ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

<<Μεταφορά C++ προγραμμάτων σε διαφορετικές πλατφόρμες>>

Του φοιτητή

Αργυριάδη Χρήστου

Αρ. Μητρώου: 02/2083

Επιβλέπων καθηγητής

Ράπτης Πασχάλης

Θεσσαλονίκη 2012

Πρόλογος

Το θέμα της παρούσας πτυχιακής εργασίας είναι να μελετηθεί η C++, ως γλώσσα προγραμματισμού μεταφέρσιμων εφαρμογών. Κατά πόσο είναι δυνατή η συγγραφή ή μεταφορά C++ προγραμμάτων λογισμικού που μπορούν να μεταγλωττιστούν και να εκτελεστούν σε διαφορετικές αρχιτεκτονικές υπολογιστών. Μετά την αδιαμφισβήτητη επιτυχία της C, ως μεταφέρσιμη γλώσσα προγραμματισμού, σε μια σειρά από διαφορετικές αρχιτεκτονικές υπολογιστών, ιδιαίτερο ενδιαφέρον παρουσιάζει το κατά πόσο η C++, η οποία επηρεάστηκε σε σημαντικό βαθμό απ' την C, αποτελεί ικανή λύση για την υλοποίηση σύγχρονων μεταφέρσιμων εφαρμογών.

Για να απαντηθεί αυτή η ερώτηση, αρχικά θα επισημανθούν τα σημεία των διαφορετικών αρχιτεκτονικών, που μπορούν να δημιουργήσουν προβλήματα στη μεταφορά C++ προγραμμάτων. Οι πλατφόρμες που θα διερευνηθούν είναι οι GNU/Linux, Microsoft Windows και Mac OS X στις αρχιτεκτονικές επεξεργαστών x86 και x86_64. Αυτό το υποσύνολο αποτελεί τις δημοφιλέστερες πλατφόρμες στους ηλεκτρονικούς υπολογιστές προσωπικής χρήσης. Έπειτα, θα διερευνηθούν κάποιες μεθοδολογίες ανάπτυξης λογισμικού που μπορούν να βοηθήσουν στη συγγραφή υψηλής ποιότητας μεταφέρσιμου κώδικα, ενώ τέλος θα παρουσιαστεί μια σειρά από εργαλεία και βιβλιοθήκες για την ανάπτυξη μεταφέρσιμων C++ εφαρμογών.

Περίληψη

Παράγοντες που επηρεάζουν την μεταφερσιμότητα των προγραμμάτων / εφαρμογών που είναι γραμμένες σε C++. Βιβλιοθήκες και εργαλεία για Linux και Windows που βοηθούν / επιτρέπουν την μεταφορά προγραμμάτων/ εφαρμογών C++. Υλοποίηση προγραμμάτων που μπορούν να εκτελεστούν σε διαφορετικές πλατφόρμες.

Abstract

Conditions that affect C++ applications platform portability. Libraries and development tools for Linux and Windows help porting of C++ applications. Cross platform development of software applications.

Ευρετήριο Περιεχομένων

Πρόλογος....2
Περίληψη....3
Abstract....3
Ευρετήριο Πινάκων....5
Εισαγωγή....6
Κεφάλαιο 1: Μεταφερσιμότητα Λογισμικού (Software Portability)....8
1.1 Μεταφορά Λογισμικού σε Διαφορετικές Πλατφόρμες....8
1.2 Δια-συστηματικότητα (Cross-Platform)....8
1.3 Πλατφόρμες....9
1.4 Τομείς που επηρεάζουν την Μεταφερσιμότητα του Λογισμικού 11
1.5 Προσεγγίσεις ανάπτυξης Μεταφέρσιμου Λογισμικού....15
Κεφάλαιο 2: Περιβάλλοντα Ανάπτυξης C++ Προγραμμάτων....18
2.1 Πλατφόρμες GNU/Linux....18
2.2 Πλατφόρμες Microsoft Windows....19
2.3 Πλατφόρμες Mac OS X....24
2.4 Δια-συστηματικοί Μεταγλωττιστές....26
Κεφάλαιο 3: Μεθοδολογίες Ανάπτυξης Μεταφέρσιμου Κώδικα....27
3.1 Υποτυπώδης μεταφέρσιμος κώδικας C++....27
3.2 Εξειδίκευση κώδικα για διαφορετικές πλατφόρμες....28
3.3 Συγγραφή συντηρήσιμου κώδικα....29
3.4 Χρήση Factory Pattern....35
Κεφάλαιο 4: Μεταφέρσιμες Βιβλιοθήκες και Εργαλειοθήκες 43
4.1 Boost C++ Libraries....43
4.2 POCO C++ Libraries....47
4.3 FLTK....49
4.4 GTK+....54
4.5 Juce....57

4.6 wxWidgets.....62
4.7 QT.....69
4.8 Wt C++ - Web Toolkit.....78
4.9 MoSync.....82
Συμπεράσματα.....88
Βιβλιογραφία – Αναφορές.....91
Παραρτήματα.....93

Ευρετήριο Σχημάτων

Σχήμα 1: Εγκατάσταση του Microsoft Visual C++ 2008 Express Edition 22
Σχήμα 2: Εγκατάσταση του MinGW.....25
Σχήμα 3: Εγκατάσταση Xcode.....26
Σχήμα 4: Αρχιτεκτονική βιβλιοθήκης POCO C++.....48
Σχήμα 5: Στιγμιότυπο παράθυρου FLTK σε περιβάλλον GNU/Linux.....54
Σχήμα 6: Στιγμιότυπο παράθυρου FLTK σε περιβάλλον MS Windows.....54
Σχήμα 7: Στιγμιότυπο παραθύρων διαλόγου GTK+ σε περιβάλλον GNU/Linux.....57
Σχήμα 8: Στιγμιότυπο παραθύρων διαλόγου GTK+ σε περιβάλλον Windows.....58
Σχήμα 9: Στιγμιότυπο ενδεικτικής εφαρμογής Juce.....62
Σχήμα 10: Αρχιτεκτονική βιβλιοθήκης wxWidgets.....64
Σχήμα 11: Στιγμιότυπο εφαρμογής wxWidgets σε GNU/Linux.....69
Σχήμα 12: Στιγμιότυπο εφαρμογής wxWidgets σε MS Windows.....69
Σχήμα 13: Αρχιτεκτονική πλαισίου ανάπτυξης QT.....70
Σχήμα 14: Στιγμιότυπο εφαρμογής QT σε GNU/Linux.....78
Σχήμα 15: Στιγμιότυπο εφαρμογής QT σε MS Windows.....79
Σχήμα 16: Αρχιτεκτονική μηχανή εκτέλεσης του MoSync.....84
Σχήμα 17: Διαδικασία Μεταγλώττισης εφαρμογών MoSync.....88
Σχήμα 18: Στιγμιότυπο του εξομοιωτή MoSync.....89

Ευρετήριο Πινάκων

Πίνακας 1: Μεταγλωττιστές για την QT.....87
Πίνακας 2: Πίνακας υποστηριζόμενων πλατφορμών από MoSync.....101

Εισαγωγή

Στόχος της πτυχιακής εργασίας είναι η μελέτη και περιγραφή των παραγόντων που επηρεάζουν την μεταφερσιμότητα των C++ προγραμμάτων σε διαφορετικές πλατφόρμες καθώς και των πρακτικών που χρησιμοποιούνται για την ανάπτυξη μεταφέρσιμου λογισμικού. Τέλος, θα γίνει μια παρουσίαση των πιο δημοφιλών βιβλιοθηκών ανάπτυξης μεταφέρσιμων C++ εφαρμογών.

Στο πρώτο κεφάλαιο, *“Μεταφερσιμότητα Λογισμικού (Software Portability)“*, γίνεται μια περιγραφή των βασικών όρων και χαρακτηριστικών που διέπουν τη μεταφερσιμότητα του λογισμικού. Ορίζονται οι πλατφόρμες υλικού και λογισμικού και επισημαίνονται οι διαφορές οι οποίες μπορούν να επηρεάσουν τη μεταφερσιμότητα μιας εφαρμογής C++. Παρουσιάζονται επίσης μερικές απ' τις σχεδιαστικές πρακτικές ανάπτυξης μεταφέρσιμων εφαρμογών.

Στο δεύτερο κεφάλαιο, *“Περιβάλλοντα Ανάπτυξης C++ Προγραμμάτων“*, παρουσιάζονται τα διάφορα περιβάλλοντα ανάπτυξης C++ εφαρμογών για τις πλατφόρμες GNU/ Linux, Microsoft Windows και Mac OS X και περιγράφεται η εγκατάσταση και χρήση τους για τη μεταγλώττιση μιας απλής εφαρμογής C++ στην κάθε μία. Αναφορά γίνεται και στους δια-συστηματικούς μεταγλωττιστές (cross compilers).

Στο τρίτο κεφάλαιο, *“Μεθοδολογίες Ανάπτυξης Μεταφέρσιμου Κώδικα“*, παρουσιάζονται διάφορες μεθοδολογίες ανάπτυξης μεταφέρσιμου λογισμικού με έμφαση στην ποιότητα του λογισμικού. Τρόποι οργάνωσης του μη μεταφέρσιμου κώδικα μιας C++ εφαρμογής με στόχο την συγγραφή ευανάγνωστου και συντηρήσιμου κώδικα.

Στο τέταρτο κεφάλαιο, *“Μεταφέρσιμες Βιβλιοθήκες και Εργαλειοθήκες “*, παρουσιάζονται κάποιες απ' τις δημοφιλέστερες βιβλιοθήκες ανάπτυξης μεταφέρσιμων C++ εφαρμογών. Περιγράφεται η αρχιτεκτονική τους και η μεταφερσιμότητα τους στις διάφορες πλατφόρμες. Τέλος, παρουσιάζονται τα εργαλεία ανάπτυξης για την υλοποίηση και διανομή της τελικής μεταφέρσιμης εφαρμογής λογισμικού.

Κεφάλαιο 1: Μεταφερσιμότητα Λογισμικού (Software Portability)

Η μεταφερσιμότητα λογισμικού είναι ένα χαρακτηριστικό του προγραμματισμού στις γλώσσες υψηλού επιπέδου. Χαρακτηρίζει τον κώδικα λογισμικού και τον βαθμό με τον οποίο μπορεί να μεταφερθεί αμετάβλητος σε διαφορετικές πλατφόρμες. Βασική αρχή για την επαναχρησιμοποίηση του μεταφέρσιμου (portable) κώδικα, είναι η εφαρμογή της αφαιρετικότητας (abstraction) ανάμεσα στην λογική της εφαρμογής και στις διεπαφές του συστήματος.

1.1 Μεταφορά Λογισμικού σε Διαφορετικές Πλατφόρμες

Ως μεταφορά ενός προϊόντος λογισμικού σε διαφορετικές πλατφόρμες (Porting), ορίζεται η διαδικασία που απαιτείται για την παραγωγή εκτελέσιμου προγράμματος, ικανό να λειτουργήσει σε διαφορετικό περιβάλλον, απ' το αρχικό περιβάλλον ανάπτυξης στο οποίο σχεδιάστηκε. Η διαφορετικότητα της πλατφόρμας μπορεί να περιλαμβάνει διαφορετική αρχιτεκτονική υλικού, λειτουργικού συστήματος ή διαφορετικές βιβλιοθήκες τρίτων.

1.2 Δια-συστηματικότητα (Cross-Platform)

Η δια-συστηματικότητα είναι ένα χαρακτηριστικό του λογισμικού το οποίο περιγράφει τη δυνατότητα του να λειτουργήσει με τον ίδιο τρόπο σε διαφορετικές πλατφόρμες. Τα cross-platform πακέτα λογισμικού μπορούν να χωριστούν σε δύο κατηγορίες ανάλογα με τη διαδικασία που απαιτείται ώστε να μεταφερθούν και να εκτελεστούν σε άλλη πλατφόρμα:

- **αυτά που απαιτούν την επαναμεταγλώττιση του λογισμικού για κάθε διαφορετική πλατφόρμα**

- **αυτά που δεν απαιτούν κάποια ειδική προετοιμασία**

Περιπτώσεις λογισμικού που απαιτούν επαναμεταγλώττιση, αποτελούν τα προϊόντα λογισμικού που έχουν γραφτεί σε παραδοσιακές γλώσσες προγραμματισμού οι οποίες παράγουν εκτελέσιμα αρχεία συστήματος. Απαραίτητη προϋπόθεση είναι η ύπαρξη μεταγλωττιστή της γλώσσας προγραμματισμού για τις υπόλοιπες πλατφόρμες στις οποίες το λογισμικό πρέπει να μεταφερθεί.

Περιπτώσεις λογισμικού που δεν απαιτούν κάποια ιδιαίτερα προετοιμασία για την εκτέλεση τους σε διαφορετικές πλατφόρμες, είναι οι διερμηνευμένες γλώσσες (interpreted languages) και αυτές που παράγουν προ-μεταγλωττισμένο κώδικα (byte-code). Σε αυτή την περίπτωση πρέπει ο διερμηνευτής (interpreter) ή τα πακέτα εκτέλεσης να είναι διαθέσιμα και κοινά στις διαφορετικές πλατφόρμες.

1.3 Πλατφόρμες

Ως πλατφόρμα μπορεί να οριστεί ο συνδυασμός υλικού και λογισμικού που χρησιμοποιείται για την εκτέλεση ενός προγράμματος λογισμικού. Συνήθως μια πλατφόρμα περιγράφεται από τον συνδυασμό της αρχιτεκτονικής του υλικού και το λειτουργικό του συστήματος που χρησιμοποιείται σε αυτό το υλικό.

Πλατφόρμα Υλικού

Μια Πλατφόρμα Υλικού μπορεί να αναφέρεται στην αρχιτεκτονική ενός Η/Υ ή της κεντρικής μονάδας επεξεργασίας (CPU) του. Κάθε πλατφόρμα υλικού ή οικογένεια κεντρικών μονάδων επεξεργασίας (CPU), έχει τη

δικής της μοναδική γλώσσα μηχανής. Για παράδειγμα, η πλατφόρμα υλικού x86 και x86_64 περιγράφει έναν Η/Υ με κεντρική μονάδα επεξεργασίας (CPU) βασισμένη στην x86 ή x86_64 αρχιτεκτονική . Άλλες δημοφιλείς πλατφόρμες υλικού είναι οι IBM Power Systems, Sun SPARC και ARM.

Πλατφόρμα Λογισμικού

Μια πλατφόρμα λογισμικού αποτελείται απ' το λειτουργικό σύστημα και πολύ συχνά απ' το αντίστοιχο περιβάλλον ανάπτυξης του λογισμικού χρήστη. Μια πλατφόρμα λογισμικού μπορεί να είναι διαθέσιμη για περισσότερες από μία πλατφόρμες υλικού. Μερικά απ' τα δημοφιλέστερα λειτουργικά συστήματα για υπολογιστές προσωπικής χρήσης είναι τα Microsoft Windows, GNU/Linux και το Apple Mac OS X.

Εικονική Μηχανή Java

Αξιοσημείωτη εξαίρεση αποτελεί στις πλατφόρμες λογισμικού η Java. Η Java εφαρμογές εκτελούνται σε ένα ανεξάρτητο λειτουργικού συστήματος, εικονικό μηχάνημα (Virtual Machine) το οποίο “κρύβει” απ' την εφαρμογή τις τεχνικές λεπτομέρειες του λειτουργικού συστήματος και της πλατφόρμας υλικού απ' την τελική εφαρμογή Java η οποία μπορεί να εκτελεστεί αυτούσια σε όλες τις πλατφόρμες στις οποίες είναι διαθέσιμη η εικονική μηχανή της Java (Java Virtual Machine). Θα μπορούσε λοιπόν η εικονική μηχανή της Java να θεωρηθεί ως μια διαφορετική πλατφόρμα (υλικού και λογισμικού) πάνω στην οποία εκτελούνται οι εφαρμογές Java. Παρόλα αυτά η εικονική μηχανή της Java αποτελεί ένα cross-platform λογισμικό το οποίο διέπεται απ' τις ίδιες αρχές οποιουδήποτε άλλου cross-platform λογισμικού.

1.4 Τομείς που επηρεάζουν την Μεταφερσιμότητα του Λογισμικού

Γλώσσα Προγραμματισμού

Ένας βασικός παράγοντας που επηρεάζει την μεταφερσιμότητα του λογισμικού είναι η γλώσσα προγραμματισμού που έχει χρησιμοποιηθεί για τη συγγραφή του. Η γλώσσα προγραμματισμού θα πρέπει να είναι υψηλού επιπέδου ώστε να παρέχει αφαιρετικά εργαλεία που κρύβουν τις τεχνικές λεπτομέρειες της πλατφόρμας. Επίσης είναι σημαντικό να υπάρχουν μεταγλωττιστές για τη συγκεκριμένη γλώσσα προγραμματισμού που μπορούν να παράγουν εκτελέσιμα για διαφορετικές πλατφόρμες.

Η C++ κληρονόμησε πολλά πλεονεκτήματα, αλλά και κάποια μειονεκτήματα, της C μιας και αρχικά ξεκίνησε ως μια επέκταση της δεύτερης. Η C αρχικά σχεδιάστηκε ως γλώσσα προγραμματισμού για την υλοποίηση λογισμικού συστήματος, αλλά γρήγορα χρησιμοποιήθηκε για την ανάπτυξη μεταφέρσιμου (portable) λογισμικού χρήστη. Η C ήδη στα τέλη του 1970 θεωρούταν μεταφέρσιμη γλώσσα προγραμματισμού και αποτέλεσε μία απ' τις βασικότερες αιτίες που το UNIX λειτουργικό σύστημα μεταφέρθηκε σε τόσες πολλές διαφορετικές πλατφόρμες υλικού.

Η τυποποίηση της, κατά ANSI αρχικά και αργότερα κατά C99 συντέλεσαν ώστε η C να γίνει ακόμα πιο μεταφέρσιμη (portable).

Μεταγλωττιστές

Η C++, έτσι όπως έχει οριστεί, αφήνει τις λεπτομέρειες υλοποίησης της γλώσσας στον κατασκευαστή του μεταγλωττιστή. Έτσι κάποια χαρακτηριστικά μπορεί να μην είναι μεταφέρσιμα μεταξύ

μεταγλωττιστών διαφορετικών κατασκευαστών. Μερικά από αυτά τα χαρακτηριστικά είναι τα παρακάτω:

- **μέγεθος βασικών αριθμητικών τύπων δεδομένων**

Το μέγεθος των τύπων δεδομένων short, int και long μπορεί να είναι διαφορετικό σε διάφορους μεταγλωττιστές. Η τυποποίηση της C++ αναφέρει ότι οι short μεταβλητές πρέπει να έχουν μέγεθος τουλάχιστον 16bit. Επίσης, οι int πρέπει να έχουν τουλάχιστον το μέγεθος των short μεταβλητών, καθώς και ότι οι long πρέπει να έχουν τουλάχιστον το μέγεθος των int. Αυτό δίνει τη δυνατότητα στους κατασκευαστές μεταγλωττιστών να υλοποιήσουν με διαφορετικά μεγέθη αυτές τις μεταβλητές. Για παράδειγμα σε ένα 32-bit περιβάλλον οι short μπορούν ορισθούν ως 16-bit και οι int και οι long ως 32-bit. Μπορούν όμως να ορισθούν και με ίδιο μέγεθος, 32-bit για τις short και 32-bit για τις int και long. Η υλοποίηση του ίδιου μεταγλωττιστή σε ένα 64-bit περιβάλλον δίνει ακόμη περισσότερες επιλογές, αφού ο κατασκευαστής μπορεί πιθανότατα να υλοποίηση το μέγεθος της int και της long μεταβλητής στο μέγεθος λέξης της πλατφόρμας 64-bit.

- **Δυαδικοί Τελεστές**

Σφάλματα μπορούν να προκύψουν αν θεωρήσουμε δεδομένο το μέγεθος των int, short ή long όταν εκτελούμε υπολογισμούς με δυαδικούς τελεστές, μιας και το μέγεθος αυτών των τύπων δεδομένων καθορίζεται απ' τον κατασκευαστή του μεταγλωττιστή. Επίσης εφαρμόζοντας ολίσθηση (shift) σε μια μεταβλητή μπορεί να προκληθεί αλλαγή του bit προσήμου ή μηδενισμό των αριστερών bit της μεταβλητής μιας και αυτό για άλλη μια φορά εξαρτάται απ' την υλοποίηση του κατασκευαστή.

- **τύποι δεδομένων χαρακτήρα με πρόσημο ή χωρίς**

Οι προδιαγραφές της C και της C++ δεν καθορίζουν αν ο τύπος δεδομένων χαρακτήρα (char) είναι ακέραιος με πρόσημο ή χωρίς. Για άλλη μια φορά ο καθορισμός αυτού έχει αφεθεί στον σχεδιαστή του μεταγλωττιστή. Η διαφορά αυτή στον τύπο δεδομένων χαρακτήρα μπορεί να δημιουργήσει πρόβλημα. Για παράδειγμα όταν χρησιμοποιηθεί η getchar() για την ανάγνωση απ' το stdin. Η getchar() επιστρέφει -1 όταν διαβάσει EOF. Αν όμως η τιμή που επιστρέφει η getchar() αποθηκευθεί σε μια μεταβλητή char η οποία έχει υλοποιηθεί απ' τον μεταγλωττιστή ως unsigned int δε θα αναγνωριστεί πότε το EOF διαβάζοντας την τιμή της μεταβλητής char.

Δεν είναι όμως καθόλου σπάνιο οι κατασκευαστές μεταγλωττιστών να παραβιάζουν τις προδιαγραφές της C++. Έτσι ακόμα και σε μια συγκεκριμένη πλατφόρμα, οι μεταγλωττιστές διαφορετικών κατασκευαστών συχνά προσφέρουν διαφορετικά σετ εντολών ή υλοποιήσεις που μπορεί να δημιουργήσουν προβλήματα στη μεταφερσιμότητα του λογισμικού. Είναι λοιπόν απαραίτητο να αποφεύγεται η χρήση συγκεκριμένων σετ εντολών (vendor-specific) που είναι διαθέσιμες μόνο στην υλοποίηση ενός συγκεκριμένου μεταγλωττιστή.

Δυαδικά Δεδομένα

Η αποθήκευση δυαδικών δεδομένων στη μνήμη ή στο δίσκο είναι απ' τις πλέον μη-μεταφέρσιμες μεθόδους αποθήκευσης δεδομένων, μιας και η δομή με την οποία αποθηκεύονται μπορεί να είναι τελείως διαφορετική από μεταφραστή σε μεταφραστή. Για να εξασφαλιστεί η μεταφερσιμότητα των αποθηκευμένων δεδομένων είναι προτιμότερο να γίνεται σε μορφή text, όταν αυτό είναι εφικτό.

Διεπαφές Λειτουργικού Συστήματος

Εξ' ορισμού οι διεπαφές λειτουργικού συστήματος (system calls) είναι άμεσα συνδεδεμένες με το εκάστοτε λειτουργικό σύστημα και ο τρόπος κλήσης μπορεί να διαφέρει σημαντικά σε κάθε λειτουργικό σύστημα. Διάφορες τυποποιήσεις έχουν σκοπό να καθορίσουν συγκεκριμένα πρότυπα τρόπου κλήσης των διεπαφών συστήματος, αλλά αφορούν συγκεκριμένες κατηγορίες λειτουργικών συστημάτων. Επίσης οι Standard Libraries της C++ προσπαθούν να χειριστούν αφαιρετικά αυτές τις διαφοροποιήσεις μεταξύ των λειτουργικών συστημάτων, αν και η χρήση βιβλιοθηκών τρίτων κατασκευαστών, είναι πολλές φορές απαραίτητη για να επιτευχθεί η φορητότητα του πηγαίου κώδικα.

Διεπαφές Χρήστη

Η διεπαφή χρήστη σε μια πλατφόρμα επιτραπέζιου υπολογιστή, είναι ίσως το λιγότερα φορητό μέρος ενός λογισμικού. Κάθε πλατφόρμα υλοποιεί τη δική του εργαλειοθήκη για την δημιουργία του γραφικού περιβάλλοντός διεπαφής χρήστη με πολύ μεγάλες διαφορές στο API.

Στα λειτουργικά συστήματα Windows, για τη δημιουργία native GUI μπορεί να χρησιμοποιηθούν εναλλακτικά οι εργαλειοθήκες Win32, MFC και προσφάτως η .NET βιβλιοθήκη. Στα Mac OS X είναι διαθέσιμη η εργαλειοθήκη Cocoa ή Carbon. Στα συστήματα POSIX πάλι υπάρχει μια πλειάδα από διαφορετικές εργαλειοθήκες όπως η Gtk+ (Gnome), η Qt (KDE) καθώς και πολλές άλλες οι οποίες όμως όλες βασίζονται στο X Window System.

Περιβάλλον Ανάπτυξης Λογισμικού (Build System)

Κάθε προγραμματιστής επιλέγει την πλατφόρμα στην οποία θα γίνει η βασική ανάπτυξη το λογισμικού. Είναι όμως απαραίτητο κατά

διαστήματα να δοκιμάζει τη μεταγλώττιση του προγράμματος και στις υπόλοιπες πλατφόρμες στις οποίες το λογισμικό θα είναι διαθέσιμο. Η μεταγλώττιση ενός προγράμματος σε διαφορετικές πλατφόρμες μπορεί να διαφέρει σημαντικά, γι' αυτό μια καλή πρακτική είναι η χρήση φορητών (cross-platform) εργαλείων μεταγλώττισης.

Επίσης κάθε προγραμματιστής ανάλογα με τη βασική πλατφόρμα ανάπτυξης που επιλέγει, μπορεί να επιλέξει διαφορετικό περιβάλλον ανάπτυξης (IDE) το οποίο προσφέρεται στη συγκεκριμένη πλατφόρμα. Υπάρχουν διάφορα περιβάλλοντα ανάπτυξης λογισμικού που είναι φορητά (cross-platform) όμως σε κάποιες περιπτώσεις ο μεταγλωττιστής μια συγκεκριμένης πλατφόρμας είναι άμεσα συνδεδεμένος με το περιβάλλον ανάπτυξης που παρέχει ο κατασκευαστή του. Σε αυτή την περίπτωση είναι σημαντικό να ρυθμιστούν τα διαφορετικά περιβάλλοντα ανάπτυξης ώστε να υπάρχει ομοιομορφία στη μορφοποίηση του κώδικα και να μη προκαλούν αλλαγές στον κώδικα οι οποίες δεν επιτρέπουν πλέον τον κώδικα να μεταγλωττιστεί σε άλλη πλατφόρμα.

1.5 Προσεγγίσεις ανάπτυξης Μεταφέρσιμου Λογισμικού

Υπάρχουν διαφορετικές προσεγγίσεις για την αντιμετώπιση του προβλήματος ανάπτυξης μεταφέρσιμου λογισμικού. Μια τέτοια προσέγγιση θα μπορούσε να θεωρηθεί η χρήση διαφορετικών εκδόσεων του πηγαίου κώδικα για κάθε πλατφόρμα. Αν και αυτή η προσέγγιση φαίνεται να είναι η πιο ξεκάθαρη, έχει αρκετά μειονεκτήματα. Το κόστος ανάπτυξης και συντήρησης μπορεί να αυξηθεί σημαντικά. Πιθανότατα διαφορετικοί προγραμματιστές θα υλοποιούν το λογισμικό σε κάθε πλατφόρμα με αποτέλεσμα, διαφορετικά προβλήματα (bugs) να προκύπτουν σε κάθε πλατφόρμα του λογισμικού. Έτσι θα είναι δύσκολο να εξασφαλιστεί η ομοιομορφία και ποιότητα του λογισμικού στις

διαφορετικές πλατφόρμες.

Ένα λογισμικό για να θεωρηθεί μεταφέρσιμο (portable) και να μπορεί να συντηρηθεί παράλληλα σε όλες τις πλατφόρμες στις οποίες είναι διαθέσιμο πρέπει να έχει ως βάση κοινό πηγαίο κώδικα με εξειδίκευση μόνο στα σημεία τα οποία είναι διαφορετικά σε κάθε πλατφόρμα.

Αφαιρετικότητα (Abstraction)

Η ιδέα της μεταφοράς κώδικα σε διαφορετικές πλατφόρμες βασίζεται στην έννοια της αφαιρετικότητας. Όπως η assembly χρησιμοποιήθηκε για να αφαιρέσει των τεχνικών λεπτομερειών που θα απαιτούσε η χρήση δυαδικού κώδικα στις διαφορετικές μηχανές, έτσι η γλώσσα προγραμματισμού C χρησιμοποιήθηκε για να προσφέρει μεγαλύτερη φορητότητα στις διαφορετικές πλατφόρμες που είναι διαθέσιμη. Αν και το επίπεδο αφαιρετικότητας που πρόσθεσε η C/C++ βοήθησε στην μεταφορά του Unix λειτουργικού σε πάρα πολλές διαφορετικές πλατφόρμες υλικού, δεν είναι αρκετή για να προσφέρει τον απαραίτητο βαθμό αφαιρετικότητας στα προγράμματα τελικού χρήστη που πρέπει να μεταγλωττιστούν στην πλειάδα των σύγχρονων πλατφορμών λογισμικού.

Μια άλλη προσέγγιση του προβλήματος είναι η χρήση προϋπάρχοντος λογισμικού αφαιρετικότητας πλατφόρμας, το οποίο θα αποκρύπτει τις ιδιαιτερότητες της κάθε πλατφόρμας απ' την εφαρμογή τελικού χρήστη. Έτσι το λογισμικό τελικού χρήστη ζητά απ' το ενδιάμεσο λογισμικό, να υλοποιήσει κατάλληλα τις κλήσεις που παρουσιάζουν διαφορετικότητα σε κάθε πλατφόρμα. Με αυτό τον τρόπο το λογισμικό τελικού χρήστη, διατηρεί κοινό πηγαίο κώδικα ο οποίος με τη βοήθεια του ενδιάμεσου λογισμικού αφαιρετικότητας πλατφόρμας μπορεί να μεταφραστεί και να εκτελεστεί αυτούσιο στις διαφορετικές

πλατφόρμες. Χαρακτηριστικό παράδειγμα ενδιάμεσου λογισμικού αφαιρετικότητας πλατφόρμας αποτελεί η εικονική μηχανή της Java. Στην προσέγγιση της Java ο ίδιος πηγαίος κώδικας, αλλά και ο μεταγλωττισμένος (byte-code) δύναται να εκτελεστεί άμεσα σε διαφορετικές πλατφόρμες μέσω της εικονικής μηχανής.

Οι ολοκληρωμένες λύσεις λογισμικού αφαίρεσης πλατφόρμας παρουσιάζουν διάφορα μειονεκτήματα, δημιουργώντας την ανάγκη σε κάποιες περιπτώσεις χρήσης μια υβριδικής προσέγγισης που συνδυάζει τις δύο παραπάνω προσεγγίσεις. Έτσι πολλές φορές χρησιμοποιείται μια κοινή βάση πηγαίου κώδικα η οποία εξειδικεύει μέρη του λογισμικού που παρουσιάζουν διαφορετικότητα σε κάθε πλατφόρμα. Χαρακτηριστικό παράδειγμα αποτελεί ο μεταφέρσιμος φυλλομετρητής Firefox του Mozilla. Ο Firefox χρησιμοποιεί αφαιρετικότητα στην υλοποίηση των χαμηλού επιπέδου μερών του, διαφορετικά δέντρα πηγαίου κώδικα για την υλοποίηση χαρακτηριστικών διαφορετικά σε κάθε πλατφόρμα (π.χ. Γραφικό περιβάλλον) και διάφορες scripting languages για άλλα χαρακτηριστικά του φυλλομετρητή.

Κεφάλαιο 2: Περιβάλλοντα Ανάπτυξης C++ Προγραμμάτων

Οι πιο δημοφιλείς πλατφόρμες για την ανάπτυξη λογισμικού τελικού χρήστη, περιλαμβάνουν τα λειτουργικά Microsoft Windows, GNU/Linux και Mac OS X σε πλατφόρμες x86, x86-64 και PowerPC. Σε κάποιες περιπτώσεις είναι δυνατή η ανάπτυξη μεταφέρσιμου λογισμικού σε μία μόνο πλατφόρμα, είναι αναπόφευκτο όμως τελικά να χρειαστούμε πρόσβαση στην κάθε μία πλατφόρμα που το μεταφέρσιμο λογισμικό στοχεύει.

2.1 Πλατφόρμες GNU/Linux

Ένα απ' τα πλεονεκτήματα της πλατφόρμας GNU/Linux είναι η ευκολία που παρέχει στην προετοιμασία του περιβάλλοντος ανάπτυξης. Οι περισσότερες διανομές GNU/Linux παρέχουν κάποιο διαχειριστή πακέτων λογισμικού ο οποίος προσφέρει εύκολη εγκατάσταση του μεταγλωττιστή C++ καθώς και μια πλειάδα από διαθέσιμες και ελεύθερες προς χρήση (GPL License) βιβλιοθήκες.

Για τις ανάγκες αυτής της εργασίας θα χρησιμοποιήσουμε τη διανομή Ubuntu η οποία είναι μια διανομή βασισμένη στο Debian. Η διανομή Ubuntu, είναι συμβατή με τα πακέτα λογισμικού τύπου .deb και χρησιμοποιεί τον διαχειριστή πακέτων APT, ο οποίος προσφέρει εύκολη λήψη, εγκατάσταση και ενημέρωση όλων των διαθέσιμων εφαρμογών λογισμικού.

Τα αποθετήρια του Ubuntu διαθέτουν ένα μετά-πακέτο το οποίο περιέχει όλα τα απαραίτητα πακέτα για τον μεταγλωττιστή της C/C++. Για να το εγκαταστήσουμε αρκεί να εκτελέσουμε το παρακάτω σε ένα παράθυρο τερματικού.

```
argy@Freak:~$ sudo apt-get install build-essential
```

Αφού ολοκληρωθεί η λήψη και εγκατάσταση μπορούμε να ελέγξουμε την έκδοση και το προεπιλεγμένο configuration του GNU C++ μεταγλωττιστή μας.

```
argy@Freak:~$ g++ -v
Using built-in specs.
Target: i686-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu/Linaro 4.4.4-14ubuntu5' --with-bugurl=file:///usr/share/doc/gcc-4.4/README.Bugs --enable-languages=c,c++,fortran,objc,obj-c++ --prefix=/usr --program-suffix=-4.4 --enable-shared --enable-multiarch --enable-linker-build-id --with-system-zlib --libexecdir=/usr/lib --without-included-gettext --enable-threads=posix --with-gxx-include-dir=/usr/include/c++/4.4 --libdir=/usr/lib --enable-nls --with-sysroot=/ --enable-clocale=gnu --enable-libstdcxx-debug --enable-objc-gc --enable-targets=all --disable-werror --with-arch-32=i686 --with-tune=generic --enable-checking=release --build=i686-linux-gnu --host=i686-linux-gnu --target=i686-linux-gnu
Thread model: posix
gcc version 4.4.5 (Ubuntu/Linaro 4.4.4-14ubuntu5)
```

Τώρα μπορούμε να δοκιμάσουμε να μεταγλωττίσουμε το ελάχιστο C++ πρόγραμμα Hello World του παραρτήματος (Παράρτημα Α) χρησιμοποιώντας τον GNU C++ μεταγλωττιστή.

```
argy@Freak:~$ g++ helloworld.cxx -o helloworld
```

2.2 Πλατφόρμες Microsoft Windows

Στα λειτουργικά Microsoft Windows, υπάρχει μια σειρά από εργαλεία ανάπτυξης λογισμικού για την C++ τα οποία παρουσιάζουν σημαντικές διαφορές από κατασκευαστή σε κατασκευαστή. Τα εργαλεία με τα οποία θα ασχοληθούμε είναι το Microsoft Visual C++ του οποίου η

Express έκδοση διατίθεται δωρεάν απ' τη Microsoft καθώς και το περιβάλλον ανάπτυξης MinGW το οποίο διατίθεται υπό τους όρους του ελεύθερου λογισμικού.

Microsoft Visual C++ Express Edition 2008

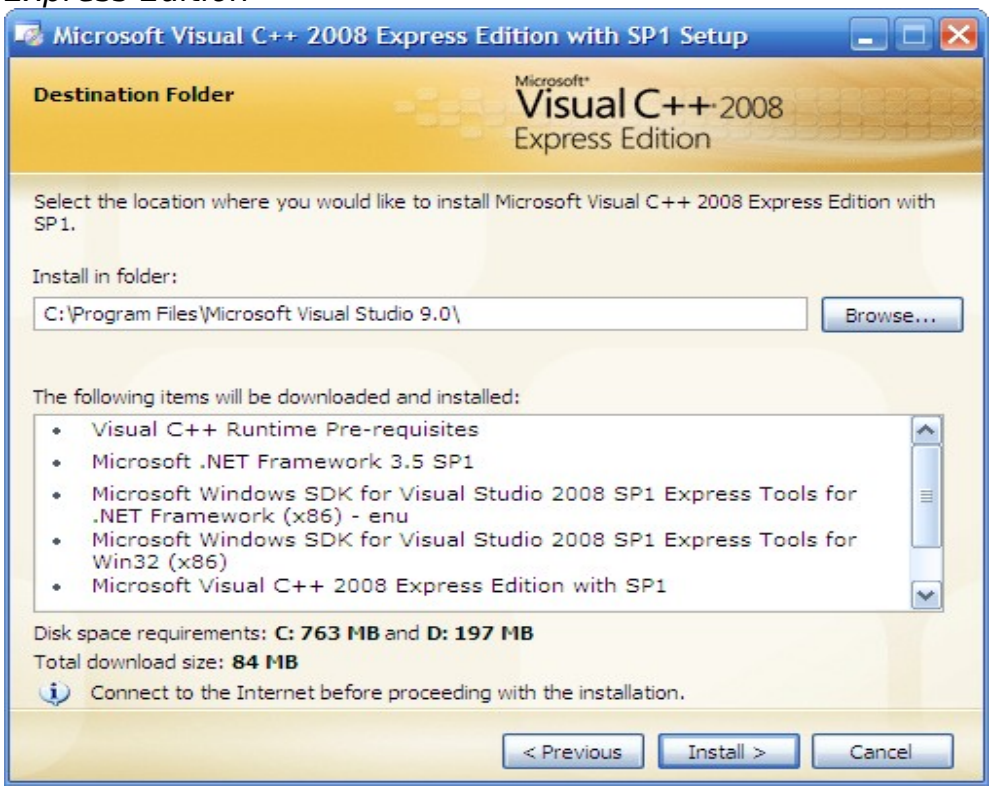
Το Microsoft Visual C++ Express Edition είναι διαθέσιμο για δωρεάν λήψη και χρήση απ' τη σελίδα της Microsoft. Πέρα απ' τον μεταγλωττιστή C/C++ εμπεριέχει και ένα ολοκληρωμένο IDE με πολλά εργαλεία ανάπτυξης και γραφικό περιβάλλον παραμετροποίησης του μεταγλωττιστή.

Η εγκατάσταση του είναι συνήθως εύκολη, μιας και οδηγός εγκατάστασης αναλαμβάνει την εγκατάσταση και παραμετροποίηση του περιβάλλοντος του συστήματος.

Με την ολοκλήρωση της εγκατάστασης δημιουργούνται δύο συντομεύσεις, μία για το γραφικό περιβάλλον ανάπτυξης (IDE) και μία για το Τερματικό Εκτέλεσης Εντολών (Visual Studio 2008 Command Prompt) με όλες τις απαραίτητες μεταβλητές συστήματος για την μεταγλώττιση C++ προγραμμάτων απ' το τερματικό.

Το Visual Studio C++ προσφέρει δύο τρόπους για μεταγλώττιση C++ εφαρμογών, είτε μέσω του Τερματικού Εντολών καλώντας απευθείας τον μεταγλωττιστή της C++ (cl.exe), είτε μέσω του γραφικού περιβάλλοντος ανάπτυξης (IDE) με τη δημιουργία λύσεων λογισμικού (Solutions) και την αυτόματη κλήση του μεταγλωττιστή.

Σχήμα 1: Εγκατάσταση του Microsoft Visual C++ 2008 Express Edition



Μπορούμε έτσι να μεταγλωττίσουμε το ενδεικτικό κομμάτι C++ κώδικα του παραρτήματος Α (Παράδειγμα ελάχιστου C++ προγράμματος Hello World) μέσω του τερματικού εντολών καλώντας απευθείας τον μεταγλωττιστή της C++.

```
C:\dev\crossessay>cl /EHsc helloworld.cxx
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 15.00.30729.01 for
80x86
Copyright (C) Microsoft Corporation.  All rights reserved.

helloworld.cxx
Microsoft (R) Incremental Linker Version 9.00.30729.01
```

```
Copyright (C) Microsoft Corporation.  All rights reserved.


/out:helloworld.exe

helloworld.obj
```

Εναλλακτικά και κυρίως για μεγαλύτερες εφαρμογές μπορούμε να χρησιμοποιήσουμε το γραφικό περιβάλλον ανάπτυξης δημιουργώντας ένα νέο έργο (Project) από το μενού File -> New Project. Για το παράδειγμα του παραρτήματος Α (Παράδειγμα ελάχιστου C++ προγράμματος Hello World) θα χρησιμοποιήσουμε το πρότυπο (Template) Win32 Console Application. Στον οδηγό εφαρμογής τερματικού μπορούμε να επιλέξουμε Empty Project και να δημιουργήσουμε ένα νέο αρχείο κώδικα C++ (C++ File) κάτω απ' τον φάκελο Source Files με το όνομα helloworld.cpp. Έπειτα από το μενού Build -> Build Helloworld μπορούμε να ζητήσουμε απ' το γραφικό περιβάλλον να μεταγλωττίσει τον κώδικα μας. Εξ' ορισμού παρέχονται δύο προ-ρυθμισμένες επιλογές μεταγλώττισης, μία για αποσφαλμάτωση (Debug) και μία για τελική διανομή (Release). Με την ολοκλήρωση της μεταγλώττισης τα τελικά εκτελέσιμα θα τοποθετηθούν στους αντίστοιχους φακέλους Debug και Release.

MinGW

Το MinGW (Minimalist GNU for Windows), είναι ένα μινιμαλιστικό περιβάλλον ανάπτυξης για εφαρμογές Microsoft Windows. Προσφέρει ένα ολοκληρωμένο σύνολο εργαλείων προγραμματισμού ανοιχτού κώδικα για τις πλατφόρμες Microsoft Windows.

Το MinGW περιλαμβάνει εκδόσεις για Microsoft Windows:

- της GNU Compiler Collection (GCC), που περιλαμβάνει υποστήριξη για μεταγλώττιση πηγαίου κώδικα σε C, C++, ADA και Fortan

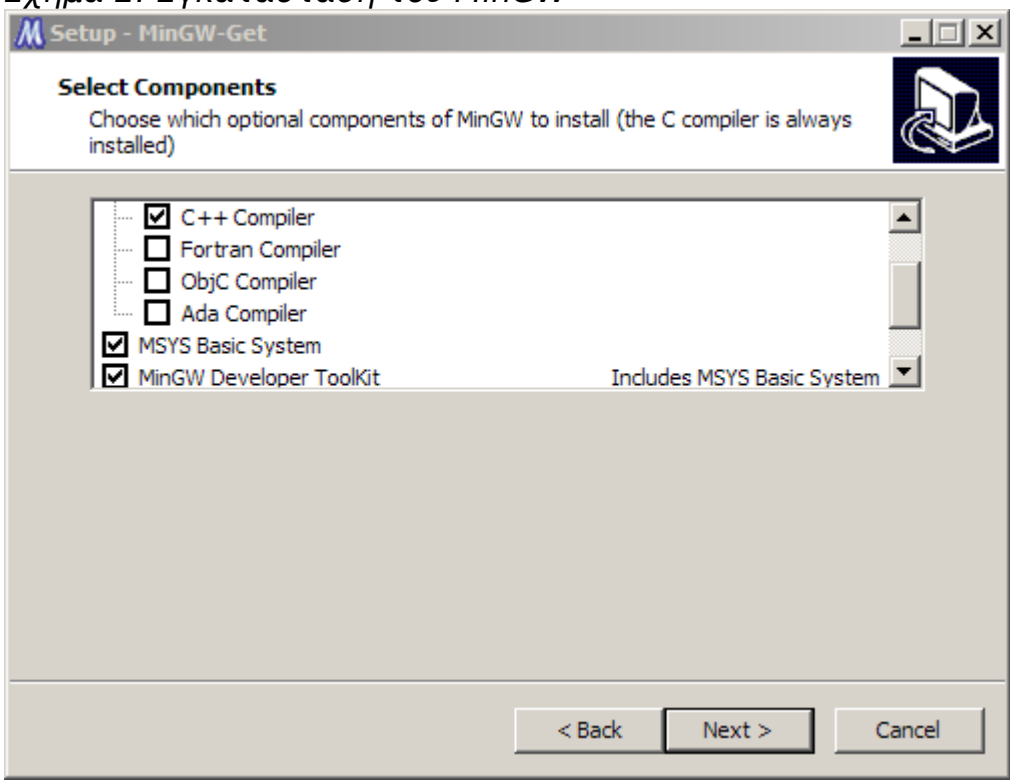
- του GNU Binutils για Windows (assembler, linker, archive manager)

- εργαλείου εγκατάστασης (εκδόσεις γραφικού περιβάλλοντος και κονσόλας τερματικού) για το MinGW και MSYS

- Το MSYS (Minimal SYStem), είναι ένας Bourne shell μεταγλωττιστής, ο οποίος προσφέρεται ως εναλλακτικό για την κονσόλα τερματικού των Windows (cmd.exe) με στόχο να διευκολύνει την μεταφορά των εφαρμογών ανοιχτού κώδικα για POSIX συστήματα στις πλατφόρμες των Microsoft Windows.

Η εγκατάσταση του περιβάλλοντος ανάπτυξης MinGW στα Windows είναι πλέον πολύ εύκολη με τη χρήση του Automatic MinGW Installer. Μετά τη λήψη του γραφικού οδηγού εγκατάστασης του MinGW, αφού επιλέξουμε τα στοιχεία που θέλουμε να εγκαταστήσουμε ο οδηγός εγκατάστασης αναλαμβάνει να λάβει τις τελευταίες εκδόσεις των εργαλείων που επιλέξαμε, απ' το internet και να τις εγκαταστήσει.

Σχήμα 2: Εγκατάσταση του MinGW



Αφού ολοκληρωθεί η εγκατάσταση είμαστε έτοιμοι να μεταγλωττίσουμε το hello world C++ κώδικα μας στην πλατφόρμα Windows Microsoft.

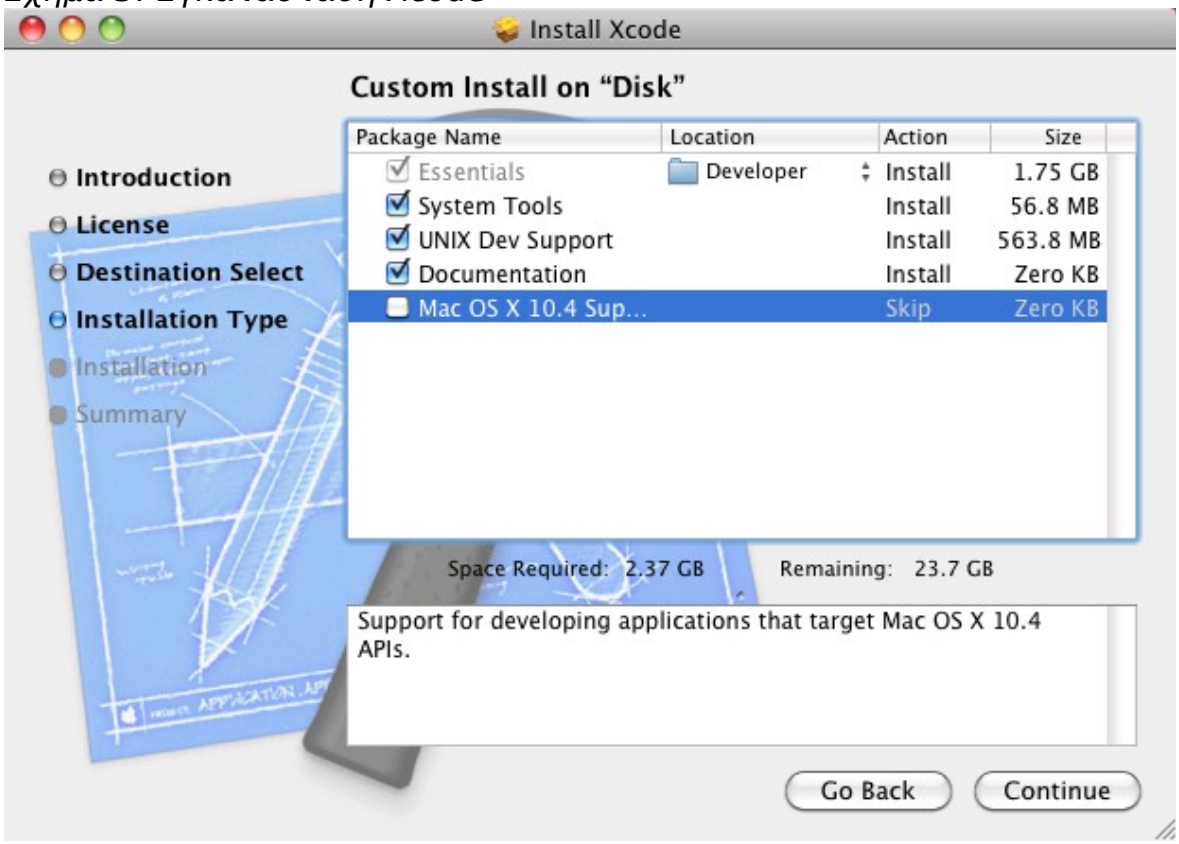
Εκτελούμε τον MinGW Shell που μόλις εγκαταστάθηκε στο σύστημα μας και εκτελούμε τον GNU C++ compiler στο hello_world.cxx κώδικα μας

```
$ g++ hello_world.cxx -o hello_world.exe
```

2.3 Πλατφόρμες Mac OS X

Στην πλατφόρμα Mac το περιβάλλον ανάπτυξης εμπεριέχεται στο πακέτο Xcode. Το πακέτο Xcode εμπεριέχει ένα πλήρες IDE με τους απαραίτητους μεταγλωττιστές και όλα τα απαραίτητα εργαλεία ανάπτυξης σε περιβάλλον Mac OS X. Ο οδηγός εγκατάστασης που είναι διαθέσιμος στο μέσο εγκατάστασης του λειτουργικού αναλαμβάνει την εγκατάσταση και παραμετροποίηση όλων των απαραίτητων στοιχείων.

Σχήμα 3: Εγκατάσταση Xcode



Αφού ολοκληρωθεί η εγκατάσταση μπορούμε να δούμε την έκδοση του C/C++ μεταγλωττιστή αν τον εκτελέσουμε σε ένα τερματικό συστήματος.

```
Sues-Mac:~ sue$ g++ -v

Using built-in specs.
```

```
Target: i686-apple-darwin10

Configured with: /var/tmp/gcc/gcc-5646~6/src/configure --disable-checking
--enable-werror    --prefix=/usr    --mandir=/usr/share/man    --enable-
languages=c,objc,c++,obj-c++ --program-transform-name=/^[cg][^.-]*$/s/$/-
4.2/   --with-slibdir=/usr/lib   --build=i686-apple-darwin10   --with-gxx-
include-dir=/usr/include/c++/4.2.1            --host=i686-apple-darwin10
--target=i686-apple-darwin10

Thread model: posix

gcc version 4.2.1 (Apple Inc. build 5646)
```

2.4 Δια-συστηματικοί Μεταγλωττιστές

Κάποιοι μεταγλωττιστές (cross compilers) είναι ικανοί να εξάγουν εκτελέσιμα αρχεία για διαφορετικές πλατφόρμες απ' τις οποίες εκτελούνται. Οι μεταγλωττιστές αυτοί χρησιμοποιούνται κυρίως για τη μεταγλώττιση εφαρμογών σε φορητά συστήματα (embedded) στα οποία δεν είναι δυνατή η εγκατάσταση λειτουργικού συστήματος και η μεταγλώττιση των εφαρμογών. Χρησιμοποιούνται όμως και στην ανάπτυξη μεταφέρσιμων εφαρμογών για την μεταγλώττιση τους για διαφορετικές πλατφόρμες.

Χαρακτηριστικό παράδειγμα αποτελεί ο μεταγλωττιστής MinGW ο οποίος μπορεί να εκτελεστεί σε μια πλατφόρμα GNU/Linux και να εξάγει εκτελέσιμα για πλατφόρμες Microsoft Windows. Για να επιτευχθεί αυτό πρέπει πρώτα να μεταγλωττιστεί με τον GNU C++ μεταγλωττιστή του GNU/Linux. Έπειτα αφού διαθέτουμε το w32 API των Windows, μπορούμε να χρησιμοποιήσουμε τον MinGW μεταγλωττιστή για να μεταγλωττίσουμε εφαρμογές C++ καθώς και τις απαραίτητες βιβλιοθήκες τους για Windows.

Κεφάλαιο 3: Μεθοδολογίες Ανάπτυξης Μεταφέρσιμου Κώδικα

3.1 Υποτυπώδης μεταφέρσιμος κώδικας C++

Θα μπορούσαμε να θεωρήσουμε δεδομένο ότι το απλό κομμάτι κώδικα όπως του παραρτήματος Α (Παράδειγμα ελάχιστου C++ προγράμματος Hello World), μπορεί να μεταγλωττιστεί και να εκτελεστεί αυτούσιο στις περισσότερες πλατφόρμες. Ακόμα όμως και σε αυτή την περίπτωση αν μελετήσουμε προσεκτικά την έξοδο του τερματικού στις διαφορετικές πλατφόρμες θα παρατηρήσουμε διαφορές.

Σε πλατφόρμα GNU/Linux:

```
argy@Freak:~$ ./helloworld  | wc -c
14
```

Σε πλατφόρμα Microsoft Windows:

```
$ helloworld.exe | wc -c
15
```

Σε πλατφόρμα Mac OS X:

```
Sues-Mac:~ sue$ ./helloworld | wc -c
14
```

Η διαφορά στο μέγεθος της εξόδου οφείλεται στον διαφορετικό τρόπο με τον οποίο κάθε πλατφόρμα ορίζει την αλλαγή γραμμής. Στις πλατφόρμες MS Windows η αλλαγή γραμμής ορίζεται με τους χαρακτήρες \r \n, ενώ στις πλατφόρμες Mac OS X και GNU / Linux ορίζεται με τον χαρακτήρα \n.

Σε πλατφόρμα GNU/Linux:

```
argy@Freak:~$ ./helloworld | od -c
```

```
0000000   H   e   l   l   o   ,       w   o   r   l   d   !  \n
0000016
```

Σε πλατφόρμα Microsoft Windows:

```
$ helloworld.exe | od -c
0000000   H   e   l   l   o   ,       w   o   r   l   d   !  \r  \n
0000017
```

Σε πλατφόρμα Mac OS X:

```
Sues-Mac:~ sue$ ./helloworld | od -c
0000000   H   e   l   l   o   ,       w   o   r   l   d   !  \n
0000016
```

Αν και το τελικό αποτέλεσμα στο τερματικό κάθε πλατφόρμας είναι το ίδιο (αλλαγή γραμμής) στην περίπτωση που θα θέλαμε να χρησιμοποιήσουμε αυτή την έξοδο σε ένα άλλο πρόγραμμα το οποίο θα χρειαζόταν να δεσμεύσει το απαιτούμενο χώρο στη μνήμη για να επεξεργαστεί την έξοδο, θα έπρεπε να δεσμεύσει διαφορετικό μέγεθος μνήμης ανάλογα με την πλατφόρμα απ' την οποία το αρχικό πρόγραμμα εκτελέστηκε. Αυτό πιθανότατα σε μια εξειδικευμένη περίπτωση θα μπορούσε να οδηγήσει σε σπατάλη χώρου μνήμης ή στην χειρότερη περίπτωση σε υπερχείλιση μνήμης (buffer overflow).

3.2 Εξειδίκευση κώδικα για διαφορετικές πλατφόρμες

Κατά τη συγγραφή μεταφέρσιμου κώδικα, απαιτείται πολύ συχνά να εξειδικεύσουμε μέρη του κώδικα τα οποία είναι διαφορετικά σε κάθε πλατφόρμα. Οι περισσότεροι μεταγλωττιστές ανάλογα με την πλατφόρμα στην οποία εκτελούνται θα ορίσουν κάποια macro η οποία θα χρησιμοποιηθεί απ' τον προ-μεταγλωττιστή (preproccessor) για να διακρίνει ποια μέρη του κώδικα πρέπει να μεταγλωττιστούν στην κάθε

πλατφόρμα.

Αν μεταγλωττίσουμε λοιπόν τον κώδικα του παραρτήματος Β (Παράδειγμα μεταγλώττισης κατά περίπτωση) θα πάρουμε την παρακάτω έξοδο για κάθε πλατφόρμα:

Σε πλατφόρμα GNU/Linux:

```
argy@Freak:~$ ./helloworld_conditional_compilation
Hello, world!
Compiled for POSIX!
```

Σε πλατφόρμα Microsoft Windows:

```
$ helloworld_conditional_compilation.exe
Hello, world!
Compiled for WIN!
```

Σε πλατφόρμα Mac OS X:

```
Sues-Mac:~ sue$ ./helloworld_conditional_compilation
Hello, world!
Compiled for MAC!
```

Επειδή όμως αυτές οι macro είναι άμεσα συνδεδεμένες με τον κατασκευαστή του μεταγλωττιστή θα ήταν πιο σωστό να ορίσουμε τις δικές μας macro για κάθε πλατφόρμα οι οποίες θα ορίζονται ως παράμετροι στον μεταγλωττιστή κατά την κλήση του κατά περίπτωση στην εκάστοτε πλατφόρμα.

3.3 Συγγραφή συντηρήσιμου κώδικα

Είναι σημαντικό να οργανώσουμε τον κώδικα μας με τέτοιο τρόπο ώστε να μην επηρεάζεται σημαντικά η ευκολία κατανόησης και συντήρησης

του κώδικα.

Όταν για παράδειγμα απαιτείται η κλήση μιας διεπαφής του λειτουργικού για την εκτέλεση μιας εργασίας ο κώδικας μας μπορεί να γίνει πολύ δύσκολα συντηρήσιμος μιας και οι διαφορές είναι σημαντικές στην κάθε μία πλατφόρμα.

Μπορούμε για παράδειγμα να μελετήσουμε τη διεπαφή που προσφέρει η κάθε πλατφόρμα για την εκτέλεση μιας απλής λειτουργίας όπως είναι η δημιουργία μιας νέας διεργασίας.

Για να δημιουργήσουμε μια νέα διεργασία στις πλατφόρμες WIN32 θα χρησιμοποιήσουμε τη διαδικασία CreateProcess(). Ο ορισμός της διεργασίας φαίνεται παρακάτω.

```
BOOL WINAPI CreateProcess(
  __in_opt     LPCTSTR lpApplicationName,
  __inout_opt  LPTSTR lpCommandLine,
  __in_opt     LPSECURITY_ATTRIBUTES lpProcessAttributes,
  __in_opt     LPSECURITY_ATTRIBUTES lpThreadAttributes,
  __in         BOOL bInheritHandles,
  __in         DWORD dwCreationFlags,
  __in_opt     LPVOID lpEnvironment,
  __in_opt     LPCTSTR lpCurrentDirectory,
  __in         LPSTARTUPINFO lpStartupInfo,
  __out        LPPROCESS_INFORMATION lpProcessInformation
);
```

Έτσι για να δημιουργήσουμε μία νέα διεργασία και να εκτελέσουμε την εφαρμογή σημειωματάριου των Windows θα πρέπει να γράψουμε ένα κομμάτι κώδικα όπως το παρακάτω:

```c
#include <windows.h>


int main(int argc, char *argv[])
{
	STARTUPINFO si;
	PROCESS_INFORMATION pi;


	ZeroMemory( &si, sizeof(si) );
	si.cb = sizeof(si);


	ZeroMemory( &pi, sizeof(pi) );
	// Start the child process.
	if(!CreateProcess(NULL, "notepad", NULL, NULL,FALSE, 0, NULL, NULL,
&si,&pi))
		return E_FAIL;
	else
		return S_OK;
}
```

Στις POSIX πλατφόρμες απ' την άλλη, θα πρέπει να χρησιμοποιήσουμε τη διαδικασία fork() και έπειτα exec().

```c
#include <unistd.h>


pid_t fork(void);
```

```c
#include <unistd.h>
```

```
extern char **environ;


int execl(const char *path, const char *arg, ...);
```

Για να δημιουργήσουμε έτσι μια νέα διεργασία και να εκτελέσουμε μια αντίστοιχη εφαρμογή σημειωματάριου στην πλατφόρμα GNU/Linux θα πρέπει να συγγράψουμε ένα κομμάτι κώδικα όπως το παρακάτω:

```
#include <sys/types.h>
#include <unistd.h>


int main(int argc, char *argv[])
{
    pid_t pid;


    pid = fork();
    if (pid ==  0 ) {
        execl("/usr/bin/gedit", "gedit", NULL);
    }
    return(0);
}
```

Είναι εμφανές ότι αν προσπαθήσουμε να συμπεριλάβουμε τους δύο διαφορετικούς τρόπους δημιουργίας μιας διεργασίας γι' αυτές τις δύο πλατφόρμες σε ένα κοινό κορμό κώδικα για την εφαρμογή μας, θα έχουμε ένα σημαντικά δυσανάγνωστο και δύσκολο προς συντήρηση κομμάτι κώδικα.

Ας προσπαθήσουμε να “κρύψουμε” αυτή τη διαφορετικότητα από το

κυρίως μέρος του προγράμματος μας δημιουργώντας μια κλάση CProcess η οποία θα αναλάβει να κάνει την κατάλληλη κλήση ανάλογα με την πλατφόρμα στην οποία βρισκόμαστε.

Μια πιθανή υλοποίηση της κλάσης CProcess θα ήταν η παρακάτω:

```
#include <iostream>


#if defined(WIN32)
#include <windows.h>
#else
#include <sys/types.h>
#include <unistd.h>
#endif
using namespace std;


class CProcess {
#if defined(WIN32)
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
#else
    pid_t pid;
#endif


   char* cmd_;


  public:
    CProcess (char* cmd);
    int run ();
```

```
};


CProcess::CProcess (char* cmd) {
  cmd_ = cmd;
#if defined(WIN32)
  ZeroMemory( &si, sizeof(si) );
  si.cb = sizeof(si);
  ZeroMemory( &pi, sizeof(pi) );
#else
  pid = fork();
#endif
}
int CProcess::run () {
#if defined(WIN32)
       if(!CreateProcess(NULL,  cmd_,  NULL,  NULL,FALSE,  0,  NULL,  NULL,
&si,&pi))
      return E_FAIL;
    else
      return S_OK;
#else
    if (pid ==  0 ) {
        execl(cmd_, "", NULL);
      return(0);
    }else{
      return(1);
    }
#endif
}
```

Χρησιμοποιώντας ένα αντικείμενο της παραπάνω κλάσης μπορούμε να δημιουργήσουμε μια νέα διεργασία στο κυρίως μέρος του προγράμματος μας χωρίς να απαιτείται να εξειδικεύσουμε για κάποια απ' τις παραπάνω πλατφόρμες.

```
int main (int argc, char *argv[]) {
  CProcess p1 (argv[1]);


  p1.run();


  return 0;
}
```

Αν και με αυτή την προσέγγιση κρύβουμε τις ιδιαιτερότητες της κάθε πλατφόρμας απ' το κυρίως πρόγραμμα η κλάση CProcess εμπεριέχει διαφορετικά μέλη για κάθε πλατφόρμα. Αν και στο συγκεκριμένο παράδειγμα αυτό μπορεί να μη δημιουργεί πρόβλημα, μιας και τα διαφορετικά μέλη είναι ιδιωτικά (private), σε πιο περίπλοκες περιπτώσεις θα αναγκαστούμε πάλι να κάνουμε διακρίσεις στο κυρίως μέρος του κώδικα για να χρησιμοποιήσουμε μια τέτοια κλάση.

3.4 Χρήση Factory Pattern

Είναι σημαντικό λοιπόν να σχεδιάσουμε μια μεταφέρσιμη διεπαφή η οποία θα είναι όμοια ανεξαρτήτως της πλατφόρμας. Στο παράδειγμα της έναρξης μιας νέας διεργασίας θα μπορούσαμε να ορίσουμε μια διεπαφή Process η οποία θα είναι κοινή για όλες τις πλατφόρμες.

```
class Process {
```

```
public:
	Process(char * cmd);
	virtual ~Process();
	int run();
private:
	ProcessImpl * m_processImpl;
	char * cmd_;
};
```

Η κλάση Process παρέχει τη δημόσια (public) διεπαφή χρήσης της ενώ παράλληλα φροντίζει να ενθυλακώνει μια κλάση με την υλοποίηση (ProcessImpl).

```
class ProcessImpl {
public:
	ProcessImpl();
	ProcessImpl(char * cmd);
	virtual ~ProcessImpl();
	virtual int runProcess();

protected:
	char * cmd_;
};
```

Χρησιμοποιώντας το Factory Pattern δημιουργούμε μια κλάση ProcessFactory η οποία θα μας παρέχει το αντίστοιχο αντικείμενο ProcessImpl για κάθε πλατφόρμα.

```
#include "ProcessImpl.h"
```

```
class ProcessFactory {
public:
      static ProcessFactory * GetProcessFactory();
      virtual ProcessImpl * createProcess(char * cmd);
};
```

Η static μέθοδος GetProcessFactory() επιστρέφει ένα αντικείμενο ProcessFactory εξειδικευμένο για κάθε πλατφόρμα. Στη συνέχεια το εκάστοτε αντικείμενο ProcessFactory θα μας παρέχει την αντίστοιχη υλοποίηση της ProcessImpl για την κάθε πλατφόρμα μέσω της μεθόδου createProcess().

Παρακάτω φαίνεται η υλοποίηση της κλάσης ProcessFactory που αναλαμβάνει την κλήση της αντίστοιχης για κάθε πλατφόρμα υλοποίησης.

```
#include "ProcessFactory.h"

class ProcessFactory;

#if defined(HAVE_WIN32)
#include "windows/WindowsFactory.h"
#endif
#if defined(HAVE_MACOS)
#include "cocoa/cocoafactory.h"
#endif
#if defined(HAVE_LINUX)
#include "linux/LinuxFactory.h"
#endif
```

```
ProcessFactory * ProcessFactory::GetProcessFactory()
{
	static ProcessFactory *processFactory = 0;

	if (!processFactory)
		#if defined(HAVE_WIN32)
		processFactory = WindowsFactory::GetFactoryInstance();
		#endif
		#if defined(HAVE_MACOS)
		processFactory = CocoaFactory::GetFactoryInstance();
		#endif
		#if defined(HAVE_LINUX)
		processFactory = LinuxFactory::GetFactoryInstance();
		#endif

	return processFactory;

}
ProcessImpl * ProcessFactory::createProcess(char * cmd) {
}
```

Η μέθοδος createProcess() στην κλάση ProcessFactory έχει οριστεί ως virtual ώστε να υλοποιηθεί απ' τις υπό-κλάσεις που θα την κληρονομήσουν ανάλογα με τις ιδιαιτερότητες τις κάθε πλατφόρμες και να επιστρέψει το αντίστοιχο αντικείμενο ProcessImpl.

Συνοψίζοντας τον μέχρι τώρα σχεδιασμό έχουμε μια κλάση Process η

οποία παρέχει την κοινή διεπαφή για όλες τις πλατφόρμες και η οποία ενθυλακώνει την υλοποίηση μέσα σε μια κλάση ProcessImpl. Η κλάση ProcessFactory είναι υπεύθυνη να παρέχει τα αντίστοιχα για κάθε πλατφόρμα αντικείμενα χωρίς να διαφοροποιείται η τελική διεπαφή.

Αυτό που μένει τώρα είναι να δημιουργήσουμε τις κλάσεις που θα κληρονομούν τα κοινά χαρακτηριστικά απ' την ProcessFactory και ProcessImpl και θα υπερφορτώνουν τα μέλη που είναι διαφορετικά σε κάθε πλατφόρμα.

Υλοποίηση υπό-κλάσεων για διαφορετικές Πλατφόρμες

Αυτό που μένει τώρα είναι να δημιουργήσουμε τις κλάσεις που θα κληρονομούν τα κοινά χαρακτηριστικά απ' την ProcessFactory και ProcessImpl και θα υπερφορτώνουν τα μέλη που είναι διαφορετικά σε κάθε πλατφόρμα.

Windows πλατφόρμες

```
#include "WindowsFactory.h"
#include "WindowsProcessImpl.h"

WindowsFactory * WindowsFactory::GetFactoryInstance() {
	static WindowsFactory *factory = 0;

	if (!factory)
		factory = new WindowsFactory;
	return factory;

}
```

```
ProcessImpl * WindowsFactory::createProcess(char * cmd) {
	return new WindowsProcessImpl(cmd);
}
```

```
#include "WindowsProcessImpl.h"

WindowsProcessImpl::WindowsProcessImpl() {
	// TODO Auto-generated constructor stub

}

WindowsProcessImpl::WindowsProcessImpl(char * cmd) : ProcessImpl(cmd) {
	ZeroMemory( &si, sizeof(si) );
	si.cb = sizeof(si);
	ZeroMemory( &pi, sizeof(pi) );
}

WindowsProcessImpl::~WindowsProcessImpl() {
	// TODO Auto-generated destructor stub
}

int WindowsProcessImpl::runProcess() {
	if(!CreateProcess(NULL,  cmd_,  NULL,  NULL,FALSE,  0,  NULL,  NULL,
&si,&pi))
		return E_FAIL;
    else
```

```
	return S_OK;
}
```

Linux πλατφόρμες

```
#include "LinuxFactory.h"
#include "LinuxProcessImpl.h"

LinuxFactory * LinuxFactory::GetFactoryInstance() {
	static LinuxFactory *factory = 0;

	if (!factory)
		factory = new LinuxFactory;
	return factory;

}

ProcessImpl * LinuxFactory::createProcess(char * cmd) {
	return new LinuxProcessImpl(cmd);
}
```

```
#include "LinuxProcessImpl.h"
#include <iostream>

LinuxProcessImpl::LinuxProcessImpl() {
	// TODO Auto-generated constructor stub

}
```

```
LinuxProcessImpl::LinuxProcessImpl(char * cmd) : ProcessImpl(cmd) {
	// TODO Auto-generated constructor stub
}

LinuxProcessImpl::~LinuxProcessImpl() {
	// TODO Auto-generated destructor stub
}

int LinuxProcessImpl::runProcess() {
	pid = fork();
    if (pid !=  0 ) {
	execl(cmd_, "", NULL);
	std::cout << "CMD: " << cmd_ << std::endl;
	return(0);
    }
    else {
	return(1);
    }
}
```

Κεφάλαιο 4: Μεταφέρσιμες Βιβλιοθήκες και Εργαλειοθήκες

Όσο αυξάνεται το μέγεθος της εφαρμογής και το πλήθος των πλατφορμών στο οποίο πρέπει να μεταφερθεί, τόσο αυξάνεται η ανάγκη απόκρυψης των ιδιαιτεροτήτων της κάθε πλατφόρμας. Υπάρχει μια πλειάδα από βιβλιοθήκες και ολοκληρωμένα πλαίσια ανάπτυξης μεταφέρσιμου λογισμικού για C++.

4.1 Boost C++ Libraries

Οι Boost C++ βιβλιοθήκες, αποτελούν μια συλλογή μοντέρνων βιβλιοθηκών βασισμένων στο C++ standard. Ο πηγαίος κώδικας διατίθεται υπό τους όρους της άδειας χρήσης Boost Software License, η οποία επιτρέπει την ελεύθερη χρήση, μετατροπή και αναδιανομή των βιβλιοθηκών.

Η κοινότητα των Boost βιβλιοθηκών συστάθηκε γύρω στο 1998 όταν η πρώτη έκδοση τυποποίησης της C++ είχε εκδοθεί. Από τότε η κοινότητα έχει αναπτυχθεί και επηρεάζει σημαντικά την διαδικασία τυποποίησης της C++. Πολλές απ' τις Boost βιβλιοθήκες έχουν προταθεί να ενσωματωθούν στις επόμενες εκδόσεις της C++ τυποποίησης.

Βασικό χαρακτηριστικό των Boost C++ βιβλιοθηκών αποτελεί η φορητότητα των βιβλιοθηκών. Παρέχουν δηλαδή μια αφαιρετική διεπαφή για χρήση των βιβλιοθηκών σε διαφορετικά περιβάλλοντα και με διαφορετικούς μεταγλωττιστές.

Αρχιτεκτονική

Μια σειρά από φορητές βιβλιοθήκες παρέχονται για τις παρακάτω

κατηγορίες:

- Αλφαριθμητικά και επεξεργασία κειμένου
- Containers
- Iterators
- Αλγόριθμοι
- Function Objects and Higher-order Programming
- Generic Programming
- Template Meta-programming
- Preprocessor Programming
- Math and Numerics
- Correctness and Testing
- Data Structures
- Image Processing
- Input / Output
- Inter-Language Support
- Memory
- Parsing

- Programming Interfaces

Μεταφερσιμότητα

Υποστηριζόμενες πλατφόρμες:

- Microsoft Windows
- Linux
- Mac OS X

Εργαλεία Ανάπτυξης

Παράλληλα με την ανάπτυξη της βιβλιοθήκης Boost, δημιουργήθηκαν και μια σειρά από βοηθητικά εργαλεία που είτε εξυπηρετούν τις ανάγκες των συγγραφέων της Boost είτε τους χρήστες της. Όπως οι βιβλιοθήκες έτσι και αυτά τα εργαλεία προσφέρονται δωρεάν μαζί με τον πηγαίο τους κώδικα.

Το **Boot.Build** είναι ένα σύστημα μεταγλώττισης (build system) C++ προγραμμάτων. Το Boost.Build αναλαμβάνει τη μεταγλώττιση του πηγαίου κώδικα με τους κατάλληλους διακόπτες για τον κάθε μεταγλωττιστή και την κάθε πλατφόρμα.

Το **Regression** αποτελεί ένα εργαλείο υλοποίησης και εκτέλεσης βασικών σεναρίων ελέγχου ποιότητας του λογισμικού.

Το **Inspect** είναι ένα εργαλείο που ανιχνεύει τα συνήθη λάθη και παραβιάσεις των οδηγιών. Μπορεί να χρησιμοποιηθεί για τον εντοπισμό σφαλμάτων σε νέες βιβλιοθήκες πριν εισαχθούν.

Το **BoostBook** είναι ένα εργαλείο τεκμηρίωσης βασισμένο στο γνωστό DocBook και στη γλώσσα XSL. Πολλές Boost βιβλιοθήκες χρησιμοποιούν το BoostBook για την τεκμηρίωση τους.

Το **bcp** αναγνωρίζει τα επιμέρους πακέτα της Boost που χρησιμοποιεί η

εκάστοτε εφαρμογή και εξάγει αναφορές χρήσης της Boost καθώς και τις σχετικές άδειες χρήσης.

Το **QuickBook** είναι ένα παρόμοιο με το Wiki-wiki στυλ σύστημα συγγραφής τεκμηρίωσης κώδικα. Το QuickBook χρησιμοποιεί μια σειρά απλούς κανόνες μορφοποίησης κειμένου και εξάγει XML κείμενα τύπου BoostBook.

Ο **Wave** είναι ένας C/C++ προ-μεταγλωττιστής πλήρως συμμορφωμένος με τα στάνταρ της γλώσσας, ικανός να χρησιμοποιηθεί πάνω από κάθε άλλο μεταγλωττιστή. Χρήσιμος για την αποσφαλμάτωση των macro του κώδικα ή ως αντικαταστάτης του ενσωματωμένου προ-μεταγλωττιστή.

Παράδειγμα Χρήσης

Μέχρι πολύ πρόσφατα η C++ δεν παρείχε κάποιες προδιαγραφές ή άλλη υποστήριξη για πολύ-νηματικό προγραμματισμό, κάνοντας τη χρήση ειδικών βιβλιοθηκών απαραίτητη. Έτσι ο κώδικα διαχείρισης νημάτων είναι πολύ διαφορετικός σε κάθε πλατφόρμα. Οι βιβλιοθήκες Boost παρέχουν ένα πακέτο για διαχείριση νημάτων, προσφέροντας κοινή διεπαφή στον προγραμματιστή και κάνοντας το τελικό λογισμικό μεταφέρσιμο.

Στο παράρτημα Z (Παράδειγμα δημιουργίας νημάτων με τη Boost C++) παρουσιάζεται η δημιουργία δύο νημάτων με τη βοήθεια της BoostThread.

Το συγκεκριμένο κομμάτι κώδικα, μπορεί πλέον να μεταγλωττιστεί αυτούσιο σε περιβάλλον GNU/Linux με τον GNU C/C++

```
g++ -I/opt/local/include -L/opt/local/lib -lboost_thread -o simpleThread
simpleThread.cpp
```

ή στα MS Windows με τον μεταγλωττιστή του Visual C++

```
cl /Ehsc /Zl /I “C:\Program Files\boost\boost_1_47” simpleThread.cpp
/link /LIBPATH:”C:\Program Files\boost\boost_1_47\lib”
```

και να εκτελεστεί και στις δύο πλατφόρμες χωρίς διαφορές στην ροή εκτέλεσης.

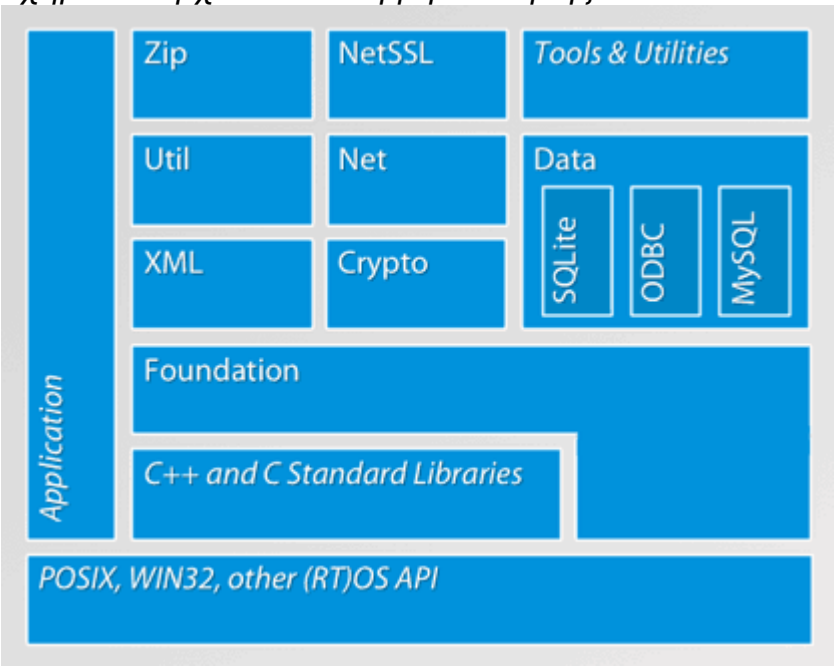
4.2 POCO C++ Libraries

Η συλλογή POCO παρέχει μια σειρά από μεταφέρσιμες βιβλιοθήκες με χαρακτηριστικά παρόμοια με την βιβλιοθήκη κλάσεων της Java, του .NET Framework ή του Cocoa της Apple. Η Poco δίνει ιδιαίτερη έμφαση στην ευκολία ανάπτυξης δικτυακών εφαρμογών σε C++, για διαφορετικές πλατφόρμες.

Αρχιτεκτονική

Οι POCO C++ βιβλιοθήκες διακρίνονται σε τέσσερις υπό-ομάδες βιβλιοθηκών οι οποίες συμπληρώνονται από κάποιες πρόσθετες προαιρετικές (add-on).

Σχήμα 4: Αρχιτεκτονική βιβλιοθήκης POCO C++



Foundation

Στην Foundation παρέχονται μια σειρά από βασικές μεταφέρσιμες βιβλιοθήκες που αποκρύπτουν τις διαφορές μεταξύ των πλατφορμών και παρέχουν κοινή προγραμματιστική διεπαφή. Στην Foundation εμπεριέχονται βιβλιοθήκες για τη διαχείριση της μνήμης, τον χειρισμό αλφαριθμητικών, τη διαχείριση ροών (streams) και νημάτων (threads), ημερομηνίας και ώρας, συστήματος αρχείων και διεργασιών.

Net

Οι βιβλιοθήκες Net είναι αφιερωμένες στις διαδικτυακές επικοινωνίες με υποστήριξη για sockets, μηνύματα MIME, επικοινωνία μέσω HTTP, FTP και Mail.

XML

Ολοκληρωμένη υποστήριξη XML με τη διεπαφή SAX2 καθώς και μέσω του DOM.

Util

Η Util περιέχει βιβλιοθήκες για τον χειρισμό αρχείων ρυθμίσεων, παραμέτρων εκτέλεσης και ένα πλαίσιο ανάπτυξης εφαρμογών διακομιστή.

Η πλήρης έκδοση παρέχει κάποια επιπλέον πακέτα όπως τα NetSSL, Crypto, Zip και Data, τα οποία όμως εξαρτώνται από κάποιες άλλες φορητές βιβλιοθήκες OpenSSL, ODBC και MySQL.

Μεταφερσιμότητα

Υποστηριζόμενες πλατφόρμες:

- Microsoft Windows

- Linux
- Mac OS X
- HP-UX, Solaris, AIX*
- Embedded Linux (uClibc, glibc)
- iOS
- Windows Embedded CE
- QNX

Παράδειγμα Χρήσης

Η POCO C++ δίνει ιδιαίτερη έμφαση στην ευκολία ανάπτυξης μεταφέρσιμων δικτυακών εφαρμογών. Το κομμάτι κώδικα του παραρτήματος Η (Χρήση sockets με την POCO C++), υλοποιεί το μοντέλο διακομιστή – πελάτη με τη χρήση sockets. Ο ίδιος ακριβώς πηγαίος κώδικας μπορεί να μεταγλωττιστεί σε GNU/Linux

```
g++ -o socket -lPocoNet socket.cpp
```

και σε MS Windows

```
cl /Ehsc /I “C:\poco-1.4.3p1\Net\include” /I “C:\poco-1.4.3p1\Foundation\include” socket.cpp /link /LIBPATH:”C:\poco-1.4.3p1\lib”
```

και να εκτελεστεί και στις δύο πλατφόρμες με το ίδιο ακριβώς αποτέλεσμα.

4.3 FLTK

Η FLTK είναι μια C++ βιβλιοθήκη για τη δημιουργία μοντέρνων GUI για διαφορετικές πλατφόρμες. Αυτή τη στιγμή η βιβλιοθήκη υποστηρίζει τις πλατφόρμες UNIX/ Linux (X11), Microsoft Windows και Mac OS X. Τα βασικά χαρακτηριστικά της βιβλιοθήκης είναι το μικρό μέγεθος, η

modular σχεδίαση και η υποστήριξη 3D γραφικών μέσω OpenGL. Αυτά τα χαρακτηριστικά κάνουν τη βιβλιοθήκη ιδανική για χρήση σε πλατφόρμες χαμηλών επιδόσεων, ενώ παράλληλα είναι δυνατή η ενσωμάτωση της στο τελικό εκτελέσιμο (statically linked) χωρίς να αυξάνει ιδιαίτερα το μέγεθος του τελικού εκτελέσιμου. Η βιβλιοθήκη διατίθεται υπό τους όρους της LGPL2 άδειας χρήσης με μια εξαίρεση που επιτρέπει της ενσωμάτωση της βιβλιοθήκης στο τελικό εκτελέσιμο, επιτρέποντας τη χρήση της σε ελεύθερες αλλά και εμπορικές εφαρμογές.

Αρχιτεκτονική

Σε αντίθεση με άλλες αντίστοιχες βιβλιοθήκες η FLTK χρησιμοποιεί ένα ανάλαφρο και απλό σχεδιασμό, ενώ περιορίζεται μόνο στην υλοποίηση της γραφικής διεπαφής χρήσης (GUI). Γι' αυτό το λόγο η βιβλιοθήκη είναι πολύ μικρή και μπορεί εύκολα να ενσωματωθεί στο τελικό εκτελέσιμο. Αν και είναι σχεδιασμένη για χρήση σε εφαρμογές C++ ο απλός σχεδιασμός της δεν κάνει χρήση κάποιον προχωρημένων χαρακτηριστικών της C++ όπως τα templates, exceptions ή RTTI, ενώ το API της βιβλιοθήκης δεν είναι ιδιαίτερα αντικειμενοστραφές (εκτός απ' την έκδοση FLTK2).

Ο απλός σχεδιασμός και η προσπάθεια μείωσης του μεγέθους της βιβλιοθήκης έχει σαν αποτέλεσμα μια σειρά μειονεκτημάτων. Τα παρεχόμενα αντικείμενα (widgets) γραφικού περιβάλλοντος είναι περιορισμένα, ενώ η μη χρήση των αντικειμένων (widgets) γραφικού περιβάλλοντος του συστήματος, έχεις σαν αποτέλεσμα το τελικό γραφικό περιβάλλον να μην εναρμονίζεται με το εκάστοτε της κάθε πλατφόρμας.

Οι υποστηριζόμενες εκδόσεις τις βιβλιοθήκης έχουν μια σειρά

διαφορετικών χαρακτηριστικών, τα οποία καθιστούν την κάθε έκδοση ασύμβατη με τις υπόλοιπες εκδόσεις.

Η έκδοση 1.1 (FLTK 1.1) θεωρείται σταθερή (stable) και υπό συντήρηση. Η συγκεκριμένη έκδοση παρέχει ένα αρκετά παλιό σύνολο αντικειμένων γραφικού περιβάλλοντος, τα οποία απέχουν πολύ απ' τα σύγχρονα γραφικά περιβάλλοντα ενώ έχει σημαντικές ελλείψεις και σε χαρακτηριστικά, όπως η υποστήριξη Unicode και αντικειμενοστραφούς API.

Η έκδοση 2.0 (FLTK 2.0) προσφέρει ένα νέο και τελείως διαφορετικό αντικειμενοστραφές API καθώς και υποστήριξη Unicode. Η ανάπτυξη της έχει σχεδόν σταματήσει και απ' ότι φαίνεται δε θα γίνει ποτέ stable. Αποτελεί όμως ήδη επιλογή πολλών προγραμματιστών λόγω του απλού και ξεκάθαρου προγραμματιστικού στυλ που προσφέρει.

Η έκδοση 1.3 (FLTK 1.3) βρίσκεται σε συνεχή ανάπτυξη αυτή την περίοδο. Προέρχεται απ' την 1.1 και στόχος είναι να αντιμετωπίσει τις αδυναμίες της. Με υποστήριξη Unicode, περισσότερα και ανανεωμένα αντικείμενα γραφικού περιβάλλοντός (widgets) είναι η έκδοση στην οποία γίνεται η περισσότερη ανάπτυξη αυτής της βιβλιοθήκης αυτή τη στιγμή. Διατηρεί το API της 1.1, χωρίς όμως να είναι πλήρως συμβατή μιας και η ένταξη νέων χαρακτηριστικών το διαφοροποιεί σε αρκετά σημεία. Αυτή θα είναι πιθανότατα η επόμενη σταθερή (stable) έκδοση της βιβλιοθήκης.

Μεταφερσιμότητα

Υποστηριζόμενες πλατφόρμες:

- Microsoft Windows
- Linux

- Mac OS X

Εργαλεία Ανάπτυξης

Η βιβλιοθήκη επίσης παρέχει έναν σχεδιαστή γραφικού περιβάλλοντος FLUID (GUI Builder) που εξάγει τον κώδικα C++ που υλοποιεί το γραφικό περιβάλλον που έχει σχεδιαστεί. Τα χαρακτηριστικά του είναι περιορισμένα, αλλά παρέχει τη δυνατότητα γρήγορης σχεδίασης του γραφικού περιβάλλοντος της εφαρμογής με προεπισκόπιση. Οι δυνατότητες που προσφέρει για την υλοποίηση της λειτουργικότητας του γραφικού περιβάλλοντος είναι περιορισμένες έως μηδαμινές και πρέπει να υλοποιηθούν μέσω τρίτου IDE.

Παράδειγμα Χρήσης

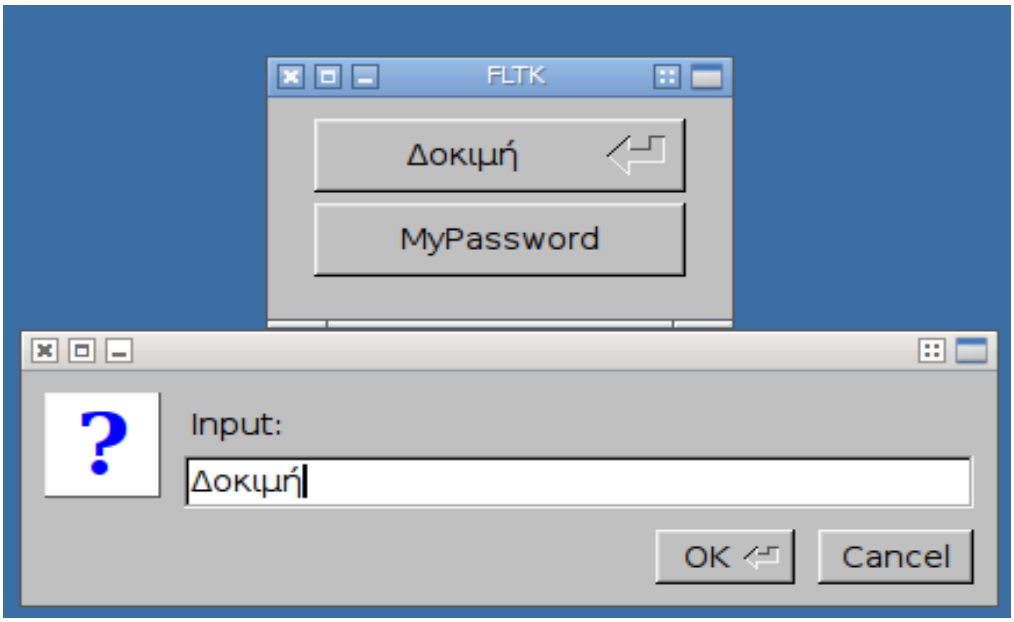
Η FLTK διαθέτει μια σειρά από αυτόνομα widgets που παρέχουν πανομοιότυπη εμπειρία χρήστη στις υποστηριζόμενες πλατφόρμες. Μπορούμε έτσι να μεταγλωττίσουμε το κομμάτι κώδικα του παραρτήματος Θ (Παράδειγμα χρήσης της FLTK) σε περιβάλλον GNU/Linux

```
fltk-config --compile ask.cxx
```

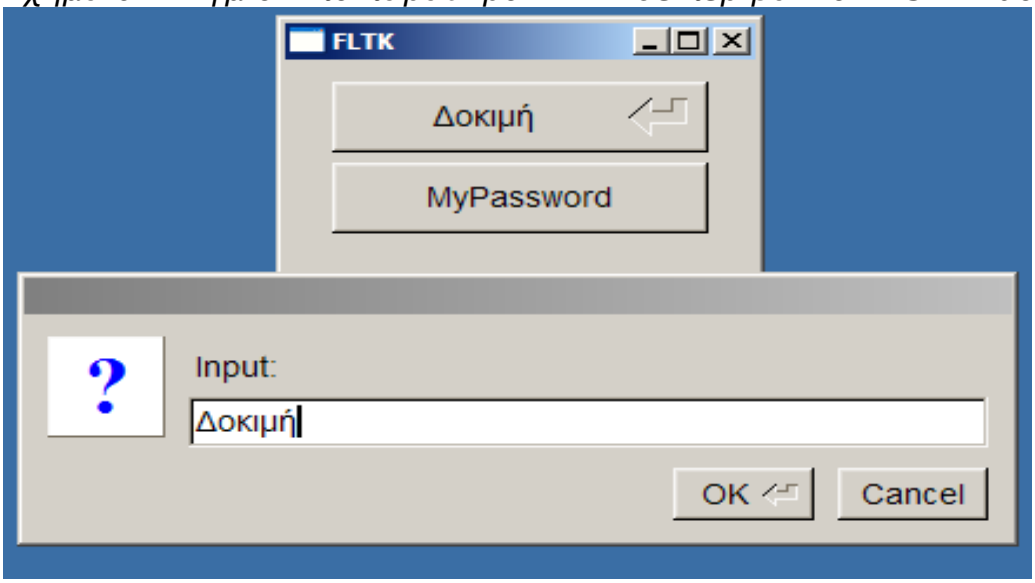
καθώς και με τη βοήθεια του Microsoft Visual Studio σε Microsoft Windows.

```
cl /Ob2 /Os /D "_CRT_SECURE_NO_DEPRECATE" /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D "WIN32_LEAN_AND_MEAN" /D "VC_EXTRA_LEAN" /D "WIN32_EXTRA_LEAN" /I c:\fltk-1.3.0\zlib /I c:\fltk-1.3.0\png /I c:\fltk-1.3.0\jpeg /I c:\fltk-1.3.0 ask.cxx /link /LIBPATH:c:\fltk-1.3.0\lib
```

Σχήμα 5: Στιγμιότυπο παράθυρου FLTK σε περιβάλλον GNU/Linux



Σχήμα 6: Στιγμιότυπο παράθυρου FLTK σε περιβάλλον MS Windows



Το οπτικό αποτέλεσμα και στις δύο πλατφόρμες είναι πανομοιότυπο.

4.4 GTK+

Η GTK+ είναι μια φορητή εργαλειοθήκη για τη δημιουργία του γραφικού περιβάλλοντος εφαρμογών. Είναι γραμμένη σε C, αλλά παρέχει μια πλειάδα από διεπαφές για διάφορες γλώσσες προγραμματισμού, καθώς και για τη C++. Η GTK+ είναι απ' τις πιο δημοφιλείς βιβλιοθήκες GUI για την πλατφόρμα GNU / Linux, παράλληλα όμως υποστηρίζει και άλλες πλατφόρμες όπως Microsoft Windows και Mac OS X. Η εργαλειοθήκη επιτρέπει τη δημιουργία γραφικών διεπαφών με εμφάνιση όμοια της εκάστοτε πλατφόρμας με τη χρήση διαφορετικών μηχανών απεικόνισης.

Αρχιτεκτονική

Η εργαλειοθήκη βασίζεται σε τέσσερις ανεξάρτητες φορητές βιβλιοθήκες οι οποίες έχουν αναπτυχθεί απ' την ίδια ομάδα.

Η **GLib** είναι μια χαμηλού επιπέδου βιβλιοθήκη η οποία παρέχει μια σειρά από δομές δεδομένων και χαρακτηριστικών αντικειμενοστρέφειας για C++. Η βιβλιοθήκη επίσης παρέχει τις απαραίτητες φορητές διεπαφές για την υλοποίηση των διεργασιών εκτέλεσης που απαιτούνται απ' το γραφικό περιβάλλον, όπως ο χειρισμός γεγονότων, νήματα και δυναμική φόρτωση αντικειμένων.

Η **Pango** βιβλιοθήκη παρέχει χαρακτηριστικά διάταξης και σχεδιασμού κειμένου με έμφαση στην διεθνοποίηση (internationalization). Χρησιμοποιείται απ' την GTK+ για τον χειρισμό του κειμένου και των γραμματοσειρών.

Η **Cairo** είναι μια 2D βιβλιοθήκη γραφικών που υποστηρίζει διαφορετικές συσκευές εξόδου, με στόχο την ομοιόμορφη απεικόνιση σε όλες τις συσκευές. Η βιβλιοθήκη επίσης εκμεταλλεύεται τα υπάρχοντα

χαρακτηριστικά επιτάχυνσης γραφικών που παρέχει το εκάστοτε υλικό.

Η **ATK (Accessibility Toolkit)** βιβλιοθήκη επιτρέπει στους προγραμματιστές να χρησιμοποιήσουν κοινά χαρακτηριστικά προσβασιμότητας. Όταν μια εφαρμογή υποστηρίζει τις διεπαφές της ATK, μπορεί να χρησιμοποιηθεί με μια σειρά εργαλείων προσβασιμότητας όπως οι αναγνώστες οθόνης (screen readers), μεγεθυντικοί φακοί, υψηλή αντίθεση χρωμάτων και εναλλακτικές συσκευές εισόδου.

Για την εκμετάλλευση των χαρακτηριστικών της C++ παρέχεται η διεπαφή GTKmm για την εργαλειοθήκη GTK++. Η διεπαφή GTKmm επιτρέπει τη δημιουργία γραφικού περιβάλλοντος (GUI) με κώδικα καθώς και με τον σχεδιαστή γραφικών διεπαφών Glade (XML). Επίσης παρέχει μια σειρά χαρακτηριστικών όπως typesafe callbacks, μια σειρά από γραφικά αντικείμενα (widgets) και επεκτασιμότητα αυτών μέσω κληρονομικότητας.

Μεταφερσιμότητα

Η GTK+ είναι πολύ δημοφιλής βιβλιοθήκη και με πολύ καλή υποστήριξη στις πλατφόρμες στις οποίες είναι διαθέσιμη. Χρησιμοποιείται σε μεγάλο βαθμό για τη δημιουργία μεταφέρσιμων εφαρμογών για Microsoft Windows, GNU/Linux και Mac OS X.

Εξαιτίας της πρωτοβουλίας GMAE (Gnome Mobile & Embedded) η GTK+ έχει υλοποιήσει αρκετά χαρακτηριστικά που απαιτούνται σε έξυπνα κινητά τηλέφωνα και άλλες φορητές συσκευές. Έχει ήδη χρησιμοποιηθεί με επιτυχία σε έξυπνα τηλέφωνα, φορητές συσκευές και το One Laptop Per Child Project.

Εργαλεία Ανάπτυξης

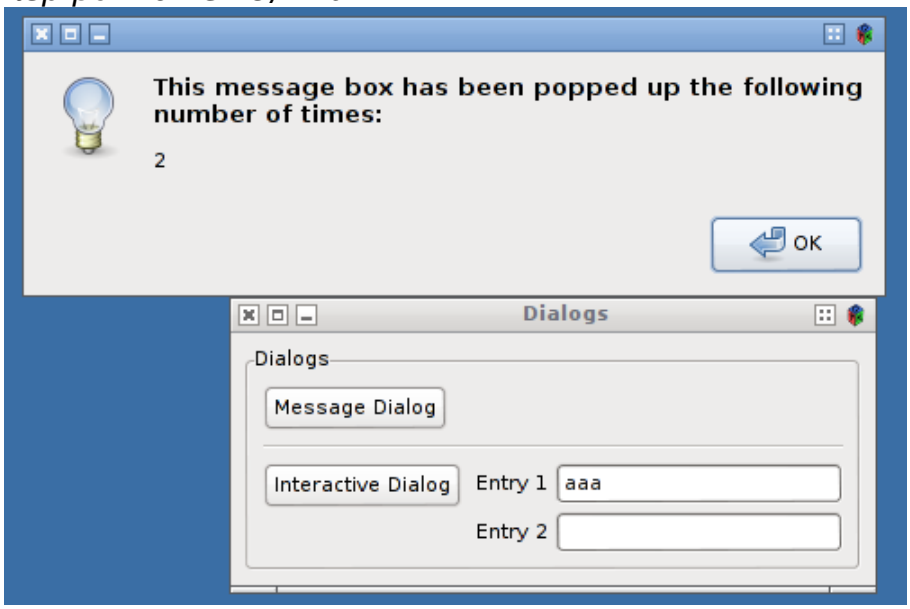
Η GTK+ δεν ενσωματώνει κάποια συγκεκριμένα εργαλεία ανάπτυξης, παρόλα αυτά μια σειρά από εργαλεία για την GTK+ έχουν αναπτυχθεί από τρίτους. Ένα από τα βασικότερα ίσως είναι το Glade.

Το Glade είναι ένα εργαλείο γρήγορης ανάπτυξης εφαρμογών (RAD), με βασικό χαρακτηριστικό την γρήγορη σχεδίαση και υλοποίηση διεπαφών χρήστη (GUI) για την GTK+ βιβλιοθήκη. Οι διεπαφές που σχεδιάζονται στο Glade αποθηκεύονται ως XML αρχεία. Με τη χρήστη του αντικειμένου GtkBuilder της GTK+ μπορεί να φορτωθεί δυναμικά από μια εφαρμογή όταν απαιτείται.

Παράδειγμα Χρήσης

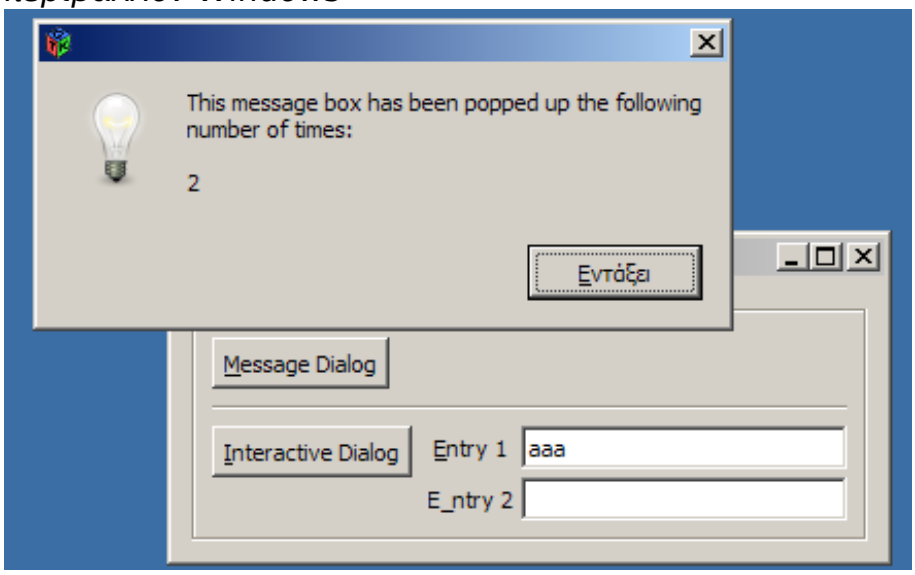
Η GTK+ επιτρέπει λοιπόν τη δημιουργία γραφικών διεπαφών σε διαφορετικές πλατφόρμες χωρίς την παραμικρή διαφοροποίηση στον πηγαίο κώδικα, παράρτημα Ι (Παράδειγμα χρήσης της GTK+).

Σχήμα 7: Στιγμιότυπο παραθύρων διαλόγου GTK+ σε περιβάλλον GNU/Linux



Η υλοποίηση των γραφικών διεπαφών ακολουθεί τη μορφή της εκάστοτε πλατφόρμας, όπως φαίνεται και στα στιγμιότυπα.

Σχήμα 8: Στιγμιότυπο παραθύρων διαλόγου GTK+ σε περιβάλλον Windows



4.5 Juce

Το Juce είναι ένα πλαίσιο ανάπτυξης μεταφέρσιμων εφαρμογών και πρόσθετων εφαρμογών σε C++ για διαφορετικές πλατφόρμες. Στόχος του πλαισίου είναι η ανάπτυξη εφαρμογών οι οποίες χωρίς καμία αλλαγή στον πηγαίο κώδικα δύναται να μεταγλωττίζονται, εκτελούνται και να φαίνονται όμοια σε κάθε πλατφόρμα. Το Juce εκδόθηκε το 2004 από την Raw Material Software υπό τους όρους δύο διαφορετικών αδειών χρήσης, της GPL και μιας εμπορικής.

Αρχιτεκτονική

Το Juce περιέχει μια πλειάδα από κλάσεις που κρύβουν όλες τις λεπτομέρειες της εκάστοτε πλατφόρμας και προσφέρουν μια

ενοποιημένη διεπαφή στον προγραμματιστή.

Οι **Core κλάσεις** παρέχουν μια σειρά από μεταφέρσιμες υλοποιήσεις για τις βασικές λειτουργίες κάθε εφαρμογής.

Παρέχει κλάσεις με υλοποιήσεις για:

- δομές δεδομένων, συλλογές αντικειμένων και χειρισμού Unicode αλφαριθμητικών (UTF-8, UTF-16 και UTF-32)
- ροές δεδομένων αρχείων, φακέλων και εισόδου εξόδου
- πολύ-νηματικού προγραμματισμού και επικοινωνίας μεταξύ διεργασιών
- μετατροπές μεταξύ τύπων δεδομένων

Οι Core κλάσεις μπορούν να χρησιμοποιηθούν ακόμη και στην ανάπτυξη εφαρμογών τερματικού αφού δεν έχουν καμία εξάρτηση γραφικού περιβάλλοντος.

Μεγάλο μέρος του πλαισίου αποτελούν οι **κλάσεις Γραφικής Διεπαφής**. Όλα τα γραφικά στοιχεία (widgets) που απαιτούν οι συνήθεις γραφικές διεπαφές παρέχονται από αντίστοιχες κλάσεις, παρέχοντας όμοια απεικόνιση σε όλες τις πλατφόρμες. Η υποστήριξη για OpenGL επιτρέπει την εύκολη ενσωμάτωση τρισδιάστατων γραφικών σε κάθε εφαρμογή Juce.

Οι **κλάσεις επεξεργασίας γραφικών**, παρέχουν υποστήριξη για 24-bit RGB και 32-bit ARGB γραφικά και επεξεργασία υψηλής ακρίβειας με εξομάλυνση γωνιών. Μια σειρά από κλάσεις γραφικών επιτρέπουν την ανάγνωση και εγγραφή γραφικών στα πιο κοινά φορμάτ (JPEG, PNG και GIF). Οι κλάσεις γραφικών επιτρέπουν ένα μεταφέρσιμο τρόπο

ενσωμάτωσης των γραφικών στην εφαρμογή (resources).

Οι **κλάσεις Ήχου** παρέχουν μια κοινή διεπαφή για αναπαραγωγή, εγγραφή και επεξεργασία του ήχου στα διαφορετικά συστήματα ήχου της κάθε πλατφόρμας. Επίσης παρέχουν υποστήριξη ανάγνωσης και εγγραφής αρχείων ήχου WAV, AIFF, Flac και Ogg-Vorbis.

Οι **κλάσεις διαχείρισης XML δεδομένων** παρέχουν δυνατότητες επεξεργασίας, ανάγνωσης και εγγραφής αρχείων XML.

Οι **κλάσεις Δικτύωσης** παρέχουν δυνατότητες χειρισμού εισερχόμενων ροών HTTP και FTP, χειρισμό URL διευθύνσεων, μεταφέρσιμες υλοποιήσεις χρήσης sockets και επικοινωνίας μεταξύ διεργασιών τοπικά και απομακρυσμένα.

Το Juce παρέχει επίσης κάποιες **κλάσεις Κρυπτογράφησης**, όπως γεννήτρια πρώτων αριθμών, κλειδιών RSA, κωδικοποίησης BlowFish και MD5.

Τέλος μια σειρά από κλάσεις γενικής χρήσης προσφέρουν επιπλέον δυνατότητες στο πλαίσιο, όπως η συμπίεση και αποσυμπίεση αρχείων Zip, δημιουργία GUID, πρόσβαση στο μητρώο των Windows, μετατροπές σε NSString του Mac και υψηλής ακρίβειας μετρητές απόδοσης εφαρμογής.

Μεταφερσιμότητα

Το Juce είναι διαθέσιμο στις παρακάτω πλατφόρμες:

- Mac OS X – Εφαρμογές και VST/ AudioUnit/RTAS/NPAPI πρόσθετα μπορούν να μεταγλωττιστούν με το Xcode για OSX 10.4 και νεότερα

- iOS – Εφαρμογές για iPhone και iPad μπορούν να μεταγλωττιστούν με τον Xcode.

- Windows - Εφαρμογές και VST/RTAS/NPAPI/ActiveX πρόσθετα μπορούν να μεταγλωττιστούν με τον MS Visual Studio για Windows XP, Vista και 7.

- Linux – Εφαρμογές και πρόσθετα μπορούν να μεταγλωττιστούν για εκδόσεις του kernel 2.6 και νεότερες

- Android – Εφαρμογές για τις Android πλατφόρμες μπορούν να μεταγλωττιστούν με το Ant και το Eclipse χρησιμοποιώντας το πακέτο ανάπτυξης Android NDK v5 και νεότερο (υπό ανάπτυξη)

Εργαλεία Ανάπτυξης

Το πλαίσιο ανάπτυξης Juce παρέχει δύο χρήσιμα εργαλεία για την οργάνωση και μεταγλώττιση των εφαρμογών σε διαφορετικές πλατφόρμες και για τον σχεδιασμό των γραφικών διεπαφών.

Το **Introjucer** είναι εργαλείο διαχείρισης εφαρμογών Juce. Στόχος του είναι η οργάνωση του πηγαίου κώδικα και των πρόσθετων αρχείων (resources) της εφαρμογής καθώς και η δημιουργία των απαραίτητων αρχείων για την εύκολη μεταγλώττιση της εφαρμογής στο περιβάλλον ανάπτυξης της κάθε πλατφόρμας. Το Introjucer δεν έχει δυνατότητες μεταγλώττισης της εφαρμογής, αλλά δημιουργεί τα απαραίτητα αρχεία ρυθμίσεων ώστε να μπορεί η τελική εφαρμογή να μεταγλωττιστεί στις διαφορετικές πλατφόρμες. Επίσης επιτρέπει την εύκολη ενημέρωση του πλαισίου ανάπτυξης Juce στην τελευταία έκδοση μέσω διαδικτύου.

Το **Jucer** είναι εργαλείο σχεδίασης γραφικών διεπαφών με τη λογική WYSISYG. Επιτρέπει στον προγραμματιστή να δημιουργήσει τις

διεπαφές της εφαρμογής χρησιμοποιώντας οπτικά μόνο εργαλεία, ενώ ο Jucer εξάγει τον C++ κώδικα της διεπαφής ο οποίος μπορεί εύκολα να ενσωματωθεί στην τελική εφαρμογή. Επίσης το Jucer επιτρέπει τη δημιουργία και τροποποίηση γραφικών στοιχείων (components) τα οποία μπορούν να χρησιμοποιηθούν ξανά μέσα στον Jucer μέσα σε άλλα γραφικά στοιχεία (components). Πέρα απ' τον κώδικα C++ των γραφικών διεπαφών το Jucer εξάγει και τα απαραίτητα μετά-δεδομένα ώστε να είναι εφικτή η επεξεργασία των γραφικών διεπαφών και μετά την ενσωμάτωση του κώδικα στην εφαρμογή.

Παράδειγμα Χρήσης

Το Juce επιδιώκει να “αποκρύψει” κάθε διαφορά ανάμεσα στις πλατφόρμες που υποστηρίζει στο επίπεδο συγγραφής κώδικα αλλά και στην τελική εφαρμογή. Εξ' ορισμού μάλιστα καλύπτει ακόμη και τον διαχειριστή παραθύρων του λειτουργικού (window manager) προσφέροντας τη μέγιστη δυνατή ομοιομορφία της εφαρμογής στις διαφορετικές πλατφόρμες.

Σχήμα 9: Στιγμιότυπο ενδεικτικής εφαρμογής Juce

Με τη βοήθεια του εργαλείου διαχείρισης Introjucer, η μεταγλώττιση μιας C++ εφαρμογής (Παράρτημα ΙΑ: Παράδειγμα Hello World στο πλαίσιο Juce) σε διαφορετικές πλατφόρμες γίνεται απλή υπόθεση. Το Introjucer δημιουργεί τα αντίστοιχα για κάθε πλατφόρμα αρχεία μεταγλώττισης. Έπειτα στη αντίστοιχη πλατφόρμα και με τα αντίστοιχα εργαλεία η εφαρμογή μπορεί εύκολα να μεταγλωττιστεί.

4.6 wxWidgets

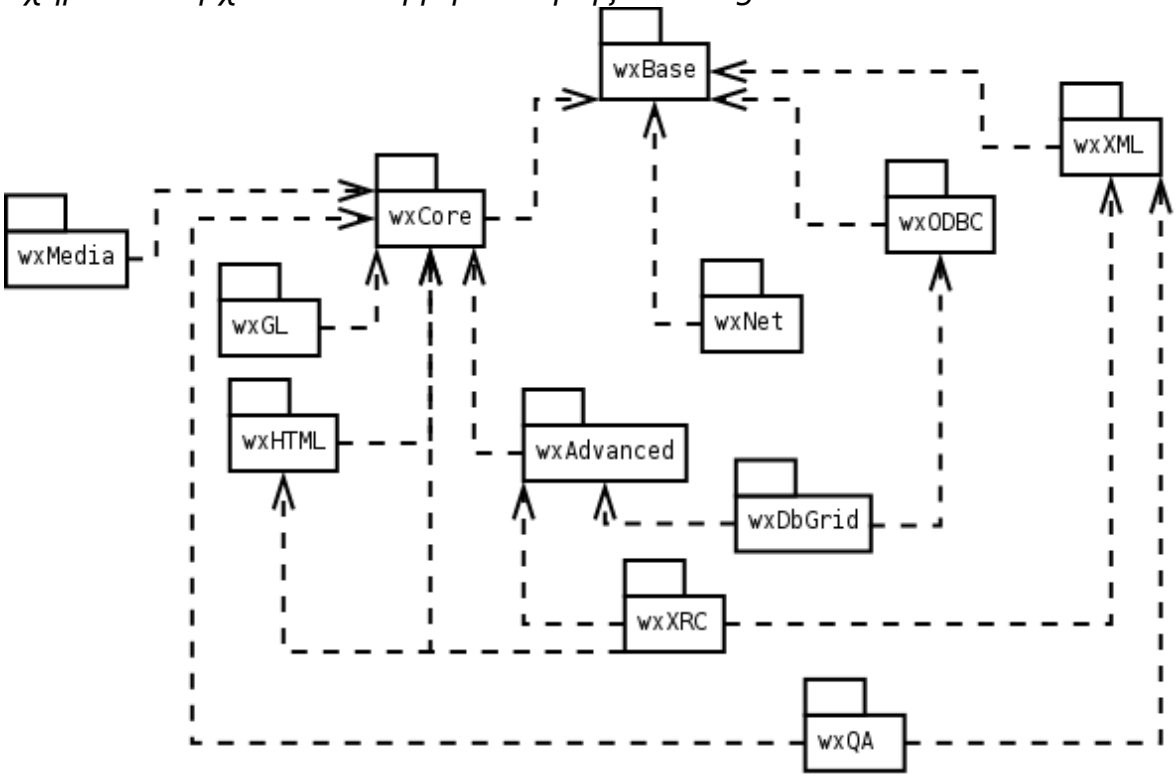
Η wxWidgets είναι μια C++ βιβλιοθήκη μεταφέρσιμων εφαρμογών για διαφορετικές πλατφόρμες. Προσφέρει μια σειρά από εργαλεία για την δημιουργία φορητών γραφικών διεπαφών, ενώ παράλληλα αποτελεί και ένα πλαίσιο (framework) ανάπτυξης λογισμικού φορητών εφαρμογών. Είναι διαθέσιμη για μια σειρά δημοφιλών γλωσσών προγραμματισμού, όπως Python, Perl, Ruby και πολλές άλλες. Σε αντίθεση με τις περισσότερες βιβλιοθήκες δημιουργίας μεταφέρσιμων διεπαφών η wxWidgets προσφέρει πραγματική ομοιομορφία με το περιβάλλον της κάθε πλατφόρμας, μιας και όπου είναι δυνατό χρησιμοποιεί το εκάστοτε API της πλατφόρμας αντί να εξομοιώνει τις γραφικές διεπαφές. Αυτό πέρα απ' την ομοιομορφία της γραφικής διεπαφής σε κάθε πλατφόρμα συμβάλει σημαντικά και στην αύξηση της απόδοσης της εφαρμογής. Η βιβλιοθήκη διατίθεται υπό τους όρους της άδειας χρήσης ανοιχτού λογισμικού LGPL με μια εξαίρεση που επιτρέπει την διάθεση της δυαδικής μορφής της τελικής εφαρμογής που χρησιμοποιεί την wxWidgets υπό τους όρους του χρήστη. Αυτό επιτρέπει τη χρήση της σε εφαρμογές ελεύθερου αλλά και εμπορικού λογισμικού.

Αρχιτεκτονική

Η wxWidgets μπορεί να χρησιμοποιηθεί ως μία μεγάλη ολοκληρωμένη μονολιθική βιβλιοθήκη ή ως διάφορες μικρότερες επιμέρους

βιβλιοθήκες.

Σχήμα 10: Αρχιτεκτονική βιβλιοθήκης wxWidgets



Η **wxBase** είναι η βασική βιβλιοθήκη με την οποία κάθε εφαρμογή wxWidgets πρέπει να χρησιμοποιεί. Περιέχει βασικές κλάσεις οι οποίες είναι απαραίτητες στις υπόλοιπες βιβλιοθήκες καθώς και φορητές αφαιρετικές κλάσεις οι οποίες κρύβουν τις διαφορές της κάθε πλατφόρμας. Η wxBase μπορεί να χρησιμοποιηθεί για την υλοποίηση εφαρμογών τερματικού που δε χρειάζονται γραφικές διεπαφές χρήστη.

Η **wxNet** παρέχει φορητές κλάσεις για δικτυακές λειτουργίες. Μια σειρά από φορητές στις διάφορες πλατφόρμες δικτυακές κλάσεις παρέχουν δυνατότητες χρήσης sockets (wxSocketClient, wxSocketServer), ροών δεδομένων (wxSocketOutputStream, wxSocketInputStream), δικτυακής επικοινωνίας διεργασιών (wxTCPServer, wxTCPClient, wxTCPConnection) και δικτυακού

συστήματος αρχείων (wxInternetFSHandler).

Η **wxXML** προσφέρει μερικές απλές κλάσεις ανάλυσης XML αρχείων.

Η **wxCore** περιέχει τις βασικές κλάσεις υλοποίησης γραφικών διεπαφών και της διαχείρισης τους. Όλες οι εφαρμογές με γραφική διεπαφή εξαρτώνται από αυτή την βιβλιοθήκη.

Η **wxAdvanced** περιέχει κάποιες εξεζητημένες κλάσεις για αντικείμενα γραφικών διεπαφών (wxCalendarCtrl, wxGrid, wxLayoutAlgorithm, wxSplashScreen, wxTaskBarIcon, wxWizard, wxSashLayoutWindow, wxSashWindow) και ελέγχου υλικού (wxJoystick, wxSound).

Η **wxMedia** παρέχει μια σειρά από κλάσεις χειρισμού πολυμέσων.

Η **wxGL** περιέχει την κλάση wxGLCanvas για την ενσωμάτωση της OpenGL βιβλιοθήκης στην wxWidgets. Σε αντίθεση με όλες τις υπόλοιπες βιβλιοθήκες δεν αποτελεί μέρος της μονολιθικής βιβλιοθήκης και αποτελεί ανεξάρτητη βιβλιοθήκη.

Η **wxHTML** είναι μια απλή βιβλιοθήκη απεικόνισης HTML δεδομένων.

Η **wxODBC** παρέχει δυνατότητες προσπέλασης διαφορετικών συστημάτων βάσεων δεδομένων.

Η **wxDbGrid**, συνδυάζει τις βιβλιοθήκες wxGrid και wxODBC για την δημιουργία της γραφικής απεικόνισης ενός πίνακα βάσης δεδομένων.

Η **wxQA** είναι μια βιβλιοθήκη που παρέχει κλάσεις σχετικές με την ποιότητα λογισμικού και παρέχει πληροφορίες αποσφαλμάτωσης απ' την τρέχουσα κατάσταση μιας εφαρμογής.

Η **wxXRC** περιέχει την κλάση wxXmlResource η οποία παρέχει

πρόσβαση σε XML αρχεία της μορφής XRC. Η wxWidgets χρησιμοποιεί XML αρχεία μορφής XRC για την αποθήκευση των αντικειμένων, μενού και εργαλειοθηκών της γραφικής διεπαφής για φόρτωση τους κατά την εκτέλεση της εφαρμογής.

Μεταφερσιμότητα

Η wxWidgets παρέχει τη δυνατότητα συγγραφής μεταφέρσιμων C++ προγραμμάτων για μια σειρά από πλατφόρμες λειτουργικών συστημάτων και υλικού. Η wxWidgets ορίζει ένα κοινό API για τις διαφορετικές πλατφόρμες, αλλά παράλληλα χρησιμοποιεί τις αντίστοιχες γραφικές διεπαφές της κάθε πλατφόρμας μέσω φορητών πακέτων (ports) για την κάθε μία.

Το **wxMSW** πακέτο είναι μια μεταφορά της wxWidgets για τις πλατφόρμες Microsoft Windows 95/98/ME/NT/2000/XP/Vista. Είναι συμβατό με μια σειρά από μεταφραστές για Windows όπως οι Microsoft Visual Studio C++, Borland 5.5, MinGW32, Cygwin και Watcom. Επίσης είναι συμβατό με 64 bit αρχιτεκτονικές των Windows.

Το **wxGTK** πακέτο είναι η μεταφορά της wxWidgets για τις Linux / Unix πλατφόρμες. Χρησιμοποιεί την GTK+ βιβλιοθήκη για την απεικόνιση της γραφικής διεπαφής όπου αυτό είναι δυνατό, ενώ εξομοιώνει όσα αντικείμενα δεν είναι διαθέσιμα.

Το **wxX11** πακέτο χρησιμοποιεί το X Window System (X11) για την απεικόνιση των γραφικών διεπαφών. Το πακέτο wxX11 είναι ιδανικό για συστήματα με περιορισμένους πόρους όπως οι φορητοί υπολογιστές παλάμης.

Το **wxMac** πακέτο κάνει δυνατή τη χρήση της wxWidgets στις πλατφόρμες Mac OS / OSX. Είναι συμβατό με τον μεταγλωττιστή της

Apple καθώς και τον MetroWerks CodeWarrior.

Το **wxOS2** πακέτο επιτρέπει τη χρήστη της wxWidgets στο λειτουργικό της IBM OS/2

Το **wxPalmOS** πακέτο επιτρέπει την χρήση της wxWidgets στους φορητούς υπολογιστές παλάμης της Palm.

Εργαλεία Ανάπτυξης

Η wxWidgets δεν περιέχει εργαλεία ανάπτυξης μέσα στο βασικό πακέτο της βιβλιοθήκης, μια σειρά όμως από ελεύθερα και εμπορικά εργαλεία ανάπτυξης είναι διαθέσιμα από τρίτους κατασκευαστές.

Το **wxGlade** είναι ένας γραφικός σχεδιαστής διεπαφών χρήστη, ανοιχτού κώδικα, γραμμένος σε Python, που εξάγει κώδικα για την wxWidgets σε διαφορετικές γλώσσες προγραμματισμού συμπεριλαμβανομένου και της C++. Όπως παραπέμπει και το όνομα του, το wxGlade δανείζεται το μοντέλο του Glade της βιβλιοθήκης GTK+/GNOME με χρήση XML αρχείων για την περιγραφή των γραφικών διεπαφών χρήστη. Το wxGlade παρέχεται για Windows και Linux ενώ μπορεί να μεταγλωττιστεί και για Mac OS X.

Το **wxFormBuilder** είναι ένα εργαλείο ανάπτυξης ανοιχτού κώδικα, το οποίο επιτρέπει την οπτική (visual) ανάπτυξη λογισμικού και εξαγωγή του αντίστοιχου κώδικα, επιτρέπει την ενσωμάτωση μη – γραφικών αντικειμένων (components) ενώ παράλληλα επιτρέπει την επέκταση των βασικών γραφικών αντικειμένων (widgets). Το wxFormBuilder αποθηκεύει τις πληροφορίες σε XML αρχεία αντί να τις ενσωματώνει απευθείας στον κώδικα. Όταν ολοκληρωθεί ο σχεδιασμός της διεπαφής χρήστη το wxFormBuilder μπορεί να εξάγει τον wxWidgets κώδικα για C++, Python ή wxWidgets resource αρχεία (XRC). Το wxFormBuilder

είναι διαθέσιμο σε Windows, Linux και Mac OS X.

Το **wxDesigner** είναι ένα μεταφέρσιμο εργαλείο ανάπτυξης γραφικών εφαρμογών wxWidgets με υποστήριξη για C++, Python, Perl, C# και δυνατότητα εξαγωγής XML. Βασικό του χαρακτηριστικό είναι η εύκολη και γρήγορη σχεδίαση διαλόγων (Dialogs) χρήστη, στοχεύοντας στην δυνατότητα επαναχρησιμοποίησης τους σε διαφορετικές γλώσσες προγραμματισμού και σωστής εμφάνισης τους στην εκάστοτε πλατφόρμα. Επίσης παράγει τον απαραίτητο κώδικα για την διαχείριση των γεγονότων της γραφικής διεπαφής.

Παράδειγμα Χρήσης

Μπορούμε να μεταγλωττίσουμε την ενδεικτική εφαρμογή wxWidgets του παραρτήματος ΙΒ (Παράδειγμα Hello World wxWidgets) σε περιβάλλον GNU/Linux

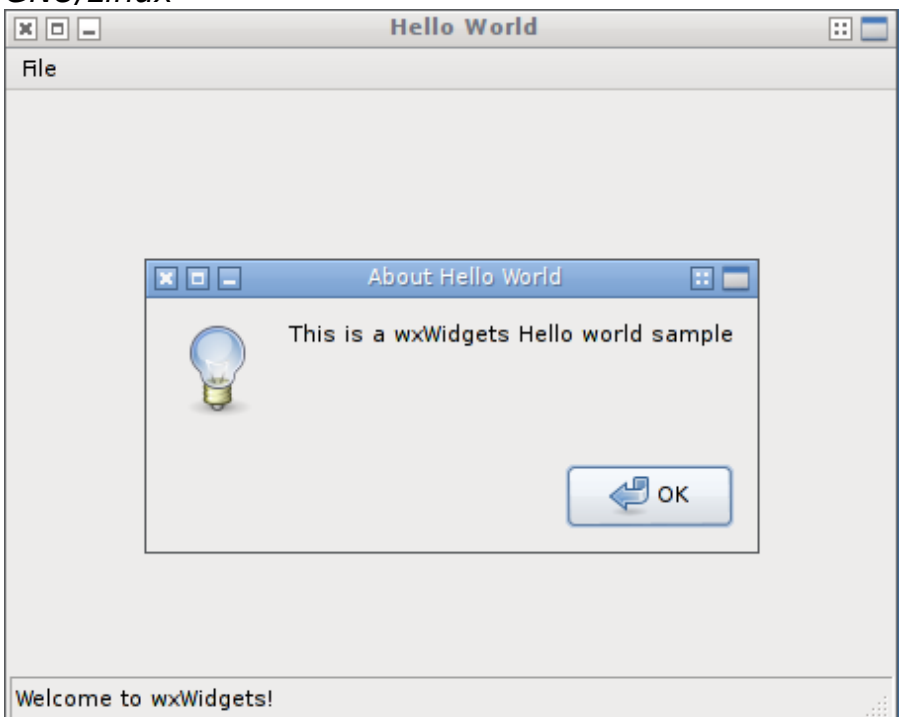
```
g++ hworld.cpp `wx-config --libs` `wx-config --cxxflags` -o hworld
```

και σε MS Windows

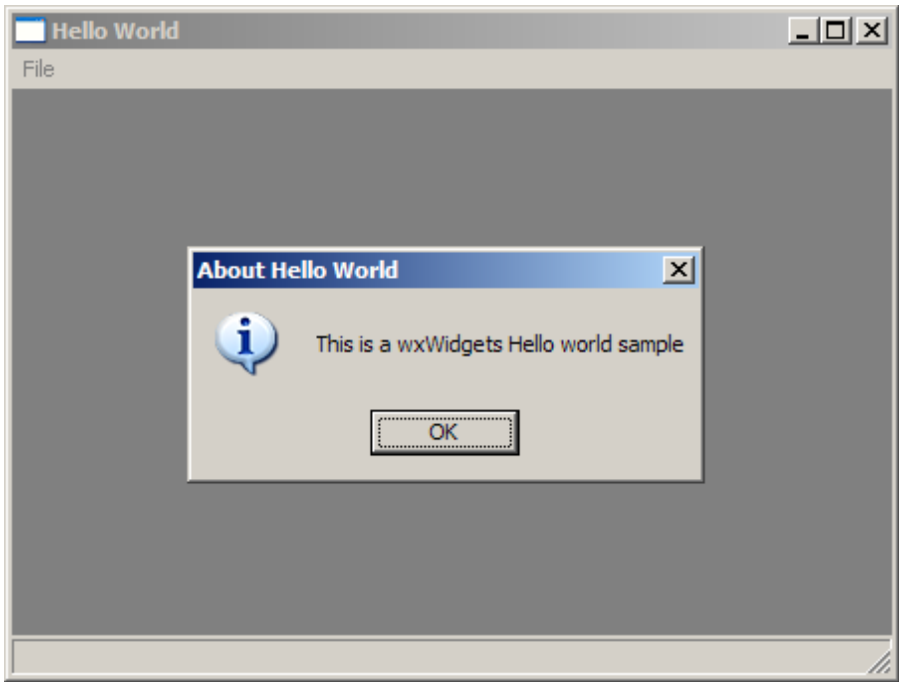
```
cl /Od /I "C:\wxWidgets-2.8.12\include" /I "C:\wxWidgets-2.8.12\lib\vc_lib\mswd" /D "WIN32" /D "__WXMSW__" /D "_WINDOWS" /D "_DEBUG" /D "__WXDEBUG__" /D "_MBCS" /Gm /EHsc /RTC1 /MDd /W3 /ZI /TP hworld.cpp /link /LIBPATH:"C:\wxWidgets-2.8.12\lib\vc_lib" /DYNAMICBASE /NXCOMPAT /MACHINE:X86 wxmsw28d_core.lib wxbase28d.lib wxtiffd.lib wxjpegd.lib wxpngd.lib wxzlibd.lib wxregexd.lib wxexpatd.lib winmm.lib comctl32.lib rpcrt4.lib wsock32.lib odbc32.lib kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
```

Εύκολα κάποιος παρατηρεί ότι η τελική εφαρμογή είναι διαφορετική από τη μία στην άλλη πλατφόρμα. Είναι όμως πλήρως εναρμονισμένη με το γραφικό περιβάλλον του λειτουργικού περιβάλλοντος μιας και χρησιμοποιεί τις βιβλιοθήκες της κάθε πλατφόρμας αντίστοιχα.

Σχήμα 11: Στιγμιότυπο εφαρμογής wxWidgets σε GNU/Linux



Σχήμα 12: Στιγμιότυπο εφαρμογής wxWidgets σε MS Windows



4.7 QT

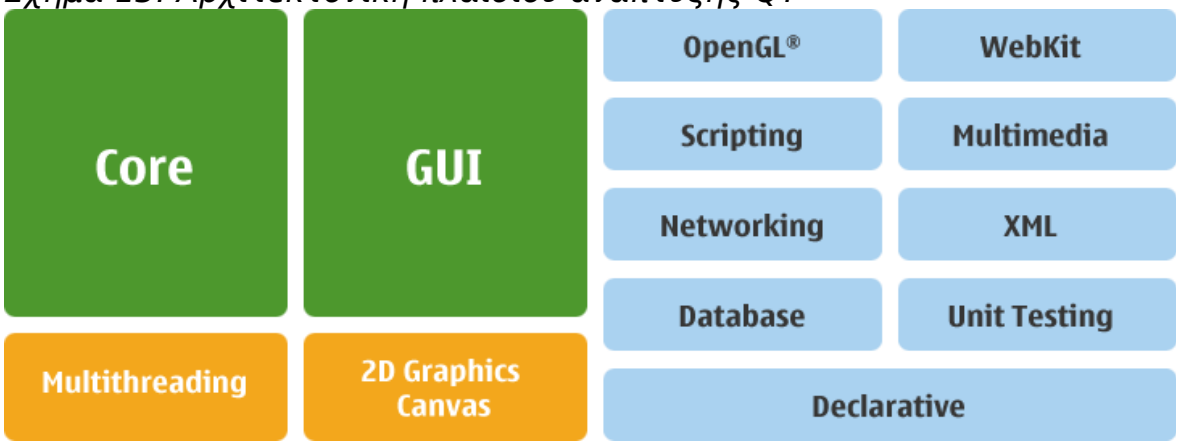
Το QT είναι ένα πλήρες πλαίσιο ανάπτυξης μεταφέρσιμων εφαρμογών και γραφικών διεπαφών. Το QT επιτρέπει την εκτέλεση εφαρμογών σε διαφορετικές πλατφόρμες με την απλή μεταγλώττιση του ίδιου πηγαίου κώδικα σε διαφορετικές πλατφόρμες. Μια μεγάλη συλλογή από πλατφόρμες και μεταγλωττιστές υποστηρίζονται, συμπεριλαμβανομένων των Windows, Mac OS X, Linux και Solaris. Το QT αναφέρεται και σε μια σειρά από φορητές συσκευές όπως το embedded Linux, Symbian, Windows CE καθώς και πολύ πρόσφατα το Android της Google.

Το QT υποστηρίζεται επαρκώς σε όλες τις διαθέσιμες πλατφόρμες που υποστηρίζει με χαρακτηριστικά πολυμέσων, 3D γραφικών, διεθνοποίηση (internationalization), SQL, XML και Unit testing, ενώ παράλληλα παρέχει μηχανισμούς άμεσης πρόσβασης χαρακτηριστικών της κάθε πλατφόρμας για εξειδικευμένες εφαρμογές.

Αρχιτεκτονική

Το QT αποτελείται από μια σειρά αυτοτελών μερών (modules) που παρέχουν όλη την απαραίτητη λειτουργικότητα για κάθε σχεδόν πιθανή εφαρμογή.

Σχήμα 13: Αρχιτεκτονική πλαισίου ανάπτυξης QT



Το **Core πακέτο** είναι το θεμελιώδες στοιχείο κάθε QT εφαρμογής καθώς και όλων των υπολοίπων μερών της, με μια σειρά από βασικές κλάσεις. Ανεξάρτητο απ' τις γραφικές διεπαφές, παρέχει φορητή λειτουργικότητα για τον χειρισμό εισόδου – εξόδου αρχείων και αντικειμένων, πολύ-νηματικού προγραμματισμού, επιπρόσθετων στοιχείων (plugins) και διαχείρισης ρυθμίσεων καθώς και έναν ολοκληρωμένο μηχανισμό επικοινωνίας μεταξύ των αντικειμένων με σήματα και υποδοχές (signals και slots).

Το **Multithreading πακέτο** προσφέρει μεταφέρσιμες κλάσεις που απλοποιούν τον παράλληλο προγραμματισμό, ενώ επίσης προσθέτουν χαρακτηριστικά για την καλύτερη εκμετάλλευση των αρχιτεκτονικών πολλαπλών πυρήνων επεξεργασίας.

Το **GUI πακέτο** περιέχει την λειτουργικότητα που απαιτείται για την ανάπτυξη εφαρμογών με γραφικές διεπαφές (GUI). Το Qt στις πιο πρόσφατες εκδόσεις του, χρησιμοποιεί πλέον το γραφικό υποσύστημα της κάθε πλατφόρμας για την οποία έχει μεταφραστεί, εκμεταλλευόμενο πλήρως τα πλεονεκτήματα της. Αποτελεί ένα πλήρες πακέτο γραφικών διεπαφών με μια σειρά από τροποποιούμενα γραφικά αντικείμενα (widgets), 2D και 3D καμβά (OpenGL), ενσωμάτωση μηχανή σχεδιασμού και απεικόνισης γραμματοσειρών, ταξιθετών (layouts) και διάφορων άλλων γραφικών εφέ.

Το **πακέτο 2D Graphics Canvas**, παρέχει μια επιφάνεια για την διαχείριση και αλληλεπίδραση μεταξύ ενός μεγάλου αριθμού δισδιάστατων γραφικών αντικειμένων. Βασικά του χαρακτηριστικά αποτελούν η δυνατότητα μεγέθυνσης, σμίκρυνσης και περιστροφή του καμβά, η υποστήριξη drag and drop και εκτύπωσης και η χρήση βιβλιοθηκών επιτάχυνσης γραφικών όπως η OpenGL, η OpenGL ES και OpenVG.

Το **πακέτο Scripting** παρέχει μια πλήρως ενσωματωμένη μηχανή σεναρίων (scripting language) η οποία συνεργάζεται πλήρως με τα αντικείμενα QObject, μεταφέρει τη λογική επικοινωνίας μεταξύ τους (Signals & Slots) στη γλώσσα σεναρίων και επιτρέπει την αλληλεπίδραση μεταξύ C++ και γλώσσας σεναρίων (scripting).

Το **Declarative πακέτο** περιέχει μια σειρά από κλάσεις για την δημιουργία δυναμικών, εναλλακτικών γραφικών διεπαφών που στοχεύουν σε φορητές συσκευές αφής. Το Declarative πακέτο επιτρέπει τη υλοποίηση των γραφικών διεπαφών και της λειτουργικότητας τους χωρίς τη χρήση κώδικα C++, αλλά με μια πιο διαδικτυακή προσέγγιση υλοποίησης τους. Με μια σειρά από QML αντικείμενα των οποίων η μορφή μπορεί να τροποποιηθεί με τη χρήση CSS και η λειτουργικότητα με χρήση της γλώσσας JavaScript, μεταφέρει την υλοποίηση των γραφικών διεπαφών απ' το προγραμματιστικό κομμάτι στο σχεδιαστικό. Στόχος είναι η εύκολη και γρήγορη δημιουργία πλούσιων και εναλλακτικών διεπαφών με τη χρήση τεχνολογιών που έχουν καθιερωθεί στο σχεδιασμό γραφικών διεπαφών, κυρίως στο διαδίκτυο.

Το **OpenGL πακέτο** προσφέρει κλάσεις για την εύκολη και γρήγορη ενσωμάτωση 3D γραφικών που εκμεταλλεύονται την επιτάχυνση γραφικών που προσφέρει το υλικό της κάθε πλατφόρμας.

Το **πακέτο Networking** παρέχει μια σειρά από κλάσεις που επιτρέπουν την υλοποίηση δικτυακών εφαρμογών σε διαφορετικές πλατφόρμες. Το πακέτο παρέχει μια σειρά από κλάσεις για την υλοποίηση εφαρμογών διακομιστή – πελάτη, κλάσεις υψηλού επιπέδου για τα πρωτόκολλα HTTP και FTP καθώς και για κλάσεις χαμηλού επιπέδου για την χρήση υποδοχών δικτύου (Sockets).

Το **πακέτο Database** προσφέρει κοινή διεπαφή για την επικοινωνία με

διαφορετικά συστήματα διαχείρισης βάσεων δεδομένων σε όλες τις πλατφόρμες. Παρέχει υποστήριξη για τα συστήματα βάσης δεδομένων MySQL, SQLite, Oracle, Sybase, DB2 καθώς και μια σειρά άλλων συστημάτων μέσω της διεπαφής ODBC. Επίσης στο επίπεδο της γραφικής διεπαφής παρέχει τη σύνδεση των δεδομένων με αντίστοιχα γραφικά αντικείμενα απεικόνισης δεδομένων.

Το **πακέτο WebKit** παρέχει ενσωμάτωση της μηχανής WebKit με το πλαίσιο Qt. Παρέχει την δυνατότητα ενσωμάτωσης ενός πλήρους φυλλομετρητής σε κάθε εφαρμογή Qt επιτρέποντας παράλληλα τον εμπλουτισμό του διαδικτυακού περιεχομένου με γραφικά αντικείμενα της Qt.

Το **πακέτο Multimedia** παρέχει κλάσεις για την υλοποίηση εφαρμογών πολυμέσων. Το συγκεκριμένο πακέτο μετονομάστηκε πρόσφατα σε Qt Mobility και αποτελεί πλέον επιπρόσθετο του πλαισίου Qt. Το Mobility πακέτο πέρα απ' την διεπαφή για αναπαραγωγή και εγγραφή πολυμέσων παρέχει πρόσβαση και στο υλικό της κάθε πλατφόρμας με έμφαση στις φορητές συσκευές.

Το **πακέτο XML** παρέχει μια σειρά από κλάσεις για την διαχείριση XML δεδομένων. Οι απαραίτητες κλάσεις για ανάγνωση και εγγραφή ροών δεδομένων XML, C++ υλοποιήσεις για την διαχείριση SAX και DOM δεδομένων καθώς και “μηχανή” για XQuery και XPath, περιέχονται σε αυτό το πακέτο.

Με το **πακέτο Unit Testing** το Qt ενσωματώνει ένα πλαίσιο ελέγχου ενοτήτων (Unit Testing) για τον έλεγχο μερών της εφαρμογής καθώς και των γραφικών διεπαφών.

Μεταφερσιμότητα

Το πλαίσιο Qt είναι διαθέσιμο για μια σειρά από πλατφόρμες υλικού (32bit και 64bit) και λογισμικού. Επίσης είναι διαθέσιμο και σε μια σειρά από φορητές συσκευές, όπως κινητά τηλέφωνα και οι υπολογιστές παλάμης.

Η εκδότρια εταιρία του Qt παρέχει υποστήριξη για τις παρακάτω πλατφόρμες:

- Linux/X11 – Qt για το X Window System (GNU/Linux, FreeBSD, HP-UX, Solaris, AIX, κ.α.)
- Mac OS X – Qt για τις πλατφόρμες της Apple Mac OS X
- Embedded Linux – Qt για embedded πλατφόρμες (υπολογιστές παλάμης, έξυπνα τηλέφωνα)
- Windows CE / Mobile – Qt για Windows CE
- Symbian – Qt για τις Symbian πλατφόρμες κινητών της Nokia
- Maemo – Qt για την Maemo πλατφόρμα για ταμπλέτες περιήγησης στο διαδίκτυο (Internet Tablets)

Πλατφόρμες	Μεταγλωττιστές
Linux (32 and 64-bit)	gcc 4.2
Microsoft Windows XP	gcc 4.4 (MinGW) (32-bit), MSVC 2005 (32 and 64-bit)
Microsoft Windows Vista	MSVC 2005, 2008
Microsoft Windows Vista 64bit	MSVC 2008
Microsoft Windows 7	MSVC 2008
Apple Mac OS X 10.6 "Snow Leopard"	Όπως παρέχεται απ' την Apple
Apple Mac OS X 10.5 "Leopard" x86_64 (Cocoa 32 and 64bit)	Όπως παρέχεται απ' την Apple
Embedded Linux QWS (ARM)	gcc
Windows CE 5.0 (ARMv4i, x86, MIPS)	MSVC 2005 WinCE 5.0 Standard (x86, pocket, smart, mipsii)
Symbian (Symbian/S60 5.0)	RVCT 2.2 [build 686], WINSCW 3.2.5 [build 482], GCCE (για εφαρμογές)

Πίνακας 1: Μεταγλωττιστές για την QT

Περισσότερες πλατφόρμες είναι διαθέσιμες αν και υποστηρίζονται μόνο από τρίτους κατασκευαστές:

- Qt για OpenSolaris – Qt για OpenSolaris πλατφόρμες

- Qt για Haiku – Qt για την Haiku πλατφόρμα
- Qt για OS/2 – Qt για την OS/2 eCS πλατφόρμα
- Qt για Amiga OS4 – Qt για την Amiga OS4 πλατφόρμα
- Qt για iPhone – Qt για την iPhone πλατφόρμα (υπό ανάπτυξη)
- Android-Lighthouse – Qt για την Android πλατφόρμα (υπό ανάπτυξη)
- Necessitas – Qt για την Android πλατφόρμα
- Qt για webOS – Qt για το webOS της Palm (υπό ανάπτυξη)
- Qt για Amazon Kindle DX – Qt για το Amazon Kindle DX (υπό ανάπτυξη)
- Qt για Wayland – Qt για τον Wayland display server (υπό ανάπτυξη)

Εργαλεία Ανάπτυξης

Το Qt έρχεται με δύο πολύ χρήσιμα μεταφέρσιμα εργαλεία ανάπτυξης που κάνουν εύκολη και γρήγορη την σχεδίαση, μεταγλώττιση και ανάπτυξη των εφαρμογών Qt σε διαφορετικές πλατφόρμες.

Το **Qt Creator** είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) που παρέχει εργαλεία για τον σχεδιασμό και την ανάπτυξη εφαρμογών που χρησιμοποιούν το πλαίσιο ανάπτυξης Qt. Το Qt Creator ενσωματώνει ένα προηγμένο επεξεργαστή πηγαίου κωδικά, αποσφαλματωτή, σχεδιαστή γραφικών διεπαφών καθώς και υποστήριξη για συστήματα διαχείρισης πηγαίου κώδικα.

Το **Qt Quick** είναι ένας υψηλού επιπέδου γραφικός σχεδιαστής διεπαφών χρήστη, που επιτρέπει στους προγραμματιστές και σχεδιαστές διεπαφών να συνεργαστούν για τη δημιουργία κινούμενων, φιλικών με φορητές συσκευές αφής εφαρμογών. Ο σχεδιαστής διεπαφών χρήστη επιτρέπει την δημιουργία διεπαφών στην περιγραφική γλώσσα QML και την προσθήκη λειτουργικότητας με τη γλώσσα JavaScript. Οι παραγόμενες διεπαφές χρήστη μπορούν να εμπλουτιστούν με C++ κώδικα.

Παράδειγμα Χρήσης

Θα χρησιμοποιήσουμε για παράδειγμα μια εφαρμογή του κλασικού παιχνιδιού τρίλιζα (Tic-Tac-Toe) υλοποιημένο σε C++ με τη χρήση της Qt (Παράρτημα ΙΓ: Ενδεικτική Εφαρμογή Τρίλιζας σε Qt).

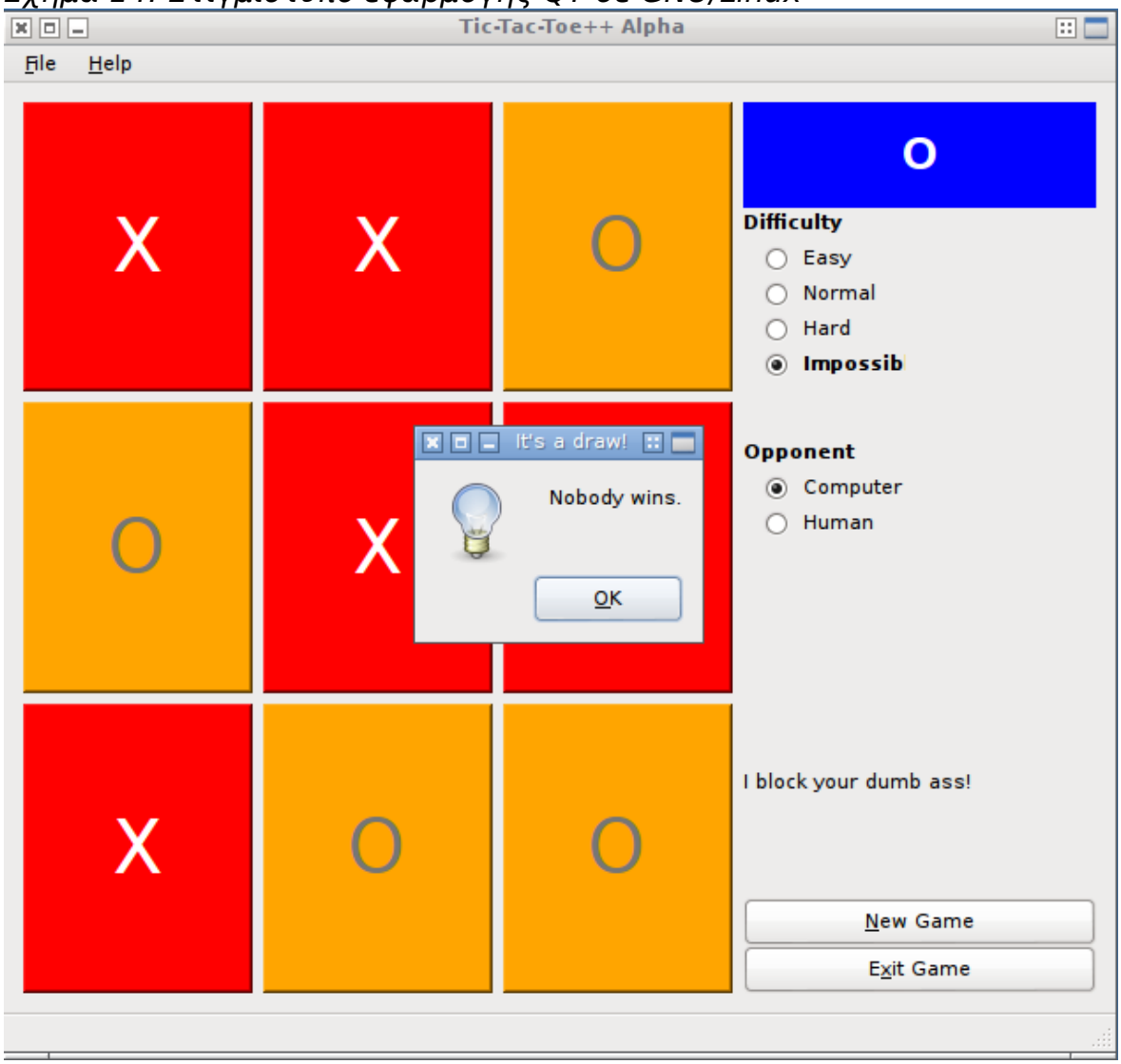
Η Qt κάνει πολύ εύκολη τη μεταφορά μιας εφαρμογής που υλοποιήθηκε σε για μια πλατφόρμα στις υπόλοιπες, με τη βοήθεια του εργαλείου qmake. Έτσι για να μεταγλωττίσουμε την παραπάνω εφαρμογή σε περιβάλλον GNU/Linux αρκεί:

```
qmake -project

qmake

make
```

Σχήμα 14: Στιγμιότυπο εφαρμογής QT σε GNU/Linux



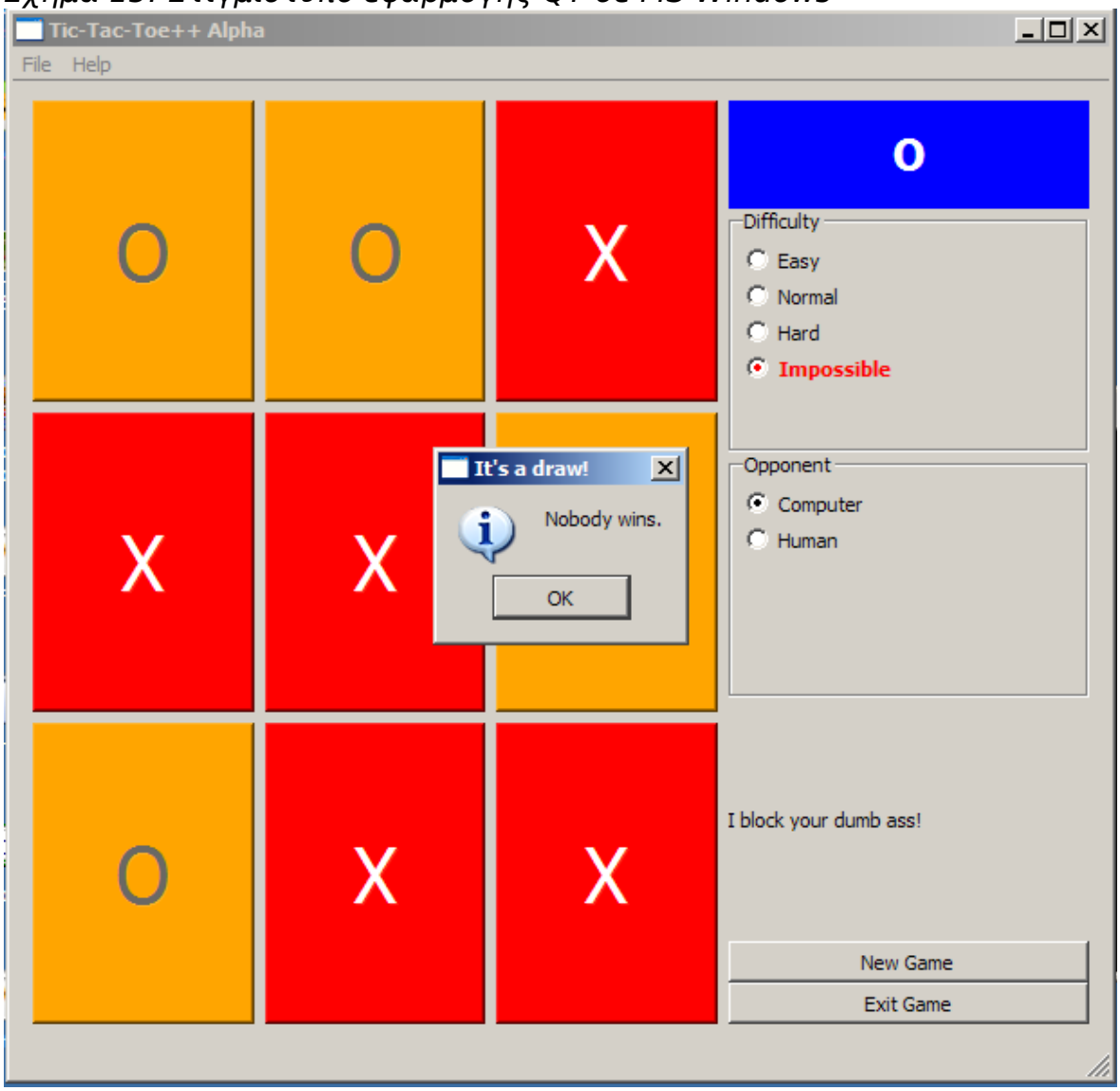
Αντίστοιχα για τη μεταγλώττιση σε περιβάλλον MS Windows:

```
qmake -project

qmake

nmake
```

Σχήμα 15: Στιγμιότυπο εφαρμογής QT σε MS Windows



4.8 Wt C++ - Web Toolkit

Η Wt είναι μια C++ βιβλιοθήκη για την ανάπτυξη διαδραστικών εφαρμογών διαδικτύου. Η βασική διαφορά απ' τις περισσότερες βιβλιοθήκες για ανάπτυξη διαδικτυακών εφαρμογών, είναι ότι υιοθετεί την λογική των εφαρμογών γραφείου (desktop) με χρήση γεγονότων (events) και αλλαγής καταστάσεων των γραφικών στοιχείων. Χρησιμοποιώντας σύγχρονες τεχνολογίες όπως η Ajax, WebSockets και

HTML5 επιτρέπει τη δημιουργία εφαρμογών διαδικτύου με υψηλή διαδραστικότητα που προσεγγίζει την αντίστοιχη των εφαρμογών γραφείου. Ένα άλλο βασικό χαρακτηριστικό της βιβλιοθήκης είναι η αφαιρετικότητα που παρέχει στον προγραμματιστή από όλες τις τεχνολογίες διαδικτύου (HTML/XHTML, JavaScript, CSS, Ajax, WebSockets, Comet, Forms, DHTML, SVG/VML/Canvas) επιτρέποντας την ανάπτυξη διαδικτυακών εφαρμογών σε απλή C++.

Αρχιτεκτονική

Η Wt υλοποιεί το μοντέλο διακομιστή – πελάτη, όπου όλη η λογική της εφαρμογής συμβαίνει στον διακομιστή ενώ η αλληλεπίδραση του χρήστη (πελάτη) είναι δυνατή μέσω οποιουδήποτε φυλλομετρητή (browser).

Το **βασικό (core) μέρος** της βιβλιοθήκης φροντίζει για την συμβατότητα με τους μεγαλύτερους φυλλομετρητές, αλλά παράλληλα παρέχει εναλλακτική λειτουργικότητα για τους απλούς HTML. Η βιβλιοθήκη αναλαμβάνει τον χειρισμό όλων των χαρακτηριστικών μιας διαδικτυακής εφαρμογής, όπως τον χειρισμό συνεδριών, cookies, ιστορικό και σελιδοδείκτες φυλλομετρητή, παρέχοντας μια απαλλαγμένη από λεπτομέρειες διεπαφή προγραμματισμού (API). Επίσης παρέχει υποστήριξη πολυγλωσσίας και διεθνοποίησης (localization).

Η βιβλιοθήκη ενσωματώνει μια σειρά από χαρακτηριστικά ασφάλειας απαραίτητα σε κάθε διαδικτυακή εφαρμογή. Οι συνεδρίες (sessions) είναι απομονωμένες σε διαφορετικά επίπεδα μνήμης πυρήνα, ώστε να εξασφαλίζεται η ιδιωτικότητα της κάθε μίας, από πιθανά λάθη (bugs) της εφαρμογής. Επίσης, παρέχει μηχανισμούς κρυπτογράφησης και πιστοποίησης πελάτη με τη χρήση SSL ή TLS μέσω του πρωτοκόλλου

HTTPS. Η εκτεταμένη χρήση της τεχνολογίας Ajax, μειώνει το απαιτούμενο εύρος χρήσης του δικτύου επιτρέποντας τη συνεχή χρήση της εφαρμογής μέσω HTTPS. Η προστασία από επιθέσεις XSS είναι ενσωματωμένη στη βιβλιοθήκη, φιλτράροντας κάθε έξοδο της εφαρμογής στον φυλλομετρητή.

Η βιβλιοθήκη παρέχει **χειρισμό γεγονότων (event handling)** με τη χρήση σημάτων και υποδοχέων (signal / slot). Διαφορετικά γεγονότα παρέχονται για τις συσκευές εισόδου (πληκτρολόγιο, ποντίκι) και την κατάσταση των γραφικών στοιχείων, τα οποία επιστρέφουν λεπτομέρειες για το κάθε γεγονός, όπως τη θέση του ποντικού, την κατάσταση ενός γραφικού πλήκτρου ή ενός κουμπιού του πληκτρολογίου. Η βιβλιοθήκη αναλαμβάνει τη μετατροπή των σημάτων της C++ στον αντίστοιχο κώδικα JavaScript επιτρέποντας χειρισμό γεγονότων στη μεριά του πελάτη παράλληλα με τη μεριά του διακομιστή. Επίσης παρέχει λειτουργικότητα Drag & Drop, χρόνο-γεγονότα (timed-events) καθώς και κλήσης διακομιστή - πελάτη (server push).

Το **σύστημα γραφικών** παρέχει μια ενοποιημένη διεπαφή (API) η οποία χρησιμοποιεί τη γραφική μηχανή του φυλλομετρητή για το σχεδιασμό γραφικών μέσω VML, SVG ή HTML5. Επίσης παρέχει κλάσεις για την μετατροπή των γραφικών σε διαφορετικά φορμάτ εικόνας (JPG, GIF, PNG) ή διανυσμάτων (SVG, PDF).

Μια σειρά από **γραφικά αντικείμενα (GUI components)** επιτρέπουν τον εύκολο σχεδιασμό γραφικών διεπαφών. Πέρα απ' τα βασικά HTML στοιχεία, παρέχονται επιπλέον γραφικά στοιχεία (widgets) με ενσωματωμένη υποστήριξη επικοινωνίας μεταξύ πελάτη και διακομιστή. Τα δεδομένα που εισάγονται απ' τον χρήστη δύναται να επικυρωθούν (validation) στην μεριά του πελάτη καθώς και του

διακομιστή πριν την καταχώρηση. Πιο σύνθετα γραφικά στοιχεία, που αποτελούνται από επιμέρους βασικά στοιχεία και χρησιμοποιούν τις δημόσιες διεπαφές της βιβλιοθήκης παρέχουν πλουσιότερα γραφικά αντικείμενα όπως ημερολόγια, γραφικά δέντρα, μενού επιλογών και παράθυρα διαλόγων. Γραφικά αντικείμενα παρέχονται για την ενσωμάτωση πολυμέσων καθώς και διαγραμμάτων.

Η βιβλιοθήκη **προσπέλασης συστημάτων βάσεων δεδομένων** προσφέρει μια αντικειμενοστραφής προσέγγιση των πινάκων των σχέσεων της βάσης δεδομένων. Κλάσεις σε C++ αντιστοιχίζονται με τους αντίστοιχους πίνακες της βάσης δεδομένων, τα πεδία των κλάσεων με τις στήλες των πινάκων και δείκτες και συλλογές δεικτών με σχέσεις μεταξύ των πινάκων. Τα ερωτήματα στη βάση επιστρέφουν αντικείμενα (database objects) που είναι άμεσα διαθέσιμα απ' τον C++ κώδικα χωρίς τη χρήση SQL.

Μια σειρά από κλάσεις ελέγχου ποιότητας λογισμικού, επιτρέπουν την αναπαραγωγή των γεγονότων της εφαρμογής και την προσπέλαση του δέντρου των γραφικών αντικειμένων (widgets). Έτσι, το περιβάλλον ελέγχου ποιότητας επιτρέπει την εξομοίωση της αρχικοποίησης της εφαρμογής και των γεγονότων χωρίς την χρήση φυλλομετρητή.

Μεταφερσιμότητα

Η Wt υλοποιεί την αρχιτεκτονική διακομιστή – πελάτη για την ανάπτυξη διαδικτυακών εφαρμογών. Οι διαδικτυακές εφαρμογές γενικά περιγράφονται ως μεταφέρσιμες εφαρμογές μιας και είναι προσβάσιμες μέσω οποιουδήποτε φυλλομετρητή, διαθέσιμου στην εκάστοτε πλατφόρμα υλικού και λογισμικού.

Οι εφαρμογές Wt μπορούν να φιλοξενηθούν στους πιο δημοφιλείς διακομιστές διαδικτυακών εφαρμογών (Apache, IIS, lighthttpd, nginx)

σε πλατφόρμες Unix / Posix, Mac OS X, Windows και Android. Επίσης ο κώδικας μπορεί να μεταγλωττιστεί στους περισσότερους μεταγλωττιστές διαθέσιμους στις παραπάνω πλατφόρμες.

4.9 MoSync

Το MoSync είναι μια πλατφόρμα ανάπτυξης εφαρμογών για φορητές συσκευές (mobile devices). Αποτελείται από μια σειρά στενά συνδεδεμένων στοιχείων - μεταγλωττιστές, βιβλιοθήκες, προφίλ συσκευών και εργαλείων – τα οποία αλληλεπιδρούν μεταξύ τους με σκοπό την ανάπτυξη μεταφέρσιμων εφαρμογών για φορητές συσκευές.

Τα εργαλεία και οι γλώσσες προγραμματισμού για την ανάπτυξη εφαρμογών για φορητές συσκευές ποικίλουν ανάλογα με τον εκάστοτε κατασκευαστή της συσκευής. Οι παλαιότερες φορητές συσκευές επέτρεπαν την ανάπτυξη λογισμικού στη γλώσσα προγραμματισμού Java μέσω της JavaME, ενώ οι πιο σύγχρονες επιβάλουν την χρήση συγκεκριμένων γλωσσών προγραμματισμού διαφορετικές για κάθε συσκευή. Το MoSync επιτρέπει την ανάπτυξη εφαρμογών με τη χρήση της γλώσσας προγραμματισμού C++, οι οποίες μπορούν να μεταφραστούν σε δυαδικό κώδικα JavaME ή σε κώδικα μηχανής της αντίστοιχης πλατφόρμας της εκάστοτε φορητής συσκευής.

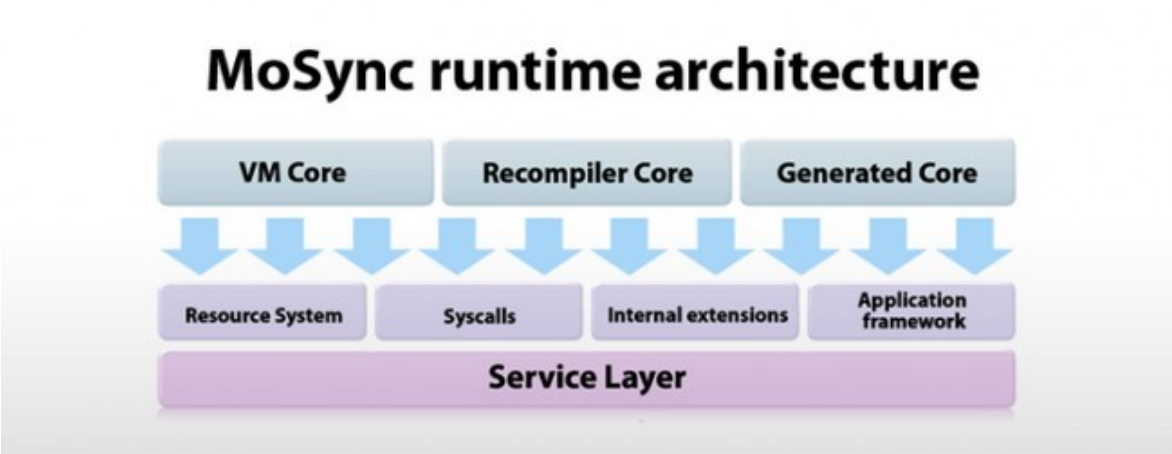
Αρχιτεκτονική

Κάθε MoSync εφαρμογή υλοποιείται αρχικά για το MoSync περιβάλλον εκτέλεσης MoRE (MoSync Runtime Environment). Το MoRE αποτελεί την υλοποίηση αναφοράς του MoSync ανεξαρτήτως κατασκευαστή. Το MoRE εκτελεί MoSync δυαδικό κώδικα και συμπεριφέρεται όπως μια ιδεατή φορητή συσκευή, χωρίς όμως να εξομοιώνει τη λειτουργία μιας συγκεκριμένης συσκευής.

Η αρχιτεκτονική του MoSync προσπαθεί να μειώσει την πολυπλοκότητα της ανάπτυξης μεταφέρσιμων εφαρμογών δίνοντας έμφαση στις διαφορές των συσκευών παρά στο πλήθος των συσκευών που θα πρέπει να υποστηρίζονται. Χαρακτηριστικά όπως το μέγεθος της οθόνης, η χωρητικότητα της μνήμης, οι μέθοδοι εισαγωγής δεδομένων και η διαθεσιμότητα χαρακτηριστικών μεταξύ των API είναι παραμετροποιήσιμα στο περιβάλλον εκτέλεσης MoRE. Ανεξαρτήτως των χαρακτηριστικών που έχουν επιλεγεί το MoSync SDK εξασφαλίζει την όμοια συμπεριφορά σε κάθε συσκευή που θα εκτελεστεί η εφαρμογή. Ο στόχος λοιπόν είναι όποια εφαρμογή εκτελείται σωστά στο περιβάλλον εκτέλεσης MoRE να εκτελείται με τον ίδιο τρόπο σε οποιαδήποτε φορητή συσκευή.

Το MoSync έχει δύο βασικές μηχανές εκτέλεσης (Runtime), μία υλοποιημένη σε C++ και μία σε Java. Η αρχιτεκτονική και των δύο όμως είναι κοινή.

Σχήμα 16: Αρχιτεκτονική μηχανή εκτέλεσης του MoSync



Το **Επίπεδο Υπηρεσιών (Service Layer)** προσφέρει υποστήριξη για

I/O, νήματα, διαχείριση μνήμης και άλλες υποστηρικτικές λειτουργίες.

Το **Πλαίσιο Εφαρμογής (Application Framework)** είναι υπεύθυνο για την εναρκτήρια κλήση της μηχανής εκτέλεσης. Το μέγεθος και οι λειτουργίες του μπορεί να διαφέρουν στην εκάστοτε πλατφόρμα, αλλά συνήθως εμπεριέχει τον χειρισμό γεγονότων, την αρχικοποίηση και τερματισμό των διαδικασιών.

Η μονάδα **Κλήσεων Συστήματος (Syscalls)** είναι υπεύθυνη για την υλοποίηση των κοινών χαρακτηριστικών των διαφορετικών πλατφορμών, όπως τα γραφικά, ο ήχος και η επικοινωνία.

Το **Σύστημα Διαχείρισης Βοηθητικών Πόρων (Resource system)** παρέχει υποδομές για τη διαχείριση των βοηθητικών δεδομένων της εφαρμογής όπως τα γραφικά και τα αρχεία ήχου.

Τα χαρακτηριστικά που δεν είναι διαθέσιμα σε όλες τις πλατφόρμες υλοποιούνται ως **Εσωτερικές Επεκτάσεις (Internal extensions)**. Τα χαρακτηριστικά αυτά υλοποιούνται ως αριθμημένες μέθοδοι που υλοποιούνται μέσω κλήσεων συστήματος (syscalls). Σε περίπτωση που το συγκεκριμένο χαρακτηριστικό δεν είναι διαθέσιμο σε μια πλατφόρμα, η κλήση επιστρέφει ένα συγκεκριμένο σφάλμα που περιγράφει το λόγο για τον οποίο δεν είναι διαθέσιμο.

Ο **Πυρήνας (Core)** είναι υπεύθυνος για την εκτέλεση των MoSync προγραμμάτων, αλληλεπιδρώντας με τις κλήσεις συστήματος και το σύστημα διαχείρισης βοηθητικών πόρων. Ο **Πυρήνας (Core)** είναι διαθέσιμος σε τρεις διαφορετικούς τύπους, απ' τους οποίους ο κάθε τύπος έχει τα δικά του μοναδικά πλεονεκτήματα.

Ένας **Πυρήνας τύπου Εικονικής Μηχανής (VM Core)** είναι μια εικονική μηχανή που φορτώνει, διερμηνεύει και εκτελεί δυαδικό κώδικα

MoSync απευθείας. Η εκτέλεση υλοποιείται στα πλαίσια μια μικρής διαδικασίας, επιτρέποντας έτσι την αποτελεσματική βελτιστοποίηση κατά την εκτέλεση (JIT). Αυτό ο τύπος χρησιμοποιείται για το JavaME.

Ο **Πυρήνας τύπου Επανα-μεταγλώττισης (Re-compiler Core)**, φορτώνει τον δυαδικό κώδικα MoSync και τον ξανά-μεταγλωττίζει στην γλώσσα μηχανής της εκάστοτε πλατφόρμας, πριν εκτελεστεί η εφαρμογή. Αυτό ο τύπος χρησιμοποιείται στα Windows Mobile και Symbian.

Ο **Πυρήνας τύπου Παραγωγής Κώδικα (Generated Core)**, δε διερμηνεύει ούτε μεταγλωττίζει δυαδικό κώδικα MoSync, αντιθέτως αποτελείται από πηγαίο κώδικα που παράχθηκε για την συγκεκριμένη πλατφόρμα. Αυτός ο τύπος χρησιμοποιείται για το iOS της Apple.

Μεταφερσιμότητα

Η MoSync υποστηρίζει μια πλειάδα φορητών συσκευών.

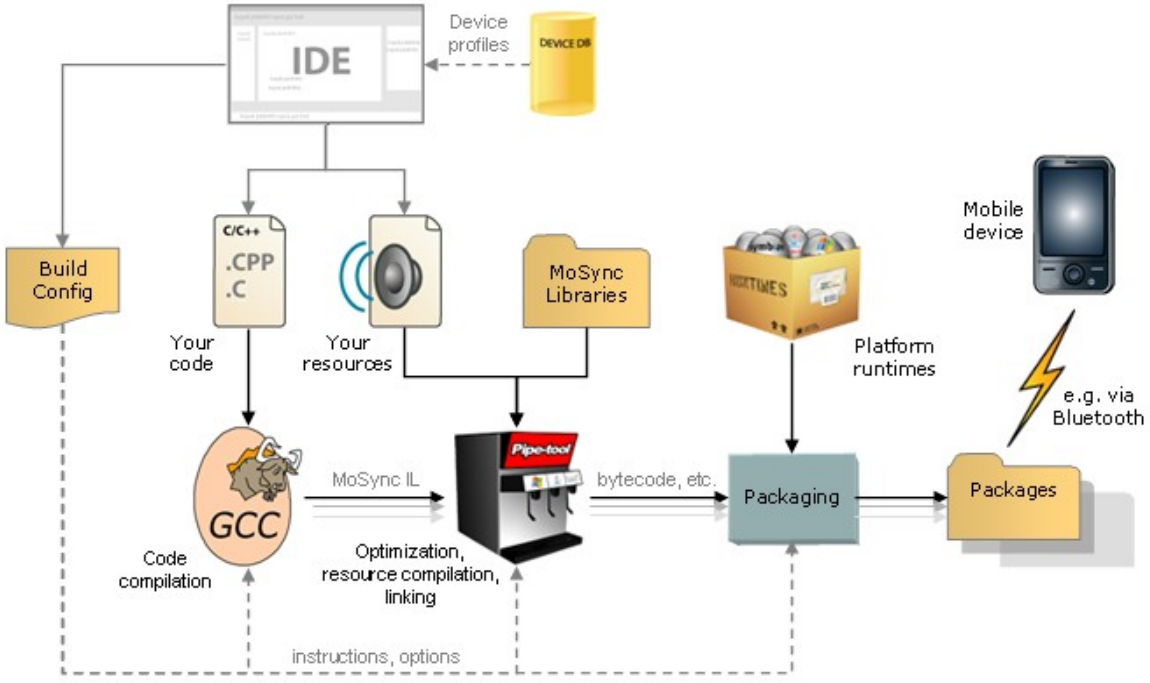
Πλατφόρμα	Εκδόσεις					
Android	1.5	1.6	2.0	2.1	2.2	2.3.3
Blackberry	4.x	5.x	6.x	7.x		
iOS	3.x	4.0.x	4.1.x	4.2.x	4.3.x	
JavaME MIDP	2.x	3.x				
Moblin	2.x					
Symbian S70	2nd	3rd	5th			
Windows Mobile	2003	5.x	6.0	6.1	6.5	
Windows Phone	7.0	7.5				

Πίνακας 2: Πίνακας υποστηριζόμενων πλατφορμών από MoSync

Εργαλεία Ανάπτυξης

Η ομάδα ανάπτυξης το MoSync SDK έχει υλοποιήσει το δικός της ενδιάμεσο μεταγλωττιστή του GNU C Compiler ο οποίος μεταγλωττίζει τον C++ κώδικα σε μια ενδιάμεση κοινή φορητή γλώσσα η οποία μετά από μια σειρά μετατροπών, αναλύσεων και βελτιώσεων (Pipe-tool) μετατρέπεται σε δυαδικό κώδικα για την εκάστοτε συσκευή. Μια βάση δεδομένων με διαφορετικά προφίλ καθοδηγεί την όλη διαδικασία ώστε να εξασφαλιστεί η συμβατότητα με κάθε φορητή συσκευή και να δημιουργηθεί το κατάλληλο πακέτο εγκατάστασης (packaging).

Σχήμα 17: Διαδικασία Μεταγλώττισης εφαρμογών MoSync



Το MoSync αποτελεί ένα ολοκληρωμένο πακέτο εργαλείων ανάπτυξης μεταφέρσιμων εφαρμογών για φορητές συσκευές, ξεκινώντας απ' το περιβάλλον συγγραφής του κώδικα (IDE), τον μεταγλωττιστή και διαχειριστή πακέτων και φτάνοντας μέχρι τους εξομοιωτές φορητών συσκευών.

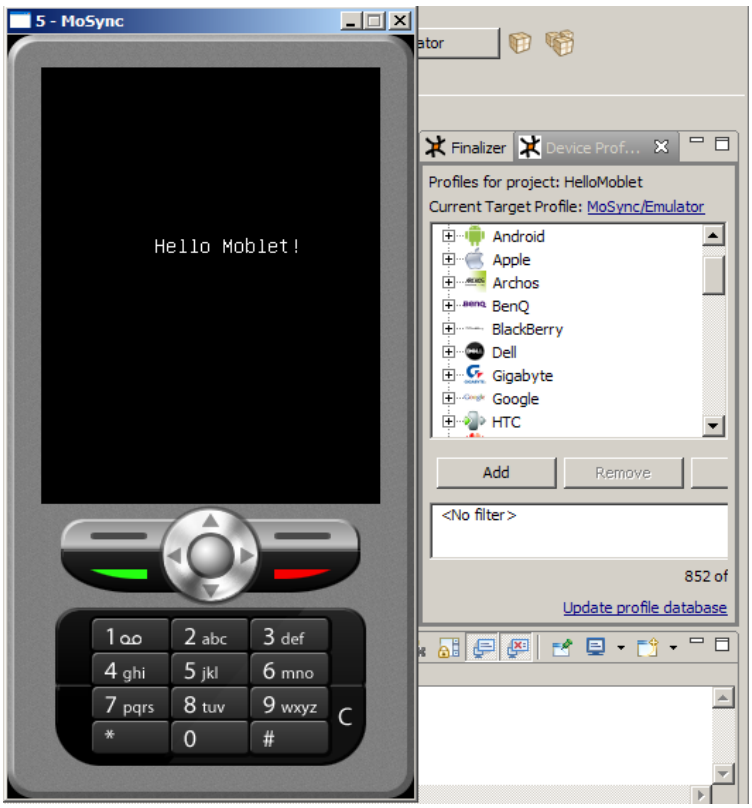
Παράδειγμα Χρήσης

Η ανάπτυξη μεταφέρσιμων εφαρμογών για φορητές συσκευές καθοδηγείται πλήρως απ' το MoSync. Μέσα απ' το Eclipse-based γραφικό περιβάλλον ανάπτυξης του MoSync μπορείς να δημιουργήσεις ένα νέο Project χρησιμοποιώντας κάποια απ' τα υπάρχοντα πρότυπα. Μπορούμε για παράδειγμα να δημιουργήσουμε ένα C++ Moblet Project για τις ανάγκες του παρακάτω παραδείγματος, όπως φαίνεται στο παράρτημα ΙΔ (Ενδεικτική εφαρμογή MoSync).

Στο παρακάτω στιγμιότυπο φαίνεται η εκτέλεση του παραπάνω

παραδείγματος κώδικα στον εξομοιωτή φορητών συσκευών. Στα δεξιά επίσης φαίνεται μέρος των προφίλ ρυθμίσεων των διαφόρων συσκευών ανά κατασκευαστή. Επιλέγοντας το αντίστοιχο προφίλ ρυθμίσεων παράγονται τα απαραίτητα για τη συγκεκριμένη εφαρμογή εκτελέσιμα.

Σχήμα 18: Στιγμιότυπο του εξομοιωτή MoSync



Συμπεράσματα

Η φορητότητα αποτελεί πλέον σημαντικό χαρακτηριστικό του λογισμικού που χρησιμοποιούμε στην καθημερινή ζωής μας. Η ανάγκη πρόσβασης στις εφαρμογές που χρησιμοποιούμε στο σπίτι μας, στον Η/Υ της εργασίας μας ή ακόμα και κατά τη διάρκεια των μετακινήσεων μας στις φορητές ηλεκτρονικές συσκευές μας είναι αδιαμφισβήτητη. Επίσης η δυνατότητα ενός προϊόντος λογισμικού να είναι διαθέσιμο στις διαφορετικές πλατφόρμες, αυξάνει το σύνολο των χρηστών στους οποίους απευθύνεται. Χαρακτηριστικό παράδειγμα οι εφαρμογές επικοινωνίας που στόχο έχουν να προσελκύσουν όσο το δυνατόν περισσότερους χρήστες, οι οποίοι ανεξαρτήτου πλατφόρμας θα μπορούν να επικοινωνούν μεταξύ τους.

Οι διαφορές ανάμεσα στις πλατφόρμες υλικού και λογισμικού, μπορούν να επηρεάσουν σημαντικά τη μεταφερσιμότητα μιας εφαρμογής λογισμικού, την οποία πρέπει να λαμβάνουν υπόψιν τους απ' την αρχή της ανάπτυξης της. Αν και η C/C++ είναι μια γλώσσα προγραμματισμού υψηλού επιπέδου, διαφοροποιήσεις στις πλατφόρμες υλικού, λογισμικού ακόμη και μεταξύ μεταγλωττιστών μπορούν να κάνουν πολύ δύσκολη έως αδύνατη τη μεταγλώττιση και εκτέλεση του ίδιου πηγαίου κώδικα σε άλλες πλατφόρμες. Η χρήση εξ' ολοκλήρου διαφορετικών εκδόσεων πηγαίου κώδικα για κάθε πλατφόρμα γρήγορα αποδεικνύεται αναποτελεσματική και ασύμφορη μιας και κάνει δύσκολη τη συντήρηση του λογισμικού και την ομοιόμορφη ανάπτυξη του οδηγώντας σε διαφορετικά προϊόντα με διαφορετικά χαρακτηριστικά και ελαττώματα. Για να μπορέσει μια εφαρμογή C++ να είναι μεταφέρσιμη απαιτείται η εφαρμογή της αρχής της αφαιρετικότητας. Διάφορες μεθοδολογίες ανάπτυξης λογισμικού βοηθούν στην συγγραφή μεταφέρσιμου και υψηλής ποιότητας κώδικα. Η συγγραφή του κώδικα

δεν είναι η μόνη εργασία η οποία επηρεάζεται απ' τις διαφορές μεταξύ των συστημάτων. Κατά τον κύκλο ανάπτυξης μιας εφαρμογής είναι απαραίτητη τακτική μεταφορά της, στις διαφορετικές πλατφόρμες που θα υποστηρίζει. Έτσι μια σειρά από εργαλεία ανάπτυξης, κατά προτίμηση μεταφέρσιμα, πρέπει να χρησιμοποιηθούν στην κάθε διαφορετική πλατφόρμα ώστε να μεταγλωττιστεί η εφαρμογή.

Η υλοποίηση των διεπαφών και η αντιμετώπιση των προβλημάτων που προκύπτουν στην κάθε πλατφόρμα, μπορεί να γίνει πολύ επίπονη διαδικασία και να αποπροσανατολίσει την ανάπτυξη της ίδιας της εφαρμογής. Γι' αυτό το λόγο είναι απαραίτητη η χρήση μιας ή περισσοτέρων μεταφέρσιμων βιβλιοθηκών. Υπάρχει μια πλειάδα από ελεύθερες βιβλιοθήκες για τη μεταφορά C++ προγραμμάτων σε διαφορετικές πλατφόρμες. Κάποιες από αυτές αντιμετωπίζουν συγκεκριμένα μόνο προβλήματα της ανάπτυξης μεταφέρσιμου λογισμικού ενώ κάποιες άλλες αποτελούν ολοκληρωμένες λύσεις για την γρήγορη ανάπτυξη μεταφέρσιμων εφαρμογών. Οι περισσότερες βιβλιοθήκες για C++ επιλέγουν να υλοποιήσουν ένα χαμηλό επίπεδο που αναλαμβάνει να υλοποιήσει διαφορετικά τις ιδιαιτερότητες της κάθε πλατφόρμας και να προσφέρει μια υψηλότερου επιπέδου διεπαφή στον προγραμματιστή. Μια άλλη προσέγγιση, η οποία γίνεται περισσότερο δημοφιλής με τον καιρό, είναι η χρήση τεχνολογιών διαδικτύου (web), επιτρέποντας την εκτέλεση της εφαρμογής-πελάτη σε οποιαδήποτε πλατφόρμα διαθέτει φυλλομετρητή.

Παρόλο που η C++ στην παρούσα της μορφή, παρουσιάζει αδυναμίες στην ανάπτυξη μεταφέρσιμων εφαρμογών, ένα υπερπλήρες οικοσύστημα εργαλείων και βιβλιοθηκών, την κατατάσσουν στις πρώτες επιλογές γλώσσας προγραμματισμού, για την ανάπτυξη μεταφέρσιμου λογισμικού. Η υψηλή δημοτικότητα της και οι σημαντικές τις επιρροές απ' τη C, τη γλώσσα προγραμματισμού με

μεταγλωττιστές διαθέσιμους για τις περισσότερες πλατφόρμες υλικού – λογισμικού, της επιτρέπουν να χαρακτηρίζεται ως μία απ' τις περισσότερο μεταφέρσιμες γλώσσες προγραμματισμού αυτή τη στιγμή.

Βιβλιογραφία – Αναφορές

- S. Logan, (2007): Cross-Platform Development in C++: Building Mac OS X, Linux and Windows
- B. Hook, (2005): Write Portable Code
- D. Williams, (2007): C++ Portability Guide - https://developer.mozilla.org/en/C++_Portability_Guide
- CreateProcess Function (Windows) - http://gsraj.tripod.com/design/creational/factory/factory.html
- How to use Process Class in Managed C++ - CodeProject - http://www.codeproject.com/KB/mcpp/SelProcessInfo.aspx
- CreateProcess Function (Windows) - http://msdn.microsoft.com/en-us/library/ms682425%28v=vs.85%29.aspx
- Boost C++ Libraries - http://www.boost.org/
- Boost C++ Libraries - Wikipedia, the free encyclopedia - http://en.wikipedia.org/wiki/Boost_C%2B%2B_Libraries
- Highscore - The Boost C++ Libraries - http://en.highscore.de/cpp/boost/frontpage.html
- Μεταφερσιμότητα, Διομήδης Δ. Σπινέλλης - http://www.dmst.aueb.gr/dds/ism/pp/langport.htm
- POCO C++ Libraries - http://pocoproject.org/
- Fast Light Toolkit (FLTK) - http://www.fltk.org/
- The GTK+ Project - http://www.gtk.org/
- gtkmm Interfaces for GTK+ and GNOME - http://www.gtkmm.org/en/

- The Juce Library - http://www.rawmaterialsoftware.com/juce.php
- wxWidgets cross-platform GUI library - http://www.wxwidgets.org/
- Qt cross-platform application and UI framework - http://qt.nokia.com/products/
- Wt C++ web toolkit - http://www.webtoolkit.eu/wt
- MoSync cross-platform mobile application - http://www.mosync.com/

Παραρτήματα

Παράρτημα Α Παράδειγμα ελάχιστου C++ προγράμματος Hello World

Ένα πολύ βασικό C++ πρόγραμμα που εμφανίζει στην στάνταρ έξοδο “Hello world!”

```cpp
// helloworld.cxx

#include <iostream>
using namespace std;

int main()
{
   cout << "Hello, world!" << endl;
   return 0;
}
```

Παράρτημα Β Παράδειγμα μεταγλώττισης κατά περίπτωση

Παράδειγμα κώδικα C++ μεταγλώττισης κατά περίπτωση πλατφόρμας

```
// helloworld_conditional_compilation.cxx

#include <iostream>
using namespace std;

int main()
{
   cout << "Hello, world!" << endl;
#ifdef __APPLE__
  cout << "Compiled for MAC!" << endl;
#elif defined(WIN32)
  cout << "Compiled for WIN!" << endl;
#else
  cout << "Compiled for POSIX!" << endl;
#endif

  return 0;
}
```

Παράρτημα Γ Δημιουργία διεργασίας σε περιβάλλον POSIX

Παράδειγμα κώδικα C++, δημιουργίας διεργασίας σε περιβάλλον POSIX

```
#include <sys/types.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    pid_t pid;

    pid = fork();
    if (pid ==  0 ) {
        execl("/usr/bin/gedit", "gedit", NULL);
    }
    return(0);
}
```

Παράρτημα Δ Δημιουργία διεργασίας σε περιβάλλον Win32

Παράδειγμα κώδικα C++, δημιουργίας διεργασίας σε περιβάλλον Win32

```
#include <windows.h>

int main(int argc, char *argv[])
{
    STARTUPINFO si;
    PROCESS_INFORMATION pi;

    ZeroMemory( &si, sizeof(si) );
    si.cb = sizeof(si);

    ZeroMemory( &pi, sizeof(pi) );
    // Start the child process.
    if(!CreateProcess(NULL, "notepad", NULL, NULL,FALSE, 0, NULL, NULL, &si,&pi))
        return E_FAIL;
    else
        return S_OK;
}
```

Παράρτημα Ε Μεταφέρσιμη κλάση δημιουργίας διεργασιών

Παράδειγμα κώδικα C++, μεταφέρσιμης κλάσης δημιουργίας διεργασιών

```
// example: class constructor
#include <iostream>

#if defined(WIN32)
#include <windows.h>
#else
#include <sys/types.h>
#include <unistd.h>
#endif
using namespace std;

class CProcess {
#if defined(WIN32)
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
#else
    pid_t pid;
#endif

   char* cmd_;

  public:
    CProcess (char* cmd);
    int run ();
};

CProcess::CProcess (char* cmd) {
  cmd_ = cmd;
#if defined(WIN32)
  ZeroMemory( &si, sizeof(si) );
  si.cb = sizeof(si);
  ZeroMemory( &pi, sizeof(pi) );
#else
  pid = fork();
#endif
}

int CProcess::run () {
#if defined(WIN32)
```

```
    if(!CreateProcess(NULL, cmd_, NULL, NULL,FALSE, 0, NULL, NULL, &si,&pi))
   return E_FAIL;
    else
   return S_OK;
#else
    if (preturn(0);
    }else{
   return(1);
    }
#endif
}

int main (int argc, char *argv[]) {
  CProcess p1 (argv[1]);
return(0);
    }else{
   return(1);
    }
#endif
}

int main (int argc, char *argv[]) {
  CProcess p1 (argv[1]);

  p1.run();

  return 0;
}
  p1.run();

  return 0;
}id ==  0 ) {
        execl(cmd_, "", NULL);
   return(0);
    }else{
   return(1);
    }
#endif
}

int main (int argc, char *argv[]) {
  CProcess p1 (argv[1]);

  p1.run();

  return 0;
```

```
}
```

Παράρτημα ΣΤ Υλοποίηση κλάσης δημιουργίας διεργασιών με χρήση του Factory pattern

```
/*
 * Process.h
 *
 *  Created on: Feb 12, 2011
 *      Author: argy
 */

#ifndef PROCESS_H_
#define PROCESS_H_

#include "ProcessFactory.h"

class Process {
public:
    Process(char * cmd);
    virtual ~Process();
    int run();
private:
    ProcessImpl * m_processImpl;
    char * cmd_;
};

#endif /* PROCESS_H_ */
```

```
/*
 * Process.cxx
 *
 *  Created on: Feb 12, 2011
 *      Author: argy
 */

#include "Process.h"
#include "ProcessFactory.h"

Process::Process(char * cmd) {
    cmd_ = cmd;
    ProcessFactory * factory = ProcessFactory::GetProcessFactory();

    if (factory)
```

```
        m_processImpl = factory->createProcess(cmd_);


}

int Process::run() {
    m_processImpl->runProcess();
}

Process::~Process() {
    // TODO Auto-generated destructor stub
}
```

```
/*
 * ProcessFactory.h
 *
 *  Created on: Feb 12, 2011
 *      Author: argy
 */

#ifndef PROCESSFACTORY_H_
#define PROCESSFACTORY_H_

#include "ProcessImpl.h"

class ProcessFactory {
public:
    static ProcessFactory * GetProcessFactory();
    virtual ProcessImpl * createProcess(char * cmd);
};
#endif /* PROCESSFACTORY_H_ */
```

```
#include "windows/WindowsFactory.h"
#endif
#if defined(HAVE_MACOS)
#include "cocoa/cocoafactory.h"
#endif
#if defined(HAVE_LINUX)
#include "linux/LinuxFactory.h"
#endif

ProcessFactory * ProcessFactory::GetProcessFactory()
```

```
{
    static ProcessFactory *processFactory = 0;

    if (!processFactory)
        #if defined(HAVE_WIN32)
        processFactory = WindowsFactory::GetFactoryInstance();
        #endif
        #if defined(HAVE_MACOS)
        processFactory = CocoaFactory::GetFactoryInstance();
        #endif
        #if defined(HAVE_LINUX)
        processFactory = LinuxFactory::GetFactoryInstance();
        #endif

    return processFactory;

}

ProcessImpl * ProcessFactory::createProcess(char * cmd) {

}
```

```
/*
 * ProcessImpl.h
 *
 *  Created on: Feb 12, 2011
 *      Author: argy
 */

#ifndef PROCESSIMPL_H_
#define PROCESSIMPL_H_

class ProcessImpl {
public:
    ProcessImpl();
    ProcessImpl(char * cmd);
    virtual ~ProcessImpl();
    virtual int runProcess();

protected:
    char * cmd_;
};
```

```
#endif /* PROCESSIMPL_H_ */
```

```
/*
 * ProcessImpl.cpp
 *
 *  Created on: Feb 12, 2011
 *      Author: argy
 */

#include "ProcessImpl.h"

ProcessImpl::ProcessImpl() {
    // TODO Auto-generated constructor stub

}

ProcessImpl::ProcessImpl(char * cmd) {
    cmd_ = cmd;
}

ProcessImpl::~ProcessImpl() {
    // TODO Auto-generated destructor stub
}

int ProcessImpl::runProcess() {

}
```

```
/*
 * WindowsFactory.h
 *
 *  Created on: Feb 14, 2011
 *      Author: argy
 */

#ifndef WINDOWSFACTORY_H_
#define WINDOWSFACTORY_H_

#include "../ProcessFactory.h"
#include "../ProcessImpl.h"

class WindowsFactory: public ProcessFactory {
```

```
public:
    static WindowsFactory * GetFactoryInstance();
    virtual ~WindowsFactory();
    ProcessImpl * createProcess(char * cmd);
private:
    WindowsFactory();
};

#endif /* WINDOWSFACTORY_H_ */
```

```
/*
 * WindowsFactory.cxx
 *
 *  Created on: Feb 14, 2011
 *      Author: argy
 */

#include "WindowsFactory.h"
#include "WindowsProcessImpl.h"

WindowsFactory * WindowsFactory::GetFactoryInstance() {
    static WindowsFactory *factory = 0;

    if (!factory)
        factory = new WindowsFactory;
    return factory;

}

ProcessImpl * WindowsFactory::createProcess(char * cmd) {
    return new WindowsProcessImpl(cmd);
}

WindowsFactory::WindowsFactory() {
    // TODO Auto-generated constructor stub

}

WindowsFactory::~WindowsFactory() {
    // TODO Auto-generated destructor stub
}
```

```
/*
 * WindowsProcessImpl.h
 *
 *  Created on: Feb 14, 2011
 *      Author: argy
 */

#ifndef WINDOWSPROCESSIMPL_H_
#define WINDOWSPROCESSIMPL_H_

#include "../ProcessImpl.h"

#include <windows.h>

class WindowsProcessImpl: public ProcessImpl {
public:
    WindowsProcessImpl();
    WindowsProcessImpl(char * cmd);
    virtual ~WindowsProcessImpl();
    virtual int runProcess();

private:
  STARTUPINFO si;
  PROCESS_INFORMATION pi;
};

#endif /* WINDOWSPROCESSIMPL_H_ */
```

```
/*
 * WindowsProcessImpl.cxx
 *
 *  Created on: Feb 14, 2011
 *      Author: argy
 */

#include "WindowsProcessImpl.h"

WindowsProcessImpl::WindowsProcessImpl() {
    // TODO Auto-generated constructor stub

}

WindowsProcessImpl::WindowsProcessImpl(char * cmd) : ProcessImpl(cmd) {
```

```
    ZeroMemory( &si, sizeof(si) );
    si.cb = sizeof(si);
    ZeroMemory( &pi, sizeof(pi) );
}

WindowsProcessImpl::~WindowsProcessImpl() {
    // TODO Auto-generated destructor stub
}

int WindowsProcessImpl::runProcess() {
    if(!CreateProcess(NULL, cmd_, NULL, NULL,FALSE, 0, NULL, NULL, &si,&pi))
        return E_FAIL;
   else
    return S_OK;
}
```

```
/*
 * LinuxFactory.h
 *
 *  Created on: Feb 12, 2011
 *      Author: argy
 */

#ifndef LINUXFACTORY_H_
#define LINUXFACTORY_H_

#include "../ProcessFactory.h"
#include "../ProcessImpl.h"

#include <sys/types.h>
#include <unistd.h>

class LinuxFactory: public ProcessFactory {

public:
    static LinuxFactory * GetFactoryInstance();
    virtual ~LinuxFactory();
    ProcessImpl * createProcess(char * cmd);

private:
    LinuxFactory();
    pid_t pid;
```

```
};

#endif /* LINUXFACTORY_H_ */
```

```
/*
 * LinuxFactory.cxx
 *
 *  Created on: Feb 12, 2011
 *      Author: argy
 */

#include "LinuxFactory.h"
#include "LinuxProcessImpl.h"

LinuxFactory * LinuxFactory::GetFactoryInstance() {
    static LinuxFactory *factory = 0;

    if (!factory)
        factory = new LinuxFactory;
    return factory;

}

ProcessImpl * LinuxFactory::createProcess(char * cmd) {
    return new LinuxProcessImpl(cmd);
}

LinuxFactory::LinuxFactory() {
    // TODO Auto-generated constructor stub

}

LinuxFactory::~LinuxFactory() {
    // TODO Auto-generated destructor stub
}
```

```
/*
 * LinuxProcessImpl.h
 *
 *  Created on: Feb 12, 2011
 *      Author: argy
 */
```

```
#ifndef LINUXPROCESSIMPL_H_
#define LINUXPROCESSIMPL_H_

#include "../ProcessImpl.h"

#include <sys/types.h>
#include <unistd.h>

class LinuxProcessImpl: public ProcessImpl {
public:
    LinuxProcessImpl();
    LinuxProcessImpl(char * cmd);
    virtual ~LinuxProcessImpl();
    virtual int runProcess();

private:
    pid_t pid;
};

#endif /* LINUXPROCESSIMPL_H_ */
```

```
/*
 * LinuxProcessImpl.cxx
 *
 *  Created on: Feb 12, 2011
 *      Author: argy
 */

#include "LinuxProcessImpl.h"
#include <iostream>

LinuxProcessImpl::LinuxProcessImpl() {
    // TODO Auto-generated constructor stub

}

LinuxProcessImpl::LinuxProcessImpl(char * cmd) : ProcessImpl(cmd) {
    // TODO Auto-generated constructor stub
}

LinuxProcessImpl::~LinuxProcessImpl() {
    // TODO Auto-generated destructor stub
```

```
}

int LinuxProcessImpl::runProcess() {
    pid = fork();
  if (pid != 0 ) {
    execl(cmd_, "", NULL);
    std::cout << "CMD: " << cmd_ << std::endl;
    return(0);
  }
  else {
    return(1);
  }
}
```

```
/*
 * newProcess.cxx
 *
 *  Created on: Feb 12, 2011
 *      Author: argy
 */

#include "Process.h"
#include <iostream>

int main (int argc, char *argv[]) {
 Process p1 (argv[1]);

 p1.run();

 return 0;
}
```

Παράρτημα Ζ Παράδειγμα δημιουργίας νημάτων με τη Boost C++

```
#include <iostream>
#include <boost/thread.hpp>
#include <boost/date_time.hpp>

class SimpleThread {
   public:
      SimpleThread(unsigned id) {
         this->id = id;
      }

      void operator()() {
         std::cout << "Thread: " << id
               << " started!" << std::endl;

         boost::posix_time::seconds workTime(id);
         // Pretend to do something useful...
         boost::this_thread::sleep(workTime);
         std::cout << "Thread: " << id
               << " completed!" << std::endl;
      }

   private:
         unsigned id;
};

int main(int argc, char* argv[]) {

   std::cout << "main: startup" << std::endl;

   SimpleThread t1(4);
   SimpleThread t2(3);

   boost::thread workerThread2(t2);
   boost::thread workerThread1(t1);

        std::cout << "main: waiting for threads" << std::endl;
   workerThread1.join();
   workerThread2.join();

        std::cout << "main: done" << std::endl;

        return 0;

}
```

Παράρτημα Η Χρήση sockets με την POCO C++

```
#include "Poco/Net/ServerSocket.h"
#include "Poco/Net/StreamSocket.h"
#include "Poco/Net/SocketStream.h"
#include "Poco/Net/SocketAddress.h"
#include "Poco/StreamCopier.h"
#include <iostream>

int main(int argc, char** argv) {

   if (argc > 1) {
      std::cout <<  "Trying connect to " <<  argv[1] <<":8888..." << std::endl;
      Poco::Net::SocketAddress sa(argv[1], 8888);
      Poco::Net::StreamSocket socket(sa);
      Poco::Net::SocketStream str(socket);
      Poco::StreamCopier::copyStream(str, std::cout);
   } else {
      Poco::Net::ServerSocket srv(8888);
      std::cout <<  "Listening on port 8888..." << std::endl;
      for (;;) {
         Poco::Net::StreamSocket ss = srv.acceptConnection ();
         Poco::Net::SocketStream str(ss);
         str << "Hi from server\r\n" << std::flush;
      }
   }
   return 0;
}
```

Παράρτημα Θ Παράδειγμα χρήσης της FLTK

```
//
// "$Id: ask.cxx 8441 2011-02-18 08:52:48Z AlbrechtS $"
//
// Standard dialog test program for the Fast Light Tool Kit (FLTK).
//
// Demonstrates how to use readqueue to see if a button has been
// pushed, and to see if a window has been closed, thus avoiding
// the need to define callbacks.
//
// This also demonstrates how to trap attempts by the user to
// close the last window by overriding Fl::exit
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software; you can redistribute it and/or
// modify it under the terms of the GNU Library General Public
// License as published by the Free Software Foundation; either
// version 2 of the License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Library General Public License for more details.
//
// You should have received a copy of the GNU Library General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA.
//
// Please report all bugs and problems on the following page:
//
//     http://www.fltk.org/str.php
//

#include <stdio.h>
#include <string.h>
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Button.H>
#include <FL/Fl_Return_Button.H>

#include <FL/fl_ask.H>
#include <stdlib.h>
```

```
void update_input_text(Fl_Widget* o, const char *input) {
  if (input) {
    o->copy_label(input);
    o->redraw();
  }
}

void rename_me(Fl_Widget*o) {
  const char *input = fl_input("Input:", o->label());
  update_input_text(o, input);
}

void rename_me_pwd(Fl_Widget*o) {
  const char *input = fl_password("Input PWD:", o->label());
  update_input_text(o, input);
}

void window_callback(Fl_Widget*, void*) {
  int hotspot = fl_message_hotspot();
  fl_message_hotspot(0);
  fl_message_title("note: no hotspot set for this dialog");
  int rep = fl_choice("Are you sure you want to quit?",
            "Cancel", "Quit", "Dunno");
  fl_message_hotspot(hotspot);
  if (rep==1)
    exit(0);
  else if (rep==2)
    fl_message("Well, maybe you should know before we quit.");
}

int main(int argc, char **argv) {
  char buffer[128] = "Test text";
  char buffer2[128] = "MyPassword";

// this is a test to make sure automatic destructors work.  Pop up
// the question dialog several times and make sure it doesn't crash.
// fc: added more fl_ask common dialogs for test cases purposes.

  Fl_Double_Window window(200, 105);
  Fl_Return_Button b(20, 10, 160, 35, buffer); b.callback(rename_me);
  Fl_Button b2(20, 50, 160, 35, buffer2); b2.callback(rename_me_pwd);
  window.end();
  window.resizable(&b);
  window.show(argc, argv);
```

```
// Also we test to see if the exit callback works:
  window.callback(window_callback);

// set default message window title
  // fl_message_title_default("Default Window Title");

  return Fl::run();
}

//
// End of "$Id: ask.cxx 8441 2011-02-18 08:52:48Z AlbrechtS $".
```

Παράρτημα Ι Παράδειγμα χρήσης της GTK+

```
#include <gtk/gtk.h>

static GtkWidget *window = NULL;
static GtkWidget *entry1 = NULL;
static GtkWidget *entry2 = NULL;

static void
message_dialog_clicked (GtkButton *button,
          gpointer   user_data)
{
  GtkWidget *dialog;
  static gint i = 1;

  dialog = gtk_message_dialog_new (GTK_WINDOW (window),
                GTK_DIALOG_MODAL | GTK_DIALOG_DESTROY_WITH_PARENT,
                GTK_MESSAGE_INFO,
                GTK_BUTTONS_OK,
                "This message box has been popped up the following\n"
                "number of times:");
  gtk_message_dialog_format_secondary_text (GTK_MESSAGE_DIALOG (dialog),
                                            "%d", i);
  gtk_dialog_run (GTK_DIALOG (dialog));
  gtk_widget_destroy (dialog);
  i++;
}

static void
interactive_dialog_clicked (GtkButton *button,
              gpointer   user_data)
{
  GtkWidget *dialog;
  GtkWidget *hbox;
  GtkWidget *stock;
  GtkWidget *table;
  GtkWidget *local_entry1;
  GtkWidget *local_entry2;
  GtkWidget *label;
  gint response;

  dialog = gtk_dialog_new_with_buttons ("Interactive Dialog",
                GTK_WINDOW (window),
                GTK_DIALOG_MODAL| GTK_DIALOG_DESTROY_WITH_PARENT,
                GTK_STOCK_OK,
                GTK_RESPONSE_OK,
```

```
                                       "_Non-stock Button",
                                       GTK_RESPONSE_CANCEL,
                NULL);

  hbox = gtk_hbox_new (FALSE, 8);
  gtk_container_set_border_width (GTK_CONTAINER (hbox), 8);
  gtk_box_pack_start (GTK_BOX (GTK_DIALOG (dialog)->vbox), hbox, FALSE, FALSE,
0);

  stock = gtk_image_new_from_stock (GTK_STOCK_DIALOG_QUESTION,
GTK_ICON_SIZE_DIALOG);
  gtk_box_pack_start (GTK_BOX (hbox), stock, FALSE, FALSE, 0);

  table = gtk_table_new (2, 2, FALSE);
  gtk_table_set_row_spacings (GTK_TABLE (table), 4);
  gtk_table_set_col_spacings (GTK_TABLE (table), 4);
  gtk_box_pack_start (GTK_BOX (hbox), table, TRUE, TRUE, 0);
  label = gtk_label_new_with_mnemonic ("_Entry 1");
  gtk_table_attach_defaults (GTK_TABLE (table),
              label,
              0, 1, 0, 1);
  local_entry1 = gtk_entry_new ();
  gtk_entry_set_text (GTK_ENTRY (local_entry1), gtk_entry_get_text (GTK_ENTRY
(entry1)));
  gtk_table_attach_defaults (GTK_TABLE (table), local_entry1, 1, 2, 0, 1);
  gtk_label_set_mnemonic_widget (GTK_LABEL (label), local_entry1);

  label = gtk_label_new_with_mnemonic ("E_ntry 2");
  gtk_table_attach_defaults (GTK_TABLE (table),
              label,
              0, 1, 1, 2);

  local_entry2 = gtk_entry_new ();
  gtk_entry_set_text (GTK_ENTRY (local_entry2), gtk_entry_get_text (GTK_ENTRY
(entry2)));
  gtk_table_attach_defaults (GTK_TABLE (table), local_entry2, 1, 2, 1, 2);
  gtk_label_set_mnemonic_widget (GTK_LABEL (label), local_entry2);

  gtk_widget_show_all (hbox);
  response = gtk_dialog_run (GTK_DIALOG (dialog));

  if (response == GTK_RESPONSE_OK)
    {
      gtk_entry_set_text (GTK_ENTRY (entry1), gtk_entry_get_text (GTK_ENTRY
(local_entry1)));
      gtk_entry_set_text (GTK_ENTRY (entry2), gtk_entry_get_text (GTK_ENTRY
(local_entry2)));
```

```
    }

  gtk_widget_destroy (dialog);
}

GtkWidget *
do_dialog (GtkWidget *do_widget)
{
  GtkWidget *frame;
  GtkWidget *vbox;
  GtkWidget *vbox2;
  GtkWidget *hbox;
  GtkWidget *button;
  GtkWidget *table;
  GtkWidget *label;

  if (!window)
    {
      window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
      gtk_window_set_screen (GTK_WINDOW (window),
                 gtk_widget_get_screen (do_widget));
      gtk_window_set_title (GTK_WINDOW (window), "Dialogs");

      g_signal_connect (window, "destroy", G_CALLBACK (gtk_widget_destroyed),
&window);
      gtk_container_set_border_width (GTK_CONTAINER (window), 8);

      frame = gtk_frame_new ("Dialogs");
      gtk_container_add (GTK_CONTAINER (window), frame);

      vbox = gtk_vbox_new (FALSE, 8);
      gtk_container_set_border_width (GTK_CONTAINER (vbox), 8);
      gtk_container_add (GTK_CONTAINER (frame), vbox);

      /* Standard message dialog */
      hbox = gtk_hbox_new (FALSE, 8);
      gtk_box_pack_start (GTK_BOX (vbox), hbox, FALSE, FALSE, 0);
      button = gtk_button_new_with_mnemonic ("_Message Dialog");
      g_signal_connect (button, "clicked",
          G_CALLBACK (message_dialog_clicked), NULL);
      gtk_box_pack_start (GTK_BOX (hbox), button, FALSE, FALSE, 0);

      gtk_box_pack_start (GTK_BOX (vbox), gtk_hseparator_new (), FALSE, FALSE,
0);

      /* Interactive dialog*/
```

```
    hbox = gtk_hbox_new (FALSE, 8);
    gtk_box_pack_start (GTK_BOX (vbox), hbox, FALSE, FALSE, 0);
    vbox2 = gtk_vbox_new (FALSE, 0);

    button = gtk_button_new_with_mnemonic ("_Interactive Dialog");
    g_signal_connect (button, "clicked",
        G_CALLBACK (interactive_dialog_clicked), NULL);
    gtk_box_pack_start (GTK_BOX (hbox), vbox2, FALSE, FALSE, 0);
    gtk_box_pack_start (GTK_BOX (vbox2), button, FALSE, FALSE, 0);

    table = gtk_table_new (2, 2, FALSE);
    gtk_table_set_row_spacings (GTK_TABLE (table), 4);
    gtk_table_set_col_spacings (GTK_TABLE (table), 4);
    gtk_box_pack_start (GTK_BOX (hbox), table, FALSE, FALSE, 0);

    label = gtk_label_new_with_mnemonic ("_Entry 1");
    gtk_table_attach_defaults (GTK_TABLE (table),
             label,
             0, 1, 0, 1);

    entry1 = gtk_entry_new ();
    gtk_table_attach_defaults (GTK_TABLE (table), entry1, 1, 2, 0, 1);
    gtk_label_set_mnemonic_widget (GTK_LABEL (label), entry1);

    label = gtk_label_new_with_mnemonic ("E_ntry 2");

    gtk_table_attach_defaults (GTK_TABLE (table),
             label,
             0, 1, 1, 2);

    entry2 = gtk_entry_new ();
    gtk_table_attach_defaults (GTK_TABLE (table), entry2, 1, 2, 1, 2);
    gtk_label_set_mnemonic_widget (GTK_LABEL (label), entry2);
  }

if (!gtk_widget_get_visible (window))
  {
    gtk_widget_show_all (window);
  }
else
  {
    gtk_widget_destroy (window);
    window = NULL;
  }

return window;
```

```
}
```

Παράρτημα ΙΑ Παράδειγμα Hello World στο πλαίσιο Juce

```
/*
  ==============================================================================

  This is an automatically generated file created by the Jucer!

  Creation date:  1 May 2011 12:12:59pm

  Be careful when adding custom code to these files, as only the code within
  the "//[xyz]" and "//[/xyz]" sections will be retained when the file is loaded
  and re-saved.

  Jucer version: 1.12

  ------------------------------------------------------------------------------

  The Jucer is part of the JUCE library - "Jules' Utility Class Extensions"
  Copyright 2004-6 by Raw Material Software ltd.

  ==============================================================================
*/

//[Headers] You can add your own extra header files here...
//[/Headers]

#include "MainComponent.h"


//[MiscUserDefs] You can add your own user definitions and misc code here...
//[/MiscUserDefs]

//==============================================================================
MainComponent::MainComponent ()
    : helloWorldLabel (0),
      quitButton (0)
{
    addAndMakeVisible (helloWorldLabel = new Label (String::empty,
                                                    L"Hello World!"));
    helloWorldLabel->setFont (Font (40.0000f, Font::bold));
    helloWorldLabel->setJustificationType (Justification::centred);
    helloWorldLabel->setEditable (false, false, false);
    helloWorldLabel->setColour (Label::textColourId, Colours::black);
    helloWorldLabel->setColour (TextEditor::textColourId, Colours::black);
    helloWorldLabel->setColour (TextEditor::backgroundColourId, Colour (0x0));
```

```
    addAndMakeVisible (quitButton = new TextButton (String::empty));
    quitButton->setButtonText (L"Quit");
    quitButton->addListener (this);


    //[UserPreSize]
    //[/UserPreSize]

    setSize (600, 300);


    //[Constructor] You can add your own custom stuff here..
    //[/Constructor]
}

MainComponent::~MainComponent()
{
    //[Destructor_pre]. You can add your own custom destruction code here..
    //[/Destructor_pre]

    deleteAndZero (helloWorldLabel);
    deleteAndZero (quitButton);


    //[Destructor]. You can add your own custom destruction code here..
    //[/Destructor]
}

//==============================================================================
void MainComponent::paint (Graphics& g)
{
    //[UserPrePaint] Add your own custom painting code here..
    //[/UserPrePaint]

    g.fillAll (Colour (0xffc1d0ff));

    g.setColour (Colours::white);
    g.fillPath (internalPath1);
    g.setColour (Colour (0xff6f6f6f));
    g.strokePath (internalPath1, PathStrokeType (5.2000f));

    //[UserPaint] Add your own custom painting code here..
    //[/UserPaint]
}

void MainComponent::resized()
```

```
{
    helloWorldLabel->setBounds (152, 80, 296, 48);
    quitButton->setBounds (getWidth() - 176, getHeight() - 60, 120, 32);
    internalPath1.clear();
    internalPath1.startNewSubPath (136.0f, 80.0f);
    internalPath1.quadraticTo (176.0f, 24.0f, 328.0f, 32.0f);
    internalPath1.quadraticTo (472.0f, 40.0f, 472.0f, 104.0f);
    internalPath1.quadraticTo (472.0f, 192.0f, 232.0f, 176.0f);
    internalPath1.lineTo (184.0f, 216.0f);
    internalPath1.lineTo (200.0f, 168.0f);
    internalPath1.quadraticTo (96.0f, 136.0f, 136.0f, 80.0f);
    internalPath1.closeSubPath();

    //[UserResized] Add your own custom resize handling here..
    //[/UserResized]
}

void MainComponent::buttonClicked (Button* buttonThatWasClicked)
{
    //[UserbuttonClicked_Pre]
    //[/UserbuttonClicked_Pre]

    if (buttonThatWasClicked == quitButton)
    {
        //[UserButtonCode_quitButton] -- add your button handler code here..

        JUCEApplication::quit();

        //[/UserButtonCode_quitButton]
    }

    //[UserbuttonClicked_Post]
    //[/UserbuttonClicked_Post]
}



//[MiscUserCode] You can add your own definitions of your custom methods or any
other code here...
//[/MiscUserCode]


//==============================================================================
#if 0
/*  -- Jucer information section --
```

```
    This is where the Jucer puts all of its metadata, so don't change anything
in here!

BEGIN_JUCER_METADATA

<JUCER_COMPONENT documentType="Component" className="MainComponent"
componentName=""
                 parentClasses="public Component" constructorParams=""
variableInitialisers=""
                 snapPixels="8" snapActive="1" snapShown="1"
overlayOpacity="0.330000013"
                 fixedSize="1" initialWidth="600" initialHeight="300">
  <BACKGROUND backgroundColour="ffc1d0ff">
    <PATH pos="0 0 100 100" fill="solid: ffffffff" hasStroke="1"
stroke="5.19999981, mitered, butt"
          strokeColour="solid: ff6f6f6f" nonZeroWinding="1">s 136 80 q 176 24
328 32 q 472 40 472 104 q 472 192 232 176 l 184 216 l 200 168 q 96 136 136 80
x</PATH>
  </BACKGROUND>
  <LABEL name="" id="be4f6f2e5725a063" memberName="helloWorldLabel"
virtualName=""
         explicitFocusOrder="0" pos="152 80 296 48" textCol="ff000000"
         edTextCol="ff000000" edBkgCol="0" labelText="Hello World!"
editableSingleClick="0"
         editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default
font"
         fontsize="40" bold="1" italic="0" justification="36"/>
  <TEXTBUTTON name="" id="bcf4f7b0888effe5" memberName="quitButton"
virtualName=""
              explicitFocusOrder="0" pos="176R 60R 120 32" buttonText="Quit"
              connectedEdges="0" needsCallback="1" radioGroupId="0"/>
</JUCER_COMPONENT>

END_JUCER_METADATA
*/
#endif
/*
  ==============================================================================

   Demonstration "Hello World" application in JUCE
   Copyright 2008 by Julian Storer.

  ==============================================================================
*/

#include "../JuceLibraryCode/JuceHeader.h"
#include "MainComponent.h"
```

```
//==============================================================================
/**
    This is the top-level window that we'll pop up. Inside it, we'll create and
    show a component from the MainComponent.cpp file (you can open this file
using
    the Jucer to edit it).
*/
class HelloWorldWindow  : public DocumentWindow
{
public:

//==============================================================================
    HelloWorldWindow()
        : DocumentWindow ("JUCE Hello World!",
                          Colours::lightgrey,
                          DocumentWindow::allButtons,
                          true)
    {
        // Create an instance of our main content component, and add it to our
window..
        setContentOwned (new MainComponent(), true);

        // Centre the window on the screen
        centreWithSize (getWidth(), getHeight());

        // And show it!
        setVisible (true);
    }

    ~HelloWorldWindow()
    {
        // (the content component will be deleted automatically, so no need to
do it here)
    }


//==============================================================================
    void closeButtonPressed()
    {
        // When the user presses the close button, we'll tell the app to quit.
This
        // HelloWorldWindow object will be deleted by the
JUCEHelloWorldApplication class.
        JUCEApplication::quit();
    }
};
```

```
//==============================================================================
/** This is the application object that is started up when Juce starts. It
handles
    the initialisation and shutdown of the whole application.
*/
class JUCEHelloWorldApplication : public JUCEApplication
{
public:

//==============================================================================
    JUCEHelloWorldApplication()
    {
    }

    ~JUCEHelloWorldApplication()
    {
    }


//==============================================================================
    void initialise (const String& commandLine)
    {
        // For this demo, we'll just create the main window...
        helloWorldWindow = new HelloWorldWindow();

        /*  ..and now return, which will fall into to the main event
            dispatch loop, and this will run until something calls
            JUCEAppliction::quit().

            In this case, JUCEAppliction::quit() will be called by the
            hello world window being clicked.
        */
    }

    void shutdown()
    {
        // This method is where you should clear-up your app's resources..

        // The helloWorldWindow variable is a ScopedPointer, so setting it to a
null
        // pointer will delete the window.
        helloWorldWindow = 0;
    }


//==============================================================================
    const String getApplicationName()
```

```
    {
        return "Hello World for JUCE";
    }

    const String getApplicationVersion()
    {
        // The ProjectInfo::versionString value is automatically updated by the
Jucer, and
        // can be found in the JuceHeader.h file that it generates for our
project.
        return ProjectInfo::versionString;
    }

    bool moreThanOneInstanceAllowed()
    {
        return true;
    }

    void anotherInstanceStarted (const String& commandLine)
    {
    }

private:
    ScopedPointer<HelloWorldWindow> helloWorldWindow;
};


//==============================================================================
// This macro creates the application's main() function..
START_JUCE_APPLICATION (JUCEHelloWorldApplication)
```

Παράρτημα ΙΒ Παράδειγμα Hello World wxWidgets

```
/*
 * hworld.cpp
 */

#include "wx/wx.h"

class MyApp: public wxApp
{
    virtual bool OnInit();
};

class MyFrame: public wxFrame
{
public:

    MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size);

    void OnQuit(wxCommandEvent& event);
    void OnAbout(wxCommandEvent& event);

    DECLARE_EVENT_TABLE()
};

enum
{
    ID_Quit = 1,
    ID_About,
};

BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU(ID_Quit, MyFrame::OnQuit)
    EVT_MENU(ID_About, MyFrame::OnAbout)
END_EVENT_TABLE()

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    MyFrame *frame = new MyFrame( _("Hello World"), wxPoint(50, 50),
                                  wxSize(450,340) );
    frame->Show(true);
    SetTopWindow(frame);
    return true;
}
```

```
MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size)
: wxFrame( NULL, -1, title, pos, size )
{
    wxMenu *menuFile = new wxMenu;

    menuFile->Append( ID_About, _("&About...") );
    menuFile->AppendSeparator();
    menuFile->Append( ID_Quit, _("E&xit") );

    wxMenuBar *menuBar = new wxMenuBar;
    menuBar->Append( menuFile, _("&File") );

    SetMenuBar( menuBar );

    CreateStatusBar();
    SetStatusText( _("Welcome to wxWidgets!") );
}

void MyFrame::OnQuit(wxCommandEvent& WXUNUSED(event))
{
    Close(TRUE);
}

void MyFrame::OnAbout(wxCommandEvent& WXUNUSED(event))
{
    wxMessageBox( _("This is a wxWidgets Hello world sample"),
                  _("About Hello World"),
                  wxOK | wxICON_INFORMATION, this);
}
```

Παράρτημα ΙΓ Ενδεικτική Εφαρμογή Τρίλιζας σε Qt

```
#ifndef WINMAIN_H
#define WINMAIN_H

#include <QtGui/QMainWindow>
#include "ui_winmain.h"
#include "Board.h"
// Don't include "btnSquare.h", it's included in Board.h


class winMain : public QMainWindow
{
    Q_OBJECT
    public:
        // Game keeping essentials
        Board * gameBoard;
        bool xHasTurn;

        /*
         * Controls we need to find.
         */
        QRadioButton *radEasy, *radNormal, * radHard, * radImp, * radComputer;
        QLabel * whoseTurn, * taunt;

        // Construct / destruct
        winMain(QWidget *parent = 0, Qt::WFlags flags = 0);
        ~winMain();

        // Returns whether or not there's a winner.
        bool winner(bool isComp);


        void pressButton(btnSquare * which);

        /// <summary>
        /// This function heavily improves program flow by
        /// checking the moves from a single function, instead
        /// of just repeating a lot of code.
        /// </summary>
        /// <param name="checks">The array of checks.</param>
        /// <param name="isX">Tell the function whether to verify against X (true)
or O (false).</param>
        /// <returns></returns>
        btnSquare * checkMoves(int check[][3], bool checkX, int i);
        btnSquare * computerMove();
```

```
        virtual void keyPressEvent(QKeyEvent * e);

    public slots:
        // Event handlers
        void newGame();
        void aboutGame();
        void aboutQt();
        void btnPressed();

    private:
        Ui::winMainClass ui;
};

#endif // WINMAIN_H

#include <QPushButton>

class btnSquare : public QPushButton
{
    Q_OBJECT
    public:
        // Construct / destruct
        btnSquare();
        ~btnSquare();

        /*
            Setters
        */

        // Mark this square with an X
        void setX();
        // Mark this square with an O
        void setO();
        // Reset this square.
        void unset();

        /*
            Checks
        */

        // Returns whether or not the square has an X.
        bool isX();
        // Returns whether or not the square has an O.
        bool isO();
```

```
        bool autoCheck(bool isX);

        // State of the square. 0 is blank, 1 is X, 2 is O
        int state;

        QString style;
};
```

Παράρτημα ΙΔ Ενδεικτική εφαρμογή MoSync

```
/*
Copyright (C) 2011 MoSync AB

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License,
version 2, as published by the Free Software Foundation.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
MA 02110-1301, USA.
*/

/** @file main.cpp
*
* This application provides a very basic example of how to use MoSync's Moblet
* framework to detect key events and use timers. It displays the text "Hello
* Moblet!" The text can be moved around using the joystick or you can click on
* the screen and the text will jump to that position. The code is
* very well commented so that you can see what's happening at each step.
*
* @author The MoSync Team
*/

//Include the header files for MoSync Moblets so that we can
//access the Moblet library code from our application.
#include <MAUtil/Moblet.h>

//Define how many pixels will be traversed each timer tick.
#define SPEED 1

//Create the wrapper for the entire application. It is here that we will manage
//the application and handle events. To create the wrapper we make our own
//implementation of the MoSync Moblet base class. There can be only one running
//Moblet in an application. We also make our application wrapper inherit from
//MAUtil::TimerListener so that it can handle timer events.
class MyMoblet : public MAUtil::Moblet, MAUtil::TimerListener
{
```

```
//Define our new class's public methods.
public:

//First, the constructor, which we will call when we need an instance
//of MyMoblet. We set some initial values for the parameters mDirX, mDirY,
//mX and mY which we will be using to store the direction of movement and
//the current location. As they are member variables, they have an "m"
//prefix; their declaration can be found in the private section of the class.
MyMoblet() : mDirx(0), mDirY(0), mX(30), mY(40)
{
//Initialize the mText variable with our display string.
mText = "Hello Moblet!";

//Get the screen dimensions using the maGetScrSize syscall
//and save it in a local variable called "screenSize". As it is a local
//variable it has no "m" prefix, but we do need to declare its
//data type: MAExtent.
MAExtent screenSize = maGetScrSize();

//MAExtent is actually a special type of integer value that contains
//both the height and width of the screen. We can extract the dimensions
//using the macros EXTENT_X() and EXTENT_Y().
mScreenWidth = EXTENT_X(screenSize);
mScreenHeight = EXTENT_Y(screenSize);

//Call the draw() function to display the text on the screen. The draw()
//function is defined later in this class.
draw();
}

//Next, the method that gets called when a key is pressed. This is a method
//we have inherited from the Moblet base class, and here we will override
//that method with some processing of our own.
void keyPressEvent(int keyCode, int nativeCode)
{

//We'll use the C/C++ "switch" statement to select between different
//possible values of the keyCode that has been passed to our application
//by the runtime and record the movement in a direction variable.
switch(keyCode) {
case MAK_LEFT: mDirx = -1; break;
case MAK_RIGHT: mDirx = 1; break;
case MAK_UP: mDirY = -1; break;
case MAK_DOWN: mDirY = 1; break;

//If the key pressed is the soft-right key OR the "0" key, just
```

```
//exit the application.
case MAK_SOFTRIGHT:
case MAK_0:
case MAK_BACK:
maExit(0);

//If it is any other key, do nothing.
default:
return;
}

//Add a timer that, every 20 milliseconds, generates a timer event
//and send it to the TimerListener. Remember that MyMoblet is itself the
//TimerListener, hence it refers to itself ("this"). The timer will run
//indefinitely (-1) until it is removed in response to a keyReleaseEvent.
addTimer(this, 20, -1);
}

//Now, the method that gets called when a key is released which we will use
//to reset our movement and location parameters, and remove the timer.
void keyReleaseEvent(int keyCode, int nativeCode)
{

//We only need to reset the direction if it is the current direction.
switch(keyCode) {
case MAK_LEFT: if(mDirx == -1) mDirx = 0; break;
case MAK_RIGHT: if(mDirx ==  1) mDirx = 0; break;
case MAK_UP: if(mDirY == -1) mDirY = 0; break;
case MAK_DOWN: if(mDirY ==  1) mDirY = 0; break;
default:
return;
}

//Here we remove the timer created by the keyPressEvent if nothing is
//still moving.
if(mDirx == 0 && mDirY == 0)
removeTimer(this);
}

//Next, the method that gets called when a pointer is pressed on the screen.
//The MoSync runtime supplies the pointer coordinates in the parameter "p",
//from which we can extract the individual x and y coordinates.
virtual void pointerPressEvent(MAPoint2d p)
{
mX = p.x;
mY = p.y;
```

```
//Use the draw function to redraw the screen.
draw();
}

//Similarly for a pointer movement to another location on the screen.
virtual void pointerMoveEvent(MAPoint2d p)
{
mX = p.x;
mY = p.y;
draw();
}

//We don't need to do anything when the pointer is lifted.
virtual void pointerReleaseEvent(MAPoint2d p)
{
}

//Now the function that will draws the background and our text. It is
//made as a separate function because both MyMoblet's constructor and its
//timer callback need to call it.
void draw()
{
//We use a set of syscalls to do the drawing. First we set the background
//colour and paint the screen. Then we reset the drawing colour and draw
//the text.
maSetColor(0);
maFillRect(0, 0, mScreenWidth, mScreenHeight);
maSetColor(0xffffff);
maDrawText(mX, mY, mText);

//We've been painting and drawing to the backbuffer, so now we need to
//show our work on the screen.
maUpdateScreen();
}

//Finally in our MyMoblet class we define the Timer callback. This is the
//function that gets called whenever our timer (see addTimer, above)
//generates an event. While the key is pressed, the timer is sending events
//50 times per second, thus we are redrawing the screen 50 frames-per-second.
void runTimerEvent()
{
//Get the dimensions of our text using the maTextScrSize syscall and
//extract the width and height in pixels using the EXTENT_X() and
//EXTENT_Y() macros.
MAExtent stringSize = maGetTextSize(mText);
```

```
int stringSizeX = EXTENT_X(stringSize);
int stringSizeY = EXTENT_Y(stringSize);

//Check if the text location is inside the boundaries of the screen,
//and if not reset it so that it is.
if((mX + stringSizeX) > mScreenWidth)  mX = mScreenWidth - stringSizeX;
if((mY + stringSizeY) > mScreenHeight) mY = mScreenHeight - stringSizeY;
if(mX < 0) mX = 0;
if(mY < 0) mY = 0;

//Call the function that draws the background and our text string.
draw();

//Record the new location of our text string.
mX += mDirx * SPEED;
mY += mDirY * SPEED;
}

//Define our new class's private methods.
private:
//Here we declare the data types of our member variables so that the
//constructor, event handlers, etc. can work with them.
const char *mText;
int mDirx, mDirY;
int mX, mY;
int mScreenWidth, mScreenHeight;
};

//Now we get to the entry point for the application - the place where
//processing starts.
extern "C" int MAMain()
{
//Create the instance of MyMoblet.
MyMoblet myMoblet;

//Run the Moblet to start the application.
MyMoblet::run(&myMoblet);

//MyMoblet will run until it is closed by the user pressing key 0. When
//it's closed we end our program in a well-behaved way by returning zero.
return 0;
}
```