

# Capitolo 4

## Libreria SAXParserLibrary

---

L'esigenza di processare un documento xml da Prolog ha fatto nascere l'opportunità di creare una libreria (prototipo) per il trasferimento di informazioni da un qualunque documento xml a tuProlog.

### **4.1    *Analisi del problema***

Un documento xml può essere immaginato come un albero dove ogni elemento rappresenta un nodo. Un documento xml può essere visto da un elaboratore come uno stream di caratteri, questo però non evidenzia le caratteristiche del documento che possiede una chiara struttura ad albero. I parser DOM consentono di trasformare un documento xml in un albero DOM, e attraverso i diversi metodi di esplorare l'intero albero.

Il parser DOM è stato utilizzato nella realizzazione del convertitore grammatica-ASF. In questo caso si vuole utilizzare il parser SAX che sfrutta questa lettura a stream ed individua l'apertura e la chiusura dei tag. Rappresentando questo aspetto su un albero astratto si può immaginare l'apertura di un tag come l'entrata in un nodo (o elemento) e la chiusura come l'uscita da un nodo. Da queste considerazioni si evince che il parser

SAX esegue una esplorazione dell'albero a discesa ricorsiva sinistra; partendo dalla radice e scendendo sempre verso il figlio di sinistra.

La libreria che si è realizzata ha il compito di offrire predicati che permettano di trasformare un documento xml in una rappresentazione ad albero in tuProlog utilizzando il parser SAX e quindi sfruttando gli aspetti sopra analizzati di esplorazione dell'albero attraverso la discesa ricorsiva.

## **4.2    *Analisi dell'approccio di conversione XML~PrologTree***

Si è visto che il parser SAX esegue una scansione del documento partendo dal nodo radice e scendendo nei figli ricorsivamente. In Prolog invece l'albero ha la struttura di un predicato che ha come nome la radice e come argomenti i figli. Una delle possibili soluzioni a questo problema di conversione è la seguente:

La lettura del documento xml può essere trasformata nella scrittura di una stringa, il che rende tutto molto semplice perché la rappresentazione di un albero in Prolog è esplicitata attraverso uno stream di caratteri che ne evidenziano un'esplorazione a discesa ricorsiva sinistra (proprio come vengono rappresentati i documenti xml). Tale stringa una volta terminata potrebbe essere "asserita" alla teoria corrente del Prolog e diventare a tutti gli effetti un fatto della teoria.

Questa soluzione però non viene ritenuta del tutto corretta perché non si vogliono generare effetti collaterali. Infatti introducendo un fatto alla teoria si va a rovinare l'eleganza del Prolog. Sarebbe invece più elegante (e più vicino alle caratteristiche di linguaggio dichiarativo) passare l'albero generato attraverso una variabile (unificazione). Questo consente

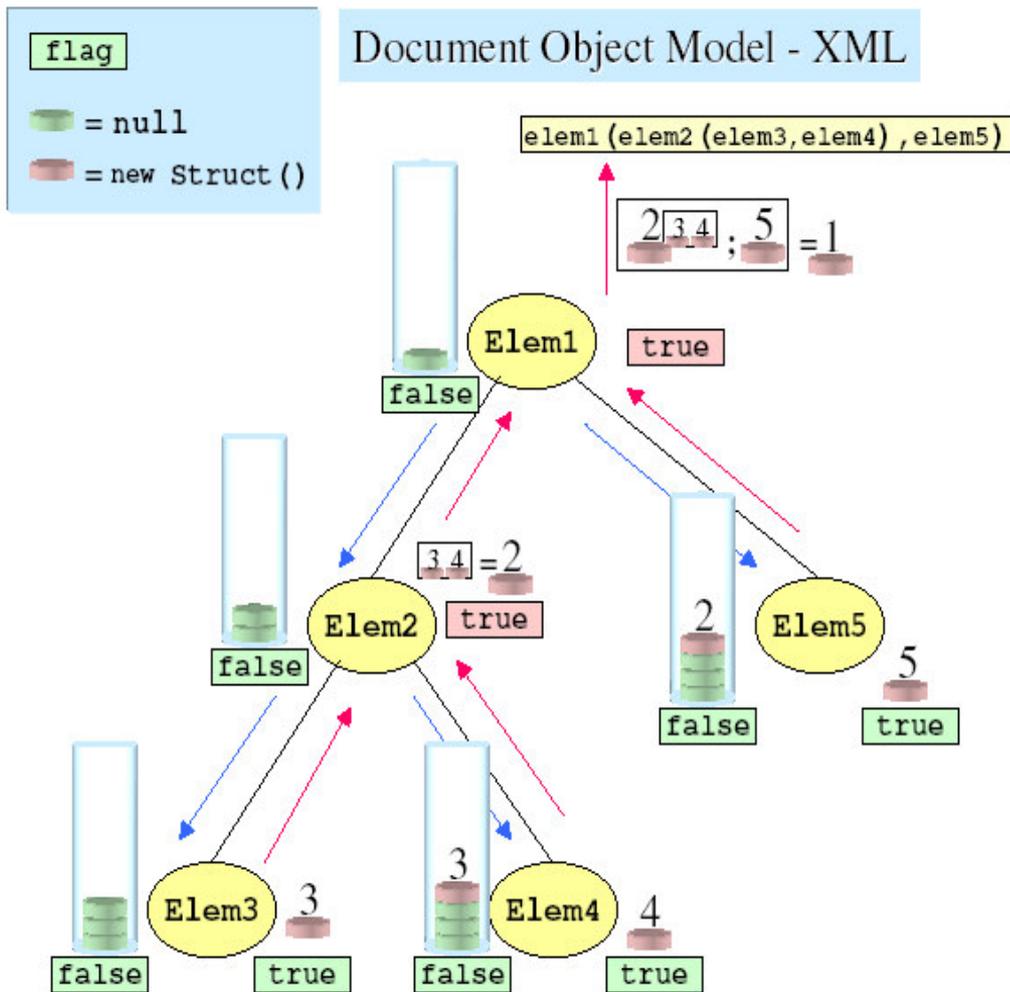
di non sporcare la teoria corrente e di verificare se le informazioni all'interno di un documento xml unificano con altre informazioni considerate in Prolog. Questo da flessibilità e modularità al sistema consentendo di snaturare il meno possibile la natura dichiarativa e logica del Prolog.

Per fare questo si è reso necessario realizzare una libreria che sfrutti le caratteristiche di tuProlog, soprattutto per come i predicati Prolog sono rappresentati in Java. Infatti è possibile istanziare classi Java che rappresentano lo scheletro dei predicati in tuProlog. La struttura dati che si va a formare è assolutamente comprensibile all'engine tuProlog che quindi consente di manipolarla attraverso i normali predicati di Prolog.

### **4.3    *Algoritmo di conversione***

Il problema della conversione è stato risolto attraverso un semplice algoritmo che tiene conto del modo di esplorazione dell'albero. La visibilità sulla struttura del documento è limitata al nodo in cui ci si trova. In altre parole il parser SAX esegue una conversione istantanea dei nodi del documento e non ha memoria sulla struttura o sul tipo di nodi già processati. Per creare una rappresentazione ad albero è invece fondamentale tenere conto dei nodi attraversati durante l'esplorazione e della loro collocazione nell'albero. Si evince che sarà necessario introdurre una memoria che tenga traccia del tipo e collocazione di ogni nodo. Lo stack è la memoria che più si avvicina a questi requisiti perché può mantenere all'interno diversi oggetti e la loro posizione nello stack permette di ricavare la collocazione nell'albero astratto.

La figura sottostante chiarisce molto bene il funzionamento dell'algoritmo di conversione.



**Figura 7 ~ Costruzione di albero Prolog di un DOM-Xml**

L'algoritmo implementato è solo un prototipo che può chiaramente essere migliorato ma il suo funzionamento risulta chiaro solo dopo aver considerato che il parser SAX può essere visto come un processo senza memoria. Ad ogni entrata in un nodo è quindi necessario inserire nello stack un elemento (oggetto). I cilindretti verdi in figura rappresentano elementi nulli. I cilindretti colorati rappresentano elementi "istanziati" che memorizzano le caratteristiche dei nodi già esplorati. Si intuisce che il processo di inserimento ed estrazione dallo stack permette di stabilire la collocazione degli elementi sull'albero. Inoltre tale processo è

riassumibile in una sequenza di inserimenti (push) che ricorda la rappresentazione polacca postfissa. Difatti per creare un nodo radice in Java (costruttore) è necessario conoscere già i nodi figli (argomenti) e questo è quello che viene rappresentato in una polacca postfissa dove i figli si trovano sempre davanti alla radice.

Nell'algoritmo si è reso necessario anche l'introduzione di un flag che consente di stabilire se si sta uscendo da un nodo foglia o da un nodo albero. Questo si può discriminare tenendo traccia delle operazioni eseguite. Anche il flag rappresenta una memoria: è "true" quando si esce da un nodo (salita) e false quando si entra (discesa). Se si esce da un nodo che rappresenta un sottoalbero (es.: Elem2) allora il flag è ancora "true" perché si era usciti poco prima da un altro nodo, questo accade solo quando un nodo è padre di altri nodi. A questo punto è possibile conoscere tutti i figli e si possono prelevare le informazioni relative dallo stack.

Al termine della conversione dell'albero si avrà un oggetto Struct che per l'engine tuProlog rappresenta un albero prolog che riassume il documento xml.