ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

XAIL: Μία Επεκτάσιμη Γλώσσα Για Διεπαφές Χρήστη

Γιαν Ντζικ

Κυριάκος Λιακόπουλος

Επιβλέποντες: **Ιζαμπώ Καράλη**, Επίκουρη Καθηγήτρια ΕΚΠΑ
Παναγιώτης Σταματόπουλος, Επίκουρος Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2008

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

XAIL: Μία Επεκτάσιμη Γλώσσα Για Διεπαφές Χρήστη

Γιαν Ντζικ

A.M.: 1115200300140

Κυριάκος Λιακόπουλος

A.M.: 1115200300094

ΕΠΙΒΛΕΠΟΝΤΕΣ:

Ιζαμπώ Καράλη, Επίκουρη Καθηγήτρια ΕΚΠΑ

Παναγιώτης Σταματόπουλος, Επίκουρος Καθηγητής ΕΚΠΑ

Περίληψη

Στην εργασία αυτή προτείνουμε μία νέα XML Γλώσσα Γραφικών Διεπαφών Χρήστη, τη XAIL, με μια διαφορετική προσέγγιση για το χειρισμό γεγονότων και τη διασύνδεση της διεπαφής με τις λειτουργίες μιας εφαρμογής. Οι XML Γλώσσες Γραφικών Διεπαφών Χρήστη, όπως η XAML, η XUL και η GladeXML, είναι αρκετά δημοφιλείς σήμερα γιατί προσφέρουν μεγαλύτερες ευκολίες ανάπτυξης και περισσότερες δυνατότητες σχεδίασης. Όμως, αυτές οι γλώσσες είναι στενά συνδεδεμένες με συγκεκριμένες εργαλειοθήκες ανάπτυξης εφαρμογών και δεν αποδεσμεύουν πλήρως τη γραφική διεπαφή (λογική παρουσίασης), από την υλοποίηση των λειτουργιών της (επιχειρησιακή λογική). Η γλώσσα που προτείνουμε σε αυτή την εργασία επιλύει αυτό το πρόβλημα με την υιοθέτηση δηλωτικής περιγραφής των ενεργειών χειρισμού γεγονότων, που χρησιμοποιούν ένα δυναμικό, κατανεμημένο σύστημα συστατικών λογισμικού, το οποίο δημιουργούμε για τις ανάγκες της γλώσσας. Με αυτό τον τρόπο, δημιουργούμε έναν πρωτότυπο τρόπο σχεδίασης και ανάπτυξης γραφικών εφαρμογών, στον οποίο η ανάπτυξη μιας γραφικής διεπαφής πραγματοποιείται ανεξάρτητα από την ανάπτυξη της λογικής της εφαρμογής.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: XML Γλώσσες Γραφικών Διεπαφών

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: XML, γραφικές διεπαφές χρήστη, επικοινωνία - ανθρώπου μηχανής, διαχωρισμός λογικής, συστατικά λογισμικού, χειρισμός γεγονότων

Abstract

In this work we propose a new XML User Interface Language, called XAIL, with a different approach to event handling and interface-functionality connection. XML User Interface Languages, such as XAML, XUL and GladeXML, are widely popular today because of the ease in development and design features they provide. However, these languages are tightly coupled with particular development toolkits and fail to decouple the user interface (presentation logic) from the implementation (business logic) completely. The language we propose, solves this issue by using declarative event handling descriptions that use a dynamic, distributed component system, which was created specifically for the language's needs. In this way, we introduce an original paradigm for graphical application design and development, where the process of graphical user interface development is independent from the application logic development.

SUBJECT AREA: XML User Interface Languages

KEYWORDS: XML, graphical user interfaces, human - computer interaction, logic separation, software components, event handling

Ευχαριστίες

Το θέμα αυτής της εργασίας ξεκίνησε πριν δύο χρόνια ως μια απλή ιδέα. Μέσα σε αυτά τα δύο χρόνια, η ιδέα αυτή, ζυμώθηκε, ωρίμασε, έγινε το θέμα αυτής της εργασίας και υλοποιήθηκε. Όλη αυτή η διαδρομή δε θα ήταν δυνατή χωρίς τη συνεργασία, τη συμβολή και τη βοήθεια διαφόρων προσώπων και για αυτό θα ήθελα να τους αποδώσω τις θερμότερες ευχαριστίες. Αρχικά, θα ήθελα να ευχαριστήσω το φίλο, συνεργάτη και συνάδελφο Κυριάκο Λιακόπουλο που δέχτηκε να συνεργαστεί μαζί μου ώστε να μετατρέψουμε εκείνη την αρχική ιδέα σε πράξη. Χωρίς τις ιδέες του, τη δουλειά και την αφοσίωσή του, αυτή η εργασία δε θα μπορούσε να έχει πραγματοποιηθεί. Φυσικά, η εργασία αυτή δε θα μπορούσε να πραγματοποιηθεί χωρίς την υποστήριξη των καθηγητών μου Ιζαμπώ Καράλη και Παναγιώτη Σταματόπουλο, που, καταρχήν δέχτηκαν την ιδέα για το θέμα και στη συνέχεια, με τις συμβουλές και την καθοδήγησή τους συνέβαλαν καθοριστικά στην ολοκλήρωσή της. Θα ήθελα, επίσης, να ευχαριστήσω τους τις εταιρίες Empneusis Internet Services και Nessos Πληροφορική που μου έδωσαν τη δυνατότητα να εργαστώ, να μάθω και να χρησιμοποιήσω τις εγκαταστάσεις τους για τις ανάγκες τις εργασίας, όποτε αυτό χρειάστηκε. Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου, που έδωσε τη δυνατότητα να ακολουθήσω το όνειρό μου και χωρίς αυτή δε θα ήμουν ο άνθρωπος που είμαι σήμερα.

Γιαν Ντζικ

Αυτή η πτυχιακή είναι αποτέλεσμα πολλών ωρών εργασίας και περισυλλογής. Αλλά, η συνεισφορά εκ μέρους μου δεν θα ήταν δυνατή χωρίς την άμεση ή έμμεση αρωγή αρκετών προσώπων. Για αυτό, θα ήθελα να ευχαριστήσω αρχικά τους γονείς μου, γιατί μου έδωσαν την δυνατότητα να σπουδάσω και μου έδωσαν την κατάλληλη παιδεία ώστε να το πετύχω. Τον αδερφό και την αδερφή μου, γιατί με ανέχονταν τις περιόδους που έπρεπε να συγκεντρωθώ στις σπουδές μου. Τους φίλους μου, για όσες φορές τους αρνήθηκα κάτι ώστε να συνεχίσω τη δουλειά πάνω στη πτυχιακή. Συγκεκριμένα, τον φίλο και συνάδελφο, Γιαν

Ντζικ, για την άρτια συνεργασία που είχαμε κατά την συγγραφή αυτής της πτυχιακής, τις εμπνευσμένες ιδέες του και όσα έμαθα από αυτόν. Τελικά ευχαριστώ τους καθηγητές μου Ιζαμπώ Καράλη και Παναγιώτη Σταματόπουλο για την απεριόριστη υπομονή και τις συμβουλές τους, και που έπαιξαν καταλυτικό ρόλο στην ολοκλήρωση αυτής της πτυχιακής εργασίας.

Κυριάκος Λιακόπουλος

Περιεχόμενα

Κατάλογος Σχημάτων

Πρόλογος

Η ενασχόλησή μου με διεπαφές χρήστη ξεκίνησε, ουσιαστικά το 2002, όταν εργαζόμουν στην Empneusis αναπτύσσοντας εφαρμογές παγκόσμιου ιστού. Από νωρίς με προβλημάτισε ο καλύτερος τρόπος με τον οποίο θα μπορεί να διαχωριστεί η παρουσίαση μιας εφαρμογής, δηλαδή αυτά που φαίνονται στο φυλλομετρητή, με τη λογική της, δηλαδή των κώδικα στον εξυπηρετητή. Ήρθα σε επαφή με διάφορες τεχνικές στο χώρο εργασίας και στην πορεία ανακάλυψα και άλλες. Το επίπεδο διαχωρισμού και η ευκολία επίτευξής του ήταν ικανοποιητικά.

Όταν άρχισα να ασχολούμαι με γραφικές διεπαφές για εφαρμογές επιφάνειας εργασίας, προς μεγάλη μου απογοήτευση, διαπίστωσα πως τα πράγματα δεν ήταν τόσο εύκολα. Παρατήρησα, πως ακόμα κι αν χρησιμοποιούσα εργαλεία σχεδίασης διεπαφών όπως το Glade, ή το Form Designer του Microsoft Visual Studio, οι συχνές αλλαγές που έκανα στην γραφική διεπαφή δυσκόλευαν πολύ την ανάπτυξη μιας εφαρμογής καθώς με κάθε αλλαγή έπρεπε να αλλάξουν αρκετά πράγματα στον κώδικα. Η εφαρμογή μοτίβων σχεδίασης όπως το MVC δεν έλυνε το πρόβλημα ικανοποιητικά.

Κάποια στιγμή το φθινόπωρο του 2006, βρέθηκα στο χώρο της Microsoft Hellas σε μια παρουσίαση για το WPF στο Microsoft .NET Framework 3.0 όπου ήρθα για πρώτη φορά σε επαφή με τη γλώσσα XAML. Παρόλο που είχα δει στο παρελθόν XML γλώσσες για γραφικές διεπαφές, όπως η XUL, η XAML με ενθουσίασε για τις δυνατότητες που προσέφερε. Παράλληλα όμως

με απογοήτευσε, καθώς παρατήρησα πως τα ίδια προβλήματα εξακολουθούν να ισχύουν και σε αυτή. Τότε, γεννήθηκε η ιδέα για τη δημιουργία μιας νέας XML γλώσσας για γραφικές διεπαφές χρήστη, απαλλαγμένη από το πρόβλημα του διαχωρισμού παρουσίασης και λογικής.

Λίγο καιρό αργότερα, συζητούσαμε με τον Κυριάκο για το θέμα αυτό. Είχε συναντήσει και ο ίδιος παρόμοια προβλήματα στις δικές του εμπειρίες με γραφικές διεπαφές και η ιδέα για μια νέα XML γλώσσα του άρεσε. Είχαμε συνεργαστεί με τον Κυριάκο σε διάφορες άλλες περιπτώσεις στο παρελθόν και γνωρίζαμε ο ένας τις ικανότητες και δυνατότητες του άλλου. Αποφασίσαμε, να υλοποιήσουμε αυτή την ιδέα από κοινού, ως πτυχιακή εργασία. Οι καθηγητές μας Παναγιώτης Σταματόπουλος και Ιζαμπώ Καράλη αναγνώρισαν την αξία της ιδέας μας, δέχτηκαν να γίνουν οι επιβλέποντες της εργασίας και έτσι η δουλειά ξεκίνησε. Η καθοδήγηση και οι πολύτιμες συμβουλές τους ήταν καθοριστικές.

Οι δυσκολίες που συναντήσαμε ήταν πολλές. Η πρώτη είχε να κάνει με την έρευνα των κατάλληλων τεχνολογιών που θα μπορούσαν να βοηθήσουν στην υλοποίηση της γλώσσας μας καθώς και της εξοικείωσης με αυτές, καθώς πολλές, όπως το CORBA, δεν τις είχαμε ξαναδεί. Στη συνέχεια, έπρεπε να υλοποιήσουμε τις σκέψεις μας. Ξεκινήσαμε με την υλοποίηση του μοντέλου συστατικών της γλώσσας μας βασιζόμενοι στο CORBA. Το στάδιο αυτό κράτησε πάνω από μισό χρόνο και αποδείχθηκε το δυσκολότερο. Έπειτα σχεδιάσαμε το συντακτικό και τις λειτουργίες της γλώσσας μας, ορίσαμε το όνομά της ως XAIL (eXtensible Application Interface Language) και τέλος, υλοποιήσαμε το διερμηνευτή της.

Η συνολική προσπάθεια ήταν δύσκολη και χρειάστηκε πολλές θυσίες. Το αποτέλεσμα είναι καλύτερο από αυτό που αρχικά περιμέναμε και είμαστε ιδιαίτερα υπερήφανοι για αυτό. Παρόλο που μετά την ολοκλήρωση αυτής της εργασίας οι δρόμοι μας θα χωρίσουν, δε θα σταματήσουμε εδώ. Πιστεύουμε στην αξία της ιδέας μας και θα συνεχίσουμε να εργαζόμαστε πάνω της με βάση αυτή την εργασία.

Γιαν Ντζικ

Κεφάλαιο 1

Εισαγωγή

Η εξέλιξη στην τεχνολογία των συστημάτων γραφικών και των βιβλιοθηκών ανάπτυξης γραφικών εφαρμογών δίνει νέες δυνατότητες στον τομέα σχεδίασης και ανάπτυξης εφαρμογών με γραφικές διεπαφές χρήστη. Δίνεται πια η δυνατότητα εκμετάλλευσης δυνατοτήτων, όπως η επιτάχυνση υλικού, η χρήση περίτεχνων οπτικών εφέ, όπως η διαφάνεια, τα κινούμενα γραφικά (animation), ακόμα και χρήση της τρίτης διάστασης[1]. Αναφερόμαστε στις εφαρμογές που αξιοποιούν αυτές τις νέες δυνατότητες ως Πλούσιες Εφαρμογές Επιφάνειας Εργασίας (RDA - Rich Desktop Applications) και Πλούσιες Εφαρμογές Διαδικτύου (RIA - Rich Internet Applications). Οι εφαρμογές αυτές προσφέρουν στο χρήστη μια πιο πλούσια εμπειρία, μέσω πιο ελκυστικών και κατανοητών[1] γραφικών διεπαφών.

Οι συνεχώς αυξανόμενες δυνατότητες των συστημάτων ανάπτυξης γραφικών εφαρμογών αλλά και η απαίτηση για Γρήγορη Ανάπτυξη Εφαρμογών - (RAD - Rapid Application Development) - δημιουργεί την ανάγκη εφαρμογής τεχνικών διαχωρισμού της διαδικασίας σχεδίασης της γραφικής διεπαφής μιας εφαρμογής και της ανάπτυξης, του προγραμματισμού, των λειτουργιών της[2]. Το πρόβλημα αυτό αντιμετωπίζεται σήμερα με διάφορα μέσα, όπως η χρήση

[1]Φυσικά αν ο σχεδιαστής μιας τέτοιας εφαρμογής αξιοποιεί με σωστό τρόπο τις νέες δυνατότητες των γραφικών συστημάτων.

εργαλείων σχεδίασης[3] που παράγουν τον κώδικα και η χρήση μοτίβων σχεδίασης (design patterns) λογισμικού, όπως το Μοντέλο - Όψη - Ελεγκτής (MVC - Model View Controller)[4]. Μετά την εμφάνιση της Επεκτάσιμης Γλώσσας Σήμανσης (XML - eXtensible Markup Language) ως γλώσσας περιγραφής δεδομένων και δεντρικών δομών, έχουν αναπτυχθεί XML γλώσσες για γραφικές διεπαφές[5, 6, 7, 8] που προσπαθούν να αποσυνδέσουν τη διαδικασία σχεδίασης από τη διαδικασία ανάπτυξης ακόμη περισσότερο.

Όμως οι XML γλώσσες μόνο εν μέρει καταφέρνουν να αποσυνδέσουν τις δύο διαδικασίες επειδή δεσμεύουν το περιβάλλον ανάπτυξης που μπορεί να χρησιμοποιηθεί (πρέπει να είναι το ίδιο με αυτό για το οποίο δημιουργήθηκε η γλώσσα) και στηρίζονται στη συγγραφή διαδικαστικού κώδικα χειριστών γεγονότων για την τροποποίηση της γραφικής διεπαφής[9, 10]. Για να επιλύσουμε αυτά τα θέματα δημιουργούμε μια καινούργια XML γλώσσα που θα εκφράζει τη λογική παρουσίασης μιας εφαρμογής ανεξάρτητα από την επιχειρησιακή λογική της. Για να το πετύχει, η λογική παρουσίασης ορίζεται πλήρως μέσω της γλώσσας με δηλωτική περιγραφή των ενεργειών που θα εκτελούνται κατά την εμφάνιση ενός γεγονότος καθώς και ορισμό των λειτουργιών της επιχειρησιακής λογικής που θα χρησιμοποιηθούν. Οι λειτουργίες της επιχειρησιακής λογικής οργανώνονται σε συστατικά λογισμικού ανεξάρτητα μεταξύ τους αλλά και από τη λογική παρουσίασης.

Στα πρώτα δύο κεφάλαια εξετάζουμε την κατάσταση της τεχνολογίας (State of the Art) σχετικά με τις Γραφικές Διεπαφές Χρήστη, τις XML γλώσσες περιγραφής γραφικών διεπαφών και με ποιούς τρόπους προσπαθούν να διαχωρίσουν τη λογική παρουσίασης με την επιχειρησιακή λογική. Συγκεκριμένα, στο κεφάλαιο 2 ορίζουμε έννοιες τις οποίες διαπραγματευόμαστε σε ολόκληρο το κείμενο όπως Γραφικές Διεπαφές Χρήστη, Γραφικά Στοιχεία, Λογική Παρουσίασης και Επιχειρησιακή Λογική κ.α. Στο ίδιο κεφάλαιο αποδεικνύουμε την ανάγκη διαχωρισμού της λογικής παρουσίασης από την επιχειρησιακή λογική και εξετάζουμε πως αυτό γίνεται στις διάφορες εργαλειοθήκες ανάπτυξης γραφικών διεπαφών. Στο κεφάλαιο 3 εξετάζουμε τις XML γλώσσες γραφικών

διεπαφών και περιγράφουμε τα χαρακτηριστικά τους.

Στο κεφάλαιο 4 αναδεικνύουμε το πρόβλημα των σημερινών γλωσσών και θέτουμε τις προδιαγραφές που το επιλύουν. Δείχνουμε πως οι σημερινές XML γλώσσες περιγραφής διεπαφών αποτυγχάνουν να αποσυνδέσουν πλήρως τη λογική παρουσίασης από την επιχειρησιακή λογική και ορίζουμε τις προδιαγραφές που θα πρέπει να πληρεί το σύστημα μιας XML γλώσσας για να επιλυθεί αυτό το πρόβλημα.

Στη συνέχεια, ορίζουμε τη γλώσσα XAIL ως μια γλώσσα που ικανοποιεί τις προδιαγραφές που έχουμε θέσει και επιλύει το πρόβλημα τις αποσύνδεσης των δύο λογικών. Συγκεκριμένα, στο κεφάλαιο 5 περιγράφουμε τα χαρακτηριστικά της γλώσσας και την αρχιτεκτονική του συστήματός της. Επιπλέον, περιγράφουμε τα στοιχεία της γλώσσας και δείχνουμε μέσα από παραδείγματα πως αυτά χρησιμοποιούνται. Στο τέλος του κεφαλαίου δείχνουμε πως χρησιμοποιείται η επιχειρησιακή λογική στη γλώσσα για να συνδεθεί με τη λογική παρουσίασης.

Συνεχίζουμε εξηγώντας την οργάνωση της επιχειρησιακής λογικής σε συστατικά λογισμικού κάτω από ένα μοντέλο συστατικών και περιγράφουμε το σύστημα που είναι υπεύθυνο για τη διαχείριση αυτών. Συγκεκριμένα, στο κεφάλαιο 6 περιγράφουμε την πρότυπη τεχνολογία στην οποία βασίζουμε το μοντέλο συστατικών που χρησιμοποιούμε στη γλώσσα XAIL. Στο κεφάλαιο 7 περιγράφουμε αυτό το μοντέλο και το σύστημα που είναι υπεύθυνο για αυτό και για τη διαχείριση των συστατικών.

Ολοκληρώνουμε την περιγραφή της γλώσσας μας και του συστήματός της με ένα παράδειγμα. Στο κεφάλαιο 8 αναπτύσσουμε μια εφαρμογή στο σύστημά μας. Ορίζουμε τη μορφή της επιχειρησιακής λογικής, τα συστατικά, και ορίζουμε τη λογική παρουσίασης στη γλώσσα XAIL. Τέλος στο παράρτημα Α′ ορίζουμε τους συντακτικούς κανόνες της γλώσσας με έναν ορισμό τύπου εγγράφου (DTD) και στο παράρτημα Β′ ορίζουμε τη γραμματική των υπολογίσημων εκφράσεων που χρησιμοποιούνται στη γλώσσα XAIL.

Κεφάλαιο 2

Γραφικές Διεπαφές Χρήστη

Σε αυτό το κεφάλαιο θα θέσουμε τα θεμέλια για τα επόμενα κεφάλαια. Θα ορίσουμε τι είναι οι Γραφικές Διεπαφές Χρήστη και θα κάνουμε μια σύντομη ιστορική αναδρομή της εξέλιξής τους. Στη συνέχεια, θα αναφερθούμε στις διαδικασίες σχεδίασης και ανάπτυξης εφαρμογών με τέτοιες διεπαφές και θα αναδείξουμε γιατί είναι σημαντικός ο διαχωρισμός των διαδικασιών σχεδίασης και ανάπτυξης της ίδιας της γραφικής διεπαφής από τις λειτουργίες της εφαρμογής που θέλει να παρουσιάσει. Τέλος, θα αναφερθούμε σε τεχνικές που χρησιμοποιούνται στην τεχνολογία λογισμικού προς αυτή την κατεύθυνση. Οι έννοιες που αναπτύσσουμε εδώ είναι κρίσιμες, καθώς ουσιαστικά σε αυτό και σε επόμενα κεφάλαια αναπτύσσεται η κατάσταση της τεχνολογίας σήμερα στη σχεδίαση και ανάπτυξη εφαρμογών με γραφικές διεπαφές χρήστη.

2.1 Ορισμός

Μία Γραφική Διεπαφή Χρήστη[1] (GUI - Graphical User Interface) είναι ένα είδος διεπαφής ανθρώπου-μηχανής, που επιτρέπει στους χρήστες του Ηλεκτρονικού Υπολογιστή - Η/Υ - να αλληλεπιδρούν με αυτόν και τις συσκευές που

[1] Από εδώ και πέρα, όταν δεν υπάρχει κίνδυνος σύγχυσης, θα αναφερόμαστε στις Γραφικές Διεπαφές Χρήστη ως Διεπαφές Χρήστη ή ως Γραφικές Διεπαφές ή πιο απλά ως Διεπαφές.

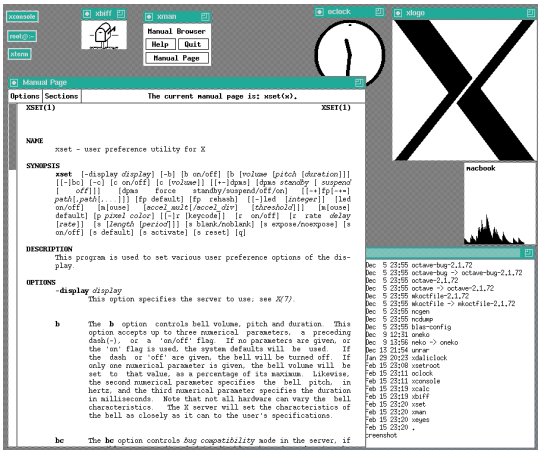
ελέγχει, χρησιμοποιώντας γραφικά, εικόνες και κείμενο για να παρουσιάσει τις πληροφορίες και τις διαθέσιμες ενέργειες προς το χρήστη[11, 1]. Συνήθως, οι ενέργειες αυτές πραγματοποιούνται με τροποποίηση των Γραφικών Στοιχείων (widgets) που αποτελούν τη διεπαφή.

2.2 Ιστορία

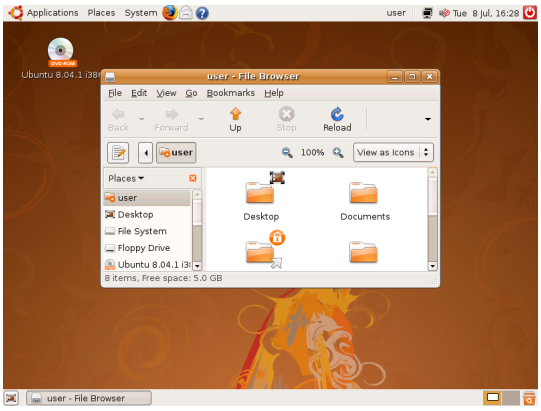
Ο πρόδρομος των Γραφικών Διεπαφών Χρήστη (GUI - Graphical User Interfaces) δημιουργήθηκε από ερευνητές του Stanford Research Institute[1, 12]. Ανέπτυξαν τη χρήση υπερσυνδέσμων κειμένου, μέσω ενός ποντικιού, για το On-Line System (NLS). Η έννοια των υπερσυνδέσμων επεκτάθηκε, πέρα από το κείμενο, στα γραφικά από ερευνητές του Xerox PARC που χρησιμποίησαν μία Γραφική Διεπαφή Χρήστη ως την κύρια διεπαφή του υπολογιστή Xerox Alto. Οι περισσότερες σύγχρονες γενικού σκοπού Γραφικές Διεπαφές Χρήστη βασίζονται σε αυτό το σύστημα[12]. Αυτό έχει ως αποτέλεσμα να αποκαλούν τέτοιου είδους διεπαφές PARC User Interface (PUI).

Ένα PARC User Interface αποτελείται από Γραφικά Στοιχεία - widgets - όπως παράθυρα, μενού, κουμπιά επιλογής και εικονίδια. Ένα PARC User Interface χρησιμοποιεί εκτός από πληκτρολόγιο και μία συσκευή δείκτη (pointing device) όπως το ποντίκι. Συνήθως δίνεται έμφαση σε αυτά τα χαρακτηριστικά της διεπαφής μέσω του όρου Παράθυρα Εικονίδια Επιλογείς και Δείκτες - ΠΕΕΔ (WIMP - Windows, Icons, Menus and Pointing Devices)[11, 1, 13].

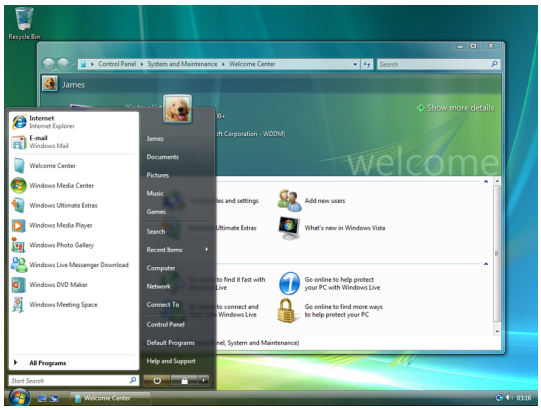
Ύστερα από το PARC το πρώτο λειτουργικό μοντέλο βασισμένο σε Γραφικές Διεπαφές Χρήστη ήταν αυτό του Apple Lisa, με τη Apple να αναπτύσσει ύστερα τον πιο επιτυχή Macintosh με το γραφικό περιβάλλον Macintosh System[12]. Οι Διεπαφές Χρήστη που είναι πιο γνωστές σήμερα στον κόσμο είναι αυτές των λειτουργικών Microsoft Windows, Mac OS X, και του συστήματος X Windowing System για λειτουργικά συστήματα UNIX. Η IBM και η Microsoft χρησιμοποίησαν πολλές από τις ιδέες της Apple για να αναπτύξουν τις προδιαγραφές Κοινής Πρόσβασης Χρήστη (Common User Access) που α-



(α′) Μία τυπική επιφάνεια εργασίας των αρχών της δεκαετίας του 1990 που βασίζεται στο X Window System.



(β′) Μία σύγχρονη επιφάνεια εργασίας σε περιβάλλον GNOME σε λειτουργικό σύστημα Linux.



(γ′) Μία σύγχρονη επιφάνεια εργασίας σε περιβάλλον Windows Vista.

(δ′) Μία σύγχρονη επιφάνεια εργασίας σε περιβάλλον Mac OS X.

Σχήμα 2.1: Στις παραπάνω εικόνες φαίνονται διάφορες Διεπαφές Χρήστη Επιφανειών Εργασίας (Desktops). Όλες οι εικόνες, εκτός από την πρώτη, είναι από επιφάνειες εργασίας που χρησιμοποιούνται σήμερα. Αν και η εμφάνισή τους διαφέρει, παρατηρούμε πως έχουν ομοιότητες σε κύρια σημεία.

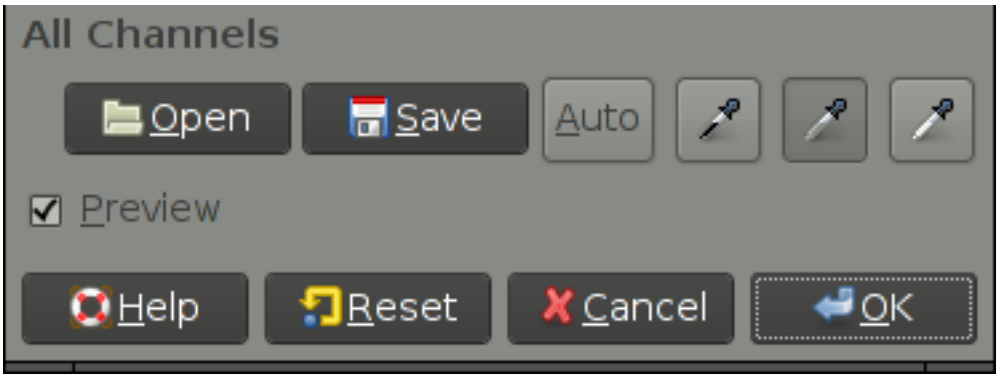
ποτέλεσαν τη βάση για τις Διεπαφές Χρήστη που εμφανίζονται στα Microsoft Windows, IBM OS/2 Presentation Manager και στα περιβάλλοντα Επιφάνειας Εργασίας για X Windows Servers (Gnome/KDE/xfce). Έτσι πολλλές από τις σύγχρονες Διεπαφές Χρήστη έχουν πολλά κοινά στοιχεία. Στο σχήμα 2.1 βλέπουμε διάφορες Διεπαφές Χρήστη από περιβάλλοντα Επιφάνειας Εργασίας. Παρατηρούμε πως παρά τη διαφορά στην εμφάνιση και την υφή των διεπαφών, είναι εμφανή τα κοινά στοιχεία στα οποία βασίζονται, όπως η χρήση κουμπιών παραθύρων για αλλαγή μεγέθους, θέσης, κλπ, και η οργάνωση σε διακριτά στοιχεία - τα Γραφικά Στοιχεία - με μία ιεραρχική οργάνωνση[14, 3].

2.3 Στοιχεία Γραφικών Διεπαφών Χρήστη

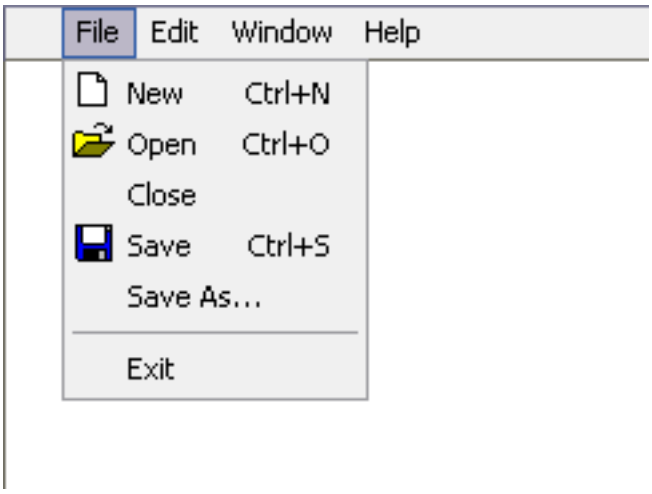
Όπως είδαμε στην προηγούμενη ενότητα, και όπως φαίνεται και στο σχήμα 2.1, μια Γραφική Διεπαφή οργανώνεται ιεραρχικά σε Γραφικά Στοιχεία. Σε αυτή την ενότητα θα ορίσουμε τι θεωρείται ως Γραφικό Στοιχείο (υποενότητα 2.3.1) και πώς αυτά συνδυάζονται στην ιεραρχική δομή που αποτελεί τη Γραφική Διεπαφή (υποενότητα 2.3.2). Τέλος, θα αναδείξουμε το ρόλο της ιεραρχικής δομής στη διαδραστικότητα των Γραφικών Διεπαφών (υποενότητα 2.3.3).

2.3.1 Γραφικό Στοιχείο

Στον τομέα ανάπτυξης γραφικών εφαρμογών, ένα Γραφικό Στοιχείο είναι ένα στοιχείο μιας Γραφικής Διεπαφής Χρήστη που εμφανίζει μια διαρρύθμιση πληροφοριών τροποποιούμενη από το χρήστη, όπως ένα παράθυρο ή ένα κουτί κειμένου (text box)[11, 1, 3]. Το χαρακτηριστικό στοιχείο ενός Γραφικού Στοιχείου είναι ότι προσφέρει ένα σημείο διάδρασης μεταξύ Διεπαφής-χρήστη για την άμεση μεταχείριση (direct manipulation)[15] κάποιου είδους δεδομένων. Τα Γραφικά Στοιχεία είναι τα βασικά κατασκευαστικά αντικείμενα τα οποία, όταν συνδυάζονται σε μια εφαρμογή, περιέχουν όλα τα δεδομένα που επεξεργάζεται η εφαρμογή καθώς και προσφέρουν τους τρόπους αλληλεπίδρασης με αυτά.



(α′) Διάφορα είδη *κουμπιών* της εργαλειοθήκης GTK+



(β′) Ένα γενικό *μενού επιλογής*.

Σχήμα 2.2: Εικόνες από διάφορα Γραφικά Στοιχεία που συναντάμε στις σύγχρονες Διεπαφές.

Από την πρωταρχική έρευνα της PARC για το Xerox Alto User Interface, έχει εξελιχθεί σήμερα μια οικογένεια κοινών επαναχρησιμοποιούμενων Γραφικών Στοιχείων με διάφορες υλοποιήσεις[3, 16]. Κάποια από αυτά είναι τα:

- Κουμπί (Σχήμα 2.2α′).
- Κουτί επιλογής.
- Μενού επιλογής (Σχήμα 2.2β′).

Κάποια Γραφικά Στοιχεία είναι περισσότερο πολύπλοκα από άλλα και μπορούμε να υποθέσουμε πως αποτελούνται από απλούστερα. Σύμφωνα με αυτή την παρατήρηση, μπορούμε να διαχωρίσουμε τα Γραφικά Στοιχεία σε δύο βασικές κατηγορίες.

Πρωτεύοντα Γραφικά Στοιχεία

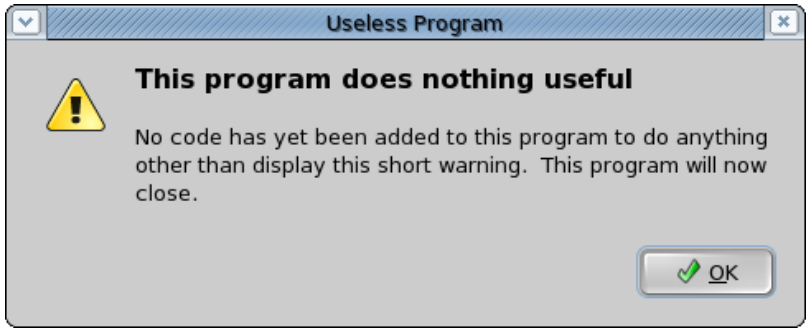
Η πρώτη κατηγορία αποτελείται από τα πιο απλά Γραφικά Στοιχεία που παρέχονται από κάποιο σύστημα. Αυτά συνήθως είναι τα Γραφικά Στοιχεία που είναι περιέκτες άλλων, όπως για παράδειγμα το κουμπί και το παράθυρο.

Σύνθετα Γραφικά Στοιχεία

Οι εφαρμογές ΠΕΕΔ, που όπως αναφέραμε στην ενότητα 2.2, ακολουθούν κάποιους κοινούς ιδιωματισμούς. Για παράδειγμα, χρησιμοποιούν επιλογείς. Το *μενού επιλογής* (βλέπε σχήμα 2.2β′) είναι ένα Γραφικό Στοιχείο το οποίο όμως αποτελείται από απλούστερα Πρωτεύοντα Γραφικά Στοιχεία: ένα παράθυρο, ένα Γραφικό Στοιχείο κάθετης διάταξης για τις γραμμές και σε κάθε γραμμή ένα *εικονίδιο* δίπλα σε μία *ετικέτα*. Γραφικά Στοιχεία, σαν το προηγούμενο, τα οποία αποτελούνται από μια ιεραρχία άλλων Γραφικών, Στοιχείων ονομάζονται Σύνθετα Γραφικά Στοιχεία ή Γραφικά Μεταστοιχεία[14]. Πολλά Σύνθετα Γραφικά Στοιχεία, όπως το *μενού επιλογής*, επειδή χρησιμοποιούνται πολύ συχνά σε Γραφικές Διεπαφές παρέχονται υλοποιημένα από τα διάφορα συστήματα ανάπτυξης γραφικών εφαρμογών τύπου ΠΕΕΔ[3].

2.3.2 Ιεραρχική Δόμηση Γραφικών Διεπαφών - Δέντρο Γραφικών Στοιχείων

Όπως αναφέραμε και στις προηγούμενες ενότητες, οι Γραφικές Διεπαφές αποτελούνται από διάφορα Γραφικά Στοιχεία που σχηματίζουν μια ιεραρχία. Είδαμε πως υπάρχουν Γραφικά Στοιχεία που δρουν ως περιέκτες (containers) άλλων και πως ομάδες από διάφορα Γραφικά Στοιχεία δημιουργούν Γραφικά Μεταστοιχεία.



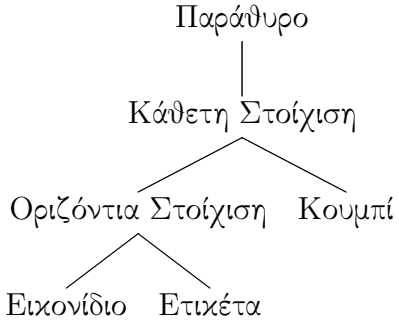
Σχήμα 2.3: Παράδειγμα μιας Γραφικής Διεπαφής. Η Διεπαφή φαίνεται ότι χρησιμοποιεί ένα *παράθυρο*, ένα *εικονίδιο*, δύο *ετικέτες* και ένα *κουμπί*.

Η ιεραρχική δομή που σχηματίζει μία Γραφική Διεπαφή είναι αυτή ενός δέντρου. Στη ρίζα βρίσκεται συνήθως ένα Γραφικό Στοιχείο περιέκτης, όπως το παράθυρο, που ουσιαστικά περιέχει όλα τα υπόλοιπα που αποτελούν τη Διεπαφή. Στους εσωτερικούς κόμβους του δέντρου βρίσκονται Γραφικά Στοιχεία περιέκτες, που όμως περιέχονται στους πατρικούς κόμβους, καθώς και Σύνθετα Γραφικά Στοιχεία. Τέλος, στα φύλλα του δέντρου βρίσκουμε Πρωτεύοντα Γραφικά Στοιχεία.

Το σχήμα 2.3 δείχνει το παράδειγμα μιας Διεπαφής[2]. Είναι ξεκάθαρο ποια γραφικά στοιχεία χρησιμοποιούνται σε αυτή: *εικονίδιο*, *ετικέτα*, *κουμπί* και *παράθυρο*. Επιπλέον όμως, χρησιμοποιούνται και Γραφικά Στοιχεία για τη στοίχι-

[2]Η Διεπαφή του παραδείγματος δεν έχει κάποιο ιδιαίτερο νόημα απλά δίνεται ως ένα απλό παράδειγμα γιατί η ιεραρχία των Γραφικών Στοιχείων που χρησιμοποιούνται είναι σχετικά μικρή.

ση των υπόλοιπων όπως η *οριζόντια στοίχιση* και η *κάθετη στοίχιση*. Όλα αυτά τα Γραφικά Στοιχεία συνδυάζονται μεταξύ τους στο δέντρο Γραφικών Στοιχείων του σχήματος 2.4. Το *παράθυρο* περιέχει μία *κάθετη στοίχιση* με δύο γραμμές. Η πρώτη περιέχει μία *οριζόντια στοίχιση* με δύο στήλες, η πρώτη με ένα *εικονίδιο* και η δεύτερη με μία *ετικέτα*. Η δεύτερη γραμμή περιέχει ένα *κουμπί*.



Σχήμα 2.4: Το δέντρο Γραφικών Στοιχείων που αποτελούν τη Διεπαφή του σχήματος 2.3.

2.3.3 Διαδραστικότητα Γραφικών Διεπαφών - Λειτουργίες Δέντρου Γραφικών Στοχείων

Οι Γραφικές Διεπαφές Χρήστη κατασκευάζονται για να είναι διαδραστικές. Αυτό σημαίνει πως εμφανίζουν ανάδραση[3] σε ενέργειες του χρήστη αλλά και της ίδιας της εφαρμογής[13]. Η ανάδραση επιτυγχάνεται με κάποια τροποποίηση στη Γραφική Διεπαφή[15].

Η τροποποίηση της Γραφικής Διεπαφής γίνεται με κάποια αλλαγή στο δέντρο των Γραφικών Στοιχείων που την αποτελούν, δηλαδή με τροποποίηση τις ιεραρχικής δομής της Διεπαφής. Μια αλλαγή στο δέντρο των Γραφικών Στοιχείων μιας Διεπαφής μπορεί να γίνει με τρεις τρόπους:

[3]Διεπαφές χωρίς ανάδραση δεν θεωρούνται συνήθως καλές, καθώς ο χρήστης τους δεν ξέρει αν και πότε εκτελείται μια ενέργεια αλλά ούτε και ποιο είναι το αποτέλεσμά της!

1. Με την τροποποίηση κάποιας ιδιότητας ενός κόμβου του δέντρου, δηλαδή κάποιου Γραφικού Στοιχείου. Για παράδειγμα, την αλλαγή της συμβολοσειράς μιας *ετικέτας*.

2. Με την προσθήκη κάποιου νέου Γραφικού Στοιχείου ως περιεχόμενο κάποιου κόμβου.

3. Με την αφαίρεση κάποιου κόμβου από το δέντρο. Δηλαδή, την εξαφάνιση κάποιου Γραφικού Στοιχείου.

Άλλες πιο πολύπλοκες αλλαγές μπορούν να εκφραστούν με κατάλληλους συνδυασμούς βασικών λειτουργιών πάνω στο δέντρο των Γραφικών Στοιχείων.

Κάθε τροποποίηση στη Γραφική Διεπαφή αλλάζει την κατάσταση στην οποία βρίσκεται εκείνη τη χρονική στιγμή. Έτσι κάθε Γραφική Διεπαφή έχει ένα σύνολο καταστάσεων στις οποίες βρίσκεται σε κάθε χρονική στιγμή λειτουργίας της[3, 17]. Οι κανόνες μετάβασης από μια κατάσταση σε άλλη είναι αυστηρά ορισμένοι και ουσιαστικά εκφράζουν τις λειτουργίες της[18]. Οπότε, για κάθε Γραφική Διεπαφή ορίζεται ένα διάγραμμα καταστάσεων[11, 1]. Κάθε κατάσταση, είναι ένα συγκεκριμένο στιγμιότυπο του δέντρου των Γραφικών Στοιχείων της Διεπαφής. Σε κάθε μετάβαση εκτελείται και κάποια ενέργεια στο δέντρο.

Η ανάπτυξη Γραφικών Διεπαφών γίνεται ορίζοντας το δέντρο των Γραφικών Στοιχείων καθώς και τις μεταβάσεις κατάστασης χρησιμοποιώντας ειδικευμένες εργαλειοθήκες λογισμικού (toolkits). Αυτές θα αναλύσουμε στις επόμενες ενότητες.

2.4 Εργαλειοθήκες Λογισμικού για Γραφικές Διεπαφές Χρήστη

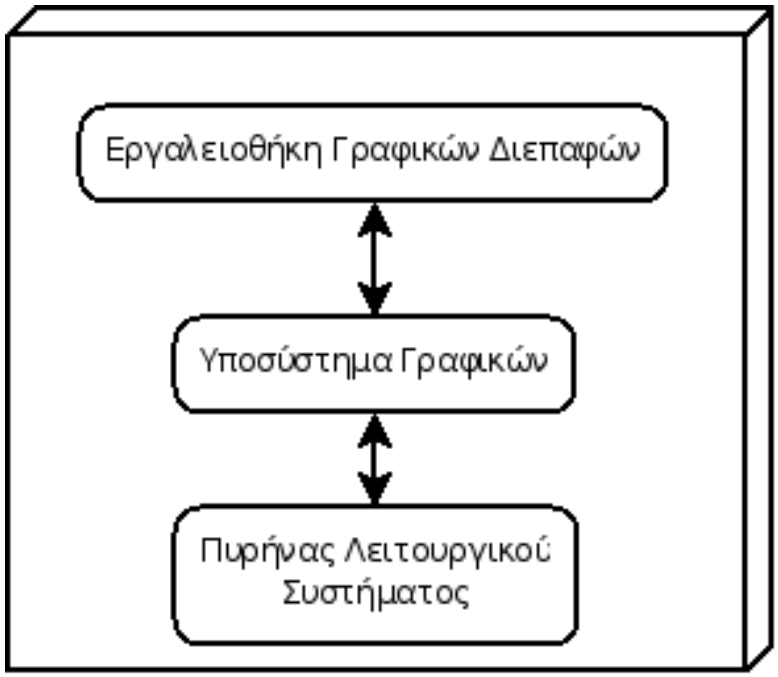
Τα σύγχρονα λειτουργικά συστήματα όπως Windows, Linux και MacOS διαθέτουν κάποιο υποσύστημα γραφικών[3]. Στο Linux και γενικότερα στα διάφορα Unix αυτό είναι παραδοσιακά ο X Server, στα Windows το GDI+ και

στα Mac OS το Quartz. Κάθε ένα από αυτά υλοποιεί βασικές λειτουργίες για την εμφάνιση γραφικών στην οθόνη, όπως χρώματα και βασικά σχήματα (ευθείες, κύκλους κλπ). Η σχεδίαση και ανάπτυξη μιας εφαρμογής με Διεπαφή Χρήστη θα ήταν πολύ προβληματική αν γινόταν άμεσα σε κάποιο από αυτά. Για το λόγο αυτό, η ανάπτυξη γραφικών εφαρμογών συνήθως γίνεται χρησιμοποιώντας κάποια εργαλειοθήκη λογισμικού (toolkit)[12, 9, 3]. Μία εργαλειοθήκη είναι ένα σύνολο βιβλιοθηκών που προσφέρουν μια Διεπαφή Προγραμματισμού Εφαρμογών (API - Application Programming Interface) για τη δημιουργία Διεπαφών Χρήστη, στο περιβάλλον μιας γλώσσας προγραμματισμού, χωρίς να απασχολείται ο προγραμματιστής με τις δύσκολες λεπτομέρειες του υποσυστήματος γραφικών του λειτουργικού συστήματος. Οι εργαλειοθήκες βασίζονται σε αυτές τις λειτουργίες για να υλοποιήσουν τα Γραφικά Στοιχεία όπως παράθυρα, επιλογείς, κουμπιά επιλογής και εικονίδια αλλά και την αλληλεπίδρασή τους με το χρήστη. Υπάρχουν διάφορες τέτοιες εργαλειοθήκες για τα πιο δημοφιλή λειτουργικά συστήματα, όπως .NET, WPF, Qt, Gtk+, Swing κλπ.

Οι διάφορες εργαλειοθήκες παρουσιάζουν πολλές διαφορές μεταξύ τους. Διαφέρουν στη γλώσσα προγραμματισμού υλοποίησης π.χ. Qt είναι σε C++, Gtk+ σε C και Swing σε Java. Διαφέρουν επίσης σε ονομασίες εννοιών, δομή του API κ.α. Παρά τις διαφορές τους όμως, παρουσιάζουν κοινά στοιχεία. Κάποιες από τις τεχνικές και τους ιδιωματισμούς που χρησιμοποιούν είναι κοινά, με κυριότερη ομοιότητα τον χειρισμό γεγονότων (event handling). Αυτό θα είναι και το θέμα της επόμενης ενότητας.

2.5 Λογική Οδηγούμενη από Γεγονότα

Το πιο κοινό στοιχείο των διάφορων εργαλειοθηκών είναι ότι ακολουθούν λογική οδηγούμενη από γεγονότα (event driven paradigm)[9, 3, 1]. Σύμφωνα με αυτή, η ροή εκτέλεσης των ενεργειών μιας εφαρμογής δεν είναι ακολουθιακή, αλλά καθορίζεται από γεγονότα που συμβαίνουν είτε εξωτερικά, είτε εσωτερικά



Σχήμα 2.5: Η δομή της στοίβας λογισμικού που χρησιμοποιείται για την κατασκευή Γραφικών Διεπαφών.

της εφαρμογής. Η εφαρμογή εκτελεί έναν ατέρμονα βρόχο, που ονομάζουμε βρόχο γεγονότων (event loop), στον οποίο περιμένει την εμφάνιση κάποιου γεγονότος. Εφόσον κάποιο γεγονός εμφανιστεί, τότε εκτελεί ενέργειες χειρισμού του, αν αυτές έχουν καθοριστεί. Σε μια γραφική εφαρμογή γεγονότα συνήθως θεωρούνται ενέργειες του χρήστη πάνω σε κάποιο Γραφικό Στοιχείο, όπως το κλικ πάνω σε ένα κουμπί, η μετακίνηση του δείκτη του ποντικιού πάνω από κάποιο Γραφικό Στοιχείο, το χτύπημα κάποιου πλήκτρου κλπ.

Το κάθε Γραφικό Στοιχείο που διαθέτει μία εργαλειοθήκη ορίζει ένα σύνολο από γεγονότα τα οποία μπορούν να συμβούν σε αυτό (ή να ξεκινήσουν από αυτό, ανάλογα πώς το προσεγγίζει κανείς). Πολλά από τα γεγονότα είναι κοινά μεταξύ των περισσότερων Γραφικών Στοιχείων και συνήθως ένα Γραφικό Στοιχείο έχει ένα μικρό αριθμό γεγονότων μοναδικών για εκείνο, τα οποία το ξεχωρίζουν από τα υπόλοιπα. Το σύνολο των γεγονότων που ορίζονται σε κάθε Γραφικό Στοιχείο ουσιαστικά αντιστοιχεί στο σύνολο των δυνατών λειτουργιών του. Για κάθε τύπο γεγονότος, ορίζεται και ένας τύπος χειριστή γεγονότος (event handler), ο οποίος εκτελείται όταν αυτό συμβεί. Σε γενικές γραμμές, μπορούμε να θεωρήσουμε έναν χειριστή γεγονότος μία ρουτίνα, η υπογραφή[4] η οποία εξαρτάται από το είδος του γεγονότος.

Η εφαρμογή, ανάλογα με τα Γραφικά Στοιχεία που χρησιμοποιεί και τα γεγονότα που την ενδιαφέρουν, ορίζει για το κάθε γεγονός έναν (ή και περισσότερους) χειριστές. Κατά την εκτέλεση της εφαρμογής, μόλις συμβεί κάποιο από αυτά τα γεγονότα εκτελείται ο αντίστοιχος χειριστής (ή οι χειριστές αν είναι πολλοί). Η κάθε εργαλειοθήκη προσεγγίζει διαφορετικά το τι είναι ένας τύπος γεγονότος και τι ένας τύπος χειριστή και με ποιο τρόπο η ρουτίνα ενός χειριστή αντιστοιχίζεται σε ένα γεγονός, κάτι που εξαρτάται και από τη γλώσσα προγραμματισμού στην όποια δίνεται το API της εργαλειοθήκης. Για παράδειγμα στο GTK+, που διαθέτει API στη γλώσσα C, ο τύπος του χειριστή είναι ένας δείκτης σε συνάρτηση.

[4] Δηλαδή ο αριθμός και ο τύπος των παραμέτρων καθώς και η τύπος της επιστρεφόμενης τιμής της.

Οι ενέργειες που εκτελούνται από ένα χειριστή γεγονότος εξαρτώνται από την εφαρμογή και είναι αυτές που ουσιαστικά καθορίζουν τη συμπεριφορά της. Στην ενότητα 2.3.3 αναφέραμε πως οι Γραφικές Διεπαφές προσφέρουν ανάδραση σε ενέργειες του χρήστη και της εφαρμογής μέσω ενεργειών τροποποίησης πάνω στο δέντρο των Γραφικών Στοιχείων. Οι ενέργειες του χρήστη μιας Διεπαφής δημιουργούν γεγονότα. Το ίδιο μπορεί να κάνει η ίδια η εφαρμογή ή κάποιο άλλο κομμάτι λογισμικού. Επομένως, οι χειριστές των γεγονότων αναλαμβάνουν να τροποποιήσουν τη Διεπαφή ώστε η εφαρμογή να εμφανίσει ανάδραση. Όμως εκτός από αυτό οι χειριστές γεγονότων εκτελούν και ενέργειες που εκφράζουν τις λειτουργίες που προσφέρει η εφαρμογή και επεξεργάζονται τα δεδομένα της[3]. Δηλαδή, ο χειρισμός ενός γεγονότος μπορεί να διαχωριστεί σε δύο διακριτές φάσεις:

1. Την εκτέλεση ενεργειών που αφορούν τις λειτουργίες της εφαρμογής, κυρίως επεξεργασία δεδομένων.

2. Την ενημέρωση της Διεπαφής για το αποτέλεσμα των ενεργειών.

Δεν είναι απαραίτητο να συμβαίνουν και τα δύο σε ένα χειριστή γεγονότος. Όμως για να προσφερθεί η αίσθηση της ανάδρασης στον χρήστη, το δεύτερο θεωρείται «σχεδόν» υποχρεωτικό. Στην ενότητα 2.3.3 αναφέραμε πως για κάθε Διεπαφή ορίζεται ένα διάγραμμα καταστάσεων στις οποίες βρίσκεται το δέντρο των Γραφικών Στοιχείων που την αποτελούν. Ουσιαστικά, οι χειριστές γεγονότων είναι υπεύθυνοι για τη μετάβαση από τη μια κατάσταση στην άλλη.

2.6 Λογική Παρουσίασης και Επιχειρησιακή Λογική

Στις προηγούμενες ενότητες εξηγήσαμε τις Γραφικές Διεπαφές, τα Γραφικά Στοιχεία που τις αποτελούν και πως με το χειρισμό γεγονότων που συμβαίνουν σε αυτά εκτελούμε τις λειτουργίες τις εφαρμογής αλλά και ενημερώνουμε μια

Διεπαφή με τα αποτελέσματα. Όλα αυτά αποτελούν μέρος της λογικής μιας γραφικής εφαρμογής[5]. Πριν όμως προχωρήσουμε περαιτέρω, είναι αναγκαίο να διαχωρίσουμε αυτή τη λογική σε δύο βασικές κατηγορίες.

Επιχειρησιακή Λογική

Αυτήν αποτελούν όλοι εκείνοι οι επιχειρησιακοί κανόνες που ορίζουν ουσιαστικά τις λειτουργίες της εφαρμογής. Η επιχειρησιακή λογική αποτελείται από εκείνον τον κώδικα που υλοποιεί τη λειτουργικότητα μιας εφαρμογής. Για παράδειγμα, σε μια εφαρμογή διαχείρισης αρχείων, όπως π.χ. ο Explorer των Windows, επιχειρησιακή λογική θεωρούμε τον κώδικα που αλληλεπιδρά με το σύστημα αρχείων του λειτουργικού συστήματος για να διαβάσει, γράψει, διαγράψει, μετακινήσει κλπ αρχεία και φακέλους.

Λογική Παρουσίασης

Αυτή αποτελείται από εκείνο τον κώδικα που παρουσιάζει τη λειτουργικότητα μιας εφαρμογής στο χρήστη μέσω μια Γραφικής Διεπαφής. Δηλαδή, προσφέρει μια φιλική διεπαφή προς την επιχειρησιακή λογική της εφαρμογής. Έχοντας πει αυτά, η λογική παρουσίασης περιλαμβάνει της Γραφικές Διεπαφές της εφαρμογής σε όλες τις καταστάσεις που μπορούν να βρεθούν. Δηλαδή, είναι το σύνολο καταστάσεων του δέντρου των Γραφικών Στοιχείων που αποτελούν τη Διεπαφή και ειδικότερα οι κανόνες μετάβασης από τη μια κατάσταση στην άλλη.

2.6.1 Χειριστές Γεγονότων: Σημεία Σύνδεσης Επιχειρησιακής Λογικής και Λογικής Παρουσίασης

Έχοντας ορίσει τα δύο είδη λογικής από τα οποία αποτελείται μια γραφική εφαρμογή, από αυτά που αναφέραμε στην ενότητα 2.5 συμπεραίνουμε πως στους

[5] Δηλαδή έχει αναπτυχθεί κώδικας που να υλοποιεί αυτή τη συμπεριφορά.

χειριστές γεγονότων εμφανίζονται και τα δύο είδη λογικής. Αυτή η παρατήρηση είναι ιδιαίτερα σημαντική καθώς υπάρχει έντονη ανάγκη για το διαχωρισμό των δύο και όσο το δυνατό λιγότερη αλληλεξάρτηση μεταξύ των υλοποιήσεών τους[9]. Επιπλέον, όπως θα δούμε στη συνέχεια, είναι το σημείο στο οποίο αποτυγχάνουν οι μέχρι σήμερα προσεγγίσεις να πετύχουν αυτό το στόχο. Είναι όμως και ένα από τα σημεία μέσα από τα οποία η εργασία αυτή αλλάζει την κατάσταση. Στην επόμενη ενότητα θα αναδείξουμε γιατί είναι τόσο αναγκαίος ο διαχωρισμός μεταξύ των δύο ειδών λογικής που αναλύσαμε.

2.7 Ανάγκη Διαχωρισμού Λογικής Παρουσίασης και Επιχειρησιακής Λογικής

Στο τέλος της προηγούμενης ενότητας καταλήξαμε στο συμπέρασμα πως οι χειριστές γεγονότων σε μια γραφική εφαρμογή περιλαμβάνουν και επιχειρησιακή λογική αλλά και λογική παρουσίασης. Αυτό σημαίνει ότι στον κώδικα ενός χειριστή γεγονότος υπάρχει ανακατεμένος κώδικας που αφορά τη λογική παρουσίασης με κώδικα που αφορά την επιχειρησιακή λογική. Κάτι τέτοιο μπορεί, όπως εξηγούμε παρακάτω, να προκαλέσει πολλά προβλήματα.

Η λογική παρουσίασης και η επιχειρησιακή λογική μπορεί να είναι αρκετά πολύπλοκες. Όταν όλη αυτή η λογική εκφράζεται μέσα στους χειριστές γεγονότων τότε αυξάνεται πάρα πολύ η σύζευξη μεταξύ της λογικής παρουσίασης και της επιχειρησιακής λογικής δυσκολεύοντας κατά πολύ την ανάπτυξη και συντήρηση μιας εφαρμογής. Αυτό σημαίνει πως ενδεχόμενες τροποποιήσεις στη Διεπαφή Χρήστη μιας εφαρμογής θα πρέπει να συνοδευτούν με πολλές αλλαγές σε διαφορετικούς χειριστές γεγονότων, αλλά και αντίστροφα, τροποποιήσεις της επιχειρησιακής λογικής απαιτούν αλλαγές στο πώς αυτή χρησιμοποιείται στη λογική παρουσίασης. Αυτό μπορεί να οδηγήσει σε πολύ αχρείαστο κώδικα, επανάληψη κώδικα και σε δυσνόητο κώδικα.

Για παράδειγμα φανταστείτε μια εφαρμογή της οποίας η επιχειρησιακή λογική προβλέπει δύο λειτουργίες σε κάποια δεδομένα κάποιας βάσης δεδομένων,

την αποθήκευση και τη διαγραφή. Ας υποθέσουμε ότι αρχικά αποφασίζουμε η Διεπαφή να έχει δύο κουμπιά, «Αποθήκευση» και «Διαγραφή», τα οποία όταν πατηθούν θα προκαλέσουν την εκτέλεση των αντίστοιχων λειτουργιών της επιχειρησιακής λογικής και θα ενημερώσουν την τιμή μιας ετικέτας στη Διεπαφή μας για να δείξουμε στο χρήστη ότι η λειτουργία εκτελέσθηκε σωστά ή ότι απέτυχε με κάποιο τρόπο. Επομένως, θα χρειαστεί να ορίσουμε δύο χειριστές γεγονότων, ένα στο κάθε κουμπί, για τα γεγονός του πατήματος. Στο χειριστή για το κουμπί «Αποθήκευση» γράφουμε τον κώδικα που αποθηκεύει τα δεδομένα μας στη βάση δεδομένων και στη συνέχεια γράφουμε στην ετικέτα μια συμβολοσειρά για να δείξουμε το αποτέλεσμα της αποθήκευσης (επιτυχία ή αποτυχία). Ομοίως και στο χειριστή για το κουμπί «Διαγραφή» με τη μόνη διαφορά πως ο κώδικας θα διαγράφει τα δεδομένα.

Σε αυτό το σημείο αξίζει να παρατηρήσουμε πως αν και η Διεπαφή μας είναι σχετικά απλή με μια απλή λογική παρουσίασης, η επιχειρησιακή λογική δεν είναι τόσο απλή καθώς απαιτεί τη σύνδεση σε κάποια βάση δεδομένων και την εκτέλεση ενεργειών σε αυτή, κάτι που συνήθως παίρνει αρκετές γραμμές κώδικα[6]. Ας κάνουμε τώρα μια αλλαγή στη λογική παρουσίασης.

Ας υποθέσουμε πως αλλάζουμε γνώμη για τη λογική παρουσίασης και πως θέλουμε να έχουμε ένα μόνο κουμπί, «Εκτέλεση», το οποίο θα εκτελεί την ενέργεια που επιλέγεται από ένα κουμπί επιλογής (radio button) με δύο επιλογές, «Αποθήκευση», «Διαγραφή». Μόνο στον τομέα των χειριστών γεγονότων πρέπει να αλλάξουν πολλά. Το κυριότερο είναι ότι οι δύο χειριστές γεγονότων που αναπτύξαμε προηγουμένως, είναι πια άχρηστοι. Θα πρέπει να ορίσουμε ένα νέο χειριστή για το κουμπί «Εκτέλεση» το οποία ανάλογα με την επιλεγμένη τιμή στο κουμπί επιλογής θα κάνει τις αντίστοιχες ενέργειες. Άρα, θα πρέπει να μεταφέρουμε τον κώδικα που έκανε την αποθήκευση και τη διαγραφή των δεδομένων από τους δύο προηγούμενους χειριστές στον καινούργιο και να τον κάνουμε να εκτελείται υπό συνθήκη. Μια φαινομενικά απλή αλλαγή στη λογική

[6]Αυτό φυσικά διαφέρει από υλοποίηση σε υλοποίηση αλλά ο σκοπός είναι να δείξουμε πως ένας χειριστής γεγονότος μπορεί να γίνει αρκετά πολύπλοκος.

παρουσίασης μπορεί να προκαλέσει πολύ πολύπλοκες αλλαγές στον κώδικα.

Φανταστείτε τι θα γινόταν με περισσότερες και πιο πολύπλοκες αλλαγές στη λογική παρουσίασης ή αν άλλαζε κάτι ή προστίθετο κάτι στην επιχειρησιακή λογική. Το πρόβλημα γίνεται ακόμα πιο σοβαρό καθώς οι περισσότερες εφαρμογές αναπτύσσονται από περισσότερα του ενός άτομα και πολλές φορές διαφορετικό άτομο αναλαμβάνει να σχεδιάσει ή/και υλοποιήσει τη λογική παρουσίασης από το άτομο που αναλαμβάνει την ανάπτυξη της επιχειρησιακής λογικής.

Για τους παραπάνω λόγους, πολύ νωρίς, δημιουργήθηκε η ανάγκη για διαχωρισμό της λογικής παρουσίασης και της επιχειρησιακής λογικής έτσι ώστε αλλαγές στη μία να μην επηρεάζουν την άλλη αλλά και να μειωθεί η σύζευξη μεταξύ τους στους χειριστές γεγονότων ώστε οι αλλαγές στους χειριστές γεγονότων να είναι όσο πιο ανώδυνες[9, 3, 2]. Για την ικανοποίηση αυτής της ανάγκης έχουν αναπτυχθεί τεχνικές διαχωρισμού της επιχειρησιακής λογικής και της λογικής παρουσίασης, με κυρίαρχες το μοτίβο σχεδίασης MVC και τις παραλλαγές του, καθώς και την αρχιτεκτονική 3 επιπέδων. Στην επόμενη ενότητα θα παρουσιάσουμε αυτές τις τεχνικές.

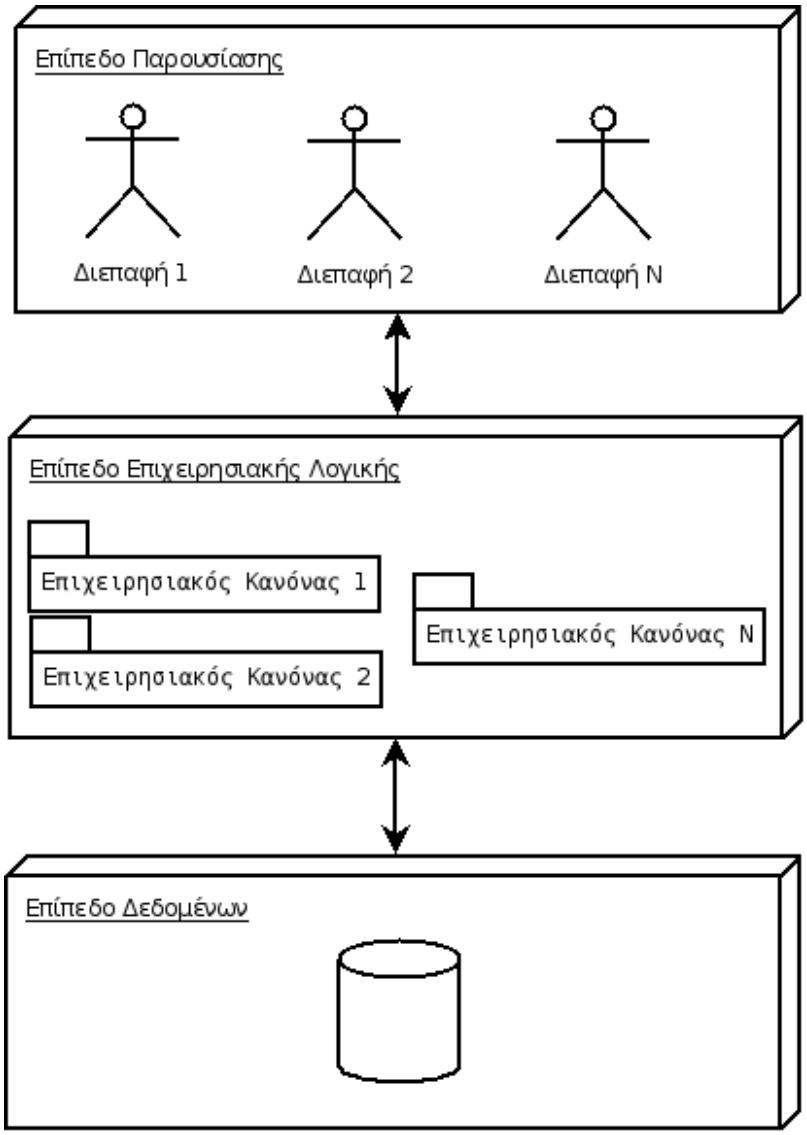
2.8 Τεχνικές Διαχωρισμού

Σε αυτή την ενότητα θα παρουσιάσουμε τις κυριότερες τεχνικές που αναπτύχθηκαν για το διαχωρισμό επιχειρησιακής λογικής και λογικής παρουσίασης. Η πρώτη είναι η αρχιτεκτονική 3 επιπέδων και η δεύτερη το μοτίβο σχεδίασης Μοντέλου-Όψης-Ελεγκτή (MVC).

2.8.1 Αρχιτεκτονική 3 επιπέδων

Η αρχιτεκτονική 3 επιπέδων (ή ακόμα και ν-επιπέδων) είναι μια συνηθισμένη πρακτική της τεχνολογίας λογισμικού που διαχωρίζει μια εφαρμογή σε 3 διακριτά επίπεδα[19]. Η δομή αυτής φαίνεται στο σχήμα 2.6

Στο κατώτερο επίπεδο τοποθετείται η λογική δεδομένων, το πρώτο από κάτω στο σχήμα 2.6. Σε αυτό το επίπεδο γίνεται η διαχείριση των δεδομένων



Σχήμα 2.6: Η αρχιτεκτονική 3 επιπέδων.

που χρησιμοποιεί η εφαρμογή. Παραδοσιακό παράδειγμα είναι η χρήση βάσης δεδομένων σε αυτό το επίπεδο. Σε μια λογιστική εφαρμογή, για παράδειγμα, σε αυτό το επίπεδο θα βρισκόταν μια βάση δεδομένων με ένα σχήμα που αποτελείται από πίνακες πελατών, τιμολογίων, εσόδων, εξόδων, συναλλαγών κλπ οι οποίοι αποθηκεύουν τα δεδομένα που η εφαρμογή διαχειρίζεται.

Στο ενδιάμεσο επίπεδο τοποθετείται η επιχειρησιακή λογική. Αυτήν αποτελούν όλοι οι επιχειρησιακοί κανόνες που ορίζουν ουσιαστικά τις λειτουργίες της εφαρμογής. Το επίπεδο αυτό αλληλεπιδρά με το επίπεδο της λογικής δεδομένων ώστε να επεξεργαστεί και να διαβάσει τα δεδομένα που χρησιμοποιεί. Στο παράδειγμα της λογιστικής εφαρμογής, εδώ τοποθετείται το κομμάτι του λογισμικού που κάνει όλη την επεξεργασία και τους υπολογισμούς των δεδομένων του επιπέδου δεδομένων, όπως υπολογισμός ΦΠΑ, υπολογισμός κέρδους, ενημέρωση στοιχείων κλπ.

Στο ανώτερο επίπεδο βρίσκεται η λογική παρουσίασης. Αυτή αποτελείται από μια διεπαφή που παρουσιάζει τις λειτουργίες της εφαρμογής με κάποιο τρόπο. Ουσιαστικά, αποτελεί τον αντιπρόσωπο της επιχειρησιακής λογικής προς το χρήστη της εφαρμογής, είτε αυτός είναι ο άνθρωπος, είτε κάποιο άλλο λογισμικό. Όταν ο χρήστης της εφαρμογής είναι ο άνθρωπος, τότε συνήθως στη λογική παρουσίασης βρίσκουμε μία Διεπαφή Χρήστη. Στο παράδειγμα μιας λογιστικής εφαρμογής, στο επίπεδο της παρουσίασης βρίσκεται η διεπαφή των λογιστικών λειτουργιών προς τον άνθρωπο, όπως μία Διεπαφή Χρήστη επιφάνειας εργασίας, ή μια διεπαφή παγκόσμιου ιστού.

Σε αυτή την αρχιτεκτονική τα επίπεδα είναι ανεξάρτητα μεταξύ τους. Δε γνωρίζει κανένα για τα εσωτερικά του άλλου. Απλά το κάθε επίπεδο δίνει προς το επόμενο και προηγούμενο του μία διεπαφή επικοινωνίας (interface) (τα βέλη μεταξύ των επιπέδων στο σχήμα 2.6), δηλαδή τα επίπεδα συμφωνούν μεταξύ τους σε κάποιο πρωτόκολλο επικοινωνίας, το οποίο όμως είναι ανεξάρτητο υλοποίησης. Άρα, το κάθε επίπεδο μπορεί να υλοποιηθεί με πολλούς διαφορετικούς τρόπους χωρίς να επηρεάζεται από το πως υλοποιείται κάποιο άλλο, αρκεί να τηρούνται οι συμφωνημένες διεπαφές. Για παράδειγμα, δεδομένου ότι

έχει καθοριστεί η διεπαφή που εξάγει το επίπεδο δεδομένων προς το επίπεδο επιχειρησιακής λογικής, το επίπεδο δεδομένων θα μπορούσε να έχει υλοποιηθεί με μια βάση δεδομένων ή με το σύστημα αρχείων χωρίς να επηρεαστεί καθόλου το πως θα υλοποιηθεί το δεύτερο επίπεδο. Με παρόμοια λογική, στην αρχιτεκτονική 3 επιπέδων μπορούμε να έχουμε πολλές διαφορετικές διεπαφές της εφαρμογής προς τον άνθρωπο, που εκτελούν όμως ακριβώς τις ίδιες λειτουργίες που υλοποιούνται με κάποιο τρόπο στο δεύτερο επίπεδο.

Η ανεξαρτησία μεταξύ των επίπεδων μας επιτρέπει να διαχωρίσουμε την εφαρμογή σε τρία[7] φυσικά επίπεδα, τοποθετώντας το καθένα σε ξεχωριστή εφαρμογή στο ίδιο μηχάνημα ή ακόμη και σε διαφορετικά μηχανήματα σε ένα δίκτυο. Αυτός ο φυσικός διαχωρισμός μεταξύ των επιπέδων είναι πολύ συνηθισμένος στις εφαρμογές παγκόσμιου ιστού.

Εφαρμογή στις Γραφικές Διεπαφές Χρήστη

Οι εφαρμογές με Γραφικές Διεπαφές που έχουν αναπτυχθεί με αρχιτεκτονική 3-επιπέδων ουσιαστικά τοποθετούν την επιχειρησιακή λογική και τη λογική παρουσίασης σε διακριτές μονάδες λογισμικού[20]. Η μονάδα της λογικής παρουσίασης χρησιμοποιεί τη διεπαφή[8] που ορίζει η επιχειρησιακή λογική στους χειριστές γεγονότων μέσω του πρωτοκόλλου επικοινωνίας που έχει καθοριστεί.

Αυτό σημαίνει πως η λογική παρουσίασης μπορεί να αλλάξει κατά το δοκούν. Όχι μόνο ως Γραφική Διεπαφή αλλά μπορεί να αλλάξει και η εργαλειοθήκη και το περιβάλλον προγραμματισμού που χρησιμοποιείται για την ανάπτυξή της. Το ίδιο ισχύει και για το επίπεδο της επιχειρησιακής λογικής. Εφόσον η διεπαφή της επιχειρησιακής λογικής παραμείνει ίδια μπορεί να αλλάξει οτιδήποτε στο εσωτερικό της ακόμα και το περιβάλλον προγραμματισμού στο οποίο αναπτύσσεται ή ακόμα και η τοποθεσία εκτέλεσής της.

Συνήθως όμως οι εφαρμογές επιφάνειας εργασίας δεν αναπτύσσονται με αυτό τον τρόπο γιατί απαιτεί τη φυσική διάσπαση του κώδικα της εφαρμογής

[7] Το πολύ για αρχιτεκτονική 3-επιπέδων.

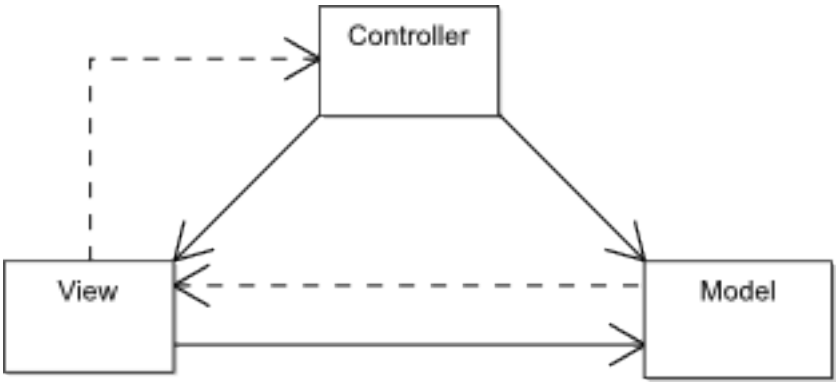
[8] Εδώ τον όρο *διεπαφή* δεν το χρησιμοποιούμε όπως ορίζεται στην επικοινωνία ανθρώπου μηχανής, αλλά όπως ορίζεται στη μηχανική λογισμικού.

και επειδή η αρχιτεκτονική δεν υποστηρίζεται άμεσα από τις εργαλειοθήκες. Προσφέρει όμως μεγαλύτερη δυνατότητα διαχωρισμού των δύο λογικών από ότι το μοτίβο σχεδίασης MVC το οποίο θα περιγράψουμε στην επόμενη ενότητα.

2.8.2 Μοντέλο Όψη Ελεγκτής

Μια άλλη ευρέος διαδεδομένη τεχνική για την ικανοποίηση της ανάγκης διαχωρισμού της επιχειρησιακής λογικής από τη λογική παρουσίασης μιας εφαρμογής, στην οποία αναφερθήκαμε στην ενότητα 2.7, είναι η τεχνική της τεχνολογίας λογισμικού: Μοντέλο Όψη Ελεγκτής (Model View Controller ή MVC)[21, 22, 4]. Η τεχνική αυτή είναι ένα μοτίβο σχεδίασης λογισμικού (design pattern) το οποίο αναπτύχθηκε ειδικά για εφαρμογές με Γραφικές Διεπαφές.

Η λογική παρουσίασης διαχωρίζεται από την επιχειρησιακή λογική με την υλοποίηση τριών αντικειμένων: το Μοντέλο, την Όψη και τον Ελεγκτή. Η Όψη και το Μοντέλο αντιστοιχούν στη Γραφική Διεπαφή και την επιχειρησιακή λογική αντίστοιχα. Ο Ελεγκτής είναι μια αφαίρεση των ενεργειών των χειριστών γεγονότων. Η Όψη και ο Ελεγκτής υλοποιούν τη λογική παρουσίασης ενώ το Μοντέλο την επιχειρησιακή λογική. Η δομή του μοτίβου σχεδίασης MVC φαίνεται στο σχήμα 2.7.



Σχήμα 2.7: Η δομή του μοτίβου σχεδίασης MVC.

Μοντέλο

Είναι μια αφαίρεση της επιχειρησιακής λογικής με καλά ορισμένο API. Ουσιαστικά περιλαμβάνει κώδικα που δίνει νόημα στα δεδομένα (π.χ. υπολογίζει αν σήμερα είναι τα γενέθλια του χρήστη ή το ολικό κόστος, τους φόρους και το κόστος μεταφοράς της παραγγελίας ενός προϊόντος). Πολλές εφαρμογές χρησιμοποιούν κάποιο μηχανισμό μόνιμης αποθήκευσης, όπως βάσεις δεδομένων για την διαχείριση των δεδομένων. Το MVC δεν αναφέρει συγκεκριμένα τι γίνεται στο επίπεδο δεδομένων γιατί εννοείται πως αυτό βρίσκεται κάτω από το Μοντέλο.

Όψη

Εμφανίζει τα δεδομένα του Μοντέλου σε μια μορφή κατάλληλη για αλληλεπίδραση με το χρήστη, παραδοσιακά σε Γραφικά Στοιχεία της Διεπαφής. Πολλαπλές Όψεις μπορούν να υπάρξουν για ένα συγκεκριμένο Μοντέλο, όπως και στην αρχιτεκτονική 3-επιπέδων όπου το επίπεδο της επιχειρησιακής λογικής θα μπορούσε να έχει πολλούς διαφορετικούς πελάτες στο επίπεδο της παρουσίασης (βλέπε ενότητα 2.8.1).

Ελεγκτής

Επεξεργάζεται και αντιδρά σε γεγονότα, συνήθως δράσεις του χρήστη, και μπορεί να προκαλέσει αλλαγές στο Μοντέλο.

Παρόλο που το MVC εμφανίζεται με διαφορετικές μορφές και παραλλαγές, η ροή ελέγχου είναι συνήθως η παρακάτω:

1. Ο χρήστης αλληλεπιδρά με τη Διεπαφή Χρήστη με κάποιο τρόπο (π.χ. πατάει ένα κουμπί).

2. Ένας Ελεγκτής χειρίζεται το γεγονός εισόδου από τη Διεπαφή Χρήστη, συνήθως μέσω ενός εγγεγραμμένου χειριστή (handler) ή callback.

3. Ο Ελεγκτής χρησιμοποιεί το Μοντέλο, πιθανώς ενημερώνοντάς το σχετικά με τη δράση του χρήστη (π.χ. ο Ελεγκτής ενημερώνει το καλάθι αγορών του χρήστη).

4. Μία Όψη χρησιμοποιεί το Μοντέλο (έμμεσα) για να δημιουργήσει τη κατάλληλη Διεπαφή Χρήστη (π.χ. η Όψη παράγει στην οθόνη μία λίστα με τα περιεχόμενα του καλαθιού αγορών του χρήστη). Η Όψη παίρνει τα δεδομένα της από το Μοντέλο. Το Μοντέλο δεν έχει καμία άμεση γνώση της Όψης.

5. Η Διεπαφή Χρήστη περιμένει περαιτέρω αλληλεπίδραση με το χρήστη ώστε να ξεκινήσει τον κύκλο από την αρχή.

Αρκετές από τις εργαλειοθήκες υποστηρίζουν άμεσα το MVC μέσα στο API, προσφέροντας αντίστοιχες κλάσεις για τα τρία συστατικά του, όπως στο Java Swing. Σε κάποια άλλα, όπως στο .Net Windows Forms, έχουν οριστεί ο Ελεγκτής και η Όψη και αφήνεται στο σχεδιαστή/προγραμματιστή να δημιουργήσει το Μοντέλο, προαιρετικά. Σε άλλες εργαλειοθήκες, όπως στο δημοφιλές GTK+, τα τρία αυτά συστατικά δεν έχουν οριστεί σαφώς στο API αλλά και πάλι η ροή ελέγχου που περιγράψαμε παραπάνω είναι η ίδια με τη διαφορά πως δεν υπάρχει κάποιος σαφής εννοιολογικός διαχωρισμός (π.χ. με τη μορφή κλάσεων ή συναρτήσεων) στο επίπεδο του API.

Κατά τη διάρκεια της ανάπτυξης συνηθίζεται τα Γραφικά Στοιχεία που χρησιμοποιούνται και συνεργάζονται μαζί για να εκτελέσουν μια λειτουργία να ομαδοποιούνται σε Γραφικά Μεταστοιχεία (βλέπε ενότητα 2.3.1). Συνηθίζεται να ορίζεται ξεχωριστή υλοποίηση του μοτίβου σχεδίασης MVC για κάθε τέτοιο Γραφικό Μεταστοιχείο ανεξάρτητα από το MVC ολόκληρης της διεπαφής δημιουργώντας έτσι μια ιεραρχία από MVC δομές.

2.9 Σύγκριση Αρχιτεκτονικής 3-Επιπέδων και MVC

Η μεγαλύτερη διαφορά μεταξύ της αρχιτεκτονικής 3-επιπέδων και του μοτίβου σχεδίασης MVC έγκειται στο ότι στην αρχιτεκτονική 3-επιπέδων γίνεται φυσική διάκριση μεταξύ των ειδών λογικής της εφαρμογής, ενώ στο μοτίβο σχεδίασης MVC η διάκριση δεν είναι φυσική. Τα διακριτά επίπεδα της αρχιτεκτονικής 3-επιπέδων και η συμφωνία μεταξύ τους σε αυστηρές διεπαφές και πρωτόκολλα επικοινωνίας επιτρέπει τη χρήση διαφορετικών ετερογενών συστημάτων ανάπτυξης και εκτέλεσης των επιπέδων. Για παράδειγμα, η λογική παρουσίασης μπορεί να είναι ορισμένη σε Adobe Flash για το περιβάλλον παγκόσμιου ιστού και ταυτόχρονα σε .NET Windows Forms ως εφαρμογή επιφάνειας εργασίας και η επιχειρησιακή λογική να βρίσκεται σε κάποιον διακομιστή παγκόσμιου ιστού. Στο μοτίβο σχεδίασης MVC τα αντικείμενα Μοντέλο, Όψη και Ελεγκτής βρίσκονται μαζί στον ίδιο χώρο διευθύνσεων της εφαρμογής. Επιπλέον, είναι υλοποιημένα στο περιβάλλον ανάπτυξης της εργαλειοθήκης που επιλέγεται για την ανάπτυξη της Γραφικής Διεπαφής. Αυτό συνήθως οδηγεί και στο να υλοποιηθεί η επιχειρησιακή λογική στο ίδιο περιβάλλον. Με άλλα λόγια οι υλοποιήσεις της λογικής παρουσίασης και της επιχειρησιακής λογικής είναι δεσμευμένες στο ίδιο περιβάλλον ανάπτυξης.

Αυτό μας οδηγεί στο συμπέρασμα πως η αρχιτεκτονική 3-επιπέδων είναι περισσότερο ευέλικτη και κατάλληλη για διαχωρισμό της επιχειρησιακής λογικής από τη λογική παρουσίασης. Όμως, όπως αναφέραμε και στην ενότητα 2.8.1, όταν η αρχιτεκτονική 3-επιπέδων εφαρμόζεται στις γραφικές εφαρμογές, η λογική παρουσίασης συνδέεται με την επιχειρησιακή λογική μέσω των χειριστών γεγονότων. Αυτό όμως σημαίνει πως είναι αναγκαίο να εφαρμοστεί το μοτίβο σχεδίασης MVC για την αποσύνδεση της Διεπαφής από τις λεπτομέρειες της επικοινωνίας με την επιχειρησιακή λογική.

2.10 Ανάπτυξη

Στην ενότητα 2.4 αναφερθήκαμε στις εργαλειοθήκες λογισμικού για τη δημιουργία Γραφικών Διεπαφών και στην ενότητα 2.5 εξηγήσαμε πώς αυτές ακολουθούν την οδηγούμενη από γεγονότα λογική. Σε αυτή την ενότητα θα μιλήσουμε για τη διαδικασία ανάπτυξης που ακολουθείται για να δημιουργηθεί μια Γραφική Διεπαφή σε κάποια εργαλειοθήκη.

Στην ανάπτυξη μιας εφαρμογής με τη χρήση κάποιας γραφικής εργαλειοθήκης ακολουθούνται κάποια βασικά βήματα, ανεξαρτήτως της εργαλειοθήκης που χρησιμοποιείται. Αρχικά ορίζεται, χρησιμοποιώντας το API, ένα δέντρο από Γραφικά Στοιχεία.

Όπως αναφέραμε και στην ενότητα 2.3.2, κάποια από τα Γραφικά Στοιχεία που παρέχονται από την εργαλειοθήκη μπορούν να λειτουργήσουν ως περιέκτες (containers) άλλων. Βασικά παραδείγματα τέτοιων είναι το παράθυρο, πίνακες και άλλα Γραφικά Στοιχεία που χρησιμοποιούνται για να στοιχίσουν άλλα Γραφικά Στοιχεία μεταξύ τους[3, 16]. Έτσι, αυτά χρησιμοποιούνται με διάφορους συνδυασμούς, σχηματίζοντας ένα δέντρο. Για παράδειγμα, το Γραφικό Στοιχείο A περιέχει τα B, το B περιέχει το C και το D, το C περιέχει μία ετικέτα και το D περιέχει ένα κουμπί. Τα φύλλα αυτού του δέντρου είναι Γραφικά Στοιχεία που παρουσιάζουν στο χρήστη δεδομένα ή μπορούν να δεχτούν κάποια αλληλεπίδραση από το χρήστη, όπως στο παράδειγμά μας το κουμπί και η ετικέτα.

Αναλόγως το API η προγραμματιστική διαδικασία που ακολουθείται για να οριστεί αυτό το δέντρο διαφέρει. Στις γραφικές εργαλειοθήκες για αντικειμενοστραφείς γλώσσες (όπως Qt για C++, Swing για Java, ṄET για C#) αυτό δημιουργείται συνήθως μέσω των μηχανισμών κληρονομικότητας (inheritance) και σύνθεσης (composition). Ο σχεδιαστής/προγραμματιστής της Διεπαφής δημιουργεί μία κλάση που κληρονομεί, για παράδειγμα, από την κλάση Window, που υλοποιεί το Γραφικό Στοιχείο του παράθυρου, και ενσωματώνει σε αυτήν τα Γραφικά Στοιχεία που θέλει να περιέχει το παράθυρο που δημιουργεί. Δηλαδή, το δέντρο των Γραφικών Στοιχείων αντιστοιχεί σε ένα δέντρο κλάσεων. Στις γραφικές εργαλειοθήκες για τις μη-αντικειμενστραφείς γλώσσες, όπως

το GTK+ για C, αντί των μηχανισμών της κληρονομικότητας και σύνθεσης, χρησιμοποιούνται συναρτήσεις του API[3].

Παράλληλα με τον ορισμό του δέντρου των Γραφικών Στοιχείων, ή αφού αυτό οριστεί, ορίζονται και οι χειριστές των γεγονότων της Διεπαφής. Η διαδικασία αυτή διαφέρει σημαντικά από εργαλειοθήκη σε εργαλειοθήκη και εξαρτάται από το αν υλοποιεί το MVC άμεσα ως μέρος του API.

Η δημιουργία κώδικα «με το χέρι» για να οριστεί το δέντρο των Γραφικών Στοιχείων αλλά και να καθοριστεί η κάθε ιδιότητα και λεπτομέρεια του κάθε Γραφικού Στοιχείου στο δέντρο μπορεί να γίνει πολύ επίπονη διαδικασία[2]. Όμως, ιστορικά κάποτε αυτός ήταν και ο μόνος τρόπος για να δει στην οθόνη ο σχεδιαστής της Διεπαφής Χρήστη αυτό που είχε σχεδιάσει στο χαρτί ή μπορεί να είχε στο μυαλό του. Η χρήση του όρου σχεδιαστής/προγραμματιστής μέχρι τώρα δεν ήταν τυχαία. Όταν σχεδιάζεται μία Διεπαφή Χρήστη με αυτόν τον τρόπο ο σχεδιαστής έπρεπε να είχε και πολύ καλές προγραμματιστικές γνώσεις καθώς και τέλεια γνώση της συγκεκριμένης γραφικής εργαλειοθήκης που χρησιμοποιούσε.

Για να ικανοποιηθεί η ανάγκη ευκολότερης σχεδίασης και ανάπτυξης Διεπαφών Χρήστη για εφαρμογές, ως επεκτάσεις των διάφορων γραφικών εργαλειοθηκών δημιουργήθηκαν εργαλεία σχεδίασης Διεπαφών Χρήστη[9, 3]. Τα εργαλεία αυτά επιτρέπουν τη σχεδίαση Διεπαφών Χρήστη με δυνατότητες WYSIWYG (What You See Is What You Get - Ότι Βλέπεις Παίρνεις) όπου ο σχεδιαστής με το ποντίκι (drag & drop) προσθέτει Γραφικά Στοιχεία στη Διεπαφή Χρήστη και θέτει τις ιδιότητές τους βλέποντας παράλληλα πώς θα φαίνεται η Διεπαφή Χρήστη κατά την εκτέλεση της εφαρμογής. Τα εργαλεία αυτά παράγουν από το αποτέλεσμα της σχεδίασης έτοιμο κώδικα που να υλοποιεί τη Διεπαφή Χρήστη που σχεδιάστηκε. Οπότε, μένει στον προγραμματιστή να ενώσει τον παραγόμενο κώδικα για τη Διεπαφή Χρήστη με τον υπόλοιπο κώδικα της εφαρμογής και να υλοποιήσει τους διάφορους χειριστές γεγονότων (event handlers) που θα χειρίζονται τις αλληλεπιδράσεις με το χρήστη.

Με την εμφάνιση εργαλείων σχεδίασης Διεπαφών που παράγουν έτοιμο

κώδικα, αυξήθηκε πολύ η παραγωγικότητα στη διαδικασία ανάπτυξης γραφικών εφαρμογών καθώς με τη χρήση τους μειώνεται πολύ ο χρόνος ανάπτυξης της Διεπαφής και επιπλέον εξαλείφονται προγραμματιστικά λάθη στον ορισμό του δέντρου των Γραφικών Στοιχείων. Έτσι, μπορεί να δοθεί περισσότερο βάρος στην καλύτερη σχεδίαση της Διεπαφής. Όμως, τέτοια εργαλεία ουσιαστικά δεν προσφέρουν κάτι παραπάνω στην αποσύνδεση της λογικής παρουσίασης και της επιχειρησιακής λογικής, καθώς απλά αυτοματοποιούν μια χειρωνακτική διαδικασία.

Κεφάλαιο 3

XML Γλώσσες Περιγραφής Διεπαφών Χρήστη

Το επόμενο βήμα σε αυτή την εξέλιξη έγινε με την εμφάνιση και εξάπλωση της XML. Έτσι όπως δημιουργήθηκε XML λεξιλόγιο για την περιγραφή σελίδων στον παγκόσμιο ιστό[5], η γλώσσα XHTML, δημιουργήθηκαν XML λεξιλόγια για την περιγραφή Διεπαφών Χρήστη που σήμερα τα αναφέρουμε ως XML γλώσσες περιγραφής Διεπαφών Χρήστη (XML User Interface Languages ή User Interface Markup Languages)[5]. Δηλαδή, όπως ένας σχεδιαστής εφαρμογών Διαδικτύου σχεδίαζε μία σελίδα γράφοντας HTML ή XHTML μπορούσε να κάνει το ίδιο ένας σχεδιαστής Διεπαφής Χρήστη χρησιμοποιώντας μία XML γλώσσα περιγραφής user interface. Αυτό μπορεί να γίνει με ένα απλό κειμενογράφο ή χρησιμοποιώντας Ό,τι Βλέπεις Παίρνεις (WYSIWYG) εφαρμογές σχεδίασης.

Δημιουργήθηκαν αρκετές τέτοιες γλώσσες, με την κάθε μία να εφαρμόζεται (συνήθως[8]) για μία και μόνο εργαλειοθήκη[10]. Κάθε μία από αυτές έχει τα δικά της χαρακτηριστικά και τις δικές τις ιδιαιτερότητες. Κάποιες από αυτές, για να λειτουργήσει η σχεδίαση, είναι μεταγλωττιζόμενες, δηλαδή η XML περιγραφή της Διεπαφής Χρήστη μεταγλωττίζεται σε κώδικα της συγκεκριμένης εργαλειοθήκης ο οποίος υλοποιεί τη Διεπαφή Χρήστη. Άλλες πάλι είναι διερ-

μηνευόμενες, δηλαδή μία βιβλιοθήκη της εργαλειοθήκης αναλαμβάνει κατά την εκτέλεση της εφαρμογής να διαβάσει την περιγραφή της Διεπαφής Χρήστη από το XML αρχείο και, ανάλογα με το τι διαβάζει, να το εμφανίσει στην οθόνη.

Όλες οι γλώσσες περιγραφής Διεπαφών Χρήστη προσφέρουν στον σχεδιαστή ένα σύνολο από Γραφικά Στοιχεία (widgets) για να μπορεί να εκφράσει διεπαφή της εφαρμογής. Συνήθως αυτά αντιστοιχούν στα Γραφικά Στοιχεία που ο σχεδιαστής θέλει να χρησιμοποιήσει, π.χ. υπάρχει ένα στοιχείο <window> για τον ορισμό ενός παραθύρου, ένα άλλο <button> για τον ορισμό ενός κουμπιού κλπ. Ο σχεδιαστής συνδυάζει τα στοιχεία αυτά κατάλληλα, θέτοντας διάφορες ιδιότητες, όπως χρώμα, μέγεθος, θέση κλπ, για να σχεδιάσει τη μορφή της διεπαφής. Εκτός από τη μορφή της διεπαφής, ο σχεδιαστής πρέπει να ορίσει τις αλληλεπιδράσεις της με τον χρήστη. Το κάθε Γραφικό Στοιχείο ορίζει ένα σύνολο από γεγονότα (events) που μπορούν να συμβούν σε αυτό, όπως αν κάποιος το έκανε κλικ ή το επέλεξε με το ποντίκι. Η συνήθης πρακτική είναι ο σχεδιαστής να ορίζει τον χειριστή του γεγονότος με ένα στοιχείο ή κάποιο χαρακτηριστικό του Γραφικού Στοιχείου. Οι χειριστές γεγονότων είναι μέρος του κώδικα της λογικής που υλοποιεί ο προγραμματιστής.

Οι γλώσσες περιγραφής Διεπαφών Χρήστη διαφέρουν μεταξύ τους ως προς τα Γραφικά Στοιχεία που προσφέρουν για τον σχεδιαστή, τους τρόπους που αυτά συνδυάζονται και στις γλώσσες προγραμματισμού που μπορεί να επιλέξει ο προγραμματιστής για να υλοποιήσει τη λογική.

Όπως είδαμε και παραπάνω, τα συστήματα αυτά διαφέρουν και ως προς τη μέθοδο που ακολουθούν για να υλοποιήσουν εν τέλει τη διεπαφή με βάση την περιγραφή τους. Σύμφωνα με την πρώτη μέθοδο, το αρχείο με την XML περιγραφή της διεπαφής «μεταγλωττίζεται» σε κώδικα μιας γλώσσας προγραμματισμού (ίδιας με αυτή που χρησιμοποιεί ο προγραμματιστής) και αυτός συνδυάζεται με τον κώδικα του προγραμματιστή για να προκύψει ο τελικός κώδικας. Σύμφωνα με τη δεύτερη μέθοδο, η XML περιγραφή της διεπαφής διαβάζεται από το αρχείο τη στιγμή που η εφαρμογή εκτελείται, δηλαδή η περιγραφή «διερμηνεύεται» χωρίς να μετατρέπεται σε κώδικα κάποιας γλώσσας. Προφανώς, η

δεύτερη μέθοδος προσφέρει μεγαλύτερη ευελιξία, καθώς επιτρέπει να αλλάζει πιο εύκολα και γρήγορα η σχεδίαση χωρίς αλλαγές στη λογική. Φυσικά, η πρώτη μέθοδος προσφέρει μεγαλύτερη απόδοση από την δεύτερη αφού δεν υπάρχει η επιβάρυνση της διερμηνείας της περιγραφής σε χρόνο εκτέλεσης.

Ακολουθεί μία σύντομη περιγραφή των γνωστών σήμερα γλωσσών περιγραφής Διεπαφών Χρήστη. Θα αναφερθούμε σε ποιες γλώσσες προγραμματισμού μπορεί να υλοποιήσει τη λογική ο προγραμματιστής, αν είναι διερμηνευόμενη ή μεταγλωττιζόμενη και για ποιο είδος εφαρμογών χρησιμοποιείται.

3.1 Γλώσσες Περιγραφής για Εφαρμογές Παγκόσμιου Ιστού

Στον χώρο των εφαρμογών Διαδικτύου, οι γλώσσες που υπάρχουν προσφέρουν χαρακτηριστικά Γρήγορης Ανάπτυξης Εφαρμογών - (RAD - Rapid Application Development) κυρίως για εργαλεία όπως ο Flash Player της Adobe ή για AJAX εφαρμογές.

3.1.1 BXML

Είναι εμπορική γλώσσα που αναπτύσσεται από την Backspace για τη σχεδίαση AJAX εφαρμογών στο δικό της περιβάλλον ανάπτυξης BPC[23]. Είναι διερμηνευόμενη και ως γλώσσα υλοποίησης της λογικής μπορεί να χρησιμοποιηθεί μόνο JavaScript.

3.1.2 MXML

Είναι γλώσσα της Macromedia (τώρα Adobe) για τη σχεδίαση εφαρμογών για τον Flash Player[24]. Είναι μεταγλωττιζόμενη και ως γλώσσα υλοποίησης της λογικής χρησιμοποιεί την ActionScript.

3.1.3 OpenLaszlo

Αναπτύσσεται από την Laszlo Systems για Flash Player εφαρμογές[25]. Είναι μεταγλωττιζόμενη και για την υλοποίηση της λογικής χρησιμοποιείται η JavaScript.

3.1.4 XUL

Σημαίνει XML User-interface Language και πρόκειται για την κύρια γλώσσα που χρησιμοποιούν οι Διαδικτυακές εφαρμογές του οργανισμού Mozilla[6]. Ο XUL κώδικας είναι διερμηνευόμενος. Μέσα στο XML αρχείο μπορεί να ενσωματωθεί Javascript κώδικας για να ορίζει τις συναρτήσεις που καλούνται από διάφορα γεγονότα (πχ click σε πλήκτρο εντολής), και γενικά συνεργάζεται με υπάρχοντα πρότυπα και τεχνολογίες όπως CSS, DTD και RDF, που την κάνει πιο εύκολη στην εκμάθηση, αν έχει κανείς εμπειρία σε προγραμματισμό και σχεδίαση εφαρμογών Παγκόσμιου Ιστού. Κύρια χρήση της είναι η ανάπτυξη Διαδικτυακών εφαρμογών που τρέχουν μέσα από τον Mozilla browser και έχουν την αισθητική (look & feel) του λειτουργικού συστήματος στο οποίο τρέχει (δηλαδή τα Γραφικά Στοιχεία που εμφανίζονται είναι τα ίδια με αυτά που χρησιμοποιεί το περιβάλλον στο οποίο τρέχει ο φυλλομετρητής (browser». Μπορεί υπό κάποιες συνθήκες να χρησιμοποιηθεί για την ανάπτυξη εφαρμογών που εκτελούνται και έξω από το περιβάλλον του Mozilla browser και να χρησιμοποιηθεί ως γλώσσα υλοποίησης της λογικής η C++.

3.2 Γλώσσες Περιγραφής για Εφαρμογές Επιφάνειας Εργασίας

Οι κυριότεροι εκπρόσωποι των γλωσσών περιγραφής που χρησιμοποιούνται για ανάπτυξη Εφαρμογών Επιφάνειας Εργασίας - (Desktop Applications εφαρμογών είναι η GladeXML και η XAML. Θα εστιάσουμε σε αυτές περισσότερο από τις υπόλοιπες καθώς είναι οι πιο πλούσιες σε χαρακτηριστικά.

3.2.1 GladeXML

Η GladeXML είναι η XML δομή που χρησιμοποιεί ο Glade Σχεδιαστής Διεπαφών (Glade Interface Designer), ένα εργαλείο σχεδίασης διεπαφών για GTK+ και GNOME εφαρμογές, για να αποθηκεύει τις φόρμες του[26]. Τα GladeXML αρχεία μπορούν να μεταγλωττιστούν σε C/C++ κώδικα ή μπορούν να διερμηνεύονται από την libglade βιβλιοθήκη, το οποίο είναι και το προτεινόμενο στην πιο πρόσφατη έκδοση[27]. Χρησιμοποιώντας την libglade, τα GladeXML αρχεία είναι δυνατόν να χρησιμοποιηθούν με πληθώρα γλωσσών προγραμματισμού όπως C, C++, Java, Perl, Python, C#, Pike, Ruby, Haskell, Objective Caml και Scheme (για αυτές τις γλώσσες υπάρχουν αντιστοιχίσεις (bindings) για τη βιβλιοθήκη).

Μέσω του Glade Σχεδιαστή Διεπαφών ο σχεδιαστής δημιουργεί φόρμες (τα παράθυρα, διάλογοι), που θα αποτελέσουν τη διεπαφή της εφαρμογής, χρησιμοποιώντας τα Γραφικά Στοιχεία του GTK+. Για να ορίσει τις αλληλεπιδράσεις των Γραφικών Στοιχείων ορίζει χειριστές σημάτων (signal handlers). Κάθε Γραφικό Στοιχείο έχει προκαθορισμένα σήματα το καθένα από τα οποία αντιστοιχεί σε ένα γεγονός του περιβάλλοντος. Οι χειριστές σημάτων αντιστοιχούν σε συναρτήσεις που υλοποιεί ο προγραμματιστής και ενσωματώνουν τη λογική της εφαρμογής. Μέσα στη λογική αυτή ο προγραμματιστής θα πρέπει να ορίσει τις οποιεσδήποτε αλλαγές στη διεπαφή που μπορούν να προκύψουν κατά την εκτέλεση της εφαρμογής, π.χ. την εμφάνιση ενός παραθύρου ή την αλλαγή του χρώματος κάποιου κειμένου.

Ο Glade Σχεδιαστής Διεπαφών χρησιμοποιεί Γραφικά Στοιχεία του GTK+ ή του GNOME και δεν έχει επιπλέον γραφικές δυνατότητες. Δηλαδή, ο σχεδιαστής δεν μπορεί να ορίσει ή να αλλάξει την εμφάνιση κάποιου Γραφικού Στοιχείου ή να σχεδιάσει οποιουδήποτε είδους γραφικά. Η εμφάνιση των Γραφικών Στοιχείων εξαρτάται από το θέμα (theme) που χρησιμοποιεί γενικότερα η GTK+ βιβλιοθήκη τα οποία αποτελούνται από pixmap γραφικά. Οι GTK+ και libglade βιβλιοθήκες καθώς και ο Glade Σχεδιαστής Διεπαφών έχουν μεταφερθεί σε πληθώρα λειτουργικών συστημάτων και αρχιτεκτονικών με ή χωρίς

X11, POSIX ή Windows, λόγω της αρχιτεκτονικής ανοιχτού κώδικα που ακολουθούν.

3.2.2 XAML

Πρόκειται για την Επεκτάσιμη Γλώσσα Σήμανσης Εφαρμογών (Extensible Application Markup Language) και είναι προϊόν της Microsoft[28]. Για την ακρίβεια, είναι μέρος του Windows Presentation Foundation (ή WPF) δηλαδή του υποσυστήματος για γραφικά του .NET Framework 3.0. Ως γλώσσα υλοποίησης της λογικής μπορεί να χρησιμοποιηθεί οποιαδήποτε .NET γλώσσα και οι XAML περιγραφές μεταγλωττίζονται σε .NET κώδικα. Το βασικό χαρακτηριστικό της γλώσσας είναι πως κάθε στοιχείο της (κάθε tag) αντιστοιχεί σε ένα αντικείμενο μιας .NET κλάσης και κάθε χαρακτηριστικό (attribute) του στοιχείου αντιστοιχεί σε μία ιδιότητα του αντικειμένου.

Ομοίως με την GladeXML, για τον ορισμό των αλληλεπιδράσεων χρησιμοποιούνται γεγονότα (events) και χειριστές γεγονότων (event handlers). Μπορεί να γραφτεί και κώδικας της λογικής μέσα στο ίδιο το XAML αρχείο. Οι χειριστές γεγονότων είναι μέθοδοι των κλάσεων που χρησιμοποιούνται στην XAML περιγραφή. Ο προγραμματιστής υλοποιεί στους χειριστές γεγονότων τη λογική της εφαρμογής συμπεριλαμβανομένων και αλλαγών στη διεπαφή κατά την εκτέλεση όπως εμφάνιση παραθύρων, διαλόγων ή αλλαγή ιδιοτήτων των Γραφικών Στοιχείων[7].

Γενικότερα, χρησιμοποιούνται διανυσματικά (vector) γραφικά για την εμφάνιση των Γραφικών Στοιχείων. Κάθε Γραφικό Στοιχείο έχει μια προεπιλεγμένη εμφάνιση ανάλογα με κάποιο θέμα (theme) αλλά ο σχεδιαστής έχει τη δυνατότητα να αλλάξει την εμφάνισή του με γραφικά δικής του σχεδίασης. Γενικότερα, η XAML δίνει τη δυνατότητα δημιουργίας και πολύπλοκων διανυσματικών γραφικών. Η XAML είναι στενά συνδεδεμένη με το .NET και χρησιμοποιείται για εφαρμογές επιφάνειας εργασίας. Ένα υποσύνολο της XAML είναι διερμηνευόμενο και μπορεί να χρησιμοποιηθεί και για εφαρμογές Διαδικτύου.

3.3 Πλεονεκτήματα

Η χρήση αυτών των γλωσσών προσφέρει ακόμα μεγαλύτερο διαχωρισμό μεταξύ σχεδίασης και ανάπτυξης της εφαρμογής. Ο σχεδιασμός της Διεπαφής Χρήστη και η ανάπτυξη της λογικής της εφαρμογής μπορούν να γίνουν ξεχωριστά από ειδικευμένα σε αυτούς τους τομείς πρόσωπα. Επιταχύνεται η διαδικασία ανάπτυξης μιας εφαρμογής καθώς δεν χρειάζεται να γραφτεί κώδικας σε κάποια γλώσσα προγραμματισμού (σε συγκεκριμένη εργαλειοθήκη Γραφικών Στοιχείων για να υλοποιηθεί η Διεπαφή Χρήστη, παρά μόνο ο XML κώδικας. Με αυτό τον τρόπο ο χρόνος που κερδίζεται μπορεί να επενδυθεί στην ανάπτυξη της λογικής της εφαρμογής βελτιώνοντας έτσι την εφαρμογή γενικότερα. Ειδικότερα, στις γλώσσες που είναι διερμηνευόμενες μπορούν να δημιουργηθούν πολλές διαφορετικές Διεπαφές Χρήστη για την ίδια εφαρμογή και να αλλάζουν δυναμικά ανάλογα με τις προτιμήσεις του χρήστη (themes ή skins), εύκολα. Κάποιες γλώσσες, όπως η XAML, δεν περιορίζονται μόνο στα προεγκατεστημένα Γραφικά Στοιχεία, αλλά προσφέρουν εκτεταμένες δυνατότητες δημιουργίας ελκυστικών γραφικών, αλλαγής της εμφάνισης των Γραφικά Στοιχεία, χρήσης εντυπωσιακών οπτικών εφέ και πολυμεσικές (multimedia) δυνατότητες[7], αυξάνοντας έτσι το επίπεδο της χρηστικότητας της εφαρμογής και της φιλικότητας και ελκυστικότητας προς το χρήστη.

Κεφάλαιο 4

Ελλείψεις XML Γλωσσών Περιγραφής Διεπαφών Χρήστη

Παρά τα πολλά πλεονεκτήματα που έχουν οι XML Γλώσσες Διεπαφής Χρήστη, παρατηρώντας προσεκτικά τις δυνατότητες που προσφέρουν, εντοπίζουμε μία σειρά από μειονεκτήματα και ελλείψεις. Τα χαρακτηριστικά αυτά εμποδίζουν τις γλώσσες να αποσυνδέσουν τη σχεδίαση της διεπαφής μέσω της XML γλώσσας από την επιχειρησιακή λογική και τη συγγραφή κώδικα έτσι ώστε να μπορεί να γίνει από ένα εξειδικευμένο άτομο (σχεδιαστή).

4.1 Δέσμευση Περιβάλλοντος Ανάπτυξης

Όλα τα συστήματα που περιγράψαμε στο κεφάλαιο 3 παρουσιάζουν στενή εξάρτηση της XML γλώσσας σχεδίασης με ένα συγκεκριμένα περιβάλλον ανάπτυξης της επιχειρησιακής λογικής της εφαρμογής[10]. Αναφέραμε πως κάθε γλώσσα έχει αναπτυχθεί για μία συγκεκριμένη εργαλειοθήκη Γραφικών Στοιχείων. Για παράδειγμα, η XAML έχει αναπτυχθεί για το WPF, η GladeXML για το Gtk+ και το GNOME και η XUL για εφαρμογές του Mozilla. Η κάθε εργαλειοθήκη όμως είναι ανεπτυγμένη για συγκεκριμένες γλώσσες προγραμματισμού. Για παράδειγμα, η XAML για γλώσσες του περιβάλλοντος .NET ενώ η GladeXML

για C και για όποιες γλώσσες υπάρχουν αντιστοιχίσεις (bindings) του GTK+ API (C++, python κ.α.).

Η επιλογή μιας XML γλώσσας περιορίζει αυτόματα την/τις γλώσσες προγραμματισμού που μπορούν να χρησιμοποιηθούν και αντίστροφα. Αυτό γιατί, οι χειριστές γεγονότων θα πρέπει να υλοποιηθούν σε μια γλώσσα προγραμματισμού που υποστηρίζεται από την εργαλειοθήκη. Ο περιορισμός αυτός είναι ένα μειονέκτημα καθώς η επιχειρησιακή λογική της εφαρμογής μπορεί να είναι πιο εύκολο να υλοποιηθεί σε γλώσσα διαφορετική από αυτές που υποστηρίζονται. Επιπλέον, γίνεται πιο δύσκολο να χρησιμοποιηθούν ως κομμάτι της έτοιμες βιβλιοθήκες υλοποιημένες σε διαφορετικές γλώσσες. Αυτού του είδους η δέσμευση στο περιβάλλον ανάπτυξης σημαίνει πως η λογική παρουσίασης δεν είναι ανεξάρτητη της επιχειρησιακής λογικής.

4.2 Διάσπαση Λογικής Παρουσίασης

Αναφέραμε προηγουμένως πως σκοπός των XML γλωσσών Διεπαφών Χρήστη είναι να εκφράζεται η λογική παρουσίασης με τη δηλωτική σύνταξη της XML ώστε να αποσυνδέεται από τον κώδικα σε μια διαδικαστική γλώσσα, δηλαδή από την επιχειρησιακή λογική. Δεν επιτυγχάνεται όμως αυτή η αποσύνδεση πλήρως[10]. Στην πραγματικότητα, ό,τι έχουμε αναφέρει στο κεφάλαιο 2 για τους χειριστές γεγονότων και τα προβλήματα σύζευξης μεταξύ λογικής παρουσίασης και επιχειρησιακής λογικής εξακολουθούν να ισχύουν. Αυτό γιατί, όπως αναφέραμε στις προηγούμενες ενότητες, η περιγραφή της αντίδρασης στην εμφάνιση κάποιου γεγονότος γίνεται με τον ορισμό σε κάποιο χαρακτηριστικό ή κάποιο στοιχείο της γλώσσας του χειριστή γεγονότος που θα εκτελεστεί.

Στην ενότητα 2.6 ορίσαμε τη λογική παρουσίασης ως τους κανόνες μετάβασης των καταστάσεων του δέντρου των Γραφικών Στοιχείων της Διεπαφής. Με βάση αυτό τον ορισμό, όμως, παρατηρούμε ότι η προσέγγιση των XML γλωσσών διασπά τη λογική παρουσίασης σε δύο κομμάτια. Το ένα είναι ο ορισμός του δέντρου των Γραφικών Στοιχείων της Διεπαφής με την XML γλώσσα. Το

άλλο είναι ο ορισμός των μεταβάσεων κατάστασης του δέντρου στον κώδικα των χειριστών γεγονότων. Στην πρώτη περίπτωση, χρησιμοποιείται μία δηλωτική περιγραφική γλώσσα. Στη δεύτερη, μία κανονική διαδικαστική[1] γλώσσα προγραμματισμού.

Πριν την εμφάνιση XML γλωσσών Διεπαφών Χρήστη, ο ορισμός του δέντρου γινόταν στην ίδια γλώσσα με τους χειριστές γεγονότων και αυτή η διάσπαση της λογικής παρουσίασης δεν ίσχυε. Και στις δύο περιπτώσεις όμως, για να αναλάβει κάποιος τη σχεδίαση/ανάπτυξη της λογικής παρουσίασης, θα πρέπει να έχει γνώσεις προγραμματισμού, γνώση της εργαλειοθήκης που χρησιμοποιείται, ή να συνεννοηθεί με το άτομο που έχει αναλάβει την ανάπτυξη της επιχειρησιακής λογικής ώστε να υλοποιήσει τη συμπεριφορά που επιθυμεί. Κατά τα άλλα, εξακολουθεί να υπάρχει η δυσκολία στην τροποποίηση μιας Διεπαφής ορισμένης σε μια τέτοια γλώσσα. Αν αποφασιστεί να τροποποιηθεί κατά κάποιο τρόπο η λογική παρουσίασης στην XML περιγραφή, οι αλλαγές αυτές θα πρέπει να συνοδευτούν με αντίστοιχες αλλαγές στον κώδικα των χειριστών γεγονότων, ο οποίος εξαρτάται από το δέντρο των Γραφικών Στοιχείων.

4.3 Συμπέρασμα: Μη Πλήρης Διαχωρισμός Λογικής Παρουσίασης και Επιχειρησιακής Λογικής

Σύμφωνα με την ανάλυση που κάναμε στις δύο προηγούμενες ενότητες, καταλήγουμε στο συμπέρασμα ότι οι XML γλώσσες περιγραφής Γραφικών Διεπαφών Χρήστη που χρησιμοποιούνται σήμερα δεν καθιστούν τη λογική παρουσίασης ανεξάρτητη της επιχειρησιακής λογικής (και αντίστροφα).

Μπορούμε να συνοψίσουμε τα αίτια στα οποία οφείλεται αυτό το γεγονός

[1]Στο συγκεκριμένο σημείο δεν χρησιμοποιούμε τον όρο «διαδικαστικός» για να κάνουμε διάκριση μεταξύ κλασικών διαδικαστικών γλωσσών όπως C και αντικειμενοστραφών γλωσσών όπως Java. Αλλά, για να δώσουμε έμφαση στη διαδικαστική λογική των χειριστών γεγονότων που ορίζονται σε αυτές, σε αντίθεση με το δηλωτικό χαρακτήρα της XML.

στα παρακάτω δύο σημεία:

1. Στην δέσμευση στο περιβάλλον ανάπτυξης της επιχειρησιακής λογικής που προκαλεί η επιλογή μιας XML γλώσσας για τον ορισμό της λογικής παρουσίασης.

2. Και, στο γεγονός ότι η λογική παρουσίασης ορίζεται σε δύο σημεία. Στην XML γλώσσα και στους χειριστές γεγονότων.

4.4 Προδιαγραφές Πλήρους Διαχωρισμού

Στο κεφάλαιο 2, και συγκεκριμένα στην ενότητα 2.7, αναδείξαμε την ανάγκη διαχωρισμού της λογικής παρουσίασης και της επιχειρησιακής λογικής μιας εφαρμογής στο σημείο που η μία να είναι ανεξάρτητη της άλλης. Στην προηγούμενη ενότητα καταλήξαμε στο συμπέρασμα πως, αν και οι XML γλώσσες περιγραφής Γραφικών Διεπαφών Χρήστη διαχωρίζουν τις δύο λογικές, δεν τις διαχωρίζουν πλήρως. Τώρα θα προσπαθήσουμε να ορίσουμε τις προδιαγραφές που θα πρέπει να έχει μία XML γλώσσα και το σύστημα λογισμικού[2] που υλοποιεί τις λειτουργίες της, ώστε να πετυχαίνει τον πλήρη διαχωρισμό και ανεξαρτησία της λογικής παρουσίασης και της επιχειρησιακής λογικής.

Παρακάτω παραθέτουμε αυτές τις προδιαγραφές:

1. Όπως οι υπάρχουσες γλώσσες, να ορίζει με τα στοιχεία της το δέντρο των Γραφικών Στοιχείων της Διεπαφής.

2. Να μην επιτρέπεται στον κώδικα που υλοποιεί την επιχειρησιακή λογική να έχει πρόσβαση στο δέντρο των Γραφικών Στοιχείων. Γενικότερα, να μην επιτρέπεται η ανάπτυξη κανενός είδος κώδικα σε γλώσσα προγραμματισμού που να έχει αυτή την πρόσβαση.

[2]Στις γλώσσες που χρησιμοποιούνται σήμερα αυτά τα συστήματα είναι οι εργαλειοθήκες Γραφικών Διεπαφών για τις οποίες δημιουργήθηκαν.

3. Οι τροποποιήσεις του δέντρου των Γραφικών Στοιχείων, όταν συμβεί κάποιο γεγονός, να περιγράφονται δηλωτικά με την XML σύνταξη της γλώσσας.

4. Όπως και στον ορισμό της αρχιτεκτονικής 3-επιπέδων (βλέπε ενότητα 2.8.1), να ορίζεται μια κοινή διεπαφή, ένα πρωτόκολλο επικοινωνίας, μεταξύ της λογικής παρουσίασης και της επιχειρησιακής λογικής, ανεξάρτητη της υλοποίησής τους.

5. Η επιχειρησιακή λογική να εξάγει τις λειτουργίες, τους επιχειρηματικούς κανόνες, που υλοποιεί, με τρόπο κατανοητό από τον άνθρωπο.

6. Να δίνεται η δυνατότητα, με δηλωτικό τρόπο, μέσα σε ένα έγγραφο της γλώσσας, ως ενέργεια χειρισμού γεγονότος, να οριστεί η χρήση μιας από τις λειτουργίες που εξάγει η επιχειρησιακή λογική.

7. Να επιτρέπεται η υλοποίηση της επιχειρησιακής λογικής σε διαφορετικά περιβάλλοντα αρκεί να υλοποιείται η συμφωνημένη διεπαφή.

8. Η γλώσσα να είναι διερμηνευόμενη. Έτσι, η λογική παρουσίασης θα μπορεί να αλλάξει πολύ εύκολα με αλλαγές στο XML κείμενο, χωρίς να χρειαστεί κάποια διαδικασία μεταγλώττισης, η οποία μπορεί να απαιτεί γνώση του συστήματος στο οποία γίνεται η ανάπτυξη (Windows, Unix). Με αυτό τον τρόπο η λογική παρουσίασης θα είναι ανεξάρτητη συστήματος, όπως γίνεται στην HTML.

Οι γενικές προδιαγραφές που ορίστηκαν παραπάνω, είναι αρκετές ώστε να επιτευχθεί η ανεξαρτησία της λογικής παρουσίασης και επιχειρησιακής λογικής σε μια XML γλώσσα Διεπαφών Χρήστη. Όμως, μια απλή αναφορά των προδιαγραφών δεν είναι απόδειξη ότι κάτι τέτοιο είναι εφικτό. Στα επόμενα κεφάλαια, θα ορίσουμε μια XML γλώσσα που θα πληροί αυτές τις προδιαγραφές ως απόδειξη.

Κεφάλαιο 5

XAIL

Οι αδυναμίες που περιγράψαμε στο κεφάλαιο 4 μας οδηγούν στο να προτείνουμε μια καινούργια XML γλώσσα περιγραφής Γραφικών Διεπαφών Χρήστη, βασισμένη στις προδιαγραφές που θέσαμε στο προηγούμενο κεφάλαιο, η οποία τις επιλύει. Τη γλώσσα αυτή την ονομάζουμε XAIL, που είναι αρχικά για το eXtensible Application Interface Language (Επεκτάσιμη Γλώσσα Διεπαφών Εφαρμογών). Στις επόμενες δύο ενότητες θα περιγράψουμε τα χαρακτηριστικά της γλώσσας και την αρχιτεκτονική του συστήματός της. Στη συνέχεια θα περιγράψουμε τις προδιαγραφές της γλώσσας και το συντακτικό της.

5.1 Χαρακτηριστικά

Το συντακτικό και τα στοιχεία της γλώσσας αυτής είναι όμοια με αυτά των υπόλοιπων γλωσσών. Εισάγουμε όμως δύο χαρακτηριστικά τα οποία αντικρούουν τα δύο κύρια σημεία στα οποία εστιάσαμε τις ελλείψεις των υπαρχουσών γλωσσών που αναφέραμε στο προηγούμενο κεφάλαιο.

Αρχικά, η περιγραφή της Διεπαφής Χρήστη μέσω της γλώσσας αποσυνδέεται πλήρως από τη συγγραφή κώδικα σε κάποια γλώσσα προγραμματισμού. Η γλώσσα καλύπτει όλο το κομμάτι της λογικής παρουσίασης της εφαρμογής, και η επιχειρησιακή λογική είναι ανεξάρτητη. Ο σχεδιαστής δεν χρειάζεται να

γράψει κώδικα σε κάποια γλώσσα προγραμματισμού για χειρισμό γεγονότων ώστε να αλλάξει τα Γραφικά Στοιχεία της Διεπαφής Χρήστη, μπορεί πολύ απλά να το εκφράσει σε XML. Η γλώσσα παρέχει σύνταξη για την τροποποίηση του XML δέντρου των Γραφικών Στοιχείων στην περίπτωση που συμβεί κάποιο γεγονός. Δηλαδή θα ορίζεται για κάποιο γεγονός σε κάποιο Γραφικό Στοιχείο το πώς θα αλλάξει το δέντρο των Γραφικών Στοιχείων και ποια κομμάτια της επιχειρησιακής λογικής θα χρησιμοποιηθούν για αυτό. Με άλλα λόγια, ο χειριστής γεγονότος ορίζεται μέσα στην ίδια XML περιγραφή. Η επικοινωνία μεταξύ της λογικής παρουσίασης και της επιχειρησιακής λογικής γίνεται σε αυτούς τους ορισμούς. Εκεί που θα απαιτηθεί να εκτελεστεί κώδικας της επιχειρησιακής λογικής, θα ορίζεται μία κλήση σε κάποιο αντικείμενο ενός κομματιού της επιχειρησιακής λογικής.

Η επιχειρησιακή λογική που χρησιμοποιείται από τη λογική παρουσίασης σε κάποια εφαρμογή οργανώνεται σε συστατικά στοιχεία λογισμικού (components). Το κάθε συστατικό υλοποιεί κάποια συγκεκριμένη λειτουργικότητα. Για παράδειγμα, ένα μπορεί να υλοποιεί διαχείριση αρχείων και καταλόγων, άλλο να υλοποιεί την αποθήκευση ρυθμίσεων της εφαρμογής και ένα άλλο την ανάκτηση/αποθήκευση δεδομένων που χειρίζεται η εφαρμογή. Επιτρέπεται κάθε συστατικό να είναι υλοποιημένο σε ξεχωριστή γλώσσα προγραμματισμού, να είναι υλοποιημένο ως δυναμική βιβλιοθήκη ή ως ξεχωριστό πρόγραμμα ή να βρίσκεται σε διαφορετικό υπολογιστή.

Για να μπορέσει η λογική παρουσίασης να επικοινωνήσει με αυτά τα πολύ διαφορετικά συστατικά λογισμικού της επιχειρησιακής λογικής χρησιμοποιείται μια ¨κοινή γλώσσα' μεταξύ του προγραμματιστή της λογικής της εφαρμογής και του σχεδιαστή. Αυτή είναι η περιγραφή της διεπαφής (interface) που θα εξάγει το κάθε συστατικό στο οποίο θα ορίζεται η λειτουργικότητα που προσφέρει. Η γλώσσα που έχουμε επιλέξει για την περιγραφή των διεπαφών είναι η IDL, μια αντικειμενοστραφής γλώσσα περιγραφής διεπαφών την οποία εξηγούμε στην ενότητα 6.6. Έτσι, αν ο σχεδιαστής γνωρίζει τη διεπαφή ενός συστατικού μπορεί να περιγράψει μέσω της XAIL μία κλήση προς μία λειτουργία, ενός

αντικειμένου, αυτού του συστατικού.

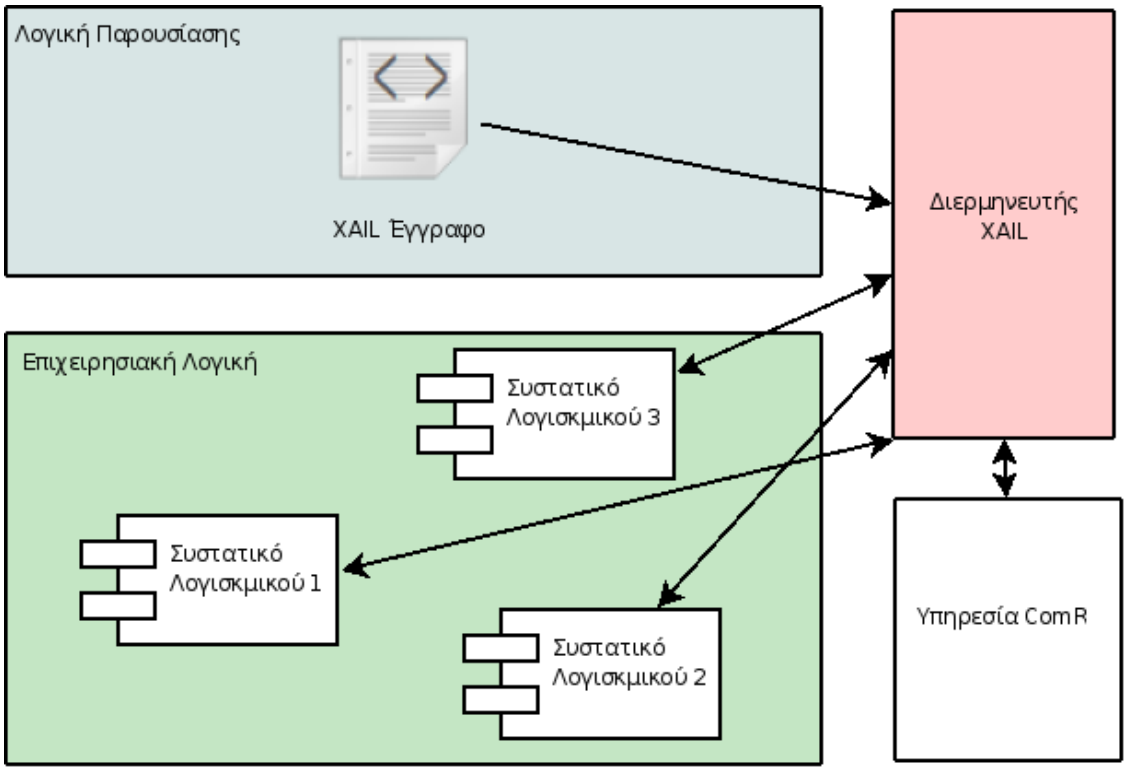
Η XAIL γλώσσα είναι διερμηνευόμενη. Δηλαδή ένα κομμάτι λογισμικού, ο διερμηνευτής, διαβάζει την περιγραφή και ανάλογα εμφανίζει τη Διεπαφή Χρήστη στο χρήστη και αντιδρά στα γεγονότα. Ο σχεδιαστής μπορεί μέσα στην περιγραφή να ορίζει ποια συστατικά θα χρησιμοποιηθούν και ποιες κλήσεις θα γίνονται σε αυτά.

5.2 Αρχιτεκτονική

Σε αυτό το σημείο, μετά την περιγραφή των χαρακτηριστικών της γλώσσας στην προηγούμενη ενότητα, θα περιγράψουμε την αρχιτεκτονική μιας εφαρμογής που έχει σχεδιαστεί με τη χρήση της γλώσσας XAIL. Στο σχήμα 5.1 φαίνεται ένα διάγραμμα της αρχιτεκτονικής αυτής.

Η λογική παρουσίασης της εφαρμογής ορίζεται σε ένα XAIL έγγραφο. Για να εκτελεστεί η εφαρμογή το έγγραφο διαβάζεται από έναν XAIL διερμηνευτή. Ο ρόλος του διερμηνευτή είναι να διαβάσει το δέντρο Γραφικών Στοιχείων που ορίζεται στο έγγραφο και να το εμφανίσει στην οθόνη. Στη συνέχεια περιμένει να συμβεί κάποιο γεγονός, οπότε και εκτελεί τις ενέργειες που έχουν δηλωθεί στο έγγραφο για το συγκεκριμένο γεγονός. Οι ενέργειες αυτές μπορούν να είναι συνδυασμός από τροποποιήσεις του δέντρου των Γραφικών Στοιχείων της Διεπαφής και κλήσεις σε κάποιο από τα συστατικά λογισμικού που χρησιμοποιεί η εφαρμογή στην επιχειρησιακή λογική της.

Τα συστατικά λογισμικού που αποτελούν την επιχειρησιακή λογική ορίζονται και αυτά στο XAIL έγγραφο. Ο διερμηνευτής αναλαμβάνει να ενεργοποιήσει αυτά τα συστατικά λογισμικού και να τα συνδέσει στην εφαρμογή. Για το σκοπό αυτό χρησιμοποιεί μια υπηρεσία, το ComR, η οποία παίζει το ρόλο του διαχειριστή των συστατικών λογισμικού. Η διαχείριση των συστατικών γίνεται σύμφωνα με το Μοντέλο Συστατικών XAIL. Η υπηρεσία ComR, η αρχιτεκτονική της καθώς και το Μοντέλο Συστατικών XAIL ορίζονται και περιγράφονται στα κεφάλαια 7.1 και 7 αντίστοιχα.



Σχήμα 5.1: Διάγραμμα αρχιτεκτονικής μιας XAIL εφαρμογής.

5.3 Προδιαγραφές Γλώσσας XAIL

5.3.1 Γενικά

Έχουμε σχεδιάσει και υλοποιήσει μια βασική και λειτουργική μορφή της γλώσσας περιγραφής διεπαφής XAIL. Ο σκοπός μας είναι να παρουσιαστεί η ιδέα μέσα από μια μελέτη περίπτωσης (case study), και όχι να κατασκευάσουμε ένα πλήρες λογισμικό έτοιμο προς χρήση από το ευρύ κοινό. Αν και κάτι τέτοιο θα ήταν επιθυμητό, ξεφεύγει από τα πλαίσια της παρούσας πτυχιακής, παρόλα αυτά, το υπάρχον σύστημα θέτει τις απαραίτητες βάσεις για περαιτέρω ανάπτυξη και επέκταση. Σε επόμενο κεφάλαιο θα δούμε δυνατές επεκτάσεις που μπορούν να κάνουν τη γλώσσα αυτή πιο πλήρη και αποτελεσματική, αλλά και πιο καινοτόμο.

5.3.2 Γραφικά Στοιχεία Διεπαφής

Τα Γραφικά Στοιχεία (widgets) είναι το βασικότερο συστατικό μιας εφαρμογής με διεπαφή (σε αντίθεση με την εφαρμογή κονσόλας). Χωρίς αυτά ο χρήστης του προγράμματος δεν μπορεί να κάνει τίποτα, γιατί δεν μπορεί ούτε να λάβει κάποια πληροφορία από το πρόγραμμα, αλλά ούτε να δώσει κάποια πληροφορία σε αυτό.

Ο χρήστης λαμβάνει πληροφορία από τη διεπαφή με διάφορους τρόπους. Είτε μέσω των ετικετών όπου υπάρχει κείμενο το οποίο διαβάζει, είτε μέσω εικόνων, είτε μέσω κάποιας μπάρας προόδου (progress bar), είτε μέσω αλλαγών στα Γραφικά Στοιχεία πχ. με αλλαγή κάποιου χρώματος από πράσινο σε κόκκινο, ο χρήστης μπορεί να καταλάβει ότι κάτι πήγε στραβά.

Επίσης ο χρήστης πρέπει να μπορεί να δώσει κάποια πληροφορία στην εφαρμογή. Είτε με το πάτημα ενός πλήκτρου εντολής, είτε με το να προκαλέσει οποιοδήποτε άλλο γεγονός, ο χρήστης μπορεί να ειδοποιήσει την εφαρμογή ότι κάποια διαδικασία πρέπει να εκτελεστεί άμεσα. Άλλος τρόπος μεταφοράς πληροφορίας στην εφαρμογή είναι η εισαγωγή κειμένου, τα κουμπιά πολλαπλής επιλογής, αλλά και πιο εξειδικευμένα Γραφικά Στοιχεία όπως ο επιλογέας

χρώματος, και πολλά άλλα.

Επίσης υπάρχουν και μερικά Γραφικά Στοιχεία τα οποία δεν έχουν ως σκοπό την ανταλλαγή δεδομένων μεταξύ διεπαφής και χρήστη, αλλά χρησιμεύουν στην ομαδοποιημένη εμφάνιση των υπόλοιπων γραφικών αντικειμένων, δηλαδή σκοπός τους είναι να παρέχουν μια αισθητικά αποδεκτή εμφάνιση της εφαρμογής. Παραδείγματα τέτοιων Γραφικών Στοιχείων είναι ο πίνακας, η οριζόντια και κάθετη ομαδοποίηση, τα πλαίσια και άλλα.

Προς το παρόν για τη γλώσσα XAIL έχουμε υλοποιήσει ενδεικτικά το Γραφικό Στοιχείο του κεντρικού παραθύρου, την ετικέτα κειμένου, το πλήκτρο εντολής, το μενού, την κάθετη και οριζόντια ομαδοποίηση, καθώς και τέσσερα γεγονότα με τα οποία ο χρήστης μπορεί να κάνει εκτέλεση κώδικα, και να διαγράψει, εισαγάγει, ή να αλλάξει τα Γραφικά Στοιχεία της διεπαφής.

Παρακάτω περιγράφουμε τα υλοποιημένα Γραφικά Στοιχεία και γεγονότα, και δείχνουμε πως συντάσσονται και χρησιμοποιούνται μέσα από παραδείγματα.

Κεντρικό Παράθυρο

Το κεντρικό παράθυρο είναι σημαντικό στοιχείο μιας διεπαφής καθώς αυτό περιέχει όλα τα υπόλοιπα Γραφικά Στοιχεία και τα γεγονότα ορισμένα για τη διεπαφή. Είναι απαραίτητο στοιχείο και κάθε εφαρμογή σε XAIL πρέπει να το περιέχει. Ορίζεται με το στοιχείο <Window>, βλέπε σχήμα 5.2 και μπορεί να πάρει το προαιρετικό χαρακτηριστικό ′title′ με το οποίο δίνεται η δυνατότητα να οριστεί ο τίτλος του παραθύρου.

```
<Window title="New XAIL Application">
    ... definition of widgets, events ...
</Window>
```

Σχήμα 5.2: Παράδειγμα: Ορισμός κεντρικού παραθύρου.

Πλήκτρο εντολής

Η δημιουργία ενός πλήκτρου εντολής (button) είναι πολύ απλή και γίνεται με το στοιχείο <Button> με κείμενο τον τίτλο του πλήκτρου που θα βλέπει ο χρήστης. Ένα απλό παράδειγμα φαίνεται στο σχήμα 5.3.

```
<Button>Cancel</Button>
```

Σχήμα 5.3: Παράδειγμα: Δημιουργία πλήκτρου εντολής με κείμενο 'Cancel'

Μενού

Τα μενού δημιουργούνται χρησιμοποιώντας τρία στοιχεία, το <MenuBar>, το <Menu> και το <Item>. Το στοχείο <MenuBar> δημιουργεί μια μπάρα για να μπουν τα μενού που θα δημιουργηθούν. Το στοιχείο <Menu> δημιουργεί το κάθε μενού το οποίο παίρνει τον τίτλο του από το χαρακτηριστικό 'label'. Μέσα στα μενού υπάρχουν οι διάφορες επιλογές των μενού οι οποίες συντάσσονται με το στοιχείο <Item> όπως φαίνεται στο απλό παράδειγμα του σχήματος 5.4.

```
<MenuBar>
    <Menu label="File">
        <Item>Open</Item>
        <Item>Quit</Item>
    </Menu>
    <Menu label="Help">
        <Item>About</Item>
        <Item>Help</Item>
    </Menu>
</MenuBar>
```

Σχήμα 5.4: Παράδειγμα: Δημιουργία μπάρας μενού με δύο μενού, τα 'File' και 'Help', που μέσα περιέχουν συνολικά τέσσερις επιλογές.

Ετικέτα

Για την παρουσίαση κειμένου χρειάζεται ένα Γραφικό Στοιχείο Ετικέτας. Αυτό συντάσσεται πολυ απλά με το στοιχείο <Label>. Ένα απλό παράδειγμα δίνεται στο σχήμα 5.5.

```
<Label>This is a label example</Label>
```

Σχήμα 5.5: Παράδειγμα: Ετικέτα.

Με το χαρακτηριστικό 'selectable' δίνεται η δυνατότητα να ρυθμίσει ο προγραμματιστής αν το κείμενο της ετικέτας μπορεί να επιλεγεί από τον χρήστη, ώστε να το κάνει copy. Ένα παράδειγμα χρήσης του χαρακτηριστικού φαίνεται στο σχήμα 5.6. Αν στο 'selectable' δοθεί η τιμή 'false' τότε το κείμενο δεν επιλέγεται, αλλιώς επιλέγεται.

```
<Label selectable="false">This text can not be selected.</Label>
<Label selectable="true">This text can be selected.</Label>
<Label>This text can be selected.</Label>
```

Σχήμα 5.6: Παράδειγμα: Χαρακτηριστικό ετικέτας 'selectable'.

Οριζόντια ομαδοποίηση

Το Γραφικό Στοιχείο που δίνει τη δυνατότητα να ομαδοποιηθούν κάποια άλλα Γραφικά Στοιχεία, ώστε να φαίνονται το ένα δίπλα στο άλλο στην ίδια σειρά, συντάσσεται με το στοιχείο <HBox>. Ένα παράδειγμα στο οποίο φαίνεται η χρήση του HBox μαζί με άλλα Γραφικά Στοιχεία δίνεται στο σχήμα 5.7.

Επίσης υπάρχει το χαρακτηριστικό 'homogenous' που με τιμές 'true' ή 'false' καθορίζεται αν τα περιεχόμενα Γραφικά Στοιχεία θα έχουν το ίδιο μέγεθος ή όχι.

Κάθετη ομαδοποίηση

Για να φαίνονται τα Γραφικά Στοιχεία το ένα κάτω από το άλλο χρειαζόμαστε την κάθετη ομαδοποίηση. Αυτή συντάσσεται με τον ίδιο τρόπο με την οριζόντια ομαδοποίηση, με τη διαφορά ότι τώρα το στοιχείο λέγεται <VBox>. Η οριζόντια και η κάθετη ομαδοποίηση μπορούν να συνδυαστούν μεταξύ τους ώστε να δημιουργηθεί το επιθυμητό αποτέλεσμα, όπως φαίνεται στο παράδειγμα του σχήματος 5.8.

Κοινά χαρακτηριστικά

Υπάρχουν μερικά χαρακτηριστικά τα οποία είναι κοινά σε δύο ή παραπάνω Γραφικά Στοιχεία και έχουν την ίδια λειτουργικότητα. Ένα τέτοιο χαρακτηριστικό, το οποίο είναι και αρκετά χρήσιμο, είναι το χαρακτηριστικό ′visible′. Αν πάρει την τιμή ′false′ τότε κάνει το συγκεκριμένο Γραφικό Στοιχείο αόρατο, σαν να μην υπήρχε, ενώ στην περίπτωση των Γραφικών Στοιχείων ομαδοποίησης, όλα

```
<HBox>
    <Label>Save changes?</Label>
    <Button>Yes</Button>
    <Button>No</Button>
</HBox>
```

Σχήμα 5.7: Παράδειγμα: Οριζόντια ομαδοποίηση τριών Γραφικών Στοιχείων.

```
<HBox>
    <Label>Save changes?</Label>
    <VBox>
        <Button>Yes</Button>
        <Button>No</Button>
    </VBox>
</HBox>
```

Σχήμα 5.8: Παράδειγμα: Σύνθετη ομαδοποίηση τριών Γραφικών Στοιχείων.

τα εσωτερικά Γραφικά Στοιχεία γίνονται αόρατα.

Άλλο χαρακτηριστικό είναι το 'focus' το οποίο καθορίζει αν το συγκεκριμένο Γραφικό Στοιχείο έχει στην αρχή την εστίαση πάνω του. Επειδή κάθε στιγμή μόνο ένα Γραφικό Στοιχείο μπορεί να έχει την εστίαση πάνω του, αν υπάρχουν πάνω από ένα Γραφικό Στοιχείο που έχουν ορίσει το χαρακτηριστικό αυτό σε 'true', τότε λαμβάνεται υπόψιν το τελευταίο που θα συναντήσει ο XAIL Διερμηνευτής.

5.3.3 Περιγραφή Γεγονότων

Σύνταξη

Τα γεγονότα (events) είναι σημαντικά για τη διεπαφή καθώς είναι το μέσο που επιτρέπει στο χρήστη να ειδοποιήσει το πρόγραμμα ότι κάτι πρέπει να γίνει τη δεδομένη στιγμή. Αυτό μπορεί να είναι είτε η εκτέλεση μιας διαδικασίας ή συνάρτησης, είτε η μορφοποίηση της ίδιας της διεπαφής ανάλογα με το τι έχει αποφασίσει ο προγραμματιστής να συμβεί.

Για τη σύνταξη ενός γεγονότος υπάρχει το στοιχείο <event> το οποίο παίρνει δύο χαρακτηριστικά, το 'type' και το 'object'. Το 'type' είναι το είδος του γεγονότος για το οποίο ο προγραμματιστής ενδιαφέρεται να ορίσει κάποια ενέργεια. Οι διαφορετικοί τύποι που υπάρχουν περιγράφονται στην επόμενη παράγραφο. Το 'object' παίρνει σαν τιμή μια XPATH έκφραση η οποία περιγράφει ένα σύνολο από κόμβους στοιχείων του XAIL αρχείου για τα οποία το συγκεκριμένο γεγονός πρέπει να καταχωρηθεί. Το παρακάτω παράδειγμα δείχνει πώς μπορούμε να ορίσουμε ένα γεγονός για όλα τα πλήκτρα εντολής της εφαρμογής, για τη περίπτωση που κάποιος κάνει κλικ με το ποντίκι πάνω σε κάποιο από αυτά.

Είναι δυνατόν το αρχείο να περιέχει πολλά στοιχεία <event> που να αφορούν το ίδιο στοιχείο χωρίς να υπάρχει πρόβλημα, ακόμα και αν αφορούν το ίδιο τύπο γεγονότος. Οι ενέργειες που ορίζονται για το ίδιο στοιχείο και για τον ίδιο τύπο γεγονότος σε διαφορετικά <event> στοιχεία λειτουργούν προσθετι-

κά, δηλαδή κάθε νέος ορισμός ενεργειών του ίδιου γεγονότος δεν αναιρεί τις προηγούμενες ορισμένες ενέργειες αλλά προστίθεται σε αυτές.

Για να ομαδοποιηθούν πολλά μαζεμένα γεγονότα, υπάρχει το στοιχείο <on> (σχήμα 5.10) το οποίο υπάρχει μόνο για διευκόλυνση και δεν είναι αναγκαίο για την ύπαρξη των γεγονότων.

Είδη γεγονότων

Το χαρακτηριστικό ΄type΄ στο στοιχείο <event> παίρνει το είδος του γεγονότος που ορίζεται. Τα είδη γεγονότων που έχουν υλοποιηθεί είναι τέσσερα και το χαρακτηριστικό ΄type΄ μπορεί να πάρει τις τιμές ’click¨, ’mouse over¨, ’focus’ και ’press¨.

Το ’click¨, όπως είδαμε σε προηγούμενο παράδειγμα, αναφέρεται στη περίπτωση που ο χρήστης κάνει κλικ με το ποντίκι πάνω στο Γραφικό Στοιχείο.

Το ’mouseOver’ αναφέρεται στη περίπτωση που ο χρήστης μεταφέρει τον δείκτη του ποντικιού πάνω από την επιφάνεια που καλύπτει το Γραφικό Στοιχείο.

Το ’focus’ αναφέρεται στην περίπτωση που η εστίαση περάσει με κάποιον τρόπο (κάνοντας κλικ ή πατώντας το tab πλήκτρο) στο συγκεκριμένο Γραφικό

```
<event type="click" object="//Button">
    ... handling actions to be performed ...
</event>
```

Σχήμα 5.9: Παράδειγμα: Ορισμός γεγονότος τύπου ΄click΄ για όλα τα πλήκτρα εντολής της εφαρμογής.

```
<on>
    <event type=... > ... actions ... </event>
    <event type=... > ... actions ... </event>
    <event type=... > ... actions ... </event>
</on>
```

Σχήμα 5.10: Παράδειγμα: Ομοαδοποίηση των γεγονότων με το στοχείο <on>.

Στοιχείο.

Το 'press' αναφέρεται στην περίπτωση που το συγκεκριμένο Γραφικό Στοιχείο είναι ήδη εστιασμένο και εκείνη τη στιγμή πατηθεί κάποιο πλήκτρο από το πληκτρολόγιο του χρήστη.

5.3.4 Ενέργειες

Οι ενέργειες που μπορούν να εκτελεστούν κατά την εμφάνιση ενός γεγονότος είναι η εισαγωγή στοιχείων, η διαγραφή, η τροποποίηση, καθώς και η εκτέλεση εντολών.

Εισαγωγή στοιχείου

Η εισαγωγή επιτρέπει τη δυναμική δημιουργία νέων συστατικών της διεπαφής. Συντάσσεται με το στοιχείο <insert> και παίρνει μια XPATH έκφραση στο χαρακτηριστικό ΄object΄ που αναφέρεται σε ποιους κόμβους του αρχείου θα γίνει η εισαγωγή των νέων στοιχείων. Τα στοιχεία που εισάγονται μπορεί να είναι ένα μοναδικό στοιχείο ή πολλά μαζί, και κάθε στοιχείο μπορεί να περιέχει και άλλα στοιχεία σε μορφή XML δέντρου, αν ακολουθεί την XAIL σύνταξη.

Αυτό που δηλώνει ο XAIL κώδικας του παραδείγματος στο σχήμα 5.11 είναι ότι αν γίνει κλικ πάνω στο στοιχείο button που έχει ορισμένο το χαρακτηριστικό ΄id΄ με τιμή ΄Button3΄, τότε μέσα στο πρώτο VBox στοιχείο που είναι κάτω από το Window θα εισαχθεί μια νέα ετικέτα με κείμενο 'Inserted label¨.

```
<event type="click" object="//Button[@id='Button3']">
    <insert object="//Window/VBox[1]">
        <Label>Inserted label</Label>
    </insert>
</event>
```

Σχήμα 5.11: Παράδειγμα: Εισαγωγή ετικέτας με το κλικάρισμα ενός πλήκτρου εντολής.

Διαγραφή στοιχείου

Η διαγραφή στοιχείων είναι σχετικά πιο απλή, συντάσσεται με το στοιχείο <remove> και παίρνει μια XPATH έκφραση στο χαρακτηριστικό ′object′ που αναφέρεται στα στοιχεία εκείνα που θέλουμε να διαγραφούν. Αν κάποιο στοιχείο περιέχει και άλλα στοιχεία, τότε διαγράφονται και αυτά. Ένα χαρακτηριστικό παράδειγμα φαίνεται στο σχήμα 5.12.

```
<event type="click" object="//Button[@id='Button3']">
    <remove object="//Window/VBox[2]" />
</event>
```

Σχήμα 5.12: Παράδειγμα: Διαγραφή του δεύτερου VBox του Window και όλων των υποστοιχείων του.

Τροποποίηση στοιχείου

Το στοιχείο <alter> χρησιμοποιείται για την τροποποίηση των στοιχείων του δέντρου, και λέγοντας τροποποίηση αναφερόμαστε στην αλλαγή τιμής του εσωτερικού κειμένου (text node) ή οποιουδήποτε χαρακτηριστικό (attribute) κάποιου στοιχείου. Αν το εσωτερικό κείμενο ή το χαρακτηριστικό που θέλουμε να αλλάξει δεν υπάρχει, τότε προστίθεται στο εκάστοτε στοιχείο.

Το χαρακτηριστικό ′object′ του <alter> αναφέρεται στα στοιχεία εκείνα που θέλουμε να τροποποιηθούν. Το χαρακτηριστικό ′attr′ παίρνει τη τιμή του ονόματος του χαρακτηριστικού που θα αλλαχθεί, ενώ αν θέλουμε να αλλάξουμε το εσωτερικό κείμενο και όχι κάποιο χαρακτηριστικό, τότε αρκεί να παραλείψουμε το ′attr′ μέσα στο <alter>.

Το εσωτερικό κείμενο του <alter> μπορεί να μπει είτε οποιοδήποτε αλφαριθμητικό είτε κάποια Υπολογίσιμη Έκφραση της XAIL (XCE - XAIL Computable Expression) το οποίο θα εξηγήσουμε πως συντάσσεται σε επόμενο κεφάλαιο, και όπως θα δούμε, δίνει στη γλώσσα κάποιες επιπλέον χρήσιμες δυνατότητες. Η τιμή που επιστρέφεται από μια Υπολογίσιμη Έκφραση της XAIL

χρησιμοποιείται ως η τιμή τροποποίησης. Στα σχήματα 5.13 και 5.14 φαίνονται δύο παραδείγματα τροποποίησης των στοιχείων του δέντρου της Διεπαφής Χρήστη.

```
<event type="click" object="//Button[@id='Button1']">
    <alter object="//Label[@id='stockLabel']">
        QuoterSystem::StockFactory.getStock("MSFT").getPrice()
    </alter>
</event>
```

Σχήμα 5.13: Παράδειγμα: Αλλαγή του κειμένου μιας ετικέτας κάνοντας χρήση XCE.

```
<event type="click" object="//Button[@id='Button1']">
    <alter object="//Label[@id='stockLabel']" attr='visible">
        "false"
    </alter>
</event>
```

Σχήμα 5.14: Παράδειγμα: Αλλαγή του χαρακτηριστικού 'visible' μιας ετικέτας χρησιμοποιώντας αλφαριθμητικό.

Εκτέλεση εντολών

Για να κάνουμε κλήση σε μεθόδους (συναρτήσεις) των αντικειμένων που υπάρχουν στα συστατικά λογισμικού (components) χρησιμοποιούμε τη στοιχείο <execute>, και μέσα στο στοιχείο αυτό ορίζουμε την κλήση με XCE, τα οποία θα τα δούμε σε επόμενο κεφάλαιο.

5.3.5 Χρήση Συστατικών Λογισμικού

Όλα τα συστατικά λογισμικού (components) που χρησιμοποιούνται από την εφαρμογή πρέπει να δηλώνονται στην αρχή του XAIL αρχείου. Αυτό γίνεται με το στοιχείο <using> το οποίο περιέχει το όνομα του συστατικού.

Το στοιχείο ‘using>, όπως και το στοιχείο <Window> του κεντρικού παραθύρου περιέχονται κάτω από το στοιχείο-ρίζα κάθε αρχείου XAIL και το οποίο ονομάζεται ‘xail’. Δηλαδή ένα αρχείο XAIL εφαρμογής έχει τη μορφή του παραδείγματος στο σχήμα 5.16.

5.3.6 Επεκτάσεις της Γλώσσας

Η γλώσσα XAIL, όπως προαναφέραμε, είναι σε πρώιμο στάδιο και έχει υλοποιηθεί σε βαθμό τέτοιο που να φαίνεται η βασική λειτουργικότητα της. Οι επεκτάσεις που μπορούν να γίνουν είναι πολλές, άλλες πιο σημαντικές και άλλες λιγότερο, και δεν περιορίζονται μόνο στην ποικιλία των γραφικών αντικειμένων και των υλοποιημένων γεγονότων που μπορούν να χρησιμοποιηθούν, αλλά περιλαμβάνουν και εννοιολογικές προεκτάσεις της γλώσσας. Για παράδειγμα θεωρούμε απαραίτητο στο μέλλον να προσθέσουμε κάποια δομή ελέγχου που θα δίνει τη δυνατότητα επιλεκτικής σχεδίασης της διεπαφής. Δηλαδή να δηλώνει κάτι σαν το εξής: αν ισχύει μία συνθήκη, πρόσθεσε ένα υπομενού στο τάδε

```
<event type="click" object="//Button[@id='startButton']">
    <execute>QuoterSystem::System.init()</execute>
</event>
```

Σχήμα 5.15: Παράδειγμα: Κλήση κώδικα αρχικοποίησης για το συστατικό ’QuoterSystem’.

```
<xail application="Test Application" version="1.0">
    <using>RandomString</using>
    <using>QuoterSystem</using>
    <Window title="Test Application" visible="true">
        ......
    </Window>
</xail>
```

Σχήμα 5.16: Παράδειγμα: Ενδεικτικό περιεχόμενο αρχείου που χρησιμοποιεί δύο συστατικά λογισμικού.

σημείο του δέντρου. Μια ενδεικτική μορφή σύνταξης αυτής της δομής φαίνεται στο σχήμα 5.17.

```
<test on="some XCE expression here...">
    <case op="greater" value="0">
        ...actions...
    </case>
    <case op="equal" value="0">
        ...actions...
    </case>
    <default>
        ...actions...
    </default>
</test>
```

Σχήμα 5.17: Παράδειγμα: Ενδεικτική μορφή σύνταξης δομής ελέγχου.

Με παρόμοια φιλοσοφία μπορεί να συντάξουμε και μια δομή επανάληψης που να δίνει τη δυνατότητα για τη δυναμική δημιουργία αντικειμένων με βάση κάποια συλλογή δεδομένων (πίνακας, σύνολο, λίστα κτλ) η οποία είναι προσπελάσιμη μέσω ενός συστατικού λογισμικού. Δηλαδή, η δομή επανάληψης παίρνει σαν παράμετρο ένα αντικείμενο που υποστηρίζει επαναλήπτες (iterators) και διατρέχει τις τιμές του χρησιμοποιώντας τες για τη δημιουργία γραφικών αντικειμένων. Στο παράδειγμα του σχήματος 5.18 δημιουργείται ένα μενού με τιμές δοσμένες από κάποια λίστα. Το '{current}' είναι μια συντακτική διευκόλυνση και δηλώνει το τρέχον αντικείμενο που βρίσκεται στη συλλογή. Στο τέλος θα δημιουργηθούν τόσα στοιχεία στο μενού όσα είναι και τα στοιχεία της λίστας.

5.4 XAIL Υπολογίσιμες Εκφράσεις

Όπως αναφέραμε και στις ενότητες 5.3.4, στο εσωτερικό κείμενο των στοιχείων <alter> και <execute> μπορούν να χρησιμοποιηθούν Υπολογίσιμες Εκφράσεις XAIL - XAIL Computable Expressions - ή αλλιώς XCE. Σε αυτή την ενότητα

θα εξηγήσουμε τι είναι αυτές και ποιος ο ρόλος τους στη γλώσσα XAIL.

Οι Υπολογίσιμες Εκφράσεις είναι εκφράσεις, πράξεις, που υπολογίζονται από το XAIL διερμηνευτή. Δεν αποτελούν μια καινούργια γλώσσα προγραμματισμού που όπως η JavaScript μπορεί να χρησιμοποιηθεί σε XML έγγραφα για να προσθέσει κάποια συμπεριφορά. Δεν παρέχονται τελεστές και εκφράσεις λογικής και επιλογής, όπως το γνωστό *if*, ή εντολές επανάληψης, δεν ορίζεται η έννοια της μεταβλητής και ούτε η έννοια της ανάθεσης τιμής. Το μόνο που επιτρέπει η γλώσσα των Υπολογίσιμων Εκφράσεων είναι αυτό που λέει το όνομά τους, δηλαδή, ο υπολογισμός τιμών.

Για το σκοπό η γλώσσα ορίζει τελεστές πράξεων: πρόσθεση (+), αφαίρεση (-), πολλαπλασιασμός (*), διαίρεση (/) και συνένωση συμβολοσειρών (+) όταν ένας από τους τελεστέους είναι συμβολοσειρά. Επιπλέον, ορίζονται τελεστές μοναδιαίας αύξησης (increment) και μείωσης (decrement).

Οι γλώσσα των Υπολογίσιμων Εκφράσεων έχει αυστηρό και στατικό σύστημα δεδομένων. Όμως οι τύποι των δεδομένων που συμμετέχουν στις εκφράσεις δεν δηλώνονται αλλά υπονοούνται και είναι στην ευθύνη του διερμηνευτή να κάνει τη σωστή απόφαση. Οι τύποι που αναγνωρίζονται στις Υπολογίσιμες Εκφράσεις είναι ο πραγματικός αριθμός διπλής ακρίβειας και η συμβολοσειρά, όπως ορίζονται στη γλώσσα IDL (CORBA::Double και CORBA::string αντίστοιχα). Οι Υπολογίσιμες Εκφράσεις έχουν πολύ στενή σχέση με τη γλώσσα IDL αφού το πιο δυνατό χαρακτηριστικό τους είναι η πραγματοποίηση κλήσεων σε IDL διεπαφές συστατικών λογισμικού που χρησιμοποιούνται από μια διεπαφή ορισμένη με τη γλώσσα XAIL.

```
<Menu label="Buy product">
    <iterate on="Supermarket::Products.listAll()">
        <Item>{current}.toString()</Item>
    </iterate>
</Menu>
```

Σχήμα 5.18: Παράδειγμα: Δημιουργία μενού με επαναληπτική δομή.

Στη συνέχεια θα δείξουμε τη σύνταξη των Υπολογίσιμων Εκφράσεων XAIL μέσα από παραδείγματα για κάθε περίπτωση. Για έναν αυστηρό EBNF ορισμό του συντακτικού των Υπολογίσιμων Εκφράσεων παραπέμπουμε τον αναγνώστη στο παράρτημα Β΄.

5.4.1 Κλήσεις Συστατικών Λογισμικού

Οι κλήσεις σε διεπαφές που εξάγει κάποιο συστατικό λογισμικού είναι ο λόγος ύπαρξης των Υπολογίσιμων Εκφράσεων XAIL. Μέσω αυτών η λογική παρουσίασης χρησιμοποιεί κατά το δοκούν την επιχειρησιακή λογική που υλοποιείται στα συστατικά λογισμικού. Μία κλήση πρέπει να ορίζει σε ποιο συστατικό λογισμικού πραγματοποιείται το οποίο πρώτα πρέπει να έχει δηλωθεί με το στοιχείο <using>, όπως έχει αναφερθεί στην ενότητα 5.3.5. Στη συνέχεια, προσδιορίζει τη διεπαφή στην οποία θα γίνει η κλήση, τη μέθοδο που θα εκτελεστεί με τις παραμέτρους της, αν είναι αναγκαίο.

```
module Product {
  interface Supplier {
    double getExpeneses();
    void addExpense(in double amount);
  };
  interface Shoe {
    string getName();
    double getPrice();
    double getCost();
    Supplier getSupplier();
  };
  interface VAT {
    double getPercentage(in Shoe product);
  };
};
```

Σχήμα 5.19: IDL διεπαφή ενός υποθετικού συστατικού λογισμικού.

Στο σχήμα 5.19 ορίζουμε τις διεπαφές ενός υποθετικού συστατικού λογι-

σμικού που χρησιμοποιούμε παραδειγματικά στο σχήμα 5.20 όπου φαίνονται οι κλήσεις προς αυτό. Έστω ότι αυτό το συστατικό λογισμικού ονομάζεται *Shop*. Αυτό που συμβαίνει στις πρώτες δύο εκφράσεις είναι αρκετά προφανές. Στη δεύτερη έκφραση, λαμβάνουμε πρώτα ένα αντικείμενο διεπαφής *Supplier* και καλούμε σε αυτό την *getExpenses()*. Στην τρίτη έκφραση, δίνουμε ως όρισμα στην *getPercentage()* το αντικείμενο της διεπαφής *Shoe*. Τέλος στην τέταρτη έκφραση συνδυάζονται οι δύο προηγούμενες περιπτώσεις χρήσης.

Σε αυτό το σημείο πρέπει να τονίσουμε πως όταν πραγματοποιούμε συνδυαστικές κλήσεις διεπαφών όπως στις δύο τελευταίες Υπολογίσιμες Εκφράσεις, δεν είναι αναγκαστικό να γίνονται σε διεπαφές του ίδιου συστατικού λογισμικού. Θα μπορούσε κάλλιστα η διεπαφή *Supplier* να βρίσκεται σε ένα άλλο συστατικό λογισμικού. Αυτό μας επιτρέπει να συνδυάζουμε τη λογική διαφορετικών συστατικών λογισμικού ώστε να διαμορφώσουμε την επιχειρησιακή λογική που απαιτείται σε μια εφαρμογή.

5.4.2 Αριθμητικοί Υπολογισμοί

Στο σχήμα 5.21 έχουμε παραδείγματα αριθμητικών υπολογισμών από Υπολογίσιμες Εκφράσεις XAIL. Όπως είναι αναμενόμενο η πρώτη έκφραση επιστρέφει 10, η δεύτερη 10, η τρίτη 25, και η τέταρτη 10. Στο συγκεκριμένο απόσπασμα XAIL του παραδείγματος η αριθμητική τιμή θα μετατραπεί σε συμβολοσειρά ώστε να εμφανιστεί ως τιμή της ετικέτας που τροποποιείται με το στοιχείο <alter>. Μάλιστα, επειδή σε κάθε <alter> τροποποιείται η ίδια ετικέτα, αυτή θα έχει στο τέλος την τιμή 10.

Στο σχήμα 5.21 οι τελεστέοι των υπολογισμών είναι σταθεροί αριθμοί και άρα το αποτέλεσμα των υπολογισμών είναι πάντα σταθερό. Σε μια πραγματική εφαρμογή τέτοιοι υπολογισμοί δε θα ήταν πολύ χρήσιμοι. Στο σχήμα 5.22 φαίνεται ένα απόσπασμα μιας XAIL εφαρμογής το οποίο εμφανίζει σε μια ετικέτα την τελική τιμή ενός προϊόντος (περιλαμβανομένης του Φ.Π.Α.). Στην υπολογίσιμη έκφραση χρησιμοποιούνται δύο κλήσεις προς συστατικά λογισμικού, τις οποίες εξηγήσαμε στην ενότητα 5.4.1, για να χρησιμοποιηθούν λειτουργίες της

```
<event type="click" object="//Button[@id='Actions']">
    <alter object="//Label[@id='ShoePrice']">
      Shop::Product.Shoe.getPrice()
    </alter>
    <alter object="//Label[@id='ShoeName']">
      Shop::Product.Shoe.getName()
    </alter>
    <alter object="//Label[@id='SupplierExpenses']">
      Shop::Product.Shoe.getSupplier().getExpenses()
    </alter>
    <alter object="//Label[@id='VAT']">
      Shop::Product.VAT.getPercentage(Shop::Product.Shoe)
    </alter>
    <execute>
      Shop::Product.Shoe.getSupplier().addExpense(
        Shop::Product.Shoe.getCost()
      )
    </execute>
</event>
```

Σχήμα 5.20: IDL διεπαφή ενός υποθετικού συστατικού λογισμικού.

```
<event type="click" object="//Button[@id='Calculations']">
    <alter object="//Label[@id='numeric']">5+5</alter>
    <alter object="//Label[@id='numeric']">(5+6)-1</alter>
    <alter object="//Label[@id='numeric']">5*5</alter>
    <alter object="//Label[@id='numeric']">(5*6)/3</alter>
</event>
```

Σχήμα 5.21: Παράδειγμα: Στοιχεία <alter> με αριθμητικούς υπολογισμούς.

επιχειρησιακής λογικής. Η κλήση *Shop::Product.Shoe.getPrice()* επιστρέφει την τιμή ενός παπουτσιού και η κλήση *getPercentage(Shop::Product.Shoe)* επιστρέφει το ποσοστό του Φ.Π.Α. που αντιστοιχεί στο συγκεκριμένο προϊόν.

5.4.3 Συνενώσεις Συμβολοσειρών

Στο σχήμα 5.23 έχουμε παραδείγματα Υπολογίσιμων Εκφράσεων XAIL που πραγματοποιούν συνενώσεις συμβολοσειρών. Αν κάποιος τελεστέος δεν είναι τύπου συμβολοσειράς, δηλαδή είναι αριθμός, τότε ο διερμηνευτής τον μετατρέπει στη συμβολική του αναπαράσταση και πραγματοποιεί τη συνένωση. Έτσι όπως είναι αναμενόμενο το αποτέλεσμα της πρώτης έκφρασης είναι ′5 5′, της δεύτερης είναι ′5 euros′ και της τρίτης ′some euros′.

```
<event type="click" object="//Button[@id='ShowFinalPrice']">
    <alter object="//Label[@id='FinalPrice']">
       Shop::Product.Shoe.getPrice() +
         (Shop::Product.VAT.getPercentage(Shop::Product.Shoe) *
          Shop::Product.Shoe.getPrice()) / 100
    </alter>
</event>
```

Σχήμα 5.22: Παράδειγμα: Υπολογισμός τιμής προϊόντος με αριθμητικούς υπολογισμούς και χρήση επιχειρησιακής λογικής.

```
<event type="click" object="//Button[@id='Button1']">
    <alter object="//Label[@id='stockLabel']">5+' 5'</alter>
    <alter object="//Label[@id='stockLabel']">
      5+' euros'
    </alter>
    <alter object="//Label[@id='stockLabel']">
      'some'+' euros'
    </alter>
</event>
```

Σχήμα 5.23: Παράδειγμα: Στοιχεία <alter> με συνενώσεις συμβολοσειρών.

Φυσικά, θα ήταν πιο φυσιολογικό οι συνενώσεις συμβολοσειρών να γίνονται στα πλαίσια της χρήσης της επιχειρησιακής λογικής. Ένα τέτοιο παράδειγμα δίνεται στο σχήμα 5.24, όπου με μια κλήση συστατικού λογισμικού παίρνουμε την τιμή ενός προϊόντος και τη συνενώνουμε με μια συμβολοσειρά που αναφέρει το νόμισμα.

5.5 XAIL Συστατικά Λογισμικού

Η λογική παρουσίασης μιας XAIL εφαρμογής ορίζεται με τη XAIL περιγραφή. Η επιχειρησιακή λογική οργανώνεται σε συστατικά στοιχεία λογισμικού που το καθένα ορίζει μία διεπαφή σε IDL. Μια XAIL εφαρμογή μπορεί να ορίζει δυναμικά ποια συστατικά θα χρησιμοποιήσει και με ποιο τρόπο. Επιπλέον, τα συστατικά μπορούν να είναι υλοποιημένα σε διαφορετικές γλώσσες προγραμματισμού (ανεξαρτησία περιβάλλοντος) και να εκτελούνται σε διαφορετικούς υπολογιστές στο ίδιο δίκτυο (ανεξαρτησία τοποθεσίας). Έχουμε ήδη επιλέξει τη γλώσσα IDL ως τρόπο περιγραφής των διεπαφών που εξάγουν τα συστατικά. Πρέπει να ορίσουμε όμως και ένα μοντέλο, μία αρχιτεκτονική, η οποία θα μπορεί μέσα σε μια XAIL εφαρμογή, ανάλογα με την XAIL περιγραφή της εφαρμογής, να συνδέει συστατικά στην εφαρμογή και να πραγματοποιεί τις κλήσεις προς αυτά.

Το μοντέλο αυτό θα ορίζεται στην IDL και στην αρχιτεκτονική CORBA (Common Object Request Broker Architecture). Η αρχιτεκτονική CORBA είναι ένα ανοιχτό πρότυπο για τη δημιουργία κατανεμημένων εφαρμογών σε ετε-

```
<event type="click" object="//Button[@id='ShowPrice']">
    <alter object="//Label[@id='Price']">
      Shop::Product.Shoe.getPrice()+' euros'
    </alter>
</event>
```

Σχήμα 5.24: Παράδειγμα: Συνένωση τιμή προϊόντος με συμβολοσειρά.

ρογενή περιβάλλοντα. Υπάρχουν και άλλα συστήματα παρόμοια με αυτό, αλλά επιλέξαμε το CORBA γιατί είναι ανοιχτό πρότυπο, τα χαρακτηριστικά του είναι κατάλληλα για τη δημιουργία ενός Μοντέλου Συστατικών (Component Model) για τη XAIL και τέλος γιατί υπάρχει πληθώρα υλοποιήσεων του προτύπου που είναι ανοικτό λογισμικό.

Προτού όμως αναλύσουμε το Μοντέλο Συστατικών της XAIL θα πρέπει πρώτα να κάνουμε μια εισαγωγή στην CORBA και την IDL, στα χαρακτηριστικά τους και πως αυτά βρίσκουν εφαρμογή στη XAIL.

Κεφάλαιο 6

CORBA & IDL

Στην ενότητα 5.5 είπαμε πως το Μοντέλο Συστατικών της XAIL στηρίζεται στην αρχιτεκτονική CORBA. Σε αυτό το κεφάλαιο θα περιγράψουμε αυτή την αρχιτεκτονική, καθώς και εξηγούμε έννοιες και στοιχεία στα οποία βασίζεται το Μοντέλο Συστατικών, για την καλύτερη κατανόηση του κεφαλαίου 7[1].

Το CORBA είναι μία αρχιτεκτονική ενδιάμεσου λογισμικού (middleware)[29, 30]. Σκοπός των αρχιτεκτονικών ενδιάμεσου λογισμικού είναι να συνδέουν διαφορετικά, ετερογενή συστήματα σε ενιαίες εφαρμογές. Το CORBA πρότυπο συντάσσεται από τον οργανισμό OMG (Object Management Group) που έχει ως σκοπό να δημιουργήσει μια πλήρη υποδομή για κατανεμημένες εφαρμογές, την OMA (Object Management Architecture). Τα πρότυπα CORBA και IDL αποτελούν πυρήνα της OMA.

Το πρότυπο CORBA ορίζει τη βασική υποδομή για την πραγματοποίηση αντικειμενοστραφών απομακρυσμένων (και μη) κλήσεων διαδικασιών (RPC - Remote Procedure Call). Αυτή η ενότητα θα παρουσιάσει μια εννοιολογική περίληψη του CORBA, καλύπτοντας τα παρακάτω θέματα:

[1] Αν και το CORBA είναι μια υπάρχουσα τεχνολογία η περιγραφή της δε τοποθετήθηκε μαζί με τα κεφάλαια 2 και 3 καθώς δεν αποτελεί μέρος της υπάρχουσας τεχνολογίας Γραφικών Διεπαφών. Είναι μία τεχνολογία στην οποία στηρίζεται η υλοποίηση του Μοντέλου Συστατικών XAIL, οπότε η περιγραφή της τοποθετήθηκε πριν από το κεφάλαιο 7 για να διευκολυνθεί η ροή του κειμένου.

1. Απομακρυσμένες Κλήσεις Διαδικασιών - Remote Procedure Calls
2. Απαιτήσεις αρχιτεκτονικής CORBA
3. Η γλώσσα διεπαφών IDL
4. Το πρωτόκολλο IIOP
5. Βασικές έννοιες CORBA

6.1 Απομακρυσμένες Κλήσεις Διαδικασιών

Τη δεκαετία του 80, αναπτύχθηκαν και διαδόθηκαν διάφοροι μηχανισμοί κλήσης απομκαρυσμένων διαδικασιών. Το δύο πιο δημοφιλή συστήματα Απομακρυσμένων Κλήσεων Διαδικασιών (Remote Procedure Calls - RPC) είναι το SUN RPC και το DCE (Distributed Computing Environment) RPC[29].

Για την εφαρμογή που την καλεί, μία κλήση απομακρυσμένης διαδικασίας μοιάζει με μία κλήση σε τοπική συνάρτηση αλλά, αντί να εκτελεστεί τοπικά, οι παράμετροι της διαδικασίας αποστέλλονται μέσω του δικτύου σε μία απομακρυσμένη εφαρμογή που θα την εκτελέσει. Οι επιστρεφόμενες τιμές της διαδικασίας, αν υπάρχουν, στέλνονται πίσω μέσω του δικτύου στην καλούσα εφαρμογή. Ο μηχανισμός RPC αποτελεί πυρήνα των κατανεμημένων συστημάτων. Η καλούσα εφαρμογή ονομάζεται πελάτης (client) και η καλούμενη εφαρμογή ονομάζεται εξυπηρετητής (server).

Το σχήμα 6.1 δείχνει τη μορφή μιας βασικής κλήσης σε απομακρυσμένη διαδικασία. Όταν ο πελάτης κάνει μία κλήση απομακρυσμένης διαδικασίας, οι παράμετροι της κλήσης μετασχηματίζονται σε ένα πακέτο αίτησης που αποστέλλεται μέσω του δικτύου στον εξυπηρετητή, χρησιμοποιώντας τυπικά πρωτόκολλο TCP ή UDP. Αυτή η διαδικασία μετασχηματισμού για μεταφορά στο δίκτυο ονομάζεται συστοίχιση (marshalling) και είναι ουσιαστικά η αντιγραφή των παραμέτρων σε μια προσωρινή μνήμη σε μορφή κατάλληλη για την εκπομπή στο δίκτυο[31]. Μία αίτηση είναι ένα μήνυμα RPC, μεταφερόμενο μέσα σε ένα TCP

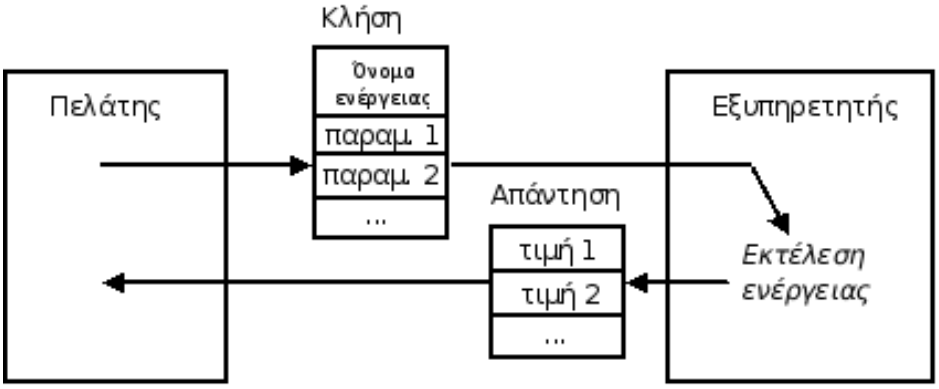
ή UDP πακέτο, το οποίο περιέχει το όνομα της απομακρυσμένης διαδικασίας και μία λίστα με τις παραμέτρους.

Όταν μία αίτηση φτάνει σε έναν εξυπηρετητή, οι παράμετροι υφίστανται την αντίστροφη διαδικασία της συστοίχισης και ο εξυπηρετητής εκτελεί τη διαδικασία. Μετά την εκτέλεση, γίνεται συστοίχιση στις επιστρεφόμενες τιμές της διαδικασίας σε ένα πακέτο απάντησης που αποστέλλεται πίσω στον πελάτη που αναμένει. Μία απάντηση είναι ένα RPC μήνυμα που περιέχει είτε τις επιστρεφόμενες τιμές ή μία κατάσταση λάθους.

Ένα ενδιαφέρον χαρακτηριστικό αυτού του παραδείγματος είναι πως η κλήση είναι σύγχρονη[30]. Δηλαδή, όταν ένας πελάτης κάνει μία κλήση, μπαίνει σε αναμονή (μπλοκάρει) μέχρι να ληφθεί η αντίστοιχη απάντηση από τον εξυπηρετητή. Οι σύγχρονες κλήσεις είναι η πιο συνηθισμένη μέθοδος απομακρυσμένων κλήσεων CORBA (αν και υπάρχει δυνατότητα να γίνουν και ασύγχρονες κλήσεις).

6.2 Απαιτήσεις Αρχιτεκτονικής CORBA

Επειδή κατανεμημένα συστήματα μπορούν να λειτουργούν πάνω σε διαφορετικές πλατφόρμες υλικού και διαφορετικά λειτουργικά συστήματα και γλώσσες προγραμματισμού, είναι επιθυμητό να αφαιρεθούν λεπτομέρειες πλατφόρμας και υλοποίησης[29, 30]. Ένας άλλος στόχος είναι να εξασφαλιστεί η απομακρυ-



Σχήμα 6.1: Μία απομακρυσμένη κλήση διαδικασίας (RPC).

σμένη κλήση να είναι σχεδόν το ίδιο εύκολη στη χρήση όπως οι εγγενείς κατασκευές των γλωσσών προγραμματισμού (συναρτήσεις, μέθοδοι). Επομένως, το CORBA σχεδιάστηκε έχοντας υπόψη τους παρακάτω στόχους:

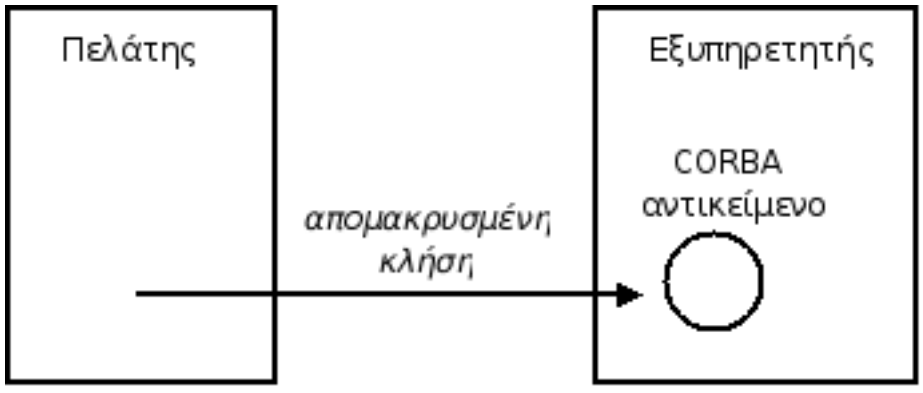
1. Αντικειμενοστραφής Σχεδίαση (Object Orientation)

2. Διαφάνεια Τοποθεσίας (Location Transparency)

3. Ανεξαρτησία Γλώσσας Προγραμματισμού (Programming Language Neutrality)

6.3 Αντικειμενοστραφής Σχεδίαση

Μία από τις κυριότερες βελτιώσεις που προσφέρει το CORBA σε σχέση με τις προηγούμενες RPC τεχνολογίες είναι η αντικειμενοστραφής σχεδίαση[29]. Οι απομακρυσμένες διαδικασίες ομαδοποιούνται σε διεπαφές (interfaces), όπως ακριβώς οι συναρτήσεις στη C++ και τη Java ομαδοποιούνται σε κλάσεις. Ένα στιγμιότυπο μιας διεπαφής ονομάζεται CORBA αντικείμενο (CORBA Object). Πρέπει να τονίσουμε εδώ πως το CORBA αντικείμενο είναι μία αφαιρετική έννοια και άρα δεν αντιστοιχείται πάντα άμεσα με κάποιο C++ ή Java αντικείμενο.



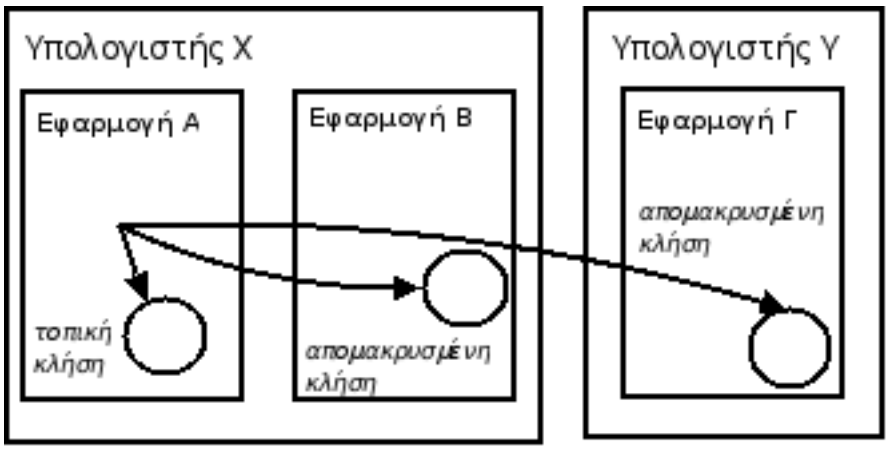
Σχήμα 6.2: Μία (απομακρυσμένη) κλήση σε ένα CORBA αντικείμενο.

Μία κλήση μιας ενέργειας πάνω σε ένα CORBA αντικείμενο φαίνεται στο σχήμα 6.2. Το CORBA αντικείμενο ζει στον εξυπηρετητή και η κλήση, ουσιαστικά μια κλήση απομακρυσμένης διαδικασίας, πραγματοποιείται από τον πελάτη. Για να πραγματοποιηθεί η απομακρυσμένη κλήση, ο πελάτης πρέπει να αποκτήσει την ταυτότητα του CORBA αντικειμένου, η οποία αναπαριστάται από μία αναφορά CORBA αντικειμένου (CORBA object reference)[31]. Μία αναφορά αντικειμένου ενθυλακώνει όλες τις πληροφορίες, συμπεριλαμβανομένης και της τοποθεσίας, που απαιτούνται για τη χρήση του CORBA αντικειμένου.

Σημειώνουμε πως στο CORBA μία συγκεκριμένη αναφορά αντικειμένου (object reference), αναφέρεται σε ένα μοναδικό CORBA αντικείμενο. Αν χρησιμοποιηθεί η ίδια αναφορά αντικειμένου σε διαφορετικές χρονικές στιγμές, οι κλήσεις οδηγούνται προς το ίδιο CORBA αντικείμενο.

6.4 Διαφάνεια Τοποθεσίας

Για να γίνει το CORBA ευκολότερο στη χρήση, υποστηρίζεται η διαφάνεια τοποθεσίας (Location Transparency) των CORBA αντικειμένων[29, 30]. Διαφάνεια τοποθεσίας σημαίνει πως δεν έχει σημασία που βρίσκεται το CORBA αντικείμενο. Πάντα η κλήση σε αυτό γίνεται χρησιμοποιώντας την ίδια σύνταξη.

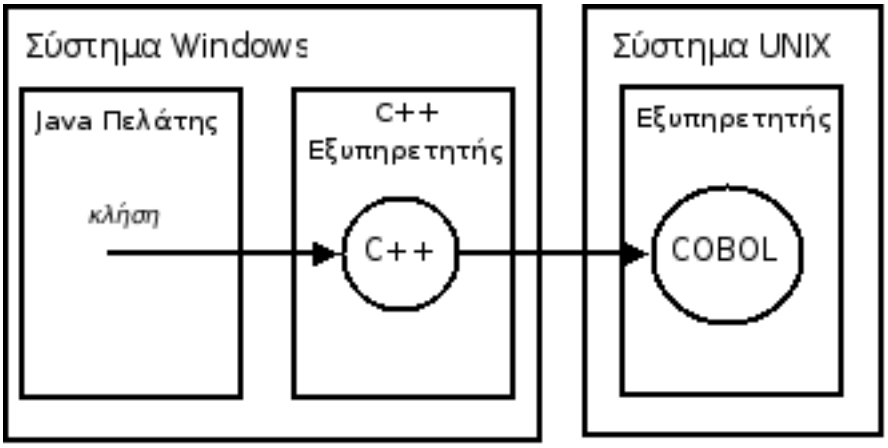


Σχήμα 6.3: Διαφάνεια τοποθεσίας CORBA αντικειμένων.

Αυτό φαίνεται στο σχήμα 6.3, το οποίο δείχνει τις εφαρμογές Α και Β που εκτελούνται στον υπολογιστή Χ και μία εφαρμογή Γ που εκτελείται στον υπολογιστή Υ. Έστω ότι ο κώδικας στην εφαρμογή Α καλεί μια ενέργεια σε ένα CORBA αντικείμενο. Δεν έχει σημασία αν το CORBA αντικείμενο βρίσκεται στο ίδιο χώρο διευθύνσεων (address space) με την εφαρμογή Α, ή σε διαφορετικό χώρο διευθύνσεων στον ίδιο υπολογιστή (εφαρμογή Β στο Χ), ή ακόμα και σε διαφορετική διεργασία σε διαφορετικό υπολογιστή (εφαρμογή Γ στον Υ). Σε κάθε περίπτωση η κλήση γίνεται χρησιμοποιώντας την ίδια σύνταξη.

6.5 Ανεξαρτησία Γλώσσας Προγραμματισμού

Το CORBA έχει σχεδιαστεί να δουλεύει με πολλαπλές γλώσσες προγραμματισμού. Και η σύνταξη μιας κλήσης (κώδικας πελάτη) και η υλοποίηση των CORBA αντικειμένων (κώδικας εξυπηρετητή) μπορεί να γίνει σε γλώσσα προγραμματισμού της επιλογής μας. Κάποιες γλώσσες προγραμματισμού για τις οποίες υπάρχουν αντιστοιχίσεις (mappings) είναι οι C, C++, Java, COBOL, Ada, Smalltalk και Lisp[29].



Σχήμα 6.4: Χρήση διαφορετικών γλωσσών προγραμματισμού.

Το σχήμα 6.4 δείχνει ένα Java πελάτη να καλεί ένα C++ CORBA αντικε-

ίμενο, το οποίο με τη σειρά του καλεί ενέργειες σε ένα CORBA αντικείμενο υλοποιημένο σε COBOL. Μπορεί να αναρωτιέστε αν έχει νόημα να μιλάμε για αντικείμενα υλοποιημένα σε μία γλώσσα όπως η COBOL, που δεν είναι καν αντικειμενοστραφής. Είναι δυνατό διότι οι πελάτες βλέπουν τον εξυπηρετητή μέσω μιας αντικειμενοστραφούς IDL διεπαφής. Ένα από τα δυνατά σημεία του CORBA είναι πως επιτρέπει να εισαγάγουμε αντικειμενοστραφείς έννοιες σε γλώσσες που δεν είναι εγγενώς αντικειμενοστραφείς.

6.6 Η Γλώσσα Διεπαφών IDL

Η διεπαφή σε ένα CORBA αντικείμενο ορίζεται χρησιμοποιώντας την Γλώσσα Ορισμού Διεπαφών IDL (Interface Description Language) της OMG[30]. Η OMG IDL είναι δικαιωματικά μία γλώσσα και δεν είναι παράγωγο κάποιας υπάρχουσας γλώσσας προργαμματισμού. Σε αντίθεση όμως με τις κανονικές γλώσσες προγραμματισμού, η OMG IDL είναι καθαρά μια δηλωτική γλώσσα - δεν υπάρχουν καθόλου συντακτικές κατασκευές για την αποτίμηση εκφράσεων ή την περιγραφή αλγορίθμων.

Το CORBA βασίζεται στην IDL και αυτό έχει πολλά πλεονεκτήματα. Η IDL έχει βελτιστοποιηθεί να είναι προσαρμόσιμη σε διαφορετικές γλώσσες προγραμματισμού, και η σύνταξη μπορεί να επεκταθεί όπως είναι απαραίτητο για να ανταποκριθεί στις ανάγκες των κατανεμημένων συστημάτων. Το μόνο μειονέκτημα είναι πως οι προγραμματιστές πρέπει να μάθουν μία καινούργια γλώσσα. Παρόλα αυτά, η OMG IDL είναι σχετικά απλή και προγραμματιστές εξοικιωμένοι με C++ ή Java θα καταλάβουν τα βασικά της IDL σε σύντομο χρονικό διάστημα.

Το σχήμα 6.5 δείχνει ένα παράδειγμα μίας IDL διεπαφής, το CustomerAccount, που μπορεί να χρησιμοποιηθεί για να αναπαραστήσει ένα λογαριασμό πελάτη σε μία τράπεζα.

Μία IDL διεπαφή είναι το ανάλογο μιας δήλωσης κλάσης για CORBA αντικείμενα. Η μόνη έκπληξη για C++ ή Java προγραμματιστές στο σχήμα 6.5 είναι

η εμφάνιση των in και out κατευθύνσεων. Μία in παράμετρος δίνεται από τον πελάτη στον εξυπηρετητή (pass by value), και μια out παράμετρος είναι μια παράμετρος που περνιέται από τον εξυπηρετητή πίσω στον πελάτη (μία παραπάνω επιστρεφόμενη τιμή).

6.7 Το Πρωτόκολλο IIOP

Η κύρια καινοτομία την περίοδο της έκδοσης του CORBA 2.0 πρότυπου ήταν η προδιαγραφή του Internet Inter-ORB Protocol (IIOP)[31]. Το IIOP ορίζει ένα πρότυπο πρωτόκολλο για την πραγμάτωση CORBA κλήσεων στο επίπεδο μεταφοράς TCP/IP. Στην πραγματικότητα, το IIOP είναι μία ειδίκευση του γενικού inter-ORB πρωτοκόλλου (GIOP - General Inter-ORB Protocol), το οποίο ορίζει το πρωτόκολλο ανεξάρτητα από το επίπεδο μεταφοράς.

Ένα από τα κέρδη του IIOP είναι ότι υποστηρίζει διαλειτουργικότητα μεταξύ ORB διαφορετικών κατασκευαστών. Αυτό σημαίνει πως ένας πελάτης γραμμένος σε ένα ORB ενός κατασκευαστή μπορεί να επικοινωνήσει με έναν εξυπηρετητή υλοποιημένο με τελείως διαφορετικό ORB. Πριν απο την εισαγωγή του IIOP, οι κατασκευαστές ORB υλοποιούσαν ιδιωτικά πρωτόκολλα για

```
//IDL
interface CustomerAccount {
    string getName();
    long getAccountNo();
    boolean depositMoney(in float amount);
    boolean transferMoney(
        in float amount,
        in long destinationAccountNo,
        out long confirmationNo
    );
};
```

Σχήμα 6.5: Παράδειγμα διεπαφής ορισμένης σε IDL.

τη μεταφορά CORBA κλήσεων και έτσι η διαλειτουργικότητα δεν ήταν δυνατή.

6.8 Βασικές Έννοιες CORBA

Αυτή η ενότητα περιγράφει τα βασικά στοιχεία του πυρήνα του CORBA πρότυπου. Θα περιγράψουμε τις ακόλουθες έννοιες:

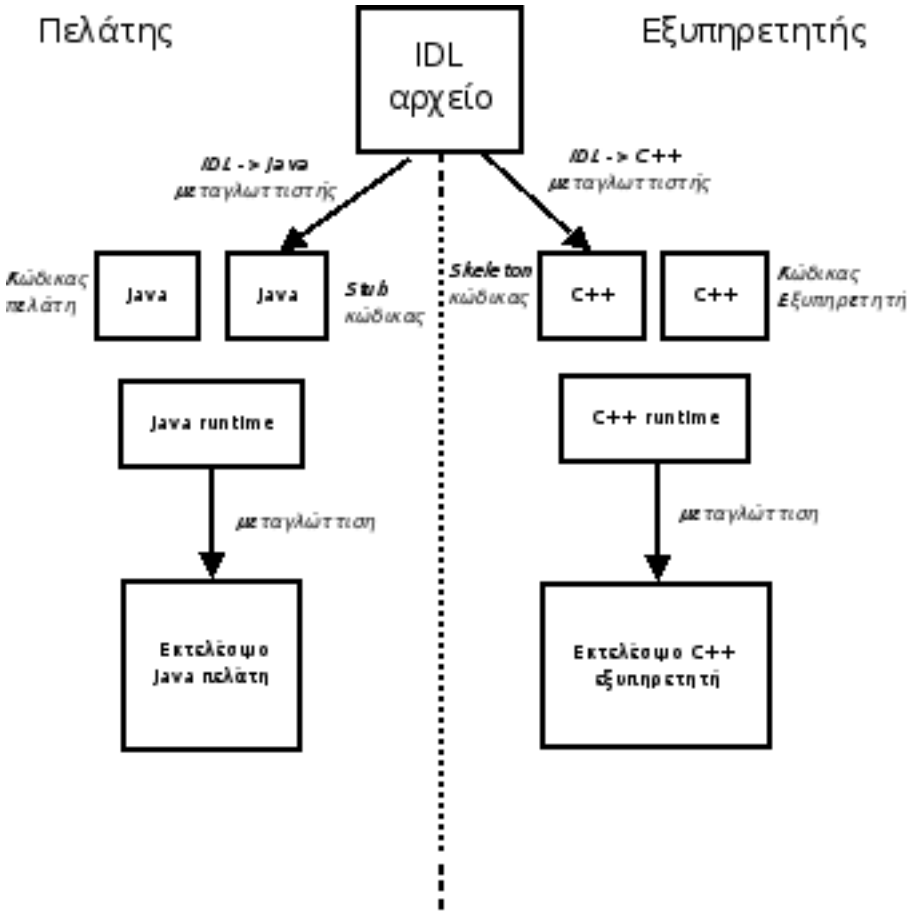
1. IDL μεταγλωττιστές.
2. Αντιστοιχίσεις (mappings) γλωσσών.
3. Κώδικας αποκόμματος και σκελετού.
4. Αναφορές αντικειμένων.
5. Προσαρμογέας Αντικειμένων.

6.8.1 IDL Μεταγλωττιστές

Ο ορισμός IDL για το σύστημα είναι το πρώτο βήμα στην ανάπτυξη μιας CORBA εφαρμογής. Επιτρέπει να ορίσουμε διεπαφές στα CORBA αντικείμενα με έναν τρόπο ανεξάρτητο πλατφόρμας, ανεξάρτητο γλώσσας προγραμματισμού και ανεξάρτητο από λεπτομέρειες υλοποίησης. Μετά τη συγγραφή της IDL της εφαρμογής, μπορεί να αποφασιστεί ποιες πλατφόρμες και ποιες γλώσσες προγραμματισμού θα χρησιμοποιηθούν για τον πελάτη και τον εξυπηρετητή.

Ένας IDL μεταγλωττιστής χρησιμοποιείται για να αντιστοιχίσει την IDL σε μία συγκεκριμένη γλώσσα προγραμματισμού[29]. Σκεφτείτε για παράδειγμα, ένα σύστημα που ο πελάτης είναι υλοποιημένος σε Java και ο εξυπηρετητής σε C++. Το σχήμα 6.6 δείχνει ένα περίγραμμα των βημάτων που ακολουθούνται.

1. Στην πλευρά του πελάτη, το IDL αρχείο δίνεται στον IDL μεταγλωττιστή για Java. Ο IDL μεταγλωττιστής αντιστοιχεί τους IDL ορισμούς σε



Σχήμα 6.6: Διαδικασία μεταγλώττισης ενός Java πελάτη και ενός C++ εξυπηρετητή.

Java, παράγοντας Java κώδικα αποκόμματος ως έξοδο. Ο κώδικας αποκόμματος παρέχει στον πελάτη τον κώδικα που χρειάζεται για να κάνει Java κλήσεις στις διεπαφές που ορίζονται στο IDL αρχείο.

2. Στην πλευρά του εξυπηρετητή, το IDL αρχείο δίνεται στο μεταγλωττιστή για C++. Ο μεταγλωττιστής παράγει C++ κώδικα σκελετού ως έξοδο. Ο κώδικας σκελετού παρέχει στον εξυπηρετητή τον κώδικα που απαιτείται για να ορίσει τις υλοποιήσεις των CORBA αντικειμένων.

Φυσικά η επιλογή των γλωσσών προγραμματισμού θα μπορούσε να αντιστραφεί ώστε ο πελάτης να είναι υλοποιημένος σε C++ και ο εξυπηρετητής σε Java.

Ένας IDL μεταγλωττιστής είναι τυπικά ένα εργαλείο της γραμμής εντολών που διαβάζει ένα IDL αρχείο και παράγει αρχεία στη γλώσσα στόχου, είτε κώδικα αποκόμματος είτε κώδικα σκελετού. Οι IDL μεταγλωττιστές είναι συνήθως ειδικευμένοι για μία μόνο γλώσσα. Θα χρησιμοποιούσαμε ένα εργαλείο για την αντιστοίχηση IDL σε Java και ένα άλλο εργαλείο για IDL σε C++.

6.8.2 Αντιστοιχίσεις Γλωσσών

Το CORBA συμπληρώνει το πυρήνα του προτύπου με μία σειρά από προδιαγραφές αντιστοιχίσεων γλωσσών (language mappings) που ορίζουν πως να μεταφραστούν IDL ορισμοί σε ισοδύναμες κατασκευές στην γλώσσα στόχου[30]. Για παράδειγμα αν η γλώσσα στόχου είναι η C++ ή η Java, η αντιστοίχιση της γλώσσας:

1. Ορίζει πώς οι IDL τύποι δεδομένων αντιστοιχούν σε ισοδύναμους C++ ή Java τύπους δεδομένων.

2. Ορίζει πώς οι IDL διεπαφές αντιστοιχούν σε κλάσεις και πως οι IDL ενέργειες αντιστοιχούν σε συναρτήσεις-μέλη (C++) ή μεθόδους (Java).

3. Περιγράφει με ποιο τρόπο μπορούν να υλοποιηθούν τα CORBA αντικείμενα στην πλευρά του εξυπηρετητή.

6.8.3 Κώδικας Αποκόμματος και Σκελετού

Οι κώδικες αποκόμματος[2] (stub) και σκελετού (skeleton) παράγονται από τους IDL μεταγλωττιστές και χρησιμοποιούνται για να κάνουν τους πελάτες και τους εξυπηρετητές ενήμερους για τους ορισμούς που εμφανίζονται στο IDL αρχείο[29]. Ο κώδικας αποκόμματος χρησιμοποιείται στην πλευρά του πελάτη και επιτρέπει στους πελάτες να καλούν ενέργειες σε απομακρυσμένα CORBA αντικείμενα χρησιμοποιώντας την ίδια σύνταξη όπως αν ήταν τοπικά αντικείμενα. Για παράδειγμα, ο Java stub κώδικας στο σχήμα 6.6 περιέχει τους ακόλουθους Java ορισμούς:

- Κάθε IDL τύπος δεδομένων αναπαρίσταται από έναν αντίστοιχο Java τύπο δεδομένων.
- Κάθε IDL διεπαφή αναπαρίσταται από μία αντίστοιχη Java διεπαφή.
- Κάθε IDL ενέργεια αναπαρίσταται από μία αντίστοιχη Java μέθοδο.

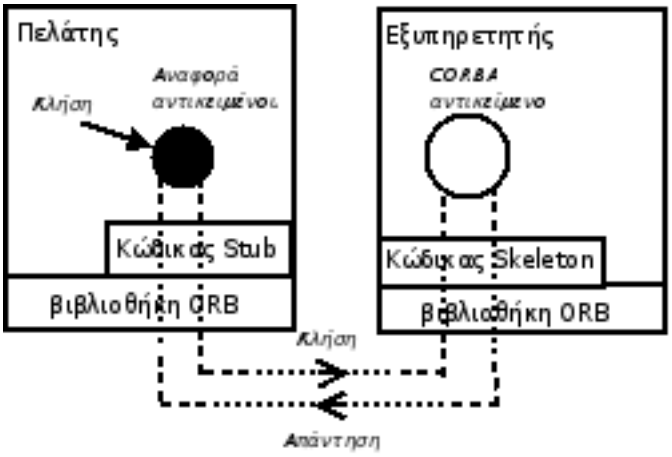
Ο κώδικας σκελετού (ένα υπερσύνολο του κώδικα αποκόμματος) χρησιμοποιείται στην πλευρά του εξυπηρετητή και επιτρέπει στους εξυπηρετητές να υλοποιήσουν CORBA αντικείμενα. Για παράδειγμα, ο C++ κώδικας αποκόμματος στο σχήμα 6.6 χρησιμοποιείται για να συσχετίσει τις IDL διεπαφές με C++ κλάσεις υλοποιημένες από τον προγραμματιστή. Όταν ο πελάτης καλεί μια συγκεκριμένη IDL ενέργεια σε μία διεπαφή, καλείται η αντίστοιχη C++ συνάρτηση-μέλος μιας C++ κλάσης.

6.8.4 Αναφορές Αντικειμένων

Το σχήμα 6.7 δείχνει πώς ο κώδικας της εφαρμογής, ο κώδικας αποκόμματος, και η βιβλιοθήκη του ORB συνδυάζονται μαζί στην πλευρά του πελάτη και πώς ο κώδικας εφαρμογής, ο κώδικας σκελετού, και η ORB βιβλιοθήκη συνδυάζονται

[2]Η μετάφραση προέρχεται από την έννοια του αγγλικού όρου stub ως απόκομμα χαρτιού (απόδειξης, επιταγής, κλπ) που χρησιμοποιείται αντί του πρωτότυπου.

μαζί στην πλευρά του εξυπηρετητή. Στον πελάτη, το CORBA API, αποτελείται από το API της βιβλιοθήκης εκτέλεσης (runtime library API), που δίνει πρόσβαση στο ORB αντικείμενο και άλλα υλοποιημένα αντικείμενα, και το API του κώδικα αποκόμματος, που δίνει πρόσβαση στις ορισμένες από το χρήστη IDL διεπαφές.



Σχήμα 6.7: Χρήση αναφοράς αντικειμένου για πραγματοποίηση απομακρυσμένης κλήσης.

Ένας πελάτης πρέπει να έχει μια αναφορά αντικειμένου (object reference) - για να κάνει κλήσεις σε ένα CORBA αντικείμενο γιατί αναφορά αντικειμένου ενθυλακώνει τις λεπτομέρειες τοποθεσίας του CORBA αντικειμένου[31]. Στη C++ και τη Java, η αναφορά αντικειμένου είναι και αυτή ένα αντικείμενο με συναρτήσεις-μέλη που έχουν προέλθει από την αντιστοίχηση στις ενέργειες της αντίστοιχης IDL διεπαφής. Όταν ένας πελάτης κάνει μια κλήση, καλεί την αντίστοιχη συνάρτηση-μέλος στην αναφορά αντικειμένου. Απ΄ όσο γνωρίζει ο πελάτης η αναφορά αντικειμένου θα μπορούσε κάλλιστα να είναι και το CORBA αντικείμενο. Η αναφορά αντικειμένου λειτουργεί ως αντικαταστάτης, ένα αντικείμενο αντιπρόσωπος (proxy object), για το CORBA αντικείμενο στην πλευρά του πελάτη[29].

Όμως, η αναφορά αντικειμένου δεν υλοποιεί στην πραγματικότητα την καλο-

ύμενη ενέργεια. Το 6.7 δείχνει τι συμβαίνει αφού κληθεί η συνάρτηση-μέλος της αναφοράς αντικειμένου: Μία κλήση απομακρυσμένης διαδικασίας δημιουργείται, με μία αίτηση, που περιέχει τις παραμέτρους της διαδικασίας, να αποστέλλεται στον απομακρυσμένο εξυπηρετητή. Στον εξυπηρετητή, η κλήση δρομολογείται στο κατάλληλο CORBA αντικείμενο με τη βοήθεια της ORB βιβλιοθήκης και του κώδικα σκελετού. Ο εξυπηρετητής εκτελεί την ενέργεια και στέλνει πίσω μια απάντηση που περιέχει την επιστρεφόμενη τιμή και τις out παραμέτρους. Πίσω στον πελάτη, η αναφορά αντικειμένου επιστρέφει την επιστρεφόμενη τιμή και τις out παραμέτρους στον κώδικα της εφαρμογής.

6.8.5 Προσαρμογέας Αντικειμένων

Οι αντιστοιχίσεις (mappings) στις C++ και Java εκμεταλλεύονται το γεγονός ότι η C++ και η Java είναι αντικειμενοστραφείς επιτρέποντας την υλοποίηση κλάσεων για CORBA αντικείμενα με τον ίδιο τρόπο όπως θα γινόταν για κανονικά Java ή C++ αντικείμενα.

Ύστερα από την υλοποίηση μιας CORBA κλάσης, πρέπει να ενημερώσουμε τον ORB πως αυτή αναπαριστά μία συγκεκριμένη IDL διεπαφή. Χρειάζεται επίσης ένας τρόπος να πούμε στον ORB πως να δημιουργεί στιγμιότυπα αυτής της κλάσης, τα CORBA αντικείμενα, προσβάσιμα σε εφαρμογές πελάτη[29]. Το κομμάτι του ORB που είναι υπεύθυνο για αυτά είναι ο προσαρμογέας αντικειμένων (object adapter)[30].

Οι ουσιαστικές ευθύνες ενός προσαρμογέα αντικειμένων είναι:

1. Να παρέχει ένα μηχανισμό συσχέτισης μιας Java ή C++ κλάσης με μία συγκεκριμένη IDL διεπαφή.

2. Να διαχειρίζεται τον κύκλο ζωής των CORBA αντικειμένων. Συγκεκριμένα, ο προσαρμογέας αντικειμένων πρέπει να παρέχει ένα τρόπο ενεργοποίησης των CORBA αντικειμένων (κάνοντάς τα προσβάσιμα σε πελάτες) και απενεργοποίησης των CORBA αντικειμένων (κάνοντάς τα μη προσβάσιμα σε πελάτες).

Δύο διαφορετικά είδη προσαρμογέων αντικειμένων χρησιμοποιούνται: ο Basic Object Adapter ή BOA και ο Portable Object Adapter ή POA[29]. Ο BOA προέρχεται από παλαιότερες εκδόσεις του CORBA προτύπου, ήταν χαλαρά ορισμένος και οι κατασκευαστές ORB οδηγήθηκαν στην υλοποίηση επεκτάσεων μη συμβατών μεταξύ τους. Άρα κώδικας γραμμένος για ένα BOA δεν είναι μεταφέρσιμος σε μια άλλη ORB υλοποίηση. Ο POA προστέθηκε στο πρότυπο στην έκδοση 2.2 με σκοπό να αντικαταστήσει τον BOA. Οι προδιαγραφές του είναι πολύ λεπτομερείς, οδηγώντας έτσι σε καλό επίπεδο μεταφερσιμότητας μεταξύ υλοποιήσεων και προσθέτοντας πολλές βελτιώσεις.

Κεφάλαιο 7

Μοντέλο Συστατικών της XAIL

Το μοντέλο συστατικών της XAIL - XAIL Component Model - στηρίζεται στο CORBA και την IDL, τα οποία περιγράψαμε στο προηγούμενο κεφάλαιο. Ένα XAIL συστατικό ορίζεται μέσα από ένα IDL αρχείο στο οποίο δηλώνονται οι διεπαφές που εξάγει. Μία XAIL εφαρμογή μπορεί δυναμικά, απλά αλλάζοντας το XAIL αρχείο, να ορίσει ποια συστατικά θα χρησιμοποιήσει και ποιες κλήσεις θα κάνει προς αυτά. Αυτό σημαίνει πως:

- Θα πρέπει να χρησιμοποιήσουμε το DII (Dynamic Invocation Interface - Διεπαφή Δυναμικής Κλήσης) που προσφέρει ένας ORB για να πραγματοποιηθούν οι κλήσεις, αφού δεν θα υπάρχει γνώση την ώρα της μεταγλώττισης (compile-time) για τις διεπαφές που θα χρησιμοποιηθούν. Δηλαδή, στην πλευρά του πελάτη, που είναι ο XAIL Διερμηνευτής, δε θα χρησιμοποιηθεί stub code των διεπαφών κάποιου συστατικού.

- Θα πρέπει να χρησιμοποιηθούν μηχανισμοί που θα ενεργοποιούν τα συστατικά που θέλει να χρησιμοποιήσει μια XAIL εφαρμογή και να τα ¨φορτώνουν¨.

- Σε κάθε συστατικό θα πρέπει να αντιστοιχίζεται ένα μοναδικό συμβολικό

όνομα.

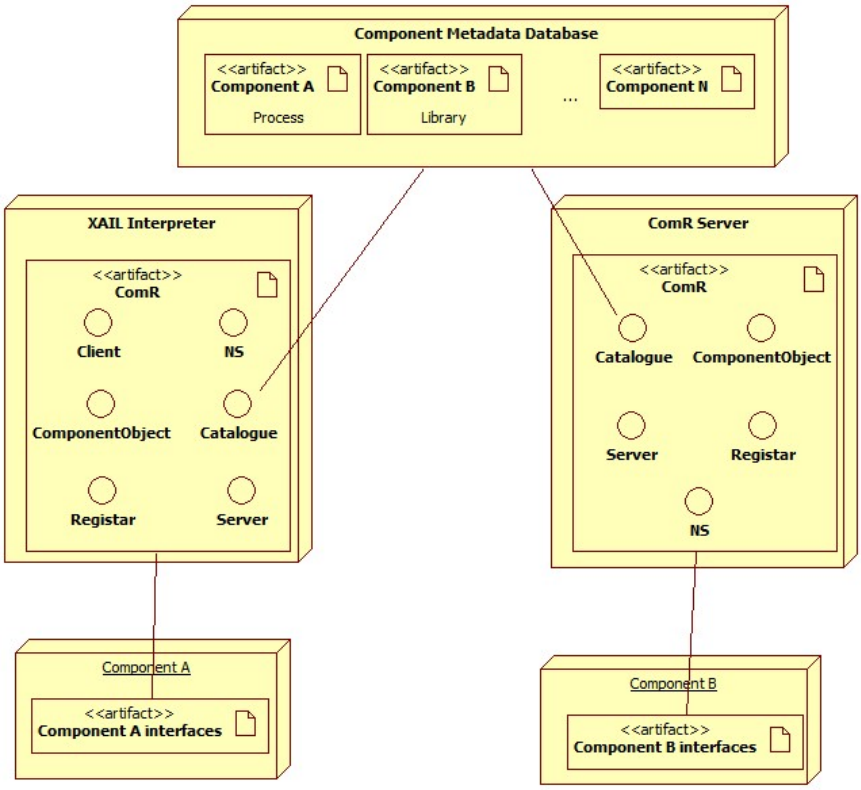
- Σε κάθε αναφορά αντικειμένου ενός αντικειμένου θα πρέπει να αντιστοιχίζεται ως συμβολικό όνομα το όνομα της αντίστοιχης διεπαφής. Όταν σε ένα XAIL αρχείο δηλώνεται μια κλήση θα γίνεται με το συμβολικό όνομα του συστατικού, το όνομα της διεπαφής και το όνομα της ενέργειας, π.χ. Quoter.Stock.getPrice().

Όπως έχουμε αναφέρει προηγουμένως, για το CORBA ορίζονται και μια σειρά από προδιαγραφές χρήσιμων υπηρεσιών, συμπληρωματικών στις υπόλοιπες CORBA λειτουργίες. Κάποιες από αυτές τις υπηρεσίες προσφέρουν τέτοια λειτουργικότητα όπως το CORBA Naming Service για την αντιστοίχηση συμβολικών ονομάτων σε αναφορές αντικειμένων και το Implementation Repository που ενεργοποιεί CORBA αντικείμενα αυτόματα. Επειδή όμως οι λειτουργίες αυτών των υπηρεσιών έχουν διαφορές από τις επιθυμητές, το XAIL Μοντέλο Συστατικών στηρίζεται σε μία διαφορετική CORBA υπηρεσία που δημιουργήσαμε, σχεδιασμένη ειδικά για το αυτό. Η υπηρεσία αυτή λέγεται Αποθήκη Συστατικών - ή ComR από το αγγλικό Component Repository - και θα είναι το αντικείμενο με το οποία θα ασχοληθούμε στις επόμενες ενότητες. Η μόνη CORBA υπηρεσία που χρησιμοποιείται στο μοντέλο συστατικών είναι το Interface Repository. Αυτή αποθηκεύει τις IDL δηλώσεις όλων των συστατικών και χρησιμοποιείται από τον XAIL Διερμηνευτή κατά τη δυναμική κατασκευή κλήσεων προς διεπαφές των συστατικών με το μηχανισμό DII.

7.1 Υπηρεσία Αποθήκης Συστατικών

Η ComR υπηρεσία είναι ο πυρήνας του XAIL Μοντέλου Συστατικών καθώς αυτή υλοποιεί όλες τις ενέργειες διαχείρισης των συστατικών. Είναι μία κατανεμημένη υπηρεσία η οποία χωρίζεται σε τρία τμήματα, όπως φαίνεται και στο σχήμα 7.1, σε αυτό που βρίσκεται στον XAIL Διερμηνευτή, σε αυτό που βρίσκεται σε κάποιο συστατικό και σε αυτό που βρίσκεται σε ένα ComR Εξυπηρετητή.

Τα τρία αυτά τμήματα υλοποιούνται σε τρεις διαφορετικές βιβλιοθήκες που το κάθε μέρος συνδέεται στο χώρο διευθύνσεων του. Το τμήμα του XAIL Διερμηνευτή θεωρείται πως είναι ο πελάτης, ενώ εξυπηρετητής θεωρούμε πως είναι κάποιο συστατικό. Τα τμήματα στο XAIL Διερμηνευτή και στον ComR Εξυπηρετητή όπως φαίνεται και στο σχήμα 7.1 χρησιμοποιούν και μία βάση δεδομένων (είτε πραγματικό σύστημα είτε απλά ένα σύνολο XML αρχείων) όπου αποθηκεύονται μεταδεδομένα για το κάθε συστατικό. Σε αυτά όμως θα αναφερθούμε αργότερα.



Σχήμα 7.1: Αρχιτεκτονική της υπηρεσίας ComR. Διακρίνονται τα τρία κύρια στοιχεία της αρχιτεκτονικής, ο XAIL Διερμηνευτής, ο ComR Εξυπηρετητής και η βάση μεταδεδομένων των συστατικών. Διακρίνονται ποιες διεπαφές υλοποιούνται σε ποιο στοιχείο.

Η υπηρεσία ComR ορίζει τις λειτουργίες που προσφέρει σε ένα IDL αρχείο όπου δηλώνονται οι διεπαφές που προσφέρει. Όπως φαίνεται στο σχήμα 7.1 οι διεπαφές είναι κατανεμημένες στα διάφορα τμήματα της υπηρεσίας. Ο XAIL Διερμηνευτής και ο ComR Εξυπηρετητής χρησιμοποιούν και οι δύο κοινές διεπαφές, το Catalogue, το ComponentObject και το NS, με διαφορετικές όμως υλοποιήσεις.

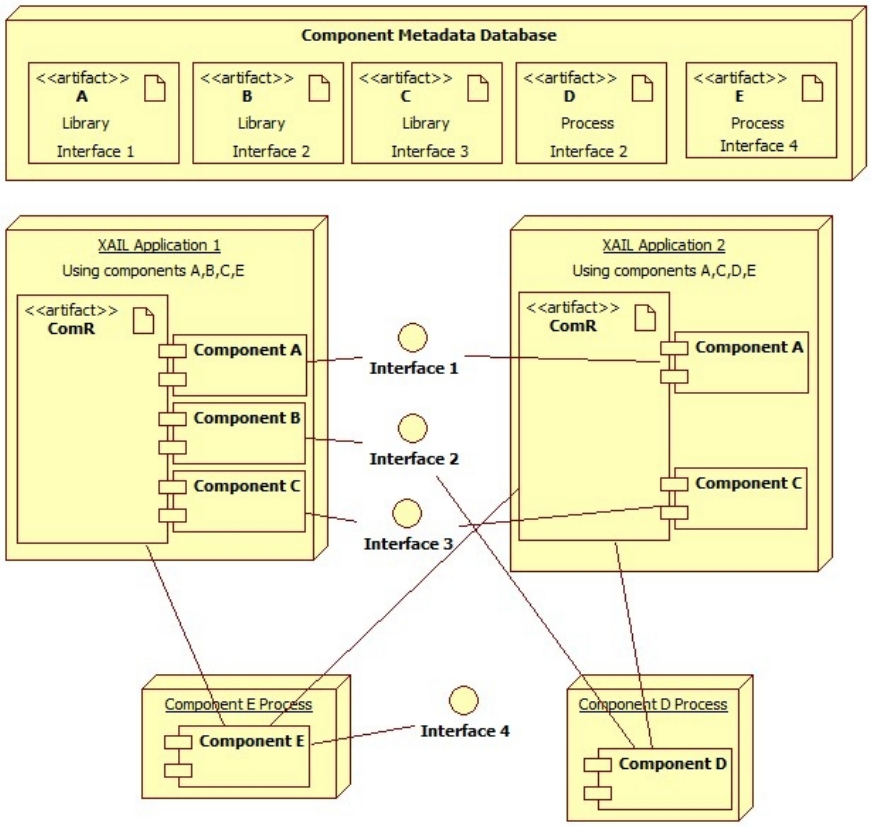
7.2 Είδη Συστατικών

Στο CORBA, οι λεπτομέρειες τοποθεσίας, υλοποίησης, πλατφόρμας κλπ αποκρύπτονται από το χρήστη ενός CORBA αντικειμένου. Κατέπέκταση και στο Μοντέλο Συστατικών της XAIL, το οποίο στηρίζεται στο CORBA, αποκρύπτονται αυτές οι λεπτομέρειες από το σχεδιαστή για τα συστατικά που χρησιμοποιεί.

Εσωτερικά όμως, το Μοντέλο Συστατικών διακρίνει τα συστατικά σε τρία είδη και αυτό ανάλογα με τον τρόπο που αυτά ενεργοποιούνται:

- Σε συστατικά που βρίσκονται υλοποιημένα μέσα σε δυναμικές βιβλιοθήκες (.so, .dll).
- Σε συστατικά που βρίσκονται υλοποιημένα μέσα σε ξεχωριστά προγράμματα.
- Σε συστατικά που είναι τοποθετημένα σε απομακρυσμένα συστήματα.

Κάθε ένα από αυτά τα είδη συστατικών ενεργοποιείται και συνδέεται με μια XAIL εφαρμογή με διαφορετικό τρόπο. Όπως φαίνεται και στο σχήμα 7.2, ένα συστατικό σε μια δυναμική βιβλιοθήκη ενεργοποιείται για και συνδέεται με κάθε XAIL εφαρμογή ξεχωριστά. Κάθε διεργασία του XAIL Διερμηνευτή έχει ένα δικό της αντίγραφο του συστατικού στο χώρο διευθύνσεων της. Ένα συστατικό σε ένα ξεχωριστό πρόγραμμα ενεργοποιείται μία φορά, για όλες τις XAIL εφαρμογές που το χρησιμοποιούν. Δηλαδή εκτελείται μόνο ένα αντίγραφο του



Σχήμα 7.2: Συστατικά στη XAIL. Το κάθε συστατικό υλοποιεί μία διεπαφή (όχι κατάνάγκη διαφορετικό). Μπορεί να είναι υλοποιημένο ως βιβλιοθήκη - οπότε έχουμε ένα στιγμιότυπο για κάθε XAIL εφαρμογή που το χρησιμοποιεί - ή ως ξεχωριστή διεργασία - οπότε έχουμε ένα στιγμιότυπο για όλες τις XAIL εφαρμογές.

συστατικού και όλες οι XAIL εφαρμογές που το χρησιμοποιούν συνδέονται με αυτό.

Για ένα συστατικό σε δυναμική βιβλιοθήκη το ComR φορτώνει τη βιβλιοθήκη στο χώρο διευθύνσεων της XAIL εφαρμογής που το χρησιμοποιεί, ενώ για ένα συστατικό σε ξεχωριστό πρόγραμμα το ComR το συνδέει με τη XAIL εφαρμογή (αν το πρόγραμμα δεν εκτελείται το ενεργοποιεί).

Ένα συστατικό που βρίσκεται σε απομακρυσμένο υπολογιστή, μπορεί να είναι και αυτό με τη σειρά του είτε βιβλιοθήκη είτε πρόγραμμα το οποίο όμως το διαχειρίζεται μία ξεχωριστή ComR υπηρεσία που εκτελείται στον απομακρυσμένο υπολογιστή. Η τοπική ComR υπηρεσία που θέλει να ενεργοποιήσει/συνδέσει το συστατικό με μία τοπική XAIL εφαρμογή επικοινωνεί με την απομακρυσμένη ComR υπηρεσία.

7.3 Διεπαφή Πελάτη

Στο σχήμα 7.3 φαίνεται η διεπαφή Πελάτη (Client), ορισμένη σε IDL. Όπως φαίνεται και στο σχήμα 7.1 το Client CORBA αντικείμενο βρίσκεται στο XAIL Διερμηνευτή. Το Client χρησιμοποιείται από τον XAIL Διερμηνευτή για να ζητά δυναμικά από το ComR τη σύνδεση της εφαρμογής με κάποιο συστατικό. Μόλις γίνει και ολοκληρωθεί επιτυχώς η αίτηση, η XAIL εφαρμογή μπορεί να έχει πρόσβαση στα CORBA αντικείμενα του συστατικού μέσω του CORBA αντικειμένου NS.

Το Client έχει ορίζει δύο ενέργειες: request() και free().

7.3.1 request()

Η request() δέχεται μία παράμετρο εισόδου, το όνομα του συστατικού και δεν έχει έξοδο. Αναλαμβάνει την ενεργοποίηση ενός συστατικού για την τρέχουσα XAIL εφαρμογή. Για να το κάνει αυτό χρησιμοποιεί το Catalogue CORBA αντικείμενο που βρίσκεται και αυτό στον XAIL Διερμηνευτή. Αυτό, μέσω της ενέργειας getComponent(), επιστρέφει το ComponentObject του συστατικού

```
//This interface is used by the XAIL runtime to dynamically
//request the use of a component. A component can be local or
//remote, a seperate process or a library (.dll, .so).
//After a component is requested and the request succeeds,
//component objects can be accessed through the NS interface.
interface Client {
   //This operation requests the usage of a component.
   //If the component is not already activated,
   //for example if the component server process is not
   //running, the operation will activate it.
   //This operation uses the Catalog interface
   //to query and activate components.
   void request(in string componentName)
      raises(ComponentNotFound, ActivationFailure, RequestFailure);

   //frees resources of using a component
   void free(in string componentName)
      raises(ComponentNotFound, ComponentReleaseFailure);
};
```

Σχήμα 7.3: Ο IDL ορισμός για τη διεπαφή του Πελάτη του ComR.

που έχει ζητηθεί. Η request() χρησιμοποιεί αυτό το αντικείμενο για να δει αν το συστατικό είναι ενεργοποιημένο και αν όχι να το ενεργοποιήσει. Σε περίπτωση που στη request() δοθεί λάθος όνομα συστατικού δημιουργείται το ComponentNotFound exception. Αν αποτύχει η ενεργοποίηση του συστατικού δημιουργείται το exception ActivationFailure, ενώ στην περίπτωση που συμβεί οποιοδήποτε άλλο σφάλμα δημιουργείται το RequestFailure exception.

7.3.2 free()

Η free() απενεργοποιεί το συστατικό του οποίου το όνομα δέχεται ως παράμετρο.

7.4 Διεπαφή Καταλόγου και Μεταδεδομένα Συστατικών

```
//The Catalog interface uses some kind of persistant
//storage that stores information about the each
//component registered in the ComR. This information
//is encapsulated by the Catalog and ComponentObject
//interfaces to provide proper querying and activation.
interface Catalog {
   //Get a ComponentObject for the component specified
   //by componentName. The Catalog searches for a
   //component definition in it's database and creates
   //the proper ComponentObject.
   ComponentObject getComponent(in string componentName)
      raises(ComponentNotFound);
};
```

Σχήμα 7.4: Ο IDL ορισμός για τη Catalogue διεπαφή του ComR.

Στο σχήμα 7.4 φαίνεται η διεπαφή Καταλόγου (Catalogue), ορισμένη σε IDL. Το CORBA αντικείμενο Catalogue χρησιμοποιείται από το Client για

να έχει πρόσβαση στο ComponentObject του συστατικού που θέλει να ενεργοποιήσει/απενεργοποιήσει. Ορίζει μία μόνο ενέργεια, την getComponent(). Η ενέργεια αυτή δέχεται ως παράμετρο ένα όνομα συστατικού και επιστρέφει ένα ComponentObject το οποίο περιέχει όλες τις απαραίτητες πληροφορίες για την ενεργοποίηση και απενεργοποίηση του συστατικού. Οι πληροφορίες αυτές αποθηκεύονται σε μία βάση μεταδεδομένων. Το Catalogue χρησιμοποιεί αυτή τη βάση για να διαβάσει τα μεταδεδομένα των συστατικών και να δημιουργεί τα αντίστοιχα ComponentObject όταν καλείται η getComponent(). Τα μεταδεδομένα που αποθηκεύονται για κάθε συστατικό φαίνονται στο σχήμα 7.5.

Component- Βιβλιοθήκη	Component - Πρόγραμμα
Όνομα component	Όνομα component
Interface component	Interface component
Αρχείο βιβλιοθήκης	Εκτελέσιμο προγράμματος
Γλώσσα υλοποίησης	-

Component - Απομακρυσμένο
Όνομα component
Interface component
Remote Catalogue object reference
Remote NS object reference

Σχήμα 7.5: Τα δεδομένα που αποθηκεύονται ανά είδος συστατικού στη βάση μεταδεδομένων.

Υπάρχουν δύο διαφορετικές υλοποιήσεις του Catalogue. Η πρώτη βρίσκεται στο κομμάτι του XAIL Διερμηνευτή και η δεύτερη βρίσκεται στον ComR Εξυπηρετητή. Το Client, όταν καλείται σε αυτό η request(), πρώτα καλεί την getComponent() στο Catalogue του XAIL Διερμηνευτή η οποία αφού βρει το συστατικό στη βάση μεταδεδομένων ελέγχει το είδος του συστατικού. Αν είναι συστατικό-βιβλιοθήκη τότε δημιουργεί και επιστρέφει το αντίστοιχο ComponentObject. Αν είναι συστατικό-πρόγραμμα τότε επιστρέφει το ComponentObject που του επιστρέφει η getComponent() του Catalogue στον ComR Εξυπηρε-

τητή. Το CORBA αντικείμενο αυτό έχει διαφορετική υλοποίηση από το αντίστοιχο για συστατικό-βιβλιοθήκη. Τέλος, αν είναι απομακρυσμένο συστατικό τότε επιστρέφει το ComponentObject που του επιστρέφει η getComponent() του απομακρυσμένου Catalogue.

7.5 Διεπαφή Αντικειμένου Συστατικού

Στο σχήμα 7.6 φαίνεται η IDL διεπαφή του Αντικειμένου Συστατικού (ComponentObject). Υπάρχουν δύο διαφορετικές υλοποιήσεις αυτού. Στη μία, για συστατικά που είναι βιβλιοθήκες, η activate() φορτώνει τη δυναμική βιβλιοθήκη και τρέχει σε αυτή μία ρουτίνα αρχικοποίησης του συστατικού. Στην άλλη, για συστατικά που είναι ξεχωριστά προγράμματα, αν το πρόγραμμα ήδη τρέχει τότε η activate() δεν κάνει τίποτα. Αν όχι, τότε το εκτελεί και επιστρέφει.

7.6 Διεπαφή Διακομιστή

Στο σχήμα 7.7 φαίνεται η IDL διεπαφή του Διακομιστή (Server). Κάθε συστατικό όταν φορτώνεται πρέπει να δημιουργήσει τα CORBA αντικείμενα και τις αντίστοιχες αναφορές αντικειμένων (object references). Πρέπει όμως αυτά να γίνουν γνωστά στην εφαρμογή που φορτώνει το συστατικό για να μπορεί στη συνέχεια να τα χρησιμοποιήσει. Δηλαδή, το συστατικό πρέπει να δημοσιοποιήσει τις αναφορές αντικειμένων που δημιουργεί στο ComR. Αυτό γίνεται μέσω του Registar. Για να αποκτήσει το συστατικό μία αναφορά σε Registar χρησιμοποιεί την registerComponent() στο Server δίνοντας το όνομά του.

7.7 Διεπαφή Εγγραφής

Στο σχήμα 7.8 φαίνεται η IDL διεπαφή Εγγραφής (Registar). Όπως φαίνεται και στο σχήμα 7.1, τα CORBA αντικείμενα των Server και Registar βρίσκονται ή στον XAIL Διερμηνευτή ή στο Catalogue Server. Έχουν και αυτά όπως

```
//Used by Client::request() to activate components.
//There should be two different servant implementations
//of this interface. One for libraries and one for
//processes.
interface ComponentObject {
  //Query if the component is active.
  //For processes this means to check if the
  //component server process is running.
  //For libraries this means to check if the
  //library has been previously dynamically opened.
  boolean isActive();

  //Activate the component.
  //If library, it will be loaded and
  //the initialization routine called.
  //The operation will return only after
  //the initialization routine successfully returns.
  //If process the process will be spawned.
  //The operation will return immediately after
  //successfully spawning the process.
  void activate()
    raises(ActivationFailure, AlreadyActive);

  //Release resources used by use of the component.
  void deactivate()
    raises(ComponentReleaseFailure);
};
```

Σχήμα 7.6: Ο IDL ορισμός για τη ComponentObject διεπαφή του ComR.

το ComponentObject δύο διαφορετικές υλοποιήσεις ανάλογα με το είδος του συστατικού. Αν το συστατικό είναι βιβλιοθήκη, τότε οι αναφορές (references) που εξάγει πρέπει να είναι γνωστά μόνο για την εφαρμογή που το φορτώνει, άρα θα χρησιμοποιηθεί το CORBA αντικείμενο που βρίσκεται στον XAIL Διερμηνευτή. Αν από την άλλη το συστατικό είναι ένα πρόγραμμα, τότε αναφορές που εξάγει πρέπει να είναι γνωστά σε όλες τις εφαρμογές που θέλουν να το χρησιμοποιήσουν, οπότε θα χρησιμοποιηθεί το CORBA αντικείμενο που βρίσκεται στον ComR Εξυπηρετητή. Το Registar παρέχει την ενέργεια registerReference() η οποία εγγράφει μία αναφορά αντικειμένου ενός CORBA αντικειμένου και τη διεπαφή που αυτό υλοποιεί στο ComR. Με την ενέργεια registrationEnd() ένα συστατικό δηλώνει πως έχει τελειώσει με τις εγγραφές των αναφορών αντικειμένων και είναι έτοιμο να δεχτεί αιτήσεις.

7.8 Διεπαφή Υπηρεσίας Ονομάτων

Στο σχήμα 7.9 φαίνεται η IDL διεπαφή της Υπηρεσίας Ονομάτων (NS, από το αγγλικό Naming Service). Αφού μία XAIL εφαρμογή φορτώσει τα συστατικά που χρειάζεται, μπορεί να χρησιμοποιήσει τα CORBA αντικείμενα μέσω των αναφορών αντικειμένων που έχουν εγγραφεί στο ComR από τα Server/Registar που είδαμε παραπάνω. Ο XAIL Διερμηνευτής για να αποκτήσει από το ComR

```
//This is a factory object for Registars to perform
//registration of CORBA Object references of a component
//to the ComR.
interface Server {
   //Create a Registar to register references for
   //the component componentName
   Registar registerComponent(in string componentName)
      raises(ComponentNotFound);
};
```

Σχήμα 7.7: Ο IDL ορισμός για τη Server διεπαφή του ComR.

```
//This is interface is used by a component implementation,
//either a process or a library for server side activation
//and object registration.
interface Registar {
  //Register a CORBA Object Refernce
  //Intended for use component implementation initialization.
  //Either a library or a process each component, has to register
  //its object refernces to the ComR. This is done in the main()
  //of a process, or in the initilization function of a library.
  //Registar will store a mapping between the object name and
  //the CORBA Object Reference so that the reference can be
  //retrieved by NS. The object name is automatically generated
  //from the RepositoryId of the IDL interface of the object.
  void registerReference(in Object objectReference)
    raises(RegistrationFailure);

  //Signifies the end of component registration.
  //When this operation is called the component cannot
  //register other objects to the ComR. Every component
  //has to call this after registering all of its references.
  //A component that does not call this will be unusable.
  void registrationEnd()
    raises(RegistrationFailure);
};
```

Σχήμα 7.8: Ο IDL ορισμός για τη Registar διεπαφή του ComR.

```
//This interface is used by the XAIL runtime to dynamically
//retreive object references.
interface NS {
   //Get an object reference specified by the objectName
   //Object names are in the form componentName::interfacePath
   //examples:
   //QuoterTransient::Quoter.StockFactory
   //QuoterService::Quoter.StockFactory
   //If the component is a process then NS checks if the component
   //has completed it's registration with Registar::registrationEnd().
   //If registration has not been completed then NS checks if
   //the process is running and waits for a small
   //amount of time before trying again to allow the process to complete
   //registration. If again registration does not occur then the
   //component is declared unusable.
   Object getReference(in string objectName)
      raises(ObjectNotFound, ComponentUnusable);
};
```

Σχήμα 7.9: Ο IDL ορισμός για τη NS διεπαφή του ComR.

κάποια αναφορά αντικειμένου χρησιμοποιεί το NS και την ενέργεια getReference(). Η παράμετρος που δέχεται είναι μία συμβολοσειρά που περιγράφει τη διεπαφή του αντικειμένου του οποίου ζητείται η αναφορά και σε ποιο συστατικό βρίσκεται. Όπως τα Catalogue, ComponentObject, Server και Registar αντικείμενα, έτσι και το NS έχει δύο υλοποιήσεις. Η πρώτη βρίσκεται στον XAIL Διερμηνευτή και έχει πρόσβαση στις αναφορές αντικειμένων που εγγράφηκαν από συστατικά που είναι βιβλιοθήκες. Η δεύτερη βρίσκεται στο Catalogue Process και έχει πρόσβαση στις αναφορές που εγγράφηκαν από συστατικά-προγράμματα. Ο XAIL Διερμηνευτής για να εκτελέσει μια ενέργεια πάνω σε ένα αντικείμενο ενός συστατικού καλεί την getReference(). Αυτή ψάχνει να βρει την αναφορά αντικειμένου της διεπαφής που της δίνεται και αν δεν το βρει μεταβιβάζει την κλήση στο NS που βρίσκεται στον ComR Εξυπηρετητή.

Κεφάλαιο 8

Παράδειγμα XAIL Εφαρμογής

Στα προηγούμενα κεφάλαια είδαμε τις προδιαγραφές και τα στοιχεία της γλώσσας XAIL, ορίσαμε τη μορφή των συστατικών λογισμικού και πως αυτά αναπτύσσονται και χρησιμοποιούνται σε μια Διεπαφή ορισμένη με XAIL. Σε αυτό το κεφάλαιο θα δώσουμε ένα παράδειγμα που θα μεταφέρει τη θεωρία των προηγούμενων κεφαλαίων σε πράξη.

8.1 Προδιαγραφές Εφαρμογής

Ας ξεκινήσουμε το παράδειγμά μας ορίζοντας τις προδιαγραφές της εφαρμογής, δηλαδή τις λειτουργίες της. Θέλουμε, λοιπόν, μια εφαρμογή λήψης τιμών μετοχών και συγκεκριμένα δύο γνωστών εταιριών, της Microsoft και RedHat. Ας ονομάσουμε την εφαρμογή Favorite Stock Quoter. Οι λειτουργίες της εφαρμογής είναι απλές. Επιλέγοντας την εταιρία διαβάζονται και εμφανίζονται τα στοιχεία της μετοχής της εταιρίας που θα περιλαμβάνουν και την τιμή[1].

[1]Σε μια πραγματική εφαρμογή λήψης τιμών, θα υπήρχαν πάνω από δύο διαθέσιμες μετοχές από τις οποίες θα μπορούσε να διαλέξει ο χρήστης και η ενημέρωση των δεδομένων θα γινόταν σε πραγματικό χρόνο. Κάτι τέτοιο όμως ξεφεύγει από το σκοπό του παραδείγματος.

8.2 Ορισμός Συστατικών Λογισμικού

Γνωρίζοντας, λοιπόν, τις προδιαγραφές λειτουργιών της εφαρμογής που πρέπει να αναπτύξουμε, θα ορίσουμε τη διεπαφή της επιχειρησιακής λογικής που θα οδηγεί την εφαρμογή. Οι λειτουργίες είναι πολύ απλές, οπότε θα αναπτύξουμε μόνο ένα συστατικό λογισμικού που θα ονομάσουμε Quoter.

```
module Quoter
{
  exception InvalidStockSymbol {};

  interface Stock;

  interface StockFactory
  {
    Stock getStock(in string stockSymbol)
      raises(InvalidStockSymbol);
  };

  interface Stock
  {
    double getPrice();
    string getSymbol();
    string getFullName();
  };
};
```

Σχήμα 8.1: Ορισμός της IDL διεπαφής του συστατικού λογισμικού Quoter.

Στο σχήμα 8.1 ορίζουμε τη διεπαφή του συστατικού λογισμικού σε IDL. Συγκεκριμένα, ορίζουμε τις διεπαφές δύο αντικειμένων, του *StockFactory* και του *Stock*. Το *StockFactory* περιέχει μία μέθοδο τη *getStock* που παίρνει ως παράμετρο το συμβολικό όνομα της μετοχής και μας επιστρέφει ένα αντικείμενο *Stock* το οποίο περιέχει όλα τα στοιχεία της. Οπότε για να έχουμε πρόσβαση στα στοιχεία μιας μετοχής πρώτα πρέπει να χρησιμοποιήσουμε αυτή τη μέθοδο.

Για τη *getStock* ορίζεται και ένας τύπος εξαίρεσης που εγείρεται όταν το *StockFactory* δεν βρει τη μετοχή που του ζητηθεί. Η διεπαφή *Stock* ορίζει μεθόδους που επιστρέφουν τις τιμές των διάφορων στοιχείων μιας μετοχής.

Αφού ορίσαμε τη διεπαφή του συστατικού λογισμικού Quoter που θα αποτελέσει την επιχειρησιακή λογική της εφαρμογής έχουμε το ελεύθερο να ορίσουμε τη λογική παρουσίασης με τη γλώσσα XAIL και να υλοποιήσουμε το συστατικό λογισμικού. Αυτά μπορούν να γίνουν παράλληλα από διαφορετικά άτομα χωρίς καμιά άλλη πληροφορία. Όμως, χάριν παραδείγματος, ας υποθέσουμε ότι πρώτα θα σχεδιαστεί η διεπαφή της εφαρμογής και η λογική παρουσίασης με τη γλώσσα XAIL.

8.3 Ορισμός Λογικής Παρουσίασης με XAIL

Αφού γνωρίζουμε την IDL διεπαφή του συστατικού λογισμικού Quoter που θέλουμε να χρησιμοποιήσουμε ως επιχειρησιακή λογική της γραφικής διεπαφής μας, μπορούμε ελεύθερα με τη γλώσσα XAIL να ορίσουμε τη λογική παρουσίασης της εφαρμογής Favorite Stock Quoter.

XAIL Ορισμός της Διεπαφής

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xail SYSTEM "xail.dtd">
<xail application="Favorite Stock Quoter" version="1.0">
  <using>Quoter</using>
  <Window title="Favorite Stock Quoter">
    <Label id="AppState">
      Welcome!! Please click one of the buttons below
      to get a quote for your favorite company.
    </Label>
    <HBox>
      <Button id="RedHat">RedHat</Button>
```

```
      <Button id="Microsoft">Microsoft</Button>
    </HBox>
    <VBox id="StockQuotePanel" visible="false">
      <HBox>
        <Label>Company: </Label>
        <Label id="StockFullName"></Label>
      </HBox>
      <HBox>
        <Label>Stock Symbol: </Label>
        <Label id="StockSymbol"></Label>
      </HBox>
      <HBox>
        <Label>Current Price: </Label>
        <Label id="StockPrice"></Label>
      </HBox>
      <Button id="HideQuote">Hide</Button>
    </VBox>
    <on>
      <event type="click" object="//Button[@id='RedHat']">
        <alter object="//Label[@id='StockFullName']">
Quoter::Quoter.StockFactory.getStock("RHAT").getFullName()
        </alter>
        <alter object="//Label[@id='StockSymbol']">
Quoter::Quoter.StockFactory.getStock("RHAT").getSymbol()
        </alter>
        <alter object="//Label[@id='StockPrice']">
Quoter::Quoter.StockFactory.getStock("RHAT").getPrice()+' USD'
        </alter>
        <alter object="//VBox[@id='StockQuotePanel']" attr="visible">
          'true'
```

```
        </alter>
      </event>
      <event type="click" object="//Button[@id='Microsoft']">
        <alter object="//Label[@id='StockFullName']">
Quoter::Quoter.StockFactory.getStock("MSFT").getFullName()
        </alter>
        <alter object="//Label[@id='StockSymbol']">
Quoter::Quoter.StockFactory.getStock("MSFT").getSymbol()
        </alter>
        <alter object="//Label[@id='StockPrice']">
Quoter::Quoter.StockFactory.getStock("MSFT").getPrice()+' USD'
        </alter>
        <alter object="//VBox[@id='StockQuotePanel']" attr="visible">
          'true'
        </alter>
      </event>
      <event type="click" object="//Button[@id='HideQuote']">
        <alter object="//VBox[@id='StockQuotePanel']" attr="visible">
          'false'
        </alter>
      </event>
    </on>
  </Window>
</xail>
```

Στο παραπάνω XAIL έγγραφο, είναι χαρακτηριστική η παρουσία του στοιχείου *using* με το οποίο δηλώνουμε προς το διερμηνευτή της XAIL ότι πρέπει να χρησιμοποιηθεί το συστατικό λογισμικού *Quoter*. Για τη διεπαφή χρησιμοποιούμε ένα παράθυρο το οποίο περιέχει μια ετικέτα (Label), και μια οριζόντια ομαδοποίηση (HBox) που στοιχίζει δύο κουμπιά. Επιπλέον, χρησιμοποιούμε ένα συνδυασμό μιας κάθετης ομαδοποίησης και τριών οριζόντιων ομαδοποιήσεων

για να δημιουργήσουμε έναν πίνακα των στοιχείων μιας επιλεγμένης μετοχής. Αυτή η δομή είναι αρχικά μη ορατή στη διεπαφή καθώς ορίζουμε το χαρακτηριστικό *visible* της κάθετης ομαδοποίησης ως *false* αποκρύπτοντας έτσι όλα τα περιεχόμενά της.

Ανάλογα με το ποιο κουμπί πατήσει ο χρήστης, θα πρέπει ο πίνακας των στοιχείων της μετοχής να γεμίζει με τα κατάλληλα δεδομένα. Αυτό ορίζεται στους χειριστές γεγονότων για τα δύο κουμπιά. Και οι δύο ενημερώνουν την κάθε ετικέτα του πίνακα με το αποτέλεσμα μιας κλήσης προς το συστατικό λογισμικού Quoter. Στις κλήσεις αυτές λαμβάνεται το αντίστοιχο *Stock* αντικείμενο που έχει οριστεί στην IDL διεπαφή του συστατικού λογισμικού (βλέπε σχήμα 8.1) και εκτελούνται σε αυτό οι ενέργειες που επιστρέφουν τα δεδομένα της μετοχής.

8.4 Υλοποίηση Συστατικού Λογισμικού

Σε αυτή την ενότητα θα δώσουμε την υλοποίηση του συστατικού λογισμικού Quoter που χρησιμοποιείται από τη διεπαφή που ορίσαμε στην προηγούμενη ενότητα με τη γλώσσα XAIL. Η ανάπτυξη της υλοποίησης του συστατικού λογισμικού γίνεται τελείως ανεξάρτητα από τη σχεδίαση της Γραφικής Διεπαφής. Ο κώδικας που αναπτύσσεται δεν έχει καμία πρόσβαση προς αυτή και δεν υπάρχει κώδικας, όπως στους «κλασικούς» χειριστές γεγονότων, που συνδυάζει λογική παρουσίασης και επιχειρησιακή λογική. Αξίζει να σημειώσουμε πως το συστατικό λογισμικού θα μπορούσε να χρησιμοποιηθεί αυτούσιο και για κάποιο τελείως διαφορετικό σκοπό εκτός από το ρόλο της επιχειρησιακής λογικής της εφαρμογής που αναπτύσσουμε.

Επιλέγουμε να υλοποιήσουμε το συστατικό λογισμικού σε περιβάλλον προγραμματισμού C++ ως δυναμική βιβλιοθήκη. Ο κώδικας που παραθέτουμε είναι από τρία ξεχωριστά αρχεία κώδικα. Το quoter.cpp, το οποίο περιλαμβάνει την υλοποίηση της ρουτίνας εισόδου στη βιβλιοθήκη και χρησιμοποιεί της διεπαφές του ComR για την εκκίνηση και εγγραφή του συστατικού λογισμικού.

Το QuoterI.h είναι το αρχείο επικεφαλίδας που δημιουργείται αυτόματα από τη μεταγλώττιση της IDL διεπαφής του συστατικού λογισμικού και περιλαμβάνει τις δηλώσεις των κλάσεων που θα υλοποιήσουν τα αντικείμενα *StockFactory* και *Stock* που ορίζονται σε αυτή. Το αρχείο QuoterI.cpp περιλαμβάνει την υλοποίηση αυτών των κλάσεων. Σε αυτό το σημείο να σημειώσουμε πως η μεταγλώττιση της IDL διεπαφής του συστατικού λογισμικού Quoter δημιουργεί και άλλα αρχεία σύμφωνα με το πρότυπο CORBA αλλά δεν τα περιλαμβάνουμε καθώς θα ήταν πέρα από το σκοπό του παραδείγματος να τα αναλύσουμε. Για τον ίδιο λόγο παραλείπουμε να εξηγήσουμε τον κώδικα γραμμή-γραμμή καθώς βασίζεται πολύ στο πρότυπο CORBA. Για την πλήρη κατανόηση του κώδικα απαιτείται γνώση του προτύπου, οπότε παραπέμπουμε τον αναγνώστη στη σχετική βιβλιογραφία.

quoter.cpp

```
//quoter.cpp

#include <iostream>

#include <tao/IFR_Client/IFR_Client_Adapter_Impl.h>
#include <tao/IFR_Client/IFR_BasicC.h>

#include "QuoterI.h"
#include "../../comr/ComRC.h"

using std::cout;
using std::endl;

//the servant
Quoter_StockFactory_i stock_factory_i;
//the reference
```

```
Quoter::StockFactory_var stock_factory;

//RootPOA
PortableServer::POA_ptr root_poa;

extern "C" int component_init(CORBA::ORB_ptr orb, XAIL::ComR::Server_ptr serve
  if(CORBA::is_nil(orb)) {
    cerr<<"QUOTER: ORB reference is nil"<<endl;
    return -1;
  }

  if(CORBA::is_nil(server)) {
    cerr<<"QUOTER: Server reference is nil"<<endl;
    return -1;
  }

  try {
    CORBA::Object_var object=orb->resolve_initial_references("RootPOA");
    root_poa=PortableServer::POA::_narrow(object.in());
    if(CORBA::is_nil(root_poa)) {
      cerr<<"QUOTER: Failed to obtain RootPOA object reference"<<endl;
      return -1;
    }

    XAIL::ComR::Registar_var registar=server->registerComponent("Quoter");
    if(CORBA::is_nil(registar.in())) {
      cerr<<"QUOTER: Registar object reference is nil"<<endl;
      return -1;
    }
```

```
    //Activate the servant to obtain the object reference
    stock_factory=stock_factory_i._this();

    //register the created reference
    registar->registerReference(stock_factory.in(),
CORBA::Object::_duplicate(stock_factory->_get_interface()));

    registar->registrationEnd();
  } catch(CORBA::Exception &e) {
    cerr<<"QUOTER: CORBA Exception: "<<e<<endl;
    return -1;
  }

  return 0;
}
```

QuoterI.h

```
//QuoterI.h

#ifndef QUOTERI_H_
#define QUOTERI_H_

#include <string>

#include "QuoterS.h"

#if !defined (ACE_LACKS_PRAGMA_ONCE)
#pragma once
#endif /* ACE_LACKS_PRAGMA_ONCE */
```

```
class  Quoter_Stock_i
  : public virtual POA_Quoter::Stock
{
  std::string symbol_;
  std::string full_name_;
  CORBA::Double price_;
public:
  // Constructor
  Quoter_Stock_i (const char *symbol, const char *full_name, CORBA::Double pri

  // Destructor
  virtual ~Quoter_Stock_i (void);

  virtual
  CORBA::Double getPrice ( )
    throw (
      CORBA::SystemException
    );

  virtual
  char * getSymbol ( )
    throw (
      CORBA::SystemException
    );

  virtual
  char * getFullName ( )
    throw (
      CORBA::SystemException
    );
```

```
};

class  Quoter_StockFactory_i
  : public virtual POA_Quoter::StockFactory
{
  Quoter_Stock_i *rhat_;
  Quoter_Stock_i *msft_;
public:
  // Constructor
  Quoter_StockFactory_i (void);

  // Destructor
  virtual ~Quoter_StockFactory_i (void);

  virtual
  ::Quoter::Stock_ptr getStock (
      const char * stockSymbol
    )
    throw (
      CORBA::SystemException,
      ::Quoter::InvalidStockSymbol
    );
};
#endif /* QUOTERI_H_  */
```

QuoterI.cpp

```
#include <iostream>

#include <cstring>
```

```
#include "QuoterI.h"

using namespace std;

// Implementation skeleton constructor
Quoter_StockFactory_i::Quoter_StockFactory_i (void)
{
  rhat_=new Quoter_Stock_i("RHAT", "RedHat, Inc.", 210);
  msft_=new Quoter_Stock_i("MSFT", "Microsoft, Inc.", 91);
}

// Implementation skeleton destructor
Quoter_StockFactory_i::~Quoter_StockFactory_i (void)
{ }

::Quoter::Stock_ptr Quoter_StockFactory_i::getStock (
    const char * stockSymbol
  )
  throw (
    CORBA::SystemException,
    ::Quoter::InvalidStockSymbol
  )
{
  if (strcmp (stockSymbol, "RHAT") == 0) {
    return this->rhat_->_this();
  } else if (strcmp (stockSymbol, "MSFT") == 0) {;
    return this->msft_->_this();
  }

  throw Quoter::InvalidStockSymbol ();
```

```
}

// Implementation skeleton constructor
Quoter_Stock_i::Quoter_Stock_i (const char *symbol,
                                const char *full_name,
                                CORBA::Double price)
  :  symbol_ (symbol),
     full_name_ (full_name),
     price_ (price)
{ }

// Implementation skeleton destructor
Quoter_Stock_i::~Quoter_Stock_i (void)
{ }

CORBA::Double Quoter_Stock_i::getPrice ( )
  throw (
    CORBA::SystemException
  )
{
  return this->price_;
}

char * Quoter_Stock_i::getSymbol ( )
  throw (
    CORBA::SystemException
  )
{
  return CORBA::string_dup (this->symbol_.c_str ());
}
```

```
char * Quoter_Stock_i::getFullName ( )
  throw (
    CORBA::SystemException
  )
{
  return CORBA::string_dup (this->full_name_.c_str());
}
```

8.5 Εκτέλεση

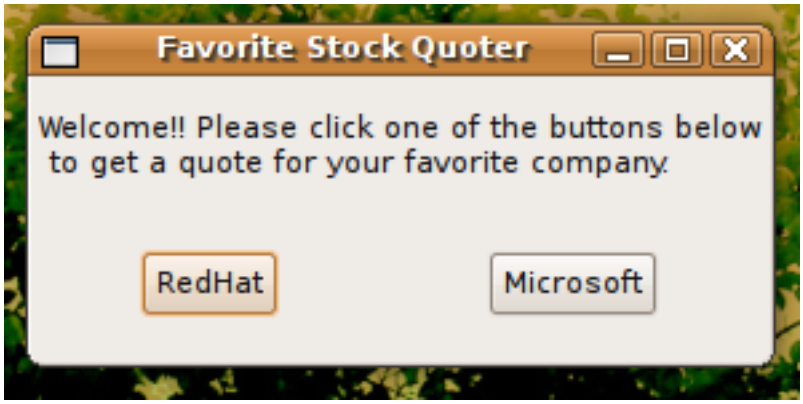
Στις προηγούμενες ενότητες ορίσαμε τη λογική παρουσίασης της εφαρμογής με τη γλώσσα XAIL και υλοποιήσαμε τη διεπαφή της επιχειρησιακής λογικής της. Σε αυτό το σημείο θα δείξουμε εικόνες από την εκτέλεση της εφαρμογής μέσω του διερμηνευτή της XAIL γλώσσας.

Αρχικά, ο διερμηνευτής θα διαβάσει το XAIL έγγραφο της ενότητας 8.3, στο οποίο ορίζεται η λογική παρουσίασης της εφαρμογής. Θα διαπιστώσει, διαβάζοντας το στοιχείο *<using>*, ότι η εφαρμογή χρησιμοποιεί το συστατικό λογισμικού *Quoter* και θα χρησιμοποιήσει την υπηρεσία ComR για να το φορτώσει. Στη συνέχεια θα εμφανίσει στην οθόνη τα Γραφικά Στοιχεία όπως ορίζονται αρχικά στο έγγραφο. Για παράδειγμα, το Γραφικό Στοιχείο VBox με StockQuotePanel ως τιμή του χαρκτηριστικού id δε θα εμφανιστεί αρχικά, αφού το χαρακτηριστικό visible έχει την τιμή false. Έτσι, ο διερμηνευτής θα εμφανίσει αρχικά στην οθόνη τη Γραφική Διεπαφή Χρήστη του σχήματος 8.2.

Σε αυτή τη διεπαφή εμφανίζονται δύο κουμπιά. Για το κάθε κουμπί, έχει οριστεί με XAIL τι θα γίνει αν αυτό πατηθεί με το ποντίκι. Συγκεκριμένα, ο διερμηνευτής θα χρησιμοποιήσει το συστατικό λογισμικό Quoter, όπως αυτό ορίζεται στις υπολογίσιμες εκφράσεις των στοιχείων <alter>, για να ενημερώσει τα κείμενα των ετικετών μιας στήλης με τα στοιχεία μιας μετοχής. Στη συνέχεια, κάνει αυτή τη στήλη ορατή τροποποιώντας το χαρακτηριστικό visible

του Γραφικού Στοιχείου VBox που περιέχει αυτές τις ετικέτες. Έτσι, αν ο χρήστης επιλέξει να πατήσει το κουμπί 'RedHat', η Γραφική Διεπαφή θα τροποποιηθεί στην εικόνα του σχήματος 8.3. Αν ο χρήστης επιλέξει να πατήσει το κουμπί 'Microsoft', η Γραφική Διεπαφή θα τροποποιηθεί στην εικόνα του σχήματος 8.4.

Τέλος, αν ο χρήστης επιλέξει να πατήσει το κουμπί 'Hide', όπως μπορούμε να καταλάβουμε από τις XAIL δηλώσεις, τα στοιχεία της μετοχής θα εξαφανιστούν και η Γραφική Διεπαφή θα τροποποιηθεί πάλι στην αρχική της εικόνα, η οποία φαίνεται στο σχήμα 8.2.



Σχήμα 8.2: Εκτέλεση μιας εφαρμογής από τον XAIL διερμηνευτή (1).

Σχήμα 8.3: Εκτέλεση μιας εφαρμογής από τον XAIL διερμηνευτή (2).

Σχήμα 8.4: Εκτέλεση μιας εφαρμογής από τον XAIL διερμηνευτή (3).

Παράρτημα Α΄

Ορισμός Τύπου Εγγράφου της Γλώσσας XAIL

Παρακάτω δίνουμε τον Ορισμό Τύπου Εγγράφου (Document Type Definition - DTD) της γλώσσας XAIL. Το DTD ορίζει τους συνταχτικούς κανόνες της γλώσσας και χρησιμοποιείται για την επαλήθευση ενός XAIL εγγράφου. Η σημασιολογία των στοιχείων της γλώσσας και παραδείγματα χρήσης τους αναφέρονται στην ενότητα 5.3.

```
<!ELEMENT xail (using*, Window)>
<!ATTLIST xail application CDATA #IMPLIED>
<!ATTLIST xail version CDATA #IMPLIED>

<!ELEMENT using (#PCDATA)>

<!ELEMENT Window ((Button|Label|VBox|HBox|MenuBar|on|event)*)>
<!ATTLIST Window id ID #IMPLIED>
<!ATTLIST Window title CDATA #IMPLIED>

<!ELEMENT Button (#PCDATA)>
```

```
<!ATTLIST Button id ID #IMPLIED>
<!ATTLIST Button focus (true|false) "false">
<!ATTLIST Button visible (true|false) "true">

<!ELEMENT Label (#PCDATA)>
<!ATTLIST Label id ID #IMPLIED>
<!ATTLIST Label focus (true|false) "false">
<!ATTLIST Label visible (true|false) "true">
<!ATTLIST Label selectable (true|false) "true">

<!ELEMENT VBox ((Button|Label|VBox|HBox|MenuBar|on|event)*)>
<!ATTLIST VBox id ID #IMPLIED>
<!ATTLIST VBox focus (true|false) "false">
<!ATTLIST VBox visible (true|false) "true">
<!ATTLIST VBox homogenous (true|false) "false">

<!ELEMENT HBox ((Button|Label|VBox|HBox|MenuBar|on|event)*)>
<!ATTLIST HBox id ID #IMPLIED>
<!ATTLIST HBox focus (true|false) "false">
<!ATTLIST HBox visible (true|false) "true">
<!ATTLIST HBox homogenous (true|false) "false">

<!ELEMENT MenuBar ((Menu|on|event)*)>
<!ATTLIST MenuBar id ID #IMPLIED>
<!ATTLIST MenuBar focus (true|false) "false">
<!ATTLIST MenuBar visible (true|false) "true">

<!ELEMENT Menu ((Item|Menu|on|event)*)>
<!ATTLIST Menu id ID #IMPLIED>
<!ATTLIST Menu focus (true|false) "false">
```

```
<!ATTLIST Menu visible (true|false) "true">
<!ATTLIST Menu title CDATA #REQUIRED>

<!ELEMENT Item (#PCDATA)>
<!ATTLIST Item id ID #IMPLIED>
<!ATTLIST Item focus (true|false) "false">
<!ATTLIST Item visible (true|false) "true">

<!ELEMENT on (event*)>
<!ATTLIST on id ID #IMPLIED>

<!ELEMENT event ((execute|alter|insert|remove)*)>
<!ATTLIST event id ID #IMPLIED>
<!ATTLIST event type (click|mouseOver|focus|press) #REQUIRED>
<!ATTLIST event object CDATA #REQUIRED>

<!ELEMENT execute (#PCDATA)>
<!ATTLIST execute id ID #IMPLIED>

<!ELEMENT alter (#PCDATA)>
<!ATTLIST alter id ID #IMPLIED>
<!ATTLIST alter object CDATA #REQUIRED>
<!ATTLIST alter attr CDATA #IMPLIED>

<!ELEMENT insert ((Button|Label|VBox|HBox|MenuBar|Menu|Item|on
|event|execute|alter|insert|remove)*)>
<!ATTLIST insert id ID #IMPLIED>
<!ATTLIST insert object CDATA #REQUIRED>

<!ELEMENT remove EMPTY>
```

```
<!ATTLIST remove id ID #IMPLIED>
<!ATTLIST remove object CDATA #REQUIRED>
```

Παράρτημα Β′

Γραμματική των Υπολογίσιμων Εκφράσεων XAIL

```
<XCE> ::= <AdditiveExpression>
      | <XCE> "," <AdditiveExpression>

<AdditiveExpression> ::= <MultiplicativeExpression>
                     | <AdditiveExpression> "+" <MultiplicativeExpression>
                     | <AdditiveExpression> "-" <MultiplicativeExpression>

<MultiplicativeExpression> ::= <UnaryExpression>
                           | <MultiplicativeExpression> "*" <UnaryExpression>
                           | <MultiplicativeExpression> "/" <UnaryExpression>

<UnaryExpression> ::= <PostfixExpression>
                  | "++" <UnaryExpression>
                  | "--" <UnaryExpression>
                  | "+" <UnaryExpression>
                  | "-" <UnaryExpression>
```

```
<PostfixExpression> ::= <PrimaryExpression>
                      | <BusinessLogicExpression>
                      | <PostfixExpression> "++"
                      | <PostfixExpression> "--"

<BusinessLogicExpression> ::= <ComponentOperationExpression>
         | <BusinessLogicExpression> "." <FollowupOperationExpression>

<FollowupOperationExpression>
         ::= {identifier} "(" <ArgumentExpressionList> ")"
         | {identifier} "(" ")"

<ComponentOperationExpression>
         ::= <ComponentPathExpression> "(" <ArgumentExpressionList> ")"
         | <ComponentPathExpression> "(" ")"

<ArgumentExpressionList> ::= <AdditiveExpression>
         | <ArgumentExpressionList> "," <AdditiveExpression>

<ComponentPathExpression> ::= <ComponentInterfacePath>
                            | {identifier} "::" <ComponentInterfacePath>

<ComponentInterfacePath> ::= {identifier}
                           | {identifier} "." <ComponentInterfacePath>

<PrimaryExpression> ::= {LiteralNumeric}
                      | {LiteralString}
                      | "(" <XCE> ")"

{identifier}  ::= [[:alpha:]][[:alpha:]_[:digit:]]*
```

```
{LiteralNumeric} ::= {literalInteger}|{literalReal}
{literalInteger} ::= [[:digit:]]+
{literalReal}
  ::= {literalInteger}\.{literalInteger}([eE][+-]?{literalInteger})?
{LiteralString}
  ::=  (\"{literalStringContentDQ}\")|(\'{literalStringContentSQ}\')
{literalStringContentSQ} ::= ({escapeChar}|{commonChar}|\"|[[:blank:]])*
{literalStringContentDQ} ::= ({escapeChar}|{commonChar}|\'|[[:blank:]])*
{escapeChar} ::= \\(n|t|r|a|0|\\|\'|\")
{commonChar} ::= [] !#-&(-[^-~]
```

Παράρτημα Γ′

Γλωσσάρι: Ελληνικά - Αγγλικά

Α

αναφορά αντικειμένου: object reference
Ανεξαρτησία Γλώσσας Προγραμματισμού: Programming Language Neutrality
αντιστοιχίσεις: mappings
Απομακρυσμένες Κλήσεις Διαδικασιών: RPC - Remote Procedure Call
αρχιτεκτονική 3 επιπέδων: 3-tier architecture

Γ

γεγονός: event
Γλώσσα Ορισμού Διεπαφών: IDL - Interface Description Language
Γλώσσες Διεπαφής Χρήστη σε XML: XML User Interface Languages
Γραφική Διεπαφή Χρήστη: GUI - Graphical User Interfaces
Γραφικό Στοιχείο: widget

Γρήγορη Ανάπτυξη Εφαρμογών: RAD - Rapid Application Development

Δ

διαλειτουργικότητα: interoperability
διανυσματικά γραφικά: vector graphics
Διαφάνεια Τοποθεσίας: Location Transparency
διεπαφή: interface
Διεπαφή Προγραμματισμού Εφαρμογών: API - Application Programming Interface

Ε

ενδιάμεσο λογισμικό: middleware
εξυπηρετητής: server
Επεκτάσιμη Γλώσσα Διεπαφών Εφαρμογών: XAIL - eXtensible Application Interface Language
Επεκτάσιμη Γλώσσα Σήμανσης: XML - eXtensible Markup Language
Επεκτάσιμη Γλώσσα Σήμανσης Εφαρμογών: XAML - Extensible Application Markup Language
επιχειρησιακή λογική: business logic
εργαλειοθήκη λογισμικού: toolkit
ετικέτα: label
Εφαρμογές Επιφάνειας Εργασίας: Desktop Applications
Εφαρμογές Παγκόσμιου Ιστού: Web Applications

Θ

θέμα: theme

Κ

Κοινή Πρόσβαση Χρήστη: Common User Access

Λ

λογική οδηγούμενη από γεγονότα: event driven paradigm
λογικής παρουσίασης: presentation logic

Μ

μελέτη περίπτωσης: case study
Μοντέλο Όψη Ελεγκτής: MVC - Model View Controller
Μοντέλο Συστατικών: Component Model
μοτίβο σχεδίασης λογισμικού: design pattern

Ο

Ότι Βλέπεις Παίρνεις: WYSIWYG - What You See Is What You Get

Π

Παράθυρα Εικονίδια Επιλογείς και Δείκτες: WIMP - Windows, Icons, Menus and Pointing Devices
πελάτης: client
πλήκτρο εντολής: button
Πλούσιες Εφαρμογές Διαδικτύου: RIA - Rich Internet Applications
Πλούσιες Εφαρμογές Επιφάνειας Εργασίας: RDA - Rich Desktop Applications
προσαρμογέας αντικειμένων: object adapter

Σ

συσκευή δείκτη: pointing device
συστατικά στοιχεία λογισμικού: components

Φ

φυλλομετρητής: browser

Χ

χαρακτηριστικό: attribute
χειρισμός γεγονότων: event handling
χειριστής γεγονότος: event handler

Παράρτημα Δ΄

Γλωσσάρι: Αγγλικά - Ελληνικά

0-9

3-tier architecture: αρχιτεκτονική 3 επιπέδων

A

API - Application Programming Interface: Διεπαφή Προγραμματισμού Εφαρμογών
attribute: χαρακτηριστικό

B

browser: φυλλομετρητής
business logic: επιχειρησιακή λογική
button: πλήκτρο εντολής

C

case study: μελέτη περίπτωσης
client: πελάτης
Common User Access: Κοινή Πρόσβαση Χρήστη
Component Model: Μοντέλο Συστατικών
components: συστατικά στοιχεία λογισμικού

D

design pattern: μοτίβο σχεδίασης λογισμικού
Desktop Applications: Εφαρμογές Επιφάνειας Εργασίας

E

event: γεγονός
event driven paradigm: λογική οδηγούμενη από γεγονότα
event handler: χειριστής γεγονότος
event handling: χειρισμός γεγονότων

G

GUI - Graphical User Interfaces: Γραφική Διεπαφή Χρήστη

I

IDL - Interface Description Language: Γλώσσα Ορισμού Διεπαφών
interface: διεπαφή
interoperability: διαλειτουργικότητα

L

label: ετικέτα
Location Transparency: Διαφάνεια Τοποθεσίας

M

mappings: αντιστοιχίσεις
middleware: ενδιάμεσο λογισμικό
MVC - Model View Controller: Μοντέλο Όψη Ελεγκτής

O

object adapter: προσαρμογέας αντικειμένων
object reference: αναφορά αντικειμένου

P

pointing device: συσκευή δείκτη
presentation logic: λογικής παρουσίασης
Programming Language Neutrality: Ανεξαρτησία Γλώσσας Προγραμματισμού

R

RAD - Rapid Application Development: Γρήγορη Ανάπτυξη Εφαρμογών
RDA - Rich Desktop Applications: Πλούσιες Εφαρμογές Επιφάνειας Εργασίας

RIA - Rich Internet Applications: Πλούσιες Εφαρμογές Διαδικτύου
RPC - Remote Procedure Call: Απομακρυσμένες Κλήσεις Διαδικασιών

S

server: εξυπηρετητής

T

theme: θέμα
toolkit: εργαλειοθήκη λογισμικού

V

vector graphics: διανυσματικά γραφικά

W

Web Applications: Εφαρμογές Παγκόσμιου Ιστού
widget: Γραφικό Στοιχείο
WIMP - Windows, Icons, Menus and Pointing Devices: Παράθυρα Εικονίδια Επιλογείς και Δείκτες
WYSIWYG - What You See Is What You Get: Ότι Βλέπεις Παίρνεις

X

XAIL - eXtensible Application Interface Language: Επεκτάσιμη Γλώσσα Διεπαφών Εφαρμογών
XAML - Extensible Application Markup Language: Επεκτάσιμη Γλώσσα Σήμανσης Εφαρμογών

XML - eXtensible Markup Language: Επεκτάσιμη Γλώσσα Σήμανσης
XML User Interface Languages: Γλώσσες Διεπαφής Χρήστη σε XML

Αναφορές

[1] A. Dix, J. E. Finlay, G. D. Abowd, and R. Beale, *Human-Computer Interaction*, 3rd ed. Prentice Hall, 2003.

[2] B. A. Myers and M. B. Rosson, "Survey on user interface programming," in *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 1992, pp. 195–202.

[3] B. A. Myers, "User interface software tools," *ACM Trans. Comput.-Hum. Interact.*, vol. 2, no. 1, pp. 64–103, 1995.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[5] E. R. Harold and W. S. Means, *XML In A Nutshell*, 3rd ed. O'Reilly, 2004.

[6] "XML User Interface Language (XUL) 1.0," http://www.mozilla.org/projects/xul/xul.html Χρόνος προσπέλασης: 10 Σεπτεμβρίου 2008 ώρα 22:11.

[7] L. A. MacVittie, *XAML in a Nutshell: A Desktop Quick Reference*. O'Reilly, 2006.

[8] M. Abrams, C. Phanouriou, A. L. Batongbacal, S. M. Williams, and J. E. Shuster, "Uiml: an appliance-independent xml user interface language," in *WWW '99: Proceedings of the eighth international conference on World Wide Web*. New York, NY, USA: Elsevier North-Holland, Inc., 1999, pp. 1695–1708.

[9] B. Myers, S. E. Hudson, and R. Pausch, "Past, present, and future of user interface software tools," *ACM Trans. Comput.-Hum. Interact.*, vol. 7, no. 1, pp. 3–28, 2000.

[10] S. Trewin, G. Zimmermann, and G. Vanderheiden, "Abstract user interface representations: how well do they support universal access?" in *CUU '03: Proceedings of the 2003 conference on Universal usability*. New York, NY, USA: ACM, 2003, pp. 77–84.

[11] B. Shneiderman and C. Plaisant, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 4th ed. Addison Wesley, 2004.

[12] B. A. Myers, "A brief history of human-computer interaction technology," *interactions*, vol. 5, no. 2, pp. 44–54, 1998.

[13] Γιάννης Ιωαννίδης και Γιώργος Λεπούρας, *Σημειώσεις Επικοινωνίας Ανθρώπου Μηχανής*. Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών, Τμήμα Πληροφορικής και Τηλεπικοινωνιών, 2005.

[14] K. Perlin and J. Meyer, "Nested user interface components," in *UIST '99: Proceedings of the 12th annual ACM symposium on User interface software and technology*. New York, NY, USA: ACM, 1999, pp. 11–18.

[15] E. L. Hutchins, J. D. Hollan, and D. A. Norman, *Direct manipulation interfaces (excerpt)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987.

[16] R. R. Swick and M. S. Ackerman, "The x toolkit: Mor bricks for building user-interfaces, or, widgets for hire," 1987.

[17] D. L. Parnas, "On the use of transition diagrams in the design of a user interface for an interactive computer system," in *Proceedings of the 1969 24th national conference*. New York, NY, USA: ACM, 1969, pp. 379–385.

[18] R. Bastide and P. A. Palanque, "A petri net based environment for the design of event-driven interfaces," in *Proceedings of the 16th International Conference on Application and Theory of Petri Nets*. London, UK: Springer-Verlag, 1995, pp. 66–83.

[19] A. O. Ramirez, "Three-tier architecture," *Linux J.*, p. 7, 2000.

[20] A. Aarsten and D. Brugali, "Client (ui) patterns for three-tier client/server applications."

[21] T. M. H. Reenskaug, "Models - views - controllers," Xerox PARC technical, 1979.

[22] G. E. Krasner and S. T. Pope, "A cookbook for using the model-view controller user interface paradigm in smalltalk-80," *J. Object Oriented Program.*, vol. 1, no. 3, pp. 26–49, 1988.

[23] "Backbase Developer Network - Client Framework Documentation," http://bdn.backbase.com/client/documentation Χρόνος προσπέλασης: 11 Σεπτεμβρίου 2008 ώρα 10:11.

[24] "An overview of MXML: The Flex markup language," http://www.adobe.com/devnet/flex/articles/paradigm.html Χρόνος προσπέλασης: 10 Σεπτεμβρίου 2008 ώρα 23:08.

[25] "OpenLaszlo Application Developer's Guide," http://www.openlaszlo.org/lps4.1/docs/developers Χρόνος προσπέλασης: 11 Σεπτεμβρίου 2008 ώρα 10:36.

[26] "Glade - a User Interface Designer for GTK+ and GNOME - Documentation," http://glade.gnome.org/documentation.html Χρόνος προσπέλασης: 10 Σεπτεμβρίου 2008 ώρα 22:46.

[27] "Libglade - Graphical Interface Description Loader API," http://library.gnome.org/devel/libglade/unstable Χρόνος προσπέλασης: 10 Σεπτεμβρίου 2008 ώρα 22:19.

[28] ".NET Framework Developer Center, XAML Overview," http://msdn.microsoft.com/en-us/library/ms752059.aspx Χρόνος προσπέλασης: 10 Σεπτεμβρίου 2008 ώρα 20:25.

[29] F. Bolton, *Pure CORBA*. Sams Publishing, 2002.

[30] *Common Object Request Broker Architecture (CORBA) Specification v.3.1, Part 1: CORBA Interfaces*, OMG, 140 Kendrick Street, Building A Suite 300, Needham, MA 02494, U.S.A., January 2008.

[31] *Common Object Request Broker Architecture (CORBA) Specification v.3.1, Part 2: CORBA Interoperability*, OMG, 140 Kendrick Street, Building A Suite 300, Needham, MA 02494, U.S.A., January 2008.