UNIVERSITÀ DEGLI STUDI
DI MODENA E REGGIO EMILIA

Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Scienze dell'Informazione

Applicazione Web per la Gestione e Fruizione di Documenti Digitali

Tesi di Laurea
Luca Bonacorsi

Relatore:
Prof. Riccardo Martoglia

Anno Accademico 2009/2010

Indice

Parte I

Capitolo 1
Introduzione alla tecnologia XML

Introduzione……...1

1.1 Introduzione alla tecnologia XML……..2

1.2 Origini e obiettivi……...2

1.3 Relazione tra XML e altri linguaggi.....……..5

1.3.1 Relazione tra XML e SGML……..5

1.3.2 Relazione tra XML e HTML……..6

1.4 Le proprietà dell'XML……..9

1.5 Struttura e sintassi del linguaggio XML……14

1.5.1 Struttura logica del linguaggio XML……....15

1.5.2 Struttura Fisica del linguaggio XML……18

1.5.3 Entità analizzabile o non analizzabile……..19

1.5.4 Entità predefinite……...20

1.5.5 Entità interne ed esterne……21

1.5.6 Sintassi XML……22

1.5.7 Apertura e chiusura dei tag……22

1.5.8 Attributi……..23

1.6 Documenti XML validi e ben formati……..24

1.6.1 Documenti ben formati……24

1.6.2 Documenti validi……..25

1.7 Definizione del tipo di documento (DTD)……..26

1.7.1 Struttura di una DTD……..27

1.7.2 Dichiarazioni di elementi……28
1.7.3 Altri tipi di dati……..29
1.7.4 Simboli relativi alla struttura……30
1.7.5 Gli attributi…….. 31
1.7.6 Entità……...33
1.8 XML Schema – Definizione (XSD)……..36
1.8.1 Tipi di dato…….…….....38
1. 9 Integrazione di grammatiche e namespace……...41
1.9.1 Sintassi dei namespace……..…….....43
1.10 Xml Parser……..44
1.10.1 Elaborazione SAX……...45
1.10.2 Elaborazione DOM……....47
1.11 Il linguaggio XPath……...48
1.12 Esempio di XML utilizzato per il progetto di libreria digitale……51
Conclusioni……..54

Capitolo 2
Introduzione alla tecnologia Adobe Flash

Introduzione……55
2.1 Introduzione ad Adobe Flash……....55
2.2 Introduzione ad Adobe Flash Player……..57
2.3 Introduzione ad ActionScript 3.0……..57
2.3.1 ActionScript 3.0 e la programmazione ad oggetti……..58
2.3.2 Elementi fondamentali della programmazione ad oggetti…….....58
2.4 Classi……..61
2.4.1 Definizioni delle classi……62
2.4.2 Attributi di classe……..63
2.4.3 Corpo della classe……64
2.5 Attributi delle proprietà di classe……66
2.5.1 Attributi dello spazio dei nomi per il controllo dell'accesso……66

2.5.2 Attributo static.....70
2.5.3 Attributi spazio dei nomi definiti dall'utente.....70
2.6 Variabili.....70
2.7 Metodi.....73
2.7.1 Metodi delle funzioni di costruzione.....74
2.7.2 Metodi statici.....76
2.7.3 Metodi di istanza.....77
2.7.4 Metodi supplementari get e set.....79
2.7.5 Metodi vincolati.....81
2.8 Enumerazioni con classi.....83
2.9 Classi delle risorse incorporate.....86
2.10 Interfacce.....86
2.10.1 Definizione di un'interfaccia.....88
2.10.2 Implementazione di un'interfaccia in una classe.....89
2.11 Ereditarietà.....91
2.11.1 Proprietà di istanza ed ereditarietà.....93
2.11.2 Sostituzione di metodi.....97
2.11.3 Istruzione super.....98
2.11.4 Sostituzione del getter e setter.....99
2.11.5 Proprietà statiche non ereditate.....100
2.11.6 Proprietà statiche e catena dell'aree di validità.....102
Conclusioni.....104

Capitolo 3
Actionscript e XML.

Introduzione.....105
3. 1 Operazioni con XML.....106
3.2 L'approccio E4X all'elaborazione XML.....106

3.3 Oggetti XML……..108
3.4 Proprietà XML……..109
3.5 Metodi XML……110
3.6 Oggetti XMLList……112
3.7 Inizializzazione delle variabili XML……..……113
3.8 Assemblaggio e trasformazione di oggetti XML……115
3.9 Letture delle strutture XML……117
3.9.1 Accesso ai nodi principali e secondari……..118
3.9.2 Accesso agli attributi……119
3.9.3 Filtraggio per attributo o elemento……121
3.10 Uso dell'attributo for..in e for each..in……..123
3.11 Uso degli spazi dei nomi XML……124
3.12 Conversione degli oggetti XML……125
3.12.1 Conversione degli oggetti XML e XMLList in stringhe……126
3.12.2 Conversione di stringhe in oggetti XML……..127
3.12.3 Conversione di valori attributo, nomi e valori di testo da stringhe…...127
3.13 Lettura di documenti XML esterni……...128
Conclusione……...129

Parte II

Capitolo 4
Progetto DigitalLab: libreria digitale via Web

Introduzione……..131
4.1 Introduzione alle librerie digitali……..……132
4.2 Archivi digitali……....133
4.3 Biblioteche e librerie digitali……..133
4.4 Kindle, I-pad e le librerie digitali……..134
4.5 La gestione dei diritti digitali……..135

4.6 Le librerie on-line. ……136
4.7 SRS ……..……137
4.7.1.Introduzione……..…137
4.7.2.Descrizione generale…….....139
4.7.3.CLASS DIAGRAM…….…142
Conclusione……...161

Capitolo 5
Progetto DigitalLab: implementazione

Introduzione……..163
5.1 Classe 'Main'……..164
5.2 Classe 'Preload'……200
5.3 Classe 'MacStyleMenu', package 'lookahead.ui'……204
5.4 Classe ' XmlAvailableEvent', package ' lookahead.events '……..208
5.5 Classe 'BitmapLoader', package 'lookahead.loaders'. ……..209
5.6 Classe 'Thumbnail', package 'lookahead.loaders'. ……216
5.7 Classe 'XmlLoaders', package 'lookahead.loaders'. ……..220
5.8 Classe Circle, package 'lookahead.ui'……..222
5.9 Classe 'PreloadItem', package 'lookahead.ui'……..224
5.10 Classe 'Sections', package 'lookahead.ui'……..224
Conclusione……..……225

Bibliografia……..229

Alla mia famiglia, sostegno fondamentale ed insostituibile.

A Carlo, la cui umanità è pari solo al suo genio.

A Elisa, mia cara compagna di vita, fonte di equilibrio e serenità.

A tutto il corpo docente. Gli insegnamenti ricevuti saranno preziosi per sempre.

Parte I

Capitolo 1

Introduzione alla tecnologia XML

In questa prima parte del mio lavoro, ho pensato di dare un'illustrazione delle principali tecnologie utilizzate per la creazione e sviluppo del mio progetto , che verrà poi analizzato in modo più dettagliato nella seconda parte, consistente in una libreria digitabile fruibile via Web.

Sia i documenti effettivamente visualizzati dall'applicazione, che gli elementi visivi e interattivi costituenti l'interfaccia, sono costruiti tramite dati XML, recuperati in fase di 'run-time'. Dati XML che sono generati da pagine JSP che tramite 'query', interrogano una sorgente di dati remota.

E' facile intuire che il cuore del progetto sia proprio il linguaggio XML e per questo motivo incomincio la mia analisi, analizzando questo linguaggio di 'mark-up' , in maniera tale che il lettore abbia una dettagliata conoscenza di quello che è uno dei pilastri per la costruzione della libreria digitale.

Iniziando quindi da una panoramica esplicativa di questo linguaggio, evidenziandone caratteristiche e proprietà , concluderemo il capitolo mettendo in risalto gli aspetti che hanno dato maggiore contributo alla realizzazione del progetto.

1.1 Introduzione alla tecnologia XML.

La sigla XML è l' acronimo di eXtensible Markup Language e letteralmente significa "Linguaggio Estendibile basato su Marcatori".

Il linguaggio XML è diventato oggi uno degli elementi fondamentali per la realizzazione di sistemi informativi, indipendentemente dalla tecnologia usata.

Nasce nel Febbraio del 1998 come formato di rappresentazione delle informazioni,su raccomandazione del W3C. Infatti Il World Wide Web Consortium (W3C), in seguito alla guerra dei browser fu costretto a seguire le individuali estensioni al linguaggio HTML. Dovette scegliere quali caratteristiche standardizzare e quali lasciare fuori dalle specifiche ufficiali dell'HTML. Fu in questo contesto che iniziò a delinearsi la necessità di un linguaggio di markup che desse maggiore libertà nella definizione dei tag, pur rimanendo in uno standard.

Il "progetto XML", che ebbe inizio alla fine degli anni '80 nell'ambito della SGML Activity del W3C, suscitò un così forte interesse a tal punto che la W3C creò un gruppo di lavoro, chiamato XML Working Group, composto da esperti mondiali delle tecnologie SGML(Standard Generalized Markup Language), ed una commissione, XML Editorial Review Board, deputata alla redazione delle specifiche del progetto.

Nel febbraio del 1998 le specifiche divennero una raccomandazione ufficiale con il nome di Extensible Mark-up Language, versione 1.0. Ben presto ci si accorse che XML non era solo limitato al contesto web, ma era qualcosa di più: uno strumento che permetteva di essere utilizzato nei più diversi contesti, dalla definizione della struttura di documenti, allo scambio delle informazioni tra sistemi diversi, dalla rappresentazione di immagini alla definizione di formati di dati.

1.2 Origine e obiettivi.

Obiettivo principale di XML è la rappresentazione di informazioni strutturate, di qualsiasi tipo, purché riconducibili a precise regole, ad una gerarchia di informazioni.

L'Extensible Markup Language (XML) nasce dall'esigenza di portare nel World Wide Web lo Standard Generalized Markup Language (SGML), ovvero lo standard internazionale per la descrizione della struttura e del contenuto di documenti elettronici di qualsiasi tipo. Il linguaggio XML può dunque essere considerato come sottoinsieme di SGML[1] , ottimizzato per il web, ne contiene quindi tutta la potenza, ma non tutte le complesse funzioni raramente utilizzate. Si caratterizza per la semplicità con cui è possibile scrivere documenti, condividerli e trasmetterli nel Web, rendendolo un potente completamento dell'HTML basato su standard. Infatti, l'utilizzo di XML permette di superare il grosso limite attuale del Web, che è quello della dipendenza da un tipo di documento HTML, singolo e non estensibile.

La nascita del XML ha dunque permesso agli utenti del World Wide Web di condividere le informazioni su sistemi differenti; ma ha anche reso possibile a gruppi di persone o ad organizzazioni di creare il proprio linguaggio di markup, specifico per il tipo di informazione che trattano (vedi Figura 1.1) ; per molte applicazioni e per diversi settori, gli esperti hanno già creato linguaggi di markup specifici, come ad esempio il Channel Definition Format, il Mathematical Markup Language ed altri.

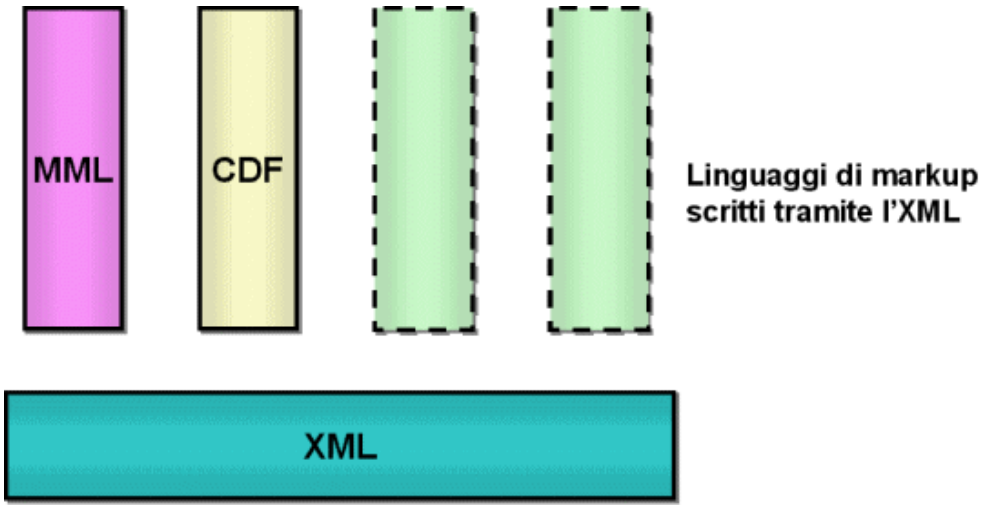

Figura 1.1: L'XML permette di scrivere linguaggi di markup

[1] Quest'ultimo è un metalinguaggio, ovvero un insieme di regole generalizzate usate per creare molteplici linguaggi

Una prima definizione dell'XML identifica in modo puntuale quelli che sono i principali obiettivi progettuali di tale linguaggio, ovvero :

- **XML deve essere utilizzabile in modo semplice su Internet**: in primo luogo, l'XML deve operare in maniera efficiente su Internet e soddisfare le esigenze delle applicazioni eseguite in un ambiente di rete distribuito.

- **XML deve supportare un gran numero di applicazioni**: deve essere possibile utilizzare l'XML con un'ampia gamma di applicazioni, tra cui strumenti di creazione, motori per la visualizzazione di contenuti, strumenti di traduzione e applicazioni di database.

- **XML deve essere compatibile con SGML**: questo obiettivo è stato definito sulla base del presupposto che un documento XML valido debba anche essere un documento SGML valido, in modo tale che gli strumenti SGML esistenti possano essere utilizzati con l'XML e siano in grado di analizzare il codice XML.

- **Deve essere facile lo sviluppo di programmi che elaborino documenti XML**: l'adozione del linguaggio è proporzionale alla disponibilità di strumenti e la proliferazione di questi è la dimostrazione che questo obiettivo è stato raggiunto.

- **Il numero di caratteristiche opzionali deve essere mantenuto al minimo possibile**: al contrario dell'SGML, l'XML elimina le opzioni, in tal modo qualsiasi elaboratore potrà pertanto analizzare qualunque documento XML, indipendentemente dai dati e dalla struttura contenuti nel documento.

- **I documenti XML dovrebbero essere leggibili da un utente e ragionevolmente chiari**: poiché utilizza il testo normale per descrivere i dati e le relazioni tra i dati, l'XML è più semplice da utilizzare e da leggere

del formato binario che esegue la stessa operazione; inoltre poiché il codice è formattato in modo diretto, è utile che l'XML sia facilmente leggibile da parte sia degli utenti che dei computer.

- **La progettazione di XML dovrebbe essere rapida**: l'XML è stato sviluppato per soddisfare l'esigenza di un linguaggio estensibile per il Web. Questo obiettivo è stato definito dopo aver considerato l'eventualità che se l'XML non fosse stato reso disponibile rapidamente come metodo per estendere l'HTML, altre organizzazioni avrebbero potuto provvedere a fornire una soluzione proprietaria, binaria o entrambe.

- **La progettazione di XML deve essere formale e concisa**: questo obiettivo deriva dall'esigenza di rendere il linguaggio il più possibile conciso, formalizzando la formulazione della specifica.

- **I documenti XML devono essere facili da creare**: i documenti XML possono essere creati facendo ricorso a strumenti di semplice utilizzo, quali editor di testo normale.

- **Non è di nessuna importanza l'economicità nel markup XML**: nell'SGML e nell'HTML la presenza di un tag di apertura è sufficiente per segnalare che l'elemento precedente deve essere chiuso. Benché così sia possibile ridurre il lavoro degli autori, questa soluzione potrebbe essere fonte di confusione per i lettori, nell'XML la chiarezza ha in ogni caso la precedenza sulla concisione.

1.3 Relazione tra XML e altri linguaggi.

1.3.1 Relazione tra XML e SGML

SGML è un meta-linguaggio, ovvero un insieme di regole generalizzate usate per creare molteplici linguaggi speciali che prendono il nome di markup language. Le

applicazioni più note di SGML sono HTML, TIM (Telecommunication Interchange Markup); inoltre SGML viene largamente usato dalle grandi industrie di tecnologia come strumento di immagazzinamento e scambio di informazioni di qualsiasi tipo.

L'SGML stesso non ha mai suscitato particolare interesse tra gli sviluppatori Web, probabilmente a causa soprattutto della sua complessità e delle difficoltà che l'utilizzo di questo linguaggio comporta.

L'obiettivo era di includere nell'XML, solo le parti dell'SGML necessarie per la pubblicazione sul Web. XML eredita da SGML la capacità di definire con estrema facilità nuovi marcatori, creando di fatto dei linguaggi di markup personalizzati, mentre la complessità e le caratteristiche opzionali che appesantivano l'SGML sono state pertanto eliminate dall'XML.

1.3.2 Relazione tra XML e HTML

L'HTML (Hypertext Markup Language) è considerato la base del World Wide Web. Questo linguaggio consente infatti di creare in maniera standardizzata pagine di informazioni formattate in grado di raggiungere, tramite Internet, un numero di utenti in costante aumento. Insieme al protocollo HTTP (Hypertext Transport Protocol), l'HTML ha rivoluzionato il modo in cui le persone inviano e ricevono informazioni, ma lo scopo principale per cui è stato realizzato è la visualizzazione dei dati. Per questo motivo, l'HTML prende in considerazione soprattutto il modo in cui le informazioni vengono presentate e non il tipo o la struttura di tali informazioni, aspetti per i quali è stato sviluppato il linguaggio XML (eXtensible Markup Language). La necessità di espandere le capacità di HTML ha spinto i produttori di browser a introdurre nuovi marcatori nella sintassi, rendendola a tutti gli effetti proprietaria e non più standard. Da ciò segue che una pagina HTML che sfrutti marcatori proprietari non può essere visualizzata correttamente se non con il browser adatto, con le ovvie conseguenze che ne derivano.

L'XML invece è uno standard semplice, indipendente dalla piattaforma e adottato su vasta scala.E' un linguaggio di markup aperto e basato su testo che fornisce informazioni di tipo strutturale e semantico relative ai dati veri e propri. Questi "dati sui dati", o metadati, offrono un contesto aggiuntivo all'applicazione che utilizza i dati e consente un nuovo livello di gestione e manipolazione delle informazioni basate su Web.

Spesso, l'XML è considerato una sostituzione dell'HTML. Sebbene questo possa essere in parte vero, in realtà i due linguaggi sono complementari e, relativamente al modo in cui vengono trattati i dati, operano su livelli differenti. Nei casi in cui l'XML viene utilizzato per strutturare e descrivere i dati sul Web, l'HTML è usato per formattare i dati.

Pregi di HTML :

Data la semplicità della struttura dei documenti HTML, rappresenta un veloce strumento per lo scambio delle informazioni sul Web. Le istanze HTML non devono essere validate ma è sufficiente che siano "ben formate", ovvero sintatticamente corrette. Il modo con cui viene visualizzato il documento dipende dallo Style-Sheet del browser stesso.

Limiti di HTML :

I limiti di HTML discendono proprio dai suoi pregi e sono dovuti soprattutto ad un uso distorto di questo strumento. Dal momento che non è richiesta una strutturazione semanticamente corretta delle informazioni, HTML diventa molto povero per le applicazioni di elaborazione dei dati, quindi il problema viene completamente demandato agli applicativi scritti in Java, Java Script, ecc. La sua struttura non è estensibile e questo lo rende praticamente inutilizzabile per le industrie che si sono viste costrette a crearsi standards differenti per ogni diversa applicazione, ed ancora una volta è compito del software elaborare i dati per il trasferimento delle informazioni sul Web. Un altro problema da non sottovalutare si può verificare dalla parte del client qualora non abbia un dispositivo video del

tipo previsto dal server (per esempio dispositivi per il braille o monitor non grafici). Inoltre l'HTML non offre i meccanismi per mantenere il controllo della formattazione. Infatti non si possono specificare le dimensioni video di un documento o controllare le dimensioni della finestra di un browser. Per superare questo problema sono stati inseriti nuovi tag e i fogli di stile; i nuovi tag forniscono istruzioni di formattazione come ad esempio il tag <FONT> per specificare il tipo di carattere; gli utenti però, possono ignorare queste specifiche e utilizzare le loro; inoltre questi tag sono di formattazione e non di descrizione (come dovrebbero essere essendo l'HTML un linguaggio di descrizione del documento). Tutto questo fa sì che uno sviluppatore di pagine Web non sappia mai con certezza cosa un client visualizzerà sullo schermo del computer.

Il W3C (ente guida per lo sviluppo del Web), si rese conto che la creazione di una moltitudine di nuovi tag che rispondessero a ogni possibile esigenza di formattazione era irrealistica ed incoerente con i principi ed i concetti dell'HTML; vennero quindi introdotti i CSS (Cascading Style Sheet) che permettono attraverso le norme di stile di definire come devono apparire determinati elementi di un documento; anche con i CSS però, si deve scegliere fra un assortimento di proprietà predefinite.

Bisogna anche specificare che lasciare al browser la decisione di come presentare un documento, fu esplicitamente voluto come impostazione del progetto HTML[2] . Il browser conosce le preferenze degli utenti e soprattutto l'ambiente in cui si trova ad operare; tutte informazioni che logicamente l'autore di un documento HTML non conosce (a parte pochi casi, come un ambiente intranet omogeneo). Il poter decidere come visualizzare un documento è molto utile ad esempio quando un utente non dispone di un browser grafico, oppure per utenti che hanno problemi di vista e hanno bisogno di caratteri molto grandi, oppure addirittura per utenti non vedenti. Sfortunatamente, le case produttrici di browser, non hanno capito la filosofia di base dell'HTML; il risultato di questo è che allo stato attuale

[2] Come si è già detto HTML è un linguaggio di descrizione del documento

un numero enorme di pagine sul Web, contengono insiemi di tag specifici per determinati browser, e anche per determinate versioni dello stesso browser; queste pagine sono molto spesso illeggibili. Gradualmente l'HTML sta passando da un linguaggio di descrizione, ad un linguaggio di presentazione, con tutti i problemi qui descritti; l'XML può fornire una soluzione, fornendo un layout a chiunque e indipendentemente dal browser utilizzato, attraverso l'utilizzo dell'Exstensible Style Language (XSL), che diventerà uno standard universalmente accettato.

L'XML si differenzia dall'HTML per tre maggiori aspetti:

- Possono essere definiti nuovi tag ed attributi.
- La struttura di un documento può essere vista in modo gerarchico nidificando i tag in ogni livello di complessità.
- Ogni documento XML può contenere una opzionale descrizione della sua grammatica, in modo che possa essere utilizzata da applicazioni che richiedono una validazione della struttura del documento.

1.4 Le proprietà dell'XML.

L'XML è importante in due classi di applicazioni Web: la creazione di documenti e lo scambio dei dati; i server Web attualmente utilizzati richiedono, per essere in grado di servire documenti XML, minime modifiche di configurazione; inoltre il metodo standard di collegamento e la connessione dei documenti XML, utilizza gli URL, che vengono interpretati correttamente della maggior parte del software per Internet.

La sintassi dell'XML è molto simile a quella dell'HTML, ma molto più rigida e severa; anche se al primo impatto questa caratteristica non sembra una proprietà positiva, (soprattutto per chi dovrà scrivere documenti XML), è stata volutamente introdotta dal gruppo di studio del W3C, per facilitare lo sviluppo di applicazioni basate sull'XML e aumentarne le prestazioni (si pensi ad un browser). Inoltre una sintassi chiara e pulita, aumenta la leggibilità di un documento (questo vale in generale); a questo proposito si può dire che una sintassi chiara e pulita, unita alla

possibilità di creare un proprio set di markup, contribuirà a rendere un file XML leggibile quanto un file di solo testo (e in alcuni casi di più).

L'XML non è limitato a un insieme fisso di tipi di elementi, ma permette di definire e utilizzare elementi e attributi personalizzati; per far questo viene fornita una sintassi con cui è possibile specificare gli elementi e gli attributi che possono essere utilizzati all'interno dei documenti. In altre parole è possibile creare un modello, chiamato Document Type Definition (DTD), che descrive la struttura e il contenuto di una classe di documenti. Il DTD è un documento di testo tramite il quale vengono definite tutte le parti di un documento XML: elementi, attributi, elenchi, entità, proprietà e relazioni.

Il linguaggio XML stesso ha un proprio DTD[3] in cui vengono elencate le regole della specifica stessa del linguaggio. Con l'XML è anche introdotta una classe di documenti che fa riferimento al solo DTD dell'XML; la creazione di un DTD personale non è quindi indispensabile, come mostra la Figura 1.2.

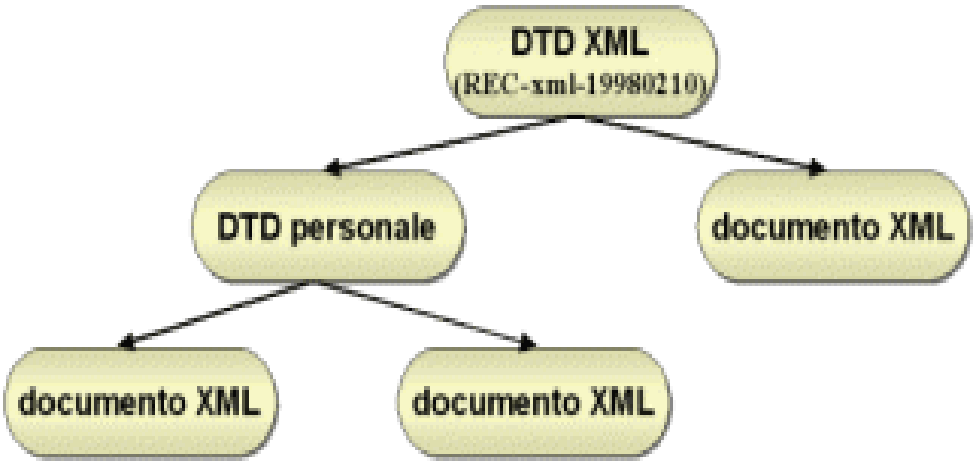

Figura 1.2: Un documento XML può fare riferimento al DTD dell'XML o a un DTD personale

Questa possibilità semplifica molto l'utilizzo dell'XML rispetto all'SGML. Nel linguaggio SGML infatti i DTD sono indispensabili (e spesso complicati da creare

[3] Descritto nella specifica REC-xml-19980210.

e da imparare), e un documento deve fare obbligatoriamente riferimento ad uno di essi.

L'XML permette di creare dei tag personalizzati; inoltre uno stesso documento XML può essere utilizzato per scopi diversi da applicazioni diverse. Come fa una applicazione a riconoscere il markup disegnato per lei e ad evitare di confonderlo con il markup disegnato per altre applicazioni? Ad esempio una applicazione potrebbe utilizzare un elemento chiamato "address" per identificare il domicilio di una persona; un'altra applicazione invece potrebbe utilizzare lo stesso elemento per identificare l'indirizzo elettronico di una persona. Anche un programmatore potrebbe avere difficoltà nel capire l'utilizzo di un determinato elemento; per risolvere questo tipo di problemi, il gruppo di lavoro del W3C ha pensato ad un metodo per individuare le convenzioni che governano l'utilizzo di un particolare set di elementi; l'idea è quella di utilizzare un namespace, cioè un documento in cui viene definito l'utilizzo di un particolare set di elementi. Possiamo vedere il namespace come una collezione di nomi di entità, definite dal programmatore, omogeneamente usate in uno o più file sorgente, il cui scopo è proprio quello di evitare confusione ed equivoci nel caso siano necessarie molte entità con nomi simili, fornendo il modo di raggruppare i nomi per categorie. Un documento XML può far riferimento ad un namespace attraverso un indirizzo Web[4].

L'XML può essere utilizzato come piattaforma per lo scambio di dati tra le applicazioni, ciò è possibile perché è orientato alla descrizione dei dati.
Poniamo il caso che si voglia scambiare le informazioni di database su Internet. Si immagini di utilizzare un browser per rinviare al server le informazioni su un questionario compilato dagli utenti. Questo processo, come molti altri, richiede un formato che possa essere personalizzato per un utilizzo specifico e che sia una soluzione aperta non proprietaria. L'XML è la soluzione per questo tipo di problema. Questo linguaggio in futuro diventerà sempre più importante per lo scambio di dati su Internet.

[4] Il documento di riferimento è il WD-xml-names-19990114.

Come si vede nella figura 1.3, l'XML è in grado di fornire una sola piattaforma per lo scambio di dati tra le applicazioni.

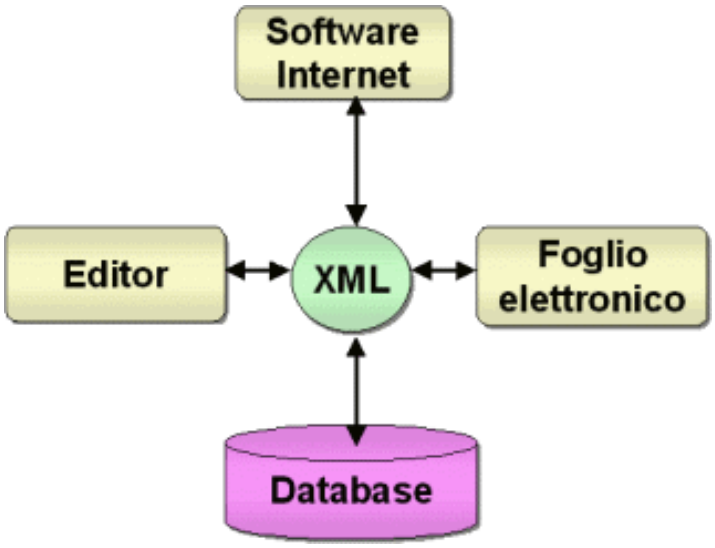

Figura 1.3: L'XML può essere utilizzato come piattaforma per lo scambio di dati

Era sempre stato difficile trovare un formato di interscambio che potesse essere utilizzato per il trasferimento di dati tra database di fornitori differenti e sistemi operativi diversi. Quel tipo di interscambio è ora diventato una delle applicazioni principali dell'XML.

Le pagine HTML hanno l'unico scopo di essere visualizzate da un browser (infatti si dice che i dati nell'HTML sono orientati al video); per questo è molto difficile l'elaborazione successiva delle informazioni contenute nelle pagine HTML. I documenti basati sull'XML invece, non fanno supposizioni su come verranno utilizzati dal client; così le informazioni ricevute possono essere utilizzate da un'applicazione che comprende il linguaggio XML, utilizzando i dati ivi contenuti in altri processi software; quindi uno stesso documento può essere facilmente utilizzato per scopi diversi.

Il collegamento ipertestuale è una caratteristica specifica dell'HTML; attualmente però, esso supporta solo un tipo di collegamento, che è quello unidirezionale; in un vero sistema ipertestuale i tipi di collegamento sono diversi.

Anche se l'XML è uno standard, molte cose sulle tecnologie correlate, quali i fogli di stile ed il collegamento, sono ancora in fase di sviluppo; quindi il modo esatto in cui il collegamento deve essere implementato nell'XML è ancora in fase di studio. Sicuramente dovrà essere compatibile con i meccanismi di collegamento HTML esistenti, supportare l'estensibilità e le proprietà intrinseche dell'XML e implementare i vari tipi di collegamenti propri di un vero sistema ipertestuale. Attualmente lo standard di riferimento è l'Extensible Linking Languge (XLL) del 3/3/1998; è stato diviso in due parti: XML Linking Language (XLink) e XML Pointer Language (XPointer).

L'XSL definisce la specifica per la presentazione e l'aspetto di un documento XML: è stato presentato nel 1997 da un consorzio di industrie software al W3C perché lo approvasse come linguaggio di stile standard per i documenti XML. L'XSL è un sottoinsieme del Document Style Semantics and Specification Language (DSSSL), il linguaggio di stile utilizzato in ambiente SGML; gode delle proprietà di essere estensibile, potente ma nello stesso tempo di facile utilizzo.

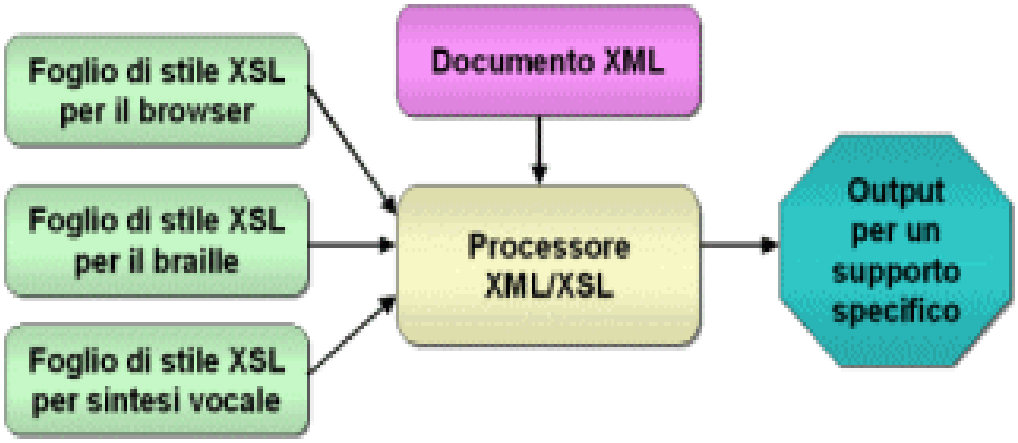

Figura 1.4: L'XSL permette di creare fogli di stile per un qualsiasi formato

Come mostra la Figura 1.4., con l'XSL è possibile creare fogli di stile che permettono la visualizzazione di un documento XML in un qualsiasi formato (audio, video, braille, etc.)[5],

[5] Attualmente lo standard di riferimento è il documento WD-xsl-19990421.

1.5 Struttura e sintassi del linguaggio XML.

Come abbiamo detto nel paragrafo precedente, una delle caratteristiche principali dell'XML è la possibilità di fornire una struttura a un documento. Ogni documento XML comprende sia una *struttura logica* che una *struttura fisica*[6].

La ***struttura logica*** è simile a un modello che indica quali elementi includere in un documento e in quale ordine.
La ***struttura fisica*** contiene i dati effettivi utilizzati in un documento, quali il testo memorizzato nella memoria del computer, un'immagine memorizzata nel WWW e così via.
Per comprendere la struttura di un documento XML, osserviamo questo modello rappresentato nella Figura 1.5:

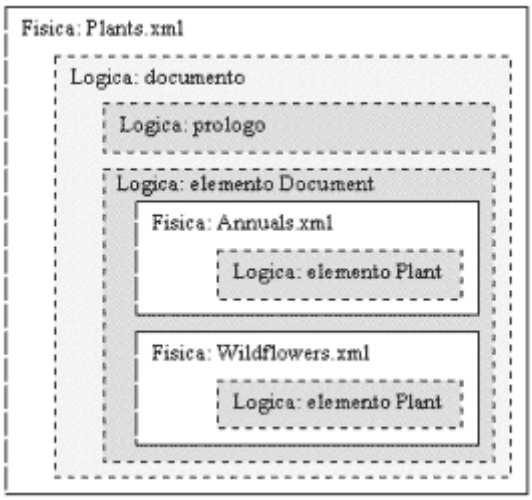

Figura 1.5:Un documento XML ha sia una struttura logica che una struttura fisica.

Un documento XML può includere un gran numero di tipi di istruzioni diverse. Gli elementi di dati del documento sono tra i più comuni. I documenti xml possono includere anche dichiarazioni markup, vale a dire istruzioni al parser XML[7] e istruzioni di elaborazione per programmi di applicazioni che elaborano i dati descritti nel documento.

[6] Strutture di questo tipo modellano i dati utilizzati dalla libreria digitale.
[7] In generale un parser è un software in grado di effettuare l'analisi sintattica di un testo rispetto ad un determinato linguaggio. Nello specifico, un parser XML è un software in grado di effettuare

1.5.1 Struttura logica del linguaggio XML

Come accennato brevemente nel paragrafo precedente, la struttura logica fa riferimento all'organizzazione delle parti di un documento: in altre parole, indica il modo in cui viene creato un documento in contrapposizione al contenuto del documento stesso. Un documento XML è costituito da dichiarazioni, elementi, istruzioni di elaborazione e commenti. Alcuni componenti sono opzionali, altri sono necessari.

Il **prologo opzionale** è il primo elemento strutturale di un documento XML, contenente le meta-infomrzioni. E' costituito da due componenti principali anch'essi opzionali: la *dichiarazione XML* e la *dichiarazione del tipo di documento*.

- ✓ La **Dichiarazione XML** identifica la versione delle specifiche XML a cui è conforme il documento. Sebbene la dichiarazione XML sia un elemento opzionale, deve sempre essere inserita in documento XML.
  Il documento inizia con una dichiarazione XML di base:

  ```
  <?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
  ```

  Come si vede nell'esempo qui sopra, la dichiarazione XML può inoltre contenere una *dichiarazione di codifica* (encoding) e una *dichiarazione di documento autonomo* (standalone).
  La dichiarazione di codifica identifica lo schema di codifica dei caratteri, ad esempio UTF-8 o EUC-JP. Schemi di codifica diversi assegnano formati di caratteri o linguaggi diversi.
  La dichiarazione di documento autonomo identifica l'esistenza delle dichiarazioni di markup esterne al documento. Questo tipo di dichiarazione può assumere valore *yes* o *no,* ad esempio se il documento non fa riferimento a informazioni contenute in altri file, va specificato il valore "yes".

---

l'analisi sintattica di un documento XML. Come minimo, un parser XML è in grado di verificare se un documento XML è ben formato.

- ✓ La **Dichiarazione del tipo di documento** è costituita da codice di markup che indica le regole grammaticali o la definizione del tipo di documento DTD per una particolare classe di documenti. Questa dichiarazione può anche essere diretta a un file esterno che contiene tutta o parte della DTD e deve essere visualizzata dopo la dichiarazione XML e prima dell'elemento Document. Queste stringhe di codice aggiungono una dichiarazione del tipo di documento all'esempio:

```
<?xml version="1.0"?>
<!DOCTYPE Wildflowers SYSTEM "Wldflr.dtd">
```

La dichiarazione del tipo di documento serve inoltre ad indicare l'*elemento radice* (detto anche Document) del documento, ossia il markup deputato ad ospitare il contenuto informativo dell'intero documento. Ogni pagina XML deve possedere uno ed un solo elemento radice. Nell'esempio sopra riportato l'elemento radice è Wildflowers.

- ✓ L**'elemento DOCUMENT,** che come detto poco fa**,** contiene tutti i dati di un documento XML inclusi tutti i sottoelementi nidificati e le entità esterne, infatti tutti gli elementi di un documento xml devono venire annidati all'interno dell'elemento radice. Volendo fare un paragone, questo elemento può essere considerato come all'unità C: del computer. Tutti i dati del computer sono memorizzati in questa singola unità in cui le cartelle e le sottocartelle contengono le singole parti di dati in una struttura logica e di semplice gestione. Queste stringhe di codice aggiungono un elemento Document, in questo caso l'elemento Plant all'esempio[8]:

```
<?xml version="1.0"?>
<!DOCTYPE Wildflowers SYSTEM "Wldflr.dtd">
<PLANT>
 <COMMON>Columbine</COMMON>
 <BOTANICAL>Aquilegia canadensis</BOTANICAL>
</PLANT>
```

[8] I dati XML utilizzato nella libreria digitale hanno come element Document il tag <immagini>.

Un documento XML può ad esempio contenere elementi nidificati[9] e altri documenti. Ogni elemento secondario, cioè un elemento diverso dall'elemento Document risiede interamente all'interno del relativo elemento principale, tramite la relazione PARENTS (genitori) e CHILDS (figli). In generale il codice xml apparirà come segue :

```
<DOCUMENT>
<PARENT1>
        <CHILD1></CHILD1>
        <CHILD2></CHILD2>
</PARENT1>
</DOCUMENT>
```

Applicando questo codice ad un esempio concreto , immaginando che questa è una descrizione di un libro:

Il Mio Primo XML

Introduzione a XML

- Che cosa è HTML
- Che cosa è XML

Sintassi di XML

- Gli elementi devono avere un Tag di chiusura
- Gli elementi devono essere annidati correttamente

Immaginiamo ora che questo documento di XML descriva il libro:

```
<libro>
 <titolo>Il Mio Primo XML</titolo>
 <prod id="33-657" supporto="carta"></prod>
 <capitolo>Introduzione all'XML
  <para>Cos'è l'HTML</para>
  <para>Cos'è l'XML</para>
 </capitolo>

 <capitolo>La sintassi dell'XML
  <para>Gli elementi devono avere un Tag di chiusura</para>
  <para>Gli elementi devono essere annidati correttamente</para>
 </capitolo>
</libro>
```

[9] La ***nidificazione*** è il processo che consente di incorporare un oggetto o un costrutto l'uno all'interno dell'altro.

Identificheremo in "libro", l'elemento della radice . Invece "titolo", "para" e "capitolo" sono elementi figli del libro. Il libro è l'elemento genitore di "titolo", "para" e "capitolo". "titolo", "para" e "capitolo" sono fratelli (o sorelle) perché hanno lo stesso genitore.

Come si diceva prima, all'interno dell' elemento DOCUMENT sono contenuti tutti i dati del documento xml, ovvero tutto ciò che sta tra il Tag d'apertura (incluso) dell'elemento stesso, sino al Tag di chiusura (incluso) dell'elemento . Il contenuto dell'elemento può dunque essere di diverso tipo. Tornando all'esempio precedente del libro, possiamo osservare che "libro" ha un contenuto d'elementi, perché contiene altri elementi. "capitolo" ha un contenuto misto perché contiene sia testo sia altri elementi. "para" ha un contenuto semplice (o testo) perché contiene soltanto del testo. "prod" ha un contenuto vuoto , perché non contiene informazioni, ma ha attributi, ovvero l'attributo chiamato "id" ha il valore "33-657", invece l'attributo chiamato "supporto" ha il valore "carta".

1.5.2 Struttura Fisica del linguaggio XML

La struttura fisica di un documento XML è costituita da tutto il contenuto del documento stesso. Le unità di memorizzazione definite *entità*, possono essere parte integrante del documento o possono essere esterne. Ogni entità è identificata da un nome univoco e da un contenuto specifico che può essere costituito da un singolo carattere all'interno del documento o da un file esterno di grandi dimensioni. In termini di struttura logica di un documento XML, le entità vengono dichiarate nel prologo e viene loro fatto riferimento nell'elemento Document.

Dopo aver dichiarato la DTD, l'entità può essere utilizzata in un punto qualsiasi del documento. Un riferimento di entità indica all'elaboratore di recuperare il contenuto di un'entità, come stabilito dalla dichiarazione di entità, e di utilizzarla all'interno del documento.

1.5.3 Entità analizzabile o non analizzabile

Un'entità può essere ***analizzabile*** *(parsed)* o ***non analizzabile*** *(unparsed)*. Per entità analizzabile si intende un'entità in grado di essere letta dall'elaboratore di XML che ne consente l'estrazione. Al termine dell'estrazione, questo tipo di entità viene visualizzata come parte del testo del documento nella posizione di riferimento dell'entità stessa. Ad esempio, una dichiarazione del tipo analizzabile potrebbe essere questa :

<!ENTITY LR1 "light requirement: mostly shade">

Ogni volta che nel documento viene fatto riferimento all'entità LR1, quest'ultima viene "espansa" e sostituita dal contenuto racchiuso tra i doppi apici. Un evidente vantaggio che ne deriva è che se si desidera modificare il valore dell'entità, è sufficiente effettuare questa operazione soltanto nella dichiarazione della stessa (la modifica si rifletterà automaticamente in qualsiasi punto del documento ove l'entità venga utilizzata).
Come abbiamo appena detto, il contenuto di ogni entità viene aggiunto al documento ogni volta che viene fatto riferimento a quell'entità. Il riferimento ha dunque la funzione di segnaposto per l'autore del contenuto e invece per l'elaboratore di XML individua il punto dove andare a collocare il contenuto effettivo. Per includere un riferimento, bisogna inserire una e commerciale (*&*) e immettere il nome dell'entità seguito da punto e virgole (*;*).

All'interno di un documento assumerebbe il seguente aspetto:

<TERM>Wild Ginger has the following &LR1;</TERM>

In tal caso si parla anche di *entità generali*, il cui utilizzo cioè è consentito all'interno dell'intero documento (validità globale). Viceversa, a quelle entità impiegate soltanto all'interno di una DTD ci si riferisce col nome di *entità di parametro*. Ad esse è possibile ricondursi nello stesso modo appena esposto per le entità generali, avendo però cura di sostituire al simbolo & quello di percentuale (*%*). Ad esempio *%CDF* è un entità di parametro.

Un'entità non analizzabile – talvolta indicata anche come entità binaria, in quanto il suo contenuto è spesso costituito da un file binario (ad esempio un'immagine) – non può essere interpretata direttamente dall'elaboratore XML e pertanto necessita di informazioni diverse e aggiuntive rispetto al caso di entità di tipo parsed.

In particolare , viene richiesta un'*annotazione* che identifica il formato o il tipo di risorsa per cui l'entità viene dichiarata. Ad esempio :

```
<!ENTITY MyImage SYSTEM "Image001.gif" NDATA GIF>
```

Questa dichiarazione stabilisce che l'entità MyImage è un file binario nell'annotazione GIF, ovvero un'immagine. Perché queste dichiarazione di entità siano valide, anche l'annotazione deve essere dichiarata, perché è proprio la *dichiarazione di annotazione* che consente all'applicazione di XML di gestire i file binari esterni. Nel caso dell'annotazione GIF utilizzata nell'esempio, può essere impiegata la dichiarazione di annotazione seguente:

```
<!NOTATION GIF SYSTEM "/Utils/Gifview.exe">
```

Questa stringa di codice indica all'elaboratore di XML ad utilizzare Gifview.exe per elaborare l'entità di tipo *GIF* ogni volta che viene rilevata. Dopo essere stata dichiarata, la dichiarazione di annotazione può essere utilizzata all'interno del documento.

1.5.4 Entità predefinite

Nel linguaggio XML alcuni caratteri sono utilizzati per contrassegnare il documento in modo specifico. Le parentesi angolari (<>) e la barra (/) sono interpretate come markup e non come dati di un carattere effettivo:

```
<PLANT>Blodroot</PLANT>
```

Questi e altri caratteri sono riservati per il markup e non possono essere utilizzati come contenuto. Se si desidera che questi caratteri siano visualizzati come dati, è

necessario utilizzare determinati codici. Nella Tabella 1 elenchiamo alcuni esempi di entità predefinite.

Notazione	Simbolo
&lt	< (parentesi angolare di apertura)
&gt	> (parentesi angolare di chiusura)
&amp	& (e commerciale)
&apos	' (apostrofo)
&quot	" (virgolette doppie)

Tabella 1. Esempi di entità predefinite in XML.

1.5.5 Entità interne ed esterne

Quando si parla di entità interna, si parla di un'entità in cui non esistono unità di memorizzazione fisica separate e il cui contenuto viene fornito nella dichiarazione corrispondente, ad esempio:

<!ENTITY LR1 "light requirement: mostly shade">

Un'entità esterna invece, fa riferimento nella dichiarazione ad una specifica un'unità di memorizzazione mediante un *identificatore pubblico o di sistema*. L'identificatore di sistema fornisce un collegamento, solitamente un URI (Uniform Resource Identifier) , alla posizione in cui si trova il contenuto dell'entità, come ad esempio:

<!ENTITY MyImage SYSTEM "http://www.wildflowers.com/Image001.gif" NDATA GIF>

In questo caso l'elaboratore di XML deve necessariamente leggere il file Image001.gif per recuperare il contenuto di questa entità.

Oltre all'identificatore di sistema, l'entità può includere un identificatore pubblico al fine di fornire un metodo opzionale e alternativo per il recupero del contenuto di un'entità da parte dell'elaboratore di XML. Questo identificatore viene in genere utilizzato se l'applicazione è collegata a una libreria del documento

disponibile pubblicamente. Un esempio di identificatore pubblico è dato dalla seguente dichiarazione:

<!ENTITY MyImage PUBLIC "-//Wildflowers/TEXT Standard images//EN"
"http://www.wildflowers.com/Image001.gif" NDATA GIF>

1.5.6 Sintassi XML

Le regole strutturali del linguaggio XML si riflettono nelle regole linguistiche o *sintassi*. Come si diceva nei paragrafi precedenti, la sintassi dell'Extensible Markup Language non si discosta molto da quella dell'HTML. Le differenze più evidenti si riscontrano casomai nel formalismo della scrittura, dal momento che l'HTML consente una stesura del codice decisamente più "disinvolta" di quella contemplata dalle specifiche XML.

1.5.7 Apertura e chiusura dei tag

Nel codice HTML un elemento contiene in genere sia tag di apertura che di chiusura. A differenza dell'HTML, l'XML richiede che un tag di chiusura venga utilizzato per ogni elemento. Si consideri ad esempio l'elemento HTML Paragraph che dovrebbe in genere includere un tag di apertura, il contenuto testuale e un tag di chiusura come mostrato di seguito:

<P>Questo è un elemento HTML Paragraph.</P>

Non sempre viene utilizzato un tag di chiusura in questo contesto. Questo avviene perché l'HTML e il linguaggio di origine SGML consentono di omettere i tag di chiusura senza invalidare il codice. Nel caso specifico, poiché un paragrafo in HTML non può essere annidato all'interno di un altro paragrafo, l'elaboratore è in grado di leggere il tag di apertura del paragrafo e di presumere che indichi anche la fine del paragrafo precedente. Queste forme di abbreviazione non sono consentite nel linguaggio XML, in accordo con gli obiettivi di chiarezza e

leggibilità base di tale linguaggio. Ed è proprio questa caratteristica che costituisce la differenza sintattica più evidente tra i due linguaggi.
Il linguaggio XML dispone di un tag specifico per gli elementi che non prevedono alcun soggetto informativo: il ***tag di elemento vuoto***. Questo markup unisce i tag di apertura e di chiusura per un elemento privo di contenuto, utilizzando un formato speciale: <NOMETAG/>.
Si noti che in questo caso la barra *segue* il nome del tag, sintassi normalmente non consentita nel linguaggio HTML

1.5.8 Attributi

In maniera simile a quanto accade in HTML, gli *attributi* sono utilizzati per arricchire, fornire informazioni supplementari sugli elementi e consentono di associare ad essi, valori senza che siano considerati parte del contenuto dell'elemento stesso[10]. Ad esempio osserviamo un comune elemento HTML e l'utilizzo di un attributo:

<A HREF = "http://www.microsoft.com">Microsoft Home Page</A>

In questo caso l'elemento Anchor indicato dal tag <A> contiene un attributo denominato HREF. Il valore dell'attributo è http://www.microsoft.com. Mentre il valore non viene mai visualizzato dall'utente, l'attributo contiene importanti informazioni relative all'elemento e fornisce la destinazione dell'ancoraggio. Questo formato del nome e del valore mostra il modo in cui sono utilizzati gli attributi nel linguaggio XML.

Un elemento può contenere uno o più attributi. I valori di attributi devono essere sempre racchiusi tra apici, sono consentite sia le doppie che le singole virgolette. I doppi apici sono i più comuni , ma a volte, se è il valore dell'attributo stesso a contenere degli apici, è necessario usare le singole.

[10] Le informazioni della libreria digitale sono fornite sia da elementi sia da attributi. Per quest'ultimo caso particolare rilevanza riveste l'attributo 'page', che specifica a quale pagina appartiene un dato documento e l'attributo 'id' che attribuisce un identificatore univoco ad ogni documento.

Spesso le stesse informazioni possono essere rappresentate sia tramite attributi che tramite elementi. La scelta tra l'uso di uno piuttosto che dell'altro è soggettiva, anche se le due soluzioni non sono in genere equivalenti. E' opportuno usare gli attributi per informazioni essenziali per l'elemento, come ad esempio gli identificativi (ID).

1.6 Documenti XML validi e ben formati.

Il linguaggio XML possiede due caratteristiche fondamentali, la capacità di fornire una struttura ai documenti e di rendere i dati autodescrittivi. Queste caratteristiche non sarebbero di alcuna utilità se non si potessero far rispettare le regole strutturali e grammaticali.

1.6.1 Documenti ben formati

Un documento XML si dice **ben formato** (*well-formed*) se rispetta tutte le specifiche del linguaggio XML, stabilite dall'ente standardizzatore (W3C), ovvero:

- ✓ il documento può contenere solamente sequenze di caratteri di markup direttamente analizzabili;
- ✓ l'intero contenuto del documento deve essere racchiuso entro un singolo elemento esterno, che può essere l'elemento radice o elemento documento;
- ✓ tutti i valori degli attributi vanno racchiusi tra virgolette;
- ✓ nessun attributo può apparire più di una volta nello stesso blocco;
- ✓ i valori degli attributi stringa non possono contenere riferimenti a entità esterne;
- ✓ tutte le entità devono essere dichiarate;
- ✓ né le entità testo né quelle parametro possono essere ricorsive, direttamente o indirettamente;
- ✓ gli elementi vuoti utilizzano il tag di elemento vuoto;

- ✓ i tag devono essere annidati in modo corretto; in particolare, una coppia di tag iniziale e finale non deve sovrapporsi parzialmente ad un'altra.

La violazione di una qualsiasi di queste regole fa in modo che il documento risultante non venga considerato ben formato. Il concetto di documento ben formato è relativamente nuovo in XML. Un documento XML ben formato è più facile da leggere per un programma ed è pronto per la distribuzione in rete.

1.6.2 Documenti validi

La definizione del tipo di documento DTD specificata nel prologo delinea tutte le regole relative a un documento, ovvero identifica la *grammatica* del linguaggio di markup da rispettare, cioè l'insieme di regole che indicano quali vocaboli (elementi) possono essere utilizzati e con che struttura è possibile comporre frasi (documenti) all'interno del linguaggio di markup. Un documento XML ***valido*** rispetta tutte queste regole rigidamente, oltre che essere conforme anche a tutti i limiti di validità identificati dalle specifiche relative all'XML.

Possiamo dunque dedurre che un documento ben formato può non essere valido rispetto ad una grammatica, mentre un documento valido è necessariamente ben formato. Tra l'altro, un documento valido per una grammatica può non essere valido per un'altra grammatica.
La caratteristica di documento valido si affianca a quella di documento ben formato per costruire documenti XML adatti ad essere elaborati automaticamente. Ed è proprio la verifica che un documento abbia queste due caratteristiche (ovvero sia ben formato e valido rispetto ad una grammatica) l'elaborazione preliminare su cui si fondano tutti i possibili impieghi di XML.

I software che si occupano di queste elaborazioni sono detti *parser* e sono degli strumenti standard disponibili sulle diverse piattaforme.

Possiamo suddividere i parser in due categorie (talvolta può essere lo stesso parser che assume due ruoli):

- **parser non validante,** è un parser che verifica soltanto se un documento è ben formato
- **parser validante,** è un parser che, oltre a verificare che un documento è ben formato, verifica se è corretto rispetto ad una data "grammatica".

Attualmente due sono gli approcci più diffusi alla creazione di "grammatiche" per documenti XML: Dtd (Document Type Definition) e XML Schema.

1.7 Definizione del tipo di documento (DTD).

Da un punto di vista cronologico, il primo approccio per la definizione di grammatiche per documenti XML è rappresentato dai **Document Type Definition (DTD)**.

Una Definizione del Tipo di Documento, DTD appunto, specifica come deve essere strutturato un dato documento: il markup che utilizza, il contenuto dei dati, i file esterni a cui fa riferimento e le istruzioni specifiche per il processore XML. Pur non essendo obbligatorio, si tratta di un componente indispensabile nel caso in cui si vogliano comporre pagine XML valide.
Una precisazione: poiché le voci *dichiarazione del tipo di documento* e *definizione del tipo di documento* (DTD) sono molto simili, spesso ci si ritrova ad utilizzarle indifferentemente. In realtà le cose non stanno affatto in questi termini: la definizione del tipo di documento (DTD), come abbiamo detto poco fa, rappresenta un insieme di dichiarazioni di markup che forniscono la grammatica per un documento o per un'intera classe di documenti. La dichiarazione del tipo di documento, invece, è un semplice riferimento (per l'elaboratore XML) alla definizione del tipo di documento che si intende utilizzare. Al più, può incorporare la DTD ma questo non deve certo indurre a pensare che i due termini si equivalgano.

1.7.1 Struttura di una DTD

XML prevede la possibilità di definire la struttura del documento non solo in un file esterno bensì anche al suo interno. Una DTD può dunque essere costituita da due parti : un sottoinsieme DTD *interno* e un sottoinsieme DTD *esterno*. Nel primo caso si tratta di una DTD incamerata nel documento stesso, mentre il secondo caso si riferisce ad una definizione del tipo di documento memorizzata in un file a parte.

Un documento può contenere uno o entrambi i tipi di sottoinsiemi. In quest'ultimo caso il set interno viene elaborato per primo, acquisendo la precedenza su qualsiasi altro sottoinsieme esterno.
Volendo utilizzare un sottoinsieme DTD in un documento, è sufficiente inserirne il contenuto (in caso di DTD interna) od un riferimento valido (se la DTD è esterna) nella dichiarazione del tipo di documento.
Qui di seguito un esempio di ciascun tipo di DTD:

- DTD Esterno
```
<?xml version="1.0"?>
<!DOCTYPE greeting SYSTEM "hello.dtd">
<greeting>Hello, world!</greeting>
```

- DTD Interno
```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE greeting [
<!ELEMENT greeting (#PCDATA)>
]>
<greeting>Hello, world!</greeting>
```

Nel primo caso il documento fa riferimento per la struttura ad un file esterno (hello.dtd), mentre nel secondo caso il contenuto di questo è inglobato dentro il documento stesso. In ogni caso la Document Type Definition, è contenuta all'interno della "Document Type Declaration", impostata dal tag , e quando ne è inglobata si apre con il carattere "[" e si chiude con il carattere"]". Inoltre nel secondo esempio ELEMENT definisce "greeting" come un elemento, che potra' essere quindi usato come tag nel documento e che contiene del testo (PCDATA=Parsed Character Data).

1.7.2 Dichiarazioni di elementi

Ogni dichiarazione di elemento contiene il nome dell'elemento e il tipo di dati definito *specifiche di contenuto* .

La sintassi di un Dtd si basa principalmente sulla presenza di due dichiarazioni: **<!ELEMENT>** e **<!ATTLIST>**. La prima definisce gli elementi utilizzabili nel documento e la struttura del documento stesso, la seconda definisce la lista di attributi per ciascun elemento.
Ad esempio, la seguente dichiarazione

```
<!ELEMENT articolo(paragrafo+)>
```

indica che <articolo> è l'elemento e che come sottoelemento l'elemento <paragrafo>. Il carattere '+', dopo il nome del sottoelemento, indica il relativo numero di occorrenze, ciò afferma che ci deve essere uno o più elementi "paragrafo" all'interno del documento.

Appartengono alla specifiche di contenuto, i seguenti tipi di dati:

- *Dichiarazione di elemento vuoto*

Per dichiarare che un elemento non può avere alcun contenuto, è possibile utilizzare la parola chiave *EMPTY* nella dichiarazione di elemento, come indicato di seguito:

<!ELEMENT TEST EMPTY>

Un elemento Test di un documento che includa la dichiarazione precedente non potrebbe mai avere alcun contenuto e sarebbe necessario che fosse indicato come elemento vuoto, ad esempio <TEST/>. Anche se gli elementi vuoti potrebbero sembrare inutili, possono contenere attributi in grado di fornire un contenuto significativo o di funzioni specifiche all'interno di un documento. Il tag <BR> in HTML è un esempio di tag di elemento vuoto.

- *Dichiarazione di tutti gli elementi*

La specifica di contenuto ANY è esattamente l'opposto della parola chiave precedente. Se una dichiarazione di elemento utilizza la parola chiave ANY per le

specifiche di contenuto, quel tipo di elemento potrà avere qualsiasi tipo di contenuto in base alle disposizioni della DTD, disposto in un ordine qualsiasi. La dichiarazione di tutti gli elementi assume questo aspetto:

<!ELEMENT TEST ANY>

- *Contenuto di vario tipo.*

Le specifiche di contenuto possono anche essere costituite da un singolo insieme di alternative separate dal simbolo pipe (|). Ad esempio:

<!ELEMENT EXAMPLE (#PCDATA|x|y|z)*>

ovvero l'elemento example dovrà contenere x,y,z.

1.7.3 Altri tipi di dati

All'interno del contenuto dei documenti, il linguaggio XML consente di utilizzare ***dati di caratteri analizzabili*** dichiarati mediante la parola chiave #PCDATA (*Parsable Character Data*) e i ***dati di caratteri normali*** dichiarati mediante la parola chiave CDATA *(Character Data).*

I dati di caratteri analizzabili sono dati di caratteri di markup, contengono quindi tag e sono soggetti all'attività di parsing. I dati di caratteri, solitamente ignorati dai parser, sono costituiti da testo ordinario che può includere caratteri in genere riservati al markup. In base all'impostazione predefinita, gli elaboratori di XML presuppongono che il contenuto di un file XML sia costituito da dati di caratteri.
Mentre i dati di caratteri analizzabili sono in genere utilizzati nel contenuto di un documento XML, i dati di carattere possono essere utilizzati nel caso in cui un autore desideri includere dati che non possono essere analizzati.

Per dichiarare una sezione come dati di carattere, è necessario indicare l'inizio della sezione con la sequenza <![CDATA[e la fine con due parentesi di chiusura

]]. Tutti i dati che risiedono all'interno di questo insieme di marcatori verranno interpretati come semplici dati non analizzabili. Ad esempio:

```
<![CDATA[<p>Questo testo è delimitato da tag di markup, qui privati del
loro significato</p>]]>
```

1.7.4 Simboli relativi alla struttura.

Il linguaggio XML utilizza una serie di simboli, o meglio una simbologia formale[11] per specificare la struttura di una dichiarazione di elementi.

La tabella seguente (Tabella 2) identifica i simboli supportati, lo scopo di ogni simbolo, un esempio di come vengono utilizzati e il loro significato.

Simbolo	Scopo	Esempio	Significato
Parentesi	Racchiudono una sequenza, un gruppo di elementi o una serie di alternative	*(content1, content2)*	L'elemento deve contenere la sequenza *content1* e *content2.*
Virgola	Separa gli elementi di una sequenza e identifica l'ordine in cui devono essere visualizzati	*(content1, content2, content3)*	L'elemento deve contenere *content1*,*content2* e *content3* nell'ordine specificato.
Pipe	Separa gli elementi in un gruppo di alternative	*(content1| content2| content3)*	L'elemento deve contenere *content1*,*content2* o *content3*.
Punto di domanda	Indica che un elemento deve essere visualizzato una sola volta o non apparire mai	*content1?*	L'elemento può contenere *content1*. Se*content1* viene visualizzato, deve apparire una sola volta.
Asterisco	Indica che l'elemento può essere visualizzato ogni volta che l'autore desidera	*content1**	L'elemento può contenere content1. Se viene visualizzato, può apparire una o più volte.

[11] Ci si riferisce alla notazione EBNF (Extended Backus Normal Form), un formalismo matematico per descrivere un linguaggio in modo non ambiguo.

Segno più	Indica che un elemento deve essere visualizzato una o più volte	*content1+*	L'elemento deve contenere content1 una volta, ma può essere visualizzato anche più di una volta.
Nessun simbolo	Indica che deve essere visualizzato un elemento	*content1*	L'elemento deve contenere content1.

Tabella 2: Simboli relativi alla struttura di una dichiarazione di elementi.

1.7.5 Gli attributi.

Gli attributi hanno lo scopo di fornire informazioni aggiuntive relativamente all'elemento o al contenuto dell'elemento cui si riferiscono. Nel linguaggio XML gli attributi vengono dichiarati all'interno della DTD, utilizzando la sintassi seguente:

<!ATTLIST ElementName AttributeName Type Default>

In questo caso <!ATTLIST> rappresenta il tag che identifica una dichiarazione di attributo. La voce ElementName rappresenta il nome dell'elemento a cui vengono applicati gli attributi; la voce AttributeName rappresenta il nome dell'attributo[12]; con Type si identifica il tipo di attributo; Default specifica le impostazioni predefinite relative all'attributo dichiarato.

La seguente tabella (Tabella 3) riassume i tipi di attributi disponibili per il linguaggio XML:

Tipo di attributo	Utilizzo
CDATA	In questo attributo possono essere utilizzati solo dati in formato carattere.
ENTITY	Il valore dell'attributo deve fare riferimento a un'entità binaria esterna dichiarata nella DTD.
ENTITIES	E' equivalente all'attributo ENTITY, ma consente l'utilizzo di più valori separati da spazi.
ID	Il valore dell'attributo deve essere un identificatore univoco. Se un documento contiene attributi ID con lo stesso valore, l'elaboratore produrrà un errore.

[12] Qualsiasi nome risulta valido purché non cominci con le lettere ((‘X’|‘x’) (‘M’|‘m’) (‘L’|‘l’)), riservate allo standard.

IDREF	Il valore deve essere un riferimento a un ID dichiarato in un altro punto del documento. Se l'attributo non corrisponde al valore dell'ID specificato, l'elaboratore produrrà un errore.
IDREFS	E' equivalente all'attributo IDREF, ma consente l'utilizzo di più valori separati da spazi.
NMTOKEN	Il valore dell'attributo consiste in una qualsiasi combinazione di caratteri del token del nome, rappresentati da lettere, numeri, punti trattini, due punti o caratteri di sottolineatura.
NMTOKENS	E' equivalente all'attributo NMTOKEN, ma consente l'utilizzo di più valori separati da spazi.
NOTATION	Il valore dell'attributo deve fare un riferimento a un'annotazione dichiarata in un altro punto della DTD. La dichiarazione può anche essere costituita da un elenco di annotazioni. Il valore deve corrispondere a una delle annotazioni dell'elenco. Ogni annotazione deve avere la relativa dichiarazione nella DTD.
Enumerated	Il valore dell'attributo deve corrispondere a uno dei valori inclusi. Ad esempio: <!ATTLIST *MyAttribute*(*content1*|*content2*)>.

Tabella 3: Tipi di attributi disponibili in XML.

La parte finale di una dichiarazione di attributo riguarda l'impostazione predefinita del contenuto dell'attributo stesso, per i cui valori di default a cui si fa riferimento sono elencati nella presente tabella (Tabella 4) :

Impostazione predefinita	Utilizzo
#REQUIRED	Ogni elemento contenente questo attributo deve specificarne un valore. Un valore mancante può causare un errore.
#IMPLIED	Questo attributo è opzionale. L'elaboratore può ignorare questo attributo se non viene rilevato alcun valore.
#FIXED *fixedvalue*	Questo attributo deve avere il valore *fixedvalue*. Se l'attributo non è incluso nell'elemento, viene stabilito il valore *fixedvalue*.
Default	Identifica un valore predefinito per un attributo. Se l'elemento non include l'attributo, viene stabilito il valore *default*.

Tabella 4 : Impostazioni predefinite (e relativo significato) per gli attributi in XML.

Utilizziamo il seguente esempio per vedere come è la nostra DTD con l'aggiunta di attributi.

```
<?xml version="1.0" ?>
<!DOCTYPE rubrica [
<!ELEMENT rubrica (persona)+>
     <!ELEMENT persona ( nome, cognome, (telefono)+,(email)*)>
       <!ELEMENT nome (#PCDATA)>
       <!ELEMENT cognome (#PCDATA)>
       <!ELEMENT telefono (#PCDATA)>
       <!ATTLIST telefono tipo (Casa|Lavoro|Cellulare) "Casa">
       <!ELEMENT email (#PCDATA)>
```

Quindi nel caso della rubrica, il nome dell'Attributo è "*tipo*", mentre il nome dell'Elemento a cui si riferisce è "*telefono*"; il valore di default è "Casa", mentre gli altri valori sono quelli specificati tra le parentesi tonde, ovvero casa,lavoro, cellulare.

1.7.6 Entità.

A parte le entità predefinite, incorporate direttamente nel codice XML ed utilizzate per visualizzare caratteri generalmente impiegati per il markup, la maggior parte delle entità viene dichiarata all'interno delle DTD. Per definire un'entità si utilizza la dichiarazione **<!ENTITY>** seguendo la stessa sintassi di base utilizzata dalle altre dichiarazioni:

```
<!ENTITY EntityName EntityDefinition>
```

Come già accennato nel corso del paragrafo, le entità della DTD possono essere analizzabili o non analizzabili. Le entità del primo tipo o entità di testo contengono testo che farà parte del documento XML. Le entità del secondo tipo o entità binarie sono in genere riferimenti a un file binario esterno. Le entità non analizzabili possono anche essere costituite da testo non analizzabile ed è quindi preferibile pensare a queste entità come elementi che non sono stati creati per essere considerati parte del codice XML.

- *Entità Interne*

Le entità interne vengono dichiarate nella DTD e includono il contenuto che verrà utilizzato nel documento. Ad esempio, questa riga di codice aggiunge un'entità interna definita *SIGNATURE*:

```
<!ENTITY SIGNATURE "Bill">
```

Ogni volta che viene fatto riferimento all'entità nel documento, *&SIGNATURE*, quest'ultima verrà sostituita dal relativo contenuto, in questo caso *Bill*.

- *Entità Esterne : parole chiave SYSTEM e PUBLIC*

Le entità esterne fanno riferimento a file esterni non inclusi nel documento principale, comprese anche altre pagine XML. Ad esempio, questa entità fa riferimento a un file GIF memorizzato in separata sede rispetto al documento XML originale: ma che verrà comunque visualizzato nel corpo del documento XML:

```
<!ENTITY IMAGE1 SYSTEM "Xmlquot.gif" NDATA GIF>
```

Una dichiarazione di entità esterna può includere la parola chiave *SYSTEM* o *PUBLIC*. Molte DTD sono sviluppate localmente, vengono cioè progettare per essere utilizzate in contesti privati. In questo caso si utilizza di norma la parola chiave *SYSTEM*., seguita da un URI (Uniform Resource Identifier) che indica all'elaboratore dove reperire l'oggetto indicato nella dichiarazione. Nell'esempio precedente è stato utilizzato il nome di un file disponibile in locale; al contrario, nella dichiarazione che segue, l'URI è un indirizzo Web che collega alla posizione dei file di riferimento:

```
<!ENTITY IMAGE1 SYSTEM http://XMLCo.com/Images/Xmlquot.gif NDATA GIF>
```

Alcune DTD, tuttavia, sono standard riconosciuti e disponibili per un'ampia gamma di applicazioni. In tal caso può risultare vantaggioso servirsi di porzioni di codice riutilizzabile, e dunque viene usata la parola chiave PUBLIC seguita dall'identificatore pubblico che l'elaboratore può impiegare se è disponibile una

libreria standard. Dopo l'identificatore pubblico è inserito un URI, simile a quello utilizzato con la parola chiave *SYSTEM*. Un esempio potrebbe essere questo:

```
<!ENTITY IMAGE1 PUBLIC "-//XMLCo//TEXT Standard Images//EN"
    "http://XMLCo.com/Images/Xmlquot.gif" NDATA   GIF>
```

- *Entità Esterne : Annotazioni e Dichiarazioni di annotazioni*

Consideriamo la dichiarazione di entità:

```
<!ENTITY IMAGE1 SYSTEM "Xmlquot.gif" NDATA GIF>
```

Un'*annotazione* NDATA GIF viene visualizzata nella parte finale della dichiarazione. Questa annotazione indica all'elaboratore il tipo di oggetto a cui viene fatto riferimento. A questo punto se viene semplicemente aggiunta la dichiarazione di entità alla DTD e viene eseguita attraverso l'elaboratore, verrà visualizzato un messaggio di errore simile al seguente:

```
Declaration 'IMAGE1' contains reference to undefined notation 'GIF'.[13]
```

L'errore si verifica perché la dichiarazione di entità fa riferimento a un tipo di file binario e all'elaboratore non è stato indicato come operare con questo file. Si tratta di un'entità non analizzabile che l'elaboratore non è in grado di comprendere. Per rimediare è sufficiente dichiarare esplicitamente l'annotazione:

```
<!NOTATION GIF SYSTEM "Iexplore.exe">
```

Questa dichiarazione indica all'elaboratore di utilizzare il programma Iexplore.exe per elaborare il file GIF ogni volta che nella DTD ne viene rilevato uno. In altre parole tramite la *dichiarazione di annotazione* si indica all'elaboratore come operare con un tipo di file binario specifico.

- *Entita di parametro*

Anche se le entità di parametro funzionano in modo simile alle entità generali, possiedono un'importante differenza sintattica. Le entità di parametro utilizzano il

[13] La dichiarazione 'IMAGE1' contiene un riferimento a un'annotazione 'GIF' non identificata.

simbolo di percentuale (%) nelle dichiarazioni e nei riferimenti. Nella dichiarazione di entità il simbolo di percentuale segue la parola chiave *!ENTITY*, ma precede il nome dell'entità come illustrato di seguito. Si noti che è richiesto uno spazio singolo prima e dopo il simbolo &:

```
<!ENTITY % ENCRYPTION "40bit CDATA #IMPLIED 128bit CDATA #IMPLIED">
```

E' ora possibile fare riferimento a questa entità in un altro punto della DTD. Ad esempio:

```
<!ELEMENT EMAIL (TO+, FROM, CC*, BCC*,  SUBJECT?, BODY?)>
   <!ATTLIST EMAIL LANGUAGE(Western|Greek|Latin|Universal)
   "Western" ENCRYPTED %ENCRYPTION;
    PRIORITY (NORMAL|LOW|HIGH) "NORMAL">
```

Il riferimento all'entità di parametro &ENCRYPTION; utilizza lo stesso formato di base del riferimento di entità generale, a eccezione del simbolo % che sostituisce il simbolo &.

Le entità di parametro possono rivelarsi un metodo utile per creare uno stile personale all'interno delle DTD e rendere queste ultime più concise e meglio organizzate. Tuttavia queste entità dovrebbero essere utilizzate con cautela, dato che possono creare situazioni complesse all'interno di un documento in grado di rendere difficoltosa la gestione.

1. 8 XML Schema – Definizione (XSD)

L'approccio basato sui Dtd ci consente di specificare la struttura del nostro documento XML e di ciascun tag utilizzabile al suo interno con una precisione a prima vista accettabile. Tuttavia, se abbiamo bisogno di un maggiore controllo sugli elementi che possono trovarsi all'interno di uno specifico tipo di documenti XML, i Dtd non risultano più sufficienti.

Ad esempio, i Dtd non mettono a disposizione un meccanismo immediato per indicare che un elemento può contenere al massimo un numero predefinito di sottoelementi, né è possibile specificare che un attributo può assumere valori di un certo tipo di dato, ad esempio valori numerici, né di specificare il numero minimo

o massimo di occorrenze di un tag in un documento o altre caratteristiche che in determinati contesti consentirebbero di ottenere un controllo ancora più accurato sulla validità di un documento XML. Queste limitazioni hanno spinto alla definizione di approcci alternativi per definire grammatiche per documenti XML. Tra questi approcci il più noto è **XML Schema**, standard definito dal World Wide Web Consortium, in modo che oltre a offrire la possibilità di validare il modello del contenuto (come la DTD), permettesse anche di descrivere e quindi validare il tipo di dato contenuto nel documento XML. XML Schema, inoltre, è a sua volta un documento XML; questo permette di avere un'immagine visuale e immediata della struttura del documento che descrive.

*Analogamente ad un Dtd***,** un XML Schema è una descrizione formale di una grammatica per un linguaggio di markup basato su XML.
A differenza di un Dtd, che utilizza una propria sintassi specifica, un XML Schema utilizza la stessa sintassi XML per definire la grammatica di un linguaggio di markup. La cosa può sembrare paradossale, ma è invece indice dell'estrema flessibilità di XML. Quindi *uno XML Schema è un documento XML che descrive la grammatica di un linguaggio XML utilizzando un linguaggio di markup specifico.* In quanto documento XML, uno XML Schema ha un root element che contiene tutte le regole di definizione della grammatica.
La **struttura generale** di uno schema XML è la seguente:

```
?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
...
</xs:schema>
```

L'elemento root del documento è rappresentato dal tag **<xs:schema>**. Esso indica al parser che in questo documento saranno utilizzati dei tag definiti dal namespace standard del W3C.

Molto brevemente, i namespace rappresentano un meccanismo per identificare tag appartenenti ad una specifica grammatica. Nel nostro caso questi **tag speciali** sono caratterizzati dal prefisso **xs:**.

XML Schema prevede il tag **<xs:element>** per la definizione degli elementi utilizzabili in un documento XML, specificando nell'attributo name il nome del relativo tag. All'interno di ciascun tag <xs:element> possiamo indicare il tipo di dato dell'elemento e possiamo definire gli eventuali attributi.
Ad esempio, la seguente definizione specifica l'elemento testo che può contenere soltanto stringhe:

```
<xs:element name="testo" type="xs:string" />
```

Questa dichiarazione corrisponde alla seguente dichiarazione Dtd:

```
<!ELEMENT testo (#PCDATA)>
```

1.8.1 Tipi di dato

XML Schema introduce il concetto di **tipo di dato semplice** per definire gli elementi che non possono contenere altri elementi e non prevedono attributi. In pratica è una sequenza di caratteri. E' una specificazione (e restrizione) di CDATA o #PCDATA.
Si possono usare tipi di dato semplici predefiniti oppure è possibile personalizzarli. Sono previsti numerosi **tipi di dato predefiniti**, alcuni dei quali sono riportati nella seguente tabella (Tabella 5) :

Tabella 5 - Tipi di dati semplice , Esempi

Tipo di dato	Descrizione
xs:string	Stringa di caratteri
xs:integer	Numero intero
xs:decimal	Numero decimale
xs:boolean	Valore booleano
xs:date	Data
xs:time	Ora
xs:uriReference	URL

Ad esempio, la seguente dichiarazione:

```
<xs:element name="quantita" type="xs:integer" />
```

permette l'utilizzo dell'elemento quantità in un documento XML consentendo soltanto un contenuto di tipo intero.mIn altre parole, sarà considerato valido l'elemento <quantita>123</quantita> mentre non lo sarà l'elemento <quantita>uno</quantita>.

Ogni tipo semplice è caratterizzato da alcune proprietà, dette ***facets***[14], che ne descrivono vincoli e formati (permessi ed obblighi). E' data possibilità di derivare nuovi tipi, sia per restrizione che per estensione di permessi ed obblighi.

XML Schema prevede dunque la possibilità di definire **tipi di dato semplici personalizzati** come derivazione di quelli predefiniti. Se, ad esempio, abbiamo bisogno di limitare il valore che può essere assegnato all'elemento <quantita> possiamo definirlo nel seguente modo:

```
<xs:element name="quantita" >
      <xs:simpleType>
      <xs:restriction base="xs:integer">
      <xs:minInclusive value="1" />
      <xs:maxInclusive value="100" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

In altre parole, la dichiarazione indica che l'elemento <quantita> è di tipo semplice e prevede una restrizione sul tipo di dato intero predefinito accettando valori compresi tra 1 e 100.

---

[14] I facets sono caratteristiche indipendenti tra di loro che specificano aspetti del tipo:

- **length, minLength, maxLength**: numero richiesto, minimo e massimo di caratteri
- **minExclusive, minInclusive, maxExclusive, maxInclusive**: valore massimo e minimo, inclusivo ed esclusivo
- **precision, scale**: numero di cifre significative e di decimali significativi
- **pattern**: espressione regolare che il valore deve soddisfare
- **enumeration**: lista all'interno dei quali scegliere il valore (simile alla lista di valori leciti degli attributi nei DTD).
- **period, duration, encoding**, ecc.

I **tipi di dato complessi** si riferiscono ad elementi che possono contenere altri elementi e possono avere attributi. Definire un elemento di tipo complesso corrisponde a definire la relativa struttura.

Lo schema generale per la definizione di un **elemento di tipo complesso** è il seguente:

```
<xs:element name="NOME_ELEMENTO">
        <xs:complexType>
          ... Definizione del tipo complesso ...
          ... Definizione degli attributi ...
        </xs:complexType>
</xs:element>
```

Se l'elemento può contenere altri elementi possiamo definire la sequenza di elementi che possono stare al suo interno utilizzando uno dei **costruttori di tipi complessi** previsti:

- **<xs:sequence>** Consente di definire una sequenza ordinata di sottoelementi
- **<xs:choice>** Consente di definire un elenco di sottoelementi alternativi
- **<xs:all>** Consente di definire una sequenza non ordinata di sottoelementi

Per ciascuno di questi costruttori e per ciascun elemento è possibile definire il numero di occorrenze previste utilizzando gli attributi **minOccurs** e **maxOccurs**. Ad esempio, se l'elemento testo può essere presente una o infinite volte all'interno di un paragrafo possiamo esprimere questa condizione nel seguente modo:

```
<xs:element name="paragrafo">
   <xs:complexType>
      <xs:element name="testo" minOccurs="1"
maxOccurs="unbounded"/>
   </xs:complexType>
</xs:element>
```

In questo caso il valore **unbounded** indica che non è stabilito un massimo numero di elementi testo che possono stare all'interno di un paragrafo.

La definizione della struttura di un elemento è **ricorsiva**, cioè contiene la definizione di ciascun elemento che può stare all'interno della struttura stessa.

Per gli **elementi vuoti** è prevista una definizione basata sul seguente schema:

```
<xs:element name="NOME_ELEMENTO">
   <xs:complexType>
      <xs:complexContent>
         <xs:extension base="xs:anyType" />
            ... Definizione degli attributi ...
      </xs:complexContent>
   </xs:complexType>
</xs:element>
```

In altri termini, un elemento vuoto è considerato un elemento di tipo complesso il cui contenuto non si basa su nessun tipo predefinito.

La definizione degli attributi è basata sull'uso del tag **<xs:attribute>**, come nel seguente esempio:

```
<xs:attribute name="titolo" type="xs:string" use="required" />
```

L'attributo **use** consente di specificare alcune caratteristiche come la presenza obbligatoria (**required**) o un valore predefinito (**default**) in combinazione con l'attributo value. Ad esempio, la seguente definizione indica un attributo il cui valore di predefinito è test:

```
<xs:attribute name="titolo" type="xs:string" use="default" value="test" />
```

Bisogna tener presente che se non si specifica esplicitamente l'obbligatorietà dell'attributo, esso è considerato opzionale.

1. 9 Integrazione di grammatiche e namespace

A partire da una grammatica definita tramite uno XML Schema, è possibile sfruttare un parser XML validante per verificare la validità di un documento XML. Il parser avrà bisogno, quindi, sia del documento XML da validare, sia dello schema XML rispetto a cui effettuare la validazione.

Ci sono diversi modi per fornire al parser informazioni sullo schema da utilizzare per la validazione. Uno di questi modi consiste nell'inserire nel documento XML

un riferimento allo schema da utilizzare. Questo riferimento viene associato all'elemento root come nel seguente esempio:

```
<articolo
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="articolo.xsd"
   titolo="Guida ad XML" >
```

Oltre all'attributo titolo sono stati aggiunti due attributi predefiniti:

L'attributo **xmlns:xsi** indica un URL che specifica la modalità con cui si indicherà il riferimento allo schema XML.

L'attributo **xsi:noNamespaceSchemaLocation** indica il nome e l'eventuale percorso del file contenente lo schema XML di riferimento.

Una delle caratteristiche auspicabili nella creazione di un nuovo linguaggio è la possibilità di integrare elementi derivanti da grammatiche diverse. Questa caratteristica consente di riutilizzare parti di grammatiche già definite evitando di dover rifare parte di lavoro già fatto in altri ambiti.

In questo modo, ad esempio, sarebbe possibile integrare un linguaggio per la descrizione di articoli tecnici con un linguaggio pre-esistente per la descrizione di bibliografie, ottenendo un nuovo linguaggio composto. Tuttavia la composizione di linguaggi pone almeno due tipi di problemi:

- un documento che utilizza due grammatiche presenta il **problema della validazione**: a quale schema si deve fare riferimento per validare un documento XML "ibrido"?
- due linguaggi potrebbero avere tag ed attributi con lo stesso nome, anche se utilizzabili in contesti diversi: come fare a risolvere questo tipo di **ambiguità**?

La soluzione a questi problemi deriva dai **namespace**. XML-Namespaces è una "recommendation" di W3C del 1999, e hanno le seguenti caratteristiche:

- Qualificano gli elementi in maniera unica su tutto il Web.
- Permettono agli sviluppatori di utilizzare mark-up provenienti da "schema" o DTD diverse.
- I componenti "schema" sono riutilizzabili.

Un namespace è un insieme di nomi di elementi e nomi di attributi identificati univocamente da un identificatore.
L'identificatore univoco individua l'insieme dei nomi distinguendoli da eventuali omonimie in altri namespace. Per fare un esempio, se nell'ambito di una grammatica per descrivere dei dati anagrafici è stato definito un elemento indirizzo, questo nome potrebbe essere confuso con l'elemento indirizzo definito nell'ambito di una grammatica che descrive messaggi di posta elettronica. L'identificatore del relativo namespace consente di distinguere i due elementi omonimi.
Nell'ambito delle tecnologie XML, un **XML Schema definisce implicitamente un namespace** degli elementi e degli attributi che possono essere usati in un documento XML. Se in un documento XML si utilizzano **elementi definiti in schemi diversi** abbiamo bisogno di un meccanismo che permetta di identificare ciascun namespace e il relativo XML Schema che lo definisce.

1.9.1 Sintassi dei namespace

In un documento XML si fa riferimento ad un namespace utilizzando un attributo speciale (**xmlns**) associato all'elemento radice, come nel seguente esempio:

```
<articolo xmlns="http://www.dominio.it/xml/articolo">
```

Questo indica che l'elemento articolo ed i suoi sottoelementi utilizzano i nomi definiti nel namespace identificato dall'identificatore http://www.dominio.it/xml/articolo.
L'identificatore di un namespace può essere rappresentato da una qualsiasi stringa, purché sia univoca. Proprio per garantirne l'univocità, è prassi ormai consolidata **utilizzare un URI** (Uniform Resource Identifier) come identificatore. Esso non ha nessun valore dichiarativo, ma solo informativo.
Per mettere in relazione un namespace con il relativo XML Schema occorre dichiararlo nel root element come nel seguente esempio:

```
<articolo
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xmlns="http://www.dominio.it/xml/articolo"
    xmlns="http://www.dominio.it/xml/bibliografia"
    xsi:schemaLocation="http://www.dominio.it/xml/articolo
     articolo.xsd"
    xsi:schemaLocation="http://www.dominio.it/xml/bibliografia
     bibliografia.xsd"
>
```

L'attributo **xmlns:xsi** indica un URL che specifica la modalità con cui viene indicato il riferimento allo schema, mentre l'attributo **xsi:schemaLocation** indica il namespace ed il file in cui è definito il relativo XML Schema separati da uno spazio.

E' possibile **combinare più namespace** facendo in modo che ciascun elemento utilizzato faccia riferimento al proprio namespace. Occorre tener presente che quando si fa **riferimento ad un namespace**, questo riferimento vale per l'elemento corrente e per tutti gli elementi contenuti, a meno che non venga specificato un diverso namespace.
Riportare il riferimento ad un namespace per ogni elemento è di solito scomodo e rende di difficile lettura il documento XML. È possibile creare delle abbreviazioni per fare riferimento ai namespace.
Queste **abbreviazioni** sono costituite da caratteri alfanumerici seguiti da due punti (:) dichiarati nell'elemento principale ed utilizzati come prefissi dei nomi degli elementi.

1.10 Xml Parser

Come abbiamo già detto, esistono fondamentalmente due tipi di processori XML, ovvero **validating** e **non-validating**. In genere un processore di tipo validating viene chiamato **parser**[15]. Sia i processori validating che quelli non-validating,

[15] I dati XML della libreria digitale vengono analizzati sintatticamente e convalidati da un parser incapsulato all'interno del flash player, tecnologia della quale si parlerà nel secondo capitolo.

devono riportare le violazioni delle regole di buona formattazione incontrate nell'entità documento ed in generale in una qualsiasi entità parsed.

Un processore validating deve anche riportare le violazioni dei vincoli espressi attraverso le dichiarazioni nel DTD; per far questo deve poter leggere e processare l'intera DTD (un esempio è IE). Un processore non-validating (quali MSXML, XMLSpy) deve poter accedere all'intera entità documento, incluso l'eventuale DTD interna (ma solo per verificare che anche questa parte del documento sia ben formattata); quest'ultimo fornisce vantaggi nelle prestazioni dato che non deve leggere ed elaborare la DTD.

Il parser *ha il compito di caricare il documento XML e di rendere le informazioni da esso contenute disponibili all'applicazione.* Può essere un componente esterno, una libreria o un insieme di classi. La struttura può essere descritta attraverso DTD oppure attraverso XML Schema.

Un parser XML può esporre i propri servizi attraverso **due tipi di interfacce**:
1. **DOM** (Documento Object Model), basato su una rappresentazione astratta in memoria.
2. **SAX** (Simple API for XML) ad eventi. L'interfaccia basata sullo standard DOM permette di vedere il documento XML come se fosse un albero e permette di navigare i vari nodi (elementi). L'interfaccia basata sullo standard SAX non mantiene il documento in memoria, ma, mentre lo carica, invoca i metodi dell'applicazione che si devono occupare di trattare gli elementi che gli sono stati passati.

1.10.1 Elaborazione SAX

SAX , Simple API[16] for XML ,è l'interfaccia standard per il parsing XML **basato su eventi**. Inizialmente nasce come API alternativa a DOM in ambiente Java , ma

[16]Application Programming Interface sono ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per un determinato

in seguito viene poi utilizzata e adottata anche al di fuori del mondo Java, come ad esempio Microsoft.

L'architettura SAX si basa su una sorta di cooperazione fra parser e applicazione al fine di rappresentare il documento nella maniera più consona ad un problema specifico. In termini pratici, *durante l'analisi, l'applicazione riceve dal parser SAX degli eventi opportuni che descrivono il riconoscimento di un particolare elemento del documento*; in questo modo l'applicativo può eventualmente scartare quelle informazioni alle quali non è interessato, rendendo l'operazione molto più veloce e meno onerosa in termini di occupazioni di risorse.

Un parser SAX funziona in un modo molto semplice: legge, carattere dopo carattere, tutto il documento XML dato in input e genera una serie di eventi (callback) ogni volta che riconosce la presenza di particolari informazioni. Tali informazioni sono solo notificate ai vari handler che poi le utilizzeranno nel modo opportuno. La sequenzialità con cui opera un parser SAX, dovendo rileggere più volte il documento, rende però dispendioso l'utilizzo delle risorse e di tempo. Per ovviare a tale problema è stato sviluppato il parser DOM.

Qui di seguito vediamo un esempio di elaborazione SAX:

Dato il documento :	**Il parser SAX registrerà gli eventi:**

`<?xml version="1.0"?>`
`<book>`
`<chapter>`
`<title>Capitolo 1</title>`
`<p>Una frase</p>`
`...`
`</chapter>`
`</book>`
	startDocument()
	startElement(book)
	startElement(chapter)
	startElement(title)
	characters(Capitolo1)
	endElement(title)
	startElement(p)
	character(Una frase)
	endElement(p)
	...
	endElement(chapter)
	endElement(book)
	endDocument()

compito. È un metodo per ottenere un'astrazione, di solito tra l'hardware e il programmatore, o tra software a basso ed alto livello. Le API permettono di evitare ai programmatori di scrivere tutte le funzioni dal nulla.

Vantaggi:

- SAX richiede pochissima memoria d'uso: il documento non è mai tutto in memoria contemporaneamente. E' ideale per documenti MOLTO grandi.
- SAX è velocissimo: non ha praticamente overhead di parsing.
- SAX è ideale per cercare velocemente un'informazione specifica all'interno di un documento XML, poiché lo si può fermare appena l'ha trovata.

Svantaggi:

- SAX non può essere usato per cercare e ricercare informazioni su un documento XML (navigare sulla struttura del documento).
- SAX non può essere usato per modificare o aggiornare la struttura del documento XML (aggiungere o togliere nodi).
- SAX non è implementato dai browser, ma solo come libreria di linguaggi server-side (o in Java anche come applet).

In base a ciò, è consigliabile usare dei parser SAX in caso di documenti XML di grandi dimensioni**,** non soggetti a modifiche e sui quali si devono eseguire operazioni di conteggio (o simili)**.**

1.10.2 Elaborazione DOM

Il DOM, a differenza di SAX, è una specifica del W3C (come XML) e le versioni si indicano per livelli. Per ogni elemento all'interno del documento XML, è prevista la definizione di un'interfaccia.

Il DOM permette di disporre - in memoria - la totalità delle informazioni contenute nel documento XML e di un meccanismo per elaborarle. Conservare in memoria tutte le informazioni è molto vantaggioso nel caso di modifica dei dati estrapolati; ciò sarebbe ancora più evidente nel caso in cui la modifica di alcune informazioni dipendesse dalla loro globalità.

In caso della mancata validità del documento, il parser si interromperà e visualizzerà un messaggio d'errore. Ma anche il parser DOM ha i suoi svantaggi, concernenti l'uso della memoria, specie quando i file XML esaminati hanno dimensioni di alcuni MB.
Il parser DOM costruisce una strutura ad albero che rappresenta il documento e fornisce all'applicazione delle API per navigare l'albero.
I parser DOM, invece, sono consigliati in caso di documenti **XML** strutturati in modo complesso, di dimensioni ridotte**,** soggetti a modifiche e la cui elaborazione dipende dalle informazioni contenute in tutto il documento**.**
Comunque SAX e DOM non sono completamente slegati: la maggior parte dei parser DOM, utilizzano, infatti, la libreria SAX

1.11 Il linguaggio XPath

Le specifiche di **XPath**[17] sono state promosse dal World Wide Web Consortium allo scopo di fornire un valido strumento per il recupero dei contenuti all'interno di un documento xml.

Fondamentalmente XPath è un linguaggio il cui scopo è quello di identificare particolari nodi o set di nodi all'interno di un documento XML che ha per sua natura una struttura ad albero all'interno del quale sono individuabili una serie di nodi. La sua sintassi è molto compatta e, per certi versi, ricorda un po' le espressioni per individuare il percorso di un file o una cartella su un file system.

- *Tipi di dati*

XPath è in grado di trattare quattro tipi differenti di dati: **stringhe**, **numeri**, **booleani** e **nodi**.
Le **stringhe** sono insiemi di caratteri delimitati dalle virgolette: dal momento che le espressioni XPath sono spesso e volentieri utilizzate come valori di attributi spesso queste risultano essere delimitate da apici.

[17] Acronimo di XML Path Language

Un **tipo numerico** è appunto un numero considerato in floating point sul quale è possibile applicare gli operatori numerici visti in precedenza ottenendo un risultato sempre di tipo numerico. XPath definisce anche il valore speciale NaN (Not a Number) che può essere utilizzato per verificare se il valore di un elemento è numerico valido o meno:

```
string(number(mioelemento)) != "NaN"
```

In XPath, come in qualasiasi altro linguaggio, un **tipo booleano** può assumere valore true o false e tale valore può essere convertito nella sua rappresentazione stringa o numerica utilizzando particolari funzioni. L'uso principale del tipo booleano all'interno di un'espressione XPath riguarda la composizione dei predicati[18].

Per **nodo** infine si intende ciascun elemento di un documento xml. Un nodo possiede generalmente un nome che può essere inteso sotto tre differenti aspetti:

- QualifiedName (QName) indica il nome con il quale il nodo appare all'interno di un'istanza di un documento xml insieme eventualmente al prefisso del namespace di appartenenza.
- Local-name indica il Qualified Name privato dell'eventuale prefisso del namespace di appartenenza.
- L'expanded-name è invece costituito dalla coppia Uri associata al namespace di appartenenza e local-name.

Per **node-set** si intende un insieme di nodi identificati da un location step (o location path)[19].

Se ad esempio si ha il seguente documento xml:

```
<cartella>
<documento>
<sezione>sezione 1.1</sezione>
```

[18] I predicati insieme all'asse e al nodetest compongono la sintassi tipica di un location step, dove il nodetest , che deve necessariamente essere presente, indica il tipo di nodi che vogliamo selezionare. L'asse invece, permette di indicare la direzione verso la quale vogliamo esaminare i nodi che abbiamo filtrato. Infine il predicato può essere utile per poter filtrare ulteriormente i nodi dal contesto correntemente selezionato in base all'impostazione di opportune condizioni.

[19] La maggior parte delle espressioni XPath individuano contenuti del documento xml preso in esame o parte di questi: tale processo avviene in uno o più passi, detti **location step**, che messi insieme danno vita ad un percorso completo di localizzazione ovvero una location path.

```
<sezione>sezione 1.2</sezione>
</documento>
<documento>
<sezione>sezione 2.1</sezione>
<sezione>sezione 2.2</sezione>
</documento>
</cartella>
```

e si considera la seguente espressione XPath:

```
/cartella/documento/sezione
```

Il node-set selezionato sarà costituito da tutti gli elementi che hanno nome "sezione". Il context-size indica il numero di nodi presenti all'interno di un node-set, il context-position invece indica la posizione di ciascun nodo all'interno del node-set di appartenenza.

- *Tipi di nodi*

XPath è in grado di recuperare ogni tipo di contenuto presente all'interno di un documento xml: non solo elementi e attributi ma anche commenti, Processing Instruction, nodi testo ed altro ancora.

Le specifiche xml prevedono che l'intero documento sia contenuto all'interno di un'unico elemento detto **radice**(root): in XPath l'elemento radice è rappresentato dal carattere / e tutte le location path espresse a partire da tale elemento,tutte quelle che iniziano cioè col carattere /, sono considerate path assolute. Ogni **nodo elemento** è generalmente composto da un tag di apertura, uno eventuale di chiusura e la porzione di documento xml compreso fra essi che a sua volta è costituito generalmente da altri nodi elemento. I **nodi attributo** servono generalmente a specificare le proprietà degli elementi all'interno dei quali sono definiti. Allo stesso modo sono definiti commenti, Processing Instruction, nodi-testo, namespaces. Individuiamo dunque in un documento XML tre tipologie di nodi fondamentali, ovvero:

● **Root Node, è**' la radice che contiene tutti gli altri nodi

● **Element Node, s**ono gli elementi del documento

●**AttributeNode, s**ono gli attributi di ciascun elemento.

Poi ci sono altri quattro nodi meno frequenti che sono **Namespace Node,** rappresentati dai namespace; **Processing Instruction Node,** ovvero le istruzioni di elaborazione esclusa la dichiarazione di documento; **Comment Node,** i

commenti del documento ed infine **Text Node,** cioè il testo presente all'interno di un elemento.

- *Le funzioni XPath*

Le **funzioni XPath** utilizzate generalmente all'interno dei predicati, assumono una struttura sintattica del tipo:

```
nomefunzione(param1, param2,..., param n)
```

ogni funzione ha un nome distinto ed accetta uno o più argomenti in base ai quali modifica il suo comportamento. Le funzioni disponibili in XPath possono essere classificate in base al tipo del valore che restituiscono o in base al tipo degli argomenti sui quali queste operano.

1.12 Esempio di XML utilizzato per il progetto di libreria digitale

Dopo aver illustrato e descritto le caratteristiche e la struttura del linguaggio XML, vorrei portare un esempio pratico di utilizzo di tale linguaggio applicato al progetto oggetto di questa tesi, ovvero quello di libreria digitale, di cui parlerò più approfonditamente nei capitolo ad esso dedicato.

```
<immagini serverYear="2007" serverMonth="11" serverDate="20"
serverHour="10" serverMinute="58" serverSecond="56"
serverMillisecond="230">
	<immagine id="1" page="i">
		<thumbnail><![CDATA[thumbnails/d.jpg]]></thumbnail>
		<url><![CDATA[images/01.jpg]]></url>
		<tooltip><![CDATA[]]></tooltip>
		<link><![CDATA[]]></link>
	</immagine>
	<immagine id="2" page="ii">
		<thumbnail><![CDATA[thumbnails/b.jpg]]></thumbnail>
		<url><![CDATA[images/02.jpg]]></url>
		<tooltip><![CDATA[]]></tooltip>
		<link><![CDATA[]]></link>
	</immagine>
	<immagine id="3" page="iii">
		<thumbnail><![CDATA[thumbnails/c.jpg]]></thumbnail>
		<url><![CDATA[images/03.jpg]]></url>
```

```
        <tooltip><![CDATA[]]></tooltip>
        <link><![CDATA[]]></link>
</immagine>
<immagine id="4" page="iv">
        <thumbnail><![CDATA[thumbnails/a.jpg]]></thumbnail>
        <url><![CDATA[images/04.jpg]]></url>
        <tooltip><![CDATA[]]></tooltip>
        <link><![CDATA[]]></link>
</immagine>
<immagine id="5" page="v">
        <thumbnail><![CDATA[thumbnails/b.jpg]]></thumbnail>
    <url><![CDATA[images/0000000(19140101)XXIX_3152_002.jpg]]><url>
        <tooltip><![CDATA[]]></tooltip>
        <link><![CDATA[]]></link>
</immagine>
<immagine id="6" page="vi">
        <thumbnail><![CDATA[thumbnails/c.jpg]]></thumbnail>
<url><![CDATA[images/0000000(19140101)XXIX_3152_003.jpg]]></url>
        <tooltip><![CDATA[]]></tooltip>
        <link><![CDATA[]]></link>
</immagine>
<immagine id="7" page="vii">
        <thumbnail><![CDATA[thumbnails/a.jpg]]></thumbnail>
<url><![CDATA[images/0000000(19140101)XXIX_3152_004.jpg]]></url>
        <tooltip><![CDATA[]]></tooltip>
        <link><![CDATA[]]></link>
</immagine>
<immagine id="8" page="viii">
        <thumbnail><![CDATA[thumbnails/b.jpg]]></thumbnail>
<url><![CDATA[images/0000000(19140101)XXIX_3152_002.jpg]]></url>
        <tooltip><![CDATA[]]></tooltip>
        <link><![CDATA[]]></link>
</immagine>
<immagine id="9" page="ix">
        <thumbnail><![CDATA[thumbnails/a.jpg]]></thumbnail>
<url><![CDATA[images/0000000(19140101)XXIX_3152_001.jpg]]></url>
        <tooltip><![CDATA[]]></tooltip>
<immagine id="10" page="x">
        <thumbnail><![CDATA[thumbnails/b.jpg]]></thumbnail>
<url><![CDATA[images/0000000(19140101)XXIX_3152_002.jpg]]></url>
        <tooltip><![CDATA[]]></tooltip>
        <link><![CDATA[]]></link>
```

```
</immagine>
<immagine id="11" page="xi">
	<thumbnail><![CDATA[thumbnails/c.jpg]]></thumbnail>
<url><![CDATA[images/0000000(19140101)XXIX_3152_003.jpg]]></url>
	<tooltip><![CDATA[]]></tooltip>
	<link><![CDATA[]]></link>
</immagine>
<immagine id="12" page="xii">
	<thumbnail><![CDATA[thumbnails/a.jpg]]></thumbnail>
<url><![CDATA[images/0000000(19140101)XXIX_3152_004.jpg]]></url>
	<tooltip><![CDATA[]]></tooltip>
	<link><![CDATA[]]></link>
</immagine>
<immagine id="13" page="xiii">
	<thumbnail><![CDATA[thumbnails/b.jpg]]></thumbnail>

<url><![CDATA[images/0000000(19140101)XXIX_3152_002.jpg]]></url>
	<tooltip><![CDATA[]]></tooltip>
	<link><![CDATA[]]></link>
</immagine>
<immagine id="14" page="xiv">
	<thumbnail><![CDATA[thumbnails/c.jpg]]></thumbnail>
<url><![CDATA[images/0000000(19140101)XXIX_3152_003.jpg]]></url>
	<tooltip><![CDATA[]]></tooltip>
	<link><![CDATA[]]></link>
</immagine>
<immagine id="15" page="xv">
	<thumbnail><![CDATA[thumbnails/a.jpg]]></thumbnail>
<url><![CDATA[images/0000000(19140101)XXIX_3152_004.jpg]]></url>
	<tooltip><![CDATA[]]></tooltip>
	<link><![CDATA[]]></link>
</immagine>
<immagine id="16" page="xvi">
	<thumbnail><![CDATA[thumbnails/b.jpg]]></thumbnail>
<url><![CDATA[images/0000000(19140101)XXIX_3152_002.jpg]]></url>
	<tooltip><![CDATA[]]></tooltip>
	<link><![CDATA[]]></link>
</immagine>
<immagine id="17" page="xvii">
	<thumbnail><![CDATA[thumbnails/a.jpg]]></thumbnail>
<url><![CDATA[images/0000000(19140101)XXIX_3152_001.jpg]]></url>
	<tooltip><![CDATA[]]></tooltip>
```

```
            <link><![CDATA[]]></link>
        </immagine>
        <immagine id="18" page="xvii">
            <thumbnail><![CDATA[thumbnails/b.jpg]]></thumbnail>
        <url><![CDATA[images/0000000(19140101)XXIX_3152_002.jpg]]></url>
            <tooltip><![CDATA[]]></tooltip>
            <link><![CDATA[]]></link>
        </immagine>
        <immagine id="19" page="xviii">
            <thumbnail><![CDATA[thumbnails/c.jpg]]></thumbnail>
        <url><![CDATA[images/0000000(19140101)XXIX_3152_003.jpg]]></url>
            <tooltip><![CDATA[]]></tooltip>
            <link><![CDATA[]]></link>
        </immagine>
        <immagine id="20" page="xx">
            <thumbnail><![CDATA[thumbnails/a.jpg]]></thumbnail>
        <url><![CDATA[images/0000000(19140101)XXIX_3152_004.jpg]]></url>
            <tooltip><![CDATA[]]></tooltip>
            <link><![CDATA[]]></link>
        </immagine>
        <immagine id="21" page="xxi">
            <thumbnail><![CDATA[thumbnails/b.jpg]]></thumbnail>
            <url><![CDATA[images/01.jpg]]></url>
            <tooltip><![CDATA[]]></tooltip>
            <link><![CDATA[]]></link>
        </immagine>
    </immagini>
```

Il lettore a questo punto ha un ‘approfondita conoscenza di come sono strutturati e gestiti i dati che descrivono i documenti visualizzati nella libreria digitale.
Come questi dati vengano recuperati e utilizzati in pratica nel programma, sarà oggetto di trattazione in opportune sezioni nei capitoli 3, 4 e 5.

Capitolo 2

Introduzione alla tecnologia Adobe Flash.

Caratteristiche essenziali della libreria digitale proposta sono un alto livello di interazione, un impatto visivo particolarmente accattivante ed una modalità di fruizione dei documenti che dà luogo ad una vera e propria esperienza multimediale. Attualmente la tecnologia più indicata per sviluppare applicazioni internet che permettano un risultato di questo tipo è Adobe Flash, con il linguaggio di programmazione orientato agli oggetti ActionScript 3.0.

Con tale linguaggio, oltre a costruire gli elementi grafico-testuali ed interattivi, si recuperano i dati XML esterni descriventi i documenti e i documenti stessi, rappresentati in questa libreria digitale da immagini 'raster'.

Procederemo ora alla presentazione dettagliata di queste tecnologie evidenziandone di volta in volta gli aspetti più importanti per la loro comprensione da parte del lettore e gli aspetti più rilevanti ai fini della realizzazione della libreria digitale.

2.1 Introduzione ad Adobe Flash.

Adobe Flash è un'applicazione per lo sviluppo di contenuti, interfacce utente e applicazioni web multimediali ed interattive.

Flash è considerato il software leader di settore per l'authoring interattivo e la distribuzione uniforme di esperienze coinvolgenti su computer, dispositivi mobili e schermi di ogni dimensione o risoluzione.

Adobe Flash permette di organizzare contenuti grafico-testuali in una modalità davvero innovativa ed unica. Questo software permette infatti di inserire grafica vettoriale, immagini raster, testi, video in più linee temporali.
Altri 'software' utilizzano una linea temporale per gestire contenuti che possono variare nel tempo, generando quindi animazioni, ma Adobe Flash permette al grafico e al programmatore di gestire 'n' (con n>=1) linee temporali contemporaneamente, dando addirittura la possibilità di inserire alcune di esse all'interno di altre. Tutto ciò rende possibile la creazione e gestione di animazioni ed elementi interattivi impensabili da realizzare con qualunque altro software.

Una linea temporale altro non è che una sequenza di fotogrammi, ognuno dei quali può contenere specifici stati di elementi multimediali.

I fotogrammi vengono letti in sequenza da un entità chiamata 'testina di lettura' o playhead in inglese.

La *playhead* legge i contenuti dei singoli fotogrammi man mano li visita ad una velocità impostabile nota come FPS (fotogrammi al secondo).
Tali contenuti vengono rappresentati in un ambiente rettangolare, lo stage, di cui si possono variare le dimensioni.

Gli elementi multimediali così organizzati sono gestibili, in fase di run-time, da un potente linguaggio di programmazione orientato agli oggetti: ActionScript 3.0.
Si deduce che Flash è un ambiente davvero potente dove sia grafici che programmatori possono dare il loro contributo per creare applicativi fortemente interattivi che, al momento, non è possibile replicare con altre tecnologie.

Grazie a Flash è possibile, quindi, creare immagini statiche e dinamiche; siti internet evoluti, dinamici e accattivanti (i filmati Flash possono essere inseriti in un file HTML, esattamente come uno dei normali contenuti di una pagina internet e divenire quindi una parte o l'intero sito internet); soluzioni per l'e-commerce e l'e-business; animazioni per il Web, il cinema e la televisione; prodotti multimediali distribuiti su CD e DVD; contenuti interattivi per la telefonia mobile

e i palmari; presentazioni multimediali; gallerie fotografiche; giochi, applicazioni multiutente e quant'altro la creatività e fantasia dello sviluppatore riesce a concepire.

2.2 Introduzione ad Adobe Flash Player.

Adobe Flash Player è un software creato da Adobe che consente di visualizzare applicazioni e contenuto interattivo sul web prodotto con Flash. Tale contenuto è rappresentato da file SWF ed FLV. Il Flash Player è diffuso in tutto il web e ed è installato sul 98% dei computer di tutto il mondo. Tale software è costantemente aggiornato e potenziato da Adobe.

Gli aggiornamenti sono importanti poiché garantiscono il corretto funzionamento di tale tecnologia e possono includere modifiche alla sicurezza o nuove funzionalità del prodotto.

Adobe Flash Player[1] è una tecnologia supportata da un'ampia gamma di piattaforme e browser ed è disponibile in diverse lingue: cinese semplificato, cinese tradizionale, inglese, francese, tedesco, giapponese, italiano, coreano e spagnolo.

2.3 Introduzione ad ActionScript 3.0

Come accennato in precedenza, Flash è dotato di un linguaggio di programmazione, ActionScript 3.0, tramite il quale è possibile gestire contenuti grafico-testuali e multimediali, dati recuperati da server remoti e in locale, stream di byte.

ActionScript 3.0 offre una buona aderenza agli standard. E' un linguaggio aderente allo standard fondato sulla bozza di specifica ECMAScript (ECMA-262). Possiede un'API XML basata su ECMAScript for XML (E4X)[2], un'estensione che

[1] E' grazie a questo software , reso disponibile all'interno del codice HTML come plug-in o controllo ActiveX, che l'utente può visualizzare ed utilizzare la libreria digitale via web.
[2]Come indicato nella specifica ECMA-357

aggiunge XML al linguaggio come tipo di dati nativo.
E' dotato di un modello di eventi basato sulla specifica degli eventi DOM (Document Object Model) Livello 3.
ActionScript 3.0 ha un tempo di esecuzione estremamente rapido, fino a otto volte più elevato, rispetto alle versioni precedenti.

2.3.1 ActionScript 3.0 e la programmazione ad oggetti.

Questo paragrafo descrive gli elementi di ActionScript che supportano la programmazione orientata agli oggetti (OOP). Non vengono descritti i principi generali della programmazione orientata agli oggetti, quali la progettazione degli oggetti, l'astrazione, l'incapsulamento, l'ereditarietà e il polimorfismo. Il capitolo spiega come è possibile applicare questi principi utilizzando ActionScript 3.0.

Poiché le basi di ActionScript sono quelle di un linguaggio per la creazione di script, il supporto della programmazione OOP da parte di ActionScript 3.0 è opzionale. I programmatori dispongono pertanto di una notevole flessibilità nella scelta dell'approccio migliore per i progetti, a seconda del livello di complessità e dell'ambito di applicazione. Per operazioni di portata limitata, l'uso di ActionScript con un paradigma di programmazione procedurale può risultare del tutto sufficiente. Per i progetti di grandi dimensioni, l'applicazione dei principi della programmazione a oggetti può rendere il codice più facile da comprendere, gestire ed estendere.

2.3.2 Elementi fondamentali della programmazione ad oggetti

- *Introduzione alla programmazione orientata agli oggetti*

La programmazione orientata agli oggetti (OOP, Object Oriented Programming) è un sistema di organizzazione del codice di un programma mediante raggruppamento all'interno di oggetti, ovvero singoli elementi che includono informazioni (valori di dati) e funzionalità. L'uso di un approccio orientato agli oggetti per organizzare un programma consente di raggruppare particolari

informazioni (ad esempio, informazioni sui brani musicali, quali titolo dell'album, titolo della traccia o nome dell'artista) insieme a funzionalità o azioni comuni[3], associate a tali informazioni (quali "aggiungi traccia all'elenco di riproduzione" o "riproduci tutti i brani di questo artista"). Tali voci vengono combinate in un'unica voce, un oggetto (ad esempio, un oggetto "Album" o "MusicTrack"). Essere in grado di raggruppare insieme tali valori e funzioni offre vari vantaggi, inclusa la necessità di tenere traccia di una sola variabile, anziché di più variabili, organizzando tutte le funzioni correlate insieme e strutturando i programmi in modo più vicino al mondo reale.

- *Comuni attività di programmazione orientata agli oggetti*

In pratica, la programmazione orientata agli oggetti consiste in due parti. Una parte riguarda le strategie e le tecniche di progettazione di un programma (spesso chiamata anche *progettazione orientata agli oggetti*). L'altra parte della programmazione a oggetti riguarda le strutture di programmazione vere e proprie disponibili in un determinato linguaggio di programmazione e necessarie per costruire un programma seguendo un approccio orientato agli oggetti. Qui di seguito elenchiamo le principali attività comuni della programmazione a oggetti:

- Definizione di classi
- Creazione di proprietà, metodi e proprietà accessor get e set (metodi supplementari)
- Controllo dell'accesso a classi, proprietà, metodi e proprietà accessor
- Creazione delle proprietà e dei metodi statici
- Creazione di strutture simili a enumerazioni
- Definizione e uso delle interfacce
- Uso dell'ereditarietà, inclusa la sostituzione degli elementi di classi.

[3] Proprio grazie a queste caratteristiche, come vedremo nel Capitolo 4, sarà possibile organizzare in maniera ordinata e modulare, la logica programmativa per la costruzione e visualizzazione dei documenti digitali.

- *Concetti e termini importanti*

L'elenco di riferimento seguente contiene alcuni termini importanti caratterizzanti il linguaggio Actionscript 3.0:

- ***Attributo***: caratteristica assegnata a un elemento di classe (quale una proprietà o un metodo) nella definizione di classe. Gli attributi vengono generalmente usati per definire se la proprietà o il metodo sarà accessibile dal codice in altre parti del programma. Ad esempio, `private` e `public` sono attributi. Un metodo private può essere chiamato solo dal codice all'interno della classe, mentre un metodo public può essere chiamato da qualsiasi codice del programma.
- ***Classe:*** definizione della struttura e comportamento degli oggetti di un determinato tipo (come un modello o progetto base per oggetti di quel tipo di dati).
- ***Gerarchia di classe***: struttura di più classi correlate che specifica quali classi ereditano funzionalità da altre classi.
- ***Funzione di costruzione***: particolare metodo definibile in una classe che viene chiamato quando viene creata un'istanza di tale classe. Una funzione di costruzione è comunemente utilizzata per specificare valori predefiniti o per eseguire operazioni di impostazione per l'oggetto.
- ***Tipo di dati***: tipo di informazioni che una particolare variabile può memorizzare. In generale, *tipo di dati* ha lo stesso significato di *classe*.
- ***Operatore punto***: segno di punto (`.`) utilizzato da ActionScript e da altri linguaggi di programmazione per indicare che un nome è riferito a un elemento secondario di un oggetto (quale una proprietà o un metodo). Ad esempio, nell'espressione `myObject.myProperty`, l'operatore punto indica che il termine`myProperty` si riferisce a un valore che è un elemento dell'oggetto denominato `myObject`.
- ***Enumerazione:*** serie di valori costanti correlati raggruppati per praticità come proprietà di un'unica classe.
- ***Ereditarietà***: meccanismo della programmazione a oggetti che consente a una

definizione di classe di includere tutte le funzionalità di una diversa definizione di classe (e, in generale, di aggiungere tale funzionalità).

- ***Istanza***: oggetto vero e proprio creato in un programma.
- ***Spazio dei nomi***: attributo personalizzato che consente un controllo più accurato su quale codice può accedere ad altro codice.

2.4 Classi

Una classe è una rappresentazione astratta di un oggetto. In una classe sono memorizzate informazioni sui tipi di dati che un oggetto può contenere e sui comportamenti in base ai quali un oggetto può funzionare. L'utilità di tale astrazione può non risultare evidente quando si creano brevi script contenenti solo pochi oggetti che interagiscono tra loro; tuttavia, con l'ampliarsi dell'area di validità di un programma e con l'aumentare del numero di oggetti da gestire, le classi consentono di controllare più accuratamente la creazione degli oggetti e il modo in cui essi interagiscono tra loro[4].

Fino alla versione di ActionScript 1.0, i programmatori potevano utilizzare gli oggetti funzione per creare costrutti somiglianti a classi. Con ActionScript 2.0 è stato introdotto il supporto delle classi con parole chiave quali `class` ed `extends`. In ActionScript 3.0, non solo è previsto il supporto delle parole chiave introdotte in ActionScript 2.0, ma vengono introdotte anche nuove funzionalità, quale il controllo dell'accesso ottimizzato mediante gli attributi `protected` e `internal` e un migliore controllo dell'ereditarietà grazie all'uso delle parole chiave `final` e `override`.

Se si ha già esperienza nella creazione di classi con linguaggi di programmazione quali Java, C++ o C#, le procedure di ActionScript risulteranno familiari.

[4] Nel Capitolo 4 verrà evidenziata la struttura a classi della libreria digitale. Ogni aspetto della libreria infatti, sarà tradotto logicamente con la classe opportuna. Il programma nella sua completezza risulta essere un insieme di oggetti comunicanti tra di loro.

ActionScript utilizza infatti molte parole chiave e nomi di attributo comuni a tali linguaggi di programmazione, quali `class`, `extends` e `public`[5].

2.4.1 Definizioni delle classi

Le definizioni delle classi in ActionScript 3.0 impiegano una sintassi simile a quella utilizzata in ActionScript 2.0. La sintassi corretta per la definizione di una classe richiede la parola chiave `class` seguita dal nome della classe. Il corpo della classe, racchiuso tra parentesi graffe (`{}`), segue il nome della classe.

Ad esempio, il seguente codice consente di creare una classe denominata Shape contenente una variabile denominata `visible`:

```
public class Shape
{
    var visible:Boolean = true;
}
```

Una modifica significativa della sintassi riguarda la definizione delle classi che si trovano all'interno di un pacchetto. In ActionScript 2.0, se una classe si trovava all'interno di un pacchetto, il nome del pacchetto doveva essere incluso nella dichiarazione della classe. In ActionScript 3.0, con l'introduzione dell'istruzione `package`, il nome del pacchetto deve essere incluso nella dichiarazione del pacchetto, anziché nella dichiarazione della classe. Ad esempio, le seguenti dichiarazioni di classe mostrano come la classe BitmapData, che fa parte del pacchetto flash.display, viene definita in ActionScript 2.0 e in ActionScript 3.0:

```
// ActionScript 2.0
class flash.display.BitmapData {}
```

[5] In questo capitolo, il termine proprietà si riferisce a qualsiasi membro di una classe o di un oggetto, incluse variabili, costanti e metodi. Inoltre, anche se spesso i termini classe e statica vengono utilizzati in modo intercambiabile, in questo capitolo essi si riferiscono a concetti distinti. Ad esempio, in questo capitolo il termine proprietà di classe si riferisce a tutti i membri di una classe e non solo ai membri statici.

```
// ActionScript 3.0
package flash.display
{
    public class BitmapData {}
}
```

2.4.2 Attributi di classe

ActionScript 3.0 consente di modificare le definizioni delle classi mediante l'impiego di uno dei seguenti attributi:

Attributo	Definizione
`dynamic`	Consente di aggiungere proprietà alle istanze in fase di runtime.
`Final`	La classe non può essere estesa da un'altra classe.
`Internal` (valore predefinito)	Visibile ai riferimenti che si trovano all'interno del pacchetto corrente.
`Public`	Visibile a tutti i riferimenti.

In tutti questi casi, a eccezione di `internal`, è necessario includere espressamente l'attributo per ottenere il comportamento associato. Ad esempio, se durante la definizione di una classe non si include l'attributo `dynamic`, non sarà possibile aggiungere proprietà a un'istanza della classe in fase di run-time. Per assegnare esplicitamente un attributo è necessario collocarlo all'inizio della definizione della classe, come indicato nel codice seguente:

```
dynamic class Shape {}
```

Si tenga presente che nell'elenco non è incluso un attributo denominato `abstract`. Questo perché le classi abstract non sono supportate in ActionScript 3.0. Si noti inoltre che l'elenco non contiene attributi denominati `private` e `protected`. Tali attributi hanno un significato unicamente all'interno di una definizione di classe e non possono essere applicati alle classi

stesse. Per fare in modo che una classe non sia pubblicamente visibile al di fuori di un pacchetto, è necessario inserire la classe in un pacchetto e contrassegnarla con l'attributo `internal`. In alternativa, è possibile omettere sia l'attributo `internal` che l'attributo `public`; in tal modo il compilatore inserirà automaticamente l'attributo `internal`. Se si desidera che una classe non sia visibile al di fuori del file di origine che la definisce, collocate la classe alla base del file di origine, sotto la parentesi graffa che chiude la definizione del pacchetto.

2.4.3 Corpo della classe

Il corpo della classe, racchiuso tra parentesi graffe, consente di definire le variabili, le costanti e i metodi della classe. Nell'esempio seguente è riportata la dichiarazione per la classe Accessibility nell'API di Adobe Flash Player:

```
public final class Accessibility
{
    public static function get active():Boolean;
    public static function updateProperties():void;
}
```

All'interno del corpo di una classe è possibile definire anche uno spazio dei nomi. Nell'esempio seguente è illustrato come uno spazio dei nomi può essere definito all'interno del corpo di una classe e utilizzato come attributo di un metodo di tale classe:

```
public class SampleClass
{
    public namespace sampleNamespace;
    sampleNamespace function doSomething():void;
}
```

ActionScript 3.0 consente di inserire nel corpo di una classe non solo definizioni, ma anche istruzioni. Le istruzioni che si trovano all'interno del corpo di una classe, ma al di fuori della definizione di un metodo, vengono eseguite una sola

volta, vale a dire nel momento in cui la definizione della classe viene rilevata la prima volta e l'oggetto classe associato viene creato.

L'esempio seguente include una chiamata a una funzione esterna, `hello()`, e un'istruzione `trace` che consente di produrre un messaggio di conferma quando la classe viene definita:

```
function hello():String
{
    trace("hola");
}
class SampleClass
{
    hello();
    trace("class created"); }
 // output when class is created
hola
class created
```

A differenza di quanto avveniva nelle precedenti versioni di ActionScript, in ActionScript 3.0 è consentito definire una proprietà statica e una proprietà di istanza con lo stesso nome all'interno di un medesimo corpo di classe. Ad esempio, il codice seguente indica una variabile statica denominata `message` e una variabile di istanza con lo stesso nome:

```
class StaticTest
{
    static var message:String = "static variable";
    var message:String = "instance variable";
}
// In your script
var myST:StaticTest = new StaticTest();
trace(StaticTest.message); // output: static variable
trace(myST.message); // output: instance variable
```

2.5 Attributi delle proprietà di classe

In relazione al modello a oggetti di ActionScript, il termine *proprietà* si riferisce a un qualsiasi membro di un una classe, incluse variabili, costanti e metodi. Nella Guida di riferimento del linguaggio e dei componenti ActionScript 3.0 il termine viene invece utilizzato in senso più ristretto per indicare unicamente membri di classi corrispondenti a variabili o definiti mediante il metodo getter o setter.

In ActionScript 3.0 è disponibile una serie di attributi utilizzabili con qualsiasi proprietà di classe. La tabella seguente (Tabella 1) riporta tali attributi.

Attributo	Definizione
`internal` (valore predefinito)	Visibile ai riferimenti che si trovano all'interno dello stesso pacchetto.
`private`	Visibile ai riferimenti che si trovano all'interno della stessa classe.
`protected`	Visibile ai riferimenti che si trovano all'interno della stessa classe e delle classi derivate.
`public`	Visibile a tutti i riferimenti.
`static`	Specifica che una proprietà appartiene alla classe e non alle sue istanze.
`UserDefinedNamespace`	Nome dello spazio dei nomi personalizzato definito dall'utente.

Tabella 1 - Attributi della proprietà classe

2.5.1 Attributi dello spazio dei nomi per il controllo dell'accesso

ActionScript 3.0 prevede quattro attributi speciali che controllano l'accesso alle proprietà definite all'interno di una classe: *`public`*, *`private`*, *`protected`* e *`internal`*.

L'attributo `public`[6] rende una proprietà visibile in qualsiasi punto dello script. Ad

[6] Come si potrà notare nel Capitolo 5, studiando il codice di implementazione della libreria digitale, si è cercato di attenersi il più possibile all'incapsulamento dei dati, per cui la maggior parte delle proprietà di ogni singola classe risulta essere `private` e l'accesso a tali proprietà è

esempio, per rendere un metodo disponibile al codice che si trova al di fuori del pacchetto, è necessario dichiarare il metodo con l'attributo `public`. Ciò vale per qualsiasi proprietà che sia stata dichiarata utilizzando le parole chiave `var`,`const` o `function`.

L'attributo `private` rende una proprietà visibile unicamente ai chiamanti entro la classe che definisce le proprietà. Tale comportamento si differenzia da quello dell'attributo `private` di ActionScript 2.0, che consentiva a una sottoclasse di accedere a una proprietà privata di una superclasse. Un'altra significativa modifica nel comportamento ha a che fare con l'accesso in fase di runtime. In ActionScript 2.0 la parola chiave `private` impediva l'accesso solo in fase di compilazione e poteva essere facilmente evitata in fase di runtime. In ActionScript 3.0 ciò non è più possibile. Le proprietà contrassegnate come `private` non risultano disponibili né in fase di compilazione né in fase di runtime.

Ad esempio, il codice seguente crea una semplice classe denominata PrivateExample con una variabile private, quindi tenta di accedere alla variabile privata dall'esterno della classe. In ActionScript 2.0, l'accesso in fase di compilazione non era consentito, ma tale divieto veniva facilmente aggirato mediante l'impiego dell'operatore di accesso alle proprietà (`[]`), che esegue una ricerca delle proprietà in fase di runtime, anziché in fase di compilazione.

Solo in fase di compilazione e poteva essere facilmente evitata in fase di runtime. In ActionScript 3.0 ciò non è più possibile. Le proprietà contrassegnate come `private` non risultano disponibili né in fase di compilazione né in fase di runtime.

```
Class PrivateExample
{
    private var privVar:String = “private variable”;
}

var myExample:PrivateExample = new PrivateExample();
```

---

garantito all'esterno da metodi `public getter` and `setter`.

```
trace(myExample.privVar);// compile-time error in strict mode
trace(myExample[“privVar”]); // ActionScript 2.0 allows access, but in
ActionScript 3.0, this is a run-time error.
```

In ActionScript 3.0, il tentativo di accedere a una proprietà privata mediante l'operatore punto (`myExample.privVar`) provoca un errore di compilazione, se ci si trova in modalità rigorosa. In caso contrario, l'errore viene riportato in fase di run-time, esattamente come accade quando si usa l'operatore di accesso alle proprietà (`myExample[“privVar”]`).

Nella tabella seguente sono riportati i risultati dei tentativi di accesso a una proprietà privata appartenente a una classe chiusa (non dinamica):

	Modalità rigorosa	**Modalità standard**
operatore punto (.)	errore di compilazione	errore di runtime
operatore parentesi ([])	errore di runtime	errore di runtime

Nelle classi dichiarate con l'attributo `dynamic`, i tentativi di accesso a una variabile privata non provocano errori di runtime. Al contrario, la variabile risulta semplicemente invisibile, di conseguenza Flash Player o Adobe® AIR™ restituiscono il valore `undefined`. Un errore in fase di compilazione si verifica invece se si utilizza un operatore punto in modalità rigorosa. L'esempio seguente è uguale a quello precedente, con la sola differenza che la classe PrivateExample viene dichiarata come classe dinamica:

```
dynamic class PrivateExample
{
    private var privVar:String = “private variable”;
}

var myExample:PrivateExample = new PrivateExample();
trace(myExample.privVar);// compile-time error in strict mode
trace(myExample[“privVar”]); // output: undefined
```

Le classi dinamiche restituiscono in genere il valore `undefined`, anziché generare un errore, se un codice esterno alla classe tenta di accedere a una proprietà privata. Nella tabella seguente si evidenzia che l'errore viene generato unicamente se si utilizza l'operatore punto per accedere a una proprietà privata in modalità rigorosa:

	Modalità rigorosa	**Modalità standard**
operatore punto (.)	errore di compilazione	undefined
operatore parentesi ([])	undefined	undefined

L'attributo `protected`, nuovo in ActionScript 3.0, rende una proprietà visibile ai chiamanti della sua stessa classe o di una sottoclasse. In altre parole, una proprietà protetta è disponibile solo all'interno della propria classe o delle classi che si trovano sotto di essa nella gerarchia di ereditarietà. Ciò vale sia che la sottoclasse si trovi nello stesso pacchetto o in un pacchetto differente.

Questa funzionalità è analoga all'attributo `private` di ActionScript 2.0. Anche l'attributo `protected` di ActionScript 3.0 è simile all'attributo Java `protected` ma, a differenza della versione Java, consente l'accesso anche ai chiamanti all'interno dello stesso pacchetto. L'attributo `protected` è utile se si dispone di una variabile o di un metodo necessario alle sottoclassi, ma che si desidera tenere nascosto da codice esterno alla catena di ereditarietà.

L'attributo `internal`, nuovo in ActionScript 3.0, rende una proprietà visibile ai chiamanti che si trovano all'interno del suo stesso pacchetto. Si tratta dell'attributo predefinito del codice all'interno di un pacchetto e può essere applicato a qualsiasi proprietà che non presenta alcuno dei seguenti attributi:

- `public`
- `private`
- `protected`
- uno spazio dei nomi definito dall'utente

L'attributo `internal` è simile al controllo dell'accesso predefinito in Java, anche se in Java non esiste un nome esplicito per tale livello di accesso, che può essere ottenuto solo mediante l'omissione di un qualsiasi altro modificatore dell'accesso. L'attributo `internal` di ActionScript 3.0 consente all'utente di dimostrare esplicitamente l'intenzione di rendere una proprietà visibile unicamente ai chiamanti che si trovano all'interno del suo stesso pacchetto.

2.5.2 Attributo static

L'attributo `static`, utilizzabile con proprietà dichiarate con le parole chiave `var`, `const` o `function`, consente di associare una proprietà alla classe anziché a istanze della classe[7]. Il codice esterno alla classe deve chiamare proprietà statiche utilizzando il nome della classe anziché il nome di un'istanza.
Le proprietà statiche non vengono ereditate dalle sottoclassi, ma fanno parte della catena delle aree di validità di una sottoclasse. In altre parole, all'interno del corpo di una sottoclasse, è possibile utilizzare una variabile o un metodo senza fare riferimento alla classe in cui sono stati creati.

2.5.3 Attributi spazio dei nomi definiti dall'utente

In alternativa agli attributi di controllo dell'accesso predefiniti, è possibile creare uno spazio dei nomi personalizzato da utilizzare come attributo. Per definizione è possibile utilizzare un solo attributo spazio dei nomi e non è possibile utilizzare tale attributo in combinazione con un qualsiasi attributo di controllo dell'accesso (`public`, `private`, `protected`, `internal`)

2.6 Variabili

Le variabili possono essere dichiarate con le parole chiave **var** o **const**. I valori

[7] La libreria digitale fa ampio uso di costanti statiche per definire valori non modificabili, da utilizzare globalmente all'interno del programma. Costanti statiche vengono usate, per esempio, per definire le dimensioni delle miniature componenti le gallerie , oppure le dimensioni dell'area dell'applicazione, o ancora il numero di miniature da visualizzare in un menù in stile Mac.

delle variabili dichiarate con la parola chiave **var** possono essere modificati più volte durante l'esecuzione di uno script. Le variabili dichiarate con la parola chiave **const** sono dette ***costanti*** e ad esse possono essere assegnati valori una sola volta. Se si tenta di assegnare un nuovo valore a una costante inizializzata, si verifica un errore.

- *Variabili statiche*

Le variabili statiche vengono dichiarate utilizzando una combinazione della parola chiave `static` e dell'istruzione `var` o `const`.
Le variabili statiche che vengono associate a una classe, anziché all'istanza di una classe, sono utili per memorizzare e condividere informazioni applicabili a un'intera classe di oggetti. Ad esempio, è appropriato l'impiego di una variabile statica per registrare quante volte viene creata un'istanza di una determinata classe oppure per memorizzare il numero massimo di istanze di una classe consentito.

Nell'esempio seguente viene creata una variabile `totalCount` per registrare il numero di volte in cui viene creata l'istanza di una classe e una costante `MAX_NUM` per memorizzare il numero massimo di istanze consentite. Le variabili `totalCount` e `MAX_NUM` sono statiche, in quanto contengono valori applicabili all'intera classe e non a una sua particolare istanza.

```
class StaticVars
{
    public static var totalCount:int = 0;
    public static const MAX_NUM:uint = 16;
}
```

Il codice esterno alla classe StaticVars e alle sue sottoclassi può fare riferimento alle proprietà `totalCount` e `MAX_NUM` solo attraverso la classe stessa.

Ad esempio, il codice seguente funziona:

```
trace(StaticVars.totalCount); // output: 0
trace(StaticVars.MAX_NUM); // output: 16
```

Non è possibile accedere a variabili statiche attraverso un'istanza della classe; il codice seguente restituisce degli errori:

```
var myStaticVars:StaticVars = new StaticVars();
trace(myStaticVars.totalCount); // error
trace(myStaticVars.MAX_NUM); // error
```

Le variabili dichiarate sia con la parola chiave `static` che `const` devono essere inizializzate nel momento in cui viene dichiarata la costante, come avviene all'interno della classe StaticVars per `MAX_NUM`. Non è possibile assegnare un valore a `MAX_NUM` all'interno della funzione di costruzione o di un metodo di istanza. Il codice seguente genera un errore in quanto non è un metodo valido per inizializzare una costante statica:

```
// !! Error to initialize static constant this way
class StaticVars2
{
    public static const UNIQUESORT:uint;
    function initializeStatic():void
    {
        UNIQUESORT = 16;
     }
}
```

- *Variabili di istanza*

Le variabili di istanza includono proprietà dichiarate con le parole chiave `var` e `const`, ma senza la parola chiave `static`. Le variabili di istanza associate a istanze di classe, anziché a un'intera classe, sono utili per la memorizzazione di valori specifici di una particolare istanza. Ad esempio, la classe Array presenta una proprietà di istanza denominata `length` nella quale è memorizzato il numero di elementi di Array contenuti in una particolare istanza della classe Array.

Le variabili di istanza, dichiarate sia con la parola chiave `var` che `const`, non possono essere sovrascritte all'interno di una sottoclasse. È tuttavia possibile ottenere una funzionalità analoga sostituendo i metodi getter e setter.

2.7 Metodi

I metodi sono funzioni che fanno parte di una definizione di classe. Una volta creata un'istanza della classe, un metodo viene associato a tale istanza. A differenza delle funzioni dichiarate all'esterno delle classi, i metodi possono essere utilizzati esclusivamente dall'istanza alla quale sono associati.

I metodi vengono definiti utilizzando la parola chiave `function`. Come avviene con qualsiasi proprietà delle classi, è possibile applicare uno degli attributi delle proprietà della classe ai metodi, inclusi spazi dei nomi privati, protetti, pubblici, interni, statici o personalizzati. È possibile utilizzare un'istruzione di funzione quale la seguente:

```
public function sampleFunction():String {}
```

Oppure è possibile utilizzare una variabile alla quale viene assegnata un'espressione della funzione, come nell'esempio seguente:

```
public var sampleFunction:Function = function () {}
```

Nella maggior parte dei casi, è consigliabile utilizzare una funzione con istruzione anziché un'espressione della funzione per le ragioni seguenti:

- Le funzioni con istruzione sono più concise e più facili da leggere.
- Le funzioni con istruzione consentono di utilizzare le parole chiave `override` e `final`.
- Le funzioni con istruzione consentono di creare un legame più solido tra l'identificatore,vale a dire, il nome della funzione, e il codice, all'interno del corpo del metodo. Poiché il valore di una variabile può essere modificato mediante un'istruzione di assegnazione, il legame tra la variabile e la relativa espressione della funzione può essere sciolto in qualsiasi momento.

Nonostante sia possibile ovviare a questo problema dichiarando la variabile con la parola chiave `const` anziché `var`, tale sistema non è consigliato, in quanto rende il codice di difficile lettura e impedisce l'uso delle parole chiave `override` e `final`.

Un caso in cui è consigliabile l'uso di un'espressione della funzione è quando si decide di associare una funzione all'oggetto prototype.

2.7.1 Metodi delle funzioni di costruzione

I metodi delle funzioni di costruzione, talvolta semplicemente dette ***funzioni di costruzione***, sono funzioni che condividono lo stesso nome della classe nella quale vengono definite[8]. Qualsiasi codice incluso in un metodo di funzione di costruzione viene eseguito ogni volta che viene creata un'istanza della classe con la parola chiave `new`.

Ad esempio, il codice seguente definisce una semplice classe chiamata Example, contenente un'unica proprietà denominata `status`. Il valore iniziale della variabile `status` viene definito all'interno della funzione di costruzione.

```
class Example
{
    public var status:String;
    public function Example()
    {
        status = "initialized";
    }
}
var myExample:Example = new Example();
trace(myExample.status); // output: initialized
```

[8] Le classi costituenti il programma della libreria digitale, come si può vedere nel Capitolo 5, hanno definito in maniera esplicita il metodo costruttore anche quando non accetta parametri. Questa scelta è stata fatta per rendere il codice più leggibile.

I metodi delle funzioni di costruzione possono essere solo pubblici, tuttavia, l'impiego dell'attributo `public` è facoltativo. Nelle funzioni di costruzione non è possibile usare nessuno degli altri specificatori del controllo di accesso, incluso `private`, `protected` o `internal`. Con i metodi delle funzioni di costruzione non è inoltre possibile utilizzare spazi dei nomi definiti dall'utente.

Una funzione di costruzione può effettuare una chiamata esplicita alla funzione di costruzione della sua superclasse diretta mediante *l'istruzione* `super()`. Se la funzione di costruzione della superclasse non viene esplicitamente chiamata, il compilatore inserisce automaticamente una chiamata prima della prima istruzione nel corpo della funzione di costruzione.
Per chiamare i metodi della superclasse, è inoltre possibile utilizzare *il prefisso* `super` come riferimento alla superclasse. Se si decide di utilizzare sia `super()` che `super` nello stesso corpo della funzione di costruzione, assicurarsi di chiamare prima `super()`. In caso contrario, il riferimento `super` non funzionerà correttamente. La funzione di costruzione `super()` deve inoltre essere chiamata prima di eventuali istruzioni `throw` o `return`.
Nell'esempio che segue è illustrato cosa accade se si tenta di utilizzare il riferimento `super` prima di chiamare la funzione di costruzione `super()`. Una nuova classe, ExampleEx, estende la classe Example. La funzione di costruzione ExampleEx tenta di accedere alla variabile status definita all'interno della sua superclasse, ma lo fa prima di chiamare `super()`. L'istruzione `trace()` che si trova all'interno della funzione di costruzione ExampleEx genera il valore `null`, in quanto la variabile `status` non risulta disponibile fino all'esecuzione della funzione di costruzione `super()`.

```
class ExampleEx extends Example
{
     public function ExampleEx()
     {
          trace(super.status);
          super();
     }
}
var mySample:ExampleEx = new ExampleEx(); // output: null
```

Nonostante sia possibile utilizzare l'istruzione `return` all'interno di una funzione di costruzione, non è consentito che venga restituito un valore. In altre parole, le istruzioni `return` non devono avere espressioni o valori associati. Di conseguenza, i metodi delle funzioni di costruzione non possono restituire valori, ovvero non è possibile specificare tipi restituiti.

Se non si definisce un metodo di funzione di costruzione nella classe, il compilatore crea automaticamente una funzione di costruzione vuota. Se la classe viene estesa a un'altra classe, il compilatore includerà una chiamata `super()` nella funzione di costruzione generata.

2.7.2 Metodi statici

I metodi statici, chiamati anche ***metodi di classe***, sono metodi che vengono dichiarati con la parola chiave `static`. I metodi statici, generalmente associati a una classe anziché all'istanza di una classe, sono utili per incorporare funzionalità relative a qualcosa di diverso dallo stato di singole istanze. Poiché i metodi statici vengono associati a un'intera classe, è possibile accedervi solo attraverso la classe stessa e non attraverso un'istanza della classe.

I metodi statici sono utili per incorporare funzioni che non si limitano a modificare lo stato delle istanze di classe. In altre parole, un metodo si definisce statico se fornisce funzionalità che non intervengono direttamente sul valore di un'istanza di classe. Ad esempio, la classe Date presenta un metodo statico denominato `parse()` che converte una stringa in un numero. Tale metodo è statico in quanto non modifica una singola istanza della classe. Il metodo `parse()` prende invece una stringa che rappresenta un valore data, l'analizza e restituisce un numero in un formato compatibile con la rappresentazione interna dell'oggetto Date. Questo metodo non è un metodo di istanza, in quanto non avrebbe senso applicarlo a un'istanza della classe Date. Il metodo statico `parse()` è in contrasto con i metodi di istanza della classe Date, quale `getMonth()`.

Il metodo `getMonth()` è un metodo di istanza, in quanto opera direttamente sul valore recuperando un componente specifico, il mese, di un'istanza Date.
Poiché i metodi statici non sono associati a singole istanze, non è possibile utilizzare le parole chiave `this` o `super` all'interno del corpo di un metodo statico. Sia il riferimento `this` che `super` hanno significato solo nel contesto di un metodo di istanza.
Diversamente da quanto accade in altri linguaggi di programmazione basati sulle classi, i metodi statici in ActionScript 3.0 non vengono ereditati.[9]

2.7.3 Metodi di istanza

I metodi di istanza sono metodi dichiarati senza la parola chiave `static`. Questi metodi, che vengono associati alle istanze di una classe anziché all'intera classe, sono utili per implementare funzionalità che riguardano singole istanze di una classe. Ad esempio, la classe Array contiene un metodo di istanza denominato `sort()` che opera direttamente sulle istanze Array.
Nel corpo di un metodo di istanza, sono valide sia le variabili statiche che di istanza, il che significa che è possibile fare riferimento alle variabili definite all'interno della stessa classe mediante un semplice identificatore. Ad esempio, la classe CustomArray seguente estende la classe Array. La classe CustomArray definisce una variabile statica denominata `arrayCountTotal` che registra il numero totale di istanze della classe, una variabile di istanza denominata `arrayNumber` che registra l'ordine in cui le istanze sono state create e un metodo di istanza denominato `getPosition()` che restituisce i valori di tali variabili.

```
public class CustomArray extends Array
{
    public static var arrayCountTotal:int = 0;
    public var arrayNumber:int;
```

[9] In breve, possiamo dire che le proprietà statiche in Actionscript 3.0 non vengono ereditate dalle sottoclassi. Ciò significa che non è possibile accedere alle proprietà statiche attraverso un'istanza di una sottoclasse. Una proprietà statica è accessibile unicamente attraverso l'oggetto di classe sul quale viene definita.

```
    public function CustomArray()
    {
         arrayNumber = ++arrayCountTotal;
    }

    public function getArrayPosition():String
    {
          return ("Array " + arrayNumber + " of " + arrayCountTotal);
    }
}
```

Se il codice esterno alla classe deve fare riferimento alla variabile statica `arrayCountTotal` mediante l'oggetto di classe, utilizzando `CustomArray.arrayCountTotal`, il codice che risiede nel corpo del metodo `getPosition()` può fare riferimento direttamente alla variabile statica `arrayCountTotal`. Ciò vale anche per le variabili statiche contenute nelle superclassi.

Anche se le proprietà statiche non vengono ereditate in ActionScript 3.0, quelle presenti nelle superclassi sono nell'area di validità. Ad esempio, la classe Array presenta poche variabili statiche, una delle quali è una costante denominata `DESCENDING`. Il codice residente in una delle sottoclassi di Array può fare riferimento alla costante statica `DESCENDING` mediante un semplice identificatore.

```
public class CustomArray extends Array
{
    public function testStatic():void
    {
         trace(DESCENDING); // output: 2
    }
}
```

Il valore del riferimento `this` nel corpo di un metodo di istanza è un riferimento all'istanza alla quale il metodo è associato. Il codice seguente illustra come il riferimento `this` punti all'istanza contenente il metodo:

```
class ThisTest
{
    function thisValue():ThisTest
    {
        return this;
    }
}
var myTest:ThisTest = new ThisTest();
trace(myTest.thisValue() == myTest); // output: true
```

L'ereditarietà dei metodi di istanza può essere controllata con le parole chiave `override` e `final`. Utilizzate l'attributo `override` per ridefinire un metodo ereditato e l'attributo `final` per impedire alle sottoclassi di sostituire un metodo[10].

2.7.4 Metodi supplementari get e set

Le funzioni get e set, chiamate anche ***getter*** **e** ***setter***, consentono di rispettare i principi di programmazione dell'information hiding e dell'incapsulamento, fornendo un'interfaccia di programmazione facile da usare per le classi create. Le funzioni get e set permettono di mantenere le proprietà della classe private all'interno della classe stessa, pur consentendo agli utenti della classe di accedere a tali proprietà come se stessero accedendo a una variabile di classe anziché chiamare un metodo di classe.

Il vantaggio di questo tipo di approccio è che consente di non utilizzare le funzioni accessor tradizionali con nomi poco pratici, quali `getPropertyName()` e `setPropertyName()`. Un altro vantaggio delle funzioni getter e setter è che

[10]Come diremo meglio in seguito, sostituire un metodo significa ridefinire il comportamento di un metodo ereditato.

evitano di gestire due funzioni pubbliche per ciascuna proprietà che consente l'accesso sia di lettura che di scrittura.

Nell'esempio seguente, la classe GetSet include le funzioni accessor get e set denominate `publicAccess()` che consentono l'accesso alla variabile privata denominata `privateProperty`:

```
class GetSet
{
     private var privateProperty:String;

     public function get publicAccess():String
     {
          return privateProperty;
     }

     public function set publicAccess(setValue:String):void
     {
          privateProperty = setValue;
     }
}
```

Se si tenta di accedere alla proprietà `privateProperty` direttamente, si verifica l'errore seguente:

```
var myGetSet:GetSet = new GetSet();

trace(myGetSet.privateProperty); // error occurs
```

Gli utenti della classe GetSet utilizzeranno invece qualcosa che appare come una proprietà denominata `publicAccess`, ma che in realtà è una coppia di funzioni get e set che operano sulla proprietà privata chiamata `privateProperty`.

Nell'esempio seguente viene creata un'istanza della classe GetSet, quindi viene impostato il valore di `privateProperty` mediante l'accessor pubblico denominato `publicAccess`:

```
var myGetSet:GetSet = new GetSet();

trace(myGetSet.publicAccess); // output: null

myGetSet.publicAccess = "hello";

trace(myGetSet.publicAccess); // output: hello
```

Le funzioni getter e setter consentono inoltre di sostituire le proprietà ereditate da una superclasse, cosa non possibile se si utilizzano normali variabili membro di classe. Le variabili membro di classe dichiarate con la parola chiave `var` non possono essere sostituite in una sottoclasse. Le proprietà create con le funzioni getter e setter invece non presentano questo limite. È possibile utilizzare l'attributo `override` sulle funzioni getter e setter ereditate da una superclasse.

2.7.5 Metodi vincolati

Un metodo vincolato, o ***chiusura di un metodo***, è semplicemente un metodo estratto dalla propria istanza. Tra gli esempi di metodi vincolati vi sono metodi passati come argomenti a una funzione o restituiti come valori da una funzione. Tra le novità di ActionScript 3.0 vi è un metodo vincolato simile a una chiusura di funzione, in quanto in grado di conservare il proprio ambiente lessicale anche se estratto dall'istanza a cui è associato. Tuttavia, la principale differenza tra un metodo vincolato e una chiusura di funzione è che il riferimento `this` del metodo vincolato resta collegato, o vincolato, all'istanza che implementa il metodo. In altre parole, il riferimento `this` in un metodo vincolato punta sempre all'oggetto originale che ha implementato il metodo. Nelle funzioni di chiusura, il riferimento `this` è generico, vale a dire che punta a qualsiasi oggetto associato alla funzione nel momento in cui viene chiamato.

Una corretta comprensione dei metodi vincolati è importante se si utilizza la parola chiave `this`. Tenete presente che la parola chiave `this` fornisce un riferimento a un oggetto principale del metodo. Per la maggior parte dei programmatori ActionScript la parola chiave `this` fa sempre riferimento all'oggetto o alla classe che contiene la definizione di un metodo. Senza i metodi vincolati, tuttavia, ciò non sarebbe sempre possibile. Nelle versioni precedenti di ActionScript, ad esempio, il riferimento `this` non era sempre riferito all'istanza che aveva implementato il metodo. Se in ActionScript 2.0 i metodi vengono estratti da un'istanza, non solo il riferimento `this` resta vincolato all'istanza

originale, ma le variabili di membro e i metodi della classe dell'istanza non risultano disponibili. Il problema non esiste in ActionScript 3.0, in quanto i metodi vincolati vengono automaticamente creati quando un metodo viene passato come parametro. I metodi vincolati garantiscono che la parola chiave `this` faccia sempre riferimento all'oggetto o alla classe nella quale il metodo viene definito.

Il codice seguente definisce una classe denominata ThisTest, contenente un metodo chiamato `foo()` che definisce a sua volta il metodo vincolato e un metodo `bar()` che restituisce il metodo vincolato. Il codice esterno alla classe crea un'istanza della classe ThisTest, chiama il metodo `bar()` e memorizza il valore restituito in una variabile denominata `myFunc`.

```
class ThisTest
{
    private var num:Number = 3;
    function foo():void // bound method defined
    {
        trace("foo's this: " + this);
        trace("num: " + num);
    }
    function bar():Function
    {
        return foo; // bound method returned
    }
}

var myTest:ThisTest = new ThisTest();
var myFunc:Function = myTest.bar();
trace(this); // output: [object global]
myFunc();
/* output:
foo's this: [object ThisTest]
output: num: 3 */
```

Le ultime due righe del codice mostrano che il riferimento `this` del metodo vincolato `foo()` continua a puntare a un'istanza della classe ThisTest, anche se il riferimento `this` che si trova nella linea precedente punta all'oggetto global. Inoltre, il metodo vincolato memorizzato nella variabile `myFunc` può ancora accedere alle variabili di membro della classe ThisTest. Se lo stesso codice viene

eseguito in ActionScript 2.0, il riferimento `this` corrisponderebbe e la variabile `num` risulterebbe `undefined`.

Un'area in cui l'inserimento dei metodi vincolati è più evidente è quella dei gestori di eventi, in quanto il metodo `addEventListener()` richiede il passaggio di una funzione o di un metodo come argomento.

2. 8 Enumerazioni con classi

Le *enumerazioni* sono tipi di dati personalizzati creati per incapsulare un piccolo gruppo di valori. ActionScript 3.0 non supporta una funzionalità di enumerazione specifica, a differenza di C++, che presenta la parola chiave **enum**, o di Java, che ha un'interfaccia di enumerazione propria.

È tuttavia possibile creare enumerazioni mediante le classi e le costanti statiche. Ad esempio, la classe in ActionScript 3.0 impiega un'enumerazione denominata PrintJobOrientation per memorizzare il set di valori che comprende **"landscape"** e **"portrait"**, come illustrato nel codice seguente:

```
public final class PrintJobOrientation
{
    public static const LANDSCAPE:String = "landscape";
    public static const PORTRAIT:String = "portrait";
}
```

Per convenzione, una classe di enumerazione viene dichiarata con l'attributo `final` in quanto non vi è necessità di estendere la classe. La classe comprende solo membri statici, di conseguenza non si potranno creare istanze della classe. Al contrario, è possibile accedere ai valori di enumerazione direttamente tramite l'oggetto di classe, come illustrato nel seguente estratto di codice:

```
var pj:PrintJob = new PrintJob();
if(pj.start())
{
    if (pj.orientation == PrintJobOrientation.PORTRAIT)
    {
```

```
        ...
    }
    ...
}
```

Tutte le classi di enumerazione in ActionScript 3.0 contengono solo variabili di tipo String, int o uint. Il vantaggio di utilizzare enumerazioni anziché stringhe di caratteri o valori numerici è che gli errori tipografici sono più facilmente rilevabili nelle enumerazioni. Se si digita in modo errato il nome di un'enumerazione, il compilatore di ActionScript genera un errore. Se si usano valori letterali, il compilatore non rileva gli errori ortografici o l'uso di un numero errato. Nell'esempio precedente, se il nome della costante di enumerazione non è corretto, il compilatore genera un errore, come illustrato nell'estratto seguente:

```
    if (pj.orientation == PrintJobOrientation.PORTRAI) // compiler
error
```

Tuttavia, il compilatore non genera un errore se la stringa di caratteri digitata contiene un errore ortografico, come nel caso seguente:

```
    if (pj.orientation == "portrai") // no compiler error
```

Una seconda tecnica per la creazione di enumerazioni prevede anch'essa la creazione di una classe separata con proprietà statiche per l'enumerazione. Ciò che differenzia questa tecnica, tuttavia, è che le proprietà statiche contengono un'istanza della classe anziché una stringa o un valore intero. Ad esempio, il seguente codice consente di creare una classe di enumerazione per i giorni della settimana:

```
public final class Day

{
    public static const MONDAY:Day = new Day();
    public static const TUESDAY:Day = new Day();
    public static const WEDNESDAY:Day = new Day();
    public static const THURSDAY:Day = new Day();
    public static const FRIDAY:Day = new Day();
    public static const SATURDAY:Day = new Day();
    public static const SUNDAY:Day = new Day();
}
```

Questa tecnica non viene utilizzata da ActionScript 3.0, ma è impiegata da numerosi sviluppatori che ne apprezzano in particolare la funzionalità di verifica tipi ottimizzata. Ad esempio, un metodo in grado di restituire un valore di enumerazione può limitare il valore restituito al tipo di dati dell'enumerazione. Il codice seguente mostra una funzione in grado di restituire non solo un giorno della settimana, ma anche una chiamata a una funzione che impiega il tipo dell'enumerazione come annotazione di tipo:

```
function getDay():Day
{
     var date:Date = new Date();
     var retDay:Day;
     switch (date.day)
     {
          case 0:
               retDay = Day.MONDAY;
               break;
          case 1:
               retDay = Day.TUESDAY;
               break;
          case 2:
               retDay = Day.WEDNESDAY;
               break;
          case 3:
               retDay = Day.THURSDAY;
               break;
          case 4:
               retDay = Day.FRIDAY;
               break;
          case 5:
               retDay = Day.SATURDAY;
               break;
          case 6:
               retDay = Day.SUNDAY;
               break;
     }
     return retDay;
}

var dayOfWeek:Day = getDay();
```

È inoltre possibile ottimizzare la classe Day in modo che a ogni giorno della settimana venga associato un numero intero e fornire un metodo `toString()` che restituisca una rappresentazione del giorno sotto forma di stringa.

2.9 Classi delle risorse incorporate

In ActionScript 3.0 vengono impiegate classi speciali, chiamate ***classi delle risorse incorporate*****,** per rappresentare risorse incorporate. Una ***risorsa incorporata*** è una risorsa, quale un suono, un'immagine o un carattere, che viene inclusa in un file SWF in fase di compilazione. L'incorporamento, anziché il caricamento dinamico, di una risorsa ne garantisce la disponibilità in fase di runtime, ma al costo di un incremento delle dimensioni del file SWF.

- *Uso delle classi delle risorse incorporate in Flash*

Per incorporare una risorsa, in primo luogo inserirla nella libreria di un file FLA. In secondo luogo, utilizzare la finestra di dialogo Proprietà del concatenamento della risorsa per specificare il nome della classe della risorsa incorporata. Se una classe con questo nome non esiste nel percorso di classe, ne viene generata una automaticamente. A questo punto, è possibile creare un'istanza della classe della risorsa incorporata e utilizzare qualsiasi proprietà e metodo definito o ereditato dalla classe. Ad esempio, il codice seguente può essere utilizzato per riprodurre l'audio incorporato collegato alla classe di una risorsa incorporata di nome PianoMusic:

```
var piano:PianoMusic = new PianoMusic();
var sndChannel:SoundChannel = piano.play();
```

2.10 Interfacce.

Un'interfaccia è una raccolta di dichiarazioni di metodi che consente a oggetti non correlati di comunicare tra loro. Ad esempio, ActionScript 3.0 definisce

l'interfaccia IEventDispatcher, che contiene dichiarazioni di metodi utilizzabili dalle classi per gestire gli oggetti evento. L'interfaccia IEventDispatcher stabilisce una modalità standard per il passaggio degli oggetti evento da un oggetto all'altro. Il codice seguente mostra la definizione dell'interfaccia IEventDispatcher:

```
public interface IEventDispatcher
{
    function addEventListener(type:String, listener:Function,
             useCapture:Boolean=false, priority:int=0,
             useWeakReference:Boolean = false):void;
    function removeEventListener(type:String, listener:Function,
             useCapture:Boolean=false):void;
    function dispatchEvent(event:Event):Boolean;
    function hasEventListener(type:String):Boolean;
    function willTrigger(type:String):Boolean;
}
```

Le interfacce si basano sulla distinzione tra l'interfaccia di un metodo e la sua implementazione. L'interfaccia di un metodo include tutte le informazioni necessarie per chiamare tale metodo, incluso il nome del metodo, tutti i suoi parametri e il tipo restituito. L'implementazione di un metodo include non solo le informazioni sull'interfaccia, ma anche le istruzioni eseguibili che attivano il comportamento del metodo. La definizione di un'interfaccia contiene soltanto interfacce di metodo e tutte le classi che implementano l'interfaccia sono responsabili della definizione delle implementazioni del metodo.

In ActionScript 3.0, la classe EventDispatcher implementa l'interfaccia IEventDispatcher mediante la definizione di tutti i metodi dell'interfaccia IEventDispatcher e l'aggiunta di corpi di metodo a ognuno dei metodi. Il codice seguente è estratto dalla definizione della classe EventDispatcher:

```
public class EventDispatcher implements IEventDispatcher
{
    function dispatchEvent(event:Event):Boolean
    {
        /* implementation statements */
    }
    ...
}
```

L'interfaccia IEventDispatcher serve da protocollo per le istanze di EventDispatcher che devono elaborare oggetti evento e passarli ad altri oggetti che hanno anch'essi implementato l'interfaccia IEventDispatcher.
Per descrivere un'interfaccia è anche possibile affermare che essa definisce un tipo di dati esattamente come fa una classe. Di conseguenza, un'interfaccia può essere usata come un'annotazione di tipo, allo stesso modo di una classe.
Come tipo di dati, l'interfaccia può inoltre essere utilizzata con operatori, quali `is` e `as`, che richiedono un tipo di dati. A differenza di quanto avviene per le classi, tuttavia, non è possibile creare istanze di un'interfaccia. Questa distinzione ha portato molti programmatori a considerare le interfacce come tipi di dati astratti e le classi come tipi di dati concreti.

2.10.1 Definizione di un'interfaccia

La struttura della definizione di un'interfaccia è simile a quella della definizione di una classe, a eccezione del fatto che l'interfaccia può contenere solo metodi e non corpi dei metodi. Le interfacce inoltre non possono includere variabili o costanti, ma possono incorporare funzioni getter e setter. *Per definire un'interfaccia, utilizzare la parola chiave* `interface`.

Ad esempio, la seguente interfaccia, IExternalizable, fa parte del pacchetto flash.utils in ActionScript 3.0. L'interfaccia IExternalizable definisce un protocollo per la serializzazione di un oggetto, vale a dire la conversione di un oggetto in un formato idoneo per la memorizzazione su un dispositivo o per la trasmissione in rete.

```
public interface IExternalizable
{
    function writeExternal(output:IDataOutput):void;
    function readExternal(input:IDataInput):void;
}
```

L'interfaccia IExternalizable viene dichiarata con il modificatore del controllo di accesso `public`. Le definizioni di interfaccia possono essere modificate

unicamente mediante gli specificatori del controllo di accesso `public` e `internal`. Le dichiarazioni dei metodi all'interno di una definizione di interfaccia non possono avere nessuno specificatore del controllo di accesso.

ActionScript 3.0 applica una convenzione in base alla quale i nomi di interfaccia iniziano con una `I` maiuscola, ma è possibile utilizzare qualsiasi identificatore valido come nome di interfaccia. Le definizioni di interfaccia vengono spesso collocate nel livello più alto di un pacchetto. Le definizioni di interfaccia non possono essere collocate all'interno di una definizione di classe o di un'altra definizione di interfaccia.
Le interfacce possono estendere altre interfacce. Ad esempio, l'interfaccia IExample seguente estende l'interfaccia IExternalizable:

```
public interface IExample extends IExternalizable
{
    function extra():void;
}
```

Tutte le classi che implementano l'interfaccia IExample devono includere implementazioni non solo del metodo `extra()`, ma anche dei metodi `writeExternal()` e `readExternal()` ereditati dall'interfaccia IExternalizable.

2.10.2 Implementazione di un'interfaccia in una classe

La classe è il solo elemento del linguaggio di ActionScript 3.0 in grado di implementare un'interfaccia. Per implementare una o più interfacce utilizzare la parola chiave `implements` nella dichiarazione della classe. Negli esempi seguenti vengono definite due interfacce, IAlpha e IBeta, e una classe, Alpha, che le implementa entrambe:

```
interface IAlpha
{
    function foo(str:String):String;
}
interface IBeta
```

```
{
    function bar():void;
}

class Alpha implements IAlpha, IBeta
{
    public function foo(param:String):String {}
    public function bar():void {}
}
```

In una classe che implementa un'interfaccia, i metodi implementati devono:

- Utilizzare l'identificatore del controllo di accesso `public`.
- Utilizzare lo stesso nome del metodo di interfaccia.
- Avere lo stesso numero di parametri, ciascuno con tipi di dati corrispondenti ai tipi di dati dei parametri del metodo di interfaccia.
- Utilizzare lo stesso tipo restituito.

  ```
  public function foo(param:String):String {}
  ```

È tuttavia possibile scegliere quale nome assegnare ai parametri dei metodi implementati. Anche se il numero dei parametri e il tipo di dati di ciascun parametro del metodo implementato devono corrispondere a quelli del metodo di interfaccia, i nomi dei parametri possono essere differenti. Nell'esempio precedente, il parametro del metodo `Alpha.foo()` è denominato `param`: .
Mentre è denominato `str` nel metodo di interfaccia `IAlpha.foo()`:

```
function foo(str:String):String;
```

Si ha un po' di flessibilità anche per quanto riguarda i valori di parametro predefiniti. Una definizione di interfaccia può includere dichiarazioni di funzioni con valori di parametro predefiniti. Un metodo che implementa una tale dichiarazione di funzione deve avere un valore di parametro predefinito che sia membro dello stesso tipo di dati del valore specificato nella definizione di interfaccia, anche se il valore vero e proprio può essere differente.

Ad esempio, il codice seguente definisce un'interfaccia che contiene un metodo con un valore di parametro predefinito 3:

```
interface IGamma
{
    function doSomething(param:int = 3):void;
}
```

La definizione di classe seguente implementa l'interfaccia Igamma, ma impiega un valore di parametro predefinito differente:

```
class Gamma implements IGamma
{
    public function doSomething(param:int = 4):void {}
}
```

Il motivo di questa flessibilità sta nel fatto che le regole di implementazione dell'interfaccia sono state specificamente studiate per garantire la compatibilità dei tipi di dati e per raggiungere tale obiettivo non è richiesta una corrispondenza dei nomi dei parametri e dei valori predefiniti.

2.11 Ereditarietà

L'ereditarietà è una forma di riutilizzo del codice che consente ai programmatori di sviluppare nuove classi basate sulle classi esistenti. Le classi esistenti vengono spesso definite ***classi base*** **o** ***superclassi*****,** mentre le nuove classi sono generalmente chiamate ***sottoclassi***.

Uno dei principali vantaggi dell'ereditarietà è che consente di riutilizzare il codice di una classe di base, senza modificare il codice esistente. Inoltre, l'ereditarietà non richiede di modificare il modo in cui le altre classi interagiscono con la classe di base. Anziché modificare una classe esistente già ampiamente testata o già in uso, l'impiego dell'ereditarietà consente di trattare tale classe come un modulo integrato da estendere con proprietà o metodi aggiuntivi. Di conseguenza, la parola chiave `extends` viene utilizzata per indicare che una classe eredita da un'altra classe[11].

[11] Il programma della libreria digitale fa ampio uso del concetto di ereditarietà. Nei capitoli 4 e 5 verrà evidenziato per esempio, come le classi del programma per funzionare correttamente,

L'ereditarietà consente inoltre di sfruttare i **vantaggi del *polimorfismo*** all'interno del codice. *Si definisce polimorfismo* la capacità di usare un unico nome metodo per un metodo in grado di comportarsi in modo differente se applicato a diversi tipi di dati. Un semplice esempio è costituito da una classe base denominata Shape con due sottoclassi denominate Circle e Square. La classe Shape definisce un metodo chiamato `area()` che restituisce l'area della figura geometrica. Se viene implementato il polimorfismo, è possibile chiamare il metodo `area()` sugli oggetti di tipo Circle e Square e ottenere i calcoli corretti. L'ereditarietà abilita il polimorfismo consentendo alle sottoclassi di ereditare e ridefinire (*sostituire*) i metodi della classe base. Nell'esempio seguente, il metodo `area()` viene ridefinito dalle classi Circle e Square:

```
class Shape
{
     public function area():Number
     {
          return NaN;
     }
}

class Circle extends Shape
{
     private var radius:Number = 1;
     override public function area():Number
     {
          return (Math.PI * (radius * radius));
     }
}

class Square extends Shape
{
     private var side:Number = 1;
     override public function area():Number
     {
```

devono estendere il framework di Flash. Non solo, ma alcune classi costituenti il programma, ne estendono altre per riutilizzarne logiche. Ad esempio, la classe Thumbnail, le cui istanze ognuna rappresenta una miniatura, estende la classe BitmapLoader, le cui istanze invece rappresentano i documenti, per utilizzarne le logiche relative al caricamento di immagini bitmap esterne.

```
        return (side * side);
    }
}

var cir:Circle = new Circle();
trace(cir.area()); // output: 3.141592653589793
var sq:Square = new Square();
trace(sq.area()); // output: 1
```

Poiché ogni classe definisce un tipo di dati, l'uso dell'ereditarietà crea una speciale relazione tra la classe base e la classe che la estende. Una sottoclasse possiede sempre tutte le proprietà della sua classe base, di conseguenza una qualsiasi istanza di una sottoclasse può sempre essere sostituita con un'istanza della classe base. Ad esempio, se un metodo definisce un parametro di tipo Shape, è consentito il passaggio di un argomento di tipo Circle, in quanto Circle è un'estensione di Shape, come indicato di seguito:

```
function draw(shapeToDraw:Shape) {}

var myCircle:Circle = new Circle();
draw(myCircle);
```

2.11.1 Proprietà di istanza ed ereditarietà

Una proprietà di istanza, sia che venga dichiarata con la parola chiave `function`, `var` o `const`, viene ereditata da tutte le sottoclassi a condizione che essa non sia stata dichiarata con l'attributo `private` nella classe base.

Ad esempio, la classe Event in ActionScript 3.0 ha un numero di sottoclassi che ereditano proprietà comuni a tutti gli oggetti evento.
Per alcuni tipi di eventi, la classe Event contiene tutte le proprietà necessarie per la definizione dell'evento. Questi tipi di eventi non richiedono proprietà di istanza oltre a quelle definite nella classe Event. Esempi di tali eventi sono l'evento `complete`, che si verifica quando i dati vengono caricati correttamente, e l'evento `connect`, che si verifica quando viene stabilita una connessione di rete.

L'esempio seguente è un estratto della classe Event che illustra alcune delle proprietà e dei metodi ereditati dalle sottoclassi. Poiché vengono ereditate, le proprietà sono accessibili da una qualsiasi istanza di una sottoclasse.

```
public class Event
{
     public function get type():String;
     public function get bubbles():Boolean;
     ...

     public function stopPropagation():void {}
     public function stopImmediatePropagation():void {}
     public function preventDefault():void {}
     public function isDefaultPrevented():Boolean {}
     ...
}
```

Vi sono poi altri tipi di eventi che richiedono proprietà univoche non disponibili nella classe Event. Tali eventi vengono definiti utilizzando le sottoclassi della classe Event, in modo da consentire l'aggiunta di nuove proprietà alle proprietà già definite nella classe Event. Un esempio di tali sottoclassi è la classe MouseEvent, che aggiunge proprietà univoche per gli eventi associate ai movimenti o ai clic del mouse, quali gli eventi `mouseMove` e `click`.

L'esempio seguente è un estratto della classe MouseEvent che mostra la definizione di proprietà esistenti all'interno della sottoclasse, ma non nella classe base:

```
public class MouseEvent extends Event
{
     public static const CLICK:String= "click";
     public static const MOUSE_MOVE:String = "mouseMove";
     ...

     public function get stageX():Number {}
     public function get stageY():Number {}
     ...
}
```

- *Specificatori del controllo di accesso ed ereditarietà*

Se viene dichiarata con la parola chiave `public`, una proprietà risulta visibile al codice ovunque si trovi. Ciò significa che la parola chiave `public`, a differenza delle parole chiave `private`, `protected` e `internal`, non pone limiti all'ereditarietà delle proprietà.

Se viene dichiarata con la parola chiave `private`, una proprietà risulterà visibile solo nella classe che la definisce e non verrà ereditata da alcuna delle sottoclassi. Ciò non accadeva nelle versioni precedenti di ActionScript, dove la parola chiave `private` si comportava in modo simile alla parola chiave `protected` di ActionScript 3.0.

La parola chiave `protected` indica che una proprietà è visibile non solo all'interno della classe che la definisce, ma anche in tutte le sue sottoclassi. A differenza della parola chiave `protected` del linguaggio di programmazione Java, la parola chiave `protected` di ActionScript 3.0 non rende una proprietà visibile a tutte le altre classi dello stesso pacchetto. In ActionScript 3.0, solo le sottoclassi possono accedere a una proprietà dichiarata con la parola chiave `protected`. Inoltre, una proprietà protetta è visibile a una sottoclasse, sia che questa si trovi nello stesso pacchetto della classe base che in un pacchetto differente.

Per limitare la visibilità di una proprietà al pacchetto nel quale essa è stata definita, utilizzare la parola chiave `internal` oppure non utilizzare alcuno specificatore del controllo di accesso. Lo specificatore del controllo di accesso `internal` è lo specificatore predefinito che viene applicato quando non ne viene specificato alcuno. Se viene contrassegnata come `internal`, una proprietà verrà ereditata unicamente dalle sottoclassi che risiedono nello stesso pacchetto.

L'esempio seguente mostra come ogni specificatore del controllo di accesso può modificare l'ereditarietà nell'ambito dei pacchetti. Il codice seguente definisce una classe di applicazione principale chiamata AccessControl e due altre classi chiamate Base ed Extender. La classe Base si trova in un pacchetto denominato

foo, mentre la classe Extender, che è una sottoclasse della classe Base, si trova in un pacchetto chiamato bar. La classe AccessControl importa unicamente la classe Extender e crea un'istanza di Extender che tenta di accedere a una variabile chiamata `str` definita nella classe Base. La variabile `str` è dichiarata come `public`, di conseguenza il codice viene compilato ed eseguito come nell'estratto seguente:

```
// Base.as in a folder named foo
package foo
{
     public class Base
     {
          public var str:String = "hello"; // change public on this line
     }
}

// Extender.as in a folder named bar
package bar
{
     import foo.Base;
     public class Extender extends Base
     {
          public function getString():String {
               return str;
          }
     }
}
// main application class in file named AccessControl.as
package
{
     import flash.display.MovieClip;
     import bar.Extender;
     public class AccessControl extends MovieClip
     {
          public function AccessControl()
          {
               var myExt:Extender = new Extender();
               trace(myExt.str);// error if str is not public
               trace(myExt.getString()); // error if str is private or
internal
          }
     }
```

```
}
```

Per vedere come gli altri specificatori del controllo di accesso possono modificare la compilazione e l'esecuzione dell'esempio precedente, impostate lo specificatore del controllo di accesso alla variabile `str` su `private`, `protected` o `internal`, dopo avere eliminato o escluso mediante commento la seguente riga dalla classe `AccessControl`:

```
trace(myExt.str);// error if str is not public
```

- *Sostituzione delle variabili non consentita*

Le proprietà dichiarate con le parole chiave `var` o `const` vengono ereditate, ma non possono essere sostituite. Sostituire una proprietà significa ridefinirla in una sottoclasse. Il solo tipo di proprietà che è possibile sostituire sono i metodi, vale a dire le proprietà dichiarate con la parola chiave `function`. Nonostante non sia possibile sostituire una variabile di istanza, si può ottenere un risultato simile mediante la creazione di metodi getter e setter per la variabile di istanza, quindi sostituire tali metodi.

2.11.2 Sostituzione di metodi

Sostituire un metodo significa ridefinire il comportamento di un metodo ereditato. I metodi statici non vengono ereditati e non possono essere sostituiti. I metodi di istanza, tuttavia, vengono ereditati dalle sottoclassi e possono essere sostituiti a condizione che vengano soddisfatti i seguenti criteri:

- Il metodo di istanza non deve essere dichiarato con la parola chiave `final` nella classe base. Se utilizzata con un metodo di istanza, la parola chiave `final` indica l'intenzione del programmatore di impedire alle sottoclassi di sostituire il metodo.
- Il metodo di istanza non deve essere dichiarato con lo specificatore del controllo di accesso `private` nella classe base. Se un metodo viene contrassegnato come `private` nella classe base, non è necessario utilizzare la parola chiave `override` per la definizione di un metodo con nome identico

nella sottoclasse, in quanto il metodo della classe base non risulterà visibile alla sottoclasse.

Per sostituire un metodo di istanza che soddisfi i criteri di cui sopra, è necessario che nella definizione del metodo all'interno della sottoclasse venga utilizzata la parola chiave `override` e che tale definizione corrisponda alla versione del metodo della superclasse, nei modi indicati di seguito:

- Il metodo di sostituzione deve avere lo stesso livello di controllo di accesso del metodo della classe base. I metodi contrassegnati come interni hanno lo stesso livello di controllo di accesso dei metodi che non presentano alcuno specificatore del controllo di accesso.
- Il metodo di sostituzione deve avere lo stesso numero di parametri del metodo della classe base.
- I parametri del metodo di sostituzione devono avere le stesse annotazioni di tipo di dati dei parametri del metodo della classe base.
- Il metodo di sostituzione deve avere lo stesso tipo restituito del metodo della classe base.

I nomi dei parametri del metodo di sostituzione, tuttavia, non devono corrispondere ai nomi dei parametri del metodo della classe base, a condizione che il numero dei parametri e il tipo di dati di ciascun parametro corrisponda.

2.11.3 Istruzione super

Quando sostituiscono un metodo, i programmatori spesso intendono aggiungere qualcosa al comportamento del metodo della superclasse da sostituire, anziché rimpiazzare completamente tale comportamento. Ciò richiede un meccanismo che consenta a un metodo in una sottoclasse di chiamare la versione di se stesso presente nella superclasse. L'istruzione `super` consente tale operazione, in quanto contiene un riferimento all'immediata superclasse. Nell'esempio seguente viene definita una classe chiamata Base contenente un metodo chiamato `thanks()` e una sottoclasse della classe Base denominata Extender che sostituisce il metodo

`thanks()`. Il metodo `Extender.thanks()` impiega l'istruzione `super` per chiamare `Base.thanks()`.

```
package {
     import flash.display.MovieClip;
     public class SuperExample extends MovieClip
     {
          public function SuperExample()
          {
               var myExt:Extender = new Extender()
               trace(myExt.thanks()); // output: Mahalo nui loa
          }
     }
}

class Base {
     public function thanks():String
     {
          return "Mahalo";
     }
}

class Extender extends Base
{
     override public function thanks():String
     {
          return super.thanks() + " nui loa";
     }
}
```

2.11.4 Sostituzione del getter e setter

Nonostante non sia possibile sostituire le variabili definite in una superclasse, è invece possibile sostituire le funzioni getter e setter.

Ad esempio, il codice seguente consente di sostituire una funzione getter denominata `currentLabel` definita nella classe MovieClip in ActionScript 3.0:

```
package
{
     import flash.display.MovieClip;
```

```
    public class OverrideExample extends MovieClip
    {
        public function OverrideExample()
        {
            trace(currentLabel)
        }
        override public function get currentLabel():String
        {
            var str:String = "Override: ";
            str += super.currentLabel;
            return str;
        }
    }
}
```

Il risultato dell'istruzione `trace()` nella funzione di costruzione della classe OverrideExample è `Override: null`, in cui è mostrato come l'esempio sia riuscito a sostituire la proprietà `currentLabel` ereditata.

2.11.5 Proprietà statiche non ereditate

Le proprietà statiche non vengono ereditate dalle sottoclassi. Ciò significa che non è possibile accedere alle proprietà statiche attraverso un'istanza di una sottoclasse. Una proprietà statica è accessibile unicamente attraverso l'oggetto di classe sul quale viene definita. Ad esempio, il codice seguente definisce una classe base chiamata Base e una sottoclasse che la estende denominata Extender. Nella classe Base viene definita una variabile statica chiamata `test`. Il codice riportato nell'estratto seguente non viene compilato in modalità rigorosa e genera un errore di runtime in modalità standard.

```
package {
    import flash.display.MovieClip;
    public class StaticExample extends MovieClip
    {
        public function StaticExample()
        {
            var myExt:Extender = new Extender();
            trace(myExt.test);// error
```

```
        }
    }
}

class Base {
    public static var test:String = "static";
}

class Extender extends Base { }
```

Il solo modo per accedere alla variabile statica `test` è mediante l'oggetto di classe, come illustrato nel codice seguente:

```
Base.test;
```

È tuttavia consentito definire una proprietà di istanza con lo stesso nome della proprietà statica. Tale proprietà di istanza può essere definita nella stessa classe della proprietà statica o in una sua sottoclasse. Ad esempio, la classe Base dell'esempio precedente potrebbe avere una proprietà di istanza denominata `test`. Il codice seguente viene compilato ed eseguito perché la proprietà di istanza viene ereditata dalla classe Extender. Il codice verrebbe compilato ed eseguito anche se la definizione della variabile di istanza test venisse spostata, ma non copiata, nella classe Extender.

```
package
{
    import flash.display.MovieClip;
    public class StaticExample extends MovieClip
    {
        public function StaticExample()
        {
            var myExt:Extender = new Extender();
            trace(myExt.test);// output: instance
        }
    }
}

class Base
{
    public static var test:String = "static";
```

```
    public var test:String = "instance";
}


class Extender extends Base {}
```

2.11.6 Proprietà statiche e catena dell'aree di validità

Nonostante non vengano ereditate, le proprietà statiche rientrano in una catena delle aree di validità della classe che definisce tali proprietà e tutte le sottoclassi della classe. Di conseguenza, le proprietà statiche vengono dette ***nell'area di validità* della classe** nella quale vengono definite e di tutte le sue sottoclassi. Ciò significa che una proprietà statica risulta direttamente accessibile all'interno del corpo della classe che definisce la proprietà statica e tutte le sottoclassi della classe.

Nell'esempio che segue vengono modificate le classi definite nell'esempio precedente, al fine di mostrare come la variabile statica `test` definita nella classe Base si trovi nell'area di validità della classe Extender. In altre parole, la classe Extender può accedere alla variabile statica `test` senza aggiungervi un prefisso corrispondente al nome della classe che definisce `test`.

```
package
{
    import flash.display.MovieClip;
    public class StaticExample extends MovieClip
    {
        public function StaticExample()
        {
            var myExt:Extender = new Extender();
        }
    }
}

class Base {
    public static var test:String = "static";
}

class Extender extends Base
{
```

```
    public function Extender()
    {
        trace(test); // output: static
    }

}
```

Se viene definita una proprietà di istanza che impiega lo stesso nome di una proprietà statica presente nella stessa classe o in una superclasse, la proprietà di istanza ha una precedenza maggiore nella catena delle aree di validità. La proprietà di istanza *prevarica* la proprietà statica, ovvero il valore della proprietà di istanza viene utilizzato al posto del valore della proprietà statica. Ad esempio, il codice seguente mostra che, se la classe Extender definisce una variabile di istanza denominata `test`, l'istruzione `trace()` impiega il valore della variabile di istanza anziché quello della variabile statica.

```
package
{
    import flash.display.MovieClip;
    public class StaticExample extends MovieClip
    {
        public function StaticExample()
        {
            var myExt:Extender = new Extender();
        }
    }
}

class Base
{
    public static var test:String = "static";
}

class Extender extends Base
{
    public var test:String = "instance";
    public function Extender()
    {
        trace(test); // output: instance
    }
}
```

Questo capitolo ha permesso al lettore di acquisire familiarità con il linguaggio utilizzato per programmare la libreria digitale , oggetto del mio progetto.
A questo punto il capitolo 5, dove si evidenzia l'implementazione del programma, risulta più facilmente comprensibile. Così come risultano essere più chiare le scelte progettuali espresse nel capitolo 4.

Capitolo 3

Actionscript e XML.

Nel capitolo vengono evidenziate gli aspetti di ActionScript 3.0 utili per accedere ad informazioni strutturate in liste di oggetti XML.

Il progetto della libreria digitale infatti, recupera dall'esterno i dati relativi ai documenti e corrispondenti miniature, sotto forma di dati XML. Tali dati in ActionScript 3.0, subiscono un'azione di parsing che traduce quello che inizialmente è semplice testo, in struttura XML gerarchiche. In questo modo con opportune sintassi e operatori diventa davvero semplice recuperare le informazioni di cui, di volta in volta, il programma ha bisogno per costruire ad esempio, un menù di miniature o per visualizzare il documento correntemente selezionato dall'utente.

3. 1 Operazioni con XML

ActionScript 3.0 comprende un gruppo di classi basate sulla specifica ECMAScript for XML (E4X) (ECMA-357 edizione 2) dotate di funzionalità potenti e di facile utilizzo per la gestione dei dati XML.

Grazie a E4X, è possibile sviluppare codice con dati XML a una velocità impensabile con le precedenti tecniche di programmazione. Un ulteriore vantaggio è rappresentato dal fatto che il codice generato è anche più facile da leggere.

3.2 L'approccio E4X all'elaborazione XML

La specifica ECMAScript for XML definisce una serie di classi e funzionalità per l'elaborazione dei dati XML. Tali classi e funzionalità sono definite globalmente *E4X*[1].

ActionScript 3.0 comprende le seguenti classi E4X: *XML, XMLList, QName e Namespace.*

I metodi, le proprietà e gli operatori delle classi E4X sono stati progettati con gli obiettivi seguenti:

- **semplicità**: laddove possibile, E4X rende il codice di gestione dei dati XML facile da scrivere e da capire;
- **uniformità**: i metodi e la struttura fondanti di E4X sono uniformi tra loro e conformi alle altri porzioni di ActionScript;
- **familiarità**: gli operatori per la manipolazione dei dati XML conosciuti, come l'operatore punto (.).

[1] ActionScript 2.0 comprendeva una classe XML. In ActionScript 3.0 è stata ridenominata XMLDocument, allo scopo di non creare un conflitto con la classe XML di ActionScript 2.0 che fa parte di E4X. In ActionScript 3.0, le classi precedenti - XMLDocument, XMLNode, XMLParser e XMLTag - sono state inserite nel pacchetto flash.xml per assicurare il supporto retroattivo. Le nuove classi E4X sono classi principali che per essere utilizzate non richiedono l'importazione di un pacchetto

Ecco un esempio di manipolazione di dati[2] con E4X:

```
var myXML:XML =
     <order>
          <item id='1'>
               <menuName>burger</menuName>
               <price>3.95</price>
          </item>
          <item id='2'>
               <menuName>fries</menuName>
               <price>1.45</price>
          </item>
     </order>
```

Come illustra la porzione di codice seguente, E4X comprende alcuni operatori intuitivi, come il punto (`.`) e l'identificatore di attributi (`@`) che vengono usati per accedere alle proprietà e agli attributi XML:

```
trace(myXML.item[0].menuName); // Output: burger
trace(myXML.item.(@id==2).menuName); // Output: fries
trace(myXML.item.(menuName=="burger").price); // Output: 3.95
```

Usare il metodo `appendChild()` per assegnare un nuovo nodo secondario al codice XML, come illustrato nello snippet di codice seguente:

```
var newItem:XML =
     <item id="3">
          <menuName>medium cola</menuName>
          <price>1.25</price>
     </item>
myXML.appendChild(newItem);
```

[2] Nonostante accada spesso che le applicazioni carichino i dati XML da un'origine esterna, come il servizio Web o un feed RSS, per motivi di chiarezza gli esempi presentati in questo capitolo assegnano valori letterali ai dati XML

Utilizzare gli operatori `@` e `.` non solo per leggere i dati, ma anche per assegnarli, come illustrato di seguito:

```
myXML.item[0].menuName="regular burger";
myXML.item[1].menuName="small fries";
myXML.item[2].menuName="medium cola";


myXML.item.(menuName=="regular burger").@quantity = "2";
myXML.item.(menuName=="small fries").@quantity = "2";
myXML.item.(menuName=="medium cola").@quantity = "2";
```

Usare un ciclo `for` per eseguire un'iterazione sui nodi XML, come illustrato di seguito:

```
var total:Number = 0;
for each (var property:XML in myXML.item)
{
    var q:int = Number(property.@quantity);
    var p:Number = Number(property.price);
    var itemTotal:Number = q * p;
    total += itemTotal;
    trace(q + " " + property.menuName + " $" + itemTotal.toFixed(2))
}
trace("Total: $", total.toFixed(2));
```

3.3 Oggetti XML

Un oggetto XML può rappresentare un elemento, un attributo, un commento, un'istruzione di elaborazione o un elemento di testo XML.

Gli oggetti XML sono classificati come oggetti con ***contenuto semplice*** o ***contenuto complesso***. Un oggetto XML che comprende nodi secondari è classificato come oggetto con contenuto complesso. Gli oggetti XML con contenuto semplice sono gli attributi, i commenti, le istruzioni di elaborazione e i nodi di testo.

Ad esempio, nel seguente oggetto XML è presente contenuto complesso perché comprende un commento e un'istruzione di elaborazione:

```
XML.ignoreComments = false;
XML.ignoreProcessingInstructions = false;
var x1:XML =
    <order>
        <!--This is a comment. -->
        <?PROC_INSTR sample ?>
        <item id='1'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>
```

Come illustra l'esempio seguente, ora è possibile usare i metodi `comments()` e `processingInstructions()` per creare nuovi oggetti XML, un commento e una istruzione di elaborazione:

```
var x2:XML = x1.comments()[0];
var x3:XML = x1.processingInstructions()[0];
```

3.4 Proprietà XML

La classe XML ha cinque proprietà statiche:

- Le ***proprietà*** `ignoreComments` ***e*** `ignoreProcessingInstructions` determinano se i commenti o le istruzioni di elaborazione devono essere ignorati nel momento in cui l'oggetto XML viene analizzato.
- La ***proprietà*** `ignoreWhitespace` determina se gli spazi vuoti devono essere ignorati nei tag degli elementi e nelle espressioni incorporate separate unicamente da spazi vuoti.

- Le ***proprietà*** `prettyIndent` ***e*** `prettyPrinting` vengono usate per formattare il testo restituito dai metodi `toString()` e `toXMLString()` della classe XML.

3.5 Metodi XML

I metodi seguenti permettono di eseguire operazioni con la struttura gerarchica degli oggetti XML:

- `appendChild()`
- `child()`
- `childIndex()`
- `children()`
- `descendants()`
- `elements()`
- `insertChildAfter()`
- `insertChildBefore()`
- `parent()`
- `prependChild()`

I metodi seguenti permettono di eseguire operazioni con gli attributi degli oggetti XML:

- `attribute()`
- `attributes()`

I metodi seguenti permettono di eseguire operazioni con le proprietà degli oggetti XML:

- `hasOwnProperty()`
- `propertyIsEnumerable()`
- `replace()`
- `setChildren()`

I metodi seguenti permettono di eseguire operazioni con i nomi completi e gli spazi dei nomi:

- `addNamespace()`

- `inScopeNamespaces()`
- `localName()`
- `name()`
- `namespace()`
- `namespaceDeclarations()`
- `removeNamespace()`
- `setLocalName()`
- `setName()`
- `setNamespace()`

I metodi seguenti permettono di eseguire operazioni con il contenuto XML e determinare alcuni tipi di contenuto XML:

- `comments()`
- `hasComplexContent()`
- `hasSimpleContent()`
- `nodeKind()`
- `processingInstructions()`
- `text()`

I metodi seguenti permettono di convertire in stringhe e formattare gli oggetti XML:

- `defaultSettings()`
- `setSettings()`
- `settings()`
- `normalize()`
- `toString()`
- `toXMLString()`

Rimangono alcuni metodi aggiuntivi:

- `contains()`
- `copy()`
- `valueOf()`
- `length()`

3.6 Oggetti XMLList

Un'istanza di XMLList rappresenta una raccolta arbitraria di oggetti XML e può contenere documenti XML completi, frammenti XML o i risultati di una query XML[3].

I metodi seguenti permettono di eseguire operazioni con la struttura gerarchica degli oggetti XMLList:

- `child()`
- `children()`
- `descendants()`
- `elements()`
- `parent()`

I metodi seguenti permettono di eseguire operazioni con gli attributi degli oggetti XMLList:

- `attribute()`
- `attributes()`

I metodi seguenti permettono di eseguire operazioni con le proprietà degli oggetti XMLList:

- `hasOwnProperty()`
- `propertyIsEnumerable()`

I metodi seguenti permettono di eseguire operazioni con il contenuto XML e determinare alcuni tipi di contenuto XML:

- `comments()`
- `hasComplexContent()`
- `hasSimpleContent()`
- `processingInstructions()`
- `text()`

I metodi seguenti permettono di convertire in stringhe e formattare gli oggetti XMLList:

[3] Una lista XML di particolare rilievo utilizzata nel programma ella libreria digitale è quella contenente tanti oggetti XML quanti sono i tag <immagine>, ognuno rappresentativo di uno specifico documento e relativa miniatura.

- `normalize()`
- `toString()`
- `toXMLString()`

Rimangono alcuni metodi aggiuntivi:

- `contains()`
- `copy()`
- `length()`
- `valueOf()`

Gli oggetti XMLList che contengono esattamente un solo elemento XML possono usare tutte le proprietà e i metodi della classe XML in quanto un oggetto XMLList con un solo elemento XML viene considerato al pari di un oggetto XML. Nella porzione di codice dell'esempio seguente, poiché `doc.div` è un oggetto XMLList che contiene un solo elemento, è possibile usare il metodo `appendChild()` della classe XML:

```
var doc:XML =

         <body>

              <div>

                   <p>Hello</p>

              </div>

         </body>;

doc.div.appendChild(<p>World</p>);
```

3.7 Inizializzazione delle variabili XML

Si può assegnare un letterale XML a un oggetto XML nel modo descritto di seguito:

```
var myXML:XML =
     <order>
          <item id='1'>
               <menuName>burger</menuName>
               <price>3.95</price>
          </item>
          <item id='2'>
               <menuName>fries</menuName>
               <price>1.45</price>
```

```
        </item>
    </order>
```

Come illustrato nello snippet di codice seguente, è possibile usare anche la funzione di costruzione `new` per creare un'istanza di un oggetto XML partendo da una stringa che contiene dati XML:

```
var str:String = "<order><item id='1'><menuName>burger</menuName>"
                                + "<price>3.95</price></item></order>";
var myXML:XML = new XML(str);
```

Se i dati XML della stringa non hanno una sintassi corretta (se manca, ad esempio, un tag di chiusura), viene generato un errore in fase di runtime.

Si possono anche passare dati in base al riferimento (da altre variabili) a un oggetto XML, come illustrato nell'esempio seguente:

```
var tagname:String = "item";
var attributename:String = "id";
var attributevalue:String = "5";
var content:String = "Chicken";
var x:XML = <{tagname}
{attributename}={attributevalue}>{content}</{tagname}>;
trace(x.toXMLString())
    // Output: <item id="5">Chicken</item>
```

Per caricare dei dati XML da un URL, usare la classe URLLoader, come illustrato nell'esempio seguente:

```
import flash.events.Event;
import flash.net.URLLoader;
import flash.net.URLRequest;

var externalXML:XML;
var loader:URLLoader = new URLLoader();
var request:URLRequest = new URLRequest("xmlFile.xml");
loader.load(request);
loader.addEventListener(Event.COMPLETE, onComplete);

function onComplete(event:Event):void
{
    var loader:URLLoader = event.target as URLLoader;
```

```
    if (loader != null)
    {
        externalXML = new XML(loader.data);
        trace(externalXML.toXMLString());
    }
    else
    {
        trace("loader is not a URLLoader!");
    }
}
```

Per leggere i dati XML da una connessione socket, usare la classe XMLSocket.

3.8 Assemblaggio e trasformazione di oggetti XML

Usare il metodo **prependChild()** o il metodo **appendChild()** per aggiungere una proprietà all'inizio o alla fine dell'elenco proprietà di un oggetto XML, come illustrato nell'esempio seguente:

```
var x1:XML = <p>Line 1</p>
var x2:XML = <p>Line 2</p>
var x:XML = <body></body>
x = x.appendChild(x1);
x = x.appendChild(x2);
x = x.prependChild(<p>Line 0</p>);
    // x == <body><p>Line 0</p><p>Line 1</p><p>Line 2</p></body>
```

Usare il metodo `insertChildBefore()` o il metodo `insertChildAfter()` per aggiungere una proprietà prima o dopo la proprietà specificata, nel modo seguente:

```
var x:XML =
    <body>
        <p>Paragraph 1</p>
        <p>Paragraph 2</p>
    </body>
var newNode:XML = <p>Paragraph 1.5</p>
x = x.insertChildAfter(x.p[0], newNode)
x = x.insertChildBefore(x.p[2], <p>Paragraph 1.75</p>)
```

Come illustrato dall'esempio seguente, è possibile utilizzare anche gli operatori parentesi graffe ({ e }) per passare dati in base al riferimento (da altre variabili) durante la costruzione di oggetti XML:

```
var ids:Array = [121, 122, 123];
var names:Array = [["Murphy","Pat"], ["Thibaut","Jean"],
["Smith","Vijay"]]
var x:XML = new XML("<employeeList></employeeList>");
for (var i:int = 0; i < 3; i++)
{
    var newnode:XML = new XML();
    newnode =
        <employee id={ids[i]}>
            <last>{names[i][0]}</last>
            <first>{names[i][1]}</first>
        </employee>;
    x = x.appendChild(newnode)
}
```

Si possono assegnare proprietà e attributi a un oggetto XML usando l'operatore =, come nell'esempio seguente:

```
var x:XML =
    <employee>
        <lastname>Smith</lastname>
    </employee>
x.firstname = "Jean";
x.@id = "239";
```

In questo modo x dell'oggetto XML viene impostato su:

```
<employee id="239">
    <lastname>Smith</lastname>
    <firstname>Jean</firstname>
</employee>
```

È possibile usare gli operatori + e += per concatenare oggetti XMLList:

```
var x1:XML = <a>test1</a>
var x2:XML = <b>test2</b>
var xList:XMLList = x1 + x2;
xList += <c>test3</c>
```

In questo modo `xList` dell'oggetto XMLList viene impostato su:

```
<a>test1</a>
<b>test2</b>
<c>test3</c>
```

3.9 Letture delle strutture XML

Una delle funzioni più sofisticate di XML è la sua abilità di creare strutture di dati complesse e nidificate mediante una stringa lineare di caratteri di testo. Durante il caricamento di dati di un oggetto XML, ActionScript analizza i dati e ne carica la struttura gerarchica in memoria (oppure genera un errore in fase di runtime se la sintassi dei dati XML contiene un errore).

Gli operatori e i metodi degli oggetti XML e XMLList rendono semplice la lettura della struttura dei dati XML.
Usare l'operatore punto (.) e l'operatore accessor discendente (..) per accedere alle proprietà secondarie di un oggetto XML. Osservare l'oggetto XML seguente:

```
var myXML:XML =
    <order>
        <book ISBN="0942407296">
            <title>Baking Extravagant Pastries with Kumquats</title>
            <author>
                <lastName>Contino</lastName>
                <firstName>Chuck</firstName>
            </author>
            <pageCount>238</pageCount>
        </book>
```

```
        <book ISBN="0865436401">
            <title>Emu Care and Breeding</title>
            <editor>
                <lastName>Case</lastName>
                <firstName>Justin</firstName>
            </editor>
            <pageCount>115</pageCount>
        </book>
    </order>
```

L'oggetto `myXML.book` è un oggetto XMLList che contiene proprietà secondarie dell'oggetto `myXML` denominato `book`. Sono due oggetti XML, che corrispondono alle due proprietà `book` dell'oggetto `myXML`.

L'oggetto `myXML..lastName` è un oggetto XMLList contenente qualsiasi proprietà discendente denominata `lastName`. Sono due oggetti XML, che corrispondono ai due `lastName` dell'oggetto `myXML`.

L'oggetto `myXML.book.editor.lastName` è un oggetto XMLList contenente qualsiasi proprietà discendente denominata `lastName` delle proprietà discendenti denominate `editor` delle proprietà discendenti denominate `book` dell'oggetto `myXML`: in questo caso, un oggetto XMLList contenente un solo oggetto XML (la proprietà `lastName` con il valore "`Case`").

3.9.1 Accesso ai nodi principali e secondari

Il metodo `parent()` restituisce il nodo principale di un oggetto XML. Per accedere a oggetti secondari specifici, si possono usare valori di indice ordinali di un elenco.

Ad esempio, si consideri un oggetto XML `myXML` che dispone di due proprietà secondarie denominate `book`. A ogni proprietà `book` secondaria è associato un numero di indice:

```
myXML.book[0]
myXML.book[1]
```

Per accedere a un elemento secondario di secondo livello è possibile specificare il numero di indice sia dell'elemento secondario di primo livello che di quello di secondo livello:

```
myXML.book[0].title[0]
```

Tuttavia, se `x.book[0]` dispone di una sola proprietà secondaria denominata `title`, è possibile omettere il riferimento al numero di indice, come illustrato di seguito:

```
myXML.book[0].title
```

Analogamente, se esiste un solo oggetto secondario book dell'oggetto `x`, e se quell'oggetto secondario dispone di un solo oggetto title, si possono omettere entrambi i riferimenti di indice, nel modo seguente:

```
myXML.book.title
```

È possibile usare il metodo `child()` per andare agli elementi secondari con nomi basati su una variabile o un'espressione, come illustrato nell'esempio seguente:

```
var myXML:XML =
         <order>
              <book>
                   <title>Dictionary</title>
              </book>
         </order>;
var childName:String = "book";
trace(myXML.child(childName).title) // output: Dictionary
```

3.9.2 Accesso agli attributi

Usare il simbolo `@` [4] per accedere agli attributi di un oggetto XML o XMLList, come nel codice seguente:

```
var employee:XML =
     <employee id="6401" code="233">
```

[4] @ è l'operatore identificatore di attributi

```
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.@id); // 6401
```

Per accedere a tutti gli attributi di un oggetto XML o XMLList si può usare il carattere jolly `*` combinato al simbolo `@`, come illustrato nel codice seguente:

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.@*.toXMLString());
// 6401
// 233
```

Si può usare il metodo `attribute()` o il metodo `attributes()` per accedere a un attributo specifico o a tutti gli attributi di un oggetto XML o XMLList, come illustrato nel codice seguente:

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.attribute("id")); // 6401
trace(employee.attribute("*").toXMLString());
// 6401
// 233
trace(employee.attributes().toXMLString());
// 6401
// 233
```

Si noti che per accedere agli attributi è anche possibile utilizzare la sintassi seguente, come illustrato nell'esempio seguente:

```
employee.attribute("id")
employee["@id"]
employee.@["id"]
```

Tutte queste sono alternative equivalenti a `employee.@id`. Tuttavia, la sintassi `employee.@id` è l'approccio da preferirsi.

3.9.3 Filtraggio per attributo o elemento

Per filtrare gli elementi con un nome o valore di attributo specifico è possibile usare gli operatori parentesi: `(` e `)`. Osservare l'oggetto XML seguente:

```
var x:XML =
    <employeeList>
        <employee id="347">
            <lastName>Zmed</lastName>
            <firstName>Sue</firstName>
            <position>Data analyst</position>
        </employee>
        <employee id="348">
            <lastName>McGee</lastName>
            <firstName>Chuck</firstName>
            <position>Jr. data analyst</position>
        </employee>
    </employeeList>
```

Tutte le espressioni seguenti sono valide:

- `x.employee.(lastName == "McGee")` - Questo è il secondo nodo `employee`.
- `x.employee.(lastName == "McGee").firstName` - Questa è la proprietà `firstName` del secondo nodo `employee`.
- `x.employee.(lastName == "McGee").@id` - Questo è il valore dell'attributo `id` del secondo nodo `employee`.
- `x.employee.(@id == 347)` - Il primo nodo `employee`.

- `x.employee.(@id == 347).lastName` - Questa è la proprietà `lastName` del primo nodo `employee`.
- `x.employee.(@id > 300)`: questo è un oggetto XMLList con entrambe le proprietà `employee`.
- `x.employee.(position.toString().search("analyst") > -1)`: questo è un oggetto XMLList con entrambe le proprietà `position`.

Se si tenta di applicare un filtro usando attributi o elementi inesistenti, Flash® Player e Adobe® AIR™ generano un'eccezione. Ad esempio, l'ultima riga del codice seguente genera un errore perché non esiste un attributo `id` nel secondo elemento `p`:

```
var doc:XML =
              <body>
                   <p id='123'>Hello, <b>Bob</b>.</p>
                   <p>Hello.</p>
              </body>;
trace(doc.p.(@id == '123'));
```

Analogamente, l'ultima riga del codice seguente genera un errore perché non esiste una proprietà `b` del secondo elemento `p`:

```
var doc:XML =
              <body>
                   <p id='123'>Hello, <b>Bob</b>.</p>
                   <p>Hello.</p>
              </body>;
trace(doc.p.(b == 'Bob'));
```

Per evitare questo tipo di errori, è possibile identificare le proprietà per cui esistono attributi o elementi corrispondenti usando i metodi `attribute()` e `elements()`, come nel codice seguente:

```
var doc:XML =
              <body>
                   <p id='123'>Hello, <b>Bob</b>.</p>
                   <p>Hello.</p>
              </body>;
```

```
trace(doc.p.(attribute('id') == '123'));
trace(doc.p.(elements('b') == 'Bob'));
```

Inoltre, è possibile utilizzare anche il metodo `hasOwnProperty()`, come nel codice seguente:

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(hasOwnProperty('@id') && @id == '123'));
trace(doc.p.(hasOwnProperty('b') && b == 'Bob'));
```

3.10 Uso delle istruzioni for..in e for each..in

ActionScript 3.0 comprende l'istruzione `for..in` e l'istruzione `for each..in` per l'iterazione negli oggetti XMLList.

Osservare, ad esempio, il seguente oggetto XML, `myXML`, e l'oggetto XMLList, `myXML.item`. L'oggetto XMLList, `myXML.item`, è composto dai due nodi `item` dell'oggetto XML.

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2' quantity='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>;
```

L'istruzione `for..in` permette di eseguire iterazioni su una serie di proprietà di un XMLList:

```
var total:Number = 0;
for (var pname:String in myXML.item)
{
    total += myXML.item.@quantity[pname] * myXML.item.price[pname];
}
```

L'istruzione `for each..in` permette di eseguire iterazioni sulle proprietà di XMLList:

```
var total2:Number = 0;
for each (var prop:XML in myXML.item)
{
    total2 += prop.@quantity * prop.price;
}
```

3.11 Uso degli spazi dei nomi XML

Gli spazi dei nomi di un oggetto (o documento) XML indicano il tipo di dati che l'oggetto contiene. Ad esempio, per inviare e consegnare dati XML a un servizio Web che usa il protocollo SOAP, si dichiara lo spazio dei nomi nel tag di apertura di XML:

```
var message:XML =
    <soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <soap:Body xmlns:w="http://www.test.com/weather/">
            <w:getWeatherResponse>
                <w:tempurature >78</w:tempurature>
            </w:getWeatherResponse>
        </soap:Body>
    </soap:Envelope>;
```

Lo spazio dei nomi è caratterizzato da un prefisso, `soap`, e da un URI che definisce lo spazio dei nomi, `http://schemas.xmlsoap.org/soap/envelope/`.

ActionScript 3.0 contiene la classe Namespace per le operazioni con gli spazi dei nomi XML. Per l'oggetto XML dell'esempio precedente, è possibile usare la classe Namespace come segue:

```
var soapNS:Namespace = message.namespace("soap");
trace(soapNS); // Output: http://schemas.xmlsoap.org/soap/envelope/

var wNS:Namespace = new Namespace("w", "http://www.test.com/weather/");
message.addNamespace(wNS);
var encodingStyle:XMLList = message.@soapNS::encodingStyle;
var body:XMLList = message.soapNS::Body;

message.soapNS::Body.wNS::GetWeatherResponse.wNS::tempurature = "78";
```

La classe XML include i seguenti metodi per le operazioni con gli spazi dei nomi: `addNamespace()`, `inScopeNamespaces()`, `localName()`, `name()`, `namespace()`, `namespaceDeclarations()`, `removeNamespace()`, `setLocalName()`, `setName()` e `setNamespace()`.

La direttiva `default xml namespace` permette di assegnare uno spazio dei nomi predefinito agli oggetti XML. Nell'esempio seguente, sia `x1` che `x2` condividono il medesimo spazio dei nomi predefinito:

```
var ns1:Namespace = new Namespace("http://www.example.com/namespaces/");
default xml namespace = ns1;
var x1:XML = <test1 />;
var x2:XML = <test2 />;
```

3.12 Conversione degli oggetti XML

Gli oggetti XML e XMLList possono essere convertiti in valori stringa. Analogamente, è possibile convertire stringhe in oggetti XML e XMLList. Tenere presente che tutti i valori di attributo, i nomi e i valori di testo XML sono stringhe. Le sezioni seguenti affrontano tutte queste forme di conversione XML.

3.12.1 Conversione degli oggetti XML e XMLList in stringhe

Le classi XML e XMLList comprendono i metodi `toString()` e `toXMLString()`.

Il metodo `toXMLString()` restituisce una stringa che comprende tag, attributi, dichiarazioni di spazio dei nomi e contenuto dell'oggetto XML. Sugli oggetti XML dal contenuto complesso (cioè con elementi secondari), il metodo `toString()` agisce esattamente come il metodo `toXMLString()`. Per gli oggetti XML dal contenuto semplice (cioè quelli caratterizzati da un solo elemento di testo), il metodo `toString()` restituisce solo il contenuto testo dell'elemento, come illustra l'esempio seguente:

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
    <order>;
trace(myXML.item[0].menuName.toXMLString());
    // <menuName>burger</menuName>
trace(myXML.item[0].menuName.toString());
    // burger
```

Se si usa il metodo `trace()` senza specificare `toString()` o `toXMLString()`, i dati vengono automaticamente convertiti usando il metodo `toString()`, come dimostra il codice seguente:

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
    <order>;
trace(myXML.item[0].menuName);
    // burger
```

Quando si usa il metodo `trace()` per le attività di debug del codice, spesso risulta utile usare il metodo `toXMLString()` per fare in modo che `trace()` restituisca dati più completi.

3.12.2 Conversione di stringhe in oggetti XML

Usare la funzione di costruzione `new XML()` per creare un oggetto XML da una stringa nel modo seguente:

```
var x:XML = new XML("<a>test</a>");
```

Se si tenta di convertire in XML una stringa che rappresenta dati XML non validi o con una sintassi XML errata, viene generato un errore in fase di runtime nel modo seguente:

```
var x:XML = new XML("<a>test"); // throws an error
```

3.12.3 Conversione di valori attributo, nomi e valori di testo da stringhe

I valori attributo, i nomi e i valori di testo XML sono tipi di dati in formato stringa e potrebbe, pertanto, essere necessario convertirli in un tipo di dati diverso.

Nel seguente codice di esempio viene utilizzata la funzione `Number()` per convertire i valori di testo in numeri:

```
var myXML:XML =
                            <order>
                                 <item>
                                      <price>3.95</price>
                                 </item>
                                 <item>
                                      <price>1.00</price>
                                 </item>
                            </order>;

var total:XML = <total>0</total>;
myXML.appendChild(total);
```

```
for each (var item:XML in myXML.item)
{
     myXML.total.children()[0] = Number(myXML.total.children()[0])
                                                          +
Number(item.price.children()[0]);
}
trace(myXML.total); // 4.35;
```

Se nel codice non fosse stata aggiunta la funzione `Number()`, il codice avrebbe interpretato l'operatore + come l'operatore di concatenazione di stringhe e il metodo `trace()` dell'ultima riga avrebbe generato il valore seguente:

```
01.003.95
```

3.13 Lettura di documenti XML esterni

La classe URLLoader [5] permette di caricare dati XML da un URL.

Per usare il codice seguente all'interno delle applicazioni, sostituire il valore **XML_URL** dell'esempio con un URL valido:

```
var myXML:XML = new XML();
var XML_URL:String = "http://www.example.com/Sample3.xml";
var myXMLURL:URLRequest = new URLRequest(XML_URL);
var myLoader:URLLoader = new URLLoader(myXMLURL);
myLoader.addEventListener("complete", xmlLoaded);
function xmlLoaded(event:Event):void
{
     myXML = XML(myLoader.data);
     trace("Data loaded.");
}
```

È inoltre possibile usare la classe XMLSocket per impostare una connessione socket XML asincrona con un server.

[5] Il programma della libreria digitale incapsula tutta la logica relativa al caricamento di dati XML esterni in una opportuna classe chiamata XmlLoader. Tale classe verrà ampiamente descritta nei capitoli 4 e 5.

Concluso dunque anche questo capitolo, il lettore ha conoscenza di tutte le logiche programmative e di recupero dati utilizzate per lo sviluppo della libreria digitale. Passiamo dunque alla trattazione dettagliata e concreta del progetto nella seconda parte del mio lavoro.

Parte II

Capitolo 4

Progetto DigitalLab: libreria digitale via Web

Dopo aver studiato e analizzato i linguaggi che sono stati utilizzati per lo sviluppo e gestione dati del mio progetto, affrontiamo in questa seconda parte i dettagli progettuali e implementativi della libreria digitale.

In questo capitolo, nello specifico, dopo alcuni cenni storici e informativi che permettono al lettore di comprendere in cosa consista nel pratico una libreria digitale , si passa allo studio progettuale del mio lavoro.

Innanzitutto vengono spiegati termini specifici, spesso utilizzati per definire alcune componenti della libreria digitale, dopo di che tramite dei class diagramm vengono modellate le logiche e le strutture dati formanti l'applicativo.

4.1 Introduzione alle librerie digitali.

Una libreria digitale è una libreria costituita da documenti proposti in formati digitali e accessibili tramite computer.

I contenuti digitali possono essere fruibili da locale oppure da remoto. Una libreria digitale è un tipo di sistema “information retrieval”.

DELOS (Digital library Reference Model) definisce una libreria digitale come: “Un' organizzazione, che può anche essere virtuale, che raccoglie, gestisce e preserva per il lungo termine contenuti digitali e offre ai suoi utenti funzionalità di fruizione per tali contenuti, di qualità misurabile in accordo con politiche codificate”.

Il termine “libreria digitale” fu utilizzato per la prima volta nella stampa nel 1988, in una ricerca per la “Corporation for National Research Initiatives”.

Tale termine si diffuse grazie ad iniziative specifiche sulle librerie digitali nel 1994 da parte di enti quali NFS/DARPA/NASA.

Il termine “libreria virtuale” fu inizialmente utilizzato in maniera interscambiabile con “libreria digitale”, ma è oggi utilizzato principalmente per librerie che sono virtuali in altri sensi (come per esempio librerie che aggregano contenuti distribuiti).

Una distinzione è spesso fatta riguardo a contenuti che sono stati creati in origine in formati digitali, conosciuti come “born-digital”, e informazioni che sono state convertite da media fisici (come la carta) attraverso operazioni di digitalizzazione.

Il termine “libreria ibrida” è qualche volta utilizzato per definire librerie costituite sia da documenti fisici che da documenti digitali.

Per esempio, “American Memory” è una libreria digitale all'interno della “Library of Congress”. Alcune importanti librerie digitali diventano dei veri e proprio archivi a lungo termine come, per esempio, “ePrint arXiv” e “Internet Archive”.

4.2 Archivi digitali.

Gli archivi fisici si differenziano dalle librerie fisiche in diversi modi. Tradizionalmente, gli archivi erano definiti come:

1. Contenitori di sorgenti primarie di informazioni (tipicamente lettere e carte direttamente prodotte da una persona singola o organizzazioni) piuttosto che sorgenti di informazioni secondarie, invece presenti nelle librerie (libri, periodici, ecc.).
2. Enti nei quali i contenuti erano organizzati in gruppi piuttosto che lasciati come elementi individuali.
3. Enti i cui contenuti erano unici.

La tecnologia utilizzata per creare librerie digitali è stata più rivoluzionaria per gli archivi in quanto ha “spezzato” gli ultimi due dei tre punti sopra citati.
In altre parole, gli archivi digitali o archivi “online”, ancora contengono sorgenti di informazioni primarie, ma è possibile descrivere tali sorgenti individualmente oltre che come gruppo. Inoltre, i documenti digitali, come tali, sono facilmente riproducibili e possono essere stati riprodotti da altre parti.
“Libreria digitale è un'interazione tra comunità ed insiemi di informazioni”.

4.3 Biblioteche e librerie digitali.

Grazie ai nuovi supporti elettronici per la lettura, una rivoluzione nel mondo dell’editoria e della lettura, e lo avevamo osservato sotto l’aspetto della tecnologia di lettura ed in particolare per l’aspetto della "carta elettronica". Questa volta proviamo a osservarlo sotto l’aspetto della distribuzione dei contenuti.

Come in tutte le rivoluzioni gli elementi chiave del cambiamento avvengono in un tempo relativamente assai breve, ma l’affermazione su larga scala di essa ha bisogno di tempi lunghi. Già al tempo di Gutenberg, una cosa fu l’affermazione della stampa come novità tecnologica, un’altra cosa fu il sorgere di strutture per la

diffusione dei prodotti stampati: per molto tempo, oltre alle officine tipografiche, furono i venditori ambulanti i grandi diffusori del libro stampato.
Oggi ci troviamo di fronte a qualcosa di simile. La tecnologia permette di realizzare e di fruire il contenuto dei libri in formato digitale, ma per chiudere il cerchio servono altri due elementi: un nuovo modo di gestire (conservare e distribuire) questi contenuti in formato digitale; un nuovo modo di ragionare e gestire questi contenuti da parte dei nuovi lettori
Circa la conservazione da lungo tempo esistono le biblioteche. Ma già adesso le biblioteche stanno diventando elettroniche, cioè digitalizzando i contenuti dei libri del passato così da poterli consultare via rete. Riguardo ai lettori, saranno i giovani a produrre nuove modalità fruizione, e per ora si possono prevedere solo parzialmente, però questo sarà il vero cuore della rivoluzione della lettura. Mentre un aspetto fondamentale, di cui ora si stanno giocando le fasi chiave della partita, è quello della distribuzione del contenuto digitale, cioè come diffondere e comprare gli e-book.

4.4 Kindle, I-pad e le librerie digitali.

Da febbraio, "Usa Today" è il primo quotidiano ad inserire stabilmente nella classifica dei libri più venduti anche gli e-book. Per ora la competizione è sostanzialmente per la lingua inglese. Infatti, per avere una massa critica bisogna porsi a livello mondiale, e per ora solo l'inglese permette risultati tali da farsi notare, per le altre lingue ci sono realtà interessanti ma bisogna andare a cercarle.

Secondo alcune fonti Amazon, la più grande libreria del mondo, avrebbe già venduto 3 milioni di Kindle, mentre che le previsioni di vendita di Apple sono sui 4-5 milioni di I-pad.
A ben leggere la situazione, la discesa in campo di Apple con il suo *I-pad,* a contendere il passo al *Kindle* di Amazon apre non tanto ad una competizione degli e-reader (ci sono altri e-reader anche migliori per la lettura), quanto piuttosto ad una competizione della gestione di diffusione e vendita dei contenuti. Apple con

I-phone (ed ora don I-pad) offre un prodotto e una serie di servizi; Amazon offre Kindle e la più vasta biblioteca/libreria del mondo.
Comunque è anche un problema economico, Amazon usa un prezzo fisso (9,90 $) per tutti i libri (tra l'altro per ora pagandolo ad un prezzo superiore a quello che versa agli editori). Ma qui che si gioca l'evoluzione della lettura nel sistema della diffusione e della vendita dei libri in formato digitale. Sarà l'interesse degli editori a poter guadagnare che farà cambiare gli attuali orizzonti. Gli analisti propendono per un costo intorno ai 5 dollari.

4.5 La gestione dei diritti digitali.

Un'altra grande questione, non tecnologica ma strategica, e che interessa tutti i prodotti digitali (oltre o assieme a quella economica in senso stretto) è quella che sul web viene stringatamente definita con la sigla DRM (*Digital Rights Management*, cioè gestione dei diritti digitali). In altre parole come gestire, e proteggere i diritti degli autori di un'opera d'ingegno, e come garantire gli interessi economici degli editori.

Le due questioni sono strettamente intrecciate, anche se alcuni vorrebbero poterle trattare in modo separato. Concettualmente sono due problematiche differenti ma in pratica è assai difficile districarle.
Non intendiamo qui trattare il problema legale, perché per questo si va nei tribunali, e con sanzioni come il mondo dei libri conosce bene da molto tempo, ma piuttosto quelle *tecniche di protezione preventiva* circa la copia o il funzionamento di programmi e di contenuti.
Nel mondo dei computer il problema è presente da tempo con l'avvento dei DVD, ma ha assunto dimensioni nuove con la musica digitale scaricabile on-line.
Un programmino inserito nel file impedisce l'uso dello stesso se non si possiede il codice, così non si può copiare senza passare dal venditore, ma ormai si giunge al punto che non si possono neppure fare più copie dello stesso file per colui che ha regolarmente pagato il prodotto.

Tralasciando il problema tecnico (e di programmi che infrangono tale blocco), ci interessa qui ragionare sulla logica che è dietro ai DRM e alle due linee principali di pensiero che sono dietro.
I primi dicono che in questo modo si proteggono i diritti degli autori e degli editori e così si assicura il giusto compenso, anche se si produce un disturbo per l'utente.
I secondi dicono che in questo modo si incentiva in realtà l'uso di prodotti "piratati", perché in realtà questo serve a tenere i prezzi alti dei prodotti originali. Se si offrissero i prodotti a prezzi più ragionevoli la gente acquisterebbe i prodotti originali senza cercare di ottenerli in modo illegale.
La questione per il mercato editoriale è tutt'altro che secondaria. Infatti, molti editori, grandi e piccoli, stanno aspettando di capire quale corrente prevarrà e quale tecnologia sarà usata.
Ovviamente i grandi editori, o case discografiche che siano, preferirebbero tutelarsi il più possibile, ma sanno che in questo modo non incentivano i guadagni.
Nel mondo musicale dopo un primo approccio con DRM molto rigidi (ma puntualmente violati) si sta andando verso la strategia di offrire una grandissima scelta a prezzo sempre più ragionevole.
Perché i libri digitali possano diventare un fenomeno di massa questa problematica deve essere risolta, altrimenti come già sta accadendo, può capitare di avere un testo che si può leggere solo su certi e-reader.

4.6 Le librerie on-line.

La stampa nasce con i tipografi, non con gli editori. Un finanziatore o un benefattore che pagasse la spesa iniziale. Le moderne "case editrici" nascono dopo, in seguito a due fattori: l'alfabetizzazione generalizzata e la conseguente possibilità di alte tirature. Prima delle case editrici sorgono le "librerie" che sono il luogo dove poter trovare e acquistare i libri. Col tempo per rifornire le librerie

diventano sempre più importanti i distributori con i loro magazzini e cataloghi sempre più vasti.

Infatti, Amazon non è e non ragiona come una editrice, ma da distributore. Inoltre propone un ecosistema chiuso, con i libri digitali su un suo supporto proprietario (il Kindle) e la vendita on-line può fare due cose: bypassare le rivendite sul territorio (le librerie), e (può) diventare direttamente anche l'editore di testi che non conosceranno mai la carta.
Tutti, e da ogni parte del mondo, lingua permettendo, possono scaricare gli e-book: è la libreria nell'era della globalizzazione e del mondo digitale.
La seconda cosa è possibile, ma per fare un buon libro oltre all'autore serve il lavoro redazionale e di impaginazione, la promozione per far conoscere il libro. Cioè significa comunque dotarsi di una redazione editoriale.
Tuttavia con un libro "senza carta" si può rinunciare a tutta la produzione e distribuzione che implica la carta: stampa, magazzini e trasporti. Ma come non serve a nulla stampare un libro se non ci sono persone che li acquistano e li leggono, così non servirà a granché produrre libri elettronici se poi non ci saranno acquirenti in grado di acquistarli, ma anche solo in grado di sapere che esistono. Sarà perciò fondamentale la diffusione.
Non sappiamo se Amazon vorrà anche essere una "casa editrice digitale", ma certamente ci abitueremo presto a nuovi modi di produrre e diffondere contenuti di testo.

4.7 SRS

4.7.1. Introduzione

- ✓ **Scopo.**

Vuole essere progettata un'applicazione web che permetta la gestione e fruizione di documenti digitali. Tali documenti sovente consistono in immagini 'bitmap' ottenute scansionando sorgenti cartacee. L'applicativo permette all'utente di

selezionare uno specifico documento tramite un sistema di navigazione sequenziale, oppure tramite icone organizzate in menu altamente interattivi. Le icone sono create, a partire dai documenti, automaticamente da script opportuni.
Inoltre, strumenti specifici, abilitano la possibilità di ridimensionare i documenti per apprezzarne i dettagli e trascinarli sullo schermo per prendere visione di eventuali porzioni non accedibili qual'ora il documento risultasse visualizzato con un'alta percentuale di ingrandimento.
Tutti i dati relativi ai documenti e corrispondenti miniature (percorsi, identificatori, ecc.) sono memorizzati in una base di dati dedicata e resi utilizzabili dall'applicazione, mediante 'query', in formato 'XML'.

- ✓ **Validità.**

Un'applicazione di questo tipo permette di prendere visione di documenti di diversa tipologia (interi libri, locandine storiche, giornali e quotidiani, ecc.) senza dover necessariamente andare in luoghi specifici adibiti alla consultazione (librerie e biblioteche, ecc.).
Stando a casa o nel luogo di lavoro, via internet, si ha quindi accesso ad un insieme di informazioni che, altrimenti, potevano essere adeguatamente ottenute solo recandosi fisicamente in ambienti dedicati.
Inoltre, grazie al sistema di ricerca incorporato nell'applicativo, è possibile ridurre drasticamente i tempi di recupero del materiale di interesse.

- ✓ **Definizioni.**

Utente: con questo termine si identifica qualunque persona che, connettendosi via web alla libreria digitale, interagisce con essa, fruendo e visualizzando uno o più documenti con gli strumenti e le modalità messe a disposizione dall'applicazione.

Documento: con tale termine si identifica, in questo libro, una controparte digitale di una sorgente di informazioni all'origine normalmente cartacea. Al momento ogni documento è, più specificatamente, un'immagine 'bitmap' ottenuta tramite la scansione di tale sorgente di informazioni.

Molto frequentemente un documento è un'immagine 'bitmap' raffigurante una coppia di facciate di un libro.

Miniatura: immagine 'bitmap' rimpicciolita ottenuta, tramite esecuzione si specifici script lato server, a partire da un documento.
Le miniature, nel progetto illustrato in questo volume, risultano organizzate in menu accedibili da parte degli utenti. Ogni miniatura è interattiva e, se cliccata, permette la visualizzazione del corrispettivo documento.

Menu in stile 'Mac': è un menu interattivo entro il quale vengono organizzate le miniature. L'utente muovendosi con il 'mouse' sulla sua superficie, può apprezzare un sistema di interazione e animazione molto simile al 'dock' tipico delle piattaforme 'Mac'.

Visualizzazione a schermo intero (full-screen): indica una modalità di visualizzazione della libreria digitale per cui tutti i suoi elementi grafico-interattivi e documenti, si ridistribuiscono e ridimensionano in modo da occupare tutta la superficie dello schermo. Il risultato è un ambiente più amplio e spazioso con il quale l'utente interagisce con maggiore facilità e chiarezza.

Stage: indica l'area dell'applicazione i cui contenuti sono visibili all'utente e l'area con la quale è possibile interagire.

4.7.2. Descrizione generale

- ✓ **Prospettive.**

L'applicativo può essere esteso per visualizzare non solo documenti intesi come rasterizzazioni di materiale cartaceo ma anche documenti rappresentati da media di tipologia diversa, come ad esempio video e audio. La libreria digitale, agganciando opportune classi a quelle già presenti, potrebbe abilitare la

visualizzazione di eventi, mostre, ecc., con tracce video-audio registrate o, legandosi ad un server opportuno, con dati ricevuti in streaming.

Inoltre, un'altra estensione interessante, è quella che permetterebbe la gestione di documenti intesi come tracce audio preregistrate. Un passo di un libro letto da un bravo narratore, una poesia letta dall'autore,: questi sono esempi di documenti che l'applicativo potrebbe facilmente gestire per permettere l'acquisizione delle informazioni in maniera estremamente coinvolgente e non-standard. L'utente, dal luogo di preferenza, è coinvolto pienamente in quella che diventa un'esperienza di consultazione e non più una semplice acquisizione di dati.
Non solo. Inserendo nel sistema di classi costituenti la libreria digitale, un package opportuno, i documenti potrebbero essere resi fruibili anche tramite formati di larga diffusione come 'PDF'.

- ✓ **Funzioni.**

Utenti e fruizione dei documenti: un utente della libreria digitale, come prima operazione, può selezionare, tra gli 'n' insiemi di documenti presenti, l'insieme d'interesse. Una volta compiuta tale scelta accede alla libreria digitale vera e propria.

Inizialmente viene proposta un' interfaccia costituita esclusivamente da un pulsante di navigazione sulla sinistra dello schermo ed uno sulla destra. Dell'insieme di documenti di interesse, viene visualizzato il primo, collocato nel centro dello schermo. Se le sue dimensioni originali lo richiedono, il documento viene opportunamente ridimensionato per poter essere visualizzato nella sua interezza. Questo significa che, in certi casi, alcuni dettagli del documento potrebbero non vedersi con nitidezza. I pulsanti sopra citati, se premuti, permettono di accedere ai documenti precedenti e successivi, qual'ora presenti, a quello correntemente visualizzato.

Ogni documento viene precaricato dal sistema. Questo significa che, dopo la selezione di uno specifico documento che è esterno all'applicazione, deve essere segnalato all'utente con opportune grafiche, che il materiale richiesto 'sta

arrivando'. In questo modo si segnala che l'applicazione non è mal funzionante, ma che le eventuali latenze sono dovute al fatto che si sta accedendo comunque a risorse in internet.

Una volta caricato il documento richiesto, l'utente può, premendo sulla sua superficie con il puntatore del 'mouse', andare in modalità a schermo intero. A questo punto la libreria digitale occupa tutta la superficie del 'monitor'. In questa visualizzazione il documento viene portato alle dimensioni originali.
Se necessario ai fini della sua fruizione, compare una barra di scorrimento alla sua destra utilizzabile per muovere il documento lungo l'asse verticale. I pulsanti di navigazione vengono riposizionati per risultare nuovamente vicini uno al bordo sinistro dello schermo, l'altro al bordo destro. Compare inoltre, in basso, un menu in stile 'Mac', contenente le miniature dei documenti che formano l'insieme dei documenti selezionato a monte dall'utente.

Premendo su una miniatura, si elimina il documento correntemente selezionato per visualizzare quello corrispondente alla miniatura stessa. In pratica, in questa visualizzazione, l'utente può navigare tra i documenti in maniera sequenziale tramite i pulsanti collocati sulla sinistra e sulla destra dello schermo, o in maniera selettiva, premendo le miniature contenute nel menu. Nel caso in cui i documenti d'interesse per l'utente siano molti, non tutte le corrispondenti miniature vengono visualizzate contemporaneamente nel menu in stile 'Mac'. Viene costruito, quindi, un secondo menu, costituito da pallini interattivi. Ogni pallino, se premuto, toglie le miniature correntemente visualizzate nel menu, per inserirne di nuove, corrispondenti ad un altro sottoinsieme di documenti.

In questa visualizzazione, se l'utente preme sulla superficie del documento correntemente selezionato, tale documento subisce un secondo ingrandimento. Ora è facile che risulti così grande da richiedere la possibilità di trascinarlo per poterlo vedere nella sua interezza. Questa visualizzazione è interessante perché permette all'utente di esplorare i dettagli del documento. Dettagli non solo legati alla fruizione delle informazioni (i caratteri con cui sono scritte le parole del documento, in questa modalità risultano, decisamente più leggibili), ma anche

legati al supporto stesso del documento. Se, per esempio, ciò che si sta visualizzando è il risultato di una scansione di un documento antico, si riescono ora a percepire anche i segni del tempo sulla carta e così via. Si riesce, cioè, ad intuire il 'sapore' del documento e non solo il 'sapere'. E' sufficiente premere il tasto 'ESC' per ripristinare la visualizzazione iniziale.

Ricerca dei documenti: durante la visualizzazione normale, (non a schermo intero), è disponibile un elemento interattivo nella parte superiore dell'applicazione, dotato di un campo di testo di 'input' grazie al quale poter compiere una selezione mirata di un documento nell'insieme selezionato a monte.
In tale campo, l'utente può scrivere quindi, un identificatore univoco associato ad un dato documento. Tale documento viene visualizzato sullo schermo e i sistemi di navigazione aggiornano le loro logiche in maniera tale che, se premuti, riconoscono quali sono i nuovi documenti (precedenti e successivi) da far visualizzare.

- ✓ **Utilizzatori.**

Utenti: non sono richieste particolari conoscenze per l'utilizzo della libreria digitale se non le elementari basi di interazione con applicazioni digitali.

4.7.3. CLASS DIAGRAM.

- ✓ **Fase di precaricamento della libreria digitale: estensione del 'framework' di Flash tramite la classe 'Preload'.**

Tutte le volte che si sviluppa un'applicazione con ActionScript 3.0 è necessario che l'insieme delle classi costituenti l'applicativo, estenda una o più classi messe a disposizione dalle librerie AS3 preesistenti che abilitano una serie di funzionalità da cui non si può prescindere (dalla gestione di elementi visuali, all'applicazione dell'interattività, dalla possibilità di manipolare testi, alle tecniche per recuperare dati esterni (es: dati XML)).

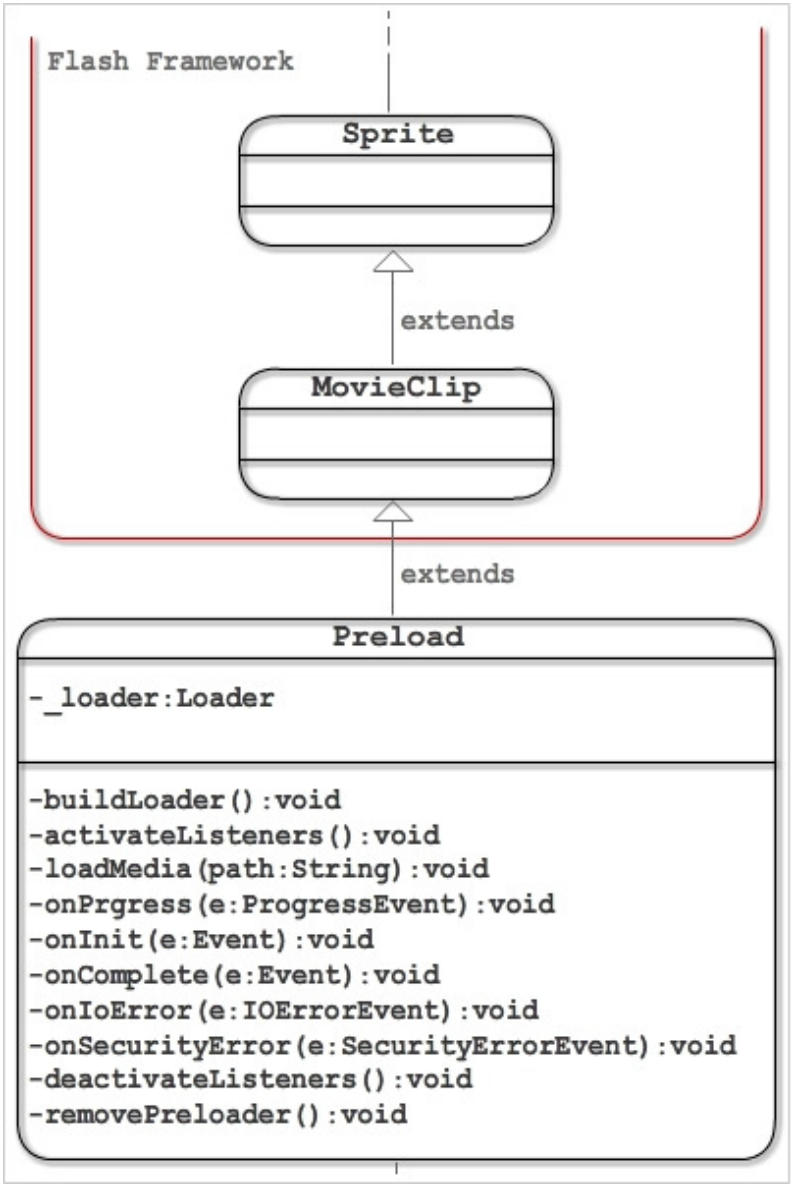

Figura 4.1:Class diagram con in evidenza la classe 'Preload'

Nel grafico appena proposto (Figura 4.1) viene messo in evidenza come la classe 'Preload', che gestisce il caricamento della parte principale dell'applicazione, per poter far visualizzare un'animazione che varia al variare del rapporto tra i 'byte' caricati della sezione principale e i 'byte' totali di tale sezione, debba necessariamente estendere le classi della 'DisplayList', ovvero quelle classi del 'framework' di Flash che permettono di gestire qualunque elemento grafico, testuale ed interattivo.

Nel caso specifico,come si può vedere nella Figura 4.2, oltre ad un'animazione, viene visualizzato un campo di testo il contenuto testuale è proprio un valore numerico ottenuto con tale rapporto e moltiplicato per cento.

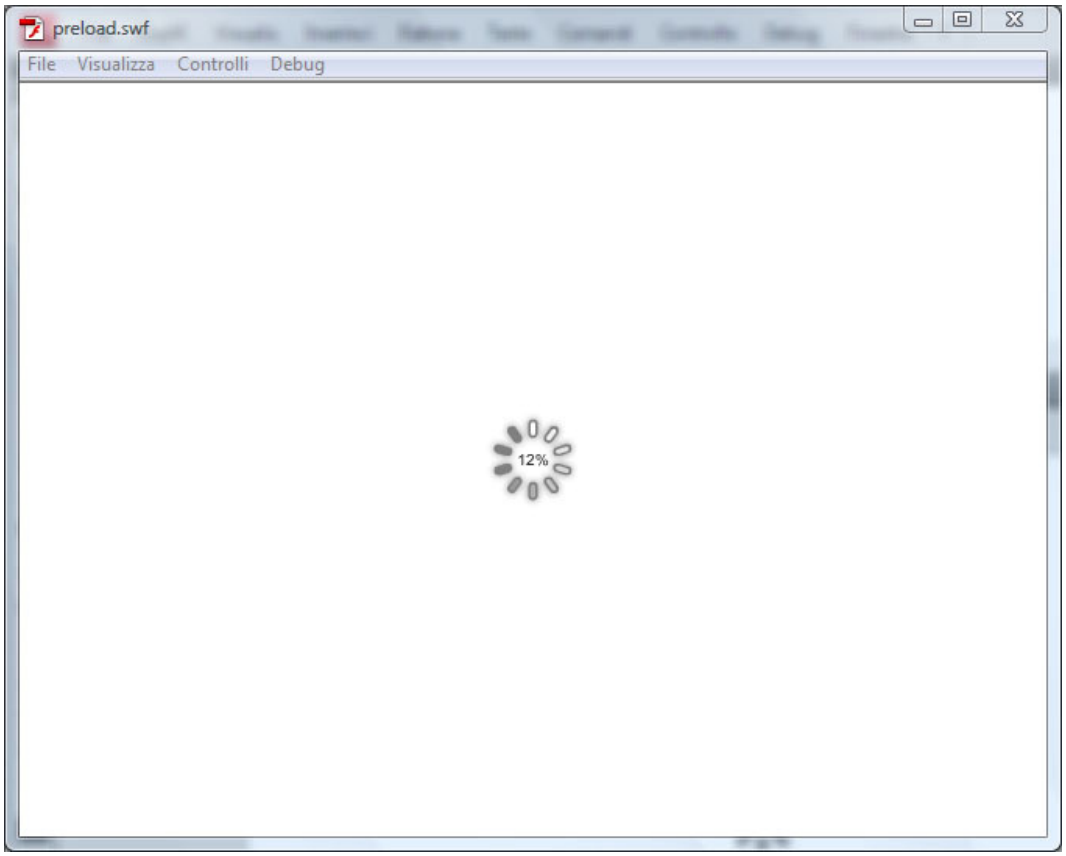

Figura 4.2: L'istanza della classe 'Preload', produce un'animazione di questo tipo.

- ✓ **Il cuore della libreria digitale: estensione del 'framework' di Flash tramite la classe 'Main'.**

La classe 'Main' è di primaria importanza poiché una sua istanza gestisce sostanzialmente tutti gli altri oggetti che rappresentano, sia a livello grafico sia a livello logico, le componenti essenziali della libreria digitale. Anche questa classe, come la classe 'Preload', si lega al 'framework' di Flash estendendo la classe 'MovieClip'.

Questo, tra le altre cose, permette ad un oggetto 'Main' di comportarsi come 'contenitore', ovvero può, a sua volta contenere elementi visuali o altri contenitori, dando origine ad una gerarchia di elementi.

Tra i contenuti dell'istanza 'Main', vale la pena citare i documenti, i pulsanti di navigazione, i menu in stile 'Mac', gli elementi per la ricerca dei documenti , come si mette in evidenza nella Figura 4.3.

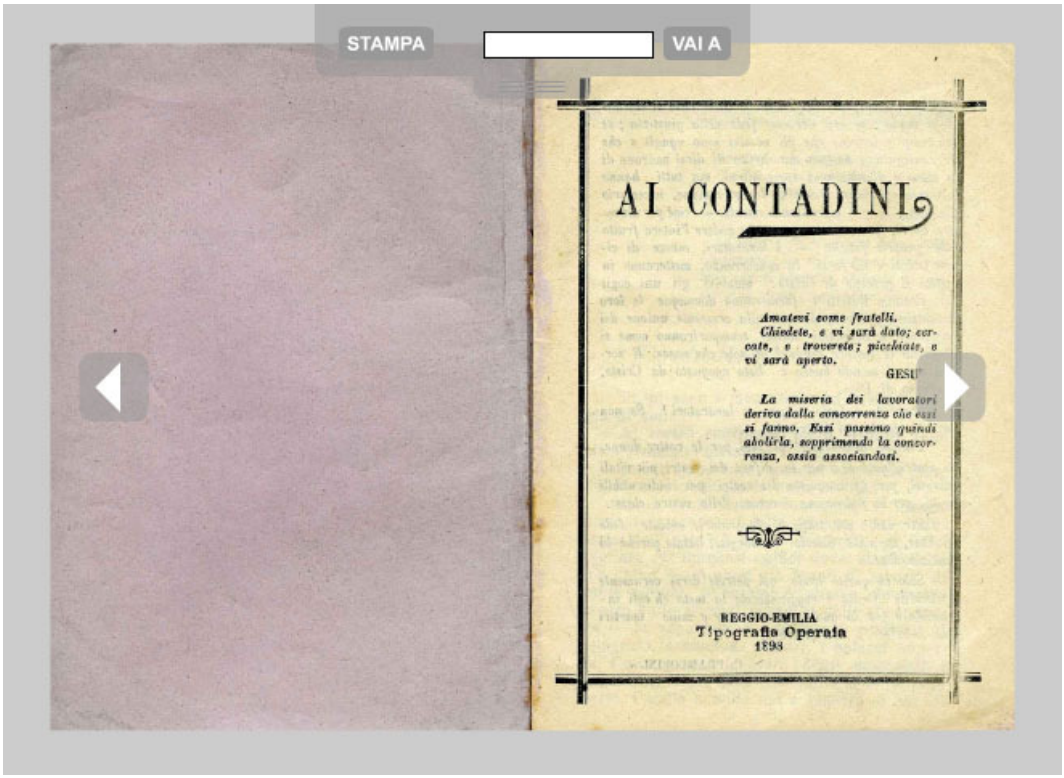

Figura 4.3: Tutto ciò che si vede è gestito dall'oggetto 'Main'.

Nello schema riportato nella pagina successiva, Figura 4.4, è stato volontariamente elencato solo un sottoinsieme dei metodi e delle proprietà di tale classe.

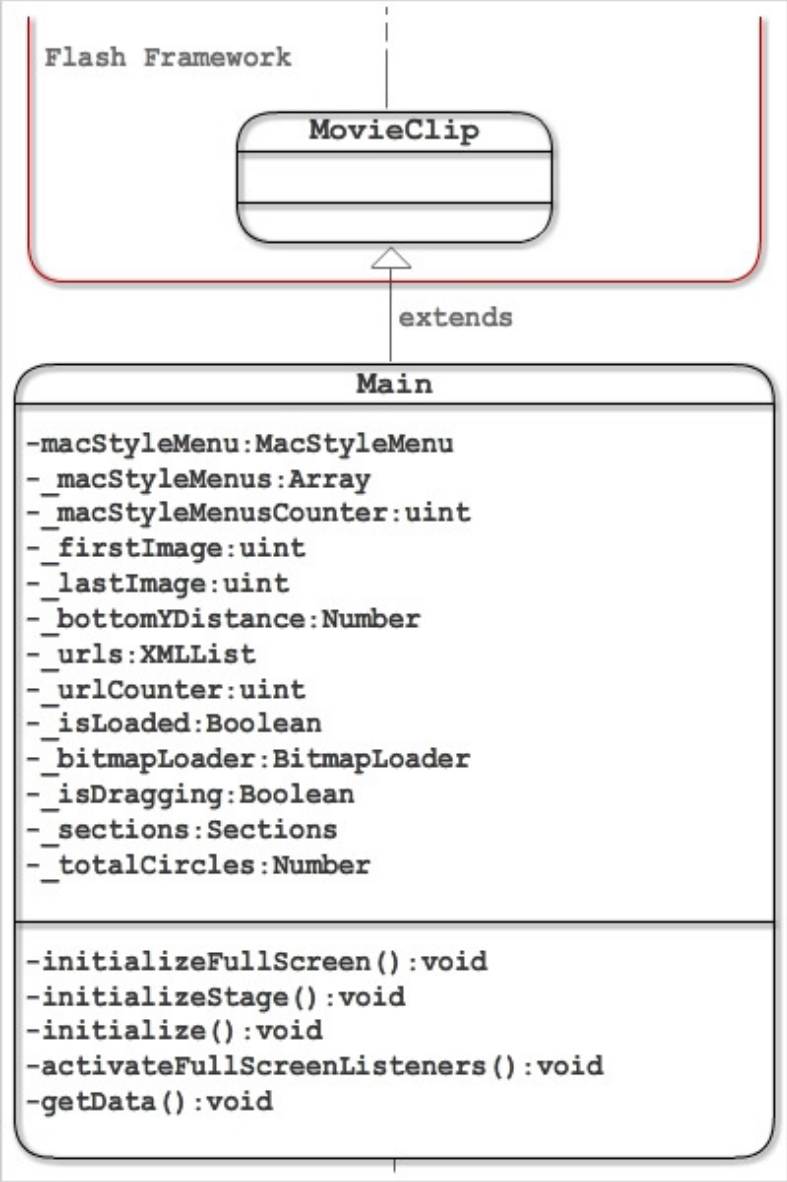

Figura 4.4: Class diagram con in evidenza la classe 'Main'.

- ✓ **Caricare e visualizzare un documento: estensione del 'framework' di Flash tramite la classe 'BitmapLoader'.**

Tutte le volte che un utente decide di visualizzare un nuovo documento, viene generata una nuova istanza della classe 'BitmapLoader'. Questa classe, al suo interno, ha tutta la logica che serve per gestire il caricamento di un'immagine 'bitmap' esterna all'applicazione.

In sostanza, selezionare un nuovo documento, significa far eseguire un opportuno metodo dell'oggetto 'Main', che istanzia la classe 'BitmapLoader'. Al costruttore di tale classe viene mandato, via parametrica, il percorso all'immagine da caricare.
Una volta completato il caricamento, l'immagine 'bitmap' risulta visibile all'utente poiché è 'incapsulata' nel contenitore rappresentato dall'oggetto 'BitmapLoader' che, a sua volta, è contenuto dall'oggetto 'Main' che ne ha indotto la creazione.
Un oggetto 'BitmapLoader' acquisisce la possibilità di contenere elementi visuali poiché estende un'opportuna classe del 'framework' di Flash, nello specifico la classe 'Sprite' come illustrato dallo schema riportato qui di seguito (Figura 4.5).

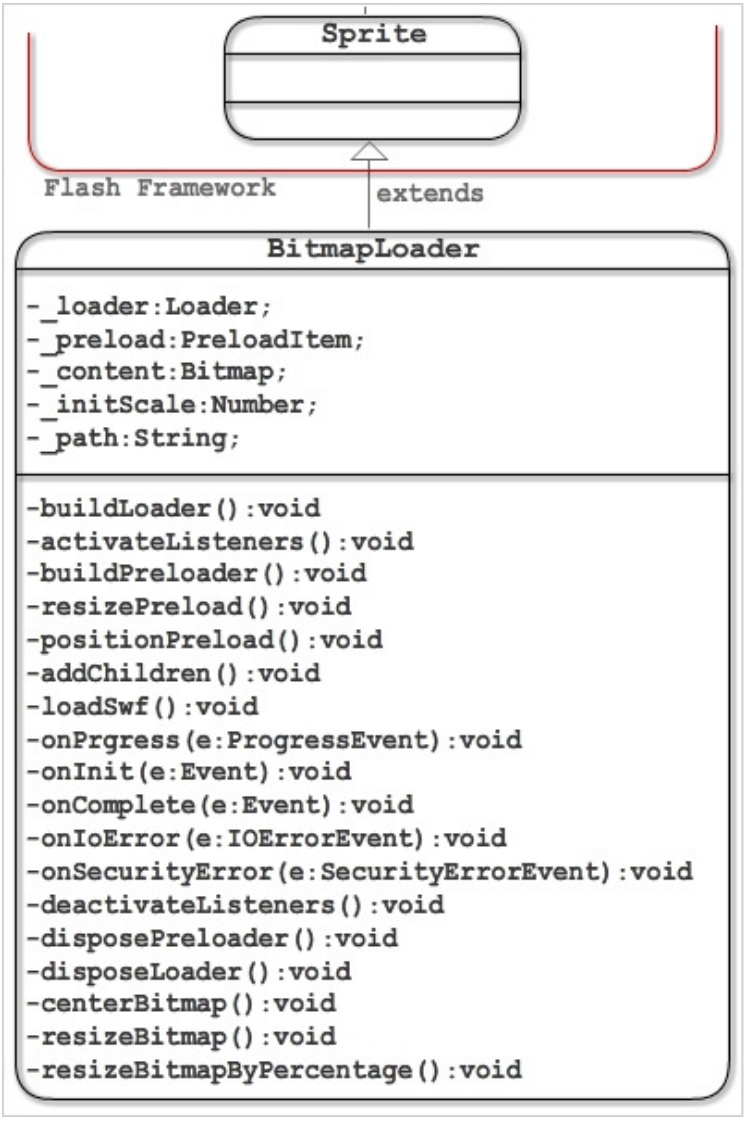

Figura 4.5: Class diagram con in evidenza la classe 'BitmapLoader'.

- ✓ **Caricare e visualizzare un documento: l'oggetto 'Main' istanzia la classe 'BitmapLoader' per accedere e visualizzare ad un'immagine 'bitmap' esterna.**

Nel grafico sotto proposto (Figura 4.6) , viene messo in evidenza che l'oggetto 'Main' è in grado di creare tanti oggetti 'BitmapLoader', quanti sono i documenti richiesti dall'utente.

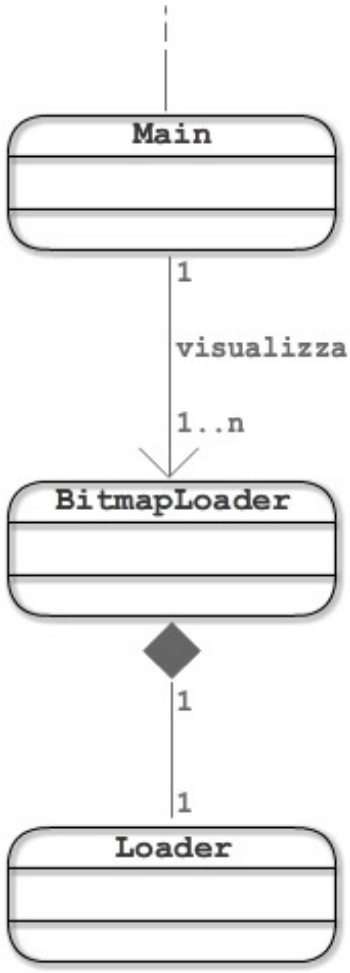

Figura 4.6: Class diagram con in evidenza le relazioni tra classe 'Main', 'BitmapLoader' e 'Loader'.

Una volta selezionato un documento, l'oggetto 'BitmapLoader' che si è occupato del caricamento del potenziale documento precedente, viene rimosso (rimuovendo, quindi dall'applicazione, l'immagine in esso contenuta), per poter generare un'istanza di 'BitmapLoader' nuova alla quale è dato il compito di

caricare il documento appena scelto. E' corretto dire, quindi, che in un dato momento, la libreria digitale visualizza un e un solo documento.

Vale la pena sottolineare che un documento è sempre presente nella libreria digitale, poiché all'avvio, il primo documento facente parte dell'insieme scelto dall'utente, è caricato automaticamente (un'istanza di 'BitmapLoader', cioè, viene creata in fase di inizializzazione dell'applicazione).

Lo schema, inoltre, mette in luce che ogni oggetto 'BitmapLoader', per composizione, gestisce un proprio oggetto 'Loader' al quale è dato l'effettivo compito di iniziare e gestire, attraverso suoi metodi e proprietà, la fase di caricamento dell'immagine esterna. La classe 'Loader' è parte del 'framework' di Flash.

- ✓ **Il menu in stile 'Mac': estensione del 'framework' di Flash tramite la classe 'MacStyleMenu'.**

La classe 'MacStyleMenu' genera istanze capaci di gestire un dato numero di miniature. Ricordando che le miniature, alla stregua dei relativi documenti, sono immagini caricate dall'esterno, si comprende come mai anche la classe 'MacStyleMenu' debba essere un contenitore in grado di poter gestire al suo interno elementi visuali. Definisce, inoltre, un puntatore grafico che evidenzia con una deliziosa animazione con frizione, quale miniatura è quella correntemente selezionata.

Un oggetto 'MacStyleMenu' acquisisce la possibilità di comportarsi come contenitore, in quanto la classe di cui è istanza estende la classe 'Sprite', inserita nel 'framework' di Flash.

La classe 'Sprite' è la classe offerta da ActionScript 3.0 con il minor 'overhead' di dati, tra quelle che si comportano come contenitori, in grado di produrre oggetti entro i quali inserire elementi grafico-testuali.

La classe 'MacStyleMenu', inoltre, definisce al suo interno metodi in grado di verificare il momento in cui l'utente interagisce con esso.

Quando questa situazione si verifica, un temporizzatore viene avviato per produrre l'animazione che, tenendo conto delle coordinate del 'mouse', riproduce fedelmente la medesima situazione che si ha con il 'dock' delle piattaforme 'Mac'. Una volta che l'utente interrompe l'interazione con il menu, anche il temporizzatore smette di eseguire le funzioni che gestiscono l'animazione, ottimizzando le risorse del computer ospitante l'applicativo.

Lo schema seguente , Figura 4.7, illustra come la classe 'MacStyleMenu' si lega al 'framework' di Flash.

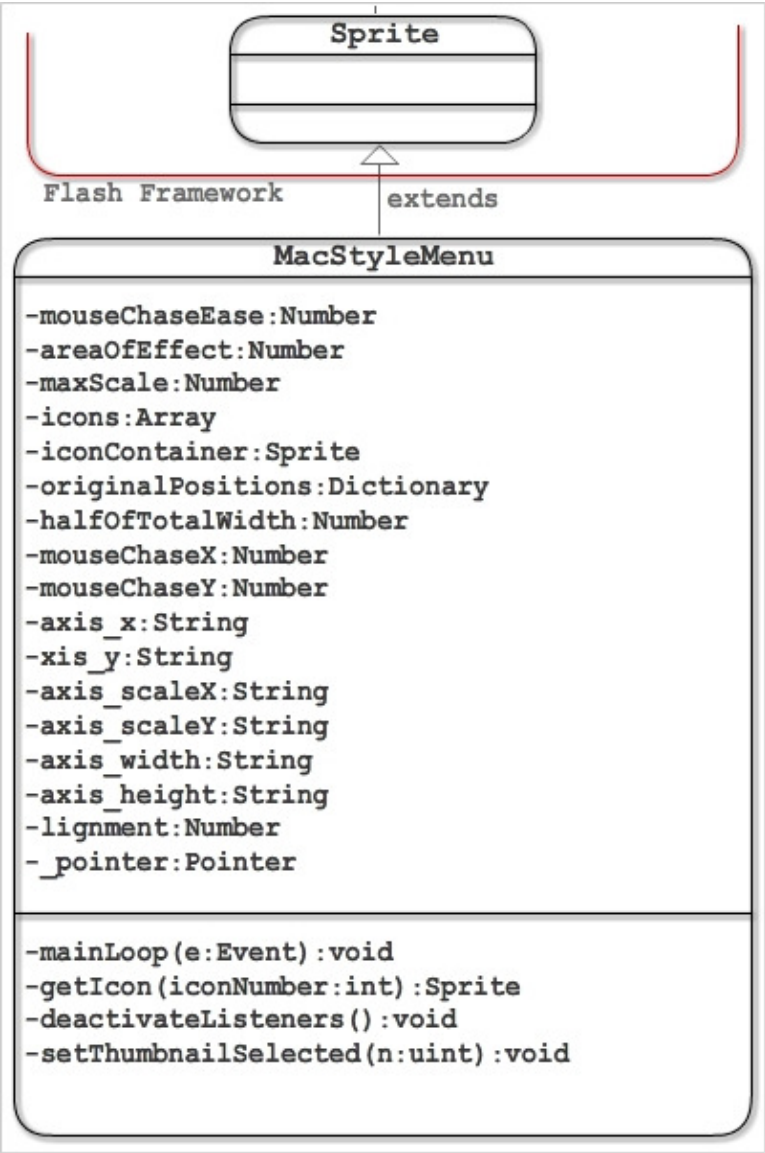

Figura 4.7 Class diagram con in evidenza la classe 'MacStyleMenu'.

- ✓ **Il menu in stile 'Mac': contenitore di miniature (istanze della classe 'Thumbnail').**

Un oggetto 'MacStyleMenu' delega, ad 'n' istanze della classe 'Thumbnail', la gestione del caricamento di altrettante miniature.

La classe 'Thumbnail' è un'estensione della classe 'BitmapLoader', da cui ne eredita tutti i comportamenti. In più, però, un oggetto 'Thumbnail' riesce a centrare la grafica che segnala l'attività di caricamento dell'immagine esterna, rispetto all'area effettiva che verrà occupata dalla miniatura. Inoltre, aggiunge un campo di testo posizionandolo sopra la miniatura. In tale campo viene visualizzato l'identificatore del documento corrispondente.

Nella figura che se gue, Figura 4.8, troviamo una rappresentazione di quanto appena detto.

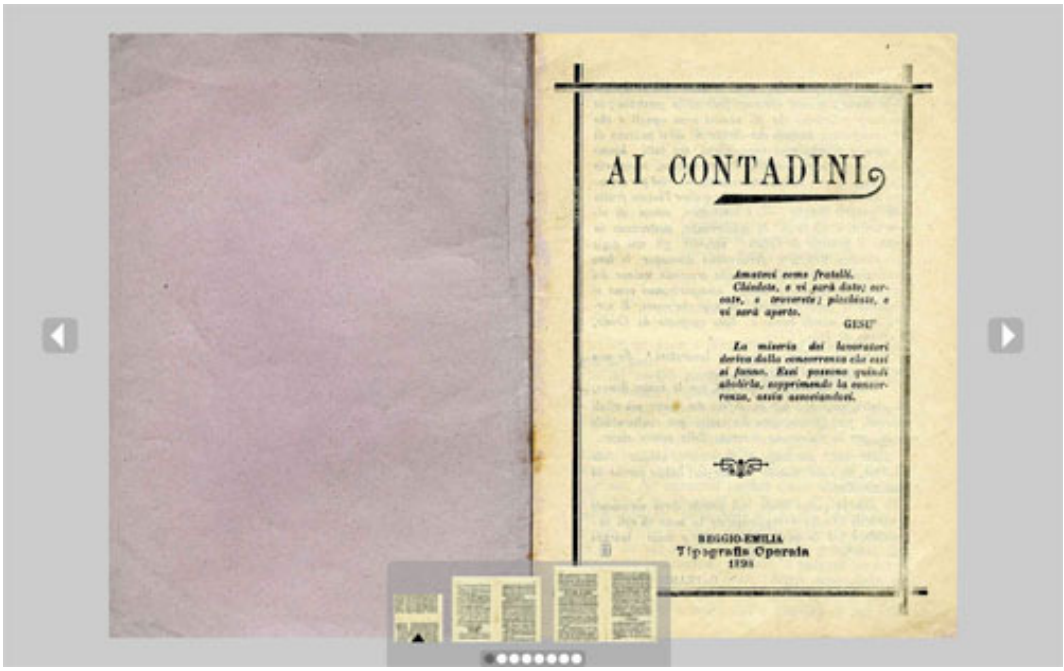

Figura 4.8: Il menu in stile 'Mac' in evidenza.

Il grafico riporta qui di seguito nella Figura 4.9, mostra la relazione tra le classi coinvolte nella creazione di un menu di miniature.

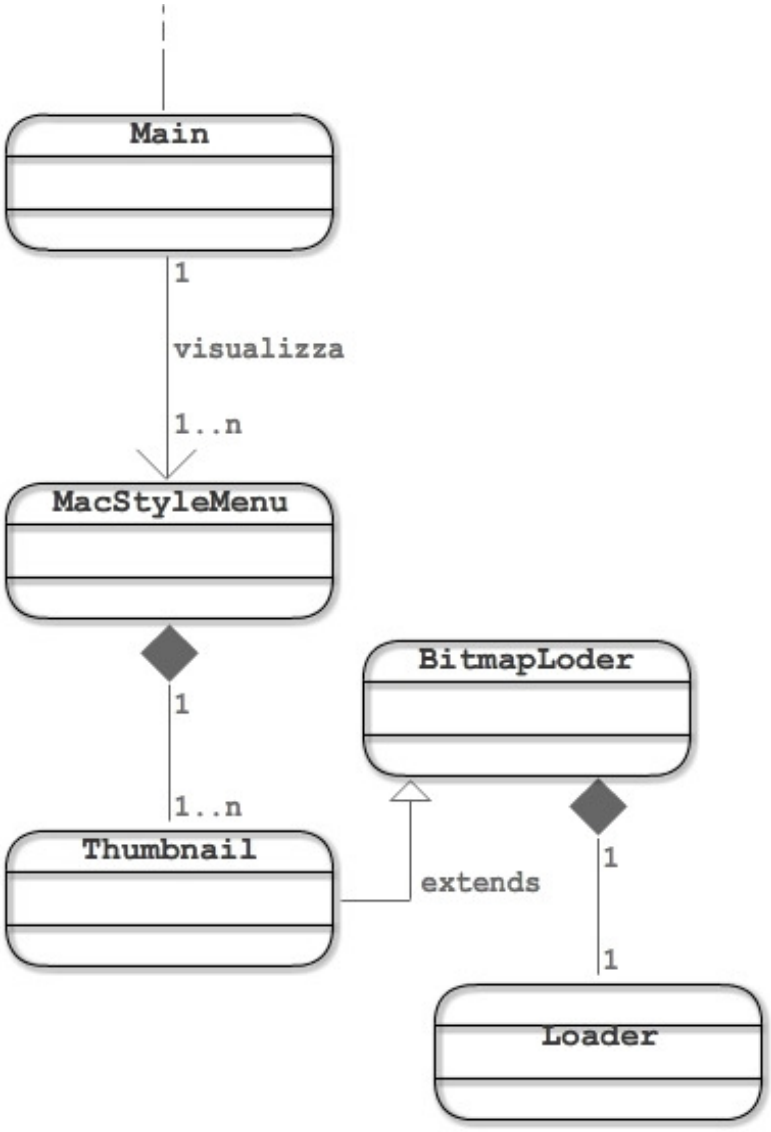

Figura 4.9: Class diagram con le classi che partecipano alla definizione di un menu.

- ✓ **Caricare i dati XML: la classe 'XmlLoader'.**

La classe 'XmlLoader' genera istanze in grado di caricare stringhe testuali esterne, dati XML compresi. Tali istanze sono in grado di comunicare all'utente, tramite opportuni messaggi, se sono avvenuti problemi legati alla fase di caricamento dei dati (es.: i dati richiesti non sono esistenti).

Al costruttore di questa classe viene passato, via parametrica, il percorso/URL della sorgente di dati.

Al contrario di molte classi prese in esame prima, la classe 'XmlLoader' produce istanze che non devono rappresentare elementi visuali o contenitori di elementi visuali. Per questa ragione, pur legandosi al 'framework' di Flash, non estende classi come 'MovieClip' o 'Sprite'. Invece, la classe estesa da 'XmlLoader', è 'EventDispatcher'.

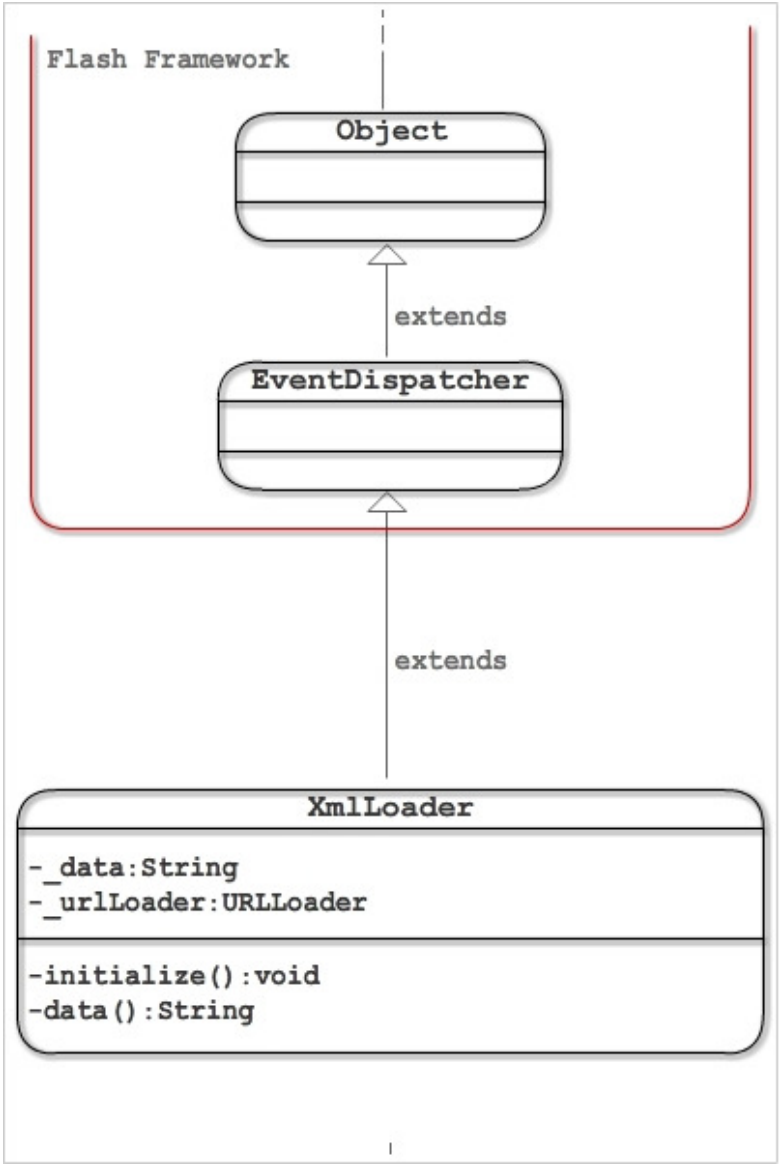

Figura 4.10: Class diagram con la classe 'XmlLoader' in evidenza.

- ✓ **Caricare i dati XML: l'oggetto 'Main' istanzia la classe 'XmlLoader' per accedere a dati XML esterni.**

L'oggetto 'Main', tramite un metodo dedicato, crea un'istanza della classe 'XmlLoader' che, al suo interno, ha ulteriori oggetti e logiche utili al recupero, caricamento e gestione del caricamento di dati testuali esterni.

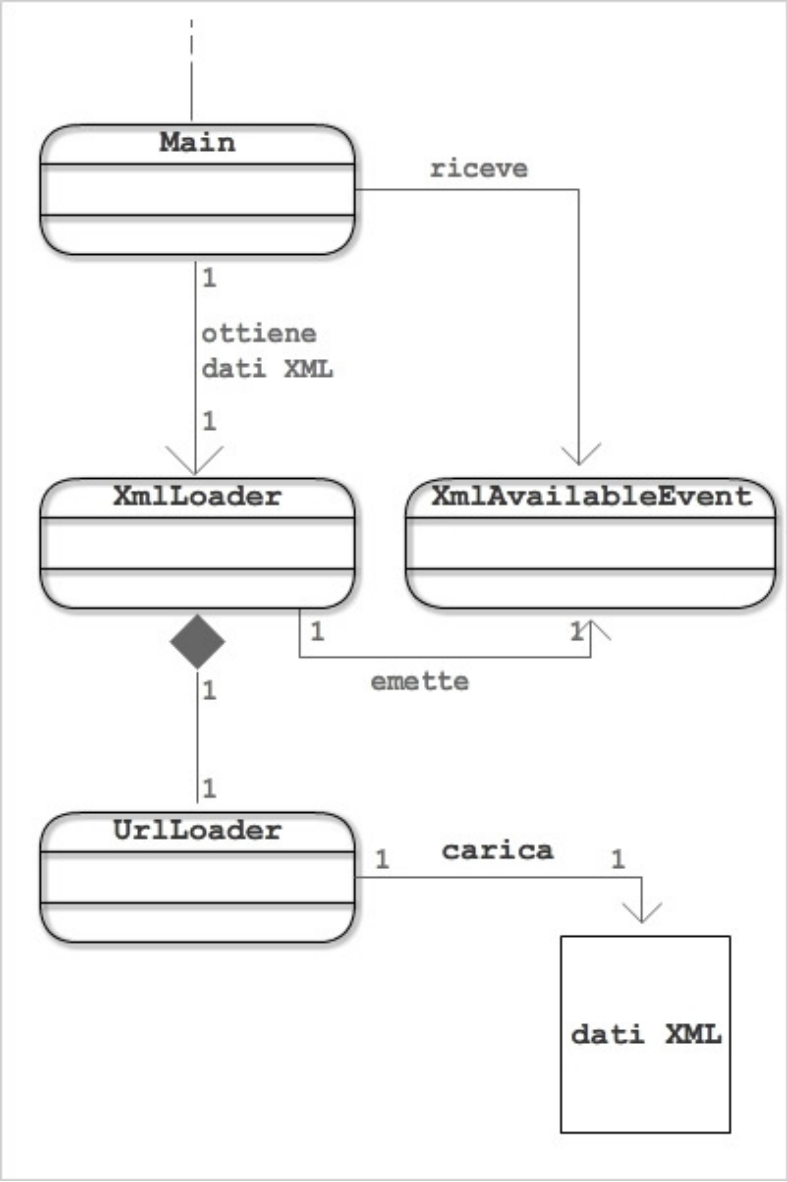

Figura 4.11: Class diagram con le classi per il caricamento ed uso di dati XML.

Facendo riferimento alla Figura 4.11, è utile sapere che un'istanza della classe 'XmlLoader' gestisce per composizione un oggetto 'URLLoader'. E' un'entità di questo tipo che realmente si occupa di recuperare i dati XML esterni, gestirne la fase di caricamento, e provvedere con opportuni messaggi a segnalare eventuali errori (es.: assenza della risorsa richiesta o problemi legati alla sicurezza).

Una volta che i dati XML sono stati caricati con successo, l'oggetto 'URLLoader', incapsulato nell'oggetto 'XmlLoader', induce l'esecuzione di un metodo di quest'ultimo il quale, a sua volta, può informare l'oggetto 'Main' che sono disponibili i dati XML e che li può utilizzare per costruire, per esempio, i menu con le miniature e permettere la visualizzazione dei documenti (infatti tutti i percorsi alle miniature e relativi documenti sono informazioni tratte dai dati XML appena caricati). Va specificato che l'oggetto 'XmlLoader' comunica all'oggetto 'Main' il fatto che i dati XML sono disponibili, tramite l'emissione di un evento personalizzato: 'XmlAvailableEvent.XML_AVAILABLE'. In questo modo queste classi risultano svincolate tra di loro. Infatti comunicano per 'eventi', non per codici interdipendenti. Questo, come del resto per altre classi studiate in precedenza, permette alla classe 'XmlLoader' di essere utilizzata così com'è in ogni progetto che richieda l'utilizzo di dati testuali esterni.

L'oggetto 'Main' è in grado di utilizzare i dati XML chiamando il metodo 'data()' esposto dalla classe 'XmlLoader' che, a sua volta 'legge' il contenuto della proprietà 'data' dell'oggetto privato 'URLLoader'.

Nella figura che segue, Figura 4.12, abbiamo schematizzato l'utilizzo dei dati XML per la costruzione dei menù e dei documenti.

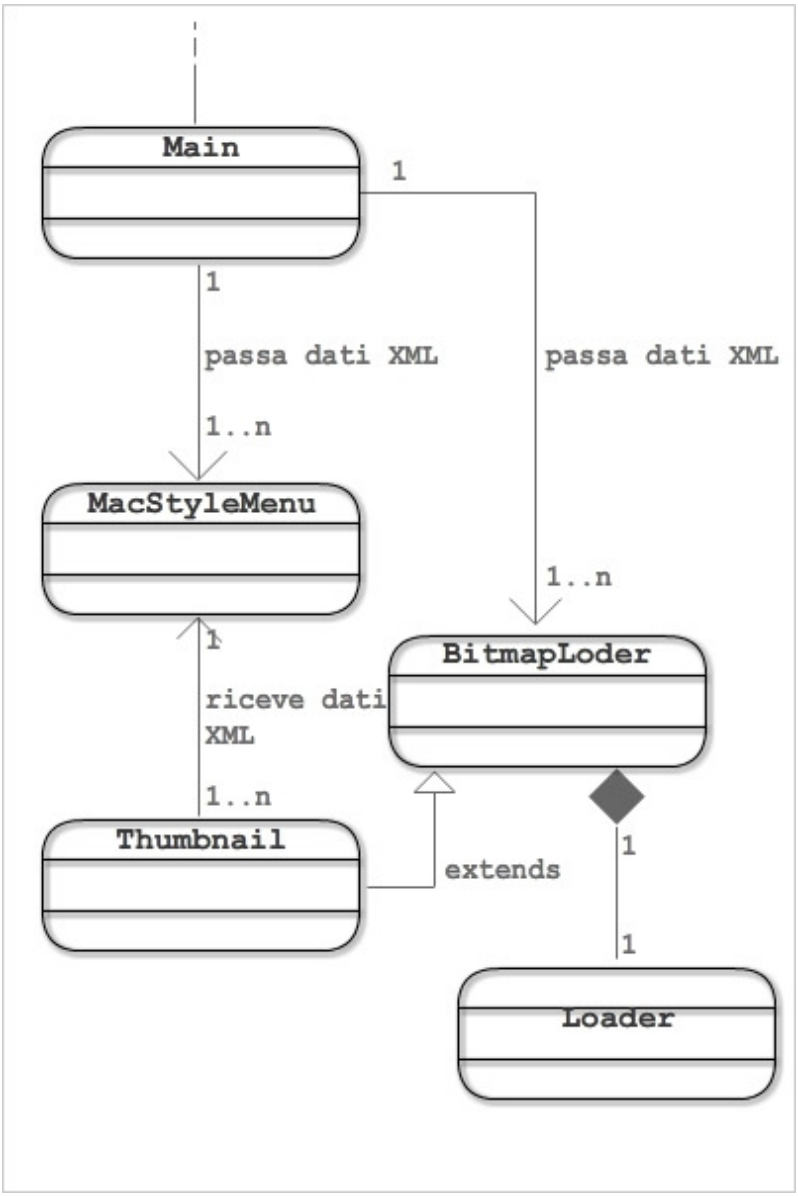

Figura 4.12 Utilizzo dei dati XML per la costruzione dei menu e i dei documenti.

Per completezza, sotto è riportato un piccolo esempio dei dati XML caricati dall'esterno ed utilizzati per costruire gli elementi dell'applicazione sopra descritti:

```
<immagini serverYear="2007" serverMonth="11" serverDate="20"
serverHour="10" serverMinute="58" serverSecond="56"
serverMillisecond="230">
      <immagine id="1" page="i">
            <thumbnail><![CDATA[thumbnails/d.jpg]]></thumbnail>
```

```
            <url><![CDATA[images/01.jpg]]></url>
            <tooltip><![CDATA[]]></tooltip>
            <link><![CDATA[]]></link>
        </immagine>
        <immagine id="2" page="ii">
            <thumbnail><![CDATA[thumbnails/b.jpg]]></thumbnail>
            <url><![CDATA[images/02.jpg]]></url>
            <tooltip><![CDATA[]]></tooltip>
            <link><![CDATA[]]></link>
        </immagine>
        <immagine id="3" page="iii">
            <thumbnail><![CDATA[thumbnails/c.jpg]]></thumbnail>
            <url><![CDATA[images/03.jpg]]></url>
            <tooltip><![CDATA[]]></tooltip>
            <link><![CDATA[]]></link>
        </immagine>
        <immagine id="4" page="iv">
            <thumbnail><![CDATA[thumbnails/a.jpg]]></thumbnail>
            <url><![CDATA[images/04.jpg]]></url>
            <tooltip><![CDATA[]]></tooltip>
            <link><![CDATA[]]></link>
        </immagine>
        <immagine id="5" page="v">
            <thumbnail><![CDATA[thumbnails/b.jpg]]></thumbnail>
          <url><![CDATA[images/0000000(19140101)XXIX_3152_002.jpg]]><url>
            <tooltip><![CDATA[]]></tooltip>
            <link><![CDATA[]]></link>
        </immagine>
</immagini>
```

- ✓ **Dettagli sulla gestione delle miniature: dividere il numero totale delle miniature in gruppi.**

Può succedere che l'insieme di documenti richiesto dall'utente sia costituito da un numero consistente di documenti. In questo caso non è proponibile gestire tutte le relative miniature subito in un unico menu in stile 'Mac'. Per questa ragione si è preferito suddividere l'insieme totale di documenti in più sottoinsiemi.

Un menu in stile 'Mac' visualizza le miniature relative ai documenti facenti parte di un dato sottoinsieme. Il numero di documenti costituenti un sottoinsieme è specificato da una variabile dedicata.

Ora, con questo valore, e il numero totale di documenti, è possibile ottenere il numero di menu in stile 'Mac' generabili dal programma. Per esempio: numero totale di documenti = 22, numero totale di documenti contenuti in un sottoinsieme = 5, si ottiene 22/5=4 con il resto di 2. Per cui il programma mette a disposizione un sistema interattivo in grado di generare 5 menu in stile 'Mac', 4 dei quali visualizzano 5 miniature e al quinto è dato il compito di visualizzare le ultime 2.

Questo sistema interattivo è generato a partire da 2 classi: la classe 'Sections' e la classe 'Circle'.

Dall'oggetto 'Main', viene istanziata una sola volta la classe 'Sections'.

Tale istanza rappresenta visivamente e logicamente il contenitore per gli elementi interattivi che, se cliccati, permettono, ognuno, di costruire il relativo menu in stile 'Mac' visualizzatore di un dato sottoinsieme di miniature.

Gli elementi interattivi appena citati sono istanze della classe 'Circle'.

Ogni oggetto 'Circle' ha 2 stati: attivo e disattivo.
Lo stato attivo è rappresentato da un cerchietto bianco, e significa che il relativo oggetto 'Circle' può essere cliccato per generare il corrispettivo menu di miniature. Una volta che un oggetto 'Circle' viene selezionato, passa allo stato 'disattivo', rappresentato da un cerchietto grigio. Questo significa che, mentre è visualizzato il menu di miniature corrispondente, non può essere ulteriormente selezionato. Appena si clicca su un altro oggetto 'Circle' sullo stato 'attivo', per costruire un nuovo menu di miniature, viene riattivato l'oggetto 'Circle' che si trovava nello stato di 'disattivo'.
Diamo rappresentazione di quanto appena spiegato nella Figura 4.13.

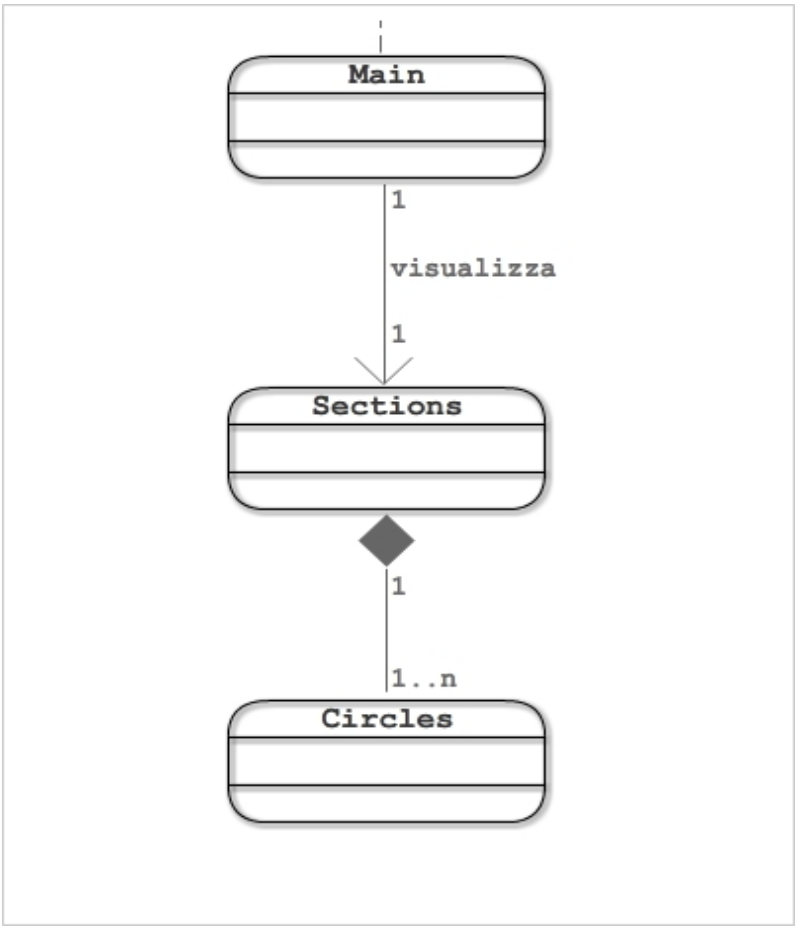

Figura 4.13: In evidenza le classi per l'implementazione dei pulsanti che permettono la creazione di menu in stile 'Mac'

A questo punto vale la pena sottolineare una caratteristica molto interessante che viene offerta dall'ambiente di sviluppo Adobe Flash ed ActionScript.
Sia la classe 'Sections' che la classe 'Circle', quando vengono istanziate producono non solo oggetti logici, ma ad essi sono associate controparti grafiche visibili dall'utente. Tali parti grafiche sono realizzate manualmente in Flash e, in un secondo tempo, dopo aver generato le relative classi, gli elementi grafici si possono associare in maniera trasparente ognuno alla sua classe di riferimento.
Questa associazione è implementabile tramite un opportuno pannello, la libreria, entro il quale sono elencati tutti gli elementi grafici.

Selezionando un elemento grafico di interesse ed accedendo alle sue proprietà, tramite una 'dialog box', è possibile legarlo ad una classe.

In questo modo, tutte le volte che il programma istanzia tale classe, genera sullo 'stage', una copia della componente grafica.

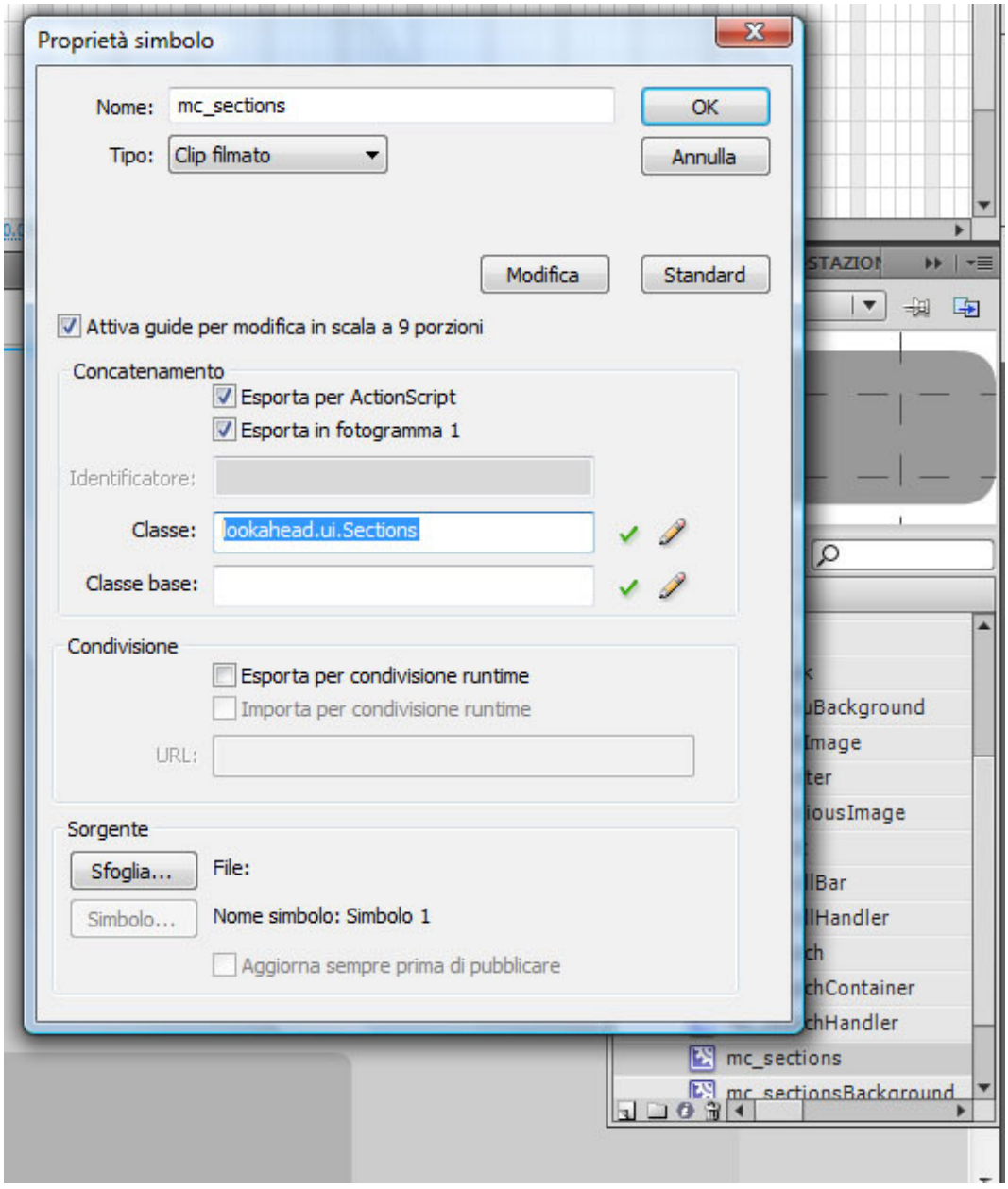

Figura 4.14: In questa immagine viene proposta l'interfaccia dalla quale associare un elemento grafico ad una classe ActionScript.

Si conclude qui la presentazione del progetto.

Nel prossimo capitolo verrà illustrato il codice di implementazione con i commenti descrittivi delle funzionalità.

Capitolo 5

Progetto DigitalLab: implementazione

In questa unità conclusiva del mio progetto, è esposto il codice sorgente dell'applicazione.

Sono elencate una ad una tutte le classi, di ognuna delle quali è evidenziato il package di appartenenza e l'eventuale legame tramite ereditarietà con altre classi. Tutti i punti del codice più importanti sono stati opportunamente e dettagliatamente commentati per permettere all'utente non solo di capirne il significato, ma anche per comprendere come la scrittura dello stesso codice ha tenuto in conto di tutte le argomentazioni trattate nei capitoli precedenti.

5.1 Classe 'Main'.

La classe 'Main', definita all'interno del file 'Main.as', è quella associata direttamente al file 'main.fla', e rappresenta l'area contenente tutti (o quasi) gli elementi grafico-interattivi che l'utente dell'applicazione può vedere ed utilizzare.

Tale classe è contenuta nel 'package' che, in AS3, è definito 'unnamed'. Questo 'package' sottende la 'directory' radice dell'applicazione.
Qui sotto è riportato il codice:

```
package
{
```

/*Questa direttiva permette alla classe corrente di utilizzare un insieme di classi le cui istanze rappresentano diversi elementi visuali e contenitori per tali elementi, da immagini 'bitmap', ad immagini vettoriali, da elementi animabili a contenitori per media caricati in 'run-time'.*/

```
import flash.display.*
```

/*La direttiva seguente permette alla classe corrente di utilizzare percorsi ed URL per accedere a risorse locali e remote, come dati 'XML' e pagine dinamiche.*/

```
import flash.net.*;
```

/*Questa direttiva abilita l'utilizzo e la gestione di eventi, da quelli emessi a causa dell'interazione del mouse con elementi interattivi, a quelli di sistema, come ad esempio gli eventi che informano l'applicazione che il caricamento di dati esterni è andato a buon fine.*/

```
import flash.events.*;
```

/*La direttiva seguente permette alla classe corrente di utilizzare le classi ActionScript 3.0 adibite alla stampa.*/

```
import flash.printing.*;
```

/*Con questa direttiva è possibile utilizzare le classi adibite alla gestione del testo e della sua formattazione. In particolare, questo 'package', mette a

disposizione l'uso della classe 'TextField' le cui istanze rappresentano visivamente campi di testo.*/

```
import flash.text.*
import flash.system.*;
```

/*Il seguente package permette l'uso di classi le cui istanze rappresentano elementi logico-matematici, come, ad esempio, punti nel piano, punti nello spazio 3D e matrici. */

```
import flash.geom.*;
import flash.external.*;
```

/*La classe 'Mouse' permette di interagire con l'icona che rappresenta il puntatore con cui l'utente interagisce con l'applicativo. */

```
import flash.ui.Mouse;
```

/*Queste tre direttive permettono di utilizzare, all'interno di questa classe, istanze delle classi create da me per rappresentare elementi specifici della libreria digitale. Sia elementi grafici, che elementi di utilità. */

```
import lookahead.loaders.*;
import lookahead.events.*;
import lookahead.ui.*;

public class Main extends MovieClip
{
/*coded by Luca Bonacorsi, lookahead=>.
Adobe Certified Instructor.*/
private static constNAVIGATION_BUTTONS_X_DISTANCE:Number=30;
```

/*Larghezza dell'area ('stage') contenente l'applicazione. */

```
public static const STAGE_WIDTH:uint=659;
```

/*Altezza dell'area ('stage') contenente l'applicazione. */

```
public static const STAGE_HEIGHT:Number=483;
```

/*Numero totale di miniature inseribili all'interno del menu in stile 'Mac'. */

```
private var _maxThumbnailsForMacStyleMenu:uint;
```

/*La variabile '_xmlLoader' conterrà un'istanza della classe 'XmlLoader' che incapsula la logica programmativa per il caricamento e gestione del caricamento di stringhe XML recuperate da locale o in remoto. */

```
private var _xmlLoader:XmlLoader;
```

/*La variabile seguente conterrà il percorso o URL ad una risorsa esterna di dati XML; */

```
private var _xmlPath:String;
```

/*La variabile '_xml' sarà un puntatore ad un'istanza della classe 'XML'. Tale istanza manterrà i valori XML, recuperati dall'esterno come semplici stringhe, sottoforma di liste di oggetti restituiti dall'azione del 'parser'. */

```
private var _xml:XML;
```

/*La variabile seguente sarà un puntatore ad un oggetto che, visivamente, rappresenterà il menu interattivo in stile Mac situato nella parte bassa dell'applicativo. Tale menu conterrà le miniature dei documenti. Tutte le miniature saranno interattive e permetteranno, tramite l'azione del 'mouse', di accedere ai documenti veri e propri che verranno visualizzati al centro dello 'stage'. */

```
private var macStyleMenu:MacStyleMenu;
```

/*Le due variabili successive, serviranno a calcolare quale miniatura dovrà vedersi all'estrema sinistra del menu e quale all'estrema destra. */

```
private var _firstImage:uint;
private var _lastImage:uint;
```

/*Questa variabile permetterà di gestire la distanza che intercorrerà tra il menu delle miniature e la parte inferiore dello 'stage'. */

```
private var _bottomYDistance:Number;
```

/*La variabile '_urls' manterrà una lista contenente i percorsi ai documenti visualizzabili nell'applicativo. Percorsi recuperati dai dati XML caricati dall'esterno. */

```
private var _urls:XMLList;
```

/*'_urlCounter' è un contatore che indica quale documento è correntemente visualizzato. */

```
private var _urlCounter:uint;
```

/*La variabile '_isLoaded' conerrà un valore 'Booleano'. Se 'true' un dato documento è stato caricato, se 'false' il documento richiesto è in fase di caricamento. */

```
private var _isLoaded:Boolean;
```

/*La variabile seguente conterrà puntatori ad istanze della classe personalizzata 'BitmapLoader', utili per caricare e gestire il caricamento di media e documenti esterni. */

```
private var _bitmapLoader:BitmapLoader;
```

/*Se la seguente variabile conterrà il valore 'true', significa che l'ingrandimento di un documento sta subendo, da parte dell'utente, un'azione di trascinamento. Altrimenti conterrà il valore 'false'. */

```
private var _isDragging:Boolean;
```

/*'_sections' manterrà un puntatore ad un oggetto che graficamente rappresenterà lo sfondo del menu delle miniature. */

```
private var _sections:Sections;
```

/*Numero totale di menu stile 'Mac' generabili in base al numero totale di miniature e relativi documenti visualizzabili nella libreria digitale. */

```
private var _totalCircles:Number;
```

/*Lista di oggetti, ognuno rappresentante un elemento grafico che, se cliccato, costruisce un nuovo menu stile 'Mac' con nuove miniature. */

```
private var _circles:Array;
```

/*Contatore per tenere traccia di quale, tra gli 'n' oggetti che se cliccati generano un nuovo menu stile 'Mac', è correntemente selezionato. */

```
private var _circleCounter:uint;
```

/*Se 'true', di un dato documento si sta visualizzando il rispettivo ingrandimento, se 'false' lo si sta visualizzando con dimensioni predefinite. */

```
private var _isEnlarged:Boolean;
```

/*Coordinate di un documento attivo, utilizzate per rendere corretto l'algoritmo di trascinamento di tale documento una volta ingrandito. */

```
private var _draggingX:Number;
private var _draggingY:Number;
```

/*Quale, tra gli 'n' documenti disponibili, è visualizzato durante la fase di 'start-up' dell'applicativo.*/

```
private var _initialPage:String;
```

/*Se il valore della variabile '_thumb' è '1', allora viene generato il menu in stile 'Mac'. Altrimenti no. */

```
private var _thumb:String;
```

/*I valori della variabile 'p2', permettono di centrare diversamente i documenti nello 'stage'. */

```
private var _p2:String;
```

/*Costruttore. Viene eseguito automaticamente una volta che si compila l'applicazione per generare il file '.swf'. */

```
public function Main()
{
```

/*Viene abilitata la possibilità di recuperare dati da domini diversi da quello che fornisce l'applicativo. */

```
Security.allowDomain('*');
```

/*L'istanza corrente di 'Main', che strutturalmente rappresenta il contenitore di tutti gli elementi grafico-testuali della libreria digitale, è in ascolto del momento in cui il sistema la riconosce come 'child' dell'oggetto 'stage', che, a sua volta, è la radice di tutti gli oggetti renderizzabili dal 'Flash Player'. L'oggetto

'stage', che è creato automaticamente da AS3, rappresenta l'area visibile dall'utente. */

```
addEventListener(Event.ADDED_TO_STAGE,onAddedToStage);
}
```

/*Questo metodo viene eseguito nel momento in cui l'oggetto 'Main' è riconosciuto 'child' dell'oggetto 'stage'. */

```
private function onAddedToStage(e:Event):void
{
```

/*Ora l'oggetto 'Main' è sicuramente inserito nella 'child-list' di 'stage'. Occorre, quindi, rimuovere l'ascolto dell'evento ' Event.ADDED_TO_STAGE' per ragioni di performance e gestione della memoria. */

```
removeEventListener(Event.ADDED_TO_STAGE,onAddedToStage);
//initializeParameters('','','','','');
```

/*Questa funzione imposta le coordinate iniziali del pannello attraverso il quale si possono ricercare documenti specifici all'interno della libreria. */

```
positionSearchContainer();
```

/*Qui vengono attivati dei temporizzatori per permettere, ad alcuni elementi dell'interfaccia, di muoversi tramite animazioni. */

```
addEventListener(Event.ENTER_FRAME,onManageCursor);
stage.addEventListener(Event.MOUSE_LEAVE,onMouseLeave);
addEventListener(Event.ENTER_FRAME,onManageMenuBackground);
addEventListener(Event.ENTER_FRAME,onManageSearchContainer);
```

/*Queste tre funzioni inizializzano componenti e funzionalità dell'applicazione e impostano i valori iniziali nelle proprietà dell'istanza 'Main'. */

```
initializeFullScreen();
initializeStage();
initialize();
```

/*L'applicativo, grazie a questa funzione, è in grado di capire quando deve visualizzarsi a tutto schermo o con dimensioni standard. */

```
                activateFullScreenListeners();
```

/*Questa funzione induce il caricamento dei dati '.xml' contenenti i percorsi sia alle miniature dei documenti, sia ai documenti stessi. */

```
                getData();
        }

        public function
initializeParameters(xmlPath:String,maxThumbnails:String,page:Stri
ng,thumb:String,p2:String):void
        {
                if(xmlPath!='')
                        _xmlPath=xmlPath;
                else
                        _xmlPath='data.xml';

                if(maxThumbnails!='')

    _maxThumbnailsForMacStyleMenu=uint(maxThumbnails);
                else
                        _maxThumbnailsForMacStyleMenu=3;

                if(page!=''&&page!='0')
                        _initialPage=page;
                else
                        _initialPage=null;

                if(thumb!='')
                        _thumb=thumb;
                else
                        _thumb='1';

                if(thumb!='')
                        _p2=p2;
                else
                        _p2='0';
        }
```

/*La funzione 'initializeFullScreen()' permette all'oggetto 'stage' di essere in grado di rilevare quando l'utente desidera fruire della libreria digitale a tutto schermo oppure desidera ripristinarla alle dimensioni normali.
Tutte le volte che l'utente clicca col 'mouse' su aree specifiche dell'applicazione, viene eseguita, a sua volta la funzione 'onToggleFullscreen()'. */

```
            private function initializeFullScreen():void
            {

      stage.addEventListener(FullScreenEvent.FULL_SCREEN,onToggleF
ullscreen);
            }
```

/*Questa funzione, a seconda del valore della variabile ' e.fullScreen' ('true' o 'false'), è in grado di rimodellare l'applicativo per la visualizzazione a tutto schermo, o ripristinare le dimensioni originali. */

```
      private function onToggleFullscreen(e:FullScreenEvent)
      {
            if(!e.fullScreen)
            {
                  _isEnlarged=false;

                  activateFullScreenListeners();

                  searchContainer_mc.visible=true;
```

/*Se, al momento del ripristino alle normali dimensioni della libreria digitale, è correntemente attivo un documento (istanza della classe 'BitmapLoader' che incapsula la 'bitmap'), tale istanza viene ridimensionata tramite un calcolo in percentuale che tiene conto sia delle effettive dimensioni della 'bitmap', sia delle dimensioni dello 'stage'. Questi calcoli sono eseguiti proprio da metodi della classe 'BitmapLoader' che sono, nello specifico, 'resizeBitmap()' oppure 'resizeByStageWidth()' a seconda del valore associato alla proprietà '_p2'. */

```
            if(_bitmapLoader!=null)
            {
                  _bitmapLoader.positionPreload();

                  if(_p2=='0')
                        _bitmapLoader.resizeBitmap();
                        else

_bitmapLoader.resizeByStageWidth();
```

/*Una volta che il documento è ridimensionato, viene ricentrato nell'applicazione. */

```
                  _bitmapLoader.centerBitmap();
      }
}
      else
```

```
{
	deactivateFullScreenListeners();

	searchContainer_mc.visible=false;
```

/*Nel caso in cui si debba configurare la visualizzazione a tutto schermo della libreria digitale, il primo controllo ad essere eseguito è quello che verifica la presenza di un documento attivo. Nel caso ci sia, l'istanza della classe 'BitmapLoader' che incapsula la 'bitmap', viene ridimensionata, eseguendo il suo metodo 'resizeBitmapForFullScreen()', in maniera tale che vada a riempire tutta l'area visibile. */

```
		if(_bitmapLoader!=null)
			{
				_bitmapLoader.positionPreload();
_bitmapLoader.resizeBitmapForFullScreen(stage.stageHeight,stage.st
ageWidth);
```

/*Una volta che il documento è ridimensionato, viene ricentrato nell'applicazione. */

```
				_bitmapLoader.centerBitmap();
		}
	}
	setLayout();
	alignBitmapLoader();

	if(_thumb=='1')

	{
		centerMacStyleMenu();
		manageMacStyleMenuVisibility();
		manageMenuBackgroundVisibility();
		centerSections();
		manageSectionsVisibility();
	}
	positionNavigationButtons();

	if(_bitmapLoader!=null)

		manageScrollBar();
	}
	private function toggleFullscreen(e:MouseEvent):void
```

```
{
```

/*Se un dato 'click' dell'utente col 'mouse', è fatto in maniera tale che le coordinate stesse del 'mouse' intersechino le aree di alcuni elementi grafici, allora, tramite l'istruzione 'return', si interrompe prematuramente l'esecuzione della funzione, poiché non si intende far eseguire il codice successivo a questo controllo 'if'. Il codice successivo, infatti, è quello che gestisce, come un interruttore, la visualizzazione della libreria digitale: a schermo intero oppure no. */

```
                if(e.target.name=='search_txt'||
                   e.target.name=='search_mc'||
                   e.target.name=='print_mc'||
                   e.target==previousImage_mc||
                   e.target==nextImage_mc||
                   e.target is ScrollHandler)
                      return;
```

/*A seconda del valore della proprietà 'displayState' dell'oggetto 'stage', il sistema è in grado di determinare se attivare la funzionalità di schermo intero o viceversa ripristinare le dimensioni normali dell'applicazione. Questo codice viene eseguito se e solo se il 'click' del 'mouse' non è avvenuto su specifici elementi interrativi. */

```
                if(stage.displayState=='fullScreen')
                {
                      stage.displayState='normal';
                }
                else
                {
                      stage.displayState='fullScreen';
                }
        }
```

/*Questa funzione determina come lo 'stage' si debba comportare una volta che viene indotto il ridimensionamento dell'applicazione. La proprietà 'stage' della classe 'Main' (come di ogni altra classe le cui istanze rappresentano elementi renderizzabili nel 'Flash Player') è un puntatore all'istanza della classe 'Stage' che rappresenta l'area visibile dell'applicazione. */

```
private function initializeStage():void
{
	stage.scaleMode=StageScaleMode.NO_SCALE;
	stage.align=StageAlign.TOP_LEFT;
}
```

/*Questa funzione permette alle grafiche di fondo di essere costantemente delle dimensioni dello 'stage' e in esso centrate (occupano sempre l'area visibile). */

```
private function setLayout():void
{
```

/*Alle larghezze di 'mask_mc' e di 'background_mc' è assegnata la larghezza dello 'stage'. */

```
mask_mc.width=background_mc.width=stage.stageWidth;
```

/*Alle altezze di 'mask_mc' e di 'background_mc' è assegnata l'altezza dello 'stage'. */

```
mask_mc.height=background_mc.height=stage.stageHeight;
```

/*'mask_mc' e 'background_mc' sono centrati rispetto lo 'stage'.

```
	mask_mc.x=background_mc.x=(stage.stageWidth/2)-
(mask_mc.width/2);
	mask_mc.y=background_mc.y=(stage.stageHeight/2)-
(mask_mc.height/2);
	}

	private function alignBitmapLoader():void
	{
```

/*Se il documento selezionato è stato caricato correttamente all'interno della libreria digitale allora, in base alla sua altezza rapportata a quella dello 'stage' viene deciso come posizionarlo. Nel caso la sua altezza sia maggiore di quella dello 'stage', i bordi superiori di entrambi vanno a collimare e si procederà con l'attivazione di una barra di scorrimento per permettere la visualizzazione delle parti nascoste, oppure il documento è semplicemente centrato nello 'stage', in quanto non presenta parti nascoste. */

```
		if(_isLoaded)
		{
			if(_bitmapLoader.height>stage.stageHeight)
```

```
                        alignBitmapLoaderOnTop();
                  else
                        centerBitmapLoader();
            }
      }

      private function alignBitmapLoaderOnTop():void
      {
            _bitmapLoader.x=stage.stageWidth/2;
            _bitmapLoader.y=_bitmapLoader.height/2;
      }

      private function centerBitmapLoader():void
      {
            _bitmapLoader.x=stage.stageWidth/2;
            _bitmapLoader.y=stage.stageHeight/2;
      }
```

/*Qui vengono inizializzate le proprietà di questo oggetto 'Main'. */

```
      private function initialize():void
      {
            _urlCounter=0;
            _isLoaded=false;
            _macStyleMenus=new Array();
            _macStyleMenusCounter=0;
            _firstImage=0;
            _lastImage=_maxThumbnailsForMacStyleMenu;
            _bottomYDistance=30;
            _isDragging=false;
            _totalCircles=1;
            _circles=new Array();
            _circleCounter=0;
            _isEnlarged=false;
            _isDraggingBitmapLoader=false;
            menuBackground_mc.visible=false;
      }
```

/*Questa funzione gestisce la posizione iniziale dell'elemento grafico-interattivo che permette, attraverso un campo di input, di ricercare specifici documenti all'interno della libreria digitale. L'elemento viene, per quanto riguarda l'asse orizzontale, centrato rispetto allo 'stage'. Mentre, per quanto riguarda l'asse verticale, viene posizionato oltre il borde superiore dello 'stage', in maniera tale che sia visibile solo una maniglietta con la quale l'utente potrà interagire per mostrare (o nascondere di nuovo), l'intero pannello. */

```
      private function positionSearchContainer():void
      {
            searchContainer_mc.x=stage.stageWidth/2-
searchContainer_mc.width/2;
```

```
                searchContainer_mc.y=-
searchContainer_mc.height+searchContainer_mc.searchHandler_mc.heig
ht;
            }
            private function getData():void
            {
```

/*Un'istanza della classe 'XmlLoader' incapsula tutta la logica necessaria per indurre e gestire il caricamento di dati '.xml' esterni. Nel caso specifico, inizia il caricamento di stringhe '.xml' inscritte in un 'file' recuperabile al percorso specificato dalla variabile '_xmlPath'. Se il caricamento dei dati '.xml' si conclude con successo, l'istanza '_xmlLoader' emette l'evento personalizzato 'XmlAvailableEvent.XML_AVAILABLE' di cui è in ascolto. L'emissione di tale evento induce l'esecuzione della funzione ' onXmlAvailable()'. */

```
                _xmlLoader=new XmlLoader(_xmlPath);

      _xmlLoader.addEventListener(XmlAvailableEvent.XML_AVAILABLE,
onXmlAvailable);
            }

            private function
onXmlAvailable(e:XmlAvailableEvent):void
            {
                  _xml=new XML(_xmlLoader.data);
                  _urls=_xml..url;

                  if(_urls.length()<_maxThumbnailsForMacStyleMenu)
                        _lastImage=_urls.length();


      if(_urls.length()>=_maxThumbnailsForMacStyleMenu)

      _totalCircles=Math.ceil(_urls.length()/_maxThumbnailsForMacS
tyleMenu);


                  if(_thumb=='1')
                  {
                        manageMacStyleMenu();
                        manageMacStyleMenuVisibility();
                        manageMenuBackgroundVisibility();
                        buildSections();
                        buildCircles();

      deactivateCircles(_circles[_circleCounter]);
                        centerSections();
                        manageSectionsVisibility();
                  }
```

```
                positionNavigationButtons();

                if(_initialPage!=null)
                      checkInitialPage();
                else
                {
                      if(_thumb=='1')
                      {

      macStyleMenu.setThumbnailSelected(_urlCounter+1);
                      }

                      buildImage();
                }
          }

          private function onManageCursor(e:Event):void
          {
                var isMenuOverAnItem:Boolean=false;

                if(macStyleMenu!=null)
                {

      if(macStyleMenu.hitTestPoint(stage.mouseX,stage.mouseY))
                      {
                            isMenuOverAnItem=true;
                      }
                }


      if(previousImage_mc.hitTestPoint(stage.mouseX,stage.mouseY)|
|

nextImage_mc.hitTestPoint(stage.mouseX,stage.mouseY))
                {
                      isMenuOverAnItem=true;
                }

      if(searchContainer_mc.hitTestPoint(stage.mouseX,stage.mouseY
)&&
                   stage.displayState=='normal')
                {
                      isMenuOverAnItem=true;
                }

                if(_sections!=null)
                {

      if(_sections.hitTestPoint(stage.mouseX,stage.mouseY))
                      {
                            isMenuOverAnItem=true;
                      }
                }

                if(bar!=null)
                {
```

```
      if(bar.hitTestPoint(stage.mouseX,stage.mouseY)||

handler.hitTestPoint(stage.mouseX,stage.mouseY))
                        {
                              isMenuOverAnItem=true;
                        }
                  }

                  if(isMenuOverAnItem)
                  {
                        len_mc.x=-len_mc.width/2;
                        len_mc.y=len_mc.height/2;

                        grab_mc.x=-grab_mc.width/2;
                        grab_mc.y=grab_mc.height/2;

                        Mouse.show();
                  }
                  else
                  {
                        Mouse.hide();

                        if(_isEnlarged)
                        {

      grab_mc.x=stage.mouseX+grab_mc.width/2;

      grab_mc.y=stage.mouseY+grab_mc.height/2;

                              len_mc.x=-len_mc.width/2;
                              len_mc.y=len_mc.height/2;

                              setChildIndex(grab_mc,numChildren-
1);
                        }
                        else
                        {

      len_mc.x=stage.mouseX+len_mc.width/2;

      len_mc.y=stage.mouseY+len_mc.height/2;

                              grab_mc.x=-grab_mc.width/2;
                              grab_mc.y=grab_mc.height/2;

                              setChildIndex(len_mc,numChildren-1);
                        }
                  }
            }

            private function onMouseLeave(e:Event):void
            {
                  trace('onMouseLeave().');
```

```
	removeEventListener(Event.ENTER_FRAME,onManageCursor);

			len_mc.x=-len_mc.width/2;
			len_mc.y=len_mc.height/2;
			//len_mc.visible=false;

			grab_mc.x=-grab_mc.width/2;
			grab_mc.y=grab_mc.height/2;
			//grab_mc.visible=false;

			Mouse.show();


	stage.addEventListener(MouseEvent.MOUSE_MOVE,mouseOff);
		}

		function mouseOn(evt: MouseEvent)
		{

	stage.removeEventListener(MouseEvent.MOUSE_MOVE,mouseOn);

	addEventListener(Event.ENTER_FRAME,onManageCursor);
		}

		function mouseOff(evt: Event)
		{

	stage.addEventListener(MouseEvent.MOUSE_MOVE,mouseOn);
		}

		private function onManageMenuBackground(e:Event):void
		{
			var thumbsXdistance:Number=10;
			var
halfThumbsXdistance:Number=thumbsXdistance/2;
			var yOffset:Number=30;
			var padding:Number=8;

			if(macStyleMenu!=null)
			{
				menuBackground_mc.x=macStyleMenu.x-
halfThumbsXdistance;
				menuBackground_mc.y=macStyleMenu.y-
yOffset;

	menuBackground_mc.width=macStyleMenu.width+padding;

	menuBackground_mc.height=macStyleMenu.height+padding;
			}
		}

		private function onManageSearchContainer(e:Event):void
		{
			var factor:Number=3;
```

```
                searchContainer_mc.x=stage.stageWidth/2-
searchContainer_mc.width/2;

                if(stage.mouseY<14)
                {
                      searchContainer_mc.y+=(0-
searchContainer_mc.y)/factor;
                }
                else
                {
                      if(stage.mouseY>searchContainer_mc.height)
                            searchContainer_mc.y+=((-
searchContainer_mc.height+searchContainer_mc.searchHandler_mc.heig
ht)-searchContainer_mc.y)/factor;
                }
          }

          private function manageMacStyleMenu():void
          {
                removeMacStyleMenu();
                buildMacStyleMenu();
                centerMacStyleMenu();
          }
```

/*Questa funzione rimuove un menu in stile 'Mac'.*/

```
          private function removeMacStyleMenu():void
          {
                if(macStyleMenu!=null)
                {
                      macStyleMenu.deactivateListeners();

                      removeChild(macStyleMenu);

                      macStyleMenu=null;
                }
          }
```

/*Questa funzione costruisce un menu in stile 'Mac'. A tale oggetto vengono passate le informazioni 'XML' contenenti i dati relativi alle miniature dei documenti da visualizzare e gestire.*/

```
          private function buildMacStyleMenu():void
          {
                var o:XML=<immagini></immagini>;

                for(var i:uint=_firstImage;i<_lastImage;i++)
                {
                      o.appendChild(_xml.immagine[i].copy());
                }

                macStyleMenu=new
MacStyleMenu(o,2,360,5,"bottom");
```

```
            addChild(macStyleMenu);
        }
```

/*Il menu in stile 'Mac' viene centrato orizzontalmente nello 'stage' e verticalmente nella parte bassa dello 'stage' stesso.*/

```
        private function centerMacStyleMenu():void
        {
            macStyleMenu.x=stage.stageWidth/2;
            macStyleMenu.y=stage.stageHeight-
_bottomYDistance;
        }
```

/*Se l'applicativo è in visualizzazione a schermo intero, il menu in stile 'Mac' viene reso visibile, altrimenti viene reso invisibile.*/

```
        private function manageMacStyleMenuVisibility():void
        {
            if(stage.displayState=='normal')
                macStyleMenu.visible=false;
            else if(stage.displayState=='fullScreen')
                macStyleMenu.visible=true;
        }
```

/*Se l'applicativo è in visualizzazione a schermo intero, lo sfondo grafico del menu in stile 'Mac' viene reso visibile, altrimenti viene reso invisibile.*/

```
        private function manageMenuBackgroundVisibility():void
        {
            if(stage.displayState=='normal')
                menuBackground_mc.visible=false;
            else if(stage.displayState=='fullScreen')
                menuBackground_mc.visible=true;
        }
```

/*Viene costruita una nuova istanza della classe 'Sections' che, graficamente, rappresenta lo sfondo degli oggetti 'Circle', i pulsanti che permettono di passare a specifici insiemi di documenti.*/

```
        private function buildSections():void
        {
            _sections=new Sections();

            addChild(_sections);
        }
```

/*Lo sfondo dei pulsanti 'Circle', viene centrato orizzontalmente sullo 'stage' e verticalmente viene collocato nella parte basse dello 'stage'.*/

```
            private function centerSections():void
            {
                  _sections.x=stage.stageWidth/2-
_sections.width/2;
                  _sections.y=stage.stageHeight-_sections.height;
            }
```

/*Questa funzione costruisce tanti pulsanti 'Circle' quanti sono gli insiemi di documenti. Ogni insieme ha altrettante miniature visualizzate da un menu in stile 'Mac'.*/

```
            private function buildCircles():void
            {
                  var circle:Circle;
                  var xOffset:Number=5;
                  var xDistance:Number=3;
                  var n:uint;
                  var urlCounter:uint;
                  var firstImage:uint;
                  var lastImage:uint;

                  for(var i:uint=0;i<_totalCircles;i++)
                  {
                        n=i;

      urlCounter=i*_maxThumbnailsForMacStyleMenu;

      firstImage=i*_maxThumbnailsForMacStyleMenu;

                        if(i==_totalCircles-1)
                              lastImage=_urls.length();
                        else

      lastImage=firstImage+_maxThumbnailsForMacStyleMenu;

                        circle=new Circle(n,
                                          urlCounter,
                                          firstImage,
                                          lastImage);


      circle.x=xOffset+i*(circle.width+xDistance);
                        circle.y=_sections.height/2-
circle.height/2;

                        _circles.push(circle);

                        _sections.addChild(circle);
                  }
_sections.sectionsBackground.width=circle.x+circle.width+xOffset;
            }
```

/*Se l'applicativo è in visualizzazione a schermo intero, lo sfondo grafico dei pulsanti 'Circle' e i pulsanti 'Circle' stessi, in esso contenuti, sono resi visibili, altrimenti viene resi invisibili.*/

```
private function manageSectionsVisibility():void
{
      if(stage.displayState=='normal')
            _sections.visible=false;
      else if(stage.displayState=='fullScreen')
            _sections.visible=true;
}
```

/*Questa funzione posiziona i pulsanti per la navigazione al documento precedente e successivo. Il pulsante che permette di accedere al documento precedente, qual'ora ci sia, viene collocato sulla sinistra dello schermo. L'altro sulla destra. Entrambi centrati verticalmente sullo 'stage'.*/

```
private function positionNavigationButtons():void
{
previousImage_mc.x=previousImage_mc.width+Main.NAVIGATION_BUTTONS_
X_DISTANCE;
      previousImage_mc.y=stage.stageHeight/2;

      nextImage_mc.x=stage.stageWidth-
previousImage_mc.width-Main.NAVIGATION_BUTTONS_X_DISTANCE;
      nextImage_mc.y=stage.stageHeight/2;
}
```

/*Nel caso venisse selezionato un nuovo documento, l'eventuale documento precedente viene eliminato.*/

```
private function removeBitmapLoader():void
{
      if(_bitmapLoader!=null)
      {

_bitmapLoader.removeEventListener(Event.INIT,onInit);

_bitmapLoader.removeEventListener(Event.COMPLETE,onComplete)
;
            _bitmapLoader.dispose();

            removeChild(_bitmapLoader);
            _bitmapLoader=null;
      }
}
```

/*Questa funzione abilita l'accesso ad un nuovo documento.*/

```
      private function buildBitmapLoader(path:String):void
```

```
		{
			_bitmapLoader=new BitmapLoader(path);


	_bitmapLoader.addEventListener(Event.INIT,onInit);

	_bitmapLoader.addEventListener(Event.COMPLETE,onComplete);

			addChild(_bitmapLoader);
		}
```

/*Quando un documento completa la fase di caricamento, a seconda di come viene visualizzato l'applicativo (a schermo intero oppure no), viene deciso come dev'essere ridimensionato.*/

```
		private function onComplete(e:Event):void
		{
			_isLoaded=true;

			if(stage.displayState=='normal')
			{
				if(_p2=='0')
					_bitmapLoader.resizeBitmap();
				else
					_bitmapLoader.resizeByStageWidth();
			}
			else
			{
_bitmapLoader.resizeBitmapForFullScreen(stage.stageHeight,stage.st
ageWidth);
			}

			_bitmapLoader.centerBitmap();

			alignBitmapLoader();
			activateListeners();
			manageScrollBar();
			orderItems();
		}

		private function checkInitialPage():void
		{
			//if(_initialPage!=null)
			{

	searchContainer_mc.search_txt.text=_initialPage;

				_initialPage=null;

				onSearch(new
MouseEvent(MouseEvent.CLICK));
			}
		}
```

```
		private function orderItems():void
		{
			if(_thumb=='1')
			{

	setChildIndex(menuBackground_mc,numChildren-1);
				setChildIndex(macStyleMenu,numChildren-1);
			}


			setChildIndex(previousImage_mc,numChildren-1);
			setChildIndex(nextImage_mc,numChildren-1);

			setChildIndex(bar,numChildren-1);
			setChildIndex(handler,numChildren-1);

			if(_thumb=='1')
			{
				setChildIndex(_sections,numChildren-1);
			}


		}

		private function jumpeToImage(n:Number):void
		{
			trace('n: '+n);

			_urlCounter=n;

			if(_thumb=='1')
			{
				macStyleMenu.setThumbnailSelected(n+1);
			}

			deactivateListeners();
			deactivateScrollBar();
			reset();
			buildImage();
		}

		private function onPreviousImage(e:MouseEvent)
		{
			var totalImages:uint=_urls.length();
			var totalUrls:uint=totalImages-1;
			var m:uint;


if(_urlCounter>0)
			{
				_urlCounter--;

				deactivateListeners();
				deactivateScrollBar();
				reset();
				buildImage();
```

```
if(_urlCounter)
                        {


      if(!((_urlCounter+1)%(_maxThumbnailsForMacStyleMenu)))
                              {

      m=_lastImage%_maxThumbnailsForMacStyleMenu;
//Main.MAX_THUMBNAILS_FOR_MAC_STYLE_MENU;

                                    if(!m)

      m=_maxThumbnailsForMacStyleMenu;

                                    _firstImage-
=_maxThumbnailsForMacStyleMenu;
//Main.MAX_THUMBNAILS_FOR_MAC_STYLE_MENU;
                                    _lastImage-=m;
```

/*Se il valore di '_thumb' è '1', viene gestita l'eventuale reimpostazione del menu in stile 'Mac'.*/

```
            if(_thumb=='1')
            {
                  manageMacStyleMenu();
```

/*Viene decrementato il contatore delle istanze della classe 'Circle'.*/

```
                        _circleCounter--;
```

/*Vengono attivate tutte le istanze della classe 'Circle', tranne quella attualmente selezionata che, cambiando di stato, non risulta più attiva.*/

```
      deactivateCircles(_circles[_circleCounter]);

      manageMacStyleMenuVisibility();

      manageMenuBackgroundVisibility();

      manageSectionsVisibility();
                                    }
                              }
                        }
```

/*Se il valore di '_thumb' è '1', allora imposto la miniatura successiva come selezionata.*/

```
                        if(_thumb=='1')
```

```
macStyleMenu.setThumbnailSelected(_urlCounter+1);

            }
      }

      private function onNextImage(e:MouseEvent)
      {
```

/*Numero totale di documenti.*/

```
            var totalImages:uint=_urls.length();
```

/*Numero totale di percorsi.*/

```
            var totalUrls:uint=totalImages-1;

            if(_urlCounter<totalUrls)
            {
                  _urlCounter++;

                  deactivateListeners();
                  deactivateScrollBar();
                  reset();
                  buildImage();


      if(!(_urlCounter%_maxThumbnailsForMacStyleMenu))

      //if(!(_urlCounter%Main.MAX_THUMBNAILS_FOR_MAC_STYLE_MENU))
                  {

      _firstImage+=_maxThumbnailsForMacStyleMenu;
//Main.MAX_THUMBNAILS_FOR_MAC_STYLE_MENU;

      _lastImage+=_maxThumbnailsForMacStyleMenu;
//Main.MAX_THUMBNAILS_FOR_MAC_STYLE_MENU;

                        if(_lastImage>totalImages)
                              _lastImage=totalImages;

                        if(_thumb=='1')
                        {
                              manageMacStyleMenu();
                              _circleCounter++;

      deactivateCircles(_circles[_circleCounter]);

      manageMacStyleMenuVisibility();

      manageMenuBackgroundVisibility();
                              manageSectionsVisibility();
                        }
                  }
```

```
                        if(_thumb=='1')
                        {

      macStyleMenu.setThumbnailSelected(_urlCounter+1);
                        }
                  }
            }

            private function
goToAnotherMacStyleMenu(o:Circle):void
            {
                  _urlCounter=o.urlCounter;

                  deactivateListeners();
                  deactivateScrollBar();
                  reset();
                  buildImage();

                  _firstImage=o.firstImage;
                  _lastImage=o.lastImage;

                  manageMacStyleMenu();
            }

            private function activateFullScreenListeners():void
            {

      removeEventListener(MouseEvent.CLICK,toggleFullscreen);

      addEventListener(MouseEvent.CLICK,toggleFullscreen)
            }

            private function deactivateFullScreenListeners():void
            {

      removeEventListener(MouseEvent.CLICK,toggleFullscreen);
            }

            private function activateListeners():void
            {
                  var circle:Circle;

                  previousImage_mc.buttonMode=true;
                  previousImage_mc.mouseChildren=false;

      previousImage_mc.addEventListener(MouseEvent.CLICK,onPreviou
sImage);

                  nextImage_mc.buttonMode=true;
                  nextImage_mc.mouseChildren=false;

      nextImage_mc.addEventListener(MouseEvent.CLICK,onNextImage);

                  handler.buttonMode=true;
                  handler.mouseChildren=false;
```

```
      stage.addEventListener(MouseEvent.MOUSE_DOWN,onDown);

      stage.addEventListener(MouseEvent.MOUSE_UP,onUp);

                  if(_thumb=='1')
                  {
                        for(var i:uint=0;i<_totalCircles;i++)
                        {
                              circle=_circles[i];

                              if(i!=_circleCounter)
                                    circle.buttonMode=true;

                              circle.mouseChildren=false;
                        }
                  }

                  searchContainer_mc.search_mc.buttonMode=true;

      searchContainer_mc.search_mc.mouseChildren=false;

      searchContainer_mc.search_mc.addEventListener(MouseEvent.CLI
CK,onSearch);

                  searchContainer_mc.print_mc.buttonMode=true;
                  searchContainer_mc.print_mc.mouseChildren=false;

      searchContainer_mc.print_mc.addEventListener(MouseEvent.CLIC
K,onPrint);
            }

            private function deactivateListeners():void
            {
                  var circle:Circle;

                  previousImage_mc.buttonMode=false;

      previousImage_mc.removeEventListener(MouseEvent.CLICK,onPrev
iousImage);

                  nextImage_mc.buttonMode=false;

      nextImage_mc.removeEventListener(MouseEvent.CLICK,onNextImag
e);

                  handler.buttonMode=false;

      stage.removeEventListener(MouseEvent.MOUSE_DOWN,onDown);

      stage.removeEventListener(MouseEvent.MOUSE_UP,onUp);

                  if(_thumb=='1')
                  {
                        for(var i:uint=0;i<_totalCircles;i++)
                        {
```

```
                        circle=_circles[i];
                        circle.buttonMode=false;
                  }
            }
      }

      private function reset():void
      {
            _isLoaded=false;
            _isEnlarged=false;
      }

      private function buildImage():void
      {
            removeBitmapLoader();
            buildBitmapLoader(_urls[_urlCounter]);
      }

      private function deactivateCircles(o:Circle):void
      {
            var circle:Circle;

            for(var i:uint=0;i<_totalCircles;i++)
            {
                  circle=_circles[i];
                  circle.gotoAndStop(1);
                  circle.buttonMode=false;
            }

            o.gotoAndStop(2);
            o.buttonMode=false;
      }

      private function onSearch(e:MouseEvent):void
      {
            var
page:String=searchContainer_mc.search_txt.text;

            page=page.toLowerCase();

            getIdFromPage(page);
      }

      private function getIdFromPage(page:String):void
      {
            var id:uint;
            var idAttribute:uint;
            var isFound:Boolean=false;
            var pages:Array;

            for each(var o:XML in _xml.immagine)
            {
                  if(o.@page.indexOf(',')>-1)
                  {
                        pages=o.@page.split(',');
```

```
                              for(var i:uint=0;i<pages.length;i++)
                              {
                                    if(pages[i]==page)
                                    {
                                          idAttribute=uint(o.@id);
                                          id=idAttribute-1;
                                          isFound=true;

                                          break;
                                    }
                              }
                        }
                        else
                        {
                              if(o.@page==page)
                              {
                                    idAttribute=uint(o.@id);
                                    id=idAttribute-1;
                                    isFound=true;

                                    break;
                              }
                        }
                  }

                  if(isFound)
                  {
                        if(_thumb=='1')
                        {

      _circleCounter=Math.ceil(idAttribute/_maxThumbnailsForMacSty
leMenu)-1;


      _firstImage=_circles[_circleCounter].firstImage;

      _lastImage=_circles[_circleCounter].lastImage;

                        manageMacStyleMenu();

      deactivateCircles(_circles[_circleCounter]);
                        manageMacStyleMenuVisibility();
                        manageMenuBackgroundVisibility();
                        manageSectionsVisibility();
                        }

                        jumpeToImage(id);
                  }
            }
```

/*Gestione barra di scorrimento.*/

```
            private function manageScrollBar():void
            {
                  if(_isLoaded)
                  {
```

```
                        if(checkHeightsDifference())
                              activateScrollBar();
                        else
                              deactivateScrollBar();
                  }
                  else
                        deactivateScrollBar();
            }

            private function checkHeightsDifference():Boolean
            {
                  if(_bitmapLoader.height>stage.stageHeight)
                        return(true);
                  else
                        return(false);
            }
```

/*Questa funzione attiva la barra di scorrimento.*/

```
            private function activateScrollBar():void
            {
                  var heightDifference:Number=bar.height-
handler.height;
                  var totalYLength:Number=stage.stageHeight-
_bitmapLoader.height;
                  var halfYLength:Number=totalYLength/2;
                  var percentage:Number;

                  if(stage.displayState=='normal')
                  {

      bar.x=_bitmapLoader.x+_bitmapLoader.width/2-bar.width;
                        percentage=(_bitmapLoader.y-
(stage.stageHeight/2-halfYLength))/(totalYLength);
                  }
                  else
                  {

      bar.x=_bitmapLoader.x+_bitmapLoader.width/2;
                        percentage=(_bitmapLoader.y-
(stage.stageHeight/2-halfYLength))/(totalYLength-Thumbnail.HEIGHT-
_bottomYDistance);
                  }

                  bar.y=stage.stageHeight/2-bar.height/2;

                  handler.x=bar.x;
                  handler.y=bar.y+heightDifference*percentage;
            }
```

/*Questa funzione rende 'invisibile' la barra di scorrimento variando i valori delle sue coordinate in maniera tale che risulti fuori dallo 'stage', cioè dall'area visibile all'utente.*/

```
private function deactivateScrollBar():void
{
	bar.x=handler.x=-100;
	bar.y=handler.y=0;
}
```

/*Questa funzione viene eseguita quando viene premuto il pulsante del 'mouse', prima del rilascio.*/

```
private function onDown(e:MouseEvent):void
{
```

/*Viene calcolata la differenza tra l'altezza della barra di scorrimento e l'altezza della maniglietta per capire di quanto si potrà muovere la maniglietta stessa.*/

```
		var heightDifference:Number=bar.height-
handler.height;
		var target:*=e.target;

		if(target is BitmapLoader)
		{
			if(stage.displayState=='fullScreen')
			{
				if(_isEnlarged)
				{
					_draggingX=_bitmapLoader.x;
					_draggingY=_bitmapLoader.y;

					startDragBitmapLoader();
				}
			}
		}
```

/*Se la pressione del pulsante del 'mouse' avviene sulla maniglia della barra di scorrimento, tale maniglia comincia a seguire le coordinate del 'mouse', rimanendo comunque 'costretta' all'interno del perimetro della barra.*/

```
		if(target is ScrollHandler)
		{
			_isDragging=true;
			handler.startDrag(false,new
Rectangle(bar.x,

	bar.y,

	0,

	heightDifference));
```

/*Parte il temporizzatore in grado di capire se è necessario aggiornare la posizione della maniglia della barra in caso in cui l'utente stia muovendo il 'mouse'.*/

```
addEventListener(Event.ENTER_FRAME,onScroll);
            }
      }
```

/*Questa funzione viene eseguita quando si rilascia il pulsante del 'mouse'.*/

```
      private function onUp(e:MouseEvent):void
      {
            var target:*=e.target;
            var id:uint;
            var pageNumber:uint;
```

/*Se il rilascio del 'mouse' avviene durante il trascinamento della maniglietta della barra di scorrimento, si interrompe tale trascinamento e si interrompe il conseguente aggiornamento della coordinata 'y' del documento selezionato.*/

```
            if(_isDragging)
            {
                  _isDragging=false;
                  handler.stopDrag();

removeEventListener(Event.ENTER_FRAME,onScroll);

                  return;
            }

            if(target is Thumbnail)
            {
```

/*Se il rilascio del 'mouse' avviene su una delle miniature interne al menu stile 'Mac' (le miniature sono rappresentate da istanze della classe Thumnail), recupero il valore della sua proprietà 'id' utile per la selezione di un nuovo documento.*/

```
                  id=target.id-1;
                  jumpeToImage(id);
            }
            else if(target is Circle)
            {
```

/*Se il rilascio del 'mouse' avviene su un oggetto 'Circle', si procede con la creazione di un nuovo menu stile 'Mac' contenente altre miniature corrispondenti ad altri documenti visualizzabili.*/

```
                    if(target.currentFrame==1)
                    {
                          _circleCounter=Circle(target).n;

goToAnotherMacStyleMenu(Circle(target));
                          deactivateCircles(Circle(target));


pageNumber=_circleCounter*_maxThumbnailsForMacStyleMenu+1;


macStyleMenu.setThumbnailSelected(pageNumber);
                    }
              }
```

/*Se il rilascio del 'mouse' avviene su un documento, viene valutata l'azione da compiere in base a come è visualizzata l'applicazione e in base al fatto se tale documento è ingrandito oppure no.*/

```
              else if(target is BitmapLoader)
              {
                    if(stage.displayState=='fullScreen')
                    {
```

/*Se il documento non è ingrandito, si procede con l'ingrandimento eseguendo la funzione 'enlargeBitmapLoader()'.*/

```
                          if(!_isEnlarged)
                          {
```

/*Dato che il documento sta per essere ingrandito, si disattiva l'eventuale barra di scorrimento che ha permesso di visualizzare le parti nascoste del documento stesso alle dimensioni 'normali'.*/

```
                                deactivateScrollBar();
                                enlargeBitmapLoader();
                          }
                          else
                          {
                                stopDragBitmapLoader();


      if(_bitmapLoader.x==_draggingX&&
_bitmapLoader.y==_draggingY)
```

```
                                    {
resetBitmapLoaderAfterEnlarging();
                                          manageScrollBar();
                                    }
                              }
                        }
                  }
            }
```

/*Questa funzione computa di quanto, il documento selezionato, deve traslare sull'asse verticale in base allo spostamento della maniglietta sulla barra.*/

```
            private function onScroll(e:Event):void
            {
```

/*Differenza tra l'altezza dello 'stage' e l'altezza del documento.*/

```
                  var totalYLength:Number=stage.stageHeight-
_bitmapLoader.height;
```

/*Si ottiene metà della differenza tra l'altezza dello 'stage' e del documento selezionato.*/

```
                  var halfYLength:Number=totalYLength/2;
```

/*Si ottiene la percentuale utile a capire di quanto muovere il documento in base alla traslazione subita dalla maniglietta sulla barra.*/

```
                  var percentage:Number=(handler.y-
bar.y)/(bar.height-handler.height);
```

/*Si fa traslare il documento sull'asse verticale in base alla percentuale ottenuta, tenendo anche in considerazione di come l'applicazione viene visualizzata al momento, o a tutto schermo o a dimensioni normali.*

```
                  if(stage.displayState=='normal')
                        _bitmapLoader.y=(stage.stageHeight/2-
halfYLength)+(totalYLength)*percentage;
                  else
                        _bitmapLoader.y=(stage.stageHeight/2-
halfYLength)+(totalYLength-Thumbnail.HEIGHT-
_bottomYDistance)*percentage;
            }
```

/*Questa funzione permette di ingrandire il documento selezionato tenendo conto sia della percentuale di ingrandimento, sia delle coordinate del 'mouse' al momento del 'click'. Selezionato un documento infatti, tramite un 'click' sulla sua

superficie, lo si ingrandisce in maniera tale che il punto su cui si è cliccato prima dell'ingrandimento risulti alle stesse coordinate anche dopo l'ingrandimento.*/

```
            private function enlargeBitmapLoader():void
            {
                  var
contentScale:Number=_bitmapLoader.content.scaleX;
                  var
a:Number=(_bitmapLoader.content.mouseX*contentScale)-
_bitmapLoader.content.width/2;
                  var
b:Number=(_bitmapLoader.content.mouseY*contentScale)-
_bitmapLoader.content.height/2;
                  var c:Number;
                  var d:Number;

                  _bitmapLoader.initScale=contentScale;

                  _isEnlarged=true;

                  _bitmapLoader.resizeBitmapByPercentage(1);
                  _bitmapLoader.centerBitmap();

                  c=a*(_bitmapLoader.content.scaleX/contentScale);
                  d=b*(_bitmapLoader.content.scaleX/contentScale);

                  _bitmapLoader.x-=c-a;
                  _bitmapLoader.y-=d-b;
            }
```

/*Una volta deselezionata l'opzione di ingrandimento, il documento corrente, viene ridimensionato e ricentrato all'interno del suo contenitore.*/

```
            private function
resetBitmapLoaderAfterEnlarging():void
            {
                  _isEnlarged=false;


      _bitmapLoader.resizeBitmapByPercentage(_bitmapLoader.initSca
le);
                  _bitmapLoader.centerBitmap();
                  alignBitmapLoader();
            }
```

/*Viene iniziata l'azione di trascinamento di un documento.*/

```
            private function startDragBitmapLoader():void
            {
                  var margin:Number=5;
```

/*Si determinano le misure dell'area entro la quale il documento può essere trascinato.*/

```
                var l:Number=-_bitmapLoader.width/2+margin;
                var t:Number=-_bitmapLoader.height/2+margin;
                var
w:Number=stage.stageWidth+_bitmapLoader.width-margin*2;
                var
h:Number=stage.stageHeight+_bitmapLoader.height-margin*2;

                _bitmapLoader.startDrag(false,new
Rectangle(l,t,w,h));
          }
```

/*Questa funzione interrompe l'azione di trascinamento di un documento.*/

```
          private function stopDragBitmapLoader():void
          {
                _bitmapLoader.stopDrag();
          }
```

/*Questa funzione predispone un documento per la stampa.*/

```
          private function onPrint(e:MouseEvent):void
          {
```

/*Se il pulsante di stampa viene premuto senza che nessun documento sia selezionato, la funzione si interrompe.*/

```
                if(_bitmapLoader==null)
                      return;
```

/*Se il pulsante di stampa viene premuto prima che il documento richiesto abbia completato il caricamento, la funzione si interrompe.*/

```
                if(_bitmapLoader.content==null)
                      return;
```

/*Si crea un'istanza della classe 'PrintJob' che in ActionScript 3.0 è in grado di gestire la fase di stampa di un documento.*/

```
                var my_pj:PrintJob=new PrintJob();
```

/*Fattore di ridimensionamento da applicare ad un documento prima della stampa.*/

```
                var scaleFactor:Number=.38;
```

/*Istanza della classe 'Sprite', che conterrà un duplicato dell'immagine-documento ottimizzato per la stampa.*/

```
var a:Sprite=new Sprite();
```

/*Si crea un nuovo 'renderizzatore' di matrici di pixel.*/

```
var b:Bitmap=new Bitmap();
```

/*Viene creata una copia della matrice di pixel che costituisce l'immagine-documento.*/

```
b.bitmapData=_bitmapLoader.content.bitmapData.clone();
```

/*La copia subisce il ridimensionamento specificato dalla variabile 'scaleFactor'.*/

```
b.scaleX=b.scaleY=scaleFactor;
```

/*In base alle dimensioni dell'immagine-documento, si decide se ruotare tale documento di 90 gradi per garantire che venga stampato correttamente sul foglio. */

```
if(b.width>b.height)
      b.rotation=90;
```

/*La 'bitmap' viene inserita nel contenitore 'a'. Una 'bitmap', per essere stampata correttamente, necessita di essere inserita all'interno di un oggetto di tipo 'Sprite'. La variabile 'a' contiene un puntatore ad un'istanza di questo tipo.*/

```
a.addChild(b);
```

/*Viene attivata la fase di stampa. Se il sistema non rileva errori, il documento viene stampato attraverso il metodo 'send()'. Altrimenti viene visualizzato un messaggio di errore.*/

```
if(my_pj.start())
{
      try
      {
            my_pj.addPage(a);
      }
      catch(e:Error)
      {
            trace('print problems.');
      }
```

```
                        my_pj.send();
                }
        }
}
```

5.2 Classe 'Preload'

/*Un'istanza di questa classe ha il compito di precaricare il programma, mostrando all'utente della grafica specifica. Inoltre un campo di testo informerà l'utente, in tempo reale, della percentuale di programma caricata fino ad un dato momento.*/

```
package
{
      import flash.display.*
      import flash.net.*;
      import flash.events.*;
      import flash.text.*
      import flash.system.*;
      import flash.utils.*;

      public class Preload extends MovieClip
      {
```

/*Il metodo costruttore, invocato all'avvio dell'applicativo, a sua volta, manda in esecuzione una serie di funzioni utili al caricamento della parte principale del programma.*/

```
            public function Preload()
            {
                  Security.allowDomain('*');

                  buildLoader();
                  activateListeners();
                  loadMedia('main.swf');
            }
```

/*Questa funzione costruisce un'istanza della classe 'Loader' avente il compito di caricare un 'file .swf' gestito dall'istanza della classe 'Main', cuore del programma.*/

```
            private function buildLoader():void
```

```
{
	_loader=new Loader();
}
```

/*L'istanza della classe 'Loader', da questo momento, è in grado di capire se il caricamento della parte principale del programma è andata a buon fine o, al contrario, se sono avvenuti errori che hanno impedito un corretto caricamento di 'main.swf'. Inoltre, ogni 'chunck' di 'byte' arrivato, manda in esecuzione la funzione 'onProgress()' grazie alla quale si può informare l'utente di quanto, del 'file' richiesto, è stato caricato fino a quel momento.*/

```
private function activateListeners():void
{
_loader.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
onProgress);

	_loader.contentLoaderInfo.addEventListener(Event.INIT,onInit
);

	_loader.contentLoaderInfo.addEventListener(Event.COMPLETE,on
Complete);
	_loader.contentLoaderInfo.addEventListener(IOErrorEvent.IO_E
RROR,onIoError);
	_loader.contentLoaderInfo.addEventListener(SecurityErrorEven
t.SECURITY_ERROR,onSecurityError);
}
```

/*Inizia il caricamento di 'main.swf'.*/

```
private function loadMedia(path:String):void
{
	_loader.load(new URLRequest(path));
}
```

/*Man mano 'chunk' di 'byte' di 'main.swf' vengono caricati nel 'Flash player', viene eseguita questa funzione in grado di calcolare la percentuale di caricamento tramite la divisione dei 'byte' totali di 'main.swf' per quelli parziali (quelli cariati fino ad un dato momento). Tale valore, opportunamente arrotondato, viene visualizzato all'interno di un campo di testo.*/

```
private function onProgress(e:ProgressEvent):void
{
	var loaded:Number;

	if(e.bytesTotal>0)
	{
		loaded=e.bytesLoaded/e.bytesTotal;
```

```
                    if(!isNaN(loaded))
                    {

preload_mc.txt.text=Math.floor(loaded*100)+' %';
                    }
                    else
                    {
                          preload_mc.txt.text='0 %';
                    }
              }
        }
```

/*Quando il primo fotogramma di 'main.swf' è stato caricato, viene chiamato il metodo 'initializeParameters()' sull'istanza di 'Main' associata a 'main.swf'. Vengono passati a tale metodo una serie di parametri, recuperati dal codice HTML, utili alla fase di inizializzazione della libreria digitale.*/

```
        private function onInit(e:Event):void
        {
```

/*Si interrompe l'ascolto dell'evento 'Event.INIT', in quanto, tale evento, è appena accaduto.*/

```
_loader.contentLoaderInfo.removeEventListener(Event.INIT,onInit);

Object(_loader.content).initializeParameters(root.loaderInfo.param
eters.xmlPath,

                root.loaderInfo.parameters.maxThumbnails,

                root.loaderInfo.parameters.page,

                root.loaderInfo.parameters.thumb,

                root.loaderInfo.parameters.p2);
        }
```

/*Nel caso sia eseguita questa funzione, significa che il caricamento del media esterno è andato a buon fine. Si deattivano, quindi, gli elementi logico-grafici che stavano gestendo tale fase di caricamento, ormai concluso, e il contenuto appena caricato viene reso renderizzabile dal 'Flash player'.*/

```
        private function onComplete(e:Event):void
        {
              deactivateListeners();
              removePreloader();

              addChild(_loader);
        }
```

/*Nel caso sia eseguita questa funzione, significa che, il media esterno che si è tentato di caricare, non è presente. Ragione per cui si deattivano gli elementi logico-grafici che stavano gestendo tale fase di caricamento.*/

```
private function onIoError(e:IOErrorEvent):void
{
	deactivateListeners();
	removePreloader();
}
```

/*Nel caso sia eseguita questa funzione, significa che si è verificato un problema legato alla sicurezza nel tentativo di caricare un media esterno. Ragione per cui si deattivano gli elementi logico-grafici che stavano gestendo tale fase di caricamento.*/

```
private function onSecurityError(e:IOErrorEvent):void
{
	deactivateListeners();
	removePreloader();
}
```

/*Questa funzione rimuove, dall'oggetto 'Loader', l'ascolto degli eventi legati all'attività di caricamento.*/

```
private function deactivateListeners():void
{
_loader.contentLoaderInfo.removeEventListener(ProgressEvent.PROGRE
SS,onProgress);

	_loader.contentLoaderInfo.removeEventListener(Event.INIT,onI
nit);

	_loader.contentLoaderInfo.removeEventListener(Event.COMPLETE
,onComplete);
	_loader.contentLoaderInfo.removeEventListener(IOErrorEvent.I
O_ERROR,onIoError);
_loader.contentLoaderInfo.removeEventListener(SecurityErrorEvent.S
ECURITY_ERROR,onSecurityError);
}
```

/*Questa funzione rimuove la grafica di segnalazione del precaricamento, rendendo l'oggetto 'preload_mc' rimuovibile dal 'garbage collector'.*/

```
private function removePreloader():void
{
	removeChild(preload_mc);
```

```
                preload_mc=null;
            }
        }
}
```

5.3 Classe 'MacStyleMenu', package 'lookahead.ui'

/*Questa classe genera istanze che rappresentano i menu in stile 'Mac'.*/

```
package
{
      import flash.display.Sprite;
      import flash.display.MovieClip;
      import flash.text.*;

      import flash.events.Event;

      import flash.net.URLLoader;
      import flash.net.URLRequest;

      import flash.utils.Dictionary;
      import flash.utils.getDefinitionByName;

      import lookahead.loaders.*;
      import lookahead.ui.*;

      public class MacStyleMenu extends Sprite
      {
            private var mouseChaseEase:Number;

            private var areaOfEffect:Number;
            private var maxScale:Number;

            private var icons:Array;
            private var iconContainer:Sprite;

            private var originalPositions:Dictionary;
            private var halfOfTotalWidth:Number;

            private var mouseChaseX:Number;
            private var mouseChaseY:Number;

            private var axis_x:String;
            private var axis_y:String;
            private var axis_scaleX:String;
            private var axis_scaleY:String;
            private var axis_width:String;
            private var axis_height:String;

            private var alignment:Number;
            private var _pointer:Pointer;
```

```
            public function MacStyleMenu($iconsXML:XML,
$maxScale:Number = 2, $areaOfEffect:Number = 200,
$mouseChaseEase:Number = 6, $alignment:String = "centerH"):void
            {
                  maxScale = $maxScale;
                  areaOfEffect = $areaOfEffect;
                  mouseChaseEase = $mouseChaseEase;

                  if($mouseChaseEase < 1) $mouseChaseEase = 1;

                  if($alignment == "top" || $alignment == "bottom"
|| $alignment == "centerH")
                  {
                        axis_x = "x";
                        axis_y = "y";
                        axis_scaleX = "scaleX";
                        axis_scaleY = "scaleY";
                        axis_width = "width";
                        axis_height = "height";

                        alignment = $alignment == "top" ? 0 :
$alignment == "bottom" ? -1 : -0.5;
                  }
                  else if($alignment == "left" || $alignment ==
"right" || $alignment == "centerV")
                  {
                        axis_x = "y";
                        axis_y = "x";
                        axis_scaleX = "scaleY";
                        axis_scaleY = "scaleX";
                        axis_width = "height";
                        axis_height = "width";

                        alignment = $alignment == "left" ? 0 :
$alignment == "right" ? -1 : -0.5;
                  }
                  else
                        throw new Error("Alignment must be
top/bottom/left/right/centerH/centerV.");

                  if($alignment == "top" || $alignment == "left")
                        alignment = 0;
                  else if($alignment == "bottom" || $alignment ==
"right")
                        alignment = -1;
                  else if($alignment == "center")
                        alignment = -0.5;

                  mouseChaseX = mouseX;
                  mouseChaseY = mouseY;

                  icons = new Array();
                  iconContainer = new Sprite();
                  _pointer=new Pointer();

                  originalPositions = new Dictionary();
```

```
                  var totalWidth:Number = 0;

                  for(var i:int = 0; i <
$iconsXML.immagine.length(); i++)
                  {
                        var icon:Thumbnail=new
Thumbnail($iconsXML.immagine[i]);

                        icon.mouseChildren=false;
                        icon.buttonMode=true;

                        addChild(icon);

                        totalWidth += icon[axis_width];
                        originalPositions[icon] = totalWidth -
icon[axis_width]/2;

                        icons.push(icon);
                        iconContainer.addChild(icon);
                  }

                  halfOfTotalWidth = totalWidth/2;

                  addEventListener(Event.ENTER_FRAME, mainLoop);

                  addChild(iconContainer);
                  addChild(_pointer);
            }

            private function mainLoop($event:Event):void
            {
                  mouseChaseX -= (mouseChaseX -
mouseX)/mouseChaseEase;
                  mouseChaseY -= (mouseChaseY -
mouseY)/mouseChaseEase;

                  var totalWidth:Number = 0;

                  for(var i:int = 0; i < icons.length; i++)
                  {
                        var icon:Thumbnail = icons[i];

                        var dX:Number = (axis_x == "x" ?
mouseChaseX : mouseChaseY) - originalPositions[icon] +
halfOfTotalWidth;
                        var dY:Number = (axis_y == "y" ?
mouseChaseY : mouseChaseX) * -1;

                        var distance:Number = Math.sqrt(dX*dX +
dY*dY);

                        var iconScale:Number = (areaOfEffect -
distance) / areaOfEffect * maxScale;

                        //trace('iconScale: '+iconScale);
```

```
                        if(iconScale>(maxScale-.3))
                              icon.n.visible=true;
                        else
                              icon.n.visible=false;

                        if(iconScale < 1) iconScale = 1;

                        icon.scaleX = iconScale;
                        icon.scaleY = iconScale;

                        icon[axis_x] = totalWidth;
                        icon[axis_y] = icon[axis_height] *
alignment;

                        totalWidth += icon[axis_width];

                        //trace('icon.isSelected:
'+icon.isSelected);

                        if(icon.isSelected)
                        {
                              _pointer.x=icon.x-
iconContainer[axis_width]/2+icon[axis_width]/2-_pointer.width/2+2;
                        }
                  }

                  iconContainer[axis_x]=-
iconContainer[axis_width]/2;
            }

            public function getIcon($iconNumber:int):Sprite
            {
                  return icons[$iconNumber];
            }

            public function deactivateListeners():void
            {
                  removeEventListener(Event.ENTER_FRAME,mainLoop);

                  mainLoop(new Event(Event.ENTER_FRAME));
            }

            public function setThumbnailSelected(n:uint):void
            {
                  var icon:Thumbnail;

                  for(var i:int=0;i<icons.length;i++)
                  {
                        icon=icons[i];

                        icon.isSelected=icon.xml.@id==n;
                  }
            }
      }
}
```

5.4 Classe 'XmlAvailableEvent', package lookahead.events.

```
package lookahead.events
{
	import flash.events.*;
```

/*Questa classe, di fatto, genera istanze che rappresentano eventi personalizzati, ovvero oggetti dotati di proprietà create dal programmatore in aggiunta a quelle 'standard' dategli dalla superclasse 'Event'.*/

```
	public class XmlAvailableEvent extends Event
	{
		/*coded by Luca Bonacorsi, lookahead=>.
		Adobe Certified Instructor.*/
```

/*Costante che specifica il nome dell'evento personalizzato sottoforma di stringa.*/

```
		public static const
XML_AVAILABLE:String='xmlAvailable';
```

/*Questa proprietà rappresenta un'informazione aggiuntiva oltre a quelle fornite dalla classe 'Event'.*/

```
		public var messageToPass:String;

		public function XmlAvailableEvent(type:String,

bubbles:Boolean=false,

cancelable:Boolean=false,

messageToPass:String='no messages.')
		{
```

/*Chiamata al costruttore della superclasse a cui vengono passati dei valori utili ad inizializzare proprietà 'standard' come il tipo di evento e così via.*/

```
			super(type,bubbles,cancelable);
```

/*Inizializzazione delle proprietà specifiche, in questo caso c'è ne una sola: 'messageToPass'.

```
			this.messageToPass=messageToPass;
		}
```

/*Tutte le volte che si crea un classe rappresentativa di un evento personalizzato, deve essere sovrascritto il metodo 'clone' della superclasse 'Event', in maniera tale che, se richiamato, generi un corretto oggetto ' XmlAvailableEvent', copia esatta dell'istanza che lo invoca.*/

```
            public override function clone():Event
            {
                  return new
XmlAvailableEvent(type,bubbles,cancelable,messageToPass);
            }
      }
}
```

5.5 Classe 'BitmapLoader', package 'lookahead.loaders'.

```
package lookahead.loaders
{
      import flash.display.*
      import flash.net.*;
      import flash.events.*;
      import flash.text.*
      import flash.system.*;

      import lookahead.ui.*;

      public class BitmapLoader extends Sprite
      {
            /*coded by Luca Bonacorsi, lookahead=>.
            Adobe Certified Instructor.*/
            public static const MAX_IMAGE_WIDTH:Number=600;
            public static const MAX_IMAGE_HEIGHT:Number=480;

            protected var _loader:Loader;
            protected var _preload:PreloadItem;
            protected var _content:Bitmap;
            protected var _initScale:Number;
            protected var _path:String;

            public function BitmapLoader(path:String)
            {
                  Security.allowDomain('*');

                  _path=path;


      addEventListener(Event.ADDED_TO_STAGE,onAddedToStage);
            }

            private function onAddedToStage(e:Event):void
            {
```

```
removeEventListener(Event.ADDED_TO_STAGE,onAddedToStage);

            buildLoader();
            activateListeners();
            buildPreloader();
            resizePreload();
            positionPreload();
            addChildren();
            loadSwf();
      }
```

/*Viene creato un oggetto 'Loader', capace di caricare documenti esterni.*/

```
      protected function buildLoader():void
      {
            _loader=new Loader();
      }
```

/*L'istanza della classe 'Loader', da questo momento, è in grado di capire se il caricamento del documento esterno (immagine 'bitmap' esterna) è andato a buon fine o, al contrario, se sono avvenuti errori che hanno impedito un corretto caricamento di tale documento. Inoltre, ogni 'chunck' di 'byte' arrivato, manda in esecuzione la funzione 'onProgress()' grazie alla quale si può informare l'utente di quanto, del 'file' richiesto, è stato caricato fino a quel momento.*/

```
            protected function activateListeners():void
            {
_loader.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS,
onProgress);

      _loader.contentLoaderInfo.addEventListener(Event.COMPLETE,on
Complete);
      _loader.contentLoaderInfo.addEventListener(IOErrorEvent.IO_E
RROR,onIoError);
      _loader.contentLoaderInfo.addEventListener(SecurityErrorEven
t.SECURITY_ERROR,onSecurityError);
            }
```

/*Questa funzione crea un'istanza della classe 'PreloadItem'. Tale classe è associata al simbolo grafico 'mc_preload'. In pratica, creando un oggetto di questo tipo, si ottiene una controparte grafica che viene visualizzata mentre il documento è in fase di caricamento. Tale grafica informa l'utente di attendere il caricamento dell'immagine esterna.*/

```
            protected function buildPreloader():void
            {
```

```
        _preload=new PreloadItem();
}
```

/*La grafica che informa l'utente che il documento è in fase di caricamento, viene centrata nello 'stage'. Ma solo se è effettivamente presente.*/

```
public function positionPreload():void
{
        if(_preload!=null)
        {
                _preload.x=stage.stageWidth/2;
                _preload.y=stage.stageHeight/2;
        }
}
```

/*La grafica che informa l'utente che il documento è in fase di caricamento, viene resa visibile all'utente, aggiungendola alla 'childlist' del contenitore rappresentato dall'istanza 'BitmapLoader'.*/

```
protected function addChildren():void
{
        addChild(_preload);
}
```

/*Questa funzione inizia il caricamento del documento esterno.*/

```
protected function loadSwf():void
{
        _loader.load(new URLRequest(_path));
}
```

/*Man mano 'chunk' di 'byte' di un documento esterno (immagine 'bitmap' esterna) vengono caricati nel 'Flash player', viene eseguita questa funzione in grado di calcolare la percentuale di caricamento tramite la divisione dei 'byte' totali del documento per quelli parziali (quelli cariati fino ad un dato momento). Tale valore, opportunamente arrotondato, viene visualizzato all'interno di un campo di testo contenuto nell'oggetto 'PreloadItem'.*/

```
protected function onProgress(e:ProgressEvent):void
{
        var loaded:Number;

        if(e.bytesTotal>0)
        {
                loaded=e.bytesLoaded/e.bytesTotal;
```

```
                    if(!isNaN(loaded))
                    {

_preload['txt'].text=Math.floor(loaded*100)+' %';
                    }
                    else
                    {
                          _preload['txt'].text='0 %';
                    }
              }
        }
```

/*Nel caso sia eseguita questa funzione, significa che il caricamento del media esterno è andato a buon fine. Si deattivano, quindi, gli elementi logico-grafici che stavano gestendo tale fase di caricamento, ormai concluso, e il contenuto appena caricato viene reso renderizzabile dal 'Flash player'.*/

```
        protected function onComplete(e:Event):void
        {
              _content=Bitmap(_loader.content);

              addChild(_content);

              deactivateListeners();
              disposePreloader();
              disposeLoader();

              dispatchEvent(e);
        }

        protected function onIoError(e:Event):void
        {
              //trace('onIoError().');

              deactivateListeners();
              disposePreloader();

              //_field.text='onIoError().';
        }
```

/*Nel caso sia eseguita questa funzione, significa che si è verificato un problema legato alla sicurezza nel tentativo di caricare un media esterno. Ragione per cui si deattivano gli elementi logico-grafici che stavano gestendo tale fase di caricamento.*/

```
        protected function
onSecurityError(e:SecurityErrorEvent):void
        {
              deactivateListeners();
              disposePreloader();
```

```
	}

	protected function closeConnection():void
	{
		if(_loader!=null)
		{
			try
			{
				trace('loader.close().');

				_loader.close();
			}
			catch(e:*)
			{
				trace('Loading not in progress.');
			}
		}
	}
```

/*Questa funzione rimuove, dall'oggetto 'Loader', l'ascolto degli eventi legati all'attività di caricamento.*/

```
	protected function deactivateListeners():void
	{
		if(_loader!=null)
		{
_loader.contentLoaderInfo.removeEventListener(ProgressEvent.PROGRE
SS,onProgress);

	_loader.contentLoaderInfo.removeEventListener(Event.COMPLETE
,onComplete);
	_loader.contentLoaderInfo.removeEventListener(IOErrorEvent.I
O_ERROR,onIoError);
_loader.contentLoaderInfo.removeEventListener(SecurityErrorEvent.S
ECURITY_ERROR,onSecurityError);
		}
	}
```

/*Questa funzione rimuove la grafica che ha informato l'utente che il documento esterno era in fase di caricamento.*/

```
	protected function disposePreloader():void
	{
		if(_preload!=null)
		{
			removeChild(_preload);

			_preload=null;
		}
	}
```

/*Questa funzione rimuove l'oggetto 'Loader' che ha gestito il caricamento del documento esterno.*/

```
protected function disposeLoader():void
{
	if(_loader!=null)
	{
		if(contains(_loader))
			removeChild(_loader);

		_loader=null;
	}
}
```

/*Metodo 'getter' per il recupero del valore della proprietà '_content' di una data istanza della classe 'BitmapLoader'. Tale contenuto è la 'bitmap' caricata dell'esterno (documento).*/

```
public function get content():Bitmap
{
	return(_content);
}

public function set initScale(n:Number):void
{
	_initScale=n;
}

public function get initScale():Number
{
	return(_initScale);
}
```

/*Questa funzione chiama in sequenza altre funzioni utili per gestire la chiusura della fase di caricamento di un'immagine esterna.*/

```
public function dispose():void
{
	closeConnection();
	deactivateListeners();
	disposePreloader();
	disposeLoader();
}
```

/*L'immagine caricata viene centrata rispetto al contenitore nel quale è inserita. Così facendo l'origine del sistema di coordinate rappresentato da tale contenitore risulta perfettamente collocato al centro dell'immagine.*/

```
public function centerBitmap():void
{
```

```
            if(_content!=null)
            {
                  _content.x=-_content.width/2;
                  _content.y=-_content.height/2;
            }
      }
```

/*La 'bitmap' caricata viene ridimensionata tenendo conto della differenza tra le misure di altezza e larghezza.*/

```
      public function resizeBitmap():void
      {
            if(_content!=null)
            {
                  if(_content.height>=_content.width)
                  {

_content.height=BitmapLoader.MAX_IMAGE_HEIGHT;
                        _content.scaleX=_content.scaleY;
                  }
                  else
                  {

_content.width=BitmapLoader.MAX_IMAGE_WIDTH;
                        _content.scaleY=_content.scaleX;
                  }
            }
      }
```

/*La 'bitmap' caricata viene ridimensionata in base all'attuale larghezza dello 'stage'.*/

```
      public function resizeByStageWidth():void
      {
            if(_content!=null)
            {
                  _content.width=stage.stageWidth;
                  _content.scaleY=_content.scaleX;
            }
      }
```

/*La 'bitmap' caricata viene ridimensionata proporzionalmente alla visualizzazione a schermo intero. Come parametri vengono passate le dimensioni dello 'stage' utili al calcolo delle nuove dimensioni del documento.*/

```
      public function
resizeBitmapForFullScreen(stageH:Number,stageW:Number):String
      {
            var h:Number=stageH/1.2;
            var w:Number=stageW/5;
```

```
                _content.width=stageW-w;
                _content.scaleY=_content.scaleX;

                return('width');
        }
```

/*La 'bitmap' caricata viene ridimensionata di una percentuale specificata dal parametro 'n'.*/

```
        public function
resizeBitmapByPercentage(n:Number):void
        {
                if(_content!=null)
                {
                        _content.scaleX=_content.scaleY=n;
                }
        }
    }
}
```

5.6 Classe 'Thumbnail', package 'lookahead.loaders'.

/*Ognuna delle istanze di questa classe rappresenta una miniatura specifica di un dato documento. Le miniature sono incapsulate, a loro volta, in istanze della classe generatrice di menu in stile 'Mac'.*/

```
package lookahead.loaders
{
    import flash.display.*
    import flash.events.*;
    import flash.net.*;
    import flash.system.*;
    import flash.text.*

    import lookahead.ui.*;

    public class Thumbnail extends BitmapLoader
    {
```

/*Misura della larghezza di ogni miniatura.*/

```
        public static const WIDTH:Number=68;
```

/*Misura dell'altezza di ogni miniatura.*/

```
        public static const HEIGHT:Number=50;
```

/*Distance, sull'asse orizzontale, tra una miniatura e l'altra.*/

```
        public static const X_DISTANCE:Number=10;
```

```
private var _xml:XML;
private var _id:uint;
private var _page:String;
private var _isSelected:Boolean;
```

/*Al costruttore viene passato il riferimento ad un oggetto 'Xml' contenente informazioni inerenti as una specifica miniatura, come ad esempio il percorso all'immagine bitmap esterna (la miniatura), il percorso al relativo ingrandimento (il documento vero e proprio), un 'id' univoco, il numero pagina a cui il documento associato appartiene (supposto si stia visualizzando una serie di documenti costitutivi un libro).*/

```
public function Thumbnail(o:XML)
{
      Security.allowDomain('*');

      super(o.thumbnail.toString());

      _xml=o;
      _id=uint(o.@id);
      _page=checkPage(o.@page);
```

/*Questa variabile terrà traccia di quale, tra 'n' miniature, è quella selezionata correntemente per azione del 'click' del 'mouse'.*/

```
      _isSelected=false;

      setN(Thumbnail.WIDTH);
}
```

/*Funzione 'getter' per recuperare il riferimento all'oggetto 'Xml' associato ad una specifica miniatura.*/

```
public function get xml():XML
{
      return(_xml);
}
```

/*Funzione 'getter' per recuperare l'identificatore univoco associato ad una specifica miniatura.*/

```
public function get id():uint
{
      return(_id);
}

public function checkPage(s:String):String
```

```
{
	var delimiter:String=',';
	var newPage:String=s;

	if(s.indexOf(delimiter)>-1)
		newPage=s.replace(delimiter,' - ');

	return(newPage)
}
```

/*Funzione 'setter' per impostare il valore della proprietà '_ isSelected' di una data miniatura, utilizzata dal programma per sancire se tale miniatura è quella correntemente selezionata (true) oppure no (false).*/

```
public function set isSelected(b:Boolean):void
{
	_isSelected=b;
}
```

/*Funzione 'getter' per recuperare il valore della proprietà '_ isSelected' per informare il programma se una data miniatura è quella correntemente selezionata.*/

```
public function get isSelected():Boolean
{
	return(_isSelected);
}
```

/*'n' è un oggetto 'TextField' associato alla miniatura. In tale campo di testo compare la pagina a cui è associato il documento corrispondente.*/

```
private function setN(w:Number):void
{
```

/*Il campo all'inizio non è visibile.*/

```
	n.visible=false;
```

/*Il testo, una volta inserito, compare centrato nel campo testuale.*/

```
	n.autoSize=TextFieldAutoSize.CENTER;
	n.text=String(_page);
```

/*Il campo di testo viene centrato orizzontalmente rispetto alla larghezza dell'immagine.*/

```
	n.x=w/2-n.width/2;
}
```

/*Si sovrascrive il metodo 'resizePreload()' della superclasse 'BitmapLoader', per poter gestire in maniera differenziata il ridimensionamento della grafica che segnala il precaricamento di un'immagine esterna. Nel caso specifico, una miniatura.*/

```
override protected function resizePreload():void
{
	_preload.scaleX=_preload.scaleY=.5;
}
```

/*Si sovrascrive il metodo 'positionPreload()' della superclasse 'BitmapLoader', per poter gestire in maniera differenziata il posizionamento della grafica che segnala il precaricamento di un'immagine esterna. Nel caso specifico, tale grafica, viene centrata rispetto alle dimensioni della miniatura in caricamento.*/

```
override public function positionPreload():void
{
	_preload.x=Thumbnail.WIDTH/2;
	_preload.y=(n.y+n.height+Thumbnail.HEIGHT/2);
}
```

/*Si sovrascrive il metodo 'onComplete()' della superclasse 'BitmapLoader', per poter gestire in maniera differenziata le azioni da eseguire qual'ora il caricamento della miniatura si completi con successo.*/

```
override protected function onComplete(e:Event):void
{
```

/*Un riferimento alL'immagine 'bitmap' appena caricata, viene memorizzato nella proprietà '_content' per facilitarne l'accesso.*/

```
	_content=Bitmap(_loader.content);
```

/*La 'bitmap' diventa visibile all'utente, in quanto, da ora, è renderizzabile dal 'Flash player'.*/

```
	addChild(_content);

	resizeShape();
	positionContent();
	deactivateListeners();
	disposePreloader();
	disposeLoader();
	setN(_content.width);
```

```
                dispatchEvent(e);
        }
```

/*Un immagine vettoriale, che fa da sfondo per la 'bitmap', viene ridimensionata in accordo con le dimensioni effettive dell'immagine.*/

```
        private function resizeShape():void
        {
                var shape:DisplayObject=getChildAt(0);
                var xDistance:Number=10;

                shape.width=_content.width+xDistance;
                shape.height=_content.height
        }
```

/*La 'bitmap' viene posizionata in maniera tale da garantire non solo che non risulti sovrapposta al campo di testo, ma inoltre la si desidera sotto tale campo, con il suo bordo superiore coincidente con il bordo inferiore del campo.*/

```
        private function positionContent():void
        {
                _content.y=n.y+n.height;
        }
    }
}
```

5.7 Classe 'XmlLoader', package 'lookahead.loaders'

/*Istanze di questa classe, passato un percorso/URL come parametro del costruttore, sono in grado di caricare, e gestire il caricamento, di dati testuali esterni all'applicativo.*/

```
package lookahead.loaders
{
    import flash.display.*;
    import flash.events.*;
    import flash.net.*;
    import lookahead.events.*

    public class XmlLoader extends EventDispatcher
    {
        private var _data:String;
        private var _urlLoader:URLLoader;
```

/*Costruttore della classe a cui viene passato il percorso ai dati 'XML' da caricare.*/

```
public function XmlLoader(path:String)
{
	initialize(path);
}

private function initialize(path:String):void
{
```

/*Si incapsula il percorso/URL alla risorsa di dati esterni, contenuta nel parametro 'path', in un'istanza della classe 'URLRequest' la quale avrebbe la possibilità, tra le altre cose, di spedire valori a tale risorsa.*/

```
		var urlRequest:URLRequest=new URLRequest(path);
```

/*Viene creata un'istanza della classe 'URLLoader', in grado di iniziare e gestire la fase di caricamento di dati testuali esterni.*/

```
		_urlLoader=new URLLoader();
```

/*L'istanza della classe 'URLLoader', da questo momento in avanti, è in grado di captare quando, i dati richiesti, sono disponibili all'interno dell'applicativo.*/

```
_urlLoader.addEventListener(Event.COMPLETE,onComplete);
```

/*Viene effettivamente iniziato il caricamento di dati esterni.*/

```
		_urlLoader.load(urlRequest);
}
```

/*Questa funzione viene eseguita quando la stringa 'XML' è stata correntemente caricata all'interno del programma e può essere utilizzata.*/

```
private function onComplete(e:Event):void
{
```

/*La stringa 'XML' caricata dall'esterno, è contenuta dalla proprietà 'data' dell'oggetto '_urlLoader'. Per rendere più leggibile il codice, copia di tale stringa viene memorizzata nella proprietà '_data'.*/

```
		_data=_urlLoader.data;
```

/*Viene ora emesso un evento che informa il resto del programma che i dati 'XML' richiesti sono stati caricati correttamente e sono pronti all'uso. Tale evento consiste in un'istanza della classe personalizzata 'XmlAvailableEvent'.*/

```
                  dispatchEvent(new
XmlAvailableEvent(XmlAvailableEvent.XML_AVAILABLE,false,false,'tra
cks\' names loaded.'));
            }
```

/*Metodo getter per recuperare, da altri punti del programma, il valore della variabile '_data', ovvero i dati 'XML' sotto forma di stringa.*/

```
            public function get data():String
            {
                  return(_data);
            }
      }
}
```

5.8 Classe Circle, package 'lookahead.ui'

/*Questa classe è associata al simbolo 'MovieClip' nominato 'mc_circle', situato all'interno della libreria di elementi grafici del file 'main.fla'.

Tutte le volte che il programma crea istanze della classe 'Circle', si creano altrettante copie del corrispettivo elemento grafico. In sostanza un'istanza della classe 'Circle' rappresenta la parte logica di un oggetto grafico, copia del simbolo 'mc_circle'. Un oggetto di questo tipo, rappresenta un elemento interattivo che, se premuto, permette di accedere ad un altro sottoinsieme di documenti visualizzabili, creando, al fine di permettere l'accesso a tali documenti, un nuovo menu in stile 'Mac' con le opportune miniature.*/

```
package lookahead.ui
{
      import flash.display.*;

      public class Circle extends MovieClip
      {
            private var _n:uint;
            private var _urlCounter:uint;
```

/*Indice numerico relativo alla prima miniatura del menu in stile 'Mac' generato nel caso si cliccasse su una data istanza della classe 'Circle'.*/

```
            private var _firstImage:uint;
```

/*Indice numerico relativo all'ultima miniatura del menu in stile 'Mac' generato nel caso si cliccasse su una data istanza della classe 'Circle'.*/

```
            private var _lastImage:uint;

            public function
Circle(n:uint,urlCounter:uint,firstImage:uint,lastImage:uint)
            {
                  _n=n;
                  _urlCounter=urlCounter;
                  _firstImage=firstImage;
                  _lastImage=lastImage;
            }
```

/*Metodo 'getter' per il recupero del valore della proprietà '_n' di una data istanza della classe 'Circle'.*/

```
            public function get n():uint
            {
                  return(_n);
            }
```

/*Metodo 'getter' per il recupero del valore della proprietà `'_urlCounter'` di una data istanza della classe 'Circle'.*/

```
            public function get urlCounter():uint
            {
                  return(_urlCounter);
            }
```

/*Metodo 'getter' per il recupero del valore della proprietà '_firstImage' di una data istanza della classe 'Circle'.*/

```
            public function get firstImage():uint
            {
                  return(_firstImage);
            }
```

/*Metodo 'getter' per il recupero del valore della proprietà '_lastImage' di una data istanza della classe 'Circle'.*/

```
            public function get lastImage():uint
            {
                  return(_lastImage);
            }
      }
}
```

5.9 Classe 'PreloadItem', package 'lookahead.ui'

/*Questa classe è associata al simbolo 'MovieClip' nominato 'mc_preload', situato all'interno della libreria di elementi grafici del file 'main.fla'. Tutte le volte che il programma crea istanze della classe 'PreloadItem', si creano altrettante 'PreloadItem' rappresenta la parte logica di un oggetto grafico, copia del simbolo 'mc_preload'. Un oggetto di questo tipo segnala agli utenti che un media esterno è in fase di caricamento. E' dotato, inoltre, di un campo di testo che mostra la percentuale, aggiornata in tempo reale, di tale caricamento.*/

```
package lookahead.ui
{
	import flash.display.*;
	import flash.text.*;

	public class PreloadItem extends MovieClip
	{
		public function PreloadItem()
		{
		}
	}
}
```

5.10 Classe 'Sections', package 'lookahead.ui'

/*Questa classe è associata al simbolo 'MovieClip' nominato 'mc_sections', situato all'interno della libreria di elementi grafici del file 'main.fla'.
Tutte le volte che il programma crea istanze della classe 'Sections', si creano altrettante copie del corrispettivo elemento grafico. In sostanza un'istanza della classe 'Sections' rappresenta la parte logica di un oggetto grafico, copia del simbolo 'mc_sections'.*/

```
package lookahead.ui
{
	import flash.display.*;

	public class Sections extends MovieClip
	{
		public function Sections()
		{
		}
	}
}
```

Il progetto appena descritto sarà a breve on-line e permetterà agli utenti di alcune biblioteche emiliane, di poter fruire di svariati servizi, quali la ricerca, consultazione e visualizzazione di documenti , anche particolarmente ricercati e antichi, accedendovi dal luogo preferito senza doversi recare fisicamente presso le biblioteche che costudiscono tali documenti.

Bibliografia

- *Extensible Markup Language (XML) 1.0, W3C Recommendation 10 February 1998,http://www.w3.org/TR/1998/REC-xml-19980210.*
- *XML-Data Specification, W3C Recommendation, http://www.w3.org/TR/1998/Rec/Note-xml-data.*
- *XML Namespace, W3C Working Draft, http://www.w3.org/TR/1998/WD-xml-names-19990114..*
- *Un sito interamente dedicato ad XML, http://www.xml.com.*
- *L'università di XML, http://www.xmlu.com.*
- *Il sito della Microsoft su XML, http://msdn.microsoft.com/xml/index.asp.*
- *Prime informazioni, approfondimenti, domande, http://xml.org, http://www.xmlinfo.com*
- *Il consorzio W3C, http://www.w3c.org.*
- *Un enorme elenco di link a risorse relative a XML, http://www.cs.caltech.edu/~adam/local/xml.html.*
- *Le FAQ (Frequented Ask Question) dei membri dell'XML W3C, http://www.ucc.ie.xml/.*
- *Interessante corso sull'XML sviluppato da tre studenti milanesi,http://digilander.iol.it/oldani/HomePino.*

- *Il newsgroup msnews.microsoft.com della Microsoft dedicato a XML, microsoft.public.xml.*

- *Sui vari motori di ricerca e software house, molti dei seguenti Link:*

  - ✓ *http://digilander.libero.it/shppotta/*
  - ✓ *http://www.fireteam.it/2009/09/introduzione-alla-tecnologia-xml/#more-696*
  - ✓ *http://www.w3c.it/papers/webday2003-interoperability.pdf*
  - ✓ *http://www.e-socrates.org/file.php/8/Informatica_di_base/*
  - ✓ *http://lingue.uniss.it/informatica/20082009/Umanistica/uma_3.pdf*
  - ✓ *http://www.slidefinder.net/X/XML_documenti_applicazioni_transazionali_Internet/5178949*
  - ✓ *http://xml.html.it/faq/leggi/1007/che-cos*
  - ✓ *http://www.gdesign.it/pages/howto/articoli/nasflash/nasflash.php*
  - ✓ *http://it.wikipedia.org/wiki/Adobe_Flash*
  - ✓ *http://agentgroup.ing.unimo.it/didattica/*
  - ✓ *http://www.mrwebmaster.it/xml/guide/attributi-xml_316.html*
  - ✓ *http://www.webmasterpoint.org/flashmx/01.asp*
  - ✓ *http://www.webstyling.it/guide/xml/attributi_xml.php*
  - ✓ *http://www.dsi.unive.it/~asorato/SlideXML/XML.pdf*
  - ✓ *http://elite.polito.it/files/courses/01KTF/Slides/DTD_v1.pdf*
  - ✓ *http://forum.html.it/forum/showthread.php?threadid=364093*
  - ✓ *http://www.giuseppesicari.it/articoli/xpath/*

- *Per le tecnologie Adobe Flash e Actionscript3.0:*
  - ✓ *http://help.adobe.com/en_US/AS3LCR/Flash_10.0/*
  - ✓ *http://livedocs.adobe.com/flash/9.0_it/main/flash_as3_programming.pdf*
  - ✓ *http://it.wikipedia.org/wiki/Programmazione_orientata_agli_oggetti*
  - ✓ *http://www.realizzazione-sito.info/linguaggio/actionscript_3?page=5*
  - ✓

- *Per la parte introduttiva al progetto di libreria digitale:*
  - ✓ *http://www.nmis.isti.cnr.it/*
  - ✓ *http://www.laterza.it/bibliotecheinrete/Cap09/Cap09_06.htm tratto da "Biblioteche in Rete : istruzioni per l'uso* " di Fabio Metitieri, Riccardo Ridi. Agg. 2006. Cap. 9.: Oltre i cataloghi: i testi
  - ✓ *http://dbdmg.polito.it/twiki/pub/Public/AMDM_Cinema/02-DigitalLibrary2.pdf*
  - ✓ *http://it.wikipedia.org/wiki/Amazon_Kindle*
  - ✓ *http://www.stpauls.it/coopera/1003cp/1003cp20.htm*