

UNIVERSITÀ DEGLI STUDI DI UDINE
FACOLTÀ DI
SCIENZE MATEMATICHE FISICHE E NATURALI

Corso di Laurea in
INFORMATICA

Tesi di laurea:

Realizzazione di un editor HTML on-line.

Laureando: Mauro Lorenzutti

Relatore: Claudio Mirolo

Tutor Aziendale: Mario Savoia

ANNO ACCADEMICO 2002 - 2003

Indice

Prefazione.....	5
1 Studio preliminare del problema.....	7
1.1 Analisi preliminare.....	7
1.2 Requisiti del progetto.....	8
1.3 Scelta della tecnologia.....	9
1.4 Piano di lavoro.....	10
2 Ricerca di un editor open-source.....	11
2.1 Introduzione.....	11
2.2 Licenza.....	11
2.3 Ekit.....	11
2.4 Struttura del progetto originale.....	13
3 File di configurazione.....	14
3.1 Finalità.....	14
3.2 Parser XML.....	14
3.3 Parsing del file XML.....	15
4 Supporto al multilingua.....	17
4.1 Considerazioni preliminari.....	17
4.2 Modifiche alla struttura di Ekit.....	17
4.3 Gestione dei tag di lingua.....	19
4.4 Definizione dei tag di lingua.....	21
5 Permessi.....	23
5.1 Gestione delle autorizzazioni.....	23
5.2 Gruppo dell'utente.....	24
6 Menu dinamici.....	25
6.1 Introduzione al problema.....	25
6.2 Barra dei menu.....	25
6.3 Barra degli strumenti.....	27
6.4 Gestione delle lingue.....	28
6.5 Implementazione delle barre dei menu degli strumenti.....	29
7. Custom tags.....	31
7.1 Definizione dei tag proprietari.....	31
7.2 Implementazione.....	35
8 Tag non permessi.....	37
8.1 Descrizione del problema.....	37
8.2 Realizzazione.....	37
9 Conversione di tag.....	39
9.1 Descrizione del problema.....	39
9.2 Realizzazione.....	39
10 Firma dell'applet.....	41
10.1 Impostazioni di sicurezza.....	41
10.2 Operazioni di firma.....	41
10.3 Implementazione.....	43
11 Implementazione su un sito internet.....	44
11.1 Integrazione col CMS.....	44
11.2 Installazione in un sito internet.....	44
Conclusioni.....	45
Bibliografia.....	47

Prefazione

La presente tesi descrive il progetto da me sviluppato durante il periodo di tirocinio svolto presso la società informatica Sth s.r.l. di Udine. Alla base di tale periodo di tirocinio vi era la necessità da parte dell'azienda di sviluppare un editor HTML interfacciabile con un *Content Management System* (CMS) da loro sviluppato per l'aggiornamento di contenuti on-line su siti internet da parte di utenti non esperti. Caratteristiche basilari di tale editor erano la completa integrazione all'interno del software proprietario di cui sopra, una facile configurabilità, un'elevata scalabilità (in quanto il CMS stesso è in continua evoluzione) ed un'estrema facilità d'uso. Per quanto concerne la tecnologia da utilizzare si è scelto di realizzare l'editor all'interno di un applet, soluzione questa che avrebbe permesso un elevato grado di interfacciamento col CMS. Riguardo la necessità di configurabilità e scalabilità, tutta la definizione dei tag supportati dall'editor (standard e proprietari), quelli non consentiti o da sostituire, e i permessi degli utenti è stata inglobata all'interno di un file XML di facile interpretazione e modifica. Inoltre, poiché l'utente medio di tale editor non sarebbe stato a conoscenza del linguaggio HTML, era necessario fornire un'interfaccia del tipo *What You See Is What You Get* (wysiwyg), ovvero presentare a video come sarebbe stata la pagina una volta visualizzata mediante un browser e non il codice HTML realmente editato. Infine, altre caratteristiche emerse durante la realizzazione del progetto hanno portato alla firma dell'applet con la conseguente capacità di accedere alla clipboard ed al filesystem del computer dell'utente.

Poiché sviluppare dalle basi tale progetto avrebbe richiesto un dispendio di risorse non accettabile per l'azienda, si è scelto di modificare un applet editor HTML open source e customizzarlo secondo le necessità sopra individuate. Questa scelta ha portato dapprima alla ricerca di un progetto che rispondesse il più possibile ai requisiti e soprattutto che fosse ben strutturato e di facile modifica, quindi allo studio del progetto ed alla riorganizzazione dello stesso. Sono state successivamente apportate tutte le modifiche necessarie affinché venissero soddisfatti tutti i requisiti.

1 Studio preliminare del problema.

1.1 Analisi preliminare.

Ai giorni nostri, vista la vastissima diffusione di internet, il sito internet rappresenta con molta probabilità il modo più veloce per farsi conoscere, per farsi pubblicità. Negli ultimi anni c'è stata una vera e propria esplosione del numero di siti aziendali comparsi sul web. Lo scopo principale per un'azienda era di farsi pubblicità e quindi il sito internet assomigliava, e molte volte ne era la trasposizione digitale, a un depliant dove un visitatore poteva trovare foto, storia e breve descrizione dei prodotti venduti dall'azienda. Per un uso così limitato la tecnologia più diffusa di allora era sufficiente: un sito internet era un insieme di pagine HTML *statiche* collegate fra loro dai link. Col passare del tempo, ma soprattutto in conseguenza di un più esigente utilizzo di internet da parte di un numero sempre maggiore di persone, proprio quell'aggettivo, *statico*, è divenuto un collo di bottiglia per l'utilizzo di un sito internet per scopi commerciali (ma anche personali). Consideriamo ad esempio infatti il sito di un rivenditore di auto usate: all'inizio avrebbe offerto nient'altro che i dati utili per contattare il commerciante e, come detto, foto, storia e poco altro. Il passo successivo logico è di presentare al navigatore anche una specie di vetrina virtuale, il sito dovrebbe contenere cioè anche la descrizione e magari le foto di un campione rappresentativo dei veicoli in vendita. A questo punto il sito internet non è più un semplice depliant pubblicitario ma diventa il punto d'incontro tra cliente e rivenditore. Nasce però il problema di tenerlo aggiornato in quanto, sempre pensando all'esempio del commerciante d'auto usate, offrire auto già vendute e non quelle disponibili non può risultare utile al rivenditore. Come detto, però, il sito è statico, questo vuol dire che l'unico modo per aggiornare le pagine HTML è di "scaricarle", correggerle (non senza la conoscenza di alcune nozioni di base del linguaggio HTML) e quindi nuovamente caricarle sul server. Tutto ciò porta ad una mole di lavoro non indifferente se il sito deve essere continuamente aggiornato, ma soprattutto, se il nostro rivenditore d'auto non conosce l'HTML, comporterà una continua serie di richieste di aggiornamento alla software house che ha realizzato il sito internet e quindi un incremento esponenziale dei costi di mantenimento.

Per ovviare al problema dell'aggiornamento dei siti, e a molti altri problemi derivanti dalla limitatezza del linguaggio HTML, sono state sviluppate tutta una serie di tecnologie che permettono di creare delle pagine *dinamiche*. Una pagina dinamica è una pagina in grado di modificare i propri contenuti in base ad alcuni input esterni. Grazie a queste nuove tecnologie, dei veri e propri linguaggi di programmazione (vedi php, asp o jsp), è infatti possibile creare programmi in esecuzione sul server attivati dalla richiesta di una pagina del sito in maniera completamente trasparente al client. Alla richiesta di visualizzare una determinata pagina il server elabora tutti gli script contenuti nella stessa e fornisce in output una pagina di puro HTML. Tale pagina risulta quindi essere *dinamica* in quanto grazie agli script in essa contenuti è possibile modificare l'HTML presente o scriverne di altro, magari estraendolo da un file diverso o da un database. Come detto, mediante questi script è possibile interagire con il filesystem del server e anche con database remoti, da ciò si deduce che è possibile scrivere degli script che permettano di salvare delle informazioni direttamente sul server. Sfruttando questa risorsa si può capire come sia semplice realizzare delle pagine che permettano di

mettere on-line delle informazioni lavorando direttamente sul sito. Ritornando all'esempio del rivenditore, si possono realizzare degli script che gli permettano di tenere aggiornato il listino direttamente lavorando sul suo sito e senza la necessità di scaricare, modificare e ricaricare sul server le pagine già esistenti.

Nel mio periodo di tirocinio presso la società Sth s.r.l., la situazione era esattamente quella appena descritta. I siti internet sviluppati da questa società mettono a disposizione del committente una sezione riservata del sito dove è possibile aggiornare i contenuti direttamente on-line. Tali contenuti, infatti, sono salvati all'interno di una base di dati e quindi, come detto sopra, facilmente accessibili e modificabili mediante script. Le pagine del sito contengono degli script che di volta in volta, o più propriamente in base alle informazioni richieste dal visitatore mediante l'ultimo link cliccato, vanno a estrarre i dati direttamente dal database e ad inserirli nella pagina HTML di output. L'inserimento di tali dati da parte dell'amministratore del sito, che, come detto, è possibile in un'area a lui solo accessibile denominata *Back Office*, si avvale di alcune pagine contenenti delle *form* HTML nelle quali scrivere i dati e di script che si occupano del salvataggio. Grazie a questo sistema si è risolto il problema dell'aggiornamento dei contenuti on-line, tuttavia chi aggiorna le pagine del sito deve conoscere forzatamente l'HTML. Infatti la *textarea* messa a disposizione per modificare i contenuti del sito è paragonabile al *notepad* di Windows, ovvero a un editor privo di comandi già pronti per formattare il testo, e quindi se si vuole evidenziare una parola è necessario conoscere il codice HTML per il *bold*, peggio ancora se si vuole creare un link o inserire una foto. Il progetto alla base di questo tirocinio e di questa tesi vuole porre rimedio proprio a questo problema.

1.2 Requisiti del progetto

Lo scopo è di realizzare un editor HTML che metta a disposizione i più comuni comandi di formattazione del testo, ovvero un editor comunemente detto *wysiwyg*. Quindi è necessario realizzare un'interfaccia che fornisca oltre all'area di editing anche tutta una serie di pulsanti e menu attraverso i quali formattare il testo secondo il linguaggio HTML. Mediante questi pulsanti, l'amministratore del sito potrà creare delle pagine HTML senza conoscerne la sintassi, un po' come reso possibile dagli editor HTML *Frontpage* di Microsoft o *Dreamweaver* di Macromedia. Un requisito fondamentale di questo progetto è la creazione di menu e barra degli strumenti personalizzati: infatti gli utenti autorizzati ad aggiornare il sito potrebbero essere più d'uno e con diversi permessi. Pertanto, in base al gruppo di appartenenza dell'utente che effettua il login nel sito, sarà necessario fornire solamente i comandi per i quali ha i permessi. A taluni utenti, infatti, non sarà concesso ad esempio l'inserimento di *form* o la modifica del *font* utilizzato. Altro requisito di primaria importanza è la possibilità di inserire dei *tag proprietari*, dei tag cioè non definiti dallo standard HTML 4.0 ma necessari all'interno del CMS nel quale verrà inserito questo progetto. Presso la Sth s.r.l. è stato sviluppato, infatti, un CMS, per la gestione di tutti i contenuti del sito, che sfrutta dei tag definiti internamente e che deve essere possibile inserire mediante l'editor. Un ulteriore requisito è rappresentato dalla necessità di scrivere il testo in un numero non definito a priori di lingue. La maggior parte dei siti internet sono stati sviluppati in più lingue e pertanto dovrà essere possibile, all'interno dell'editor stesso, impostare quali lingue dovranno essere supportate e passare da una lingua all'altra senza la necessità di

cambiare pagina. Ovviamente dovrà anche essere messa a disposizione del CMS preesistente la possibilità di estrarre il testo dall'editor per effettuare il salvataggio in remoto. Inoltre dovrà essere possibile il caricamento di file preesistenti ed il salvataggio del testo HTML scritto nell'editor in locale, cioè direttamente sul computer dell'utente, al fine di permetterne l'elaborazione off-line ed il salvataggio dei dati in caso di perdita della connessione. Un ultimo servizio che dovrà mettere a disposizione questo editor è la conversione automatica di tag non standard HTML 4.0 (ad esempio il tag *font* deve essere sostituito dal tag *span*) e l'eliminazione di tag non permessi (si pensi ad un utente al quale non è concesso ridefinire il tipo di font che però carica un file preesistente che include questa caratteristica).

1.3 Scelta della tecnologia

Il primo passo da effettuare per lo svolgimento del progetto consiste nella scelta della tecnologia da impiegare. L'informatica di oggi, infatti, mette a disposizione un vasto insieme di strumenti da poter utilizzare. Sono state prese in esame più di una tecnologia (Client Server, Macromedia Flash, JWS Java Web Start, ActiveX e Java-JavaScript) e sono stati analizzati pro e contro di ciascuna.

La prima possibilità presa in esame è stata di sviluppare due programmi, uno da eseguire sul client, l'editor vero e proprio, ed uno da eseguire sul server per permettere il salvataggio del testo. Tale eventualità è stata però scartata per diversi motivi, tra i quali anche per una questione di sicurezza: per la realizzazione del programma server sarebbe stato necessario mettere a disposizione sul server una porta da sfruttare per la comunicazione col programma client, aumentando in questo modo, seppur di un fattore non elevato, la possibilità di attacchi dall'esterno. Inoltre sarebbe stato necessario obbligare tutti gli utenti autorizzati ad aggiornare il sito ad installare un programma sul loro computer. Quest'ultima restrizione avrebbe causato una drastica limitazione nell'utilizzo del *Back Office* in quanto non sarebbe più stato possibile aggiornare il sito da una postazione qualsiasi che avesse avuto accesso ad internet. Soprattutto per tale motivo si è deciso di seguire una strada diversa.

La seconda tecnologia considerata è stato Flash di Macromedia che, nella nuova versione MX, mette a disposizione degli sviluppatori tutta una serie di strumenti che vanno ben oltre alla sola creazione di contenuti multimediali. Sebbene meritasse un'analisi più approfondita, questa possibilità è stata accantonata principalmente per la natura proprietaria di tale prodotto. Infatti la realizzazione del progetto mediante Flash MX avrebbe richiesto l'acquisto di una licenza causando un incremento dei costi da sostenere, soprattutto considerando la possibilità di utilizzare strumenti open source. Un altro ostacolo è dovuto alla necessità da parte dei futuri utilizzatori dell'editor di scaricare il plug-in necessario per la corretta visualizzazione di progetti Flash mediante un qualsiasi browser. Inoltre tale soluzione risulta penalizzata dal mancato supporto da parte di Flash di caratteri accentati, apostrofi e di altri caratteri speciali.

Lo sviluppo dell'editor mediante ActiveX in realtà non è stato preso in considerazione per un'enorme limitazione derivante dal tipo di tecnologia. Sviluppando il progetto in ActiveX si sarebbe limitata la compatibilità con la sola piattaforma Win32, una restrizione inaccettabile.

Tale limitazione non è certo un problema utilizzando Java e quindi la scelta si è ridotta a sole due tecnologie: la recente JWS (Java Web Start) e l'applet in collaborazione con JavaScript. Java Web Start è una tecnologia sviluppata da Sun che permette l'esecuzione di applicazioni Java attraverso il Web e che possono essere eseguite mediante un qualsiasi browser o anche come applicazione a sé stante. Un'implementazione di questo tipo avrebbe diversi vantaggi, primo fra tutti proprio la portabilità derivante da Java. Inoltre utilizzando tale tecnologia sarebbe possibile effettuare anche il salvataggio in locale del testo scritto mediante l'editor per procedere alla sincronizzazione dei dati col server in un secondo tempo, possibilità questa difficilmente realizzabile mediante un applet per motivi di sicurezza. Un problema è però rappresentato dalla necessità di effettuare un consistente download da parte di ciascun utente per poter sfruttare tutte le caratteristiche offerte da JWS. Per questo motivo e per altri di seguito descritti si è optato per lo sviluppo mediante applet Java.

L'utilizzo di applet si porta dietro tutti i vantaggi di Java, a cominciare dalla portabilità. Il problema sopra descritto riguardante il lavoro *offline* è risolto mediante firma dell'applet stesso come spiegato nel capitolo 10. Anche per l'utilizzo di un applet è comunque necessario installare nel proprio browser un plug-in in quanto non tutte le funzionalità di Java vengono incluse in tutti i browser. Comunque, come verrà descritto più approfonditamente in seguito, il progetto è stato sviluppato cercando di mantenere la compatibilità anche con i "vecchi" plug-in senza costringere gli utilizzatori ad installare l'ultimo rilasciato da Sun (che peraltro comporta un download significativamente più ponderoso dei plug-in precedenti). Ciò che però più di tutto ha fatto propendere per applet Java e JavaScript è stata la possibilità di partire da progetti open source da plasmare in base alle esigenze interne dell'azienda senza la necessità di sviluppare il progetto da zero.

Per quanto concerne invece la definizione di tag proprietari, dei tag non permessi, delle conversioni di tag ed anche per la strutturazione di menu e barra degli strumenti si è decisa la stesura di un file XML che permettesse una chiara e facilmente modificabile configurazione.

1.4 Piano di lavoro

Il progetto del tirocinio sarà pertanto l'estensione di un editor HTML visuale basato su un applet per soddisfare le necessità particolari di Sth s.r.l. e dal bisogno di interagire con un CMS preesistente. Il primo passo sarà quindi la ricerca di un programma open source distribuito con una licenza che ne permetta l'impiego all'interno di software proprietario e che ne permetta la modifica. I passi successivi dovranno comprendere, come detto, il supporto allo sviluppo di testo multilingua, il supporto a tag proprietari, la conversione/eliminazione di tag particolari, l'interazione col filesystem locale e lo sviluppo di un sistema di configurazione esterna che non comporti la ricompilazione dell'applet qualora necessiti di modifiche alla configurazione.

2 Ricerca di un editor open-source.

2.1 Introduzione

Il mondo dell'open source è in costante crescita e i progetti sviluppati secondo tale dottrina sono un numero sempre maggiore e di ottima qualità. Effettuando una ricerca sul web si possono trovare numerosi progetti che propongono editor HTML visuali per l'inserimento di dati on-line. Un ottimo punto di riferimento è senz'altro rappresentato dal sito <http://www.bris.ac.uk/is/projects/cms> che raccoglie una lista di progetti open source e commerciali di cui sopra. Scorrendo tale lista, limitatamente ai prodotti open source, si possono subito analizzare due caratteristiche di fondamentale importanza di ciascun progetto: la tecnologia con cui è stato sviluppato e la licenza tramite la quale viene distribuito. Si può notare come molti dei progetti riportati limitino la loro compatibilità al binomio Win/IE e cioè a Windows e Internet Explorer. Altri progetti, invece, consentono un utilizzo mediante i soli browser Mozilla e Netscape. Di tale limitazione si è già discusso nella definizione dei requisiti del progetto decretandone la non idoneità. Pertanto tutti questi progetti non sono stati considerati nella ricerca ma si è ristretto il campo a quanti fossero sviluppati in tecnologia Java e Javascript. A questo punto si è presa in esame la licenza di distribuzione soffermandosi su due in particolare che ricoprivano la maggior parte dei programmi: la licenza GPL e la licenza LGPL.

2.2 Licenza

Le licenze considerate sono state la GPL e la LGPL, ovvero *Lesser GPL*. Dopo una attenta analisi di entrambe, si è constatato che la licenza LGPL rispondeva meglio alle nostre esigenze. Benché infatti entrambe le licenze permettano la distribuzione, l'utilizzo e la modifica del software, differiscono per quanto riguarda l'inclusione del software come modulo all'interno di un altro prodotto. La licenza GPL impone che i sorgenti del modulo vengano resi pubblici insieme a tutti i sorgenti del progetto completo, cioè anche del software che include il modulo. Tale licenza imporrebbe quindi che i sorgenti dell'applet modificato vengano resi pubblici congiuntamente ai sorgenti del CMS sviluppato internamente da Sth s.r.l., condizione questa non accettata dalla società. La licenza LGPL (GNU Lesser General Public License) invece permette l'utilizzo dell'applet come modulo del CMS obbligando la distribuzione dei sorgenti limitatamente all'applet stesso. Pertanto si è deciso l'utilizzo di un progetto open source distribuito sotto questa licenza. Per una trattazione più completa di questa licenza si rimanda al testo completo scaricabile dal sito <http://www.opensource.org>.

2.3 Ekit

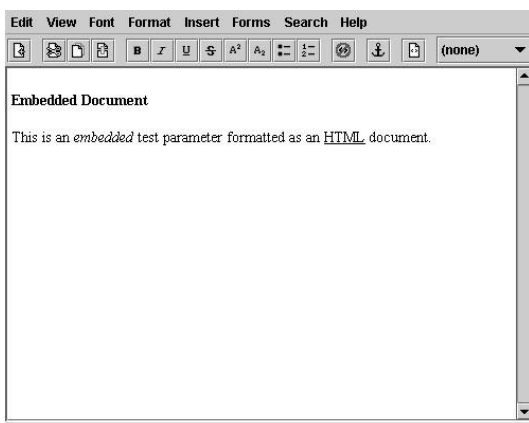
Il programma che risponde a tutti i requisiti di cui sopra è stato individuato nell'editor HTML Ekit v0.9f, un editor visuale sviluppato in tecnologia Java e distribuito sotto i termini della licenza LGPL. Questo progetto è stato sviluppato da Howard Kistler per hexidec e incorpora sia un applet sia un'applicazione stand-alone per l'editing di testo HTML. È possibile scaricare i sorgenti del progetto e valutarne una versione demo direttamente on-line dal sito internet <http://www.hexidec.com>. L'editor Ekit necessita per il suo

funzionamento in qualità di applet di un browser con installato il plug-in Java 1.3 in quanto sfrutta le librerie Swing per la visualizzazione del testo formattato in HTML. Tale plug-in è liberamente scaricabile dal sito ufficiale di Sun all'indirizzo <http://java.sun.com/products/plugin/1.3/plugin-install.html>.

L'applet mette già a disposizione un menu ed una barra degli strumenti con tutti i comandi necessari per gestire correttamente la formattazione del testo. Consente infatti la gestione del formato e dello stile del carattere, anche mediante il caricamento di un foglio di stile esterno (un file CSS), la possibilità di tagliare-copiare-incollare porzioni di testo, la possibilità di inserire dei link e delle tabelle, la possibilità di effettuare delle ricerche e delle sostituzioni all'interno del testo ma anche la gestione dell'annullamento e della ripetizione delle operazioni. Inoltre è messa a disposizione la possibilità di inserire nella pagine delle form HTML, funzionalità questa introdotta ultimamente e, come si può leggere dalla documentazione, non ancora sufficientemente testata e quindi piuttosto instabile ma comunque non necessaria per gli scopi di Sth s.r.l. Permette la visualizzazione del testo formattato come verrà visualizzato dal browser ed anche il codice sorgente, il testo HTML, in due diversi pannelli, entrambi modificabili e selezionabili mediante un semplice pulsante. Ciò che invece non permette di fare è di caricare delle immagini o dei documenti dal client e nemmeno di effettuare un salvataggio in locale, per i motivi di sicurezza dell'applet.

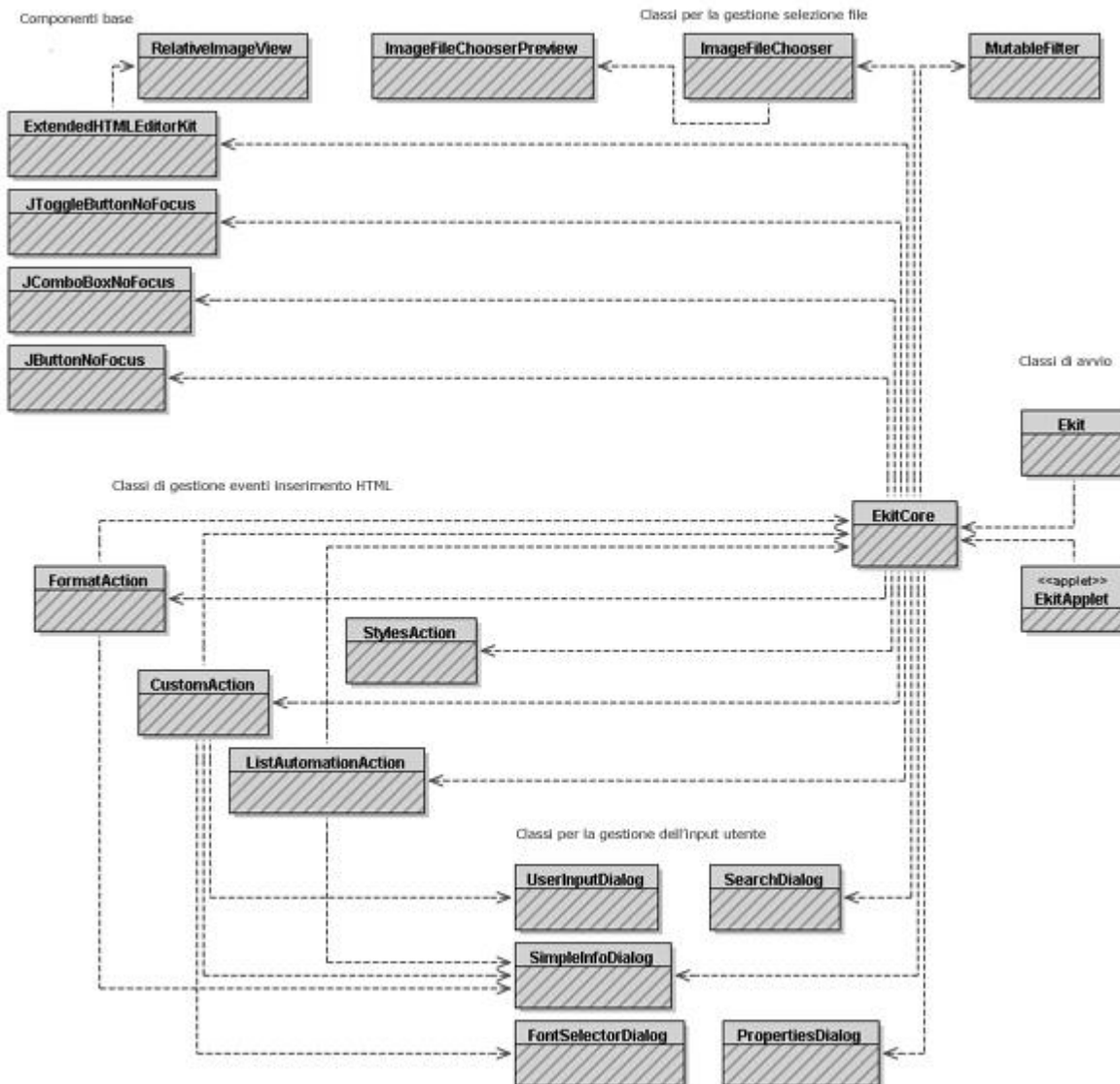
Per quanto riguarda la configurabilità dell'applet è possibile agire su alcune variabili passate come parametri all'interno del tag *applet* nella pagina HTML contenente la definizione dell'applet. È infatti possibile impostare il documento di stile da caricare, il testo HTML preimpostato da visualizzare all'avvio, la presenza o meno della barra degli strumenti e delle icone associate ai comandi e di visualizzare contemporaneamente o meno l'area di testo e il relativo codice sorgente, similmente a quanto permettono programmi commerciali quali *Dreamweaver* di Macromedia. Inoltre è anche possibile impostare la regione e la lingua dei menu, del testo alternativo dei pulsanti e di tutto il testo contenuto nei messaggi di errore o nelle maschere di inserimento dati. Le lingue supportate sono numerose, fra cui inglese, italiano, francese, tedesco e spagnolo, tutte definite in file di configurazione testuali.

Queste sono le funzionalità già presenti nell'applet, l'applicazione stand-alone contiene altre funzioni che non è possibile includere e quindi non considerate, e costituiscono il punto di partenza del mio progetto per la realizzazione dell'editor HMTL *wysiwyg* secondo i requisiti esposti in precedenza. Di seguito viene riportata una istantanea del progetto così com'era prima di essere modificato:



2.4 Struttura del progetto originale

Nella figura che segue è possibile osservare come era strutturato il progetto iniziale, ovvero la base di partenza del tirocinio in questione. Una osservazione: alcune classi di minore rilevanza, e comunque eliminate dal progetto finale, non sono rappresentate per permettere una maggiore leggibilità.



3 File di configurazione

3.1 Finalità

L'editor di testo da me modificato durante il tirocinio presso la società Sth s.r.l. deve poter essere utilizzato all'interno di un gran numero di siti internet, tutti diversi per contenuti, impostazione, committente e personale addetto all'aggiornamento. Inoltre questo progetto deve essere integrato all'interno di un CMS preesistente ma in continua evoluzione. Per questa serie di motivazioni, il passaggio da un sito ad un altro o da una versione del CMS ad un'altra comporterà tutta una serie di modifiche sulla struttura dell'applet. Come detto in precedenza, il menu potrà contenere elementi diversi in base alle esigenze del committente e in base ai permessi concessi ad un dato utente che lo utilizzerà, inoltre l'editor dovrà poter utilizzare nuovi tag proprietari qualora ne vengano introdotti. È inoltre necessaria la definizione di tutti i tag (o attributi di tag) non permessi, elenco questo che può variare per ogni sito oltre che per ogni utente, e la lista delle conversioni di tag. Risulta evidente come le varianti siano molte e frequenti e quindi non sia conveniente correggere il codice sorgente e ricompilare l'applet ogniqualvolta venga introdotta una modifica, sia per ragioni di tempo sia per evitare un'esplosione del numero di versioni diverse. La soluzione a questo problema è l'introduzione di un file di configurazione per ciascun sito dal quale l'applet andrà a reperire tutti i dati che gli serviranno. Tale file di configurazione è stato scritto nel linguaggio XML (eXtensible Markup Language) e quindi ha richiesto l'utilizzo di un parser XML per estrapolarne le informazioni contenute.

3.2 Parser XML

Il parser da me utilizzato è *Xerces-1.4.4* sviluppato da *Apache Software Foundation* e distribuito sotto i termini della licenza *Apache Software License*, liberamente scaricabile dal sito <http://www.apache.org>. Questa licenza impone a chiunque utilizza questo software di specificarne le origini e il nome e di distribuire una copia della licenza insieme al prodotto stesso. Questo parser si basa sulle API SAX (Simple API for XML), un'interfaccia standard per il parsing di testo XML basato sugli eventi. Ciò significa che al riconoscimento di un tag XML viene generato un evento che dovrà essere gestito dalla classe che gestisce il parsing del file. Il software *Xerces* implementa questa interfaccia e fornisce un motore di parsing perfettamente funzionante.

Oltre ad includere tutte le classi di questo progetto, è stato necessario includere anche le classi SAX (org.xml.sax e sottoclassi), questo per evitare l'installazione del nuovo plug-in Java 1.4. Infatti queste ultime classi sono state introdotte a partire dal JDK 1.4 e quindi non supportate dal plug-in 1.3, ovvero quello già richiesto dal progetto originale. Si è scelto di includere tali classi per evitare il passaggio al nuovo plug-in in quanto avrebbe comportato un download molto più consistente rispetto a quello finora necessario ed avrebbe costretto quanti avessero precedentemente installato la versione 1.3 ad un aggiornamento evitabile.

A questo punto il lavoro è consistito nella definizione della sintassi dell'XML per il file di configurazione e nella successiva definizione di una nuova classe che, importando le API SAX e il pacchetto *Xerces*, gestisse gli eventi generati durante il parsing del file.

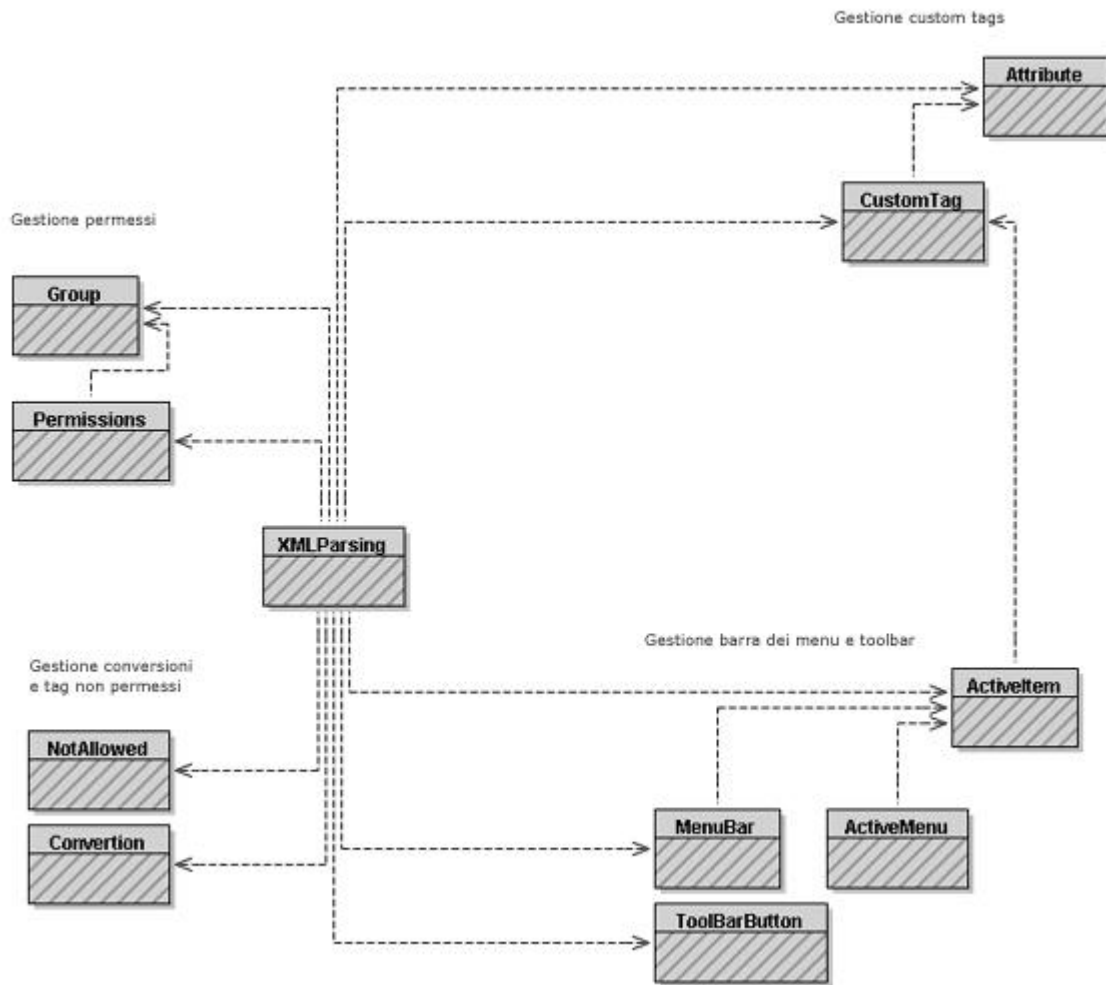
3.3 Parsing del file XML

Come verrà illustrato nel capitolo 7, i tag da me definiti all'interno del file per la formattazione dei dati sono molteplici e richiedono quindi un processo di parsing non elementare. Innanzitutto sono stati definiti i tag principali, ovvero i tag che avrebbero racchiuso le liste di elementi dello stesso tipo. Pertanto sono stati introdotti il tag *MenuBar*, che contiene una lista di menu attivabili nella barra degli strumenti, il tag *ToolBar*, che contiene la lista di tutti i pulsanti attivabili nella barra degli strumenti, il tag *Permissions*, che racchiude la definizione di tutti i permessi e dei relativi livelli, il tag *Conversions*, che racchiude la lista di tutte le conversioni di tag, ed infine il tag *NotAllowed*, che contiene la lista di tutti i tag o attributi non utilizzabili (e quindi da eliminare in fase di salvataggio o caricamento) all'interno dello specifico sito. Ciascuno di questi tag, come detto, contiene un elenco di elementi ai quali è stata associata una classe che ne raccoglie tutte le caratteristiche espresse mediante sottotag o attributi. Pertanto, come verrà meglio spiegato in seguito, al tag *Group* (sottotag di *Permissions*), che specifica un nuovo gruppo, è stata associata la classe "Group" che mantiene i dati relativi al nome del gruppo ed al relativo livello specificati all'interno del tag. Pertanto alla fine della procedura di parsing sarà stato popolato un vettore con un insieme di oggetti *Group* rappresentanti ciascuno un gruppo. Per tutti gli altri tag principali il procedimento è analogo, portando quindi alla creazione di cinque vettori accessibili mediante altrettanti metodi pubblici della classe che gestisce il parsing.

La classe *XMLParsing* si occupa appunto di effettuare il parsing di un file XML passato come parametro. Tale classe gestisce due eventi generati dal parser *Xerces*: uno quando viene incontrata l'apertura di un tag e uno quando viene incontrata la chiusura di un tag. In questo ambito viene operata la distinzione fra oggetti e attributi degli oggetti, ovvero all'evento riferito all'apertura di un tag vengono gestiti gli oggetti mentre alla chiusura di un tag vengono gestiti gli attributi. Alla generazione di un evento di apertura tag, infatti, viene effettuato il confronto per verificare se tale a tale tag è associata una classe. In caso positivo viene creato un nuovo oggetto del tipo specificato dal tag, accodato nel vettore relativo, e impostato come oggetto corrente. Inoltre, nel caso ne contenga, vengono gestiti gli attributi. Nel caso invece il confronto abbia esito negativo viene semplicemente ignorato l'evento. Alla generazione di un evento di chiusura di un tag vengono gestiti i sottotag dei tag di cui sopra, ovvero gli attributi degli oggetti. In questo caso viene controllato quale tag contiene quello in esame e vengono quindi settati i dati contenuti nell'oggetto attuale. Consideriamo infatti il tag *name* che è contenuto in quasi tutti i tag contenitori: all'occorrenza di tale tag viene verificato a quale dei tag contenitori fa riferimento e quindi viene richiamato il metodo dell'oggetto relativo necessario per impostarne il nome. In questo modo vengono gestiti tutti i tag e i relativi attributi del file di configurazione portando alla creazione di un insieme di oggetti facilmente manipolabili dal programma.

Al termine della procedura di parsing vengono anche gestiti i permessi. Infatti vengono eliminati dai vettori tutti quegli oggetti il cui gruppo ha un livello inferiore al gruppo dell'utente attualmente loggato e che ha istanziato l'applet. In questo modo il resto del programma viene sollevato dal compito di gestire i permessi in quanto è certo che tutti gli elementi caricati dal file di configurazione rispettano già il sistema di sicurezza.

Nell'immagine che segue è possibile vedere come è stata strutturata la procedura per il parsing del file XML di configurazione:



4 Supporto al multilingua

4.1 Considerazioni preliminari

Molti dei siti internet sviluppati dalla società presso la quale ho svolto il mio tirocinio supportano altre lingue oltre l'italiano e il loro numero varia in base alle esigenze del committente, in alcuni casi un sito può arrivare ad essere tradotto anche in sei lingue diverse. Il *Back Office* già esistente metteva a disposizione un numero di *textarea* per l'inserimento del testo pari al numero delle lingue da supportare. Tutto questo testo veniva poi salvato all'interno di un unico campo di testo in un database contenente le pagine del sito. Per poter distinguere il contenuto di ciascuna lingua il testo era contenuto all'interno di tag proprietari che ne specificavano la lingua.

Le possibili soluzioni alla gestione di più lingue sono due: inizializzare un applet per ogni lingua, delegando al programma server-side di salvataggio la gestione dei tag per la delimitazione delle lingue, oppure inizializzare un unico applet che metta a disposizione un tab per ciascuna lingua e che sia in grado di gestire i tag di lingua. Per evitare una crescita esponenziale della dimensione della pagina ed anche per una più immediata interazione tra utente e sistema ho scelto la seconda via. Pertanto si è reso necessario modificare il progetto per gestire un numero di tab pari al numero di lingue necessarie e non noto a priori.

4.2 Modifiche alla struttura di Ekit

Il progetto Ekit, come detto, metteva a disposizione sia un applet sia un'applicazione stand-alone. Per permettere entrambe queste due possibilità, venivano utilizzate due classi per l'avvio del programma, una per avviare l'applet (la classe *EkitApplet*) ed una per avviare l'applicazione (La classe *Ekit*). Entrambe queste classi non facevano altro che chiamare la classe di avvio vera e propria (*EkitCore*) passando degli opportuni parametri. Quest'ultima classe si occupava di creare la barra dei menu, la barra degli strumenti, il pannello contenente il testo e quello contenente il codice sorgente.

EkitApplet	
Init()	Ekitcore
- Avvia l'applet creando un nuovo EkitCore.	

Ekit	
main(String[])	Ekitcore
- Crea un nuovo oggetto Ekit	
new()	
- Crea un nuovo EkitCore	
new(String, String, String, URL, boolean, boolean,	

boolean, boolean, String, String, boolean)

- Crea un nuovo EkitCore specificando i parametri
passati come argomento all'avvio dell'applicazione

EkitCore

new(String, String, String, URL, boolean, boolean, ...

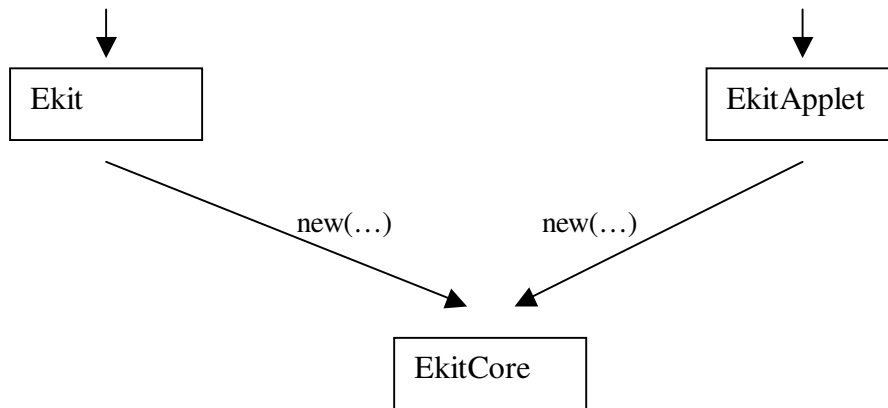
boolean, String, String, boolean)

- Attiva tutte le funzioni

...

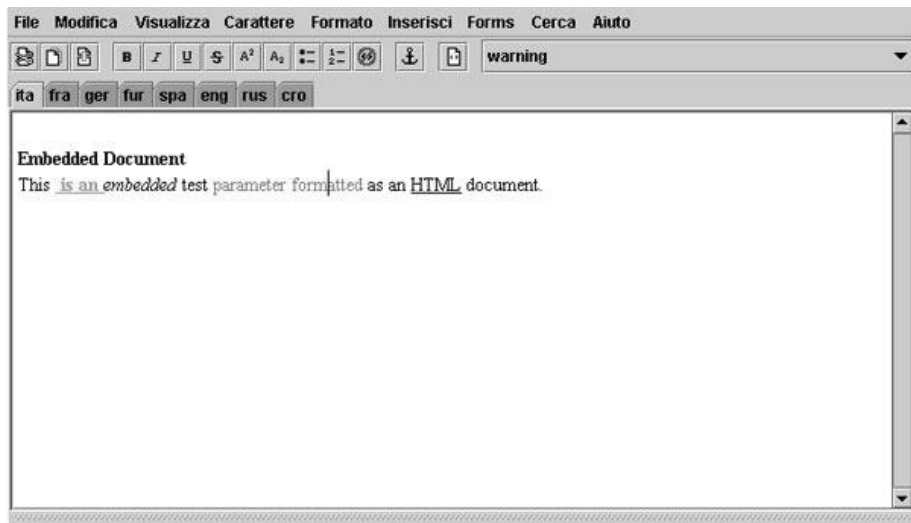
main(String[])

<applet><param...>...</applet>



Come si può vedere dagli schemi, l'unica funzione delle classi *Ekit* e *EkitApplet* è di inizializzare l'*EkitCore* in base a certi parametri passati da linea di comando o attraverso il tag *applet*. È poi compito della classe *EkitCore* creare le barre dei menu e degli strumenti. Secondo questo sistema, però, tutti i comandi attivabili da menu e toolbar erano legati in modo indissolubile all'*EkitCore* che, inoltre, si occupava di creare anche le due aree di editing, quella con il codice sorgente e quella di testo. Pertanto, volendo gestire più di una lingua sarebbe stato necessario inizializzare un numero di oggetti *EkitCore* pari al numero di lingue, questo per poter avere un'area di testo e una per il sorgente per ciascuna lingua. Tale operazione portava però ad un conflitto con la generazione della barra dei menu e degli strumenti, infatti non era possibile delegarla ad un unico *EkitCore* in quanto tutti gli eventi sarebbero stati riferiti sempre ad esso. Quindi ho scelto di rendere le due barre indipendenti dall'*EkitCore* portando la loro generazione all'interno della classe *EkitApplet* e perdendo, così facendo, la possibilità di avviare il progetto come applicazione stand-alone (che comunque non era richiesta). In questo modo è stato necessario innanzitutto spostare il codice di creazione delle due barre ed il codice relativo alla gestione degli eventi da loro generati al di fuori della classe *EkitCore* e quindi ridefinire tutti i metodi che venivano attivati dalla pressione di un pulsante o dalla selezione di un oggetto del menu affinché eseguissero le loro operazioni non più sull'*EkitCore* che le conteneva bensì su quello

selezionato al momento della generazione dell'evento. In questo modo si è resa possibile l'istanziamento di più di un *EkitCore* e quindi la contemporanea presenza di più aree di testo, tutte sottoforma di un tab distinto. Di seguito è riportata un'istantanea del progetto con il supporto al multilingua:



4.3 Gestione dei tag di lingua

Il progetto esistente metteva a disposizione all'interno della classe *EkitCore* due metodi pubblici per leggere (mediante una funzione JavaScript) l'HTML editato dall'utente. Queste due funzioni sono state rese private e sono state introdotte due nuove funzioni pubbliche all'interno della classe *EkitApplet* che richiamassero la rispettiva funzione da ciascun *EkitCore*, separassero il testo mediante i tag di lingua e fornissero così il testo già pronto per essere salvato nel database. In questo modo ho provveduto al salvataggio del contenuto, ciò che rimaneva da fare era permettere la modifica di un articolo preesistente o la creazione di uno nuovo. Per modificare del testo già esistente è necessario passarlo al momento dell'inizializzazione all'applet mediante un parametro del tag *applet*. Il valore di tale parametro viene impostato direttamente dal CMS che si occupa di passare il testo da modificare all'applet e di impostare quali lingue sono necessarie sfruttando un altro parametro. Qualora non sia stata attivata nessuna lingua, l'applet viene automaticamente inizializzato con la sola lingua italiana. Qualora il parametro che si occupa di passare il testo all'applet sia vuoto, l'applet presenta tante pagine vuote quante sono le lingue. Nel caso in cui, invece, sia stato impostato un certo numero di lingue e il testo reimpostato non sia vuoto, vengono istanziati i vari *EkitCore* (sempre uno per lingua) e il testo viene filtrato per dividerlo per lingua: vengono cercati i tag di apertura e quelli di chiusura corrispondenti a ciascuna lingua e il testo fra essi contenuto viene visualizzato dal corrispondente *EkitCore*. Se i tag di apertura e di chiusura di una lingua non dovessero essere trovati, allora l'area di testo di tale lingua sarà completamente bianca e pronta alla stesura di un nuovo testo. Per ultimo, se nel testo fornito all'applet non viene trovato nessun tag di lingua allora in automatico viene passato alla prima lingua attivata (solitamente l'italiano). Tutto questo procedimento per la gestione delle lingue si è reso necessario per evitare di limitare il numero di lingue attivabili e il momento della loro attivazione: è infatti possibile in ogni

momento aggiungere o togliere delle lingue. Di seguito viene riportato un esempio di passaggio di parametri all'applet:

```
<APPLET ...>
```

...

```
<PARAM NAME="DOCUMENT" VALUE='<we_l ita><P>Testo <b>Italiano</b></P></we_l><we_l  
eng><P>Testo <i>Inglese</i></P></we_l>'>
```

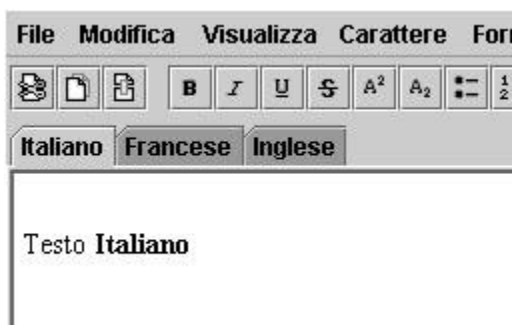
```
<PARAM NAME="LANGUAGES" VALUE="ita fra eng">
```

```
<PARAM NAME="DESCLANGUAGES" VALUE="Italiano Francese Inglese">
```

...

```
</APPLET>
```

Dall'esempio sopra riportato è possibile notare come è possibile impostare il testo al momento del caricamento dell'applet: mediante il parametro denominato "DOCUMENT". Il testo passato nell'esempio è scritto in due lingue delimitate dal tag *we_l* e riconoscibili dall'attributo specificato all'apertura del tag stesso: *ita* per l'italiano e *eng* per l'inglese. Il parametro seguente, ovvero "LANGUAGES", specifica quali lingue deve poter supportare l'applet, e cioè quanti tab deve istanziare. In questo esempio sono state attivate tre lingue: italiano (ita), francese (fra) e inglese (eng). Come si può vedere il testo sopra preimpostato mediante il parametro "DOCUMENT" definisce solamente due lingue. In questo caso il tab contenente il testo francese presenterà l'area di editing completamente bianca e pronta per la stesura di un nuovo testo. Infine, mediante il parametro "DESCLANGUAGES" vengono passate all'applet le descrizioni delle lingue attivate da visualizzare sulla linguetta del tab per selezionarle, come mostrato nell'immagine seguente:



Tutti i parametri sopra citati vengono impostati dinamicamente dal CMS. Infatti tale motore contiene al suo interno una struttura dati che descrive le lingue utilizzate e che può essere adoperata per passare tali valori all'applet. Per quanto riguarda il testo da caricare mediante il parametro "DOCUMENT" si possono distinguere due diverse operazioni concesse agli utenti: modificare una pagina preesistente o creare una nuova pagina. Nel primo caso il CMS provvederà ad impostare questo campo con il testo della pagina che l'utente vuole modificare mentre nel secondo caso il parametro verrà lasciato vuoto e quindi pronto all'inserimento di testo ex-novo.

Come sopra descritto, è stato introdotto un file XML per rendere l'applet il più configurabile e scalabile possibile. Poteva quindi risultare ovvia la definizione delle lingue da supportare all'interno di questo file. Ciò non era però conveniente in quanto le lingue vengono definite all'interno del CMS e quindi risulta più immediato e non ridondante il passaggio diretto di questa informazione dal CMS all'applet mediante un parametro.

4.4 Definizione dei tag di lingua

Ciò che invece viene definito all'interno del file di configurazione è la sintassi dei tag di configurazione. Questo perché, come già successo nel passaggio dalla prima versione del CMS alla seconda, i tag di lingua possono variare e quindi risulta conveniente poterli modificare senza dover ricompilare i sorgenti. Pertanto è stato introdotto il tag *LanguageTag*, all'interno del file XML, per la descrizione dei tag di lingua. Di seguito viene riportata la definizione dei tag di lingua utilizzati nell'esempio del paragrafo precedente:

```
<LanguageTag>
  <opener>we_l ###</opener>
  <closer>we_l</closer>
</LanguageTag>
```

La classe che effettua il parsing si occupa di mantenere una struttura dati che descriva la stringa del tag di apertura e quella di chiusura da passare alla procedura di gestione delle lingue. Tale procedura provvede ad analizzare la sintassi di tale tag, ricercando la presenza o meno del carattere speciale '#'. Nell'esempio sopra riportato, nel tag di apertura (*opener*) è presente la sottostringa '###' che deve essere sostituita con il codice identificativo di una lingua. Pertanto il tag di apertura per la gestione delle lingue sarà del tipo:

```
<we_l ita>    per l'italiano
<we_l fra>    per il francese
<we_l eng>    per l'inglese
```

e così via per ciascuna lingua. La definizione del tag di chiusura, invece, non contiene la sottostringa '###' pertanto sarà comune a ciascuna lingua e sarà della forma `</we_l>`.

Nella precedente versione del CMS i tag di lingua utilizzati erano diversi e di seguito è riportata la loro definizione:

```
<LanguageTag>
  <opener>tm_###</opener>
  <closer>tm_###</closer>
</LanguageTag>
```

In questo caso sia l'apertura sia la chiusura sono personalizzate per ciascuna lingua, pertanto i tag di lingua saranno della forma:

<tm_ita> e </tm_ita> apertura e chiusura per l'italiano

<tm_fra> e </tm_fra> apertura e chiusura per il francese

<tm_eng> e </tm_eng> apertura e chiusura per l'inglese

senza limitazioni al numero di lingue.

In questo modo è possibile specificare un testo fisso (quali *we_l* o *tm_*) e definire la presenza (e quindi la posizione) o meno dell'identificativo di lingua per i tag di gestione delle lingue direttamente nel file di configurazione.

5 Permessi

5.1 Gestione delle autorizzazioni

I siti internet sviluppati da Sth s.r.l. hanno la peculiarità, come detto, di permettere l'aggiornamento dei contenuti direttamente on-line, previo un login. Al momento della creazione di un nuovo utente (e non la registrazione pubblica per la quale il gruppo è prefissato e solitamente non consente l'accesso all'area di aggiornamento e quindi all'editor in questione) gli viene assegnata l'appartenenza ad un determinato gruppo, al quale fanno riferimento un certo numero di operazioni consentite. Pertanto quando un utente effettua il login nel sito deve poter eseguire solamente le operazioni concesse al gruppo a cui appartiene. Poiché il sistema di aggiornamento dati precedente è stato sostituito dal progetto in questione, è stato necessario realizzare una gestione dei gruppi e dei permessi anche all'interno dell'applet. Pertanto è stato inserito nel file di configurazione un tag contenitore di definizione dei gruppi. All'interno di questo tag è possibile elencare una serie di gruppi con il relativo livello nella gerarchia (a livello maggiore corrisponde maggiore libertà). Di seguito è possibile vedere un esempio di questa soluzione:

```
<Permissions>
  <Group>
    <name>admin</name>
    <level>9</level>
  </Group>
  <Group>
    <name>poweruser</name>
    <level>7</level>
  </Group>
  <Group>
    <name>owner</name>
    <level>5</level>
  </Group>
  <Group>
    <name>guest</name>
    <level>0</level>
  </Group>
</Permissions>
```

La classe che effettua il parsing del file di configurazione gestisce l'evento generato all'apertura del tag *Permissions* inizializzando un vettore che conterrà i vari gruppi. Tale classe regola l'evento dell'apertura del tag *Group* istanziando un nuovo oggetto *Group* ed aggiungendolo al vettore precedentemente creato

contenente tutte le istanze di *Group*. I tag *name* e *level* vengono gestiti passando i dati contenuti all'ultimo gruppo istanziato. Al termine del parsing la classe che lo ha effettuato mette a disposizione un metodo pubblico per richiedere il vettore contenente tutti i gruppi. In questo modo è possibile definire un insieme di gruppi diversi per ogni sito.

5.2 Gruppo dell'utente

A questo punto è necessario implementare un sistema per informare l'editor del gruppo di appartenenza di chi lo sta utilizzando. Il passaggio di tale informazione avviene mediante un parametro aggiuntivo alla definizione dell'applet, come riportato di seguito:

```
<APPLET ...>
    ...
    <PARAM NAME="GROUP" VALUE="%q_lg_group%">
    ...
</APPLET>
```

Il valore di tale parametro è preimpostato al codice `%q_lg_group%` che verrà automaticamente sostituito dal CMS, che mantiene al suo interno una struttura dati che memorizza l'utente attualmente nel sistema e tutti i suoi dati, al momento dell'apertura della pagina contenente l'applet. Così facendo, quando viene caricato l'applet, questi è a conoscenza del gruppo di appartenenza dell'utente che lo sta utilizzando e quindi gli può assegnare un certo livello in base a quanto definito nel file di configurazione. Come verrà spiegato nel capitolo 6, tale livello servirà per definire i comandi per i quali l'utente è autorizzato mediante confronto col livello di appartenenza di un determinato comando (infatti ad ogni voce di menu e toolbar è associato anche un gruppo minimo per la visualizzazione).

6 Menu dinamici

6.1 Introduzione al problema

La gestione dei permessi precedentemente descritta viene utilizzata nella creazione della barra dei menu e della barra degli strumenti. Infatti, come più volte ripetuto, è necessario differenziare gli utenti al fine di fornir loro solamente un certo insieme di comandi. Pertanto, poiché ad ogni utente è associato un determinato gruppo, anche ad ogni elemento delle barre è stato associato un gruppo minimo per la visualizzazione, ovvero per visualizzare un certo comando l'utente deve appartenere al gruppo di quel comando o ad un gruppo di livello superiore. Come appare chiaro da quanto appena detto, è stato necessario definire all'interno del file di configurazione tutti gli elementi dei menu e della toolbar ed assegnare loro il gruppo di appartenenza. Di conseguenza sono stati creati due nuovi tag, uno per la barra dei menu e uno per la barra degli strumenti.

6.2 Barra dei menu

Iniziamo con l'analizzare la definizione dei menu. All'interno del relativo tag (*MenuBar*) è possibile indicare una lista di menu specificando per ciascuno il nome (utilizzato per la gestione interna e non modificabile), il gruppo ed una ulteriore proprietà per indicare se tale menu è attivo o meno, a prescindere dal gruppo. Inoltre, come attributo del tag *Menu* è possibile specificare il testo da visualizzare nelle diverse lingue. Quello che segue è un esempio di definizione:

```
<MenuBar>
  <Menu it="File" en="File">
    <name>File</name>
    <active>>false</active>
    <permission>admin</permission>
  </Menu>
  <Menu it="Modifica" en="Edit">
    <name>Edit</name>
    <active>>true</active>
    <permission>poweruser</permission>
  </Menu>
</MenuBar>
```

Come si può evincere dall'esempio, sono stati creati due menu a discesa nella barra dei menu, il primo chiamato *File* a cui è stato assegnato il gruppo *admin* ma che comunque non verrà attivato in base a quanto impostato dal parametro *active*. Il secondo, denominato *Edit*, appartenente al gruppo *poweruser* e attivo. Il parametro *active* è stato introdotto perché può essere utile, passando da un sito ad un altro, disabilitare

completamente il menu senza doverne cancellare la definizione e quindi rendendo più laboriosa una futura reintroduzione. In questo modo invece è sufficiente modificare un unico parametro.

Una strutturazione simile è stata mantenuta per la definizione degli elementi di ogni singolo menu. Infatti per ciascuno dei menu elencati è possibile specificare un insieme di elementi da visualizzare, ognuno dei quali avrà un nome, con relativo testo da visualizzare in più lingue, uno stato (attivato/disattivato) e l'appartenenza ad un gruppo. Di seguito viene mostrata la definizione di alcuni comandi del menu *Edit*:

```
<Menu>
```

```
  <name>Edit</name>
```

```
  <active>true</active>
```

```
  <permission>poweruser</permission>
```

```
  <Item it="Taglia" en="Cut">
```

```
    <name>Cut</name>
```

```
    <active>true</active>
```

```
    <permission>poweruser</permission>
```

```
  </Item>
```

```
  <Item it="Copia" en="Copy">
```

```
    <name>Copy</name>
```

```
    <active>true</active>
```

```
    <permission>poweruser</permission>
```

```
  </Item>
```

```
  <Item it="Incolla" en="Paste">
```

```
    <name>Paste</name>
```

```
    <active>true</active>
```

```
    <permission>poweruser</permission>
```

```
  </Item>
```

```
  <Item>
```

```
    <name>separator</name>
```

```
    <active>true</active>
```

```
    <permission>poweruser</permission>
```

```
  </Item>
```

```
  <Item it="Annulla" en="Undo">
```

```
    <name>Undo</name>
```

```
    <active>true</active>
```

```
    <permission>poweruser</permission>
```

```
  </Item>
```

```
  <Item it="Ripeti" en="Redo">
```

```

        <name>Redo</name>
        <active>true</active>
        <permission>poweruser</permission>
    </Item>
</Menu>

```

Da questo estratto è possibile notare la definizione all'interno del menu *Edit* dei classici comandi *taglia-copia-incolla* e *annulla-ripeti*. Come per il singolo menu, anche per tutti i suoi elementi è quindi possibile specificare gli stessi parametri con le stesse finalità. Si noti inoltre la possibilità di definire i separatori come qualsiasi altro elemento di menu. L'utilità di definire il gruppo anche per il separatore deriva dalla necessità di dividere insiemi di elementi diversi presenti solamente per certi gruppi, altrimenti si potrebbero avere due separatori contigui se l'utente non ha i permessi necessari per visualizzare i comandi compresi.

Questa rappresentazione altamente modulare dei menu si è resa necessaria per rendere la generazione della barra dei menu il più configurabile possibile. Semplicemente spostando o rinominando delle parti di codice XML è possibile modificare l'ordine dei menu o dei loro elementi e cambiarne il testo visualizzato o aggiungere la traduzione per una nuova lingua. Inoltre risulta immediata anche l'assegnazione di un menu o di un elemento ad un determinato gruppo o la più immediata attivazione/disattivazione.

Per quanto riguarda la visualizzazione delle icone a fianco degli elementi di menu principali, tale opzione era già messa a disposizione e quindi è stata mantenuta inalterata. L'attivazione o disattivazione di tale opzione è ottenibile modificando il parametro del tag *applet* sotto riportato:

```

<APPLET ...>
    ...
    <PARAM NAME="MENUICONS" VALUE="true">
    ...
</APPLET>

```

Impostando il valore a "true" vengono visualizzate le icone, impostandolo a "false" invece non vengono caricate.

6.3 Barra degli strumenti

In egual modo è stata impostata la definizione della barra degli strumenti: per ogni pulsante è possibile definire il gruppo e lo stato attivo/non attivo. Di seguito viene riportato un esempio:

```

<ToolBar>
    <Button it="Nuovo" en="New">
        <name>New</name>

```

```

        <active>true</active>
        <permission>poweruser</permission>
    </Button>
    <Button it="Apri" en="Open">
        <name>Open</name>
        <active>>false</active>
        <permission>admin</permission>
    </Button>
</ToolBar>

```

Come si vede in questo esempio la barra degli strumenti è composta da due soli pulsanti: il pulsante *New*, che è attivo e appartiene al gruppo *poweruser*, ed il pulsante *Open*, disattivato. Anche per la barra degli strumenti è possibile definire dei separatori e cambiare l'ordine dei pulsanti. Le traduzioni introdotte verranno visualizzate come testo alternativo qualora l'utente si posizioni sopra un pulsante e attenda qualche istante, al fine di informare sulla funzione del pulsante stesso.

Anche la possibilità di nascondere tale barra era già prevista nel progetto originale, così come la gestione delle icone, e quindi si è preferito mantenere quella impostazione. Pertanto la possibilità di disabilitare la barra degli strumenti è offerta da un parametro del tag *applet* e non dal file di configurazione:

```

<APPLET ...>
    ...
    <PARAM NAME="TOOLBAR" VALUE="true">
    ...
</APPLET>

```

Secondo questa impostazione la barra degli strumenti viene visualizzata, mentre se il valore è "false" non viene attivata.

6.4 Gestione delle lingue

Nella definizione di menu e pulsanti della toolbar, come sopra visto, è possibile specificare il testo da visualizzare in più lingue. Il numero e il tipo di lingue impostabili non è prefissato e quindi è possibile aggiungere nuove traduzioni in qualsiasi momento. Negli esempi sopra riportati erano presenti le traduzioni in due sole lingue: *it*, ovvero italiano, e *en*, inglese. Per impostare la lingua da visualizzare è possibile agire su due parametri del tag *applet*:

```

<APPLET ...>
    ...

```

```
<PARAM NAME="LANGCODE" VALUE="it">
<PARAM NAME="LANGCOUNTRY" VALUE="IT">
...
</APPLET>
```

Il parametro che definisce la lingua in cui devono essere visualizzati i testi della barra dei menu e della toolbar è il primo e può essere impostato direttamente dal CMS. Il secondo parametro, oltre al primo, viene utilizzato dal progetto originale in quanto era già prevista la possibilità di modificare la lingua di menu, toolbar e messaggi d'errore. Per i messaggi di errore il sistema di traduzioni è rimasto invariato mentre, come si è visto, per il resto è stato rifatto, principalmente per evitare ridondanza. Infatti in seguito verrà spiegato come introdurre dei tag proprietari e quindi dei comandi di menu non previsti: in questo caso le traduzioni sarebbero state forzatamente nel file di configurazione da me sviluppato e non in quelli già presenti, portando ad una difformità nel trattamento degli elementi non accettabile e che avrebbe causato una complessità molto maggiore nella gestione. Inoltre chi dovrà personalizzare l'applet in relazione al singolo sito internet potrà, in questo modo, agire unicamente sul file XML, svincolandosi dai problemi di formattazione dei file testuali preesistenti.

Qualora la lingua selezionata tramite il parametro di cui sopra non sia disponibile, le due barre vengono automaticamente visualizzate in lingua inglese.

6.5 Implementazione delle barre dei menu degli strumenti

Nel progetto originale le barre dei menu e degli strumenti venivano impostate *manualmente*: entrambe contenevano sempre gli stessi elementi e nella stessa posizione. Un requisito di questo progetto, invece, è la personalizzazione di tali barre senza la necessità di ricompilare i sorgenti. Se non ci fosse stata quest'ultima limitazione si sarebbe potuta mantenere l'implementazione preesistente, semplicemente riscrivendo il codice per ogni diverso sito al fine di presentare delle barre diverse. Ovviamente non è stata seguita questa strada (per tutti i motivi già citati) ma si è scelto di rendere "dinamica" la costruzione di tali barre. Dopo il parsing del file di configurazione viene messa a disposizione una struttura dati contenente tutti gli elementi da inserire nelle due barre già ordinati. Prima di rendere accessibili tali dati, però, vengono filtrati per eliminare gli elementi non attivi (quelli cioè per cui il parametro *active* è impostato a *false*) e quelli per cui l'utente non ha i permessi necessari. Sfruttando tale struttura dati, è stato realizzato un ciclo che analizza e inserisce gli elementi presenti in tale struttura nella corretta posizione nelle due barre fino ad esaurimento degli stessi. Il parametro *name* viene utilizzato a questo punto per l'istanziamento del comando associato all'elemento di menu. Infatti ho scelto di sfruttare quanto già messo a disposizione dalle API Java in fatto di gestione dell'HTML, come i comandi per l'inserimento di tag standard o la gestione di annullamento/ripetizione delle operazioni. Pertanto, qualora il parametro *name* corrisponda ad un identificatore associato ad uno di questi comandi predefiniti, viene sfruttato tale comando. Gli eventi associati agli altri comandi, quelli non messi a disposizione dalle classi del JDK 1.3, viene effettuata in loco e gestita da codice appositamente scritto (si

pensi ai tag proprietari). Mediante questo procedimento piuttosto laborioso si è riusciti a rendere la definizione delle barre il più scalabili e personalizzabili possibile, fornendo la possibilità di inserire nuovi comandi, con l'unica accortezza di mantenere inalterato il parametro *name* per i comandi che sfruttano le API.

7. Custom tags

7.1 Definizione dei tag proprietari

Come più volte detto, l'editor in questione dovrà supportare dei tag proprietari del CMS che lo contiene. Poiché il CMS stesso è in costante evoluzione, i tag da supportare non sono noti a priori e quindi è necessario includere nel file di configurazione anche la loro definizione. Dato che l'unico modo consentito per inserire un tag è mediante i menu o la toolbar, si è scelto di integrare all'interno della caratterizzazione di questi la stessa definizione dei tag proprietari. Di seguito viene riportata la definizione del tag utilizzato per creare un link ad un documento remoto:

```
<Item it="Documento remoto" en="Remote document">
  <name>customTag</name>
  <active>true</active>
  <permission>admin</permission>
  <CustomTag>
    <tag>we_doc</tag>
    <styled>{ font-weight: bold; text-decoration: underline; color: #ff0000 }</styled>
    <closeTag>true</closeTag>
    <Attribute en="Name" it="Nome">
      <name>name</name>
      <type>text</type>
    </Attribute>
  </CustomTag>
</Item>
```

Questa descrizione viene inclusa all'interno del menu prescelto come se si trattasse di un qualsiasi altro elemento di menu. Ciò che differenzia questo tag da tutti gli altri è il nome, comune a tutti quelli proprietari, *customTag*. Quando la funzione che genera la barra dei menu trova un elemento con questo nome, riconosce la presenza di un tag proprietario e lo gestisce diversamente dagli altri. Quando l'utente seleziona questo elemento dal menu, l'applet gestisce l'evento in base all'oggetto *CustomTag*, oggetto creato dal parser secondo i parametri specificati all'interno di *CustomTag*. L'attributo *tag* in questo contesto precisa il tag da inserire, *closeTag* è invece un attributo booleano per indicare se è di tipo aperto/chiuso o meno. L'attributo *styled* descrive, qualora il tag sia di tipo aperto/chiuso e racchiuda del testo da visualizzare, lo stile del testo. Come detto, questo comando permette l'inserimento di un link ad un documento remoto. È quindi necessario fornire all'utente una maschera di ingresso per inserire il nome del documento precedentemente caricato e al quale è stato associato appunto un nome che il CMS è in grado di gestire. Per tale motivo è stata aggiunta la definizione di un attributo da richiedere:

```
<Attribute en="Name" it="Nome">
    <name>name</name>
    <type>text</type>
</Attribute>
```

Di seguito è possibile vedere come si presenta la finestra di dialogo all'utente:



Questo attributo è di tipo testuale e chiamato *name*. L'applet si occuperà di presentare all'utente una finestra di dialogo dove richiede l'inserimento di un *name*, visualizzando la richiesta nella lingua prescelta, fornendo una textarea. Quando l'utente avrà fornito questa informazione verrà inserito nel punto specificato dal cursore il tag che segue:

```
<we_doc name="nome inserito">testo precedentemente selezionato</we_doc>
```

A questo punto è stato inserito correttamente il link.

Oltre all'inserimento di attributi di tipo testuale è anche possibile richiedere la scelta fra un predefinito insieme di opzioni (combobox) o la possibilità di fornire la scelta fra due sole opzioni (checkbox). Di seguito viene riportato un esempio di checkbox:

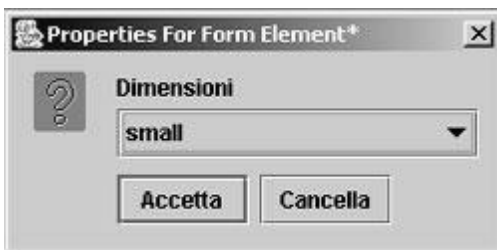
```
<Attribute en="Visible" it="Visibile">
    <name>visible</name>
    <type>bool</type>
</Attribute>
```



In questo caso viene visualizzata una checkbox che permette di decidere se il parametro *visible* è vero o falso. Quella che segue è invece la definizione di una combobox:

```
<Attribute en="Size" it="Dimensioni">
    <name>size</name>
    <type>combo</type>
    <v>small</v>
    <v>medium</v>
    <v>large</v>
</Attribute>
```

In questo caso viene visualizzata una comboBox che permette di scegliere fra tre diversi valori di cui *small* è quello di default.



Ovviamente è possibile combinare le tre possibilità sopra descritte per ottenere delle finestre di dialogo più complesse. Di seguito viene riportato un esempio che implementa tutti i tre tipi di input possibili nella stessa finestra e un'immagine del risultato:

```
<Attribute en="Name" it="Nome">
    <name>name</name>
    <type>text</type>
</Attribute>
<Attribute en="Visible" it="Visibile">
    <name>visible</name>
    <type>bool</type>
</Attribute>
<Attribute en="Mode" it="Modalità">
    <name>mode</name>
    <type>combo</type>
    <v>popup</v>
    <v>nopopup</v>
```

</Attribute>

<Attribute en="Size" it="Dimensioni">

<name>size</name>

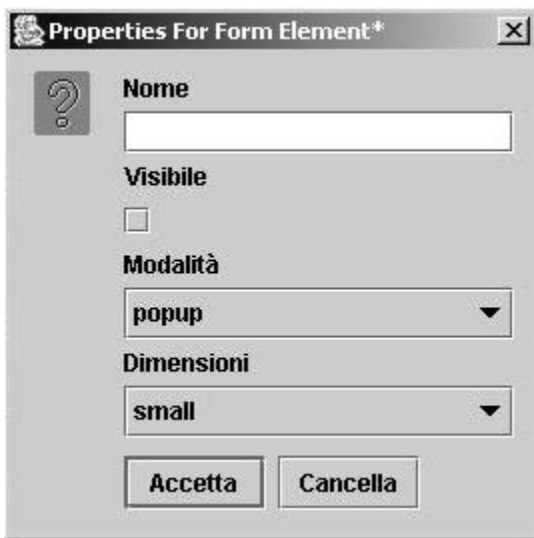
<type>combo</type>

<v>small</v>

<v>medium</v>

<v>large</v>

</Attribute>



Inoltre è anche possibile specificare una pagina esterna da aprire. Infatti l'inserimento di un'immagine remota avviene visualizzando una pagina nella quale l'utente può cliccare sull'immagine da inserire. Una funzione Javascript si occuperà di passare i parametri da questa pagina all'applet. Anche questa pagina deve quindi essere specificata all'interno della definizione del tag proprietario che ne fa uso, come riportato di seguito:

<Item>

<name>customTag</name>

<active>>true</active>

<permission>admin</permission>

<CustomTag>

<tag>we_img</tag>

<closeTag>>false</closeTag>

<page>immagini.php</page>

</CustomTag>

</Item>

Alla selezione di questo comando, quindi, l'applet aprirà la pagina *immagini.php*. Questa pagina visualizza tutte le immagini remote fra cui l'utente deve scegliere quale inserire mediante un click sulla stessa. Una funzione Javascript fornirà all'applet i nomi dei parametri del tag ed i rispettivi valori. Questa pagina è solamente un esempio in quanto è possibile crearne di nuove specificando i nomi dei parametri all'applet mediante una funzione Javascript. Se necessario è anche possibile aggiungere degli attributi nel modo visto sopra da richiedere all'utente mediante una finestra di dialogo.

7.2 Implementazione

Per distinguere i tag proprietari da quelli standard, o meglio i comandi non supportati da Java da quelli invece già forniti, viene utilizzato il parametro *name* che è comune a tutti i tag di questo tipo: "customTag". Qualora la routine di generazione delle due barre incontri un tag di questo tipo istanzia un generico elemento di menu. A tale elemento viene associato un identificatore univoco al fine di distinguere alla generazione di un evento di menu (ovvero l'evento generato dalla selezione di un elemento delle due barre da parte dell'utente) quale sia la fonte ed agire di conseguenza. In questo caso viene isolato il tag proprietario a cui fa riferimento l'evento e vengono svolte le operazioni definite nel file di configurazione (apertura di pagine esterne o presentazione di finestre di dialogo o entrambe le possibilità in sequenza). Quando l'utente ha fornito i feedback richiesti viene effettuato l'inserimento vero e proprio del codice HTML all'interno del testo. Tale inserimento, però, non è affatto banale ed ha portato alla scoperta di un problema proprio della classe `HTMLToolkit` del package `javax.swing.text.html` del JDK 1.3. Per gestire la visualizzazione di testo formattato in HTML e l'editing nello stesso pannello è stato utilizzato l'`HTMLToolkit`, ovvero una classe Java predefinita per l'editing visuale di HTML. Tale classe permette la definizione degli stili dei tag mediante l'impostazione di uno `StyleSheet`. Inoltre permette l'inserimento di testo HTML mediante il metodo `insertHTML` che richiede come parametri il documento contenente l'HTML, la posizione, la stringa HTML, due parametri per la gestione della struttura dati interna che rappresenta un albero a partire dai tag e il tag che contiene il testo. Per l'inserimento di tag proprietari si è reso necessario estendere la classe che gestisce la visualizzazione dell'HTML (`HTMLDocument`) per poter registrare i nuovi tag estendendo la sottoclasse `HTMLDocument.HTMLReader` che gestisce gli eventi interni per la presentazione di testo formattato HTML. In questa classe sono anche stati definiti gli stili del testo racchiuso dai tag proprietari secondo quanto specificato nel file di configurazione mediante il tag `styled`. A questo punto il documento riconosce e può gestire i tag proprietari. Il problema è sorto però al momento dell'inserimento nel documento di tali tag sfruttando il metodo sopra descritto `insertHTML`. Infatti passando come parametro un tag sconosciuto (`HTML.UnknownTag`), invece di un tag standard (`HTML.Tag`), il tag stesso ed il testo racchiuso al suo interno veniva ignorato completamente all'atto della visualizzazione e quindi cancellato. Tale problema ha richiesto un'approfondita ricerca nei forum della Sun (www.java.sun.com) e su numerosi altri forum in rete. Il risultato è stata la scoperta che questo è un bug delle API Java in quanto non è possibile inserire degli `HTML.UnknownTag` (ovvero come i tag proprietari vengono modellati in Java) sfruttando il metodo `insertHTML`. La soluzione è consistita nell'includere la stringa di testo HTML all'interno di un tag

standard, tag, questo, che le classi preposte al trattamento dell'HTML fornite dal JDK 1.3 sono in grado di gestire. La spiegazione di tale mancanza sembra essere che il parser HTML non sa in che posizione inserire i tag sconosciuti all'interno della sua struttura dati. Pertanto ho sovrascritto il metodo insertHTML affinché racchiudesse la stringa contenente il testo HTML all'interno del tag bold ().

In questo modo l'HTMLEditorKit è in grado di fornire una visualizzazione formattata secondo lo stile definito nel file di configurazione anche dei tag proprietari, mantenendo la filosofia dell'editor wysiwyg e quindi fornendo un immediato output all'utente su come verrà visualizzata la pagina che sta editando.

Un'ultima nota riguardo la definizione dei tag proprietari: con il sistema di definizione implementato sarebbe possibile definire come tag proprietari anche i tag standard mantenendo comunque le stesse funzionalità. Infatti, potendo definire lo stile del testo contenuto all'interno del tag proprietario, si possono implementare tutti i tag di formattazione del testo. Nel progetto è stata mantenuta l'impostazione iniziale che prevedeva lo sfruttamento delle classi predefinite in quanto dover ridefinire tutti i tag sarebbe stato un lavoro inutile. Comunque tale possibilità potrebbe essere sfruttata per implementare dei tag che in un futuro potrebbero essere introdotti nello standard senza per questo dover modificare il codice sorgente.

8 Tag non permessi

8.1 Descrizione del problema

Una delle peculiarità dei siti internet sviluppati da Sth s.r.l è rappresentata dalla pulizia e dalla conformità grafica mantenuta all'interno di tutte le pagine. Poiché questo progetto mette a disposizione un gran numero di comandi per la formattazione del testo, la possibilità di modificare direttamente il codice HTML, nonché la possibilità di importare delle pagine editate in precedenza e magari con altri strumenti, risulta evidente come questa fedeltà e purezza grafica possa venire in breve tempo stravolta. Per evitare tutto ciò, oltre alla possibilità di non fornire all'utente determinati comandi sfruttando la gestione dei permessi, è stata creata una procedura che all'atto del salvataggio del documento in remoto (ovvero della richiesta da parte del CMS del testo contenuto nell'applet) elimina tutti i tag e gli attributi non permessi. In questo modo è possibile evitare che l'utente inserisca nelle sue pagine una form semplicemente elencando tutti i tag relativi, quali i tag *form*, *button*, *input* ..., oppure è possibile permettere l'utilizzo del tag *font* per specificare magari il colore del testo, ma non consentire la definizione della grandezza del carattere impostando l'attributo *size* come non permesso.

8.2 Realizzazione

Come per tutte le altre opzioni personalizzabili viste sinora, anche per la definizione dei tag e degli attributi da filtrare è stato definito un tag all'interno del file di configurazione che ne contenesse la lista. Di seguito viene presentato un breve estratto di tale lista dal file di configurazione:

```
<NotAllowed>
  <No>
    <tag>applet</tag>
    <closeTag>true</closeTag>
  </No>
  <No>
    <tag>param</tag>
    <closeTag>>false</closeTag>
  </No>
  <No>
    <tag>font</tag>
    <attrib>size</attrib>
    <attrib>color</attrib>
  </No>
</NotAllowed>
```

Il motore di parsing, all'evento generato quando viene trovata l'apertura del tag *NotAllowed*, inizializza un nuovo vettore che conterrà l'insieme di oggetti di tipo *NotAllowed* (ovvero gli oggetti che descrivono il tag o gli attributi non permessi) specificati in seguito. All'apertura del tag *No* viene istanziato un nuovo oggetto di questo tipo ed inserito nel vettore di cui sopra. Come si può notare nel codice di esempio sopra riportato, le possibilità sono due: è possibile definire non valido un tag e tutti i suoi attributi oppure un insieme di attributi di un tag. Nel primo caso non viene permesso l'utilizzo del tag *applet*. Inoltre viene specificato che tale tag prevede un tag di chiusura che deve essere eliminato. In questo caso la procedura di eliminazione dei tag non permessi provvederà a cancellare la definizione dell'applet con tutti i suoi parametri (<applet ...>) e la chiusura del tag (</applet>), lasciando intatto quanto contenuto fra l'apertura e la chiusura. Nel secondo caso viene definito come non valido il tag *param* che, a differenza di *applet*, non prevede un tag di chiusura. In questo caso verrà cancellato il tag e tutti i suoi attributi. Infine, nell'ultimo caso, vengono definiti come non validi due attributi del tag *font*. Infatti, a differenza delle due definizioni precedenti, all'interno del tag *No* è presente, oltre ad un tag, una lista di attributi. In questo caso la procedura provvederà ad eliminare solamente gli attributi del tag a cui fanno riferimento ed i rispettivi valori, lasciando intatto il resto degli attributi ed il tag stesso. In questo modo, sempre relativamente all'esempio, è possibile evitare che l'utente modifichi la dimensione del carattere (*size*) ed il colore (*color*) ma lasciandolo libero di settare, ad esempio, lo stile del testo selezionato.

9 Conversione di tag

9.1 Descrizione del problema

L'idea di base che ha portato all'implementazione di una procedura per la sostituzione di un tag con un altro tag è la volontà di scrivere solamente codice HTML standard, evitando l'utilizzo di tag deprecati. Tale procedura ha il compito di sostituire tutti quei tag riconosciuti dai browser ma dichiarati deprecati (ovvero il cui supporto in futuro non sarà garantito) con altri tag di egual funzione ma standard. Un esempio banale riguarda la sostituzione del tag *font* con il nuovo tag *span* per la gestione degli stili mediante gli style sheets. Tale conversione nasconde però delle insidie in quanto non tutti i tag standard sono supportati dalla classe *HTMLEditorKit*, dalla classe Java predefinita utilizzata per la visualizzazione in modalità anteprima del codice HTML scritto. Pertanto è stata introdotta una doppia conversione: una all'atto di salvare il lavoro svolto, appunto per convertire i tag deprecati, ed una all'apertura di un documento preesistente, per evitare incompatibilità. È inoltre possibile convertire dei tag in entrambe le occasioni: è il caso dei tag riconosciuti dall'*HTMLEditorKit* che è possibile inserire mediante l'editor.

9.2 Realizzazione

Anche in questo caso è stato definito un tag (*Conversions*) che contenesse tutte le definizioni delle conversioni. Viene di seguito riportato un esempio di conversioni:

```
<Conversions>
  <Conv>
    <old>font</old>
    <new>span</new>
    <closeTag>>true</closeTag>
    <when>onSave</when>
  </Conv>
  <Conv>
    <old>span</old>
    <new>font</new>
    <closeTag>>true</closeTag>
    <when>onLoad</when>
  </Conv>
  <Conv>
    <old>b</old>
    <new>strong</new>
    <closeTag>>true</closeTag>
    <when>onLS</when>
```

</Conv>

</Conversions>

Il motore di parsing effettua il parsing come in tutti gli altri casi fornendo un vettore di oggetti *Conversion*, cioè di strutture dati che descrivono il tipo di conversione da effettuare. L'esempio sopra riportato descrive tutta la casistica possibile.

Nel primo tipo viene definita la conversione del tag *font* con il tag *span* al momento del salvataggio del lavoro, ossia ogniqualvolta vengono richiamati i metodi che leggono il testo contenuto nell'applet. Tale definizione specifica inoltre la presenza del tag di chiusura in quanto dovrà essere sostituito anch'esso. Nel secondo caso, invece, viene definita esattamente la conversione opposta ma da effettuarsi al momento dell'apertura di un nuovo testo. Ciò si è reso necessario in quanto il tag *span* non è tuttora supportato dalla classe *HTMLEditorKit*. Nell'ultimo caso si è definita la conversione del tag *b* con il tag *strong*. Tale conversione non sarebbe strettamente necessaria in quanto il tag *bold* non è stato ancora deprecato, ma è preferibile in quanto il tag *strong* include un valore semantico maggiore e di conseguenza viene trattato in modo diverso dagli analizzatori dei motori di ricerca. Si noti che tale conversione viene effettuata *onLS*, cioè sia al momento dell'apertura sia del salvataggio di testo.

Con questo sistema è quindi possibile definire tutte le conversioni necessarie in qualsiasi momento, anche qualora vengano definiti in futuro deprecati altri tag.

10 Firma dell'applet

10.1 Impostazioni di sicurezza

Scorrendo i requisiti dell'editor in questione è possibile notare fra questi anche la necessità di effettuare il salvataggio ed il caricamento di documenti dal computer ove viene eseguito l'applet (dal client), ovvero di interagire con il filesystem locale. Questa esigenza si scontra con le impostazioni di sicurezza degli applet. Infatti di default un applet non può interagire in alcun modo con la macchina locale per evitare l'apertura di falle di sicurezza. Si pensi ad esempio se aprendo la pagina di un sito contenente un applet questo possa a nostra insaputa leggere il contenuto dei nostri file e spedirlo a qualcun altro. Ovviamente questo comportamento è inaccettabile, da qui le restrizioni imposte agli applet, restrizioni che comunque non riguardano solamente l'accesso al filesystem ma anche, ad esempio, l'accesso alla clipboard. Quest'ultimo esempio non è affatto casuale in quanto firmare l'applet permetterà l'accesso alla clipboard e quindi il copia/incolla sfruttando il sistema operativo.

10.2 Operazioni di firma

Per firmare un applet vengono messi a disposizione all'interno del JDK due comandi indispensabili: *keytool* e *jarsigner*. Mediante il primo comando è possibile generare una coppia di chiavi (pubblica e privata) ed un certificato digitale. Il comando *jarsigner* consente, invece, la firma vera e propria dell'applet, ovvero assegna al file *jar* il certificato precedentemente creato. Vediamo di seguito le operazioni che si sono rese necessarie per firmare l'editor in questione:

- Creazione della coppia di chiavi e del certificato:

```
keytool -genkey -aliaskey -keypass password1 -keystore store.txt -storepass password2
```

Con questo comando si genera la coppia di chiavi che viene denominata con il nome mnemonico *aliaskey* e protetta mediante una password (*password1*). Tale coppia di chiavi viene salvata all'interno di un database protetto (*store.txt*) che può contenere molte coppie di chiavi, ciascuna identificata dall'alias. Anche per accedere a tale database è comunque necessaria un password che nell'esempio è *password2*. Una volta eseguito tale comando vengono richiesti i dati del possessore delle chiavi: nome e cognome, nome dell'unità aziendale, nome dell'azienda, località, provincia e codice identificante lo Stato. Forniti tali dati viene creato (o modificato se già esistente) il file *store.txt* per contenere la coppia di chiavi.

- Firma del file *jar*:

```
jarsigner -keystore store.txt -storepass password2 -keypass password1 -signedjar sekitapplet.jar ekitapplet.jar aliaskey
```

Con questo comando viene creato il file *sekitapplet.jar* etichettando il file *ekitapplet.jar* con un certificato digitale ottenuto dalla chiave privata precedentemente generata. Per fare questa operazione è pertanto

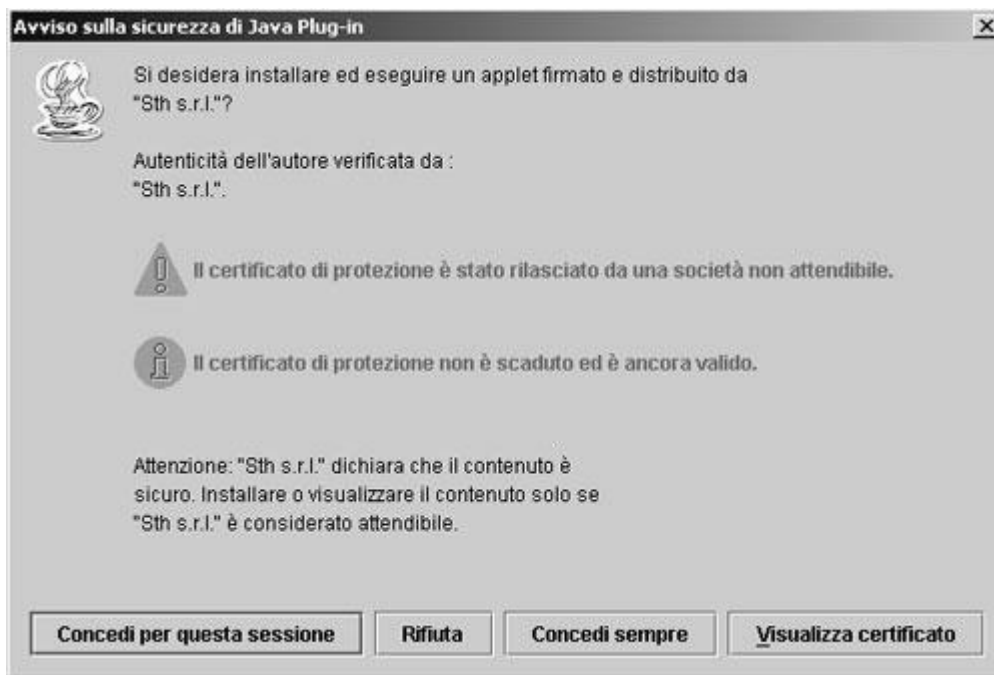
necessario specificare, oltre al file origine e quello destinazione, l'alias della chiave da utilizzare (*aliaskey*), la password per poterla utilizzare (*password1*), il file contenente le chiavi (*store.txt*) e la password per accedervi (*password2*).

- Esportazione della chiave pubblica:

```
keytool -export -keystore store.txt -storepass password2 -alias aliaskey -keypass password1 -file publicKey.txt
```

Questo comando permette, sempre specificando keystore e alias, di esportare in un file la chiave pubblica precedentemente generata. Tale chiave viene distribuita insieme all'applet affinché sia possibile verificare l'identità del firmatario.

Al termine di tali operazioni si ottiene un file *jar* firmato. All'apertura dell'applet appena firmato mediante un browser, con precedentemente installato un plug-in Java, viene visualizzata una richiesta di accettazione dell'esecuzione dell'applet stesso. Insieme a tale richiesta vengono forniti tutti i dati contenuti nel certificato affinché l'utente possa riconoscere la fonte dell'applet e possa deciderne l'esecuzione o meno in base alla fiducia che ripone nel firmatario. Di seguito viene riportato un esempio di richiesta:



Nell'immagine sopra riportata è possibile vedere la richiesta di eseguire un applet firmato da Sth s.r.l. Si può notare come la richiesta riporti la dicitura "Il certificato di protezione è stato rilasciato da una società non attendibile", questo perché il browser contiene un elenco di società considerate affidabili e fra queste non figura Sth s.r.l. Comunque l'utente può decidere se reputare Sth s.r.l. non affidabile e quindi rifiutare l'esecuzione dell'applet, può ritenere completamente affidabile l'emittente e quindi concedere sempre il permesso (la richiesta non verrà più effettuata per nessun programma firmato da questo emittente) o può

decidere di concedere i privilegi richiesti solamente per questa sessione, ovvero solamente per questa esecuzione e di decidere nuovamente in seguito per future esecuzioni. Qualora l'utente sia indeciso può visualizzare il certificato per intero all'interno del quale sono specificati tutti i dati forniti all'atto della creazione della coppia di chiavi mediante il comando *"keytool -genkey ..."*.

10.3 Implementazione

Come sopra spiegato, all'apertura della pagina contenente l'applet viene presentata all'utente una richiesta di concessione di permessi. Qualora l'utente conceda tali permessi l'applet mette a disposizione due nuove caratteristiche: il salvataggio ed il caricamento di file dal computer locale ed il copia/incolla mediante clipboard. Quest'ultima peculiarità consente di effettuare il copia/incolla di testo sfruttando il sistema operativo e quindi al di fuori dell'applet stesso: è pertanto possibile copiare del testo da una qualsiasi fonte ed incollarlo nell'area di editing dell'applet e viceversa. Nel caso l'utente non abbia concesso i permessi richiesti, il copia/incolla funzionerà solamente all'interno dell'applet stesso e verrà gestito mediante le classi predefinite Java. Per quanto riguarda invece l'interazione col filesystem locale, l'applet è in grado di aprire dei file HTML e visualizzarne il contenuto all'interno dell'editor e quindi consentendone la modifica o il salvataggio mediante il CMS. È altresì possibile effettuare il salvataggio su un file HTML nel filesystem locale del testo editato all'interno dell'applet. Queste due caratteristiche rendono l'applet un editor HTML utilizzabile anche off-line. Il progetto iniziale forniva anche la possibilità di salvare il testo editato mediante l'editor in formato rtf ma solamente qualora venisse eseguito come applicazione stand-alone e non come applet. Tale caratteristica è stata introdotta all'interno dell'applet ed è stata potenziata con la possibilità di aprire dei file rtf e, mediante una conversione automatica, presentarli all'interno dell'applet in formato HTML pronti per essere modificati. Ovviamente anche in questo caso se l'utente non concede i permessi necessari l'applet viene eseguito ma non mette a disposizione tutte queste funzionalità.

11 Implementazione su un sito internet

11.1 Integrazione col CMS

Una volta concluso lo sviluppo vero e proprio si è reso necessario integrare l'applet con il CMS. Innanzitutto è stato creato un file php contenente la definizione dell'applet e che risultasse facile includere all'interno di tutte le pagine che avessero avuto la necessità di utilizzare l'editor ed è stato salvato nella cartella contenente il *Back Office* del CMS. Il file XML di configurazione è stato inserito nella cartella contenente tutta la configurazione del CMS al fine di concentrare in un solo punto tutte le opzioni impostabili, evitando dispersione delle informazioni. Il file jar, contenente l'editor vero e proprio, e la chiave pubblica utilizzata per la firma digitale sono state salvate all'interno della cartella contenente i sorgenti del CMS, ovvero tutti i file php che gestiscono ogni sito internet ma che non vengono mai modificati. Inoltre ho provveduto a stilare una breve documentazione che descrivesse come installare l'editor all'interno di un sito e come modificarne la configurazione.

11.2 Installazione in un sito internet

A questo punto si è passati all'installazione dell'editor in un sito internet. La fase in questione risulta essere molto facile e di rapida esecuzione. Infatti è sufficiente verificare che i file di cui sopra siano presenti nel sito in questione, altrimenti è necessario copiarli nelle cartelle previste. Per inserire l'applet all'interno di una qualsiasi pagina internet che fornisca la possibilità di editare del testo HTML è sufficiente copiare la definizione dell'applet contenuta nel file php di cui sopra e modificare la variabile del CMS che dovrà essere passata all'editor e che conterrà il testo da visualizzare. Per gestire il salvataggio del testo scritto all'interno dell'applet è necessario inserire un campo nascosto di una form e riempirlo, nel momento in cui l'utente clicca sul pulsante che attiva il salvataggio, con il testo contenuto nell'editor. Tale stratagemma si è reso necessario al fine di non modificare la struttura del CMS mantenendo così la possibilità di utilizzare l'applet anche nei siti internet precedentemente sviluppati. Non rimane che modificare il file XML in base alle esigenze specifiche del sito e l'editor è pronto da usare.

Conclusioni

Il progetto svolto durante il tirocinio ha reso l'aggiornamento dei siti internet molto più facile e fruibile anche da un'utenza non a conoscenza del linguaggio HTML. L'unico problema derivante dal passaggio da semplice area di testo di una form all'editor in questione è la necessità da parte degli utenti di installare il plug-in Java, plug-in peraltro non disponibile nelle versioni precedenti il sistema operativo *Mac OS X* per Macintosh. Pertanto la soluzione al problema è consistita nel realizzare un sistema automatico che permettesse, qualora non sia già disponibile, di installare in automatico il plug-in scaricandolo dal sito della Sun o, per quanti non potessero o non volessero installare tale plug-in, di presentare la precedente interfaccia HTML e quindi non utilizzare l'editor. Comunque bisogna dire che l'utenza di questo applet è ristretta ai soli autorizzati a modificare i contenuti del sito internet, utenza questa che il più delle volte si concretizza unicamente nel committente, e che pertanto qualora insorgano problemi possono essere agevolmente gestiti.

Questo progetto ha consentito di approfondire le tecniche di programmazione Object-oriented e la conoscenza del linguaggio Java. Inoltre ha permesso di studiare a fondo le problematiche relative allo sviluppo di applet, soprattutto inerenti la sicurezza e la firma degli stessi. Lo sviluppo di questo progetto ha consentito anche lo studio del linguaggio XML e la stesura di un parser per la lettura di dati formattati secondo questo linguaggio. Non ultimo ha permesso la comprensione delle problematiche riguardanti l'ingegnerizzazione del software, in particolare la manutenzione, e lo studio delle licenze con cui è attualmente possibile divulgare il software.

L'applet realizzato viene tutt'ora installato all'interno dei siti internet di maggiore rilevanza prodotti dalla società per cui è stato sviluppato. Il maggior problema, che limita l'utenza di riferimento, è dato dalla pesantezza derivante dall'utilizzo dell'applet stesso: è innanzitutto necessario installare un plug-in di dimensioni non irrilevanti e quindi utilizzare un applet del peso superiore ai 600 KB. Ciò ha portato a limitare i possibili utilizzatori a quanti abbiano un accesso a banda larga alla rete in quanto connettendosi con un comune modem 56.6 Kbit/s i tempi di attesa sarebbero eccessivi. Ne deriva che una futura modifica del progetto dovrebbe mirare in primo luogo ad ottimizzare il codice al fine di renderlo più snello e quindi consentire un download più rapido. Tale inconveniente risulta essere l'unico presentatosi nel seppur breve periodo di utilizzo dell'editor.

Bibliografia

Carli Massimo, “Sondaggi con Java”, in: *DEV*, nr.104, febbraio 2003, Gruppo Editoriale Infomedia.

Eckel Bruce, *Thinking in Java*, New Jersey, Prentice Hall PTR.

Howard Kistler, *Hexidex codex : Java : Ekit*, <http://www.hexidec.com/ekit.php>

Hustead Robert, *Mapping XML to Java*, settembre 2000,
<http://developer.java.sun.com/developer/technicalArticles/xml/mapping>

Java 2 Platform SE v1.3.1, API Specification, <http://java.sun.com/j2se/1.3/docs/api>

Maone Giorgio, “Tecnologie Web Client a confronto”, in: *DEV*, nr.104, febbraio 2003, Gruppo Editoriale Infomedia.

Naccarato Giuseppe, “XML e Document Type Definition”, in: *io Programmo*, anno VI nr.5, giugno 2002, Edizioni Master.

Open Source Iniziative OSI, www.opensource.org

Pawlan Monica, Dodda Satya, *Signed Applets, Browsers and File Access*, aprile 1998,
<http://developer.java.sun.com/developer/technicalArticles/Security/Signed>

SourceForge.net : Project: Ekit (Java HTML Editor), <http://sourceforge.net/projects/ekit/>

The Source for Java Technology, www.java.sun.com

Xerces2 Java Parser, <http://xml.apache.org/xerces2-j/index.html>