UNIVERSITÀ DI PISA

FACOLTÀ DI LETTERE E FILOSOFIA

FACOLTÀ DI SCIENZE MM. FF. NN.

CORSO DI LAUREA SPECIALISTICA IN INFORMATICA UMANISTICA

Dal web alla carta: realizzazione di un sistema di utilizzo dei contenuti presenti in archivi *on line* per la creazione automatizzata di documenti destinati alla stampa professionale.

Relatori	*Candidato*	*Controrelatore*
Prof. Mirko Tavoni	Andrea Grande	Prof.ssa Maria Simi
Theo van Boxel		

Anno Accademico
2009/2010

Indice

Introduzione

Il *database publishing* è un settore dell'editoria elettronica che si è sviluppato in risposta ad una precisa esigenza: rendere molto più rapido ed efficiente, rispetto ai metodi tradizionali, il processo di creazione di impaginati i cui contenuti presentano caratteristiche tali da poter essere archiviati in una base di dati.

L'idea fondamentale è quella di sfruttare la costante disponibilità di informazioni, già presenti in rete e strutturate secondo una gerarchia conosciuta, per automatizzare molte fasi del lavoro riducendo notevolmente non solo i tempi necessari per la produzione e l'aggiornamento di documenti cartacei, ma anche il rischio di commettere errori dovuti alla ripetitività che caratterizza molte delle azioni da compiere.

In tale contesto si inserisce il progetto descritto in questa tesi: l'elaborazione di un sistema che consente di creare in maniera automatizzata un impaginato (destinato alla stampa professionale) i cui contenuti sono importati da un archivio *on line*.

La relazione è suddivisa in due parti:

- nella prima si introduce il contesto di riferimento e si motiva la scelta di standard, tecnologie e software utilizzati nella realizzazione del progetto;

- nella seconda parte è descritto, nel dettaglio, un esempio di applicazione del sistema studiato in un contesto reale.

Nel primo capitolo, che si apre con un'introduzione al mondo del *database publishing*, sono prese in esame le tecnologie scelte per strutturare un sistema alternativo all'uso dei costosi strumenti proposti dal mercato; fra esse riveste un ruolo fondamentale XML (*eXtensible Murkup Language*), standard che, per sua

natura, si pone ad un livello intermedio fra la base di dati di riferimento e la fase di impaginazione vera e propria del documento cartaceo.

Nel secondo capitolo viene motivata la scelta di Adobe InDesign come software di riferimento per realizzare il progetto. Del programma di impaginazione certamente più diffuso in ambito professionale sono descritte una serie di caratteristiche che lo rendono il più adatto per lavori di *database publishing*.

Le ultime versioni del software prevedono un'integrazione avanzata dello standard XML associata a un ambiente di sviluppo (*ExtendScript ToolKit*) dedicato alla programmazione diretta da parte dell'utente per automatizzare il *workflow* di impaginazione e ridurre al minimo la quantità di operazioni da svolgere manualmente.

Il terzo capitolo è completamente dedicato all'illustrazione del lavoro svolto, passo dopo passo.

L'analisi delle singole fasi di realizzazione è preceduta da un'introduzione al contesto reale di riferimento per l'uso esemplificativo del sistema: la creazione automatizzata del catalogo cartaceo di una casa editrice locale di Pisa.

- In primo luogo è discussa la progettazione di una struttura per il file XML che conterrà i dati prelevati dal database *on line*;
- in secondo luogo è descritto lo studio di un'applicazione PHP che permetta di generare tale file XML e popolarlo con i contenuti estratti in automatico dall'archivio;
- successivamente sono esaminate le operazioni da compiere in InDesign per predisporre il *template* di partenza dell'impaginato all'inserimento e alla successiva formattazione dei contenuti importati dal file XML;
- in ultima istanza è analizzato, in ogni sua parte, il nucleo del lavoro, ciò che caratterizza il sistema: la creazione di un programma JavaScript che possa essere mandato in esecuzione all'interno del software di impaginazione per

eseguire in automatico tutte le operazioni di inserimento e formattazione di testi e immagini nel documento.

1

Database publishing: una soluzione accessibile

1.1 - Come nasce il *database publishing*

Impaginare documenti destinati alla stampa professionale è sempre stato un lavoro prettamente manuale: il compito da svolgere da parte del grafico copre il *workflow* di un progetto di questo tipo in tutte le sue fasi, dall'ideazione di una gabbia tipografica, cioè la struttura nella quale inserire i contenuti, fino alla predisposizione per la stampa del PDF finale passando attraverso l'insieme delle procedure di impaginazione e formattazione del documento.

Sebbene questo tipo di approccio non sia particolarmente oneroso nella realizzazione di prodotti brevi e composti da un numero molto limitato di pagine (come ad esempio pieghevoli, piccole brochure etc.), problemi rilevanti in termini di costi, tempi necessari per portare a termine il lavoro e rischio di errori emergono nei casi in cui il prodotto da realizzare sia caratterizzato da un numero più elevato di pagine all'interno delle quali i contenuti assumono una forma ed una struttura anche molto articolate.

I software per l'impaginazione professionale sono stati arricchiti nel tempo di nuove funzionalità, accessibili da interfaccia GUI[1], che rendono più agevole il lavoro sotto alcuni aspetti; un esempio è la possibilità di dotare il documento di *autoflow*[2] del testo.

Si tratta di opzioni divenute certamente indispensabili ma che, soprattutto nella realizzazione di documenti dalla natura grafica e dalla struttura più complesse,

[1] L'interfaccia utente grafica GUI (dal corrispondente termine inglese *graphical user interface*) è un paradigma di sviluppo che mira a consentire all'utente di interagire con il computer manipolando graficamente degli oggetti; è lo strato di un'applicazione software che si occupa del dialogo con l'utente del sistema utilizzando un ambiente grafico.

[2] Con questo termine si fa riferimento al meccanismo di scorrimento dinamico del testo in blocchi concatenati e attraverso le varie pagine del documento.

non alleggeriscono il *designer* di un lavoro che richiede estrema attenzione e precisione nell'eseguire operazioni spesso molto ripetitive e per questo facilmente soggette a errori.

Esistono, in particolare, delle tipologie di impaginato per le quali la questione relativa alla ripetitività delle azioni da compiere è portata all'estremo; documenti cartacei come cataloghi prodotti, *report*, manuali multilingua, listini, sono accomunati da una caratteristica precisa ed evidente: la natura ben strutturata a livello logico dei contenuti, che possono essere testuali e grafici. Non a caso i dati destinati a essere inseriti in questa tipologia di impaginati sono spesso presentabili all'interno di database, archivi *on line* o fogli di calcolo.

L'esigenza che è andata definendosi sempre di più nel corso del tempo è stata quella di creare flussi di lavoro che, a partire dal recupero delle informazioni di partenza, ne sfruttasse la natura rigidamente strutturata per automatizzare il processo di impaginazione e di inserimento dei contenuti all'interno del documento destinato alla stampa nei casi in cui si rendesse necessaria la creazione di una versione cartacea di presentazione dei contenuti stessi.

È a partire da questa esigenza che ha avuto origine negli ultimi anni un'area tuttora in evoluzione dell'editoria elettronica che prevede l'utilizzo di strumenti, tecnologie e software specifici alla quale si fa riferimento col termine *database publishing*.

1.2 - Tipologie di *database publishing*

Esistono più modelli di *database publishing*, ma, in generale, si può effettuare una macrodistinzione fra due tipologie principali.

Un primo modello è quello usato nelle soluzioni *web-to-print* che forniscono agli utenti di un sito la possibilità di scegliere *on line* una soluzione di impaginazione fra una serie di *template* predefiniti e personalizzarla inserendo contenuti tramite moduli HTML per poi valutare immediatamente il risultato finale.

In questo caso la sorgente iniziale di dati è l'input del fruitore del sito: i contenuti inseriti vengono archiviati all'interno di un database e tale archiviazione

permette all'utilizzatore che non completa il lavoro e si riconnette al sistema in un secondo momento di recuperare i dati precedentemente immessi (la form HTML viene automaticamente precompilata) e di riprendere l'*editing* laddove lo aveva lasciato.

È ben rappresentativa di questo approccio la soluzione ideata dai creatori del sito web della MOO, una società di stampa che consente di impaginare prodotti cartacei e in particolare biglietti da visita, cartoline, *minicard* e *greeting card*, caricando direttamente testi e immagini personali o scegliendo fra i contenuti già disponibili.

Ogni passo della realizzazione, dalla scelta del tipo di carta a quella del *layout*, dal caricamento dei contenuti alla conferma di pagamento, può essere effettuato tramite un'interfaccia web chiara e intuitiva.

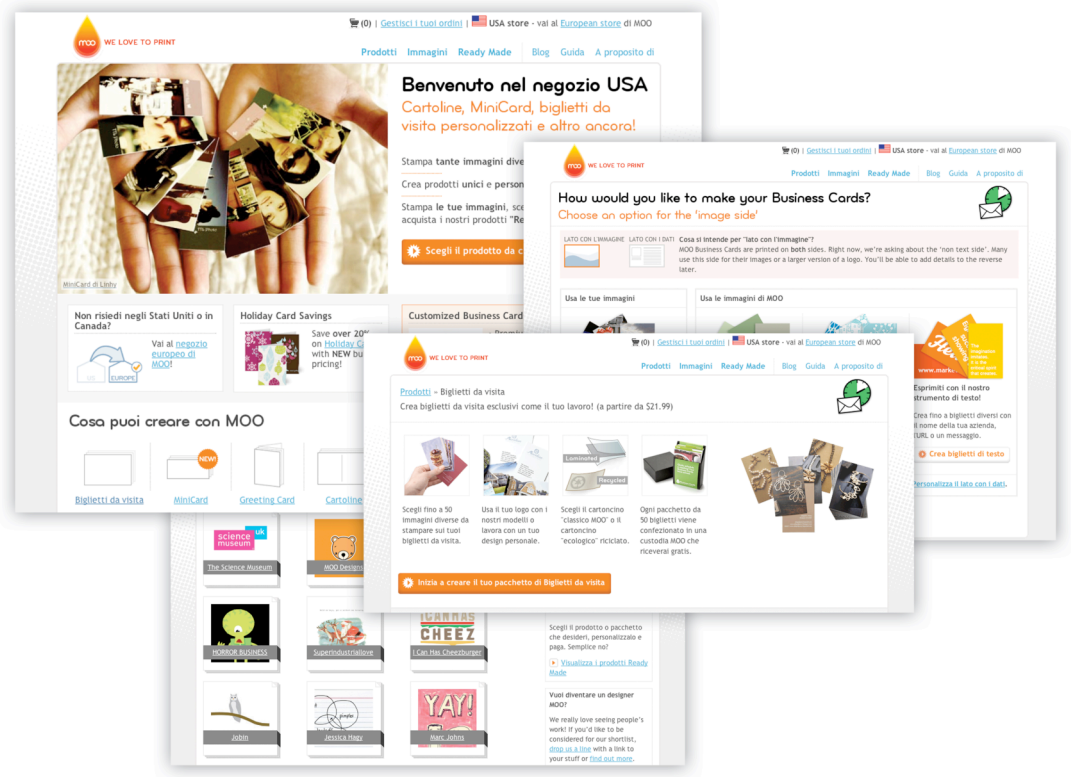

Figura 1 – Pagine del sito di MOO relative al sistema di creazione *on line* di impaginati

Una seconda tipologia di *database publishing*, quella entro la quale si colloca

il progetto di questa tesi, si fonda sull'idea di "riempire" con i contenuti archiviati in una base di dati (ad esempio le informazioni relative ai prodotti all'interno di un catalogo) un *template* preformattato e realizzato *ad hoc* da un grafico.

Questi *template* sono tipicamente creati tramite applicazioni desktop di impaginazione professionale e si presentano come documenti in cui sono definiti esclusivamente la struttura dell'impaginato e gli spazi entro i quali avverrà la collocazione dei diversi elementi grafici e testuali; le aree destinate a contenere testi ed eventuali immagini sono inizialmente riempiti da contenuti generici e fittizi, detti *placeholder*, che permettono a chi crea la struttura di valutare il risultato finale.

Solo una volta definite nel dettaglio tutte le impostazioni relative al *template*, si procede alla sostituzione dei *placeholder* con i contenuti reali che provengono direttamente dal database.

Un tale approccio permette una creazione rapida dell'impaginato e un rapido aggiornamento in caso di modifiche apportate ai dati, con un intervento manuale sul prodotto finale molto limitato o del tutto assente.

Altre varianti di *database publishing* utilizzano il *rendering* di contenuti per la creazione diretta di PDF, modalità che impedisce l'intervento manuale sull'output finale, poiché il documento PDF non è facilmente modificabile. Questa potrebbe non rappresentare una limitazione in casi come la generazione di un *report* per il quale la modificabilità manuale non è necessaria.

1.3 - I software specializzati

Esistono diverse applicazioni usate per la produzione di impaginati professionali di tutti i tipi, alcune fra le principali sono elencate di seguito:

- Adobe InDesign
- QuarkXpress
- Xyvision

Generalmente, per queste applicazioni è previsto anche l'utilizzo di una

versione server, che prevede l'invio di comandi via interfaccia web al posto dell'interazione desktop; QuarkXpress Server e Adobe InDesign Server, ad esempio, si avvantaggiano entrambe delle opzioni di design utilizzabili nelle rispettive versioni base. Sono stati, inoltre, implementati diversi *plug-in* per l'integrazione delle funzionalità di questi software con strumenti di impaginazione automatica di documenti strutturati come i cataloghi prodotti. Alcuni di essi funzionano in più di una applicazione; fra i principali ricordiamo:

- InCatalog
- EasyCatalog

Ci sono infine applicazioni specializzate (sulle quali è opportuno soffermarsi) proprio nella creazione di documenti strutturati a partire da archivi di dati:

- Adobe FrameMaker
- Corel Ventura
- Arbortext Advanced Print Publisher

1.3.1 - Adobe FrameMaker

In virtù delle sue caratteristiche fortemente orientate alla creazione di documenti ben strutturati, FrameMaker è un software utilizzato soprattutto per produrre la documentazione presso industrie in cui sono presenti molti modelli dello stesso prodotto, ad esempio quelle aerospaziali, o nelle industrie farmaceutiche per le quali la traduzione e la standardizzazione sono requisiti fondamentali ai fini della comunicazione sui prodotti.

Il modello di funzionamento è basato sull'uso degli standard SGML (*Standard Generalized Markup Language*) e XML (*eXtensible Markup Language*): l'autore lavora con una specifica DTD di FrameMaker, la EDD (*Element Definition Document*), che definisce la struttura di un documento in cui le unità principali sono progettate come elementi annidati l'uno nell'altro in base alle relazioni che li legano.

A questi elementi possono essere aggiunti attributi o metadati usati per filtrare i contenuti nei processi di output come la produzione di impaginati per la stampa o per la visualizzazione su web; la struttura è sempre sotto controllo di chi crea l'impaginato ed è in stretta relazione con il *layout* del documento.

Dalla sua introduzione sul mercato fino agli ultimi anni nei quali ha avuto una scarsa evoluzione, FrameMaker è stato al passo coi tempi nel supporto a XML per la creazione di *workflow* finalizzati al recupero e all'inserimento di contenuti all'interno di documenti da pubblicare in forma cartacea e su web.

Originariamente scritto per il sistema operativo della *Sun*, FrameMaker diventa molto presto uno strumento diffuso per la scrittura tecnica e visto il fiorire del mercato del *database publishing*, viene introdotta successivamente anche una versione per piattaforma Macintosh, seguita, negli anni novanta da ulteriori versioni destinate ad altre piattaforme basate su UNIX.

Effettuato il *porting* per Microsoft Windows, il software raggiunge definitivamente il mercato professionale per pubblicazioni altamente tecniche, e raggiunge anche costi elevati. Nonostante il successivo ribasso dei prezzi il tentativo di destinare questo software anche al mercato del *desktop publishing*[3] non professionale si rivela un fallimento; lo strumento risulta complesso per un utilizzo meno specialistico e questo ne impedisce la totale affermazione nel settore determinando un netto calo di vendite che porta l'azienda produttrice sull'orlo della bancarotta.

In questo contesto la Adobe acquisisce la Frame Technology Corporation e rilancia il prodotto riportando l'attenzione esclusivamente sul mercato professionale e facendo nuovamente di FrameMaker un'applicazione largamente usata in quel contesto nonostante l'interruzione della produzione per sistema operativo Mac OS.

L'aggiunta del supporto a SGML, e successivamente a XML contribuisce all'affermazione del software come strumento molto utilizzato per produrre documenti fortemente strutturati con relativa facilità, grazie anche all'approccio di

[3] Insieme delle procedure di creazione, impaginazione e produzione di materiale stampato dedicato alla produzione editoriale svolte attraverso un personal computer.

tipo WYSIWYG[4] che dà pieno controllo all'utente professionale e che permette la creazione di impaginati finali di ottima qualità.

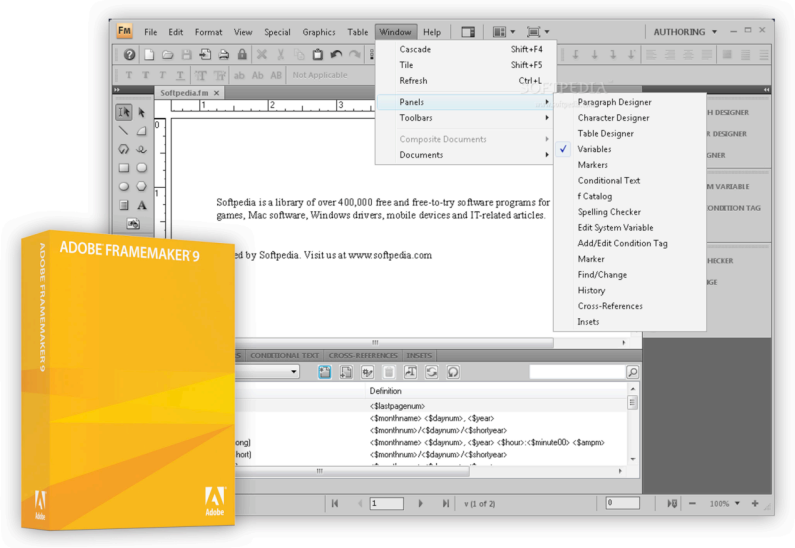

Figura 2 – Interfaccia del software per il *database publishing* Adobe FrameMaker

1.3.2 - Corel Ventura

Ventura Publisher, sviluppato dalla Ventura software, una piccola azienda fondata da John Meyer, Don Heiskel e Lee Jay Lorenzen, è stato il primo pacchetto di applicazioni di impaginazione per piattaforma IBM, la cui prima versione ufficiale risale al 1986.

Il software è originariamente progettato per interfacciarsi con un'ampia varietà di programmi per la grafica piuttosto che come loro sostituto.

I contenuti, infatti, non vengono manipolati solo internamente, bensì vengono importati ed esportati in file gestiti da altri programmi di *word processing* compreso Microsoft Word. Questo permette all'utente di continuare ad usare il programma preferito per apportare significative modifiche al proprio documento.

[4] Acronimo che sta per l'inglese *What You See Is What You Get* ("quello che vedi è quello che ottieni" o "ottieni quanto vedi"). Si riferisce al problema di ottenere sulla carta testo e/o immagini che abbiano una disposizione grafica uguale a quella visualizzata sullo schermo del computer.

I paragrafi sono marcati con *tag* descrittivi che sono completamente controllati dall'utente all'interno di Ventura e che possono comunque essere modificati al pari dei testi stessi lavorando sul file tramite l'utilizzo di altri software.

Concetti come il *paragraph tagging* e l'aggiunta di attributi e caratteri speciali anticipano l'uso di approcci basati su standard XML. Analogamente, la creazione di documenti risultanti dall'unione di file relativi ai singoli capitoli danno la possibilità di generare impaginati costituiti da centinaia o anche migliaia di pagine con la stessa facilità con cui si gestiscono documenti di poche pagine.

Un punto di forza di Ventura Publisher, almeno nella sua versione originale, è la portabilità su molte piattaforme, l'abilità di produrre documenti con un alto grado di strutturazione interna e la possibilità di esportare ed importare automaticamente testi da formati nativi di diversi *word processor*.

L'applicazione, acquisita da Corel nel 1993, viene reinserita sul mercato col nome Corel Ventura senza particolari modifiche nel codice sorgente.

La prima vera versione Corel del programma è la 5.0 rilasciata nel 1994, nella quale, invece, sono implementati cambiamenti sostanziali sia all'interfaccia sia alla struttura dei documenti. L'ultima *release*, attualmente in vendita, è la 10 che si pone come principale concorrente di Adobe FrameMaker.

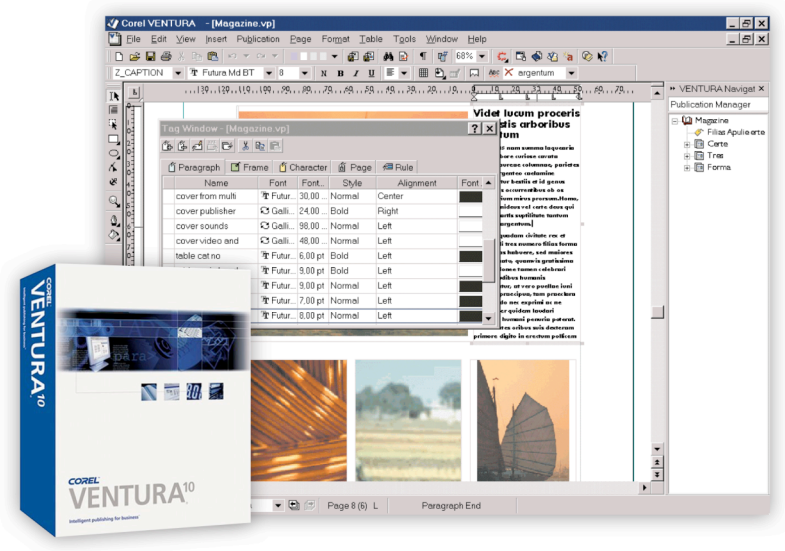

Figura 3 - Interfaccia del software per il *database publishing* Corel Ventura

1.3.3 - Arbortext Advanced Print Publisher

Una realtà più recente nel mondo del *database publishing* è rappresentata dal software Arbortext Advanced Print Publisher creato dalla PTC: un motore di composizione di stampa automatizzato che consente di pubblicare contenuto in più lingue con requisiti di *layout* diversificati e complessi.

È possibile produrre volumi elevati di pagine basate su regole definite in fogli di stile e realizzare vari tipi di impaginati: documentazione tecnica (manuali dell'utente, dell'operatore e di assistenza, cataloghi illustrati e materiali per il *training*), pubblicazioni di carattere scientifico, tecnico, medico, documenti amministrativi, finanziari e legali, elenchi e pubblicità.

Tramite i fogli di stile si possono associare proprietà di formattazione agli elementi in una gerarchia XML, creare *layout* di pagina e specificare riferimenti incrociati, indicizzazione e sommari. Grazie al supporto nativo per XML, per importare contenuto da altri sistemi di informazione non sono necessari passaggi di conversione file o pre-elaborazione.

L'architettura del software è aperta, si può integrare questa soluzione con una vasta gamma di database, applicazioni per la gestione del contenuto e prodotti dell'*information technology* utilizzati in molti flussi di lavoro.

Figura 4 – Logo del software Arbortext

1.4 - Il problema dei costi

Le applicazioni specializzate appena passate in rassegna sono strumenti indubbiamente validi per la creazione automatica di impaginati per la stampa a

partire da archivi di informazioni ma presentano alcune limitazioni non trascurabili, non ultima quella della mancata portabilità su più piattaforme.

In particolare, dal punto di vista di chi usufruisce di tali strumenti, non è sempre possibile sostenere i costi aggiuntivi necessari per affiancarli alle applicazioni di *desktop publishing* utilizzate per tutti i tipi di prodotti, che sono ovviamente irrinunciabili.

Si tratta di un problema da considerare sia in relazione all'acquisto di software che integrano le funzionalità di base delle applicazioni di impaginazione professionale più comunemente usate presso aziende e case editrici ad ogni livello, sia nel caso in cui si esternalizzi il lavoro affidandone la gestione a fornitori di servizi di *database publishing* personalizzati.

Se infatti il costo delle applicazioni di base utilizzate per qualunque tipo di impaginato è sostenibile anche per piccole realtà locali, ben diverso è il discorso per quanto riguarda la possibilità di fornirsi di estensioni, *plug in* e complesse applicazioni dedicate all'automatizzazione del lavoro che non raramente raggiungono costi di migliaia di euro.

1.5 - Un'alternativa: la scelta delle tecnologie

Il punto di partenza del progetto elaborato ai fini di questa tesi è lo studio di un metodo di lavoro che possa costituire una valida alternativa alle soluzioni più onerose dal punto di vista economico avvalendosi delle possibilità in chiave *database publishing* già fornite dal più diffuso fra i software di impaginazione, Adobe InDesign, di integrare le numerose funzionalità messe a disposizione dall'applicazione con soluzioni verticali che possono essere sviluppate anche internamente, attraverso linguaggi di programmazione (come JavaScript[5],

[5] Linguaggio di *scripting* orientato agli oggetti comunemente usato nei siti web. Fu originariamente sviluppato con il nome di Mocha e successivamente di LiveScript, ma in seguito è stato rinominato JavaScript ed è stato formalizzato con una sintassi più vicina a quella del linguaggio Java.

AppleScript[6] o VBScript[7]) e attraverso l'uso di uno strumento estremamente potente come XML che, non a caso, è alla base del funzionamento delle applicazioni specializzate descritte in precedenza.

1.5.1 - La via dell'XML

Se si pensa alla natura strutturata delle informazioni conservate in basi di dati non è difficile immaginare come le medesime informazioni possano essere inserite all'interno di un file XML modellato sulla base di una grammatica specifica che definisca i rapporti fra i diversi elementi che lo costituiscono.

L'esempio del catalogo di una casa editrice rende molto bene questa idea: la suddivisione per categorie di appartenenza dei diversi libri pubblicati può essere intuitivamente strutturata in un albero di questo tipo:

```
<catalogo>
      <categoria>
         <libro></libro>
         <libro></libro>
         <libro></libro>
         …
      </categoria>
      <categoria>
         <libro></libro>
         <libro></libro>
         <libro></libro>
         …
      </categoria>
      …
</catalogo>
```

[6] Sistema di *scripting* introdotto da Apple Computer e integrato nel sistema operativo Mac OS.

[7] Abbreviazione di Microsoft's Visual Basic Scripting Editino, è un sottoinsieme di Visual Basic utilizzato come linguaggio di *scripting general-purpose*.

Le informazioni relative a ciascun oggetto libro possono poi essere inserite in nodi specifici:

```
<libro>
    <collana></collana>
    <copertina></copertina>
    <legatura></legatura>
    <formato></formato>
    <pagine></pagine>
    <illustrazioni></illustrazioni>
    <genere></genere>
    <anno></anno>
    <isbn></isbn>
    <prezzo></prezzo>
    <autore></autore>
    <titolo></titolo>
    <descrizione></descrizione>
</libro>
```

XML, già largamente usato per lo scambio di dati è diventato una tecnologia sempre più importante anche nello svolgimento di lavori di impaginazione professionale, che prevedono una gestione di contenuti spesso di quantità considerevole; le motivazioni sono da individuare nelle nuove opportunità che questa tecnologia fornisce nell'ambito di interesse:

- una prima forma di automatizzazione del lavoro che consiste nell'importazione diretta dei contenuti strutturati all'interno del *layout* del documento;
- un riutilizzo di contenuti: XML può essere usato come formato intermedio per uno scambio di informazioni normalmente impossibile fra applicazioni diverse.

Se il software di impaginazione utilizzato integra la possibilità di gestire informazioni così strutturate, l'utilizzo di XML diventa il passaggio intermedio ideale nel processo di inserimento di contenuti in un impaginato cartaceo a partire dalla base di dati di origine.

Figura 5 – *Database publishing* basato su XML

Tramite la creazione di *query* SQL specifiche per l'estrazione dei dati dalle tabelle presenti in un database e implementate attraverso l'uso di linguaggi di programmazione lato server come PHP, è possibile generare file XML opportunamente strutturati e pensati per essere la fonte di contenuti alla quale attingere in fase di realizzazione del prodotto per la stampa.

I vantaggi rispetto ad un approccio tradizionale sono evidenti se si considera che qualunque aggiornamento effettuato sui dati di origine presenti nel database può essere rapidamente effettuato anche sui contenuti dell'impaginato grazie al passaggio intermedio in XML.

1.5.2 - Automatizzare il flusso di lavoro

In accordo a quanto appena detto, XML contribuirebbe in modo evidente a rendere il lavoro del grafico meno ripetitivo e molto più rapido: i contenuti, già pronti e strutturati, necessitano soltanto di essere importati adeguatamente all'interno di un *template* predisposto in precedenza. Tuttavia molte operazioni necessarie restano a carico di chi impagina il documento; i testi, le immagini, le tabelle e qualunque tipo di contenuto importato nella struttura delle pagine deve essere sistemato manualmente e ricontrollato più volte.

È necessario, dunque, implementare un programma, scritto in uno fra i linguaggi interpretabili dal software utilizzato, per automatizzare tutto il *workflow* a partire dall'importazione della struttura XML nel documento fino alla resa del prodotto pronto per l'esportazione in PDF e la stampa, passando attraverso l'inserimento dei contenuti e la formattazione degli stessi all'interno delle pagine.

Nel prossimo capitolo vengono esaminati nello specifico gli strumenti messi a disposizione dal software scelto per implementare il sistema di impaginazione.

Sono descritte le modalità per mezzo delle quali è possibile creare flussi di lavoro dedicati per l'automatizzazione del processo di impaginazione di documenti destinati alla stampa nel contesto di riferimento del *database publishing*.

2
Il software di riferimento

Quelle discusse nel primo capitolo sono le premesse che hanno indirizzato alla scelta di Adobe InDesign come software attraverso il quale realizzare la soluzione di impaginazione automatica oggetto di questa tesi.

Delle potenzialità che l'applicazione deve avere nell'ottica di sviluppare un sistema completo e versatile per la produzione di impaginati a partire da archivi *on line* si è già accennato analizzando, in linea teorica, l'impostazione da dare al lavoro perché il risultato sia soddisfacente.

In questo capitolo vengono passate in rassegna nel dettaglio le caratteristiche peculiari che permettono di tradurre le suddette potenzialità nella concreta creazione di un sistema di *database publishing*, facendo dell'uso di InDesign, che gode di larga diffusione come software di impaginazione professionale, uno strumento completo anche in questo ambito specifico.

2.1 - L'integrazione di XML

Già implementata in versioni precedenti, l'integrazione di XML nel flusso di lavoro è stata notevolmente incrementata e migliorata nelle ultime *release* del software.

Data la sempre crescente esigenza di sfruttare le grandi potenzialità di questa tecnologia nell'ambito di progetti di *database publishing*, il supporto all'utilizzo di XML è venuto a costituire una delle prerogative più importanti per gli sviluppatori che hanno dotato l'interfaccia dell'applicazione di una serie di elementi i quali, integrandosi con la tradizionale *layout view* del documento sul quale si lavora, rendono sempre più efficiente e fruibile tale supporto anche a *designer* "puri", con conoscenza limitata del linguaggio di *markup*.

2.1.1 – Gli elementi dell'interfaccia

I principali strumenti pensati per rendere pratico e funzionale il controllo ed il passaggio di contenuti da una struttura XML al *layout* del documento destinato alla stampa sono descritti di seguito.

Pannello Struttura

Mostra l'albero XML di riferimento e ne permette la modifica ricoprendo le funzionalità di base di un *editor* specifico per il linguaggio; menu e bottoni consentono di aggiungere, eliminare, modificare, manipolare la visualizzazione di elementi e attributi e anche di validare la struttura sulla base di una grammatica specifica.

L'inserimento dei contenuti nel *layout* della pagina avviene tramite semplice operazione di *drag and drop* degli elementi nei box predisposti.

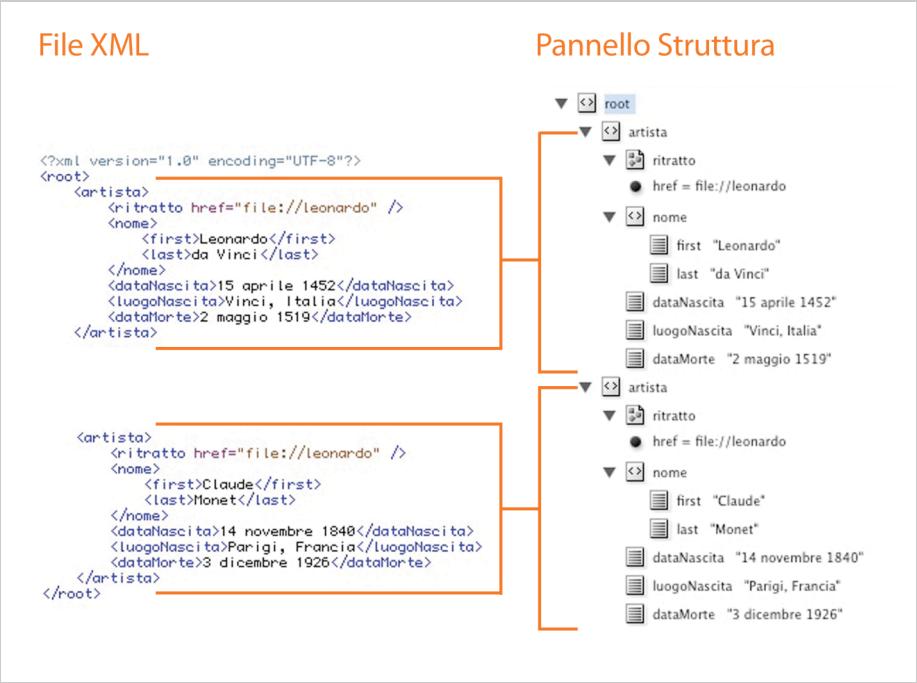

Figura 6 - Rappresentazione della struttura XML nel Pannello Struttura di InDesign

Pannello dei Tag

Contiene una lista di tutti gli elementi XML e permette di selezionarli singolarmente ed associarli a specifici contenuti nella pagina.

Può essere usato per creare, modificare o cancellare *tag*, per identificare gli elementi marcati e già inseriti nel documento, per assegnare le etichette a contenuti o *placeholder*.

In questo pannello sono sempre elencati i nomi di tutti i *tag* XML disponibili e a ciascuno è associato un colore diverso per rendere più immediata l'individuazione dei contenuti nel documento.

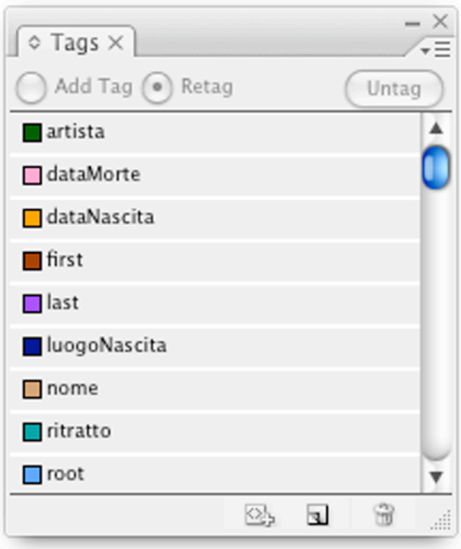

Figura 7 - Il pannello dei *tag*

Story editor

Una visualizzazione in stile *word-processor* di un singolo *frame* di testo o più *frame* di testo collegati.

Ogni modifica strutturale o grafica apportata ai contenuti tramite questo strumento si riflette sull'impaginato reale; risulta estremamente utile quando si lavora con XML perché mostra i contenuti all'interno dei *tag* ai quali sono assegnati.

Il termine *story* fa riferimento all'espressione *text stories* a sua volta usata per definire quei contenuti che, data la loro lunghezza, riempiono automaticamente un certo numero di *frame* di testo collegati fra loro all'interno dell'impaginato.

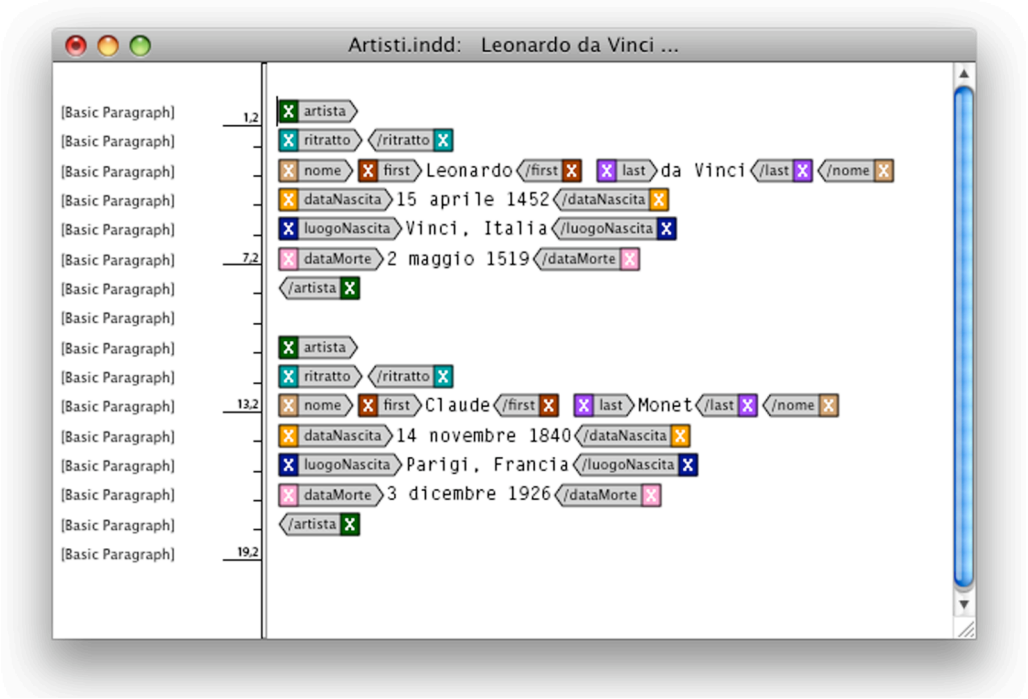

Figura 8 – Lo *Story editor*

Un documento XML deve necessariamente avere un singolo *tag* radice che contenga tutti gli altri elementi: in InDesign questo elemento è sempre mostrato nel Pannello Struttura.

Ciascun elemento XML può essere associato ad un oggetto all'interno del documento: ad esempio ad una *text story*, un singolo paragrafo, un elemento grafico. In accordo al meccanismo delle *stories*, quando un elemento XML è associato a un *frame* di testo, il programma applica il medesimo *tag* a tutti i box che sono collegati ad esso.

2.1.2 - Le modalità di importazione

In InDesign è prevista una serie di opzioni utili per controllare in che modo i contenuti devono essere gestiti dal programma in fase di importazione di file XML.

Il settaggio di queste preferenze si ripercuote anche sul modo in cui viene aggiornato un albero XML già importato nel programma:

Opzione	Descrizione
Merge content	Sostituisce tutti i contenuti XML presenti nel *layout.*
Append content	Inserisce (aggiunge) contenuti XML importati nell'attuale documento.
Create link	Crea un collegamento attivo al file XML presente sul computer. In caso di modifica dell'XML il pannello link indica che il file non è aggiornato.
Apply XSLT	Usa un foglio di stile XSLT[8] per manipolare l'XML durante l'importazione. L'XSLT è usato per ordinare, filtrare o cambiare completamente la struttura del file.
Clone repeating text elements	Replica automaticamente testi e oggetti taggati o non taggati. Opzione da usare nel caso di *layout* ripetitivi, come ad esempio biglietti da visita e cataloghi.
Only import elements that match existing structure	Filtra gli elementi dell'XML in rapporto alla struttura del documento, impedendo l'importazione di elementi che non sono già presenti nella struttura.
Import text element into tables if tags match	Filtra gli elementi in base alla struttura di una tabella inserendo i contenuti

[8] L'XSLT (*eXtensible Stylesheet Language Transformations*) è il linguaggio di trasformazione dell'XML; deriva direttamente dal linguaggio XSL, infatti i file di questo formato sono essenzialmente file di testo, contengono elementi ed attributi ed hanno l'estensione ".xsl".

	nelle celle appositamente marcate.
Do not import contents of whitespace-only elements	Impedisce l'importazione di spazi bianchi presenti nel file XML fra i diversi elementi in modo da utilizzare le impostazioni di spaziatura e allineamento definite direttamente nel *layout*.
Delete elements, frames and content that do not match imported XML	Cancella gli oggetti marcati nella pagina nel caso in cui l'XML importato non contenga elementi corrispondenti. Questa opzione previene l'inserimento di *frame* vuoti nel *layout* finale.

Tabella 1 - Opzioni per l'importazione e l'aggiornamento di strutture XML

2.1.3 - Formattazione dei contenuti XML

La natura dell'XML, che è semplicemente quella di dotare i contenuti di una struttura ben definita, non prevede alcuna specifica per la formattazione dei testi e per la loro collocazione nelle pagine; è quindi necessario definire *come* questi contenuti debbano fluire nell'impaginato e associare ad essi l'aspetto desiderato.

Formattazione manuale

Una volta importato l'XML e inseriti i contenuti testuali nel documento tramite operazione di *drag and drop* dal pannello Struttura, i testi possono essere formattati manualmente esattamente come si farebbe con contenuti inseriti in maniera diretta, senza passaggio attraverso XML.

La scelta di una formattazione manuale ha senso nei casi in cui i contenuti recuperati da XML rappresentino una piccola parte del totale.

Formattazione automatica

L'alternativa alla formattazione manuale è l'utilizzo della *mappatura sui tag* di stili paragrafo e stili carattere precedentemente creati in maniera coerente rispetto all'impostazione grafica scelta per il prodotto finale.

Si tratta di un approccio molto indicato nei casi di principale interesse per questa tesi, nei quali i contenuti dell'impaginato sono estratti nella loro totalità da una struttura definita e popolata a priori a partire da una base di dati.

Esistono tre modalità differenti di mappatura dei *tag*:

- per nome, nel caso in cui i nomi dei *tag* XML coincidano coi nomi degli stili carattere o paragrafo da assegnare;

- tramite caricamento delle impostazioni di mappatura da altri documenti InDesign creati in precedenza;

- attraverso un'associazione fatta singolarmente per i diversi *tag* tramite la definizione di corrispondenze con la lista predefinita di stili.

2.1.4 - Uso di oggetti ancorati

Un fenomeno frequente negli impaginati creati a partire da strutture XML è il regolare ripetersi di elementi come titoli e immagini; la posizione dei contenuti degli elementi XML varia, tuttavia, da pagina a pagina in relazione alla lunghezza di tutti gli altri testi presenti: ad esempio in ciascuna pagina di un catalogo di libri l'effettiva posizione in cui verrà collocata la breve descrizione di un testo pubblicato cambia in base allo spazio occupato dall'elemento che, all'interno della pagina, la precede.

L'uso degli oggetti ancorati in InDesign serve proprio a rendere flessibile la creazione di elementi ripetuti: le immagini e i box di testo interessati vengono appunto "ancorati" ad uno specifico elemento della pagina.

2.2 - Un ulteriore passo in avanti: la programmazione

Sulla base di questa rapida panoramica sulle funzionalità specifiche e sulle modalità essenziali di integrazione di XML nel software d'impaginazione di

riferimento si deduce che l'uso di tali strumenti comporta già un notevole passo in avanti rispetto alla ripetitività di molte operazioni tradizionalmente effettuate a mano dal grafico pagina per pagina.

Si nota anche come tale gestione del flusso di lavoro comporti una prima forma di automatizzazione dei *task* da compiere; una volta terminata l'importazione di un file XML è infatti possibile inserire tutti i tipi di contenuto nel documento e formattarli in maniera abbastanza rapida avvalendosi dei mezzi appena descritti.

L'obiettivo finale di questa tesi va però ancora oltre e consiste nel creare un sistema che riduca ulteriormente il numero di operazioni da compiere manualmente, combinando i vantaggi, già considerevoli, apportati dall'integrazione di XML con un'altra importante possibilità prevista in InDesign: quella di usufruire di un ambiente di sviluppo dedicato per la scrittura di programmi *ad hoc* eseguibili da interfaccia grafica tramite l'apposito pannello degli *script*.

2.2.1 - Gli strumenti a disposizione

La possibilità di creare *script* per l'automatizzazione di operazioni è probabilmente il più potente strumento del software InDesign; la funzionalità che più di ogni altra permette di risparmiare tempo e ridurre fortemente il rischio di commettere errori.

Qualsiasi operazione compiuta tramite interfaccia grafica può essere svolta attraverso uno *script*: creare *frame* in cui inserire testo, immagini, e anche stampare o esportare le pagine di un documento; in definitiva ogni azione che cambi la forma di un documento o i suoi contenuti può essere realizzata mandando in esecuzione programmi scritti con fini specifici. Alcune procedure sono addirittura effettuabili esclusivamente tramite *script*.

Esistono, allo stesso tempo, anche operazioni non effettuabili da codice: non è ad esempio possibile aggiungere nuovi tipi di oggetti ad un documento oppure funzionalità di base del software come un nuovo motore di composizione del testo.

Nel nostro contesto di riferimento l'aspetto veramente decisivo per la scelta dello *scripting* è la possibilità da esso fornita di poter automatizzare tutta una serie di

procedure necessarie per la creazione di impaginati generati a partire da informazioni strutturate in file XML ma lunghe e complicate da svolgere manualmente.

Anche per quanto riguarda l'uso di *script* creati *ad hoc* per gestire determinate operazioni, il software fornisce un elemento dell'interfaccia attraverso il quale è possibile scegliere quale operazione avviare in automatico: si tratta di un altro pannello al quale si è accennato in precedenza, quello degli *script* appunto, che rappresenta il modo più semplice per mandare in esecuzione programmi predefiniti già presenti così come quelli specifici scritti dall'utilizzatore di InDesign.

Tutti gli *script*, sia quelli predefiniti che quelli aggiunti dall'utente sono raggruppati in cartelle che prendono il nome dal linguaggio utilizzato per la scrittura del codice (JavaScript, AppleScript, VBScript).

L'albero visualizzato corrisponde ad una gerarchia presente nel *file system* della macchina sulla quale si lavora ed è quindi modificabile anche in maniera diretta accedendo ad una specifica cartella senza bisogno di lanciare l'applicazione.

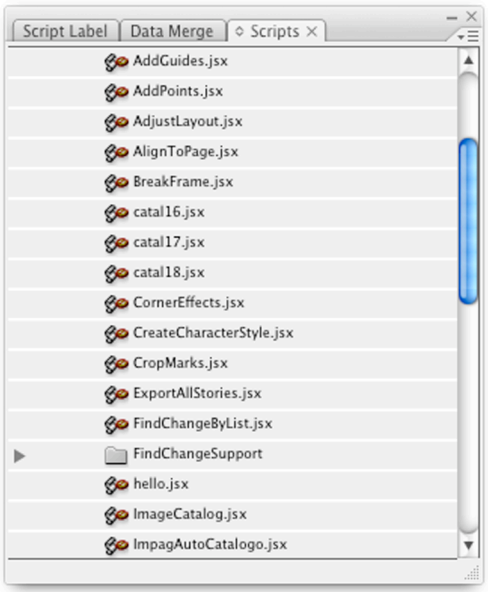

Figura 9 - Il pannello degli *script*

Il software, dunque, supporta più linguaggi di programmazione per la stesura del codice necessario all'automatizzazione del lavoro; la scelta del linguaggio dipende dalla piattaforma che si intende utilizzare: AppleScript sarà scelto per

usufruire del programma esclusivamente su Mac Os, VBScript in Windows, JavaScript, come vedremo la soluzione adottata per il sistema realizzato in questa tesi, permette invece un utilizzo *cross-platform* del sistema.

Le due componenti essenziali per la scrittura e l'utilizzo di *script* all'interno di un documento InDesign sono:

- l'ambiente di sviluppo *ExtendScript ToolKit*
- *l'InDesign Object Model*.

2.2.2 - L'ExtendScript ToolKit (ESTK)

Il supporto a JavaScript in InDesign è basato su un'implementazione del linguaggio realizzata dalla stessa Adobe ed utilizzata anche in altre applicazioni del pacchetto *Creative suite* che prende il nome di *ExtendScript*. Quest'ultimo supporta tutte le funzionalità di JavaScript ed è sostanzialmente una forma estesa del linguaggio.

Gli *script* sono composti di solo codice e perciò possono essere creati in un qualunque *editor* che permetta di salvare il file con l'estensione appropriata (ad esempio ".jsx"). Tuttavia l'ambiente più indicato per la scrittura del codice è l'*ExtendScript ToolKit*, appositamente creato per questo scopo, che fornisce una serie di strumenti atti a rendere più efficiente[9] lo sviluppo di programmi destinati a essere lanciati in InDesign.

L'interfaccia dell'ESTK presenta diverse aree, le più importanti sono:

- La finestra principale per la scrittura del codice.
- La *JavaScript Console* nella quale è possibile scrivere comandi e mandarli rapidamente in esecuzione per controllare il risultato ottenuto in maniera immediata (nello spazio sottostante).

[9] Uno strumento fondamentale per la programmazione è il meccanismo di *debugging*

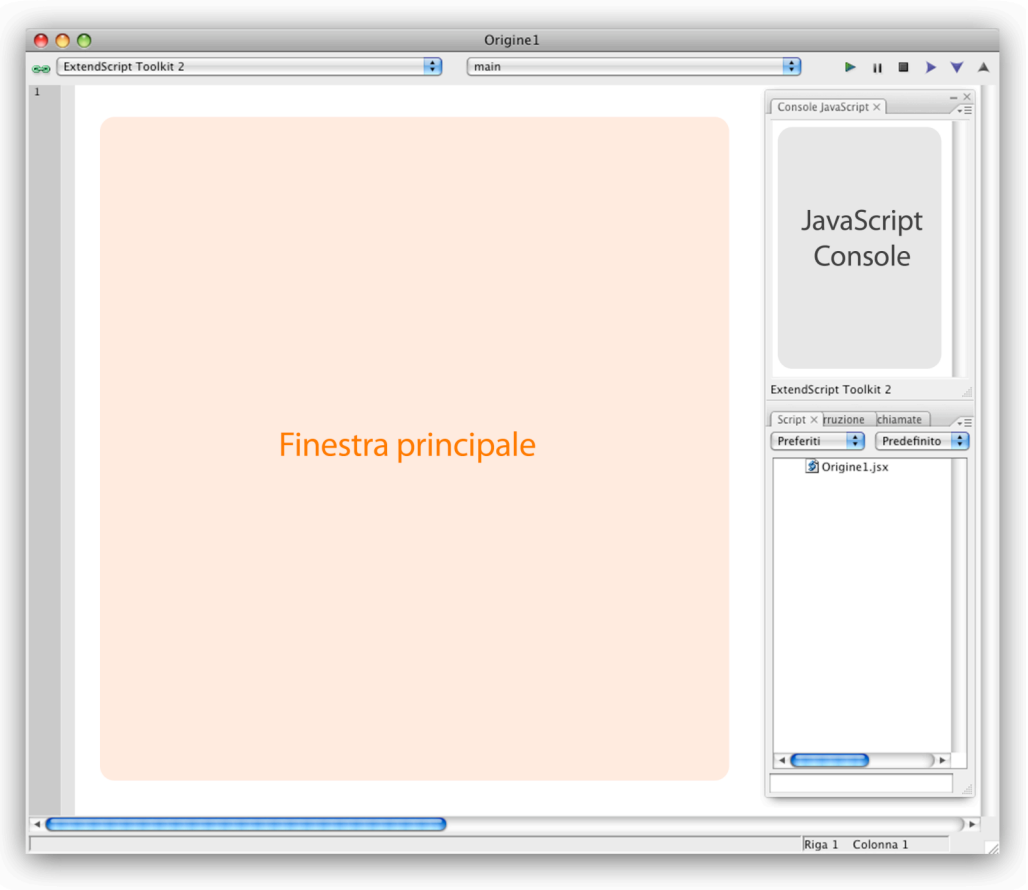

Figura 10 - L'interfaccia dell'*Extend Script ToolKit*

2.2.3 - Il modello a oggetti di InDesign

I documenti InDesign hanno sempre una struttura definita: i paragrafi sono contenuti in *frame* di testo i quali, a loro volta, sono contenuti in una pagina; una pagina è parte di un foglio e più fogli vanno a costituire il documento nella sua interezza che comprende anche colori, stili e livelli.

Esiste dunque un ordine che viene in realtà dato ai documenti nel momento stesso in cui li si progetta e li si crea.

L'applicazione "pensa ai contenuti" in modo analogo: un documento è composto da pagine in cui si collocano oggetti (*frame* di testo, forme, immagini, e così via); i box di testo contengono caratteri, parole, paragrafi. Tutti questi elementi sono gli oggetti che costituiscono una pubblicazione in InDesign e con i quali si

lavora nel progettare algoritmi e nel tradurli in uno *script*.

È proprio di una gerarchia di oggetti che si parla quando ci si riferisce all'*Object Model* di InDesign.

In accordo con i principi di base della programmazione *object oriented*[10], in questo modello ogni oggetto ha specifiche proprietà, ad esempio le proprietà di un oggetto testo includono il *font* utilizzato per formattarlo, la misura e l'interlinea applicata al paragrafo. Tali proprietà possono avere diversi valori (la grandezza di un testo può essere definita da un numero che corrisponde alla misura in punti) e possono essere di tipo *read/write* oppure *read only*.

Sugli oggetti si possono eseguire specifiche azioni, cioè applicare ad essi dei metodi; l'oggetto *document*, per esempio, prevede metodi per la stampa, l'esportazione e il salvataggio. In relazione all'azione che descrivono, i metodi possono poi richiedere o meno l'utilizzo di parametri: ad esempio il metodo *place* di un oggetto *document* prende un parametro che definisce il file da importare nel documento stesso.

L'immagine di seguito rappresenta uno schema riassuntivo della gerarchia del modello a oggetti di InDesign. È importante sottolineare che il diagramma non comprende tutti gli oggetti utilizzabili nello *scripting* del software, ma è piuttosto uno schema concettuale utile a comprendere le relazioni fra i diversi tipi di oggetto:

[10] La programmazione orientata agli oggetti (OOP, *Object Oriented Programming*) è un paradigma di programmazione, che prevede di raggruppare in un'unica entità (la classe) sia le strutture dati che le procedure che operano su di esse, creando per l'appunto un "oggetto" software dotato di proprietà (dati) e metodi (procedure) che operano sui dati dell'oggetto stesso. La programmazione orientata agli oggetti può essere vista come una modulazione di oggetti software sulla base degli oggetti del mondo reale.

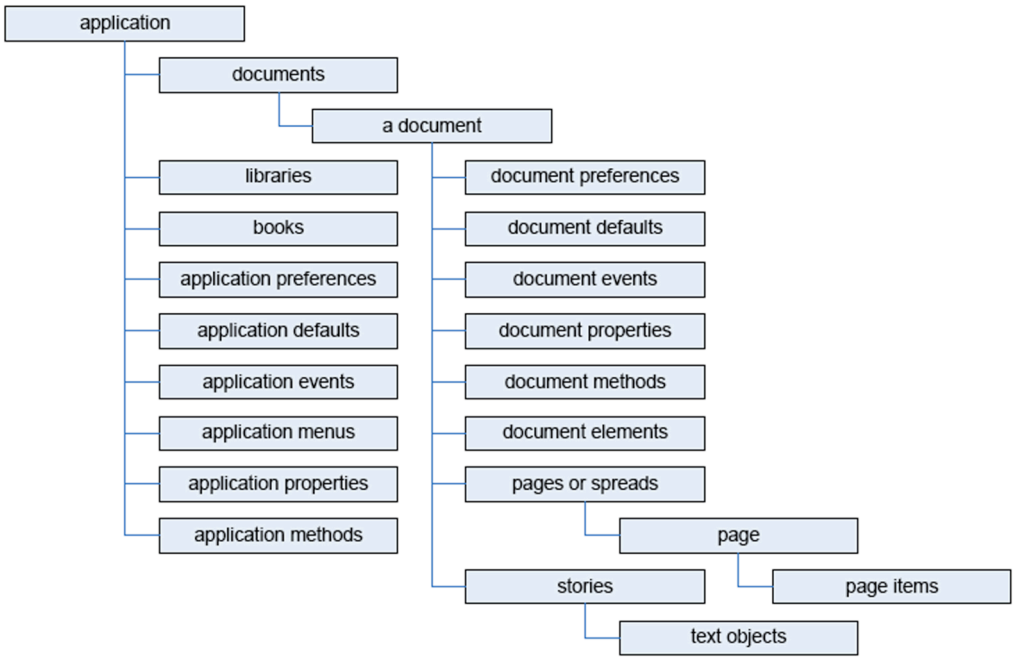

Figura 11 - *InDesign Object Model*: uno schema concettuale

2.3 - *Scripting* e XML: il punto di incontro

Poiché XML permette esclusivamente di gestire i contenuti e non la loro formattazione, integrarne l'utilizzo in un contesto come quello del *layout* di un impaginato non è banale.

L'approccio di InDesign a XML discusso nella prima parte di questo capitolo, sebbene molto flessibile, presenta alcune limitazioni.

- Una volta importati nel documento, gli elementi XML diventano elementi di InDesign che corrispondono alla struttura dell'albero; è quindi importante considerare che la rappresentazione degli elementi XML data dal software è qualcosa di diverso dagli elementi XML medesimi.
- Ogni elemento XML può comparire solo una volta nel *layout*. La sua duplicazione all'interno delle pagine comporta necessariamente una duplicazione dell'elemento XML corrispondente.
- L'ordine in cui gli elementi XML appaiono nel *layout* dipende strettamente

dall'ordine che è stato dato loro nella struttura XML.

- Qualsiasi testo che venga inserito nella *story* associata a un elemento diventa parte del contenuto di quell'elemento.

2.3.1 - Le XML rules

InDesign fornisce un potente set di strumenti per lo *scripting* nella gestione di contenuti XML: gli strumenti che permettono di definire le cosiddette *XML rules*.

Queste semplificano notevolmente il processo di scrittura di programmi per la gestione di elementi XML e migliorano l'efficienza di tutte le operazioni di ricerca, modifica e formattazione dei contenuti.

L'approccio all'uso di strutture XML tramite *XML rules* prevede tre componenti essenziali:

- *XML rule processor*: un oggetto usato per individuare elementi in una struttura XML usando XPath[11] e applicare la *XML rule* appropriata. È importante sottolineare che uno *script* può contenere più *processor objects* e che ognuno di essi è associato ad un set di *XML rules*.
- *Glue code*: un insieme di funzioni fornite da Adobe per semplificare il processo di scrittura delle *XML rules* e l'interazione con l'*XML rule processor*.
- *XML rules*: le azioni vere e proprie che possono essere aggiunte ad uno *script*, elaborate sotto forma di codice. Una regola comprende una condizione X-Path ed una funzione da applicare nel momento in cui tale condizione si verifica. La funzione può contenere una serie di operazioni definibili tramite *scripting* in InDesign, fra le quali modifica della struttura XML, applicazione

[11] XPath è il linguaggio di XSL che consente di individuare gli elementi e gli attributi di un documento XML sui quali verranno applicate le operazioni necessarie per la presentazione dei dati. Esso svolge il compito che nei CSS ha il selettore, cioè l'elemento sintattico di una regola CSS che individua gli elementi da formattare. A differenza dei selettori CSS, peró, XPath è molto più potente e flessibile: consente di creare espressioni, dette espressioni XPath o pattern, che individuano i vari nodi dell'albero di rappresentazione di un documento XML.

della formattazione, creazione di nuove pagine, di oggetti pagina e di interi documenti.

In uno *script* si può definire un numero qualsiasi di *rules* e applicarle all'intera struttura XML di un documento InDesign oppure a suoi sottogruppi.

Quando un *XML rule* è attivata da un oggetto *rule processor*, essa viene usata per applicare modifiche agli elementi coinvolti.

Le *XML rules*, in definitiva, rappresentano qualcosa di simile all'XSLT. Come quest'ultimo usa X-path in contesti generici per la localizzaione di elementi specifici nella struttura, per poi trasformarli in qualche modo, le *XML rules* usano XPath per localizzare e manipolare elementi XML in InDesign; e ancora, come un *template* XSLT usa un *parser* XML per applicare trasformazioni ai dati XML al di fuori di InDesign, un *XML rule processor* è usato per applicare trasformazioni all'interno dell'applicazione di impaginazione professionale.

Le *XML rules* in InDesign supportano un sottoinsieme delle specifiche di XPath 1.0, alcune delle funzionalità principali incluse sono:

- Ricerca di un elemento per nome attraverso l'indicazione del percorso a partire dall'elemento radice: `documento/titolo`.
- Ricerca di elementi tramite l'uso di wildcards[12]: `/doc/*/subtree/node()`.
- Ricerca di un elemento sulla base del valore di uno specifico attributo: `doc/para[@font='Courier']`.
- Ricerca di un elemento con un attributo il cui valore non corrisponde a quello specificato: `doc/para[@font !='Courier']`.
- Ricerca di un elemento figlio in base alla posizione: `/doc/para[3]`.

[12] Carattere che, all'interno di una stringa, non rappresenta sé stesso bensì un insieme di altri caratteri o sequenze di caratteri.

2.3.2 - Perché usare le XML rules?

Nelle prime versioni di InDesign non era implementato l'utilizzo di XPath per navigare la struttura XML ed era per questo necessario scrivere funzioni ricorsive per iterare attraverso essa esaminando ciascun elemento: un approccio difficile e lento.

L'introduzione delle *XML rules* e l'uso di XPath hanno reso più semplice la ricerca di elementi nell'XML delegando agli oggetti *rule processor* questo compito che viene così svolto in maniera molto più efficiente.

2.3.3 - Struttura di una XML rule

Una regola XML è composta da tre parti:

- un nome (stringa);
- un comando XPath (stringa);
- una funzione *apply*.

Il comando XPath individua un punto nella struttura XML; non appena l'*XML rule processor* trova un elemento corrispondente, esegue la funzione *apply* definita.

Di seguito è riportato un esempio:

```
function RuleName() {
      this.name = "RuleNameAsString";
      this.xpath = "ValidXPathSpecifier";
      this.apply = function (element, ruleSet, ruleProcessor){
                        //Blocco istruzioni;
                        return true;
                        }
      };
```

Spesso è necessario utilizzare set di *XML rules*, ovvero *array* di regole che sono applicate, nell'ordine in cui appaiono, da un oggetto *rule processor*.

Un esempio di *XML rule set*:

```
var myRuleSet = new Array (new SortByName,
                           new AddStaticText,
                           new LayoutElements,
                           new FormatElements);
```

Le *XML rules* contenute nell'*array* sono definite in altri punti all'interno dello *script*.

2.3.4 - Il Glue code

Oltre all'oggetto *XML rule processor*, il software InDesign fornisce un pacchetto di funzioni che hanno lo scopo di rendere più facile la scrittura di regole XML.

Sono le funzioni che nel loro complesso costituiscono il *Glue code*:

- `__processRuleSet(root, ruleSet)`: funzione invocata per mandare in esecuzione un insieme di *XML rules*. Richiede due parametri:
  a. un elemento XML che definisce il punto della struttura a partire dal quale si iniziano ad applicare le regole.
  b. Il set delle regole medesime

- `__processChildren(ruleProcessor)`: funzione usata per far sì che il *rule processor* passato come parametro applichi le *XML rules* agli elementi figli dell'elemento di partenza. Di *default*, quando un *XML rule processor* applica una regola al figlio di un elemento XML, il controllo non ritorna alla regola stessa; la funzione `__processChildren` può essere usata per restituire il controllo alla funzione *apply* della *XML rule* dopo che gli elementi figli sono stati processati.

- `__skipChildren(ruleProcessor)`: funzione che impedisce al *rule*

processor di agire sui figli dell'elemento XML attualmente soggetto all'applicazione di una regola. È da usare nei casi in cui si debba spostare o cancellare l'elemento XML corrente oppure si voglia ottimizzare lo *script* saltando parti irrilevanti della struttura.

2.3.5 - L'iterazione attraverso la struttura XML

L'iterazione attraverso una struttura XML è effettuata da un oggetto *XML rule processor* che processa ciascun elemento in base all'ordine nel quale appare nella gerarchia del documento.

L'oggetto attraversa in un unico verso la struttura e visita ogni elemento due volte (nell'ordine padre-figlio-padre, esattamente seguendo la gerarchia dei *tag* annidati in un file XML); per ciascun elemento visitato il *rule processor* prova ad applicare le *XML rules* corrispondenti nell'ordine in cui esse appaiono nel *XML rule set* corrente.

La funzione `__processRuleSet` applica le regole in modalità *depth first*[13]: per ogni "ramo", il *rule processor* visita tutti gli elementi prima di passare al ramo successivo. Dopo l'applicazione di una *XMl rule* ad un elemento, il *processor* continua a cercare regole da applicare ai discendenti di quell'elemento. Questo comportamento può essere alterato tramite l'utilizzo delle funzioni `__processChildren` e `__skipChildren` appena descritte.

In figura 12 e in figura 13 sono rispettivamente illustrate una normale iterazione nella quale si assume che la regola sia applicabile a tutti gli elementi nella struttura XML ed un'iterazione effettuata in caso di utilizzo della funzione `__processChildren`:

[13] La strategia adottata da questo schema di visita consiste nel visitare il grafo sempre più in profondità (fin quando `e possibile). Si comincia con il visitare un nodo qualsiasi v che viene marcato come "scoperto"; la visita prosegue ispezionando tutti gli archi uscenti dal nodo corrente *v* alla ricerca di un nodo *w* collegato a *v* e non ancora visitato.

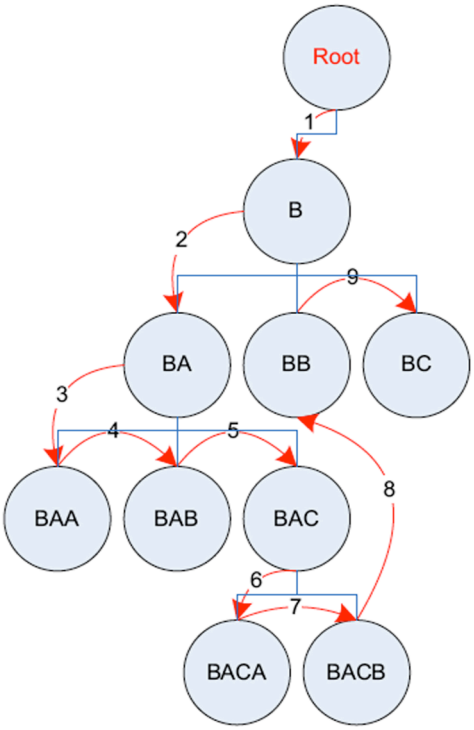

Figura 12 – Normale iterazione a partire dall'elemento radice

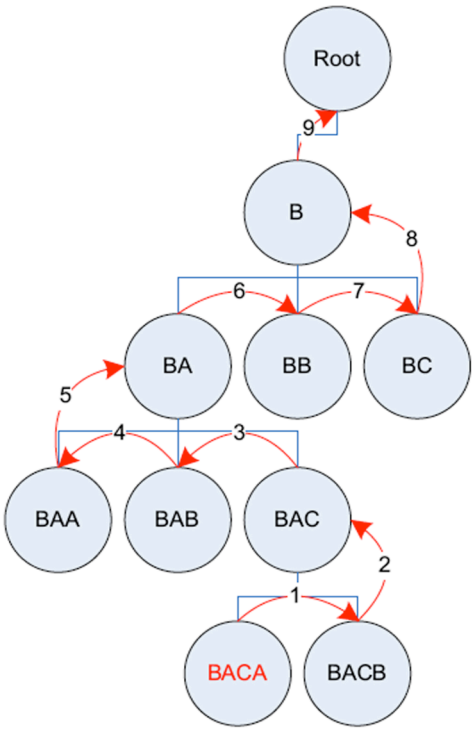

Figura 13 – Iterazione con uso di __processChildren

In figura 14, infine, è mostrata un'iterazione effettuata da un oggetto *rule processor* nel caso di applicazione di regole che, comportano un doppio passaggio per ciascun elemento.

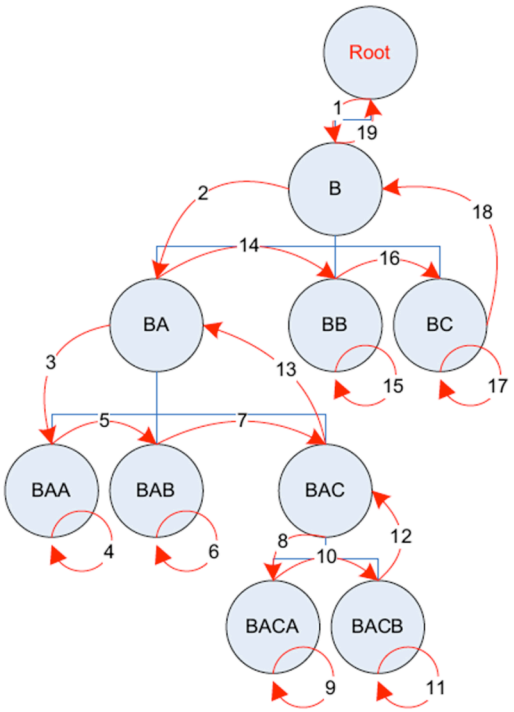

Figura 14 – Iterazione con doppio passaggio per ciascun elemento

2.3.6 - La gestione degli errori

Essendo parte del modello di programmazione di InDesign, gli *script* che utilizzano *XML rules* non differiscono da quelli ordinari e beneficiano degli stessi meccanismi di gestione degli errori.

Fra le diverse tipologie di errore che possono essere "catturate" usando gli strumenti che il linguaggio utilizzato fornisce come ad esempio blocchi `try…catch` le più rilevanti in riferimento all'uso di regole XML sono:

- errori specifici per l'*XML rules processing* che si possono verificare

durante l'inizializzazione di un oggetto *XML rule processor*,

- errori che si verificano nel caso in cui una regola utilizzi uno specificatore XPath non conforme in riferimento alle limitazioni descritte sopra.

2.3.7 - La gestione del flusso di controllo

Nel momento in cui viene mandato in esecuzione uno *script* che contiene *XML rules* il flusso del controllo passa dalla funzione principale a ciascuna regola, e da queste alle funzioni definite nel *Glue code*; tali funzioni passano il controllo all'oggetto *XML rules processor* il quale inizia l'iterazione attraverso la struttura XML visitando gli elementi ed applicando opportunamente le operazioni indicate nelle *XML rules*.

I risultati sono restituiti a ritroso lungo la catena fino a che vengono gestiti da una funzione o causano il lancio di un eccezione nel caso si verifichino errori.

Il diagramma in figura 15 fornisce un'idea della gestione del flusso di controllo:

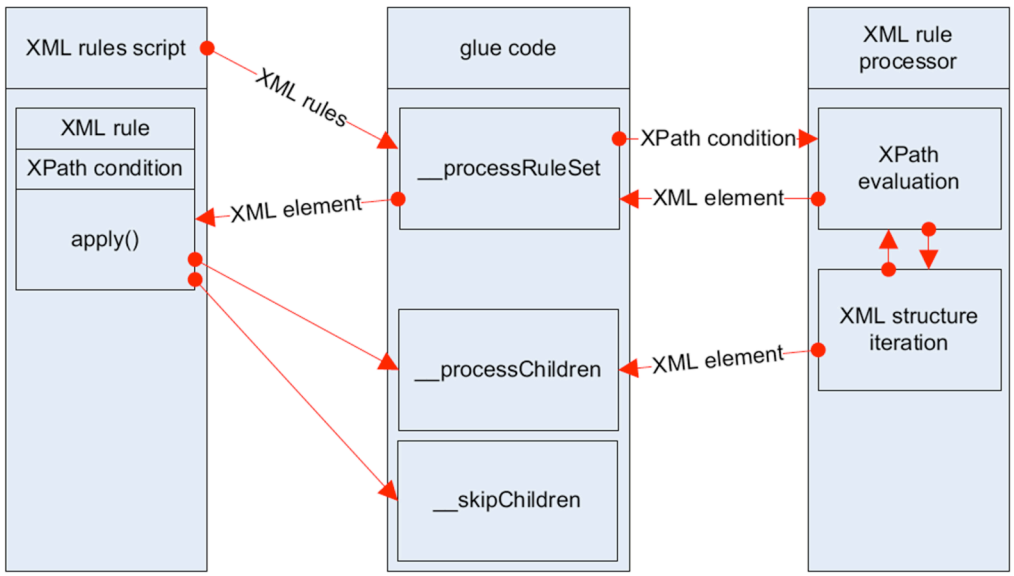

Figura 15 – Gestione del flusso di controllo all'attivazione di una *XML rule*

3
L’esempio d’uso: creazione del catalogo per un editore

Finora è stata discussa in linea teorica una modalità di approccio a progetti di *database publishing* diversa rispetto a quella tradizionale; modalità in virtù della quale il risultato finale, analogo a quello ottenibile attraverso strumenti specifici ma costosi, sia raggiunto in modo alternativo.

È un metodo di lavoro che sfrutta le potenzialità presenti in tecnologie usate spesso per scopi diversi, messe in combinazione con quelle offerte dal software InDesign, ampiamente trattate nel secondo capitolo.

3.1 - Il contesto

Lo studio e la progettazione di uno strumento di impaginazione professionale effettuata a partire da archivi *on line* ha trovato lo spunto per concretizzarsi nell’ambito di un lavoro più ampio di ristrutturazione completa del sito web di una casa editrice locale di Pisa.

È stata presa in esame la necessità di rinnovare, accanto all’interfaccia web e alla gestione interna del sito, anche la logica di creazione di cataloghi cartacei da destinare alla stampa e alla diffusione.

Presupposto fondamentale è stata la volontà di realizzare un sistema che potesse rendere la predisposizione dei contenuti destinati alla stampa del catalogo molto più flessibile, e quindi adattabile alle diverse esigenze che in momenti specifici possono crearsi nell’ambito dell’attività di una piccola casa editrice.

Il catalogo è l’elemento centrale attorno al quale è costruito il sito web; il costante riferimento ad un database rende l’aggiornamento delle pagine dedicate alle categorie di testi un’operazione estremamente semplice da effettuare grazie all’interfaccia intuitiva fornita dal sistema di gestione dei contenuti (CMS) utilizzato.

Tipicamente le risorse economiche e umane necessarie per la creazione e l'aggiornamento dei cataloghi pubblicati da un editore impongono la scelta di stampare una quantità consistente di copie, destinate a rimanere in circolazione per periodi molto lunghi.

Adottare questa soluzione implica l'impossibilità di comunicare su carta l'introduzione di novità e di aggiornamenti che riguardano le pubblicazioni in maniera tanto tempestiva quanto è possibile fare su web.

La gestione e la produzione del catalogo cartaceo può però seguire una logica più vicina a quella che sta alla base dell'aggiornamento del sito stesso: è possibile fare del materiale stampato relativo ai prodotti una risorsa che, rispecchiando costantemente le informazioni archiviate nel database di riferimento, venga finalmente a costituire una forma di comunicazione aggiornabile in tempi ridotti e soprattutto molto più flessibile nella struttura così come nei contenuti.

Questa logica si concretizza nell'uso del sistema di automatizzazione studiato che sfrutta questa semplicità di gestione dei dati archiviati *on line* anche per recuperare e inserire contenuti nelle pagine del catalogo da stampare; un approccio tanto più valido se associato all'uso di tecnologie ormai affermate, come quella della stampa digitale, che permette di produrre un numero variabile di copie a costi estremamente accessibili.

3.2 - Le fasi di lavoro

Una volta effettuata la scelta degli standard e del software di cui servirsi per realizzare il sistema, ha avuto inizio la progettazione dettagliata nella quale si è suddiviso il lavoro in fasi successive, strettamente connesse al *workflow* necessario alla creazione automatizzata del catalogo cartaceo.

- Nella prima fase sono presi in considerazione tutti gli aspetti legati all'utilizzo di XML come livello intermedio fra database e impaginato destinato alla stampa: dalla valutazione della struttura nella quale inserire i

contenuti estratti dall'archivio alla definizione di una grammatica sulla base della quale costruire le relazioni fra i vari elementi.

- Stabilita la struttura in maniera dettagliata è necessario studiare un metodo per la generazione del file XML da importare nel software di impaginazione. A questo scopo è finalizzata la progettazione di uno *script* in linguaggio PHP che integri nella gestione *back-end* del sito la possibilità di creare una struttura XML a partire dalla base di dati *on line* tramite *query* SQL.

- La terza fase del lavoro prevede in primo luogo l'elaborazione di un *template* di esempio ben definito nella sua impostazione grafica per la versione cartacea del catalogo; in secondo luogo una serie di prove di popolamento del *layout* attraverso l'importazione e la gestione dei contenuti XML che si può effettuare dall'interfaccia grafica di InDesign.

- L'ultima fase è quella nella quale vengono studiati algoritmi e soluzioni programmative volte all'automatizzazione dell'impaginazione dei contenuti.

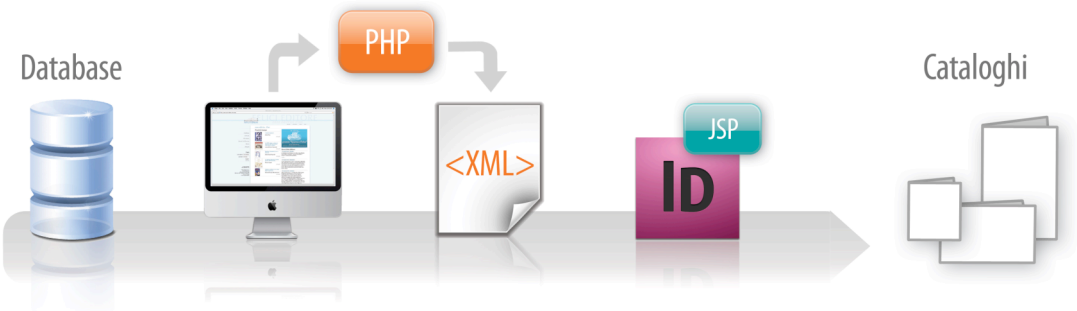

Figura 16 – Le fasi di lavoro

3.3 - Prima fase: XML

3.3.1 - Una DTD per il catalogo

La natura strutturata delle informazioni relative ai libri presenti nel catalogo della casa editrice si riflette nella predisposizione di un database relazionale impostato in modo tale da poter gestire operazioni di diversa natura, comprese quelle di *e-commerce* per la vendita *on line* dei testi.

Ai fini di questa tesi è importante individuare quelle relazioni che esistono fra le tabelle contenenti i dati necessari per la creazione del catalogo, vale a dire tutte le informazioni relative alle caratteristiche fisiche dei libri ed ai loro contenuti.

Informazioni che concettualmente si possono rappresentare con un albero di questo tipo:

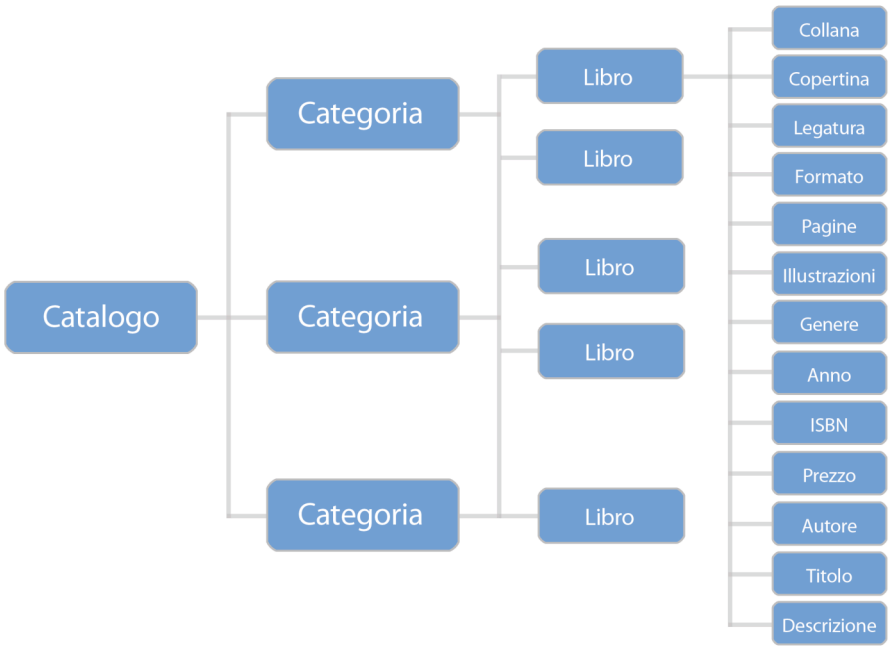

Figura 17 – Albero XML delle informazioni necessarie per il catalogo

Il primo passo verso la creazione di un file XML la cui struttura rifletta quella dell'archivio è la definizione di una sorta di grammatica che delinei i rapporti reciproci fra gli elementi.

Tale definizione viene effettuata in una particolare struttura dati denominata *Document Type Definition* (DTD) costituita da un elenco di dichiarazioni specificate in base alla sintassi XML che individuino gli elementi di interesse attraverso un identificatore generico (il nome, costituito da una stringa di caratteri) e un contenuto, ovvero l'insieme di sotto-elementi che ciascun elemento può contenere e i relativi rapporti in ordine e ricorrenza.

A ogni elemento possono essere associati uno o più attributi che ne specificano ulteriori caratteristiche o funzioni non strutturali. Attraverso questo meccanismo si può creare una DTD per definire un modello e poi assicurarsi che il documento creato sia conforme a tale modello effettuando un'analisi sintattica automatica o *parsing*. Un documento è valido solo se risponde ai vincoli espressi nella sua DTD, qualora ne sia stata creata una. È poi necessario che il documento XML sia ben formato ovvero che tutti i *tag* di apertura e chiusura degli elementi siano collocati in maniera corretta senza i cosiddetti errori di *overlapping*[14].

In accordo all'albero in figura 17, la grammatica creata per l'XML prevede un elemento radice chiamato `<catalogo>` che contiene tutti gli altri, al suo interno sono previsti uno o più elementi `<categoria>` che corrispondono ai diversi ambiti a cui appartengono le pubblicazioni (ad esempio Arte, Narrativa, Poesia, Medicina e così via). Ciascuna categoria contiene uno o più elementi `<libro>` ognuno dei quali comprende una serie di elementi il cui contenuto è relativo alle caratteristiche di ciascun prodotto: l'elemento `<copertina>`, che segue l'elemento `<collana>`, è un riferimento all'immagine ad alta risoluzione in cui è raffigurata la copertina di un libro, di seguito è prevista una serie di elementi in cui sono contenute tutte le informazioni rilevanti: tipo di legatura, formato, numero di pagine, illustrazioni, genere, anno di pubblicazione, ISBN, prezzo, autore, titolo e infine una breve descrizione.

La DTD serve anche a stabilire con precisione quali attributi possono essere assegnati agli elementi e per ciascun attributo è definito il tipo di valore che può assumere.

[14] Nei linguaggi di *markup* il fenomeno di *overlapping* consiste nell'apertura o chiusura errata di un *tag* nei casi in cui non venga rispettato il corretto annidamento degli elementi nella struttura del documento.

L'inserimento delle immagini relative alle copertine dei libri si effettua sulla base del valore dell'attributo `href` dell'elemento `<copertina>`. Tale valore, che come vedremo più avanti sarà generato opportunamente, è una stringa che indica il percorso della cartella all'interno del *file system* che contiene le immagini ad alta risoluzione di ogni copertina.

Un altro elemento che richiede l'inserimento di informazioni specifiche è la categoria della quale occorre indicare due attributi:

- il nome;
- un codice numerico che si riferisce alle percentuali cromatiche (valori di quadricromia) del colore associato alla categoria stessa nelle pagine del sito;

si tratta di scelte fatte in previsione della gestione automatica del contenuto del file che verrà effettuata in seguito e per la quale risulterà decisiva la strutturazione data in questa fase.

La DTD completa sulla base della quale si dovrà validare la struttura XML del catalogo è riportata di seguito:

```
<!ELEMENT catalogo (categoria+)>
<!ELEMENT categoria (libro+)>
<!ELEMENT libro (collana, copertina, legatura, formato,
pagine, illustrazioni, genere, anno, isbn, prezzo, autore,
titolo, descrizione)>
<!ELEMENT copertina (#PCDATA)>
<!ELEMENT collana (#PCDATA)>
<!ELEMENT aspetto (#PCDATA)>
<!ELEMENT formato (#PCDATA)>
<!ELEMENT pagine (#PCDATA)>
<!ELEMENT illustrazioni (#PCDATA)>
<!ELEMENT genere (#PCDATA)>
<!ELEMENT anno (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT prezzo (#PCDATA)>
```

```
<!ELEMENT autore (#PCDATA)>
<!ELEMENT titolo (#PCDATA)>
<!ELEMENT descrizione (#PCDATA)>
<!ATTLIST categoria
      nome CDATA #REQUIRED
      colore CDATA #REQUIRED>
<!ATTLIST copertina
      href CDATA #REQUIRED>
<!ATTLIST illustrazioni
      autore CDATA #IMPLIED>
```

L'elemento radice <catalogo> contiene uno o più elementi <categoria>; a ciascuna categoria appartiene almeno un libro ed ogni elemento libro presenta una serie di elementi figli (<collana>, <copertina>, <collana>, <legatura>, <formato>, <pagine>, <illustrazioni>, <genere>, <anno>, <isbn>, <prezzo>, <autore>, <titolo>, <descrizione>).

3.3.2 – La struttura XML del catalogo

A questo punto è possibile impostare la struttura stessa nella quale gli elementi sono messi in relazione fra loro:

```
<catalogo>
      <categoria nome="…" colore="…">
          <libro>
            <collana>…</collana>
            <copertina href="#" />
            <legatura>…</legatura>
            <formato>…</formato>
            <pagine>…</pagine>
            <illustrazioni>…</illustrazioni>
            <genere>…</genere>
            <anno>…</anno>
```

```
            <isbn>…</isbn>
            <prezzo>…</prezzo>
            <autore>…</autore>
            <titolo>…</titolo>
            <descrizione>…</descrizione>
         </libro>
         <libro>…</libro>
         <libro>…</libro>
         …
      </categoria>
      <categoria>
         <libro>…</libro>
         <libro>…</libro>
         <libro>…</libro>
         …
      </categoria>
      …
</catalogo>
```

Per testare l'efficienza della struttura ed avere un'idea precisa su quello che sarà il risultato della generazione automatica del file a partire dai dati archiviati nel database *on line*, il file è stato popolato manualmente attingendo dai contenuti presenti nelle pagine del vecchio sito relative a libri appartenenti ad alcune fra le categorie previste[15] :

```
<?xml version="1.0" encoding="UTF-8"?>
<catalogo>
  <categoria nome="arte" colore="887700">
    <libro>
     <collana>Arte e tecnica</collana>
     <copertina href="file://immagini/ar01.jpg" />
     <legatura>rilegato</legatura>
```

[15] I valori utilizzati per l'attributo colore dell'elemento categoria dipendono dalle scelte effettuate per la nuova grafica del sito web della casa editrice.

```
  <formato>24x31</formato>
  <pagine>92</pagine>
  <illustrazioni>colori-b/n</illustrazioni>
  <genere>nome_genere</genere>
  <anno>2006</anno>
  <isbn>88-6019-099-1</isbn>
  <prezzo>40,00</prezzo>
  <autore>Annamaria Cavanna</autore>
  <titolo>La pala di Marco di Martino nella Scuola Grande
  della Carità e la pittura a Venezia nella seconda metà
  del Trecento</titolo>
  <descrizione>Tra gli anni Ottanta e Novanta del Trecento
  il pittore Marco di Martino esegue un'importante pala
  per una delle più prestigiose confraternite di Venezia,
  la Scuola Grande di Santa Maria della Carità: si tratta
  dell'unica opera tutt'oggi nota che rechi la firma di
  questo artista. Il libro ripercorre le vicende del
  dipinto dal punto di vista storico e iconografico e
  analizza la figura di questo pittore, sensibile nei
  confronti della prestigiosa tradizione locale incarnata
  dalla figura di Paolo Veneziano, ma tempestivamente
  aggiornato sugli sviluppi contemporanei della civiltà
  figurativa veneziana, da Catarino, a Guglielmo Veneziano
  a Jacobello di Bonomo. L'ultima sezione del libro, a
  cura di Pierluigi Carofano e di Mario Amedeo Lazzari, è
  dedicata alle indagini diagnostiche ai raggi x  e
  all'infrarosso condotte di recente sulla
  pala.</descrizione>
</libro>
<libro>
  <collana>Acquarello. Quaderni del disegno,
  dell'incisione, della grafica</collana>
  <copertina href="file://immagini/ar02.jpg" />
  <legatura>brossura</legatura>
  <formato>24x22</formato>
```

```
    <pagine>88</pagine>
    <illustrazioni>colori-b/n</illustrazioni>
    <genere>nome_genere</genere>
    <anno>2006</anno>
    <isbn>88-6019-080-0</isbn>
    <prezzo>18,00</prezzo>
    <autore>Franco Paliaga</autore>
    <titolo>Livorno nel Seicento: il  porto, le navi, il
    mare. I disegni degli artisti toscani e i dipinti di
    Pietro Ciafferi</titolo>
    <descrizione>Al momento della sua edificazione e della
    nomina ufficiale a città, nel 1606, Livorno divenne nel
    giro di pochi decenni un porto commerciale di livello
    internazionale. In concomitanza con la costruzione e lo
    sviluppo dello scalo marittimo, numerosi furono gli
    artisti, prevalentemente di origine toscana, che
    testimoniarono con i loro splendidi disegni la fervente
    e frenetica attività che si svolgeva lungo i moli,
    ritraendo anche imbarcazioni e navi. Il libro
    ricostruisce la vita, le attività e le strutture del
    porto di Livorno nel Seicento attraverso i disegni, i
    dipinti e le incisioni di questi artisti. Grazie ad un
    materiale grafico inedito o sconosciuto al grande
    pubblico emergono le personalità di Agostino Tassi,
    Filippo Napoletano, Baccio del Bianco, Stefano Della
    Bella, Pietro Ciafferi e Ascanio Della Penna. Di questi
    artisti, che sono da ritenersi gli interpreti più
    rappresentativi del genere paesaggistico toscano delle
    origini, sono tracciate anche delle brevi schede
    biografiche connesse alla loro attività
    livornese.</descrizione>
  </libro>
</categoria>
<categoria nome="storia" colore="14691002">
  <libro>
```

```
    <collana>I libri di Leonardo</collana>
    <copertina href="file://immagini/st01.jpg" />
    <aspetto>brossura</aspetto>
    <formato>17x24</formato>
    <pagine>148</pagine>
    <illustrazioni>colori</illustrazioni>
    <genere>nome_genere</genere>
    <anno>2007</anno>
    <isbn>88-6019-187-8</isbn>
    <prezzo>18,00</prezzo>
    <autore>Angelo Nesti (a cura di)</autore>
    <titolo>Pisa e le acque</titolo>
    <descrizione>Il dibattito intorno alle acque della piana
    di Pisa ha coinvolto generazioni di tecnici e scienziati
    al servizio della dinastia medicea e lorenese. All'apice
    della gran quantità di interventi, progetti e
    realizzazioni (mancate, malfatte o ben riuscite) che
    hanno per oggetto il tratto pisano dell'Arno e le zone
    limitrofe si collocano gli "otto" pareri presentati in
    questo volume.
    Redatti tra la fine del XVI secolo e la metà del XVIII,
    molti di questi sono stati pubblicati nella prima
    edizione della Raccolta d'autori che trattano del moto
    delle acque e nelle successive; altri sono conservati
    manoscritti nei fondi della Biblioteca Universitaria di
    Pisa; altri ancora, pur pubblicati, non hanno incontrato
    ampia diffusione.
    La pubblicazione di queste preziose relazioni risulta
    pertanto fondamentale per comprendere l'atteggiamento
    culturale con cui la società e i suoi governanti si
    ponevano rispetto alle acque, al di là degli aspetti
    tecnico-idraulici o politico-economici ad esse
    sottesi.</descrizione>
  </libro>
</categoria>
```

```
</catalogo>
```

3.4 - Seconda fase: generazione automatica del file XML

Definire nel dettaglio la struttura XML per la produzione del catalogo cartaceo è un'operazione essenziale e propedeutica alle fasi successive del lavoro. Come premesso all'inizio del capitolo, il progetto di impaginazione automatica è inserito in un contesto reale più ampio di rinnovamento del sito web della casa editrice.

Questa rielaborazione è completa: riguarda la grafica e la struttura delle pagine del sito così come la gestione dei contenuti.

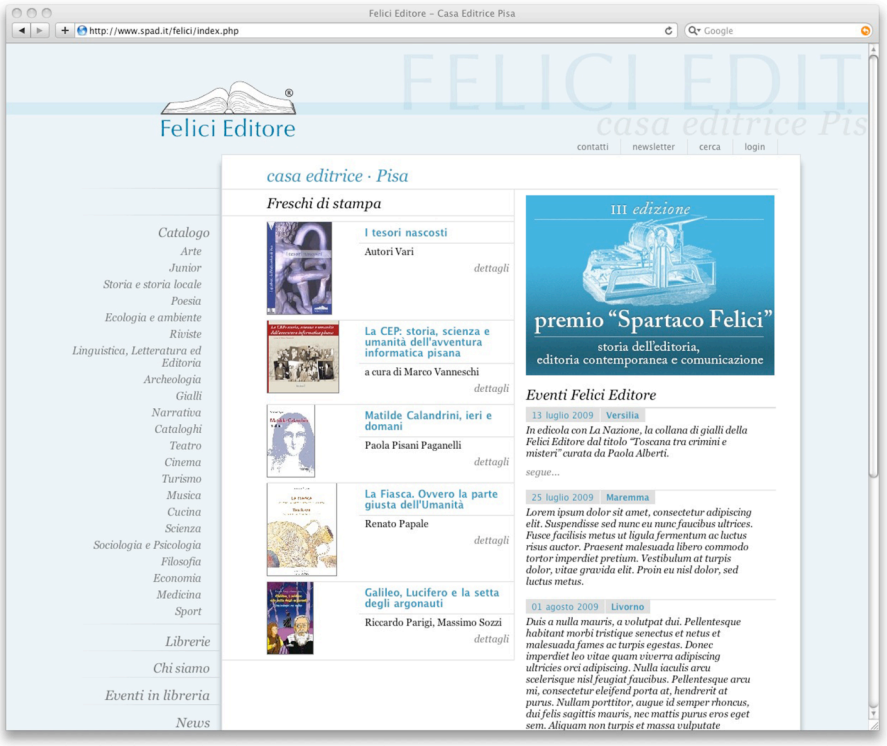

Figura 18 - La nuova homepage del sito dell'editore Felici di Pisa

Il contesto del *database publishing* impone a questo punto la progettazione di un sistema per la generazione automatica del file XML; più precisamente un

meccanismo tramite il quale estrarre tutte le informazioni necessarie dall'archivio *on line* e utilizzarle per popolare opportunamente la struttura XML.

A tal fine è stato scritto un algoritmo tradotto poi nello *script* PHP riportato di seguito:

```
<?
// nome del server
$nome_server = "localhost";
// nome del database a cui connettersi
$nome_db = "Prova_felici";
// username o login d'accesso al database
$nome_utente = "root";
// password d'accesso al database
$db_password = "root";
$connessione=mysql_connect($nome_server,$nome_utente,$db_passw
ord) or die ("ATTENZIONE: non riesco a connettermi con il
server $nome_server<br>");
echo "Passo 1 - Connessione con il server $nome_server
correttamente stabilita<br><br>";
$database = mysql_select_db($nome_db, $connessione)
or die ("ATTENZIONE: non riesco a connettermi con il database
$nome_db<br>");
echo "Passo 2 - Connessione con il database $nome_db
correttamente stabilita<br>";

$query_rsRecordset = "SELECT * FROM books ORDER BY ID DESC";
$rsRecordset = mysql_query($query_rsRecordset, $connessione)
or die(mysql_error());
$row_rsRecordset = mysql_fetch_assoc($rsRecordset);
$totalRows_rsRecordset = mysql_num_rows($rsRecordset);

if($totalRows_rsRecordset > 0) {
      $strXML = "<?xml version=\"1.0\"?>\n";
```

```
$strXML=$strXML . "<catalogo>\n";

do {
      $strXML = $strXML."<libro>\n";
      $strXML =
      $strXML."<id>".$row_rsRecordset['ID']."</id>\n";
      $strXML =
      $strXML."<categoria>".$row_rsRecordset['categoria']
      ."</categoria>\n";
      $strXML =
      $strXML."<collana>".$row_rsRecordset['collana']."</
      collana>\n";
      $strXML =
      $strXML."<legatura>".$row_rsRecordset['legatura']."
      </legatura>\n";
      $strXML =
      $strXML."<formato>".$row_rsRecordset['formato']."</
      formato>\n";
      $strXML =
      $strXML."<pagine>".$row_rsRecordset['pagine']."</pa
      gine>\n";
      $strXML =
      $strXML."<illustrazioni>".$row_rsRecordset['illustr
      azioni']."</illustrazioni>\n";
      $strXML =
      $strXML."<lingua>".$row_rsRecordset['genere']."</li
      ngua>\n";
      $strXML =
      $strXML."<anno>".$row_rsRecordset['anno']."</anno>\
      n";
      $strXML =
      $strXML."<isbn>".$row_rsRecordset['isbn']."</isbn>\
      n";
```

```
            $strXML =
            $strXML."<prezzo>".$row_rsRecordset['prezzo']."</pr
            ezzo>\n";
            $strXML =
            $strXML."<autore>".$row_rsRecordset['autore']."</au
            tore>\n";
            $strXML =
            $strXML."<titolo>".$row_rsRecordset['titolo']."</ti
            tolo>\n";
            $strXML =
            $strXML."<descrizione>".$row_rsRecordset['descrizio
            ne']."</descrizione>\n";
            $strXML = $strXML."</libro>\n";
            }
      while ($row_rsRecordset=
            mysql_fetch_assoc($rsRecordset));
            $strXML = $strXML . "</catalogo>";
            $XMLFile = fopen("catalogo.xml", "w") or die
            ("can't open file");

      fwrite($XMLFile, $strXML);
      fclose($XMLFile);
}

      mysql_free_result($rsRecordset);
?>
```

Una volta stabilita la connessione col database, alcune funzioni messe a disposizione dal linguaggio PHP per l'estrazione dei singoli *record* da una tabella sono usate in combinazione con l'esecuzione di un ciclo `do-while` per aggiornare una variabile di tipo stringa il cui contenuto è sostanzialmente quello del file XML da generare.

La creazione effettiva dell'XML è implementata utilizzando i metodi `fopen`, `fwrite` ed `fclose` che servono rispettivamente per creare, riempire e chiudere il nuovo file.

Nello *script* si utilizza una *query* SQL per estrarre da una tabella-prodotti i dati relativi a ciascun libro. Naturalmente questo rappresenta solo il nucleo essenziale della soluzione per l'estrazione dei dati: si ipotizza la connessione ad un database costituito da un'unica tabella già progettata per contenere tutte le informazioni necessarie al catalogo cartaceo.

Nel contesto reale, la struttura della base di dati gestita attraverso il CMS messo a disposizione da un'azienda web specializzata si presenta ben più complessa perché utilizzata in diversi contesti e per diverse tipologie di prodotto. I dati da recuperare sono dislocati in tabelle legate fra loro da un complesso sistema di relazioni regolato dal meccanismo delle chiavi primarie e secondarie. Inoltre per poter considerare completo il file generato, è necessario integrare lo *script* nel meccanismo di gestione *back-end* del sito e modificarlo con l'aggiunta degli attributi XML il cui valore ricopre un ruolo fondamentale nell'impaginazione del catalogo.

Per questo motivo l'idea contenuta nello *script* è stata poi modellata sulla struttura dell'archivio lavorando in stretta collaborazione con l'azienda fornitrice del sistema di gestione dei contenuti; tale collaborazione ha avuto non solo lo scopo di adeguare l'applicazione PHP al contesto reale del nuovo sito web della casa editrice, ma è stata anche fondamentale per arricchire il programma di tutti i dettagli necessari per la generazione del file XML nella sua completezza.

3.5 - Terza fase: definizione del *template* per il catalogo cartaceo

La predisposizione della struttura XML è completata; il file generato direttamente dall'applicazione PHP è pronto per essere importato in InDesign e utilizzato per impaginare il prodotto cartaceo.

L'uso della struttura XML è però solo uno dei passi da effettuare nel processo realizzativo del catalogo: come ogni documento destinato alla stampa professionale,

esso è il frutto di un lavoro che richiede competenze di grafica e di tipografia per la progettazione dell'impaginato in tutti i suoi aspetti.

Nella prospettiva di testare il sistema di creazione automatica del catalogo in un contesto realistico è stato affrontato l'*iter* di progettazione del documento stesso fin dalle fasi iniziali.

3.5.1 - La specifica dei contenuti

Automatizzare il processo di impaginazione del catalogo apre diverse possibilità nella scelta dei contenuti da inserire; il potente strumento di *scripting* permette infatti di implementare soluzioni diverse sfruttando la possibilità di prelevare contenuti direttamente dal database di riferimento in maniera flessibile.

In sostanza, disponendo sempre di tutte le informazioni necessarie e potendo, in pochi *click*, creare un prodotto cartaceo a partire da esse, diventa facile realizzare cataloghi specifici (ad esempio sulle ultime uscite oppure su una categoria in particolare) oltre a quello generico che contiene le informazioni relative a tutte le pubblicazioni.

Lo scenario in cui si inserisce l'esempio d'uso del sistema descritto in queste pagine prevede la necessità di creare il catalogo completo dei libri pubblicati dalla casa editrice, e di questo bisogna tener conto in fase di progettazione del *template*.

3.5.2 - Il lavoro del grafico

Un'operazione preliminare importante nell'ideazione di un *layout* per il nuovo catalogo dell'editore è stato lo studio e l'analisi della vecchia versione dell'impaginato, relativa al periodo compreso tra il 2005 e il 2008.

Figura 19 – Il catalogo 2005-2008

Dall'osservazione del catalogo emerge la precisa strutturazione del documento: per ogni pagina vengono presentati due libri di cui si mostrano una piccola anteprima della copertina e tutte le informazioni sul testo corredate da una descrizione dei contenuti. Si tratta di una impostazione abbastanza standard alla quale è opportuno attenersi anche nel produrre la versione rinnovata del documento.

Nel vecchio catalogo si notano alcune imprecisioni e alcune sbavature in relazione all'inserimento dei contenuti entro i limiti della gabbia tipografica. Ci sono poi alcuni errori probabilmente non notati in fase di revisione.

3.5.3 - Il nuovo progetto grafico

Inizia a questo punto il lavoro di impostazione del nuovo *template* da riempire successivamente coi contenuti del file XML generato.

È un rinnovamento per lo più estetico e di formato: i contenuti sono ripartiti secondo un *layout* diverso che comunque prevede l'inserimento di due articoli per

pagina in una brossura di formato A5 orizzontale, scelta finalizzata a dare un'impostazione più ariosa e chiara ai contenuti.

Creazione delle mastro

L'impostazione del *layout* è iniziata con la creazione dei fogli mastro nella quale si è tenuto conto del modo in cui i contenuti devono essere distribuiti nelle pagine. Nel caso specifico sono stati creati due modelli: nel primo è definita la gabbia tipografica all'interno della quale si andranno ad inserire i contenuti principali del catalogo, ovvero le informazioni relative a ciascun libro. Le due pagine che compongono il foglio mastro hanno una struttura identica; lo spazio è suddiviso in due parti uguali ognuna destinata ad accogliere i contenuti riguardanti un singolo libro.

Un aspetto importantissimo di cui si è tenuto conto in questa fase di creazione del *layout* è il fatto che l'ordine degli elementi negli spazi preposti all'interno della pagina deve rispecchiare pienamente quello in cui compaiono nella struttura XML affinché l'impaginazione automatica vada a buon fine.

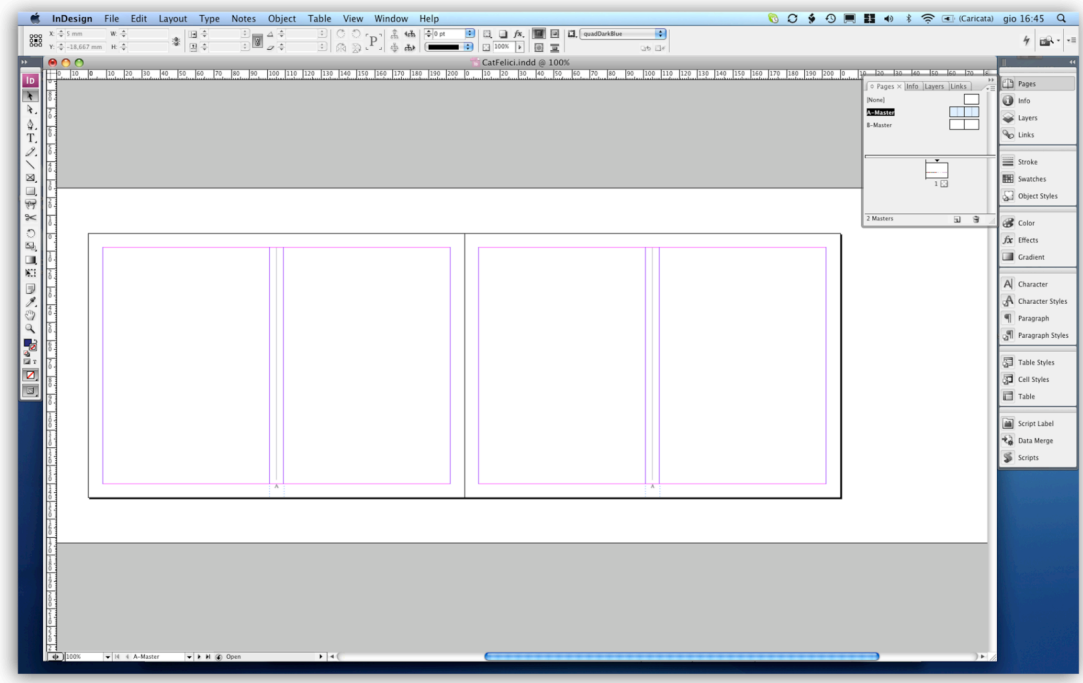

Figura 20 - Il foglio mastro principale per le pagine del catalogo

Il secondo modello è invece relativo alle pagine che separano ciascuna sezione del catalogo dalla successiva. L'impostazione è molto essenziale e prevede solo l'inserimento del nome della categoria che viene introdotta.

Impostazione degli stili

Nel catalogo che si sta creando, così come in tutti i tipi di pubblicazione, l'aspetto delle diverse parti del testo si pone in stretta relazione col ruolo svolto da ciascun elemento della pagina per la comunicazione chiara delle informazioni.

A questo stadio del lavoro l'inserimento di contenuto in una pagina mastro determina il risultato illustrato in figura 21 nel quale, come si può vedere non è applicato alcun tipo di formattazione ai contenuti.

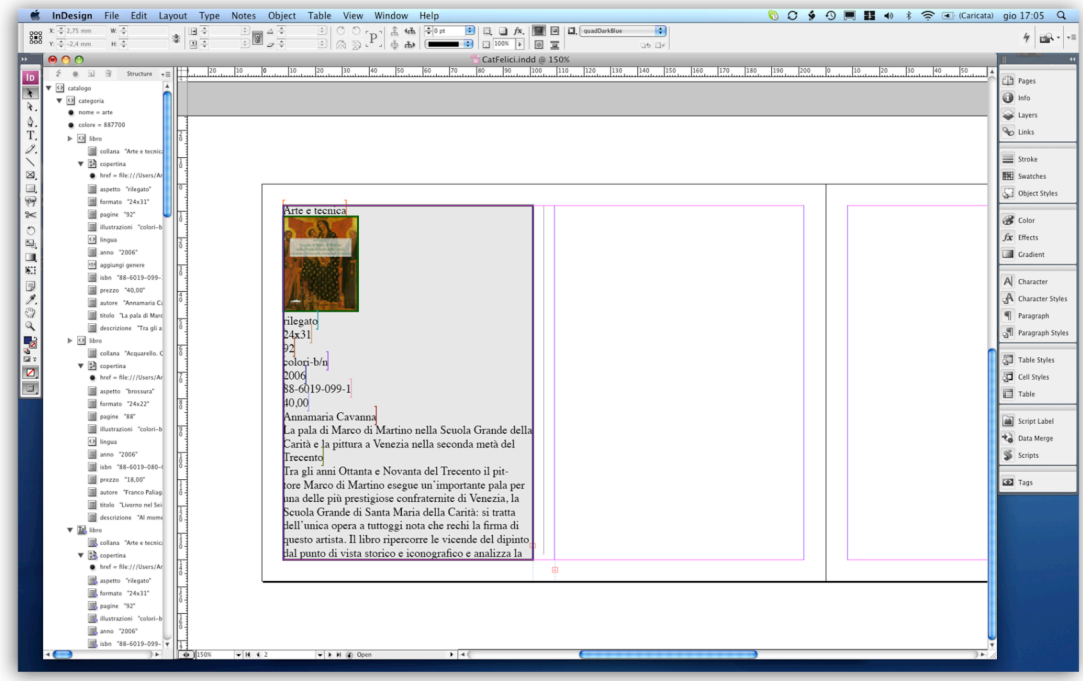

Figura 21 – Il contenuto della pagina prima dell'applicazione degli stili di formattazione

Applicare manualmente tutte le impostazioni di formattazione, dalla scelta del carattere alla grandezza, al colore e al posizionamento per ogni singolo elemento, si rivelerebbe un'attività alla quale dedicare molte ore di lavoro; tante più quanto più ricco è il contenuto e quanto più elevato è il numero di pagine del documento.

Una delle caratteristiche del software InDesign che permettono un notevole risparmio di tempo per eseguire operazioni noiose e ripetitive come questa è la possibilità di definire *stili carattere* e *stili paragrafo* che possono poi essere applicati con la rapidità di un *click* del mouse agli elementi del testo interessati.

Per la realizzazione del *template* del catalogo della casa editrice è dunque opportuno definire stili specifici tenendo sempre presente il modo in cui verrà riempito il documento che si sta progettando.

A proposito della formattazione dei contenuti XML, nel secondo capitolo sono state illustrate diverse modalità di formattazione automatica: nell'esempio d'uso è stata scelta la mappatura dei *tag* sugli stili in base al nome; tale scelta impone di applicare ai diversi stili lo stesso nome dei *tag* XML che contengono le informazioni corrispondenti: d'altronde dedicare un minimo di attenzione a questa semplice operazione comporta un vantaggio enorme in termini di tempo nel momento in cui testi e immagini saranno inseriti nelle pagine ed il software assocerà in automatico i corretti stili di formattazione ad ogni singolo elemento.

Grazie alla possibilità di clonare oggetti ripetuti, il meccanismo può essere applicato anche agli elementi testuali statici che non sono contenuti nell'XML ma sono strettamente correlati ad essi e si ripresentano identici in tutte le pagine.

Nel caso del catalogo al quale si sta lavorando gli unici contenuti di questo tipo sono le etichette che precedono le informazioni relative alla collana di appartenenza, all'aspetto fisico del libro, al codice identificativo ISBN, al genere e all'anno di pubblicazione.

Per sfruttare fino in fondo i vantaggi dell'uso degli stili di formattazione dell'impaginato si è poi definito uno *stile oggetto* da applicare all'immagine della copertina dei libri; essa rappresenta un esempio di oggetto ancorato del quale si possono definire le opzioni di posizionamento rispetto agli altri elementi nella pagina e ai margini della pagina stessa.

Il risultato dell'applicazione dei diversi stili ad una pagina del catalogo illustrato in figura 22 dimostra l'efficacia di questo strumento.

I testi e le immagini sono ora collocati nella posizione desiderata all'interno della pagina; caratteristiche estetiche come i font, i colori, le dimensioni, l'interlinea, i margini sono applicate correttamente e tale operazione non ha richiesto l'impiego di lunghi tempi di lavorazione

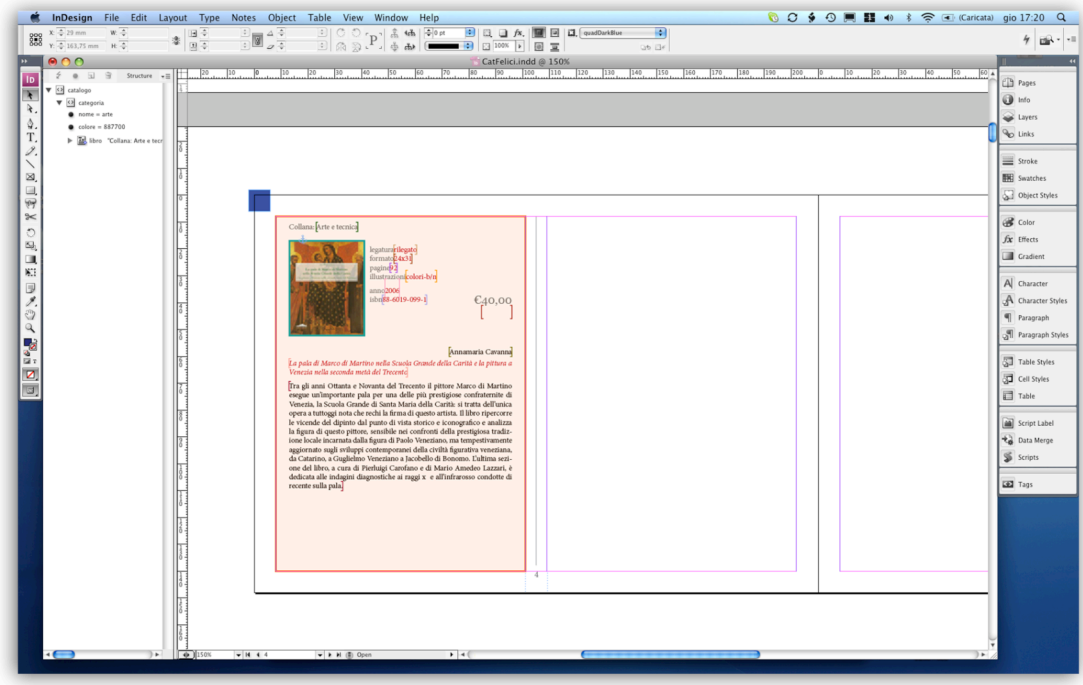

Figura 22 - Il risultato dell'applicazione degli stili di formattazione

La definizione dei fogli mastro, degli stili carattere e paragrafo e degli stili oggetto e la creazione di una copertina per il catalogo costituiscono, nel loro insieme, la stesura del *template* di partenza.

Grazie ai meccanismi di automatizzazione che saranno elaborati, l'aggiunta di fogli nel catalogo avverrà dinamicamente in base alla quantità di contenuti importati nell'impaginato.

3.6 - Quarta fase: programma per l'impaginazione automatica

L'ultima fase del lavoro ne rappresenta anche il nucleo principale: ciò che caratterizza maggiormente questo diverso approccio al *database publishing* pensato per piccole realtà editoriali come alternativa all'uso di applicazioni specializzate e costose.

Si parla della scrittura di un programma JavaScript appositamente creato per automatizzare tutto il processo di impaginazione del catalogo la cui struttura è stata appena delineata.

Il contesto di funzionamento dello *script* è quello dell' *Object model* di InDesign (descritto nel secondo capitolo) che è stato studiato in maniera approfondita allo scopo di sfruttare al meglio gli strumenti (oggetti e loro proprietà e metodi) messi a disposizione del programmatore.

La documentazione[16] relativa all'ExtendScript (la versione estesa del linguaggio JavaScript creata da Adobe) è stata un costante riferimento nella stesura del codice e nella ricerca di soluzioni efficienti in termini di complessità degli algoritmi.

L'ambiente di sviluppo è l'*ExtendScript ToolKit*, anch'esso illustrato in precedenza, che costiutuisce il miglior *editor* da utilizzare, in quanto pensato per integrarsi pienamente con il software InDesign.

3.6.1 - Lo script nel dettaglio

La prima parte del codice scritto per automatizzare l'impaginazione del catalogo contiene una serie di funzioni standard predefinite, necessarie per poter creare ed utilizzare regole XML.

Fra esse sono presenti anche le funzioni `__processChildren` e `__skipChildren` il cui funzionamento è stato illustrato in precedenza.

[16] http://www.indesignscriptingreference.com/CS3/JavaScript/

Tali funzioni sono precedute dalla definizione di un costruttore per l'oggetto *ruleProcessor* che sarà usato per individuare, all'interno della struttura XML, gli elementi ai quali applicare, di volta in volta, le regole definite più avanti nello *script*.

```
function ruleProcessorObject(ruleSet, ruleProcessor) {
    this.ruleSet = ruleSet;
    this.ruleProcessor = ruleProcessor;
}


function __makeRuleProcessor(ruleSet, prefixMappingTable){

    var pathArray = new Array();
    for (i=0; i<ruleSet.length; i++)
    {
         pathArray.push(ruleSet[i].xpath);
    }

    try{
        var ruleProcessor =
        app.xmlRuleProcessors.add(pathArray,
        prefixMappingTable);
    }
    catch(e){
        throw e;
    }
    var rProcessor =  new ruleProcessorObject(ruleSet,
    ruleProcessor);
    return rProcessor;
}

function __deleteRuleProcessor(rProcessor) {
```

```
    rProcessor.ruleProcessor.remove();

    delete rProcessor.ruleProcessor;
    delete rProcessor.ruleSet;

    delete   rProcessor;
}

function __processRuleSet (root, ruleSet, prefixMappingTable)
    {
    var mainRProcessor = __makeRuleProcessor(ruleSet,
    prefixMappingTable);

        try {
            __processTree(root, mainRProcessor);
            __deleteRuleProcessor(mainRProcessor);
        } catch (e) {
            __deleteRuleProcessor(mainRProcessor);
            throw e;
        }
}

function __processTree (root, rProcessor)
    {
    var ruleProcessor = rProcessor.ruleProcessor;
    try
    {
        var matchData =
        ruleProcessor.startProcessingRuleSet(root);
        __processMatchData(matchData, rProcessor);

        ruleProcessor.endProcessingRuleSet();
    }
    catch (e)
    {
```

```
            ruleProcessor.endProcessingRuleSet();
            throw e;
        }
    }

function __processChildren(rProcessor){
    var ruleProcessor = rProcessor.ruleProcessor;
    try
    {
        var matchData =
        ruleProcessor.startProcessingSubtree();
        __processMatchData(matchData, rProcessor);
    }
    catch (e)
    {
        ruleProcessor.halt();
        throw e;
    }
}

function __processMatchData(matchData, rProcessor){
    var ruleProcessor = rProcessor.ruleProcessor;
    var ruleSet = rProcessor.ruleSet;
    while (matchData != undefined)
    {
        var element = matchData.element;
        var matchRules = matchData.matchRules;
        var applyMatchedRules = true;

        for (var i=0; i<matchRules.length &&
        applyMatchedRules && !ruleProcessor.halted; i++)
        {
        applyMatchedRules = (false ==
        ruleSet[matchRules[i]].apply(element, rProcessor));
        }
```

```
        matchData = ruleProcessor.findNextMatch();
    }
}

function __skipChildren(rProcessor){
    rProcessor.ruleProcessor.skipChildren();
}
```

Si nota come all'interno delle funzioni sopra riportate siano presenti gli opportuni blocchi `try_catch` per la gestione degli errori nell'uso di regole XML.

Settaggio delle preferenze

Subito dopo inizia la sezione del programma che contiene le istruzioni più specificamente finalizzate alla creazione del documento.

Come detto, tramite codice è possibile compiere pressocchè tutte le azioni effettuabili dall'interfaccia grafica del programma InDesign; fra queste rientrano le impostazioni delle preferenze generali per un documento, che normalmente sono settate dal menu dedicato (figura 23).

Da qui è possibile definire numerose opzioni che riguardano le caratteristiche generali di un documento e variano in base alle esigenze dettate dal lavoro; ad impostazioni generali se ne affiancano di più specifiche riguardanti ad esempio la composizione del testo, l'interfaccia, strumenti grammaticali come la correzione automatica, la qualità di visualizzazione dei contenuti e la scelta delle unità di misura da utilizzare.

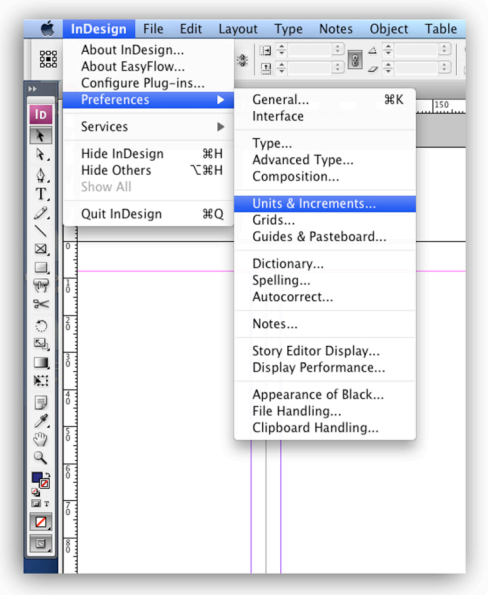

Figura 23 – Menu delle preferenze di InDesign

In particolare è necessario impostare in millimetri la misurazione orizzontale e verticale visualizzata tramite righelli. I righelli sono poi impostati in maniera tale da far coincidere l'inizio della misurazioni con il punto di origine della pagina come illustrato in figura 24:

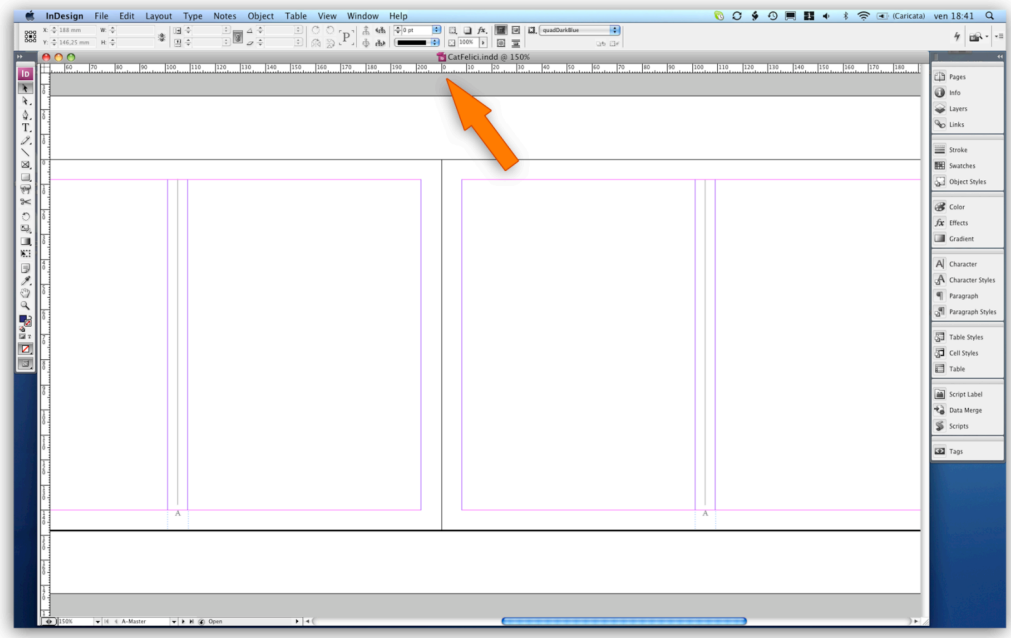

Figura 24 – L'origine della misurazione coincide con quella della pagina

La traduzione in linguaggio JavaScript nel contesto del modello a oggetti di InDesign è la seguente:

```
var myDocument = app.documents.item(0);

myDocument.viewPreferences.horizontalMeasurementUnits
=MeasurementUnits.millimeters;
myDocument.viewPreferences.verticalMeasurementUnits =
MeasurementUnits.millimeters;
myDocument.viewPreferences.rulerOrigin =
RulerOrigin.pageOrigin;
```

In primo luogo è definita la variabile `myDocument` che contiene il documento sul quale si sta lavorando; successivamente vengono impostati, secondo le esigenze, i valori delle proprietà dell'oggetto che si riferiscono alla scelta delle unità di misura.

Sempre alla fase di impostazione delle preferenze appartengono i due blocchi successivi.

Nel primo sono stabilite le modalità di visualizzazione degli elementi XML importati nel documento:

```
var myXMLViewPreferences = myDocument.xmlViewPreferences;

myXMLViewPreferences.showAttributes = true;
myXMLViewPreferences.showStructure = true;
myXMLViewPreferences.showTaggedFrames = true;
myXMLViewPreferences.showTagMarkers = true;
myXMLViewPreferences.showTextSnippets = true;
```

La variabile `myXMLViewPreferences` è utilizzata per impostare su *true* le proprietà dell'oggetto `document` relative alla visibilità di una serie di elementi: in particolare questo frammento di codice permette di aprire il pannello Struttura,

visualizzare al suo interno elementi ed attributi XML e attivare la visualizzazione di marcatori grafici anche sui contenuti importati nelle pagine del documento.

Figura 25 - I contenuti degli elementi XML sono marcati con parentesi quadre di diversi colori

Nel secondo blocco sono invece impostate le preferenze di importazione del file XML: alcune delle opzioni elencate e spiegate in Tabella 1 nel secondo capitolo.

```
var myXMLImportPreferences = myDocument.xmlImportPreferences;

myXMLImportPreferences.allowTransform = false;
myXMLImportPreferences.createLinkToXML = false;
myXMLImportPreferences.ignoreUnmatchedIncoming = false;
myXMLImportPreferences.ignoreWhitespace = false;
myXMLImportPreferences.importCALSTables = false;
myXMLImportPreferences.importStyle =
```

```
XMLImportStyles.mergeImport;
myXMLImportPreferences.importTextIntoTables = false;
myXMLImportPreferences.importToSelected = false;
myXMLImportPreferences.removeUnmatchedExisting = false;
myXMLImportPreferences.repeatTextElements = false;
```

Tutte le proprietà sono settate su *false* perché si desidera che la struttura XML non subisca variazioni particolari in fase di importazione.

Anche in questo caso si tratta di scelte effettuabili accedendo alla voce corrispondente del menu dedicato nell'interfaccia dell'applicazione:

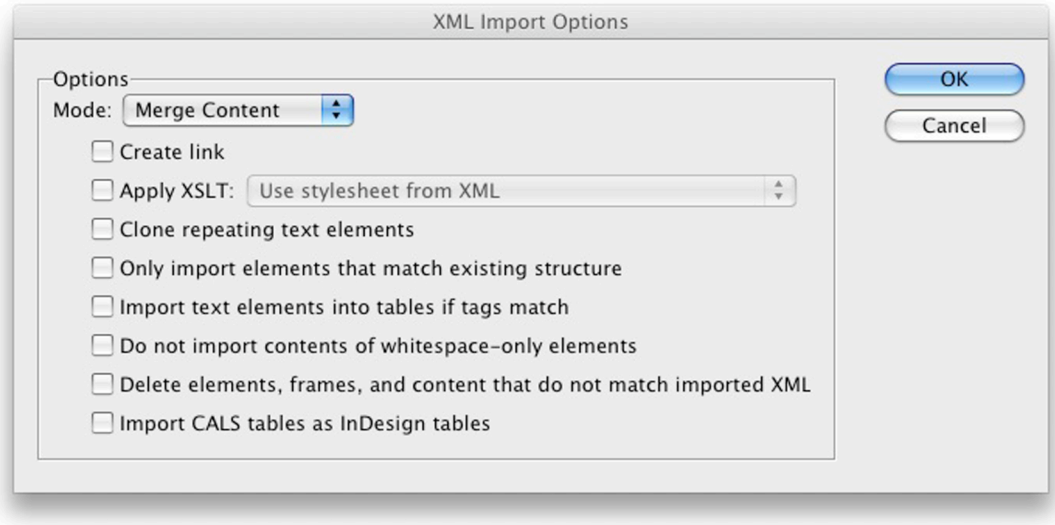

Figura 26 – La finestra delle opzioni di importazione di file XML

Importazione dell'XML

Terminata l'esecuzione delle istruzioni per il settaggio delle preferenze si passa alla prima operazione diretta sul documento: l'importazione del file XML.

```
myDocument.importXML(File("/percorso/catalogo.xml"));
```

Questa singola istruzione equivale alla procedura di importazione di un file XML eseguibile da menu contestuale all'interno del Pannello struttura.

Il metodo `importXML` dell'oggetto `document` prende come parametro il percorso assoluto o relativo del file di interesse.

Un presupposto fondamentale è che in fase di generazione del file XML tramite PHP si sia stabilita precisamente la collocazione del documento all'interno del file system. La scelta più ovvia è quella di creare un'unica cartella che contenga tutti i tipi di documento che sono coinvolti nel flusso di lavoro, in particolare il *template* InDesign (.indd) e il file XML.

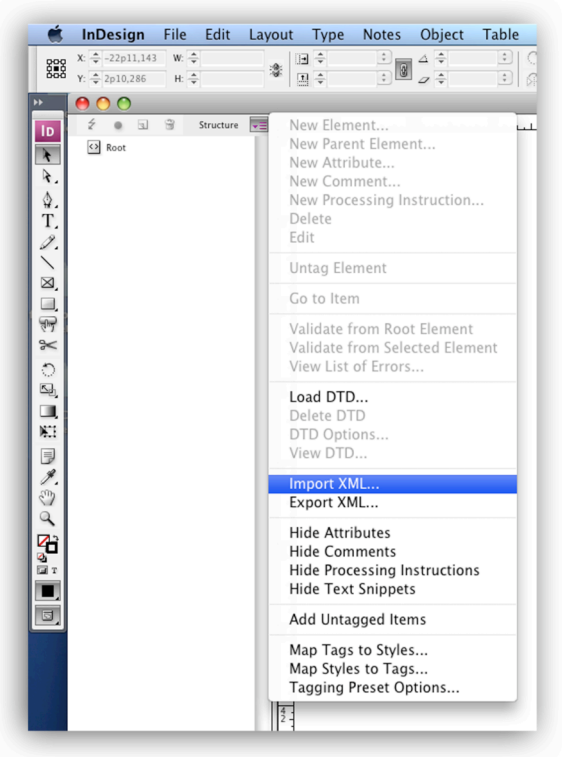

Figura 27 – Importazione dell'XML dal menu del pannello Struttura

Mappatura dei tag

Il passo successivo da effettuare è la mappatura dei *tag* sugli stili già impostati in fase di creazione del *template*.

```
myDocument.xmlImportMaps.add(myDocument.xmlTags.item("collana"
), myDocument.paragraphStyles.item("collana"));

myDocument.xmlImportMaps.add(myDocument.xmlTags.item("aspetto"
), myDocument.paragraphStyles.item("aspetto"));

myDocument.xmlImportMaps.add(myDocument.xmlTags.item("formato"
), myDocument.paragraphStyles.item("formato"));

myDocument.xmlImportMaps.add(myDocument.xmlTags.item("lingua")
, myDocument.paragraphStyles.item("lingua"));

myDocument.xmlImportMaps.add(myDocument.xmlTags.item("anno"),
myDocument.paragraphStyles.item("anno"));

myDocument.xmlImportMaps.add(myDocument.xmlTags.item("isbn"),
myDocument.paragraphStyles.item("isbn"));

myDocument.xmlImportMaps.add(myDocument.xmlTags.item("prezzo")
, myDocument.paragraphStyles.item("prezzo"));

myDocument.xmlImportMaps.add(myDocument.xmlTags.item("illustra
zioni"), myDocument.paragraphStyles.item("illustrazioni"));

myDocument.xmlImportMaps.add(myDocument.xmlTags.item("pagine")
, myDocument.paragraphStyles.item("pagine"));

myDocument.xmlImportMaps.add(myDocument.xmlTags.item("descrizi
one"), myDocument.paragraphStyles.item("descrizione"));
```

```
myDocument.xmlImportMaps.add(myDocument.xmlTags.item("autore")
, myDocument.paragraphStyles.item("autore"));

myDocument.xmlImportMaps.add(myDocument.xmlTags.item("titolo")
, myDocument.paragraphStyles.item("titolo"));

myDocument.mapXMLTagsToStyles();
```

Dopo la specifica dei singoli stili da utilizzare effettuata facendo uso del metodo `add` degli oggetti `xmlImportMaps` presenti nel documento, gli stili sono applicati ai contenuti marcati coi *tag* corrispondenti tramite il metodo `mapXMLTagsToStyles` dell'oggetto `document`.

Funzioni per l'aggiunta di nuove pagine

A questo punto dello script vengono definite due funzioni fondamentali che saranno invocate dalla funzione principale del programma.

La prima, chiamata `aggiungiIntroCategoria`, contiene le istruzioni necessarie per aggiungere un foglio che introduce una nuova categoria del catalogo.

Al nuovo foglio viene applicata la mastro corrispondente (B-Master).

```
function aggiungiIntroCategoria(){
    var myDocument = app.documents.item(0);
    with(myDocument){
        var nuovaIntroCat = spreads.add();
        nuovaIntroCat.appliedMaster =
        app.activeDocument.masterSpreads.item("B-Master");

    }
    return nuovaIntroCat;
}
```

La seconda funzione, `aggiungiConteCategoria`, è invocata per aggiungere al catalogo un nuovo foglio destinato a contenere informazioni sui libri; una volta effettuato l'inserimento al foglio viene applicata la prima mastro disegnata dal grafico (A-Master).

Inoltre viene gestito l'inserimento di elementi grafici che servono come ulteriore elemento di distinzione visiva fra una categoria e l'altra: in base alla valutazione di un serie di condizioni, tramite l'istruzione `switch` viene collocato automaticamente nella parte superiore di ogni pagina un quadrato del colore associato alla categoria che si sta costruendo. L'elemento valutato corrisponde all'unico parametro passato alla funzione il cui valore numerico rappresenta le percentuali per la rappresentazione in quadricromia di ogni colore. Si spiega così la presenza di un attributo `colore` previsto per ogni elemento `<categoria>` nella struttura XML di partenza.

Anche in questo caso sono utilizzati stili oggetto specifici per associare ad ogni elemento decorativo il colore corrispondente.

```
function aggiungiConteCategoria(coloreCat){

    var myDocument = app.documents.item(0);
    with(myDocument){
        var nuovoFoglioCat = spreads.add();
        nuovoFoglioCat.appliedMaster =
        app.activeDocument.masterSpreads.item("A-Master");
        var quadLeft =
        nuovoFoglioCat.pages.item(0).rectangles.add({geometri
        cBounds:["-2mm","-2mm", "6mm", "6mm"]});
        var quadRight =
        nuovoFoglioCat.pages.item(1).rectangles.add({geometri
        cBounds:["-2mm","204mm", "6mm", "212mm"]});


        switch (coloreCat){
```

```
case '887700':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadBlue"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadBlue"));break;

case '24969118':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadBrown"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadBrown"));break;

case '14691002':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadChocolate"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadChocolate"));break;

case '7100781':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadCrimson"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadCrimson"));break;

case '8426464':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadDarkCyan"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadDarkCyan"));break;

case '883510029':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadDarkGreen"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadDarkGreen"));break;
```

```
case '5610061':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadDarkMagenta"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadDarkMagenta"));break;

case '663810026':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadDarkOliveGreen"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadDarkOliveGreen"));break;

case '558300':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadDarkOrchid"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadDarkOrchid"));break;

case '80535837':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadDarkSlateGray"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadDarkSlateGray"));break;

case '15351001':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadGoldenrod"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadGoldenrod"));break;

case '513610013':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadOlive"));
quadLeft.applyObjectStyle(myDocument.objectStyles
```

```
.item("quadOlive"));break;

case '879192':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadDarkSlateBlue"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadDarkSlateBlue"));break;

case '219910012':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadFireBrick"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadFireBrick"));break;

case '0411000':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadOrange"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadOrange"));break;

case '85100118':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadIndigo"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadIndigo"));break;

case '0871000':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadOrangeRed"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadOrangeRed"));break;

case '8631498':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadTeal"));
```

```
        quadLeft.applyObjectStyle(myDocument.objectStyles
        .item("quadTeal"));break;

        case '317610029':
        quadRight.applyObjectStyle(myDocument.objectStyle
        s.item("quadSaddleBrown"));
        quadLeft.applyObjectStyle(myDocument.objectStyles
        .item("quadSaddleBrown"));break;

        case '100981011':
        quadRight.applyObjectStyle(myDocument.objectStyle
        s.item("quadDarkBlue"));
        quadLeft.applyObjectStyle(myDocument.objectStyles
        .item("quadDarkBlue"));break;

        default:
        quadRight.applyObjectStyle(myDocument.objectStyle
        s.item("quadVerde"));
        quadLeft.applyObjectStyle(myDocument.objectStyles
        .item("quadVerde"));

    }
```

Il cuore del programma

Il nucleo centrale del programma è costituito dalla funzione `main`; l'esecuzione di questa funzione innesca l'impaginazione vera e propria, portata a termine grazie all'invocazione delle funzioni appena descritte ed all'integrazione dell'XML effettuata scrivendo *XML rules* specifiche.

Oltre ad un controllo preliminare che serve per accertarsi che il documento sia aperto, il primo frammento analizzato contiene l'inizializzazione della variabile `myRuleSet`, un array costituito dal set di regole XML necessarie per il funzionamento del programma.

Successivamente gli elementi XML sono inseriti in una variabile `elemento`; ciò permette di passare l'elemento radice come primo parametro alla funzione `__processRuleSet` invocata subito dopo.

```
function main(){
     if (app.documents.length != 0){
          var myDocument = app.documents.item(0);
          var myRuleSet = new Array (new insertContenuti, new
          ProcessCollana, new ProcessCopertina, new
          ProcessLegatura, new ProcessFormato, new
          ProcessPagine, new ProcessIllustrazioni, new
          ProcessAnno, new ProcessIsbn, new ProcessPrezzo);
          with(myDocument){
          var elements = xmlElements;
          __processRuleSet(elements.item(0), myRuleSet);
          }
     }
     else{
          alert("No open document");
     }
…
```

Segue la costruzione delle regole XML che costituiscono il set appena inizializzato. Le prime sono finalizzate a processare il contenuto degli elementi figli dell'elemento libro. Ogni regola si presenta con la struttura standard analizzata in precedenza:

- la definizione di un nome;
- un comando XPath usato per localizzare l'elemento a cui applicare la regola
- una funzione in cui è definito quale modifica apportare

Per queste prime regole create tale modifica si attua sostanzialmente in due operazioni:

- l'inserimento della corretta etichetta statica prima del contenuto dell'elemento esaminato (ad esempio *anno*: 2008 dove *anno* è il testo statico inserito e *2008* è l'effettivo contenuto testuale di un dato elemento `anno`);
- l'inserimento di un ritorno a capo dopo ciascun elemento. Questa seconda istruzione, finalizzata ad ottenere tutte le informazioni in una lista verticale nelle pagine del catalogo, equivale ad una operazione che, secondo una modalità tradizionale, andrebbe fatta manualmente modificando il testo nello *story editor* o nella *layout view* stessa.

```
function ProcessCollana(){
        this.name = "ProcessCollana";
        this.xpath = "/catalogo/categoria/libro/collana";
        this.apply = function(myElement, myRuleProcessor){
            with(myElement){
            insertTextAsContent("Collana: ",
            XMLElementPosition.beforeElement);
            insertTextAsContent("\r",
            XMLElementPosition.afterElement);
            }
            return true;
        }
    }


    function ProcessCopertina(){
        this.name = "ProcessCopertina";
        this.xpath = "/catalogo/categoria/libro/copertina";
        this.apply = function(myElement, myRuleProcessor){
            var cop = myElement.contents;
            myElement.insertTextAsContent("\r",
```

```
            XMLElementPosition.afterElement);
            return true;
        }
    }


    function ProcessLegatura(){
        this.name = "ProcessLegatura";
        this.xpath = "/catalogo/categoria/libro/aspetto";
        this.apply = function(myElement, myRuleProcessor){
            with(myElement){
            var etichettaLegatura =
            insertTextAsContent("legatura",
            XMLElementPosition.beforeElement);
            etichettaLegatura.appliedCharacterStyle =
            myDocument.characterStyles.item("caratteristica")
            ;
            insertTextAsContent("\r",
            XMLElementPosition.afterElement);
            }
            return true;
        }
    }


    function ProcessFormato(){
        this.name = "ProcessFormato";
        this.xpath = "/catalogo/categoria/libro/formato";
        this.apply = function(myElement, myRuleProcessor){
            with(myElement){
            var etichettaFormato =
            insertTextAsContent("formato",
            XMLElementPosition.beforeElement);
            etichettaFormato.appliedCharacterStyle =
            myDocument.characterStyles.item("caratteristica")
```

```
        ;
        insertTextAsContent("\r",
        XMLElementPosition.afterElement);
        }
        return true;
    }
}


function ProcessPagine(){
    this.name = "ProcessPagine";
    this.xpath = "/catalogo/categoria/libro/pagine";
    this.apply = function(myElement, myRuleProcessor){
        with(myElement){
        var etichettaPagine =
        insertTextAsContent("pagine",
        XMLElementPosition.beforeElement);
        etichettaPagine.appliedCharacterStyle =
        myDocument.characterStyles.item("caratteristica")
        ;
        insertTextAsContent("\r",
        XMLElementPosition.afterElement);
        }
        return true;
    }
}


function ProcessIllustrazioni(){
    this.name = "ProcessIllustrazioni";
    this.xpath =
    "/catalogo/categoria/libro/illustrazioni";
    this.apply = function(myElement, myRuleProcessor){
        with(myElement){
        var etichettaIllustrazioni =
```

```
            insertTextAsContent("illustrazioni",
            XMLElementPosition.beforeElement);
            etichettaIllustrazioni.appliedCharacterStyle
            =myDocument.characterStyles.item("caratteristica"
            );
            insertTextAsContent("\r",
            XMLElementPosition.afterElement);
            }
            return true;
        }
}


function ProcessAnno(){
        this.name = "ProcessAnno";
        this.xpath = "/catalogo/categoria/libro/anno";
        this.apply = function(myElement, myRuleProcessor){
            with(myElement){
            var etichettaAnno = insertTextAsContent("anno",
            XMLElementPosition.beforeElement);
            etichettaAnno.appliedCharacterStyle =
            myDocument.characterStyles.item("caratteristica")
            ;
            insertTextAsContent("\r",
            XMLElementPosition.afterElement);
            }
            return true;
        }
}


function ProcessIsbn(){
        this.name = "ProcessIsbn";
        this.xpath = "/catalogo/categoria/libro/isbn";
        this.apply = function(myElement, myRuleProcessor){
```

```
            with(myElement){
            var etichettaIsbn = insertTextAsContent("isbn",
            XMLElementPosition.beforeElement);
            etichettaIsbn.appliedCharacterStyle =
            myDocument.characterStyles.item("caratteristica")
            ;
            insertTextAsContent("\r",
            XMLElementPosition.afterElement);
            }
            return true;
        }
    }


    function ProcessPrezzo(){
        this.name = "ProcessPrezzo";
        this.xpath = "/catalogo/categoria/libro/prezzo";
        this.apply = function(myElement, myRuleProcessor){
            with(myElement){
            insertTextAsContent("\u20ac",
            XMLElementPosition.beforeElement);
            insertTextAsContent("\r",
            XMLElementPosition.afterElement);
            }
            return true;
        }
    }
```

Si nota che, nel caso dell'elemento prezzo, la regola varia leggermente perché diverse sono le modalità di presentazione dell'informazione sul prezzo dei libri previste nel *template*. La funzione `apply` della regola `ProcessPrezzo` è utilizzata per inserire il simbolo dell'euro (€) in posizione antecedente al valore dell'elemento processato in un dato momento.

La regola XML più articolata è costruita all'interno della funzione `InsertContenuti`.

Alla specifica del nome segue quella del percorso XPath: il nodo di partenza per l'applicazione della regola è l'elemento `<catalogo>`; il metodo `count()` è usato per ottenere il numero di elementi categoria attualmente presenti nella struttura XML. Il conteggio è necessario all'impostazione dei cicli `for` annidati che seguono; l'iterazione principale serve in prima istanza ad aggiungere (invocando la funzione `aggiungiIntroCategoria`) una pagina introduttiva ogni volta che si incontra una nuova categoria. Le informazioni contenute nel file XML (in particolare il nome della categoria stessa) sono analizzate per generare il contenuto della pagina e per applicare gli stili di formattazione.

Concluso l'inserimento della pagina introduttiva si passa a quello delle pagine informative sui libri presentati; a questo scopo è invocata la funzione `aggiungiConteCategoria`. Per aggiungere il numero necessario di *frame* destinati ad accogliere i contenuti relativi a ciascun libro, gli elementi figli di ogni elemento `<categoria>` sono visitati utilizzando un `for` annidato: il blocco di istruzioni di questa seconda iterazione serve a popolare opportunamente le pagine del catalogo. La verifica di una serie di condizioni permette di gestire in maniera opportuna la collocazione dei contenuti.

```
function insertContenuti(){
    this.name = "insertContenuti";
    this.xpath = "/catalogo";
    this.apply = function (myElement, myRuleProcessor){
        var categorie = myElement.xmlElements.count();
        for (i=0; i<categorie; i++){
            var nuovaSezione = aggiungiIntroCategoria();
            var testo =
            nuovaSezione.pages.item(0).textFrames.add({geomet
```

```
ricBounds:["89mm", "0mm", "81mm", "420mm"]});
var nomeCat =
myElement.xmlElements.item(i).xmlAttributes.item(
0).value;
var coloreCat =
myElement.xmlElements.item(i).xmlAttributes.item(
1).value;
testo.contents = nomeCat;
testo.lines.item(0).applyParagraphStyle
(myDocument.paragraphStyles.item("nomeCat"));
var myCat = myElement.xmlElements.item(i);
var libri = myCat.xmlElements.count();
var nuovoFoglioCat =
aggiungiConteCategoria(coloreCat);
var posLibro = 0;
    for (j=0; j<libri; j++){
        var conteLibro =
        myCat.xmlElements.item(j);
        switch (posLibro){
            case 0: conteLibro.placeIntoFrame
            (nuovoFoglioCat.pages.item(0),
            ["8mm", "8mm", "140mm", "101mm"]);
            break;
            case 1: conteLibro.placeIntoFrame
            (nuovoFoglioCat.pages.item(0),
            ["8mm", "109mm", "140mm", "202mm"]);
            break;
            case 2: conteLibro.placeIntoFrame
            (nuovoFoglioCat.pages.item(1),
            ["8mm", "8mm", "140mm", "101mm"]);
            break;
            case 3: conteLibro.placeIntoFrame
            (nuovoFoglioCat.pages.item(1),
            ["8mm", "109mm", "140mm", "202mm"]);
            break;
```

```
                    }

                    if(posLibro<3) {posLibro++}
                    else if (posLibro==3){
                    if(myCat.xmlElements.item(j+1)!=null){
                        nuovoFoglioCat =
                        aggiungiConteCategoria(coloreCat);
                        posLibro = 0;
                    }
                    else posLibro = 0;
                    }
                    else {
                        nuovoFoglioCat =
                        aggiungiConteCategoria(coloreCat);
                        posLibro = 0;
                    }
                }
            }
    return true;
    }
}
```

Uno stile per le copertine

L'ultimo frammento dello *script* è rappresentato dalla funzione `stile_oggetto` usata per applicare a tutte le immagini delle copertine dei libri le impostazioni di formattazione adeguate.

```
stile_oggetto();
function stile_oggetto(){
    var myDoc = app.activeDocument;
    for(var i=0; i<myDoc.allGraphics.length; i++){
        var myGraphic = myDoc.allGraphics[i];
```

```
    myGraphic.parent.applyObjectStyle(myDoc.objectStyles.item
("copertina"));
    }
}
```

3.6.2 – Uso dello script

Lo *script* appena descritto nel dettaglio può essere mandato in esecuzione non solo per creare l'impaginato completo a partire dalla struttura del *template*, ma anche ogni volta che si renda necessario un aggiornamento del catalogo cartaceo in seguito ad una modifica apportata ai contenuti del database di origine.

Una volta effettuato il lancio dello *script* si può osservare il documento che si "costruisce da solo" in pochi secondi.

Al termine dell'esecuzione il documento non viene esportato automaticamente in formato PDF; Questa scelta è motivata dalla necessità di lasciare la possibilità di effettuare controlli finali sul documento, di apportare eventuali modifiche mirate a elementi specifici dell'impaginato, e di scegliere, di volta in volta, differenti settaggi in fase di impostazione delle caratteristiche del documento PDF destinato alla stampa.

In figura 28 è illustrato il risultato finale che si ottiene al termine dell'esecuzione del programma appena descritto. A *template* differenti possono essere associati *script* specifici e questo permette di generare le più svariate forme di catalogo stampato a partire dal medesimo archivio e quindi dalla medesima struttura XML.

Figura 28 – Il catalogo cartaceo

Conclusioni

Tutte le grandi aziende che hanno necessità di produrre costantemente cataloghi aggiornati sfruttano le possibilità di automatizzazione del flusso di lavoro messe a disposizione da alcuni software; come abbiamo visto, però, questi programmi sono caratterizzati da costi molto elevati e non alla portata di tutti.

Lo strumento realizzato ai fini di questa tesi non si pone l'obiettivo di sostituirsi completamente a tali software molto complessi e articolati, ma punta ad essere, soprattutto per piccole realtà editoriali, una valida alternativa ad essi per quanto riguarda lo specifico aspetto di automatizzazione della creazione di impaginati a partire dai contenuti di un database.

Creato per una casa editrice, può essere adattato e sfruttato in maniera più generale da altri tipi di azienda che abbiano necessità di avere un catalogo cartaceo in costante aggiornamento senza che ciò comporti oneri eccessivi in termini economici e operativi.

Questo meccanismo, infatti, implica una serie di vantaggi per chi lo usa, fra i quali:

- risparmio di tempo rispetto a metodologie tradizionali (copia-incolla);
- meno errori nell'impaginato finale e quindi meno cicli di revisione;
- possibilità di impaginare i dati con *layout* differenti;
- tempi minimi di aggiornamento dell'impaginato; possibilità di gestire aggiornamenti anche giornalieri;
- possibilità di apportare modifiche manuali mirate prima dell'esportazione in PDF

Benefici considerevoli possono inoltre scaturire dalla combinazione di diverse componenti: ad esempio avvalersi delle tecnologie avanzate di stampa digitale anziché di quella tipografica rende ancora più vantaggioso un approccio di

questo tipo alla realizzazione di impaginati professionali strutturati sulla base di un archivio *on line*.

In questo modo anche la fase conclusiva di tutto il flusso di lavoro è impostata in un'ottica di continuo aggiornamento che vada di pari passo con piccole correzioni e aggiunte applicate al database di partenza, fonte dei dati e delle informazioni comunicate su tutti i supporti, compreso quello della carta stampata.

In conclusione è opportuno sottolineare due aspetti importanti.

In primo luogo la possibilità di creare più cataloghi specifici lanciando *script* all'interno di *template* diversi a seconda dei casi: nel contesto di riferimento dell'impaginazione del catalogo per una casa editrice può essere ad esempio utile poter pubblicare versioni differenti attingendo a sottoinsiemi di informazioni; l'approccio studiato è abbastanza flessibile da permettere la creazione di cataloghi relativi ad un'unica categoria di testi (per esempio i libri per ragazzi). A ciascun impaginato può essere data un'impostazione grafica differente a tema con l'ambito di interesse; in definitiva a partire da una fonte di contenuti che mantiene sempre la medesima struttura, possono essere modellati e creati tanti diversi documenti destinati alla stampa che variano nelle scelte estetiche e nei contenuti.

È importante, inoltre, ricordare quanto può essere vantaggioso l'uso di questo sistema anche per la creazione di documenti più semplici come biglietti da visita e diplomi che presentano contenuti sempre riconducibili ad una gerarchia ben definita e strutturabile in un archivio.

Appendice

Listato completo del codice:

```
function ruleProcessorObject(ruleSet, ruleProcessor) {
    this.ruleSet = ruleSet;
    this.ruleProcessor = ruleProcessor;
}
function __makeRuleProcessor(ruleSet, prefixMappingTable){
     var pathArray = new Array();
     for (i=0; i<ruleSet.length; i++)
     {
          pathArray.push(ruleSet[i].xpath);
     }
     case no rules are processed
     try{
          var ruleProcessor =
          app.xmlRuleProcessors.add(pathArray,
          prefixMappingTable);
     }
     catch(e){
          throw e;
     }
     var rProcessor =  new ruleProcessorObject(ruleSet,
     ruleProcessor);
     return rProcessor;
}
function __deleteRuleProcessor(rProcessor) {
          rProcessor.ruleProcessor.remove();
          delete rProcessor.ruleProcessor;
          delete rProcessor.ruleSet;

          delete   rProcessor;
}
function __processRuleSet (root, ruleSet, prefixMappingTable)
     {
     var mainRProcessor = __makeRuleProcessor(ruleSet,
     prefixMappingTable);
          try {
               __processTree(root, mainRProcessor);
               __deleteRuleProcessor(mainRProcessor);
          } catch (e) {
               __deleteRuleProcessor(mainRProcessor);
               throw e;
          }
}
```

```
function __processTree (root, rProcessor)
     {
     var ruleProcessor = rProcessor.ruleProcessor;
     try
     {
          var matchData =
          ruleProcessor.startProcessingRuleSet(root);
          __processMatchData(matchData, rProcessor);

          ruleProcessor.endProcessingRuleSet();
     }
     catch (e)
     {
          ruleProcessor.endProcessingRuleSet();
          throw e;
     }
 }
function __processChildren(rProcessor){
     var ruleProcessor = rProcessor.ruleProcessor;
     try
     {
          var matchData =
          ruleProcessor.startProcessingSubtree();
          __processMatchData(matchData, rProcessor);
     }
     catch (e)
     {
         ruleProcessor.halt();
         throw e;
     }
}
function __processMatchData(matchData, rProcessor){
     var ruleProcessor = rProcessor.ruleProcessor;
     var ruleSet = rProcessor.ruleSet;
     while (matchData != undefined)
     {
          var element = matchData.element;
          var matchRules = matchData.matchRules;
          var applyMatchedRules = true;

          for (var i=0; i<matchRules.length &&
          applyMatchedRules && !ruleProcessor.halted; i++)
          {
          applyMatchedRules = (false ==
          ruleSet[matchRules[i]].apply(element, rProcessor));
          }
          matchData = ruleProcessor.findNextMatch();
     }
}

function __skipChildren(rProcessor){
     rProcessor.ruleProcessor.skipChildren();
```

```
}

var myDocument = app.documents.item(0);

myDocument.viewPreferences.horizontalMeasurementUnits
=MeasurementUnits.millimeters;
myDocument.viewPreferences.verticalMeasurementUnits =
MeasurementUnits.millimeters;
myDocument.viewPreferences.rulerOrigin =
RulerOrigin.pageOrigin;

var myXMLViewPreferences = myDocument.xmlViewPreferences;

myXMLViewPreferences.showAttributes = true;
myXMLViewPreferences.showStructure = true;
myXMLViewPreferences.showTaggedFrames = true;
myXMLViewPreferences.showTagMarkers = true;
myXMLViewPreferences.showTextSnippets = true;

var myXMLImportPreferences = myDocument.xmlImportPreferences;

myXMLImportPreferences.allowTransform = false;

myXMLImportPreferences.createLinkToXML = false;

myXMLImportPreferences.ignoreUnmatchedIncoming = false;

myXMLImportPreferences.ignoreWhitespace = false;

myXMLImportPreferences.importCALSTables = false;

myXMLImportPreferences.importStyle =
XMLImportStyles.mergeImport;

myXMLImportPreferences.importTextIntoTables = false;

myXMLImportPreferences.importToSelected = false;

myXMLImportPreferences.removeUnmatchedExisting = false;

myXMLImportPreferences.repeatTextElements = false;


myDocument.importXML(File("/percorso/catalogo.xml"));

myDocument.xmlImportMaps.add(myDocument.xmlTags.item("collana"
), myDocument.paragraphStyles.item("collana"));

myDocument.xmlImportMaps.add(myDocument.xmlTags.item("aspetto"
), myDocument.paragraphStyles.item("aspetto"));

myDocument.xmlImportMaps.add(myDocument.xmlTags.item("formato"
), myDocument.paragraphStyles.item("formato"));

myDocument.xmlImportMaps.add(myDocument.xmlTags.item("lingua")
, myDocument.paragraphStyles.item("lingua"));
```

```
myDocument.xmlImportMaps.add(myDocument.xmlTags.item("anno"),
myDocument.paragraphStyles.item("anno"));

myDocument.xmlImportMaps.add(myDocument.xmlTags.item("isbn"),
myDocument.paragraphStyles.item("isbn"));

myDocument.xmlImportMaps.add(myDocument.xmlTags.item("prezzo")
, myDocument.paragraphStyles.item("prezzo"));

myDocument.xmlImportMaps.add(myDocument.xmlTags.item("illustra
zioni"), myDocument.paragraphStyles.item("illustrazioni"));

myDocument.xmlImportMaps.add(myDocument.xmlTags.item("pagine")
, myDocument.paragraphStyles.item("pagine"));

myDocument.xmlImportMaps.add(myDocument.xmlTags.item("descrizi
one"), myDocument.paragraphStyles.item("descrizione"));

myDocument.xmlImportMaps.add(myDocument.xmlTags.item("autore")
, myDocument.paragraphStyles.item("autore"));

myDocument.xmlImportMaps.add(myDocument.xmlTags.item("titolo")
, myDocument.paragraphStyles.item("titolo"));

myDocument.mapXMLTagsToStyles();

function aggiungiIntroCategoria(){
    var myDocument = app.documents.item(0);
    with(myDocument){
        var nuovaIntroCat = spreads.add();
        nuovaIntroCat.appliedMaster =
        app.activeDocument.masterSpreads.item("B-Master");

    }
    return nuovaIntroCat;
}

function aggiungiConteCategoria(coloreCat){

    var myDocument = app.documents.item(0);
    with(myDocument){
        var nuovoFoglioCat = spreads.add();
        nuovoFoglioCat.appliedMaster =
        app.activeDocument.masterSpreads.item("A-Master");
        var quadLeft =
        nuovoFoglioCat.pages.item(0).rectangles.add({geometri
        cBounds:["-2mm","-2mm", "6mm", "6mm"]});
        var quadRight =
        nuovoFoglioCat.pages.item(1).rectangles.add({geometri
        cBounds:["-2mm","204mm", "6mm", "212mm"]});
        switch (coloreCat){
            case '887700':
```

```
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadBlue"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadBlue"));break;

case '24969118':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadBrown"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadBrown"));break;

case '14691002':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadChocolate"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadChocolate"));break;

case '7100781':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadCrimson"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadCrimson"));break;

case '8426464':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadDarkCyan"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadDarkCyan"));break;

case '883510029':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadDarkGreen"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadDarkGreen"));break;

case '5610061':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadDarkMagenta"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadDarkMagenta"));break;

case '663810026':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadDarkOliveGreen"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadDarkOliveGreen"));break;

case '558300':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadDarkOrchid"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadDarkOrchid"));break;
```

```
case '80535837':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadDarkSlateGray"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadDarkSlateGray"));break;

case '15351001':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadGoldenrod"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadGoldenrod"));break;

case '513610013':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadOlive"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadOlive"));break;

case '879192':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadDarkSlateBlue"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadDarkSlateBlue"));break;

case '219910012':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadFireBrick"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadFireBrick"));break;

case '0411000':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadOrange"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadOrange"));break;

case '85100118':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadIndigo"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadIndigo"));break;

case '0871000':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadOrangeRed"));
quadLeft.applyObjectStyle(myDocument.objectStyles
.item("quadOrangeRed"));break;

case '8631498':
quadRight.applyObjectStyle(myDocument.objectStyle
s.item("quadTeal"));
```

```
            quadLeft.applyObjectStyle(myDocument.objectStyles
            .item("quadTeal"));break;

            case '317610029':
            quadRight.applyObjectStyle(myDocument.objectStyle
            s.item("quadSaddleBrown"));
            quadLeft.applyObjectStyle(myDocument.objectStyles
            .item("quadSaddleBrown"));break;

            case '100981011':
            quadRight.applyObjectStyle(myDocument.objectStyle
            s.item("quadDarkBlue"));
            quadLeft.applyObjectStyle(myDocument.objectStyles
            .item("quadDarkBlue"));break;

            default:
            quadRight.applyObjectStyle(myDocument.objectStyle
            s.item("quadVerde"));
            quadLeft.applyObjectStyle(myDocument.objectStyles
            .item("quadVerde"));

        }

function main(){
    if (app.documents.length != 0){
        var myDocument = app.documents.item(0);
        var myRuleSet = new Array (new insertContenuti, new
        ProcessCollana, new ProcessCopertina, new
        ProcessLegatura, new ProcessFormato, new
        ProcessPagine, new ProcessIllustrazioni, new
        ProcessAnno, new ProcessIsbn, new ProcessPrezzo);
        with(myDocument){
        var elements = xmlElements;
        __processRuleSet(elements.item(0), myRuleSet);
        }
    }
    else{
        alert("No open document");
    }
function ProcessCollana(){
        this.name = "ProcessCollana";
        this.xpath = "/catalogo/categoria/libro/collana";
        this.apply = function(myElement, myRuleProcessor){
            with(myElement){
            insertTextAsContent("Collana: ",
            XMLElementPosition.beforeElement);
            insertTextAsContent("\r",
            XMLElementPosition.afterElement);
            }
            return true;
        }
    }
```

```
function ProcessCopertina(){
    this.name = "ProcessCopertina";
    this.xpath = "/catalogo/categoria/libro/copertina";
    this.apply = function(myElement, myRuleProcessor){
        var cop = myElement.contents;
        myElement.insertTextAsContent("\r",
        XMLElementPosition.afterElement);
        return true;
    }
}
function ProcessLegatura(){
    this.name = "ProcessLegatura";
    this.xpath = "/catalogo/categoria/libro/aspetto";
    this.apply = function(myElement, myRuleProcessor){
        with(myElement){
        var etichettaLegatura =
        insertTextAsContent("legatura",
        XMLElementPosition.beforeElement);
        etichettaLegatura.appliedCharacterStyle =
        myDocument.characterStyles.item("caratteristica")
        ;
        insertTextAsContent("\r",
        XMLElementPosition.afterElement);
        }
        return true;
    }
}
function ProcessFormato(){
    this.name = "ProcessFormato";
    this.xpath = "/catalogo/categoria/libro/formato";
    this.apply = function(myElement, myRuleProcessor){
        with(myElement){
        var etichettaFormato =
        insertTextAsContent("formato",
        XMLElementPosition.beforeElement);
        etichettaFormato.appliedCharacterStyle =
        myDocument.characterStyles.item("caratteristica")
        ;
        insertTextAsContent("\r",
        XMLElementPosition.afterElement);
        }
        return true;
    }
}
function ProcessPagine(){
    this.name = "ProcessPagine";
    this.xpath = "/catalogo/categoria/libro/pagine";
    this.apply = function(myElement, myRuleProcessor){
        with(myElement){
        var etichettaPagine =
        insertTextAsContent("pagine",
        XMLElementPosition.beforeElement);
```

```
            etichettaPagine.appliedCharacterStyle =
            myDocument.characterStyles.item("caratteristica")
            ;
            insertTextAsContent("\r",
            XMLElementPosition.afterElement);
            }
            return true;
        }
    }


    function ProcessIllustrazioni(){
        this.name = "ProcessIllustrazioni";
        this.xpath =
        "/catalogo/categoria/libro/illustrazioni";
        this.apply = function(myElement, myRuleProcessor){
            with(myElement){
            var etichettaIllustrazioni =
            insertTextAsContent("illustrazioni",
            XMLElementPosition.beforeElement);
            etichettaIllustrazioni.appliedCharacterStyle
            =myDocument.characterStyles.item("caratteristica"
            );
            insertTextAsContent("\r",
            XMLElementPosition.afterElement);
            }
            return true;
        }
    }
    function ProcessAnno(){
        this.name = "ProcessAnno";
        this.xpath = "/catalogo/categoria/libro/anno";
        this.apply = function(myElement, myRuleProcessor){
            with(myElement){
            var etichettaAnno = insertTextAsContent("anno",
            XMLElementPosition.beforeElement);
            etichettaAnno.appliedCharacterStyle =
            myDocument.characterStyles.item("caratteristica")
            ;
            insertTextAsContent("\r",
            XMLElementPosition.afterElement);
            }
            return true;
        }
    }
    function ProcessIsbn(){
        this.name = "ProcessIsbn";
        this.xpath = "/catalogo/categoria/libro/isbn";
        this.apply = function(myElement, myRuleProcessor){
            with(myElement){
            var etichettaIsbn = insertTextAsContent("isbn",
            XMLElementPosition.beforeElement);
```

```
                etichettaIsbn.appliedCharacterStyle =
                myDocument.characterStyles.item("caratteristica")
                ;
                insertTextAsContent("\r",
                XMLElementPosition.afterElement);
                }
                return true;
            }
        }
        function ProcessPrezzo(){
            this.name = "ProcessPrezzo";
            this.xpath = "/catalogo/categoria/libro/prezzo";
            this.apply = function(myElement, myRuleProcessor){
                with(myElement){
                insertTextAsContent("\u20ac",
                XMLElementPosition.beforeElement);
                insertTextAsContent("\r",
                XMLElementPosition.afterElement);
                }
                return true;
            }
        }
    function insertContenuti(){
        this.name = "insertContenuti";
        this.xpath = "/catalogo";
        this.apply = function (myElement, myRuleProcessor){
            var categorie = myElement.xmlElements.count();
            for (i=0; i<categorie; i++){
                var nuovaSezione = aggiungiIntroCategoria();
                var testo =
                nuovaSezione.pages.item(0).textFrames.add({geomet
                ricBounds:["89mm", "0mm", "81mm", "420mm"]});
                var nomeCat =
                myElement.xmlElements.item(i).xmlAttributes.item(
                0).value;
                var coloreCat =
                myElement.xmlElements.item(i).xmlAttributes.item(
                1).value;
                testo.contents = nomeCat;
                testo.lines.item(0).applyParagraphStyle
                (myDocument.paragraphStyles.item("nomeCat"));
                var myCat = myElement.xmlElements.item(i);
                var libri = myCat.xmlElements.count();
                var nuovoFoglioCat =
                aggiungiConteCategoria(coloreCat);
                var posLibro = 0;
                    for (j=0; j<libri; j++){
                        var conteLibro =
                        myCat.xmlElements.item(j);
                        switch (posLibro){
                            case 0: conteLibro.placeIntoFrame
                            (nuovoFoglioCat.pages.item(0),
```

```
                        ["8mm", "8mm", "140mm", "101mm"]);
                        break;
                        case 1: conteLibro.placeIntoFrame
                        (nuovoFoglioCat.pages.item(0),
                        ["8mm", "109mm", "140mm", "202mm"]);
                        break;
                        case 2: conteLibro.placeIntoFrame
                        (nuovoFoglioCat.pages.item(1),
                        ["8mm", "8mm", "140mm", "101mm"]);
                        break;
                        case 3: conteLibro.placeIntoFrame
                        (nuovoFoglioCat.pages.item(1),
                        ["8mm", "109mm", "140mm", "202mm"]);
                        break;
                    }
                    if(posLibro<3) {posLibro++}
                    else if (posLibro==3){
                    if(myCat.xmlElements.item(j+1)!=null){
                        nuovoFoglioCat =
                        aggiungiConteCategoria(coloreCat);
                        posLibro = 0;
                    }
                    else posLibro = 0;
                    }
                    else {
                        nuovoFoglioCat =
                        aggiungiConteCategoria(coloreCat);
                        posLibro = 0;
                    }
                }
            }
    return true;
    }
}

stile_oggetto();
function stile_oggetto(){
    var myDoc = app.activeDocument;
    for(var i=0; i<myDoc.allGraphics.length; i++){
        var myGraphic = myDoc.allGraphics[i];
    myGraphic.parent.applyObjectStyle(myDoc.objectStyles.item
("copertina"));
    }
}
```

Bibliografia e Sitografia

Documenti elettronici, testi e manuali

ADOBE. (2008), *Adobe InDesign CS3, Scripting guide: JavaScript*, Adobe Press, (http://wwwimages.adobe.com/www.adobe.com/products/indesign/scripting/pdfs/InDesignCS3_ScriptingGuide_JS.pdf).

CANDUCCI M. (2004), *PHP 5*, Apogeo, Milano.

KAHREL P. (2006), *Scripting InDesign with JavaScript*, O'Really Media.

MAIVALD J. e PALMER C. (2008), *A designer's guide to Adobe InDesign and XML*, Adobe Press.

PIERAZZO E. (2005), *La codifica dei testi. Un'introduzione*, Carocci, Roma.

Adobe InDesign CS3 and XML: a Technical Reference, (http://wwwimages.adobe.com/www.adobe.com/products/indesign/scripting/pdfs/indesign_and_xml_technical_reference.pdf).

Siti internet

InDesign scripting reference: Javascript:
http://www.indesignscriptingreference.com/CS3/JavaScript/.

Wikipedia, the free enciclopedia:
http://en.wikipedia.org/wiki/Main_Page.

inEditoria.net, un punto d'incontro per gli utenti italiani di Adobe InDesign:
http://www.ineditoria.net/.

MacTech, a journal of macintosh technology:
http://www.mactech.com.

InDesign Central:
http://www.indesigncentral.com.

InDesign Magazine:
http://www.indesignmag.com/default.asp.

Felici Editore – Casa editrice, Pisa:
http://www.felicieditore.it.

X-ray Magazine:
http://www.xraymag.com/.

Moo, we love to print:
http://us.moo.com/it/.