**ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ**

**ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ**

**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ**

# ΕΥΦΥΗΣ ΔΙΑΧΕΙΡΙΣΗ XML ΔΕΔΟΜΕΝΩΝ ΜΕ ΤΗ ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ PROLOG

**Διπλωματική Εργασία**

ΤΟΥ

**Γεώργιου Αλεξιάδη (ΑΕΜ: 400)**



**Επιβλέπων Καθηγητής: Ιωάννης Βλαχάβας**

**Θεσσαλονίκη 2003**

*Στη μνήμη του πατέρα μου,*

*στη μητέρα μου Δέσποινα*

*και στην αδερφή μου Ειρήνη.*

# Πρόλογος

Στα πλαίσια της διπλωματικής αυτής εργασίας παρουσιάζεται η ανάπτυξη ενός συστήματος για την ευφυή διαχείριση XML δεδομένων με τη γλώσσα προγραμματισμού Prolog.

Η XML αποτελεί μια σύγχρονη γλώσσα προγραμματισμού, με την οποία μπορούμε να περιγράψουμε οποιοδήποτε είδος εγγράφου, καθώς διαθέτει ένα εξαιρετικά ευέλικτο συντακτικό. Όσο σημαντική είναι η ίδια η XML, άλλο τόσο σημαντική είναι και η διαχείριση των δεδομένων που είναι γραμμένα στη συγκεκριμένη γλώσσα προγραμματισμού. Ο όρος διαχείριση έχει να κάνει με την εισαγωγή, αποθήκευση και διαγραφή δεδομένων από τα XML έγγραφα.

Στην εργασία αυτή, η διαχείριση XML δεδομένων πραγματοποιείται με τη χρήση ενός αντικειμενοστραφούς μοντέλου βάσης δεδομένων, του X-DEVICE. Όλο το πρόγραμμα είναι υλοποιημένο στη γλώσσα προγραμματισμού Prolog και συγκεκριμένα στην έκδοση Eclipse Prolog. Επίσης, για την αναπαράσταση των δεδομένων με αντικειμενοστραφείς βάσεις δεδομένων χρησιμοποιείται το σύστημα ADAM, το οποίο αποτελεί μια εφαρμογή αντικειμενοστραφούς βάσεως δεδομένων σε Prolog.

Η εκπόνηση της εργασίας έγινε στο Εργαστήριο Γλωσσών Προγραμματισμού και Τεχνολογίας Λογισμικού (Programming Languages and Software Engineering Lab – PlaSE Lab) του τμήματος Πληροφορικής της Σχολής Θετικών Επιστημών του Αριστοτελείου Πανεπιστημίου Θεσσαλονίκης, σε συνεργασία με την ομάδα Λογικού Προγραμματισμού και Ευφυών Συστημάτων (Logic Programming and Intelligent Systems Group - LPIS Group).

Στο σημείο αυτό θα ήθελα να ευχαριστήσω θερμά τον κ. Νικόλαο Βασιλειάδη για την άψογη συνεργασία, συμπαράσταση και παροχή συμβουλών καθ' όλη τη διάρκεια ανάπτυξης της εργασίας. Επίσης, θα ήθελα να ευχαριστήσω φυσικά και τον καθηγητή μου κ. Ιωάννη Βλαχάβα για την ανάθεση της συγκεκριμένης διπλωματικής εργασίας και για το ευχάριστο και δημιουργικό περιβάλλον συνεργασίας. Τέλος, οφείλω να ευχαριστήσω την οικογένειά μου, τους φίλους και συμφοιτητές μου για την ηθική τους συμπαράσταση.

**Νοέμβριος 2003**

# Περιεχόμενα

# 1 ΕΙΣΑΓΩΓΗ

Το αντικείμενο μελέτης της διπλωματικής αυτής εργασίας είναι η *Ευφυής Διαχείριση XML Δεδομένων με τη Γλώσσα Προγραμματισμού Prolog*. Η γλώσσα XML (Extensible Markup Language – Επεκτάσιμη Γλώσσα Σήμανσης) είναι το τρέχον πρότυπο για την ανταλλαγή δομημένων και ημιδομημένων πληροφοριών στο Internet, η οποία δεν αντικαθιστά τη γνωστή και ευρέως διαδεδομένη γλώσσα προγραμματισμού HTML, αλλά χρησιμοποιείται σε συνδυασμό με αυτήν για να αυξήσει τις δυνατότητες των ιστοσελίδων ως προς ποικίλα χαρακτηριστικά αναπαράστασης και μεταφοράς οποιουδήποτε είδους εγγράφου στο Internet. Η XML διαθέτει ένα εξαιρετικά ευέλικτο συντακτικό και γι' αυτό είναι κατάλληλη για την περιγραφή οποιουδήποτε είδους εγγράφου, από μία απλή μουσική παρτιτούρα μέχρι και σύνθετες βάσεις δεδομένων.

Ο όρος διαχείριση δεδομένων περιλαμβάνει διεργασίες όπως η εισαγωγή, διαγραφή, επεξεργασία και αποθήκευση των δεδομένων αυτών. Έχουν αφιερωθεί αρκετά χρόνια στην ανάπτυξη θεωριών και συστημάτων γύρω από τη διαχείριση δεδομένων και υπάρχουν ήδη πολλές προτάσεις σχετικά με μεθοδολογίες διαχείρισης ημιδομημένων δεδομένων, δηλαδή XML δεδομένων. Στη συγκεκριμένη εργασία, η διαχείριση των XML δεδομένων υλοποιείται με αντικειμενοστραφείς βάσεις δεδομένων. Για το σκοπό αυτό χρησιμοποιείται το μοντέλο X-DEVICE [1], το οποίο είναι ένα συμπερασματικό αντικειμενοστραφές μοντέλο βάσης δεδομένων. Η κατασκευή του προγράμματος υλοποίησης του συστήματος διαχείρισης XML δεδομένων, βασίζεται στον αλγόριθμο του X-DEVICE.

Το σύστημα για τη διαχείριση των XML δεδομένων είναι υλοποιημένο στη γλώσσα προγραμματισμού Prolog. Πιο συγκεκριμένα, χρησιμοποιείται η έκδοση ECLiPSe (ECLiPSe Common Logic Programming System) Prolog, διότι είναι αρκετά επεκτάσιμη και διαθέτει δυνατότητες διασύνδεσης στο Internet. Για την αναπαράσταση των XML δεδομένων με μια αντικειμενοστραφή βάση δεδομένων, γίνεται χρήση του συστήματος ADAM [2]. Το ADAM αποτελεί ουσιαστικά μια εφαρμογή αντικειμενοστραφούς βάσης δεδομένων σε Prolog.

Το θεωρητικό υπόβαθρο και η υλοποίηση της εργασίας αυτής περιγράφονται στα ακόλουθα κεφάλαια:

Το **δεύτερο κεφάλαιο** παρουσιάζει την Επεκτάσιμη Γλώσσα Σήμανσης (XML). Πρώτα παρέχεται ένας ορισμός της XML, ενώ στη συνέχεια περιγράφονται διάφοροι περιορισμοί, στους οποίους η λύση δίδεται με την XML. Επίσης, γίνεται σύγκριση μεταξύ των γλωσσών προγραμματισμού SGML, HTML και XML. Το δεύτερο μέρος του κεφαλαίου αυτού παρουσιάζει την ανατομία ενός XML εγγράφου (στοιχεία, ιδιότητες, σχόλια, εντολές επεξεργασίας και οντότητες), τον ορισμό του DTD (Document Type Definition – Ορισμός Τύπου Εγγράφου) που συνοδεύει το XML έγγραφο και τις έννοιες «σωστά διατυπωμένο» και «έγκυρο» έγγραφο. Τέλος, παρατίθεται μια λίστα από εφαρμογές XML για τη βελτίωση εγγράφων και μια λίστα από πρακτικές χρήσεις της γλώσσας XML.

Το **τρίτο κεφάλαιο** ασχολείται με το θέμα της διαχείρισης των XML δεδομένων. Παρουσιάζονται οι δύο βασικές μεθοδολογίες διαχείρισης XML δεδομένων: η πρώτη με σχεσιακές βάσεις δεδομένων και η δεύτερη με αντικειμενοστραφείς βάσεις δεδομένων. Όσο αφορά τη διαχείριση XML δεδομένων με σχεσιακές βάσεις δεδομένων, περιγράφονται και συγκρίνονται 3 βασικές τεχνικές: η τεχνική Βασικής Παρεμβολής (Basic Inlining), η τεχνική Διαμοιραζόμενης Παρεμβολής (Shared Inlining) και η τεχνική Υβριδικής Παρεμβολής (Hybrid Inlining). Όσο αφορά τη διαχείριση XML δεδομένων με αντικειμενοστραφείς βάσεις δεδομένων, γίνεται αναλυτική περιγραφή του αντικειμενοστραφούς μοντέλου αναπαράστασης XML δεδομένων του X-DEVICE.

Στο **τέταρτο κεφάλαιο** γίνεται μια εισαγωγή στις αντικειμενοστραφείς βάσεις δεδομένων (Object Oriented Databases – OODBs) και στα συστήματα OODB. Το βασικό θέμα ενασχόλησης στο συγκεκριμένο κεφάλαιο είναι το OODB σύστημα ADAM. Πιο συγκεκριμένα, γίνεται λόγος για τις έννοιες: μετα-κλάσεις (meta-classes), κλάσεις (classes) και στιγμιότυπα (instances), μέθοδοι (methods), εγκλεισμός (encapsulation) και σχισμές (slots), κληρονομικότητα (inheritance), τύποι (types), πλειάδες (tuples) και μέθοδοι γενικής χρήσης (generic methods). Επίσης, παραθέτονται κάποια παραδείγματα ερωτημάτων (queries) προς το σύστημα ADAM και στο τέλος του κεφαλαίου παρουσιάζεται αναλυτικά η ιεραρχία των μετα-κλάσεων στο σύστημα ADAM.

Στο **πέμπτο κεφάλαιο** αναλύεται η υλοποίηση του συστήματος που αναπτύχθηκε για τη διαχείριση των XML δεδομένων με τη γλώσσα προγραμματισμού Prolog. Αρχικά, παρουσιάζεται η έννοια των Γραμματικών Οριστικών Προτάσεων (Definite Clause Grammars – DCGs), οι οποίες χρησιμοποιούνται στα στάδια συντακτικής ανάλυσης των δεδομένων, και δίδεται ένα παράδειγμα γραμματικής με χρήση DCGs. Στο επόμενο μέρος του κεφαλαίου περιγράφεται λεπτομερώς η αρχιτεκτονική του συστήματος και όλα τα στάδια που ακολουθούνται για την παραγωγή του τελικού αποτελέσματος. Η περιγραφή αυτή πραγματοποιείται με την παράθεση τμημάτων κώδικα από το πρόγραμμα που αναπτύχθηκε και υλοποιεί, σε Prolog, τη διαχείριση XML δεδομένων. Τέλος, παρουσιάζεται το περιβάλλον της Eclipse Prolog και ένα παράδειγμα εκτέλεσης του αναπτυχθέντος συστήματος.

Στο **έκτο κεφάλαιο** παρουσιάζονται τα προβλήματα που προέκυψαν κατά τη διάρκεια της ανάπτυξης του συστήματος διαχείρισης XML δεδομένων, ενώ επιχειρείται και κριτική του προγράμματος, με την εξαγωγή συμπερασμάτων. Επίσης, παρατίθεται μία σειρά από θέματα που δε λύθηκαν και θα μπορούσαν να αποτελέσουν μελλοντικό στόχο άλλων διπλωματικών εργασιών.

Το **Παράρτημα Α** περιλαμβάνει τον κώδικα του προγράμματος που συγγράφηκε για τη διαχείριση XML δεδομένων σε Prolog, ο οποίος είναι διαμοιρασμένος σε τέσσερα αρχεία, καθένα από τα οποία υλοποιεί και ένα από τα τέσσερα στάδια που απαιτούνται για τη διαχείριση των δεδομένων.

Το **Παράρτημα Β** περιέχει τον αλγόριθμο του X-DEVICE, ο οποίος αναπαριστά ένα XML έγγραφο, μαζί με το DTD του, σε μία αντικειμενοστραφή βάση δεδομένων.

Στο **Παράρτημα Γ** παρατίθεται η ιεραρχία κλάσεων μιας πανεπιστημιακής βάσης δεδομένων. Η παραπομπή στο παράρτημα αυτό γίνεται στο τέταρτο κεφάλαιο, όπου τα περισσότερα παραδείγματα αφορούν τη συγκεκριμένη βάση.

Τέλος, στο **Παράρτημα Δ** παρουσιάζονται τα αποτελέσματα από την εκτέλεση του προγράμματος για ένα συγκεκριμένο σύνολο δεδομένων (XML έγγραφο και DTD).

# 2 ΕΠΕΚΤΑΣΙΜΗ ΓΛΩΣΣΑ ΣΗΜΑΝΣΗΣ (EXTENSIBLE MARKUP LANGUAGE - XML)

## 2.1 Ορισμός της XML

Η γλώσσα *XML* πήρε το όνομά της από τα αρχικά των λέξεων *Extensible Markup Language* (Επεκτάσιμη Γλώσσα Σήμανσης) και ορίστηκε από την ομάδα XML Working Group της Κοινοπραξίας Παγκόσμιου Ιστού (World Wide Web Consortium, ή αλλιώς W3C) [3] ως εξής:

«Η XML αποτελεί ένα υποσύνολο της γλώσσας *SGML* (Standard Generalized Markup Language – Πρότυπη Γενικευμένη Γλώσσα Σήμανσης). Σκοπός της είναι να διευκολύνει την αποστολή, τη λήψη και την επεξεργασία της γενικευμένης γλώσσας SGML στον Ιστό, όπως ακριβώς γίνεται τώρα με την *HTML* (Hypertext Markup Language – Γλώσσα Σήμανσης Υπερκειμένου). Η XML έχει σχεδιαστεί με στόχο την ευκολία της υλοποίησης και τη δυνατότητα παράλληλης χρήσης της με τις γλώσσες SGML και HTML» [4].

### 2.1.1 Η Ανάγκη για XML

Η HTML παρέχει ένα σταθερό σύνολο προκαθορισμένων στοιχείων, με τα οποία μπορούμε να συμβολίζουμε τα περιεχόμενα μιας τυπικής ιστοσελίδας γενικού περιεχομένου. Παραδείγματα τέτοιων στοιχείων είναι οι επικεφαλίδες, οι ενότητες, οι λίστες, οι πίνακες, οι εικόνες και οι σύνδεσμοι. Αν και το σύνολο των προκαθορισμένων στοιχείων HTML έχει εμπλουτιστεί σημαντικά από την πρώτη έκδοση της HTML, η γλώσσα αυτή εξακολουθεί να είναι ακατάλληλη για τον προσδιορισμό πολλών ειδών εγγράφων. Τέτοια παραδείγματα εγγράφων που δεν μπορούν να προσδιοριστούν επαρκώς με την HTML είναι τα ακόλουθα:

- *Έγγραφα που δεν αποτελούνται από τυπικά στοιχεία (επικεφαλίδες, ενότητες, λίστες, πίνακες κλπ.)*. Για παράδειγμα, η HTML δεν έχει τα απαραίτητα στοιχεία που χρειάζονται για τη σήμανση μιας μουσικής παρτιτούρας ή ενός συνόλου μαθηματικών εξισώσεων.

- *Βάσεις δεδομένων, όπως για παράδειγμα ένας κατάλογος βιβλίων*. Μπορούμε να χρησιμοποιήσουμε μια σελίδα HTML για να αποθηκεύσουμε και να εμφανίσουμε στατικές πληροφορίες (όπως μια λίστα με περιγραφές βιβλίων). Όμως, αν θέλουμε να ταξινομήσουμε, να επιλέξουμε, να εντοπίσουμε και να δουλέψουμε με τις πληροφορίες μας με κάποιο διαφορετικό τρόπο, για κάθε πληροφορία (πεδίο) θα πρέπει να οριστεί και από μία ετικέτα (αυτό κάνουν και τα προγράμματα βάσεων δεδομένων, όπως η Microsoft Access). Η HTML δε διαθέτει τα απαραίτητα στοιχεία για κάτι τέτοιο.

- *Έγγραφα που θέλουμε να οργανώσουμε ιεραρχικά σε μορφή δένδρου*. Ας υποθέσουμε, για παράδειγμα, ότι γράφουμε ένα βιβλίο και θέλουμε να το χωρίσουμε σε μέρη, κεφάλαια, ενότητες επιπέδου Α, ενότητες επιπέδου Β, ενότητες επιπέδου Γ, παραγράφους και εικόνες. Σε αυτή την περίπτωση, ένα πρόγραμμα θα μπορούσε να χρησιμοποιήσει αυτό το δομημένο έγγραφο για να δημιουργήσει έναν πίνακα περιεχομένων, να εμφανίσει τη διάρθρωσή του με διάφορα επίπεδα λεπτομερειών, να εξάγει κάποιες συγκεκριμένες ενότητες και να διαχειριστεί με άλλους τρόπους τις πληροφορίες. Το στοιχείο επικεφαλίδας της HTML όμως επισημαίνει μόνο το κείμενο της επικεφαλίδας. Το πραγματικό κείμενο και τα στοιχεία που ανήκουν σε μια ενότητα του εγγράφου δεν περιέχονται μέσα στο στοιχείο της αντίστοιχης επικεφαλίδας, γι΄ αυτό και τα στοιχεία αυτά δεν μπορούν να χρησιμοποιηθούν για να αποδώσουν με σαφήνεια την ιεραρχική δομή του εγγράφου.

Λύση σε τέτοιου είδους περιορισμούς δίνει η XML.

### 2.1.2 Η Λύση της XML

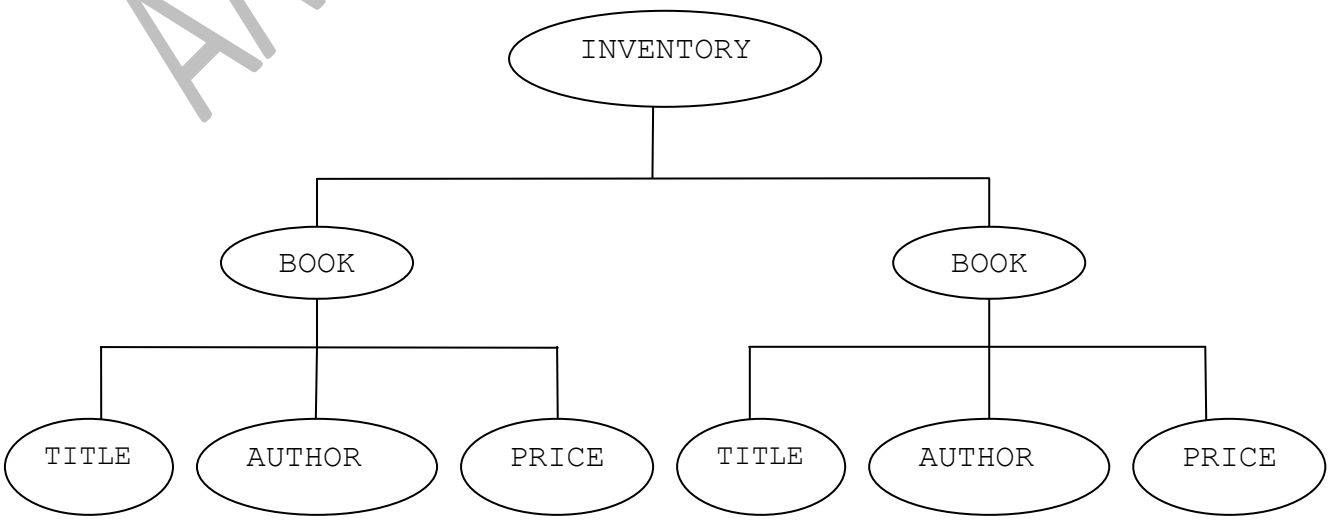
Αν και η HTML είναι προς το παρόν η πιο διαδεδομένη γλώσσα στο χώρο της δημιουργίας ιστοσελίδων, διαθέτει περιορισμένες δυνατότητες αναπαράστασης

πληροφοριών. Αντίθετα, η XML διαθέτει ένα εξαιρετικά ευέλικτο συντακτικό και μπορούμε να δημιουργούμε δικά μας στοιχεία και να τους δίνουμε όποιες ονομασίες θέλουμε – αυτή την έννοια έχει ο όρος *επεκτάσιμη* (extensible) στην ονομασία της γλώσσας. Επομένως, μπορούμε να χρησιμοποιήσουμε την XML για να περιγράψουμε οποιοδήποτε είδος εγγράφου, από μουσικές παρτιτούρες και συνταγές μαγειρικής μέχρι σύνθετες βάσεις δεδομένων. Μπορούμε, για παράδειγμα, να περιγράψουμε μια λίστα βιβλίων, όπως το έγγραφο XML που ακολουθεί στο Σχήμα 2.1 [5].

```
<INVENTORY>
     <BOOK>
        <TITLE>The Adventures of Huckleberry Finn</TITLE>
        <AUTHOR>Mark Twain</AUTHOR>
        <PRICE>$5.49</PRICE>
     </BOOK>
     <BOOK>
        <TITLE>The Scarlet Letter</TITLE>
        <AUTHOR>Nathaniel Hawthorne</AUTHOR>
        <PRICE>$4.25</PRICE>
     </BOOK>
</INVENTORY>
```

**Σχήμα 2.1: Παράδειγμα εγγράφου XML**

Όπως βλέπουμε από το παραπάνω παράδειγμα, ένα έγγραφο XML είναι δομημένο ιεραρχικά σε μορφή δένδρου, με στοιχεία που περικλείονται μέσα σε άλλα και με ένα στοιχείο ανωτάτου επιπέδου (σε αυτό το παράδειγμα `INVENTORY`) – γνωστό ως βασικό στοιχείο (root element) – το οποίο περιέχει όλα τα υπόλοιπα. Η δομή του εγγράφου XML του παραδείγματός μας μπορεί να αποδοθεί ως εξής:



Μπορούμε επομένως πολύ εύκολα να χρησιμοποιήσουμε την XML για να ορίσουμε ένα ιεραρχικά δομημένο έγγραφο, για παράδειγμα, ένα βιβλίο με μέρη, κεφάλαια και ενότητες διαφορετικών επιπέδων.

Επίσης, ένα έγγραφο XML – σε συνδυασμό με ένα stylesheet (φύλλο στυλ) ή μια συμβατική σελίδα HTML – μπορεί να απεικονιστεί εύκολα σε ένα φυλλομετρητή Ιστού (Web browser). Χάρη στην ικανότητα που έχει ένα έγγραφο XML να δομεί και να περιγράφει τόσο αποτελεσματικά τις πληροφορίες που περιέχει (μέσω ετικετών), ο φυλλομετρητής μπορεί να εντοπίζει, να εξάγει, να ταξινομεί, να φιλτράρει, να τακτοποιεί και να χειρίζεται με εξαιρετικά ευέλικτους τρόπους τις πληροφορίες αυτές. Έτσι, η XML προσφέρει την ιδανική λύση στο πρόβλημα χειρισμού της ταχύτατα αυξανόμενης ποσότητας και πολυπλοκότητας των πληροφοριών που χρειάζεται να δημοσιευθούν στον Ιστό.

### 2.1.3 SGML, HTML και XML

Η γλώσσα SGML (Standard Generalized Markup Language) είναι η μητέρα όλων των γλωσσών σήμανσης [5]. Η HTML και η XML έχουν προέλθει και οι δυο από την SGML (αν και με πολύ διαφορετικούς τρόπους). Η SGML ορίζει ένα βασικό συντακτικό και επιτρέπει τη δημιουργία δικών μας στοιχείων (γι΄ αυτό και ο όρος *generalized* – γενικευμένη). Αν θέλουμε να χρησιμοποιήσουμε την SGML για να περιγράψουμε κάποιο συγκεκριμένο έγγραφο, πρέπει να επινοήσουμε ένα κατάλληλο σύνολο στοιχείων και μια δομή εγγράφου.

Ένα σύνολο στοιχείων γενικής χρήσης που χρησιμοποιείται για να περιγράψει ένα συγκεκριμένο είδος εγγράφου είναι γνωστό ως εφαρμογή SGML. Μπορούμε να ορίσουμε τη δική μας εφαρμογή SGML για να περιγράψουμε ένα συγκεκριμένο τύπο εγγράφου. Επίσης, ένας οργανισμός προτυποποίησης μπορεί να ορίσει μια εφαρμογή SGML για να περιγράψει κάποιο ευρέως χρησιμοποιούμενο είδος εγγράφου. Το πιο γνωστό παράδειγμα αυτού του τελευταίου είδους εφαρμογής είναι η HTML. Η SGML αναπτύχθηκε ουσιαστικά το 1986, ενώ η HTML, που είναι μια εφαρμογή της SGML, δημιουργήθηκε το 1991 με σκοπό την περιγραφή ιστοσελίδων.

Η SGML μπορεί να φαίνεται η ιδανική επεκτάσιμη γλώσσα για την περιγραφή εγγράφων στον Ιστό. Ωστόσο, τα μέλη του W3C που ασχολούνται με αυτά τα θέματα θεώρησαν την SGML πολύ σύνθετη και δύσχρηστη για την αποδοτική μεταφορά πληροφοριών στον Ιστό. Αυτό που χρειαζόταν ήταν ένα αποδοτικό υποσύνολο της

SGML, σχεδιασμένο ειδικά για τη μεταφορά πληροφοριών στον Ιστό. Το 1996, η αντίστοιχη ερευνητική ομάδα του W3C ανέπτυξε αυτό το υποσύνολο, το οποίο ονόμασε Extensible Markup Language (Επεκτάσιμη Γλώσσα Σήμανσης). Όπως αναφέραμε παραπάνω, η XML σχεδιάστηκε με στόχο την ευκολία υλοποίησης, κάτι που δε χαρακτηρίζει την SGML.

Η XML, όπως και η SGML, επιτρέπει τον ορισμό δικών μας συνόλων στοιχείων όταν περιγράφουμε κάποιο συγκεκριμένο έγγραφο. Επίσης, όπως και με την SGML, μια εφαρμογή XML (γνωστή και ως *λεξιλόγιο* – vocabulary) μπορεί να οριστεί από έναν ιδιώτη ή από έναν οργανισμό προτυποποίησης. Εφαρμογή XML είναι ένα σύνολο στοιχείων γενικής χρήσης και μια δομή εγγράφου που μπορεί να χρησιμοποιηθεί για την περιγραφή εγγράφων συγκεκριμένου τύπου (για παράδειγμα, εγγράφων που περιέχουν μαθηματικούς τύπους ή διανυσματικά γραφικά).

Το συντακτικό της XML προσφέρει λιγότερες επιλογές από εκείνο της SGML και έτσι είναι πιο εύκολο για τους χρήστες να διαβάζουν τα έγγραφα XML και για τους προγραμματιστές να γράφουν φυλλομετρητές, scripts (σενάρια) και websites (ιστοσελίδες) που μπορούν να προσπελαστούν και να εμφανίζουν τις πληροφορίες του εγγράφου.

### 2.1.4 Η XML Αντικαθιστά την HTML;

Προς το παρόν η απάντηση σε αυτό το ερώτημα είναι όχι. Η HTML εξακολουθεί να είναι η πρωταρχική γλώσσα για την εμφάνιση πληροφοριών στον Ιστό [5]. Αντί να αντικατασταθεί η HTML, χρησιμοποιείται προς το παρόν σε συνδυασμό με την XML και αυξάνει τις δυνατότητες των ιστοσελίδων ως προς τα ακόλουθα:

⇒ Μεταφορά, ουσιαστικά, οποιουδήποτε είδους εγγράφου.

⇒ Ταξινόμηση, φιλτράρισμα, αναδιάταξη, εντοπισμός και διαχείριση των πληροφοριών με άλλους τρόπους.

⇒ Παρουσίαση εξαιρετικά δομημένων πληροφοριών.

## 2.2 Δημιουργία XML Εγγράφου

### 2.2.1 Ανατομία XML Εγγράφου

Τα περιεχόμενα ενός XML εγγράφου μπορούν να είναι κάποια από τα ακόλουθα [6]:

- *Elements* (Στοιχεία)
- *Attributes* (Ιδιότητες)
- *Comments* (Σχόλια)
- *Processing Instructions* (Εντολές επεξεργασίας - αναφέρονται και ως «στιγμιότυπο εγγράφου»)

**Στοιχεία.** Τα στοιχεία δείχνουν τη λογική δομή ενός εγγράφου XML και περιέχουν τις πληροφορίες του. Ένα έγγραφο XML αποτελείται από ένα βασικό στοιχείο (root element). Κάθε στοιχείο αποτελείται από ένα *start tag* (ετικέτα αρχής), το περιεχόμενο και ένα *end tag* (ετικέτα τέλους). Το περιεχόμενο μπορεί να αποτελείται από δεδομένα χαρακτήρων, άλλα (ένθετα) στοιχεία ή από συνδυασμό και των δύο. Ένα παράδειγμα στοιχείου φαίνεται στο Σχήμα 2.2 [7].

```
<BOOK>
   <TITLE>TCP/IP Illustrated</TITLE>
   <AUTHOR>Stevens W.</AUTHOR>
   <PUBLISHER>Addison-Wesley</PUBLISHER>
   <PRICE>$65.95</PRICE>
</BOOK>
```

**Σχήμα 2.2: Παράδειγμα XML στοιχείου**

Το βασικό στοιχείο είναι το `BOOK`. Η ετικέτα αρχής του στοιχείου αυτού είναι το `<BOOK>`, η ετικέτα τέλους το `</BOOK>` και το περιεχόμενό του είναι 4 ένθετα στοιχεία. Τα 4 αυτά ένθετα στοιχεία είναι τα `TITLE`, `AUTHOR`, `PUBLISHER` και `PRICE`. Καθένα από αυτά περιέχει μόνο δεδομένα χαρακτήρων.

**Ιδιότητες.** Μια ιδιότητα περιλαμβάνεται στην ετικέτα αρχής ενός στοιχείου και πρόκειται για ένα ζεύγος name – value, όπου name είναι το όνομα της ιδιότητας και value η τιμή της, το οποίο συνδέεται με το στοιχείο. Θεωρούμε το προηγούμενο παράδειγμα με το στοιχείο `BOOK`, στο οποίο προσθέτουμε την ιδιότητα `YEAR="1994"`, οπότε προκύπτει το Σχήμα 2.3.

```
<BOOK YEAR="1994">
   <TITLE>TCP/IP Illustrated</TITLE>
   <AUTHOR>Stevens W.</AUTHOR>
   <PUBLISHER>Addison-Wesley</PUBLISHER>
   <PRICE>$65.95</PRICE>
</BOOK>
```

**Σχήμα 2.3: Παράδειγμα XML στοιχείου με ιδιότητα**

Η ιδιότητα `YEAR="1994"` δηλώνει την ημερομηνία έκδοσης του συγκεκριμένου βιβλίου. Τα attributes παίρνουν βασικά μόνο απλές τιμές, δηλαδή δεν έχουν κάποια δομή. Η διαδικασία προσθήκης ιδιοτήτων αποτελεί μια εναλλακτική λύση που μας δίνει τη δυνατότητα να συμπεριλάβουμε πληροφορίες σε ένα στοιχείο. Η χρήση ιδιοτήτων έχει ως σκοπό την αποθήκευση διάφορων χαρακτηριστικών του στοιχείου, τα οποία δεν είναι απαραίτητο να εμφανιστούν.

**Σχόλια.** Σχόλιο είναι μια σημείωση που προσθέτουμε στο έγγραφό μας και η οποία μπορεί να διαβαστεί από οποιονδήποτε εκτός από τον επεξεργαστή. Ένα σχόλιο αρχίζει με τους χαρακτήρες `<!--` , δέχεται ως περιεχόμενο οποιαδήποτε ακολουθία χαρακτήρων και τελειώνει με τους χαρακτήρες `-->`. Επίσης, μπορεί να χρησιμοποιηθεί οπουδήποτε μέσα στο έγγραφο. Ένα παράδειγμα σχόλιου φαίνεται στο Σχήμα 2.4.

```
<SXOLH_THETIKWN_EPISTHMWN>
     <TMHMA>Tmima Pliroforikis</TMHMA>
     <!-- Auto einai ena sxolio -->
</SXOLH_THETIKWN_EPISTHMWN>
```

**Σχήμα 2.4: Παράδειγμα σχόλιου**

**Εντολές Επεξεργασίας.** Οι εντολές επεξεργασίας χρησιμοποιούνται για την παροχή πληροφοριών, τις οποίες ο επεξεργαστής XML μεταφέρει σε εφαρμογές. Έχουν τη γενική μορφή: `<?target instruction?>`, όπου target είναι το όνομα της εφαρμογής στην οποία απευθύνεται η εντολή και instruction είναι το σύνολο των πληροφοριών που μεταφέρονται στην εφαρμογή (εντολές). Όπως και τα σχόλια, μπορούν να τοποθετηθούν σε οποιοδήποτε σημείο του εγγράφου, αρκεί να μη βρίσκονται μέσα σε άλλα tags (ετικέτες). Το Σχήμα 2.5 δείχνει ένα παράδειγμα εντολής επεξεργασίας [5].

```
<?xml-stylesheet TYPE="text/css" HREF="Book.css"?>
```

**Σχήμα 2.5: Παράδειγμα εντολής επεξεργασίας**

Η εντολή αυτή λέει στην εφαρμογή να χρησιμοποιήσει το φύλλο επάλληλων στυλ (cascading stylesheet) που βρίσκεται στο αρχείο Book.css.

### 2.2.2 Ορισμός Τύπου Εγγράφου (Document Type Definition – DTD)

Ένα έγγραφο XML μπορεί να περιλαμβάνει μια κατάλληλη δήλωση τύπου εγγράφου (Document Type Declaration). Η δήλωση αυτή περιέχει έναν ορισμό τύπου εγγράφου (Document Type Definition – DTD), ο οποίος αποτελείται από τις δηλώσεις που προσδιορίζουν τα στοιχεία, τις ιδιότητες και άλλα χαρακτηριστικά του εγγράφου. Η δήλωση τύπου εγγράφου έχει την εξής γενική μορφή:

```
<!DOCTYPE  Όνομα  DTD>
```

όπου το `Όνομα` προσδιορίζει το όνομα του βασικού στοιχείου.

Το DTD αποτελείται από μια αριστερή αγκύλη ( `[` ) ακολουθούμενη από μια σειρά δηλώσεων σήμανσης (markup declarations) και μια δεξιά αγκύλη ( `]` ). Οι δηλώσεις σήμανσης περιγράφουν τη λογική δομή του εγγράφου. Ένα παράδειγμα DTD απεικονίζεται στο Σχήμα 2.6 που ακολουθεί [7].

```
<!DOCTYPE BOOK
  [
     <!ELEMENT BOOK   (TITLE, AUTHOR+, PUBLISHER, PRICE)>
     <!ATTLIST BOOK      YEAR  CDATA  #REQUIRED>
     <!ELEMENT AUTHOR    (#PCDATA)>
     <!ELEMENT TITLE     (#PCDATA)>
     <!ELEMENT PUBLISHER (#PCDATA)>
     <!ELEMENT PRICE     (#PCDATA)>
  ]
>
```

**Σχήμα 2.6: Παράδειγμα DTD**

Το DTD αυτό αναφέρεται στο βασικό στοιχείο `BOOK` και προσδιορίζει τα στοιχεία του. Το στοιχείο `BOOK` περιέχει ένα στοιχείο τύπου `TITLE`, ένα ή περισσότερα στοιχεία τύπου `AUTHOR` (η σημασία του συμβόλου '+' μετά από το `AUTHOR` εξηγείται σε άλλο κεφάλαιο αναλυτικά), ένα στοιχείο τύπου `PUBLISHER` και

ένα τύπου PRICE. Καθένα από αυτά τα στοιχεία είναι ακολουθίες χαρακτήρων (PCDATA). Επίσης, περιέχει και την ιδιότητα YEAR η οποία παίρνει μια αριθμητική τιμή και είναι απαραίτητη (REQUIRED). Για παράδειγμα, ένα έγγραφο XML που πληρεί τους κανόνες του παραπάνω DTD, περιλαμβάνει δεδομένα όπως αυτά στο Σχήμα 2.7.

```
<BOOK YEAR="1994">
   <TITLE>TCP/IP Illustrated</TITLE>
   <AUTHOR>Stevens W.</AUTHOR>
   <PUBLISHER>Addison-Wesley</PUBLISHER>
   <PRICE>$65.95</PRICE>
</BOOK>
```

**Σχήμα 2.7: Δεδομένα εγγράφου XML που πληρεί τους κανόνες του DTD του**

Αν θέλουμε να είμαστε βέβαιοι ότι το έγγραφό μας ακολουθεί μια συγκεκριμένη δομή ή σύνολο κανόνων, η παρουσία ενός DTD που θα ορίζει τη δομή αυτή, επιτρέπει σε κάποιον επεξεργαστή XML να ελέγξει κατά πόσο το έγγραφό μας ακολουθεί τη προβλεπόμενη δομή.

Εκτός από το DTD, υπάρχει και το πρότυπο *XML Schema (XML Σχήμα)* [6]. Σε ένα σύστημα διαχείρισης βάσης δεδομένων, το σχήμα είναι μια τυπική περιγραφή της δομής της βάσης. Ορίζει τις εσωτερικές δομές, όπως πίνακες και πεδία και τις σχέσεις μεταξύ τους. Ένα σχήμα περιγράφεται με περιορισμούς (constraints). Υπάρχουν δύο βασικοί τύποι περιορισμών: οι περιορισμοί περιεχομένου (*content constraints)* που καθορίζουν πού και πότε μπορούν να χρησιμοποιηθούν στοιχεία, και οι περιορισμοί τύπου δεδομένων *(datatype constraints)* που καθορίζουν τους τύπους δεδομένων που μπορούν να έχουν τα στοιχεία. Παρόλο που και το DTD και το XML Schema παρέχουν κάποια δομή για τα έγγραφα, παρουσιάζουν αρκετές θεμελιώδεις διαφορές μεταξύ τους:

1. Τα DTDs είναι γραμμένα με βάση την τυπική μορφή EBNF (Extended Backus – Naur Form), ενώ τα Schemas είναι γραμμένα σε XML.
2. Τα DTDs έχουν ελάχιστους περιορισμούς τύπου δεδομένων. Για παράδειγμα, ένα στοιχείο <TELEPHONE> θα μπορούσε να οριστεί ότι περιέχει CDATA, αλλά με τη χρήση DTD δε θα μπορούσε να περιοριστεί σε αριθμητικές τιμές.

Αντίθετα, με τη χρήση του XML Schema μπορούμε να εφαρμόσουμε πιο ειδικούς περιορισμούς πάνω στα δεδομένα.

3. Στα DTDs υπάρχει ένα περιορισμένο σύνολο μοντέλων περιεχομένου, ενώ στα XML Schemas υπάρχει μεγαλύτερη ευελιξία.

### 2.2.3 Οντότητες (Entities)

Ο μηχανισμός *οντότητας* XML αποτελεί ένα εργαλείο εξοικονόμησης χρόνου, καθώς και έναν τρόπο ενσωμάτωσης διαφόρων τύπων δεδομένων σε έγγραφα XML. Σε ένα τέτοιο έγγραφο μπορούμε να ορίσουμε ως οντότητα κάποιο μπλοκ κειμένου XML που χρησιμοποιείται συχνά. Αυτό μας δίνει τη δυνατότητα να εισάγουμε γρήγορα το κείμενο στο έγγραφό μας, όποτε το θελήσουμε. Μπορούμε επίσης να ορίσουμε ως οντότητα ένα εξωτερικό αρχείο και έτσι να συνδέουμε τα δεδομένα του αρχείου με το έγγραφό μας. Τα στοιχεία αυτά μπορεί να αποτελούνται από κείμενο XML, από κάποιο άλλο κείμενο ή από άλλα δεδομένα διαφορετικά από κείμενο. Επίσης, μπορούμε να ορίσουμε μια οντότητα στον ορισμό τύπου εγγράφου (DTD), χρησιμοποιώντας σύνταξη παρόμοια με εκείνη που χρησιμοποιούμε για τη δήλωση ενός στοιχείου ή μιας ιδιότητας σε ένα έγκυρο έγγραφο XML [5].

**Τύποι Οντοτήτων.** Οι οντότητες διακρίνονται σε πολλές κατηγορίες:

- *Γενικές (general)*. Η γενική οντότητα έχει περιεχόμενο εγγράφου – δηλαδή κείμενο XML, άλλου είδους κείμενο ή δεδομένα εκτός κειμένου, τα οποία μπορούμε να χρησιμοποιήσουμε στο βασικό στοιχείο.

- *Παραμέτρων (parameter)*. Η οντότητα παραμέτρων περιέχει κείμενο XML, το οποίο μπορεί να συμπεριληφθεί στο DTD.

- *Εσωτερικές (internal)*. Η εσωτερική οντότητα περιέχεται μέσα σε ένα αλφαριθμητικό εισαγωγικό.

- *Εξωτερικές (external)*. Η εξωτερική οντότητα περιέχεται σε ένα ξεχωριστό αρχείο.

- *Αναλυόμενες (parsed)*. Η αναλυόμενη οντότητα αποτελείται από κείμενο XML (δεδομένα χαρακτήρων, σήμανση ή και τα δυο). Όταν προσθέτουμε μια αναφορά σε μια αναλυόμενη οντότητα του εγγράφου, η αναφορά αντικαθίσταται από τα περιεχόμενα της οντότητας (γνωστά επίσης και ως *κείμενα αντικατάστασης*), τα οποία ενσωματώνονται πλέον στο έγγραφο.

- *Μη αναλυόμενες (unparsed)*. Η μη αναλυόμενη οντότητα μπορεί να περιέχει οποιοδήποτε είδος δεδομένων (δεδομένα XML ή δεδομένα μη XML). Συνήθως δεν περιλαμβάνει δεδομένα XML, γι΄ αυτό και τα περιεχόμενά της δεν εισάγονται άμεσα στο έγγραφο μέσω κάποιας αναφοράς οντότητας.

Ακολουθεί ένα παράδειγμα DTD (Σχήμα 2.8) που ορίζει το εξωτερικό αρχείο Topics.xml (ένα αρχείο που περιέχει κατάλογο με τα θέματα του άρθρου το οποίο περιέχεται στο έγγραφο) ως εξωτερική οντότητα με το όνομα topics και ορίζει ένα αλφαριθμητικό εισαγωγικών (`"A Short History of Xml"`) ως εσωτερική οντότητα με το όνομα `title`.

```
<!DOCTYPE  ARTICLE
   [
    <!ELEMENT ARTICLE (TITLEPAGE, INTRODUCTION, SECTION*)>
    <!ELEMENT TITLEPAGE      (#PCDATA)>
    <!ELEMENT INTRODUCTION   (#PCDATA)>
    <!ELEMENT SECTION        (#PCDATA)>

    <!ENTITY  topics  SYSTEM  "Topics.xml">
    <!ENTITY  title  "A Short History of Xml">
   ]
>
```

**Σχήμα 2.8: Παράδειγμα DTD με εσωτερική και εξωτερική οντότητα**

Θα μπορούσαμε επομένως να προσθέσουμε τον τίτλο του άρθρου σε οποιοδήποτε σημείο του εγγράφου και να συμπεριλάβουμε την αναφορά οντότητας `&title;` - όπως στο στοιχείο που φαίνεται στο Σχήμα 2.9.

```
<TITLEPAGE>
     Title:  &title;
     Author: Michael Young
</TITLEPAGE>
```

**Σχήμα 2.9: Παράδειγμα στοιχείου με αναφορά οντότητας**

Μια σημαντική παρατήρηση είναι ότι ο μηχανισμός οντοτήτων της XML μοιάζει με τον ορισμό μακροεντολών (macros) σε μια γλώσσα προγραμματισμού (όπως αυτές που δηλώνονται με τη χρήση της εντολής προεπεξεργαστή `#define` στη C).

### 2.2.4 Εγκυρότητα Εγγράφου

Ένα έγγραφο μπορεί να χαρακτηριστεί ως «*σωστά διατυπωμένο*» (well-formed) ή «*έγκυρο*» (valid) ανάλογα με τις απαιτήσεις που πληρεί. Κάθε XML έγγραφο είναι ουσιαστικά «σωστά διατυπωμένο» [6]. Ένα έγγραφο χαρακτηρίζεται ως «σωστά διατυπωμένο» όταν ικανοποιεί κάποιους κανόνες, όπως οι ακόλουθοι:

- ⇒ Πρέπει να έχει ετικέτες αρχής και τέλους (start και end tag αντίστοιχα) σε κάθε στοιχείο
- ⇒ Πρέπει να έχει 1 και μόνο βασικό στοιχείο (root element)
- ⇒ Τα άδεια στοιχεία δομούνται σωστά
- ⇒ Οι ετικέτες αρχής και τέλους μπορεί να είναι με κεφαλαία γράμματα ή πεζά, αρκεί να ταιριάζουν μεταξύ τους
- ⇒ Τα νέα στοιχεία πρέπει να ενσωματώνονται σωστά το ένα μέσα στο άλλο
- ⇒ Οι τιμές των ιδιοτήτων πρέπει να είναι πάντα μέσα σε εισαγωγικά

Ένα έγκυρο έγγραφο είναι σωστά διατυπωμένο και γίνεται έγκυρο με βάση ένα DTD (ή κάποιο άλλο σχήμα). Το DTD παρέχει σε έναν επεξεργαστή XML ένα πρότυπο σχέδιο, ώστε ελέγχοντας την εγκυρότητα του εγγράφου να μπορεί να επιβάλλει την επιθυμητή δομή και να εγγυηθεί ότι το έγγραφό μας πληρεί τις απαραίτητες προϋποθέσεις.

Η συμπερίληψη DTD και ο έλεγχος εγκυρότητας είναι ιδιαίτερα σημαντικές διαδικασίες όταν η επεξεργασία των εγγράφων γίνεται από προσαρμοσμένο λογισμικό,

το οποίο αναμένει μια συγκεκριμένη δομή εγγράφου. Αν όλοι οι χρήστες συμπεριλάβουν στα XML έγγραφά τους ένα κοινό DTD και αν στα έγγραφα γίνεται έλεγχος εγκυρότητας, οι χρήστες μπορούν να είναι σίγουροι ότι τα έγγραφά τους θα αναγνωρίζονται από το λογισμικό επεξεργασίας. Για παράδειγμα, αν μια ομάδα μαθηματικών επιστημόνων θέλει να δημιουργήσει μαθηματικά έγγραφα που θα εμφανιστούν μέσω κάποιου συγκεκριμένου προγράμματος, μπορούν να συμπεριλάβουν στα έγγραφά τους ένα κοινό DTD που θα ορίζει την απαιτούμενη δομή, τα στοιχεία, τις ιδιότητες και άλλα χαρακτηριστικά των εγγράφων.

## 2.3 Εφαρμογές XML για Βελτίωση Εγγράφων

Εκτός από τις εφαρμογές XML που χρησιμοποιούνται για την περιγραφή ειδικών τάξεων εγγράφων, έχουν οριστεί και αρκετές άλλες εφαρμογές, τις οποίες μπορούμε να χρησιμοποιήσουμε σε οποιοδήποτε είδος εγγράφου. Οι εφαρμογές αυτές διευκολύνουν τη δημιουργία των εγγράφων και επιτρέπουν τη βελτιστοποίησή τους. Παραδείγματα τέτοιων εφαρμογών είναι τα ακόλουθα [5]:

- *Επεκτάσιμη Γλώσσα Φύλλων Στυλ (Extensible Stylesheet Language – XSL).* Επιτρέπει τη δημιουργία ισχυρών φύλλων στυλ, χρησιμοποιώντας το συντακτικό της XML [8]. Ένα φύλλο στυλ XSL συνδέεται με ένα έγγραφο XML και δίνει εντολές στο φυλλομετρητή (browser) πώς να εμφανίσει τα δεδομένα XML. Έτσι, δίνεται η δυνατότητα να ανοίγουμε ένα έγγραφο XML κατευθείαν στο φυλλομετρητή, χωρίς τη χρήση ενδιάμεσης σελίδας HTML. Η χρήση ενός φύλλου στυλ XSL είναι πολύ πιο ισχυρή και ευέλικτη από τη χρήση φύλλων επάλληλων στυλ (CSS – Cascading Style Sheets). Ενώ το φύλλο στυλ CSS επιτρέπει απλώς τον ορισμό της μορφοποίησης κάθε στοιχείου XML, το φύλλο στυλ XSL δίνει τη δυνατότητα πλήρους ελέγχου του αποτελέσματος. Συγκεκριμένα, η XSL μας επιτρέπει να επιλέξουμε με ακρίβεια τα δεδομένα της XML που θέλουμε να εμφανίσουμε, να παρουσιάσουμε τα δεδομένα με οποιαδήποτε σειρά ή διάταξη, να κάνουμε τροποποιήσεις ή να προσθέσουμε πληροφορίες. Η XSL μας δίνει επίσης τη δυνατότητα να προσπελαύνουμε όλα τα τμήματα ενός εγγράφου XML (όπως στοιχεία, ιδιότητες, σχόλια και εντολές επεξεργασίας), να ταξινομούμε και να φιλτράρουμε με ευκολία τα δεδομένα XML και να συμπεριλαμβάνουμε

σενάρια στο φύλλο στυλ, ενώ μας παρέχει ένα σύνολο χρήσιμων μεθόδων, με τις οποίες μπορούμε να επεξεργαζόμαστε τις πληροφορίες μας.

- *Σχήμα XML (XML Schema)*. Επιτρέπει να γράφουμε λεπτομερή σχήματα (schemas) για τα έγγραφά μας, χρησιμοποιώντας το καθιερωμένο συντακτικό της XML. Παρέχει μια πιο ισχυρή εναλλακτική λύση από ό,τι το DTD.

- *Γλώσσα Σύνδεσης XML (XML Linking Language – XLink)*. Επιτρέπει τη σύνδεση των εγγράφων μας και τη χρήση πολλαπλών περιορισμών συνδέσμων, καθώς και άλλα προηγμένα χαρακτηριστικά. Είναι σαφώς πιο ισχυρή από το μηχανισμό σύνδεση της HTML [9].

- *Γλώσσα Δεικτών XML (XML Pointer Language – XPointer)*. Επιτρέπει τον ορισμό ευέλικτων προορισμών σύνδεσης. Μπορούμε να χρησιμοποιήσουμε την XPointer σε συνδυασμό με την XLink και να συνδεθούμε σε οποιαδήποτε θέση στο έγγραφο προορισμού – όχι μόνο σε κάποιο ειδικά επισημασμένο προορισμό σύνδεσης [10].

- *Γλώσσα Ερωταποκρίσεων XML (XML Query Language – XQuery)*. Είναι μια δυναμική και εύκολη γλώσσα, η οποία σχεδιάστηκε για την επεξεργασία δεδομένων είτε σε μορφή XML είτε σε κάποια άλλη μορφή, για παράδειγμα μια βάση δεδομένων, της οποίας η δομή είναι όμοια με αυτήν της XML [11].

- *Γλώσσα Χρήσης Μονοπατιών XML (XML Path Language – XPath)*. Είναι το αποτέλεσμα μιας προσπάθειας να παραχθεί μια κοινή σύνταξη και σημασιολογία για την επίτευξη λειτουργικότητας μεταξύ του XSLT (XSL Transformations) και του XPointer. Ο αρχικός σκοπός της είναι να διευθυνσιοδοτεί τα μέρη ενός XML εγγράφου. Για το σκοπό αυτόν, παρέχει βασικές ευκολίες για το χειρισμό αλφαριθμητικών, αριθμών και λογικών τιμών. Παίρνει το όνομά της από τη χρήση μονοπατιών για την πλοήγηση στην ιεραρχική δομή του εγγράφου. Μοντελοποιεί ένα XML έγγραφο ως ένα δένδρο από κόμβους [12].

## 2.4 Πρακτικές Χρήσεις της XML

Τέλος, παραθέτουμε έναν κατάλογο με μερικές από τις πρακτικές χρήσεις της γλώσσας XML. Στον κατάλογο αυτόν συμπεριλαμβάνονται οι τρόποι με τους οποίους χρησιμοποιείται προς το παρόν η XML, καθώς και οι χρήσεις που έχουν προταθεί από διάφορες ομάδες [5].

- *Δημοσίευση βάσεων δεδομένων*. Όπως συμβαίνει με τις αποκλειστικές μορφές βάσεων δεδομένων, η XML μπορεί να χρησιμοποιηθεί για να τοποθετήσει ετικέτες σήμανσης σε οποιοδήποτε πεδίο πληροφοριών μέσα σε οποιαδήποτε βάση δεδομένων. Με τον τρόπο αυτόν μας δίνει την ευκαιρία να εμφανίζουμε τα δεδομένα, καθώς επίσης να αναζητούμε συγκεκριμένα δεδομένα, να τα ταξινομούμε, να τα φιλτράρουμε και να τα επεξεργαζόμαστε με πολλούς και ποικίλους τρόπους. Για τους παραπάνω σκοπούς, εταιρίες όπως η Oracle και η Microsoft, χρησιμοποιούν σε εφαρμογές components (συστατικά) βασισμένα σε XML.

- *Δόμηση εγγράφων*. Η δενδροειδής δομή των εγγράφων XML κάνει τη γλώσσα αυτή ιδανική για την περιγραφή της δομής εγγράφων, όπως μυθιστορήματα, βιβλία γενικού ενδιαφέροντος και θεατρικά έργα. Για παράδειγμα, μπορούμε να χρησιμοποιήσουμε την XML για να χωρίσουμε το έργο σε πράξεις, σκηνές, ομιλητές, ατάκες, σκηνικές οδηγίες κ.ά. Κάτι τέτοιο επιτρέπει στο λογισμικό να εμφανίζει ή να τυπώνει το έγγραφο με την κατάλληλη μορφοποίηση, να εντοπίζει, να εξάγει, ή να χειρίζεται τις πληροφορίες του εγγράφου, να δημιουργεί αυτόματα πίνακες περιεχομένων, διαρθρώσεις και περιλήψεις, και γενικά να χειρίζεται τις πληροφορίες μας με διάφορους τρόπους.

- *Επεκτάσιμη Γλώσσα Σήμανσης Υπερκειμένου (XHTML – Extensible Hypertext Markup Language).* Η γλώσσα αυτή επιτρέπει σε ένα συγγραφέα να διατηρήσει τη σημασιολογία και την παρουσίαση ενός HTML εγγράφου, και παρέχει μία ομαλή μετάβαση από HTML σε XML. Πέραν τούτου, οι συγγραφείς πρέπει να είναι πολύ προσεκτικοί με τη δόμηση των εγγράφων και αυτό είναι αρκετά καλό γι΄ αυτούς. Ένας καλός λόγος πάντως, για να

στραφούμε προς τη γλώσσα XHTML, είναι η δυνατότητα μεταφερσιμότητας και η ευκολία συμβατότητας που παρέχει [6].

- *Παρουσίαση διανυσματικών γραφικών (VML ή Vector Markup Language – Γλώσσα Σήμανσης Διανυσμάτων)*. Πρόκειται για μια XML εφαρμογή, η οποία ορίζει το σχήμα για την κωδικοποίηση διανυσματικών πληροφοριών με πρόσθετη σήμανση, για να περιγράψει τον τρόπο αναπαράστασης των πληροφοριών [13].

- *Περιγραφή παρουσιάσεων με πολυμέσα (SMIL ή Synchronized Multimedia Integration Language και HTML+TIME ή HTML Interactive Multimedia Extensions).* Η SMIL επιτρέπει απλή συγγραφή διαδραστικών οπτικοακουστικών παρουσιάσεων. Χρησιμοποιείται για πολυμεσικές παρουσιάσεις που ενσωματώνουν στη ροή ήχου και βίντεο εικόνες, κείμενο ή άλλους τύπους μέσων. Είναι εύκολη στη εκμάθηση και μοιάζει με την HTML [14].

- *Μορφοποίηση μαθηματικών τύπων και επιστημονικών πληροφοριών στον Ιστό (MathML ή Mathematical Markup Language)*. Η MathML είναι μία χαμηλού επιπέδου προδιαγραφή για την περιγραφή των μαθηματικών ως βάση για την επικοινωνία μεταξύ μηχανών. Αποτελεί βασικό θεμέλιο για την ενσωμάτωση και υψηλού επιπέδου παρουσίαση μαθηματικών εκφράσεων σε ιστοσελίδες [15].

- *Επικοινωνία μεταξύ εφαρμογών στον Ιστό με ανοιχτό και επεκτάσιμο τρόπο, χρησιμοποιώντας μηνύματα που βασίζονται στην XML*. Τα μηνύματα αυτά είναι ανεξάρτητα από τα λειτουργικά συστήματα, τα μοντέλα αντικειμένων και τις γλώσσες προγραμματισμού που χρησιμοποιούνται (SOAP ή Simple Object Access Protocol).

- *Ανταλλαγή χρηματοπιστωτικών πληροφοριών.* Οι πληροφορίες αυτές ανταλλάσσονται με έναν ανοιχτό αναγνώσιμο τρόπο ανάμεσα σε χρηματοπιστωτικά προγράμματα (όπως το Quicken και το Microsoft Money)

και οργανισμούς, όπως τράπεζες και αμοιβαία κεφάλαια (OFX ή Open Financial Exchange).

- *Δημιουργία, διαχείριση και χρήση πολύπλοκων ψηφιακών φορμών για τις εμπορικές συναλλαγές μέσω Internet.* Οι φόρμες μπορεί να περιλαμβάνουν ψηφιακές υπογραφές που τις κάνουν νομικά δεσμευτικές (XFDL ή Extensible Forms Description Language).

- *Αποθήκευση φωνητικών σεναρίων για μεταφορά τους μέσω τηλεφώνου.* Φωνητικά σενάρια μπορούν για παράδειγμα να χρησιμοποιηθούν για την εκφώνηση οδηγιών φωνητικού ταχυδρομείου, αξιών μετοχών και δελτίων καιρού. (VoxML)

- *Ορισμός καναλιών*. Τα κανάλια είναι ιστοσελίδες που προωθούνται (στέλνονται αυτόματα) σε συνδρομητές (CDF ή Channel Definition Format).

- *Περιγραφή πακέτων λογισμικού και των αλληλεξαρτήσεών τους*. Αυτές οι περιγραφές επιτρέπουν τη διανομή και την αναβάθμιση λογισμικού μέσω δικτύων (OSD ή Open Software Description).

Άλλες πρακτικές χρήσεις της XML είναι οι ακόλουθες:

- *Ανταλλαγή περιγραφών θέσεων εργασίας και βιογραφικών σημειωμάτων (HRMML ή Human Resource Management Markup Language).*

- *Περιγραφή μοριακών δομών (CML ή Chemical Markup Language).*

- *Κωδικοποίηση και εμφάνιση πληροφοριών σχετικά με ακολουθίες DNA, RNA και πρωτεϊνών (BSML ή Bioinformatic Sequence Markup Language).*

- *Κωδικοποίηση γενεαλογικών στοιχείων (GedML ή Genealogical Data Markup Language).*

- *Ανταλλαγή πληροφοριών σχετικά με την αστρονομία (AML ή Astronomical Markup Language).*

- *Συγγραφή μουσικής παρτούρας (MusicML ή Music Markup Language).*

- *Αποστολή ηλεκτρονικών επαγγελματικών καρτών μέσω ηλεκτρονικού ταχυδρομείου (Vcard).*

- *Αποθήκευση πληροφοριών εντοπισμού για υπηρεσίες courier.* Η Federal Express, για παράδειγμα, χρησιμοποιεί την XML γι΄ αυτόν το σκοπό.

- *Υποβολή αγγελιών σε εφημερίδες σε ψηφιακή μορφή (Ad Markup).*

- *Αρχειοθέτηση νομικών εγγράφων και ανταλλαγή νομικών πληροφοριών ηλεκτρονικά. (XCL ή XML Court Interface).*

- *Κωδικοποίηση δελτίων καιρού (OMF ή Weather Observation Markup Format).*

- *Ανταλλαγή πληροφοριών σχετικά με συναλλαγές ακίνητης περιουσίας (RETS ή Real Estate Transaction Standard).*

- *Ανταλλαγή πληροφοριών σχετικά με ασφάλειες.*

- *Ανταλλαγή ειδήσεων και πληροφοριών με τη χρήση προτύπων Ιστού (XMLNews).*

- *Αναπαράσταση πληροφοριών θεολογικού περιεχομένου και σήμανση εκκλησιαστικών κειμένων (ThML ή Theological Markup Language και LitML ή Liturgigal Markup Language).*

# 3 ΔΙΑΧΕΙΡΙΣΗ XML ΔΕΔΟΜΕΝΩΝ

## 3.1 Εισαγωγή

Η επιτυχία του Διαδικτύου (Internet) εξαρτάται από τη διαθεσιμότητα των εφαρμογών που παρέχουν αξιόπιστες ηλεκτρονικές υπηρεσίες. Πάντα, όμως, οι εφαρμογές εξαρτώνται από τα δεδομένα εισόδου και κυρίως από τη σωστή δομή τους. Μέχρι τώρα, η αναπαράσταση και ανταλλαγή πληροφοριών στο Internet πραγματοποιείται μέσω HTML σελίδων, χωρίς την ύπαρξη κάποιας εννοιολογικής δομής που να χαρακτηρίζει τα δεδομένα. Η γλώσσα XML είναι το τρέχον πρότυπο για την ανταλλαγή δομημένων και ημιδομημένων πληροφοριών στο Internet [4]. Ωστόσο, η αποθήκευση και διαχείριση των πληροφοριών αυτών είναι εξίσου σημαντική. Η ενσωμάτωση, ο διαμοιρασμός, η επαναχρησιμοποίηση και η επαύξηση των πληροφοριών, που συλλαμβάνονται από XML έγγραφα, είναι σημαντικές διεργασίες για την κατασκευή μεγάλης κλίμακας εφαρμογών με βιομηχανική ισχύ.

Η διαχείριση δεδομένων ή πληροφοριών πραγματοποιείται στις σημερινές εφαρμογές από τα *συστήματα διαχείρισης βάσεων δεδομένων (DataBase Management Systems - DBMSs)*. Πάνω από τρεις δεκαετίες έρευνας έχουν αφιερωθεί στην ανάπτυξη θεωριών και συστημάτων για τη σύλληψη, αποθήκευση, διαχείριση και ανάκτηση δεδομένων για έναν ή πολλούς χρήστες. Ένας τέτοιος όγκος έρευνας και ανάπτυξης θα πρέπει να επαναχρησιμοποιηθεί για τη διαχείριση ημιδομημένων δεδομένων, δηλαδή XML ή SGML δεδομένων, με την ελάχιστη δυνατή προσπάθεια. Υπάρχουν ήδη αρκετές προτάσεις για μεθοδολογίες αποθήκευσης, ανάκτησης και διαχείρισης ημιδομημένων δεδομένων, που είναι αποθηκευμένα σε σχεσιακές, αντικειμενοστραφείς βάσεις δεδομένων ή συνδυασμό και των δύο αυτών ειδών [1].

## 3.2 Μέθοδοι Διαχείρισης XML Δεδομένων

Υπάρχουν δύο βασικές μέθοδοι για τη διαχείριση XML εγγράφων και τη διατύπωση ερωτήσεων προς τη βάση δεδομένων. Η πρώτη μέθοδος χρησιμοποιεί μηχανές

ερωτήσεων ειδικού σκοπού και συστήματα αποθήκευσης ημιδομημένων δεδομένων [16, 17, 18, 19, 20]. Αυτά τα συστήματα βάσεων δεδομένων έχουν κατασκευαστεί ειδικά για την αποθήκευση XML εγγράφων και τη διατύπωση ερωτήσεων προς αυτά. Όμως, αυτή η προσέγγιση έχει δύο μειονεκτήματα. Το πρώτο έχει να κάνει με το γεγονός ότι τα απλά συστήματα βάσεων XML δεδομένων δεν έχουν τη δυνατότητα σύνθετης αποθήκευσης και διατύπωσης ερωτήσεων, που ήδη υποστηρίζονται από τα υπάρχοντα συστήματα βάσεων δεδομένων. Το δεύτερο μειονέκτημα αφορά το ότι τα απλά συστήματα βάσεων δεδομένων δεν επιτρέπουν στους χρήστες να διατυπώνουν ερωτήσεις που δεν περιλαμβάνουν συνδέσεις XML εγγράφων με άλλα δομημένα δεδομένα αποθηκευμένα σε αυτές τις βάσεις δεδομένων.

Η δεύτερη μέθοδος διαχείρισης XML δεδομένων αφορά τη σύλληψη και διαχείριση XML δεδομένων μέσα σε υπάρχοντα μοντέλα είτε σχεσιακών [21, 22, 23, 24] είτε αντικειμενοστραφών [25, 26, 27, 28] είτε αντικειμενοσχεσιακών βάσεων δεδομένων [29, 30]. Τα XML έγγραφα έχουν από τη φύση τους μια ιεραρχική δομή που ταιριάζει καλύτερα στο αντικειμενοστραφές μοντέλο. Επίσης, οι αναφορές μεταξύ των εγγράφων ή μεταξύ στοιχείων στο εσωτερικό ενός εγγράφου, παίζουν σημαντικό ρόλο και συνδυάζονται απόλυτα με τα αντικείμενα σε ένα αντικειμενοστραφές μοντέλο. Αυτό το ταίριασμα μεταξύ του αντικειμενοστραφούς μοντέλου και του μοντέλου εγγράφων μπορεί κανείς να το συναντήσει σε έναν μεγάλο αριθμό προηγούμενων προσεγγίσεων για την αποθήκευση SGML πολυμεσικών εγγράφων σε αντικειμενοστραφείς βάσεις δεδομένων [31, 32, 33].

## 3.3 Διαχείριση XML Δεδομένων με Σχεσιακές Βάσεις Δεδομένων

Κατά την αναπαράσταση XML δεδομένων με σχέσεις, τα XML DTDs μετατρέπονται σε σχεσιακά σχήματα (relational schemata). Τα σχεσιακά σχήματα συνήθως προέρχονται από ένα μοντέλο δεδομένων όπως το γνωστό μοντέλο Οντότητας-Σχέσης (Entity-Relationship, E-R) που χρησιμοποιείται στις βάσεις δεδομένων. Η μετατροπή του E-R μοντέλου σε σχεσιακό σχήμα είναι ξεκάθαρη, γιατί υπάρχει ένας σαφής διαχωρισμός μεταξύ των οντοτήτων και των ιδιοτήτων τους. Κάθε οντότητα και οι ιδιότητές της αναπαριστώνται από μία σχέση.

Σε ένα σχεσιακό σχήμα, μία σχέση απεικονίζει ένα στοιχείο του DTD και οι ιδιότητες (attributes) της σχέσης απεικονίζουν τις ιδιότητες του στοιχείου. Ωστόσο, δεν

υπάρχει αντιστοίχιση ανάμεσα στα στοιχεία (elements) του DTD και τις οντότητες (entities) του E-R μοντέλου, καθώς επίσης ανάμεσα στις ιδιότητες των στοιχείων του DTD και τις ιδιότητες των οντοτήτων του E-R μοντέλου. Πολλές φορές, οι ιδιότητες των οντοτήτων του E-R μοντέλου αναπαριστούν ακόμη και στοιχεία του DTD. Ένα παράδειγμα για την περίπτωση αυτή φαίνεται στο Σχήμα 3.1.

```
<!ELEMENT BOOK (BOOKTITLE, AUTHOR)>
<!ELEMENT ARTICLE (TITLE, AUTHOR*, CONTACTAUTHOR?)>
<!ELEMENT CONTACTAUTHOR EMPTY>
<!ATTLIST CONTACTAUTHOR AUTHORID IDREF #IMPLIED>
<!ELEMENT MONOGRAPH (TITLE, AUTHOR, EDITOR)>
<!ELEMENT EDITOR (MONOGRAPH*)>
<!ATTLIST EDITOR NAME CDATA #REQUIRED>
<!ELEMENT AUTHOR (NAME, ADDRESS)>
<!ATTLIST AUTHOR AUTHORid ID #REQUIRED>
<!ELEMENT NAME (FIRSTNAME?, LASTNAME)>
<!ELEMENT FIRSTNAME (#PCDATA)>
<!ELEMENT LASTNAME (#PCDATA)>
<!ELEMENT ADDRESS ANY>
```

**Σχήμα 3.1: Παράδειγμα DTD**

Σε ένα E-R μοντέλο το στοιχείο `AUTHOR` του DTD θα ήταν μια οντότητα και τα στοιχεία `FIRSTNAME`, `LASTNAME` και `ADDRESS` θα ήταν οι ιδιότητες αυτής της οντότητας. Για το σχεδιασμό ενός DTD δεν υπάρχει κάποιος κανόνας που να λέει ότι η οντότητα `AUTHOR` θα αναπαρασταθεί από ένα στοιχείο και οι ιδιότητες `FIRSTNAME`, `LASTNAME` και `ADDRESS` της οντότητας από ιδιότητες του στοιχείου αυτού. Επομένως, η απευθείας αναπαράσταση των στοιχείων σε σχέσεις μπορεί να οδηγήσει σε υπερβολική κατάτμηση (fragmentation) του εγγράφου [24].

Όταν αναπαριστούμε XML δεδομένα με σχέσεις, προκύπτουν δύο βασικοί περιορισμοί. Ο πρώτος έγκειται στο ότι το σχεσιακό μοντέλο δεν υποστηρίζει ιδιότητες πολλαπλών τιμών, με αποτέλεσμα όταν ένα στοιχείο έχει ένα υποστοιχείο με τις παραστάσεις πολλαπλής εμφάνισης '*' , δηλαδή εμφάνιση του υποστοιχείου καμία ή περισσότερες φορές, ή '+', δηλαδή εμφάνιση του υποστοιχείου τουλάχιστον μία ή περισσότερες φορές, τότε το υποστοιχείο αναπαριστάται σε μια διαφορετική σχέση και η σχέση μεταξύ του στοιχείου και του υποστοιχείου αναπαριστάται με χρήση ενός ξεχωριστού κλειδιού (foreign key). Η δημιουργία ερωτήσεων και η ανακατασκευή XML εγγράφων απαιτούν τη χρήση πολλών SQL (Structured Query Language)

συνδέσεων (joins) μεταξύ των σχέσεων ανάμεσα στο στοιχείο και το υποστοιχείο, που είναι χρονοβόρα. Από την άλλη πλευρά, οι αντικειμενοστραφείς βάσεις δεδομένων υποστηρίζουν ιδιότητες πολλαπλών τιμών (σε λίστα). Με αυτό τον τρόπο, τα υποστοιχεία (ή καλύτερα οι αναφορές στα υποστοιχεία) μπορούν να αποθηκευτούν με το στοιχείο-γονέα και να ανακτηθούν με έναν μη δαπανηρό τρόπο.

Ο δεύτερος περιορισμός αφορά το ότι για την αναπαράσταση των συσχετίσεων μεταξύ στοιχείων-σχέσεων, πρέπει να δημιουργηθούν χειρωνακτικά ιδιότητες συνδέσεων (join attributes). Από την άλλη μεριά, στις αντικειμενοστραφείς βάσεις δεδομένων, οι σχέσεις μεταξύ στοιχείων-κλάσεων αναπαριστώνται στο σχήμα με ιδιότητες αναφοράς αντικειμένου (δείκτες), οι οποίες χρησιμοποιούνται για την απάντηση ερωτήσεων που περιέχουν εκφράσεις μονοπατιού.

Τέλος, ένα άλλο πρόβλημα είναι ότι οι σχέσεις αποτελούν σύνολα χωρίς κάποια σειρά μεταξύ των ιδιοτήτων τους ή των πλειάδων τους. Όμως, στα XML έγγραφα η σειρά των στοιχείων είναι σημαντική, ειδικά όταν περιέχουν πληροφορίες κειμένου (π.χ. βιβλία, άρθρα, περιεχόμενα ιστοσελίδων). Βέβαια, υπάρχουν μερικές σχεσιακές προσεγγίσεις οι οποίες κρατούν επιπλέον πληροφορίες διάταξης, ώστε να είναι ικανές να ανακατασκευάσουν τα αρχικά XML έγγραφα. Ωστόσο, αυτές οι προσεγγίσεις προσθέτουν επιπλέον πολυπλοκότητα στο σχεσιακό σχήμα, στην επεξεργασία ερωτήσεων και στους αλγορίθμους ανακατασκευής XML εγγράφων [1].

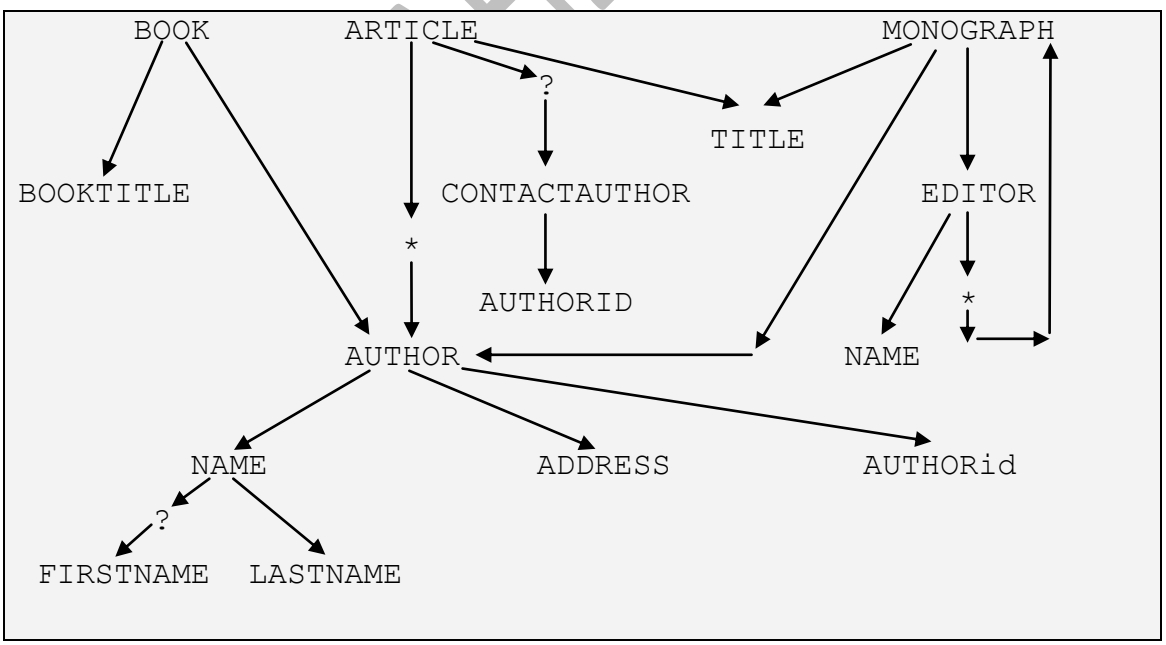
### 3.3.1 Τεχνική Βασικής Παρεμβολής (Basic Inlining)

Η *Τεχνική Βασικής Παρεμβολής* λύνει το πρόβλημα της κατάτμησης εγγράφων, παρεμβάλλοντας όσο το δυνατόν περισσότερους απογόνους ενός στοιχείου σε μια μοναδική σχέση. Αλλά, η τεχνική αυτή δημιουργεί σχέσεις για κάθε στοιχείο, γιατί ένα XML έγγραφο μπορεί να έχει ως ρίζα ένα οποιοδήποτε στοιχείο του DTD. Για παράδειγμα, το στοιχείο `AUTHOR` στο DTD που αναφέρθηκε προηγουμένως θα μπορούσε να αναπαρασταθεί με μία σχέση με ιδιότητες τα στοιχεία `FIRSTNAME`, `LASTNAME` και `ADDRESS`. Επίσης, θα μπορούσαν να δημιουργηθούν ξεχωριστές σχέσεις για τα στοιχεία `FIRSTNAME`, `LASTNAME` και `ADDRESS` [24].

Δύο προβλήματα που παρουσιάζονται στην τεχνική αυτή είναι τα ακόλουθα: η μη υποστήριξη ιδιοτήτων πολλαπλών τιμών και η αναδρομή. Στο προαναφερθέν DTD, όταν δημιουργούμε μία σχέση για το στοιχείο `ARTICLE`, δεν μπορούμε να παρεμβάλλουμε ένα σύνολο από συγγραφείς (στοιχεία `AUTHOR`), γιατί το παραδοσιακό

σχεσιακό μοντέλο δεν υποστηρίζει ιδιότητες πολλαπλών τιμών. Έτσι, ακολουθούμε τη βασική τεχνική για την αποθήκευση συνόλων μέσα σε ένα σύστημα σχεσιακής βάσης δεδομένων (Relational DataBase Management System, RDBMS), δημιουργώντας μία σχέση για το στοιχείο `AUTHOR` και συνδέοντας τους συγγραφείς με τα αντίστοιχα άρθρα (στοιχεία `ARTICLE`) χρησιμοποιώντας ένα ξεχωριστό κλειδί. Με τη χρήση της παρεμβολής ελαττώνεται σημαντικά το επίπεδο εμφώλευσης στην αναδρομή. Επομένως, η τεχνική αυτή εκφράζει την αναδρομική σχέση χρησιμοποιώντας την έννοια των σχεσιακών κλειδιών (relational keys) και χρησιμοποιεί σχεσιακή αναδρομική επεξεργασία για την ανάκτηση της σχέσης. Για να το πετύχει αυτό με ένα γενικό τρόπο, η Τεχνική Βασικής Παρεμβολής εισάγει την έννοια του *DTD γράφου (DTD graph).*

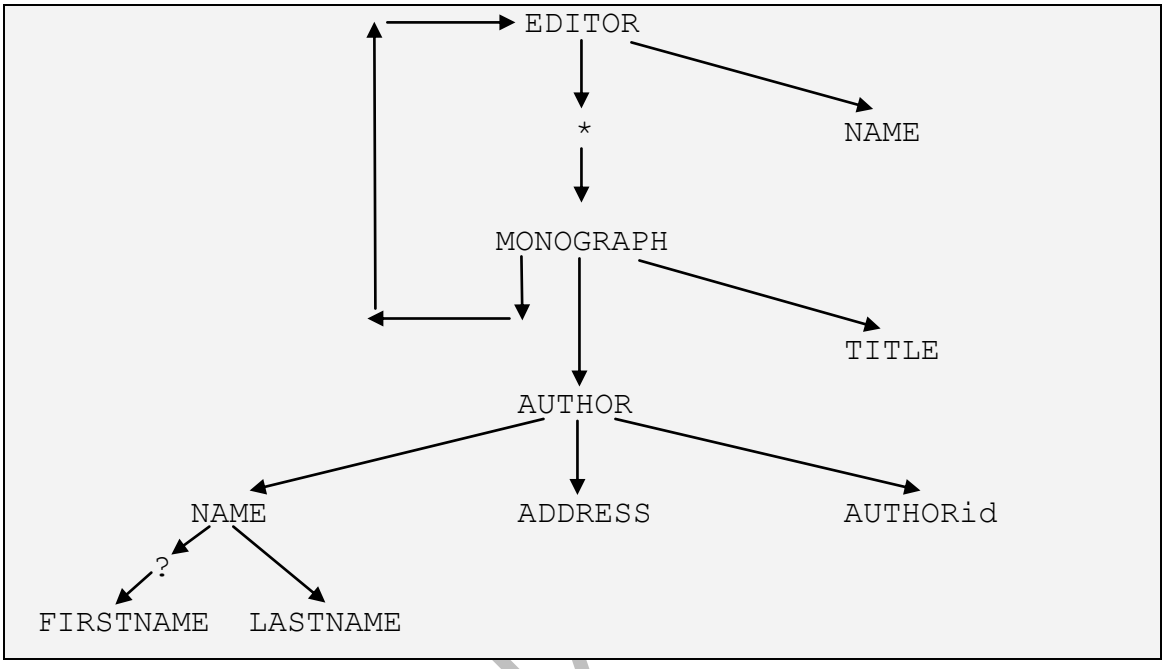
Ένας DTD γράφος αναπαριστά τη δομή ενός DTD. Οι κόμβοι του είναι στοιχεία, ιδιότητες και τελεστές του DTD. Κάθε στοιχείο εμφανίζεται ακριβώς μία φορά στο γράφο, ενώ οι ιδιότητες και οι τελεστές εμφανίζονται τόσες φορές, όσες εμφανίζονται και στο DTD. Ο DTD γράφος που αντιστοιχεί στο προαναφερθέν DTD απεικονίζεται στο Σχήμα 3.2. Οι κύκλοι στο DTD γράφο υποδηλώνουν την παρουσία αναδρομής.




**Σχήμα 3.2: Ο DTD γράφος του DTD που απεικονίζεται στο Σχήμα 3.1**

Το σχήμα που δημιουργείται για ένα DTD είναι η ένωση των συνόλων των σχέσεων που δημιουργούνται για κάθε στοιχείο. Προκειμένου να καθοριστεί το σύνολο

των σχέσεων για ένα συγκεκριμένο στοιχείο, η συγκεκριμένη τεχνική δημιουργεί μία δομή γράφου που ονομάζεται *γράφος στοιχείου (element graph).* Ο γράφος στοιχείου για το στοιχείο `EDITOR` του παραπάνω DTD γράφου φαίνεται στο Σχήμα 3.3. Ο γράφος αυτός επεκτείνει το σχετικό μέρος του DTD γράφου σε μία δενδρική δομή.




**Σχήμα 3.3: Ο γράφος του στοιχείου EDITOR του DTD που απεικονίζεται στο Σχήμα 3.1**

Με δεδομένο έναν γράφο στοιχείου, οι σχέσεις δημιουργούνται με τον ακόλουθο τρόπο. Για κάθε βασικό στοιχείο (root element) του γράφου δημιουργείται μία σχέση. Όλοι οι απόγονοι του στοιχείου παρεμβάλλονται σε αυτή τη σχέση, λαμβάνοντας υπόψιν τις εξής εξαιρέσεις: (α) για τα παιδιά που βρίσκονται ακριβώς κάτω από έναν κόμβο '*' δημιουργούνται ξεχωριστές σχέσεις (αυτό αντιστοιχεί στη δημιουργία μίας νέας σχέσης για ένα παιδί που παίρνει πολλαπλές τιμές), και (β) για κάθε κόμβο που οδηγεί σε έναν κόμβο που ο αλγόριθμος έχει ήδη επισκεφθεί δημιουργείται μία ξεχωριστή σχέση (αυτό αντιστοιχεί στη δημιουργία μιας νέας σχέσης για το χειρισμό της αναδρομής). Το Σχήμα 3.4 απεικονίζει το σχεσιακό σχήμα που παράγεται από το DTD που απεικονίζεται στο Σχήμα 3.1. Στο σχήμα αυτό παρατηρούνται μερικά χαρακτηριστικά. Το όνομα των ιδιοτήτων στις σχέσεις καθορίζεται από το μονοπάτι που ξεκινάει από το βασικό στοιχείο της σχέσης. Κάθε σχέση έχει ένα *ID* πεδίο που παίζει το ρόλο του κλειδιού στη σχέση αυτή. Όλες οι σχέσεις που αντιστοιχούν σε κόμβους στοιχείων που έχουν κάποιο γονέα,

περιλαμβάνουν ένα πεδίο parentedID που παίζει το ρόλο ενός foreign key. Για παράδειγμα, η σχέση ARTICLE.AUTHOR έχει ένα foreign key ARTICLE.AUTHOR.parentID, το οποίο συνδέει τους συγγραφείς (AUTHORS) με τα άρθρα (ARTICLES).

```
BOOK(BOOKID:integer, BOOK.BOOKTITLE:string,
     BOOK.AUTHOR.NAME.FIRSTNAME:string,
     BOOK.AUTHOR.NAME.LASTNAME:string,
     BOOK.AUTHOR.ADDRESS:string,
     BOOK.AUTHOR.AUTHORid:string)
BOOKTITLE(BOOKTITLEID:integer, BOOKTITLE:string)
ARTICLE(ARTICLEID:integer, ARTICLE.TITLE:string,
        ARTICLE.CONTACTAUTHOR.AUTHORID:string)
ARTICLE.AUTHOR(ARTICLE.AUTHORID1:integer,
               ARTICLE.AUTHOR.parentID:integer,
               ARTICLE.AUTHOR.NAME.FIRSTNAME:string,
               ARTICLE.AUTHOR.NAME.LASTNAME:string,
               ARTICLE.AUTHOR.ADDRESS:string,
               ARTICLE.AUTHOR.AUTHORid:string)
CONTACTAUTHOR(CONTACTAUTHORID:integer,
              CONTACTAUTHOR.AUTHORID:string)
TITLE(TITLEID:integer, TITLE:string)
MONOGRAPH(MONOGRAPHID:integer, MONOGRAPH.parentID:integer,
          MONOGRAPH.TITLE:string,
          MONOGRAPH.EDITOR.NAME:string,
          MONOGRAPH.AUTHOR.NAME.FIRSTNAME:string,
          MONOGRAPH.AUTHOR.NAME.LASTNAME:string,
          MONOGRAPH.AUTHOR.ADDRESS:string,
          MONOGRAPH.AUTHOR.AUTHORid:string)
EDITOR(EDITORID:integer, EDITOR.parentID:integer,
       EDITOR.NAME:string)
EDITOR.MONOGRAPH(EDITOR.MONOGRAPHID:integer,
         EDITOR.MONOGPAPH.parentID:integer,
         EDITOR.MONOGRAPH.TITLE:string,
         EDITOR.MONOGRAPH.AUTHOR.NAME.FIRSTNAME:string,
         EDITOR.MONOGRAPH.AUTHOR.NAME.LASTNAME:string,
         EDITOR.MONOGRAPH.AUTHOR.ADDRESS:string,
         EDITOR.MONOGRAPH.AUTHOR.AUTHORid:string)
AUTHOR(AUTHORID1:integer, AUTHOR.NAME.FIRSTNAME:string,
       AUTHOR.NAME.LASTNAME:string, AUTHOR.ADRRESS:string,
       AUTHOR.AUTHORid:string)
NAME(NAMEID:integer, NAME.FIRSTNAME:string,
     NAME.LASTNAME:string)
FIRSTNAME(FIRSTNAMEID:integer, FIRSTNAME:string)
LASTNAME(LASTNAMEID:integer, LASTNAME:string)
ADDRESS(ADDRESSID:integer, ADDRESS:string)
```

**Σχήμα 3.4: Το σχεσιακό σχήμα που παράγεται από το DTD που απεικονίζεται στο Σχήμα 3.1, με βάση την Τεχνική Βασικής Παρεμβολής**

Ενώ η Τεχνική Βασικής Παρεμβολής είναι καλή για συγκεκριμένους τύπους ερωτήσεων, όπως η ερώτηση "δώσε μία λίστα όλων των συγγραφέων των βιβλίων", για άλλες ερωτήσεις μπορεί να είναι αρκετά αναποτελεσματική. Για παράδειγμα, ερωτήσεις όπως η ερώτηση "δώσε μία λίστα όλων των συγγραφέων που έχουν μικρό όνομα Γιώργος" πρέπει να εκτελεστούν ως σύνδεση (join) 5 ξεχωριστών ερωτήσεων. Ένα άλλο μειονέκτημα της συγκεκριμένης τεχνικής είναι ο μεγάλος αριθμός σχέσεων που δημιουργούνται. Η επόμενη τεχνική που παρουσιάζεται προσπαθεί να λύσει αυτά τα προβλήματα.

### 3.3.2 Τεχνική Διαμοιραζόμενης Παρεμβολής (Shared Inlining)

Η Τεχνική Διαμοιραζόμενης Παρεμβολής προσπαθεί να επιλύσει τα προβλήματα που παρουσιάζει η Τεχνική Βασικής Παρεμβολής, εξασφαλίζοντας ότι ένας κόμβος στοιχείου (element node) αναπαριστάται με μία ακριβώς σχέση. Η αρχική ιδέα πίσω από αυτή την τεχνική είναι η αναγνώριση των κόμβων που αναπαριστώνται με πολλαπλές σχέσεις στην Τεχνική Βασικής Παρεμβολής (όπως τα στοιχεία `FIRSTNAME`, `LASTNAME` και `ADDRESS` στο Σχήμα 3.1), και ο διαμοιρασμός τους δημιουργώντας ξεχωριστές σχέσεις για τα στοιχεία αυτά [24].

Αρχικά πρέπει να αποφασιστεί ποιες σχέσεις πρέπει να δημιουργηθούν. Σύμφωνα με την τεχνική αυτή, δημιουργούνται σχέσεις για όλα εκείνα τα στοιχεία του DTD γράφου, των οποίων οι κόμβοι έχουν βάθος μεγαλύτερο της μονάδας. Αυτοί οι κόμβοι είναι ακριβώς εκείνοι που αναπαριστώνται ως πολλαπλές σχέσεις στην Τεχνική Βασικής Παρεμβολής. Οι κόμβοι που έχουν βάθος (βάθος ενός κόμβου είναι το μέγιστο επίπεδο των κόμβων που βρίσκονται κάτω από το συγκεκριμένο κόμβο στη δενδρική δομή) ίσο με τη μονάδα παρεμβάλλονται στους πατρικούς κόμβους. Για τους κόμβους στοιχείων που έχουν μηδενικό βάθος δημιουργούνται ξεχωριστές σχέσεις, γιατί δεν είναι προσβάσιμοι από οποιοδήποτε άλλο κόμβο. Όπως και στην Τεχνική Βασικής Παρεμβολής, για τα στοιχεία που βρίσκονται ακριβώς κάτω από έναν κόμβο '*' δημιουργούνται ξεχωριστές σχέσεις. Τέλος, για κάποιο από όλα τα αναδρομικά στοιχεία που έχουν βάθος μεγαλύτερο της μονάδας δημιουργείται μία ξεχωριστή σχέση. Μπορούμε να συναντήσουμε τέτοια αναδρομικά στοιχεία, ψάχνοντας για ισχυρά συνδεδεμένα στοιχεία στο DTD γράφο.

Αφού αποφασιστεί ποιοι κόμβοι στοιχείων θα γίνουν ξεχωριστές σχέσεις, είναι σχετικά εύκολο να κατασκευαστεί το σχεσιακό σχήμα. Κάθε κόμβος στοιχείου X που

είναι μία ξεχωριστή σχέση, παρεμβάλλει όλους τους κόμβους Υ που είναι προσβάσιμοι από αυτόν, έτσι ώστε το μονοπάτι από τον Χ στον Υ να μην περιέχει έναν κόμβο (διαφορετικό από τον Χ) που προορίζεται να γίνει μία ξεχωριστή σχέση. Το Σχήμα 3.5 δείχνει το σχήμα που παράγεται από το DTD γράφο που απεικονίζεται στο Σχήμα 3.2. Ένα αξιοσημείωτο χαρακτηριστικό αυτής της τεχνικής είναι ο μικρός αριθμός σχέσεων που δημιουργούνται, συγκριτικά με αυτόν του σχήματος που προκύπτει με τη χρήση της Τεχνικής Βασικής Παρεμβολής (Σχήμα 3.4).

Η παρεμβολή ενός στοιχείου Χ σε μία σχέση που αντιστοιχεί σε ένα άλλο στοιχείο Υ δημιουργεί προβλήματα όταν ένα XML έγγραφο έχει ως βασικό στοιχείο το στοιχείο Χ. Για τη διατύπωση ερωτήσεων σε τέτοια στοιχεία, χρησιμοποιούμε τα πεδία *ISROOT*.

Ο διαμοιρασμός στοιχείων στη συγκεκριμένη τεχνική εμφανίζει κάποιους περιορισμούς στην επεξεργασία ερωτήσεων. Από την άλλη μεριά όμως παρουσιάζει πλεονεκτήματα: για παράδειγμα, για μία ερώτηση επιλογής που αφορά όλους τους συγγραφείς χρειάζεται να προσπελαστεί μία μόνο σχέση, ενώ στην Τεχνική Βασικής Παρεμβολής χρειάζεται να προσπελαστούν 5 σχέσεις. Πέρα από το γεγονός ότι η Τεχνική Διαμοιραζόμενης Παρεμβολής παρουσιάζει κάποια από τα μειονεκτήματα της Τεχνικής Βασικής Παρεμβολής και μοιράζεται κάποια από τα πλεονεκτήματά της, η δεύτερη τεχνική αποδίδει σημαντικά καλύτερα, καθώς μειώνει τον αριθμό των συνδέσεων, ξεκινώντας από ένα συγκεκριμένο κόμβο στοιχείου. Έτσι, για να έχουμε αποτελεσματικότερη και ταχύτερη επεξεργασία ερωτήσεων, εισάγουμε μία τεχνική που συνδυάζει τη μείωση των συνδέσεων της Τεχνικής Βασικής Παρεμβολής και τα χαρακτηριστικά διαμοιρασμού της Τεχνικής Διαμοιραζόμενης Παρεμβολής. Η τεχνική αυτή ονομάζεται Τεχνική Υβριδικής Παρεμβολής.

```
BOOK(BOOKID:integer, BOOK.BOOKTITLE.ISROOT:boolean,
     BOOK.BOOKTITLE:string)
ARTICLE(ARTICLEID:integer,
        ARTICLE.CONTACTAUTHOR.ISROOT:boolean,
        ARTICLE.CONTACTAUTHOR.AUTHORID:string)
MONOGRAPH(MONOGRAPHID:integer, MONOGRAPH.parentID:integer,
          MONOGRAPH.parentCODE:integer,
          MONOGRAPH.EDITOR.ISROOT:boolean,
          MONOGRAPH.EDITOR.NAME:string)
TITLE(TITLEID:integer, TITLE.parentID:integer,
      TITLE.parentCODE:integer, TITLE:string)
AUTHOR(AUTHORID1:integer, AUTHOR.parentCODE:integer,
       AUTHOR.parentID:integer,
       AUTHOR.NAME.ISROOT:boolean,
       AUTHOR.NAME.FIRSTNAME.ISROOT:boolean,
       AUTHOR.NAME.FIRSTNAME:string,
       AUTHOR.NAME.LASTNAME.ISROOT:boolean,
       AUTHOR.NAME.LASTNAME:string,
       AUTHOR.ADDRESS.ISROOT:boolean,
       AUTHOR.ADRRESS:string, AUTHOR.AUTHORid:string)
```

**Σχήμα 3.5: Το σχεσιακό σχήμα που παράγεται από το DTD που απεικονίζεται στο Σχήμα 3.2, με βάση την Τεχνική Διαμοιραζόμενης Παρεμβολής**

### 3.3.3 Τεχνική Υβριδικής Παρεμβολής (Hybrid Inlining)

Η τεχνική αυτή είναι ίδια με την Τεχνική Διαμοιραζόμενης Παρεμβολής με τη διαφορά ότι αυτή παρεμβάλλει μερικά στοιχεία, τα οποία δεν παρεμβάλλονται στην Τεχνική Διαμοιραζόμενης Παρεμβολής. Συγκεκριμένα, η Τεχνική Υβριδικής Παρεμβολής παρεμβάλλει επιπρόσθετα στοιχεία με βάθος μεγαλύτερο της μονάδας, τα οποία δεν είναι αναδρομικά ή προσβάσιμα από έναν κόμβο '*'. Τα υποστοιχεία και τα αναδρομικά στοιχεία αντιμετωπίζονται όπως και στην Τεχνική Διαμοιραζόμενης Παρεμβολής. Το Σχήμα 3.6 δείχνει το σχεσιακό σχήμα που παράγεται από τη χρήση της τεχνικής αυτής. Παρατηρούμε ότι το σχήμα αυτό συνδυάζει χαρακτηριστικά και των δύο προηγούμενων τεχνικών, όπως για παράδειγμα το στοιχείο AUTHOR παρεμβάλλεται μέσα στα στοιχεία BOOK και MONOGRAPH παρόλο που είναι διαμοιραζόμενο, ενώ τα στοιχεία MONOGRAPH και EDITOR αναπαριστώνται ακριβώς μία φορά [24].

```
BOOK(BOOKID:integer, BOOK.BOOKTITLE.ISROOT:boolean,
     BOOK.BOOKTITLE:string, AUTHOR.NAME.FIRSTNAME:string,
     AUTHOR.NAME.LASTNAME:string, AUTHOR.ADRRESS:string,
     AUTHOR.AUTHORid:string)
ARTICLE(ARTICLEID:integer,
        ARTICLE.CONTACTAUTHOR.ISROOT:boolean,
        ARTICLE.CONTACTAUTHOR.AUTHORID:string,
        ARTICLE.TITLE.ISROOT:boolean,
        ARTICLE.TITLE:string)
MONOGRAPH(MONOGRAPHID:integer, MONOGRAPH.parentID:integer,
          MONOGRAPH.parentCODE:integer,
          MONOGRAPH.TITLE:string,
          MONOGRAPH.EDITOR.ISROOT:boolean,
          MONOGRAPH.EDITOR.NAME:string,
          AUTHOR.NAME.FIRSTNAME:string,
          AUTHOR.NAME.LASTNAME:string,
          AUTHOR.ADRRESS:string, AUTHOR.AUTHORid:string)
AUTHOR(AUTHORID1:integer, AUTHOR.parentID:integer,
       AUTHOR.parentCODE:integer,
       AUTHOR.NAME.ISROOT:boolean,
       AUTHOR.NAME.FIRSTNAME.ISROOT:boolean,
       AUTHOR.NAME.FIRSTNAME:string,
       AUTHOR.NAME.LASTNAME.ISROOT:boolean,
       AUTHOR.NAME.LASTNAME:string,
       AUTHOR.ADDRESS.ISROOT:boolean,
       AUTHOR.ADRRESS:string, AUTHOR.AUTHORid:string)
```

**Σχήμα 3.6: Το σχεσιακό σχήμα που παράγεται από το DTD που απεικονίζεται στο Σχήμα 3.1, με βάση την Τεχνική Υβριδικής Παρεμβολής**



## 3.4 Διαχείριση XML Δεδομένων με Αντικειμενοστραφείς Βάσεις Δεδομένων

Οι προσεγγίσεις για την αποθήκευση XML δεδομένων και τη διατύπωση ερωτήσεων σε αντικειμενοστραφείς βάσεις δεδομένων χειρίζονται συνήθως τους τύπους των στοιχείων ως κλάσεις και τα στοιχεία ως αντικείμενα. Οι ιδιότητες (attributes) των στοιχείων χειρίζονται ως ιδιότητες κειμένου, ενώ οι σχέσεις μεταξύ των στοιχείων και των παιδιών τους χειρίζονται ως ιδιότητες αναφοράς των αντικειμένων. Υπάρχουν πολλές παραλλαγές του σχήματος αυτού μεταξύ των διαφόρων προσεγγίσεων. Για παράδειγμα, σε μερικές περιπτώσεις [31, 28] όλα τα στοιχεία χειρίζονται ως αντικείμενα, ακόμη κι όταν το περιεχόμενό τους είναι `PCDATA`, δηλαδή αλφαριθμητικά. Ωστόσο, μια τέτοια αναπαράσταση απαιτεί πολλές κλάσεις και αντικείμενα, το οποίο οδηγεί σε σπατάλη χώρου και μείωση απόδοσης, επειδή οι ερωτήσεις πρέπει να προσπελάσουν πολύ περισσότερα αντικείμενα από όσα

πραγματικά χρειάζεται. Σε άλλες περιπτώσεις [26, 1], το πρόβλημα αυτό αντιμετωπίζεται με την αναπαράσταση των στοιχείων περιεχομένου `PCDATA` με ιδιότητες κειμένου.

Επιπρόσθετα, μερικές άλλες προσεγγίσεις [34, 29, 33] προχωρούν παραπέρα, χειριζόμενες μερικά στοιχεία με μία εσωτερική δομή ως ιδιότητες κειμένου. Η εσωτερική δομή αυτών των στοιχείων μπορεί να προσπελασθεί μέσω ειδικών μεθόδων επεξεργασίας κειμένου ικανών να χειριστούν XML δεδομένα. Η απόφαση για το ποια στοιχεία πρέπει να χειριστούν ως κλάσεις ή ιδιότητες κειμένου είτε αφήνεται στο σχεδιαστή της βάσης δεδομένων [34, 33] είτε μπορεί να παρθεί ευριστικά βάσει της συχνότητας χρήσης των δεδομένων και των στατιστικών των ερωτήσεων [29]. Αυτή η προσέγγιση μπορεί μερικές φορές να αποδειχθεί περισσότερο αποτελεσματική όσον αφορά τις απαιτήσεις σε χώρο αποθήκευσης, και γρηγορότερη όσον αφορά την επεξεργασία ερωτήσεων, λόγω της μικρότερης κατάτμησης των στοιχείων, όμως, η επεξεργασία ερωτήσεων είναι πιο σύνθετη, διότι πρέπει να χρησιμοποιηθούν διαφορετικές μέθοδοι προσπέλασης για διάφορα τμήματα των ίδιων εκφράσεων μονοπατιού (path expressions). Επιπλέον, η υλοποίηση απαιτεί την επέκταση της ίδιας της αντικειμενοστραφούς βάσης δεδομένων, για να χειριστεί τέτοιες ιδιότητες κειμένου XML και πιθανώς την επέκταση της βασικής αντικειμενοστραφούς γλώσσας ερωτήσεων, ώστε να είναι ενήμερη για αυτές τις ιδιότητες.

Η τεχνική της παρεμβολής (inlining), που αναφέρθηκε στην προηγούμενη ενότητα [24], για σχεσιακές βάσεις δεδομένων, χρησιμοποιείται για την αποφυγή δημιουργίας πολλών κλάσεων στο σχήμα. Παρόλα αυτά, η αιτιολόγηση για τη χρήση αυτής της τεχνικής στηρίζεται στο γεγονός ότι μειώνει τον αριθμό των πινάκων και των συνδέσεων μεταξύ τους, ενώ στις αντικεμενοστρεφείς βάσεις δεδομένων δεν υπάρχουν συνδέσεις και επομένως δεν υπάρχει λόγος χρήσης της τεχνικής inlining στις βάσεις αυτές. Επιπλέον, η ανάλυση των εκφράσεων μονοπατιού γίνεται σύνθετη, καθώς μερικά τμήματα του μονοπατιού αποτελούνται από απλά ονόματα ιδιοτήτων, ενώ κάποια άλλα αποτελούνται από τμήματα μονοπατιού που παρεμβάλλονται ως μία απλή ιδιότητα σε μία μεγαλύτερη κύρια κλάση [1].

### 3.4.1 Το Αντικειμενοστραφές Μοντέλο Αναπαράστασης XML Δεδομένων του X-DEVICE

Το *X-DEVICE* είναι ένα συμπερασματικό αντικεινοστραφές μοντέλο βάσης δεδομένων για τη διαχείριση XML δεδομένων. Αποτελεί μία επέκταση του υπάρχοντος αντικειμενοστραφούς συστήματος γνώσης *DEVICE* [35]. Το DEVICE ενσωματώνει δηλωτικούς κανόνες υψηλού επιπέδου (δηλαδή συμπερασματικούς κανόνες και κανόνες παραγωγής) μέσα σε μία ενεργή αντικειμενοστραφή βάση δεδομένων, η οποία υποστηρίζει μόνο κανόνες οδηγούμενους από γεγονότα (event-driven rules) [36]. Το X-DEVICE επεκτείνει το σύστημα DEVICE ενσωματώνοντας XML δεδομένα μέσα σε μία αντικειμενοστραφή βάση δεδομένων, με την αυτόματη αναπαράσταση των DTDs των XML εγγράφων σε αντικειμενικά σχήματα (object schemata), χωρίς να χάνεται η αρχική σειρά των στοιχείων του εγγράφου.

Ο ορισμός του DTD (DTD definition) μεταφράζεται σε ένα αντικειμενοστραφές σχήμα που περιλαμβάνει κλάσεις (classes) και ιδιότητες (attributes), ενώ τα XML δεδομένα μεταφράζονται σε αντικείμενα (objects) της αντικειμενοστραφούς βάσης δεδομένων. Σημειώνεται ότι όταν ένα XML έγγραφο δε συνοδεύεται από το DTD του, τότε μπορούμε να υποθέσουμε ότι ένας εμπορικός XML συντάκτης (editor) μπορεί να παράγει το DTD αυτό.

Το σύστημα X-DEVICE αναπαριστά ένα υποσύνολο των τύπων κόμβων (node types) του μοντέλου, δηλαδή κόμβων εγγράφου, στοιχείου, τιμών, ιδιοτήτων και αναφορών κόμβων. Η αναπαράσταση ενός DTD με ένα αντικειμενοστραφές μοντέλο δεδομένων πραγματοποιείται ως εξής:

- Ο τύπος κόμβου εγγράφου (document node) αναπαριστάται με μία κλάση `xml_doc` και κάθε κόμβος εγγράφου είναι ένα στιγμιότυπο αυτής της κλάσης.
- Οι κόμβοι στοιχείου (element nodes) αναπαριστώνται είτε με ιδιότητες αντικειμένου είτε με κλάσεις. Πιο συγκεκριμένα:
  - Αν ένας κόμβος στοιχείου έχει περιεχόμενο `PCDATA` (χωρίς ιδιότητες), αναπαριστάται με μια ιδιότητα της κλάσης του κόμβου στοιχείου-γονέα. Το όνομα αυτής της ιδιότητας είναι ίδιο με το όνομα του στοιχείου και ο τύπος της είναι αλφαριθμητικό (string).
  - Αν ένας κόμβος στοιχείου έχει είτε κόμβους στοιχείων-παιδιών είτε κόμβους ιδιοτήτων, αναπαριστάται με μία κλάση, η οποία είναι

στιγμιότυπο της μετα-κλάσης `xml_seq`. Οι ιδιότητες της κλάσης περιλαμβάνουν και τις ιδιότητες του στοιχείου και τους κόμβους στοιχείων-παιδιών.

- Οι κόμβοι τιμών (value nodes) αναπαριστώνται με τις τιμές των ιδιοτήτων των αντικειμένων.
- Οι κόμβοι ιδιοτήτων (attribute nodes) αναπαριστώνται με τις ιδιότητες των αντικειμένων. Οι ιδιότητες διαχωρίζονται από τα στοιχεία-παιδιά μέσω της μετα-ιδιότητας `att_lst`.
- Οι αναφορές κόμβων (node references) αναπαριστώνται ως αναφορές αντικειμένων.

Το Σχήμα 3.9 δείχνει ένα XML έγγραφο που αντιστοιχεί στο DTD που απεικονίζεται στο Σχήμα 3.7. Το Σχήμα 3.10 δείχνει πώς το έγγραφο αναπαριστάται με ένα σύνολο αντικειμένων, ενώ το Σχήμα 3.8 δείχνει το σχήμα των κλάσεων (class schema).

Τα στοιχεία σε ένα DTD μπορούν να συνδυαστούν είτε με *διαδοχή (sequencing)* είτε με *εναλλαγή (alternation)*. Διαδοχή σημαίνει ότι ένα συγκεκριμένο στοιχείο πρέπει να περιέχει όλα τα καθορισμένα στοιχεία-παιδιά του με μία συγκεκριμένη σειρά. Αυτό επιτυγχάνεται με την ύπαρξη πολλαπλών ιδιοτήτων στην κλάση που αναπαριστά το στοιχείο-γονέα, όπου κάθε ιδιότητα αντιστοιχεί σε κάθε στοιχείο-παιδί της διαδοχής. Η σειρά επιτυγχάνεται με μία τεχνική εκτός του βασικού αντικειμενοστραφούς μοντέλου βάσεων δεδομένων, με την εισαγωγή μιας μετα-ιδιότητας (meta-attribute), της `elem_ord`, που καθορίζει τη σωστή σειρά των στοιχείων-παιδιών. Αυτή η μετα-ιδιότητα χρησιμοποιείται όταν το αρχικό XML έγγραφο (είτε ολόκληρο είτε ένα μέρος αυτού) ανακατασκευάζεται και επιστρέφεται στο χρήστη.

Από την άλλη πλευρά, εναλλαγή σημαίνει ότι οποιοδήποτε από τα καθορισμένα στοιχεία-παιδιά μπορεί να περιληφθεί στο στοιχείο-γονέα. Η εναλλαγή επίσης επιτυγχάνεται με μία τεχνική εκτός του βασικού αντικειμενοστραφούς μοντέλου βάσεων δεδομένων, με τη δημιουργία μιας νέας κλάσης για κάθε εναλλαγή των στοιχείων, η οποία είναι ένα στιγμιότυπο (instance) της μετα-κλάσης `xml_alt` και παίρνει ένα μοναδικό όνομα που παράγεται από το σύστημα. Οι ιδιότητες αυτής της κλάσης καθορίζονται από τα στοιχεία που παίρνουν μέρος στην εναλλαγή. Οι τύποι των ιδιοτήτων καθορίζονται όπως και στην περίπτωση της διαδοχής. Η δομή μιας κλάσης εναλλαγής μπορεί να μοιάζει με μία κλάση διαδοχής, όμως η συμπεριφορά των

αντικειμένων εναλλαγής είναι διαφορετική, γιατί πρέπει να έχουν ακριβώς μία τιμή για καθεμία από τις ιδιότητες που καθορίζονται στην κλάση (Σχήμα 3.10).

```
<!ELEMENT BOOK    (TITLE, AUTHOR+, SECTION+)>
<!ELEMENT TITLE   (#PCDATA)>
<!ELEMENT AUTHOR  (#PCDATA)>
<!ELEMENT SECTION (TITLE, (P | FIGURE | SECTION)*)>
<!ATTLIST SECTION  id          ID      #IMPLIED
                   difficulty  CDATA   #IMPLIED>
<!ELEMENT P      (#PCDATA)>
<!ELEMENT FIGURE (TITLE, IMAGE)>
<!ATTLIST FIGURE   width       CDATA   #REQUIRED
                   height      CDATA   #REQUIRED>
<!ELEMENT IMAGE EMPTY>
<!ATTLIST IMAGE    source      CDATA   #REQUIRED>
```

**Σχήμα 3.7: Παράδειγμα DTD**

```
xml_seq BOOK
   attributes
      TITLE           (string, single, mandatory)
      AUTHOR          (string, list, mandatory)
      SECTION         (SECTION, list, mandatory)
   meta_attributes
      elem_ord        [TITLE, AUTHOR, SECTION]


xml_seq SECTION
   attributes
      TITLE           (string, single, mandatory)
      section_alt1    (section_alt1, list, optional)
      id              (string, single, optional)
      difficulty      (string, single, optional)
   meta_attributes
      elem_ord        [TITLE, section_alt1]
      att_lst         [id, difficulty]
      alias           [P-section_alt1,FIGURE-section_alt1,
                       SECTION-section_alt1]


xml_alt section_alt1
   attributes
      P               (string, single, optional)
      FIGURE          (FIGURE, single, optional)
      SECTION         (SECTION, single, optional)
```

```
xml_seq FIGURE
   attributes
      TITLE            (string, single, mandatory)
      IMAGE            (IMAGE, single, mandatory)
      width            (string, single, mandatory)
      height           (string, single, mandatory)
   meta_attributes
      elem_ord         [TITLE, IMAGE]
      att_lst          [width, height]

xml_seq IMAGE
   attributes
      source           (string, single, mandatory)
   meta_attributes
      att_lst          [source]
```

**Σχήμα 3.8: Το σχήμα των κλάσεων που αντιστοιχεί στο DTD που απεικονίζεται στο Σχήμα 3.7**

Η κλάση εναλλαγής εμφωλεύεται πάντα σε ένα στοιχείο-γονέα. Η κλάση του στοιχείου-γονέα έχει μία ιδιότητα με το όνομα της κλάσης εναλλαγής, το οποίο παράγεται από το σύστημα, και η οποία πρέπει να είναι αδιαφανής στο χρήστη για τη διατύπωση ερωτήσεων σχετικών με την κλάση. Έτσι, παρέχεται μία μετα-ιδιότητα (alias) μαζί με τα ψευδώνυμα (aliases) της ιδιότητας αυτής, π.χ. τα ονόματα των ιδιοτήτων της κλάσης εναλλαγής.

Τα στοιχεία μικτού περιεχομένου αντιμετωπίζονται παρόμοια με τα στοιχεία εναλλαγής. Η μοναδική διαφορά είναι ότι ένα από τα εναλλακτικά στοιχεία-παιδιά μπορεί επίσης να είναι απλό κείμενο (PCDATA), το οποίο επιτυγχάνεται με τη δημιουργία μιας ιδιότητας αλφαριθμητικού (string) της κλάσης εναλλαγής με το όνομα content.

Ένα άλλο θέμα που πρέπει να διευθετηθεί είναι η απεικόνιση της εμφάνισης τελεστών για τα στοιχεία, τις διαδοχές και τις εναλλαγές. Πιο συγκεκριμένα, οι τελεστές αυτοί αντιμετωπίζονται ως εξής:

- Το σύμβολο '*' μετά από ένα στοιχείο-παιδί προκαλεί τη δήλωση της αντίστοιχης ιδιότητας της κλάσης του στοιχείου-γονέα ως προαιρετικής (optional) και πολλαπλών τιμών ιδιότητας (multi-valued).
- Το σύμβολο '+' μετά από ένα στοιχείο-παιδί προκαλεί τη δήλωση της αντίστοιχης ιδιότητας της κλάσης του στοιχείου-γονέα ως υποχρεωτικής (mandatory) και πολλαπλών τιμών ιδιότητας (multi-valued).

- Το σύμβολο '?' μετά από ένα στοιχείο-παιδί προκαλεί τη δήλωση της αντίστοιχης ιδιότητας της κλάσης του στοιχείου-γονέα ως προαιρετικής (optional) και μονότιμης ιδιότητας (single-valued).
- Τέλος, η απουσία οποιουδήποτε συμβόλου σημαίνει ότι η αντίστοιχη ιδιότητα πρέπει να δηλωθεί ως υποχρεωτική (mandatory) και μονότιμη ιδιότητα (single-valued).

Η σειρά των εμφανίσεων των στοιχείων-παιδιών είναι σημαντική για τα XML έγγραφα, οπότε οι πολλαπλών τιμών ιδιότητες υλοποιούνται ως λίστες και όχι ως σύνολα.

Ο συνδυασμός των τελεστών και των εμφωλευμένων διαδοχών των στοιχείων-παιδιών απαιτεί ειδικό χειρισμό. Για παράδειγμα, θεωρούμε την παρακάτω δήλωση DTD: `<!ELEMENT a (b, (c, d)*, e)>`. Η διαδοχή `(c, d)` μπορεί να εμφανιστεί πολλές φορές μέσα στο στοιχείο `a`. Όμως, η δομή μιας κλάσης πρέπει να είναι πεπερασμένη. Αυτή η περίπτωση αντιμετωπίζεται από μια κλάση διαδοχής, παραγόμενη από το σύστημα, η οποία περιλαμβάνει τα εμφωλευμένα στοιχεία `c` και `d`. Το μοναδικό όνομα αυτής της κλάσης είναι επίσης και όνομα της ιδιότητας της κλάσης του παιδιού-γονέα. Τέλος, υπάρχει μία μετα-ιδιότητα με τα ψευδώνυμα για το όνομα της κλάσης αυτής, όπως συμβαίνει και στην περίπτωση των εμφωλευμένων στοιχείων-παιδιών εναλλαγής.

Τα κενά στοιχεία (empty elements) αντιμετωπίζονται ανάλογα με την εσωτερική τους δομή. Αν ένα κενό στοιχείο δεν έχει ιδιότητες, τότε αντιμετωπίζεται ως ένα στοιχείο `PCDATA`, δηλαδή απεικονίζεται με μία ιδιότητα αλφαριθμητικού της κλάσης του στοιχείου-γονέα. Η μοναδική τιμή που μπορεί να πάρει η ιδιότητα αυτή είναι yes, αν υπάρχει το κενό στοιχείο. Αν δεν υπάρχει το κενό στοιχείο, τότε η αντίστοιχη ιδιότητα δεν έχει τιμή. Από την άλλη πλευρά, αν ένα κενό στοιχείο έχει ιδιότητες, τότε αναπαριστάται με μία κλάση [1].

Σημειώνουμε ότι το μοντέλο αντικειμενοστραφές μοντέλο βάσης δεδομένων X-DEVICE υλοποιείται από τον αλγόριθμο που βρίσκεται στο Παράρτημα Β, ο οποίος χρησιμοποιείται για την υλοποίηση εκείνων των σταδίων που περιγράφονται στο Κεφάλαιο 5.

```
<bib>
  <BOOK YEAR="1994">
     <TITLE>TCP/IP Illustrated</TITLE>
     <AUTHOR>
          <LAST>Stevens</LAST><FIRST>W.</FIRST>
     </AUTHOR>
     <PUBLISHER>Addison-Wesley</PUBLISHER>
     <PRICE>65.95</PRICE>
 </BOOK>

 <BOOK YEAR="2000">
     <TITLE>Data on the Web</TITLE>
     <AUTHOR>
          <LAST>Abiteboul</LAST><FIRST>Serge</FIRST>
     </AUTHOR>
     <AUTHOR>
          <LAST>Buneman</LAST><FIRST>Peter</FIRST>
     </AUTHOR>
     <AUTHOR>
          <LAST>Suciu</LAST><FIRST>Dan</FIRST>
     </AUTHOR>
     <PUBLISHER>Morgan Kaufmann Publishers</PUBLISHER>
     <PRICE>39.95</PRICE>
 </BOOK>

 <BOOK YEAR="1999">
     <TITLE>The Economics of Technology and Content for
            Digital TV</TITLE>
     <EDITOR>
         <LAST>Gerbarg</LAST><FIRST>Darcy</FIRST>
         <AFFILIATION>CITI</AFFILIATION>
     </EDITOR>
     <PUBLISHER>Kluwer Academic Publishers</PUBLISHER>
     <PRICE>129.95</PRICE>
 </BOOK>
</bib>
```

**Σχήμα 3.9: Ένα δείγμα XML εγγράφου που αντιστοιχεί στο DTD που απεικονίζεται στο Σχήμα 3.7**

```
object        1#bib
  instance    bib
  attributes
     BOOK     [2#BOOK, 3#BOOK, 4#BOOK]

object        2#BOOK
  instance    BOOK
  attributes
     YEAR       '1994'
     TITLE      'TCP/IP Illustrated'
     book_alt1  [5#book_alt1]
     PUBLISHER  'Addison-Wesley'
     PRICE      '65.95'

object        3#BOOK
  instance    BOOK
  attributes
     YEAR       '2000'
     TITLE      'Data on the Web'
     book_alt1  [6#book_alt1, 7#book_alt1, 8#book_alt1]
     PUBLISHER  'Morgan Kaufmann Publishers'
     PRICE      '39.95'

object        4#BOOK
  instance    BOOK
  attributes
     YEAR       '1999'
     TITLE      'The Economics of Technology and Content
                 for Digital TV'
     book_alt1  [9#book_alt1]
     PUBLISHER  'Kluwer Academic Publishers'
     PRICE      '129.95'
```

```
object        5#book_alt1
  instance    book_alt1
  attributes
     AUTHOR     10#AUTHOR
     EDITOR     ø

object        6#book_alt1
  instance    book_alt1
  attributes
     AUTHOR     11#AUTHOR
     EDITOR     ø

object        7#book_alt1
  instance    book_alt1
  attributes
     AUTHOR     12#AUTHOR
     EDITOR     ø
```

```
object        10#AUTHOR
  instance    AUTHOR
  attributes
     LAST     'Stevens'
     FIRST    'W.'

object        11#AUTHOR
  instance    AUTHOR
  attributes
     LAST     'Abiteboul'
     FIRST    'Serge'

object        12#AUTHOR
  instance    AUTHOR
  attributes
     LAST     'Buneman'
     FIRST    'Peter'
```

```
object          8#book_alt1
  instance      book_alt1
  attributes
     AUTHOR       13#AUTHOR
     EDITOR       ø

object          9#book_alt1
  instance      book_alt1
  attributes
     AUTHOR       ø
     EDITOR       14#EDITOR
```

```
object          13#AUTHOR
  instance      AUTHOR
  attributes
     LAST       'Suciu'
     FIRST      'Dan'

object          14#EDITOR
  instance      EDITOR
  attributes
     LAST           'Gerbarg'
     FIRST          'Darcy'
     AFFILIATION    'CITI'
```

**Σχήμα 3.10: Αναπαράσταση του XML εγγράφου που απεικονίζεται στο Σχήμα 3.9 με ένα σύνολο αντικειμένων, σύμφωνα με το X-DEVICE**

# 4 ΑΝΤΙΚΕΙΜΕΝΟ-ΣΤΡΑΦΕΙΣ ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ – ΤΟ ΣΥΣΤΗΜΑ ADAM

## 4.1 Εισαγωγή

Οι αντικειμενοστραφείς βάσεις δεδομένων (Object-Oriented Databases - OODBs) αποτελούν τομή του αντικειμενοστραφούς προγραμματισμού και των συμβατικών, σχεσιακών βάσεων δεδομένων [37]. Οι OODBs μειώνουν το "σημασιολογικό χάσμα" μεταξύ των εννοιών και των οντοτήτων του πραγματικού κόσμου και των υπολογιστικών μοντέλων αναπαράστασής τους. Ακόμη παραπέρα, οι OODBs ευνοούν την αμφιμονοσήμαντη απεικόνιση του πραγματικού κόσμου στα αντικείμενα του μοντέλου και συμβάλλουν καταλυτικά στην ανάπτυξη εφαρμογών που χειρίζονται πολύπλοκες δομές δεδομένων και συσχετίσεων μεταξύ τους, όπως εφαρμογές σχεδίασης CAD/CAM, προσομοίωσης (simulation), γραφικών περιβαλλόντων διασύνδεσης χρήστη (graphical user interfaces), κλπ.

Οι αντικειμενοστραφείς βάσεις δεδομένων αποθηκεύουν στη βάση δεδομένων τα δεδομένα αλλά και τα προγράμματα χειρισμού τους. Τα πλεονεκτήματα μιας τέτοιας προσέγγισης είναι εμφανή: κοινή χρήση προγραμμάτων, επαναχρησιμοποίηση, δόμηση και ευκολία συντήρησης.

Στα συστήματα OODB τα δεδομένα απεικονίζονται με *αντικείμενα (objects)*. Τα αντικείμενα είναι ανεξάρτητες οντότητες που εγκλείουν την εσωτερική τους κατάσταση και τη συμπεριφορά τους. Η εσωτερική κατάσταση των αντικειμένων είναι ουσιαστικά τα δεδομένα μιας βάσης και είναι ανάλογα των πλειάδων (tuples). Η συμπεριφορά των αντικειμένων καθορίζεται από ένα σύνολο από διαδικασίες ή προγράμματα που επιδρούν στα δεδομένα και ονομάζονται *μέθοδοι (methods)*. Κάθε μέθοδος έχει την υπογραφή της (signature), η οποία αποτελείται από το όνομα και τον

τύπο των παραμέτρων εισόδου και εξόδου. Το σύνολο των μεθόδων ορίζει ένα περιβάλλον αλληλεπίδρασης (interface) με τα αντικείμενα της βάσης.

Τα αντικείμενα αποκρύπτουν την εσωτερική τους κατάσταση από το υπόλοιπο σύστημα. Αυτό ονομάζεται *εγκλεισμός (encapsulation)*. Το υπολογιστικό μοντέλο των συστημάτων OODB είναι η ανταλλαγή μηνυμάτων μεταξύ αντικειμένων και του χρήστη. Τα προγράμματα και οι ερωτήσεις διατυπώνονται ομοιόμορφα ως ακολουθίες μηνυμάτων. Τα *μηνύματα (messages)* ουσιαστικά είναι κλήσεις των μεθόδων, οι οποίες εκτελούνται από τα αντικείμενα που λαμβάνουν τα μηνύματα. Τα αντικείμενα που αποστέλλουν τα μηνύματα δεν γνωρίζουν και δεν ενδιαφέρονται να γνωρίζουν για τον τρόπο υλοποίησης της διαδικαστικής κλήσης (δηλ. του μηνύματος), αλλά μόνο για τη λειτουργικότητα και τα αποτελέσματα που παρέχει αυτή η κλήση.

Υπάρχουν τρία είδη αντικειμένων σε ένα σύστημα OODB:

- Τα αντικείμενα-δεδομένα, που ονομάζονται επίσης και αντικείμενα-στιγμιότυπα (instances),
- Οι κλάσεις (classes), οι οποίες είναι αφηρημένες περιγραφές της δομής και της συμπεριφοράς των αντικειμένων-στιγμιότυπων, και
- Οι μετα-κλάσεις (meta-classes), οι οποίες είναι αφηρημένες περιγραφές των κλάσεων.

Τα αντικείμενα συνδέονται μεταξύ τους με συσχετίσεις (relationships). Η πιο βασική συσχέτιση είναι η `is_instance_of` (είναι_στιγμιότυπο_του), η οποία συνδέει τα αντικείμενα-στιγμιότυπα με τις κλάσεις τους και τις κλάσεις με τις μετα-κλάσεις τους. Για παράδειγμα, ο Πίνακας 4.1 περιγράφει την κλάση `person` ως στιγμιότυπο της μετα-κλάσης `class`, η οποία είναι μια πρωταρχική μετα-κλάση του συστήματος. Το αντικείμενο `10#person` είναι ένα στιγμιότυπο της κλάσης `person`. Τα αντικείμενα προσδιορίζονται μονοσήμαντα με μία *ταυτότητα αντικειμένου (Object IDentifier - OID)*. Η ταυτότητα των κλάσεων και μετα-κλάσεων είναι το όνομά τους, ενώ η ταυτότητα των αντικειμένων-στιγμιότυπων παρέχεται αυτόματα από το σύστημα και συνήθως αποτελείται από έναν αριθμό και μία ένδειξη της κλάσης στην οποία ανήκει το αντικείμενο.

| **class** | person |
|---|---|
| is_instance_of | class |
| attributes | name(string[20]) |
| | birthdate(integer) |
| | friends(list_of person) |
| methods | get_age ::= (today()-self.birthdate)/365 |
| **instance** | 10#person |
| is_instance_of | person |
| name | 'George' |
| birthdate | 12121981 |
| friends | [11#person,22#person,33#person] |

**Πίνακας 4.1: Παράδειγμα κλάσης και αντικειμένου**

Η ταυτότητα χρησιμεύει στην αποστολή μηνυμάτων στα αντικείμενα ή στη σύνδεση αντικειμένων μέσω συσχετίσεων που ορίζονται από το χρήστη. Για παράδειγμα, το ακόλουθο μήνυμα αποστέλλεται στο αντικείμενο `10#person` και ανακαλεί την ηλικία του:

`get_age(Age)` $\Rightarrow$ `10#person` .

Το χαρακτηριστικό `friends` της κλάσης `person` είναι ένα χαρακτηριστικό πολλαπλών-τιμών (multi-valued), δηλαδή ένα χαρακτηριστικό του οποίου η τιμή δεν είναι ένα απλό στοιχείο δεδομένων, αλλά μία λίστα ή ένα σύνολο στοιχείων. Αυτή είναι μια πολύ σημαντική διαφορά των OODB με τις σχεσιακές βάσεις δεδομένων. Επιπλέον, κάθε στοιχείο αυτής της λίστας (στο συγκεκριμένο παράδειγμα) δεν είναι ένας απλός τύπος δεδομένων, π.χ. ακέραιος, αλλά μία ταυτότητα κάποιου άλλου αντικειμένου της κλάσης person. Με τον τρόπο αυτό υλοποιούνται οι συσχετίσεις που ορίζονται από το χρήστη.

Τα *σύνθετα αντικείμενα (complex objects)* είναι εκείνα τα αντικείμενα που συσχετίζονται με άλλα μέσω των συσχετίσεων που ορίζονται από το χρήστη. Τα χαρακτηριστικά που υλοποιούν αυτές τις συσχετίσεις ονομάζονται *σύνθετα χαρακτηριστικά (complex attributes)* και οι τιμές τους είναι οι ταυτότητες άλλων αντικειμένων, που ονομάζονται επίσης και δείκτες (pointers) ή δεσμοί (links). Ένα σύνθετο αντικείμενο ουσιαστικά αποτελείται από το σύνολο όλων αυτών των συσχετιζόμενων αντικειμένων και συνήθως αντιπροσωπεύεται από το αρχικό αντικείμενο, δηλαδή από το αντικείμενο από το οποίο ξεκινάει η αλυσίδα των συσχετίσεων.

Τέλος, υπάρχει η συσχέτιση `is_a` (είναι_ένα), η οποία εξειδικεύει τη δομή και τη συμπεριφορά μιας κλάσης, που ονομάζεται υπερ-κλάση (super-class), σε μία άλλη κλάση, που ονομάζεται υπο-κλάση (sub-class). Η υπο-κλάση κληρονομεί όλους τους ορισμούς χαρακτηριστικών και μεθόδων της υπερ-κλάσης, εκτός αν ρητά καθορίζεται κάτι άλλο. Η συσχέτιση `is_a` ορίζει μια ιεραρχία κληρονομικότητας ανάμεσα στις κλάσεις. Το σύνολο όλων των στιγμιότυπων μιας κλάσης είναι υποσύνολο του συνόλου των στιγμιότυπων όλων των υπερ-κλάσεών της, άμεσων και μη. Για παράδειγμα, ο Πίνακας 4.2 περιέχει τον ορισμό της κλάσης `emp`, η οποία είναι υπο-κλάση της κλάσης `person` και κληρονομεί όλα τα χαρακτηριστικά και τις μεθόδους της.

| **class** | emp |
|---|---|
| is_instance_of | class |
| is_a | person |
| attributes | manager(emp) |
| | salary(integer) |
| methods | get_tax ::= self.salary * 15% |

**Πίνακας 4.2: Παράδειγμα υπο-κλάσης**

Τα στιγμιότυπα της κλάσης `emp` μπορούν να λάβουν μηνύματα που σχετίζονται και με τις μεθόδους που ορίζονται αποκλειστικά για την κλάση `emp` αλλά και για μεθόδους των υπερ-κλάσεών της. Επίσης, μια μέθοδος που ορίζεται σε κάποια κλάση μπορεί να αντικατασταθεί (ως προς τον κώδικα - όχι ως προς την υπογραφή της) από μια μέθοδο η οποία θα είναι περισσότερο συγκεκριμένη στα αντικείμενα της υπο-κλάσης. Η αντικατάσταση μπορεί να είναι πλήρης ή απλά η υπο-κλάση να κληρονομεί και να επαυξάνει την μέθοδο της υπερ-κλάσης.

Υπάρχουν πολλά βιβλία αφιερωμένα στις OODBs, όπως τα [38, 39, 2, 40], τα οποία παρουσιάζουν το θέμα εκτενέστερα, όσον αφορά τις θεωρητικές λεπτομέρειές του (γλώσσες ερωτήσεων, αποδοτικότητα, κλπ) και περιγράφουν αρκετά εμπορικά και ερευνητικά πρωτότυπα OODB συστήματα.

## 4.2 Το Σύστημα ADAM

Το σύστημα ADAM [41] είναι ένα OODB σύστημα που αντιμετωπίζει τα δεδομένα ως αντικείμενα και τις ιδιότητές τους ως χαρακτηριστικά ή σχισμές (slots), όπως σε ένα κέλυφος εμπείρων συστημάτων που υποστηρίζει πλαίσια [42]. Το ADAM ακολουθεί

πλήρως το αντικειμενοστραφές μοντέλο προγραμματισμού και παρέχει μόνο την ανταλλαγή μηνυμάτων και την εκτέλεση μεθόδων ως μέσο επικοινωνίας μεταξύ του χρήστη και των αντικειμένων [2]. Ένα από τα κύρια χαρακτηριστικά του είναι ότι απεικονίζει όλα τα χαρακτηριστικά του και υλοποιεί όλες τις λειτουργικές του ανάγκες, όπως τα γεγονότα, τους κανόνες [43], τους περιορισμούς ακεραιότητας των δεδομένων [44], τις συσχετίσεις των δεδομένων [45], κλπ., με αντικείμενα. Το ADAM αρχικά αναπτύχθηκε με τη χρήση της Quintus Prolog και της C, στο πανεπιστήμιο του Aberdeen [41], αλλά έχει υλοποιηθεί και σε άλλες εκδόσεις της γλώσσας Prolog, όπως SICStus και ECLiPSe.

### 4.2.1 Μετα-κλάσεις (Meta-classes), Κλάσεις (Classes) και Στιγμιότυπα (Instances)

Στο σύστημα ADAM, τα αντικείμενα διαχωρίζονται σε μετα-κλάσεις, κλάσεις και στιγμιότυπα κλάσεων. Αρχικά, όταν το σύστημα δημιουργείται, υπάρχει μόνο η βασική μετα-κλάση του συστήματος, η `meta_class`. Όλες οι υπόλοιπες κλάσεις του συστήματος δημιουργούνται αποστέλλοντας μηνύματα σε μετα-κλάσεις, όπως η `meta_class`, η οποία ορίζει βασικές μεθόδους, όπως η `new` (για τη δημιουργία νέων αντικειμένων), η `put_slot` (για την εισαγωγή ενός χαρακτηριστικού σε μια κλάση), η `put_method` (για την εισαγωγή κάποιας μεθόδου σε μια κλάση), κλπ.

Τα μηνύματα αποστέλλονται με τη χρήση του τελεστή "=>". Η σύνταξη ενός μηνύματος είναι η ακόλουθη:

*Μήνυμα = <Όνομα_μεθόδου> Παράμετροι* "=>" *<Παραλήπτης_μηνύματος>*
*Παράμετροι =* "(" *Στοιχεία* ")" | [ ]
*Στοιχεία* = *<Παράμετροι_εισόδου>* "," *<Παράμετροι_εξόδου>* |
*<Παράμετροι_εισόδου>* | *<Παράμετροι_εξόδου>*

όπου *Παραλήπτης_μηνύματος* είναι η ταυτότητα ενός αντικειμένου. Εάν το σύστημα δε μπορεί να βρει κάποια μέθοδο για να ανταποκριθεί σε ένα μήνυμα που αποστέλλεται σε ένα αντικείμενο, τότε προκύπτει μήνυμα λάθους.

Τα νέα αντικείμενα (όποιου είδους και να είναι) δημιουργούνται αποστέλλοντας το μήνυμα `new` στην κλάση της οποίας στιγμιότυπο θα είναι το συγκεκριμένο αντικείμενο. Έτσι, για να δημιουργηθεί μία κλάση θα πρέπει να σταλεί ένα μήνυμα

new σε μία μετα-κλάση, το οποίο περιλαμβάνει το όνομα της κλάσης, τα χαρακτηριστικά της (όνομα, τύπος, ιδιότητες) και τις μεθόδους που υποστηρίζει. Στο Σχήμα 4.1 δίνεται ένα παράδειγμα δημιουργίας της κλάσης person (Πίνακας 4.1) στο σύστημα ADAM, η οποία είναι ένα στιγμιότυπο της μετα-κλάσης class. Σημειώνεται ότι, στο ADAM, το σώμα μιας μεθόδου αποτελείται από κώδικα στη γλώσσα Prolog.

```
:- new([person,[
     slot(slot_tuple(name,global,single,total,string)),
     slot(slot_tuple(birthdate,global,single,optional,
                     integer)),
     slot(slot_tuple(friends,global,set,optional,
                     person)),
     method((get_age(global,single,[],integer,[Age]):-
               today(Today),
               get_birthdate(BirthDate) => self,
               Age is (Today - BirthDate)/365)
      ]]) ⇒ class.
```

**Σχήμα 4.1: Δημιουργία κλάσης στο σύστημα ADAM**

Μια νέα μετα-κλάση δημιουργείται με τον ίδιο τρόπο, αποστέλλοντας δηλαδή το μήνυμα new σε μια άλλη μετα-κλάση. Επίσης, τα στιγμιότυπα των κλάσεων δημιουργούνται πάλι με την αποστολή του μηνύματος new, αλλά αντί ονόματος αντικειμένου δίνεται μια ελεύθερη μεταβλητή, η οποία ταυτοποιείται με μια ταυτότητα αντικειμένου, της μορφής 10#person, που δημιουργείται από το σύστημα. Επίσης, αντί για ορισμό χαρακτηριστικών, δίνονται τιμές σε αυτά που έχουν οριστεί στην αντίστοιχη κλάση ή στις υπερ-κλάσεις της. Αν κάποιο χαρακτηριστικό χαρακτηρίζεται ως προαιρετικό (optional), τότε δεν είναι υποχρεωτικό να δοθεί τιμή μέσα στο μήνυμα new. Στο Σχήμα 4.2 δίνεται ένα παράδειγμα δημιουργίας στιγμιότυπου της κλάσης person (Πίνακας 4.1) στο σύστημα ADAM.

```
: - new([OID,[
          name(['George']),
          birthdate([12121981]),
          friends([11#person,22#person,33#person])
       ]]) ⇒ person.
```

**Σχήμα 4.2: Δημιουργία στιγμιότυπου κλάσης στο σύστημα ADAM**

Οποιοδήποτε αντικείμενο μπορεί να διαγραφεί αποστέλλοντας το μήνυμα

```
delete => <oid>
```

όπου `oid` είναι η ταυτότητα του αντικειμένου. Για παράδειγμα, εάν θέλουμε να διαγράψουμε το αντικείμενο με την ταυτότητα `11#person`, τότε αποστέλλουμε το μήνυμα

```
delete => 11#person
```

ενώ εάν θέλουμε να διαγράψουμε την κλάση `person`, αποστέλλουμε το μήνυμα

```
delete => person.
```

Όταν διαγράφεται ένα αντικείμενο, όλες οι αναφορές σε αυτό αφαιρούνται αυτόματα.

Για να δούμε τη δομή και τις τιμές ενός αντικειμένου, αποστέλλουμε το μήνυμα

```
describe => <oid>
```

όπου `oid` είναι η ταυτότητα του αντικειμένου. Για παράδειγμα, εάν θέλουμε να δούμε τη δομή και τις τιμές του αντικειμένου με την ταυτότητα `11#person`, αποστέλλουμε το μήνυμα

```
describe => 22#person
```

ενώ εάν θέλουμε να δούμε τη δομή και τις τιμές της κλάσης `person`, αποστέλλουμε το μήνυμα

```
describe => person.
```

### 4.2.2 Μέθοδοι (Methods)

Μία μέθοδος είναι ένας κανόνας Prolog που συνδέεται με μία κλάση. Μία μέθοδος καλείται αποστέλλοντας μηνύματα σε στιγμιότυπα κλάσεων. Όταν ορίζεται μία μέθοδος, της δίδονται τα ακόλουθα στοιχεία: όνομα (name), *ορατότητα* (visibility), *πληθάριθμος* (cardinality), λίστα με τύπους ορισμάτων και τύπος αποτελέσματος.

Η ορατότητα μιας μεθόδου μπορεί να πάρει μία από τις τιμές `global`, `family` και `local`. Μια μέθοδος `global` έχει το χαρακτηριστικό ότι μπορεί να κληθεί εξωτερικά από την κλάση στην οποία έχει οριστεί. Μια μέθοδος `family` μπορεί να κληθεί το πολύ από τις υπο-κλάσεις της κλάσης στην οποία έχει οριστεί. Αντίθετα, μια μέθοδος `local` μπορεί να κληθεί μόνο από άλλες μεθόδους της ίδιας κλάσης στην οποία έχει οριστεί.

Ο πληθάριθμος μιας μεθόδου μπορεί να πάρει μία από τις τιμές `single` και `set`. Μια μέθοδος `single` επιστρέφει μία μόνο τιμή, ενώ μια μέθοδος `set` μπορεί να

επιστρέψει πολλές τιμές μέσω του μηχανισμού οπισθοδρόμησης (backtracking) της Prolog.

### 4.2.3 Εγκλεισμός (Encapsulation) - Σχισμές (Slots)

Στο σύστημα ADAM, οι ιδιότητες ενός αντικειμένου και οι σχέσεις του μεταξύ άλλων αντικειμένων αποθηκεύονται σε σχισμές (slots). Ένα slot είναι προσβάσιμο μόνο από τις μεθόδους της κλάσης στην οποία έχει οριστεί το slot και δεν κληρονομείται από μεθόδους των υπο-κλάσεών της. Αυτή η δύναμη του εγκλεισμού (encapsulation) των slots στις κλάσεις τους προωθεί την αντίληψη των αντικειμένων ως αφηρημένες οντότητες και αυξάνει την ευκολία με την οποία η αναπαράσταση ενός αντικειμένου μπορεί να αλλάξει.

Οι τελεστές για την πρόσβαση στις τιμές ενός slot και την αναβάθμισή τους είναι οι ακόλουθοι:

- `slotname ← []`

  Ο τελεστής αυτός αφαιρεί όλες τις τιμές από το slot με όνομα `slotname`.
- `slotname ← old/new`

  Ο τελεστής αυτός αντικαθιστά την παλιά τιμή `old` με τη νέα τιμή `new` στο slot με όνομα `slotname`.
- `slotname ← value`

  Ο τελεστής αυτός προσθέτει την τιμή value στο slot με όνομα `slotname`. Εάν το slot δεν μπορεί να πάρει πολλές τιμές (multi-valued) και έχει πάρει ήδη μία, τότε αποτυγχάνει η λειτουργία του τελεστή.
- `Value <= slotname`

  Με τον τελεστή αυτό επιστρέφονται όλες οι τιμές του slot με όνομα `slotname` στη μεταβλητή `Value` (μεταβλητή Prolog). Εάν το slot παίρνει πολλές τιμές, οι τιμές αυτές επιστρέφονται μία προς μία μέσω του μηχανισμού οπισθοδρόμησης της Prolog. Επίσης, εάν γίνει προσπάθεια για ανάκτηση τιμής από ένα άδειο slot, τότε αποτυγχάνει η λειτουργία του παραπάνω τελεστή.

Όταν δημιουργείται ένα slot, προστίθεται αυτόματα ένας αριθμός από μεθόδους στην κλάση στην οποία έχει οριστεί το slot, για να παρέχουν πρόσβαση στο slot. Αυτές οι μέθοδοι επιτρέπουν την ανάκτηση τιμών από slots, την προσθήκη τιμών σε slots, τη

διαγραφή τους από αυτά, όπως επίσης και την αντικατάστασή τους με άλλες τιμές. Τα ονόματα των μεθόδων αυτών είναι το όνομα του slot τους μαζί με το πρόθεμα `get_`, `put_`, `delete_` και `update_`.

Όπως και με τις μεθόδους, όταν ορίζεται ένα slot, του δίδονται τα ακόλουθα στοιχεία: όνομα (name), *ορατότητα* (visibility), *πληθάριθμος* (cardinality), *κατάσταση* (status) και τύπος αποτελέσματος.

Η ορατότητα ενός slot χρησιμοποιείται ουσιαστικά για να καθορίσει την ορατότητα των συσχετιζόμενων μεθόδων και παίρνει μία από τις τιμές `global`, `family` και `local`. Η σημασία των τιμών αυτών δόθηκε προηγουμένως.

Ο πληθάριθμος ενός slot μπορεί να πάρει μία από τις τιμές `single` και `set`. Ένα slot `single` μπορεί να πάρει μία μονό τιμή, ενώ ένα slot `set` πολλές.

Η κατάσταση ενός slot υποδηλώνει ποιες λειτουργίες αναβάθμισης μπορούν να πραγματοποιηθούν στο slot αυτό και παίρνει μία από τις τιμές `total`, `mandatory` και `optional`. Ένα slot `total` απαιτεί την ύπαρξη κάποιας τιμής (όχι `null`) σε αυτό κατά τη δημιουργία αντικειμένου και όταν η τιμή αυτή διαγράφεται, τότε διαγράφεται και το ίδιο το αντικείμενο. Ένα slot `mandatory` δεν απαιτεί την ύπαρξη κάποια τιμής κατά τη δημιουργία αντικειμένου, αλλά αν δοθεί κάποια τιμή σε αυτό, κάθε προσπάθεια διαγραφής της τιμής οδηγεί στη διαγραφή του αντικειμένου στο οποίο έχει οριστεί το slot. Αντίθετα, ένα slot `optional` δεν απαιτεί την ύπαρξη κάποια τιμής κατά τη δημιουργία αντικειμένου και η διαγραφή κάποιας τιμής του δε δημιουργεί πρόβλημα.

Ο τύπος αποτελέσματος ενός slot μπορεί να είναι είτε πραγματικός αριθμός (float) είτε ακέραιος (integer) είτε αλφαριθμητικό (string) είτε οποιοσδήποτε άλλος όρος της Prolog (plog). Αυτό που χρίζει σημασίας να τονίσουμε είναι πως ο τύπος αποτελέσματος ενός slot μπορεί να είναι το όνομα κάποιας κλάσης. Με τον τρόπο αυτό υλοποιούνται ιδιότητες αναφοράς και σχέσεις μεταξύ κλάσεων.

### 4.2.4 Κληρονομικότητα (Inheritance)

Ένα αντικείμενο μπορεί, στο σύστημα ADAM, να έχει περισσότερες από μία υπερ-κλάσεις. Αυτό το γεγονός ονομάζεται πολλαπλή κληρονομικότητα (multiple inheritance). Αν ο ορισμός μιας κλάσης δεν αναφέρει κάποια υπερ-κλάση της κλάσης, χρησιμοποιείται η εξ' ορισμού υπερ-κλάση του συστήματος `object`. Τα ονόματα των υπερ-κλάσεων μιας κλάσης αποθηκεύονται σε ένα slot του συστήματος με όνομα

`is_a` (πρόκειται για τη συσχέτιση `is_a` που αναφέραμε στην εισαγωγή του κεφαλαίου αυτού).

Ένα αντικείμενο θεωρείται ότι είναι ένα στιγμιότυπο των υπερ-κλάσεών του, αλλά και μιας πιο άμεσης κλάσης που ονομάζεται "η πιο εξειδικευμένη κλάση" (most specialized class). Η σχέση μεταξύ του αντικειμένου και της πιο εξειδικευμένης κλάσης του μπορεί να βρεθεί σε ένα slot του συστήματος με όνομα `is_instance_of` (πρόκειται για τη συσχέτιση `is_instance_of` που επίσης αναφέραμε στην εισαγωγή του κεφαλαίου αυτού). Όταν αποστέλλεται ένα μήνυμα σε ένα αντικείμενο, η αναζήτηση της μεθόδου που θα ανταποκριθεί στο μήνυμα ξεκινάει από την πιο εξειδικευμένη κλάση, της οποίας στιγμιότυπο είναι το αντικείμενο.

Οι μέθοδοι με ορατότητα `global` ή `family` κληρονομούνται από όλες τις υπο-κλάσεις της κλάσης στην οποία έχουν οριστεί. Όμως, μία υπο-κλάση μπορεί να υπερκαλύπτει τις κληρονομημένες από την υπερ-κλάση της μεθόδους, οπότε δημιουργούνται αμφιβολίες για την κλήση της κατάλληλης μεθόδου σε περίπτωση αποστολής μηνύματος. Εκείνο που είναι απαραίτητο τις περισσότερες φορές είναι η κλήση της πιο γενικής μεθόδου έναντι της πιο εξειδικευμένης. Αυτό επιτυγχάνεται ως εξής: η μέθοδος μιας κλάσης C αποστέλλει ένα μήνυμα στο άτομο `super`, οπότε και η αναζήτηση μεθόδου που ανταποκρίνεται στο μήνυμα ξεκινάει από τις υπερ-κλάσεις της C. Εάν η κλάση C έχει περισσότερες από μία υπερ-κλάσεις, τότε η αναζήτηση μεθόδου πραγματοποιείται με τη σειρά που έχει οριστεί εξαρχής.

### 4.2.5 Τύποι (Types)

Όπως αναφέραμε και παραπάνω, όλα τα slot και όλες οι μέθοδοι στο σύστημα ADAM διαθέτουν κάποιο συγκεκριμένο τύπο. Αυτό γίνεται για τους εξής λόγους:

- Για να εξασφαλίσουμε ότι τα δεδομένα που αποθηκεύονται στη βάση δεδομένων και οι αναφορές σε αυτά ακολουθούν τις προδιαγραφές του σχήματος .
- Για να έχουμε ένα καλύτερο περιβάλλον διασύνδεσης (interface).
- Για ευκολότερη και γρηγορότερη εκσφαλμάτωση (debugging) των εφαρμογών που γράφονται σε Prolog.

Πρέπει να αναφέρουμε πως υπάρχει και ένας άλλος τύπος που μπορεί να χρησιμοποιηθεί είτε για την περιγραφή ορισμάτων και αποτελεσμάτων μεθόδων είτε ως τύπος ενός slot και ονομάζεται `plog`. Κάθε δομή δεδομένων της Prolog είναι αποδεκτή ως τύπος `plog`. Ο τύπος `plog` παρέχει ουσιαστικά ένα μέσο για την αποφυγή του συστήματος τύπων του ADAM και χρησιμοποιείται από το σύστημα όταν ορίζονται μέθοδοι όπως η `new`, οι οποίες παίρνουν ως ορίσματα τυχαίους Prolog όρους για τη χρήση τους ως μεθόδους. Επίσης, αποθαρρύνεται η χρήση του τύπου `plog` στο επίπεδο στιγμιότυπων (instances).

### 4.2.6 Πλειάδες (Tuples)

Τα slots μπορούν να περιλαμβάνουν σύνθετες τιμές με εσωτερική δομή, οι οποίες ονομάζονται *πλειάδες (tuples)*. Μια πλειάδα έχει ένα όνομα και μία ή περισσότερες ιδιότητες που μπορούν να περιέχουν αριθμούς, αναφορές ή άλλες πλειάδες. Στο Σχήμα 4.3 δημιουργείται ένας νέος τύπος πλειάδας (`slot_desc_tuple`), ο οποίος χρησιμοποιείται από το σύστημα για να περιγράψει τα slots.

```
new_tuple(meta_class,
          slot_desc_tuple(
               name: string, class: string,
               visibility: string, cardinality: string,
               status: string, type: string
         ))
```

**Σχήμα 4.3: Δημιουργία του τύπου πλειάδας `slot_desc_tuple`**

Το πρώτο όρισμα του `new_tuple` υποδηλώνει το μέρος (η κλάση `meta_class`) όπου θα αποθηκευτεί ο ορισμός της πλειάδας. Οι πλειάδες μπορούν να χρησιμοποιηθούν για να αναπαραστήσουν σύνθετες τιμές ή σχέσεις [41].

### 4.2.7 Μέθοδοι Γενικής Χρήσης (Generic Methods)

Στα συστήματα που υποστηρίζουν κληρονομικότητα, μπορούν να οριστούν μέθοδοι γενικής χρήσης (generic methods), οι οποίες επιδρούν σε διαφορετικούς τύπους αντικειμένων. Για παράδειγμα, η μέθοδος `get`, στο σύστημα ADAM, χρησιμοποιείται για να επιστρέψει με επανάληψη (iteration) όλα τα στιγμιότυπα μιας κλάσης.

Από εδώ και πέρα τα παραδείγματα που θα παρουσιάσουμε αφορούν την πανεπιστημιακή βάση δεδομένων που βρίσκεται στο Παράρτημα Γ. Το παρακάτω

πρόγραμμα (Σχήμα 4.4) τυπώνει τα επίθετα όλων των φοιτητών της βάσης αυτής. Υποθέτουμε βέβαια ότι, πέρα από τις κλάσεις που αναπαριστώνται στο Παράρτημα Γ, στη βάση υπάρχουν καταχωρημένα και κάποια στιγμιότυπά τους.

```
(get(Instance) => student,
 get_sname(Name) => Instance,
 write(Name), nl,
 fail
 ; true).
```

**Σχήμα 4.4: Παράδειγμα χρήσης της μεθόδου `get` για στιγμιότυπα**

Επίσης, η μέθοδος `get` μπορεί να χρησιμοποιηθεί για να επιστρέψει με επανάληψη όλες τις κλάσεις μιας μετα-κλάσης. Για παράδειγμα, το Σχήμα 4.5 δείχνει τον τρόπο με τον οποίο η μέθοδος `get` επιστρέφει όλες τις κλάσεις της μετα-κλάσης `class`.

```
?- get(C) => class.

Αποτελέσματα
C = person ;
C = student ;
C = course ;
......
```

**Σχήμα 4.5: Παράδειγμα χρήσης της μεθόδου `get` για κλάσεις**

Πέρα από τη μέθοδο `get`, υπάρχουν και άλλες μέθοδοι, εξίσου σημαντικές, οι οποίες παράγονται αυτόματα μόλις δημιουργείται ένα slot. Οι μέθοδοι αυτές έχουν το πρόθεμα `get_by` και χρησιμοποιούνται είτε για την αναπαράσταση αντίστροφων σχέσεων μεταξύ κλάσεων είτε για την αναπαράσταση δευτερευόντων πινάκων (secondary indices). Για παράδειγμα, στην πανεπιστημιακή βάση δεδομένων του Παραρτήματος Γ, εάν θέλουμε όλους τους φοιτητές με επίθετο `Alexiadis`, καλούμε τη μέθοδο `get_by` ως εξής:

```
get_by_sname(['Alexiadis'], P) => student .
```

### 4.2.8 Δημιουργία, Περιγραφή και Διαγραφή Σχισμών και Μεθόδων (Slot and Method Creation, Description and Deletion)

Οι σχισμές και οι μέθοδοι μπορούν να προσαρτηθούν σε μια κλάση είτε όταν η κλάση δημιουργείται με το μήνυμα `new` είτε χρησιμοποιώντας τις μεθόδους `put_slot` και `put_method` αντίστοιχα. Το όρισμα της μεθόδου `put_slot` είναι μια πλειάδα τύπου `slot_desc_tuple`. Για παράδειγμα, στο Σχήμα 4.6 φαίνεται η δημιουργία του slot `length` στην κλάση `course`, το οποίο εκφράζει τη διάρκεια της σειράς μαθημάτων.

```
put_slot([
          slot_desc_tuple(length, global, single,
                          optional, integer)
         ]) => course.
```

**Σχήμα 4.6: Παράδειγμα δημιουργίας slot με τη μέθοδο `put_slot`**

Παρόμοια, το όρισμα της μεθόδου `put_method` είναι μια πλειάδα τύπου `method_desc_tuple`. Για παράδειγμα, η δημιουργία της μεθόδου `put_year` της κλάσης `student` γίνεται όπως φαίνεται στο Σχήμα 4.7. Η μέθοδος αυτή θέτει τον περιορισμό πως το έτος φοίτησης όλων των φοιτητών πρέπει να είναι μικρότερο του 5.

```
put_method([
     (put_year(global, single, [integer], [], [Year]):-
               year < 5,
               put_year([Year]) => super)
           ]) => student.
```

**Σχήμα 4.7: Παράδειγμα δημιουργίας μεθόδου με τη μέθοδο `put_method`**

Ένα slot μπορεί να διαγραφεί από μια κλάση με τη μέθοδο `delete_slot`, η οποία παίρνει το ίδιο όρισμα με τη μέθοδο `put_slot`. Για παράδειγμα, το Σχήμα 4.8 δείχνει το μήνυμα που αποστέλλεται στην κλάση `course` για να διαγραφεί το slot `length`.

```
delete_slot([
          slot_desc_tuple(length, global, single,
                          optional, integer)
            ]) => course.
```

**Σχήμα 4.8: Παράδειγμα διαγραφής slot με τη μέθοδο `delete_slot`**

Αντίστοιχα, η διαγραφή μιας μεθόδου υλοποιείται με τη μέθοδο `delete_method`, η οποία παίρνει σαν όρισμα τη μέθοδο αυτή με τύπο `method_desc_tuple`. Η διαγραφή, για παράδειγμα, της μεθόδου `put_year` γίνεται όπως φαίνεται στο Σχήμα 4.9.

```
delete_method([
          method_desc_tuple(put_year, global, single,
                            [], [integer], [])
              ]) => student.
```

**Σχήμα 4.9: Παράδειγμα διαγραφής μεθόδου με τη μέθοδο delete_slot**

Στο σύστημα ADAM, όλα τα αντικείμενα είναι στιγμιότυπα κάποιου άλλου αντικειμένου. Η μετα-κλάση του συστήματος `meta_class` διαφέρει από τις υπόλοιπες στο ότι είναι στιγμιότυπο του ίδιου του εαυτού της. Κάθε κλάση έχει εξ' ορισμού ένα slot που περιγράφει τα slots των στιγμιότυπών της. Το slot αυτό ονομάζεται `slot_desc` και περιλαμβάνει πλειάδες του τύπου `slot_desc_tuple` που μελετήσαμε προηγουμένως. Οι τιμές του `slot_desc` μπορούν να ανακτηθούν αποστέλλοντας το μήνυμα που φαίνεται στο Σχήμα 4.10 στην επιθυμητή κλάση.

```
?- get_slot_desc(D) => student.

Αποτελέσματα
D = slot_desc_tuple(fname, student, local, single, total,
string) ;
D = slot_desc_tuple(sname, student, local, single, total,
string) ;
......
```

**Σχήμα 4.10: Παράδειγμα ανάκτησης των τιμών `slot_desc` από μια κλάση**

Τέλος, μπορούμε αποστέλλοντας το μήνυμα που φαίνεται στο Σχήμα 4.11 να ανακτήσουμε τις μεθόδους που έχουν οριστεί σε μια κλάση.

```
?- get_method_desc(M) => student.

Αποτελέσματα
M = method_desc_tuple(student, delete_sname, global,
single, [string], []) ;
M = method_desc_tuple(student, get_sname, global, single,
[], string) ;
......
```

**Σχήμα 4.11: Παράδειγμα ανάκτησης των μεθόδων μιας κλάσης**

## 4.3 Παραδείγματα Ερωτημάτων (Queries)

Με βάση όλα όσα αναφέρθηκαν στο κεφάλαιο αυτό, μπορούμε να απαντήσουμε απλά και σύνθετα ερωτήματα με εύκολο και κατανοητό τρόπο. Για παράδειγμα, έστω ότι θέλουμε να ανακτήσουμε τα ονοματεπώνυμα όλων των φοιτητών που βρίσκονται στο τρίτο έτος σπουδών τους. Η απάντηση στο ερώτημα αυτό δίδεται με τη συγγραφή του κώδικα που ακολουθεί (Σχήμα 4.12).

```
( get_by_year([3], UG) => undergrad,
  get_fname(FN) => UG,
  get_sname(SN) => UG,
  write(FN), write(' '),
  write(SN), nl,
  fail
  ; true ).
```

**Σχήμα 4.12: Παράδειγμα υλοποίησης απλού ερωτήματος στο ADAM**

Ένα άλλο ερώτημα, λίγο πιο σύνθετο από το προηγούμενο, είναι το ακόλουθο: "*Ποιοι καθηγητές διδάσκουν ποια μαθήματα με κωδικό σειράς ΙΙΙΑ*". Στο ερώτημα αυτό, η απάντηση δίδεται με τη συγγραφή του κώδικα που ακολουθεί (Σχήμα 4.13).

```
( get_by_code(['IIIA'], CO) => course,
  get_by_course([CO], TCL) => tcl,
  get_section(S) => TCL,
  get_lecturer(LECT) => TCL,
  get_fname(FN) => LECT,
  get_sname(SN) => LECT,
  write(FN), write(' '),
  write(SN), write(' '),
  write(S), nl,
  fail
  ; true ).
```

**Σχήμα 4.13: Παράδειγμα υλοποίησης σύνθετου ερωτήματος στο ADAM**

Ένα τελευταίο παράδειγμα ερωτήματος στο σύστημα ADAM θα μπορούσε να είναι το εξής: "*Ποιοι φοιτητές διδάσκονται από τον καθηγητή Βλαχάβα*". Η απάντηση στο ερώτημα αυτό δίδεται με τη συγγραφή του παρακάτω κώδικα (Σχήμα 4.14).

```
( get_by_fname(['Vlahavas'], NP) => lecturer,
  get_by_lecturer([NP], TCL) => tcl,
  get_course(CO) => TCL,
  get_by_course([CO], EN) => enrollment,
  get_undergrad(UG) => EN,
  get_fname(FN) => UG,
  get_sname(SN) => UG
  write(FN), write(' '),
  write(SN), nl,
  fail
  ; true ).
```

**Σχήμα 4.14: Παράδειγμα υλοποίησης ερωτήματος στο ADAM**

Αξίζει να σημειώσουμε ότι, παρόλο που η βασική δομή του παραπάνω κώδικα είναι ξεκάθαρη και απλή, μπορεί για ερωτήματα της ίδιας κατηγορίας να μεγαλώσει αρκετά και να καταντήσει χρονοβόρα η κατασκευή της. Είναι απαραίτητο, επομένως, να μειώσουμε τη βασική δομή, με την προσθήκη νέων μεθόδων που αναπαριστούν παραγόμενες σχέσεις. Για παράδειγμα, μπορούμε, στο τελευταίο παράδειγμα, να προσθέσουμε κάποια μέθοδο στην κλάση `lecturer`, η οποία επιστρέφει όλες τις σειρές μαθημάτων που διδάσκονται από κάποιο στιγμιότυπο της κλάσης (Σχήμα 4.15).

```
put_method([
      (get_teaches(global, set, [ ], course, [C]):-
               message_recipient(MR),
               get_by_lecturer([MR],TCL) => tcl,
               get_course(C) => TCL)
          ]) => lecturer.
```

**Σχήμα 4.15: Προσθήκη της μεθόδου `get_teaches` στην κλάση `lecturer`**

Η εντολή `message_recipient` είναι μία δήλωση του συστήματος ADAM, η οποία χρησιμοποιείται μαζί με μεθόδους, για να υποδείξει το αντικείμενο που έλαβε το μήνυμα, στο οποίο η μέθοδος ανταποκρίθηκε.

Επιπρόσθετα, μπορεί να οριστεί και μία άλλη μέθοδος στην κλάση `course`, η οποία επιστρέφει όλους τους φοιτητές που έχουν εγγραφεί σε μια σειρά μαθημάτων (Σχήμα 4.16).

```
put_method([
      (get_taken_by(global, set, [], undergrad, [U]):-
               message_recipient(MR),
               get_by_course([MR], EN) => enrollment,
               get_undergrad(U) => EN)
          ]) => course.
```

**Σχήμα 4.16: Προσθήκη της μεθόδου `get_taken_by` στην κλάση `course`**

Με την προσθήκη, λοιπόν, των δύο παραπάνω μεθόδων στις αντίστοιχες κλάσεις, η ερώτηση που υλοποιείται στο Σχήμα 4.13 μπορεί να μετασχηματιστεί σε αυτήν που φαίνεται στο Σχήμα 4.17 που ακολουθεί, μειώνοντας έτσι τον αριθμό των κλήσεων που απαιτούνται για την υλοποίηση του επιθυμητού ερωτήματος.

```
( get_by_fname(['Vlahavas'], NP) => lecturer,
  get_teaches(CO) => NP,
  get_taken_by(UG) => CO,
  get_fname(FN) => UG,
  get_sname(SN) => UG
  write(FN), write(' '),
  write(SN), nl,
  fail
  ; true ).
```

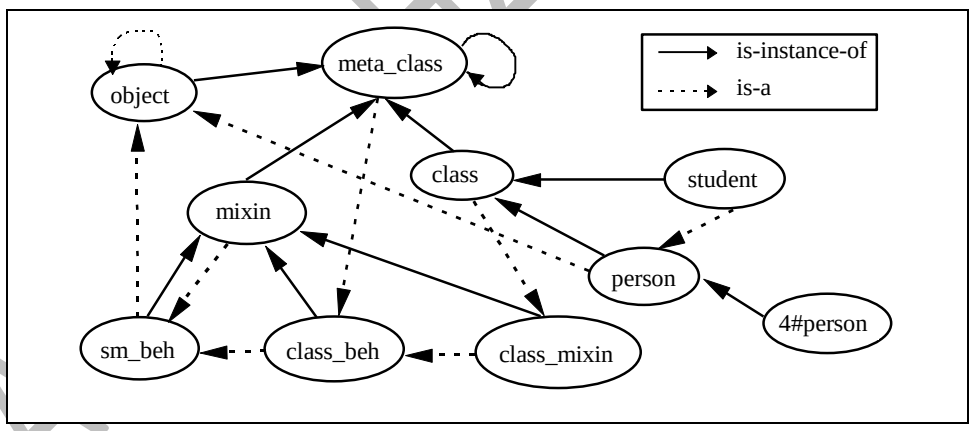
**Σχήμα 4.17: Παράδειγμα συντομότερης υλοποίησης ερωτήσεως**

**σε σχέση με αυτή στο Σχήμα 4.13**

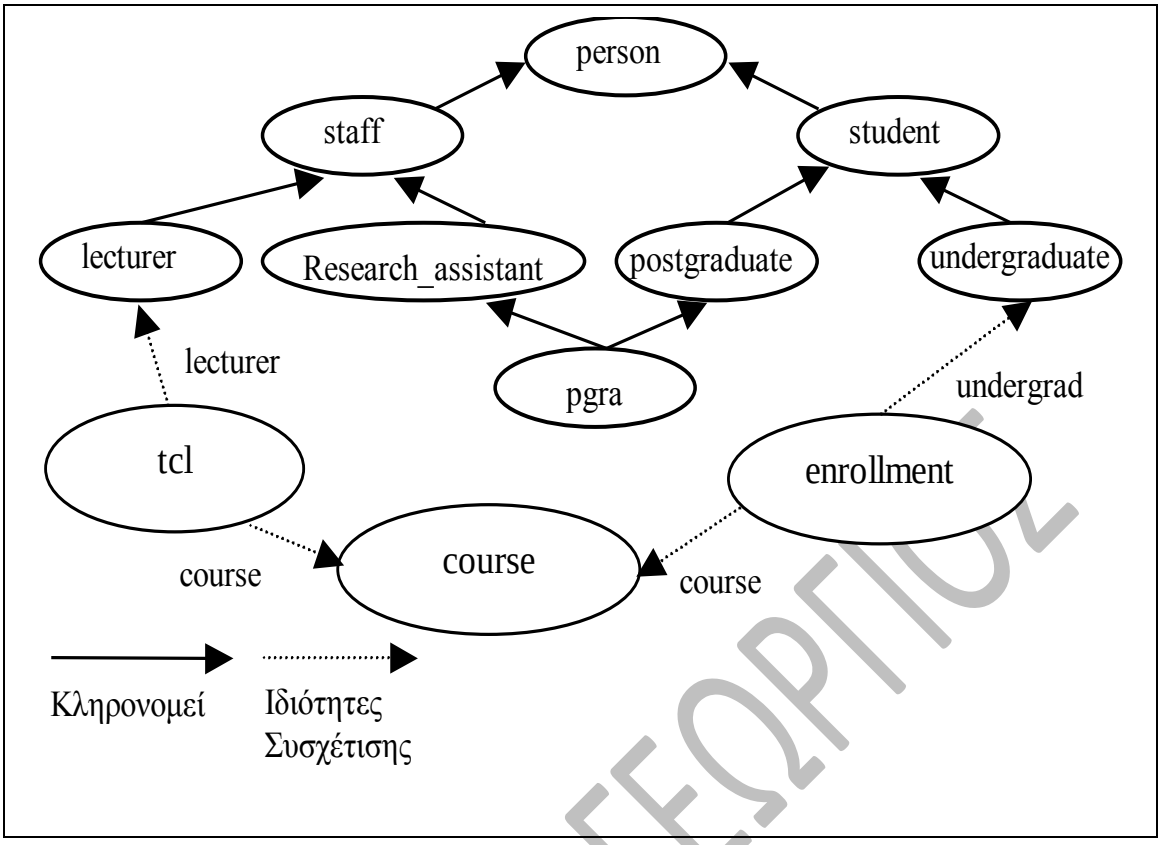
## 4.4 Η Ιεραρχία των Μετα-κλάσεων στο Σύστημα ADAM

Όπως αναφέραμε στην εισαγωγή του κεφαλαίου αυτού, μια μετα-κλάση είναι μια κλάση, της οποίας όλα τα στιγμιότυπα είναι κλάσεις. Η ιεραρχία των μετα-κλάσεων της βάσης δεδομένων του Παραρτήματος Γ αναπαριστάται στο Σχήμα 4.18, ενώ στο Σχήμα 4.19 απεικονίζεται η ιεραρχία των κλάσεων και των σχέσεων μεταξύ τους. Η κλάση `mixin` είναι μια κλάση που χρησιμοποιείται για να κρατάει τη συμπεριφορά άλλων κλάσεων, συμβάλλει στην υλοποίηση σύνθετων σχέσεων και θεωρείται πάντοτε ως υπερ-κλάση, αλλά δεν μπορεί να θεωρηθεί ως στιγμιότυπο του εαυτού της.

Για τη δημιουργία της κλάσης `person` που απεικονίζεται στο Σχήμα 4.18, πρέπει να αποσταλεί το μήνυμα `new` στη μετα-κλάση `class`. Η αναζήτηση μεθόδου που ανταποκρίνεται στο μήνυμα αυτό ξεκινάει από τη μετα-κλάση `meta_class`, δηλαδή τη μετα-κλάση του αντικειμένου (`class`) το οποίο έλαβε το μήνυμα. Αντίστοιχα, για τη δημιουργία ενός στιγμιότυπου της κλάσης `person` πρέπει να αποσταλεί το μήνυμα `new` στην κλάση `person`. Η αναζήτηση μεθόδου που ανταποκρίνεται στο μήνυμα αυτό ξεκινάει από τη μετα-κλάση της κλάσης `person`, δηλαδή τη μετα-κλάση `class`.




**Σχήμα 4.18: Η ιεραρχία των μετα-κλάσεων στο σύστημα ADAM**




**Σχήμα 4.19: Η ιεραρχία των μετα-κλάσεων στο σύστημα ADAM**

Και στις δύο περιπτώσεις αποστολής των παραπάνω μηνυμάτων, αν η κλάση (ή μετα-κλάση) δεν περιέχει τον ορισμό της μεθόδου new, τότε αυτή αναζητείται στις υπερ-κλάσεις της κλάσης (ή μετα-κλάσης) λόγω κληρονομικότητας. Έτσι, αν για παράδειγμα αποστέλλεται μήνυμα στην κλάση person και δεν υπάρχει μέθοδος `new` στη μετα-κλάση της (`class`), τότε η αναζήτηση της μεθόδου new γίνεται στην υπερ-κλάση της (`class_mixin`). Αυτή είναι, με λίγα λόγια, η διαδικασία που ακολουθείται στο σύστημα ADAM για τη δημιουργία αντικειμένων [2].

# 5 ΥΛΟΠΟΙΗΣΗ ΣΥΣΤΗΜΑΤΟΣ

## 5.1 Εισαγωγή

Το κεφάλαιο αυτό περιλαμβάνει στο πρώτο του μέρος την έννοια των Γραμματικών Οριστικών Προτάσεων (Definite Clause Grammars – DCGs), οι οποίες χρησιμοποιούνται για την υλοποίηση ενός μέρους του αντικειμένου της διπλωματικής αυτής εργασίας, που είναι η ευφυής διαχείριση XML δεδομένων με τη γλώσσα προγραμματισμού Prolog. Πιο συγκεκριμένα, οι DCGs χρησιμεύουν στο στάδιο συντακτικής ανάλυσης (parsing) των εισαγόμενων στο σύστημα αρχείων.

Το δεύτερο και βασικό μέρος του κεφαλαίου αυτού παρουσιάζει την αρχιτεκτονική του συστήματος διαχείρισης XML δεδομένων με αντικειμενοστραφείς βάσεις δεδομένων και περιγράφει αναλυτικά κάθε στάδιο που ακολουθείται για να παραχθεί το τελικό αποτέλεσμα.

Το πρόγραμμα για την αναπαράσταση των XML δεδομένων σε αντικειμενοστραφείς βάσεις δεδομένων είναι υλοποιημένο στη γλώσσα προγραμματισμού Prolog και πιο συγκεκριμένα στην πλατφόρμα ECLiPSe (ECLiPSe Common Logic Programming System) Prolog. Χρησιμοποιούμε τη συγκεκριμένη πλατφόρμα, γιατί είναι γρήγορη και αρκετά επεκτάσιμη. Μπορεί, για παράδειγμα, να ενσωματώσει δυνατότητες Προγραμματισμού βασισμένου σε περιορισμούς (Constraint Programming) και διαθέτει ακόμη δυνατότητες διασύνδεσης στο Internet. Πέρα από την Eclipse Prolog, γίνεται επίσης χρήση του συστήματος ADAM, το οποίο, όπως παρουσιάσαμε στο προηγούμενο κεφάλαιο, αποτελεί την εφαρμογή μιας αντικειμενοστραφούς βάσης δεδομένων στη γλώσσα Prolog.

## 5.2 Γραμματικές Οριστικών Προτάσεων (Definite Clause Grammars – DCGs)

Οι *Γραμματικές Οριστικών Προτάσεων (Definite Clause Grammars – DCGs)* είναι οι κατεξοχήν γραμματικές που χρησιμοποιεί η Prolog για τη διαδικασία της συντακτικής

ανάλυσης (parsing) και αποτελούν μια γενίκευση των *Γραμματικών Χωρίς Συμφραζόμενα (Context Free Grammars – CFGs)*. Γενικά, μια γραμματική ορίζεται από τα παρακάτω στοιχεία:

- Ένα σύνολο με *μη τερματικά στοιχεία (non terminals).*
- Ένα σύνολο με *τερματικά στοιχεία (terminals).*
- Ένα σύνολο *αρχικών στοιχείων*.
- Τους γραμματικούς κανόνες που χαρακτηρίζονται είτε σαν *κανόνες παραγωγής (production rules)* είτε σαν *κανόνες επαναγραφής (rewriting rules).*

Τα μη τερματικά στοιχεία μπορεί να είναι οποιοσδήποτε έγκυρος όρος Prolog, εκτός από μεταβλητή, αριθμό και αλφαριθμητικό. Τα τερματικά στοιχεία είναι όροι Prolog μέσα σε αγκύλες ("`[`", "`]`"). Το χαρακτηριστικό σύμβολο των γραμματικών κανόνων είναι το "`-->`". Όταν γίνεται συντακτική ανάλυση, κάθε γραμματικός κανόνας μετασχηματίζεται σε κλασικό κανόνα της Prolog, όπως επίσης μετασχηματίζεται κάθε όρος που βρίσκεται εκτός αγκυλών. Τα σύμβολα "`;`", "`|`", "!" και "`-->`" δε χρειάζεται να περικλείονται από αγκύλες.

Ο ορισμός των DCGs επεκτείνει το γενικό ορισμό μιας γραμματικής ως εξής:

- Τα μη τερματικά στοιχεία επεκτείνονται και σε σύνθετους όρους.
- Το δεξί μέλος ενός κανόνα μπορεί να περιέχει:
    1. Μη τερματικά στοιχεία.
    2. Λίστες τερματικών στοιχείων.
    3. Διάφορες κλήσεις διαδικασιών που περικλείονται από άγκιστρα ("`{`", "`}`").

Η τελευταία επέκταση (3) περιλαμβάνει ειδικές συνθήκες που πρέπει να πληρούνται για να εφαρμοσθεί ο γραμματικός αυτός κανόνας.

Η σύνταξη των γραμματικών κανόνων απεικονίζεται στο Σχήμα 5.1 που ακολουθεί.

```
γραμματικός_κανόνας -->  κεφαλή_κανόνα, ['-->'],
                         σώμα_κανόνα.

κεφαλή_κανόνα  -->  μη_τερματικό_στοιχείο.

σώμα_κανόνα --> σώμα_κανόνα, [','], σώμα_κανόνα.
σώμα_κανόνα --> σώμα_κανόνα, [';'], σώμα_κανόνα.
σώμα_κανόνα --> στοιχείο_σώματος_κανόνα.

στοιχείο_σώματος_κανόνα --> ['!'].
στοιχείο_σώματος_κανόνα --> ['{'], στόχοι_prolog, ['}'].
στοιχείο_σώματος_κανόνα --> μη_τερματικό_στοιχείο.
στοιχείο_σώματος_κανόνα --> τερματικό_στοιχείο.
```

**Σχήμα 5.1: Σύνταξη γραμματικών κανόνων**

Η γραμματική μπορεί να προσπελαθεί με το κατηγόρημα `phrase/3` του συστήματος. Το πρώτο όρισμα του `phrase/3` είναι το όνομα της γραμματικής που χρησιμοποιείται, ενώ το δεύτερο είναι μια λίστα με τα δεδομένα που πρόκειται να αναλυθούν συντακτικά. Εάν η συντακτική ανάλυση είναι επιτυχής, θα επιτύχει και το κατηγόρημα `phrase/3`. Για παράδειγμα, εάν είχαμε τη γραμματική:

```
a --> [] ; [b], a.
```

η κλήση `phrase(a,X,[])`, βάσει του μηχανισμού οπισθοδρόμησης (backtracking) της Prolog, θα έδινε ως αποτελέσματα τα ακόλουθα:

```
X = [b]; X = [b, b]; X = [b, b, b]; ....
```

### 5.2.1 Παράδειγμα Γραμματικής με Χρήση DCGs

Το παρακάτω παράδειγμα (Σχήμα 5.2) αναπαριστά μία απλή γραμματική που στηρίζεται στη χρήση DCGs.

```
πρόταση --> ονοματική_φράση(Αριθμός),
            ρηματική_φράση(Αριθμός).

ονοματική_φράση(Αριθμός) -->
            άρθρο(Αριθμός), ουσιαστικό(Αριθμός).
ονοματική_φράση(Αριθμός) -->
            άρθρο(Αριθμός), επίθετο(Αριθμός),
            ουσιαστικό(Αριθμός).

ρηματική_φράση(Αριθμός) --> ρήμα(Αριθμός).

άρθρο(ενικός) --> [ο].

ουσιαστικό(ενικός) --> [άνθρωπος].
ουσιαστικό(ενικός) --> [καθηγητής].
ουσιαστικό(ενικός) --> [σκύλος].

επίθετο(ενικός) --> [νέος].

ρήμα(ενικός) --> [διδάσκει].
ρήμα(ενικός) --> [ονειρεύεται].
```

**Σχήμα 5.2: Παράδειγμα γραμματικής με χρήση DCGs**

Η παραπάνω γραμματική μπορεί με επιτυχία να αναλυθεί συντακτικά καλώντας το κατηγόρημα `phrase/3`. Για παράδειγμα, οι ακόλουθες κλήσεις επιτυγχάνουν:

```
phrase(πρόταση, [ο, καθηγητής, διδάσκει], []).
phrase(πρόταση, [ο, σκύλος, ονειρεύεται], []).
```

Αντίθετα, οι κλήσεις που αποτυγχάνουν μπορεί να είναι οι εξής:

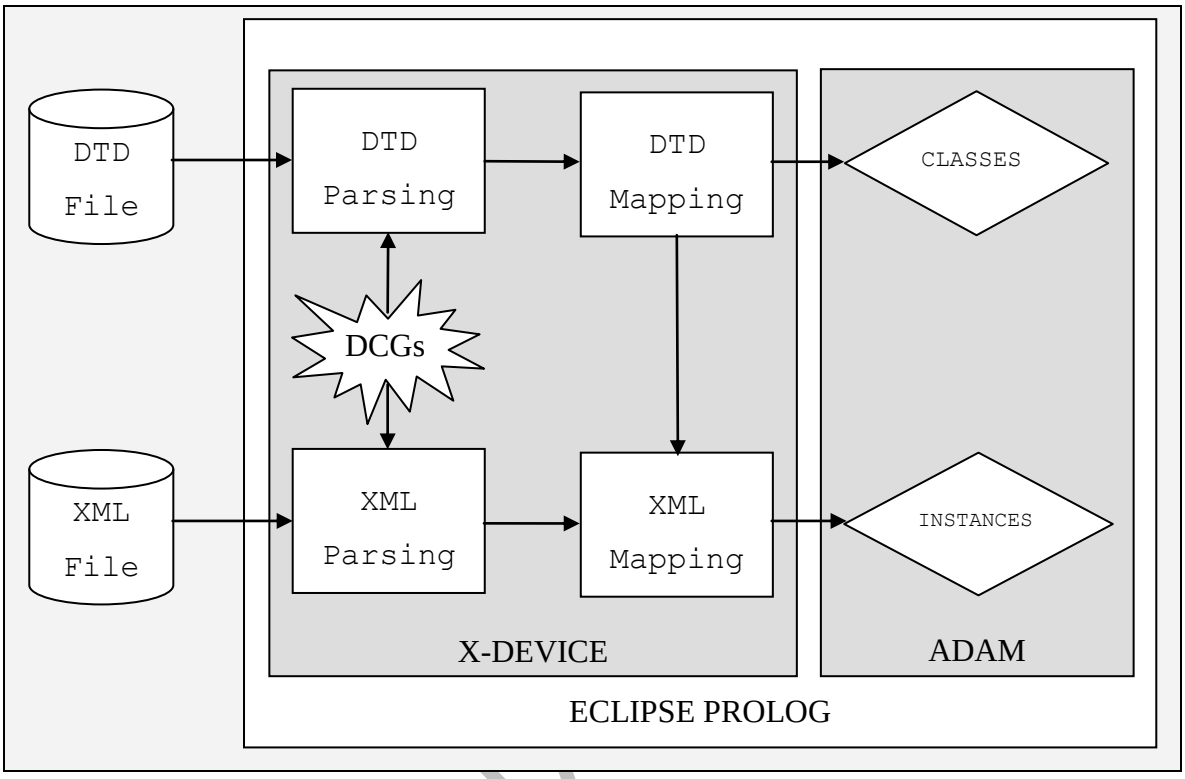
```
phrase(πρόταση, [ο, άνθρωπος, ονειρεύονται], []).
phrase(πρόταση, [οι, καθηγητές, ονειρεύονται], []).
phrase(πρόταση, [ο, σκύλος, τρώει], []).
```

## 5.3 Υλοποίηση της Διαχείρισης XML Δεδομένων με Αντικειμενοστραφείς Βάσεις Δεδομένων

Όπως αναφέραμε και στην εισαγωγή του κεφαλαίου, στο σημείο αυτό θα παρουσιάσουμε πρώτα την αρχιτεκτονική του συστήματος που υλοποιεί τη διαχείριση XML δεδομένων με αντικειμενοστραφείς βάσεις δεδομένων και έπειτα θα αναλύσουμε όλα τα στάδια που ακολουθήσαμε για την υλοποίηση αυτή, εισάγοντας αντίστοιχα τμήματα κώδικα σε Prolog που έχουν συγγραφεί για το κάθε στάδιο.

### 5.3.1 Αρχιτεκτονική Συστήματος

Η αρχιτεκτονική του συστήματος φαίνεται στο Σχήμα 5.3.




**Σχήμα 5.3: Αρχιτεκτονική συστήματος**

Σημειώνουμε ότι στη συγκεκριμένη υλοποίηση υποθέτουμε ότι το DTD είναι εξωτερικό και δεν περιέχεται μέσα στο ίδιο το XML αρχείο. Υποθέτουμε δηλαδή ότι το DTD και τα XML δεδομένα βρίσκονται σε ξεχωριστά αρχεία. Βέβαια, η άρση αυτής της υπόθεσης δεν απαιτεί σημαντικές τροποποιήσεις της εφαρμογής.

Το πρώτο στάδιο που πραγματοποιείται είναι η συντακτική ανάλυση (parsing) του αρχείου που περιλαμβάνει το DTD (DTD File). Αφού εισάγουμε, λοιπόν, το DTD αρχείο, αυτό διαβάζεται, μετατρέπεται σε ένα αλφαριθμητικό (string) και στη συνέχεια όλες οι λέξεις και τα σύμβολα του string εισάγονται σε μια λίστα ως tokens (διαδικασία tokenization). Έπειτα, καλείται το κατηγόρημα `phrase/3`, που παρουσιάσαμε προηγουμένως, με είσοδο τη λίστα αυτή. Βάσει της γραμματικής που έχει οριστεί (DCG) για το συγκεκριμένο στάδιο, το κατηγόρημα αυτό επιστρέφει σε μία λίστα τα αποτελέσματα από την κλήση του, τα οποία αναπαριστούν τα στοιχεία του DTD μαζί με τις ιδιότητές τους.

Το δεύτερο στάδιο είναι το στάδιο κατασκευής των κλάσεων. Αφού προηγηθεί το στάδιο της συντακτικής ανάλυσης του DTD αρχείου, τα στοιχεία της λίστας που παράγεται ως έξοδος ταξινομούνται κατά τέτοιο τρόπο, ώστε πρώτα να επεξεργάζονται τα στοιχεία που δεν προορίζονται να γίνουν κλάσεις, αλλά απλές ιδιότητες, και έπειτα τα στοιχεία που θα αναπαρασταθούν με κλάσεις. Έπειτα δημιουργούνται στο σύστημα ADAM οι κατάλληλες κλάσεις με τις αντίστοιχες σχισμές (slots) τους. Βέβαια, πριν από την κατασκευή των κλάσεων, επιβάλλεται η δημιουργία των απαραίτητων μετα-κλάσεων, στις οποίες θα αποσταλούν μηνύματα για τη δημιουργία των κλάσεων.

Το τρίτο στάδιο αφορά τη συντακτική ανάλυση του αρχείου που περιλαμβάνει τα XML δεδομένα (XML File). Έτσι, αφού εισάγουμε το επιθυμητό XML αρχείο, το οποίο πρέπει και να υπακούει στο DTD του, πραγματοποιείται η ίδια ακριβώς διαδικασία με εκείνη του πρώτου σταδίου, με τη διαφορά ότι χρησιμοποιείται διαφορετική γραμματική. Σημειώνεται ότι, αφού βέβαια διασφαλιστεί η συμβατότητα του XML αρχείου με τη γραμματική που ορίζει το DTD αρχείο, το στάδιο αυτό μπορεί να θεωρηθεί ότι εκτελείται παράλληλα με το πρώτο, καθώς είναι ανεξάρτητα μεταξύ τους.

Τέλος, το τέταρτο στάδιο πραγματεύεται την κατασκευή στιγμιότυπων κλάσεων. Αφού, λοιπόν, προηγηθεί το parsing του XML αρχείου, βάσει των στοιχείων της παραγόμενης λίστας και των κλάσεων που δημιουργούνται στο τρίτο στάδιο, κατασκευάζονται τα κατάλληλα στιγμιότυπα των κλάσεων αυτών. Προφανώς, δηλαδή, το στάδιο αυτό επιβάλλει πρώτα την εκτέλεση του δεύτερου σταδίου, καθώς για την κατασκευή στιγμιότυπων πρέπει να αποσταλούν μηνύματα στις κλάσεις που δημιουργούνται από το δεύτερο στάδιο.

Με την ολοκλήρωση όλων των παραπάνω σταδίων (με την προβλεπόμενη σειρά), επιτυγχάνεται η αναπαράσταση του DTD και των XML δεδομένων ως κλάσεις και αντικείμενα σε μία αντικειμενοστραφή βάση δεδομένων.

### 5.3.2 Περιγραφή της Υλοποίησης

Στο σημείο αυτό θα γίνει μία λεπτομερής ανάλυση και περιγραφή με τμήματα κώδικα σε Prolog και των τεσσάρων σταδίων που παρουσιάστηκαν προηγουμένως. Αυτό που πρέπει να διευκρινιστεί είναι πως όλοι οι κανόνες που εκτελούν τις βασικές λειτουργίες κάθε σταδίου βρίσκονται στο αρχείο "xmlmapping.pl" που εκτελεί και το τελευταίο στάδιο της υλοποίησης του προγράμματος, για λόγους αναπαράστασης.

**Στάδιο Συντακτικής Ανάλυσης DTD (DTD parsing).** Αρχικά, εισάγεται το DTD αρχείο και διαβάζεται με το κατηγόρημα `readfile/2`, το οποίο περιγράφεται στο τελευταίο στάδιο για το λόγο που αναφέραμε προηγουμένως. Το κατηγόρημα αυτό μετατρέπει το DTD σε ένα μεγάλο αλφαριθμητικό (string), το οποίο στη συνέχεια διαμερίζεται στις λέξεις και τα σύμβολα που το απαρτίζουν. Αυτό επιτυγχάνεται με το κατηγόρημα `tokenize/2` που φαίνεται στο Σχήμα 5.4.

```
tokenize([],[]):- !.
tokenize(String,[Word|Rest]):-
          grab_word(String,Chars,NewString),
          name(Word,Chars),
          tokenize(NewString,Rest).
```

**Σχήμα 5.4: Κατηγόρημα `tokenize/2`**

Το κατηγόρημα `tokenize/2` παίρνει τη λίστα με τους χαρακτήρες ASCII που αναπαριστούν το string (μεταβλητή `String`) και επιστρέφει τη λίστα `[Word|Rest]`, όπου `Word` είναι κάθε λέξη ή σύμβολο που υπάρχει στο string και `Rest` η λίστα με τις υπόλοιπες λέξεις ή σύμβολα που πρόκειται να αποσπαστούν. Ο διαχωρισμός των λέξεων και των συμβόλων πραγματοποιείται με το κατηγόρημα `grab_word/3`, δείγμα του οποίου απεικονίζεται στο Σχήμα 5.5. Το πρώτο όρισμα του κατηγορήματος αυτού είναι η λίστα με τους χαρακτήρες ASCII που αναπαριστούν το string (μεταβλητή `String`), το δεύτερο είναι η λίστα με τους χαρακτήρες ASCII (μεταβλητή `Chars`) του συμβόλου ή της λέξης που θα αποσπαστεί από το string, ενώ το τρίτο είναι το υπόλοιπο τμήμα της λίστας με τους χαρακτήρες ASCII που αναπαριστούν το string (μεταβλητή `NewString`), το οποίο απομένει ύστερα από την απόσπαση. Για παράδειγμα, στο Σχήμα 5.5 η πρώτη γραμμή μεταφράζεται ως εξής: εάν ο τελευταίος χαρακτήρας ASCII της λίστας είναι το 62, που αντιστοιχεί στο σύμβολο ">", τότε αποσπά το χαρακτήρα αυτό από τη λίστα και επιστρέφει την κενή λίστα ως το υπόλοιπο τμήμα που απομένει από την αρχική λίστα, με αποτέλεσμα να τερματίσει και ο κανόνας `tokenize/2`. Επίσης, αν παρατηρήσουμε τον πέμπτο και τον έκτο κανόνα `grab_word/3`, καταλαβαίνουμε εύκολα ότι οι κανόνες αυτοί συμβάλλουν στο να αγνοείται οποιαδήποτε ακολουθία από κενούς χαρακτήρες μέσα στο string.

```
grab_word([62],[62],[]).
grab_word([10],[],[]).
grab_word([60,33,Char|Tail],Tail1,Tail2):-
          Char \= 32,
          grab_word([60,33,32,Char|Tail],Tail1,Tail2),!.
grab_word([Char,62|Tail],Tail1,Tail2):-
          Char \= 32,
          grab_word([Char,32,62|Tail],Tail1,Tail2),!.
grab_word([13,10|Tail],[],Tail1):-
          first(X,Tail),
          X is 32,
          grab_word(Tail,[],Tail1),!.
grab_word([Char1,X,Char2|Tail],Tail1,Tail2):-
          (X is 40; X is 41; X is 42;
           X is 43; X is 44; X is 63),
          Char1 \= 32,
          Char2 is 32,
         grab_word([Char1,32,X,Char2|Tail],Tail1,Tail2),!.
grab_word([32|Tail],[],Tail1):-
          first(X,Tail),
          X is 32,
          grab_word(Tail,[],Tail1),!.
grab_word([32|Tail],[],Tail):- !.
grab_word([],[],[]).
grab_word([Char1,Char2|Tail1],[NewChar|Tail2],Rest):-
          Char1 \= 13,
          Char2 \= 40, Char2 \= 41,
          Char2 \= 42, Char2 \= 43,
          Char2 \= 44, Char2 \= 63,
          grab_word([Char2|Tail1],Tail2,Rest),!,
          lower_case(Char1,NewChar).
```

**Σχήμα 5.5: Κατηγόρημα `grab_word/3`**

Επίσης, στον τελευταίο κανόνα `grab_word/3`, ο οποίος είναι και ο βασικότερος από τους υπόλοιπους, καλείται το κατηγόρημα `lower_case/2`, το οποίο μετατρέπει τα κεφαλαία γράμματα σε μικρά (Σχήμα 5.6).

```
lower_case(Char,NewChar):-
          Char >= 65,
          Char =< 90,
          NewChar is Char + 32.
lower_case(Char,Char):-
          Char < 65.
lower_case(Char,Char):-
          Char > 90.
```

**Σχήμα 5.6: Κατηγόρημα `lower_case/2`**

Στη συνέχεια, η λίστα με τις λέξεις και τα σύμβολα που παράγεται από το κατηγόρημα `tokenize/2` υφίσταται κάποια επεξεργασία. Πρώτα αφαιρούνται όλα τα περιττά κενά ή tabs που τυχόν περιέχει, με τη χρήση του κατηγορήματος `remove_spaces/2` (Σχήμα 5.7).

```
remove_spaces([],[]).
remove_spaces([A|R],List):-
          (A == ''; A == '\t'),
          remove_spaces(R,List),!.
remove_spaces([A|R],[A|List]):-
          remove_spaces(R,List),!.
```

**Σχήμα 5.7: Κατηγόρημα `remove_spaces/2`**

Έπειτα, χρησιμοποιείται το κατηγόρημα `add_null/2`, το οποίο παίρνει σαν είσοδο τη λίστα που επιστρέφει το κατηγόρημα `remove_spaces/2` και εισάγει την τιμή "`null`" σε εκείνα τα σημεία της λίστας όπου δεν έχει δοθεί κάποια από τις τιμές "`?`", "`+`" και "`*`" (Σχήμα 5.8). Αυτό γίνεται προκειμένου να διευκολυνθεί η επεξεργασία της λίστας από τους υπόλοιπους κανόνες του προγράμματος.

Στην περίπτωση που ένα στοιχείο αποτελείται από ένα μόνο στοιχείο μέσα σε παρενθέσεις και υπάρχει ένας από τους χαρακτήρες "`?`", "`+`" και "`*`"μετά τη δεξιά παρένθεση, χρησιμοποιείται το επόμενο κατηγόρημα `change_null/2`, το οποίο, για λόγους αναπαράστασης, αντικαθιστά την τιμή `null` που είχε δοθεί στο στοιχείο με την τιμή που βρίσκεται μετά τη δεξιά παρένθεση (Σχήμα 5.9). Για παράδειγμα, εάν είχαμε στο DTD τη δήλωση του στοιχείου `BIB`: `<!ELEMENT BIB (BOOK)*>`, για το οποίο υποθέτουμε πως δεν έχει ιδιότητες, τότε η συντακτική ανάλυση θα μετέτρεπε το στοιχείο `BIB` στη μορφή: `element(bib, child([book/(null)])/(*), [])`. Αντίθετα, εάν είχαμε τη δήλωση: `<!ELEMENT BIB (BOOK*)>`, η συντακτική ανάλυση θα έδινε: `element(bib, child([book/(*)])/(null), [])`. Αυτό που επιθυμούμε είναι, στην περίπτωση που η τιμή του στοιχείου-παιδιού μέσα στη λίστα `child` είναι `null` και έξω από τη λίστα `child` διαφορετική της τιμής `null`, να αλλάζει η πρώτη τιμή και να γίνεται ίση με τη δεύτερη. Δηλαδή, στην πρώτη παραπάνω περίπτωση, η τιμή `null` που βρίσκεται στη λίστα `child`, εφόσον η τιμή έξω από τη λίστα `child (*)` είναι διαφορετική της `null`, θα αλλάξει με τη βοήθεια του κατηγορήματος `change_null/2` και θα γίνει ίση με την τιμή `(*)`.

```
add_null([],[]).
add_null([A,B,A1,A2,A3,A4,C|D1],[A,B,A1,A2,A3,A4,C|D2]):-
          A == '<!' ,
          B == attlist,
          C == > ,
          add_null(D1,D2),!.
add_null([A,B,A1,A2,A3,A4,A5,C|D1],
         [A,B,A1,A2,A3,A4,A5,C|D2]):-
          A == '<!' ,
          B == attlist,
          C == > ,
          add_null(D1,D2),!.
add_null([A,B|Tail],[A,null,B|Tail1]):-
          (((B == ',' ; B == ')' ),
            A \= '?' , A \= '+' ,
            A \= '*' , A \= '#pcdata' )
           ;
           (A == ')' ,
            B \= '?' , B \= '+' , B \= '*' )
           ;
           (B == > ,
            A \= '?' , A \= '+' , A \= '*' )),
          add_null([B|Tail],[B|Tail1]),!.
add_null([A|Tail],[A|Tail1]):- add_null(Tail,Tail1).
```

**Σχήμα 5.8: Κατηγόρημα `add_null/2`**

```
change_null([],[]).
change_null([A,B,(C),D|Tail],[A,B,(C),D|Tail1]):-
          A == '(',
          C \= null,
          D == ')',
          change_null(Tail,Tail1),!.
change_null([A,B,(C),D,E|Tail],[A,B,E,D,E|Tail1]):-
          A == '(',
          C == null,
          D == ')',
          change_null(Tail,Tail1),!.
change_null([A|Tail],[A|Tail1]):- change_null(Tail,Tail1).
```

**Σχήμα 5.9: Κατηγόρημα `change_null/2`**

Μετά από την επεξεργασία της παραγόμενης από τη διαδικασία tokenization λίστας, καλείται μέσω του κατηγορήματος `pass_list/2` (Σχήμα 5.10) το κατηγόρημα `phrase/3` του συστήματος με δεύτερο όρισμα τη λίστα αυτή. Αυτό που επιστρέφει το κατηγόρημα `phrase/3` είναι μια λίστα, της οποίας κάθε στοιχείο έχει τη μορφή `element_declaration(E)`, που σημαίνει ότι ικανοποιείται ο

αντίστοιχος γραμματικός κανόνας. Οι γραμματικοί κανόνες `element_declaration/1` απεικονίζονται στο Σχήμα 5.11.

```
pass_list(B,S):-
          phrase(element_declaration(S),B,[]).
```

**Σχήμα 5.10: Κατηγόρημα `pass_list/2`**

```
element_declaration([element(Name,Content,Attributes)])
     -->  start_symbol,
          element_word,
          element_name(Name),
          element_content(Content),
          end_symbol,
          attribute_declaration(Name,Attributes),!.
element_declaration([element(Name,Content,[])]) -->
          start_symbol,
          element_word,
          element_name(Name),
          element_content(Content),
          end_symbol.
element_declaration([element(Name,Content,Attributes)|
                     Tail]) -->
          start_symbol,
          element_word,
          element_name(Name),
          element_content(Content),
          end_symbol,
          attribute_declaration(Name,Attributes),!,
          element_declaration(Tail).
element_declaration([element(Name,Content,[])|Tail]) -->
          start_symbol,
          element_word,
          element_name(Name),
          element_content(Content),
          end_symbol,
          element_declaration(Tail).
```

**Σχήμα 5.11: Κατηγόρημα `element_declaration/1`**

Τόσο ο πρώτος όσο και ο τρίτος κανόνας `element_declaration/1` αναλύουν την περίπτωση που ένα στοιχείο έχει αντίστοιχα 1 ή περισσότερες ιδιότητες (attributes), ενώ ο δεύτερος και ο τέταρτος κανόνας αναλύουν την περίπτωση που ένα στοιχείο δεν έχει καμία ιδιότητα. Το κατηγόρημα `element_content/1` (Σχήμα 5.12) ενεργοποιεί, βάσει του περιεχομένου του τρέχοντος στοιχείου, τον αντίστοιχο

γραμματικό κανόνα. Το περιεχόμενο ενός στοιχείου μπορεί να είναι είτε #PCDATA, οπότε καλείται το κατηγόρημα pcdata/1, είτε EMPTY, οπότε καλείται το κατηγόρημα empty/1, είτε ANY, οπότε καλείται το κατηγόρημα any/1, είτε άλλα στοιχεία, περίπτωση στην οποία καλείται το κατηγόρημα children/1.

```
element_content(Content) -->
          empty(Content);
          any(Content);
          pcdata(Content);
          children(Content).
```

**Σχήμα 5.12: Κατηγόρημα element_content/1**

Το κατηγόρημα attribute_declaration/2, που εμφανίζεται στους κανόνες element_declaration/1 στο Σχήμα 5.11, χρησιμοποιείται στην περίπτωση που ένα στοιχείο έχει ιδιότητες και περιγράφεται στο Σχήμα 5.13.

```
attribute_declaration(Element,[Att|RestAtt]) -->
          start_symbol,
          attribute_word,
          attribute_name(Element),
          attribute_content(Att),
          end_symbol,
          attribute_declaration(Element,RestAtt),!.
attribute_declaration(Element,[Att]) -->
          start_symbol,
          attribute_word,
          attribute_name(Element),
          attribute_content(Att),
          end_symbol.
```

**Σχήμα 5.13: Κατηγόρημα attribute_declaration/2**

Έστω, για παράδειγμα ότι έχουμε ένα DTD που περιλαμβάνει τις δηλώσεις:

```
<!ELEMENT  AUTHOR  (LAST, FIRST)>
<!ELEMENT  FIRST   (#PCDATA)>
<!ELEMENT  LAST    (#PCDATA)>
```

Ο γραμματικός κανόνας element_declaration/1 που θα επιτύχει για την ανάλυση του στοιχείου AUTHOR είναι ο τελευταίος στο Σχήμα 5.11, καθώς το στοιχείο αυτό δεν έχει ιδιότητες και δεν είναι το τελευταίο στοιχείο που αναλύεται. Το περιεχόμενο του στοιχείου επιστρέφεται με την κλήση του κατηγορήματος element_content/1 (Σχήμα 5.12), και επειδή το στοιχείο AUTHOR έχει στοιχεία-παιδιά, καλείται με τη σειρά του το κατηγόρημα children/1, το οποίο στην περίπτωσή μας θα επιστρέψει μια λίστα, έστω List, με τα στοιχεία-παιδιά. Έτσι, το αποτέλεσμα που θα πάρουμε για το στοιχείο AUTHOR είναι της μορφής:

```
element(author,
        child([first/(null),last/(null)])/(null),
        []).
```

Με τον ίδιο τρόπο, με τη διαφορά ότι καλείται το κατηγόρημα pcdata/1 αντί για το κατηγόρημα children/1, για τα στοιχεία FIRST και LAST θα πάρουμε τα εξής αποτελέσματα:

```
element(first, '#pcdata', []) και
element(last, '#pcdata', []) .
```

Όλα τα βήματα που παρουσιάσαμε για την υλοποίηση του σταδίου συντακτικής ανάλυσης ενός DTD αρχείου συνοψίζονται στο βασικό κατηγόρημα dtd_parsing/2 που φαίνεται στο Σχήμα 5.14.

```
dtd_parsing(DtDFile,List):-
          readfile(DtDFile,A),
          tokenize(A,B),
          remove_spaces(B,BN),
          add_null(BN,C),
          change_null(C,CC),
          pass_list(CC,List).
```

**Σχήμα 5.14: Κατηγόρημα dtd_parsing/2**

Σε αυτό το σημείο πρέπει να διευκρινίσουμε ότι το πρόγραμμα που υλοποιεί το στάδιο συντακτικής ανάλυσης ενός DTD αρχείου θέτει κάποιους περιορισμούς στη μορφή του DTD αρχείου, προκειμένου να επιτύχει η συντακτική ανάλυση. Οι περιορισμοί αυτοί είναι οι εξής:

- Η δήλωση `#pcdata` πρέπει να δηλώνεται σαν μία λέξη (χωρίς κενά) και να περικλείεται μέσα σε παρενθέσεις.
- Τα σύμβολα έναρξης μιας δήλωσης "`<!`" δεν πρέπει να διαχωρίζονται με κενά, ώστε να θεωρούνται ουσιαστικά ως ένα σύμβολο.
- Κάθε ιδιότητα στοιχείου πρέπει να δηλώνεται αμέσως μετά τη δήλωση του στοιχείου.
- Για περισσότερες από 1 ιδιότητες, πρέπει να υπάρχει ξεχωριστή δήλωση "`<!ATTLIST …>`" για την καθεμιά.
- Τα στοιχεία που έχουν περιεχόμενο `#pcdata` δεν έχουν ιδιότητες.
- Δεν υποστηρίζονται `entities`, `nmtokens` και σχόλια.
- Δεν υποστηρίζονται οι περιπτώσεις τιμών `optional(Default)` και `mandatory(Default)`.
- Δεν υποστηρίζεται εναλλαγή στα στοιχεία (alternation).

**Στάδιο Αναπαράστασης DTD με Κλάσεις (DTD mapping).** Αρχικά, δημιουργούνται οι απαραίτητες μετα-κλάσεις, στις οποίες θα αποσταλούν αργότερα μηνύματα για τη δημιουργία κλάσεων. Οι μετα-κλάσεις αυτές παρουσιάζονται στο Σχήμα 5.15.

```
:- new([xml_doc,[
     slot(slot_tuple(dtd_uri,global,single,total,string)),
     slot(slot_tuple(xml_uri,global,single,total,string)),
     slot(slot_tuple(children,global,set,total,plog))
      ]]) => class.

:- new([xml_elem,[
     is_a([class]),
     slot(slot_tuple(alias,global,set,optional,plog)),
     slot(slot_tuple(empty,global,set,optional,string))
      ]]) => meta_class.

:- new([xml_seq,[
     is_a([xml_elem]),
     slot(slot_tuple(elem_ord,global,single,optional,plog)),
     slot(slot_tuple(att_lst,global,set,optional,string))
      ]]) => meta_class.

:- new([xml_alt,[
     is_a([xml_elem])
      ]]) => meta_class.
```

**Σχήμα 5.15: Δημιουργία μετα-κλάσεων**

Η λίστα που παράγεται από το προηγούμενο στάδιο (DTD parsing) ταξινομείται πρώτα κατάλληλα και έπειτα παρέχεται για τη δημιουργία των κατάλληλων κλάσεων. Η ταξινόμηση που υφίσταται περιλαμβάνει 3 φάσεις. Στην πρώτη φάση, όλα τα στοιχεία της λίστας που έχουν περιεχόμενο `'#pcdata'` ή `empty` τοποθετούνται στην αρχή της λίστας. Αυτό επιτυγχάνεται με το κατηγόρημα `put_first_pcdemp/4` (Σχήμα 5.16).

```
put_first_pcdemp([],[],[],[]).
put_first_pcdemp([element(Element,'#pcdata'/(Sign),
                          Attributes)|R],
                 [element(Element,'#pcdata'/(Sign),
                          Attributes)|PcdList],
                 EmptyList,RestList):-
     put_first_pcdemp(R,PcdList,EmptyList,RestList),!.
put_first_pcdemp([element(Element,child([empty/(Sign)])
                          /(Sign1),Attributes)|R],PcdList,
                 [element(Element,child([empty/(Sign)])
                          /(Sign1),Attributes)|EmptyList],
                 RestList):-
     put_first_pcdemp(R,PcdList,EmptyList,RestList),!.
put_first_pcdemp([element(Element,child(A)/(Sign),
                          Attributes)|R],
                 PcdList,EmptyList,
                 [element(Element,child(A)/(Sign),
                          Attributes)|RestList]):-
     put_first_pcdemp(R,PcdList,EmptyList,RestList),!.
```

**Σχήμα 5.16: Κατηγόρημα `put_first_pcdemp/4`**

Η δεύτερη φάση ταξινόμησης έχει να κάνει με τα στοιχεία εκείνα, των οποίων τα στοιχεία-παιδιά έχουν όλα περιεχόμενο `'#pcdata'` ή `empty`, και όχι κάποιο άλλο στοιχείο-παιδί. Τα στοιχεία αυτά, λοιπόν, τοποθετούνται στη λίστα αμέσως μετά τα στοιχεία με περιεχόμενο `'#pcdata'` και `empty`. Αυτό επιτυγχάνεται με το κατηγόρημα `child_pcdemp/5` (Σχήμα 5.17).

```
child_pcdemp(_,_,[],[],[]).
child_pcdemp(P,E,[element(El,child(Ch)/(Sign),A)|R1],R2,
             [element(El,child(Ch)/(Sign),A)|R]):-
     aux_check(P,E,Ch),
     child_pcdemp(P,E,R1,R2,R),!.
child_pcdemp(P,E,R1,[element(El,child(Ch)/(Sign),A)|R2],
             [element(El,child(Ch)/(Sign),A)|R]):-
     child_pcdemp(P,E,R1,R2,R).
```

**Σχήμα 5.17: Κατηγόρημα `child_pcdemp/5`**

Η τελευταία φάση ταξινόμησης ασχολείται με τα υπόλοιπα στοιχεία που προορίζονται να γίνουν κλάσεις. Τα στοιχεία αυτά πρέπει να ταξινομηθούν κατά τέτοιο τρόπο, ώστε για ένα στοιχείο-παιδί να έχει δημιουργηθεί η κλάση του προτού δημιουργηθεί η κλάση του στοιχείου-γονέα του. Αυτή η φάση επιτυγχάνεται με το

κατηγόρημα `correct_position/5`, δείγμα του οποίου απεικονίζεται στο Σχήμα 5.18.

```
correct_position([element(E1,child(Ch1)/(Sign1),A1)],
                 [element(E1,child(Ch1)/(Sign1),A1)],
                 [],[],[]).

correct_position([element(E1,child(Ch1)/(Sign1),A1)|R],
                 [element(E1,child(Ch1)/(Sign1),A1)|L1],
                 L2,L3,S):-
     append(R,S,O),
     member(element(E2,child(Ch2)/_,_),O),
     E1 \= E2,
     member(E1/_,Ch2),
     findall(Ch/_,(member(Ch/_,Ch1),
                   member(element(Ch,_,_),O)),[]),
     append([element(E1,child(Ch1)/(Sign1),A1)],S,S1),
     correct_position(R,L1,L2,L3,S1),!.
```

**Σχήμα 5.18: Κατηγόρημα `correct_position/5`**

Ο τελευταίος κανόνας `correct_position/5` εκφράζει το εξής: εάν το τρέχον στοιχείο που εξετάζεται είναι παιδί ενός άλλου στοιχείου της υπόλοιπης λίστας και τα παιδιά του δεν έχουν στοιχεία-παιδιά, τότε το στοιχείο αυτό μπαίνει στην αρχή της λίστας.

Αφού λοιπόν η λίστα που παράγεται από το στάδιο του DTD parsing περάσει και τις τρεις φάσεις ταξινόμησης που περιγράψαμε, καλείται στη συνέχεια το κατηγόρημα `create_schema/6`, το οποίο αποφασίζει για το ποια στοιχεία θα αναπαρασταθούν ως κλάσεις και ποια ως ιδιότητες άλλων κλάσεων. Δείγμα του κατηγορήματος αυτού φαίνεται στο Σχήμα 5.19.

Ο πρώτος κανόνας `create_schema/6` ασχολείται με τα στοιχεία που έχουν περιεχόμενο `'#pcdata'` και τα εισάγει σε μια λίστα (`ElemAttr`), η οποία περιλαμβάνει όλα τα στοιχεία που πρόκειται να αναπαρασταθούν ως ιδιότητες σε κλάσεις. Αντίθετα, ο δεύτερος κανόνας `create_schema/6` ασχολείται με τα στοιχεία που πρόκειται να αναπαρασταθούν με κλάσεις και καλεί το κατηγόρημα `create_class/5` (Σχήμα 5.20), το οποίο δημιουργεί τις κλάσεις αυτές.

```
create_schema(_,[],_,_,_,_).
create_schema(TopElement,
              [element(Element,'#pcdata'/(Sign),[])|R],
              ElemAttr,EmptyElem,AttLst,ElemEmpty):-
     (Sign == * ; Sign == + ; Sign == ? ; Sign == null),
     top_element(TopElement),
     Element \= TopElement,
     append(ElemAttr,[Element],ElemAttr1),
     create_schema(TopElement,R,ElemAttr1,EmptyElem,
     AttLst,ElemEmpty),!.
create_schema(TopElement,
              [element(Element,child([Child/(Sign)|RC])
                       /_,[])|R],
              ElemAttr,EmptyElem,AttLst,ElemEmpty):-
     top_element(TopElement),
     check_empty(Element,Child,EmptyElem,EmptyElem1),
     create_class(xml_seq,Element,[Child/(Sign)|RC],
                  ElemAttr,EmptyElem1),
     put_att_slots(AttLst,Element,[]),
     create_schema(TopElement,R,ElemAttr,EmptyElem1,
                   [],ElemEmpty),!.
```

**Σχήμα 5.19: Κατηγόρημα `create_schema/6`**

```
create_class(ClassType,Element,ChildrenElements,
             ElemAttr,EmptyElem):-
     SlotNames = [],
     ElemOrdAtt = [],
     EmptyAtt = [],
     SlotDefs = [],
     create_class1(ClassType,Element,ChildrenElements,
                   ElemAttr,EmptyElem,SlotNames,
                   ElemOrdAtt,EmptyAtt,SlotDefs).
```

**Σχήμα 5.20: Κατηγόρημα `create_class/5`**

Το κατηγόρημα `create_class1/9` αποτελεί ουσιαστικά μια προέκταση του κατηγορήματος `create_class/5` και περιγράφεται στο Σχήμα 5.21. Η δημιουργία των κλάσεων πραγματοποιείται με τον πρώτο κανόνα `create_class1/9`, με την αποστολή του μηνύματος `new` στην κατάλληλη κλάση `ClassType`.

Τέλος, στο Σχήμα 5.19 παρατηρούμε ότι υπάρχει και ένα άλλο κατηγόρημα, το κατηγόρημα `put_att_slots/3`, το οποίο έχει σαν είσοδο μια σειρά από σχισμές (slots) και τις προσθέτει στην κατάλληλη κλάση, αμέσως μετά τη δημιουργία της κλάσης αυτής. Το κατηγόρημα αυτό περιγράφεται στο Σχήμα 5.22.

```
create_class1(ClassType,Element,[],_,_,_,
              ElemOrdAtt,EmptyAtt,SlotDefs):-
     append([elem_ord([ElemOrdAtt])],[empty(EmptyAtt)],
            ClassAtts),
     append(ClassAtts,SlotDefs,Slots),
     new([Element,Slots]) => ClassType.
create_class1(ClassType,Element,[empty/(Sign)|RC],ElemAttr,
              EmptyElem,SlotNames,ElemOrdAtt,EmptyAtt,
              SlotDefs):-
     not_member(Element,ElemAttr),
     append(EmptyAtt,[Element],EmptyAtt1),
     create_class1(ClassType,Element,RC,ElemAttr,EmptyElem,
                   SlotNames,ElemOrdAtt,EmptyAtt1,
                   SlotDefs),!.
create_class1(ClassType,Element,[Child/(Sign)|RC],ElemAttr,
              EmptyElem,SlotNames,ElemOrdAtt,EmptyAtt,
              SlotDefs):-
     check_elemattr(Child,ElemAttr,Type),
     SlotName = Child,
     append(SlotNames,[SlotName],SlotNames1),
     aux(Sign,Card,Req),
     check_mandopt(Req,MO),
     append(ElemOrdAtt,[Child],ElemOrdAtt1),
     append(SlotDefs,[slot(slot_tuple(SlotName,global,
                                      Card,MO,Type))],
            SlotDefs1),
     create_class1(ClassType,Element,RC,ElemAttr,EmptyElem,
                   SlotNames1,ElemOrdAtt1,EmptyAtt,
                   SlotDefs1),!.
```

**Σχήμα 5.21: Κατηγόρημα `create_class1/9`**

```
put_att_slots([],_,[]):- !.
put_att_slots([],Element,[A|B]):-
     put_att_lst([A]) => Element,
     put_att_slots([],Element,B),!.
put_att_slots([slot_tuple(A,B,C,D,E)|R],Element,AttNames):-
     put_slot([slot_tuple(A,B,C,D,E)]) => Element,
     append(AttNames,[A],AttNames1),
     put_att_slots(R,Element,AttNames1).
```

**Σχήμα 5.22: Κατηγόρημα `put_att_slots/3`**

Θεωρούμε το παράδειγμα με τα στοιχεία `AUTHOR`, `LAST` και `FIRST` που αναφέραμε στο στάδιο DTD parsing. Η εφαρμογή του σταδίου αναπαράστασης του DTD σε κλάσεις στο παράδειγμα αυτό θα έχει ως αποτέλεσμα τη δημιουργία της

κλάσης `author`, με slots `last` και `first`. Έτσι, εάν στείλουμε το μήνυμα `describe` στην κλάση `author`, θα εμφανιστούν όλα τα slots και οι μέθοδοι της κλάσης. Στο Σχήμα 5.23 φαίνονται τα βασικά slots που δημιουργούνται. Εκτός από τα slots αυτά, δημιουργούνται από το σύστημα και κάποια βασικά slots και βασικές μέθοδοι, όπως έχουμε αναφέρει στο προηγούμενο κεφάλαιο.

```
?- describe => author.

Slots
slot_desc_tuple(last, author, local, single, total,
string)
slot_desc_tuple(first, author, local, single, total,
string)

Slot Values
is_a: object
empty:
alias:
att_lst:
elem_ord: [last, first]
```

**Σχήμα 5.23: Εμφάνιση των slots μιας κλάσης**

Όλα τα βήματα που παρουσιάσαμε για την υλοποίηση του σταδίου DTD mapping συνοψίζονται στο βασικό κατηγόρημα `dtd_mapping/2` που φαίνεται στο Σχήμα 5.24.

```
dtd_mapping(List,TopElement):-
     put_first_pcdemp(List,P,E,R),
     child_pcdemp(P,E,R1,R2,R),
     reverse_list(R2,R2New),
     correct_position(R2New,C1,C2,C3,[]),
     append(P,E,O),
     append(C1,C2,C4),
     append(C4,C3,CP),
     append(R1,CP,T),
     append(O,T,List1),
     last(element(TopElement,_,_),List1),
     asserta(top_element(TopElement)),
     create_schema(TopElement,List1,[],[],[],[]).
```

**Σχήμα 5.24: Κατηγόρημα `dtd_mapping/2`**

**Στάδιο Συντακτικής Ανάλυσης XML (XML parsing).** Παρόμοια με το στάδιο DTD parsing, στο στάδιο αυτό εισάγεται αρχικά το XML αρχείο και διαβάζεται με το κατηγόρημα `readfile/2`. Το κατηγόρημα αυτό μετατρέπει το XML σε ένα μεγάλο αλφαριθμητικό (string), το οποίο στη συνέχεια διαμερίζεται στις λέξεις και τα σύμβολα που το απαρτίζουν. Αυτό επιτυγχάνεται με το κατηγόρημα `tokenize1/2`, το οποίο λειτουργεί με τον ίδιο τρόπο με τον οποίο λειτουργεί και το κατηγόρημα `tokenize/2` του σταδίου συντακτικής ανάλυσης του DTD. Ο διαχωρισμός των λέξεων και των συμβόλων πραγματοποιείται με το κατηγόρημα `grab_word1/3`, το οποίο επίσης λειτουργεί όπως και το κατηγόρημα `grab_word/3` του σταδίου DTD parsing.

Στη συνέχεια, καλείται μέσω του κατηγορήματος `pass_list1/2`, το οποίο λειτουργεί με τον ίδιο τρόπο όπως και το κατηγόρημα `pass_list/2` του σταδίου DTD parsing, το κατηγόρημα `phrase/3` του συστήματος με δεύτερο όρισμα τη λίστα που παράγεται από τη διαδικασία tokenization. Αυτό που επιστρέφει ο κανόνας `phrase/3` είναι μια λίστα της μορφής:

`[element(ΌνομαΒασικούΣτοιχείου, Περιεχόμενο, Ιδιότητες)],`

όπου `Περιεχόμενο` είναι το περιεχόμενο του βασικού στοιχείου, δηλαδή τα στοιχεία-παιδιά του, καθένα από τα οποία αναπαριστάται με τη μορφή:

`child(Όνομα, Περιεχόμενο, Ιδιότητες).`

Οι γραμματικοί κανόνες που παράγουν τις παραπάνω μορφές είναι οι `element_dec 1` (Σχήμα 5.25) και `elem_cont/1` (Σχήμα 5.26).

```
element_dec([element(Name,[],Attributes)]) -->
          start_tag(Name,Attributes),
          end_tag(Name),!.
element_dec([element(Name,[A|Content],Attributes)]) -->
          start_tag(Name,Attributes),
          elem_child([A]),
          elem_child(Content),
          end_tag(Name),!.
element_dec([element(Name,A,Attributes)]) -->
          start_tag(Name,Attributes),
          elem_child(A),
          end_tag(Name).
```

**Σχήμα 5.25: Κανόνες `element_dec/1`**

```
elem_cont([child(Name,[A|Content],Attributes)|R]) -->
          start_tag(Name,Attributes),
          elem_child([A]),
          elem_cont1(Content),
          end_tag(Name),
          elem_cont(R),!.
elem_cont([child(Name,[A|Content],Attributes)]) -->
          start_tag(Name,Attributes),
          elem_child([A]),
          elem_cont1(Content),
          end_tag(Name),!.
elem_cont([child(Name,[],Attributes)|R]) -->
          start_tag(Name,Attributes),
          end_tag(Name),
          elem_cont(R),!.
elem_cont([child(Name,[],Attributes)]) -->
          start_tag(Name,Attributes),
          end_tag(Name),!.
```

**Σχήμα 5.26: Κανόνες `element_cont/1`**

Έστω ότι έχουμε τα εξής XML δεδομένα:

```
<bib>
     <book year="2003">ContentA</book>
     <book>ContentB</book>
</bib>
```

Το αποτέλεσμα από τη συντακτική ανάλυση των δεδομένων αυτών θα είναι η λίστα:

```
[element(bib,[child(book,ContentA,[year-"2003"]),
              child(book,ContentB,[])],[])] .
```

Όλα τα βήματα που παρουσιάσαμε για την υλοποίηση του σταδίου XML parsing συνοψίζονται στο βασικό κατηγόρημα `xml_parsing/2` που φαίνεται στο Σχήμα 5.27.

```
xml_parsing(XmlFile,List):-
          readfile(XmlFile,A),
          tokenize1(A,B),
          remove_spaces(B,BN),
          tokenize2(BN,BN1),
          pass_list1(BN1,List).
```

**Σχήμα 5.27: Κατηγόρημα `xml_parsing/2`**

**Στάδιο Αναπαράστασης XML με Στιγμιότυπα (XML mapping).** Αφού έχει προηγηθεί η συντακτική ανάλυση των XML δεδομένων και έχουν δημιουργηθεί οι κλάσεις (βάσει της συντακτικής ανάλυσης του DTD), το τελευταίο στάδιο είναι η δημιουργία στιγμιότυπων των κλάσεων αυτών. Τα κατηγορήματα που εκκινούν τη διαδικασία της δημιουργίας στιγμιότυπων είναι τα `create_all_instances/2` και `create_instances/2`, τα οποία απεικονίζονται στο Σχήμα 5.28.

```
create_instances(child(Child,Content,Atts),OID):-
          (get_elem_ord(ElemOrder) => Child -> true ;
           ElemOrder = []),
          (findall(A,get_att_lst(A) => Child,AttLst) ->
           true ; AttLst = []),
          (findall(E,get_empty(E) => Child,EmptyAtts) ->
           true ; EmptyAtts = []),
           create_instances1(child(Child,Content,Atts),
                             ElemOrder,AttLst,EmptyAtts,
                             SlotTuples),
           check_oid(Child,SlotTuples,OID).

create_all_instances([element(Child,Content,Atts)],
                     OIDTopElem):-
           create_instances(child(Child,Content,Atts),
                                  OIDTopElem).
```

**Σχήμα 5.28: Κατηγορήματα `create_all_instances/2` και `create_instances/2`**

Το κατηγόρημα `create_instances/2` έχει σαν πρώτο όρισμα ένα στοιχείο μαζί με τα περιεχόμενα και τις ιδιότητές του. Την πρώτη φορά που καλείται ο κανόνας `create_instances/2`, έχει σαν πρώτο όρισμα το βασικό στοιχείο (root element). Η κλήση αυτή περιλαμβάνει την ανάκτηση των slots `ElemAttr`, `EmptyAtts` και `AttLst` της κλάσης που αναπαριστά το βασικό στοιχείο και στη συνέχεια καλεί ένα νέο κατηγόρημα, το `create_instances1/5`. Το νέο αυτό κατηγόρημα αποτελεί ουσιαστικά προέκταση του αρχικού, καθώς έχει τα ίδια ορίσματα με το αρχικό συν άλλα 3 ορίσματα που αντιπροσωπεύουν τα slots που ανακτήθηκαν προηγουμένως. Ένα δείγμα του κατηγορήματος `create_instances1/5` φαίνεται στο Σχήμα 5.29.

```
create_instances1(child(_,[],[]),_,[],_,[]).
create_instances1(child(Child,Content,
                        [Att1-Val1|RA]),ElemOrder,
                  [A|R],EmptyAtts,[Value|SlotTuples]):-
     get_slot_desc(slot_desc_tuple(A,Child,_,
                                   Card,Req,_)) => Child,
     Req == total,
     Card == single,
     change_to_atom(Val1,Val11),
     Value =.. [Att1,[Val11]],
     create_instances1(child(Child,Content,RA),ElemOrder,
                       R,EmptyAtts,SlotTuples),!.
create_instances1(child(Child,[child(Elem,Content,
                                     Atts)|RC],[]),
                  [Elem1|RE],[],EmptyAtts,
                  [Value|SlotTuples]):-
     get_slot_desc(slot_desc_tuple(Elem1,Child,_,Card,
                                   Req,Type)) => Child,
     Type \= string,
     Req == total,
     Card == single,
     create_instances(child(Elem,Content,Atts),OIDElem),
     Value =.. [Elem,[OIDElem]],
     create_instances1(child(Child,RC,[]),RE,[],
                       EmptyAtts,SlotTuples),!.
```

**Σχήμα 5.29: Κατηγόρημα `create_instances1/5`**

Όταν τερματίσει ο κανόνας `create_instances1/5`, επιστρέφει τις λίστες `ElemAttr`, `EmptyAtts` και `AttLst`, και καλείται στη συνέχεια το κατηγόρημα `check_oid/3` (Σχήμα 5.30), το οποίο ελέγχει αν το τρέχον στιγμιότυπο έχει ξαναδημιουργηθεί ή όχι. Αν έχει ξαναδημιουργηθεί, τότε δε δημιουργείται νέο, αλλά στη θέση του νέου μπαίνει η ταυτότητα του ήδη υπάρχοντος. Αν δεν έχει ξαναδημιουργηθεί, τότε αποστέλλεται μήνυμα `new` στην κατάλληλη κλάση και δημιουργείται.

```
check_oid(Child,SlotTuples,OID):-
          get(OID) => Child,
          check_slots(OID,SlotTuples),!.
check_oid(Child,SlotTuples,OID):-
          new([OID,SlotTuples]) => Child.
```

**Σχήμα 5.30: Κατηγόρημα `check_oid/3`**

Το κατηγόρημα check_oid/3 καλεί το κατηγόρημα check_slots/2 (Σχήμα 5.31), το οποίο ανακτά όλα τα slots όλων των στιγμιότυπων που έχουν ήδη δημιουργηθεί και ελέγχει αν κάποιο από τα στιγμιότυπα αυτά είναι πανομοιότυπο, έχει δηλαδή τις τιμές όλων των slots ίδιες, με το τρέχον στιγμιότυπο. Ο έλεγχος αυτός είναι αρκετά σημαντικός, καθώς αποφεύγεται η δημιουργία περιττών στιγμιότυπων και εξοικονομείται χώρος.

```
check_slots(_,[]).
check_slots(OID,[S|RS]):-
          S =.. [Slot,[Content]],
          name(get,Get),
          name('_',Und),
          name(Slot,SlotNums),
          append(Get,Und,GetUnd),
          append(GetUnd,SlotNums,GetNums),
          name(GetSlot,GetNums),
          GetSlotTuple =.. [GetSlot,Content],
          (GetSlotTuple => OID -> true ; fail),
          check_slots(OID,RS).
```

**Σχήμα 5.31: Κατηγόρημα check_slots/2**

Αν θεωρήσουμε το παράδειγμα με τα στοιχεία BOOK και BIB στο προηγούμενο στάδιο της υλοποίησης, το αποτέλεσμα που θα παίρναμε από την εκτέλεση του κανόνα create_all_instances/2 θα ήταν η δημιουργία ενός στιγμιότυπου bib και δύο στιγμιότυπων book, με την προϋπόθεση ότι τα περιεχόμενα των books είναι διαφορετικά και τα στοιχεία book αναπαριστώνται με κλάσεις. Επίσης, πέρα από τη δημιουργία στιγμιότυπων, το κατηγόρημα create_all_instances/2 επιστρέφει την ταυτότητα του στιγμιότυπου του βασικού στοιχείου, η οποία απαιτείται για τη δημιουργία ενός στιγμιότυπου της κλάσης xml_doc.

Το βασικό κατηγόρημα που υλοποιεί το στάδιο XML mapping είναι το κατηγόρημα xml_mapping/2 (Σχήμα 5.32), το οποίο απλά καλεί το κατηγόρημα create_all_instances/2.

```
xml_mapping(List,OIDTopElem):-
          create_all_instances(List,OIDTopElem).
```

**Σχήμα 5.32: Κατηγόρημα xml_mapping/2**

Περιγράψαμε λοιπόν αναλυτικά τα στάδια που ακολουθήσαμε για τη διαχείριση XML δεδομένων με αντικειμενοστραφείς βάσεις δεδομένων. Το βασικό κατηγόρημα που εκτελεί και τα τέσσερα στάδια είναι το κατηγόρημα `map_files/3` που παρουσιάζεται στο Σχήμα 5.33.

```
map_files(DtDFile,XmlFile,DocOID):-
          dtd_to_classes(DtDFile),
          xml_to_objects(XmlFile,OIDTopElem),
          atom_string(DtDURI,DtDFile),
          atom_string(XmlURI,XmlFile),
          new([DocOID,[
               dtd_uri([DtDURI]),
               xml_uri([XmlURI]),
               children([OIDTopElem])
              ]]) => xml_doc.
```

**Σχήμα 5.33: Βασικό κατηγόρημα `map_files/3`**

Το παραπάνω κατηγόρημα περιλαμβάνει τα κατηγορήματα `dtd_to_classes/1` και `xml_to_objects/2`, τα οποία περιγράφονται στο Σχήμα 5.34. Το κατηγόρημα `dtd_to_classes/1` εκκινεί τα δύο πρώτα στάδια (DTD parsing και mapping), ενώ το κατηγόρημα `xml_to_objects/2` εκκινεί τα υπόλοιπα στάδια (XML parsing και mapping). Αφού εκτελεστούν τα δύο αυτά κατηγορήματα και επιστραφεί η ταυτότητα του στιγμιότυπου του βασικού στοιχείου (`OIDTopElem`), το κατηγόρημα `map_files/3` αποστέλλει μήνυμα new στην κλάση `xml_doc`, ώστε να δημιουργηθεί ένα στιγμιότυπό της με slots τα Dtd και Xml αρχεία που εισάγει ο χρήστης και την ταυτότητα `OIDTopElem`, και επιστρέφει την ταυτότητα του στιγμιότυπου αυτού (`DocOID`).

```
dtd_to_classes(DtDFile):-
          dtd_parsing(DtDFile,List),
          dtd_mapping(List,TopElement).

xml_to_objects(XmlFile,OIDTopElem):-
          xml_parsing(XmlFile,List),
          xml_mapping(List,OIDTopElem).
```

**Σχήμα 5.34: Κατηγορήματα `dtd_to_classes/1` και `xml_to_objects/2`**

Σημειώνουμε ότι κάθε στάδιο υλοποιείται σε ξεχωριστό αρχείο το καθένα. Συγκεκριμένα, το πρώτο στάδιο υλοποιείται στο αρχείο "dtdparser.pl", το δεύτερο στο αρχείο "dtdmapping.pl", το τρίτο στο αρχείο "xmlparser.pl" και το τελευταίο στο αρχείο "xmlmapping.pl".

Λαμβάνοντας υπόψη όλα όσα προαναφέραμε, το μόνο που χρειάζεται να γράψει κανείς απευθείας στο περιβάλλον της Eclipse Prolog, προκειμένου να χρησιμοποιήσει το πρόγραμμα για τη διαχείριση XML δεδομένων, είναι η κλήση:

```
map_files(<DtdFile>,<XmlFile>,OID)
```

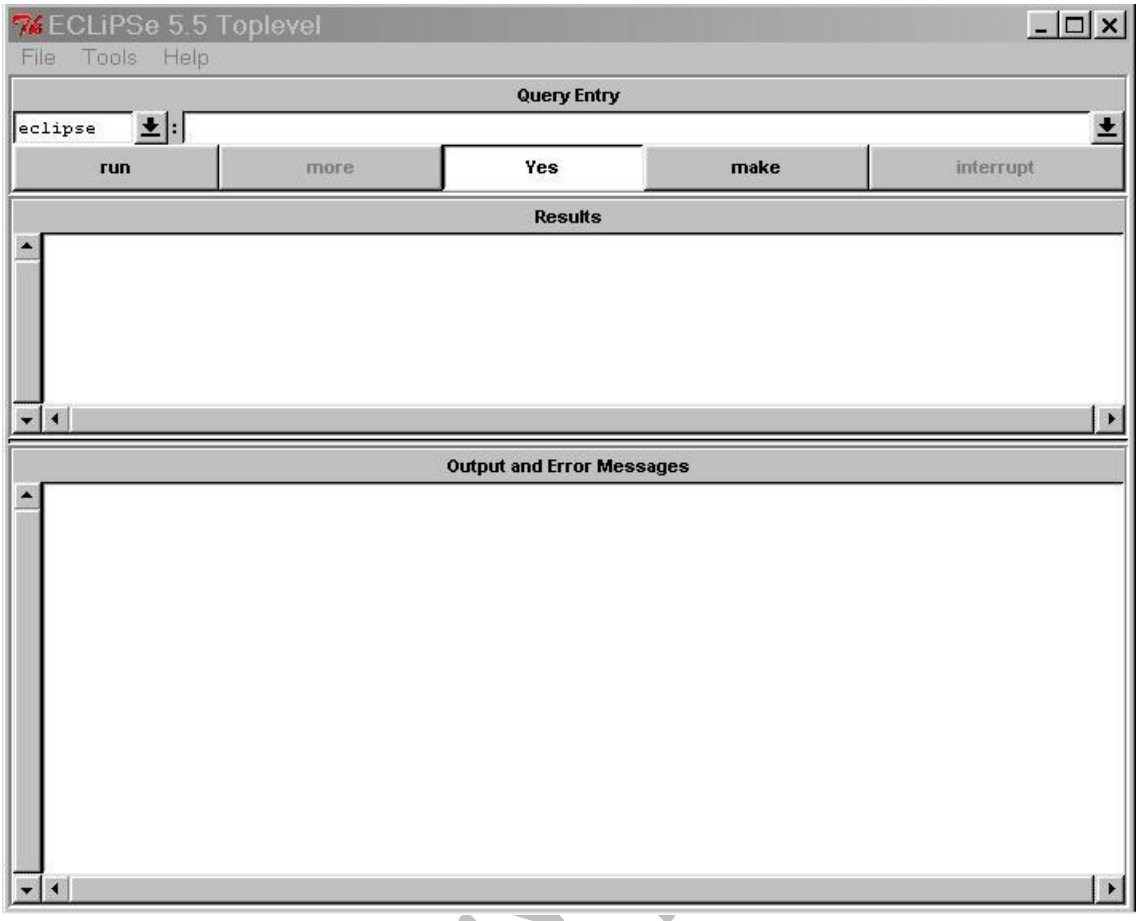
όπου `<DtdFile>` είναι το όνομα (και η διαδρομή στο δίσκο) του εξωτερικού DTD αρχείου, `<XmlFile>` το όνομα (και η διαδρομή στο δίσκο) του XML αρχείου και `OID` η ταυτότητα ενός στιγμιότυπου της κλάσης `xml_doc`, η οποία επιστρέφεται μετά από την επιτυχή απεικόνιση των DTD και XML αρχείων στο σύστημα ADAM.

Ο κώδικας που υλοποιεί το σύστημα διαχείρισης των XML δεδομένων με τη γλώσσα προγραμματισμού Prolog παρατίθεται στο Παράρτημα Α, ενώ στο Παράρτημα Δ παρουσιάζεται ένα ολοκληρωμένο παράδειγμα εκτέλεσης του αναπτυχθέντος προγράμματος μαζί με τα παραγόμενα αποτελέσματα.

## 5.4 Εκτέλεση Προγράμματος

### 5.4.1 Περιβάλλον Eclipse Prolog

Το περιβάλλον της έκδοσης Eclipse της γλώσσας προγραμματισμού Prolog φαίνεται στην Εικόνα 5.1. Στο πεδίο Query Entry εισάγουμε τα ερωτήματα και τα εκτελούμε επιλέγοντας την εντολή run. Τα αποτελέσματα και τα τυχόν λάθη από την εκτέλεση του ερωτήματος απεικονίζονται στα κάτω πεδία Results και Output and Error Messages.

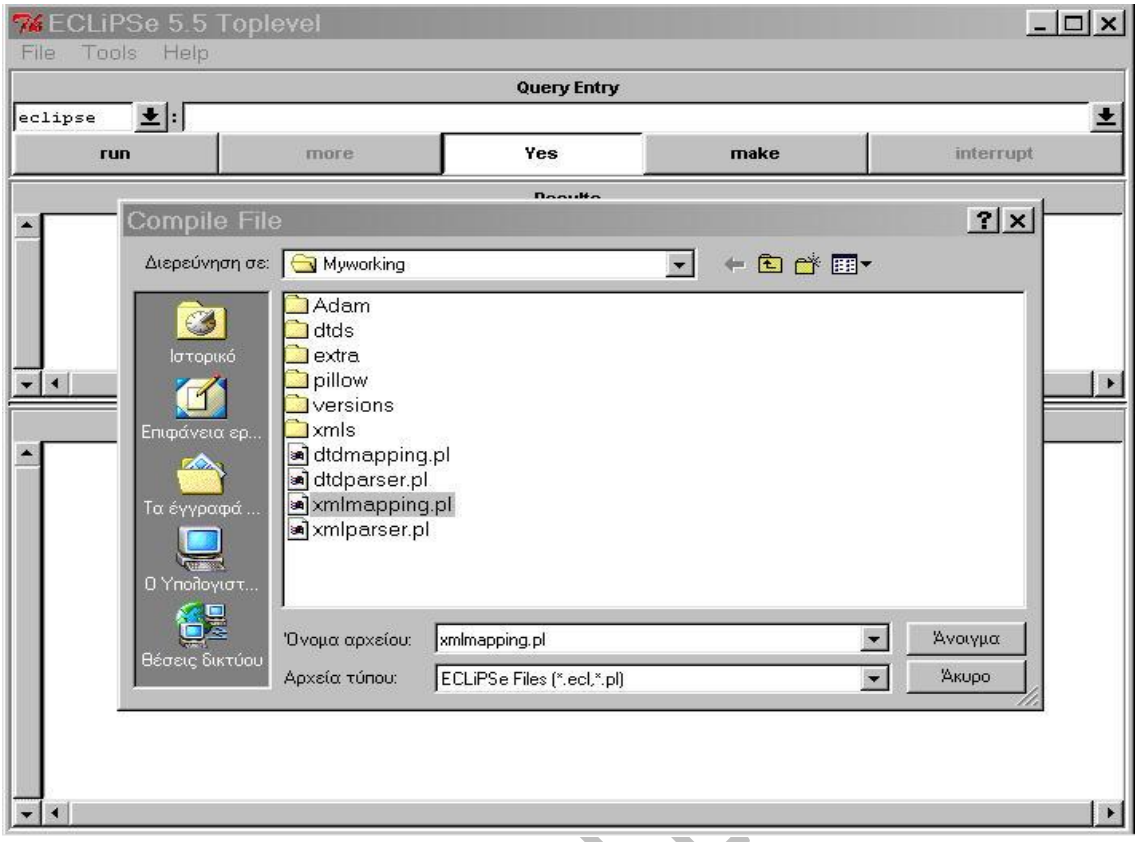



**Εικόνα 5.1: Περιβάλλον Eclipse Prolog**

Για να κάνουμε compilation ενός προγράμματος, από το μενού `File` επιλέγουμε πρώτα `Compile` και στη συνέχεια το επιθυμητό αρχείο, π.χ. το αρχείο "xmlmapping.pl" (Εικόνα 5.2).

### 5.4.2 Παράδειγμα Εκτέλεσης

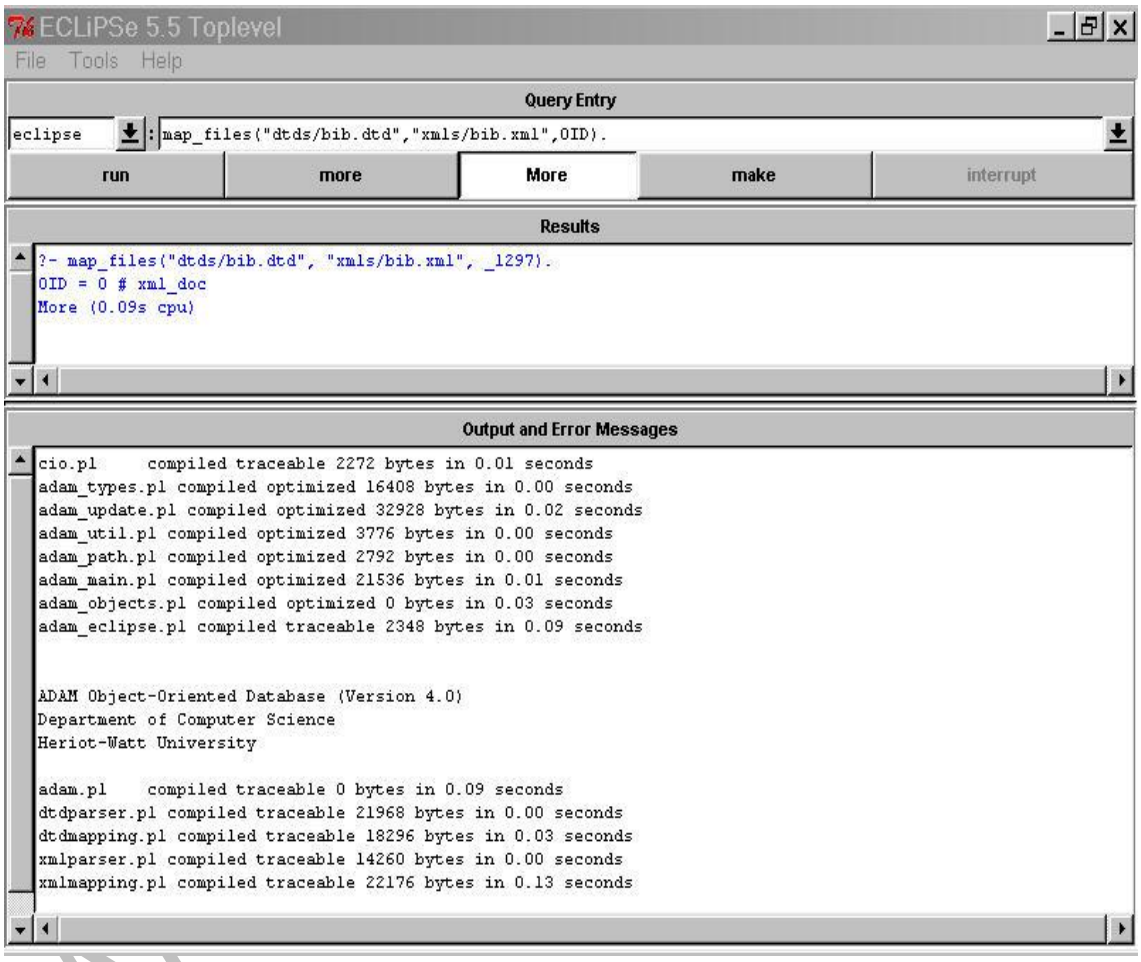
Στην ενότητα αυτή παρουσιάζεται ο τρόπος εκτέλεσης του αναπτυχθέντος προγράμματος με δεδομένα τα DTD και XML αρχεία που παρατίθενται στο Παράρτημα Δ. Πρώτα, λοιπόν, πρέπει να γίνει compilation του αρχείου που περιλαμβάνει τον κώδικα του προγράμματος. Το αρχείο αυτό είναι το "xmlmapping.pl", το οποίο με τη σειρά του κάνει αυτόματα compilation των αρχείων "adam.pl", "dtdparser.pl", "dtdmapping.pl" και "xmlparser.pl". Η διαδικασία compilation αναφέρθηκε προηγουμένως (Εικόνα 5.2). Τα αποτελέσματά της φαίνονται στην Εικόνα 5.3.




**Εικόνα 5.2: Compilation του αρχείου "xmlmapping.pl"**

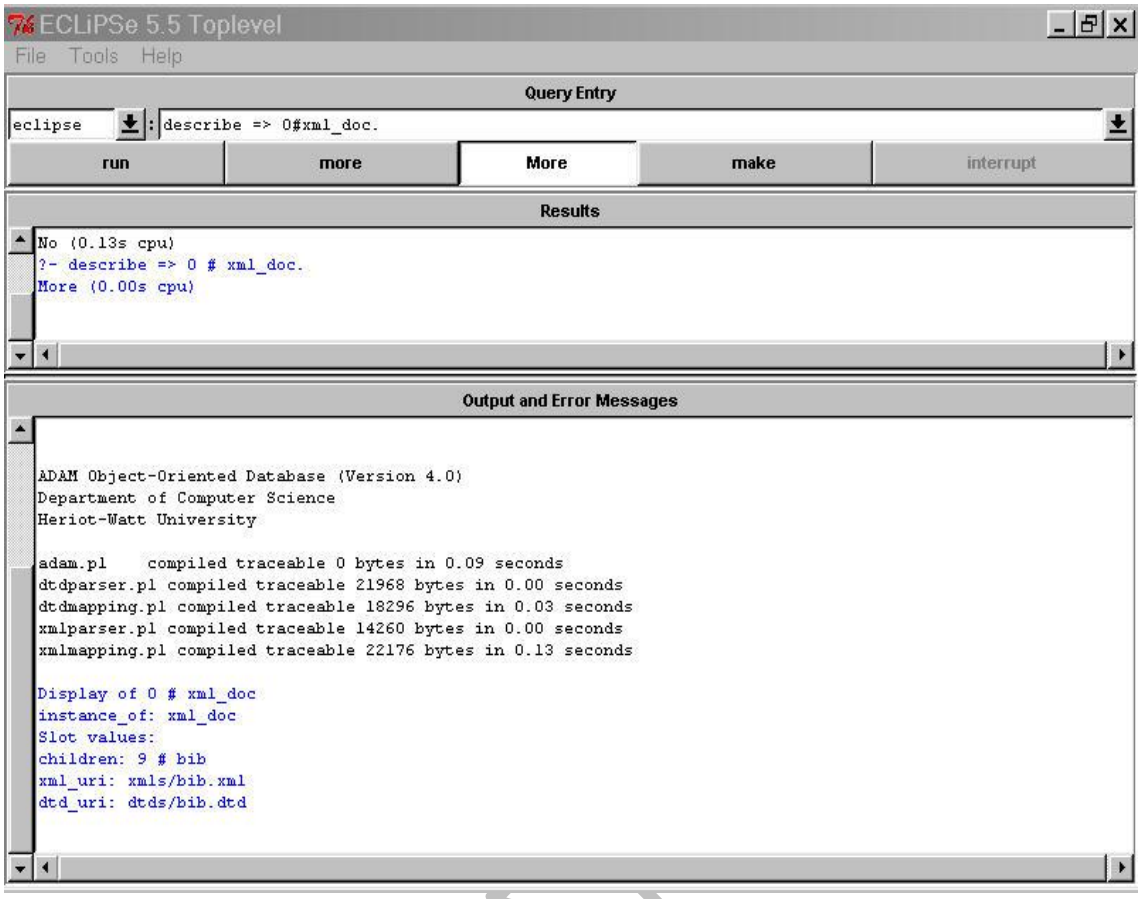

**Εικόνα 5.3: Αποτελέσματα compilation του αρχείου "xmlmapping.pl"**

Το επόμενο βήμα είναι η εκτέλεση του βασικού κατηγορήματος `map_files/3`, το οποίο υλοποιεί όλα τα στάδια για την αναπαράσταση των XML δεδομένων με κλάσεις και στιγμιότυπα. Το αποτέλεσμα από την εκτέλεση του κατηγορήματος αυτού απεικονίζεται στην Εικόνα 5.4, όπου επιστρέφεται σε μια μεταβλητή (`OID`) η ταυτότητα ενός στιγμιότυπου της κλάσης `xml_doc` (`0#xml_doc`).

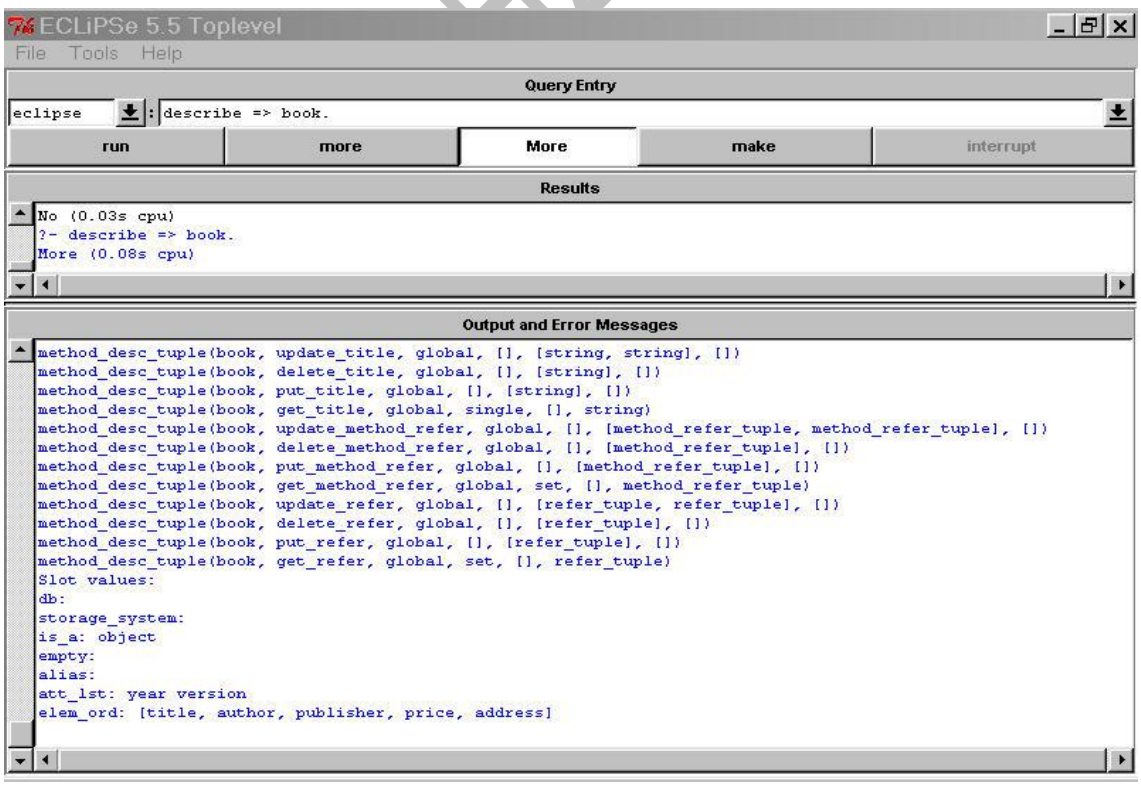



**Εικόνα 5.4: Εκτέλεση του κατηγορήματος `map_files/3`**

Μετά την εκτέλεση του παραπάνω ερωτήματος, επομένως, επιτυγχάνεται η επιθυμητή αναπαράσταση. Προκειμένου να δούμε τα περιεχόμενα του στιγμιότυπου `0#xml_doc`, πρέπει να αποστείλουμε το μήνυμα `describe` στο στιγμιότυπο αυτό (Εικόνα 5.5). Παρόμοια, για να δούμε τα περιεχόμενα μιας κλάσης, αποστέλλεται το μήνυμα `new` στην κλάση αυτή (Εικόνα 5.6).

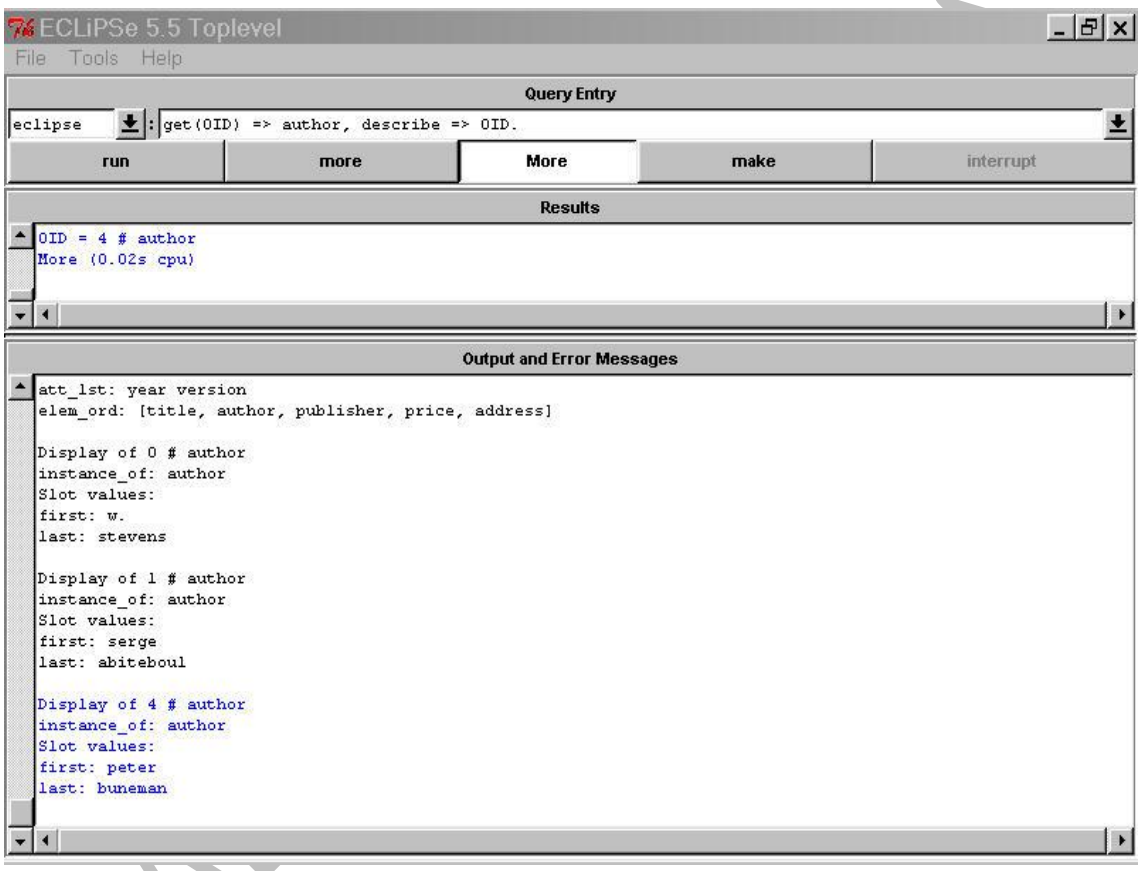



**Εικόνα 5.5: Εμφάνιση των περιεχομένων του στιγμιότυπου `0#xml_doc`**




**Εικόνα 5.6: Εμφάνιση των περιεχομένων της κλάσης `book`**

Τέλος, για να δούμε τα περιεχόμενα όλων των στιγμιότυπων της κλάσης `author`, πρέπει πρώτα να αποσταλεί το μήνυμα `get(OID)` στην κλάση αυτή, ώστε να επιστραφούν οι ταυτότητες όλων των στιγμιότυπων, και μετά να αποσταλεί μήνυμα `describe` στο καθένα από τα στιγμιότυπα αυτά (Εικόνα 5.7). Σημειώνεται πως όταν προκύπτουν περισσότερα από ένα αποτελέσματα, τότε για να τα πάρουμε όλα ένα προς ένα, χρησιμοποιούμε την επιλογή `More` εώς ότου δε βρεθεί άλλο και επιστραφεί μήνυμα `No`.




**Εικόνα 5.7: Εμφάνιση των περιεχομένων όλων των στιγμιότυπων της κλάσης `author`**

Όλες οι κλάσεις και τα στιγμιότυπα που κατασκευάζονται από την υλοποίηση του συστήματος για το συγκεκριμένο παράδειγμα παρουσιάζονται στο Παράρτημα Δ.

# 6 ΕΠΙΛΟΓΟΣ - ΣΥΜΠΕΡΑΣΜΑΤΑ

## 6.1 Εισαγωγή

Το σύστημα που αναπτύχθηκε για τη διαχείριση XML δεδομένων με τη γλώσσα προγραμματισμού Prolog δοκιμάστηκε για διαφορετικά σύνολα δεδομένων και αποδείχθηκε ότι ανταποκρίνεται στις προδιαγραφές για τις οποίες σχεδιάστηκε.

Η εκτέλεση του προγράμματος που υλοποιεί το σύστημα έδειξε για το παράδειγμα XML δεδομένων (XML έγγραφο και DTD) που παρατίθεται στο Παράρτημα Δ ότι δίνει ικανοποιητικά αποτελέσματα. Πιο συγκεκριμένα, τόσο η συντακτική ανάλυση του DTD όσο και του XML εγγράφου παράγουν σωστά τις επιθυμητές λίστες από tokens, οι οποίες χρησιμοποιούνται στα στάδια αναπαράστασης των δεδομένων με κλάσεις και στιγμιότυπα. Ακόμη, η αναπαράσταση των XML δεδομένων με κλάσεις και στιγμιότυπα συνέβαλε στη σωστή και επιτυχή δημιουργία των απαραίτητων κλάσεων και στιγμιότυπων, χωρίς την ύπαρξη πλεονασμών ή ελλείψεων.

Στις επόμενες ενότητες αναφέρονται κάποιες δυσκολίες που παρουσιάστηκαν κατά την ανάπτυξη του προγράμματος και κάποια θέματα που δε λύθηκαν και τίθενται ως μελλοντικός στόχος άλλων διπλωματικών εργασιών.

## 6.2 Προβλήματα – Δυσκολίες

Σε γενικές γραμμές δεν εμφανίστηκαν ιδιαίτερα σημαντικά προβλήματα κατά τη διάρκεια κατασκευής του προγράμματος. Η μόνη δυσκολία παρουσιάστηκε κατά τη διάρκεια υλοποίησης των σταδίων που πραγματοποιούν συντακτική ανάλυση των δεδομένων. Η δυσκολία αυτή έγκειται αρχικά στο γεγονός ότι σε κάθε έγγραφο τα δεδομένα μπορεί να συγγράφονται με τη χρήση μιας σειράς ακανόνιστων κενών χαρακτήρων, χαρακτήρων tab και χαρακτήρων αλλαγής γραμμής, οι οποίοι πρέπει να αγνοούνται από το πρόγραμμα, με αποτέλεσμα να απαιτείται η μελέτη κάθε δυνατού

συνδυασμού των παραπάνω χαρακτήρων με τα υπόλοιπα δεδομένα του εγγράφου. Αυτό φυσικά είναι μία χρονοβόρα διαδικασία, η οποία όμως υλοποιήθηκε με επιτυχία.

Εκτός από τους παραπάνω χαρακτήρες, απαιτείται κάποιος διαχωρισμός μεταξύ όλων των χαρακτηριστικών συμβόλων που περιλαμβάνονται στα δεδομένα, όπως για παράδειγμα τα σύμβολα "<", "!" και "/", προκειμένου να έχουμε τις επιθυμητές παραγόμενες λίστες. Η διαδικασία αυτή είναι επίσης χρονοβόρα, απαιτεί αρκετή μελέτη, αλλά υλοποιείται με επιτυχία.

Συνοψίζουμε, λοιπόν, ότι γενικά δεν εμφανίστηκε κάποιο σημαντικό πρόβλημα κατά την κατασκευή του προγράμματος, παρά μόνο στα στάδια της συντακτικής ανάλυσης των δεδομένων, όπου απαιτούνται οι διαδικασίες που περιγράψαμε παραπάνω, οι οποίες τελικά υλοποιούνται επιτυχώς.

## 6.3 Μελλοντικοί Στόχοι

Στην ενότητα αυτή παρουσιάζονται θέματα που δεν λύθηκαν και αποτελούν μελλοντικούς στόχους άλλων διπλωματικών εργασιών. Η πρώτη βασική δυνατότητα που δεν υποστηρίζει το πρόγραμμα είναι η δυνατότητα να υπάρχει εναλλαγή (alternation) μεταξύ των στοιχείων (elements) ενός XML εγγράφου. Για παράδειγμα, εάν στο DTD υπάρχει η δήλωση

```
<! ELEMENT BIB (BOOK|MAGAZINE)>
```

σημαίνει ότι το στοιχείο BIB έχει είτε ένα στοιχείο `BOOK` είτε ένα στοιχείο `MAGAZINE`, αλλά δε μπορεί να έχει και τα δύο. Αυτό το παράδειγμα δείχνει τη λειτουργία της εναλλαγής, η οποία συμβολίζεται με το χαρακτήρα "|" (OR).

Μια άλλη δυνατότητα που θα μπορούσε να προστεθεί στο πρόγραμμα είναι η υποστήριξη οντοτήτων (entities), σχολίων και ιδιοτήτων τύπου `nmtoken` και `nmtokens`. Επίσης, αν και έχουν προστεθεί οι περιπτώσεις ιδιοτήτων με προεπιλεγμένες τιμές, δεν υλοποιούνται σε τελικό στάδιο, αλλά αφήνονται προς μελλοντική υλοποίηση.

Με βάση όλα τα παραπάνω, μπορεί να αποτελέσει αντικείμενο διπλωματικής εργασίας η κατασκευή ενός συστήματος διαχείρισης XML δεδομένων, το οποίο θα υποστηρίζει όλες τις παραπάνω δυνατότητες και θα αποτελεί ουσιαστικά μια επέκταση του συστήματος που κατασκευάστηκε στη συγκεκριμένη διπλωματική εργασία.

Τέλος, θα μπορούσε για την εμφάνιση των αποτελεσμάτων που παράγονται από την εκτέλεση του προγράμματος πάνω σε κάποια δεδομένα, να υπάρχει ένα

λειτουργικό και ευχάριστο περιβάλλον διασύνδεσης (user interface). Και αυτό λοιπόν είναι κάτι που αποτελεί μελλοντικό στόχο προς υλοποίηση.

## 6.4 Επίλογος

Το σύστημα που υλοποιείται στη διπλωματική αυτή εργασία ικανοποιεί πλήρως και με επιτυχία τις προδιαγραφές για τις οποίες σχεδιάστηκε. Μπορεί να χρησιμοποιηθεί και να επεκταθεί, ώστε να ικανοποιεί τις οποιεσδήποτε απαιτούμενες προδιαγραφές με απόλυτη επιτυχία.

# ΠΑΡΑΡΤΗΜΑ Α

## Κώδικας Υλοποίησης του Συστήματος Διαχείρισης XML Δεδομένων με τη Γλώσσα Προγραμματισμού Prolog.

Το παράρτημα αυτό περιλαμβάνει τον κώδικα υλοποίησης του συστήματος, ο οποίος έχει μοιραστεί σε 4 αρχεία: "dtdparser.pl", "dtdmapping.pl", "xmlparser.pl" και "xmlmapping.pl". Μπορεί εύκολα κανείς να αντιληφθεί ότι καθένα από τα αρχεία αυτά εκτελεί ένα από τα στάδια υλοποίησης που αναφέραμε στο Κεφάλαιο 5. Δηλαδή, το αρχείο "dtdparser.pl" περιλαμβάνει τον κώδικα για τη συντακτική ανάλυση του DTD αρχείου, ενώ το αρχείο "xmlparser.pl" περιέχει τον κώδικα για τη συντακτική ανάλυση του XML αρχείου. Αντίστοιχα, το αρχείο "dtdmapping.pl" πραγματεύεται την αναπαράσταση των δεδομένων του DTD αρχείου με κλάσεις, ενώ το αρχείο "xmlmapping.pl" κατασκευάζει, βάσει των δεδομένων του XML αρχείου, στιγμιότυπα των κλάσεων που δημιουργήθηκαν από τον κώδικα του αρχείου "dtdmapping.pl".

### Dtdparser.pl

```
*****************************TOKENIZATION*******************************

tokenize([],[]):- !.
tokenize(String,[Word|Rest]):-
      grab_word(String,Chars,NewString),
      name(Word,Chars),
      tokenize(NewString,Rest).

grab_word([62],[62],[]).
grab_word([10],[],[]).
grab_word([60,33,Char|Tail],Tail1,Tail2):-
            Char \= 32,
            grab_word([60,33,32,Char|Tail],Tail1,Tail2),!.
grab_word([Char,62|Tail],Tail1,Tail2):-
            Char \= 32,
            grab_word([Char,32,62|Tail],Tail1,Tail2),!.
grab_word([13,10|Tail],[],Tail1):-
            first(X,Tail),
            X is 32,
            grab_word(Tail,[],Tail1),!.
grab_word([13,10|Tail],[],Tail1):-
            first(13,Tail),
            second(10,Tail),
```

```
              grab_word(Tail,[],Tail1),!.
grab_word([13,10|Tail],[],Tail).
grab_word([9|Tail],[],Tail1):-
              first(X,Tail),
              X is 9,
              grab_word(Tail,[],Tail1),!.
grab_word([9|Tail],Tail1,Tail2):-
              first(X,Tail),
              X \= 32,
              grab_word([9,32|Tail],Tail1,Tail2),!.
grab_word([9|Tail],[],Tail):-!.
grab_word([Char1,X,Char2|Tail],Tail1,Tail2):-
              (X is 40; X is 41;
               X is 42; X is 43;
               X is 44; X is 63),
              Char1 \= 32,
              Char2 \= 32,
              grab_word([Char1,32,X,32,Char2|Tail],Tail1,Tail2),!.
grab_word([Char1,X,Char2|Tail],Tail1,Tail2):-
              (X is 40; X is 41;
               X is 42; X is 43;
               X is 44; X is 63),
              Char1 is 32,
              Char2 \= 32,
              grab_word([Char1,X,32,Char2|Tail],Tail1,Tail2),!.
grab_word([Char1,X,Char2|Tail],Tail1,Tail2):-
              (X is 40; X is 41;
               X is 42; X is 43;
               X is 44; X is 63),
              Char1 \= 32,
              Char2 is 32,
              grab_word([Char1,32,X,Char2|Tail],Tail1,Tail2),!.
grab_word([32|Tail],[],Tail1):-
              first(X,Tail),
              X is 32,
              grab_word(Tail,[],Tail1),!.
grab_word([32|Tail],[],Tail):- !.
grab_word([],[],[]).
grab_word([Char1,Char2|Tail1],[NewChar|Tail2],Rest):-
              Char1 \= 13,
              Char2 \= 40, Char2 \= 41,
              Char2 \= 42, Char2 \= 43,
              Char2 \= 44, Char2 \= 63,
              grab_word([Char2|Tail1],Tail2,Rest),!,
              lower_case(Char1,NewChar).

lower_case(Char,NewChar):-
       Char >= 65,
       Char =< 90,
       NewChar is Char + 32.
lower_case(Char,Char):-
       Char < 65.
lower_case(Char,Char):-
       Char > 90.

================================================================================
**********************************DCGs**************************************

start_symbol      --> ['<!'].

end_symbol        --> [>].

left_parenthesis  --> ['('].

right_parenthesis --> [')'].

comma             --> [','].
```

```
special_sign(Sign) -->     ([*], {Sign = *});
                           ([+], {Sign = +});
                           ([?], {Sign = ?}).

element_word   --> [element].

attribute_word --> [attlist].

element_name(Name) --> [Name].

attribute_name(Name) --> [Name].

empty(empty) --> [empty].

any(any) --> [any].

pcdata('#pcdata'/Times)  -->      left_parenthesis,
                                  ['#pcdata'],
                                  right_parenthesis,
                                  special_sign(Times),!.
pcdata('#pcdata'/(null)) -->      left_parenthesis,
                                  ['#pcdata'],
                                  right_parenthesis,
                                  [null].

child1(Children/Times)   -->      (element_name(Name), {Children=Name};
                                   seq1(SeqList), {Children=SeqList}),
                                  special_sign(Times),!.
child1(Children/(null))  -->      (element_name(Name), {Children=Name};
                                   seq1(SeqList), {Children=SeqList}),
                                  [null].

child2(Child/Times)  -->   element_name(Name), {Child=Name},
                           special_sign(Times),!.
child2(Child/(null)) -->   element_name(Name), {Child=Name},
                           [null].

seqChild([Child|Rest]) --> comma,
                           child1(Child),
                           seqChild(Rest).
seqChild([]) -->    [].

seq1(Output) -->    left_parenthesis,
                    child1(Child),
                    seqChild(Rest),
                    right_parenthesis,
                    {Output=child([Child|Rest])}.

children(Children/Times)  -->     seq1(SeqList), {Children=SeqList},
                                  special_sign(Times),!.
children(Children/(null)) -->     seq1(SeqList), {Children=SeqList},
                                  [null].
children(child([Child])/(null)) --> child2(Child).

element_content(Content) -->
             empty(Content);
             any(Content);
             pcdata(Content);
             children(Content).

element_declaration([element(Name,Content,Attributes)]) -->
             start_symbol,
             element_word,
             element_name(Name),
             element_content(Content),
             end_symbol,
             attribute_declaration(Name,Attributes),!.
```

```
element_declaration([element(Name,Content,[])]) -->
              start_symbol,
              element_word,
              element_name(Name),
              element_content(Content),
              end_symbol.
element_declaration([element(Name,Content,Attributes)|Tail]) -->
              start_symbol,
              element_word,
              element_name(Name),
              element_content(Content),
              end_symbol,
              attribute_declaration(Name,Attributes),!,
              element_declaration(Tail).
element_declaration([element(Name,Content,[])|Tail]) -->
              start_symbol,
              element_word,
              element_name(Name),
              element_content(Content),
              end_symbol,
              element_declaration(Tail).

attribute_type(Card) -->
              ([cdata]; [id]; [entity]; [nmtoken]), {Card=single}.
attribute_type(Card) -->
              ([entities]; [nmtokens]), {Card=set}.

attribute_default(Req) -->
              ['#required'], {Req=mandatory}.
attribute_default(Req) -->
              ['#implied'], {Req=optional}.
attribute_default(Req) -->
              [AttValue], {Req=optional(AttValue)}.
attribute_default(Req) -->
              ['#fixed', AttValue], {Req=mandatory(AttValue)}.

attribute_content(att(Att/Card/Req)) -->
              attribute_name(Att),
              attribute_type(Card),
              attribute_default(Req).

attribute_declaration(Element,[Att|RestAtt]) -->
              start_symbol,
              attribute_word,
              attribute_name(Element),
              attribute_content(Att),
              end_symbol,
              attribute_declaration(Element,RestAtt),!.
attribute_declaration(Element,[Att]) -->
              start_symbol,
              attribute_word,
              attribute_name(Element),
              attribute_content(Att),
              end_symbol.

================================================================================
********************************************************************************

pass_list(B,S):-
              phrase(element_declaration(S),B,[]).

================================================================================
*******************************EXTRA KANONEΣ************************************

add_null([],[]).
add_null([A,B,A1,A2,A3,A4,C|D1],[A,B,A1,A2,A3,A4,C|D2]):-
              A == '<!' ,
              B == attlist,
```

```
              C == > ,
              add_null(D1,D2),!.
add_null([A,B,A1,A2,A3,A4,A5,C|D1],[A,B,A1,A2,A3,A4,A5,C|D2]):-
              A == '<!' ,
              B == attlist,
              C == > ,
              add_null(D1,D2),!.
add_null([A,B|Tail],[A,null,B|Tail1]):-
              (((B == ',' ; B == ')' ),
                A \= '?' , A \= '+' , A \= '*' , A \= '#pcdata' )
                  ;
               (A == ')' ,
                B \= '?' , B \= '+' , B \= '*' )
               ;
               (B == > ,
                A \= '?' , A \= '+' , A \= '*' )),
              add_null([B|Tail],[B|Tail1]),!.
add_null([A|Tail],[A|Tail1]):- add_null(Tail,Tail1).

change_null([],[]).
change_null([A,B,(C),D|Tail],[A,B,(C),D|Tail1]):-
       A == '(',
       C \= null,
       D == ')',
       change_null(Tail,Tail1),!.
change_null([A,B,(C),D,E|Tail],[A,B,E,D,E|Tail1]):-
       A == '(',
       C == null,
       D == ')',
       change_null(Tail,Tail1),!.
change_null([A|Tail],[A|Tail1]):- change_null(Tail,Tail1).

first(X,[X|_]).
second(X,[_,X|_]).
```

==========

==========

## Dtdmapping.pl

```
*****************************ΜΕΤΑ-ΚΛΑΣΕΙΣ*****************************

:- new([xml_doc,[
              slot(slot_tuple(dtd_uri,global,single,total,string)),
              slot(slot_tuple(xml_uri,global,single,total,string)),
              slot(slot_tuple(children,global,set,total,plog))
      ]]) => class.

:- new([xml_elem,[
              is_a([class]),
              slot(slot_tuple(alias,global,set,optional,plog)),
              slot(slot_tuple(empty,global,set,optional,string))
      ]]) => meta_class.

:- new([xml_seq,[
              is_a([xml_elem]),
              slot(slot_tuple(elem_ord,global,single,optional,plog)),
              slot(slot_tuple(att_lst,global,set,optional,string))
      ]]) => meta_class.

:- new([xml_alt,[
              is_a([xml_elem])
      ]]) => meta_class.
```

==========

```
**********************************ΚΛΑΣΕΙΣ**********************************

create_class(ClassType,Element,ChildrenElements,ElemAttr,EmptyElem):-
       SlotNames = [],
       ElemOrdAtt = [],
       EmptyAtt = [],
       SlotDefs = [],
       create_class1(ClassType,Element,ChildrenElements,ElemAttr,EmptyElem,
                     SlotNames,ElemOrdAtt,EmptyAtt,SlotDefs).

create_class1(ClassType,Element,[],_,_,_,ElemOrdAtt,EmptyAtt,SlotDefs):-
       append([elem_ord([ElemOrdAtt])],[empty(EmptyAtt)],ClassAtts),
       append(ClassAtts,SlotDefs,Slots),
       new([Element,Slots]) => ClassType.

create_class1(ClassType,Element,[empty/(Sign)|RC],ElemAttr,EmptyElem,
                SlotNames,ElemOrdAtt,EmptyAtt,SlotDefs):-
       not_member(Element,ElemAttr),
       append(EmptyAtt,[Element],EmptyAtt1),
       create_class1(ClassType,Element,RC,ElemAttr,EmptyElem,
                     SlotNames,ElemOrdAtt,EmptyAtt1,SlotDefs),!.

create_class1(ClassType,Element,[Child/(Sign)|RC],ElemAttr,EmptyElem,
                SlotNames,ElemOrdAtt,EmptyAtt,SlotDefs):-
       check_elemattr(Child,ElemAttr,Type),
       SlotName = Child,
       append(SlotNames,[SlotName],SlotNames1),
       aux(Sign,Card,Req),
       check_mandopt(Req,MO),
       append(ElemOrdAtt,[Child],ElemOrdAtt1),
       append(SlotDefs,[slot(slot_tuple(SlotName,global,Card,MO,Type))],
              SlotDefs1),
       create_class1(ClassType,Element,RC,ElemAttr,EmptyElem,
                     SlotNames1,ElemOrdAtt1,EmptyAtt,SlotDefs1),!.

================================================================================
**********************************SCHEMA**********************************

create_schema(_,[],_,_,_,_).
create_schema(TopElement,
              [element(Element,'#pcdata'/(Sign),[])|R],
              ElemAttr,EmptyElem,AttLst,ElemEmpty):-
       (Sign == * ; Sign == + ; Sign == ? ; Sign == null),
       top_element(TopElement),
       Element \= TopElement,
       append(ElemAttr,[Element],ElemAttr1),
       create_schema(TopElement,R,ElemAttr1,EmptyElem,AttLst,ElemEmpty),!.

create_schema(TopElement,
              [element(Element,child([empty/(Sign)])/_,[])|R],
               ElemAttr,EmptyElem,_,ElemEmpty):-
       (Sign == * ; Sign == + ; Sign == ? ; Sign == null),
       top_element(TopElement),
       not_member(Element,ElemEmpty),
       Element \= TopElement,
       append(EmptyElem,[Element],EmptyElem1),
       append(ElemAttr,[Element],ElemAttr1),
       create_schema(TopElement,R,ElemAttr1,EmptyElem1,[],ElemEmpty),!.

create_schema(TopElement,
              [element(Element,child([Child/(Sign)|RC])/(Sign1),
                                                  [att(Att/Card/Req)|RA])|R],
               ElemAttr,EmptyElem,AttLst,ElemEmpty):-
       top_element(TopElement),
       check_mandopt(Req,MO),
       append(AttLst,[slot_tuple(Att,global,Card,MO,string)],AttLst1),
       check_empty(Element,Child,ElemEmpty,ElemEmpty1),
```

```
       create_schema(TopElement,
                     [element(Element,child([Child/(Sign)|RC])/(Sign1),RA)|R],
                     ElemAttr,EmptyElem,AttLst1,ElemEmpty1),!.


create_schema(TopElement,
              [element(Element,child([Child/(Sign)|RC])/_,[])|R],
              ElemAttr,EmptyElem,AttLst,ElemEmpty):-
       top_element(TopElement),
       check_empty(Element,Child,EmptyElem,EmptyElem1),
       create_class(xml_seq,Element,[Child/(Sign)|RC],ElemAttr,EmptyElem1),
       put_att_slots(AttLst,Element,[]),
       create_schema(TopElement,R,ElemAttr,EmptyElem1,[],ElemEmpty),!.

================================================================================
*********************************EXTRA KANONEΣ*****************************

aux(Sign,set,optional):-     Sign == * ,!.
aux(Sign,set,mandatory):-    Sign == + ,!.
aux(Sign,single,optional):-  Sign == ? ,!.
aux(Sign,single,mandatory):- Sign == null.

check_elemattr(Child,ElemAttr,string):-
       member(Child,ElemAttr),!.
check_elemattr(Child,ElemAttr,Child):-
       get(Elem) => xml_seq,
       Elem == Child.

check_empty(Element,empty,EmptyElem,EmptyElem1):-
       append(EmptyElem,[Element],EmptyElem1),!.
check_empty(_,_,E,E).

check_mandopt(optional,optional).
check_mandopt(mandatory,total).
check_mandopt(optional(A),optional(A)).
check_mandopt(mandatory(A),total(A)).

remove_spaces([],[]).
remove_spaces([A|R],List):-
       (A == ''; A == '\t'),
       remove_spaces(R,List),!.
remove_spaces([A|R],[A|List]):-
       remove_spaces(R,List),!.

aux_check(_,_,[]).
aux_check(P,E,[Ch/_|RC]):-
       member(element(Ch,_,_),P),
       aux_check(P,E,RC),!.
aux_check(P,E,[Ch/_|RC]):-
       member(element(Ch,_,_),E),
       aux_check(P,E,RC).

last(X,[X]).
last(X,[_|T]):- last(X,T).

not_member(_,[]).
not_member(E,[H|T]):-
       E \= H,
       not_member(E,T).

reverse_list([],[]).
reverse_list([S1|L],L1):-
       reverse_list(L,L2),
       append(L2,[S1],L1).

================================================================================
```

```
*****************************ΒΟΗΘΗΤΙΚΟΙ ΚΑΝΟΝΕΣ************************

put_att_slots([],_,[]):- !.
put_att_slots([],Element,[A|B]):-
       put_att_lst([A]) => Element,
       put_att_slots([],Element,B),!.
put_att_slots([slot_tuple(A,B,C,D,E)|R],Element,AttNames):-
       put_slot([slot_tuple(A,B,C,D,E)]) => Element,
       append(AttNames,[A],AttNames1),
       put_att_slots(R,Element,AttNames1).
put_first_pcdemp([],[],[],[]).
put_first_pcdemp([element(Element,'#pcdata'/(Sign),Attributes)|R],
                 [element(Element,'#pcdata'/(Sign),Attributes)|PcdList],
                 EmptyList,RestList):-
       put_first_pcdemp(R,PcdList,EmptyList,RestList),!.
put_first_pcdemp([element(Element,child([empty/(Sign)])/(Sign1),Attributes)|R]
                                                              , PcdList,
                 [element(Element,child([empty/(Sign)])/(Sign1),
                                                    Attributes)|EmptyList],
                 RestList):-
       put_first_pcdemp(R,PcdList,EmptyList,RestList),!.
put_first_pcdemp([element(Element,child(A)/(Sign),Attributes)|R],
                 PcdList,EmptyList,
                [element(Element,child(A)/(Sign),Attributes)|RestList]):-
       put_first_pcdemp(R,PcdList,EmptyList,RestList),!.

child_pcdemp(_,_,[],[],[]).
child_pcdemp(P,E,[element(El,child(Ch)/(Sign),A)|R1],R2,[element(El,child(Ch)/
                                                          (Sign),A)|R]):-
       aux_check(P,E,Ch),
       child_pcdemp(P,E,R1,R2,R),!.
child_pcdemp(P,E,R1,[element(El,child(Ch)/(Sign),A)|R2],[element(El,child(Ch)/
                                                          (Sign),A)|R]):-
       child_pcdemp(P,E,R1,R2,R).

correct_position([],[],[],[],_).
correct_position([element(E1,child(Ch1)/(Sign1),A1)],
                 [element(E1,child(Ch1)/(Sign1),A1)],[],[],[]).

correct_position([element(E1,child(Ch1)/(Sign1),A1)|R],
                 [element(E1,child(Ch1)/(Sign1),A1)|L1],L2,L3,S):-
       append(R,S,O),
       member(element(E2,child(Ch2)/_,_),O),
       E1 \= E2,
       member(E1/_,Ch2),
       findall(Ch/_,(member(Ch/_,Ch1),member(element(Ch,_,_),O)),[]),
       append([element(E1,child(Ch1)/(Sign1),A1)],S,S1),
       correct_position(R,L1,L2,L3,S1),!.

correct_position([element(E1,child(Ch1)/(Sign1),A1)|R],L1,
                 [element(E1,child(Ch1)/(Sign1),A1)|L2],L3,S):-
       append(R,S,O),
       member(element(E2,child(Ch2)/_,_),O),
       E1 \= E2,
       member(E1/_,Ch2),
       findall(Ch/_,(member(Ch/_,Ch1),member(element(Ch,_,_),O)),H),
       H \= [],
       append([element(E1,child(Ch1)/(Sign1),A1)],S,S1),
       correct_position(R,L1,L2,L3,S1),!.

correct_position([element(E1,child(Ch1)/(Sign1),A1)|R],L1,L2,
                 [element(E1,child(Ch1)/(Sign1),A1)|L3],S):-
       append(R,S,O),
       member(element(E2,_,_),O),
       E1 \= E2,
       member(E2/_,Ch1),
       append([element(E1,child(Ch1)/(Sign1),A1)],S,S1),
       correct_position(R,L1,L2,L3,S1),!.
```

```
================================================================================
================================================================================
```

## Xmlparser.pl

```
**************************TOKENIZATION*********************************

tokenize1([],[]):- !.
tokenize1(String,[Word|Rest]):-
       grab_word1(String,Chars,NewString),
       name(Word,Chars),
       tokenize1(NewString,Rest).

grab_word1([62],[62],[]).
grab_word1([10],[],[]).
grab_word1([Char1,Char2|Tail],Tail1,Tail2):-
              Char1 \= 32,
              (Char2 is 60; Char2 is 62),
              grab_word1([Char1,32,Char2|Tail],Tail1,Tail2),!.
grab_word1([Char1,Char2|Tail],Tail1,Tail2):-
              Char2 \= 32,
              (Char1 is 60; Char1 is 62),
              grab_word1([Char1,32,Char2|Tail],Tail1,Tail2),!.
grab_word1([13,10|Tail],[],Tail1):-
              first(X,Tail),
              X is 32,
              grab_word1(Tail,[],Tail1),!.
grab_word1([13,10|Tail],[],Tail1):-
              first(13,Tail),
              second(10,Tail),
              grab_word1(Tail,[],Tail1),!.
grab_word1([13,10|Tail],[],Tail).
grab_word1([9|Tail],[],Tail1):-
              first(X,Tail),
              X is 9,
              grab_word1(Tail,[],Tail1),!.
grab_word1([9|Tail],Tail1,Tail2):-
              first(X,Tail),
              X \= 32,
              grab_word1([9,32|Tail],Tail1,Tail2),!.
grab_word1([9|Tail],[],Tail):- !.
grab_word1([Char1,X,Char2|Tail],Tail1,Tail2):-
              (X is 47; X is 61),
              Char1 \= 32,
              Char2 \= 32,
              grab_word1([Char1,32,X,32,Char2|Tail],Tail1,Tail2),!.
grab_word1([Char1,X,Char2|Tail],Tail1,Tail2):-
              (X is 47; X is 61),
              Char1 is 32,
              Char2 \= 32,
              grab_word1([Char1,X,32,Char2|Tail],Tail1,Tail2),!.
grab_word1([Char1,X,Char2|Tail],Tail1,Tail2):-
              (X is 47; X is 61),
              Char1 \= 32,
              Char2 is 32,
              grab_word1([Char1,32,X,Char2|Tail],Tail1,Tail2),!.
grab_word1([32|Tail],[],Tail1):-
              first(X,Tail),
              X is 32,
              grab_word1(Tail,[],Tail1),!.
grab_word1([32|Tail],[],Tail):-!.
grab_word1([Char1,47,Char2|Tail],Tail1,Tail2):-
              Char1 \= 60,
              Char2 \= 62,
              grab_word1([47,Char2|Tail],Tail1,Tail2),!.
```

```
grab_word1([],[],[]).
grab_word1([Char1,Char2|Tail1],[NewChar|Tail2],Rest):-
       Char1 \= 13,
       Char2 \= 47,
       Char2 \= 61,
       grab_word1([Char2|Tail1],Tail2,Rest),!,
       lower_case(Char1,NewChar).

tokenize2([>],[>]).
tokenize2([A,B,C|R],F):-
       A == > ,
       B \= < ,
       C \= < ,
       name(B,B1),
       name(C,C1),
       append(B1,[32],B2),
       append(B2,C1,D1),
       name(D,D1),
       tokenize2([A,D|R],F),!.
tokenize2([>,<|R],[>,<|A]):- tokenize2(R,A),!.
tokenize2([>,F,<|R],[>,F,<|A]):- tokenize2(R,A),!.
tokenize2([A|R],[A|B]):-
       A \= >,
       tokenize2(R,B),!.

================================================================================
**********************************DCGs************************************

start_symb  --> [<].

end_symb    --> [>].

slash       --> [/].

ison        -->     [=].

elem_name(Name)  --> [Name].

att_name(Name)   --> [Name].

att_value(Value) --> [Value].

att_cont([AttName-AttValue]) -->
                    att_name(AttName),
                    ison,
                    att_value(AttValue).
att_cont([AttName-AttValue|R]) -->
                    att_name(AttName),
                    ison,
                    att_value(AttValue),
                    att_cont(R).

start_tag(Name,[])  -->          start_symb,
                                 elem_name(Name),
                                 end_symb.
start_tag(Name,Attributes) -->   start_symb,
                                 elem_name(Name),
                                 att_cont(Attributes),
                                 end_symb.

end_tag(Name) -->    start_symb,
                     slash,
                     elem_name(Name),
                     end_symb.

content([Content]) --> [Content], {Content \= <}.

elem_child(Content) -->
```

```
            content(Content);
            elem_cont(Content).

elem_cont1([A|B]) -->      elem_child([A]),
                           elem_cont1(B),!.
elem_cont1([]) --> [].
elem_cont([child(Name,[A|Content],Attributes)|R]) -->
            start_tag(Name,Attributes),
            elem_child([A]),
            elem_cont1(Content),
            end_tag(Name),
            elem_cont(R),!.
elem_cont([child(Name,[A|Content],Attributes)]) -->
            start_tag(Name,Attributes),
            elem_child([A]),
            elem_cont1(Content),
            end_tag(Name),!.
elem_cont([child(Name,[],Attributes)|R]) -->
            start_tag(Name,Attributes),
            end_tag(Name),
            elem_cont(R),!.
elem_cont([child(Name,[],Attributes)]) -->
            start_tag(Name,Attributes),
            end_tag(Name),!.

element_dec([element(Name,[],Attributes)]) -->
            start_tag(Name,Attributes),
            end_tag(Name),!.
element_dec([element(Name,[A|Content],Attributes)]) -->
            start_tag(Name,Attributes),
            elem_child([A]),
            elem_child(Content),
            end_tag(Name),!.
element_dec([element(Name,A,Attributes)]) -->
            start_tag(Name,Attributes),
            elem_child(A),
            end_tag(Name).

================================================================================
********************************************************************************

pass_list1(B,S) :-
            phrase(element_dec(S),B,[]).

================================================================================
================================================================================
```

**XmlmappIng.pl**

```
*****************************COMPILATION*********************************

:- compile('Adam/adam.pl').
:- compile(dtdparser).
:- compile(dtdmapping).
:- compile(xmlparser).

================================================================================
*****************************ΣΤΙΓΜΙΟΤΥΠΑ*********************************

/**********Τερματική συνθήκη***********/
create_instances1(child(_,[],[]),_,[],_,[]).

/**************Ιδιότητες**************/
create_instances1(child(Child,Content,[Att1-Val1|RA]),ElemOrder,
                [A|R],EmptyAtts,[Value|SlotTuples]):-
       get_slot_desc(slot_desc_tuple(A,Child,_,Card,Req,_)) => Child,
```

```
       Req == total,
       Card == single,
       change_to_atom(Val1,Val11),
       Value =.. [Att1,[Val11]],
       create_instances1(child(Child,Content,RA),ElemOrder,R,EmptyAtts,
                                                              SlotTuples),!.

create_instances1(child(Child,Content,[Att1-Val1|RA]),ElemOrder,
                  [A|R],EmptyAtts,[Value|SlotTuples]):-
       get_slot_desc(slot_desc_tuple(A,Child,_,Card,Req,_)) => Child,
       Req == total,
       Card == set,
       atom_string(Val1,Val11),
       name(Val11,String),
       tokenize3(String,SetList),
       Value =.. [Att1,SetList],
       create_instances1(child(Child,Content,RA),ElemOrder,R,EmptyAtts,
                                                              SlotTuples),!.

 create_instances1(child(Child,Content,[Att1-Val1|RA]),ElemOrder,
                   [A|R],EmptyAtts,[Value|SlotTuples]):-
       get_slot_desc(slot_desc_tuple(A,Child,_,Card,Req,_)) => Child,
       Req == optional,
       Card == single,
       (Att1 == A,
        change_to_atom(Val1,Val11),
        Value =.. [Att1,[Val11]],
        create_instances1(child(Child,Content,RA),ElemOrder,R,EmptyAtts,
                                                              SlotTuples)
       )
       ;
       (Att1 \= A,
        create_instances1(child(Child,Content,[Att1-Val1|RA]),ElemOrder,
                          R,EmptyAtts,[Value|SlotTuples])
       ),!.

create_instances1(child(Child,Content,[Att1-Val1|RA]),ElemOrder,
                  [A|R],EmptyAtts,[Value|SlotTuples]):-
       get_slot_desc(slot_desc_tuple(A,Child,_,Card,Req,_)) => Child,
       Req == optional,
       Card == set,
       (Att1 == A,
        atom_string(Val1,Val11),
        name(Val11,String),
        tokenize3(String,SetList),
        Value =.. [Att1,SetList],
        create_instances1(child(Child,Content,RA),ElemOrder,R,EmptyAtts,
                                                              SlotTuples)
       )
       ;
       (Att1 \= A,
        create_instances1(child(Child,Content,[Att1-Val1|RA]),ElemOrder,
                          R,EmptyAtts,[Value|SlotTuples])
       ),!.

create_instances1(child(Child,Content,[Att1-Val1|RA]),ElemOrder,
                   [A|R],EmptyAtts,[Value|SlotTuples]):-
       get_slot_desc(slot_desc_tuple(A,Child,_,_,Req,_)) => Child,
       Req == optional(Default),
       Att1 == A,
       change_to_atom(Val1,Val11),
       Value =.. [Att1,[Val11]],
       create_instances1(child(Child,Content,RA),ElemOrder,R,EmptyAtts,
                                                              SlotTuples),!.

create_instances1(child(Child,Content,[Att1-Val1|RA]),ElemOrder,
                  [A|R],EmptyAtts,[Value|SlotTuples]):-
       get_slot_desc(slot_desc_tuple(A,Child,_,_,Req,_)) => Child,
```

```
        Req == optional(Default),
        Att1 \= A,
        change_to_atom(Default,Default1),
        Value =.. [A,[Default1]],
        create_instances1(child(Child,Content,[Att1-Val1|RA]),ElemOrder,
                          R,EmptyAtts,SlotTuples),!.

create_instances1(child(Child,Content,[Att1-Val1|RA]),ElemOrder,
                  [A|R],EmptyAtts,[Value|SlotTuples]):-
        get_slot_desc(slot_desc_tuple(A,Child,_,_,Req,_)) => Child,
        Req == total(Default),
        Att1 == A,
        change_to_atom(Default,Default1),
        Value =.. [A,[Default1]],
        create_instances1(child(Child,Content,RA),ElemOrder,R,EmptyAtts,
                                                              SlotTuples),!.

create_instances1(child(Child,Content,[Att1-Val1|RA]),ElemOrder,
                  [A|R],EmptyAtts,[Value|SlotTuples]):-
        get_slot_desc(slot_desc_tuple(A,Child,_,_,Req,_)) => Child,
        Req == total(Default),
        Att1 \= A,
        change_to_atom(Default,Default1),
        Value =.. [A,[Default1]],
        create_instances1(child(Child,Content,[Att1-Val1|RA]),ElemOrder,
                          R,EmptyAtts,SlotTuples),!.

create_instances1(child(A,B,[]),C,D,E,S):-
        D \= [],
        create_instances1(child(A,B,[]),C,[],E,S),!.

/*************Type # String*************/
create_instances1(child(Child,[child(Elem,Content,Atts)|RC],[]),[Elem1|RE],
                  [],EmptyAtts,[Value|SlotTuples]):-
        get_slot_desc(slot_desc_tuple(Elem1,Child,_,Card,Req,Type)) => Child,
        Type \= string,
        Req == total,
        Card == single,
        create_instances(child(Elem,Content,Atts),OIDElem),
        Value =.. [Elem,[OIDElem]],
        create_instances1(child(Child,RC,[]),RE,[],EmptyAtts,SlotTuples),!.

create_instances1(child(Child,[child(Elem,Content,Atts)|RC],[]),[Elem1|RE],
                  [],EmptyAtts,[Value|SlotTuples]):-
        get_slot_desc(slot_desc_tuple(Elem1,Child,_,Card,Req,Type)) => Child,
        Type \= string,
        Req == total,
        Card == set,
        (Elem == Elem1,
         create_instances(child(Elem,Content,Atts),OIDElem),
         Value =.. [Elem,[OIDElem]],
         create_instances1(child(Child,RC,[]),[Elem1|RE],[],EmptyAtts,
                                                              SlotTuples)
        )
        ;
        (Elem \= Elem1,
         create_instances1(child(Child,[child(Elem,Content,Atts)|RC],[]),RE,
                           [],EmptyAtts,[Value|SlotTuples])
        ),!.

create_instances1(child(Child,[child(Elem,Content,Atts)|RC],[]),[Elem1|RE],
                  [],EmptyAtts,[Value|SlotTuples]):-
        get_slot_desc(slot_desc_tuple(Elem1,Child,_,Card,Req,Type)) => Child,
        Type \= string,
        Req == optional,
        Card == single,
        (Elem == Elem1,
         create_instances(child(Elem,Content,Atts),OIDElem),
```

```
        Value =.. [Elem,[OIDElem]],
        create_instances1(child(Child,RC,[]),RE,[],EmptyAtts,SlotTuples)
       )
       ;
       (Elem \= Elem1,
        create_instances1(child(Child,[child(Elem,Content,Atts)|RC],[]),RE,
                          [],EmptyAtts,[Value|SlotTuples])
       ),!.

create_instances1(child(Child,[child(Elem,Content,Atts)|RC],[]),[Elem1|RE],
                  [],EmptyAtts,[Value|SlotTuples]):-
       get_slot_desc(slot_desc_tuple(Elem1,Child,_,Card,Req,Type)) => Child,
       Type \= string,
       Req == optional,
       Card == set,
       (Elem == Elem1,
        create_instances(child(Elem,Content,Atts),OIDElem),
        Value =.. [Elem,[OIDElem]],
        create_instances1(child(Child,RC,[]),[Elem1|RE],[],EmptyAtts,
                                                                SlotTuples)
       )
       ;
       (Elem \= Elem1,
        create_instances1(child(Child,[child(Elem,Content,Atts)|RC],[]),RE,
                          [],EmptyAtts,[Value|SlotTuples])
       ),!.

/*************Type = String*************/
create_instances1(child(Child,[child(Elem,Content,[])|RC],[]),[Elem1|RE],
                  [],EmptyAtts,[Value|SlotTuples]):-
       get_slot_desc(slot_desc_tuple(Elem1,Child,_,Card,Req,Type)) => Child,
       Type == string,
       Req == total,
       Card == single,
       return_value(Elem,Content,Value),
       create_instances1(child(Child,RC,[]),RE,[],EmptyAtts,SlotTuples),!.

create_instances1(child(Child,[child(Elem,Content,[])|RC],[]),[Elem1|RE],
                  [],EmptyAtts,[Value|SlotTuples]):-
       get_slot_desc(slot_desc_tuple(Elem1,Child,_,Card,Req,Type)) => Child,
       Type == string,
       Req == total,
       Card == set,
       (Elem == Elem1,
        return_value(Elem,Content,Value),
        create_instances1(child(Child,RC,[]),[Elem1|RE],[],EmptyAtts,
                                                                SlotTuples)
       )
       ;
       (Elem \= Elem1,
        create_instances1(child(Child,[child(Elem,Content,[])|RC],[]),RE,
                          [],EmptyAtts,[Value|SlotTuples])
       ),!.

create_instances1(child(Child,[child(Elem,Content,[])|RC],[]),[Elem1|RE],
                  [],EmptyAtts,[Value|SlotTuples]):-
       get_slot_desc(slot_desc_tuple(Elem1,Child,_,Card,Req,Type)) => Child,
       Type == string,
       Req == optional,
       Card == single,
       (Elem == Elem1,
        return_value(Elem,Content,Value),
        create_instances1(child(Child,RC,[]),RE,[],EmptyAtts,SlotTuples)
       )
       ;
       (Elem \= Elem1,
        create_instances1(child(Child,[child(Elem,Content,[])|RC],[]),RE,
                          [],EmptyAtts,[Value|SlotTuples])
```

```
       ),!.

create_instances1(child(Child,[child(Elem,Content,[])|RC],[]),[Elem1|RE],
                  [],EmptyAtts,[Value|SlotTuples]):-
       get_slot_desc(slot_desc_tuple(Elem1,Child,_,Card,Req,Type)) => Child,
       Type == string,
       Req == optional,
       Card == set,
       (Elem == Elem1,
        return_value(Elem,Content,Value),
        create_instances1(child(Child,RC,[]),[Elem1|RE],[],EmptyAtts,
                                                                SlotTuples)
       )
       ;
       (Elem \= Elem1,
        create_instances1(child(Child,[child(Elem,Content,[])|RC],[]),RE,
                          [],EmptyAtts,[Value|SlotTuples])
       ),!.

/**************Δημιουργία Στιγμιότυπων*************/
create_instances(child(Child,Content,Atts),OID):-
       (get_elem_ord(ElemOrder) => Child -> true ; ElemOrder = []),
       (findall(A,get_att_lst(A) => Child,AttLst) -> true ; AttLst = []),
       (findall(E,get_empty(E) => Child,EmptyAtts) -> true ; EmptyAtts = []),
       create_instances1(child(Child,Content,Atts),ElemOrder,
                         AttLst,EmptyAtts,SlotTuples),
       check_oid(Child,SlotTuples,OID).

create_all_instances([element(Child,Content,Atts)],OIDTopElem):-
       create_instances(child(Child,Content,Atts),OIDTopElem).

================================================================================
*********************************ΒΟΗΘΗΤΙΚΟΙ ΚΑΝΟΝΕΣ**************************

change_to_atom(A,B):-      string(A),
                           atom_string(B,A),!.
change_to_atom(A,B):-      number(A),
                           term_string(A,C),
                           atom_string(B,C),!.
change_to_atom(A,A).

tokenize3([],[]):- !.
tokenize3(String,[Word1|Rest]):-
       grab_word2(String,Chars,NewString),
       name(Word,Chars),
       change_to_atom(Word,Word1),
       tokenize3(NewString,Rest).

grab_word2([],[],[]).
grab_word2([32|Tail],[],Tail1):-
       first(X,Tail),
       X is 32,
       grab_word2(Tail,[],Tail1),!.
grab_word2([32|Tail],[],Tail):- !.
grab_word2([Char1|Tail1],[NewChar|Tail2],Rest):-
       grab_word2(Tail1,Tail2,Rest),!,
       lower_case(Char1,NewChar).

return_value(Elem,Content,C):-
       Content \= [],
       append([C1],[],Content),
       change_to_atom(C1,C2),
       C =.. [Elem,[C2]],!.
return_value(Elem,[],C):-
       C =.. [Elem,[]].

check_slots(_,[]).
check_slots(OID,[S|RS]):-
```

```
        S =.. [Slot,[Content]],
        name(get,Get),
        name('_',Und),
        name(Slot,SlotNums),
        append(Get,Und,GetUnd),
        append(GetUnd,SlotNums,GetNums),
        name(GetSlot,GetNums),
        GetSlotTuple =.. [GetSlot,Content],
        (GetSlotTuple => OID -> true ; fail),
        check_slots(OID,RS).

check_oid(Child,SlotTuples,OID):-
        get(OID) => Child,
        check_slots(OID,SlotTuples),!.
check_oid(Child,SlotTuples,OID):-
        new([OID,SlotTuples]) => Child.

================================================================================
**********************************ΒΑΣΙΚΟΙ ΚΑΝΟΝΕΣ*****************************

readfile(File,String):-
        open(File,read,Stream),
        read_string(Stream,end_of_file,_,Utf8String),
        close(Stream),
        string_list(Utf8String,String,utf8).

map_files(DtDFile,XmlFile,DocOID):-
        dtd_to_classes(DtDFile),
        xml_to_objects(XmlFile,OIDTopElem),
        atom_string(DtDURI,DtDFile),
        atom_string(XmlURI,XmlFile),
        new([DocOID,[
               dtd_uri([DtDURI]),
               xml_uri([XmlURI]),
               children([OIDTopElem])
            ]]) => xml_doc.

dtd_to_classes(DtDFile):-
        dtd_parsing(DtDFile,List),
        dtd_mapping(List,TopElement).

dtd_parsing(DtDFile,List):-
        readfile(DtDFile,A),
          tokenize(A,B),
        remove_spaces(B,BN),
        add_null(BN,C),
        change_null(C,CC),
        pass_list(CC,List).

dtd_mapping(List,TopElement):-
        put_first_pcdemp(List,P,E,R),
        child_pcdemp(P,E,R1,R2,R),
        reverse_list(R2,R2New),
        correct_position(R2New,C1,C2,C3,[]),
        append(P,E,O),
        append(C1,C2,C4),
        append(C4,C3,CP),
        append(R1,CP,T),
        append(O,T,List1),
        last(element(TopElement,_,_),List1),
        asserta(top_element(TopElement)),
        create_schema(TopElement,List1,[],[],[],[]).

xml_to_objects(XmlFile,OIDTopElem):-
        xml_parsing(XmlFile,List),
        xml_mapping(List,OIDTopElem).
```

```
xml_parsing(XmlFile,List):-
       readfile(XmlFile,A),
         tokenize1(A,B),
       remove_spaces(B,BN),
       tokenize2(BN,BN1),
       pass_list1(BN1,List).


xml_mapping(List,OIDTopElem):-
       create_all_instances(List,OIDTopElem).
```

==========

==========

# ΠΑΡΑΡΤΗΜΑ Β

## Αποθήκευση ενός XML Εγγράφου σε μια Αντικειμενοστραφή Βάση Δεδομένων (OODB).

Το παράρτημα αυτό περιλαμβάνει τον αλγόριθμο για την αναπαράσταση ενός XML εγγράφου, μαζί με το DTD του, σε μία αντικειμενοστραφή βάση δεδομένων (OODB).

```
function map_document(URI) returns DocOID
        DTD := retrieve_dtd(URI)
        DocType := retrieve_doc_type(URI)
        create_schema(DocType,DTD)
        Doc := retrieve_doc(URI)
        TopElemNodes := create_instances(Doc)
        new([DocOID,[uri([URI]),children(TopElemNodes)]]) => xml_doc

function create_schema(TopElement,DTD) returns ∅
        global ElemAttr, EmptyElem:= ∅
        for_each X in DTD
            such_that
                ((X = '<!ELEMENT' ElemName '(#PCDATA)' '>' or
                  X = '<!ELEMENT' ElemName '(#PCDATA)*' '>') and
                ElemName ≠ TopElement and
                '<!ATTLIST' ElemName * '>' ∉ DTD)
            do
                ElemAttr := ElemAttr ∪ {ElemName}
                DTD := DTD - {X}
        for_each X in DTD
            such_that
                X = '<!ELEMENT' ElemName 'EMPTY' '>'
            do
                EmptyElem := EmptyElem ∪ {ElemName}
                if (ElemName ≠ TopElement and
                    '<!ATTLIST' ElemName * '>' ∉ DTD) then
                        ElemAttr := ElemAttr ∪ {ElemName}
                        DTD := DTD - {X}
        for_each X in DTD
                such_that
                    X = '<!ELEMENT' ElemName ElementDecl '>'
                do
                    if ElementDecl = '(' ChildrenElements ')' then
                        if ChildrenElements = Elem1|...|Elemn then
                            ChildrenElements := '(' ChildrenElements ')'
                        else
                            ChildrenElements = ∅
                        create_class(xml_seq,ElemName,ChildrenElements)
                        DTD := DTD - {X}
                        if '<!ATTLIST' ElemName Attributes '>' ∈ DTD then
                            AttLst := ∅
                            for_each A in Attributes
                                such_that
                                    A = AttName AttType AttDefault
                                do
                                    Type := string
                                    case AttType of
```

```
                                        'CDATA','ID','IDREF','ENTITY','NMTOKEN':
                                           Card := single
                                        'IDREFS','ENTITIES','NMTOKENS':
                                           Card := list
                                        otherwise:
                                           Card := single
                                     case AttDefault of
                                        '#IMPLIED' :
                                           Req := optional
                                        '#REQUIRED':
                                           Req := mandatory
                                        '#FIXED' AttValue :
                                           Req := mandatory(AttValue)
                                        AttValue :
                                           Req := optional(AttValue)
                                    AttLst := AttLst ∪ {slot_desc(AttName,Type,Card,Req)}
                          put_att_lst(AttLst) => ElemName
                          DTD := DTD - {'<!ATTLIST' ElemName Attributes '>'}

function create_class(ClassType,ElemName,ChildrenElements) returns SlotNames
    SlotDefs, SlotNames, ElemOrdAtt, EmptyAtt, AliasAtt:= ∅
    AltCount, SeqCount:= 1
    for_each C in ChildrenElements
       such_that
           C = Elem OccurenceOperator
       do
           if Elem = '(' EncapsulatedElements ')' then
               if EncapsulatedElements = Elem1|...|Elemn then
                    AltClassName := ElemName ∪ '_alt' ∪ AltCount
                    AltCount := AltCount + 1
                    Aliases := create_class(xml_alt,AltClassName,EncapsulatedElements)
                    SlotName := AltClassName
                    Type := AltClassName
               elseif EncapsulatedElements = Elem1,...,Elemn then
                    SeqClassName := ElemName ∪ '_seq' ∪ SeqCount
                    SeqCount := SeqCount + 1
                    Aliases := create_class(xml_seq,SeqClassName,EncapsulatedElements)
                    SlotName := SeqClassName
                    Type := SeqClassName
               SlotNames := SlotNames ∪ Aliases
               for_each A in Aliases
                    AliasAtt := AliasAtt ∪ {A - SlotName}
           else
               if Elem = '#PCDATA'
                    SlotName := content
               else
                    SlotName := Elem
               SlotNames := SlotNames ∪ {SlotName}
               if Elem ∈ ElemAttr or Elem = '#PCDATA' then
                    Type := string
               else
                    Type := Elem
            case OccurenceOperator of
                    ∅, '?' :
                            Card := single
                    '*', '+' :
                            Card := list
            if ClassType = xml_alt then
                    Req := optional
            else
                    case OccurenceOperator of
                       ∅, '+' :
                            Req := mandatory
                       '?', '*':
                            Req := optional
                    ElemOrdAtt := ElemOrdAtt ∪append {Elem}
            if Elem ∈ EmptyElem then
               EmptyAtt := EmptyAtt ∪ {Elem}
            SlotDefs := SlotDefs ∪ {slot_desc(SlotName,Type,Card,Req)}
       ClassAtts := {elem_ord(ElemOrdAtt)} ∪ {empty(EmptyAtt)} ∪ {alias(AliasAtt)}
       new([ElemName, ClassAtts ∪ SlotDefs]) => ClassType

function create_instances(Doc) returns TopElemNodes
    Doc := Doc - {'<!DOCTYPE' RootElement * '>'}
```

```
    TopElemNodes := ∅
    for_each X in Doc
        such_that
             X = '<'RootElement Attributes '>' Contents '</'RootElement'>'
        do
             TopElemNode := create_instance(RootElement,Attributes,Contents)
             TopElemNodes := TopElemNodes ∪ {TopElemNode}

function create_instance(Class,Attributes,Contents) returns OID
   global Contents
   get_elem_order(ElemOrder) => Class
   get_att_lst(AttList) => Class
   get_alias(Alias) => Class
   get_empty(EmptyAtts) => Class
   SlotTuples := ∅
   for_each Att in AttList
        do
           get_slot_desc(slot_desc(Att,AttType,Card,Req)) => Class
           if (Req = mandatory or Req = optional) then
                Attributes := Attributes - {Att '=' Val}
           elseif Req = optional(Default) then
                if {Att '=' Val} ∉ Attributes then
                       Val := Default
                else
                       Attributes := Attributes - {Att '=' Val}
           elseif Req = mandatory(Default) then
                Val := Default
                Attributes := Attributes - {Att '=' *}
           if Val ≠ ∅ then
                if Card = single then
                       Value := {Val}
                else
                       Value := Val
                SlotTuples:= SlotTuples ∪ {Att(Value)}
   for_each Elem in ElemOrder
        do
           get_slot_desc(slot_desc(Elem,Type,Card,Req)) => Class
           if Type = string then
             if (Elem \= content or Elem ∈ EmptyAtts) then
               Value := ∅
               if (Req = mandatory and Card = single) then
                  Contents = [Head|Tail]
                  Val := get_val(Head,Elem,EmptyAtts)
                  Value := {Val}
                  Contents := Tail
               elseif (Req = mandatory and Card = list) then
                  repeat
                       Contents = [Head|Tail]
                       Val := get_val(Head,Elem,EmptyAtts)
                       Value := Value ∪ {Val}
                       Contents := Tail
                  until not check_head(Contents,Elem)
               elseif (Req = optional and Card = list) then
                  while check_head(Contents,Elem)
                       Contents = [Head|Tail]
                       Val := get_val(Head,Elem,EmptyAtts)
                       Value := Value ∪ {Val}
                       Contents := Tail
               elseif (Req = optional and Card = single) then
                  if check_head(Contents,Elem) then
                       Contents = [Head|Tail]
                       Val := get_val(Head,Elem,EmptyAtts)
                       Value := {Val}
                       Contents := Tail
             else
                 Contents = [Head|Tail]
                 if string(Head) then
                       Value := {Head}
                       Contents := Tail
                 else
                       Value := {""}
           elseif (*-Elem) ∉ Alias then
                Value := ∅
                if (Req = mandatory and Card = single) then
                   Contents = ['<'Elem AttElem '>' ElemContents '</'Elem'>'|Tail]
```

```
                Val := create_instance(Elem,AttElem,ElemContents)
                Value := {Val}
                Contents := Tail
            elseif (Req = mandatory and Card = list) then
                repeat
                    Contents = ['<'Elem AttElem '>' ElemContents '</'Elem'>'|Tail]
                    Val := create_instance(Elem,AttElem,ElemContents)
                    Value := Value ∪ {Val}
                    Contents := Tail
                until Contents \= ['<'Elem AttElem '>' ElemContents '</'Elem'>'|Tail]
            elseif (Req = optional and Card = list) then
                while Contents = ['<'Elem AttElem '>' ElemContents '</'Elem'>'|Tail]
                    Val := create_instance(Elem,AttElem,ElemContents)
                    Value := Value ∪ {Val}
                    Contents := Tail
            elseif (Req = optional and Card = single) then
                if Contents = ['<'Elem AttElem '>' ElemContents '</'Elem'>'|Tail] then
                    Val := create_instance(Elem,AttElem,ElemContents)
                    Value := {Val}
                    Contents := Tail
        else
            Value := ∅
            if (Req = mandatory and Card = single) then
                Val := create_encaps_instance(Elem,Alias,Contents)
                Value := {Val}
            elseif (Req = mandatory and Card = list) then
                Val := create_encaps_instance(Elem,Alias,Contents)
                repeat
                    Value := Value ∪ {Val}
                    Val := create_encaps_instance(Elem,Alias,Contents)
                until Val = ∅
            elseif (Req = optional and Card = list) then
                Val := create_encaps_instance(Elem,Alias,Contents)
                while Val ≠ ∅
                    Value := Value ∪ {Val}
                    Val := create_encaps_instance(Elem,Alias,Contents)
            elseif (Req = optional and Card = single) then
                Val := create_encaps_instance(Elem,Alias,Contents)
                if Val ≠ ∅
                    Value := {Val}
        SlotTuples := SlotTuples ∪ {Elem(Value)}
OID := create_object(Class,SlotTuples)

function create_encaps_instance(SystemClass,Aliases,Contents) returns OID
    get_instance_of(MetaClass) => SystemClass
    if MetaClass = xml_seq then
       OID := create_encaps_instance(SystemClass,∅,Contents)
    else
       OID := create_encaps_alt_instance(SystemClass,Aliases,Contents)

function create_encaps_alt_instance(SystemClass,Aliases,Contents) returns OID
    global Contents
    get_alias(Alias) => SystemClass
    get_empty(EmptyAtts) => SystemClass
    SlotTuples, ElemContents, OID:= ∅
    if (Contents = [Head|Tail] and string(Head) and content-SystemClass ∈ Aliases) then
       Contents := Tail
       OID := create_object(SystemClass,content([Head]))
    elseif ((Contents = ['<'Elem AttElem '>' ElemContents '</'Elem'>'|Tail] or
             Contents = ['<'Elem AttElem '/>'|Tail]) and
             Elem-SystemClass ∈ Aliases) then
       get_slot_desc(slot_desc(Elem,Type,Card,Req)) => SystemClass
       if Type = string then
           Value := ∅
          if Card = single then
              if ElemContents ≠ ∅ then
                     Value := {ElemContents}
              else
                     Value := {yes}
              Contents := Tail
          elseif Card = list then
              ElemContents := ∅
              while (Contents = ['<'Elem'>' ElemContents '</'Elem'>'|Tail] or
                     Contents = ['<'Elem'/>'|Tail])
```

```
                if ElemContents ≠ ∅ then
                    Val := ElemContents
                else
                    Val := yes
                Value := Value ∪ {Val}
                Contents := Tail
          SlotTuple := Elem(Value)
       elseif (Elem-*) ∉ Alias then
          Value := ∅
          if Card = single then
               Val := create_instance(Elem,AttElem,ElemContents)
               Value := {Val}
               Contents := Tail
          elseif Card = list then
               ElemContents := ∅
               while (Contents = ['<'Elem AttElem '>' ElemContents '</'Elem'>'|Tail] or
                       Contents = ['<'Elem AttElem '/>'|Tail])
                   Val := create_instance(Elem,AttElem,ElemContents)
                   Value := Value ∪ {Val}
                   Contents := Tail
          SlotTuple := Elem(Value)
       else
          (Elem-EncapsClass) ∈ Alias
          Value := ∅
          if Card = single then
               Val := create_encaps_instance(EncapsClass,Alias,Contents)
               Value := {Val}
          elseif Card = list then
               Val := create_encaps_instance(EncapsClass,Alias,Contents)
               while Val ≠ ∅
                   Value := Value ∪ {Val}
                   Val := create_encaps_instance(Elem,Alias,Contents)
          SlotTuple := EncapsClass(Value)
       OID := create_object(SystemClass,[SlotTuple])

function create_object(Class,SlotTuples) returns OID
    if (get(OID) => Class) and
       (for_each X in SlotTuples
               such that
                       (X = Slot(Value)) and (get_Slot(Value) => OID)) then
       true
    else
       new([OID, SlotTuples]) => Class

function get_head(Head,Elem,EmptyAtts) returns Val
    if (Head = '<' Elem '/>' or Head = '<' Elem '>' '</' Elem '>') then
       if Elem ∈ EmptyAtts
               Val := yes
       else
               Val := ""
    else
       Head = '<' Elem '>' Val '</' Elem '>'

function check_head(Contents,Elem) returns Flag
    if (Contents = ['<' Elem '/>' | Tail] or
       Contents = ['<' Elem '>' '</' Elem '>' | Tail] or
       Contents = ['<' Elem '>' Val '</' Elem '>' | Tail]) then
               Flag := true
       else
               Flag := false
```

# ΠΑΡΑΡΤΗΜΑ Γ

## Ιεραρχία Κλάσεων μιας Πανεπιστημιακής Βάσης Δεδομένων.

```
new([person, [
      slot(slot_tuple(fname, global, single, total, string)),
      slot(slot_tuple(sname, global, single, total, string))
   ]]) => class.

new([staff, [
      is_a([person]),
      slot(slot_tuple(phone, global, single, optional, integer))
   ]]) => class.

new([ra, [
      is_a([staff])
   ]]) => class.

new([lecturer, [
      is_a([staff])
   ]]) => class.

new([student, [
      is_a([person]),
      slot(slot_tuple(year, global, single, total, integer))
   ]]) => class.

new([undergrad, [
      is_a([student])
   ]]) => class.

new([postgrad, [
      is_a([student]),
      slot(slot_tuple(subject, global, single, total, string))
   ]]) => class.

new([pgra, [
      is_a([postgrad, ra]),
      slot(slot_tuple(percentra, global, single, optional,
                      integer))
   ]]) => class.
```

```
new([course, [
      slot(slot_tuple(code, global, single, total, string)),
      slot(slot_tuple(level, global, single, optional, integer))
   ]]) => class.

new([tcl, [
    slot(slot_tuple(lecturer, global, single, total, lecturer)),
    slot(slot_tuple(course, global, single, total, course)),
    slot(slot_tuple(section, global, single, total, string))
   ]]) => class.

new([enrollment, [
    slot(slot_tuple(undergrad, global, single, total,
                    undergrad)),
    slot(slot_tuple(course, global, single, total, course)),
    slot(slot_tuple(grade, global, single, optional, integer))
   ]]) => class.
```

# ΠΑΡΑΡΤΗΜΑ Δ

## Παράδειγμα Υλοποίησης Συστήματος.

Στο παράρτημα αυτό παρουσιάζεται ένα παράδειγμα υλοποίησης του συστήματος διαχείρισης XML δεδομένων. Το DTD αρχείο του παραδείγματος περιέχει τα εξής δεδομένα:

```
<!ELEMENT  BIB        (BOOK)*>
<!ELEMENT  BOOK       (TITLE, AUTHOR+, PUBLISHER, PRICE, ADDRESS)>
<!ATTLIST  BOOK        YEAR CDATA #IMPLIED>
<!ATTLIST  BOOK        VERSION CDATA #IMPLIED>
<!ELEMENT  AUTHOR     (LAST, FIRST)>
<!ELEMENT  TITLE      (#PCDATA)>
<!ELEMENT  LAST       (#PCDATA)>
<!ELEMENT  FIRST      (#PCDATA)>
<!ELEMENT  PUBLISHER  (#PCDATA)>
<!ELEMENT  PRICE      (#PCDATA)>
<!ELEMENT  ADDRESS     EMPTY>
<!ATTLIST  ADDRESS     URL CDATA #REQUIRED>
```

Το XML αρχείο του παραδείγματος περιλαμβάνει τα ακόλουθα δεδομένα:

```
<bib>
     <book year="1994">
            <title>TCP/IP Illustrated</title>
            <author>
                  <last>Stevens</last>
                  <first>W.</first>
            </author>
            <author>
                  <last>Abiteboul</last>
                  <first>Serge</first>
            </author>
            <publisher>Addison-Wesley</publisher>
            <price>65.95</price>
            <address URL="www.csd.auth.gr"></address>
    </book>
    <book year="2000">
            <title>Data on the Web</title>
            <author>
                  <last>Abiteboul</last>
                  <first>Serge</first>
            </author>
            <author>
```

```
                <last>Buneman</last>
                <first>Peter</first>
            </author>
            <author>
                <last>Suciu</last>
                <first>Dan</first>
            </author>
            <publisher>Morgan Kaufmann Publishers</publisher>
            <price>39.95</price>
            <address URL="www.csd.auth.gr"></address>
      </book>
      <book year="1999">
            <title>The Economics of Technology and Content
                 for Digital TV</title>
            <author>
                <last>Gerbarg</last>
                <first>Darcy</first>
            </author>
            <publisher>Kluwer Academic Publishers</publisher>
            <price>129.95</price>
            <address URL="www.csd.auth.gr"></address>
      </book>
</bib>
```

Οι κλάσεις που δημιουργούνται βάσει των παραπάνω δεδομένων είναι οι κλάσεις `bib`, `book`, `author` και `address`. Τα περιεχόμενα των κλάσεων αυτών περιγράφονται παρακάτω.

## ➢ Κλάση bib

```
instance_of: xml_seq
```

**Slots:**

```
slot_desc_tuple(refer, bib, system, set, optional, refer_tuple)
slot_desc_tuple(book, bib, local, set, optional, book)
```

**Methods:**

```
method_desc_tuple(object, describe, global, [], [], [])
method_desc_tuple(object, get_in_class, global, set, [], object)
method_desc_tuple(object, update_instance_of, global, [], [object,
                object], [])
method_desc_tuple(object, delete_instance_of, global, [], [object],
                [])
method_desc_tuple(object, put_instance_of, global, [], [object], [])
method_desc_tuple(object, get_instance_of, global, single, [], object)
method_desc_tuple(object, delete, global, [], [], [])
method_desc_tuple(bib, update_book, global, [], [book, book], [])
method_desc_tuple(bib, delete_book, global, [], [book], [])
```

```
method_desc_tuple(bib, put_book, global, [], [book], [])
method_desc_tuple(bib, get_book, global, set, [], book)
method_desc_tuple(bib, update_method_refer, global, [],
                  [method_refer_tuple, method_refer_tuple], [])
method_desc_tuple(bib, delete_method_refer, global, [],
                  [method_refer_tuple], [])
method_desc_tuple(bib, put_method_refer, global, [],
                  [method_refer_tuple], [])
method_desc_tuple(bib, get_method_refer, global, set, [],
                  method_refer_tuple)
method_desc_tuple(bib, update_refer, global, [], [refer_tuple,
                  refer_tuple], [])
method_desc_tuple(bib, delete_refer, global, [], [refer_tuple], [])
method_desc_tuple(bib, put_refer, global, [], [refer_tuple], [])
method_desc_tuple(bib, get_refer, global, set, [], refer_tuple)
```

**Slot values:**

```
db:
storage_system:
is_a: object
empty:
alias:
att_lst:
elem_ord: [book]
```

## ➢ Κλάση book

```
instance_of: xml_seq
```

**Slots:**

```
slot_desc_tuple(refer, book, system, set, optional, refer_tuple)
slot_desc_tuple(method_refer, book, system, set, optional,
                method_refer_tuple)
slot_desc_tuple(title, book, local, single, total, string)
slot_desc_tuple(author, book, local, set, total, author)
slot_desc_tuple(publisher, book, local, single, total, string)
slot_desc_tuple(price, book, local, single, total, string)
slot_desc_tuple(address, book, local, single, total, address)
slot_desc_tuple(year, book, local, single, optional, string)
slot_desc_tuple(version, book, local, single, optional, string)
```

**Methods:**

```
method_desc_tuple(object, describe, global, [], [], [])
method_desc_tuple(object, get_in_class, global, set, [], object)
method_desc_tuple(object, update_instance_of, global, [], [object,
                  object], [])
method_desc_tuple(object, delete_instance_of, global, [], [object],
                  [])
method_desc_tuple(object, put_instance_of, global, [], [object], [])
method_desc_tuple(object, get_instance_of, global, single, [], object)
method_desc_tuple(object, delete, global, [], [], [])
method_desc_tuple(book, update_version, global, [], [string, string],
                  [])
method_desc_tuple(book, delete_version, global, [], [string], [])
method_desc_tuple(book, put_version, global, [], [string], [])
method_desc_tuple(book, get_version, global, single, [], string)
method_desc_tuple(book, update_year, global, [], [string, string], [])
method_desc_tuple(book, delete_year, global, [], [string], [])
method_desc_tuple(book, put_year, global, [], [string], [])
method_desc_tuple(book, get_year, global, single, [], string)
method_desc_tuple(book, update_address, global, [], [address,
                  address], [])
method_desc_tuple(book, delete_address, global, [], [address], [])
method_desc_tuple(book, put_address, global, [], [address], [])
method_desc_tuple(book, get_address, global, single, [], address)
method_desc_tuple(book, update_price, global, [], [string, string],
                  [])
method_desc_tuple(book, delete_price, global, [], [string], [])
method_desc_tuple(book, put_price, global, [], [string], [])
method_desc_tuple(book, get_price, global, single, [], string)
method_desc_tuple(book, update_publisher, global, [], [string,
                  string], [])
method_desc_tuple(book, delete_publisher, global, [], [string], [])
method_desc_tuple(book, put_publisher, global, [], [string], [])
method_desc_tuple(book, get_publisher, global, single, [], string)
method_desc_tuple(book, update_author, global, [], [author, author],
                  [])
method_desc_tuple(book, delete_author, global, [], [author], [])
method_desc_tuple(book, put_author, global, [], [author], [])
method_desc_tuple(book, get_author, global, set, [], author)
method_desc_tuple(book, update_title, global, [], [string, string],
                  [])
```

```
method_desc_tuple(book, delete_title, global, [], [string], [])
method_desc_tuple(book, put_title, global, [], [string], [])
method_desc_tuple(book, get_title, global, single, [], string)
method_desc_tuple(book, update_method_refer, global, [],
                  [method_refer_tuple, method_refer_tuple], [])
method_desc_tuple(book, delete_method_refer, global, [],
                  [method_refer_tuple], [])
method_desc_tuple(book, put_method_refer, global, [],
                  [method_refer_tuple], [])
method_desc_tuple(book, get_method_refer, global, set, [],
                  method_refer_tuple)
method_desc_tuple(book, update_refer, global, [], [refer_tuple,
                  refer_tuple], [])
method_desc_tuple(book, delete_refer, global, [], [refer_tuple], [])
method_desc_tuple(book, put_refer, global, [], [refer_tuple], [])
method_desc_tuple(book, get_refer, global, set, [], refer_tuple)
```

**Slot values:**

```
db:
storage_system:
is_a: object
empty:
alias:
att_lst: year version
elem_ord: [title, author, publisher, price, address]
```

## ➢ Κλάση author

```
instance_of: xml_seq
```

**Slots:**

```
slot_desc_tuple(refer, author, system, set, optional, refer_tuple)
slot_desc_tuple(method_refer, author, system, set, optional,
                method_refer_tuple)
slot_desc_tuple(last, author, local, single, total, string)
slot_desc_tuple(first, author, local, single, total, string)
```

**Methods:**

```
method_desc_tuple(object, describe, global, [], [], [])
method_desc_tuple(object, get_in_class, global, set, [], object)
method_desc_tuple(object, update_instance_of, global, [], [object,
```

```
                object], [])
method_desc_tuple(object, delete_instance_of, global, [], [object],
                [])
method_desc_tuple(object, put_instance_of, global, [], [object], [])
method_desc_tuple(object, get_instance_of, global, single, [], object)
method_desc_tuple(object, delete, global, [], [], [])
method_desc_tuple(author, update_first, global, [], [string, string],
                [])
method_desc_tuple(author, delete_first, global, [], [string], [])
method_desc_tuple(author, put_first, global, [], [string], [])
method_desc_tuple(author, get_first, global, single, [], string)
method_desc_tuple(author, update_last, global, [], [string, string],
                [])
method_desc_tuple(author, delete_last, global, [], [string], [])
method_desc_tuple(author, put_last, global, [], [string], [])
method_desc_tuple(author, get_last, global, single, [], string)
method_desc_tuple(author, update_method_refer, global, [],
                [method_refer_tuple, method_refer_tuple], [])
method_desc_tuple(author, delete_method_refer, global, [],
                [method_refer_tuple], [])
method_desc_tuple(author, put_method_refer, global, [],
                [method_refer_tuple], [])
method_desc_tuple(author, get_method_refer, global, set, [],
                method_refer_tuple)
method_desc_tuple(author, update_refer, global, [], [refer_tuple,
                refer_tuple], [])
method_desc_tuple(author, delete_refer, global, [], [refer_tuple], [])
method_desc_tuple(author, put_refer, global, [], [refer_tuple], [])
method_desc_tuple(author, get_refer, global, set, [], refer_tuple)
```

**Slot values:**

```
db:
storage_system:
is_a: object
empty:
alias:
att_lst:
elem_ord: [last, first]
```

## ➢ Κλάση address

```
instance_of: xml_seq
```

**Slots:**

```
slot_desc_tuple(refer, address, system, set, optional, refer_tuple)
slot_desc_tuple(method_refer, address, system, set, optional,
                method_refer_tuple)
slot_desc_tuple(url, address, local, single, total, string)
```

**Methods:**

```
method_desc_tuple(object, describe, global, [], [], [])
method_desc_tuple(object, get_in_class, global, set, [], object)
method_desc_tuple(object, update_instance_of, global, [], [object,
                  object], [])
method_desc_tuple(object, delete_instance_of, global, [], [object],
                  [])
method_desc_tuple(object, put_instance_of, global, [], [object], [])
method_desc_tuple(object, get_instance_of, global, single, [], object)
method_desc_tuple(object, delete, global, [], [], [])
method_desc_tuple(address, update_url, global, [], [string, string],
                  [])
method_desc_tuple(address, delete_url, global, [], [string], [])
method_desc_tuple(address, put_url, global, [], [string], [])
method_desc_tuple(address, get_url, global, single, [], string)
method_desc_tuple(address, update_method_refer, global, [],
                  [method_refer_tuple, method_refer_tuple], [])
method_desc_tuple(address, delete_method_refer, global, [],
                  [method_refer_tuple], [])
method_desc_tuple(address, put_method_refer, global, [],
                  [method_refer_tuple], [])
method_desc_tuple(address, get_method_refer, global, set, [],
                  method_refer_tuple)
method_desc_tuple(address, update_refer, global, [], [refer_tuple,
                  refer_tuple], [])
method_desc_tuple(address, delete_refer, global, [], [refer_tuple],
                  [])
method_desc_tuple(address, put_refer, global, [], [refer_tuple], [])
method_desc_tuple(address, get_refer, global, set, [], refer_tuple)
```

**Slot values:**

db:

storage_system:

is_a: object

empty: address

alias:

att_lst: url

elem_ord:

Τέλος, τα στιγμιότυπα που δημιουργούνται βάσει των δεδομένων του συγκεκριμένου παραδείγματος περιγράφονται ως εξής:

- **Στιγμιότυπο 0 # xml_doc**

instance_of: xml_doc

**Slot values:**

children: 9 # bib

xml_uri: xmls/bib.xml

dtd_uri: dtds/bib.dtd

- **Στιγμιότυπο 9 # bib**

instance_of: bib

**Slot values:**

book: 3 # book 6 # book 8 # book

- **Στιγμιότυπο 3 # book**

instance_of: book

**Slot values:**

version:

year: "1994"

address: 2 # address

price: 65.95

publisher: addison-wesley

author: 0 # author 1 # author

title: tcp / ip illustrated

- ## Στιγμιότυπο 6 # book

instance_of: book

**Slot values:**

year: "2000"

address: 2 # address

price: 39.95

publisher: morgan kaufmann publishers

author: 1 # author 4 # author 5 # author

title: data on the web

- ## Στιγμιότυπο 8 # book

instance_of: book

**Slot values:**

version:

year: "1999"

address: 2 # address

price: 129.95

publisher: kluwer academic publishers

author: 7 # author

title: the economics of technology and content for digital tv

- ## Στιγμιότυπο 0 # author

instance_of: author

**Slot values:**

first: w.

last: stevens

- ## Στιγμιότυπο 1 # author

instance_of: author

**Slot values:**

first: serge

last: abiteboul

- **Στιγμιότυπο 4 # author**

instance_of: author

**Slot values:**

first: peter

last: buneman

- **Στιγμιότυπο 5 # author**

instance_of: author

**Slot values:**

first: dan

last: suciu

- **Στιγμιότυπο 7 # author**

instance_of: author

**Slot values:**

first: darcy

last: gerbarg

first: w.

last: stevens

- **Στιγμιότυπο 2 # address**

instance_of: address

**Slot values:**

url: www.csd.auth.gr

# Βιβλιογραφία


[1] Basileiades N., Vlahavas I., Sampson D. (2003), *Using Logic for Querying XML Data*.

[2] Gray P., Kulkarni K., Paton N. (1992), *Object-Oriented Databases, A semantic Data Model Approach*.

[3] *World Wide Web Consortium (W3C)*. http://www.w3.org

[4] W3C Consortium (2000), *Extensible Markup Language (XML) 1.0 (Second Edition)*. http://www.w3.org/TR/REC-xml

[5] Young Michael J., *Βήμα Βήμα XML*.

[6] Navarro A., White C., Burman L. (2000), *Mastering XML*.

[7] W3C Consortium (2002), *XML Query Use Cases*. http://www.w3.org/TR/xmlquery-use-cases

[8] W3C Consortium (2001), *Extensible Stylesheet Language (XSL) Version 1.0*. http://www.w3.org/TR/WD-xsl

[9] W3C Consortium (2001), *XML Linking Language (XLink) Version 1.0*. http://www.w3.org/TR/xlink

[10] W3C Consortium (2002), *XML Pointer Language (XPointer)*. http://www.w3.org/TR/xptr

[11] Bothner Per (2002), *What is XQuery?* http://www.gnu.org/software/qexo/XQuery-Intro.html

[12] W3C Consortium (1999), *XML Path Language (XPath) Version 1.0*. http://www.w3.org/TR/xpath

[13] W3C Consortium (1998), *Vector Markup Language (VML)*. http://www.w3.org/TR/NOTE-VML

[14] W3C Consortium (2002), *Synchronized Multimedia*. http://www.w3.org/AudioVideo

[15] W3C Consortium (2002), *Mathematical Markup Language (MathML)*. http://www.w3.org/Math, http://www.w3.org/Math/whatIsMathML.html

[16] Buneman P., Davidson S.B., Hillebrand G.G. and Suciu D. (1996), *A query language and optimization techniques for unstructured data*. Proceedings of the ACM SIGMOD Conference, 505-516.

[17] Goldman R. and Widom J. (1997), *DataGuides: Enabling query formulation and optimization in semistructured databases*. Proceeding of the International Conference on Very Large Databases (VLDB), 436-445.

[18] Xyleme Lucie (2001), *A dynamic warehouse for XML data of the Web*. IEEE Data Engineering Bulletin, 40-47.

[19] McHugh J., Abiteboul S., Goldman R., Quass D. and Widom J. (1997), *Lore: A database management system for semistructured data*. ACM SIGMOD Record, 54-66.

[20] Naughton J.F., DeWitt D.J., Maier D., Aboulnaga A., Chen J., Galamis L. (2001), *The Niagara Internet query system*. IEEE Data Engineering Bulletin, 27-33.

[21] Deutsch A., Fernandez M.F. and Suciu D. (1999), *Storing semistructured data with STORED*. Proceedings of the ACM SIGMOD Conference, 431-442.

[22] Floresku D. and Kossmann D. (1999), *Storing and querying XML data using an RDBMS*. IEEE Data Engineering Bulletin, 27-34.

[23] Schmidt A., Kersten M.L., Windhouwer M. and Waas F. (2000), *Efficient relational storage and retrieval of XML documents*. Proceedings of the International Workshop on the Web and Databases, 47-52.

[24] Shanmugasundaram j., Tufte K., Zhang C., He G., DeWitt D.J. and Naughton J.F. (1999), *Relational Databases for Querying XML Documents: Limitations and opportunities*. Proceedings of the International Conference on Very Large Databases (VLDB), 302-314.

[25] Chung T.S., Park S., Han S.Y. and Kim H.J. (2001), *Extracting object-oriented database schemas from XML DTDs using inheritance*. Proceedings of the International Conference on Electronic Commerce and Web Technologies (EC-Web), 49-59.

[26] Nishioka S. and Onizuka M. (2001), *Mapping XML to object-relational model*. Proceedings of the International Conference on Internet Computing, 171-177.

[27] Renner A. (2001), *XML data and object databases: A perfect couple?* Proceedings of the International Conference on Data Engineering, 143-148.

[28] Yeh C.L. (2000), *A logic programming approach to supporting the entries of XML documents in an object database*. Proceedings of the International Symposium on Practical Aspects of Declarative Languages (PADL), 278-292.

[29] Hosoya H. and Pierce B. (2000), *XDuce: A typed XML processing language*. Proceedings of the International Workshop on Web and Database, 111-116.

[30] Shimura T., Yoshikawa M. and Uemura S. (1999), *Storage and retrieval of XML documents using object-relational databases*. Proceedings of the International Conference on Database and Expert Systems Applications, 206-217.

[31] Abiteboul S., Cluet S., Christophides V., Milo T., Moerkotte G. and Simeon J. (1997), *Querying documents in object databases*. International Journal on Digital Libraries, 5-19.

[32] Abiteboul S., Quass D., McHugh J., Widom J. and Wiener J.L. (1997), *The Lorel query language for semistructured data*. International Journal on Digital Libraries, 68-88.

[33] Ozsu M.T., Iglinski P., Szafron D., El-Medani S. and Junghanns M. (1997), *An object-oriented SGML/HyTime compliant multimedia database management system*. Proceedings of the ACM Multimedia Conference, 239-249.

[34] Boehm K., Aberer K., Neuhold E.J. and Yang X. (1997), *Structured document storage and refined declarative and navigational access mechanisms in HyperStorM*. Very Large Databases (VLDB) Journal, 296-311.

[35] Bassiliades N., Vlahavas I. and Elmagarmid A.K. (2000), *E-DEVICE: An extensible active knowledge base system with multiple rule type support*. IEEE Transactions on Knowledge and Data Engineering, 824-844.

[36] Diaz O. and Jaime A. (1997), *EXACT: An extensible approach to active object-oriented databases*. Very Large Databases (VLDB) Journal, 282-295.

[37] Atkinson M.P., Bancilhon F., DeWitt D.J., Dittrich K.R., Maier D. and Zdonik S.B. (1989), *The Object-Oriented Database System Manifesto*. International Conference on Deductive and Object-Oriented Databases, 223-240.

[38] Kim W. (1990), *Introduction to Object-Oriented Databases*.

[39] Zdonik S. and Maier D. (1990), *Readings in Object-Oriented Systems*.

[40] Σελλής Τ. και Βασιλειάδης Π. (1997), *Αντικειμενοστραφή Συστήματα Διαχείρισης Βάσεων Δεδομένων*.

[41] Paton N.W. (1989), *ADAM: An object-oriented database system implemented in Prolog*. British National Conference on Databases, 147-161.

[42] Paton N.W. and Diaz O (1991)., *Object-oriented databases and frame-based systems: a comparison*.

[43] Diaz O., Paton N. and Gray P.M.D. (1991), *Rule management in object oriented databases: A uniform approach*. International Conference on Very Large Databases (VLDB), 317-326.

[44] Bassiliades N. and Gray P.M.D. (1995), *CoLan: A functional constraint language and its implementation*.

[45] Diaz O. (1995), *The Operational Semantics of User-Defined Relationships in Object Oriented Database Systems*.