Ατομική Διπλωματική Εργασία

ΑΥΤΟΜΑΤΟΣ ΕΛΕΓΧΟΣ ΓΡΑΦΙΚΗΣ ΔΙΑΠΡΟΣΩΠΙΑΣ ΣΥΣΤΗΜΑΤΩΝ ΛΟΓΙΣΜΙΚΟΥ

Χάρης Παναγή

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Δεκέμβριος 2009

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Αυτόματος έλεγχος γραφικής διαπροσωπίας συστημάτων λογισμικού

Χάρης Παναγή

Επιβλέπων Καθηγητής
Ανδρέας Ανδρέου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Δεκέμβριος 2009

Ευχαριστίες

Θέλω να ευχαριστήσω αρχικά τον επιβλέποντα καθηγητή μου κ. Ανδρέα Ανδρέου για την υποστήριξη και καθοδήγηση που μου προσέφερε κατά την διάρκεια της εκπόνησης της παρούσας Διπλωματικής Εργασίας.

Ακόμη θέλω να εκφράσω τις ευχαριστίες μου στον κ. Αναστάση Σοφοκλέους, ο οποίος κατά την διάρκεια της εκπόνησης της Διπλωματικής αυτής Εργασίας βρισκόταν παρών από την αρχή μέχρι το τέλος και μέσω της διδασκαλίας του, των εποικοδομητικών συζητήσεων, των συμβουλών αλλά και της καθοδήγησης του έκανε δυνατή την ολοκλήρωση αυτής της εργασίας.

Ευχαριστίες πρέπει να δοθούν στον κ. Κωνσταντίνο Παττίχη, ο οποίος ευγενικά παραχώρησε μέρος από τον χρόνο του για να αποτελέσει τον δεύτερο εξεταστή της παρούσας εργασίας.

Θέλω ακόμη να ευχαριστήσω θερμά τους φίλους μου Κωνσταντίνο Χριστοφή, Ανδρέα Ιακώβου, Γρηγόρη Γιαννάκη, Παναγιώτη Παναγιώτου, Παναγιώτα Παντελή και Γιώτα Χαπούπη για όλη την βοήθεια και συμπαράσταση.

Τέλος θέλω να ευχαριστήσω την οικογένεια μου για όλη της την υποστήριξη.

Περίληψη

Ο έλεγχος λογισμικών συστημάτων αποτελεί μια από τις πιο ακριβές σε χρόνο και κόστος διαδικασίες στην ανάπτυξη λογισμικού και σύμφωνα με κάποιες εκτιμήσεις το κόστος αυτό φτάνει το 50 – 70% του συνολικού κόστους παραγωγής του λογισμικού προϊόντος.

Για να μειωθούν λοιπόν αυτά τα κόστη αλλά και για να αυξηθεί η αξιοπιστία των λογισμικών προϊόντων, διάφοροι ερευνητές και εταιρείες προσπαθούν να αυτοματοποιήσουν την διαδικασία ελέγχου λογισμικών συστημάτων.

Δημιουργήθηκαν έτσι κάποιες τεχνικές για έλεγχο του γραφικού περιβάλλοντος των λογισμικών συστημάτων. Σε αυτή την διπλωματική εργασία παρουσιάζεται μια νέα προσέγγιση στο θέμα. Η προσέγγιση αυτή δέχεται ως είσοδο τις προδιαγραφές του υπό έλεγχο λογισμικού συστήματος γραμμένες σε γλώσσα μοντελοποίησης Spec#® ως το σύνολο των προδιαγραφών όλων των στοιχείων του γραφικού περιβάλλοντος του λογισμικού και με βάση αυτές, το σύστημα θα μπορεί να παράγει αυτόματα και τυχαία ένα σύνολο από σενάρια ελέγχου που αφορούν τα στοιχεία του λογισμικού συστήματος ξεχωριστά, αλλά και στο σύνολο τους αφού η συμπεριφορά, η κατάσταση και η σειρά αλληλεπίδρασης επηρεάζει το τελικό αποτέλεσμα και την έξοδο του προγράμματος. Στην συνέχεια χρησιμοποιούμε βιβλιοθήκες από το έτοιμο σύστημα RANOREX® για την αυτόματη εκτέλεση των σεναρίων ελέγχου και για να πάρουμε τα τελικά αποτελέσματα του ελέγχου. Τα αποτελέσματα που παίρνουμε από την εφαρμογή χρησιμεύουν για ανίχνευση λογικών λαθών στο υπό έλεγχο λογισμικό σύστημα, αλλά και για έλεγχο λειτουργιών που δούλευαν σωστά και θέλουμε να ελεγχθούν μετά την προσθήκη νέων λειτουργιών στο υπό έλεγχο λογισμικό σύστημα.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Ορισμός προβλήματος.

Η ευρεία χρήση γραφικού περιβάλλοντος στα σύγχρονα λογισμικά συστήματα έχει προκαλέσει την δημιουργία ακόμη πιο πολύπλοκων γραφικών περιβάλλοντων. Η αύξηση αυτής της πολυπλοκότητας προκάλεσε προβλήματα στον έλεγχο της ορθότητας αυτών των λογισμικών συστημάτων, αφού όσο διευκολύνουν την αλληλεπίδραση του χρήστη, τόσο δυσκολεύουν στην διαδικασία παραγωγής τους τον προγραμματιστή. Έτσι, δεδομένου της σημαντικής θέσης που έχει το γραφικό περιβάλλον στα σύγχρονα λογισμικά συστήματα, ο έλεγχος της ορθότητάς του επηρεάζει την ευρωστία, την ασφάλεια και τη δυνατότητα χρησιμοποίησης του.

Ο έλεγχος αυτών των συστημάτων γραφικού περιβάλλοντος είναι η διαδικασία με την οποία ένα πρόγραμμα υπόκειται σε έλεγχο κατά πόσο τηρεί τις καταγραμμένες προδιαγραφές που τέθηκαν κατά το δεύτερο στάδιο κύκλου ζωής λογισμικού, όπως αυτό ορίζεται από την Τεχνολογία Λογισμικού. Για να γίνει ο έλεγχος αυτός, απαιτείται τεράστια προσπάθεια από την πλευρά του προγραμματιστή σε χρόνο, κόπο και χρήμα για να να δημιουργήσει τέτοια σενάρια ελέγχου ώστε να είναι σίγουρος ότι το λογισμικό δουλεύει σωστά.

όμως, ο έλεγχος της ορθότητας ενός συστήματος με γραφικό περιβάλλον είναι δύσκολος για διάφορους λόγους: Καταρχήν ο χώρος των δυνατών αλληλεπιδράσεων σε ένα γραφικό περιβάλλον είναι τεράστιος. Κάθε διαφορετική ακολουθία αλληλεπίδρασης με το πρόγραμμα θα προκαλέσει το σύστημα λογισμικού να μεταβεί σε διαφορετική κατάσταση και στη τελική θα δημιουργήσει διαφορετικό αποτέλεσμα.

Με βάση αυτό, θα πρέπει ο προγραμματιστής να δημιουργήσει τέτοια σενάρια ελέγχου, ώστε να καλύψει όλες τις δυνατές περιπτώσεις. Τα σενάρια αυτά, δημιουργούνταν στο χέρι και στη συνέχεια ο προγραμματιστής θα έπρεπε να τα εκτελέσει και πάλι χειροκίνητα και να γράψει κάτω τα αποτελέσματα.

Αυτό όμως δημιουργεί νέο πρόβλημα. Ο προγραμματιστής ή ο οποιοσδήποτε άλλος δημιουργός αυτών των σεναρίων, λόγω της εμπειρίας και της γνώσης που κατέχει, είναι δυνατόν να αγνοήσει περιπτώσεις – σενάρια τα οποία γνωρίζει πως δεν έχουν κάποια λογική συνοχή και που ο εξειδικευμένος χρήστης του υπό έλεγχο λογισμικού συστήματος δεν θα χρησιμοποιήσει. Αυτά τα σενάρια όμως ενδέχεται να περιέχουν λάθη και κάποιος ανειδίκευτος χρήστης πιθανόν να τα εκτελέσει. Σε αυτή την περίπτωση, το σύστημα λογισμικού θα κάνει παραγωγή λάθος αποτελέσματος κατά την εκτέλεσή του. Για το λόγο αυτό, επιβάλλεται η ύπαρξη κάποιου συστήματος αυτόματου ελέγχου λογισμικού συστήματος με γραφικό περιβάλλον στο οποίο θα πρέπει να παράγονται εντελώς τυχαία σενάρια ελέγχου χωρίς να υπάρχει εκ των προτέρων καμία γνώση για το είδος και την λειτουργία του υπό έλεγχο λογισμικού συστήματος.

1.2 Σκοπός της διπλωματικής εργασίας

Ο στόχος αυτής της διπλωματικής εργασίας στην επίλυση του πιο πάνω αναφερόμενου προβλήματος είναι πως προτείνει μια νέα προσέγγιση στον αυτόματο έλεγχο λογισμικού συστήματος με γραφικό περιβάλλον με αυτόματη παραγωγή και εκτέλεση τυχαίων σεναρίων ελέγχου.

Έχει δημιουργηθεί μια εφαρμογή, το Automatic Gui Tester (AutoGT) το οποίο αυτοματοποιεί ολόκληρη την διαδικασία του έλεγχου λογισμικών συστημάτων. Δέχεται ως είσοδο το υπό έλεγχο λογισμικό σύστημα, εξαγάγει όλα τα στοιχεία του γραφικού του περιβάλλοντος και τα παρουσιάζει. Στην συνέχεια, δέχεται ως δεύτερη είσοδο τις προδιαγραφές και με βάση αυτές, παράγει γενικά ή ειδικά, ανάλογα με το πόσο συγκεκριμένα θέλει να κάνει έλεγχο ο χρήστης, τα σενάρια ελέγχου. Τέλος εκτελεί τα σενάρια ελέγχου αυτόματα, και παρουσιάζει τα αποτελέσματα της εκτέλεσης στον χρήστη.

Κεφάλαιο 2

Έλεγχοι λογισμικού

2.1 Έλεγχος λογισμικού

Ο έλεγχος λογισμικού είναι η διαδικασία επικύρωσης και επαλήθευσης του λογισμικού συστήματος που πραγματοποιείται πάνω σε ένα ολοκληρωμένο διαδραστικό σύστημα α) για να αξιολογήσει κατά πόσο οι λειτουργίες που εκτελεί είναι ταυτόσημες και ικανοποιούν τις επιχειρηματικές και τεχνικές προδιαγραφές που τέθηκαν, κατά την διάρκεια του σταδίου ορισμού των προδιαγραφών, που αποτελούν νομικό έγγραφο και καθοδηγούν ολόκληρη την διαδικασία σχεδιασμού και υλοποίησης και β) ότι το υπό έλεγχο λογισμικό σύστημα εκτελεί τις λειτουργίες του όπως αναμενόταν. Η ικανοποίηση των προδιαγραφών θα καθορίσει το ποσοστό αξιοπιστίας του υπό έλεγχο λογισμικού συστήματος καθώς και της εταιρείας που ανέλαβε την υλοποίηση. Αυτοί οι έλεγχοι συνήθως πραγματοποιούνται από τρίτα άτομα για

να διαπιστώσουν οι χρήστες και οι ιδιοκτήτες της εταιρείας για την οποία παράγεται το λογισμικό σύστημα αν το προϊόν που παράγγειλαν είναι αυτό που θα τους παραδοθεί.

Σήμερα υπάρχουν αρκετοί και διάφοροι τρόποι ελέγχου λογισμικού και χωρίζονται σε τρεις βασικές κατηγορίες: το black box, το white box και το grey box testing.

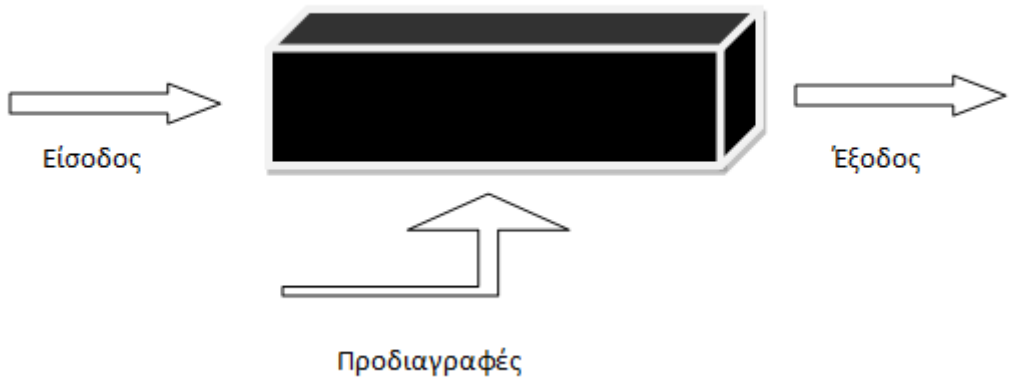
2.2. Είδη ελέγχου λογισμικών συστημάτων

2.2.1 Black box testing

Ο όρος black box [1,2] στην Τεχνολογία Λογισμικού είναι ένας τεχνικός όρος και αναφέρεται στο έλεγχο λογισμικού συστήματος χωρίς να εξετάζεται ο κώδικας του υπό έλεγχο λογισμικού συστήματος. Αυτός ο τρόπος ελέγχου χειρίζεται το υπό έλεγχο λογισμικό σύστημα ως ένα «μαύρο κουτί», ελέγχοντας μέσω της διεπαφής του λογισμικού συστήματος την έξοδό του για συγκεκριμένες εισόδους. Το άτομο που πραγματοποιεί αυτού του είδους τον έλεγχο, δημιουργεί τέτοια σενάρια ελέγχου τα οποία εξαντλητικά θα πρέπει να ελέγξουν όλες τις λειτουργίες που αναγράφονται στις προδιαγραφές του υπό έλεγχο λογισμικού συστήματος. Έτσι αυτού του είδους ο έλεγχος προσπαθεί να ανιχνεύσει λάθη σχετικά με α) λανθασμένες ή ανύπαρκτες λειτουργίες, β) λάθη στην διεπαφή του λογισμικού συστήματος, γ) λάθη στις δομές δεδομένων ή την επικοινωνία με βάσεις δεδομένων, δ) λάθη συμπεριφοράς ή απόδοσης του λογισμικού συστήματος και ε) λάθη στην αρχικοποίηση και/ ή τερματισμό του λογισμικού συστήματος.

Αυτού του είδους ο έλεγχος χρησιμοποιεί τις προδιαγραφές (Σχ.2.1) με βάση τις οποίες δημιουργήθηκε το λογισμικό σύστημα για να ελέγξει ότι πληρούνται. Για αυτό τον λόγο, τα σενάρια ελέγχου πρέπει να είναι ενδελεχή, ούτως ώστε το άτομο που θα αξιολογήσει τα αποτελέσματα του ελέγχου να μπορεί εύκολα να επαληθεύσει ότι η έξοδος, το αποτέλεσμα δηλαδή του λογισμικού συστήματος είναι το ίδιο με αυτό που δόθηκε στο σενάριο ελέγχου.

Το πρόβλημα με το black box testing είναι πως κανείς δεν μπορεί με σιγουριά να είναι βέβαιος πως εξετάστηκαν όλα τα δυνατά μονοπάτια του υπό έλεγχο λογισμικού συστήματος.



Σχήμα 2.1 Black box testing.

2.2.2 White box testing

Σε αντίθεση με το black box testing, το white box testing [3] είναι ο έλεγχος που πραγματοποιείται όταν ο προγραμματιστής έχει πρόσβαση στις εσωτερικές δομές δεδομένων, τον αλγόριθμο και τον κώδικα που τα υλοποιεί. Η μέθοδος αυτή βασίζεται στις γνώσεις και ικανότητα του προγραμματιστή να αναγνωρίσει όλα τα πιθανά μονοπάτια στον κώδικα ούτως ώστε να δημιουργήσει τέτοια σενάρια ελέγχου που θα καλύψουν εντελώς τον κώδικα του υπό έλεγχο λογισμικού συστήματος.

Με χρήση των τεχνικών ελέγχου white box ο προγραμματιστής μπορεί να δημιουργήσει σενάρια ελέγχου που α) εκτελούν και ελέγχουν ανεξάρτητα μονοπάτια ή κομμάτια κώδικα, β) εκτελούν και ελέγχουν τις δομές ελέγχου και τον κώδικα που περιέχεται σε αυτές, τόσο για την αληθή όσο και για την ψευδή περίπτωση (control flow - branch testing technique), γ) εκτελούν και ελέγχουν τις δομές επανάληψης και τον κώδικα που περιέχεται σε αυτές καθώς και τις συνθήκες εισόδου και εξόδου από αυτές, δ) ελέγχουν τις εσωτερικές δομές δεδομένων για διατήρηση ορθών δεδομένων (data flow testing technique).

Το πρόβλημα με αυτού του είδους τον έλεγχο βρίσκεται στο γεγονός πως εάν γίνουν κάποιες αλλαγές στον κώδικα του λογισμικού συστήματος, τότε τα σενάρια ελέγχου που έχουν δημιουργηθεί είναι άχρηστα. Εάν όχι, τότε σίγουρα θα χρειάζονται σημαντικές αλλαγές για να μπορέσουν να επαναχρησιμοποιηθούν. Ακόμη, ενώ μπορεί να καλύψει όλα τα πιθανά μονοπάτια του υπό έλεγχο λογισμικού συστήματος, αδυνατεί να ανιχνεύσει μη υλοποιημένες προδιαγραφές.

2.2.3 Grey box testing

Πρόκειται [4] για τον συνδυασμό των πιο πάνω μεθόδων ελέγχου, του black box και του white box testing. Το άτομο που διενεργεί τον έλεγχο έχει γνώση του κώδικα του υπό έλεγχο λογισμικού συστήματος και κατά την δημιουργία σεναρίων ελέγχου, αφιερώνει στον κώδικα ένα περιορισμένο αριθμό σεναρίων ελέγχου για να πραγματοποιήσει τον έλεγχο. Στα υπόλοιπα σενάρια ελέγχου ο προγραμματιστής χρησιμοποιεί μια black box προσέγγιση, με σενάρια ελέγχου τα οποία, δεδομένου κάποιων εισόδων, ελέγχουν το αποτέλεσμα ή έξοδο του λογισμικού συστήματος.

Όπως και στο black box testing, η προσέγγιση που χρησιμοποιείται βασίζεται στη διεπαφή ή γραφικό περιβάλλον του λογισμικού συστήματος με μόνη διαφορά ότι το άτομο που διενεργεί τον έλεγχο γνωρίζει τον κώδικα ή μέρος του κώδικα του λογισμικού συστήματος. Γι' αυτό τον λόγο παράγονται καλύτερα σενάρια ελέγχου που δίνουν καλύτερα αποτελέσματα από τις προηγούμενες δύο μεθόδους ελέγχου αφού υπάρχει περισσότερη γνώση σε ότι αφορά το υπό έλεγχο λογισμικό σύστημα.

Ο βασικός σκοπός χρήσης του grey box testing είναι η επαναχρησιμοποίηση των σεναρίων ελέγχου αφού γίνουν αλλαγές και / ή προσθήκες στον κώδικα του λογισμικού συστήματος και η ανίχνευση λαθών που σχετίζονται με κακό σχεδιασμό και κακή υλοποίηση του υπό έλεγχο συστήματος λογισμικού.

2.3 Μέθοδος ελέγχου capture / playback

Στην αγορά υπάρχουν αρκετές εφαρμογές λογισμικού των οποίων σκοπός είναι ο έλεγχος του γραφικού περιβάλλοντος κάποιου λογισμικού. Μια από τις τεχνικές που χρησιμοποιείται ευρέως σήμερα σε διάφορα τέτοια λογισμικά συστήματα είναι η τεχνική Capture / Playback [5].

Με μια απλή περιδιάβαση σε ένα τέτοιο λογισμικό σύστημα ελέγχου παρατήρησα πως η τεχνική αυτή χρησιμοποιεί ένα εργαλείο που ηχογραφεί χειρονακτικές αλληλεπιδράσεις του χρήστη με το υπό έλεγχο λογισμικό σύστημα. Κατά την διάρκεια αυτών των αλληλεπιδράσεων το σύστημα αυτό ζητούσε συνεχώς στοιχεία επαλήθευσης των κινήσεων αυτών. Μετά το πέρας της ηχογράφησης των αλληλεπιδράσεων με το υπό έλεγχο λογισμικό σύστημα το σύστημα δίνει την επιλογή να γίνει εκτέλεση των ηχογραφημένων

αλληλεπιδράσεων. Αν το υπό έλεγχο λογισμικό σύστημα αντιδράσει διαφορετικά σε κάποια από τις εκτελέσεις, τότε το σύστημα που κάνει τον έλεγχο παρουσιάζει την αντίδραση σαν ελάττωμα στο υπό έλεγχο λογισμικό σύστημα. Οι κατασκευαστές αυτών των συστημάτων υποστηρίζουν πως αυτοματοποιούν έτσι την διαδικασία ελέγχου. Αυτή η διατύπωση δεν ευσταθεί για τον απλό λόγο ότι τα σενάρια ελέγχου σε αυτά τα συστήματα γίνονται χειρωνακτικά με πολύ κόπο, σκέψη και προσπάθεια από πλευράς των χρηστών τέτοιων προγραμμάτων και ουσιαστικά αυτοί είναι που κάνουν όλη την δουλειά.

2.4 Έτοιμα συστήματα ελέγχου γραφικού περιβάλλοντος λογισμικών συστημάτων

Σε αυτό το σημείο θα παρουσιάσουμε κάποια ολοκληρωμένα συστήματα ελέγχου γραφικού περιβάλλοντος λογισμικών συστημάτων.

2.4.1. CompuWare TestPartner

Τα σύστημα αυτό [6] παρέχει λειτουργίες για έλεγχο στοιχείων του γραφικού περιβάλλοντος λογισμικών συστημάτων μέσω test scripts. Τα test scripts αυτά μπορούν να γραφούν σε γλώσσα προγραμματισμού Visual Basic εάν ο χρήστης του συστήματος αυτού είναι έμπειρος προγραμματιστής. Στην περίπτωση όμως που ο χρήστης δεν είναι γνώστης προγραμματιστικών μεθόδων, το σύστημα πάλι μπορεί να χρησιμοποιηθεί μέσω του Visual Navigator, μιας λειτουργίας που παρέχεται από το σύστημα και μέσω αυτής να δημιουργήσει και να εκτελέσει τα test scripts. Παρόλα αυτά τα test scripts στην δεύτερη περίπτωση δεν θα είναι το ίδιο αποτελεσματικά.

2.4.2 IBM Rational Test Tools

Η εταιρεία Rational Software, θυγατρική εταιρεία της IBM, παρέχει συστήματα ελέγχου λογισμικών συστημάτων για διάφορα στάδια υλοποίησης. Για τον σκοπό ελέγχου του γραφικού περιβάλλοντος λογισμικών συστημάτων η εταιρεία αυτή διαθέτει δύο εφαρμογές.

2.4.2.1 Rational Robot

Το σύστημα αυτό [7] συνεργάζεται με το Visual Studio 6 της Microsoft και χρησιμοποιεί λειτουργία capture / playback για να δημιουργήσει τα test scripts. Τα test scripts δημιουργούνται μέσω μιας διαδικασίας μετάφρασης των αλληλεπιδράσεων του χρήστη με το

υπό έλεγχο λογισμικό σύστημα και καταγράφονται σε γλώσσα SQA Basic η οποία είναι παρόμοια με την Visual Basic. Στο σύστημα αυτό απουσιάζουν λειτουργίες για αποδοτική αναγνώριση και επαλήθευση των στοιχείων του γραφικού περιβάλλοντος.

2.4.2.2 Rational Visual Test

Το σύστημα αυτό [8] αρχικά δημιουργήθηκε από την Microsoft και στην συνέχεια αγοράστηκε από την Rational Software. Παρέχει στους χρήστες ένα API (Application Programming Interface) μέσω του οποίου οι χρήστες μπορούν να δημιουργήσουν εφαρμογές ελέγχου του γραφικού περιβάλλοντος άλλων λογισμικών συστημάτων στο χέρι, ή με χρήση λειτουργίας capture / playback η οποία καταγράφει το test script σε γλώσσα Test Basic. Οι εφαρμογές που δημιουργούνται χρειάζονται συνεχή επίβλεψη από τους χρήστες, ώστε να τους παρέχουν δεδομένα ελέγχου.

2.4.3 Mercury Interactive Tools

Η εταιρεία Mercury Interactive διαθέτει διάφορα εργαλεία και εφαρμογές για έλεγχο τόσο εφαρμογών λογισμικού τόσο και διαδικτυακών εφαρμογών. Οι πιο διαδεδομένες από αυτές περιγράφονται πιο κάτω.

2.4.3.1 WinRunner

Το σύστημα αυτό [9] μπορεί να χρησιμοποιηθεί για έλεγχο σε λογισμικά συστήματα προορισμένα για Windows αλλά και σε διαδικτυακές εφαρμογές. Η χρήση του βασίζεται και πάλι σε λειτουργίες capture / playback. Οι χρήστες μπορούν να προσθέσουν ή να αφαιρέσουν στοιχεία του γραφικού περιβάλλοντος του υπό έλεγχο λογισμικού συστήματος μέσω κάποιου είδους χαρτογράφησης του γραφικού περιβάλλοντος, μπορούν να αλληλεπιδράσουν με τις φόρμες του συστήματος και να προσθέσουν δεδομένα ελέγχου. Τα δεδομένα αυτά όμως πρέπει να δημιουργηθούν χειροκίνητα από τους χρήστες του συστήματος. Ακόμη κάποιες από τις λειτουργίες του είναι ειδικές και δεν μπορούν να γενικευτούν για χρήση σε οποιαδήποτε εφαρμογή.

2.4.3.2 LoadRunner

Το σύστημα αυτό [10] χρησιμοποιεί test scripts που δημιουργούνται από άλλες εφαρμογές της Mercury Interactive για εκτέλεση ελέγχου. Στη συνέχεια το σύστημα εκτελεί load testing για να ελέγξει την απόδοση του συστήματος.

2.4.4 Abbot

Το σύστημα Abbot [11] χρησιμοποιείται σε εφαρμογές Java. Χρησιμοποιεί λειτουργίες για να αναπαραστήσει αλληλεπιδράσεις χρηστών με το υπό έλεγχο λογισμικό σύστημα. Ο χρήστης και σε αυτό το σύστημα πρέπει να γράψει στο χέρι τα test scripts με χρήση γλώσσας Java.

2.4.5 Guitar

Αυτό το σύστημα [12] αποτελεί μια ολοκληρωμένη λύση για έλεγχο λογισμικών συστημάτων. Περιλαμβάνει λειτουργία παραγωγής σεναρίων ελέγχου και λειτουργία αναπαραγωγής για να εκτελέσει τον έλεγχο στην υπό έλεγχο εφαρμογή.

Κεφάλαιο 3

Ανάλυση προβλήματος – Αρχές σχεδιασμού

3.1 Ανάλυση

Αρχικά, για να μπορέσω να αρχίσω την υλοποίηση έπρεπε να καταγράψω και να οργανώσω τις διάφορες διαδικασίες και στάδια του συστήματος, ούτως ώστε να υπάρχει καταγραμμένη και αναλυμένη ολόκληρη η εικόνα του συστήματος. Αυτό εξυπηρετεί 2 (δύο) σκοπούς. (α) για ευκολία, αφού από την στιγμή που θα είναι καταγραμμένα θα γνωρίζω ακριβώς τι θα πρέπει να κάνω, βήμα προς βήμα και (β) για καλύτερη οργάνωση της όλης διαδικασίας του προγραμματισμού.

Έτσι, βήμα προς βήμα τα στάδια μιας αυτοματοποιημένης διαδικασίας ελέγχου λογισμικού είναι τα ακόλουθα:

1. Ο προγραμματιστής θα αποφασίσει ποιο λογισμικό σύστημα θα υποστεί τον έλεγχο. Αυτό σημαίνει ότι ο προγραμματιστής θα πρέπει να έχει στην κατοχή του το εκτελέσιμο αρχείο του υπό έλεγχο λογισμικού συστήματος.

2. Το σύστημα θα πρέπει να μπορεί να αναλύσει δυναμικά την δομή του υπό έλεγχου λογισμικού συστήματος και θα πρέπει να μπορεί να ανιχνεύει όλα τα στοιχεία που υπάρχουν στο γραφικό περιβάλλον του υπό έλεγχο λογισμικού συστήματος. Συγκεκριμένα θα πρέπει να ανιχνεύει στο υπό έλεγχο λογισμικό σύστημα όλες τις λεπτομέρειες για τα στοιχεία του γραφικού του περιβάλλοντος. Τέτοια στοιχεία είναι το είδος του στοιχείου, δηλαδή αν είναι button, radiobutton, checkbox, textbox κτλ, το control id ή control name του κάθε χειριστηρίου δηλαδή ένας μοναδικός αριθμός ή λέξη που το χαρακτηρίζει μοναδικά και μας επιτρέπει να το χειριζόμαστε, το όνομά του, η τιμή του κτλ. Αυτά τα δεδομένα που θα εξαχθούν από το γραφικό περιβάλλον

του υπό έλεγχο λογισμικού συστήματος θα πρέπει να αποθηκευτούν σε μορφή η οποία είναι κατανοητή και ευκολοδιάβαστη. Μια τέτοια μορφή αποθήκευσης θα μπορούσε να είναι απλό κείμενο (text) ή XML. Ένα XML αρχείο [15] που σημαίνει extensible markup language αρχείο είναι ένα αρχείο το οποίο κωδικοποιείται με βάση ένα σύνολο κανόνων ή προδιαγραφές που καθορίζονται από το XML Specification 1.0, το οποίο δημιουργήθηκε από την οργάνωση W3C.

3. Το σύστημα θα πρέπει μέσα από την αποθηκευμένη μορφή των στοιχείων του γραφικού περιβάλλοντος του υπό έλεγχο λογισμικού συστήματος να μπορεί να εξαγάγει τα δεδομένα, δηλαδή τα στοιχεία του γραφικού περιβάλλοντος και να τα παρουσιάσει στον χρήστη. Με αυτό τον τρόπο ο προγραμματιστής θα μπορεί να έχει μια εικόνα με όλα τα δεδομένα που αποτελούν το γραφικό περιβάλλον του υπό έλεγχο λογισμικού συστήματος.

4. Το σύστημα θα πρέπει να δέχεται ως είσοδο τις προδιαγραφές με τις οποίες το σύστημα δημιουργήθηκε. Αφού πρόκειται για black box testing τότε η είσοδος προδιαγραφών είναι βασική προϋπόθεση για τον σωστό έλεγχο κάποιου λογισμικού συστήματος. Οι προδιαγραφές μπορούν να καταγραφούν σε διάφορες μορφές. Μια τέτοια μορφή θα μπορούσε να είναι η UML ή Unified Modeling Language. Η UML είναι μια πολύ διαδεδομένη, τυποποιημένη και γενικής χρήσης γλώσσα μοντελοποίησης η οποία χρησιμοποιείται ευρέως στην Τεχνολογία Λογισμικού. Η γλώσσα αυτή περιλαμβάνει ένα σύνολο από τεχνικές γραφικής σημειογραφίας για την δημιουργία οπτικών μοντέλων συστημάτων λογισμικού. Μια άλλη μορφή θα μπορούσε να ήταν η AsmL. AsmL αποτελεί συντομογραφία του Abstract State Machine Language. Η γλώσσα αυτή είναι μια γλώσσα προγραμματισμού και μοντελοποίησης που αναπτύχθηκε από την Microsoft Research και χρησιμοποιήθηκε αρχικά για επιστημονική έρευνα. Η γλώσσα αυτή επεκτάθηκε και δημιουργήθηκε τελικά η Spec# ή Spec Sharp μια γλώσσα προγραμματισμού και μοντελοποίησης προδιαγραφών που αποτελεί επέκταση των δυνατοτήτων της πολύ διαδεδομένης γλώσσας προγραμματισμού C#. Αυτή η γλώσσα μοντελοποίησης δημιουργήθηκε ως μια πιο οικονομικά αποδεκτή και αποδοτική προσπάθεια για την ανάπτυξη και διατήρηση υψηλής ποιότητας λογισμικών συστημάτων.

5. Το σύστημα που θα δημιουργηθεί θα πρέπει να χρησιμοποιεί τις προδιαγραφές, μέσω της γλώσσας μοντελοποίησης και να εμφανίζει όλα τα στοιχεία του γραφικού περιβάλλοντος του υπό έλεγχο λογισμικού σε κάποιου είδους πίνακα ώστε μπορεί ο

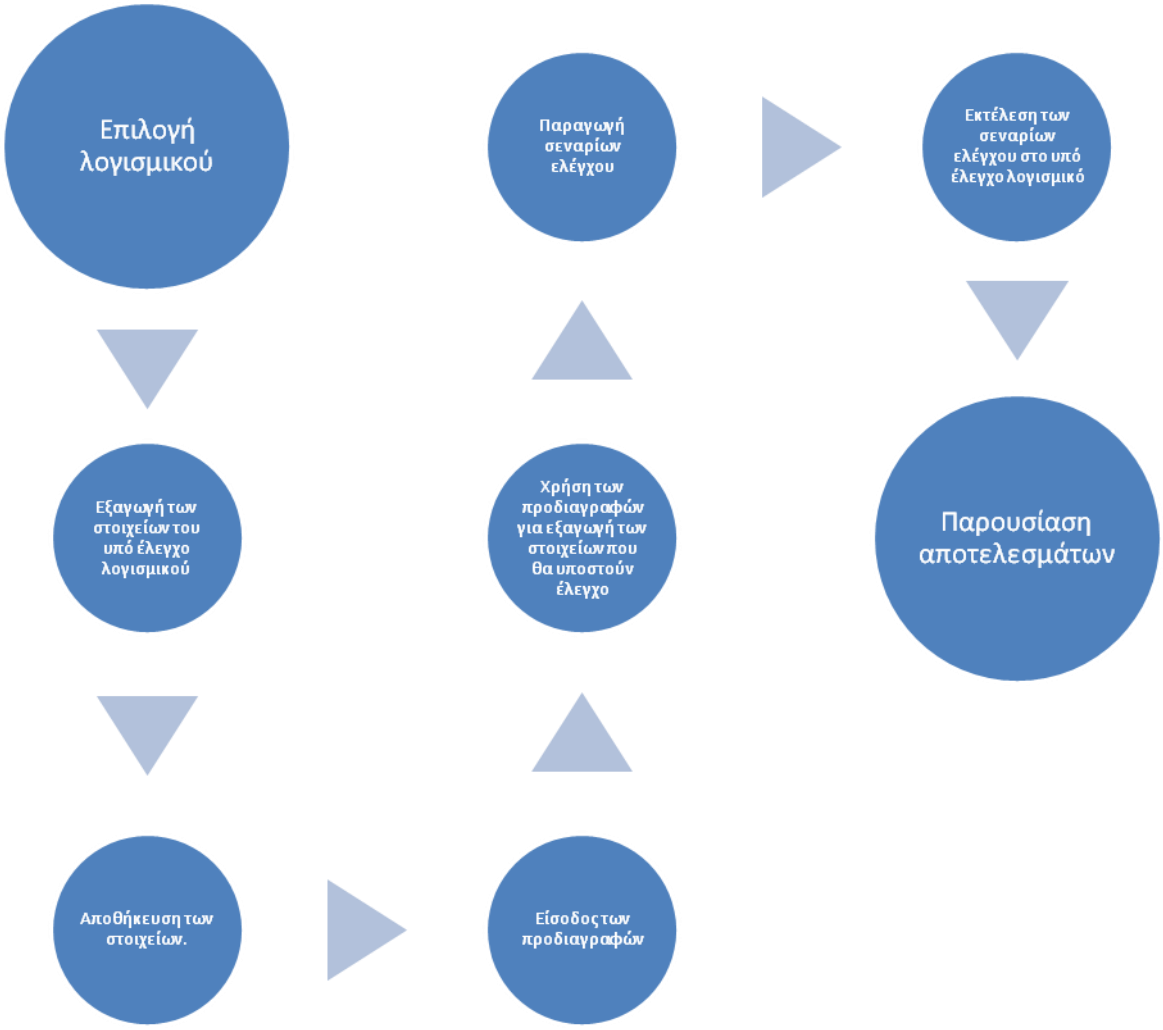
προγραμματιστής να επιλέξει τα στοιχεία αυτά τα οποία θέλει να ελέγξει, σε περίπτωση που θέλει να ελέγξει ένα μεμονωμένο και συγκεκριμένο υποσύνολο των στοιχείων που αποτελούν το γραφικό περιβάλλον του υπό έλεγχο λογισμικού συστήματος.

6. Το σύστημα στην συνέχεια θα πρέπει να μπορεί, με βάση τις επιλογές που έκανε ο προγραμματιστής για το σύνολο ή υποσύνολο των στοιχείων του υπό έλεγχο λογισμικού συστήματος στο οποίο θα γίνει ο έλεγχος, να παράγει αυτόματα σενάρια ελέγχου οποιουδήποτε μήκους. Έτσι ο χρήστης δεν θα χρειάζεται να καταναλώνει χρόνο για την καταγραφή σεναρίων ελέγχου, αφού το σύστημα θα το κάνει γι' αυτόν.

7. Το σύστημα θα πρέπει να μπορεί να διαβάζει τα σενάρια ελέγχου, τα οποία θα είναι αποθηκευμένα σε κάποιο αρχείο κειμένου (text file) και με τέτοια τυποποιημένη μορφή, ούτως ώστε να μπορεί το σύστημα να μπορεί να τα διαβάζει. Στην συνέχεια, καθώς τα διαβάζει θα πρέπει να μπορεί να εκτελεί στο υπό έλεγχο λογισμικό σύστημα όλες τις εντολές που δίδονται από αυτά τα σενάρια.

8. Το σύστημα, καθώς εκτελεί τα σενάρια ελέγχου στο υπό έλεγχο λογισμικό σύστημα, θα πρέπει να αποθηκεύει σε αρχείο όλες τις πληροφορίες και αποτελέσματα σχετικά με την κάθε λειτουργία που εκτελεί. Αξίζει να σημειωθεί ότι τα αποτελέσματα της εκτέλεσης θα πρέπει να συγκρίνονται αυτόματα με τα αποτελέσματα του σεναρίου ελέγχου ούτως ώστε να μπορούν να παρουσιάσουν στον χρήστη οπτικά αποτελέσματα. Τα αποτελέσματα θα πρέπει να μπορούν να αποθηκεύονται σε μορφή απλού αρχείου κειμένου (simple text file), ή σε μορφή html (hypertext markup language) η οποία σύμφωνα με την βιβλιογραφία είναι μια περιγραφική γλώσσα η οποία χρησιμοποιεί μια ειδική σημειογραφία για καταγραφή του κειμένου και μπορεί να διαβαστεί από οποιονδήποτε φυλλομετρητή ιστοσελίδων. Ένας άλλος τρόπος καταγραφής των αποτελεσμάτων θα μπορούσε να είναι το XML, το οποίο περιγράψαμε αναλυτικά πιο πάνω. Για καλύτερη οπτική αναπαράσταση των αποτελεσμάτων, θα μπορούσαμε να μετατρέπαμε το XML αρχείο ώστε να χρησιμοποιεί CSS ή διαφορετικά Cascading Style Sheets. Η CSS, η οποία είναι μια γλώσσα σημειολογίας (semantics language) χρησιμοποιείται κυρίως για τον έλεγχο της εμφάνισης κάποιου εγγράφου που γράφτηκε με βάση τους κανόνες του HTML ή του XML.

9. Το σύστημα μετά την εμφάνιση των αποτελεσμάτων θα πρέπει να δίνει στον προγραμματιστή την επιλογή να ξαναεκτελέσει κάποιο άλλο σύνολο σεναρίων ελέγχου στην περίπτωση που η συγκεκριμένη εκτέλεση δεν έδειξε κάποιο λάθος.



Σχήμα 3.1 Πρώτη προσέγγιση λειτουργιών του συστήματος

3.2 Αρχές σχεδιασμού του συστήματος

Προτού αναλύσουμε με λεπτομέρεια το σχεδιασμό του συστήματος, θα ήταν καλό να ορίσουμε κάποιες αρχές με βάση τις οποίες θα γίνει ο σχεδιασμός. Αυτές οι αρχές παραθέτονται πιο κάτω:

1. Απλότητα. Ο χειρισμός του συστήματος θα πρέπει να είναι όσο το δυνατό πιο απλός. Ο χειριστής αν και θα είναι εξοικειωμένος με συστήματα πληροφορικής δεν θα πρέπει να δυσκολεύεται να χειριστεί το σύστημα αφού σκοπός μας είναι να ευκολύνουμε την όλη διαδικασία.

2. Ευρωστία. Το σύστημα πρέπει να μπορεί να συνεχίζει την λειτουργία του ακόμη και στην περίπτωση που ανίχνευσε λάθη στην υλοποίηση του υπό έλεγχο λογισμικού συστήματος και να μπορεί να εκτελέσει το σύνολο των σεναρίων ελέγχου που καθορίστηκαν για την συγκεκριμένη εκτέλεση, ούτως ώστε να εξαχθούν τα αποτελέσματα.

3. Αυτοματοποίηση. Το σύστημα θα πρέπει να αυτοματοποιεί όσο το δυνατό περισσότερες λειτουργίες στον έλεγχο λογισμικών συστημάτων ούτως ώστε να μπορούν στην συνέχεια χωρίς κόπο οι χρήστες του να παίρνουν τα αποτελέσματα της εκτέλεσης και να κάνουν τις αναγκαίες αλλαγές στο υπό έλεγχο λογισμικό σύστημα. Οι αυτοματοποιημένες λειτουργίες θα πρέπει να περιλαμβάνουν τουλάχιστον την εξαγωγή των στοιχείων του γραφικού περιβάλλοντος, την παραγωγή σεναρίων ελέγχου και την εκτέλεση των σεναρίων αυτών.

4. Απόδοση. Το σύστημα θα πρέπει να είναι αποδοτικό αφού η διαδικασία ελέγχου λογισμικών συστημάτων αποτελεί σήμερα μια από τις πιο επίπονες, χρονοβόρες και ακριβές διαδικασίες μέσα στο σύνολο των διαδικασιών παραγωγής συστημάτων λογισμικού όπως αυτές ορίζονται από την Τεχνολογία Λογισμικού. Έτσι οποιαδήποτε έλλειψη απόδοσης θα αναγκάσει τους χειριστές του να το εγκαταλείψουν.

5. Γενίκευση. Το σύστημα θα πρέπει να γενικεύει τις λειτουργίες του στο σύνολο, ή αρχικά σε ένα μεγάλο υποσύνολο των λογισμικών γραφικού περιβάλλοντος. Αυτό θα πρέπει να εξασφαλιστεί για να διασφαλιστεί η επιβίωση του συστήματος.

Σχήμα 3.2 Βασικές αρχές σχεδιασμού

Κεφάλαιο 4

Μεθοδολογία

4.1 Εισαγωγή

Η προσέγγιση του θέματος, όπως αυτή παρατίθεται σε αυτή την Διπλωματική Εργασία χωρίζεται σε επτά (7) βήματα.

4.2 Πρώτο βήμα – Αναγνώριση και εισαγωγή του λογισμικού συστήματος στο οποίο θα γίνει ο έλεγχος

Στο πρώτο αυτό βήμα, γίνεται εισαγωγή στο σύστημα που δημιουργήθηκε το υπό έλεγχο λογισμικό σύστημα. Αυτό, θα χρησιμοποιηθεί στη συνέχεια κατά την διάρκεια του ελέγχου.

4.3 Δεύτερο βήμα – Εισαγωγή του μοντέλου του γραφικού περιβάλλοντος σε μορφή XML

Αυτό το βήμα, χωρίζεται σε δύο μονοπάτια. Ακολουθώντας το πρώτο μονοπάτι, ο χρήστης θα κάνει χρήση του εργαλείου Ranorex Spy, για αποθήκευση του μοντέλου του γραφικού περιβάλλοντος του υπό έλεγχο λογισμικού συστήματος σε μορφή XML. Ακολουθώντας το δεύτερο μονοπάτι, ο χρήστης θα κάνει εισαγωγή του αποθηκευμένου μοντέλου του γραφικού περιβάλλοντος του υπό έλεγχο λογισμικού συστήματος στο δικό μας σύστημα. Ο χρήστης του συστήματος μπορεί να ακολουθήσει μόνο το δεύτερο μονοπάτι, στην περίπτωση που έχει ήδη αποθηκευμένο το μοντέλο του υπό έλεγχο λογισμικού συστήματος σε κάποια μονάδα μόνιμης αποθήκευσης. Σε περίπτωση όμως που θα ακολουθήσει το πρώτο μονοπάτι, ο χρήστης θα πρέπει υποχρεωτικά να ακολουθήσει και το δεύτερο μονοπάτι, προκειμένου να μπορέσει να προχωρήσει στο επόμενο βήμα.

4.4 Τρίτο βήμα – Παρουσίαση του μοντέλου από το σύστημα

Στο βήμα αυτό ο χρήστης μπορεί να μελετήσει τα συστατικά μέρη του γραφικού περιβάλλοντος του υπό έλεγχο λογισμικού συστήματος, έτσι ώστε να μπορέσει να κάνει τις απαραίτητες αλλαγές στα συστατικά του μοντέλου του λογισμικού συστήματος για το οποίο δημιουργήθηκαν οι προδιαγραφές, ώστε να συνάδουν με τις ονομασίες των στοιχείων του πραγματικού συστήματος λογισμικού.

4.5 Τέταρτο βήμα – Εισαγωγή των προδιαγραφών

Σε αυτό το βήμα, γίνεται εισαγωγή των προδιαγραφών σε μορφή Spec# (Spec Sharp), από αρχείο με extension .ssc. Οι προδιαγραφές θα πρέπει απαραίτητα να δημιουργηθούν εκ των προτέρων έτσι ώστε να μπορέσει να γίνει η εισαγωγή τους στο βήμα αυτό.

4.6 Πέμπτο βήμα - Επιλογή των στοιχείων του λογισμικού συστήματος στα οποία θα γίνει έλεγχος

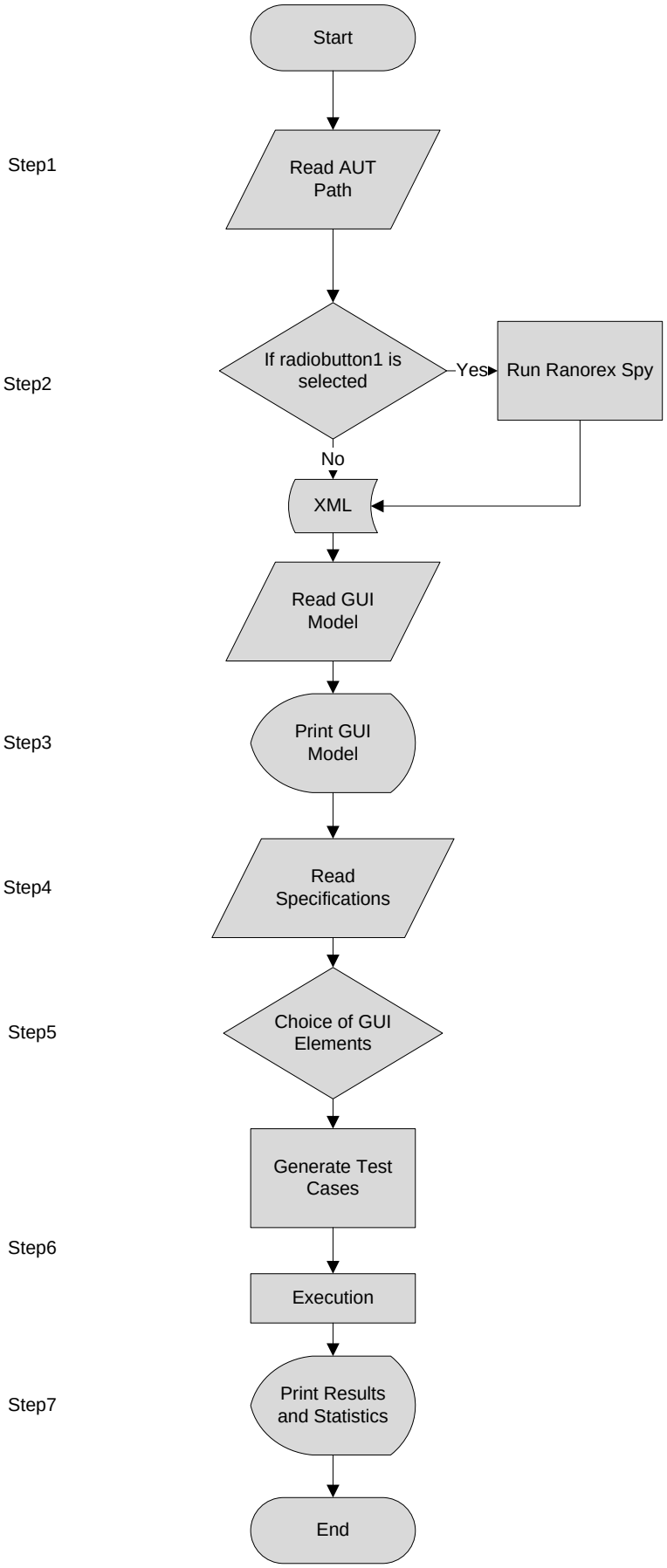
Με την εισαγωγή των προδιαγραφών στο προηγούμενο βήμα, ο χρήστης τώρα μπορεί να παρατηρήσει στην οθόνη του συστήματος, όλα τα στοιχεία τα οποία περιέχονται στις προδιαγραφές. Εδώ θα μπορεί να επιλέξει τα στοιχεία του γραφικού περιβάλλοντος του υπό έλεγχο λογισμικού συστήματος στο σύνολό τους, ή ένα μικρότερο υποσύνολο για να εκτελεστεί σε επόμενο βήμα ο έλεγχος.

4.7 Έκτο βήμα – Επιλογή παραμέτρων ελέγχου

Στο βήμα αυτό, δίνεται η δυνατότητα στο χρήστη του συστήματος να κάνει κάποιες επιλογές. Οι επιλογές αφορούν α) τον αριθμό παραγωγής και εκτέλεσης των σεναρίων ελέγχου και β) τον τρόπο παραγωγής των σεναρίων ελέγχου. Εδώ ο χρήστης μπορεί να επιλέξει εάν η παραγωγή των σεναρίων θα γίνει τυχαία και αυτόματα, ή εάν επιθυμεί να γίνει παραγωγή συγκεκριμένου σεναρίου ελέγχου. Υπάρχουν δύο λόγοι για τους οποίους δίνεται η δυνατότητα αυτή στον χρήστη. Ο πρώτος λόγος που γίνεται αυτό είναι για να μπορεί ο χρήσης να εκτελέσει σενάρια που δυνατόν να απέτυχαν στο παρελθόν και μετά από κάποιες αλλαγές στην υλοποίηση του λογισμικού συστήματος να γίνει επανέλεγχος συγκεκριμένου σεναρίου. Ο δεύτερος λόγος αφορά σενάρια που ενώ παρουσίασαν επιτυχημένη εκτέλεση στο παρελθόν, μετά από αλλαγές ή πρόσθεση λειτουργιών στην υλοποίηση του συστήματος να χρειάζεται να γίνει έλεγχος εάν όντως δεν έχει επηρεαστεί η σωστή του λειτουργία.

4.8 Έβδομο βήμα – Παρουσίαση αποτελεσμάτων – Στατιστικά που αφορούν τον έλεγχο

Μετά την ολοκλήρωση του ελέγχου του λογισμικού συστήματος, όπως αυτός έγινε στο έκτο βήμα, στο βήμα αυτό το σύστημα ελέγχου παρουσιάζει στον χρήστη σε ευανάγνωστη μορφή τα αποτελέσματα του ελέγχου. Ακόμη δίνεται η δυνατότητα στο χρήστη να μελετήσει στατιστικά στοιχεία που αφορούν την εκτέλεση ελέγχου όπως τον αριθμό των συνολικών ελέγχων που έγιναν, τον αριθμό των σωστών και λανθασμένων ελέγχων, σε ποια στοιχεία του λογισμικού συστήματος έγινε έλεγχος και πόσες φορές έγινε έλεγχος σε αυτά, καθώς και το χρόνο εκτέλεσης του κάθε σεναρίου ελέγχου και το συνολικό χρόνο ελέγχου.



Σχήμα 4.1 Μεθοδολογία

Κεφάλαιο 5

Αρχιτεκτονική συστήματος – Σύντομη περιγραφή

5.1 Αρχιτεκτονική

Αφού έχει γίνει περιγραφή της μεθοδολογίας που θα ακολουθηθεί για την υλοποίηση του συστήματος [24] σε αυτό το σημείο θα περιγράψουμε τα βασικά στοιχεία της αρχιτεκτονικής του συστήματος.

5.2 Συστατικά μέρη του συστήματος

Τα βασικά συστατικά μέρη του συστήματος είναι τα ακόλουθα:

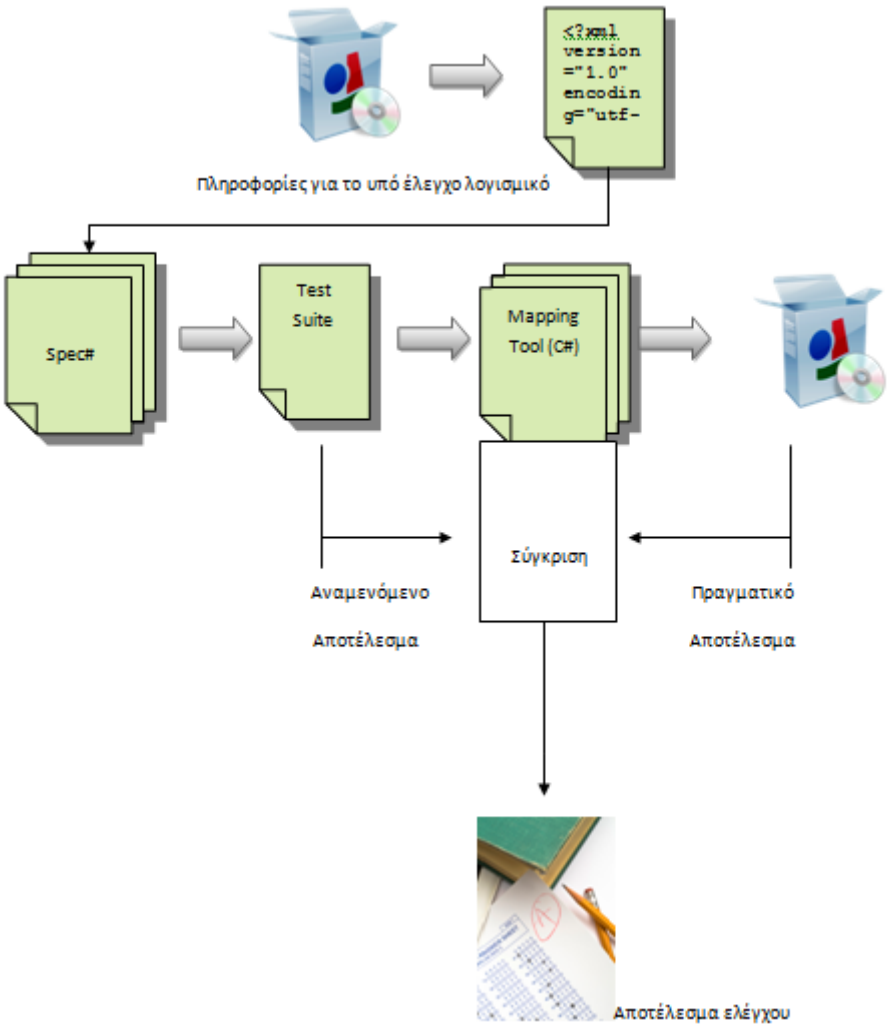
1. Υποσύστημα αναπαράστασης γραφικού περιβάλλοντος του υπό έλεγχο λογισμικού. Το υποσύστημα αυτό αποτελείται βασικά από το Ranorex Spy.

2. Προδιαγραφές γραμμένες σε κώδικα Spec#. Αν και δεν αποτελεί υποσύστημα, οι προδιαγραφές αποτελούν βασικό συστατικό στοιχείο του συστήματος αφού από αυτές θα παραχθούν τα σενάρια ελέγχου και τα αναμενόμενα αποτελέσματα.

3. Υποσύστημα mapping tool το οποίο δέχεται είσοδο τις προδιαγραφές (σε κώδικα Spec#), το εκτελέσιμο αρχείο των προδιαγραφών και το υπό έλεγχο λογισμικό σύστημα. Το υποσύστημα είναι το βασικότερο μέρος ολόκληρου του συστήματος αφού εκτελώντας τις επιμέρους διεργασίες του λαμβάνουμε το αποτέλεσμα του ελέγχου.

Πρέπει να σημειωθεί ότι το κάθε υποσύστημα αποτελείται από άλλα μικρότερα υποσυστήματα.

Στο σχήμα 5.1 πιο κάτω παρουσιάζεται σε διάγραμμα ολόκληρο το σύστημα.



Σχήμα 5.1 Αρχιτεκτονική συστήματος αυτόματου ελέγχου λογισμικού

Κεφάλαιο 6

Αρχιτεκτονική συστήματος - Ανάλυση

6.1 Αναπαράσταση γραφικού περιβάλλοντος

6.1.1 Εισαγωγή

Το γραφικό περιβάλλον διεπαφής χρήστη ή GUI (Graphical User Interface) είναι ο τρόπος με τον οποίο στα σημερινά λογισμικά συστήματα ο χρήστης επικοινωνεί με τον υπολογιστή. Ένα λογισμικό σύστημα με γραφικό περιβάλλον χρησιμοποιεί ένα σύνολο από τεχνολογίες και υλικό ώστε να μπορεί να επιτρέψει στον χρήστη να αλληλεπιδρά με τον υπολογιστή.

Υπάρχουν διάφορα στυλ αλληλεπίδρασης του ανθρώπου με τον υπολογιστή. Μια από τις πιο διαδεδομένες είναι η WIMP [13] (Window, Icon, Menu, Pointer) η οποία επικρατεί στον τομέα τις Πληροφορικής εδώ και 20 χρόνια.

Σχήμα 6.1 Στυλ αλληλεπίδρασης WIMP

Σκοπός χρήσης γραφικού περιβάλλοντος στις εφαρμογές λογισμικού είναι κυρίως η ενίσχυση της αποτελεσματικότητας των χρηστών κατά την χρήση των λογισμικών αυτών μέσα από την ευχρηστία που επιτρέπουν.

6.1.2 Επιμέρους συστατικά του γραφικού περιβάλλοντος λογισμικών συστημάτων

Ένα γραφικό περιβάλλον μπορεί να αναλυθεί στα συστατικά του. Αλληλεπιδρώντας με αυτά ο χρήστης έχει την δυνατότητα να χειριστεί το λογισμικό και να εκτελέσει τις λειτουργίες του. Αυτά παρατίθενται σε συντομία πιο κάτω:

1. Φόρμα. Η φόρμα είναι το όνομα που δόθηκε στο API (Application Programming Interface). Χρησιμοποιείται ως το βασικό μέρος της εφαρμογής πάνω στο οποίο βρίσκονται όλα τα στοιχεία του γραφικού περιβάλλοντος λογισμικού τα οποία συνιστούν την εφαρμογή.

2. Label. Η ετικέτα (label) είναι ίσως το απλούστερο συστατικό του λογισμικού συστήματος που επιτρέπει στον προγραμματιστή να ονομάσει άλλα συστατικά του γραφικού περιβάλλοντος. Ορισμένα χαρακτηριστικά του είναι το χρώμα, το μέγεθος, το είδος των χαρακτήρων. Μπορούν να αλλάζουν χαρακτηριστικά εάν αυτό επιθυμεί ο προγραμματιστής περνώντας το δρομέα του ποντικιού (mouse) από πάνω από το κείμενο τους ή μπορούν να εκτελούν κάποιο κώδικα όταν ο χρήστης πατήσει πάνω τους, αν και αυτό δεν συνηθίζεται πλέον.

3. Textbox. Το κουτί κειμένου (textbox) είναι ένα κουτί μέσα στο οποίο ο χρήστης μπορεί να δώσει δεδομένα στο πρόγραμμα υπό μορφή κειμένου. Ο προγραμματιστής μπορεί να δώσει κάποια αρχική τιμή σε αυτό.

4. Button. Το κουμπί (button) αποτελεί το πιο απλό αλλά ενεργό συστατικό του γραφικού περιβάλλοντος του λογισμικού συστήματος και αυτό γιατί ο χρήστης μπορεί να εκτελέσει τις λειτουργίες της εφαρμογής λογισμικού με το πάτημα του κουμπιού. Ένα βασικό χαρακτηριστικό του κουμπιού είναι η ετικέτα που έχει. Μέσω αυτής, ο προγραμματιστής δίνει στο κουμπί ένα όνομα το οποίο εξηγεί στο χρήστη την λειτουργία που πρόκειται να εκτελεστεί με το πάτημα του κουμπιού.

5. Checkbox. Το checkbox χρησιμοποιείται από τους προγραμματιστές για να δώσουν στους χρήστες της εφαρμογής λογισμικού την επιλογή true / false. Έχουν και αυτά ετικέτα όπως και τα κουμπιά στα οποία υπάρχει ένας επεξηγηματικός τίτλος για την λειτουργία του checkbox. Ο προγραμματιστής μπορεί να αρχικοποιήσει την τιμή του checkbox ώστε να διατηρεί κάποια συγκεκριμένη τιμή κατά την δημιουργία της φόρμας.

6. Radiobutton. Το radiobutton χρησιμοποιείται από τους προγραμματιστές για να δώσουν στους χρήστες πολλαπλές επιλογές, πέραν του true / false. Συνήθως χρησιμοποιούνται σε γκρουπ πέραν των δύο. Έχουν και αυτά ετικέτα για να εξηγούν το ρόλο τους. Και εδώ οι προγραμματιστές έχουν την επιλογή, εάν το θέλουν, να αρχικοποιήσουν την τιμή του radiobutton κατά την δημιουργία της φόρμας.

7. Υπάρχουν και άλλα πολλά άλλα στοιχεία του γραφικού περιβάλλοντος τα οποία δεν θα περιγραφούν στο παρόν άρθρο αλλά δίδονται ως στοιχεία που μπορούν να εισαχθούν σε μια εφαρμογή λογισμικού μέσα από το Microsoft Visual Studio® για εφαρμογές Windows, το Eclipse για εφαρμογές Java κτλ.

6.1.3 Αναπαράσταση γραφικού περιβάλλοντος υπό έλεγχου λογισμικού συστήματος σε αναγνώσιμη μορφή.

Η συνέχεια έδειξε πως έπρεπε με κάποιο τρόπο να γίνει η εξαγωγή των στοιχείων του γραφικού περιβάλλοντος του υπό έλεγχο λογισμικού συστήματος. Μετά από πολλή έρευνα σε διάφορα επιστημονικά άρθρα [25] που αφορούν παρόμοια δουλειά αλλά και στο διαδίκτυο, σχετικά με το αυτόματο έλεγχο γραφικού περιβάλλοντος λογισμικών

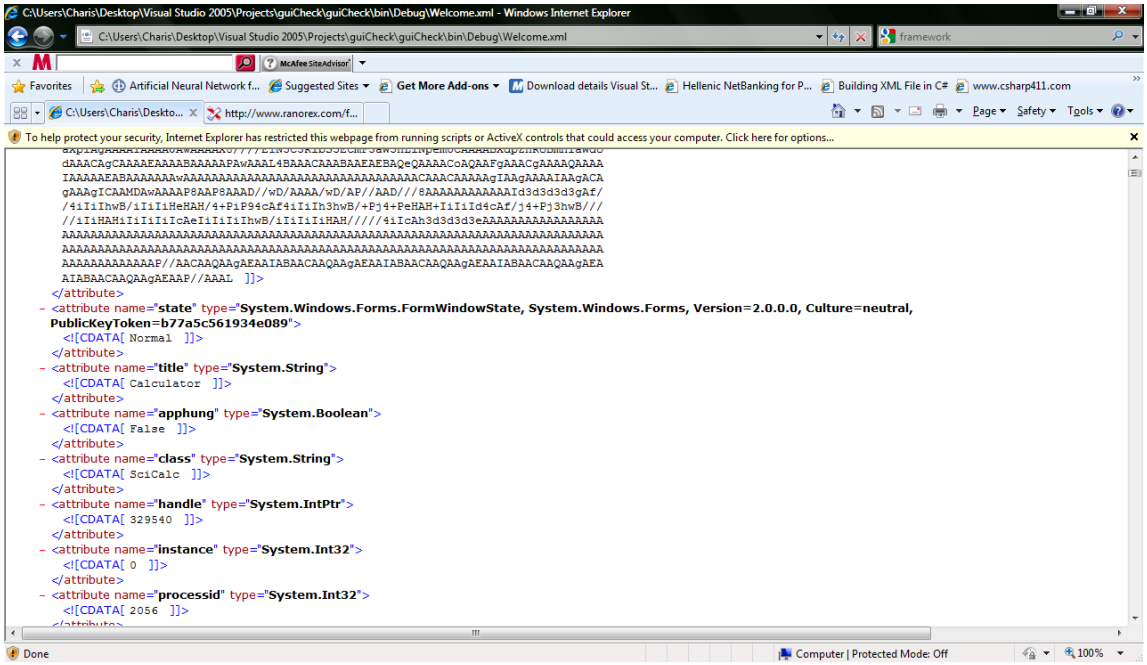
συστημάτων, καθώς και μετά από δοκιμασίες διαφόρων εργαλείων που εξυπηρετούν το σκοπό, συμπέρανα πως το εργαλείο Ranorex® Spy αποτελούσε το κατάλληλο εργαλείο για την εξαγωγή των δεδομένων που χρειαζόμουν.

Η αυστριακή εταιρεία Ranorex GmbH, [14] η οποία είναι βασικά οίκος λογισμικού παράγει προϊόντα που ειδικεύονται αποκλειστικά στον χώρο του ελέγχου λογισμικών συστημάτων.

Το Ranorex αποτελεί ένα αυτόνομο σύστημα ελέγχου λογισμικών συστημάτων, με το οποίο κάποιος μπορεί να υλοποιήσει εφαρμογές με τις οποίες μπορεί να γίνει αυτόματος έλεγχος λογισμικού συστήματος. Οι εφαρμογές που δημιουργούνται με το εργαλείο αυτό είναι εξειδικευμένες για τον έλεγχο ενός συγκεκριμένου συστήματος λογισμικού.

Προφανώς, το σύστημα αυτό σε ολόκληρη του την μορφή δεν μου είναι ιδιαίτερα βοηθητικό. Όμως παρατήρησα πως το εργαλείο που χρησιμοποιεί το σύστημα αυτό διαθέτει μια εφαρμογή η οποία εξάγει πληροφορίες σχετικά με το γραφικό περιβάλλον του υπό έλεγχο λογισμικού συστήματος και τις αποθηκεύει σε μορφή XML. Έτσι μετά από κάποιες δοκιμές αποφάσισα να το χρησιμοποιήσω. Δυστυχώς όμως το εργαλείο αυτό είναι ακριβό και έτσι χρησιμοποίησα μόνο την δοκιμαστική του έκδοση, η οποία λήγει μετά το πέρασμα 30 ημερών από την ημέρα εγκατάστασης. Αυτό δυσκόλεψε την κατάσταση για τον λόγο ότι έπρεπε να γίνεται εγκατάσταση των Windows στον υπολογιστή μου κάθε 30 ημέρες. Εν πάσει περιπτώσει, το εργαλείο χρησιμοποιήθηκε με πολύ καλά αποτελέσματα.

Η μορφή των δεδομένων που εξάγονται από το συγκεκριμένο εργαλείο φαίνεται στο σχήμα 6.2 πιο κάτω.



Σχήμα 6.2 Δομή των στοιχείων του GUI σε XML μορφή.

6.1.4 Υλοποίηση

Αφού όλα τα δεδομένα που χρειαζόμασταν αρχικά βρίσκονταν σε αυτό το XML αρχείο, έπρεπε με κάποιο τρόπο να τα εξάγουμε από αυτό και να τα παρουσιάσουμε στο χρήστη, με κάποιο κατανοητό τρόπο.

Μετά από κάποια έρευνα σχετικά με το XML και τον τρόπο που είναι δομημένο, και αφού μελετήθηκαν οι προδιαγραφές του και κατανοήθηκε η γραμματική του, υλοποιήθηκε ένα κομμάτι κώδικα, το οποίο διαπερνά το κείμενο του XML αρχείου, αναλύει το κείμενο αυτό και στην συνέχεια εξάγει και παρουσιάζει όλα τα χρήσιμα δεδομένα του στον χρήστη.

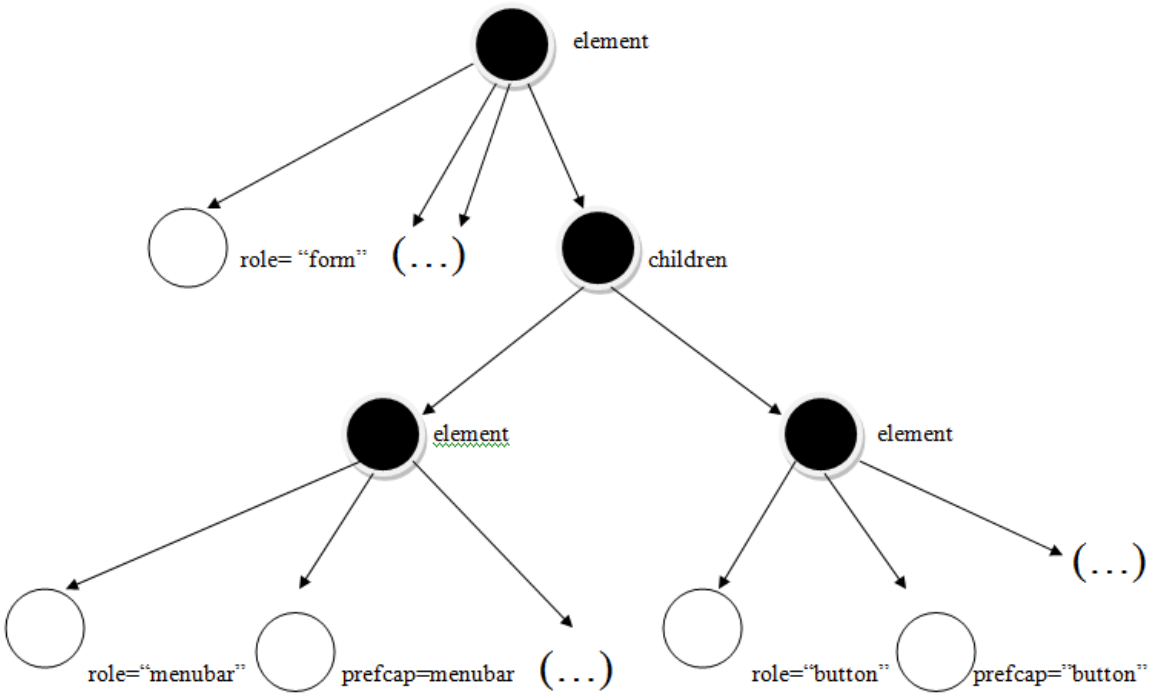
Παράδειγμα ενός XML αρχείου όπου φαίνεται η δομή βλέπουμε στο σχήμα 6.3 πιο κάτω:

```
<?xml version="1.0" encoding="utf-8" ?>
- <snapshot>
  <timestamp>02/01/2009 13:55:01</timestamp>
  <description />
- <element role="form" prefcap="form" capabilities="nativewindow" flavor="win32"
visible="True" valid="True" enabled="True" hasfocus="False" index="0"
rect="138,138,270,264">
    - <children>
        + <element role="menubar" prefcap="menubar"
        capabilities="accessible" flavor="msaa" visible="True" valid="True"
        enabled="True" hasfocus="False" index="32" rect="141,161,254,19">
        + <element role="button" prefcap="button"
        capabilities="nativewindow" flavor="win32" visible="True"
        valid="True" enabled="True" hasfocus="False" index="8"
        rect="200,357,36,29">
      </children>
</element>
```

Σχήμα 6.3 Δομή αρχείου XML

Παρατηρήθηκε πως το αρχείο είναι δομημένο σε δεντρική μορφή και μόνο οι εσωτερικοί του κόμβοι (nodes) καθορίζουν την τοπολογία του δέντρου αυτού.

Έτσι μπορούμε να καθορίσουμε εύκολα το σχήμα του δέντρου σύμφωνα με το σχήμα 6.3 , το οποίο βλέπουμε στο σχήμα 6.4.



Σχήμα 6.4 Παρουσίαση του XML ως δεντροδιάγραμμα

Με βάση την πιο πάνω μορφολογία αλλά και με την χρήση της βιβλιοθήκης System.Xml η οποία υπάρχει στο Visual Studio 2005 της Microsoft έγινε δυνατή η υλοποίηση του XMLParser.

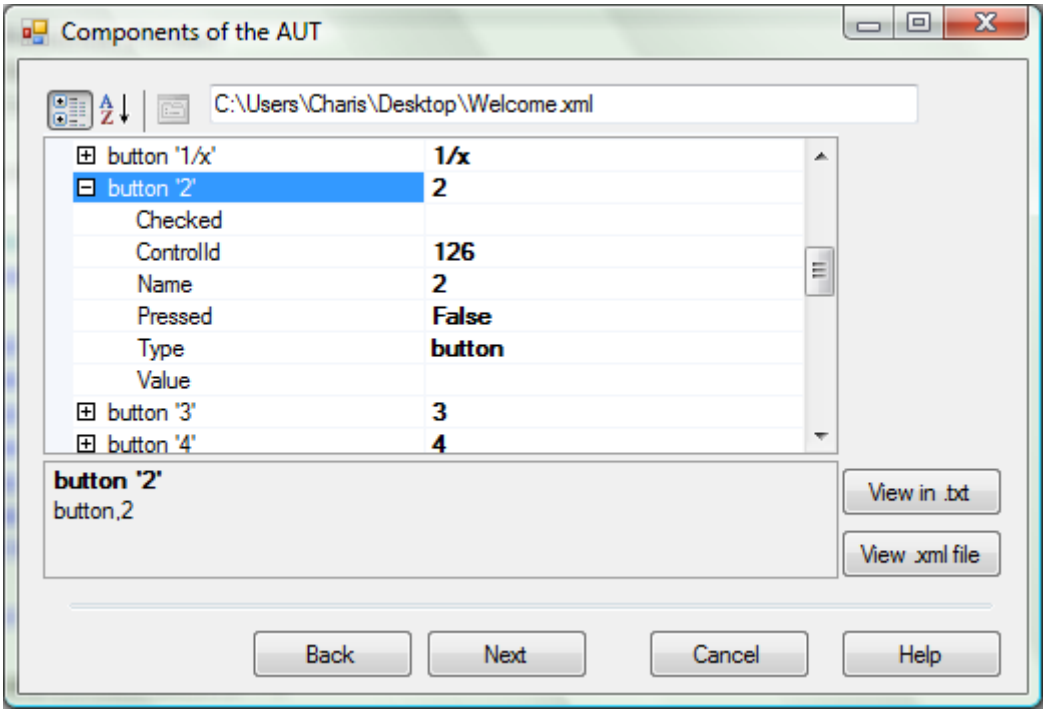
Ο XMLParser υλοποιήθηκε με βάση τα πιο κάτω χαρακτηριστικά:

- Απλός. Δεν χρησιμοποιεί πολλές κλάσεις για να κάνει την ανάλυση του XML αρχείου.
- Αποδοτικός. Η απόδοση χρειάζεται για να μπορεί να χειρίζεται μεγάλα αρχεία σε ελάχιστο χρόνο.
- Υποστηρίζει XML namespaces.
- Βασίζεται στις προδιαγραφές του XML.
- Έχει υποστεί τέτοιες μετατροπές ώστε να χειρίζεται ειδικά τα αρχεία που παράγονται από το Ranorex Spy. Με αυτή την βάση ο Parser που υλοποιήθηκε μετατρέπει όλα τα elements που ανιχνεύει στο αρχείο XML σε ένα πίνακα αντικειμένων Element[] στον οποίο και αποθηκεύονται μαζί με τα χαρακτηριστικά που τα περιγράφουν.

Στη συνέχεια, έχει υλοποιηθεί ένας αριθμός κλάσεων, με βάση τον οποίο διαβάζονται όλα τα αντικείμενα που βρίσκονται στον πίνακα Element[] και παρουσιάζονται στην οθόνη του συστήματος μας μέσα σε ένα propertyGrid.

Η όλη διαδικασία υλοποιήθηκε για να μπορεί να έχει ο χρήστης του συστήματος μας μια πρώτη ιδέα για όλες τις λεπτομέρειες που περιγράφουν το υπό έλεγχο λογισμικό σύστημα και για όλα τα επιμέρους στοιχεία του με βάση την δομή των αρχείων που μας παρέχει το Ranorex Spy.

Στο πιο κάτω σχήμα βλέπουμε την οθόνη του συστήματος στην οποία παρουσιάζονται οι λεπτομέρειες των στοιχείων του γραφικού περιβάλλοντος του υπό έλεγχο λογισμικού και συγκεκριμένα του κουμπιού «2» του Microsoft Calculator.



Σχήμα 6.5 Οθόνη παρουσίασης των αποτελεσμάτων της ανάλυσης που έγινε στο αρχείο XML

6.2 Προδιαγραφές

6.2.1 Εισαγωγή

Ο σκοπός ενός ελέγχου γραφικού περιβάλλοντος σε ένα λογισμικό είναι να ελέγξει κατά πόσο η υλοποίηση του συστήματος συνάδει με τις προδιαγραφές του [2]. Όπως έχει ήδη αναφερθεί το έγγραφο των προδιαγραφών αποτελεί νομικό έγγραφο και συμβόλαιο μεταξύ της εταιρείας που παράγει το λογισμικό και των πελατών, αφού μέσα σε αυτό το έγγραφο βρίσκονται καταγραμμένες οι απαιτήσεις του πελάτη. Έτσι το ζητούμενο αποτέλεσμα μιας υλοποίησης είναι η ικανοποίηση όλων των καταγραμμένων αναγκών του πελάτη.

6.2.2 Εύρεση κατάλληλης γλώσσας για καταγραφή των προδιαγραφών

Οι προδιαγραφές ενός συστήματος, όπως και το ίδιο το σύστημα μπορούν να μοντελοποιηθούν με χρήση κάποιας γλώσσας μοντελοποίησης ή modeling language. Μέσα από αυτές τις γλώσσες μπορούν να εκφραστούν πληροφορίες, γνώση ή συστήματα υπό μορφή μοντέλων σε κάποια δομημένη και κατανοητή μορφή, η οποία προκαθορίζεται με συγκεκριμένους κανόνες, ανάλογα με την γλώσσα.

Η μορφή των μοντέλων αυτών μπορεί να είναι είτε γραφική είτε σαν κείμενο. Οι γλώσσες γραφικής αναπαράστασης μοντέλων χρησιμοποιούν τεχνικές διαγραμμάτων με συγκεκριμένα σύμβολα. Αντίθετα, οι γλώσσες αναπαράστασης μοντέλων υπό μορφή κειμένου χρησιμοποιούν τυποποιημένες λέξεις και παραμέτρους για να αναπαραστήσουν τα μοντέλα.

Αρχικά, η κατεύθυνση της έρευνας [16, 17, 18] για το ποια γλώσσα μοντελοποίησης θα χρησιμοποιηθεί στο σύστημα έτεινε προς την UML ή Unified Modeling Language, η οποία είναι μια ευρέως διαδεδομένη γλώσσα μοντελοποίησης με γραφική αναπαράσταση μοντέλων. Κατά την διάρκεια της έρευνας, βρέθηκε μια άλλη γλώσσα, η AsmL, [19, 20] συντομογραφία του Abstract State Machine Language, η οποία παράγει μοντέλα υπό μορφή κειμένου. Με την γλώσσα αυτή μπορούμε να δημιουργήσουμε μοντέλα αφηρημένα, τα οποία τρέχουν με την βοήθεια του εργαλείου SpecExplorer [21, 22, 23]. Κατά την μελέτη της γλώσσας αυτής και του εργαλείου SpecExplorer παρατήρησα πως το εργαλείο δέχεται σαν είσοδο και γλώσσα Spec#.

Μελετώντας την αρθρογραφία σχετικά με την γλώσσα Spec#, παρατήρησα πως αποτελεί επέκταση της γλώσσας προγραμματισμού C# και πως μπορεί κάποιος να γράψει τον κώδικα της στο περιβάλλον του Visual Studio, το οποίο διαθέτει και τον κατάλληλο μεταγλωττιστή (compiler). Αυτό που με οδήγησε στο να την χρησιμοποιήσω στο σύστημα μου είναι το βασικό της πλεονέκτημα, δηλαδή το γεγονός πως οι προδιαγραφές που καταγράφονται σε αυτή την γλώσσα είναι εκτελέσιμες, και για αυτόν τον λόγο προτιμήθηκε.

6.2.3 Spec#

Η γλώσσα Spec# είναι μια γλώσσας περιγραφής προδιαγραφών. Αποτελεί μια νέα προσπάθεια υποβοήθησης της ανάπτυξης και συντήρησης υψηλής ποιότητας λογισμικών συστημάτων. Η χρήση της είναι σημαντική για την παραγωγή σεναρίων ελέγχου για εφαρμογές αλλά και για την παραγωγή των αναμενόμενων αποτελεσμάτων από την εκτέλεση του υπό έλεγχου λογισμικού συστήματος.

Με την γλώσσα αυτή μπορεί κάποιος να καταγράψει τις προδιαγραφές ως ένα σύνολο από Pre-conditions, Actions και Post-Conditions. Τα Pre-Conditions παρουσιάζονται στην γλώσσα αυτή με την χρήση της λέξης “requires” ενώ τα Post-Conditions παρουσιάζονται στο κείμενο της γλώσσας χρησιμοποιώντας την λέξη “ensures”. Τα actions παρουσιάζονται με την χρήση του συμβολισμού “[Action]” ή με άλλες εντολές της γλώσσας.

6.2.4 Καταγραφή των προδιαγραφών σε γλώσσα Spec#

Όπως έχει προαναφερθεί το περιβάλλον του Visual Studio μπορεί να ενσωματώσει την γλώσσα Spec# απλά κατεβάζοντας και εκτελώντας ένα αρχείο plug-in για την έκδοση του Visual Studio που διαθέτει ο προγραμματιστής. Το εκτελέσιμο αυτό αρχείο περιέχει το συντακτικό της γλώσσας, τον μεταγλωττιστή κτλ.

Η γραφή προδιαγραφών με την γλώσσα αυτή δεν είναι δύσκολη αφού αποτελεί υπερσύνολο της γλώσσας προγραμματισμού C#.

Αρχικά, αφού δημιουργείται ένα project spec# μέσω του μενού του Visual Studio (File -> New -> Project -> Spec# Project -> Windows Application), αρχίζει η καταγραφή των προδιαγραφών.

Ως παράδειγμα εδώ θα γίνει υλοποίηση των προδιαγραφών για την εφαρμογή Calculator της Microsoft. Η εφαρμογή αυτή αποτελεί πολύ καλό παράδειγμα ελέγχου γραφικού περιβάλλοντος αφού η χρήση της είναι γνωστή σε όλους και δεν χρειάζεται να παρουσιαστούν οι λειτουργίες της, είναι απλή και αποτελείται από πολλά στοιχεία που χαρακτηρίζουν ένα γραφικό περιβάλλον.

Πρώτα δημιουργούνται κλάσεις για το κάθε είδους στοιχείο που περιέχεται στο υπό έλεγχο λογισμικό. Εδώ θα υλοποιήσουμε τις κλάσεις button και text αντίστοιχα για τα buttons και τα textboxes του υπό έλεγχο λογισμικού συστήματος. Το κάθε αντικείμενο που δημιουργείται περιγράφεται με τα χαρακτηριστικά Name, Pressed, Type, ControlId, Checked και Value τα οποία αρχικοποιούνται με τα αρχικά χαρακτηριστικά του υπό έλεγχο λογισμικού, όπως αυτά αποθηκεύονται από το Ranorex Spy. Για να γίνει αυτό γίνεται αυτόματη αποθήκευση ενός αρχείου κειμένου (text file) με συγκεκριμένη δομή, έτσι ώστε να μπορούν να διαβάζονται τα χαρακτηριστικά των στοιχείων του γραφικού περιβάλλοντος του υπό έλεγχο λογισμικού συστήματος.

Στην συνέχεια δημιουργούνται οι μέθοδοι που εκτελούν όλες τις άλλες λειτουργίες του συστήματος. Στο παράδειγμα θα υλοποιηθούν οι μέθοδοι:

1. `static button addValues(string name, string pressed, string type,int controlId,string checked1, string value1)` η οποία είναι τύπου button και η

λειτουργία της είναι η αρχικοποίηση των χαρακτηριστικών του κάθε αντικειμένου button που δημιουργείται.

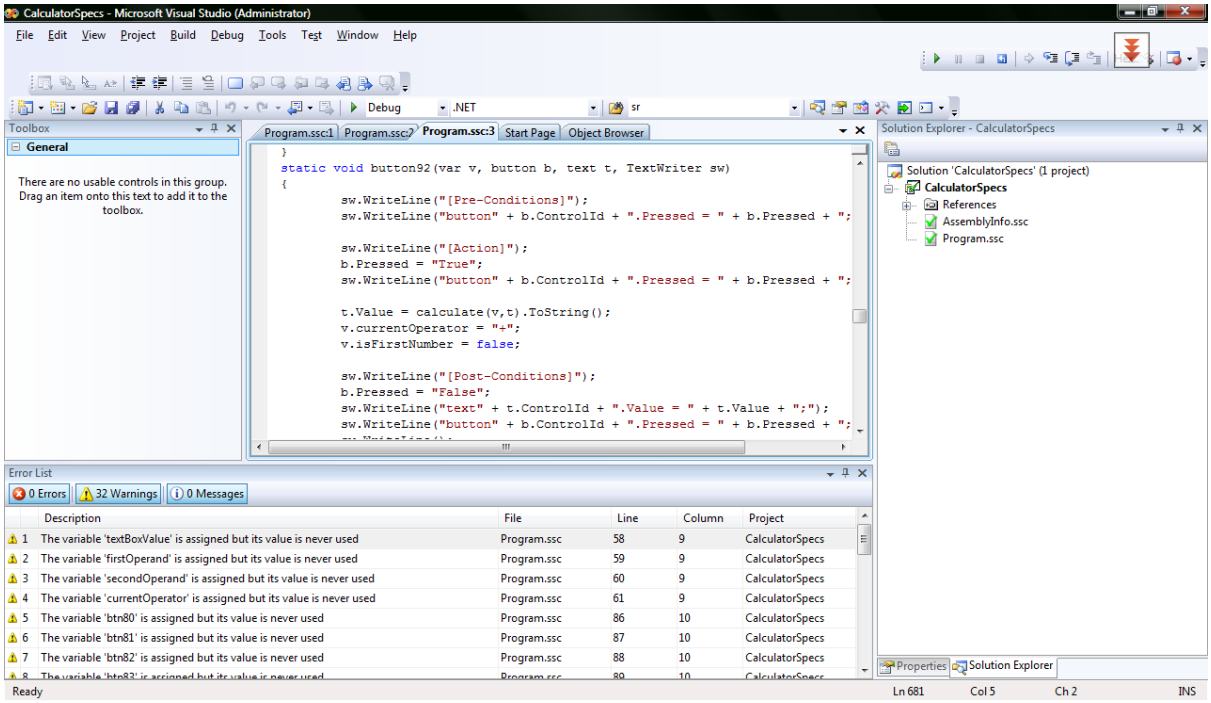
2. `static text addTextValues(string name, string pressed, string type,int controlId,string checked1, string value1)` η οποία είναι τύπου text και η λειτουργία της είναι η αρχικοποίηση των χαρακτηριστικών του κάθε αντικειμένου text που δημιουργείται.
3. `static double calculate(var v, text txt)` η οποία είναι τύπου double και η λειτουργία της είναι η εκτέλεση των υπολογισμού της κάθε πράξης που εκτελείται στο υπό έλεγχο λογισμικό σύστημα, δηλαδή το Calculator της Microsoft.

4. `static var numericButton(button b,var v, text t)` η οποία είναι τύπου var και η λειτουργία της είναι να εκτελεί τις λειτουργίες της εφαρμογής όταν κάποιος κάνει κλικ σε αριθμητικές εντολές π.χ. κλικ στο «2». Ο τύπος var, που είναι και ο τύπος επιστροφής της μεθόδου αυτής δημιουργήθηκε για να μπορούν να επιστρέφονται περισσότερα από ένα δεδομένα στην εφαρμογή.

5. `static void operationButton (var v, button b, text t)` η οποία είναι τύπου void, δηλαδή δεν επιστρέφει τίποτα στην μέθοδο που την καλεί και η λειτουργία της είναι να εκτελεί τις αριθμητικές πράξεις όταν κάποιος κάνει κλικ σε εντολές πράξεων, π.χ. κλικ στο «+».

Αυτό που απομένει από την δημιουργία των προδιαγραφών του υπό έλεγχο λογισμικού συστήματος είναι η δημιουργία μεθόδων για το κάθε ένα από τα στοιχεία του γραφικού περιβάλλοντος. Αυτές οι μέθοδοι κάνουν κλήση στις μεθόδους που έχουν περιγραφεί πιο πάνω και λειτουργούν όπως λειτουργούν ή θα έπρεπε να λειτουργούν τα χειριστήρια του υπό έλεγχο λογισμικού συστήματος. Αυτές οι μέθοδοι χρησιμοποιούνται επίσης και για την παραγωγή σεναρίων ελέγχου (test cases) αφού κατά την εκτέλεση των προδιαγραφών (έχει αναφερθεί πιο πάνω πως οι προδιαγραφές με την χρήση της γλώσσας Spec# είναι εκτελέσιμες) παράγουν ένα αρχείο που περιέχει όλα τα σενάρια ελέγχου που θα εκτελεστούν.

Αξίζει να σημειωθεί εδώ πως τα σενάρια ελέγχου που παράγονται δεν αγνοούν το αποτέλεσμα των λειτουργιών, αφού στα σενάρια αυτά περιέχεται το αναμενόμενο αποτέλεσμα που θα πρέπει να επιστρέφει στο χρήστη το υπό έλεγχο λογισμικό σύστημα.



Σχήμα 6.6 Περιβάλλον γραφής κώδικα Spec# του Microsoft Visual Studio 2008

Για λόγους ορθότητας κατά την δημιουργία και καταγραφή των προδιαγραφών έγιναν οι πιο κάτω παραδοχές:

1. Πρέπει στην αρχή του αρχείου των προδιαγραφών να υπάρχουν ορισμοί των κλάσεων των αντικειμένων, για τα στοιχεία του γραφικού περιβάλλοντος του υπό έλεγχο λογισμικού συστήματος όπως πιο κάτω:

```
class button
      {
         i.  public string Name;
        ii.  public string Pressed;
       iii.  public string Type;
        iv.  public int ControlId;
         v.  public string Checked;
        vi.  public string Value;

      }
```

2. Στις κλάσεις αυτές πρέπει να υπάρχουν μεταβλητές οι οποίες αποθηκεύουν όλα τα δεδομένα που εξάγονται από το γραφικό περιβάλλον του υπό έλεγχο λογισμικού συστήματος. Για σκοπούς ορθότητας, ευκολίας και αυτοματισμού, κατά την δημιουργία των αντικειμένων των κλάσεων αυτών, αυτά αρχικοποιούνται από το αρχείο SpecsParameters.txt το οποίο βρίσκεται αποθηκευμένο στο φάκελο του συστήματος.

3. Για την αρχικοποίηση των αντικειμένων των προαναφερομένων κλάσεων πρέπει να δημιουργηθούν μέθοδοι οι οποίες επιστρέφουν το αντικείμενο όπως στο πιο κάτω παράδειγμα:

```
static button addButtonValues(string name, string pressed, string
type,int controlId,string checked1, string value1)
      {
                  button btn = new button();
                  btn.Name = name;
                  btn.Pressed = pressed;
                  btn.Type = type;
                  btn.ControlId = controlId;
                  btn.Checked = checked1;
                  btn.Value = value1;
                  return btn;
      }
```

4. Πριν την καταγραφή των μεθόδων των στοιχείων του γραφικού περιβάλλοντος του υπό έλεγχο λογισμικού συστήματος πρέπει να υπάρχει το αναγνωριστικό «// Elements » ούτως ώστε να αναγνωρίζει το σύστημα ότι κάτω από αυτή υπάρχουν καταγραμμένες οι μέθοδοι των στοιχείων του γραφικού περιβάλλοντος.

5. Πρέπει να υπάρχουν μέθοδοι οι οποίοι αναπαριστούν την λειτουργία των στοιχείων του γραφικού περιβάλλοντος. Σε αυτές τις μεθόδους πρέπει να υπάρχει απαραίτητα ένας δείκτης σε αρχείο ο οποίος θα καταγράφει σε αυτό την τριάδα Pre-Condition – Action – Post Condition. Ακόμη θα πρέπει απαραίτητα να στέλνονται ως παράμετροι τα αντικείμενα των κλάσεων που επηρεάζονται από την μέθοδο.

6. Η δομή μιας τέτοιας μεθόδου φαίνεται πιο κάτω:

```
static void button90(var v, button b, text t, TextWriter sw)
{
      sw.WriteLine("[Pre-Conditions]");
      sw.WriteLine("button" + b.ControlId + ".Pressed = " + b.Pressed
      + ";");

      sw.WriteLine("[Action]");
      b.Pressed = "True";
      sw.WriteLine("button" + b.ControlId + ".Pressed = " + b.Pressed
      + ";");

      t.Value = calculate(v,t).ToString();
      v.currentOperator = "/";
      v.isFirstNumber = false;

      sw.WriteLine("[Post-Conditions]");
      b.Pressed = "False";
      sw.WriteLine("text" + t.ControlId + ".Value = " + t.Value +
      ";");
      sw.WriteLine("button" + b.ControlId + ".Pressed = " + b.Pressed
      + ";");
      sw.WriteLine();
```

```
}
```

7. Απαραίτητα θα πρέπει να υπάρχουν και τα τρία αναγνωριστικά Pre-Condition – Action – Post Condition όπως φαίνονται πιο πάνω ώστε να μπορούν να διαβάζονται σωστά τα σενάρια ελέγχου που θα παράγονται από το σύστημα.

8. Στο τέλος των Post-Conditions πρέπει να υπάρχει απαραίτητα κενή γραμμή.

6.3 Σενάρια ελέγχου

6.3.1 Εισαγωγή

Κατά την διαδικασία ελέγχου λογισμικών συστημάτων, επιβάλλεται να υπάρχει μια συλλογή από σενάρια ελέγχου (test cases) σκοπός της οποίας είναι για να χρησιμοποιηθεί και να δοκιμάσει το υπό έλεγχο λογισμικό σύστημα έτσι ώστε να αποδειχτεί ότι το σύστημα αυτό συμπεριφέρεται όπως αναμενόταν. Τα σενάρια αυτά περιέχουν λεπτομερείς οδηγίες για την λειτουργία του λογισμικού συστήματος για το οποίο καταγράφονται.

Υπάρχουν δύο είδη σεναρίων ελέγχου α) τα τυπικά σενάρια ελέγχου και β) τα άτυπα σενάρια ελέγχου. Η διαφορά αυτών των δύο τρόπων δημιουργίας και καταγραφής σεναρίων ελέγχου βρίσκεται στο γεγονός ότι στα τυπικά σενάρια ελέγχου είναι γνωστή η είσοδος και το αναμενόμενο αποτέλεσμα. Αυτά τα χαρακτηριστικά αναγράφονται στα σενάρια ελέγχου και συμβολίζονται με τα αναγνωριστικά pre-conditions και post-conditions ανάλογα.

6.3.2 Δημιουργία σεναρίων ελέγχου

Τα σενάρια όπως αναφέρθηκε και πιο πάνω δημιουργούνται αυτόματα με την εκτέλεση των προδιαγραφών. Για την δημιουργία του απαιτείται κάποια μικρή μετατροπή στο κώδικα των προδιαγραφών. Η μετατροπή αυτή γίνεται εύκολα αφού χρειάζεται μόνο μια μικρή προσθήκη κάποιων γραμμών κώδικα σε κάθε αναπαράσταση στοιχείου γραφικού περιβάλλοντος που βρίσκεται στις προδιαγραφές.

Για να γίνει κατανοητός πιο εύκολα ο τρόπος που γίνεται αυτή η μετατροπή στο κώδικα των προδιαγραφών μπορούμε να δούμε τον πιο κάτω κώδικα που αναφέρεται στο κουμπί button128:

```
static var button128(button b,var v, text t, TextWriter sw)
              {

              double number  = double.Parse(b.Name);
              if (v.currentOperator == "=")
              {
              v.currentOperator = "";
              t.Value = number.ToString();
              v.isFirstNumber = false;
              }
              if (v.isFirstNumber)
              {
                     if (t.Value == "0")
                            {
                                   t.Value = number.ToString();
                            }
                     else
                            {
                                   t.Value = t.Value + number.ToString();
                            }
              }
              else
              {
              v.isFirstNumber = true;
              if (v.currentOperator !="")
                     v.firstOperand = double.Parse(t.Value);
              t.Value = number.ToString();
              }
              return v;
              }
```

Ο κώδικας μετά την μετατροπή θα έχει ως ακολούθως:

```
static var button128(button b,var v, text t, TextWriter sw)
              {
              sw.WriteLine("[Pre-Conditions]");
              sw.WriteLine("button" + b.ControlId + ".Pressed = " + b.Pressed +
";");

              sw.WriteLine("[Action]");
              b.Pressed = "True";
              sw.WriteLine("button" + b.ControlId + ".Pressed = " + b.Pressed +
";");

              double number  = double.Parse(b.Name);
              if (v.currentOperator == "=")
              {
              v.currentOperator = "";
              t.Value = number.ToString();
              v.isFirstNumber = false;
              }
              if (v.isFirstNumber)
              {
                     if (t.Value == "0")
                            {
                                   t.Value = number.ToString();
                            }
                     else
                            {
                                   t.Value = t.Value + number.ToString();
                            }
              }
              else
              {
              v.isFirstNumber = true;
              if (v.currentOperator !="")
                     v.firstOperand = double.Parse(t.Value);
              t.Value = number.ToString();
              }
              sw.WriteLine("[Post-Conditions]");
```

```
                sw.WriteLine("text" + t.ControlId + ".Value = " + t.Value + ";");
                b.Pressed = "False";
                sw.WriteLine("button" + b.ControlId + ".Pressed = " + b.Pressed +
";");
                sw.WriteLine();
                return v;
                }
```

Παρατηρείται πως προστέθηκαν μόνο 10 γραμμές κώδικα . Ο κώδικας αυτός ενδέχεται να αλλάξει ανάλογα με την υλοποίηση. Στην περίπτωση όμως του συγκεκριμένου υπό έλεγχο λογισμικού συστήματος ο κώδικας αυτός παραμένει ο ίδιος για όλα τα κουμπιά (buttons). Στο πιο πάνω κώδικα το sw που φαίνεται σε κάθε γραμμή αναφέρεται σε αντικείμενο τύπου StreamWriter, το οποίο γράφει σε αρχείο κειμένου (text file).

Τα σενάρια ελέγχου που παράγονται μπορούν να είναι είτε τυχαία είτε συγκεκριμένα, και δημιουργούνται με βάση ενός test string το οποίο δημιουργείται αυτόματα, είτε εισάγεται από τον χρήστη του συστήματος. Στην περίπτωση της αυτόματης και τυχαίας δημιουργίας του test string λαμβάνονται υπόψη δυο παράμετροι. Η πρώτη παράμετρος αφορά μια λίστα αντικειμένων στην οποία αποθηκεύονται τα στοιχεία του γραφικού περιβάλλοντος του υπό έλεγχο λογισμικού συστήματος τα οποία ο χρήστης θέλει να συμπεριλάβει στον έλεγχο. Η δεύτερη παράμετρος αφορά το μήκος του σεναρίου ελέγχου που θέλει να δημιουργήσει. Με την λέξη μήκος εννοείται ο αριθμός των στοιχείων του γραφικού περιβάλλοντος που θέλει ο χρήστης να χειριστεί κατά την διάρκεια του έλεγχου. Για παράδειγμα στην συγκεκριμένη εφαρμογή που θα ελεγχθεί, εάν ο χρήστης θα δώσει παράμετρο 10, τότε στην εκτέλεση του σεναρίου θα πατηθούν δέκα κουμπιά (buttons).

Αυτή η μέθοδος παραγωγής σεναρίων εκτέλεσης μπορεί να παρομοιαστεί με την μέθοδο ελέγχου που χρησιμοποιεί test monkeys με σημαντική διαφορά πως στην δική μας περίπτωση τα σενάρια ελέγχου περιέχουν και το αναμενόμενο αποτέλεσμα. Η διαδικασία ελέγχου monkey αναφέρεται σε ένα αυτοματοποιημένο εργαλείο που δεν έχει καμία γνώση όσον αφορά το υπό έλεγχο λογισμικό σύστημα και εκτελεί τυχαία κλικ σε στοιχεία του γραφικού περιβάλλοντος του λογισμικού συστήματος αυτού.

Τα σενάρια ελέγχου που παράγονται από το σύστημα που υλοποιήθηκε έχουν την ακόλουθη μορφή:

```
[Pre-Conditions]
button128.Pressed = False;
[Action]
button128.Pressed = True;
```

[Post-Conditions]

text403.Value = 4;

button128.Pressed = False;

Η καταγραφή των σεναρίων ελέγχου έγινε με αυτό τον τρόπο ούτως ώστε να αναγνωρίζονται και να διαβάζονται εύκολα από το επόμενο μέρος του συστήματος, το οποίο κάνει mapping στο γραφικό περιβάλλον του υπό έλεγχο λογισμικού εκτελώντας τις εντολές που του δίνονται μέσω των σεναρίων αυτών. Ακόμη μπορεί να παρατηρηθεί πως τα σενάρια ελέγχου διατηρούν κάποια δομημένη μορφή και αποτελούν τυπικά σενάρια ελέγχου σύμφωνα με τον ορισμό που δόθηκε στην εισαγωγή (§ 6.3.1) αυτού του υποκεφαλαίου.

6.4 Εργαλείο mapping

6.4.1 Εισαγωγή

Ο σκοπός ύπαρξης ενός εργαλείου mapping είναι να μειώσει στο ελάχιστο την χειρωνακτική εργασία που πρέπει να κάνει ο προγραμματιστής κατά την διάρκεια του ελέγχου. Στο σύστημα που υλοποιήθηκε έγινε εξάλειψη της οποιασδήποτε συμμετοχής χρήστη κατά την διαδικασία εκτέλεσης των σεναρίων ελέγχου.

6.4.2 Υλοποίηση

Το υποσύστημα αυτό χρησιμοποιεί τρεις βιβλιοθήκες από το έτοιμο σύστημα Ranorex, την RanorexCore.dll, την RanorexNet.dll και την RanorexSpy.dll οι οποίες έδωσαν την δυνατότητα της εκτέλεσης εντολών αλληλεπίδρασης του συστήματος με το υπό έλεγχο λογισμικό σύστημα αυτόματα.

Το υποσύστημα που υλοποιήθηκε καλεί αρχικά το υπό έλεγχο λογισμικό σύστημα με την χρήση του ονόματος της φόρμας του προγράμματος. Στην συνέχεια καλεί ένα δείκτη σε ένα αρχείο τύπου StreamReader ο οποίος θα αρχικοποιηθεί με το αρχείο που είναι αποθηκευμένα τα σενάρια ελέγχου.

Το σύστημα διατρέχει το αρχείο με τα σενάρια ελέγχου γραμμή προς γραμμή και με την βοήθεια κανονικών εκφράσεων τεμαχίζει την κάθε πρόταση ώστε να ξεχωρίσει τις επιμέρους λέξεις και αποθηκεύει τις λέξεις αυτές στις κατάλληλες μεταβλητές. Οι μεταβλητές αυτές

περιέχουν το ControlId του στοιχείου, το είδος του στοιχείου καθώς και την κατάσταση που βρίσκεται π.χ. Checked, Pressed κτλ.

Το πρόγραμμα ψάχνει να βρει την πρόταση [Pre-Conditions] στο αρχείο. Έτσι με την εύρεση της πρότασης [Pre-Conditions] ο δείκτης στο αρχείο προχωρεί στην επόμενη γραμμή και αποθηκεύει τις πληροφορίες που πρέπει όπως αυτές αναφέρθηκαν πιο πάνω. Στη συνέχεια δημιουργεί ένα αντικείμενο με τον κατάλληλο τύπο, και προσπαθεί να το αντιστοιχίσει με το ανάλογο στοιχείο στο υπό έλεγχο λογισμικό σύστημα. Αν αποτύχει να το αντιστοιχίσει τότε επιστρέφεται κενό αντικείμενο και προχωρεί στην επόμενη πρόταση. Αν επιτύχει τότε το σύστημα συγκρίνει τα Pre-Conditions με αυτά που είναι αποθηκευμένα στα αντίστοιχα αντικείμενα.

Αφού τελειώσουν οι αρχικοί έλεγχοι των Pre-Conditions τότε ο δείκτης στο αρχείο των σεναρίων ελέγχου θα δείξει στην πρόταση [Actions]. Με την αναγνώριση της πρότασης αυτής, το σύστημα ελέγχου καταλαβαίνει πως θα πρέπει να εκτελέσει κάποια δράση στο υπό έλεγχο σύστημα λογισμικού. Κάνοντας ανάγνωση της πρότασης, τεμαχισμό και αποθήκευση των λέξεων της πρότασης στις κατάλληλες μεταβλητές το σύστημα αλληλεπιδρά με το υπό έλεγχο λογισμικό σύστημα, δημιουργεί το κατάλληλο στοιχείο και εκτελεί την ανάλογη δράση στο γραφικό περιβάλλον του υπό έλεγχο συστήματος λογισμικού.

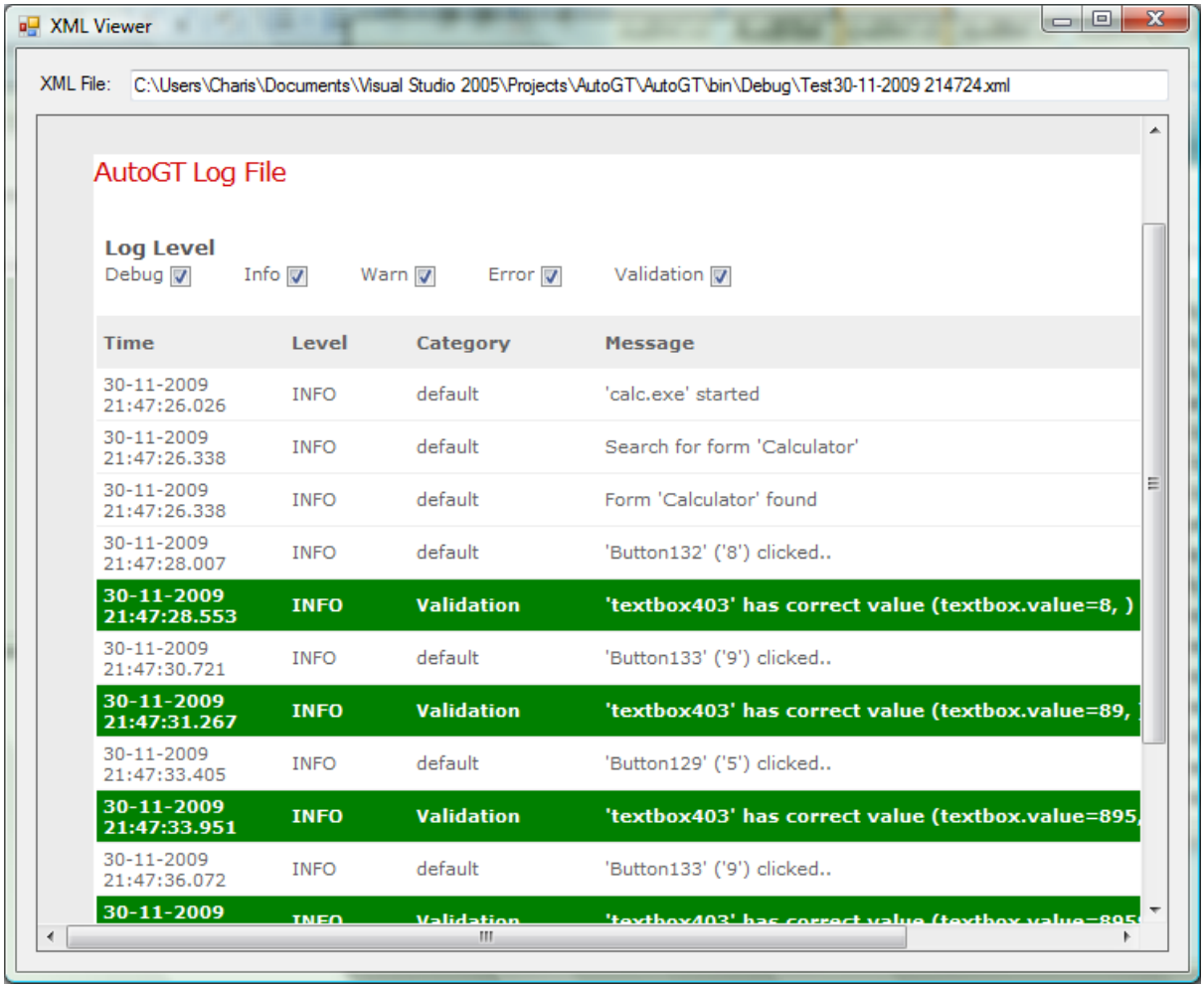
Ολοκληρώνοντας την δράση του, το σύστημα θα αναγνωρίσει την πρόταση [Post-Conditions] και με αυτό δεικνύεται στο σύστημα ότι θα πρέπει να κάνει και πάλι έλεγχο αν η δράση έφερε το σωστό αποτέλεσμα, με βάση τις προδιαγραφές. Και πάλι, το σύστημα διαβάζει από το αρχείο που βρίσκονται αποθηκευμένα τα σενάρια ελέγχου ποιους ελέγχους πρέπει να κάνει, κάνει το τεμαχισμό των προτάσεων, αποθηκεύει τα δεδομένα, δημιουργεί τα κατάλληλα αντικείμενα και εκτελεί τους ανάλογους ελέγχους όπως αυτοί ορίστηκαν στο σενάριο ελέγχου.

Τελειώνοντας την εκτέλεση όλων των σεναρίων ελέγχου το σύστημα παρουσιάζει τα αποτελέσματα του έλεγχου στο χρήστη ανοίγοντας ένα αρχείο .xml στο οποίο είναι αποθηκευμένα.

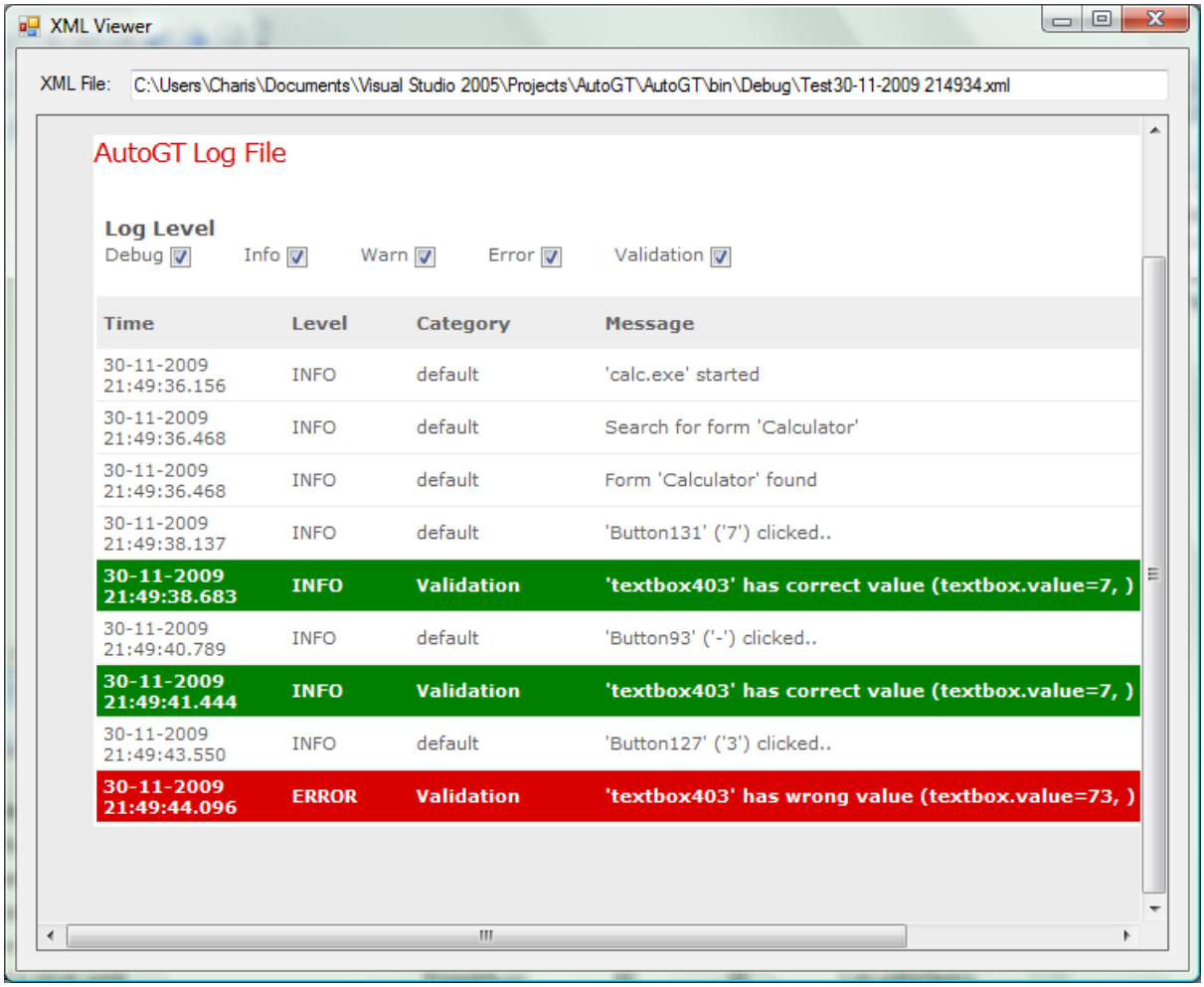
Αξίζει να σημειωθεί πως για την καλύτερη παρουσίαση των αποτελεσμάτων, το αρχείο xml κάνει αναφορά σε αρχεία .css τα οποία κάνουν την παρουσίαση των αποτελεσμάτων πιο οργανωμένη και ευανάγνωστη.

6.5 Βοηθητικά εργαλεία

Για την καλύτερη οπτική αναπαράσταση των αποτελεσμάτων αλλά και όλων των αρχείων xml που χρησιμοποιούνται από το σύστημα, έχει δημιουργηθεί ένα πρόγραμμα το οποίο ανοίγει αρχεία .xml. Το πρόγραμμα αυτό δέχεται σαν είσοδο ένα αρχείο .xml και χρησιμοποιώντας ένα web browser control παρουσιάζει καλύτερα αισθητικά το αρχείο.



Σχήμα 6.7 Παρουσίαση αποτελεσμάτων ελέγχου χωρίς εύρεση λαθών



Σχήμα 6.8 Παρουσίαση αποτελεσμάτων ελέγχου με ανίχνευση λάθους

Κεφάλαιο 7

Εκτέλεση του συστήματος

7.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα παρουσιάσουμε βήμα προς βήμα την εκτέλεση του συστήματος για δυο εφαρμογές.

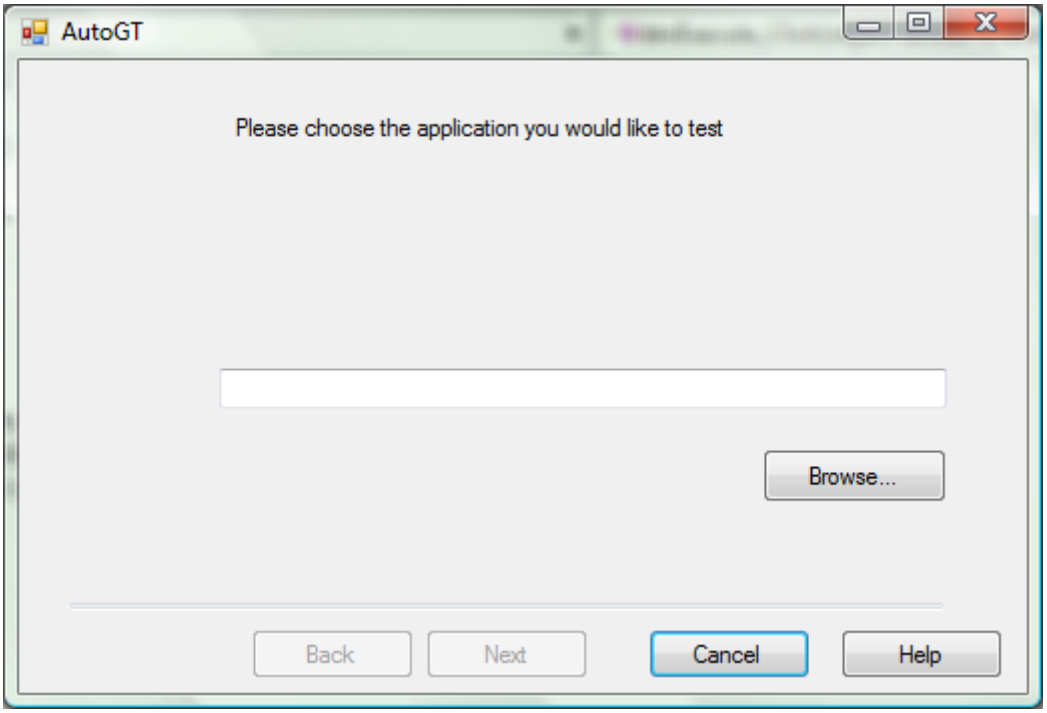
Στο υποκεφάλαιο 6.2 θα παρουσιαστούν δύο εκτελέσεις τους συστήματος ελέγχου με είσοδο την εφαρμογή Calculator (calc.exe) της Microsoft. Στην πρώτη εκτέλεση δεν θα γίνει ανεύρεση λάθους ενώ στην δεύτερη εκτέλεση θα ανιχνευτούν λάθη. Τα λάθη αυτά αφορούν και αποτελούν διαφορές μεταξύ των προδιαγραφών και της πραγματικής υλοποίησης.

Στο υποκεφάλαιο 6.3 θα παρουσιαστεί εκτέλεση του συστήματος ελέγχου με είσοδο μια user defined εφαρμογή, η οποία δεν διαθέτει καμία λειτουργικότητα. Ο λόγος που θα γίνει παρουσίαση της εφαρμογής αυτής είναι για να δείξουμε ότι το σύστημα μπορεί να χειριστεί και άλλα στοιχεία πέραν των buttons και των textboxes, δηλαδή των checkboxes και των radiobuttons.

7.2 Calculator

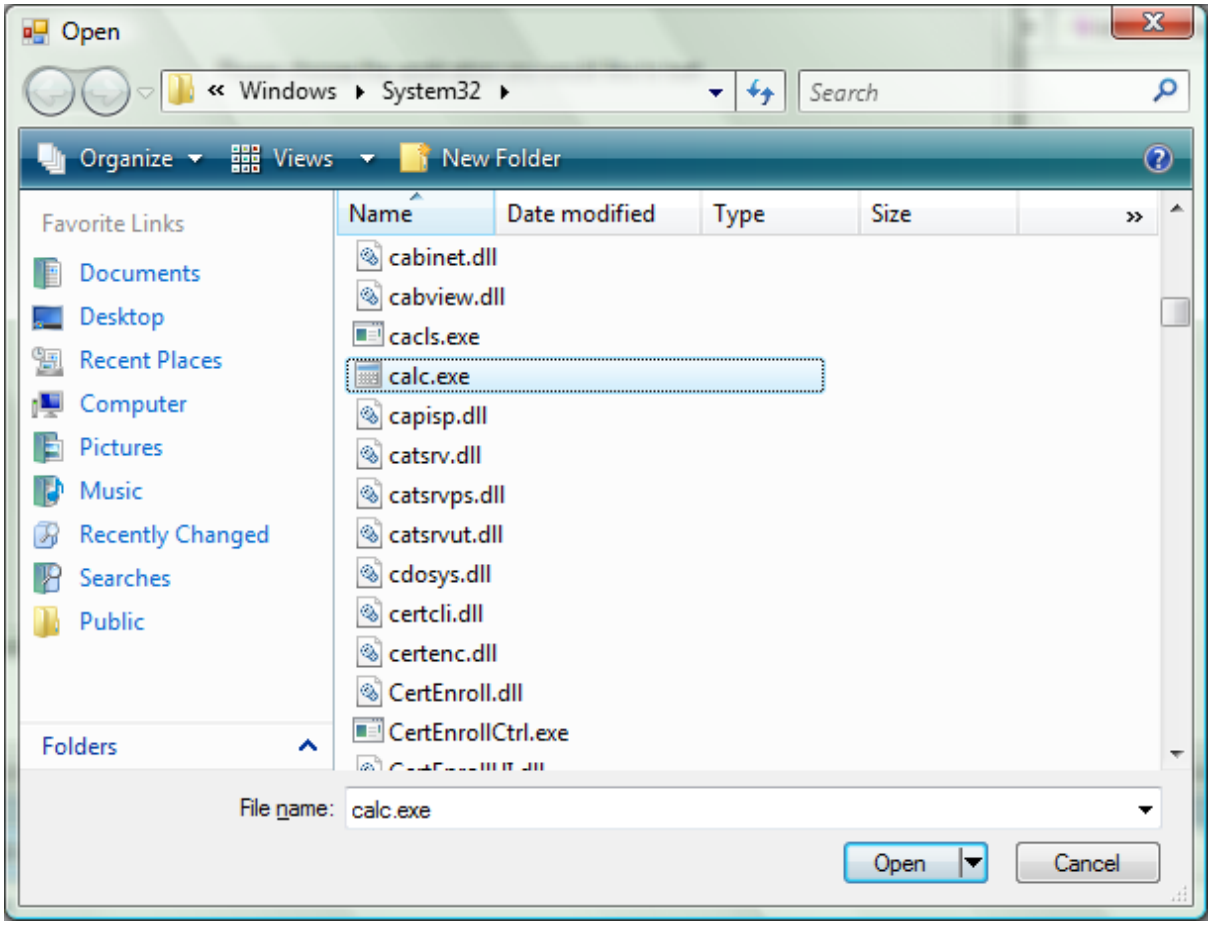
7.2.1 Πρώτη εκτέλεση

Το σύστημα αποτελείται από πολλές οθόνες, κατάλληλα οργανωμένες υπό μορφή Wizard. Στην πρώτη οθόνη του συστήματος, η φόρμα που εμφανίζεται ζητά από τον χρήστη να δώσει είσοδο το μονοπάτι (path) του υπό έλεγχο λογισμικού συστήματος όπως αυτό φαίνεται στο Σχ. 7.1. Η είσοδος αυτή θα χρησιμοποιηθεί στο στάδιο της εκτέλεσης των σεναρίων ελέγχου στην υπό έλεγχο εφαρμογή.



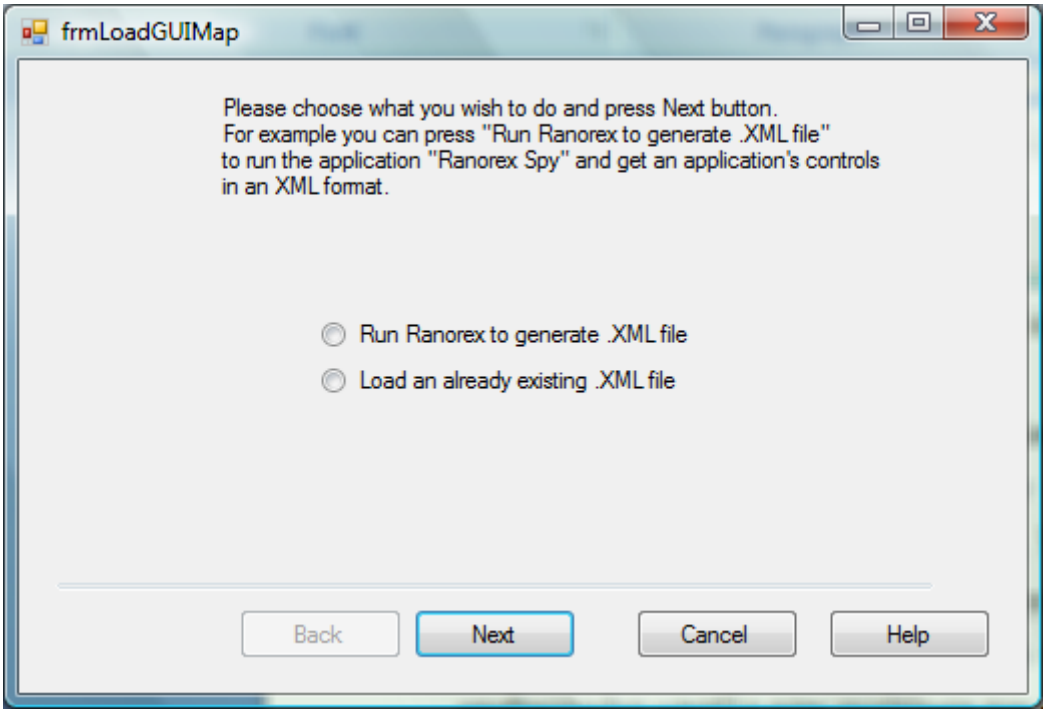
Σχήμα 7.1 Πρώτη οθόνη του συστήματος

Πατώντας το κουμπί «Browse…» ανοίγει ένα παράθυρο OpenFileDialog για επιλογή αρχείου. Αυτό φαίνεται στο Σχ. 7.2 πιο κάτω. Επιλέγοντας το εκτελέσιμο αρχείο και κάνοντας κλικ στην εντολή Ok, στο σύστημα αποθηκεύεται το μονοπάτι (path) του εκτελέσιμου αρχείου του υπό έλεγχο λογισμικού συστήματος.

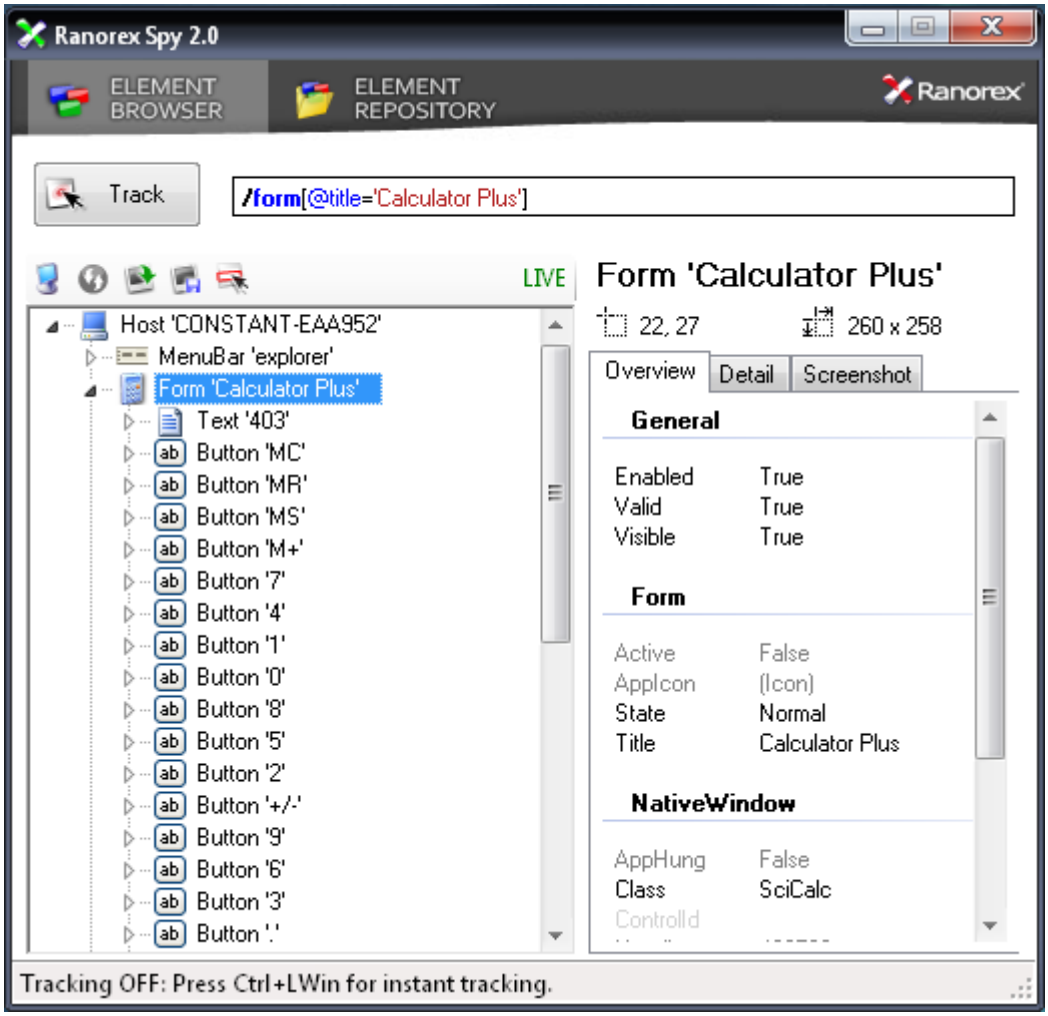


Σχήμα 7.2 Είσοδος εκτελέσιμου αρχείου στο σύστημα

Μετά την επιλογή του εκτελέσιμου αρχείου του υπό έλεγχο λογισμικού συστήματος και κάνοντας κλικ στην εντολή «Next» στην πρώτη οθόνη του συστήματος εμφανίζεται η δεύτερη οθόνη, όπως αυτή φαίνεται στο Σχ. 7.3. Εδώ ο χρήστης πρέπει να διαλέξει εάν θέλει να κάνει χρήση του εργαλείου Ranorex Spy (Σχ. 7.4) για να γίνει παραγωγή μοντέλου του γραφικού περιβάλλοντος του υπό έλεγχο λογισμικού συστήματος, ή εάν θέλει να φορτώσει ένα ήδη αποθηκευμένο μοντέλο στην περίπτωση που αυτό έχει δημιουργηθεί στο παρελθόν. Με την εντολή «Next» το σύστημα εκτελεί την επιλογή του χρήστη. Στην περίπτωση που ο χρήστης επιλέξει να γίνει εκτέλεση του εργαλείου Ranorex Spy, μετά την αποθήκευση του μοντέλου που θα δημιουργηθεί θα πρέπει να κάνει κλικ στην δεύτερη επιλογή, δηλαδή να φορτώσει το αποθηκευμένο αρχείο του μοντέλου.

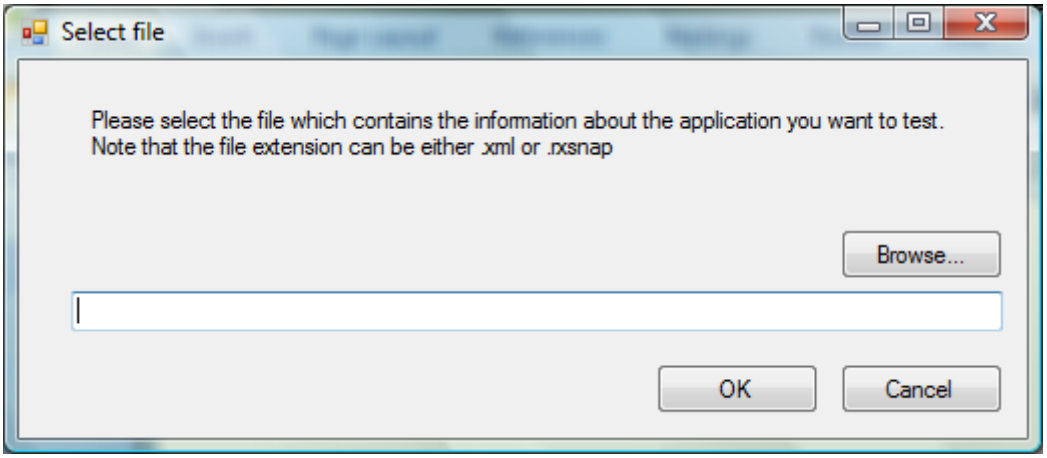


Σχήμα 7.3 Δεύτερη οθόνη του συστήματος



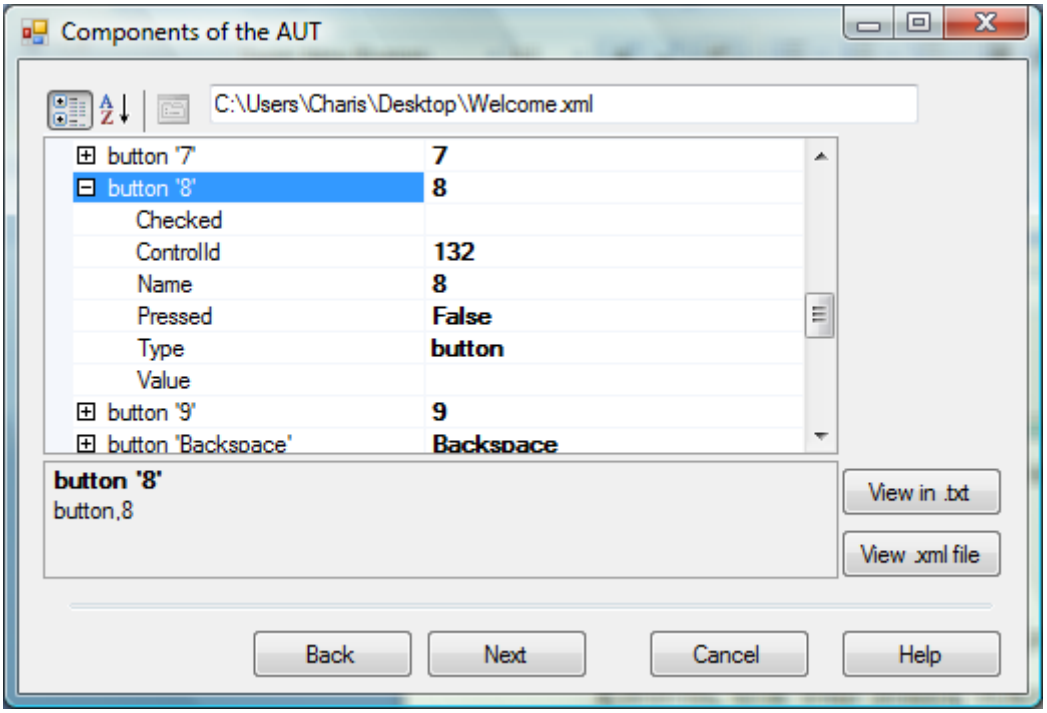
Σχήμα 7.4 Οθόνη του εργαλείου Ranorex Spy

Με την εντολή «Next» το σύστημα συνεχίζει τη λειτουργία του και ζητά από τον χρήστη να του δώσει ως είσοδο ένα αρχείο .xml ή .rxsnap (Σχ. 7.5). Τα αρχεία αυτά περιέχουν τα μοντέλα του γραφικού περιβάλλοντος του υπό έλεγχο λογισμικού συστήματος.



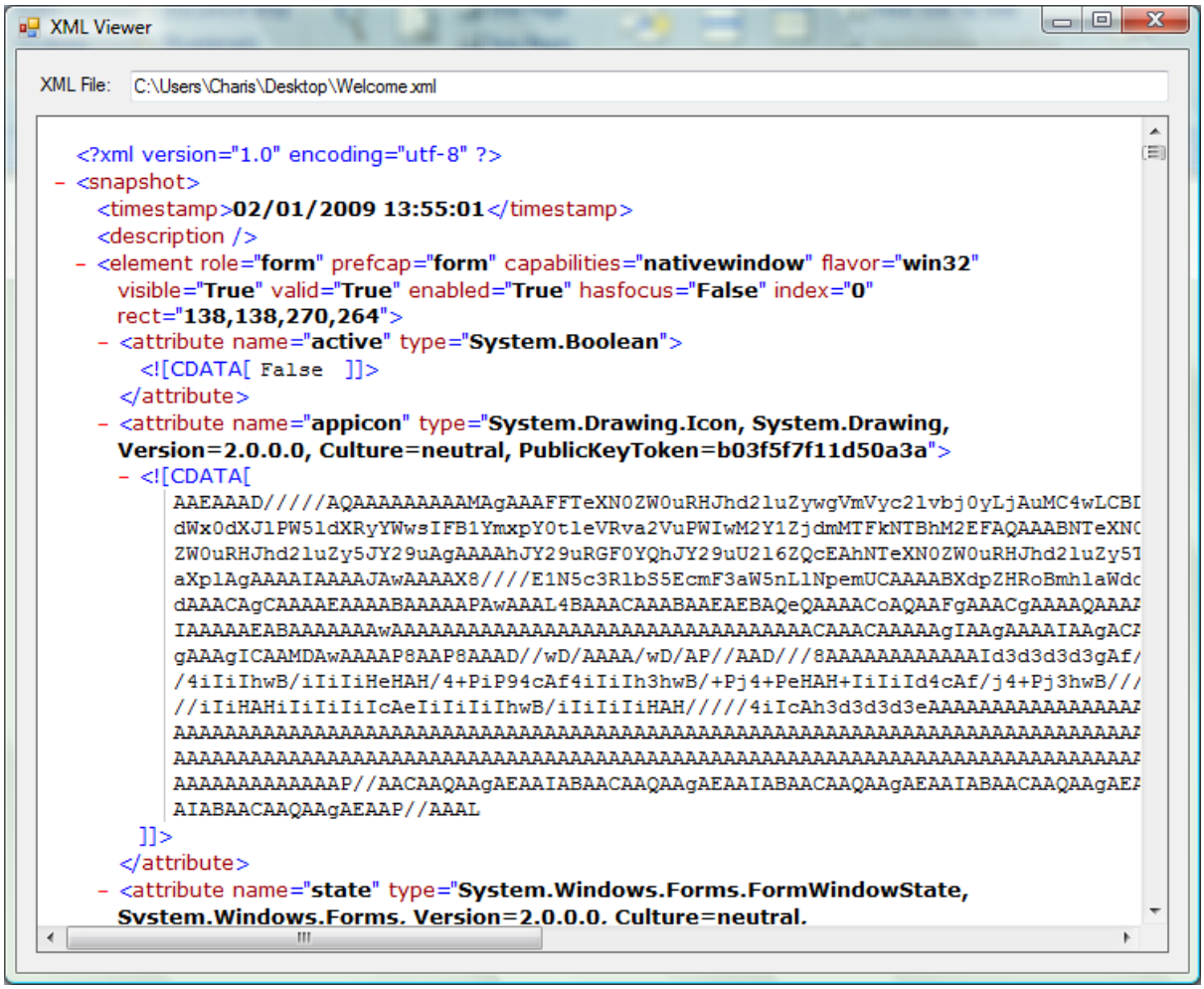
Σχήμα 7.5 Οθόνη εισόδου μοντέλου του γραφικού περιβάλλοντος του υπό έλεγχο λογισμικού συστήματος

Πατώντας «Browse…» εμφανίζεται οθόνη OpenFileDialog, όμοια με αυτή του Σχ. 7.2. Κάνοντας κλικ στην εντολή «Ok» το σύστημα θα εμφανίσει την τρίτη οθόνη (Σχ. 7.6) στην οποία παρουσιάζονται τα στοιχεία του υπό έλεγχο λογισμικού συστήματος, όπως αυτά εξάγονται από το αρχείο .xml ή .rxsnap και παρουσιάζονται στον χρήστη σε propertyGrid.



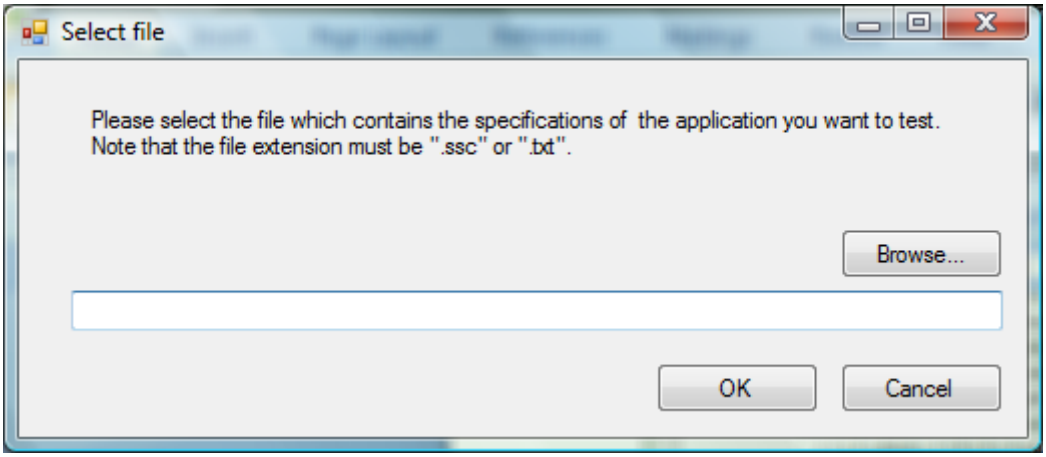
Σχήμα 7.6 Τρίτη οθόνη συστήματος

Παρατηρούμε ότι τα στοιχεία του γραφικού περιβάλλοντος καθώς και τα χαρακτηριστικά τους εμφανίζονται σε οργανωμένη και ευανάγνωστη μορφή. Με την εντολή «View in .txt» εμφανίζονται τα στοιχεία του γραφικού περιβάλλοντος σε απλή μορφή και με την εντολή «View .xml file» παρουσιάζεται το αρχείο .xml του οποίου η εισαγωγή έγινε στην προηγούμενη οθόνη. Αξίζει να σημειώσουμε πως για την παρουσίαση του αρχείου σε μορφή xml γίνεται χρήση μιας νέας φόρμας με web browser control έτσι ώστε η εμφάνιση του αρχείου αυτού να είναι ευανάγνωστη και καλύτερα οργανωμένη (Σχ.7.7).



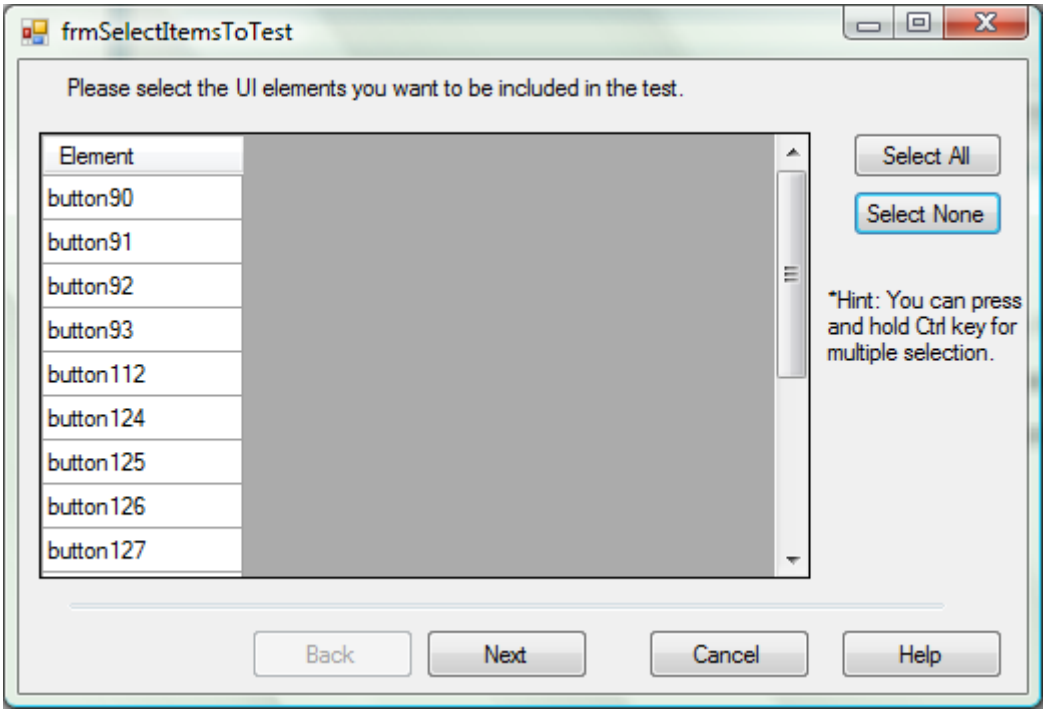
Σχήμα 7.7 Xml viewer

Κάνοντας κλικ στην εντολή «Next» το σύστημα ζητά την είσοδο αρχείου .ssc, (Σχ. 7.8) το οποίο περιέχει τις προδιαγραφές του υπό έλεγχο λογισμικού συστήματος.



Σχήμα 7.8 Είσοδος προδιαγραφών από αρχείο Spec#

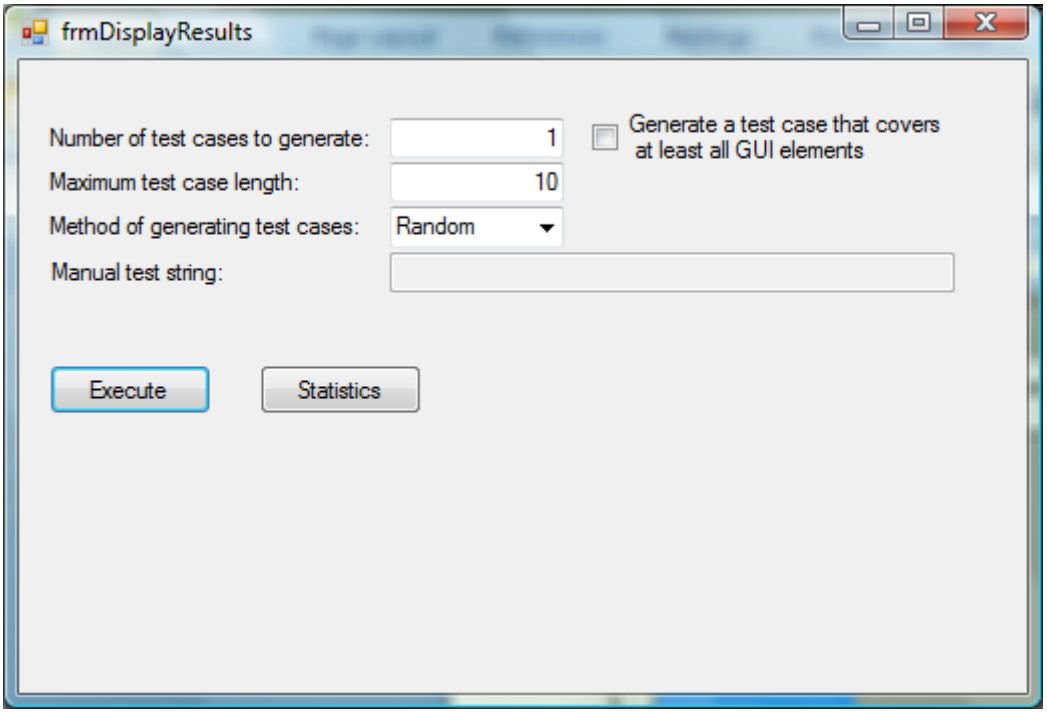
Με την είσοδο των προδιαγραφών το σύστημα παρουσιάζει στο χρήστη όλα τα στοιχεία του γραφικού περιβάλλοντος τα οποία περιέχονται στις προδιαγραφές και μπορούν να δεχθούν ελέγχους. (Σχ. 7.9) Εδώ ο χρήστης μπορεί να επιλέξει το σύνολο των στοιχείων ή ένα μικρότερο υποσύνολο το οποίο θα υποβληθεί σε έλεγχο.



Σχήμα 7.9 Τέταρτη οθόνη του συστήματος

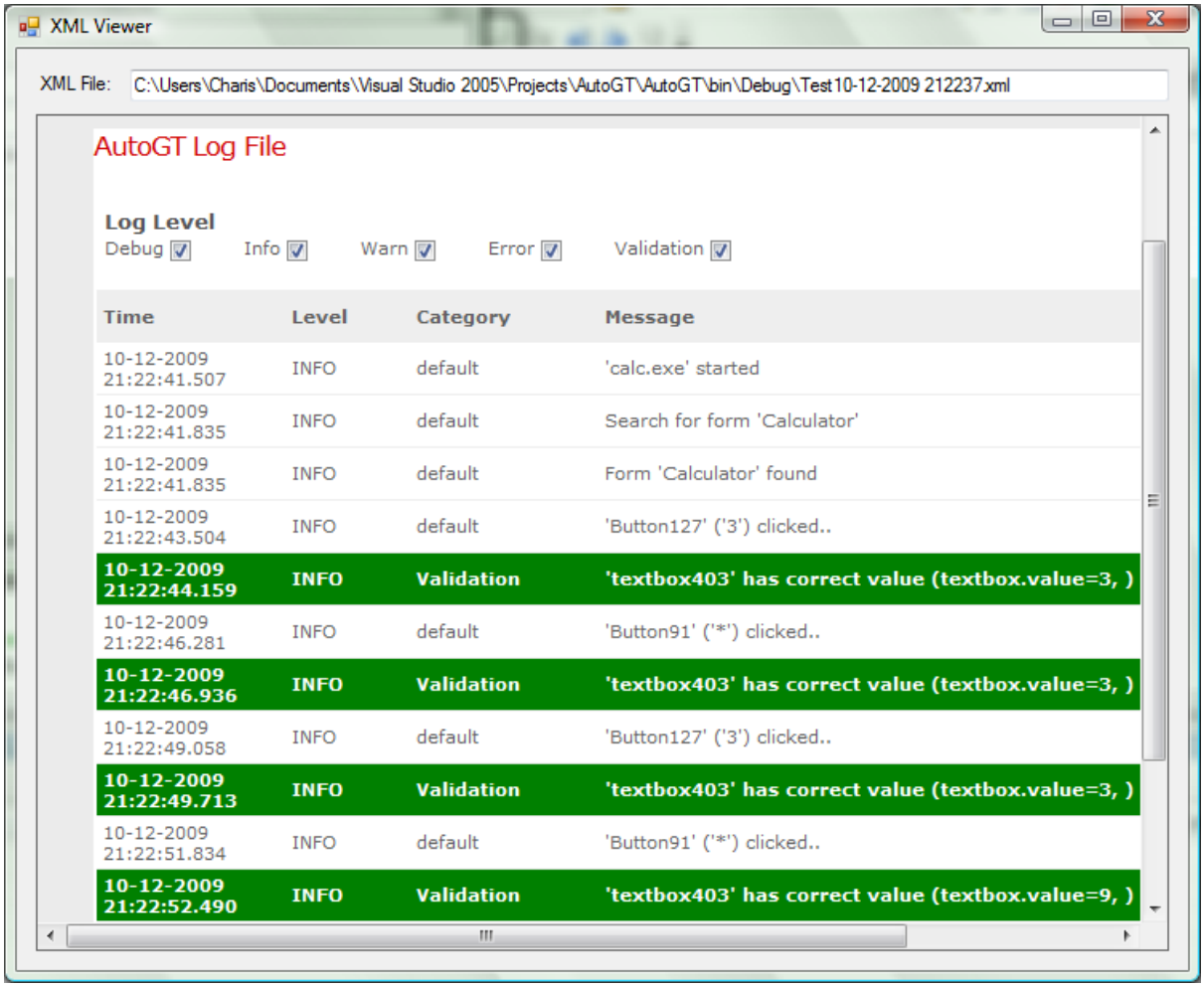
Με το πάτημα της εντολή «Select All» και «Select None» ο χρήστης μπορεί να επιλέξει όλα τα στοιχεία του γραφικού περιβάλλοντος ή κανένα αντίστοιχα. Αφού κάνει τις επιλογές του ο χρήστης θα πρέπει να κάνει κλικ στην εντολή «Next» ούτως ώστε να εμφανιστεί η τελευταία οθόνη (Σχ. 7.10) του συστήματος. Εδώ ο χρήστης μπορεί να επιλέξει ως μέθοδο παραγωγής σεναρίων ελέγχου την τυχαία παραγωγή ή την χειροκίνητη. Ακόμη μπορεί να επιλέξει τον

αριθμό των σεναρίων ελέγχου που θέλει να εκτελέσει και το μέγεθος του σεναρίου. Του δίνεται ακόμη η δυνατότητα παραγωγής σεναρίου ελέγχου ακανόνιστου μεγέθους με την προϋπόθεση ότι αυτό θα καλύψει όλα τα στοιχεία του γραφικού περιβάλλοντος τουλάχιστον μια φορά.



Σχήμα 7.10 Τέταρτη οθόνη του συστήματος

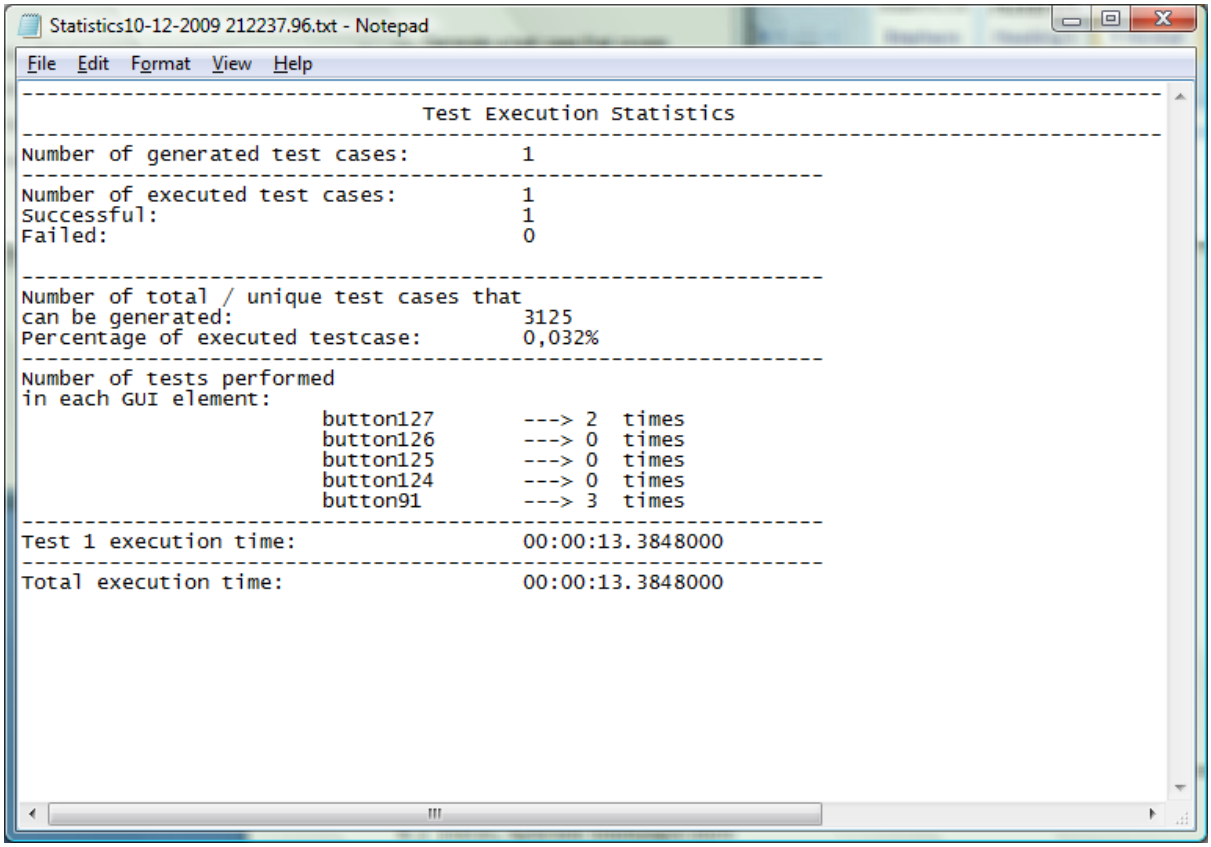
Αμέσως μετά την εκτέλεση των σεναρίων ελέγχου που παράχθηκαν, το σύστημα παρουσιάζει τα αποτελέσματα της εκτέλεσης.



Σχήμα 7.11 Παρουσίαση αποτελεσμάτων

Όπως φαίνεται από το Σχ. 7.11 πιο πάνω, η εκτέλεση ελέγχου δεν ανίχνευσε καμία διαφορά μεταξύ της υλοποίησης και των προδιαγραφών.

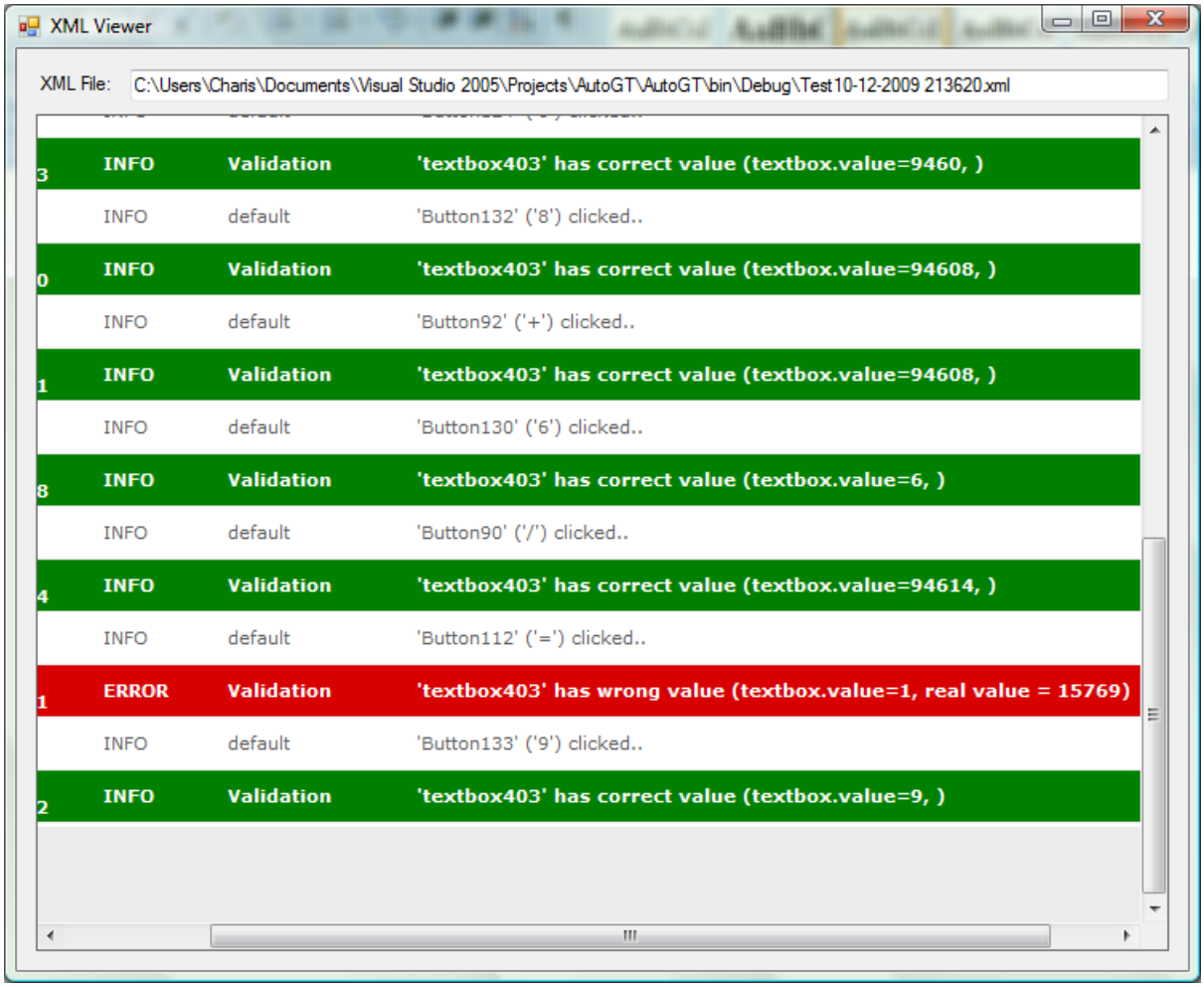
Αυτό φαίνεται και από τα στατιστικά που παράγει το σύστημα. Όπως φαίνεται πιο κάτω στο Σχ. 7.12 έγινε έλεγχος πέντε (5) στοιχείων του υπό έλεγχο λογισμικού συστήματος σε ένα σενάριο ελέγχου και δεν παρατηρήθηκε κανένα λάθος στην εκτέλεση. Ακόμη βλέπουμε πως το button127 πατήθηκε δύο (2) φορές, το button91 τρεις, ενώ τα button126, button125 και button124 δεν χρησιμοποιήθηκαν καθόλου στην διαδικασία ελέγχου. Στο κάτω μέρος παρατηρούμε ότι ο χρόνος που χρειάστηκε το σύστημα να διενεργήσει τον συγκεκριμένο έλεγχο είναι περίπου δεκατρία (13) δευτερόλεπτα.



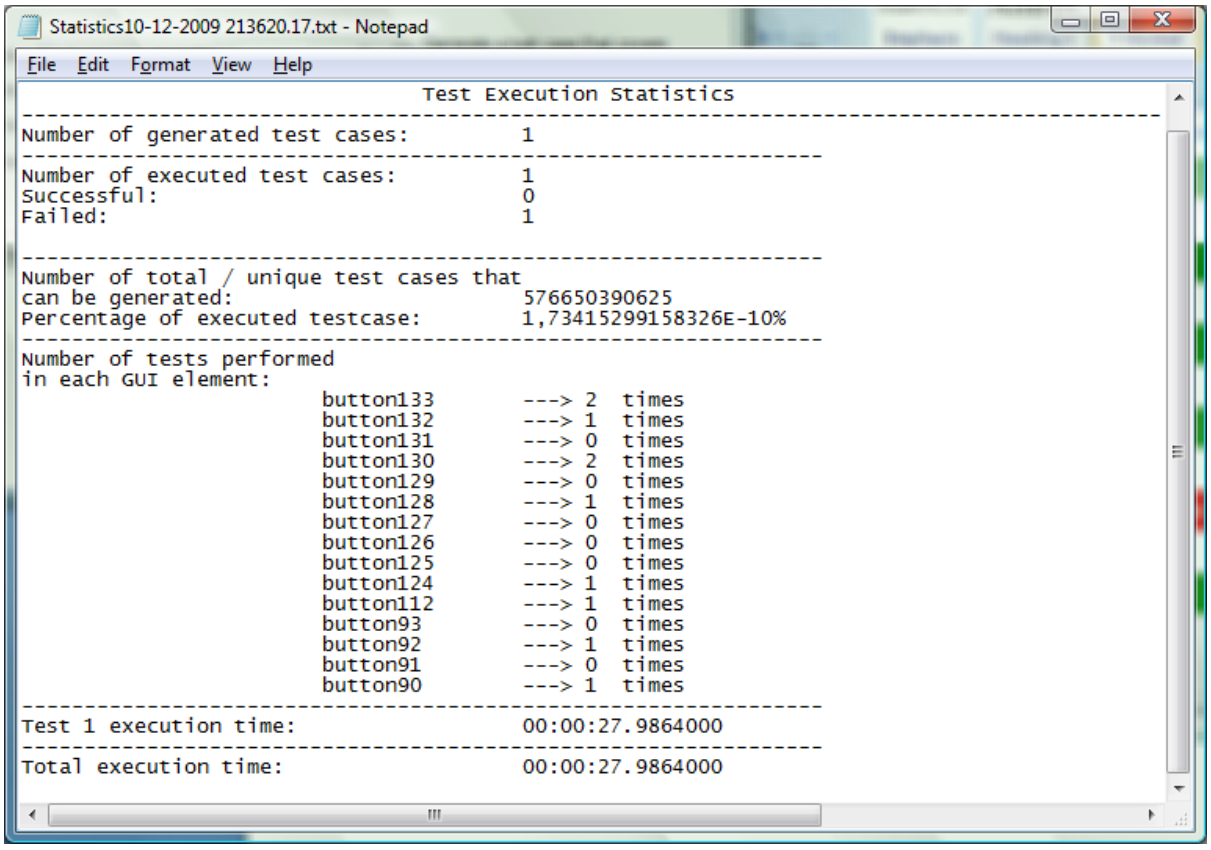
Σχήμα 7.12 Στατιστικά στοιχεία του ελέγχου

7.2.2 Δεύτερη εκτέλεση

Και σε αυτή την εκτέλεση το σύστημα ακολουθεί ακριβώς την ίδια διαδικασία. Η παραγωγή όμως διαφορετικών σεναρίων ελέγχου μας επέτρεψε να ανιχνεύσουμε ένα λάθος – διαφορά μεταξύ της υλοποίησης και του ορισμού των προδιαγραφών (Σχ. 7.13) Από το αποτέλεσμα της εκτέλεσης διαφαίνεται πως υπάρχει διαφορετική υλοποίηση της ακολουθίας /, = στο υπό έλεγχο λογισμικό σύστημα και στις προδιαγραφές.



Σχήμα 7.13 Παρουσίαση αποτελεσμάτων – Ανίχνευση λάθους



Σχήμα 7.14 Στατιστικά στοιχεία του ελέγχου – Ανίχνευση λάθους

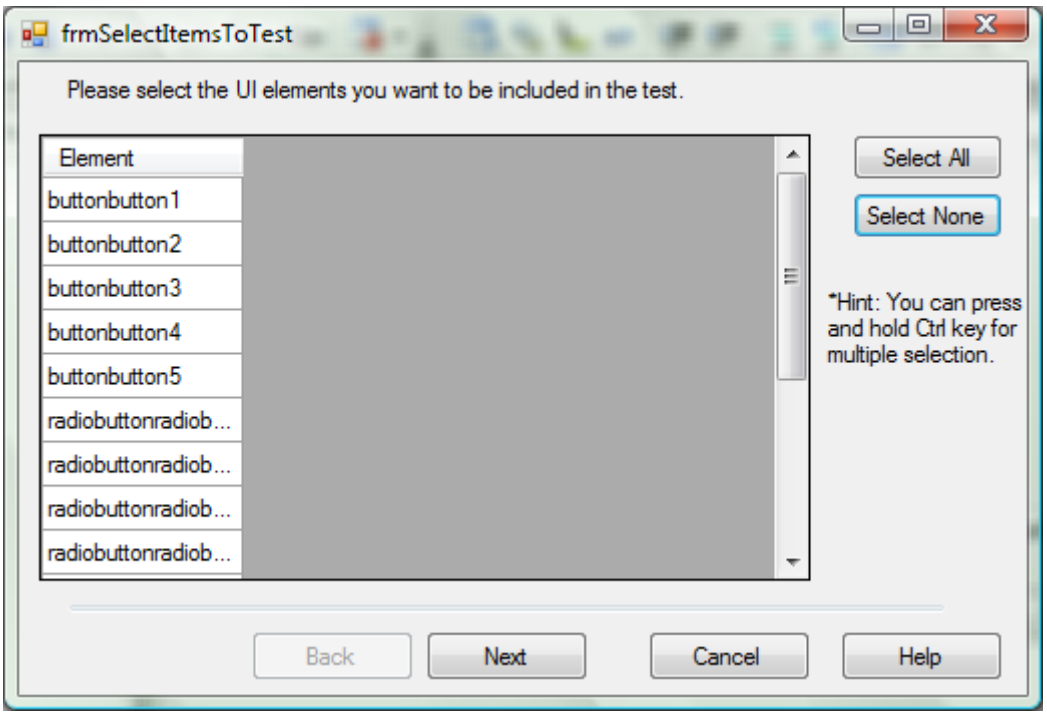
Στο Σχ. 7.14 μέσα από τα στατιστικά της εκτέλεση του ελέγχου παρουσιάζεται ακόμα μια φορά η διαφορά στην υλοποίηση και στις προδιαγραφές.

7.3. Εφαρμογή ελέγχου

Στην εκτέλεση αυτή το υπό έλεγχο λογισμικό σύστημα είναι μια απλή εφαρμογή χωρίς καμία λειτουργικότητα. Ο λόγος που θα παρουσιαστεί ο έλεγχος αυτής της εφαρμογής είναι για να αποδειχθεί ότι το σύστημα ελέγχου μπορεί να αναγνωρίσει και να ελέγξει συστήματα διαφορετικά από το Calculator, τα οποία χρησιμοποιούν .NET framework και διαθέτουν άλλα στοιχεία πέραν των buttons και των textboxes, δηλαδή radiobuttons και checkboxes.

Το σύστημα συμπεριφέρεται αρχικά όπως και στην πρώτη εκτέλεση (Σχ. 7.1 – 7.8) και για τον λόγο αυτό δεν θα παρουσιάσουμε ολόκληρη την διαδικασία. Αντίθετα, είναι αρκετό να παρουσιάσουμε την εκτέλεση από το σημείο μετά την είσοδο των προδιαγραφών.

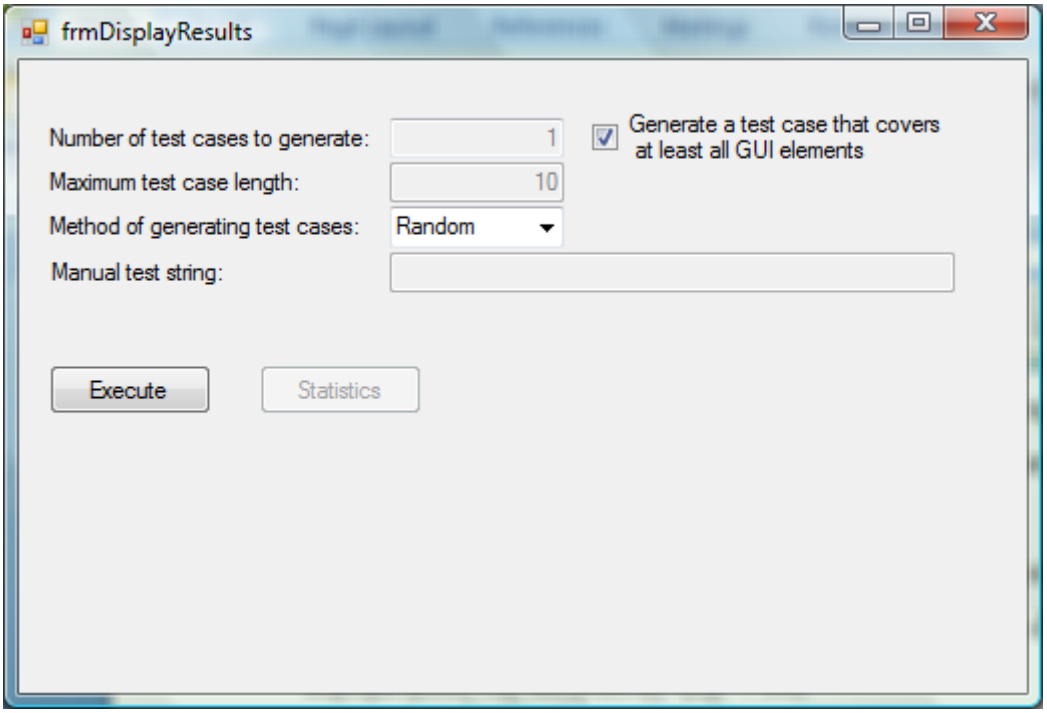
Έτσι, αφού γίνει η είσοδος των προδιαγραφών, παρουσιάζονται τα στοιχεία του γραφικού περιβάλλοντος στον χρήστη όπως στο Σχ. 7.15.



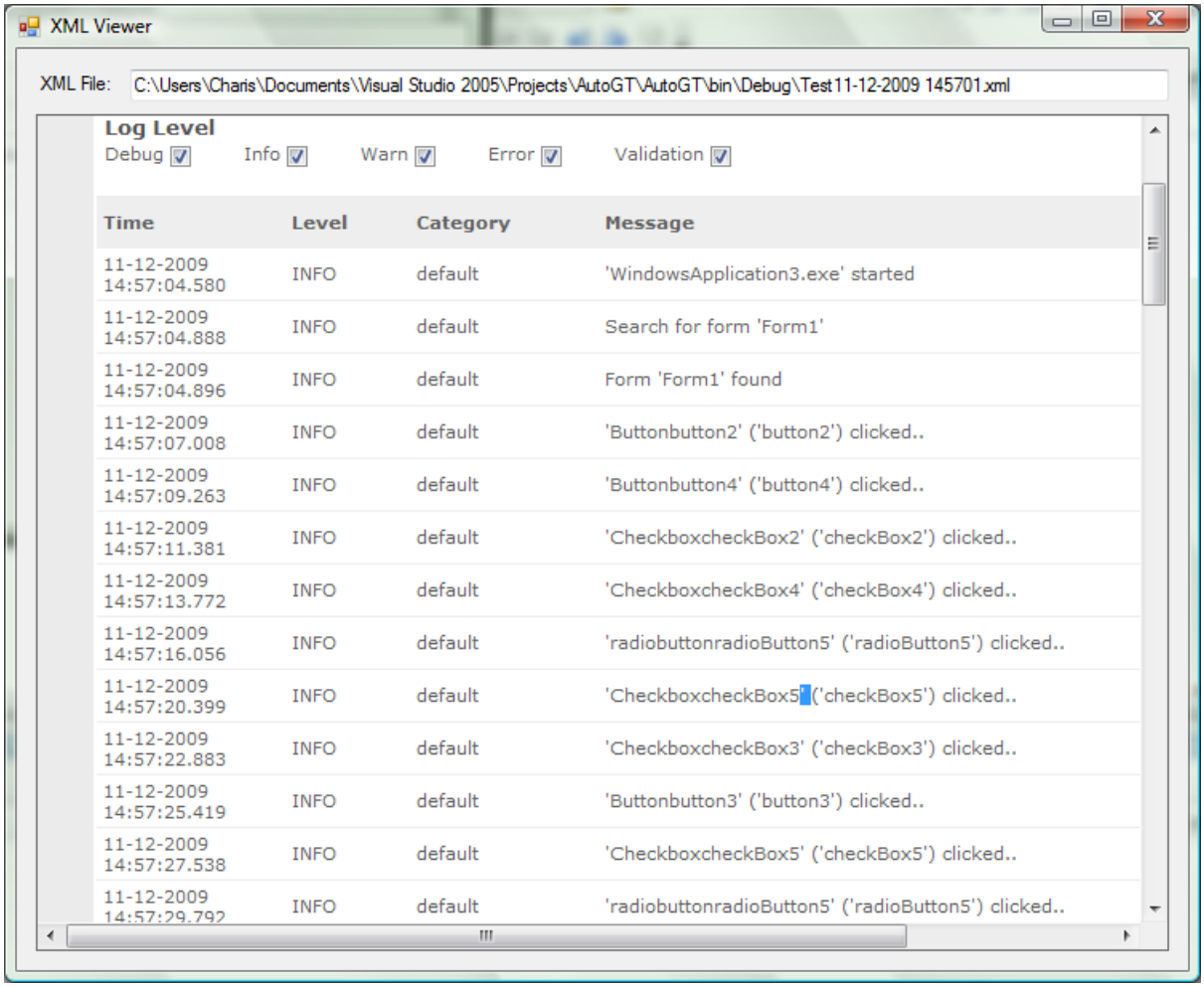
Σχήμα 7.15 Οθόνη παρουσίασης και επιλογής των στοιχείων του γραφικού περιβάλλοντος που περιέχονται στις προδιαγραφές

Αφού επιλέξουμε τα στοιχεία που θέλουμε, για σκοπούς ελέγχου και παρουσίασης θα επιλεχθούν όλα τα στοιχεία και θα κάνουμε κλικ στην εντολή «Next» έτσι ώστε να καθορίσουμε τις παραμέτρους της εκτέλεσης. (Σχ. 7.16)

Στην εκτέλεση αυτή θα επιλέξουμε να χρησιμοποιήσουμε την επιλογή που μας επιτρέπει να έχουμε ακανόνιστο μήκος στο σενάριο εκτέλεσης φτάνει να γίνει έλεγχος σε όλα τα στοιχεία του γραφικού περιβάλλοντος.

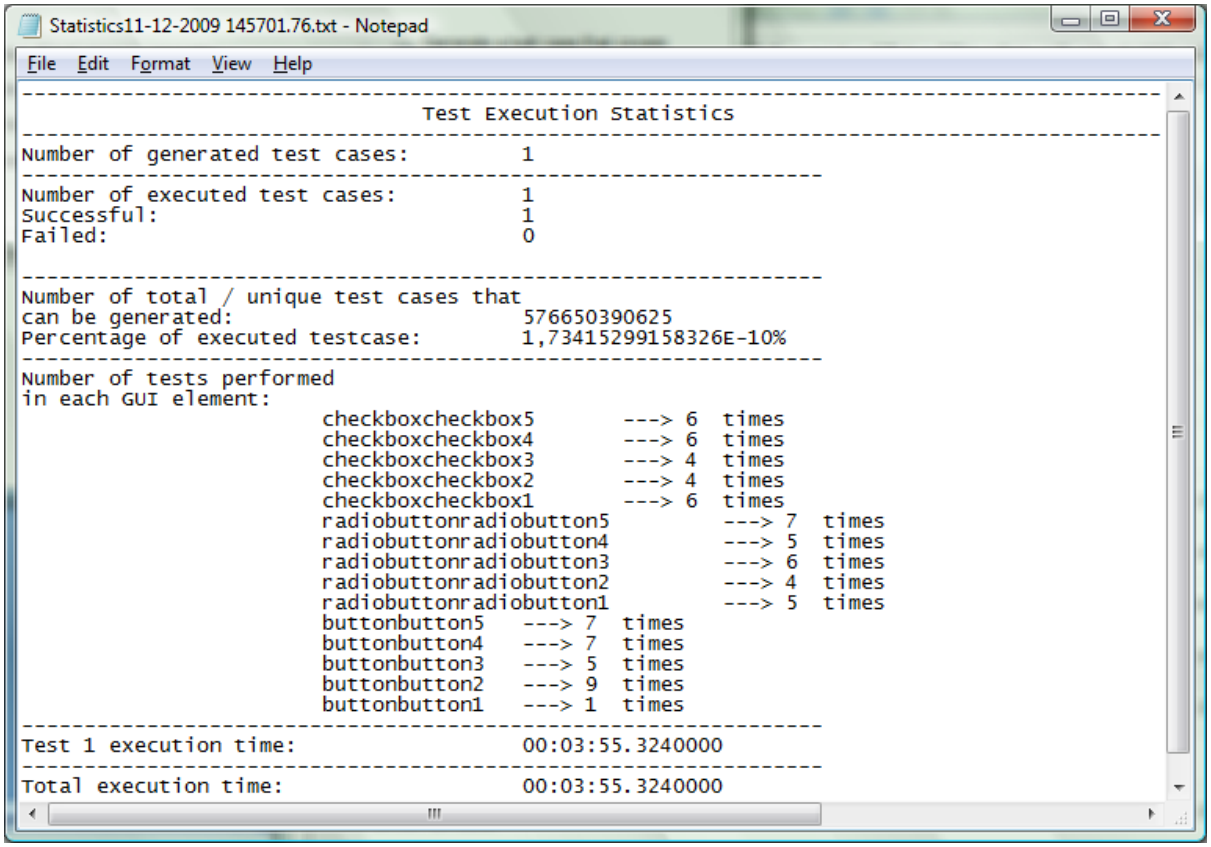


Σχήμα 7.16 Οθόνη καθορισμού των προδιαγραφών με επιλογή σεναρίου ελέγχου όλων των στοιχείων του γραφικού περιβάλλοντος.



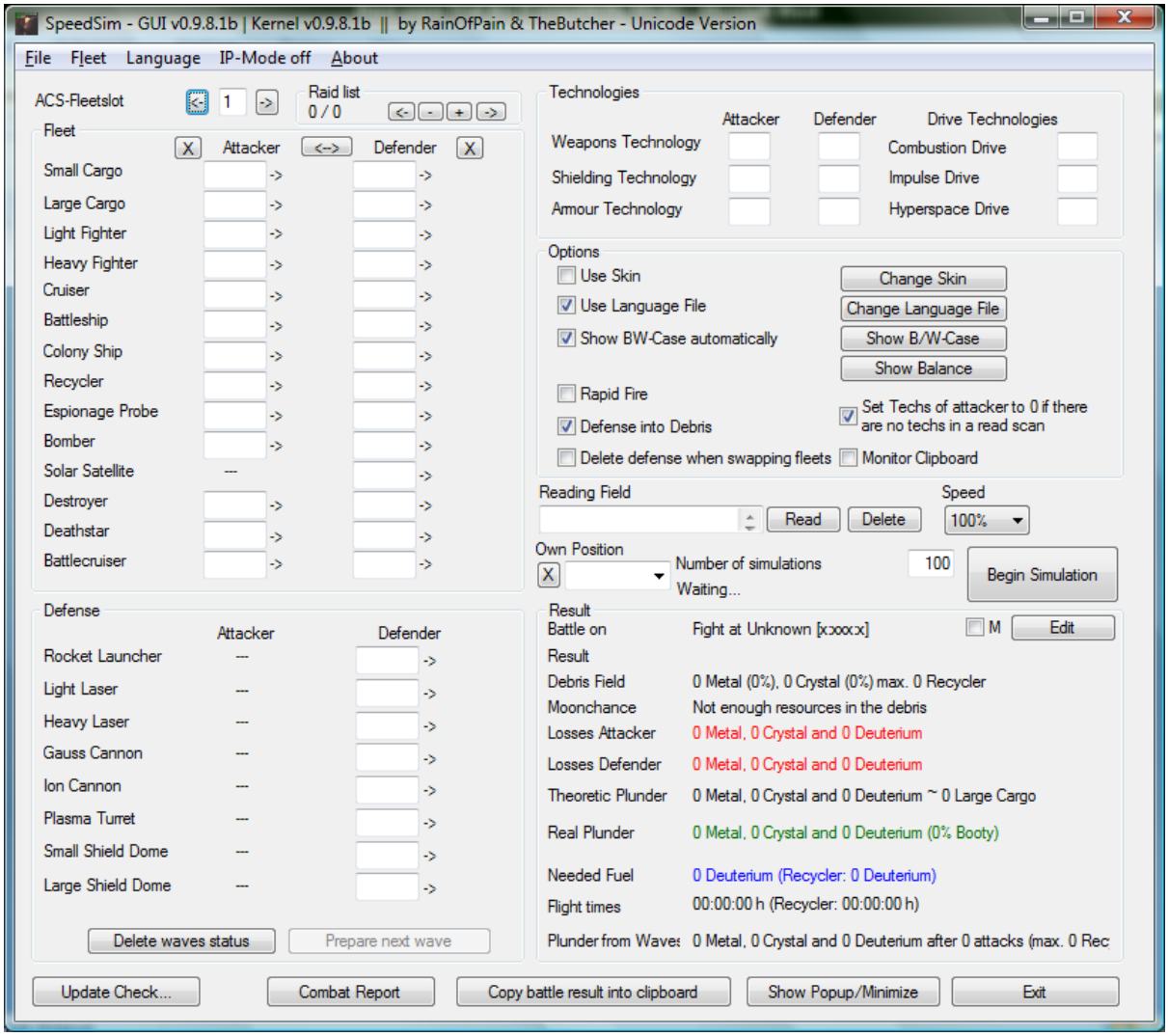
Σχήμα 7.17 Παρουσίαση αποτελεσμάτων ελέγχου

Όπως φαίνεται στο Σχ. 7.17 το σύστημα δεν ανίχνευσε καμία διαφορά μεταξύ προδιαγραφών και υλοποίησης. Αυτό φαίνεται και στο Σχ. 7.18 πιο κάτω στο οποίο παρουσιάζονται τα στατιστικά στοιχεία της εκτέλεσης του ελέγχου.



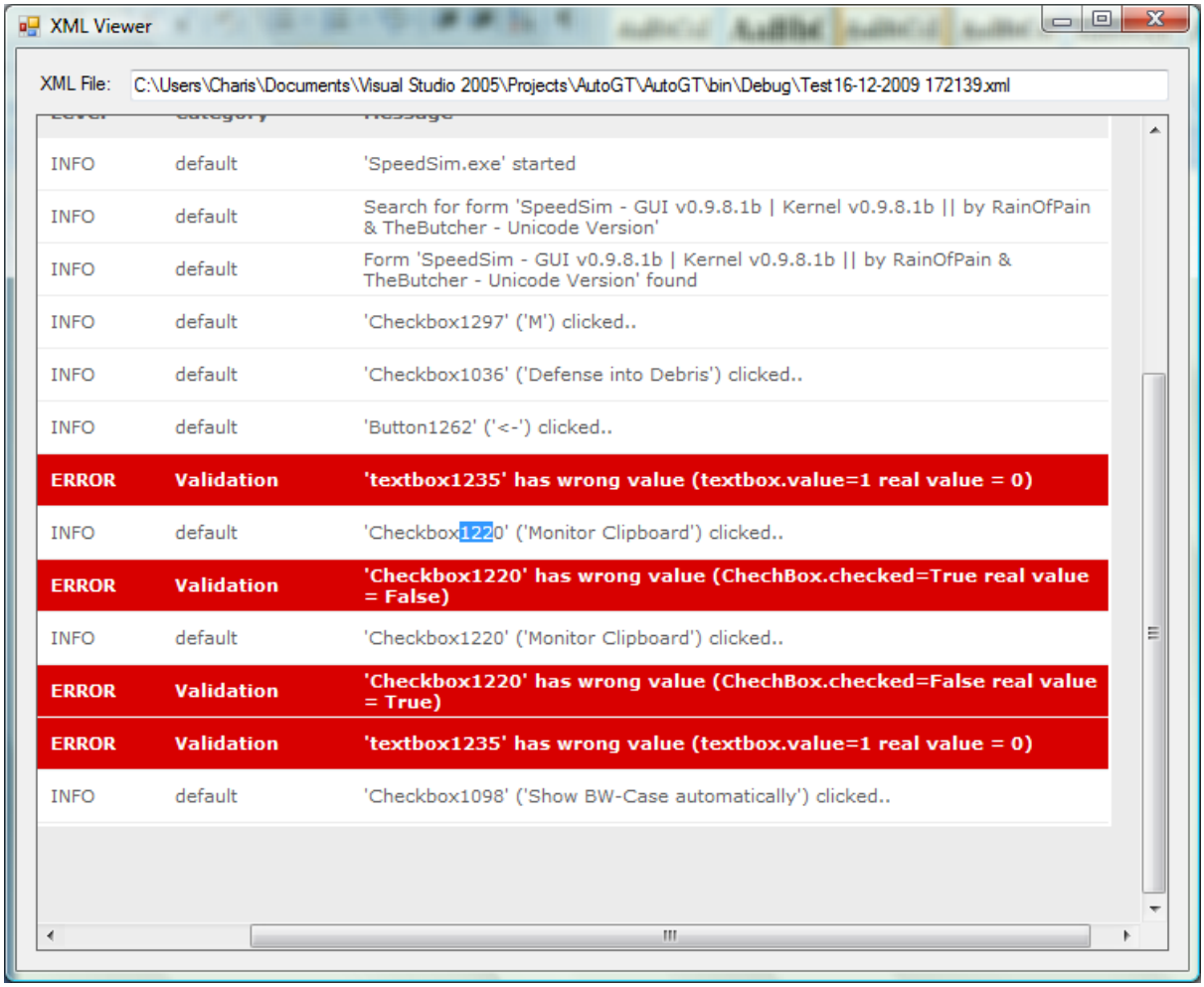
Σχήμα 7.18 Παρουσίαση στατιστικών στοιχείων της εκτέλεσης ελέγχου

7.4 Εφαρμογή SpeedSim

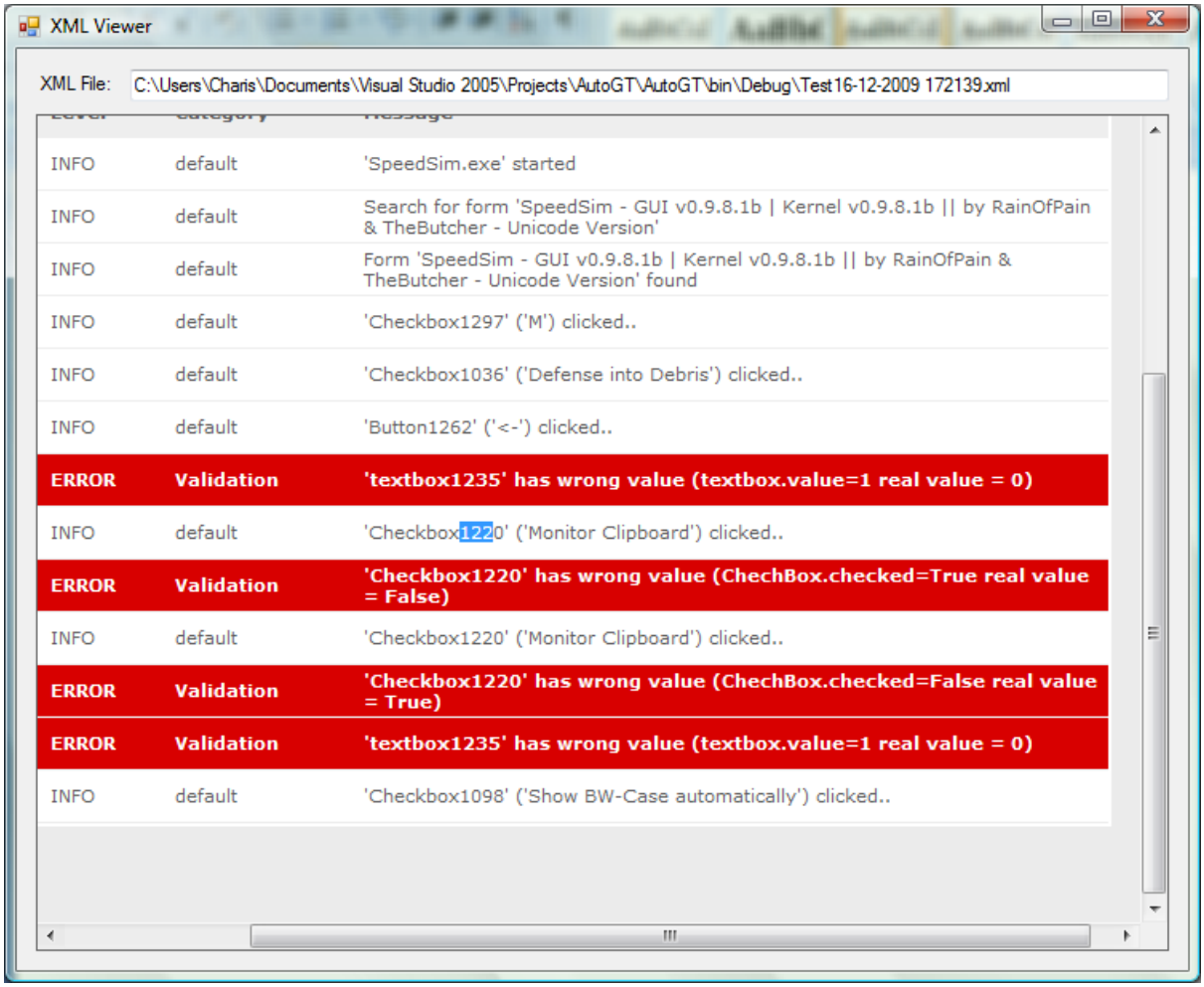


Οι διαδικασίες που ακολουθούνται και σε αυτό το παράδειγμα είναι οι ίδιες (Σχ. 7.1 – 7.8) με τα προηγούμενα παραδείγματα.

Η διαφορά βρίσκονται στο ότι πρόκειται για διαφορετική εφαρμογή. Στην περίπτωση της εφαρμογής αυτής που πάλι πρόκειται οι προδιαγραφές όσων αφορούν τα checkboxes είναι λανθασμένες και για τον λόγο αυτό ο έλεγχος δίνει λάθος αποτελέσματα σε αλληλεπιδράσεις που αφορούν αυτά. Ο έλεγχος δείχνει λάθος και στην περίπτωση που το σύστημα προσπαθεί να μειώσει την τιμή του textbox πέραν της τιμής 1. Στις προδιαγραφές φαίνεται ότι μπορεί να πάρει τιμή κάτω από το 1 ενώ στην υλοποίηση αυτό εμποδίζεται. Τα αποτελέσματα του ελέγχου φαίνονται στο Σχήμα 7.19.



Σχήμα 7.19 Παρουσίαση των αποτελεσμάτων ελέγχου



Σχήμα 7.20 Παρουσίαση των στατιστικών στοιχείων του ελέγχου

Κεφάλαιο 8

Πειραματική αξιολόγηση – Ανάλυση αποτελεσμάτων

8.1 Εισαγωγή

Για σκοπούς αξιολόγησης και εξαγωγής αποτελεσμάτων του συστήματος που υλοποιήθηκε έγινε μια σειρά δοκιμών με διαφορετικά σενάρια ελέγχου, εκ των οποίων ένα μικρό μέρος παρουσιάστηκε σε προηγούμενο κεφάλαιο.

8.2 Πειραματική αξιολόγηση

Έχει αποδειχτεί πως ο αυτόματος έλεγχος λογισμικών συστημάτων με γραφικό περιβάλλον είναι κατορθωτός με την χρήση των προδιαγραφών. Ολόκληρη η διαδικασία είναι αρκετά γρήγορη και δεν απαιτεί ιδιαίτερη προσπάθεια από την πλευρά του χρήστη. Όμως λόγω της απουσίας όμως των απαιτούμενων για τον έλεγχο προδιαγραφών, το σύστημα ελέγχθηκε σε περιορισμένο αριθμών εφαρμογών.

Παρόλα αυτά, μέσα από την πειραματική αξιολόγηση του συστήματος προέκυψε ότι μπορεί να υπάρξουν ασυνέπειες μεταξύ των προδιαγραφών και της υλοποίησης λόγω μη ορθότητας στην καταγραφή των προδιαγραφών. Αυτό συνέβηκε γιατί δεν υπήρχε καμία γνώση για την υλοποίηση της εφαρμογής πέραν από την περιδιάβαση σε αυτή. Το γεγονός όμως ότι οι προδιαγραφές δεν καταγράφτηκαν με βάση την υλοποίηση βοήθησε στην εξεύρεση διαφορών μεταξύ των δύο, που ήταν και το ζητούμενο.

Ακόμη προέκυψε ότι πρέπει να τηρηθούν αυστηρά οι κανόνες σημειογραφίας κατά την καταγραφή των απαιτήσεων που αφορούν την έξοδο σε αρχείο των διάφορων δεδομένων που θα προκύψουν από την εκτέλεση των προδιαγραφών. Σε αντίθετη περίπτωση το σύστημα δεν θα αντιδράσει σωστά. Με αυτό εννοείται ότι η τριάδα Pre-Conditions, Actions και Post-Conditions αποτελεί απαραίτητη προϋπόθεση σωστής εκτέλεσης των σεναρίων ελέγχου.

Το σύστημα μπορεί να ελέγξει λογισμικά συστήματα μέσω σεναρίων ελέγχου που είναι είτε μικρά είτε μεγάλα σε μήκος. Ακόμη δεν υπάρχει περιορισμός στον αριθμό των σεναρίων που θα εκτελεστούν παρά μόνο όταν γίνει κάλυψη όλων των δυνατών περιπτώσεων ελέγχου.

Ακόμη παρατηρήθηκε ότι λόγω του μεγάλου αριθμού στοιχείων του γραφικού περιβάλλοντος και του μεγάλου μήκους σεναρίων ελέγχου φαίνεται ότι ο έλεγχος όλων των περιπτώσεων είναι αδύνατο να εκτελεστεί με το χέρι. Παράδειγμα σε αυτό αποτελεί η περίπτωση εκτέλεσης σεναρίων ελέγχου με δεκαπέντε (15) στοιχεία γραφικού περιβάλλοντος και μήκος σεναρίων ελέγχου δέκα (10). Παρατηρήθηκε ότι για κάλυψη όλων των δυνατών περιπτώσεων εκτέλεσης, χρειάζονται πεντακόσια εβδομήντα έξι δισεκατομμύρια εξακόσια πενήντα εκατομμύρια τριακόσιες ενενήντα χιλιάδες εξακόσιες είκοσι πέντε (576.650.390625) διαφορετικές εκτελέσεις ή σενάρια ελέγχου, που αυτό εκ των πραγμάτων είναι ένας πραγματικά τεράστιος αριθμός περιπτώσεων ιδίως για ένα μικρό πρόγραμμα όπως είναι το Calculator της Microsoft.

Μια άλλη παρατήρηση που έγινε κατά την πειραματική αξιολόγηση είναι πως ο χρόνος εκτέλεσης κάποιου σεναρίου ελέγχου μεγαλώνει όσο μεγαλώνει ο αριθμός των Pre-Condition, Actions και Post-Conditions, γεγονός που είναι όμως λογικό αφού το σύστημα κάνει περισσότερους ελέγχους στα στοιχεία του γραφικού περιβάλλοντος. Είναι αποδεδειγμένο πειραματικά πως το σύστημα χρειάζεται περίπου 20 (είκοσι) δευτερόλεπτα για να εκτελέσει ένα σενάριο ελέγχου μήκους 10 (δέκα). Ο χρόνος αυτός είναι θεωρητικά μεγάλος αλλά περιορίζεται από τις βιβλιοθήκες που χρησιμοποιήθηκαν για την εκτέλεση των αλληλεπιδράσεων του συστήματος με το υπό έλεγχο λογισμικό σύστημα και πρακτικά δεν μπορεί να αλλάξει. Παραδείγματα χρόνου εκτέλεσης φαίνονται στο Σχ. 8.1.

Μήκος σεναρίου ελέγχου	Χρόνος (Μέσος όρος σε δευτερόλεπτα)
5	12.0690000
10	23.7580000
15	35.4380000
20	46.1120000
50	116.8540000

Σχήμα 8.1 Μέσος όρος εκτέλεσης ενός σεναρίου ελέγχου

Όσον αφορά τον χώρο που χρειάζεται για αποθήκευση των σεναρίων ελέγχου, των αποτελεσμάτων και των στατιστικών, το σύστημα δεν έχει ιδιαίτερες απαιτήσεις αφού ο

χώρος που καταναλώνει για αποθήκευση σεναρίων ελέγχου μήκους 50 (πενήντα) είναι μόλις 8 (οκτώ) KB για κάθε σενάριο, για την αποθήκευση των αποτελεσμάτων 15 (δεκαπέντε) KB και για τα στατιστικά μόλις 2 (δύο) KB. Σημειώνεται ενδεικτικά ότι για 150 (εκατό πενήντα) ελέγχους που διενεργήθηκαν χρειάστηκαν μόλις 900 (εννιακόσια) KB.

Κεφάλαιο 9

Γενικά συμπεράσματα και μελλοντική εργασία

9.1 Γενικά συμπεράσματα

Η παρούσα διπλωματική εργασία έχει παρουσιάσει ένα ολοκληρωμένο σύστημα για αυτόματο έλεγχο λογισμικών συστημάτων που διαθέτουν γραφικό περιβάλλον με βάση τις προδιαγραφές.

Το σύστημα αυτό δέχεται ως είσοδο το υπό έλεγχο σύστημα και χρησιμοποιεί το εργαλείο Ranorex Spy για να εξαγάγει τα συστατικά στοιχεία του υπό έλεγχο λογισμικού συστήματος και των χαρακτηριστικών τους, Στην συνέχεια τα καταγραμμένα σε μορφή XML συστατικά στοιχεία του γραφικού περιβάλλοντος του υπό έλεγχο συστήματος δίνονται ως είσοδος υπό μορφή αρχείου στο σύστημα ελέγχου που υλοποιήθηκε. Τότε το σύστημα αναλύει το αρχείο αυτό και με βάση τον αλγόριθμο που υλοποιήθηκε δημιουργεί μια λίστα στην οποία περιέχονται όλα τα συστατικά στοιχεία του γραφικού περιβάλλοντος και τα παρουσιάζει στον χρήστη κατάλληλα οργανωμένα και σε ευανάγνωστη μορφή.

Στη συνέχεια το σύστημα δέχεται ως είσοδο τις προδιαγραφές, καταγραμμένες σε γλώσσα Spec#, αναγνωρίζει τα συστατικά στοιχεία του γραφικού περιβάλλοντος του υπό έλεγχο λογισμικού συστήματος που βρίσκονται καταγραμμένα σε αυτές και τα παρουσιάζει στο χρήστη για να μπορέσει τότε αυτός με την σειρά του να κάνει επιλογή των στοιχείων εκείνων που θα υποστούν των έλεγχο.
Αφού ο χρήστης του συστήματος ελέγχου κάνει τις επιλογές του, το σύστημα ζητά από τον χρήστη να ορίσει τις παραμέτρους του ελέγχου που θα εκτελεστεί και αφού γίνει αυτό αρχίζει ο έλεγχος.

Τα αποτελέσματα του ελέγχου καθώς και στατιστικά στοιχεία που αφορούν τον έλεγχο παρουσιάζονται σε ευανάγνωστη μορφή από ένα αρχείο XML και ένα αρχείο TXT αντίστοιχα.

Ο χρήστης τότε μπορεί να αναλύσει τα αποτελέσματα του ελέγχου του υπό έλεγχο συστήματος. Τα αποτελέσματα αυτά παρέχουν στο χρήστη του συστήματος τις αναγκαίες κατευθυντήριες γραμμές ώστε να μπορέσει εύκολα να αναγνωρίσει τις διαφορές σε προδιαγραφές και υλοποίηση. Έτσι ολόκληρη η διαδικασία ελέγχου βελτιώνεται σημαντικά και παράλληλα το κόστος της σε χρόνο, κόπο και χρήμα παραμένει χαμηλό.

Κατά την διάρκεια της εκπόνησης της διπλωματικής εργασίας, έχουν αναγνωριστεί 3 (τρία) βασικά κομμάτια που ο ρόλος τους είναι κρίσιμος για την διεκπεραίωση του ελέγχου. Πρώτο, η σωστή παρουσίαση των συστατικών στοιχείων και των χαρακτηριστικών τους λαμβάνουν σημαντικότατο ρόλο, γιατί χωρίς την ύπαρξη αυτών δεν θα ήταν δυνατό να γίνει ο έλεγχος. Δεύτερο, η ύπαρξη ή η καταγραφή των τυπικών προδιαγραφών του υπό έλεγχο συστήματος με χρήση της γλώσσας Spec#. Λόγω του γεγονότος ότι οι καταγραμμένες προδιαγραφές με βάση την γλώσσα αυτή μπορούν να εκτελεστούν, μας δίνουν το πλεονέκτημα να κάνουμε αναπαράσταση της εκτέλεσης του υπό έλεγχο συστήματος λογισμικού και με βάση την εκτέλεση αυτή να δημιουργηθούν τα σενάρια ελέγχου. Τρίτο κρίσιμο στοιχείο είναι η υλοποίηση του Mapping Tool, το οποίο δέχεται ως είσοδο τα σενάρια ελέγχου, και με βάση τον αλγόριθμο του τα εκτελεί.

Παρόλο που το σύστημα αυτό παρουσιάστηκε ως σύστημα ελέγχου, η χρήση του μπορεί να επεκταθεί και για regression testing αφού είναι αποδοτικό και για τέτοια χρήση.

Το σύστημα που υλοποιήθηκε αν και είναι ολοκληρωμένο και λειτουργεί ορθά στα σενάρια που ελέγχθηκε, περιέχει κάποιους περιορισμούς:

1. Δεν είναι γενικευμένο και η χρήση του προορίζεται μόνο σε εφαρμογές Windows

2. Δεν μπορεί να διενεργήσει έλεγχο σε εξόδους ή αλληλεπιδράσεις του συστήματος με συσκευές πέραν της οθόνης, δηλαδή ήχο, εκτυπωτές και άλλα.

3. Χειρίζεται εφαρμογές στις οποίες τα συστατικά στοιχεία του γραφικού περιβάλλοντος παρουσιάζονται μόνο στην πρώτη οθόνη του υπό έλεγχο συστήματος λογισμικού.

4. Τα συστατικά στοιχεία που μπορεί να χειριστεί περιορίζονται μόνο σε buttons, checkboxes, radiobuttons και textboxes.

Συνοψίζοντας και δεδομένου του γεγονότος ότι τα αποτελέσματα είναι τόσο θετικά όσο και ενθαρρυντικά η παρούσα διερευνητική Διπλωματική Εργασία καθώς και το σύστημα που υλοποιήθηκε αποτελούν μια πολύ καλή βάση για επέκταση και υλοποίηση ενός ολοκληρωμένου και ανεξάρτητου συστήματος αυτόματου ελέγχου του γραφικού περιβάλλοντος και σε μεγαλύτερα ακόμη λογισμικά συστήματα με βάση τις προδιαγραφές.

9.2 Μελλοντική εργασία

Λόγω του γεγονότος ότι το θέμα της Διπλωματικής αυτής Εργασίας είναι επίκαιρο και σημαντικό για τον κλάδο της Πληροφορικής προορίζεται να συνεχιστεί ώστε να εξαλειφθούν όλοι οι περιορισμοί και οι ελλείψεις του.

Η μελλοντική εργασία θα πρέπει να ασχοληθεί με την πρόσθεση στο σύστημα όλων των υπολοίπων στοιχείων που πιθανό να αποτελούν ένα γραφικό περιβάλλον. Η εργασία αυτή δεν είναι δύσκολη γιατί στην υλοποίηση που έγινε υπάρχουν οι βάσεις για την πρόσθεση των υπολοίπων συστατικών.

Ακόμη θα πρέπει να επεκταθεί το σύστημα ώστε να μπορεί να αναγνωρίζει και τις υπόλοιπες φόρμες που αποτελούν ένα σύστημα λογισμικού και όχι απλώς την πρώτη. Ένας πιθανός τρόπος υλοποίησης αυτής της λειτουργίας είναι με την συσχέτιση αρχείων μοντελοποίησης του γραφικού περιβάλλοντος με τις φόρμες του λογισμικού συστήματος, ούτως ώστε να μπορεί το σύστημα να αναγνωρίζει ορθά και να ελέγχει τα στοιχεία των υπολοίπων φορμών του υπό έλεγχο συστήματος λογισμικού.

Μια άλλη επέκταση του συστήματος που θα πρέπει να υλοποιηθεί είναι η αυτοματοποίηση της διαδικασίας καταγραφής των προδιαγραφών σε γλώσσα Spec#. Η διαδικασία αυτή αποτελεί την μοναδική χειρονακτική εργασία που εκτελεί ο χρήστης του υφιστάμενου συστήματος. Η καταγραφή των προδιαγραφών, ακολουθώντας τους κανόνες που ορίζονται από την Τεχνολογία Λογισμικού, γίνεται κατά το δεύτερο στάδιο του κύκλου ζωής. Όμως στην περίπτωση που η καταγραφή των προδιαγραφών γίνει σε άλλη γλώσσα μοντελοποίησης πέραν της Spec#, π.χ. σε UML, το σύστημα θα πρέπει να μπορεί να μεταφράσει τις προδιαγραφές στην γλώσσα μοντελοποίησης που αναγνωρίζεται από το σύστημα.

Σημαντικό είναι να προστεθούν γενετικοί αλγόριθμοι ή νευρωνικά δίκτυα στην υλοποίηση στο στάδιο παραγωγής σεναρίων ελέγχου. Αυτό θα επιτρέψει την σημαντική μείωση του αριθμού των σεναρίων που παράγονται από το σύστημα έχοντας πάντα υπόψη ότι η χρήση τέτοιων αλγόριθμων και ιδίως των γενετικών δίνουν τις περισσότερες φορές πολύ καλά αποτελέσματα.

Ακόμη η πρόσθεση δυνατότητας ελέγχου εφαρμογών διαδικτύου και ιστοσελίδων είναι μια λειτουργικότητα που θα μπορούσε να προστεθεί εάν και εφόσον κριθεί αυτό αναγκαίο. Η γνώμη μου είναι πως ο ρόλος του διαδικτύου είναι πολύ σημαντικός στην σημερινή εποχή και βλέπουμε συνεχώς την δημιουργία online εφαρμογών οι οποίες χρησιμοποιούνται από τους χρήστες απευθείας από το διαδίκτυο χωρίς να γίνει οποιαδήποτε εγκατάσταση λογισμικού συστήματος στον υπολογιστή τους. Για τον λόγο αυτό πιστεύω πως η ύπαρξη τέτοιων δυνατοτήτων ελέγχου επιβάλλεται.

Βιβλιογραφία

[1] Moez Krichen and Stavros Tripakis, “Black-Box Conformance Testing for Real-Time Systems”, Springer-Verlag Berlin Heidelberg, pp.109-126, 2004

[2] Luay H. Tahat, Boris Vaysburg, Bogdan Korel and Atef J. Bader, “Requirement-Based Automated Black-Box Test Generation”, IEEE, 2001

[3] White Box Testing, 2009, http://www.webopedia.com/TERM/W/White_Box_ Testing.html

[4] What is gray box testing, 2009, http://www.robdavispe.com/free2/software-qa-testing-test-tester-2210.html

[5] Capture – Replay Tool, 2009, http://www.soft.com

[6] CompuWare TestPartner, 2009, http://web.umit.com/content/brochure/TestPartner_ Fact_Sheet.pdf

[7] Rational Robot, 2009, http://www-01.ibm.com/software/awdtools/tester/robot/

[8] Rational Visual Test, 2009, http://www-01.ibm.com/software/awdtools/visualtest/ support/index.html

[9] WinRunner, 2009, http://www.fdscallcentre.com.tw/homepage/data/mi/DS-0414-0203_WR7.6.pdf

[10] LoadRunner, 2009, http://www.fdscallcentre.com.tw/homepage/data/mi/DS-0399-0203_LR.pdf

[11] Abbot, 2009, http://abbot.sourceforge.net/doc/overview.shtml

[12] Guitar, 2009, http://guitar.sourceforge.net/

[13] WIMP, 2009, http://en.wikipedia.org/wiki/WIMP_(computing)

[14] Ranorex, 2009, http://www.ranorex.com

[15] XML, 2009, http://www.w3.org/XML/

[16] Lionel Briand, Yvan Labiche, "A UML-Based Approach to System Testing" pp. 10-42, Springer – Verlag, 2002

[17] Y.G. Kim, H.S. Hong, S.M. Cho, D.H. Bae and S.D. Cha, "Test Cases Generation from UML State Diagrams", IEEE Proceedings – Software, Vol. 146, No.4 pp. 187-192, 1999

[18] Jeff Offutt and Aynur Abdurazik, "Generating Tests from UML Specifications", pp. 416 – 429, Springer – Verlag, 1999

[19] Mike Barnett, Wolfgang Grieskamp, Lev Nachmanson, Wolfram Schulte, Nikolai Tillmann and Magus Veanes, "Model-Based Testing with AsmL .Net", Microsoft Research, 2003

[20] AsmL Reference Manual, 2009, http://www.codeplex.com/AsmL

[21] Colin Campbell, Wolfgang Grieskamp, Lev Nachmanson, Wolfram Schulte, Nikolai Tillman and Margus Veanes, "Testing Concurrent Object-Oriented Systems with Spec Explorer", pp.523 – 547, Springer – Verlag, 2005

[22] Spec Explorer Reference Manual, 2009, http://research.microsoft.com/en-us/projects/SpecExplorer/

[23] Ana, C.R. Paiva, joao C.P. Faria, Nikolai Tillmann and Raul A.M. Vidal, "A Model-to-Implementation Mapping Tool for Automated Model-Based GUI Testing", pp. 450-464, Springer – Verlag, 2005

[24] Atif Memon, Adithya Nagarajan and Qing Xie, "Automated Regression Testing for Evolving GUI Software", Proceedings of the International Conference on Software Maintenance, 2003

[25] Atif Memon, Ishan Banerjee and Adithya Nagarajan, “GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing”, Proceedings of the 10th Working Conference on Reverse Engineering, 2003