Πανεπιστήμιο Πελοποννήσου

Σχολή Θετικών Επιστημών και Τεχνολογίας
Τμήμα Επιστήμης και Τεχνολογίας Υπολογιστών
Πρόγραμμα Μεταπτυχιακών Σπουδών

Ανάλυση και οπτικοποίηση Υψηλού επιπέδου ενδιάμεσης αναπαράστασης γλώσσας C εκφρασμένης σε XML μέσω της τεχνολογίας Document Object Model (DOM)

Διπλωματική Εργασία

Χρηστάκης Λέζος

Επιβλέπων: Κωνσταντίνος Μασσέλος
<Τρίπολη, Φεβρουάριος 2012>

Περίληψη

Σε αυτή την εργασία παρουσιάζεται ένα εργαλείο ανάγνωσης, ανάλυσης και οπτικοποίησης XML δεδομένων μιας υψηλού επιπέδου ενδιάμεσης αναπαράστασης (High-level Intermediate Representation, HIR) για τη γλώσσα προγραμματισμού C. Η λειτουργία αυτή αποτελεί επέκταση σε υπάρχον εργαλείο το οποίο λαμβάνει C κώδικα ως είσοδο και παράγει την αντίστοιχη High-level IR σε μορφή XML αλλά και οπτικοποιημένη. Στόχος της παρούσας διπλωματικής εργασίας είναι να επιτευχθεί η φορητότητα των High-Level IR δεδομένων μέσω XML. Για την ανάγνωση και ανάλυση των XML δεδομένων γίνεται χρήση του Document Object Model (DOM) μέσα από το Microsoft XML Core Services (MSXML) ενώ για την οπτικοποίηση χρησιμοποιείται η εφαρμογή οπτικοποίησης γράφων Graphviz.

Abstract

This thesis presents a tool for parsing and visualizing XML data of a High-level Intermediate Representation (HIR) for the C programming language. The application that has been developed is an extension to an existing tool which receives C code as input and produces the corresponding High-level IR in two forms: XML and visualized GIF. The aim of this thesis is to achieve portability of the High-level IR data through XML. The Document Object Model (DOM) as implemented in the Microsoft XML Core Services (MSXML) is used for parsing the XML documents and the open source graph visualization software Graphviz is used for visualizing the HIR.

Ευχαριστίες

Η παρούσα διπλωματική εργασία πραγματοποιήθηκε στα πλαίσια απόκτησης Μεταπτυχιακού Διπλώματος Ειδίκευσης (Μ.Δ.Ε.) στην "Επιστήμη και Τεχνολογία Υπολογιστών" του τμήματος Επιστήμης και Τεχνολογίας Υπολογιστών του Πανεπιστημίου Πελοποννήσου.

Θα ήθελα να ευχαριστήσω τον κ. Γρηγόρη Δημητρουλάκο, ΕΕΔΙΠ στο τμήμα Επιστήμης και Τεχνολογίας Υπολογιστών, για τη συνεισφορά του στην παρούσα εργασία, τις συμβουλές, τη βοήθεια και την πρόθυμη υποστήριξη του σε διάφορα ζητήματα που προέκυπταν κατά την υλοποίηση.

Θα ήθελα επίσης να ευχαριστήσω τα μέλη του οικογενειακού και φιλικού μου περιβάλλοντος για την πολύπλευρη συμπαράσταση τους καθ' όλη τη διάρκεια των σπουδών μου.

Η στοιχειοθεσία του κειμένου έγινε με το LaTeX.
Χρησιμοποιήθηκε η γραμματοσειρά Linux Libertine.

Περιεχόμενα

Κατάλογος σχημάτων

Εισαγωγή

Οι περισσότεροι μεταγλωττιστές μεταφράζουν τον πηγαίο κώδικα σε κάποια μορφή ενδιάμεσης αναπαράστασης και από εκεί κάνουν τη μετατροπή σε κώδικα μηχανής. Υπάρχουν πάρα πολλές ενδιάμεσες αναπαραστάσεις σε χρήση και κατηγοριοποιούνται ανάλογα με το αν είναι πιο κοντά σε μία γλώσσα υψηλού επιπέδου ή πιο κοντά στον κώδικα μηχανής. Οι ενδιάμεσες αναπαραστάσεις που βρίσκονται κοντά στις γλώσσες υψηλού επιπέδου ονομάζονται υψηλού επιπέδου ενδιάμεσες αναπαραστάσεις (High-Level Intermediate Representations, HIRs).

Μία υλοποίηση μιας τέτοιας υψηλού επιπέδου ενδιάμεσης αναπαράστασης για τη γλώσσα C μπορεί να βρεθεί στο [13].

Η συγκεκριμένη υλοποίηση παρέχει:

- Ένα εργαλείο για τη μετατροπή C κώδικα στην HIR, το front end μέρος δηλαδή ενός μεταγλωττιστή για C. Η εφαρμογή παίρνει κώδικα C σαν είσοδο και παράγει την HIR σε XML αλλά και οπτικοποιημένη σε μορφή GIF.

- Μια βιβλιοθήκη με κλάσεις για όλα τα συντακτικά αντικείμενα που αποτελούν αυτή την υψηλού επιπέδου ενδιάμεση αναπαράσταση. Την βιβλιοθήκη αυτή τη χρησιμοποιεί το παραπάνω εργαλείο για να αναπαραστήσει εσωτερικά τα συντακτικά αντικείμενα της HIR. Η βιβλιοθήκη είναι σε μορφή C++ Static Library και στην εργασία θα αναφέρεται ως "HLIR Library".

Στην παρούσα εργασία κατασκευάστηκε ένα εργαλείο επέκτασης της παραπάνω HIR με σκοπό την επίτευξη της φορητότητας της μέσω XML. Το εργαλείο αυτό παίρνει την HIR που παράγει η παραπάνω εφαρμογή σε μορφή XML και την οπτικοποιεί σε μορφή GIF. Είναι γραμμένο σε C++ και χρησιμοποιεί μια σειρά τεχνολογιών όπως το Microsoft XML Core Services (MSXML) για την ανάλυση του XML εγγράφου, την HLIR Library που αναφέρθηκε παραπάνω για την εσωτερική αναπαράσταση της HIR και την εφαρμογή οπτικοποίησης γράφων Graphviz για τη δημιουργία του HIR δέντρου και την οπτικοποίηση του σε μορφή GIF. Το εργαλείο αυτό θα αναφέρεται στο υπόλοιπο της εργασίας ως "HLIR XML Parser".

Ακολουθεί μια σύντομη περιγραφή του κάθε κεφαλαίου της εργασίας:

Στο πρώτο κεφάλαιο γίνεται μια εισαγωγική παρουσίαση στους μεταγλωττιστές με έμφαση στις ενδιάμεσες αναπαραστάσεις. Γίνεται συνοπτική αναφορά

στη διαδικασία που ακολουθείται κατά τη μεταγλώττιση και εκτενέστερη στις υψηλού επιπέδου ενδιάμεσες αναπαραστάσεις.

Στο δεύτερο κεφάλαιο παρουσιάζεται το MSXML, μια συλλογή υπηρεσιών που διευκολύνουν τη χρήση του XML. Το MSXML μεταξύ άλλων παρέχει κατάλληλες προγραμματιστικές διεπαφές, σε διάφορες γλώσσες προγραμματισμού, για την ανάλυση ενός XML εγγράφου μέσω του Document Object Model (DOM). Μέσω των διεπαφών αυτών ο HLIR XML Parser διαβάζει και αναλύει XML έγγραφα. Το κεφάλαιο αυτό αποτελεί ουσιαστικά μια τεκμηρίωση των κυριότερων διεπαφών του MSXML για το DOM και του τρόπου με τον οποίο αυτές υλοποιούνται στην C++.

Το τρίτο κεφάλαιο αναφέρεται στο βασικό εργαλείο που αναπτύχθηκε για αυτή την εργασία, τον HLIR XML Parser. Εξηγείται ο τρόπος λειτουργίας του ενώ αναλύονται λεπτομερώς όλες οι κλάσεις και συναρτήσεις που το απαρτίζουν. Γίνεται επίσης αναφορά στα βασικά στοιχεία της HLIR Library και στον τρόπο αλληλεπίδρασης της με τον HLIR XML Parser. Τέλος εκτελείται δοκιμαστικά η εφαρμογή για ορισμένα XML αρχεία και παρουσιάζεται η έξοδος που παράγει.

Στο τέταρτο και τελευταίο κεφάλαιο αναπτύχθηκε ένα γραφικό περιβάλλον εργασίας (Graphical User Interface, GUI) για τον HLIR XML Parser το οποίο και αναλύεται. Για ευκολία στην υλοποίηση της εφαρμογής αυτής η ανάπτυξη έγινε σε C#.

Κεφάλαιο 1

Μεταγλωττιστές & ενδιάμεσες αναπαραστάσεις

1.1 Μεταγλωττιστές

1.1.1 Εισαγωγή

Ένας *μεταγλωττιστής* είναι ένα πρόγραμμα το οποίο διαβάζει ένα πρόγραμμα γραμμένο σε μία γλώσσα (*source* γλώσσα) και το μεταφράζει σε ένα ισοδύναμο πρόγραμμα σε μία άλλη γλώσσα (*target* γλώσσα). Η source γλώσσα είναι συνήθως μια γλώσσα υψηλού επιπέδου όπως η C++ και η Java, ενώ η target γλώσσα είναι συνήθως μια γλώσσα χαμηλού επιπέδου όπως η assembly ή η γλώσσα μηχανής. Κατά τη διάρκεια της μετάφρασης ο μεταγλωττιστής γνωστοποιεί σφάλματα και προειδοποιήσεις ώστε να βοηθήσει τον προγραμματιστή να κάνει διορθώσεις στο source πρόγραμμα και να ολοκληρωθεί η μετάφραση.

Η κατασκευή ενός μεταγλωττιστή είναι μια πολύπλοκη διαδικασία. Παλαιότερα η διαδικασία αυτή θεωρούνταν εξαιρετικά χρονοβόρα, σήμερα όμως έχει απλουστευθεί αρκετά χάρις στην καλύτερη κατανόηση και οργάνωση της αλλά και την ανάπτυξη διάφορων προγραμματιστικών εργαλείων για την υλοποίηση των διάφορων σταδίων της.

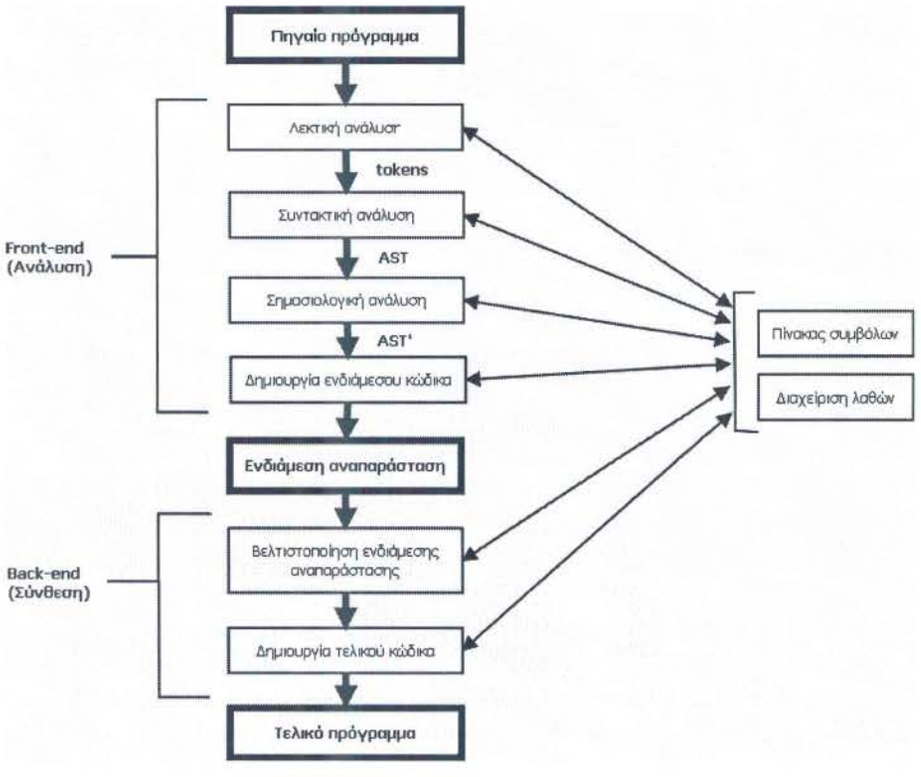
Ένα εργαλείο που εκτελεί διαφορετική εργασία από ένα μεταγλωττιστή και δεν πρέπει να συγχέονται είναι ο *διερμηνέας*. Αντί να μετατρέπει ένα πρόγραμμα σε μια target γλώσσα, ο διερμηνέας εκτελεί κατευθείαν τις λειτουργίες που καθορίζονται στο source πρόγραμμα.

1.1.2 Η δομή ενός μεταγλωττιστή

Ένας τυπικός μεταγλωττιστής αποτελείται από διάφορες φάσεις κάθε μια από τις οποίες μεταβιβάζει την έξοδο που παράγει στην επόμενη η οποία τη λαμβάνει ως είσοδο. Όπως φαίνεται και στο σχήμα 1.1 οι φάσεις αυτές μπορούν να χωριστούν σε δύο στάδια, στην *ανάλυση* και τη *σύνθεση*. Κατά την ανάλυση διαβάζεται το source πρόγραμμα, αναγνωρίζονται τα συντακτικά μέρη του και

δημιουργείται μια ενδιάμεση αναπαράσταση του. Η ενδιάμεση αυτή αναπαράσταση μετατρέπεται στην target γλώσσα κατά τη σύνθεση.

Το στάδιο της ανάλυσης ονομάζεται και *front-end* του μεταγλωττιστή ενώ η σύνθεση λέγεται και *back-end* μέρος.



Σχήμα 1.1: Φάσεις ενός μεταγλωττιστή.

Τα δύο στάδια του μεταγλωττιστή περιλαμβάνουν τις παρακάτω φάσεις για το χειρισμό των διάφορων τμημάτων της διαδικασίας:

Front-end (Ανάλυση)

1. **Λεκτική ανάλυση (Scanning)**: Στην πρώτη φάση ενός μεταγλωττιστή ο λεκτικός αναλυτής διαβάζει τους χαρακτήρες που συνθέτουν το source πρόγραμμα με τη σειρά από τα αριστερά προς τα δεξιά και τους ομαδοποιεί σε λεκτικές μονάδες που ονομάζονται *tokens*. Τα tokens είναι σειρές από χαρακτήρες που έχουν κάποιο νόημα να είναι ομαδοποιημένοι. Παραδείγματα από tokens αποτελούν οι δεσμευμένες λέξεις μιας γλώσσας, τα identifiers, οι ακέραιοι αριθμοί, οι δεκαδικοί, οι τελεστές και τα ειδικά σύμβολα. Οι περισσότεροι λεκτικοί αναλυτές χρησιμοποιούν *συνήθης εκφράσεις (regular expressions)* για την αναγνώριση των tokens. Οι σειρά από tokens που δημιουργείται κατά τη λεκτική ανάλυση δίνεται σαν είσοδος

στην επόμενη φάση, τη συντακτική ανάλυση. Η λεκτική ανάλυση λέγεται και *σάρωση (scanning).*

2. **Συντακτική ανάλυση (Parsing)**: Ο συντακτικός αναλυτής κάνει χρήση της γραμμικής σειράς από tokens που παρήγαγε ο scanner και τα ομαδοποιεί σε γραμματικές φράσεις. Η ομαδοποίηση αυτή γίνεται με βάση μια *context-free γραμματική*, ένα σύνολο κανόνων που καθορίζουν έγκυρους συνδυασμούς tokens και μη τερματικών συμβόλων που δέχεται η γλώσσα. Η συντακτική ανάλυση λέγεται και *parsing* ενώ οι γραμματικές φράσεις που δημιουργήθηκαν συνήθως αναπαρίστανται με ένα δέντρο. Το δέντρο αυτό το παράγει ο parser σαν έξοδο και ονομάζεται *syntax tree* ή *parse tree.*

3. **Σημασιολογική ανάλυση**: Στη σημασιολογική ανάλυση γίνεται χρήση των πληροφοριών του πίνακα συμβόλων ώστε να ελεγχθεί το parse tree για σημασιολογικά λάθη. Μία δήλωση μπορεί να είναι συντακτικά σωστή, να συμβαδίζει με έναν γραμματικό κανόνα, αλλά να παρακούει τους σημασιολογικούς κανόνες της source γλώσσας. Η φάση αυτή σχετίζεται κυρίως με το *type checking*, τον έλεγχο των τύπων των μεταβλητών. Ελέγχεται αν όλα τα μέρη ενός τελεστή είναι τύπου που επιτρέπεται από τη γλώσσα. Στη σημασιολογική ανάλυση ανακαλύπτονται σφάλματα όπως αδήλωτες μεταβλητές, η κλήση μίας συνάρτησης με λάθος αριθμό παραμέτρων, παραβάσεις πρόσβασης και λάθος τύποι δεδομένων στα μέρη ενός τελεστή.

4. **Δημιουργία ενδιάμεσου κώδικα**: Κατά τη διαδικασία της μετάφρασης ο μεταγλωττιστής μπορεί να δημιουργήσει μία ή και περισσότερες *ενδιάμεσες αναπαραστάσεις (Intermediate Representations, IRs)* του προγράμματος οι οποίες μπορούν να έχουν διάφορες μορφές. Οι ενδιάμεσες αναπαραστάσεις κατηγοριοποιούνται ανάλογα με το αν είναι πιο κοντά στη source ή την target γλώσσα. Μία τέτοια αναπαράσταση μπορεί να θεωρηθεί σαν ένα πρόγραμμα για μια αφηρημένη μηχανή, ανεξάρτητη από την πραγματική μηχανή. Μία IR πρέπει να είναι εύκολο να παραχθεί από το parse tree αλλά και να μεταφραστεί στο target πρόγραμμα.

Back-end (Σύνθεση)

5. **Βελτιστοποίηση ενδιάμεσης αναπαράστασης**: Η φάση αυτή δέχεται κάποια ενδιάμεση αναπαράσταση σαν είσοδο και παράγει μια βελτιστοποιημένη έκδοση της. Σκοπός των βελτιώσεων που πραγματοποιεί είναι το τελικό (target) πρόγραμμα που θα δημιουργηθεί να τρέχει πιο γρήγορα ή και να καταλαμβάνει και λιγότερο χώρο στη μνήμη. Πρόκειται για μια προαιρετική φάση καθώς μπορεί να κάνει τον μεταγλωττιστή αρκετά πιο αργό. Οι περισσότεροι μεταγλωττιστές έχουν κάποια ρύθμιση που την απενεργοποιεί.

6. **Δημιουργία τελικού κώδικα**: Στη φάση αυτή δημιουργείται το τελικό πρόγραμμα στην target γλώσσα. Σαν είσοδο παίρνει μια ενδιάμεση αναπαράσταση του αρχικού προγράμματος ενώ η έξοδος είναι συνήθως σε γλώσσα assembly ή κώδικα μηχανής. Εδώ αποφασίζονται οι θέσεις μνήμης για κάθε μεταβλητή και κάθε εντολή της ενδιάμεσης αναπαράστασης μετατρέπεται σε μια σειρά εντολών μηχανής που εκτελούν την ίδια λειτουργία.

7. **Βελτιστοποίηση τελικού κώδικα**: Στον τελικό κώδικα όπως και στην ενδιάμεση αναπαράσταση μπορεί προαιρετικά να εφαρμοστούν κάποιες βελτιστοποιήσεις. Εδώ λαμβάνονται υπόψη διάφορα χαρακτηριστικά του υλικού ώστε να ξαναοργανωθεί ο τελικός κώδικας.

Υπάρχουν και κάποιες δραστηριότητες που αλληλεπιδρούν με διάφορες φάσεις κατά τη διάρκεια και τον δύο σταδίων του μεταγλωττιστή. Οι δραστηριότητες αυτές είναι οι παρακάτω:

- **Διαχείριση του Πίνακα συμβόλων**: Κατά τη διάρκεια της ανάλυσης συλλέγονται πληροφορίες για το source πρόγραμμα και αποθηκεύονται σε μια δομή δεδομένων που λέγεται *πίνακας συμβόλων (symbol table)*. Στη σύνθεση οι πληροφορίες αυτές του πίνακα συμβόλων λαμβάνονται υπόψη για τη δημιουργία του target προγράμματος. Οι πληροφορίες αυτές αφορούν τις μεταβλητές που χρησιμοποιούνται στο source πρόγραμμα και διάφορα χαρακτηριστικά τους όπως ο τύπος και το πεδίο ισχύος τους. Μια μεταβλητή αναγνωρίζεται από τον λεκτικό αναλυτή και προστίθεται στον πίνακα συμβόλων. Στις δύο επόμενες φάσεις, της συντακτικής και σημασιολογικής ανάλυσης, ενημερώνεται η καταχώρηση της μεταβλητής στον πίνακα ώστε να περιλαμβάνει πληροφορίες για τον τύπο και το πεδίο ισχύος της.

- **Χειρισμός λαθών**: Κάθε φορά που διαπιστώνεται κάποια ανωμαλία στο πρόγραμμα εκδίδονται διαγνωστικά μηνύματα και ρυθμίζονται οι πληροφορίες που τροφοδοτούνται από την κάθε φάση στην επόμενη, ώστε κάθε φάση να μπορεί να συνεχίσει τη δουλειά της. Οι τρείς πρώτες φάσεις της ανάλυσης αντιμετωπίζουν το μεγαλύτερο μέρος των λαθών. Η φάση της λεκτικής ανάλυσης προσέχει για σκόρπια tokens, η συντακτική ανάλυση αναφέρει λάθος συνδυασμούς από tokens και η σημασιολογική ανάλυση αναφέρει λάθη που αφορούν τους τύπους των μεταβλητών και των δεδομένων. Η διαδικασία της μεταγλώττισης θα πρέπει να φτάνει μέχρι το τέλος της ώστε να ανακαλύπτονται όσο το δυνατόν περισσότερα από τα υπάρχοντα λάθη.

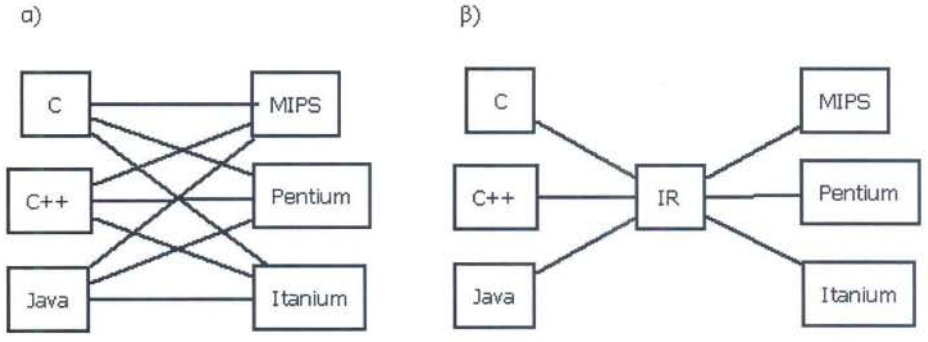
1.2 Ενδιάμεσες αναπαραστάσεις

Οι περισσότεροι μεταγλωττιστές μεταφράζουν τον πηγαίο κώδικα σε κάποια μορφή *ενδιάμεσης αναπαράστασης* και από εκεί κάνουν τη μετατροπή σε κώδικα

μηχανής. Μια ενδιάμεση αναπαράσταση είναι μια έκδοση του κώδικα ανεξάρτητη τόσο από τη source γλώσσα όσο και από τη μηχανή.

Μετά το στάδιο της ανάλυσης είναι εφικτό να γίνει η μετάφραση κατευθείαν σε κώδικα μηχανής, με τη χρήση όμως μιας ενδιάμεσης αναπαράστασης αυξάνεται η φορητότητα και επεκτασιμότητα. Υποθέτουμε πως ζητούνται μεταγλωττιστές για N source γλώσσες σε M target μηχανές. Όπως φαίνεται και στο σχήμα 1.2 χωρίς τη χρήση ενδιάμεσης αναπαράστασης θα χρειάζονταν να κατασκευαστούν N*M μεταγλωττιστές. Κάνοντας χρήση μιας κοινής ενδιάμεσης αναπαράστασης θα χρειάζονταν μόνο ένα front-end για κάθε source γλώσσα και ένα back-end για κάθε target μηχανή, N front-ends + M back-ends συνολικά. Ακόμα και στην περίπτωση ενός front-end και ενός back-end μια καλή IR μπορεί να διαχωρίσει τη διαδικασία ώστε το front-end να μην ασχολείται με λεπτομέρειες που αφορούν τη μηχανή και ομοίως το back-end να μην ασχολείται με πληροφορίες σχετικές με την source γλώσσα.

Οι περισσότερες βελτιστοποιήσεις κατά τη διάρκεια της μετάφρασης γίνονται όσο ο κώδικας βρίσκεται σε κάποια μορφή ενδιάμεσης αναπαράστασης. Οι βελτιστοποιήσεις αυτές έχουν τη δυνατότητα να είναι ανεξάρτητες από το back-end και τη μηχανή.



Σχήμα 1.2: Μεταγλωττιστές για τρείς γλώσσες υψηλού επιπέδου και τρείς target μηχανές: (α) χωρίς ενδιάμεση αναπαράσταση, (β) με ενδιάμεση αναπαράσταση.

Μια καλή ενδιάμεση αναπαράσταση πρέπει να έχει τις παρακάτω ιδιότητες:

- Να είναι εύκολο να παραχθεί από το front-end αλλά και βολικό να μετατραπεί σε κώδικα για οποιαδήποτε επιθυμητή target μηχανή.
- Να είναι απλή και να βγάζει εύκολα νόημα ώστε να είναι εύκολη η υλοποίηση βελτιστοποιήσεων πάνω της.

Οι ενδιάμεσες αναπαραστάσεις κατηγοριοποιούνται ανάλογα με το αν είναι πιο κοντά σε μία γλώσσα υψηλού επιπέδου ή πιο κοντά στον κώδικα μηχανής. Ένας μεταγλωττιστής μπορεί αντί να δημιουργήσει μία και μοναδική ενδιάμεση αναπαράσταση, να κάνει χρήση μιας διαδοχικής σειράς ενδιάμεσων αναπαραστάσεων. Έτσι έχουμε το διαχωρισμό τους σε:

- **Ενδιάμεσες αναπαραστάσεις υψηλού επιπέδου (High-level Intermediate Representations, HIRs)**: HIR ονομάζονται οι ενδιάμεσες αναπαραστάσεις που είναι κοντά σε μια γλώσσα υψηλού επιπέδου. Μια HIR συνήθως διατηρεί πληροφορίες όπως βρόγχοι επανάληψης, εντολές επιλογής, μεταβλητές, εκφράσεις και συναρτήσεις ενώ πολλές φορές είναι δυνατή και η μετατροπή πίσω στην source γλώσσα ή σε μία άλλη γλώσσα υψηλού επιπέδου. Μία τέτοια αναπαράσταση επιτρέπει βελτιστοποιήσεις βασισμένες στις ιδιότητες της source γλώσσας όπως memory dependence analysis και loop transformations.
- **Ενδιάμεσες αναπαραστάσεις μέσου επιπέδου (Medium-level Intermediate Representations, MIRs)**: Οι MIRs προσπαθούν να είναι ανεξάρτητες τόσο από τη source γλώσσα όσο και από τη μηχανή.
- **Ενδιάμεσες αναπαραστάσεις χαμηλού επιπέδου (Low-level Intermediate Representations, LIRs)**: Οι LIRs τείνουν αρκετά προς τον target κώδικα και για αυτό συχνά είναι και εξαρτώμενες από τη μηχανή.

Σχήμα 1.3: Ένας μεταγλωττιστής μπορεί να κάνει χρήση μιας ακολουθίας ενδιάμεσων αναπαραστάσεων.

Οι δύο σημαντικότεροι και πιο συχνά χρησιμοποιούμενοι τύποι ενδιάμεσων αναπαραστάσεων είναι οι παρακάτω:

- **Δέντρα και γράφοι**, συμπεριλαμβανομένων των parse trees και των (abstract) syntax trees (AST). Μια τέτοια αναπαράσταση μπορεί να θεωρηθεί το AST το οποίο δημιουργείται κατά τη διάρκεια της συντακτικής ανάλυσης. Πρόκειται ουσιαστικά για μια πολύ υψηλού επιπέδου ενδιάμεση αναπαράσταση. Καθώς το στάδιο της ανάλυσης προχωράει, προστίθενται πληροφορίες στους κόμβους του δέντρου με τη μορφή ιδιοτήτων.
- **Γραμμικές αναπαραστάσεις**, όπως ο "κώδικας τριών διευθύνσεων". Σε αντίθεση με μια IR δενδροειδούς μορφής ο κώδικας τριών διευθύνσεων δεν χρησιμοποιεί ιεραρχική δομή. Είναι μια ακολουθία της μορφής *x:=y op z* όπου *x*, *y* και *z* είναι μεταβλητές, σταθερές ή προσωρινές μεταβλητές που δημιούργησε ο μεταγλωττιστής και *op* είναι κάποιος τελεστής. Στον κώδικα τριών διευθύνσεων δεν επιτρέπονται πολύπλοκες αριθμητικές εκφράσεις. Είναι φανερό πως μία παράσταση της μορφής *x+y*z* δεν επιτρέπεται και θα παρασταθεί σαν δύο εκφράσεις: *t1:=y*z* και *t2:=x+t1* όπου τα *t1* και *t2* είναι προσωρινές μεταβλητές που δημιούργησε ο μεταγλωττιστής.
- Υπάρχουν και κάποιες **υβριδικές αναπαραστάσεις** οι οποίες συνδυάζουν τις δύο παραπάνω περιπτώσεις.

Κατηγοριοποιώντας την ενδιάμεση αναπαράσταση της οποίας γίνεται χρήση στην παρούσα εργασία, θα λέγαμε ότι είναι:

- Δενδροειδούς μορφής,
- Υψηλού επιπέδου ενδιάμεση αναπαράσταση.

Η συγκεκριμένη HIR αποτελεί ουσιαστικά ένα παράγωγο του AST. Το πρότυπο front-end που αναπτύχτηκε για αυτή την IR δημιουργεί αρχικά ένα AST και στη συνέχεια ελέγχει τους κόμβους του από τους οποίους λαμβάνει ορισμένους υπόψη για την παραγωγή της ενδιάμεσης αναπαράστασης. Μια υλοποίηση της συγκεκριμένης υψηλού επιπέδου ενδιάμεσης αναπαράστασης είναι διαθέσιμη σε μορφή C++ Static Library στον σχετικό ιστότοπο που αναφέρεται στην εισαγωγή της εργασίας. Η βιβλιοθήκη αυτή ονομάζεται "HLIR Library".

Κεφάλαιο 2

Χρήση του DOM μέσω του MSXML στη C++

2.1 Εισαγωγή

2.1.1 Τι είναι το MSXML

Το *Microsoft XML Core Services (MSXML)* είναι μια συλλογή υπηρεσιών που σκοπό έχουν την διευκόλυνση της χρήσης του XML. Πρόκειται στην ουσία για κάποιες διεπαφές, συμβατές με διάφορες γλώσσες προγραμματισμού, μέσα από τις οποίες υλοποιούνται διάφορες τεχνολογίες και τεχνικές (DOM, SAX κ.α.) για το χειρισμό XML δεδομένων.

2.1.2 Τι είναι το XML DOM

Το *Document Object Model (DOM)* όπως υλοποιείται στο MSXML παρέχει μία προγραμματιστική αναπαράσταση για έγγραφα XML, τμήματα εγγράφων XML, κόμβους, ή σύνολα κόμβων. Παρέχει επίσης και ένα *Application Programming Interface (API)* για την εργασία με XML δεδομένα. Για την XML αναπαράσταση ακολουθούνται οι προδιαγραφές του W3C για το DOM. Ως API τα XML DOM αντικείμενα είναι αντικείμενα COM και μπορούν να χρησιμοποιηθούν σε XML εφαρμογές γραμμένες σε γλώσσες προγραμματισμού όπως C/C++, Visual Basic, VBScript και Jscript.

2.1.3 Τρόπος λειτουργίας του DOM

Το MSXML διαβάζει ένα αρχείο XML και αναλύει τα περιεχόμενα του σε ένα σύνολο αντικειμένων που περιέχουν πληροφορία τα οποία ονομάζονται *κόμβοι (nodes)*. Οι κόμβοι αυτοί αντιπροσωπεύουν τη δομή και το περιεχόμενο του αρχείου, επιτρέποντας στις εφαρμογές να διαβάζουν και να ελέγχουν τις πληροφορίες του αρχείου χωρίς να κατανοούν ουσιαστικά το XML συντακτικό. Αφού

διαβαστεί και αναλυθεί ένα αρχείο γίνεται δυνατή η πλοήγηση σε αυτό προς οποιαδήποτε κατεύθυνση.

2.2 Προετοιμασία για εργασία με MSXML/DOM/C++

Ο πιο απλός τρόπος εργασίας με το συνδυασμό MSXML/DOM/C++ είναι κάνοντας χρήση του *Ολοκληρωμένου Περιβάλλοντος Ανάπτυξης (IDE) Visual Studio* της Microsoft. Κάνοντας χρήση του συγκεκριμένου εργαλείου επιτυγχάνεται καλύτερη οργάνωση του κώδικα και του έργου ενώ αυτοματοποιούνται και διάφορες χρήσιμες εργασίες. Η έκδοση του Visual Studio της οποίας γίνεται χρήση στην παρούσα εργασία είναι η 2008.

Για να εργαστούμε με το MSXML χρειάζεται να έχουμε:

- Μια κατάλληλη έκδοση του MSXML εγκατεστημένη στον υπολογιστή μας.
- Τις βιβλιοθήκες του MSXML ενσωματωμένες στο project του Visual Studio.

Αναλυτικά:

Για να επιβεβαιώσουμε ότι το MSXML είναι εγκατεστημένο ψάχνουμε για τις βιβλιοθήκες **msxmlX.dll** στο φάκελο *$sysRoot\system32*, όπου X είναι ο αριθμός της έκδοσης του MSXML και $sysRoot είναι ο φάκελος του συστήματος. Εξ ορισμού ο φάκελος αυτός είναι ο *c:\windows* για μηχανήματα που τρέχουν windows XP, Vista ή 7. Για παράδειγμα εγκαθιστώντας το MSXML 6.0 τοποθετούνται στο φάκελο του συστήματος τα αρχεία **msxml6.dll**, **msxml6a.dll** και **msxml6r.dll** και καταχωρούνται και στη registry του συστήματος.

Αφού εγκατασταθεί το MSXML πρέπει να ρυθμιστεί το project της εφαρμογής ώστε οι κλήσεις προς τα APIs που υποστηρίζονται από το MSXML να εκτελούνται σωστά κατά την εκτέλεση της. Στη Visual C++ πρέπει να γίνουν import τα MSXML headers και βιβλιοθήκες στο project.

Υπάρχουν δύο τρόποι για να ρυθμιστεί μια εφαρμογή ώστε να κάνει χρήση των DOM διεπαφών: Μπορούμε να συμπεριλάβουμε τα κατάλληλα headers και βιβλιοθήκες χειροκίνητα ή αυτόματα.

2.2.1 Χειροκίνητη προσθήκη των headers και libraries

1. Εντοπισμός του φακέλου στον οποίο είναι εγκατεστημένο το MSXML SDK. Εξ ορισμού, για το MSXML 6.0, ο φάκελος αυτός είναι ο *C:\ Program Files\ MSXML 6.0*, με τους υποφάκελους inc και lib. Στα windows 7 ο φάκελος αυτός είναι ο *C:\ Program Files\ Microsoft SDKs\ Windows\ v6.0A* με τους υποφακέλους Include και Lib αντίστοιχα.

2. Ρύθμιση του include directory στο Visual Studio: Από το μενού **Tools** επιλογή του **Options**. Στο παράθυρο διαλόγου **Options** επέκταση του **Projects and Solutions**. Επιλογή του **VC++ Directories**. Επιλογή του **Include files** από το drop down **Show directories for**. Κλικ στο κουμπί **New Line** για

προσθήκη νέου φακέλου στη λίστα. Αναζήτηση και επιλογή του φακέλου που περιέχει το header αρχείο του MSXML. Για το MSXML 6.0 ο κατάλογος αυτός είναι εξ ορισμού ο *C:\ Program Files\ MSXML 6.0\ inc.* Ο αντίστοιχος κατάλογος στα windows 7 είναι ο *C:\ Program Files\ Microsoft SDKs\ Windows\ v6.0A\ Include.*

3. Ρύθμιση του lib directory στο Visual Studio: Από το μενού **Tools** επιλογή του **Options**. Στο παράθυρο διαλόγου **Options** επέκταση του **Projects and Solutions**. Επιλογή του **VC++ Directories**. Επιλογή του **Library files** από το drop down **Show directories for**. Κλικ στο κουμπί **New Line** για προσθήκη νέου φακέλου στη λίστα. Αναζήτηση και επιλογή του φακέλου που περιέχει τις MSXML βιβλιοθήκες. Για το MSXML 6.0 ο κατάλογος αυτός είναι εξ ορισμού ο *C:\ Program Files\ MSXML 6.0\ lib.* Ο αντίστοιχος κατάλογος στα windows 7 είναι ο *C:\ Program Files\ Microsoft SDKs\ Windows\ v6.0A\ Lib.*

4. Μετατροπή του project ώστε να κάνει link με την MSXML 6.0 βιβλιοθήκη: Από το **Solutions Explorer**, δεξί κλικ στο project και επιλογή του **Properties**. Επέκταση του **Configuration Properties** και του **Linker** και επιλογή του **Command Line**. Εισαγωγή του **msxml6.lib** στο πλαίσιο κειμένου **Additional options**.

5. Προσθήκη των ακόλουθων γραμμών στον κώδικα για να συμπεριληφθεί το κατάλληλο header αρχείο του MSXML.

```
#include <objbase.h>
#include <msxml6.h>
```

2.2.2 Αυτόματη προσθήκη των headers και libraries

- Προσθήκη των δυο ακόλουθων γραμμών στον κώδικα:

```
#import <msxml6.dll> raw_interfaces_only
using namespace MSXML2;
```

Μπορούν να συμπεριληφθούν headers και βιβλιοθήκες και άλλων εκδόσεων του MSXML με τον ίδιο τρόπο, χρησιμοποιώντας το κατάλληλο όνομα για το DLL της εκάστοτε έκδοσης (π.χ. msxml3.dll, msxml4.dll, msxml5.dll, κτλ.) στο import directive.

Η Microsoft προτείνει τα headers και οι βιβλιοθήκες να συμπεριλαμβάνονται χειροκίνητα αντί να γίνεται χρήση του #import directive.

2.3 Εκδόσεις MSXML

Ο ακόλουθος πίνακας παραθέτει τις εκδόσεις του MSXML, τα αντίστοιχα ονόματα των header αρχείων, των αρχείων lib και των DLL.

Έκδοση MSXML	Όνομα αρχείου Header	Όνομα αρχείου Lib	Όνομα αρχείου DLL
.x	msxml.h	msxml.lib	msxml2.dll
3.0	msxml2.h	msxml2.lib	msxml3.dll
4.0	msxml2.h	msxml2.lib	msxml4.dll
6.0	msxml6.h	msxml6.lib	msxml6.dll

Πίνακας 2.1: Εκδόσεις MSXML.

Η σύνδεση με τις βιβλιοθήκες και τα header αρχεία γίνεται με βάση τον παραπάνω πίνακα ανάλογα με την έκδοση του MSXML της οποίας γίνεται χρήση. Στην παρούσα εργασία γίνεται χρήση της έκδοσης 6.0.

Η έκδοση 6.0 του MSXML συμπεριλαμβάνεται στα παρακάτω πακέτα λογισμικού της Microsoft:

- Microsoft SQL Server 2005,
- Visual Studio 2005,
- .NET Framework 3.0,
- Windows Vista,
- Windows 7,
- Windows XP Service Pack 3

Για παλαιότερα λειτουργικά συστήματα στα οποία δεν συμπεριλαμβάνεται το MSXML 6.0 υπάρχει η δυνατότητα χρήσης του μέσω αρχείου εγκατάστασης που προσφέρεται από τον ιστότοπο της Microsoft[1]. Η συγκεκριμένη εγκατάσταση είναι κατάλληλη για τις παρακάτω εκδόσεις του λειτουργικού συστήματος Windows:

- Windows 2000 Service Pack 4,
- Windows Server 2003,
- Windows Server 2003 Service Pack 1,
- Windows XP Service Pack 1,
- Windows XP Service Pack 2

[1]http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=3988

2.4 MSXML DOM API

Ακολουθεί μια τεκμηρίωση των βασικών προγραμματιστικών διεπαφών του MSXML για το DOM. Αναλύονται τα βασικά Objects/Interfaces, Properties, Methods και Enumerated Constants που παρέχει το MSXML δίνοντας κυρίως έμφαση στην υλοποίηση τους στην C++. Για μια πλήρη εικόνα όλων των MSXML DOM διεπαφών μπορείτε να επισκεφθείτε την επίσημη τεκμηρίωση της Microsoft[2].

Οι διεπαφές που χρησιμοποιήθηκαν στην παρούσα εργασία είναι οι παρακάτω:

Enumerated Constants

- IXMLDOMNodeType

Objects/Interfaces, Properties και Μέθοδοι

- IXMLDOMDocument
  - documentElement
  - load(xmlSource)
  - parseError
- IXMLDOMElement
- IXMLDOMNode
  - GetnodeType()
  - GetnodeTypeString()
  - GetnodeName()
  - GetchildNodes()
  - Getattributes()
  - Gettext()
  - hasChildNodes()
- IXMLDOMNodeList
  - Getlength()
  - Getitem(itemindex)
- IXMLDOMNamedNodeMap
  - Getlength()

[2]http://msdn.microsoft.com/en-us/library/windows/desktop/ms764730%28v=VS.85%29.aspx

- Getitem(itemindex)
- GetNameditem(itemname)

• IXMLDOMParseError

- Getreason()

Αναλυτικά:

2.4.1 Enumerated Constants

IXMLDOMNodeType

Το Enumerated Constant *IXMLDOMNodeType* καθορίζει τους έγκυρους τύπους που μπορεί να έχουν οι κόμβοι του DOM. Κάθε κόμβος μπορεί να έχει έναν μόνο τύπο και ανάλογα με τον τύπο αυτό καθορίζονται και παράμετροι όπως το αν ο κόμβος θα μπορεί να έχει παιδιά. Οι enumerators του *IXMLDOMNodeType* παρουσιάζονται παρακάτω:

- **NODE_ELEMENT (1)**: Ένας κόμβος αυτού του τύπου αντιπροσωπεύει ένα element. Η ιδιότητα *nodeTypeString* του κόμβου έχει την τιμή "*element*". Ένας κόμβος τύπου Element μπορεί να έχει τους ακόλουθους τύπους κόμβων ως παιδιά: Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference. Ένας κόμβος Element μπορεί να είναι παιδί των: Document, DocumentFragment, EntityReference, Element.

- **NODE_ATTRIBUTE (2)**: Ένας κόμβος αυτού του τύπου αντιπροσωπεύει ένα attribute ενός element. Η ιδιότητα *nodeTypeString* του κόμβου έχει την τιμή "*attribute*". Ένας κόμβος τύπου Attribute μπορεί να έχει τους ακόλουθους τύπους κόμβων ως παιδιά: Text, EntityReference. Ένας κόμβος Attribute δεν μπορεί να είναι παιδί κάποιου άλλου κόμβου. Δεν θεωρείται παιδί ενός Element παρόλο που συνδέεται με αυτό.

- **NODE_TEXT (3)**: Ένας κόμβος αυτού του τύπου αντιπροσωπεύει το κείμενο που περιλαμβάνεται σε ένα element. Η ιδιότητα *nodeTypeString* του κόμβου έχει την τιμή "*text*". Ένας κόμβος τύπου Text δεν μπορεί να έχει άλλους κόμβους ως παιδιά ενώ μπορεί να είναι παιδί των: Attribute, DocumentFragment, Element, EntityReference.

- **NODE_CDATA_SECTION (4)**: Ένας κόμβος αυτού του τύπου αντιπροσωπεύει μια ενότητα CDATA στο XML. Η ιδιότητα *nodeTypeString* του κόμβου έχει την τιμή "*cdatasection*". Οι ενότητες CDATA χρησιμοποιούνται για να συμπεριλάβουμε κείμενο το οποίο σε διαφορετική περίπτωση θα αναγνωρίζονταν σαν XML κώδικας. Ένας κόμβος τύπου CDATASection δεν

μπορεί να έχει άλλους κόμβους ως παιδιά ενώ μπορεί να είναι παιδί των: DocumentFragment, EntityReference, Element.

- **NODE_ENTITY_REFERENCE (5)**: Ένας κόμβος αυτού του τύπου αντιπροσωπεύει μια αναφορά σε ένα entity στο XML έγγραφο. Η ιδιότητα *nodeTypeString* του κόμβου έχει την τιμή "*entityreference*". Ένας κόμβος τύπου EntityReference μπορεί να έχει τους ακόλουθους τύπους κόμβων ως παιδιά: Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference. Ένας κόμβος EntityReference μπορεί να είναι παιδί των: Attribute, DocumentFragment, Element, EntityReference.

- **NODE_ENTITY (6)**: Ένας κόμβος αυτού του τύπου αντιπροσωπεύει ένα entity. Η ιδιότητα *nodeTypeString* του κόμβου έχει την τιμή "*entity*". Ένας κόμβος τύπου Entity μπορεί να έχει ως παιδιά κόμβους οι οποίοι το αναπαριστούν (για παράδειγμα κόμβους Text και EntityReference) ενώ μπορεί να εμφανιστεί σαν παιδί του κόμβου DocumentType.

- **NODE_PROCESSING_INSTRUCTION (7)**: Ένας κόμβος αυτού του τύπου αντιπροσωπεύει ένα processing instruction του XML αρχείου. Η ιδιότητα *nodeTypeString* του κόμβου έχει την τιμή "*processinginstruction*". Ένας κόμβος τύπου ProcessingInstruction δεν μπορεί να έχει άλλους κόμβους ως παιδιά ενώ μπορεί να είναι παιδί των: Document, DocumentFragment, Element, EntityReference.

- **NODE_COMMENT (8)**: Ένας κόμβος αυτού του τύπου αντιπροσωπεύει ένα σχόλιο στο XML αρχείο. Η ιδιότητα *nodeTypeString* του κόμβου έχει την τιμή "*comment*". Ένας κόμβος τύπου Comment δεν μπορεί να έχει άλλους κόμβους ως παιδιά ενώ μπορεί να είναι παιδί των: Document, DocumentFragment, Element, EntityReference.

- **NODE_DOCUMENT (9)**: Ένας κόμβος αυτού του τύπου αντιπροσωπεύει ένα κεντρικό αντικείμενο το οποίο λειτουργεί σαν ρίζα του XML δέντρου και παρέχει πρόσβαση σε όλο το XML έγγραφο. Η ιδιότητα *nodeTypeString* του κόμβου έχει την τιμή "*document*". Ένας κόμβος τύπου Document μπορεί να έχει τους ακόλουθους τύπους κόμβων ως παιδιά: Element (το πολύ ένα), ProcessingInstruction, Comment, DocumentType. Ένας κόμβος Document δεν μπορεί να εμφανιστεί σαν παιδί κάποιου άλλου κόμβου.

- **NODE_DOCUMENT_TYPE (10)**: Ένας κόμβος αυτού του τύπου αντιπροσωπεύει μία δήλωση document type η οποία υποδεικνύεται από το <!DOCTYPE> tag. Η ιδιότητα *nodeTypeString* του κόμβου έχει την τιμή "*documenttype*". Ένας κόμβος τύπου DocumentType μπορεί να έχει τους ακόλουθους τύπους κόμβων ως παιδιά: Notation, Entity. Ένας κόμβος DocumentType μπορεί να εμφανιστεί σαν παιδί του κόμβου Document.

- **NODE_DOCUMENT_FRAGMENT (11)**: Ένας κόμβος αυτού του τύπου αντιπροσωπεύει ένα document fragment. Η ιδιότητα *nodeTypeString* του κόμβου έχει την τιμή "*documentfragment*". Ένα document fragment συνδέει έναν κόμβο ή ένα υποδέντρο με ένα έγγραφο χωρίς να συμπεριλαμβάνεται πραγματικά σε αυτό. Ένας κόμβος τύπου DocumentFragment μπορεί να έχει τους ακόλουθους τύπους κόμβων ως παιδιά: Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference. Ένας κόμβος DocumentFragment δεν μπορεί να εμφανιστεί σαν παιδί κάποιου άλλου κόμβου.

- **NODE_NOTATION (12)**: Ένας κόμβος αυτού του τύπου αντιπροσωπεύει ένα notation στη δήλωση του document type. Η ιδιότητα *nodeTypeString* του κόμβου έχει την τιμή "*notation*". Ένας κόμβος τύπου Notation δεν μπορεί να έχει άλλους κόμβους ως παιδιά ενώ μπορεί να εμφανιστεί σαν παιδί του κόμβου DocumentType.

2.4.2 Objects/Interfaces, Properties και Μέθοδοι

IXMLDOMDocument/DOMDocument

Το αντικείμενο αυτό αντιπροσωπεύει το ανώτατο επίπεδο του XML αρχείου. Περιλαμβάνει μέλη για ανάκτηση και δημιουργία όλων των άλλων XML αντικειμένων.

Όταν σε ένα αρχείο-αντικείμενο *DOMDocument* γίνεται χρήση μιας μεθόδου που δημιουργεί κάποιο αντικείμενο (όπως η *createElement*) οι κόμβοι (nodes) που δημιουργούνται είναι συσχετισμένοι με το αρχείο (η ιδιότητα *ownerDocument* του κόμβου δείχνει στο αρχείο) αλλά ο κόμβος αυτός δεν είναι μέρος του δέντρου του αρχείου. Ο κόμβος γίνεται μέρος του δέντρου του αρχείου μόνο όταν προστίθεται σε αυτό καλώντας τις μεθόδους *insertBefore, replaceChild* ή *appendChild* (ή αν πρόκειται για attribute και την *setAttributeNode*).

Η κλάση *DOMDocument* αντιπροσωπεύει τον ανώτατο κόμβο στο XML δέντρο. Μόνο ένα αντικείμενο τέτοιου τύπου μπορεί να δημιουργηθεί για ένα αρχείο XML: το αρχείο (document). Η πρόσβαση σε όλα τα υπόλοιπα αντικείμενα ή και η δημιουργία τους είναι δυνατή μέσω του αντικειμένου αυτού.

Μέθοδοι και ιδιότητες του *IXMLDOMDocument* των οποίων έγινε χρήση στην παρούσα εργασία:

- **documentElement**: Περιέχει τη ρίζα του XML αρχείου. Επιστρέφει ένα *IXMLDOMElement* το οποίο αναπαριστά το element της ρίζας του XML δέντρου. Επιστρέφει *Null* αν δεν υπάρχει ρίζα.

- **load(xmlSource)**: Φορτώνει ένα έγγραφο XML από την τοποθεσία που του καθορίζουμε.

- **parseError**: Επιστρέφει ένα αντικείμενο *IXMLDOMParseError* το οποίο περιλαμβάνει πληροφορίες σχετικά με το τελευταίο σφάλμα που παρουσιάστηκε.

IXMLDOMElement

Το *IXMLDOMElement* αντιπροσωπεύει ένα element. Οι κόμβοι τύπου element είναι από τα πιο συχνά εμφανιζόμενα αντικείμενα στο δέντρο ενός XML εγγράφου. Μπορούν να έχουν attributes συσχετισμένα με αυτά. Τα attributes όμως δεν ορίζονται ως απόγονοι ενός element και δεν θεωρούνται μέρος του δέντρου του εγγράφου. Για το λόγο αυτό το *IXMLDOMElement* παρέχει μεθόδους για τη διευκόλυνση της διαχείρισης των attributes.

Για την πρόσβαση στα attributes τα οποία συσχετίζονται με ένα element υπάρχει η δυνατότητα μέσω της μεθόδου *Getattributes()* να επιστραφεί μια συλλογή τύπου *IXMLDOMNamedNodeMap* η οποία θα περιλαμβάνει όλα τα attributes του element.

IXMLDOMNode

Το *IXMLDOMNode* αντιπροσωπεύει έναν απλό κόμβο στο δέντρο του εγγράφου. Το αντικείμενο αυτό αποτελεί τη βασική διεπαφή για την προσπέλαση δεδομένων μέσω του DOM. Η διεπαφή αυτή υποστηρίζει τύπους δεδομένων, namespaces, DTDs και XML schemas.

Μέθοδοι του *IXMLDOMNode* των οποίων έγινε χρήση στην παρούσα εργασία:

- **GetnodeType()**: Επιστρέφει τον τύπο του κόμβου σε μορφή *IXMLDOMNodeType*.
- **GetnodeTypeString()**: Επιστρέφει τον τύπο του κόμβου σε μορφή αλφαριθμητικού (*bstr_t*).
- **GetnodeName()**:Επιστρέφει το όνομα του κόμβου σε μορφή αλφαριθμητικού (*bstr_t*).
- **GetchildNodes()**: Επιστρέφει ένα *IXMLDOMNodeListPtr* το οποίο αντιπροσωπεύει όλους τους κόμβους που είναι απευθείας απόγονοι του κόμβου για τον οποίο καλείται η μέθοδος.
- **Getattributes()**: Επιστρέφει ένα *IXMLDOMNamedNodeMap* το οποίο αντιπροσωπεύει όλα τα attributes του κόμβου για τον οποίο καλείται η μέθοδος.
- **Gettext()**: Επιστρέφει το περιεχόμενο ενός κόμβου σε απλό κείμενο.
- **hasChildNodes()**: Επιστρέφει Boolean, *true* αν ο κόμβος έχει έστω και έναν άμεσο απόγονο ή *false* αν δεν έχει κανέναν.

IXMLDOMNodeList

Το *IXMLDOMNodeList* είναι μια συλλογή από κόμβους (*IXMLDOMNode*) στους οποίους επιτρέπεται η άμεση πρόσβαση μέσω ευρετηρίου και επαναληπτικής προσπέλασης.

Μέθοδοι του *IXMLDOMNodeList* των οποίων έγινε χρήση στην παρούσα εργασία:

- **Getlength()**: Επιστρέφει ένα long integer το οποίο αντιπροσωπεύει τον αριθμό των κόμβων της συλλογής.
- **Getitem(itemindex)**: Παίρνει ένα long integer ως παράμετρο το οποίο αντιπροσωπεύει τη θέση ενός κόμβου στο ευρετήριο της συλλογής και επιστρέφει ένα *IXMLDOMNode* το οποίο αντιπροσωπεύει τον κόμβο αυτό.

IXMLDOMNamedNodeMap

Το *IXMLDOMNamedNodeMap* είναι μια συλλογή από κόμβους (nodes) στους οποίους επιτρέπει την πρόσβαση ονομαστικά ή με βάση τη σειρά τους. Η συλλογή αυτή χρησιμοποιείται για την πρόσβαση στα Attributes ενός Element. Ένα *IXMLDOMNamedNodeMap* δηλαδή συνήθως αντιπροσωπεύει μια συλλογή με όλα τα Attributes ενός Element. Κάθε Attribute αποτελεί και έναν κόμβο.

Σύμφωνα με τον ορισμό του World Wide Web Consortium (W3C) για το Document Object Model (DOM) η σειρά των κόμβων σε ένα *NamedNodeMap* δεν είναι αναγκαίο να διατηρηθεί αμετάβλητη. Η υλοποίηση της Microsoft διατηρεί τη σειρά των Attributes όπως εμφανίζονται και στον αρχικό XML κώδικα. Τα Attributes που προσθέτονται μπαίνουν στο τέλος της λίστας.

Όπως και στη λίστα κόμβων (*IXMLDOMNamedNodeList*), σε ένα *NamedNodeMap* η προσθήκη και διαγραφή κόμβων καθώς και οι μετατροπές εντός των κόμβων αντανακλώνται αμέσως στην συλλογή.

Όταν στο σχήμα του XML (DTD σχήμα για παράδειγμα) έχει οριστεί σε ένα attribute μια τιμή ως προεπιλογή και γίνει διαγραφή αυτού του attribute τότε ξαναδημιουργείται αυτόματα έχοντας ως τιμή αυτή που ορίζει το σχήμα ως προεπιλογή.

Μέθοδοι του *IXMLDOMNamedNodeMap* των οποίων έγινε χρήση στην παρούσα εργασία:

- **Getlength()**: Επιστρέφει ένα long integer το οποίο αντιπροσωπεύει τον αριθμό των κόμβων της συλλογής.
- **Getitem(itemindex)**: Παίρνει ένα long integer ως παράμετρο το οποίο αντιπροσωπεύει τη θέση ενός κόμβου στο ευρετήριο της συλλογής και επιστρέφει ένα *IXMLDOMNode* το οποίο αντιπροσωπεύει τον κόμβο αυτό.
- **GetNamedItem(itemname)**: Παίρνει ένα αλφαριθμητικό (*bstr_t*) ως παράμετρο και επιστρέφει ένα *IXMLDOMNode* το όνομα του οποίου είναι ίδιο με το αλφαριθμητικό.

IXMLDOMParseError

Το *IXMLDOMParseError* περιέχει αναλυτικές πληροφορίες για το τελευταίο σφάλμα κατά τη σάρωση του XML.

Μέθοδοι του *IXMLDOMParseError* των οποίων έγινε χρήση στην παρούσα εργασία:

- **Getreason()**: Επιστρέφει μια περιγραφή του σφάλματος.

2.5 Smart Pointers

Ένας *smart pointer* είναι ένας τύπος δεδομένων ο οποίος μιμείται έναν δείκτη ενώ παράλληλα προσφέρει και επιπλέον χαρακτηριστικά που έχουν να κάνουν κυρίως με τη διαχείριση της μνήμης. Τα επιπλέον αυτά χαρακτηριστικά έχουν στόχο τη μείωση των σφαλμάτων που συμβαίνουν λόγο κακής χρήσης των δεικτών.

Με τη χρήση των smart pointers ο παραγόμενος κώδικας απλουστεύεται. Η χρήση των smart pointer MSXML αντικειμένων είναι ευκολότερη από τη χρήση δεικτών σε απλά MSXML αντικείμενα αφού μοιάζουν με την αντίστοιχη σύνταξη των διεπαφών αυτών σε γλώσσες όπως C# και Visual Basic. Αυτό είναι χρήσιμο ειδικά για προγραμματιστές που χρησιμοποιούν συχνά αυτές τις γλώσσες.

Όλες οι διεπαφές του MSXML για το DOM είναι διαθέσιμες και ως smart pointers προσθέτοντας την κατάληξη **Ptr** στο όνομα τους.

Για παράδειγμα οι διεπαφές που αναφέρθηκαν στην προηγούμενη παράγραφο αντιστοιχούν στις ακόλουθες smart pointer κλάσεις:

Smart Pointer Classes

- IXMLDOMDocumentPtr
- IXMLDOMElementPtr
- IXMLDOMNodePtr
- IXMLDOMNodeListPtr
- IXMLDOMNamedNodeMapPtr
- IXMLDOMParseErrorPtr

2.6 Παράδειγμα - ένας απλός XML Parser

Για να γίνει πιο κατανοητός ο τρόπος λειτουργίας του DOM στη C++ μέσω του MSXML αναπτύχθηκε ως επίδειξη των σχετικών διεπαφών μια απλή εφαρμογή που αναλύει ένα έγγραφο XML.

```
#include <iostream>
#include <stdio.h>

#include <string>
#include <comdef.h> // Gia ti metatropi apo _bstr_t se std::string
    .

#import <msxml6.dll>

using namespace std;
using namespace MSXML2;

// Synartisi pou typonei ton typo enos komvou kathos kai to
// periexomeno tou an prokeitai gia text, CDATA h sxolio.
void printNodeType(MSXML2::IXMLDOMNodePtr xNode, MSXML2::
    DOMNodeType xNodeType)
{
  cout << "Node Type: ";
  switch(xNodeType) {
    case MSXML2::NODE_TEXT:
      // Text komvos.
      cout << "-Node TEXT-" << endl
        << "-Content:" << xNode->Gettext();
      break;
    case MSXML2::NODE_CDATA_SECTION:
      // CDATA komvos.
      cout << "-Node CDATA-" << endl
        << "-Content:" << xNode->Gettext();
      break;
    case MSXML2::NODE_COMMENT:
      // Komvos sxoliou.
      cout << "-Node COMMENT-" << endl
        << "-Content:" << xNode->Gettext();
      break;
    case MSXML2::NODE_ELEMENT:
      // Element.
      cout << "-Node ELEMENT-";
      break;
    default:
      break;
  }
}

// Synarthsh pou typvnei ola ta attributes enos komvou.
void printNodeAttributes(MSXML2::IXMLDOMNodePtr xNode)
{
  // Anakthsh twn attributes tou xNode.
  MSXML2::IXMLDOMNamedNodeMapPtr xNodeAttributesList;
  xNodeAttributesList = xNode->Getattributes();

  // Anakthsh tou plithous twn attributes tou xNode.
  long numAttributes = xNodeAttributesList->Getlength();
```

```
52    // Ektypwsh twn attributes (ean yparxei toulaxiston ena).
      if(numAttributes>0) {
54      MSXML2::IXMLDOMNodePtr xNodeAttribute;

56      cout << endl << "Attributes:" << endl;

58      string xNodeAttributeNameString;
        string xNodeAttributeValue;
60      for(int i=0; i<numAttributes; i++) {
          xNodeAttribute = xNodeAttributesList->Getitem(i);
62        xNodeAttributeNameString =
            _bstr_t(xNodeAttribute->GetnodeName());
64
          cout << "-" << xNodeAttributeNameString << " = "
66             << xNodeAttribute->Gettext() << endl;
        }
68    }
  }
70
  // Anadromiki synarthsh pou typwnei plhrofories gia enan komvo.
72 void recursiveNodeParse(MSXML2::IXMLDOMNodePtr xNode, int Level)
  {
74    Level++;

76    // Anakthsh tou typou tou xNode se morfh DOMNodeType.
      MSXML2::DOMNodeType xNodeType;
78    xNodeType = xNode->GetnodeType();

80    // Anakthsh tou typou tou xNode se morfh std::string.
      string xNodeTypeString;
82    xNodeTypeString = _bstr_t(xNode->GetnodeTypeString());

84    // Anakthsh tou onomatos tou xNode.
      string xNodeNameString;
86    xNodeNameString = _bstr_t(xNode->GetnodeName());

88    // Ektypwsh twn stoixeiwn tou xNode.
      cout << "Level: " << Level << endl;
90    cout << "Node Name: " << xNodeNameString << endl;
      //cout << "Node Type: " << xNodeTypeString << endl;
92    printNodeType(xNode, xNodeType);

94    // Ektypwsh twn attributes tou komvou (an prokeitai gia element)
        .
      if(xNodeType == MSXML2::NODE_ELEMENT) {
96      printNodeAttributes(xNode);
      }
98
      // Ean o komvos DEN exei paidia
100   // stamataei h synarthsh kai epistrefei.
      if(!xNode->hasChildNodes()){
102     cout << endl << "Node has 0 Child nodes." << endl
          << "-----------------------------------" << endl;
104     return;
```

```
  }
106
  // Anakthsh twn paidiwn tou xNode.
108 MSXML2::IXMLDOMNodeListPtr nodelist;
  nodelist = xNode->GetchildNodes();
110
  // Anakthsh tou plithous twn paidiwn tou xNode.
112 long numChildNodes = nodelist->Getlength();

114 cout << endl << "Node has "
       << numChildNodes << " Child nodes." << endl
116    << "-------------------------------" << endl;

118 if(numChildNodes<1) {
    return;
120 }

122 // Anadromiki klhsh ths recursiveNodeParse
    // gia ola ta paidia tou xNode.
124 MSXML2::IXMLDOMNodePtr childNode;
  for(int i=0; i<numChildNodes; i++) {
126   childNode = nodelist->Getitem(i);
    recursiveNodeParse(childNode, Level);
128 }
}
130
int main()
132 {
  // Arxikopoihsh ths COM vivliothikis.
134 CoInitialize(NULL);
  /*
136 HRESULT hr = CoInitialize(NULL);
  if(FAILED(hr)) {
138   cout << endl << "Could not initialize the COM library!"
         << endl;
140   return 0;
  }
142 */

144 {
    // Dhmioyrgia enos antikeimenou IXMLDOMDocumentPtr.
146   MSXML2::IXMLDOMDocumentPtr pXMLDom;
    pXMLDom.CreateInstance(__uuidof(MSXML2::DOMDocument60),
148           NULL, CLSCTX_INPROC_SERVER);
    /*
150   HRESULT hr2 =
    pXMLDom.CreateInstance(__uuidof(MSXML2::DOMDocument60),
152           NULL, CLSCTX_INPROC_SERVER);
    if (FAILED(hr2)) {
154   cout << endl << "Failed to instantiate an XML DOM!"
         << endl;
156   return;
    }
158 */
```

```
160     try {
          // Anoigma tou XML arxeiou.
162       if(pXMLDom->load("test.xml") == VARIANT_TRUE) {
            // Anakthsh tou komvou-riza tou XML dentrou.
164         MSXML2::IXMLDOMElementPtr pRoot =
                pXMLDom->documentElement;
166
            int Level = 0;
168         recursiveNodeParse(pRoot, Level);
          } else {
170         // An to anoigma tou XML arxeiou den einai
            // epityxes typose to parakato minima.
172         printf("Failed to load DOM from the xml file.
              %s\n",
174           (LPCSTR)pXMLDom->parseError->Getreason());
          }
176
        } catch(_com_error errorObject) {
178       printf("Exception thrown, HRESULT: 0x%08x",
                errorObject.Error());
180     }
    }
182
    // Kleisimo ths COM vivliothikis.
184 CoUninitialize();
    return 0;
186 }
```

Listing 2.1: Εφαρμογή που αναλύει ένα έγγραφο XML.

Για να γίνει εφικτή η χρήση των διεπαφών του MSXML συμπεριλαμβάνεται αρχικά το αρχείο "*msxml6.dll*" όπως φαίνεται στις πρώτες γραμμές του κώδικα.

Στην εφαρμογή διακρίνονται εκτός από την κεντρική και άλλες τρείς συναρτήσεις. Η συνάρτηση ***recursiveNodeParse***, η ***printNodeType*** και η ***printNodeAttribues***.

Στην κεντρική συνάρτηση δημιουργείται ένα αντικείμενο *pXMLDom* τύπου *IXMLDOMDocumentPtr* και μέσω της μεθόδου *load(xmlSource)* του αντικειμένου αυτού φορτώνεται το αρχείο "*test.xml*". Δημιουργείται επίσης και το αντικείμενο *pRoot* τύπου *IXMLDOMElementPtr* το οποίο αντιπροσωπεύει τον κεντρικό κόμβο του XML δέντρου και του εκχωρείται η τιμή του πεδίου *documentElement* του *pXMLDom*. Στη συνέχεια καλείται η συνάρτηση *recursiveNodeParse* δίνοντας της ως παράμετρο το *pRoot* ώστε να εκκινήσει η αναδρομική σάρωση όλων των κόμβων του XML δέντρου.

Η συνάρτηση ***recursiveNodeParse*** εκτελείται αναδρομικά με σκοπό να διαπεράσει όλους τους κόμβους του δέντρου. Επισκέπτεται τους κόμβους πραγματοποιώντας ***αναζήτηση κατά βάθος (Depth First Search, DFS)***, ελέγχει δηλαδή πρώτα τους απογόνους ενός κόμβου και έπειτα τους κόμβους που βρίσκονται στο ίδιο επίπεδο με αυτόν. Η επίσκεψη των κόμβων κατά βάθος δεν γίνεται για συγκεκριμένο λόγο και θα μπορούσε εναλλακτικά να πραγματοποιείται διαφορετικά

όλη η διαδικασία. Η συνάρτηση αυτή παίρνει ως παραμέτρους έναν κόμβο προς ανάλυση, δηλαδή ένα αντικείμενο τύπου *IXMLDOMNodePtr*, με όνομα *xNode* και τον ακέραιο *Level*. Ο ακέραιος αυτός κρατάει πληροφορία σχετικά με το βάθος του δέντρου στο οποίο βρίσκεται κάθε φορά η συνάρτηση. Πρώτο βήμα της συνάρτησης είναι η αύξηση του αριθμού αυτού κατά ένα. Η πρώτη κλήση της συνάρτησης (μέσα από την main) γίνεται με την τιμή 0 στο *Level*. Αφού αλλάξει η τιμή του *Level*, ορίζονται οι μεταβλητές *xNodeType* τύπου *DOMNodeType*, *xNodeTypeString* τύπου *std::string* και *xNodeNameString* τύπου *std::string*. Οι *xNodeType* και *xNodeTypeString* κρατάνε πληροφορία για τον τύπο του κόμβου ενώ η *xNodeTypeString* έχει ως τιμή το όνομα του κόμβου. Και στις τρείς αυτές μεταβλητές εκχωρούνται τιμές κάνοντας χρήση των κατάλληλων μεθόδων του *xNode*. Στη συνέχεια τυπώνονται τα στοιχεία αυτά του κόμβου και αν πρόκειται για Element καλείται η συνάρτηση *printNodeAttributes* ώστε να τυπωθούν πληροφορίες και για τα attributes του. Για την εμφάνιση του τύπου του κόμβου θα μπορούσε να τυπωθεί απλά η μεταβλητή *xNodeTypeString*, η σχετική γραμμή όμως έχει γίνει σχόλιο και γίνεται εναλλακτικά χρήση της συνάρτησης *printNodeType* ώστε να γίνει πιο κατανοητή η χρήση του enum *DOMNodeType* το οποίο χρησιμοποιείται στη συνάρτηση αυτή.

Αφού τυπώθηκαν τα στοιχεία του κόμβου σειρά έχει η αναδρομική κλήση της συνάρτησης για όλους τους αμέσως επόμενους απογόνους του *xNode*. Ανακτούνται οι απόγονοι αυτοί μέσω της μεθόδου *GetChildNodes* και εκχωρούνται στην *IXMLDOMNodeListPtr nodelist*. Με την μέθοδο *GetLength* ανακτάται ο αριθμός των κόμβων που περιέχει η *nodelist* και εκχωρείται στον ακέραιο *numChildNodes*. Στη συνέχεια μέσα σε μία δομή επανάληψης ανακτούνται οι κόμβοι της *nodelist* με τη μέθοδο *Getitem(i)*, για κάθε *i*=0 έως *i*=*numChildNodes*-1 και πραγματοποιείται η αναδρομική κλήση της συνάρτησης για καθέναν από αυτούς.

Η συνάρτηση ***printNodeType*** επιδεικνύει τη χρήση του enum *DOMNodeType*. Συγκεκριμένα παίρνει ένα *DOMNodeType* με όνομα *xNodeType* σαν παράμετρο και ελέγχοντας την τιμή του με μια switch τυπώνεται το κατάλληλο μήνυμα για κάθε περίπτωση.

Τέλος, η συνάρτηση ***printNodeAttributes*** παίρνει έναν κόμβο με όνομα *xNode* ως παράμετρο και τυπώνει τα attributes του. Για να καλεστεί η συγκεκριμένη συνάρτηση έχει πρότινος γίνει έλεγχος για το αν ο κόμβος είναι Element και μόνο σε αυτή την περίπτωση καλείται. Αρχικά ανακτούνται τα attributes του *xNode* μέσω της μεθόδου *Getattributes* και εκχωρούνται στην *IXMLDOMNamedNodeMapPtr xAttributesList*. Με την μέθοδο *GetLength* ανακτάται ο αριθμός των κόμβων που περιέχει η *xAttributesList* και εκχωρείται στον ακέραιο *numAttributes*. Τα attributes ενός Element προσπελαύνονται κι αυτά σαν απλά Nodes, σαν *IXMLDOMNodePtr*. Στη συνέχεια μέσα σε μία δομή επανάληψης ανακτούνται οι κόμβοι της *xAttributesList* με τη μέθοδο *Getitem(i)*, για κάθε *i*=0 έως *i*=*numAttributes*-1 και έπειτα ανακτούνται και τυπώνονται τα στοιχεία για κάθε έναν από τους κόμβους αυτούς με τη μέθοδο που περιγράφηκε και παραπάνω.

```
<?xml version="1.0" ?>

<voters foo="this is the value of foo" bar="this is the value of
    bar">
  <!-- This is a comment -->
  <voter>
    John Doe
  </voter>
  <voter>

  </voter>
  <voter myattr="mia timi" myattr2="mia alli timi">
    <!-- Registered voter information -->
    <name>Chris Lezos</name>
    <address>
      <street1>Sterioti</street1>
      <street2 />
      <city>Sparti</city>
      <state>Lakonia</state>
      <zip>23100</zip>
    </address>
    <phone>6993736656</phone>
  </voter>
  <voter_alternative />
</voters>
```

Listing 2.2: Ένα τυπικό XML αρχείο.

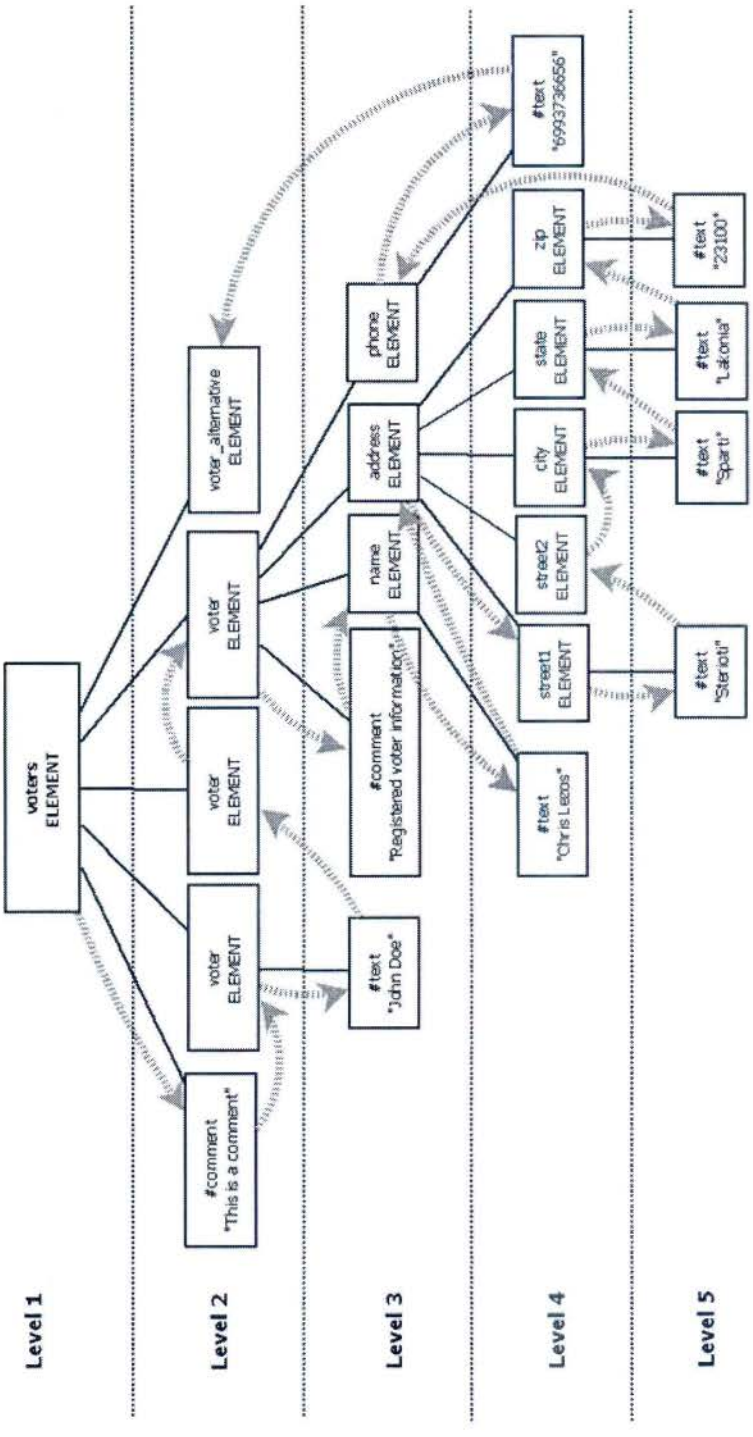
Μετά από εκτέλεση της εφαρμογής έχοντας ως είσοδο το παραπάνω XML αρχείο λαμβάνουμε στην έξοδο τις παρακάτω πληροφορίες:

```
Level: 1
Node Name: voters
Node Type: -Node ELEMENT-
Attributes:
-foo = this is the value of foo
-bar = this is the value of bar

Node has 5 Child nodes.
-----------------------------------
Level: 2
Node Name: #comment
Node Type: -Node COMMENT-
-Content: This is a comment
Node has 0 Child nodes.
-----------------------------------
Level: 2
Node Name: voter
Node Type: -Node ELEMENT-
Node has 1 Child nodes.
-----------------------------------
Level: 3
Node Name: #text
Node Type: -Node TEXT-
-Content:John Doe
Node has 0 Child nodes.
-----------------------------------
Level: 2
Node Name: voter
Node Type: -Node ELEMENT-
Node has 0 Child nodes.
-----------------------------------
Level: 2
Node Name: voter
Node Type: -Node ELEMENT-
Attributes:
-myattr = mia timi
-myattr2 = mia alli timi

Node has 4 Child nodes.
-----------------------------------
Level: 3
Node Name: #comment
Node Type: -Node COMMENT-
```

```
-Content: Registered voter
    information
Node has 0 Child nodes.
-----------------------------------------
Level: 3
Node Name: name
Node Type: -Node ELEMENT-
Node has 1 Child nodes.
-----------------------------------------
Level: 4
Node Name: #text
Node Type: -Node TEXT-
-Content:Chris Lezos
Node has 0 Child nodes.
-----------------------------------------
Level: 3
Node Name: address
Node Type: -Node ELEMENT-
Node has 5 Child nodes.
-----------------------------------------
Level: 4
Node Name: street1
Node Type: -Node ELEMENT-
Node has 1 Child nodes.
-----------------------------------------
Level: 5
Node Name: #text
Node Type: -Node TEXT-
-Content:Sterioti
Node has 0 Child nodes.
-----------------------------------------
Level: 4
Node Name: street2
Node Type: -Node ELEMENT-
Node has 0 Child nodes.
-----------------------------------------
Level: 4
Node Name: city
Node Type: -Node ELEMENT-
Node has 1 Child nodes.
-----------------------------------------
Level: 5
Node Name: #text
Node Type: -Node TEXT-
-Content:Sparti
Node has 0 Child nodes.
-----------------------------------------
Level: 4
Node Name: state
Node Type: -Node ELEMENT-
Node has 1 Child nodes.
-----------------------------------------
Level: 5
Node Name: #text
Node Type: -Node TEXT-
-Content:Lakonia
Node has 0 Child nodes.
-----------------------------------------
Level: 4
Node Name: zip
Node Type: -Node ELEMENT-
Node has 1 Child nodes.
-----------------------------------------
Level: 5
Node Name: #text
Node Type: -Node TEXT-
-Content:23100
Node has 0 Child nodes.
-----------------------------------------
Level: 3
Node Name: phone
Node Type: -Node ELEMENT-
Node has 1 Child nodes.
-----------------------------------------
Level: 4
Node Name: #text
Node Type: -Node TEXT-
-Content:6993736656
Node has 0 Child nodes.
-----------------------------------------
Level: 2
Node Name: voter_alternative
Node Type: -Node ELEMENT-
Node has 0 Child nodes.
-----------------------------------------
```

Listing 2.3: Η έξοδος της εφαρμογής.

Όπως επιβεβαιώνεται και από την έξοδο της εφαρμογής η επίσκεψη στους κόμβους του XML δέντρου γίνεται κατά βάθος. Η σειρά με την οποία επισκέπτεται η εφαρμογή τους κόμβους φαίνεται πιο καθαρά και στο παρακάτω σχήμα:



Σχήμα 2.1: Η σειρά με την οποία η εφαρμογή επισκέπτεται τους κόμβους του XML δέντρου.

Κεφάλαιο 3

HLIR XML Parser

3.1 Εισαγωγή

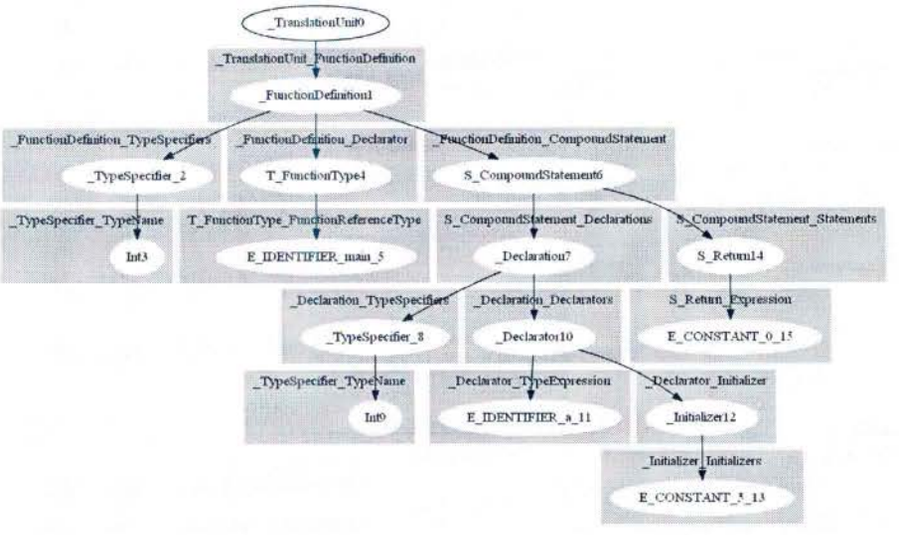
Η HIR της οποίας έχει γίνει χρήση είναι μια δενδροειδούς μορφής αναπαράσταση. Χωρίζει το πρόγραμμα σε ένα σύνολο συντακτικών αντικειμένων και με βάση ένα πίνακα συσχετίσεων τα αντικείμενα αυτά μπορούν να ενωθούν μεταξύ τους. Κάθε συντακτικό αντικείμενο είναι και ένας κόμβος στο δέντρο ενώ κάθε τύπος συντακτικού αντικειμένου μπορεί να συσχετιστεί με συγκεκριμένους μόνο τύπους, να έχει δηλαδή συγκεκριμένους τύπους σαν παιδιά του στο δέντρο. Για παράδειγμα για τον παρακάτω κώδικα σε γλώσσα C η αντίστοιχη αναπαράσταση του στη συγκεκριμένη HIR είναι αυτή του σχήματος 3.1.

```
1 int main()
2 {
3   int a=5;
4   return 0;
5 }
```

Ο HLIR XML Parser είναι το βασικό εργαλείο που αναπτύχτηκε για την παρούσα εργασία. Η εφαρμογή κάνει χρήση του MSXML για την ανάλυση ενός XML εγγράφου με τρόπο ίδιο με αυτόν του παραδείγματος που παρουσιάστηκε στο προηγούμενο κεφάλαιο. Αφού διαβάσει το XML αρχείο επισκέπτεται τους κόμβους του και δημιουργεί τα κατάλληλα συντακτικά αντικείμενα που απαρτίζουν το δέντρο της ενδιάμεσης αναπαράστασης. Έπειτα πραγματοποιείται η οπτικοποίηση του δέντρου μέσω της εφαρμογής Graphviz.

3.2 Λειτουργία της εφαρμογής και Παράμετροι

Ο HLIR XML Parser τρέχει σε *περιβάλλον γραμμής εντολών (Command Line Interface, CLI).* Σαν πρόγραμμα γραμμής εντολών η ρύθμιση της εφαρμογής για πράγματα όπως ο ορισμός του XML αρχείου εισόδου γίνεται μέσω κάποιον πα-



Σχήμα 3.1: Παράδειγμα προγράμματος στην HIR.

ραμέτρων που δέχεται. Συγκεκριμένα είναι δυνατή η εκτέλεση της εφαρμογής με τους παρακάτω τρόπους:

- hlir_xml_parser.exe {FILENAME|-help|-about}
- hlir_xml_parser.exe {FILENAME} [-showlog] [-nograph]

Όπου:

- Ότι περικλείεται σε braces {} είναι υποχρεωτικό να συμπληρωθεί,
- Ότι περικλείεται σε brackets [] είναι προαιρετικό,
- Η κάθετη γραμμή | συμβολίζει τον λογικό τελεστή OR,
- *FILENAME* είναι το path του xml αρχείου εισόδου,
- Η παράμετρος *-help* εμφανίζει βοήθεια σχετικά με τη λειτουργία της εφαρμογής,
- Η παράμετρος *-about* εμφανίζει πληροφορίες σχετικά με την εφαρμογή,
- Με την παράμετρο *-showlog* ενεργοποιημένη ανοίγει το αρχείο καταγραφής συμβάντων (log) της εφαρμογής με τη λήξη της εκτέλεσης της,
- Με την παράμετρο *-nograph* ενεργοποιημένη δεν δημιουργείται αρχείο GIF αλλά μόνο log.

3.3 Οπτικοποίηση γράφων με το Graphviz

Το Graphviz[1] είναι μια ανοικτού κώδικα εφαρμογή οπτικοποίησης γράφων. Παίρνει σαν είσοδο ένα κείμενο με την περιγραφή ενός γράφου σε γλώσσα DOT[2] και δημιουργεί οπτικοποιημένα διαγράμματα σε διάφορα format όπως GIF, PNG, PDF και άλλα. Στον ιστότοπο της εφαρμογής υπάρχει τεκμηρίωση που περιγράφει την εγκατάσταση και χρήση του εργαλείου αλλά και της γλώσσας DOT.

Ένα μικρό παράδειγμα των δυνατοτήτων του Graphviz φαίνεται στη συνέχεια.

```
digraph People {
Giannis -> Dimitra ;
Spiros -> Christos ;
Spiros -> Maria ;
Anna -> Spiros ;
Anna -> Giannis ;
Anna -> Dimitra ;
}
```

Στο παραπάνω κομμάτι κώδικα φαίνεται η περιγραφή ενός γράφου σε γλώσσα DOT. Θεωρούμε ότι ο κώδικας αυτός βρίσκεται στο αρχείο κειμένου *dot_test.dot*. Τρέχοντας το Graphviz από την γραμμή εντολών με τον ακόλουθο τρόπο παράγεται το αρχείο GIF του σχήματος 3.2.

```
$ dot -Tgif dot_test.dot -o dot_test.gif
```

Για την οπτικοποίηση της HIR ο HLIR XML Parser δημιουργεί το κατάλληλο αρχείο κειμένου σε γλώσσα DOT και στη συνέχεια μέσω της συνάρτησης *system()* καλεί το Graphviz για την μετατροπή σε αρχείο GIF όπως φαίνεται παρακάτω:

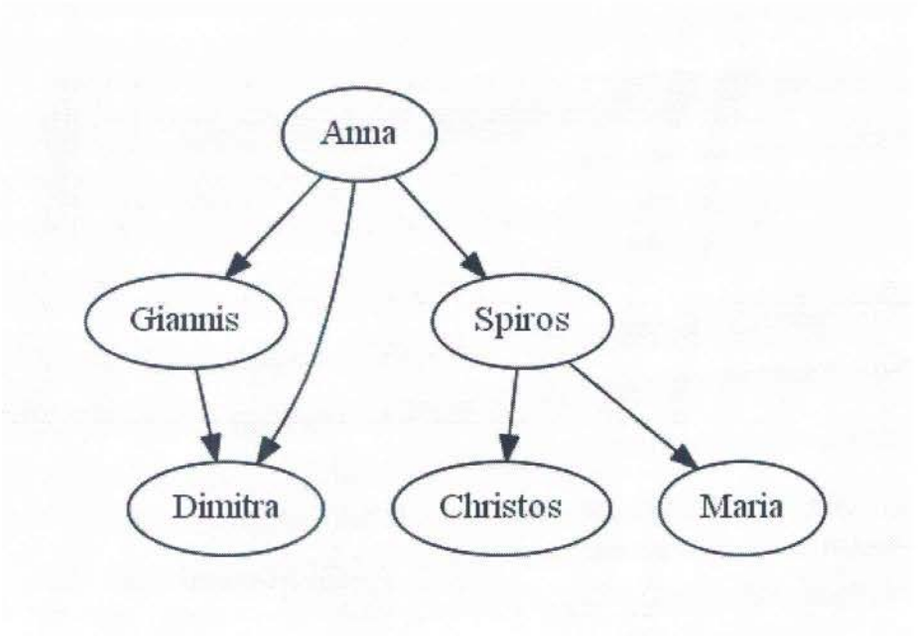
```
system("dot -Tgif HLIR_graph.dot -o HLIR.gif");
```

3.4 HLIR Library

Η HLIR Library διατίθεται σε μορφή C++ Static Library από το σχετικό ιστότοπο της HIR ο οποίος αναφέρεται στην εισαγωγή. Παρέχεται σε συμπιεσμένη μορφή .zip και πρέπει να αποσυμπιεστεί προτού χρησιμοποιηθεί.

Ακολουθεί μια λίστα με τις κυριότερες διεπαφές που προσφέρει η βιβλιοθήκη. Οι διεπαφές αυτές χρησιμοποιήθηκαν και στον HLIR XML Parser.

[1] http://www.graphviz.org/
[2] http://www.graphviz.org/doc/info/lang.html



Σχήμα 3.2: Το αρχείο dot_test.gif.

- **HLIR_ELEMENT_TYPE** - Το enum *HLIR_ELEMENT_TYPE* παρέχει ένα σύνολο κωδικών, έναν για κάθε τύπο συντακτικού αντικείμενου που υποστηρίζει η HIR.

- **CHLIR_SyntaxElement** - Η κλάση *CHLIR_SyntaxElement* αναπαριστά ένα συντακτικό αντικείμενο στην HLIR Library. Κάθε τύπος συντακτικού αντικειμένου της HIR αντιπροσωπεύεται από μια ξεχωριστή κλάση και όλες αυτές οι κλάσεις κληρονομούν την *CHLIR_SyntaxElement*. Όλες οι κλάσεις-παιδιά της *CHLIR_SyntaxElement* περιλαμβάνουν μεταξύ άλλων και τις παρακάτω virtual συναρτήσεις:

  - **virtual void HLIRGraphEmmiter(CHLIR_SyntaxElement *parent)** - Συνάρτηση η οποία διατρέχει το δέντρο και δημιουργεί το DOT αρχείο.
  - **virtual int InstallContextElement(CHLIR_SyntaxElement *el)** - Η συνάρτηση αυτή καλείται για ένα συντακτικό αντικείμενο ώστε να συνδεθεί με κάποιο άλλο αντικείμενο ως πατέρας του. Παίρνει ένα δείκτη προς το υποψήφιο παιδί με όνομα *el* σαν παράμετρο, ελέγχει αν επιτρέπεται η σύνδεση μεταξύ τους και τοποθετεί το *el* στη λίστα με τα παιδιά του συντακτικού αντικειμένου για το οποίο καλείται. Αν η συσχέτιση δεν επιτρέπεται εμφανίζεται σχετικό μήνυμα λάθους και σταματάει η εκτέλεση της εφαρμογής.

Ακολουθώντας τα παρακάτω βήματα μπορεί να γίνει χρήση της βιβλιοθήκης από το τρέχων project στο Visual Studio:

1. Εντοπισμός του φακέλου στον οποίο βρίσκονται τα αρχεία της HLIR Library. Ο φάκελος αυτός περιέχει δύο υποφακέλους με τα ονόματα **include** και **lib**.

2. Ρύθμιση του include directory στο Visual Studio: Από το μενού **Tools** επιλογή του **Options**. Στο παράθυρο διαλόγου **Options** επέκταση του **Projects and Solutions**. Επιλογή του **VC++ Directories**. Επιλογή του **Include files** από το drop down **Show directories for**. Κλικ στο κουμπί **New Line** για προσθήκη νέου φακέλου στη λίστα. Αναζήτηση και επιλογή του φακέλου που περιέχει τα header αρχεία της HLIR Library.

3. Ρύθμιση του lib directory στο Visual Studio: Από το μενού **Tools** επιλογή του **Options**. Στο παράθυρο διαλόγου **Options** επέκταση του **Projects and Solutions**. Επιλογή του **VC++ Directories**. Επιλογή του **Library files** από το drop down **Show directories for**. Κλικ στο κουμπί **New Line** για προσθήκη νέου φακέλου στη λίστα. Αναζήτηση και επιλογή του φακέλου που περιέχει το αρχείο .lib της HLIR Library.

4. Μετατροπή του project ώστε να κάνει link με την HLIR Library: Από το **Solutions Explorer**, δεξί κλικ στο project και επιλογή του **Properties**. Επέκταση του **Configuration Properties** και του **Linker** και επιλογή του **Command Line**. Εισαγωγή του **hlir_library.lib** στο πλαίσιο κειμένου **Additional options**.

5. Προσθήκη των ακόλουθων γραμμών στον κώδικα για να συμπεριληφθούν τα κατάλληλα header αρχεία της HLIR Library.

```
#include <HLIRDefines.h>
#include <HLIRSyntaxElements.h>
#include <Types.h>
#include <TypeDefines.h>
```

3.5 Οι συναρτήσεις Decode

Η αναπαράσταση των συντακτικών αντικειμένων της HIR και των ιδιοτήτων τους γίνεται διαφορετικά στο XML από ότι στις κλάσεις της HLIR Library. Για παράδειγμα η τιμή της ιδιότητας signness ενός συντακτικού αντικειμένου που αντιπροσωπεύει ένα integer στο XML μπορεί να έχει τις αλφαριθμητικές τιμές "signed" και "unsigned" ενώ στην HLIR Library οι συγκεκριμένες τιμές αναπαρίστανται με το enum *INTEGER_SIGNNESS* και μπορεί να είναι *SGN_SIGNED* και *SGN_UNSIGNED*.

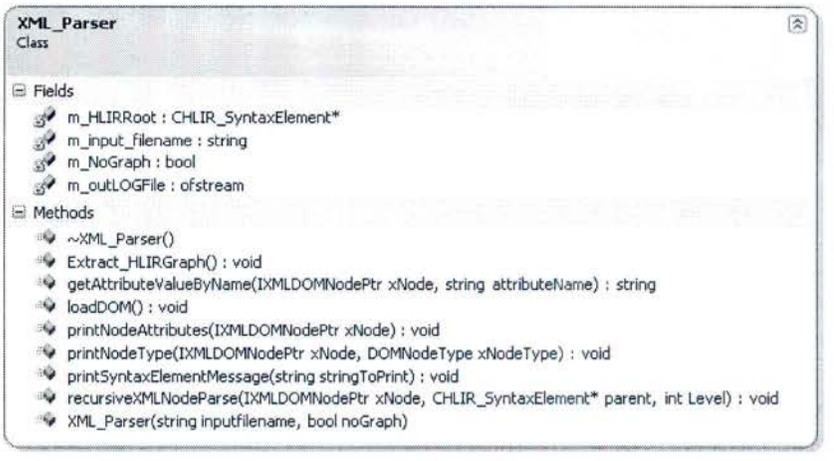
Οι συναρτήσεις decode είναι ένα σύνολο βοηθητικών συναρτήσεων όπου παίρνουν ένα string ως παράμετρο και επιστρέφουν μια τιμή σε μορφή κατάλληλη για να αναγνωριστεί από τις διεπαφές της HLIR Library. Η σημαντικότερη από αυτές τις συναρτήσεις είναι η *DecodeXmlElementString*. Η συνάρτηση αυτή παίρνει το όνομα ενός XML κόμβου σε string και επιστρέφει τον αντίστοιχο *HLIR_ELEMENT_TYPE* κωδικό του συγκεκριμένου τύπου συντακτικού αντικειμένου της HIR.

Όλες οι συναρτήσεις decode φαίνονται παρακάτω:

- **HLIR_ELEMENT_TYPE DecodeXmlElementString(string xNodeName)**
- **INTEGER_SIGNNESS DecodeXmlIntegerTypeSignness(string IntegerTypeSignnessString)**
- **INTEGER_PRECISION DecodeXmlIntegerTypePrecision(string IntegerTypePrecisionString)**
- **FLOATING_PRECISION DecodeXmlFloatingTypePrecision(string FloatingTypePrecisionString)**
- **CONSTANT_CATEGORY DecodeXmlConstantCategory(string ConstantCategoryString)**
- **unsigned char DecodeXmlConstantSignness(string ConstantSignnessString)**
- **unsigned char DecodeXmlConstantLongness(string ConstantLongnessString)**
- **unsigned char DecodeXmlConstantFloatness(string ConstantFloatnessString)**
- **unsigned char DecodeXmlConstantWideness(string ConstantWidenessString)**
- **unsigned char DecodeXmlConstantEschexseq(string ConstantEschexseqString)**
- **unsigned char DecodeXmlConstantEscoctseq(string ConstantEscoctseqString)**
- **unsigned char DecodeXmlConstantEscdecseq(string ConstantEscdecseqString)**
- **unsigned char DecodeXmlConstantEscuniseq(string ConstantEscuniseqString)**
- **unsigned char DecodeXmlStringWideness(string StringWidenessString)**
- **TYPE_QUALIFIER DecodeXmlTypeQualifier(string TypeQualifierSemanticsString)**

3.6 Η κλάση XML_Parser

Η κλάση *XML_Parser* είναι η βασική δομή της εφαρμογής. Μέσω των μεθόδων της κλάσης αυτής είναι δυνατή η ανάγνωση του XML εγγράφου μέσω του MSXML, η δημιουργία των αντικειμένων της HLIR Library ανάλογα με την XML είσοδο και η σύνδεση μεταξύ τους, η δημιουργία αρχείου καταγραφής δραστηριότητας της εφαρμογής (log) καθώς και η κατασκευή του κατάλληλου αρχείου dot και η κλήση του Graphviz για την οπτικοποίηση του αρχείου αυτού σε μορφή GIF.



Σχήμα 3.3: Τα πεδία και οι μέθοδοι της κλάσης XML_Parser.

Μόλις εκτελείται η εφαρμογή δημιουργείται ένα αντικείμενο XML_Parser με όνομα *parser* και σε αυτό εκτελούνται όλες οι λειτουργίες του προγράμματος.

Τα πεδία και οι μέθοδοι της κλάσης XML_Parser φαίνονται στο σχήμα 3.3 ενώ οι λειτουργίες τους περιγράφονται αναλυτικά παρακάτω:

- **ofstream m_outLOGFile**; - Πεδίο το οποίο αναπαριστά ένα αρχείο στο οποίο θα καταγράφεται η δραστηριότητα της εφαρμογής (log).
- **string m_input_filename**; - Το πεδίο αυτό παίρνει ως τιμή το path του XML αρχείου εισόδου.
- **bool m_NoGraph**; - Bool πεδίο το οποίο λειτουργεί σαν διακόπτης για το αν θα παράγει αρχεία DOT και GIF η κλάση ή όχι.
- **CHLIR_SyntaxElement *m_HLIRRoot**; - Δείκτης σε ένα αντικείμενο τύπου *CHLIR_SyntaxElement* ο οποίος αναπαριστά τη ρίζα του HIR δέντρου.

- **XML_Parser(string inputfilename, bool noGraph)** - Ο constructor της κλάσης. Παίρνει ως παραμέτρους ένα string με όνομα *inputfilename* το οποίο έχει ως τιμή το path του XML αρχείου εισόδου και ένα Bool με όνομα *noGraph*. Οι τιμές των παραμέτρων αυτών εκχωρούνται στα πεδία *m_input_filename* και *m_NoGraph* αντίστοιχα. Επιπλέον ανοίγει το αρχείο *m_outLOGFile* περνώντας του ως παράμετρο το "hlir_xml_parser_log.txt" και καλείται η μέθοδος *loadDOM()*.

- **XML_Parser()** - Ο destructor της κλάσης. Κλείνει το αρχείο *m_outLOGFile*.

- **void loadDOM()** - Στη μέθοδο αυτή δημιουργείται ένα αντικείμενο *pXMLDom* τύπου *IXMLDOMDocumentPtr* και μέσω της μεθόδου *load(xmlSource)* του αντικειμένου αυτού φορτώνεται το *m_input_filename*. Δημιουργείται επίσης και το αντικείμενο *pRoot* τύπου *IXMLDOMElementPtr* το οποίο αντιπροσωπεύει τον κεντρικό κόμβο του XML δέντρου και του εκχωρείται η τιμή του πεδίου *documentElement* του *pXMLDom*. Στη συνέχεια καλείται η μέθοδος *recursiveXMLNodeParse* δίνοντας της ως παράμετρο το *pRoot* ώστε να εκκινήσει η αναδρομική σάρωση όλων των κόμβων του XML δέντρου. Μόλις ολοκληρωθεί αυτή η διαδικασία ελέγχεται η τιμή του πεδίου *m_NoGraph* και εάν είναι ίση με *true* καλείται η μέθοδος *Extract_HLIRGraph* για τη δημιουργία του αρχείου DOT και έπειτα καλείται το Graphviz μέσω της συνάρτησης *system()* για την μετατροπή του DOT αρχείου σε GIF.

- **void recursiveXMLNodeParse(MSXML2::IXMLDOMNodePtr xNode, CHLIR_SyntaxEleme *parent, int Level)** - Η μέθοδος αυτή εκτελείται αναδρομικά με σκοπό να διαπεράσει όλους τους κόμβους του XML δέντρου. Λειτουργεί ακριβώς με τον ίδιο τρόπο όπως η συνάρτηση *recursiveNodeParse* του παραδείγματος που παρατέθηκε στο προηγούμενο κεφάλαιο. Πριν κληθεί αναδρομικά για τα παιδιά του προς εξέταση κόμβου η μέθοδος εκτελεί και κάποιες επιπλέον λειτουργίες σε σχέση με τη συνάρτηση *recursiveNodeParse*. Αρχικά παίρνει μια παράμετρο παραπάνω από τη συνάρτηση *recursiveNodeParse*, ένα δείκτη προς ένα συντακτικό αντικείμενο *CHLIR_SyntaxElement* με όνομα *parent*. Ο δείκτης αυτός αντιπροσωπεύει τον πατέρα του προς εξέταση κόμβου στην HIR. Αφού ανακτήσει τα στοιχεία του κόμβου και τυπώσει τα κατάλληλα μηνύματα στο αρχείο log η μέθοδος καλεί τη συνάρτηση *DecodeXmlElementString* με το όνομα του κόμβου ως παράμετρο. Η *DecodeXmlElementString* επιστρέφει τον αντίστοιχο *HLIR_ELEMENT_TYPE* κωδικό. Μέσω μιας δομής επιλογής ελέγχεται ο κωδικός αυτός ως προς όλους τους τύπους συντακτικών αντικειμένων και ανάλογα την περίπτωση δημιουργείται ένα κατάλληλου τύπου αντικείμενο με όνομα *newobject*. Έπειτα καλείται η μέθοδος *InstallContextElement* για το *CHLIR_SyntaxElement* που δείχνει ο *parent* περνώντας το *newobject* σαν παράμετρο. Με την ενέργεια αυτή ορίζεται ουσιαστικά το *newobject* σαν παιδί του *parent*. Στη συνέχεια ακολουθεί η αναδρομική κλήση της μεθόδου για όλα τα παιδιά του προς εξέταση κόμβου δίνοντας το *newobject* ως παράμετρο *father*. Για να

γίνει πιο κατανοητή η λειτουργία της συγκεκριμένης μεθόδου παρατίθεται ο παρακάτω αλγόριθμος.

```
void recursiveXMLNodeParse(MSXML2::IXMLDOMNodePtr xNode,
                           CHLIR_SyntaxElement *parent){

  CHLIR_SyntaxElement *newobject;
  string xNodeName = To onoma tou xNode;

  switch( DecodeXmlElementString(xNodeName) ){
    case S_ForLoop:
      newobject = CS_HLIRForLoop();
      parent->InstallContextElement(newobject);
    case S_If:
      newobject = CS_HLIRIf();
      parent->InstallContextElement(newobject);
      .
      .
      .
  }

  // Anadromiki klhsh ths synarthshs gia ola ta paidia tou komvou
  For loop {
    recursiveXMLNodeParse(child, newobject);
  }

}
```

- **void printNodeType(MSXML2::IXMLDOMNodePtr xNode, MSXML2::DOMNodeType xNodeType)** - Παίρνει έναν XML κόμβο IXMLDOMNodePtr με όνομα *xNode* ως παράμετρο και τον τύπο του σε μορφή *DOMNodeType* και τυπώνει τον τύπο αυτό στο log αρχείο *m_outLOGFile*.

- **void printNodeAttributes(MSXML2::IXMLDOMNodePtr xNode)** - Παίρνει έναν XML κόμβο IXMLDOMNodePtr με όνομα *xNode* ως παράμετρο και τυπώνει στο log αρχείο *m_outLOGFile* τα attributes του κόμβου αυτού και τις τιμές τους.

- **void printSyntaxElementMessage(string stringToPrint)** - Τυπώνει ένα μήνυμα στο log αρχείο *m_outLOGFile*. Το μήνυμα αυτό το παίρνει ως παράμετρο σε μορφή string.

- **string getAttributeValueByName(MSXML2::IXMLDOMNodePtr xNode, string attributeName)** - Παίρνει έναν XML κόμβο IXMLDOMNodePtr με όνομα *xNode* και το όνομα ενός attribute του κόμβου αυτού σε μορφή string ως παραμέτρους και επιστρέφει την τιμή του attribute σε μορφή string.

- **void Extract_HLIRGraph()** - Η συνάρτηση αυτή καλεί τη μέθοδο *HLIR-GraphEmmiter* για το αντικείμενο στο οποίο δείχνει ο δείκτης *m_HLIRRoot* ώστε να διατρεχθεί το HIR δέντρο και να δημιουργηθεί το αρχείο *HLIR_graph.dot*.

3.7 Παραδείγματα χρήσης

Ακολουθούν μερικά χαρακτηριστικά παραδείγματα χρήσης του εργαλείου. Παρατίθενται για κάθε παράδειγμα ο αρχικός C κώδικας, η HIR σε μορφή XML και σε οπτικοποιημένη μορφή όπως παράγεται από τον HLIR XML Parser.

Στο πρώτο παράδειγμα (listing 3.1, listing 3.4, σχήμα 3.4) φαίνεται η δήλωση και η αρχικοποίηση μιας μεταβλητής και ενός πίνακα, στο δεύτερο (listing 3.2, listing 3.5, σχήμα 3.5) φαίνεται μια δομή επιλογής if και στο τρίτο (listing 3.3, listing 3.6, σχήμα 3.6) φαίνεται μια συνάρτηση με δύο παραμέτρους που έχει στο σώμα της μια δήλωση μεταβλητής και μια εκχώρηση τιμής.

```
int a=5;
int b[3]={1,2,3};
```

Listing 3.1: Το πρώτο παράδειγμα σε C κώδικα.

```
int main()
{
  int a=5;
  if( a==5 )
  {
    a=6;
  }
  return 0;
}
```

Listing 3.2: Το δεύτερο παράδειγμα σε C κώδικα.

```
int average(int a, int b)
{
  int c;
  c = (a+b)/2;
  return c;
}
```

Listing 3.3: Το τρίτο παράδειγμα σε C κώδικα.

```
<?xml version="1.0"?>
<translation_unit>
  <declaration>
    <type_specifier>
      <integer_type signness="signed" precision="int">int</
          integer_type>
    </type_specifier>

    <declarators semantics="pure_declarator">
```

```
      <declarator>
        <type_expression>
          <identifier name="a"/>
        </type_expression>
        <initialization>
          <initializer>
            <constant type="decimal" signness="signed" longness="
                regular" floatness="na" wideness="na" eschexseq="
                na" escoctseq="na" escdecseq="na" escuniseq="na"
                value="5"/>
          </initializer>
        </initialization>
      </declarator>
    </declarators>
  </declaration>
  <declaration>
    <type_specifier>
      <integer_type signness="signed" precision="int">int</
          integer_type>
    </type_specifier>

    <declarators semantics="pure_declarator">
      <declarator>
        <type_expression>
          <array_type>
            <array_size>
              <constant type="decimal" signness="signed" longness=
                  "regular" floatness="na" wideness="na" eschexseq
                  ="na" escoctseq="na" escdecseq="na" escuniseq="
                  na" value="3"/>
            </array_size>
            <array_type_expression>
              <identifier name="b"/>
            </array_type_expression>
          </array_type>
        </type_expression>
        <initialization>
          <initializer>
            <initializer>
              <constant type="decimal" signness="signed" longness=
                  "regular" floatness="na" wideness="na" eschexseq
                  ="na" escoctseq="na" escdecseq="na" escuniseq="
                  na" value="1"/>
            </initializer>
            <initializer>
              <constant type="decimal" signness="signed" longness=
                  "regular" floatness="na" wideness="na" eschexseq
                  ="na" escoctseq="na" escdecseq="na" escuniseq="
                  na" value="2"/>
            </initializer>
            <initializer>
              <constant type="decimal" signness="signed" longness=
                  "regular" floatness="na" wideness="na" eschexseq
                  ="na" escoctseq="na" escdecseq="na" escuniseq="
```

```
                    na" value="3"/>
48              </initializer>
              </initializer>
50          </initialization>
        </declarator>
52      </declarators>
    </declaration>
54
</translation_unit>
```

Listing 3.4: Η HIR του πρώτου παραδείγματος σε μορφή XML.

```
1  <?xml version="1.0"?>
   <translation_unit>
3
     <function_definition>
5      <return_type>
         <type_specifier>
7          <integer_type signness="signed" precision="int">int</
               integer_type>
         </type_specifier>
9      </return_type>

11     <function_declarator semantics="type_expression">
         <function_type>
13         <function_reference_type>
             <identifier name="main"/>
15         </function_reference_type>

17         <function_parameter_declarations/>
         </function_type>
19     </function_declarator>

21     <compound_statement>
         <declaration>
23         <type_specifier>
             <integer_type signness="signed" precision="int">int</
                 integer_type>
25         </type_specifier>

27         <declarators semantics="pure_declarator">
             <declarator>
29             <type_expression>
                 <identifier name="a"/>
31             </type_expression>
               <initialization>
33               <initializer>
                   <constant type="decimal" signness="signed"
                       longness="regular" floatness="na" wideness="na
                       " eschexseq="na" escoctseq="na" escdecseq="na"
                        escuniseq="na" value="5"/>
35               </initializer>
               </initialization>
37           </declarator>
```

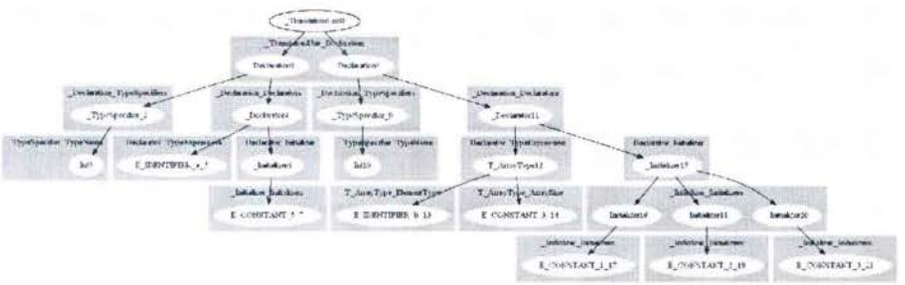
```
        </declarators>
      </declaration>
      <if_statement>
        <expression>
          <EqualTo>
            <argument semantics="left">
              <identifier name="a"/>
            </argument>

            <argument semantics="right">
              <constant type="decimal" signness="signed" longness=
                  "regular" floatness="na" wideness="na" eschexseq
                  ="na" escoctseq="na" escdecseq="na" escuniseq="
                  na" value="5"/>
            </argument>
          </EqualTo>
        </expression>

        <if_body>
          <compound_statement>
            <expression_statement>
              <Assignment>
                <argument semantics="lefthandside">
                  <identifier name="a"/>
                </argument>

                <argument semantics="righthandside">
                  <constant type="decimal" signness="signed"
                      longness="regular" floatness="na" wideness="
                      na" eschexseq="na" escoctseq="na" escdecseq=
                      "na" escuniseq="na" value="6"/>
                </argument>
              </Assignment>
            </expression_statement>
          </compound_statement>
        </if_body>


      </if_statement>

      <return_statement>
        <expression>
          <constant type="decimal" signness="signed" longness="
              regular" floatness="na" wideness="na" eschexseq="na"
               escoctseq="na" escdecseq="na" escuniseq="na" value=
              "0"/>
        </expression>
      </return_statement>
    </compound_statement>
  </function_definition>
</translation_unit>
```

Listing 3.5: Η HIR του δεύτερου παραδείγματος σε μορφή XML.

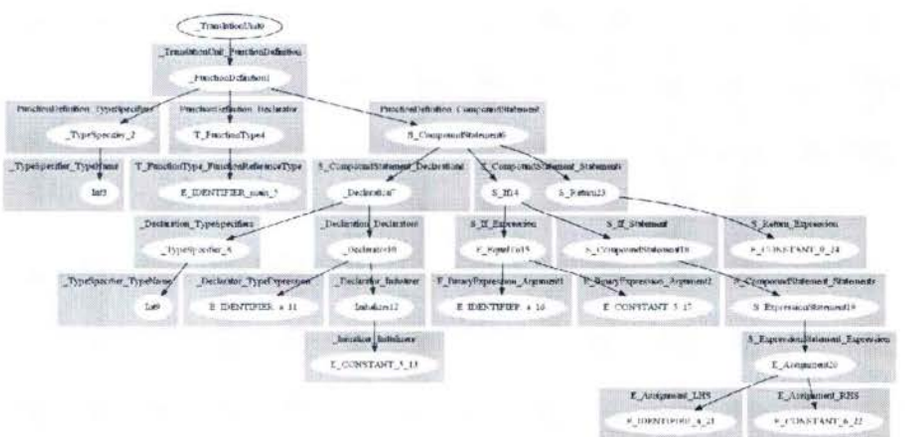
```
<?xml version="1.0"?>
<translation_unit>

  <function_definition>
    <return_type>
      <type_specifier>
        <integer_type signness="signed" precision="int">int</
            integer_type>
      </type_specifier>
    </return_type>

    <function_declarator semantics="type_expression">
      <function_type>
        <function_reference_type>
          <identifier name="average"/>
        </function_reference_type>

        <function_parameter_declarations>
          <declaration>
            <type_specifier>
              <integer_type signness="signed" precision="int">int<
                  /integer_type>
            </type_specifier>

            <declarators semantics="specialized_declarator">
              <identifier name="a"/>
            </declarators>
          </declaration>
          <declaration>
            <type_specifier>
              <integer_type signness="signed" precision="int">int<
                  /integer_type>
            </type_specifier>

            <declarators semantics="specialized_declarator">
              <identifier name="b"/>
            </declarators>
          </declaration>
        </function_parameter_declarations>
      </function_type>
    </function_declarator>

    <compound_statement>
      <declaration>
        <type_specifier>
          <integer_type signness="signed" precision="int">int</
              integer_type>
        </type_specifier>

        <declarators semantics="pure_declarator">
          <declarator>
            <type_expression>
              <identifier name="c"/>
```

```
            </type_expression>

          </declarator>
        </declarators>
      </declaration>
      <expression_statement>
        <Assignment>
          <argument semantics="lefthandside">
            <identifier name="c"/>
          </argument>

          <argument semantics="righthandside">
            <Division>
              <argument semantics="left">
                <grouping_expression>
                  <Addition>
                    <argument semantics="left">
                      <identifier name="a"/>
                    </argument>

                    <argument semantics="right">
                      <identifier name="b"/>
                    </argument>
                  </Addition>
                </grouping_expression>
              </argument>

              <argument semantics="right">
                <constant type="decimal" signness="signed"
                    longness="regular" floatness="na" wideness="na
                    " eschexseq="na" escoctseq="na" escdecseq="na"
                     escuniseq="na" value="2"/>
              </argument>
            </Division>
          </argument>
        </Assignment>
      </expression_statement>

      <return_statement>
        <expression>
          <identifier name="c"/>
        </expression>
      </return_statement>
    </compound_statement>
  </function_definition>
</translation_unit>
```

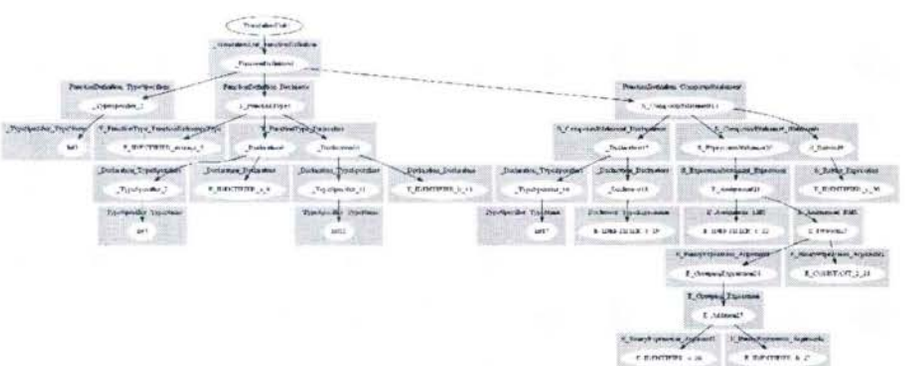
Listing 3.6: Η HIR του τρίτου παραδείγματος σε μορφή XML.



Σχήμα 3.4: Η οπτικοποιημένη HIR του πρώτου παραδείγματος.



Σχήμα 3.5: Η οπτικοποιημένη HIR του δεύτερου παραδείγματος.



Σχήμα 3.6: Η οπτικοποιημένη HIR του τρίτου παραδείγματος.

Κεφάλαιο 4

HLIR XML Parser GUI

4.1 Εισαγωγή

Ο HLIR XML Parser είναι μια εφαρμογή που τρέχει σε γραμμή εντολών. Λόγο της δυσχρηστίας της γραμμής εντολών δημιουργήθηκε και μια συμπληρωματική εφαρμογή που προσθέτει γραφικό περιβάλλον στον HLIR XML Parser. Για να υλοποιηθεί αυτό το *γραφικό περιβάλλον διεπαφής (Graphic User Interface, GUI)* έγινε χρήση της γλώσσας C#. Η επιλογή αυτή έγινε κυρίως λόγο των ευκολιών που παρέχει η συγκεκριμένη τεχνολογία στην ανάπτυξη παραθυρικών εφαρμογών σε συνδυασμό με το .NET Framework και το Visual Studio.

Το GUI που αναπτύχθηκε αποτελεί ξεχωριστή εφαρμογή από τον HLIR XML Parser. Περιλαμβάνει κάποιες φόρμες με διάφορα στοιχεία ελέγχου και ελάχιστες μόνο λειτουργίες σχετικές με την HIR και την ανάλυση του XML εγγράφου. Με βάση τις επιλογές του χρήστη μέσω των στοιχείων ελέγχου των φορμών του GUI η εφαρμογή τρέχει τον HLIR XML Parser δίνοντας του κατάλληλες παραμέτρους. Για παράδειγμα στην κεντρική φόρμα του GUI υπάρχει ένα TextBox το οποίο παίρνει ως τιμή ένα path. Όταν εκτελείται ο HLIR XML Parser περνιέται το path αυτό ως παράμετρος για τη θέση του XML αρχείου εισόδου. Παίρνει δηλαδή τις κατάλληλες παραμέτρους μέσω των στοιχείων ελέγχου που παρέχει, τρέχει τον HLIR XML Parser στο παρασκήνιο και μόλις ολοκληρωθεί η εκτέλεση λαμβάνει τα αποτελέσματα και τα παρουσιάζει στο χρήστη.

Μια επιπλέον λειτουργία που παρέχει το GUI και δεν είναι ενσωματωμένη στον HLIR XML Parser είναι η δυνατότητα ελέγχου εγκυρότητας του XML εγγράφου κάνοντας χρήση ενός αρχείου DTD. Για να δουλέψει η συγκεκριμένη λειτουργία θα πρέπει φυσικά να αναπτυχθεί και το αντίστοιχο DTD.

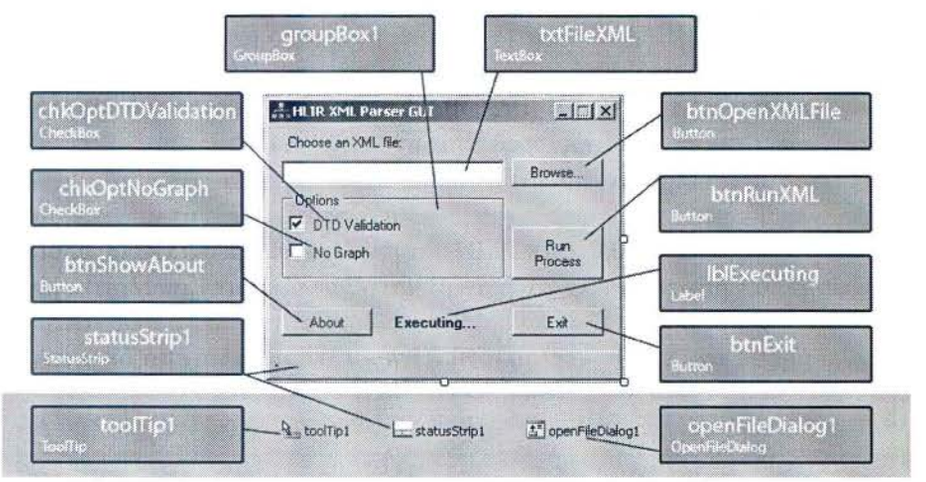
4.2 Λειτουργία της εφαρμογής

Το HLIR XML Parser GUI αποτελείται από τις παρακάτω φόρμες:

- **frmMain.cs**: Το κεντρικό παράθυρο της εφαρμογής.

- **frmRunResult.cs**: Παράθυρο που παρουσιάζει στο χρήστη την έξοδο από την εκτέλεση του HLIR XML Parser.
- **frmAbout.cs**: Πληροφορίες για την εφαρμογή.

Κατά την εκτέλεση της εφαρμογής εμφανίζεται αρχικά το κεντρικό παράθυρο, η φόρμα frmMain (σχήμα 4.1). Ο χρήστης προτρέπεται να πληκτρολογήσει το path ενός XML αρχείου στο σχετικό textbox ή κάνοντας κλικ στο button "Browse" να επιλέξει ένα XML αρχείο μέσω ενός παραθύρου διαλόγου. Στα αριστερά του παραθύρου υπάρχουν δύο checkboxes με τις ενδείξεις "DTD Validation" και "No Graph" ενώ στα δεξιά υπάρχει το button "Run Process". Αφού ο χρήστης επιλέξει ένα αρχείο και κάνει κλικ στο "Run Process" εκτελείται ο HLIR XML Parser με το συγκεκριμένο αρχείο ως είσοδο.



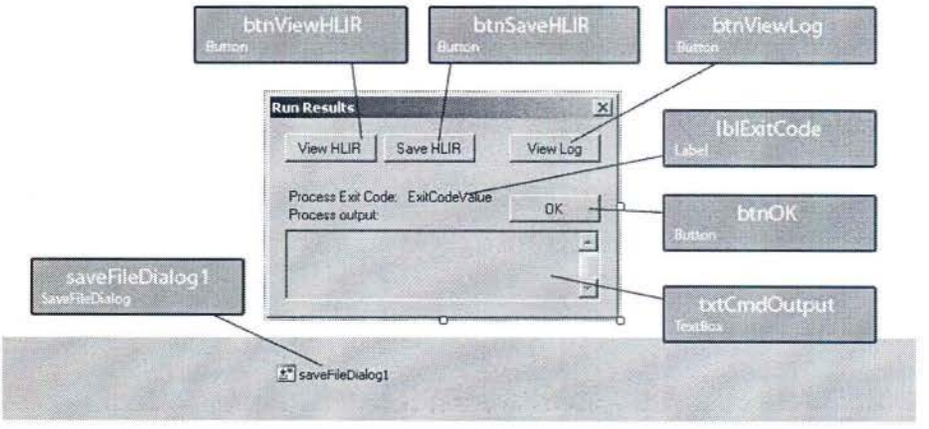
Σχήμα 4.1: Η φόρμα frmMain και τα στοιχεία ελέγχου της.

Αν η επιλογή "DTD Validation" είναι ενεργοποιημένη θα πραγματοποιηθεί πριν την εκτέλεση έλεγχος εγκυρότητας του XML με βάση το αρχείο "hlir.dtd". Το συγκεκριμένο DTD αρχείο δεν ανεπτύχθη στα πλαίσια αυτής της εργασίας αλλά ο μηχανισμός που πραγματοποιεί τον έλεγχο έχει υλοποιηθεί. Στην περίπτωση που είναι τσεκαρισμένη η επιλογή "No Graph" ο HLIR XML Parser θα εκτελεστεί με την παράμετρο *-nograph* ώστε να μην παραχθεί αρχείο GIF και dot.

Για όσο διάστημα εκτελείται ο HLIR XML Parser τα στοιχεία ελέγχου της φόρμας αποκρύπτονται και στη θέση τους εμφανίζεται το μήνυμα "Executing...".

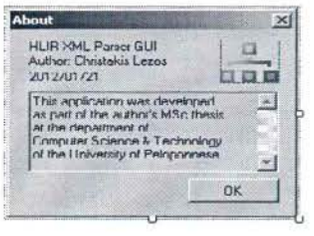
Κάνοντας κλικ στο button "About" εμφανίζεται η φόρμα frmAbout ενώ το button "Exit" κλείνει την εφαρμογή.

Μόλις ολοκληρωθεί η εκτέλεση του HLIR XML Parser όλα τα στοιχεία ελέγχου της frmMain επιστρέφουν στην αρχική τους κατάσταση και εμφανίζεται στο προσκήνιο η φόρμα frmRunResult (σχήμα 4.2). Η φόρμα αυτή επιτρέπει στο χρήστη μέσα από τα buttons "View HLIR" και "Save HLIR" να προβάλει και να



Σχήμα 4.2: Η φόρμα frmRunResult και τα στοιχεία ελέγχου της.

αποθηκεύσει αντίστοιχα το GIF αρχείο που παρήγαγε ο HLIR XML Parser ενώ κάνοντας κλικ στο button "View Log" μπορεί να προβάλει το log αρχείο που παρήχθει. Στο κάτω μέρος του παραθύρου εμφανίζεται η command line έξοδος του HLIR XML Parser ώστε σε περίπτωση σφάλματος να παρέχονται στο χρήστη οι σχετικές πληροφορίες. Κάνοντας κλικ στο button "OK" κλείνει η frmRunResults και ο χρήστης επιστρέφει στην κεντρική φόρμα.



Σχήμα 4.3: Η φόρμα frmAbout.

4.3 Κλάσεις & μέθοδοι

Στο σχήμα 4.4 φαίνονται οι κλάσεις της εφαρμογής, μια για κάθε φόρμα, μαζί με τα πεδία και τις μεθόδους τους.

Οι λειτουργίες που περιγράφηκαν στην προηγούμενη παράγραφο έχουν υλοποιηθεί στις παρακάτω μεθόδους.

Για τη φόρμα/κλάση *frmMain*:

- **private void hideControls()** - Η μέθοδος αυτή κάνει όλα τα στοιχεία ελέγχου της φόρμας μη ορατά.

Σχήμα 4.4: Διάγραμμα κλάσεων του HLIR XML Parser GUI.

- **private void showControls()** - Η μέθοδος αυτή κάνει όλα τα στοιχεία ελέγχου της φόρμας ορατά.
- **private static bool xmlDTDisValid = true;** - Bool πεδίο που κρατά πληροφορία για το αν το XML αρχείο είναι έγκυρο με βάση το αρχείο *hlir.dtd*. Αρχικοποιείται σε *true* και η μέθοδος *DTDValidation* του αλλάζει τιμή σε *false* αν προκύψει κάποιο σφάλμα κατά τον DTD έλεγχο.
- **private static string xmlDTDvalidationError = "";** - String πεδίο το οποίο αρχικοποιείται σαν κενό και αλλάζει τιμή αν προκύψει λάθος κατά τον DTD έλεγχο.
- **private void DTDValidation(string filename)** - Μέθοδος η οποία παίρ-

νει ένα string με το path ενός XML αρχείου ως παράμετρο και ελέγχει την εγκυρότητα του ως προς το αρχείο *hlir.dtd*. Επειδή το προς έλεγχο XML αρχείο μπορεί να μην έχει ενσωματωμένη την κατάλληλη δήλωση του DTD *<!DOCTYPE translation_unit SYSTEM "hlir.dtd">* η μέθοδος την προσθέτει στο XML και σώζει ένα προσωρινό αντίγραφο του με όνομα *temp_for_dtd_validation.xml*. Στη συνέχεια πραγματοποιείται ο έλεγχος εγκυρότητας στο προσωρινό αυτό αρχείο κάνοντας χρήση των διεπαφών *XmlReaderSettings*, *XmlReader* και *ValidationEventHandler*. Αν κατά τον έλεγχο προκύψει κάποιο λάθος τα πεδία *xmlDTDisValid* και *xmlDTDvalidationError* παίρνουν τις κατάλληλες τιμές.

- **private void btnShowAbout_Click(object sender, EventArgs e)** - Η μέθοδος αυτή εμφανίζει τη φόρμα *frmAbout* σαν modal dialog box. Την εκτελεί το *Click* event του button *btnShowAbout*, τρέχει δηλαδή μόλις γίνει κλικ στο *btnShowAbout*.

- **private void btnExit_Click(object sender, EventArgs e)** - Κλείνει τη φόρμα και τερματίζει και την εφαρμογή. Την εκτελεί το *Click* event του button *btnExit*.

- **private void btnOpenXMLFile_Click(object sender, EventArgs e)** - Ανοίγει ένα παράθυρο διαλόγου για την επιλογή ενός XML αρχείου. Το path του αρχείου που επιλέγεται δίνεται σαν τιμή στο property *Text* του textbox *txtFileXML*. Την εκτελεί το *Click* event του button *btnOpenXMLFile*.

- **private void btnRunXML_Click(object sender, EventArgs e)** - Αρχικά η μέθοδος αυτή ορίζει ένα string με το όνομα *filename* και του εκχωρεί την τιμή του property *Text* του textbox *txtFileXML*. Έτσι το *filename* αντιπροσωπεύει το path του XML αρχείου εισόδου. Έπειτα ελέγχει αν το αρχείο αυτό είναι υπαρκτό και τερματίζει στην περίπτωση που δεν υπάρχει. Αν είναι τσεκαρισμένο το checkbox *chkOptDTDValidation* καλείται η μέθοδος *DTDValidation* με το *filename* σαν παράμετρο. Δημιουργείται το string *arguments* το οποίο παίρνει την τιμή του *filename* + το string " -nograph" αν το checkbox *chkOptNoGraph* είναι τσεκαρισμένο. Επόμενο και τελευταίο βήμα της συνάρτησης είναι η εκτέλεση του αρχείου *hlir_xml_parser.exe* με την τιμή του *arguments* για παραμέτρους. Η εκτέλεση αυτή πραγματοποιείται μέσω της διεπαφής *Process*. Ακριβώς πριν από την εκτέλεση του HLIR XML Parser καλείται η μέθοδος *hideControls* και γίνεται ορατό το Label *lblExecuting*. Αφού τερματίσει τη λειτουργία του ο HLIR XML Parser καλείται η μέθοδος *showControls*, αποκρύπτεται ξανά το Label *lblExecuting* και δημιουργείται ένα αντικείμενο τύπου *frmRunResult* με όνομα *formrunresult* που αντιπροσωπεύει την αντίστοιχη φόρμα. Δίνεται η τιμή του exit code από την εκτέλεση του HLIR XML Parser σαν τιμή στο πεδίο *exitCode* του αντικειμένου *formrunresult* και η command line έξοδος σαν τιμή στο

πεδίο *cmdOutput*. Τέλος εμφανίζει τη συγκεκριμένη φόρμα σαν modal dialog. Τη μέθοδο αυτή την εκτελεί το *Click* event του button *btnRunXML*.

Για τη φόρμα/κλάση *frmRunResult*:

- **public string exitCode = "No exit code!";** - String πεδίο το οποίο έχει σαν τιμή το exit code από την εκτέλεση του HLIR XML Parser. Είναι public και παίρνει τιμή στη μέθοδο *btnRunXML_Click* της φόρμας *frmMain* πριν γίνει ορατή η φόρμα *frmRunResult*.
- **public string cmdOutput = "No CMD output!";** - String πεδίο το οποίο έχει σαν τιμή την command line έξοδο από την εκτέλεση του HLIR XML Parser. Είναι public και παίρνει τιμή στη μέθοδο *btnRunXML_Click* της φόρμας *frmMain* πριν γίνει ορατή η φόρμα *frmRunResult*.
- **public static string deleteFile(string filename)** - Μέθοδος η οποία παίρνει ένα string με το path ενός αρχείου ως παράμετρο και το διαγράφει στην περίπτωση που είναι υπαρκτό.
- **private void saveFileWithDialog(string filename, string extension)** - Μέθοδος η οποία παίρνει ένα string με το όνομα ενός αρχείου και ένα string με μια κατάληξη αρχείου ως παραμέτρους και ανοίγει ένα παράθυρο διαλόγου για αποθήκευση αρχείου. Έπειτα πραγματοποιεί αντιγραφή του *filename* από τον τρέχοντα κατάλογο στο path που έδωσε ο χρήστης στο παράθυρο διαλόγου.
- **private void frmRunResult_Load(object sender, EventArgs e)** - Δίνει στο property *Text* του Label *lblExitCode* την τιμή του πεδίου *exitCode* και στο property *Text* του Label *lblCmdOutput* την τιμή του πεδίου *cmdOutput*. Έπειτα εξετάζει αν υπάρχει το αρχείο *HLIR.gif* και εάν δεν υπάρχει απενεργοποιεί τα buttons *btnViewHLIR* και *btnSaveHLIR*. Τη μέθοδο αυτή την εκτελεί το *Load* event της φόρμας, τρέχει δηλαδή μόλις φορτώνεται η φόρμα.
- **private void frmRunResult_FormClosing(object sender, FormClosingEventArgs e)** - Καλεί τη μέθοδο *deleteFile* για τα αρχεία *HLIR.gif*, *HLIR_graph.dot* και *hlir_xml_parser_log.txt*. Τη μέθοδο αυτή την εκτελεί το *FormClosing* event της φόρμας, τρέχει δηλαδή μόλις κλείνει η φόρμα.
- **private void btnOK_Click(object sender, EventArgs e)** - Εκτελείται το αρχείο *HLIR.gif* μέσω της διεπαφής *Process.Start()*. Τη μέθοδο αυτή την εκτελεί το *Click* event του button *btnOK*.
- **private void btnViewHLIR_Click(object sender, EventArgs e)** - Κλείνει τη φόρμα. Τη μέθοδο αυτή την εκτελεί το *Click* event του button *btnViewHLIR*.
- **private void btnSaveHLIR_Click(object sender, EventArgs e)** - Καλεί τη μέθοδο *saveFileWithDialog* με παραμέτρους *("HLIR.gif", "GIF files (*.gif)|*.gif")*. Τη μέθοδο αυτή την εκτελεί το *Click* event του button *btnSaveHLIR*.

- **private void btnViewLog_Click(object sender, EventArgs e)** - Εκτελείται το αρχείο *hlir_xml_parser_log.txt* μέσω της διεπαφής *Process.Start()*. Τη μέθοδο αυτή την εκτελεί το *Click* event του button *btnViewLog*.

Συμπεράσματα - Μελλοντικές κατευθύνσεις

Σκοπός του εργαλείου που αναπτύχτηκε στην παρούσα εργασία ήταν η παροχή δυνατότητας φορητότητας μιας HIR μέσω XML. Αυτό επετεύχθη για την HIR για γλώσσα C της οποίας έγινε χρήση. Απεικονίζοντας τα δεδομένα αυτά σε XML παρέχεται ένας τρόπος αποθήκευσης της HIR ώστε αργότερα να υπάρχει η δυνατότητα ανάκτησης και ανάγνωσης της από κάποια εφαρμογή. Ακριβώς αυτό επιτυγχάνει ο HLIR XML Parser. Αν θεωρητικά υπήρχε υλοποιημένο το back end μέρος ενός compiler το οποίο λάμβανε τη συγκεκριμένη HIR σαν είσοδο τότε ο χρήστης θα είχε τη δυνατότητα να παράγει εκτελέσιμο αρχείο χρησιμοποιώντας ένα XML αρχείο αντί για τον πηγαίο κώδικα.

Ακολουθεί μια σειρά προτάσεων για τη μελλοντική εξέλιξη του εργαλείου:

Ο HLIR XML Parser θα μπορούσε με μικρές αλλαγές να χρησιμοποιηθεί και για άλλες υψηλού επιπέδου ενδιάμεσες αναπαραστάσεις δενδροειδούς μορφής για τις οποίες έχει καθοριστεί κάποιος τρόπος αναπαράστασης τους σε XML.

Με μικρές πάλι αλλαγές μπορεί να γίνει μεταφορά της εφαρμογής και σε άλλες πλατφόρμες πέραν των Windows. Για να γίνει κάτι τέτοιο, για παράδειγμα για το Linux, θα πρέπει να αλλάξει ο τρόπος με τον οποίο γίνεται η ανάλυση των XML εγγράφων καθώς το MSXML είναι διαθέσιμο μόνο στα Windows. Υπάρχουν πάρα πολλές εναλλακτικές του MSXML για τη συγκεκριμένη εργασία και οι περισσότερες από αυτές υποστηρίζονται από το Linux και άλλα λειτουργικά συστήματα. Ένα τέτοιο εργαλείο με το οποίο έγιναν κάποιες δοκιμές σε Linux είναι ο rapidXML Parser. Το εργαλείο αποτελείται στην ουσία από 4 αρχεία πηγαίου C++ κώδικα με κλάσεις οι οποίες μπορούν να χρησιμοποιηθούν συμπεριλαμβάνοντας τα αρχεία αυτά στο δικό μας κώδικα.

Σε περίπτωση μετατροπής της HIR που χρησιμοποιήθηκε, αν για παράδειγμα στο μέλλον προστεθεί σε αυτήν υποστήριξη και για άλλες γλώσσες πέραν της C, θα μπορούσε να επεκταθεί και ο HLIR XML Parser ώστε να υποστηρίζει αυτές τις νέες δυνατότητες. Υπό την προϋπόθεση ότι η μετατροπή αυτή θα υλοποιηθεί με την προσθήκη κάποιων επιπλέων συντακτικών αντικειμένων στην HIR και σε κάποια έκδοση της HLIR Library προστεθούν κλάσεις που να αναπαριστούν αυτά τα αντικείμενα, η αναβάθμιση του HLIR XML Parser δεν θα είναι δύσκολη διαδικασία. Θα πρέπει απλά να επεκταθούν οι έλεγχοι που πραγματοποιεί ώστε

να είναι σε θέση να αναλύει και τα νέα αυτά αντικείμενα.

Τέλος, μια απλή λειτουργία που λείπει από το εργαλείο και θα μπορούσε να ενσωματωθεί στο μέλλον είναι ο έλεγχος εγκυρότητας του xml που λαμβάνει σαν είσοδο κάνοντας χρήση ενός σχετικού αρχείου DTD το οποίο πρέπει πρώτα να αναπτυχθεί.

Βιβλιογραφία

[1] Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. *Compilers: Principles, techniques and tools.* Pearson, 2nd Edition, 2007.

[2] Steven S. Muchnick. *Advanced Compiler Design and Implementation.* Academic Press, 1997.

[3] Anthony A. Aaby. *Compiler Construction using Flex and Bison.* Walla Walla College, 2004.

[4] John R. Levine. *Flex & Bison.* O'Reilly, 2009.

[5] Παναγιώτης Πιντελας. *Μεταγλωττιστές.* Ελληνικό Ανοικτό Πανεπιστήμιο, 2001.

[6] Andrew W. Appel, Jens Palsberg. *Modern Compiler Implementation in Java.* Cambridge University Press, 2nd Edition, 2002.

[7] MSDN Library MSXML Documentation.
http://msdn.microsoft.com/en-us/library/ms763742%28v=vs.85%29.aspx.
Ημερομηνία τελευταίας πρόσβασης: 2012/02/08.

[8] Andrei Alexandrescu. *Modern C++ Design: Generic Programming and Design Patterns Applied.* Addison Wesley, 2001.

[9] Stephen R. Davis, Chuck Sphar. *C# 2005 For Dummies.* Wiley Publishing, 2006.

[10] Kevin H. Goldberg. *XML Visual QuickStart Guide.* Peachpit Press, 2nd Edition, 2009.

[11] W3Schools DTD Tutorial.
http://www.w3schools.com/dtd/default.asp.
Ημερομηνία τελευταίας πρόσβασης: 2012/02/08.

[12] Graphviz Documentation.
http://www.graphviz.org/Documentation.php.
Ημερομηνία τελευταίας πρόσβασης: 2012/02/08.

[13] *http://www.cs.berkeley.edu/_ cil_/cintermediate.*
Ημερομηνία τελευταίας πρόσβασης: 2012/02/08.