_path;

                                        partial_path = generate(figli[i].getDom());

                                        Iterator itr = partial_path.listIterator();

                                        LinkedList next;

                                        next = (LinkedList) itr.next();
                                        //Si aggiunge tmp col suo peso
                                        // alle liste generate
                                        while (next != null) {

                                                if (tmp.hasIstiChild())
                                                        next.add(new FileNode(root, new Double(tmp
                                                                        .getIstiVal())));
                                                else
                                                        next.add(new FileNode(root, null));

                                                all_paths.add(next);

                                                if (itr.hasNext())
                                                        next = (LinkedList) itr.next();
                                                else
                                                        next = null;
                                        }
                                }
                        }
                }
                return all_paths;
        }

        /**
         * validatePaths()
         *
```

```
 * Scarta i path non interessanti dalla lista
 * di tutti i path. I path non interessanti sono quelli
 * che non contengono choice al loro interno.
 * Inoltre dei path non rimuovibili rimuove il primo
 * elemento fino ad ottenere un path il cui ultimo elemento
 * foglia e' figlio di choice
 *
 * @param           all_paths - una lista contenente tutti i path
 * @return          all_paths - modificata contenente solo i paths interessanti
 */
public static LinkedList validatePaths(LinkedList all_paths) {

          // iteratore sulla lista di liste (paths)
          Iterator itr = all_paths.listIterator();

          LinkedList removable = new LinkedList();
          LinkedList one_path;

          one_path = (LinkedList) itr.next();

          FileNode tmp = null;
          tmp = (FileNode) one_path.getFirst();

          while (one_path != null) {
                    // iteratore sul singolo path
                    Iterator iter = one_path.listIterator();

                    FileNode nnext;

                    nnext = (FileNode) iter.next();

                    boolean contains_choice = false;

                    while (nnext != null && !contains_choice) {

                              if(nnext.isChoice())
                                        contains_choice = true;

                              if (iter.hasNext())
                                        nnext = (FileNode) iter.next();
                              else
                                        nnext = null;
                    }
                    // se un path non contiene choice va eliminato
                    if(!contains_choice)
                              itr.remove();

                    if (itr.hasNext())
                              one_path = (LinkedList) itr.next();
                    else
                              one_path = null;
          }

          itr = all_paths.listIterator();


          one_path = (LinkedList) itr.next();

          tmp = null;

          while (one_path != null) {

                    Iterator iter = one_path.listIterator();

                    FileNode nnext;

                    nnext = (FileNode) iter.next();

                    boolean is_choice = false;

                    while (nnext != null && !is_choice) {

                              // rimuovo tutti i nodi finchè non trovo una choice
                              if(nnext.isChoice())
                                        is_choice = true;
                              else{
                                        tmp = nnext;
                                        iter.remove();
                              }

                              if (iter.hasNext())
                                        nnext = (FileNode) iter.next();
                              else
                                        nnext = null;
                    }
                    // se trovo una choice riaggiungo
                    // alla lista l'ultimo elemento eleiminato
                    one_path.addFirst(tmp);

                    if(all_paths.indexOf(one_path) != all_paths.lastIndexOf(one_path))
```

```
                                removable.add(one_path);

                        if (itr.hasNext())
                                one_path = (LinkedList) itr.next();
                        else
                                one_path = null;
                }
                // rimuovo tutti i paths non validi
                for(int i = 0; i < removable.size(); i++)
                    all_paths.remove(removable.get(i));

                removable.clear();
                return all_paths;
        }

        /**
         * calculate()
         *
         * Calcola per ogni path valido il peso associato alla foglia
         * Scorre tutto il path e moltiplica tutti i pesi dei nodi (con peso)
         * che incontra. Questo sara' il peso da associare alla foglia
         *
         * @param           all_paths -         tutti i path validi
         * @return          Vector              -         il vettore contenente i pesi di tutti i path.
         *                                                                      Al primo elemento del vettore
corrisponde il
         *                                                                      primo path contenuto in all_paths e
cosi via
         */
        public static Vector calculate(LinkedList all_paths){
                // iteratore sulla lista di liste (paths)
                Iterator itr = all_paths.listIterator();

                LinkedList one_path;
                Vector w = new Vector();

                one_path = (LinkedList) itr.next();

                while (one_path != null) {

                        double count = 1;
                        // iteratore sul path
                        Iterator iter = one_path.listIterator();

                        FileNode nnext;

                        nnext = (FileNode) iter.next();


                        while (nnext != null) {
                                // prelevo il peso di ogno nodo che incontro
                                Double p = nnext.getPeso();
                                // se e' diverso da null lo moltiplico
                                // per count, variabile che memorizza il
                                // peso del path
                                if(p != null)
                                        count = count * p.doubleValue();

                                if (iter.hasNext())
                                        nnext = (FileNode) iter.next();
                                else
                                        nnext = null;
                        }
                        // memorizzo il peso appena calcolato
                        // nel vettore da restituire
                        w.addElement(new Double(count));

                        if (itr.hasNext())
                                one_path = (LinkedList) itr.next();
                        else
                                one_path = null;
                }

                return w;

        }

        /**
         * leaves()
         *
         * Restituisce un vettore contenente tutte le foglie.
         * Vengono ottenute prelevando da ogni path il primo
         * elemento della lista.
         *
         * @param           all_paths -         tutti paths
         * @return          Vector      -       le foglie
         */
        public static Vector leaves(LinkedList all_paths){
                //iteratore sulla lista di liste (paths)
                Iterator itr = all_paths.listIterator();
```

```
                Vector leafs = new Vector();
                LinkedList one_path;

                one_path = (LinkedList) itr.next();

                while (one_path != null) {

                        // aggiungo al vettore delle foglie
                        // il primo elemento di ogni path
                        leafs.addElement(one_path.getFirst());

                        if (itr.hasNext())
                                one_path = (LinkedList) itr.next();
                        else
                                one_path = null;
                }

                return leafs;

        }

        public static void printPaths(LinkedList all_paths) {

                Iterator itr = all_paths.listIterator();

                LinkedList next;

                next = (LinkedList) itr.next();
                System.out.print("[ ");

                while (next != null) {
                        System.out.print("[ ");
                        Iterator iter = next.listIterator();

                        FileNode nnext;

                        nnext = (FileNode) iter.next();

                        while (nnext != null) {

                                System.out.println(nnext.toString2() + ", ");
                                if (iter.hasNext())
                                        nnext = (FileNode) iter.next();
                                else
                                        nnext = null;
                        }
                        System.out.print(" ]\n");

                        if (itr.hasNext())
                                next = (LinkedList) itr.next();
                        else
                                next = null;
                }
                System.out.print(" ]");
        }
}
```

ThreeWeights.java

```
package TSS.ChoiceAnalysis;

import java.io.*;
import java.util.*;

import javax.xml.parsers.*;

import org.w3c.dom.*;
import org.xml.sax.*;

import TAXInterface.file.*;
import TSS.ChoiceAnalysis.Paths;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */
```

```
public class TreeWeights {
	// vettori delle foglie e dei relativi pesi
	private Vector leaves;
	private Vector weights;

	/**
	 * costruttore
	 *
	 * @param leaves    -         un vettore contenete le foglie di un albero
	 * @param weights   -         un vettore parallelo a quello delle foglie
	 *                                                    contenente i pesi delle foglie
	 */
	public TreeWeights(Vector leaves, Vector weights){

		this.leaves = leaves;
		this.weights = weights;
	}

	/**
	 * getWeight()
	 *
	 * Calcola ricorsivamente il peso dell'albero come somma dei pesi
	 * delle sue foglie. Le foglie dell'albero sono
	 * contenute nel vettore delle foglie leaves e i pesi
	 * delle foglie sono contenuti nel vettore weights
	 *
	 * @param           root      -         la radice di un albero di cui
	 *                                                    si deve calcolare il peso come
	 *                                                    somma dei pesi delle foglie
	 * @param           p                   -         il peso iniziale dell'albero
	 * @return          peso      -         il peso dell'albero
	 */
	public double getWeight(Node root, double p){

		FileNode r = new FileNode(root);
		// vettore contenente i figli della radice
		FileNode[] children = (FileNode[]) r.getChildren();

		double peso=p;
		// se il nodo radice è una foglia si controlla se
		// appartiene al vettore dell foglie. se si si preleva
		// il relativo peso alla corrispondente posizione
		// nel vettore dei pesi e lo sommiamo a 'peso', il peso
		// totale dell'albero
		if(children.length == 0){
			if(leaves.contains(r)){
				peso += ((Double) weights.elementAt(leaves.indexOf(r))).doubleValue();
			}
		}
		else{
			// altrimenti si applica la funzione ad ognuno dei suoi figli
			// e se ne somma il risultato a 'peso'
			for(int i=0; i<children.length; i++){
				peso += getWeight(children[i].getDom(), p);
			}
		}
		return peso;

	}

	public static void main(String[] args){

		File file = null;
		String path = "C:\\Documents and Settings\\Administrator\\workspace\\TAXIFusioneInterface\\xml
files\\cicciobariccio2.xml";
		DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
		DocumentBuilder builder = null;
		try {
			builder = factory.newDocumentBuilder();
		} catch (ParserConfigurationException e) {
			e.printStackTrace();
		}
		Document document = null;
		try {
			file = new File(path);

			document = builder.parse(file);
		} catch (SAXException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}


		LinkedList all_paths = Paths.generate(document.getDocumentElement());
		LinkedList all_valid_paths = Paths.validatePaths(all_paths);
		System.out.println("PATH VALIDI\n");
		Paths.printPaths(all_valid_paths);
		Vector w = Paths.calculate(all_valid_paths);
```

```
                    Vector leafs = Paths.leaves(all_valid_paths);
                    System.out.println("foglie SIZE: " + leafs.size());
                    System.out.println(leafs);
                    System.out.println("pesi SIZE: " + w.size());
                    System.out.println(w);
                    TreeWeights tw = new TreeWeights(leafs, w);

                    System.out.println(tw.getWeight(document.getDocumentElement(), 0));
          }
}
```

TSS.StrategySelection

MultiComponent.java

```
package TSS.StrategySelection;

import java.awt.*;
import java.awt.event.*;
import java.util.Vector;

import javax.swing.*;
import javax.swing.event.*;

import org.w3c.dom.Document;

import TAXInterface.graphics.Alert;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

/*
 * Interfaccia con tre radio button per
 * la scelta della modalità con cui generare
 * le istanze.
 * 2 pulsanti: uno di OK e uno
 * di ANNULLA.
 */
public class MultiComponent {

          /**
           *
           */
          private static final long serialVersionUID = -6221276519506858623L;

          static String first = new String("Coverage                          ");
          static String second = new String("Number of instances        ");
          static String third = new String("Mixed mode                        ");
          static JFrame frame;
          static JSpinner tf;
          static JSpinner spin;
          static SpinnerNumberModel model;
          static JTextField tf2;
          static JTextField tfInst;
          public static String one;
          public static String two;
          public static String three;
          private static Document[] combinations;
          private double[] weights;
          private static boolean valid0;
          public static Vector FINAL_COMBINATIONS;

          public MultiComponent(Document[] combinations, double[] weights) {
                    MultiComponent.combinations = combinations;
                    this.weights = weights;
                    double sommapesi = 0;
                    for(int i = 0; i < weights.length; i++)
```

```
          sommapesi += weights[i];

for(int i = 0; i < weights.length; i++)
           weights[i] = weights[i]/sommapesi;

Sorter.mergeSort(MultiComponent.combinations, this.weights);
//creiamo il frame e gli passo il titolo.
frame = new JFrame("Selection Mode");

//settiamo lo sfondo del frame color bianco..
frame.setBackground(Color.white);

//..il layout del frame con il BorderLayout..
frame.setLayout(new BorderLayout());

//..le cordinate iniziali..
frame.setLocation(450, 350);

//..le dimensioni..
frame.setSize(3000, 300);

//..e l'icona sulla barra del titolo.
frame.setIconImage(new ImageIcon("icons\\taxi6.gif").getImage());

//creiamo 3 pannelli.
JPanel radioPanel = new JPanel();
JPanel labelPane = new JPanel();
JPanel buttonPanel = new JPanel();
buttonPanel.setLayout(new BorderLayout());

//creiamo un pulsante do Ok con un'icona che passiamo come parametro.
JButton ok = new JButton("OK", new ImageIcon("icons\\ok.png"));


/*aggiungiamo l'action listener con il metodo actionPerformed
 *  per l'azione del pulsante OK.
 */

System.out.println("STO PER AGGIUNGERE ACTIONLISTENER!!!!!!!!!!!!");
ok.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
                    //Qui dentro va il codice che viene eseguito
                    //ogni volta che OK viene premuto
                    System.out.println("<<<STO PER PREMERE IL PULSANET OK!!!!!!!!!!!!");
                    ok_actionPerformed(e);
                    System.out.println("<<<HO PREMUTO IL PULSANET OK!!!!!!!!!!!!");
          }
});

//creiamo il pulsante CANCEL con la sua icona
JButton canc = new JButton("Cancel", new ImageIcon("icons\\canc.png"));

/*aggiungiamo l'action listener con il metodo actionPerformed
 *  per l'azione del pulsante CANCEL.
 */
canc.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {

                    //Qui dentro va il codice che viene eseguito
                    //ogni volta che CANCEL viene premuto
                    canc_actionPerformed(e);
          }
});

/*aggiungiamo a buttonPanel i pulsanti
 * OK e Cancel rispettivamente a
 * sinistra e destra
 */
buttonPanel.add(ok, BorderLayout.WEST);
buttonPanel.add(canc, BorderLayout.EAST);

//coloriamo di bianco il buttonPanel.
buttonPanel.setBackground(Color.white);

//aggiungiamo il buttonPanel in fondo al frame in basso.
frame.add(buttonPanel, BorderLayout.SOUTH);

//creaiamo un panel imagePanel
//settiamogli il BorderLayout
//coloriamolo di bianco.
JPanel imagePanel = new JPanel();
imagePanel.setLayout(new BorderLayout());
imagePanel.setBackground(Color.white);

//creaiamo una label, chiamata img, con un icona.
JLabel img = new JLabel(new ImageIcon("icons\\logo3.gif"));

//settiamole il colore bianco.
img.setBackground(Color.white);
```

```
//aggiungo img a imagepanel.
imagePanel.add(img);

//e a sua volta aggiungo imagePanel nel centro del frame.
frame.add(imagePanel, BorderLayout.CENTER);

//radiopanel e labelpanel col background bianco.
radioPanel.setBackground(Color.white);
labelPane.setBackground(Color.white);

/*creazione di una textfield chiamat label.
 * Prende come parametro una stringa, cioè
 * la stringa da visualizzare nella textarea
 */
JTextField label = new JTextField("    Select  generation  mode:   ");

//creiamo un bordo color bianco intorno alla textfield,
//settiamo il font e colore.
label.setBorder(BorderFactory.createLineBorder(Color.white));
Font f = new Font("Mirror", Font.PLAIN, 16);
label.setFont(f);
Color c = new Color(16,96,156);
label.setForeground(c);
label.setBackground(Color.white);
label.setEnabled(true);
label.setEditable(false);

//aggiungiamo la txtfield a pannello labelPane.
labelPane.add(label);

//aggiungiamo labelPane in alto al frame.
frame.add(labelPane, BorderLayout.NORTH);

//set del layout di radioPanel.
radioPanel.setLayout(new GridLayout(0, 1));

//creiamo un gruppo di radiobutton group.
ButtonGroup group = new ButtonGroup();

//primo radiobutton first.
JRadioButton firstButton = new JRadioButton(first, false);

//pannello fp per firstButton.
JPanel fp = new JPanel();

//applichiamo borderlayout a fp e coloriamo di bianco lo sfondo.
fp.setLayout(new BorderLayout());
fp.setBackground(Color.white);

//label l, settiamole il testo con la stringa passata per parametro.
JLabel l = new JLabel();
l.setText("   % test units");



/*impostiamo lo spinnernumbermodel, che
 * prende un intero (0) come valore iniziale
 * dello spinner, un intero che identifica il
 * valore minimo che può assumere lo spinner(0),
 * un'altro intero invece per il massimo(100)
 * e l'ultimo intero che rappresenta la grandezza
 * dello step dello spinner(di quanto si incrementa o
 * si decrementa ogni volta che si clicca)
 */
model = new SpinnerNumberModel(0, 0, 100, 1);

//creiamo spinner tf.
tf = new JSpinner(model);
tf.setSize(0,0);
/*Un editor for il JSpinner il cui modello è un SpinnerNumberModel.
 * Il valore dell'editor è visualizzato con un JFormattedTextField
 * il cui formato è definito da un'istanza NumberFormatter
 */
JSpinner.NumberEditor editor = new JSpinner.NumberEditor(tf);
JFormattedTextField form = editor.getTextField();

//setta la lunghezza del form JFormattedTextField.
form.setColumns(2);

//setta il colore di background del form.
form.setBackground(SystemColor.window);

//setta il colore di foreground del form.
form.setForeground(SystemColor.windowText);

//setta i bordi del form a null.
form.setBorder(null);

//applicamo toString a form..
```

```
one = form.toString();

//setta l'editor dello spinner con il parametro editor(numberEditor).
tf.setEditor(editor);

//rendiamo indeditabile il testo dentro il field dello spinner.
editor.getTextField().setEditable(false);


//editor.getAccessibleContext().setAccessibleName("Session Size:");
//settiamo il bordo dello spinner
tf.setBorder(BorderFactory.createLoweredBevelBorder());


tf.setEnabled(false);


/*aggiungiamo al pannello fp: il firstbutton a sinistra,
 * la textfield a destra del pannello e
 * la label l nel centro.
 */
fp.add(firstButton, BorderLayout.WEST);
fp.add(tf, BorderLayout.EAST);
fp.add(l, BorderLayout.CENTER);

//inizialmente di default firstbutton non è selezionato.
firstButton.setSelected(false);

//impostiamo il nome del comando,
//passato come stringa, per il pulsante firstbutton
//e settiamo anche il colore delllo sfondo
//e il colore del font.
firstButton.setActionCommand(first);
firstButton.setBackground(Color.white);
firstButton.setForeground(c);

//aggiungiamo firstbutton a group
group.add(firstButton);


radioPanel.add(fp);

//creiamo anche il secondbutton..
JRadioButton secondButton = new JRadioButton(second, false);

/*creiamo anche JPanel fp2.
 *  settiamogli il BorderLayout e il
 *  lo sfondo color bianco.
 */
JPanel fp2 = new JPanel();
fp2.setLayout(new BorderLayout());
fp2.setBackground(Color.white);

//creazione di una label l2..
JLabel l2 = new JLabel();

//settiamo il testo della label l2, passandogli la
//stringa come parametro.
l2.setText("N° instances               ");

/*inizializziamo la nostra textfield tf2,
 * passandoglicome parametro un nuovo oggetto Document di
 * tipo customtextfield(con la lunghezza massima del field),
 * una stringa di testo uguale a null e un intero che
 * rappresenta la lunghezza della textfield(in colonne).
 */
tf2 = new JTextField(new CustomTextField(10), null, 4);

tf2.setText("0");
//tf2.setSize(tf.getWidth(), tf.getHeight());

//creiamo un bordo LoweredBevel alla textfield.
tf2.setBorder(BorderFactory.createLoweredBevelBorder());

//inizialmente di default è disabilitata.
tf2.setEnabled(false);

/*Aggiungiamo al panel fp2 il secondbutton
 * e lo posizionamo con layout BorderLayout
 * nella parte sinistra(west), mentre aggiungiamo
 * e posizioniamo la textfield tf2 nella parte destra.
 * Poi mettiamo anche la label l2 nel centro.
 */
fp2.add(secondButton, BorderLayout.WEST);
fp2.add(tf2, BorderLayout.EAST);
fp2.add(l2, BorderLayout.CENTER);

/*diamo il nome(stringa second)
 * al comando del radiobutton secondbutton e
 * impostiamogli il background a bianco e il
```

```
 * foreground con il colore c precedentemente creato.
 */
secondButton.setActionCommand(second);
secondButton.setBackground(Color.white);
secondButton.setForeground(c);

//inizialmente di default secondbutton è disabilitato.
secondButton.setSelected(false);

//aggiungiamo secondbutton a group.
group.add(secondButton);


radioPanel.add(fp2);



/*****************************----MIXED MODE----***********************************/


/*come per gli altri 2 radiobutton,
 * ci creiamo anche il terzo radiobutton thirdbutton e
 * lo impostiamo subito a disabilitato..
 * le impostazioni sono circa le stesse degli altri 2.
 * Mixed mode però avrà prima un jspinner per la
 * percentuale e subito accanto una textfield per il
 * numero di istanze.
 */
JRadioButton thirdButton = new JRadioButton(third, false);

/**creo pannello fpMixed dove verranno messi
 * gli altri 2 pannelli, quello dello spinner e
 * quello della textarea. Inoltre gli settiamo il
 * BorderLayout.
 */
JPanel fpMixed = new JPanel();
fpMixed.setLayout(new BorderLayout());

/*Percentual**/

/*creiamo un pannello per lo spinner
 * per la percentule dell istanze
 * di mixed mode.
 */
JPanel fpPerc = new JPanel();

//applichiamo borderlayout a fpPerc e coloriamo di bianco lo sfondo.
fpPerc.setLayout(new BorderLayout());
fpPerc.setBackground(Color.white);

//label lPerc, settiamole il testo con la stringa passata per parametro.
JLabel lPerc = new JLabel();
lPerc.setText("% test units");

/*impostiamo lo spinnernumbermodel, che
 * prende un intero (0) come valore iniziale
 * dello spinner, un intero che identifica il
 * valore minimo che può assumere lo spinner(0),
 * un'altro intero invece per il massimo(100)
 * e l'ultimo intero che rappresenta la grandezza
 * dello step dello spinner(di quanto si incrementa o
 * si decrementa ogni volta che si clicca)
 */
SpinnerNumberModel modelPerc = new SpinnerNumberModel(0, 0, 100, 1);

//creiamo lo spinner spin e passiamogli modelPerc come parametro.
spin = new JSpinner(modelPerc);

/*Un editor per il JSpinner il cui modello è un SpinnerNumberModel.
 * Il valore dell'editor è visualizzato con un JFormattedTextField
 * il cui formato è definito da un'istanza NumberFormatter
 */
JSpinner.NumberEditor editorPerc = new JSpinner.NumberEditor(spin);
JFormattedTextField formPerc = editorPerc.getTextField();

//setta la lunghezza del form.
formPerc.setColumns(2);

//setta il colore di background del form.
formPerc.setBackground(SystemColor.window);

//setta il colore di foreground del form.
formPerc.setForeground(SystemColor.windowText);

//setta i bordi del form a null.Niente bordo.
formPerc.setBorder(null);

three = formPerc.toString();

//setta l'editor della spinner con il parametro editorPerc(numberEditor).
```

```
spin.setEditor(editorPerc);

//rendiamo non editabile il testo dentro il field dello spinner.
editorPerc.getTextField().setEditable(false);

//impostiamo il bordo dello spinner.
spin.setBorder(BorderFactory.createLoweredBevelBorder());

//di default lo spin è disabilitato.
spin.setEnabled(false);

/*aggiungiamo al pannello fp il radiobutton
 * thirdbutton sulla sinistra del pannello,
 * la label lPerc("% istanze") nel centro e
 * spin a destra accanto alla label.
 */
fpPerc.add(thirdButton, BorderLayout.WEST);

fpPerc.add(lPerc, BorderLayout.CENTER);

fpPerc.add(spin, BorderLayout.EAST);
/*******fine perc********/

/*****numero instances****/

/*creiamo anche JPanel fpInst.
 *  settiamogli il BorderLayout e il
 *  lo sfondo color bianco.
 */
JPanel fpInst = new JPanel();
fpInst.setLayout(new BorderLayout());
fpInst.setBackground(Color.white);

//creazione di una label lInst..
JLabel lInst = new JLabel();

//settiamo il testo della label lInst, passandogli la
//stringa come parametro.
lInst.setText("   N° instances");

/*inizializziamo la nostra textfield tfInst,
 * passandoglicome parametro un nuovo oggetto Document di
 * tipo customtextfield(con la lunghezza massima del field),
 * una stringa di testo uguale a 0("0": lo spin
 * inizialmente avrà un valore di default 0) e un intero che
 * rappresenta la lunghezza della textfield(in colonne).
 */
tfInst = new JTextField(new CustomTextField(10), "0", 4);


//creiamo un bordo LoweredBevel alla textfield.
tfInst.setBorder(BorderFactory.createLoweredBevelBorder());

//inizialmente di default è disabilitata.
tfInst.setEnabled(false);


/*Aggiungiamo al panel fpInst la label lInst
 * e lo posizionamo con layout BorderLayout
 * nella parte centrale(center), mentre aggiungiamo
 * e posizioniamo la textfield tfInst nella parte destra.
 */

fpInst.add(lInst, BorderLayout.CENTER);

fpInst.add(tfInst, BorderLayout.EAST);

/***********fine instances**********/


/*Aggiungiamo il pannello fpPerc(quello
 * con lo spinner) al pannello
 * principale fpMixed(di mixed mode) nella parte
 * sinistra(con il
 * BorderLayout) e il pannello fpInst(quello con la
 * textfield) nella parte centrale
 */
fpMixed.add(fpPerc, BorderLayout.WEST);
fpMixed.add(fpInst, BorderLayout.CENTER);

thirdButton.setActionCommand(third);
thirdButton.setSelected(false);
thirdButton.setBackground(Color.white);
thirdButton.setForeground(c);
group.add(thirdButton);

radioPanel.add(fpMixed);
/************************   FINE MIXED MODE*************************************************/
```

```
                //creazione del radiolistener per gli ascoltatori
                //dei radiobutton
                RadioListener myListener = new RadioListener();


                /*Ad ogni radiobutton gli aggiungiamo un actionlistener
                 * al quale
                 * passo come parametro il radiolistener precedentemente
                 * creato.
                 */
                firstButton.addActionListener(myListener);
                //firstButton.addChangeListener(myListener);
                //firstButton.addItemListener(myListener);


                secondButton.addActionListener(myListener);
                //secondButton.addChangeListener(myListener);
                //secondButton.addItemListener(myListener);

                thirdButton.addActionListener(myListener);
                //thirdButton.addChangeListener(myListener);
                //thirdButton.addItemListener(myListener);


                //creiamo un bordo elegante(EtchedBorder), di colore grigio,
                //intorno ai 3 radiobutton, con un titoletto.
                Font font = new Font("Sanserif", Font.ROMAN_BASELINE, 14);
                radioPanel.setBorder(BorderFactory.createTitledBorder(BorderFactory
                                .createEtchedBorder(), "Selection",0, 0, font, Color.gray));

                //posizioniamo a sinistra del frame il nostro radiopanel
                frame.add(radioPanel, BorderLayout.WEST);

                frame.setVisible(true);
                frame.setEnabled(true);
                frame.pack();
        }


        //metodo contenente il codice da eseguire del pulsante cancel.
        private static void canc_actionPerformed(ActionEvent e){

                //quando si preme cancel il frame si chiude.
                frame.dispose();
        }



        private static void ok_actionPerformed(ActionEvent e){

                //CASO1   se lo spinner è abilitato, allora si esegue il metodo coverage.
                if(tf.isEnabled()){
                        coverage_actionPerformed(e);
                }

                //CASO2   se è abilitata la textfield...si entra
                if(tf2.isEnabled()){

                        boolean ok = true;
                        valid0 = true;
                        do{

                                ok = instances_actionPerformed(e);

                        }
                        while(!ok && valid0==false);

                        if(ok && valid0){

                                //SIP si = new SIP();

                                //si.send_doc_preprocessor(MultiComponent.FINAL_COMBINATIONS,
alberini_choise.get_stop());
                                frame.dispose();
                                //System.exit(1);
                        }
                }

                //CASO3   se è abilitata lo spin di mixed...si entra
                if(spin.isEnabled()){

                        boolean ok = true;
                        valid0 = true;
                        do{
                                ok = mixed_actionPerformed(e);

                        }
                        while(!ok && valid0==false);
```

```
                                if(ok && valid0){
                                //SIP si = new SIP();
                                //si.send_doc_preprocessor(alberini_choise.get_alberini(), alberini_choise.get_stop());
                                //si.send_doc_preprocessor(MultiComponent.FINAL_COMBINATIONS,
alberini_choise.get_stop());
                                frame.dispose();

                                }
                        }
                }



                /**Metodo che esegue il codice del modo coverage....
                 *
                 * @param e: l'evento ActionEvent
                 */
                private static void coverage_actionPerformed(ActionEvent e){

                        int centopercento = combinations.length;
                        int percentuale = ((Integer) tf.getValue()).intValue();
                        int totale;
                        if((centopercento * percentuale)%100 == 0) {
                                totale = (centopercento * percentuale)/100;
                        } else {
                                totale = (centopercento * percentuale)/100 + 1;
                        }

                        FINAL_COMBINATIONS = new Vector(totale);
                        int count = 0;
                        for(int i = combinations.length-1; i >= 0; i--){
                                if(count++ != totale)
                                        FINAL_COMBINATIONS.addElement(combinations[i]);
                                else break;
                        }

                        System.out.println(FINAL_COMBINATIONS);

                        //SIP si = new SIP();
                        //si.send_doc_preprocessor(alberini_choise.get_alberini(), alberini_choise.get_stop());
                        //si.send_doc_preprocessor(MultiComponent.FINAL_COMBINATIONS, alberini_choise.get_stop());

                        // crea l'array contenente la percentuale richiesta
                        frame.dispose();
                }
                private static boolean instances_actionPerformed(ActionEvent e){

                        boolean isOk = false;

                        if(tf2.getText().equals("")){
                                tf2.setText("0");
                        }
                        int totale = Integer.parseInt(tf2.getText());


                        if(totale > combinations.length){
                                new Alert("So many subtrees don't exist!");
                                isOk = false;
                                tf2.setText("0");
                                valid0 = false;
                                //CustomTextField.cache.append("0");
                        }
                        else{
                                FINAL_COMBINATIONS = new Vector(totale);
                                int count = 0;
                                for(int i = combinations.length-1; i >= 0; i--){
                                        if(count++ != totale)
                                                FINAL_COMBINATIONS.addElement(combinations[i]);
                                        else break;
                                }
                                isOk = true;

                        }

                        return isOk;
                }


                private static boolean mixed_actionPerformed(ActionEvent e){

                        int totale;

                        int centopercento = combinations.length;

                        int percentuale = ((Integer) spin.getValue()).intValue();


                        if((centopercento * percentuale)%100 == 0) {
```

```
                totale = (centopercento * percentuale)/100;
        } else {
                totale = (centopercento * percentuale)/100 + 1;
        }


        boolean isOk;

        if(tfInst.getText().equals("")){
                tfInst.setText("0");
        }

        int numInst = Integer.parseInt(tfInst.getText());


        if(numInst > totale){
                new Alert("So many subtrees don't exist!");
                tfInst.setText("0");
                valid0 = false;
                isOk = false;

        }



        else{

                FINAL_COMBINATIONS = new Vector(totale);
                int count = 0;
                for(int i = combinations.length-1; i >= 0; i--){
                        if(count++ != numInst)
                                FINAL_COMBINATIONS.addElement(combinations[i]);
                        else break;

                }
                isOk = true;
        }

return isOk;
}




/**Questo metodo RadioListener ascolta i radiobutton
 * e implementa ActionListener.
 *
 *
 */
 // Abbiamo bisogno di un solo tipo di evento.
class RadioListener implements ActionListener,

ChangeListener, // solo per curiosità.
ItemListener { //  solo per curiosità.

        public void actionPerformed(ActionEvent e) {

                /**
                 * Nucleo del metodo. Qui si fanno 3 if,
                 * uno per ogni radiobutton. Questi if
                 * quando si ha l'evento di selezione di
                 * un certo radiobutton abilitano solo il
                 * field del radiobutton selezionato mentre
                 * disabilitano gli altri 2.
                 * E' come una mutua esclusione.
                 */

                //Caso di selezione del primo radiobutton.
                if (e.getActionCommand() == first) {
                        //viene abilitato solo lo spinner del primo radiobutton.
                        tf.setEnabled(true);
                        tf2.setEnabled(false);
                        spin.setEnabled(false);
                        tfInst.setEnabled(false);


                }
                //Caso di selezione del secondo radiobutton.
                if(e.getActionCommand() == second){
                        //viene abilitato solo la textfield del secondo radiobutton.
                        tf2.setEnabled(true);
                        tf.setEnabled(false);
                        spin.setEnabled(false);
                        tfInst.setEnabled(false);

                }
                //Caso di selezione del terzo radiobutton.
                if(e.getActionCommand() == third){
                        //viene abilitato solo lo spinner e
```

```
                                        //la textfield del radiobutton mixed mode.
                                        //tutto il resto è lasciato disabilitato.
                                        spin.setEnabled(true);
                                        tfInst.setEnabled(true);
                                        tf.setEnabled(false);
                                        tf2.setEnabled(false);

                                }
                        }


                        //metodi non implementati dato che viene implementata ActionListener.
                        public void itemStateChanged(ItemEvent e) {
                                /**
                                 System.out.println("ItemEvent received: "
                                 + e.getItem()
                                 + " is now "
                                 + ((e.getStateChange() == ItemEvent.SELECTED) ? "selected."
                                 : "unselected"));*/
                        }

                        public void stateChanged(ChangeEvent e) {
                                //System.out.println("ChangeEvent received from: " + e.getSource());
                        }

                }

}
```

CustomTextField.java

```
package TSS.StrategySelection;

import javax.swing.text.*;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

public class CustomTextField extends PlainDocument {

          /*
           * Classe per la gestione delle textarea
           */
          private static final long serialVersionUID = 6269678590301794773L;

          //cache di tipo  StringBuffer per contenere i caratteri immessi.
          static StringBuffer cache = new StringBuffer();

          //intero per la lunghezza massima consentita.
          static int lunghezzaMax;

          /*
           * costruttore che prende come parametro un intero
           *
           */
          public CustomTextField(int lunghezzaMax) {
                    CustomTextField.lunghezzaMax = lunghezzaMax;

          }


          /**metodo insertString() prende come parametro
           * un intero: l'offset all'interno del field,
           * una stringa e un   AttributeSet: gli attributi
           * da associare ai caratteri inseriti.
           *
           * @Exception: può lanciare una BadLocationException
           */
          public void insertString(int off, String s, AttributeSet aset)
                              throws BadLocationException {
                    int len = getLength();
```

```
                /*controlliamo che la lunghezza della stringa
                 * immessa nel field non sia più
                 * lunga di lunghezzaMax.
                 * Altrimenti ritorna.
                 */
                if (len >= CustomTextField.lunghezzaMax) {
                        return;
                }

                //settaggio della dimensione della cache(StringBuffer) a 0.
                cache.setLength(0);

                char c;
                /*Questo ciclo non fa altro che scandire
                 * ogni carattere della textfield e
                 * controllare che sia un numero
                 * (o un punto o una virgola) e inserire il
                 * carattere nella variabile (StringBuffer)
                 * cache.
                 */
                for (int i = 0; i < s.length(); i++) {

                        c = s.charAt(i);

                        if(isNumber(c)){

                                cache.append(c);

                                if(cache.length() >= lunghezzaMax-len)
                                        break;
                        }
                        else break;


                        /*Dopo il ciclo for, se la cache è maggiore
                         * di 0 chiamiamo il metodo della superclasse insertString()
                         * che prende come parametri un intero
                         * (lo stesso passato come parametro(off: offset)
                         * nell'invocazione del metodo la cache
                         * (con applicato il toString) come String e
                         * il solito AttributeSet aset(attributi).
                         */
                if (cache.length() > 0)


                                super.insertString(off, cache.toString(), aset);

                }

        }

        /**
         * @param c: il carattere da analizzare.
         * @return un booleano: true se il char c
         * passato come parametro è un numero da 0 a 9
         * o un punto o una virgola.
         * false altrimenti.
         */
        public boolean isNumber(char c) {

                //il cuore del metodo è lo switch(c).
                switch (c) {

                case '0':
                        return true;
                case '1':
                        return true;
                case '2':
                        return true;
                case '3':
                        return true;
                case '4':
                        return true;
                case '5':
                        return true;
                case '6':
                        return true;
                case '7':
                        return true;
                case '8':
                        return true;
                case '9':
                        return true;
                default:
                        return false;
                }
        }
}
```

Sorter.java

```
package TSS.StrategySelection;

import org.w3c.dom.Document;
import org.w3c.dom.Node;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

public class Sorter {
          /**
           * mergeSort()
           *
           * Ordina l' array a in ordine crescente
           * applicando gli stessi spostamenti
           * effettuati su a anche su doc
           *
           * @param                    doc       -         un vettore contenente Document
           * @param                    a         -         un vettore di double
           * @modifies        a
           */
          public static void mergeSort(Document[] doc, double[] a) {

                    /*
                     * EFFETTI: ordinamento di a in senso crescente, a viene modificata in
                     * modo che i suoi elementi risultino ordinati in senso crescente.
                     * Applica anche su doc gli stessi spostamenti effettuati su a
                     */
                    if (a.length > 1) {
                              msort(doc, a, 0, a.length);
                    }
          }

          /**
           * msort()
           *
           * ordina la porzione di array compreso tra min e max usando l'algoritmo di
           * merge sort applicando gli stessi spostamenti effettuati su a anche su doc
           *
           * @param doc -
           *             l'arrai di documenti parallelo a quello dei pesi a
           * @param a   -
           *             l'array contenente i pesi da ordinare
           * @param min -
           *             estremo inferiore della porzione di array da ordinare
           * @param max -
           *             estremo superiore della porzione di array da ordinare
           */
          private static void msort(Document[] doc, double[] a, int min, int max) {
                    /*
                     * ordina la porzione di array compreso tra min e max usando l'algoritmo
                     * di merge sort applicando gli stessi spostamenti effettuati su a anche
                     * su doc
                     */
                    int mid;
                    if (min < max - 1) {
                              mid = (min + max) / 2;
                              msort(doc, a, min, mid);
                              msort(doc, a, mid, max);
                              merge(doc, a, min, mid, max);
                    }
          }

          /**
           * merge()
           *
           * Effettua il merge delle due porzioni ordinate di array comprese tra min e
           * mid e mid e max
           *
           * @param doc -
           *             un vettore di documenti il cui peso e' memorizzato in a
           * @param a -
           *             il vettore dei pesi da ordinare
           * @param min -
           *             estremo inferiore della prima porzione di array da unire
           * @param mid -
```

```
	 *            estremo inferiore della prima porzione di array da unire e
	 *            estremo superiore della seconda porzione di array da unire
	 * @param max -
	 *            estremo superiore della seconda porzione di array da unire
	 */
	private static void merge(Document[] doc, double[] a, int min, int mid,
			int max) {
		// fa il merge delle due porzioni ordinate di array comprese tra min e
		// mid e mid e max
		double[] r = new double[max - min];
		Node[] m = new Node[max - min];
		int i = 0;
		int j = min;
		int h = mid;
		while (i < r.length) {
			if ((j < mid) & (h < max))
				if (a[j] < a[h]) {
					r[i] = a[j];
					m[i] = doc[j];
					j++;
				} else {
					r[i] = a[h];
					m[i] = doc[h];
					h++;
				}
			else if (j < mid) {
				r[i] = a[j];
				m[i] = doc[j];
				j++;
			} else {
				r[i] = a[h];
				m[i] = doc[h];
				h++;
			}
			i++;
		}
		for (i = 0; i < r.length; i++) {
			a[min + i] = r[i];
			doc[min + i] = (Document) m[i];
		}
	}
}
```

VPinterfaces

DBInterface.java

```
package VPinterfaces;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

public class DBInterface{

	//Il Frame principale
	private static JFrame frame;
```

```
  //I bottoni del frame
  private JButton loadDB;
  private JButton loadTAXI;

  public DBInterface(){

            frame = new JFrame("Application");
            frame.setBackground(Color.white);
            frame.setLayout(new BorderLayout());
            frame.setLocation(450, 350);

            frame.setIconImage(new ImageIcon("icons\\taxi6.gif").getImage());

            //pannelli da posizionare nel frame
            JPanel northpanel = new JPanel();
            JPanel centralpanel = new JPanel();
            JPanel southpanel = new JPanel();

            loadDB = new JButton("Load Database", new ImageIcon("icons\\u.gif"));
            loadTAXI = new JButton("Load TAXI", new ImageIcon("icons\\taxi6.gif"));

            JLabel label = new JLabel("Select an application to start:");


            loadDB.addActionListener(new ActionListener() {
     public void actionPerformed(ActionEvent e) {

            loadDB_actionPerformed(e);
     }
  });

            loadTAXI.addActionListener(new ActionListener() {
     public void actionPerformed(ActionEvent e) {

            loadTAXI_actionPerformed(e);
     }
  });

            northpanel.add(label, BorderLayout.CENTER);
            northpanel.setBackground(Color.white);

            centralpanel.add(loadDB, BorderLayout.CENTER);
            centralpanel.add(loadTAXI, BorderLayout.CENTER);
            centralpanel.setBackground(Color.white);

            southpanel.setBackground(Color.white);


            frame.add(centralpanel, BorderLayout.CENTER);
            frame.add(northpanel, BorderLayout.NORTH);
            frame.add(southpanel, BorderLayout.SOUTH);

            frame.setVisible(true);
            frame.setEnabled(true);
            frame.pack();
  }



  //OMAR DEVI INSERIRE IL CODICE QUI IN QUESTO METODO E POI FA TUTTO LUI!!!!
  private static void loadDB_actionPerformed(ActionEvent e){

            //CODICE DA INSERIRE QUI. TUTTO CIO' CHE è QUI VIENE ESEGUITO QUANDO
            //NELLA FINESTRA DI DIALOGO SI SELEZIONA LOAD DATATBASE!!!

            System.out.println("database......LOADING");
            //frame.dispose();
  }

  private static void loadTAXI_actionPerformed(ActionEvent e){

            System.out.println("TAXI......LOADING");
            //CODICE DA INSERIRE QUI. TUTTO CIO' CHE è QUI VIENE ESEGUITO QUANDO
            //NELLA FINESTRA DI DIALOGO SI SELEZIONA LOAD TAXI!!!
            //frame.dispose();
  }

}
```

Interface1.java

```
package VPinterfaces;

import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

public class Values {

          public Values(){
          JFrame frame = new JFrame("Values");
          frame.setLocation(650, 500);
          frame.setIconImage(new ImageIcon("icons\\taxi6.gif")
            .getImage());

          JPanel panel = new JPanel();
          panel.setLayout(new BorderLayout());
          panel.setSize(frame.getSize());
          panel.setBackground(Color.WHITE);
          panel.setSize(frame.getSize());

          JPanel buttonPanel = new JPanel();
          JPanel stringpanel = new JPanel();
          buttonPanel.setBackground(Color.WHITE);
          stringpanel.setBackground(Color.WHITE);


          stringpanel.setLayout(new GridLayout());
          JLabel label1 = new JLabel("          ");
          JLabel label2 = new JLabel("INSERT VALUE");
          label1.setFont(new Font("Verdana", Font.BOLD, 14));
          stringpanel.add(label1);
          stringpanel.add(label2);


          JPanel pannelloPippo = new JPanel();
          pannelloPippo.setBackground(Color.WHITE);

          JTextField labelPippo = new JTextField("Pippo");
          labelPippo.setColumns(6);
          JTextField labeltxt = new JTextField("TXT");
          labeltxt.setColumns(6);

          pannelloPippo.add(labelPippo);
          pannelloPippo.add(labeltxt);
          pannelloPippo.setBorder(BorderFactory.createTitledBorder(BorderFactory
                              .createLoweredBevelBorder(), "MODIFY",0, 0, new Font("Verdana", Font.PLAIN, 10), new
Color(27, 115, 201)));


          JButton ok = new JButton("OK", new ImageIcon("icons\\ok.png"));

          buttonPanel.add(ok);

          panel.add(stringpanel, BorderLayout.NORTH);
          panel.add(pannelloPippo, BorderLayout.CENTER);
          panel.add(buttonPanel, BorderLayout.SOUTH);


          frame.add(panel);
          frame.pack();
          frame.setVisible(true);


          //se vuoi modificare la stringa pippo, fai
          //labelPippo.setText("NUOVO TESTO").


          ok.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                              ok_actionPerformed(e);
                    }
```

```
          });

          }
          public static void ok_actionPerformed(ActionEvent event) {

                    //QUELLO CHE FA IL PULSANTE OK QUI DENTRO......
                              }
}
```

Mode.java

```
package VPinterfaces;

import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

public class Mode {

          public Mode(){

                    JFrame frame = new JFrame("Mode");
                    frame.setIconImage(new ImageIcon("icons\\taxi6.gif")
                      .getImage());
                    Dimension dim = new Dimension(250, 150);
                    frame.setPreferredSize(dim);
                    frame.setLocation(650, 500);


                    JPanel panel = new JPanel();
                    panel.setBackground(Color.WHITE);
                    panel.setLayout(new BorderLayout());


                    JPanel labPanel = new JPanel();
                    labPanel.setBackground(Color.WHITE);
                    JLabel lab = new JLabel("SELECT MODE:");
                    labPanel.add(lab);

                    JPanel buttonPanel = new JPanel();
                    buttonPanel.setBackground(Color.WHITE);
                    JButton manual = new JButton("Manual", new ImageIcon("icons\\pencil.gif"));
                    JButton random = new JButton("Random", new ImageIcon("icons\\mac06.gif"));
                    buttonPanel.add(manual);
                    buttonPanel.add(random);


                    panel.add(labPanel, BorderLayout.NORTH);
                    panel.add(buttonPanel, BorderLayout.SOUTH);



                    frame.add(panel);
                    frame.pack();
                    frame.setVisible(true);


                    manual.addActionListener(new ActionListener() {
                              public void actionPerformed(ActionEvent e) {
                                        manual_actionPerformed(e);
                              }
                    });

                    random.addActionListener(new ActionListener() {
                              public void actionPerformed(ActionEvent e) {
                                        random_actionPerformed(e);
                              }
                    });

          }
```

```
	public static void manual_actionPerformed(ActionEvent event) {

		//QUELLO CHE FA IL PULSANTE manual QUI DENTRO......

	}

	public static void random_actionPerformed(ActionEvent event) {

		//QUELLO CHE FA IL PULSANTE random QUI DENTRO......

	}
}
```

Restrictions.java

```
package VPinterfaces;

import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

/**
 * @author		Romina Paradisi
 *			256283
 *			paradisi@cli.di.unipi.it
 *
 * @author		Maurizio Santoni
 *			244237
 *			santon@cli.di.unipi.it
 */

public class Restrictions {

	public Restrictions(){

		JFrame frame = new JFrame("Restrictions");
		frame.setIconImage(new ImageIcon("icons\\taxi6.gif")
		  .getImage());
		Dimension dim = new Dimension(300, 200);
		frame.setPreferredSize(dim);
		frame.setLocation(650, 500);


		JPanel panel = new JPanel();
		panel.setBackground(Color.WHITE);
		panel.setLayout(new BorderLayout());


		JLabel title = new JLabel("RESTRICTION INTERFACE");


		JPanel miopanel = new JPanel();
		JLabel empty = new JLabel("                          ");
		miopanel.setBackground(Color.WHITE);
		JTextField num = new JTextField("N°");
		num.setColumns(6);
		//num.setText("Inserisco nuove stringhe in num");
		JButton stop = new JButton("STOP", new ImageIcon("icons\\c.gif"));
		miopanel.add(num);
		miopanel.add(empty);
		miopanel.add(stop);



		JTextArea txt = new JTextArea("Restrictions........");


		txt.setBorder(BorderFactory.createTitledBorder(BorderFactory
				.createLoweredBevelBorder(), "Restrictions",0, 0,
				new Font("Verdana", Font.PLAIN, 12), new Color(27, 115, 201)));


		panel.add(title, BorderLayout.NORTH);
		panel.add(miopanel, BorderLayout.CENTER);
		panel.add(txt, BorderLayout.SOUTH);


		frame.add(panel);
		frame.pack();
		frame.setVisible(true);
```

```
                stop.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                stop_actionPerformed(e);
                        }
                });



        }
        public static void stop_actionPerformed(ActionEvent event) {

                //QUELLO CHE FA IL PULSANTE stop QUI DENTRO......
                System.out.println("Ho premuto STOP!!");
        }
}
```

TAXInstancesVisualizer

Apertura.java

```
package TAXInstancesVisualizer;

import java.io.*;
import javax.swing.*;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

public class Apertura{

        /**
         * E' semplicemente una funzione che ha come parametri f e a

         * @param f - il file da aprire.
         * @param a - la JTextArea dove visualizzare.
         * @return - Ritorna la nuova jTextArea da visualizzare
         */
        public JTextArea Apri(File f,JTextArea a){

                try{
                        BufferedReader fa=new BufferedReader(new FileReader(f));
                        int x=(int)fa.read();
                        String s5="";
                        while(x>=0){
                                char ch=(char)x;
                                s5+=""+ch;
                                /*Qua apriamo il file in lettura,e leggiamo
                                 *  il file carattere per carattere
                                 *  ("avremmo potuto farlo anche linea per linea)
                                 * Poichè l'fa.read() ritorna il codice UNICODE
                                 * del carattere letto lo convertiamo in carattere
                                 * che appendiamo ogni volta alla stringa s5...
                                 */
                                x=fa.read();
                        }
                        fa.close();

                        /*Appende la stringa s5 in coda alla jtextarea
                         * (che è vuota e quindi la mette all'inizio..).
                         */
                        a.append(s5);

                }//chiude il try

                catch(Exception e){
```

```
                        System.out.println(e);
                }
                return a;
        }//chiusura if
}
```

Editor.java

```
package TAXInstancesVisualizer;

import java.awt.*;
import java.awt.event.*;
import java.io.*;

import javax.swing.*;

import TAXInterface.graphics.Alert;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
 *                  244237
 *                  santon@cli.di.unipi.it
 */

//Editor estende JFrame
public class Editor extends JFrame{

        private static final long serialVersionUID = -5028314895963838552L;

        static File aperto;
        JScrollPane cj;
        static JFrame f;
        public static JTextArea a;
        JPanel panel;
        String path;

        /*costruttore che prende come parametro
         * una String.
         */
        public Editor(String path){

                this.path = path;
        }

        /**
         * Questo metodo(non ha parametri) rappresenta il vero e proprio
         * cuore del notepad per la visualizzazione delle
         * istanze.
         * Il nucleo di questo metodo è la textarea in cui viene
         * visualizzata l'istanza.
         * Questa classe richiama la classe Aperturain fondo al metodo.
         *
         */
        public void createPad(){


                f=new JFrame("Notepad to display instances");
                JToolBar toolbar = new JToolBar();

                JButton open = new JButton(new ImageIcon("icons\\folder_empty_open.png"));
                open.setText("Open an instance");
                open.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                open_actionPerformed(e);
                        }
                });
                toolbar.add(open);

                f.setIconImage(new ImageIcon("icons\\taxi6.gif").getImage());
                f.getContentPane().setPreferredSize(new Dimension(500, 400));
                f.setPreferredSize(new Dimension(500, 400));
                a=new JTextArea();
                cj=new JScrollPane(a);

                /*Inseriamo una JTextArea dentro un JScrollPane che permetta
                 di muovere a nostro piacimento la finestra.
                 */
```

```
                Container c=f.getContentPane();
                panel=new JPanel();
                /*Questo pannello viene inserito dentro il contenitore
                 * per disegnare il JScrollPane
                 */

                c.add(panel);
                panel.setLayout(new BorderLayout());
                panel.add(BorderLayout.NORTH,toolbar);
                panel.add(cj);
                f.setLocation(new Point(400, 250));
                f.pack();
                f.setVisible(true);
                File file = new File(path);
                Apertura aa=new Apertura();
                a.setEditable(false);
                a=aa.Apri(file,a);
        }
        /**Questo metodo aprendo una finestra di dialogo
         * permette all'utente di scegliere un file da
         * file system dalla directory Instances.
         *
         * @param event - L'evento scatenato dalla pressione del
         * bottone Open instances della toolbar(l'evento che era stato captato
         * dall'actionListener).
         */
        public static void open_actionPerformed(ActionEvent event) {


                FileDialog file = new FileDialog(f, "Open file",
                                FileDialog.LOAD);
                file.setDirectory("C:\\Documents and Settings\\Administrator\\workspace\\TAXI FINAL
FUSION\\Instances");
                //file.setFile("*.xml");// Set initial filename filter
                file.setVisible(true);
                String curFile, filename = "";
                String dir = "";
                File tester;
                if ((curFile = file.getFile()) != null) {
                        dir = file.getDirectory();
                        filename = dir + curFile;
                        tester = new File(filename);

                        while (tester.exists() == false) {
                                new Alert("WARNING: Instances not exists!");
                                file = new FileDialog(f, "Open file",
                                                FileDialog.LOAD);

                                //file.setFile("*.xml"); // Set initial filename filter
                                file.setVisible(true);

                                curFile = file.getFile();

                                dir = file.getDirectory();
                                filename = dir + curFile;
                                tester = new File(filename);

                        }

                        Apertura aa=new Apertura();
          a=aa.Apri(tester,a);
          f.setVisible(true);
                }


        }

}
```

Informazioni.java

```
package TAXInstancesVisualizer;

import javax.swing.*;

/**
 * @author          Romina Paradisi
 *                  256283
 *                  paradisi@cli.di.unipi.it
 *
 * @author          Maurizio Santoni
```

```
 *                  244237
 *                  santon@cli.di.unipi.it
 */

//estende JFrame

public class Informazioni extends JFrame {

          private static final long serialVersionUID = -2922857542556118283L;

          /**
           * Viene creata una stringa e viene passata come parametro
           * ad un JoptionPane nel quale verrà visualizzata.
           *
           */
          public void Guida() {
                    final String s1 = "Notepad is a simple text editor\nthat can be used to open\n instances… ";
                    JOptionPane.showMessageDialog(null, s1, "Guida in linea", 1);
          }

          /**
           * Viene creata una stringa e viene passata come parametro
           * ad un JoptionPane nel quale verrà visualizzata.
           *
           */
          public void Interrogativo() {
                    final String s1 = "\nJava Notepad\n\nVersion: 1.0.0\n";
                    JOptionPane.showMessageDialog(null, s1, "Information about product", 1);
          }

}
```

BIBLIOGRAFIA

[1] Antonia Bertolino, Jinghua Gao, Eda Marchetti, and Andrea Polini. "*Systematic Generation of XML Instances to Test Complex Software Applications*". Supported by TELCERT.

[2] Sito ufficiale di TELCERT. "*About TELCERT*" http://www.opengroup.org/telcert/about.htm.

[3] Tesi di laurea di Elisa Belfiori e Maria de los Milagros Cappelli. "*TAXI: Applicazione per la generazione automatica di istanze da XML Schema*" a.a 2004/2005.

[4] T.J. Ostrand and M.J. Balcer. "The category-partition method for specifying and generating functional tests".

[5] Antonia Bertolino, Jinghua Gao, Eda Marchetti. "*XML-based Automation of Partiton Testing*".

[6] XMLTestSuite. "*Extensible markup language (xml) conformance test suites*". http://www.w3.org/XML/Test/, 2005.

[7] NIST. "*Software diagnostic&conformance testing division: Web technologies*". http://xw2k.sdct.itl.nist.gov/brady/xml/index.asp/ , 2003.

[8] SQC. Xml Schema Quality Checker. http://www.alphaworks.ibm.com/tech/xmlsqc , 2001..

[9] W3CXMLValidator. http://www.w3.org/2001/03/webdata/xsv, 2001.

[10] XMLSpy. http://www.altova.com/products_ide.html , 2005.

[11] RISE. "Technical report".

[12] W3CXMLSchema: W3c xmlschema. http://www.w3.org/XML/Schema , 1998.

[13] RTTS: Rtts: Xml proven xml strategy. http://www.rttsweb.com/services/index.cfm, nd.

[14] XMLJudge: Xml judge. http://www.topologi.com/products/utilities/xmljudge.html, nd.

[15] EasyCheXML: easychexml: http://www.stonebroom.com/xmlcheck.htm, nd.

[16] Li, J.B., Miller, J. In: "*Testing the Semantics of W3C XML Schemas*". COMPSAC 2005, 2005.

[17] Toxgene: Toxgene: http://www.cs.toronto.edu/tox/toxgene, 2005.

[18] SunXMLInstanceGenerator: Sun xml instance generator: http://www.sun.com/software/xml/developers/instancegenerators/index.html, 2003.

[19] Bertolino A., Gao J., Marchetti E., Polini A.: "*Partition testing from xml schema*", submitted to IEEE software, 2006.

[20] Ostrand T., Balcer M.: "The category-partition method for specifying and generating functional tests. Communications of ACM" 31(6), 1988.

[21] DocumentObjectModel: Document object model: http://www.w3.org/DOM/, 2003.

[22] Sito ufficiale della SUN. http://java.sun.com

[23] Davide Marche. "Corso Java". Disponibile on line: http://www.ilsoftware.it/articoli.asp, 2006.

[24] Andrea Leoncini. "*Al lavoro con le swing*", nd.

[25] Claudio De Sio Cesari. "*Object Oriented && Java 5*", modulo 15 "*Interfacce grafiche (GUI) con AWT, Applet e Swing*", http://www.claudiodesio.com, nd.

[26] Pietro Castellucci. Guida Java – "*Contenitori e gestione dei layout*", http://java.html.it/lezione/765/contenitori-e-gestione-dei-layout, nd.

[27] DOM(italiano) Giuseppe Capodieci "*Parsing di documenti XML*"; disponibile online: http://www.latoserver.it/java/parsing-XML/, nd.

[28] XML e DOM: Massimo Autiero "*Costruire e pubblicare documenti XML con Visual Basic*" disponibile online: http://www.itportal.it/developer/vb/xml2/default.asp, nd.

[29] SAX Tutorials, Code Examples & Articles : http://www.troobloo.com/tech/sax.shtml

[30] DocumentObjectModel. Document object model : http://www.w3.org/DOM/ , 2005.

[31] Sito http://www.mokabyte.it.

[32] Claudio Garau. "*Parsing SAX e parsing DOM*", disponibile on line: http://www.claudiogarau.it/php/Parsing-SAX-e-parsing-DOM.php, 2006.

[33] Antonio Cisternino. "*MokaPackages; java.awt*", http://www.mokabyte.it/1999/11/index.htm, 1999.

[34] Tony Sintes. "*Events and Listeners*", disponibile online a: http://www.javaworld.com/javaqa/2000-08/01-qa/0801-events.html, 2000.

[35] Sito ufficiale di Eclipse: "*About Eclipse*", http://www.eclipse.org .

[36] Werner Randelshofer. "*How to do a fast spalshscreen in java*", disponibile ondine al sito: http://www.randelshofer.ch/oop/javasplash/javasplash.html, nd.

[37] The Java ™ Tutorials. "*How to use Tooltips*", http://java.sun.com/docs/books/tutorial/uiswing/components/tooltip.htm, nd.

[38] Donato Cappetta: "*Introduzione a Java*", disponibile ondine al sito http://www.lattecafe.net/java, nd.

[39] The Java ™ Tutorials. "*How to use Toolbars*", http://java.sun.com/docs/books/tutorial/uiswing/components/toolbar.htm, nd.

[40] Sito http://java.html.it/guida/lezione/766/menu, "*Menu*", nd.

[41] Michele Sciabarrà. "Corso Java", disponibile ondine al sito: http://www.corsojava.it/testi/java/basi/index.jsp, nd.

[42] Andrea Gini. "*Corso di swing, 2 Parte*", disponibile online al sito: http://www.mokabyte.it/2000/10/swing_02.htm, 2000.

[43] Philip Milne. "Creating TreeTables in Swing". Sun developer Network (SDN). http://java.sun.com/products/jfc/tsc/articles/treetable1, nd.

[44] Java ™ 2 Platform std. Edition v 1.4.2. http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/JToolBar.html.

[45] Andrea Gini. "Corso di swing, 5 Parte", disponibile ondine al sito: http://www.mokabyte.it/2001/01/swing5.htm, 2001.

[46] Java ™ 2 Platform std. Edition v 1.4.2. http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/JMenuBar.html.

[47] Roger S. Pressmann. "*Principi di ingegneria del software*", McGraw-Hill, 2004.

[48] Elliotte Rusty Harold & W. Scott Means, "*XML in a nutshell*", O'REILLY, 2002.

[49] Technology Enhanced Learning Conformance - European Requirements and Testing: http://www.opengroup.org/telcert .

[50] EJBSourceGenerator. Ejbsourcegenerator.: http://ejbgen.sourceforge.net/ , 2003.

[51] JavaXMLBindlets. JavaXMLbindlets. http://wwws.sun.com/software/xml/developers/instancegenerator/index.html , 2003.

[52] Andrea Gini. "*Corso di swing, 1 Parte*", disponibile online al sito: http://www.mokabyte.it/2000/09/swing_01.htm, 2000.

[53] W3C: http://www.w3c.it/w3cin7punti.html..

[54] Ultimo articolo scientifico redatto su TAXI.