ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Περιβάλλον Πλαίσιο για το Στατικό και Δυναμικό Έλεγχο Συστημάτων Λογισμικού με τη βοήθεια Μοντέλων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

των

Ξενούλα Λ. Κάκκαρου
Όλγα - Χαρά Σ. Καραμπέρη

Επιβλέπων : Κωνσταντίνος Κοντογιάννης
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2015

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Περιβάλλον Πλαίσιο για το Στατικό και Δυναμικό Έλεγχο Συστημάτων Λογισμικού με τη βοήθεια Μοντέλων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

των

Ξενούλα Λ. Κάκκαρου
Όλγα - Χαρά Σ. Καραμπέρη

Επιβλέπων : Κωνσταντίνος Κοντογιάννης
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 28η Σεπτεμβρίου 2015.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Κωνσταντίνος Κοντογιάννης Αναπλ. Καθηγητής Ε.Μ.Π.	Ιωάννης Βασιλείου Καθηγητής Ε.Μ.Π.	Στέφανος Κόλλιας Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2015

(Υπογραφή) *(Υπογραφή)*

....

Ξενούλα Λ. Κάκκαρου **Όλγα-Χαρά Σ. Καραμπέρη**

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε κατά το ακαδημαϊκό έτος 2014-2015 στον τομέα Τεχνολογίας πληροφορικής και υπολογιστών της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου.

Υπεύθυνος κατά την εκπόνηση της διπλωματικής ήταν ο Αναπληρωτής Καθηγητής κ. Κωνσταντίνος Κοντογιάννης στον οποίο οφείλουμε ιδιαίτερες ευχαριστίες για την ανάθεση αυτής ,τη δυνατότητα που μας δόθηκε να ασχοληθούμε με ένα τόσο ενδιαφέρον θέμα καθώς και για την υποστήριξη και την καθοδήγηση που μας παρείχε κατά τη συγγραφή της εργασίας. Ακόμα θα θέλαμε να ευχαριστήσουμε τον Καθηγητή κ. Ιωάννη Βασιλείου και τον Καθηγητή κ. Στέφανο Κόλλια για την τιμή που μας έκαναν να συμμετάσχουν στην επιτροπή εξέτασης της εργασίας.

Κυρίως όμως θέλουμε να εκφράσουμε την ευγνωμοσύνη μας στις οικογένειές μας για την αμέριστη στήριξή τους όλα αυτά τα χρόνια ώστε να υλοποιήσουμε τους επιδιωκόμενους στόχους μας. Τέλος θα ήταν παράλειψη να μην ευχαριστήσουμε όλους τους δικούς μας ανθρώπους για την διάθεση και την κατανόηση που επέδειξαν κατά τη διάρκεια της πορείας μας.

Ξενούλα Λ. Κάκκαρου
Όλγα Χαρά Σ. Καραμπέρη

Περίληψη

Στόχος της παρούσας διπλωματικής εργασίας είναι ο σχεδιασμός και η υλοποίηση ενός περιβάλλοντος-πλαισίου που επιτρέπει τον στατικό και δυναμικό έλεγχο συστημάτων λογισμικού με τη βοήθεια μοντέλων στόχων. Η διαχείριση έργων ανάπτυξης λογισμικού αποτελεί μια σύνθετη διαδικασία με κύριο στόχο της την εξασφάλιση ότι το τελικό σύστημα λογισμικού θα παραδοθεί εγκαίρως, θα ικανοποιεί τις ανάγκες των τελικών χρηστών, θα λαμβάνει υπ' όψιν τις απαιτήσεις ποιότητας και ταυτόχρονα θα διατηρεί το κόστος στα προβλεπόμενα όρια. Στην ανάπτυξη πολυσύνθετων συστημάτων λογισμικού ο περισσότερος χρόνος και η μεγαλύτερη προσπάθεια καταναλώνονται στην επαλήθευση του συστήματος λογισμικού παρά στην ανάπτυξη. Η λύση που προτείνεται για την μείωση του χρόνου και του κόστους στην επαλήθευση λογισμικού, είναι η ανάπτυξη πλαισίων ελέγχου τα οποία θα είναι σε θέση να εξετάζουν την ορθή λειτουργία ενός συστήματος λογισμικού και την ικανοποίηση των απαιτήσεων του, σε όλα τα στάδια της ανάπτυξης του. Η προσέγγιση μας στο πρόβλημα βασίζεται στην μοντελοποίηση των απαιτήσεων του υπό έλεγχο συστήματος με τη βοήθεια μοντέλων δέντρων στόχων. Η διαδικασία ελέγχου των απαιτήσεων, ξεκινά με την ανακάλυψη ενδείξεων αποτυχίας κάποιας απαίτησης του υπό έλεγχο συστήματος και απαιτεί την εύρεση των πιθανών αρχικών αιτιών. Η επαλήθευση των αρχικών αιτιών μπορεί να επιτευχθεί με την εκτέλεση ενός συνόλου στρατηγικών ελέγχου. Στα πλαίσια αυτής της διπλωματικής εργασίας ορίζουμε στρατηγικές ελέγχου που βασίζονται στις μεθόδους δυναμικής και στατικής ανάλυσης. Έτσι επιτυγχάνεται όχι μόνο η μείωση του όγκου δεδομένων που πρέπει να ελεγχθεί, αλλά και ο ακριβής προσδιορισμός της βασικής αιτίας που οδήγησε στη δυσλειτουργία του συστήματος.

Λέξεις Κλειδιά: << Δέντρα στόχων, στατική ανάλυση, δυναμική ανάλυση, ανάλυση αρχικών αιτιών, μοντελοκεντρικός έλεγχος συστημάτων, τεχνολογία λογισμικού, έλεγχος μονάδων, ανάλυση δεδομένων παρακολούθησης, προτασιακή ικανοποιησιμότητα >>

Abstract

The main goal of this diploma thesis is the design and implementation of a goal-driven framework that applies static and dynamic analysis on software systems. The management of software development projects is a complex process and its main objective is to ensure that the final software system will be delivered on time and meets its end users requirements, taking into account quality requirements and simultaneously keeping the cost within the specified limits. During the development of complex software systems, the most of time and effort needed is consumed in verifying the correct functionality of the software system, rather than in the implementation of the system. The solution proposed for the reduction of the time and cost in software verification processes, is the development of frameworks that are able to verify the correct functionality of the a software system and make sure that it meets its requirements during all development stages. Our approach to this challenge, utilizes goal trees to model and define system requirements. The requirements verification process gets started upon detecting events that may be evidence of a requirement failure of the system under test and requires root-cause analysis. The verification of root causes can be achieved by executing a set of test strategies. For the purposes of this diploma thesis we define test strategies based on dynamic and static analysis methods. The advantage of using these framework is the not only the reduction of the data volume that needs to be analyzed but also the precise definition of the root cause leading to system malfunction.

Keywords : << goal tree, static analysis, dynamic analysis, root-cause analysis, model-based testing, software engineering, unit testing, log analysis, propositional satisfiability >>

Πίνακας Περιεχομένων

Πίνακας Σχημάτων

Κεφάλαιο 1: Εισαγωγή

Η εισαγωγή αυτή περιγράφει το κίνητρο για τη μοντελοποίηση συστημάτων με σκοπό την δοκιμή και τον έλεγχο λογισμικού (Model Based Software Testing), δηλαδή την χρήση μοντέλων που αντιπροσωπεύουν τη συμπεριφορά του συστήματος υπό παρακολούθηση , για τον αποτελεσματικό έλεγχο της λειτουργίας του (κεφάλαιο 1.1) . Παρουσιάζονται τα οφέλη μιας τέτοιας προσέγγισης ελέγχου συστημάτων , καθώς και η σημαντικότερη πρόκληση που αντιμετωπίζουν , την εύρεση και ανάλυση των αρχικών αιτιών μιας λανθασμένης συμπεριφοράς του συστήματος (κεφάλαιο 1.2) . Στη συνέχεια, συνοψίζεται η επιστημονική συνεισφορά της διπλωματικής εργασίας (κεφάλαιο 1.3) και περιγράφεται η δομή της (κεφάλαιο 1.4).

1.1 Κίνητρο

Σύγχρονες έρευνες δείχνουν πως σε ολόκληρο τον κόσμο διατίθενται μεγάλα ποσά σε έργα λογισμικού. Ένα αξιόλογο ποσοστό αυτών των έργων παρουσιάζουν προβλήματα. Υπάρχουν περιπτώσεις που κάποια έργα λογισμικού παρέχουν διαφορετικές υπηρεσίες από αυτές που προτάθηκαν αρχικά, άλλα είτε παρουσιάζουν καθυστερήσεις είτε ξεπερνούν τις αρχικές εκτιμήσεις κόστους, ενώ υπάρχουν και οι περιπτώσεις πολλών έργων που καταλήγουν σε πλήρη αποτυχία.
Η διαχείριση έργων ανάπτυξης λογισμικού αποτελεί μία σύνθετη διαδικασία και κύριος στόχος της είναι να εξασφαλιστεί ότι το τελικό πληροφοριακό σύστημα θα παραδοθεί εγκαίρως, θα ικανοποιεί τις ανάγκες των τελικών χρηστών , θα λαμβάνει υπ' όψιν τις απαιτήσεις ποιότητας και ταυτόχρονα θα διατηρεί το κόστος στα προβλεπόμενα όρια. Είναι γεγονός πως όταν τα λάθη βρίσκονται αργά, τότε το κόστος του να τα διορθώσουμε είναι πολύ μεγαλύτερο από αυτό του να τα διορθώσουμε έγκαιρα. Στην ανάπτυξη λογισμικού πολυσύνθετων συστημάτων, ο περισσότερος χρόνος και η μεγαλύτερη προσπάθεια ξοδεύονται στην επαλήθευση παρά στην κατασκευή.
Μία διαδικασία ανάπτυξης λογισμικού αποτελεί μια συστηματική προσέγγιση για την ανάλυση, σχεδιασμό, κατασκευή και εξέλιξη ενός πληροφοριακού συστήματος. Αποτελείται από διάφορες φάσεις ανάπτυξης (development phases) , οι οποίες υλοποιούνται από την αρμόδια ομάδα μηχανικών λογισμικού. Σε αυτή τη διπλωματική εργασία θα ασχοληθούμε με την φάση του ελέγχου λογισμικού και αποδοχής (software testing) . Η φάση αυτή περιλαμβάνει δραστηριότητες όπως τον μοναδιαίο έλεγχο (unit testing), τον έλεγχο μονάδων λογισμικού (module testing), τον λειτουργικό έλεγχο (functiona testing) , τον έλεγχο των προδιαγραφών του συστήματος (testing specifications) [1] κ.λ.π. Κάθε μια δραστηριότητα από αυτές , έχει έναν συνδυασμό συνθηκών εισόδου, ένα σύνολο προδιαγραφών ελέγχου, ένα χρονοδιάγραμμα για τη φάση του ελέγχου (testing plan), και παρέχει ένα σύνολο αποτελεσμάτων ή μία αναφορά σφαλμάτων.

Σε αυτή τη διπλωματική εργασία παρουσιάζεται μία προσέγγιση με στόχο την διαχείριση της πολυπλοκότητας και ποικιλίας που χαρακτηρίζουν τις διαδικασίες επαλήθευσης λογισμικού.

Προτείνεται η αξιοποίηση μοντέλων για την αναπαράσταση του λογισμικού και την ανάλυση των διαδικασιών του ελέγχου του. Βασικός στόχος είναι μία ενιαία αρχιτεκτονική που θα ενσωματώνει μοντέλα προσανατολισμένα στην εύρεση προβλημάτων. Παρόλο που αυτός ο τύπος ελέγχου λογισμικού απαιτεί σημαντική προσπάθεια κατά τη διάρκεια σύνθεσης του μοντέλου, παρέχει κάποια σημαντικά πλεονεκτήματα συγκριτικά με παραδοσιακούς τύπους ελέγχου λογισμικού [2]:

- Είναι μία γενική μέθοδος επαλήθευσης που μπορεί να εφαρμοστεί σε μεγάλο πεδίο εφαρμογών όπως είναι τα ενσωματωμένα συστήματα (embedded systems), η τεχνολογία λογισμικού και ο σχεδιασμός υλικού (hardware design).

- Υποστηρίζει μερική επαλήθευση, δηλαδή για παράδειγμα οι ιδιότητες μπορούν να ελεγχθούν ξεχωριστά, έτσι ώστε να ελεγχθούν οι βασικές ιδιότητες πρώτα. Δεν χρειάζεται πλήρης προδιαγραφή των απαιτήσεων.

- Παραχωρεί διαγνωστικές πληροφορίες σε περίπτωση που μια ιδιότητα ανατραπεί. Αυτό είναι πολύ χρήσιμο για σκοπούς αποσφαλμάτωσης.

- Προκαλεί ένα γοργά αυξανόμενο ενδιαφέρον στη βιομηχανία.

- Έχει μαθηματικό υπόβαθρο. Βασίζεται στη θεωρία αλγόριθμων γραφικών παραστάσεων, δομές δεδομένων και λογικής.

- Μπορεί εύκολα να ενσωματωθεί σε ήδη υπάρχοντες κύκλους ανάπτυξης. Μελέτες έχουν δείξει ότι μπορεί να οδηγήσει σε χαμηλότερο χρόνο ανάπτυξης.

- Παρέχει την δυνατότητα αυτοματοποίησης της διαδικασίας επαλήθευσης λογισμικού , μειώνοντας τόσο το χρόνο όσο και το κόστος της ολοκλήρωσής της.

Συμπεραίνουμε λοιπόν πως το Model Based Testing φαίνεται να είναι ένα ισχυρό εργαλείο για τη μελλοντική αγορά , το οποίο κάνει την επαλήθευση εφαρμογών λογισμικού ευκολότερη και ταχύτερη.

1.2 Περιγραφή Προβλήματος

Ο στόχος της εργασίας αυτής είναι η ανάπτυξη ενός περιβάλλοντος-πλαισίου με χρήση μοντέλων για την αναπαράσταση του λογισμικού και την ανάλυση των διαδικασιών του ελέγχου του. Υπάρχουν πολλά μοντέλα για να κάνουμε Model Based Testing σε ένα σύστημα, όπως για παράδειγμα οι πίνακες απόφασης, οι μηχανές πεπερασμένων καταστάσεων, οι γραμματικές, οι αλυσίδες Markov και τα UML. Στη παρούσα διπλωματική εργασία παρουσιάζουμε την μοντελοποίηση συστημάτων χρησιμοποιώντας δέντρα στόχων (Goal Models).
Εξηγούμε πως μία λειτουργία ενός συστήματος μπορεί να διαιρεθεί σε απλούστερες λειτουργίες , ώστε να διευκολυνθεί η ανάλυση των αρχικών αιτιών της αποτυχίας μιας λειτουργίας (Root Cause Analysis) . Στη συνέχεια , παρουσιάζουμε πως είναι δυνατή η πυροδότηση μίας υπόθεσης

σφάλματος μίας λειτουργίας, για την εκκίνηση της διαδικασίας ανάλυσης αρχικών αιτιών. Τέλος, θα δούμε πως μπορούμε να αποφανθούμε για τη ισχύ ή την άρνηση μίας υπόθεσης σφάλματος, επιλέγοντας στρατηγικές επαλήθευσης των αρχικών αιτιών .

Κατά την σχεδίαση του περιβάλλοντος-πλαισίου, αναφερόμαστε στα παρακάτω ζητήματα :
- Πως μπορούμε να μοντελοποιήσουμε ένα σύστημα λογισμικού ;

- Πως μπορούμε διαμερίσουμε μία λειτουργία του σε απλούστερες λειτουργίες που ελέγχονται ταχύτερα και ευκολότερα ;

- Πως μπορεί μία υπόθεση σφάλματος να πυροδοτήσει τη διαδικασία επαλήθευσης της αποτυγχάνουσας συνιστώσας ;

- Πως μπορούμε να αποφανθούμε ποιές είναι οι πιθανές αρχικές αιτίες για την αποτυχία μιας λειτουργίας του συστήματος ;

- Πως μπορούμε να ελέγξουμε την ισχύ ή την άρνηση μίας πιθανής αρχικής αιτίας αποτυχίας μιας λειτουργίας του συστήματος ;

1.3 Συνεισφορά της Διπλωματικής Εργασίας

Η συνεισφορά της παρούσας διπλωματικής εργασίας συνοψίζεται στα εξής σημεία:

- Στον σχεδιασμό, την υλοποίηση και αξιολόγηση μίας αρχιτεκτονικής μοντελοποίησης πληροφοριακών συστημάτων, η οποία θα υποστηρίζει την αυτοματοποίηση της διαδικασίας επαλήθευσης λογισμικού , βασισμένη σε μοντέλα και υποθέσεις που μπορούν να ορίσουν οι χρήστες - μηχανικοί επαλήθευσης λογισμικού . Το περιβάλλον πλαίσιο που θα παρουσιάσουμε στη εργασία αυτή μπορεί να χρησιμοποιηθεί στη φάση ελέγχου λογισμικού και αποδοχής , που αποτελεί την τελική φάση του κύκλου ανάπτυξης λογισμικού.

- Στην επέκταση του μοντέλου Δέντρων-Στόχων με alarms και στρατηγικές επαλήθευσης υποθέσεων , σε στόχο την αποσφαλμάτωση ενός πληροφοριακού συστήματος. Τα alarms μοντελοποιούν οντότητες που σηματοδοτούν την εύρεση σφαλμάτων σε ένα σύστημα και εκκινούν την διαδικασία εύρεσης των αρχικών αιτιών. Οι στρατηγικές αποτελούν ενέργειες στις οποίες πρέπει να βασιστεί ο χρήστης για τον έλεγχο της ισχύος μίας πιθανής αιτίας αποτυχίας του συστήματος. Το μοντέλο Δέντρων-Στόχων μπορεί να χρησιμοποιηθεί για την μοντελοποίηση οποιουδήποτε πληροφοριακού συστήματος, καθώς και να επεκταθεί ώστε να περιγράψει ακόμα και σύνθετα μοντέλα συστημάτων.

- Στην αυτοματοποίηση της διαδικασίας εύρεσης αιτιών αποτυχίας μίας λειτουργίας ενός συστήματος (Root Cause Analysis) [3] . Με αυτόν τον τρόπο , το πρόβλημα μπορεί να αντιμετωπιστεί στην ρίζα του, υπογραμμίζοντας τις αιτίες που πραγματικά συνεισφέρουν σε αυτό. Η διαδικασία Root Cause Analysis βοηθά τις ομάδες ανάπτυξης λογισμικού, να αποφύγουν την τάση του να ξεχωρίζουν έναν παράγοντα που οδηγεί στην αποτυχία μίας λειτουργίας, οδηγώντας σε μία ελλιπή λύση . Προβάλλει την γενικότερη εικόνα όλων των σημαντικών

συνιστωσών που οδηγούν στο πρόβλημα, παρέχοντας μία σφαιρική εικόνα.

- Στην χρήση της διαδικασίας ανάλυσης δεδομένων και καταγραφών παρακολούθησης ενός συστήματος (Log Analysis) [4]. Η διατήρηση συμπτωμάτων (logging) είναι η διαδικασία κατά την οποία συλλέγονται και καταγράφονται γεγονότα κατά την λειτουργία και αλληλεπίδραση στοιχείων και χρηστών του συστήματος, και παρέχει πολύτιμη πληροφορία για την επαλήθευση των πιθανών αιτιών (root causes) ενός προβλήματος. Στην εργασία αυτή, χρησιμοποιούμε τα δεδομένα από την παρακολούθηση του συστήματος -αντικειμένου , ώστε να αποδείξουμε την ισχύ των αποτελεσμάτων της διαδικασίας Root Cause Analysis.

1.4 Οργάνωση του Κειμένου

Τα επόμενα κεφάλαια της διπλωματικής εργασίας οργανώνονται ως εξής :
Στο κεφάλαιο 2, παρουσιάζονται έννοιες και τεχνολογίες που χρησιμοποιήθηκαν για την εκπόνηση της διπλωματικής εργασίας. Αναφέρονται οι έννοιες της μοντελοποίησης λογισμικού αλλά και η ιδέα των Δέντρων Στόχων στην οποία βασίστηκε η μοντελοποίηση των λειτουργιών και των στρατηγικών επαλήθευσης . Παρουσιάζονται τα εργαλεία που βοήθησαν στην υλοποίηση του περιβάλλοντος-πλαισίου και στον ορισμό των Δέντρων Στόχων, με ποιο σημαντικά το EMF και την βιβλιοθήκη SAT4J. Τέλος δίνεται μια επισκόπηση των κύριων εννοιών αυτής της εργασίας : Model Based Testing, Root Cause Analysis, Log Analysis,Static Analysis, Dynamic Analysis, Unit Testing.

Στο κεφάλαιο 3 δίνεται η αρχιτεκτονική του περιβάλλοντος-πλαισίου ,αναλύονται μία προς μία οι συνιστώσες (components) που την απαρτίζουν και ορίζονται οι αλληλεπιδράσεις μεταξύ τους.

Στο κεφάλαιο 4 παρουσιάζεται ένα εννοιολογικό μοντέλο των δέντρων στόχων, το οποίο χρησιμοποιείται για την μοντελοποίηση των υποθέσεων και των στρατηγικών επαλήθευσης τους. Οι υποθέσεις αποτελούν τις ρίζες των δέντρων , οι οποίες απαρτίζονται από κόμβους-φύλλα, δηλώνοντας πως μία λειτουργία ενός συστήματος μπορεί να εκφραστεί σαν ένα σύνολο απλούστερων συνιστωσών. Τα απαραίτητα alarm για την πυροδότηση της διαδικασίας αποσφαλμάτωσης βρίσκονται στις ρίζες των δέντρων , καθώς συνδέονται με μία υπόθεση. Οι στρατηγικές επαλήθευσης αντίστοιχα είναι μοντελοποιημένες στα φύλλα - κόμβους. Στο ίδιο κεφάλαιο δίνεται μία περιγραφή κάθε κλάσης του συστήματος και η αντίστοιχη συνεισφορά στο περιβάλλον-πλαίσιο.

Στο κεφάλαιο 5 παρουσιάζουμε τους κανόνες που χρησιμοποιούνται για την οδήγηση του SAT-Solver και την μετατροπή των δέντρων στόχων στους κανόνες αυτούς. Στην συνέχεια παρουσιάζουμε λεπτομέρειες του βρόχου ελέγχου που χρησιμοποιεί το περιβάλλον-πλαίσιο και την αξιοποίηση του SAT-Solver.

Στο κεφάλαιο 6 παρουσιάζεται μία μελέτη χρήσης του περιβάλλοντος-πλαισίου που αναπτύξαμε εφαρμόζοντας τις τεχνικές ελέγχου σε μια προσομοίωση ενός συστήματος ATM.

Κεφάλαιο 2: Σχετικές έννοιες

2.1 Εισαγωγή

Το παρόν κεφάλαιο αποτελεί μια συνοπτική παρουσίαση του συνόλου των τεχνολογιών που χρησιμοποιήθηκαν για την ανάπτυξη του περιβάλλοντος κατά την εκπόνηση της διπλωματικής εργασίας , καθώς και το θεωρητικό υπόβαθρο που απαιτείται για την κατανόηση της. Το γνωστικό αντικείμενο που πραγματεύεται το κεφάλαιο αυτό είναι αρκετά ευρύ και ξεπερνά τα πλαίσια αυτής της διπλωματικής. Για το λόγο αυτό περιορίζεται μόνο σε βασικές έννοιες. Για περισσότερες πληροφορίες ο αναγνώστης μπορεί να ανατρέξει στις βιβλιογραφικές αναφορές, στο κεφάλαιο 8.

2.2 Μοντελοποίηση συστημάτων λογισμικού (Model Driven Architecture)

Η Μοντελοκεντρική Αρχιτεκτονική (Model Driven Architecture, MDA) αποτελεί μια σύγχρονη προσέγγιση ανάπτυξης και σχεδίασης λογισμικού. Η προσέγγιση αυτή παρέχει ένα σύνολο κατευθυντήριων γραμμών για την διάρθρωση των προδιαγραφών ενός συστήματος λογισμικού , οι οποίες εκφράζονται ως μοντέλα. Σκοπός της μοντελοποίησης ενός συστήματος είναι η εξαγωγή πληροφορίας και η αναπαράστασή της σε ένα μοντέλο το οποίο μπορεί να γίνει εύκολα κατανοητό από έναν μηχανικό. Η μοντελοποίηση συστημάτων λογισμικού έχει ως τεχνολογικό υπόβαθρο μια σειρά από ήδη υπάρχουσες τεχνολογίες, οι οποίες παρουσιάζονται συνοπτικά στη συνέχεια του κεφαλαίου.

2.2.1 MOF

Η έννοια της μετα-μοντελοποίησης αποτελεί ουσιαστική βάση για τον καθορισμό και την επεξεργασία των γλωσσών μοντελοποίησης και των μοντέλων τους. Το Meta Object Facility [5]

προτάθηκε από τον οργανισμό Object Management Group (OMG) ως ένα πρότυπο της μοντελοποίησης συστημάτων . Αποτελεί ουσιαστικά μια αφηρημένη γλώσσα και ένα περιβάλλον πλαίσιο για την προδιαγραφή, κατασκευή και διαχείριση των μεταμοντέλων. Το πρότυπο αυτό παρέχει ένα ανεξάρτητο πλαίσιο διαχείρισης μεταδεδομένων , δίνοντας τη δυνατότητα ανάπτυξης και δια-λειτουργικότητας των συστημάτων μεταδεδομένων, διατηρώντας τη συμβατότητα μεταξύ προϊόντων διαφορετικών κατασκευαστών . Η αρχιτεκτονική που ορίζεται στο πρότυπο Meta Object Facility χωρίζεται σε τέσσερα επίπεδα[6][7]. Τα επίπεδα αυτά περιγράφονται παρακάτω:

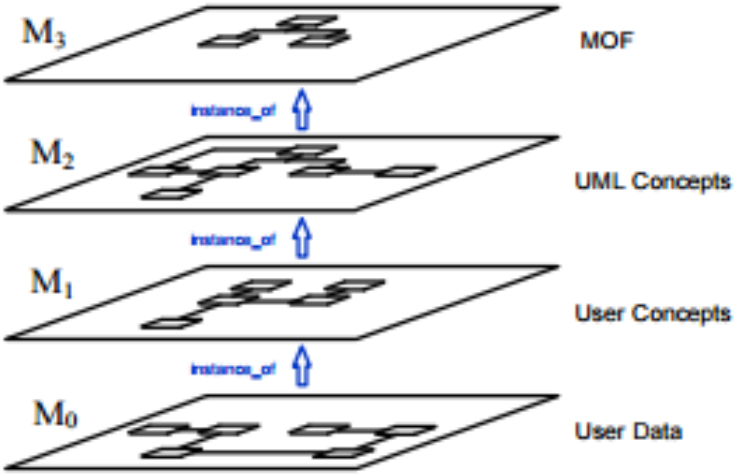
1. Το επίπεδο M3 ή αλλιώς επίπεδο μετα-μετα-μοντέλου (meta-meta model layer - MOF). Το επίπεδο αυτό είναι το ανώτερο επίπεδο της αρχιτεκτονικής και αποτελεί την γλώσσα η οποία χρησιμοποιείται από το MOF για την κατασκευή των μεταμοντέλων (M2-models).

2. Το επίπεδο M2 ή αλλιώς μεταμοντέλο (meta model layer). Το επίπεδο αυτό αποτελεί τη γλώσσα μοντελισμού την οποία χρησιμοποιούμε για να περιγράψουμε τα μοντέλα του κατωτέρου επιπέδου M1. Το πιο χαρακτηριστικό παράδειγμα ενός μεταμοντέλου επιπέδου 2 είναι το UML μεταμοντέλο. Τα μεταμοντέλα σε αυτό το επίπεδο υπακούουν σε κάποιο μετα-μετα-μοντελο. Για παράδειγμα το UML μεταμοντέλο αποτελεί στιγμιότυπο του MOF.

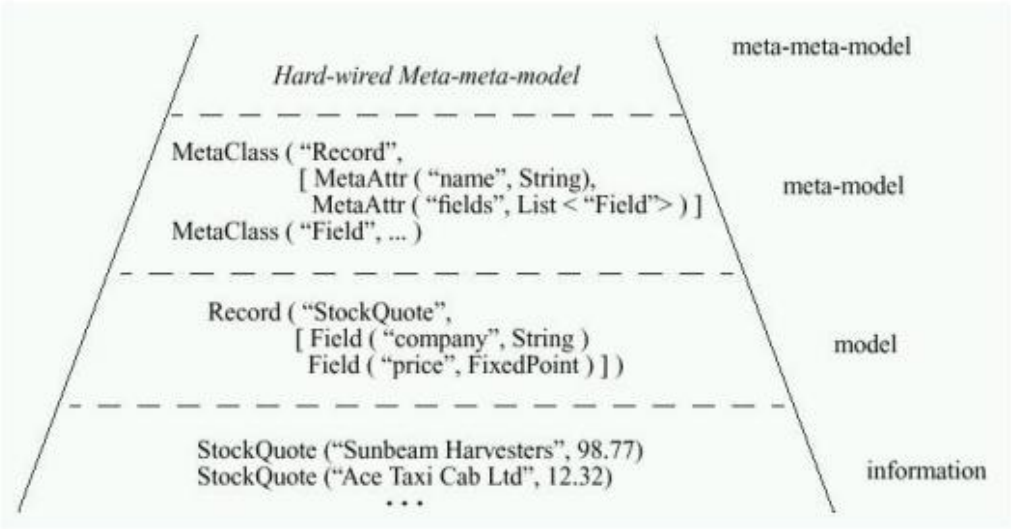
3. Το επίπεδο M1 ή αλλιώς επίπεδο μοντέλου (model layer). Στο επίπεδο αυτό ανήκουν τα μοντέλα , τα οποία είναι ένα στιγμιότυπο ενός μεταμοντέλου του επιπέδου 2. Στο πεδίο των συστημάτων λογισμικού, χαρακτηριστική περίπτωση στοιχείων του επιπέδου M1 είναι τα μοντέλα των λογισμικών συστημάτων τα οποία περιγράφονται με UML μοντέλα (πχ διαγράμματα κλάσεων, ακολουθιακά διαγράμματα κτλ).

4. Το επίπεδο M0 η αλλιώς επίπεδο πληροφορίας (information layer) . Στο επίπεδο αυτό βρίσκονται όλα τα δεδομένα ή τα αντικείμενα που θέλουμε να μοντελοποιήσουμε .

Στα σχήματα 2.1 και 2.2 φαίνεται η αρχιτεκτονική των τεσσάρων επιπέδων που μόλις περιγράψαμε.



Σχήμα 2.1: Αρχιτεκτονική Μετα-δεδομένων Τεσσάρων επιπέδων (MOF-UML)



Σχήμα 2.2: Αρχιτεκτονική Μετα-δεδομένων Τεσσάρων επιπέδων

Η αρχιτεκτονική που παρουσιάστηκε δεν έχει όμως απόλυτο χαρακτήρα. Το MOF έχει σχεδιαστεί έτσι ώστε να μπορεί να χρησιμοποιηθεί σε αρχιτεκτονικές με αριθμό επιπέδων μεγαλύτερο των δύο ανάλογα με τις μοντελιστές ανάγκες του εκάστοτε προβλήματος. Τέλος θα πρέπει να σημειωθεί πως το MOF παρέχει τη δυνατότητα μετακίνησης των μοντέλων που

περιγράφει, από εφαρμογή σε εφαρμογή καθώς και τη δυνατότητα αποθήκευσης τους σε άλλες μορφές όπως το XMI.

2.2.2 XMI

Όταν η UML εμφανίστηκε για πρώτη φορά , δεν υπήρξε τυποποιημένη μορφή για την ανταλλαγή των μοντέλων UML, τα περισσότερα εργαλεία είχαν την δική τους μορφή κειμένου κάνοντας αδύνατη την τροποποίηση και την μεταφορά των μοντέλων. Για τη λύση αυτού του προβλήματος ο οργανισμός Object Management Goup (OMG) πρότεινε το πρότυπο XML Metadata Interchange (XMI) που έχει πλέον πιστοποιηθεί κατά ISO/IEC (ISO /IEC 19503:2005 Information technology). Η δυνατότητα της αποθήκευσης και τις μεταφόρτωσης XMI αρχείων είναι τώρα ένα τυποποιημένο χαρακτηριστικό γνώρισμα που παρέχεται από όλα σχεδόν τα εργαλεία UML, από τα πιο ισχυρά (και ακριβά) εργαλεία μοντελοποίησης , όπως το Rational Rose μέχρι τα λιγότερα ισχυρά εργαλεία όπως το MagicDraw και το QuickUML. Γενικά τα περισσότερα εργαλεία παρέχουν τη δυνατότητα να αποθηκευθεί το μοντέλο σε μορφή XMI ως ξεχωριστή μορφή. Ωστόσο ορισμένα εργαλεία όπως τα εργαλεία ελεύθερου λογισμικού Argo /UML και Poseidon , χρησιμοποιούν τη μορφή XMI ως τη βασική τους μορφή. Το μεγάλο πλεονέκτημα της μορφής XMI είναι ότι βασίζεται στη γλώσσα Extensive Markup Language (XML) και μπορεί να χρησιμοποιήσει όλο το φάσμα των εργαλείων της γλώσσας XML. Κάποια από τα πλεονεκτήματα της χρήσης της μορφής XMI φαίνεται παρακάτω[8]:

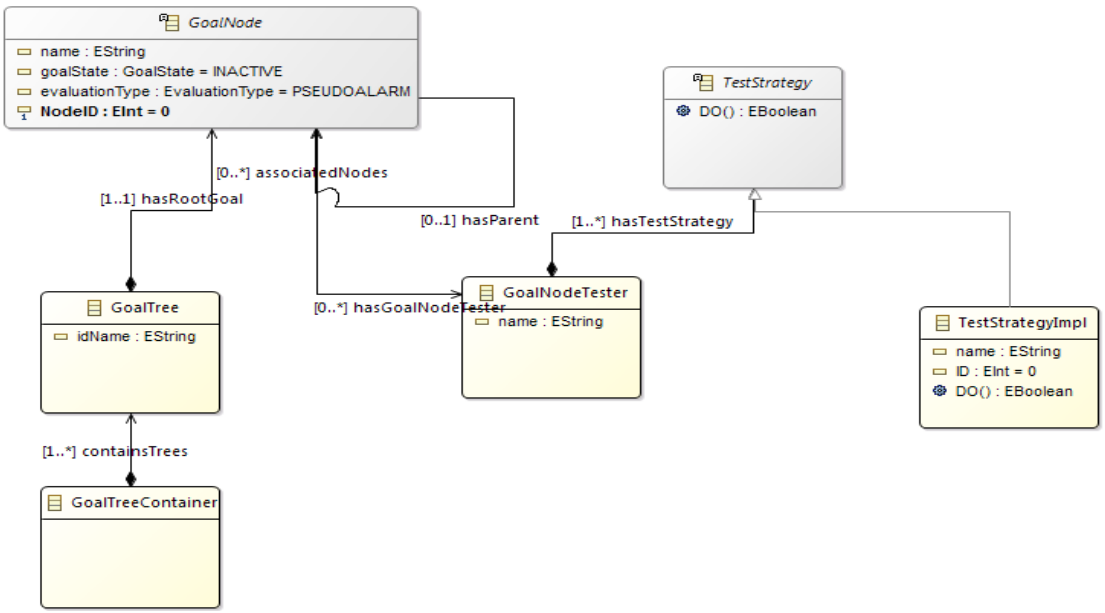
- Η ανάλυση ή η τροποποίηση κώδικα μπορεί να αυτοματοποιηθεί μέσω της χρήσης UML εργαλείων. Για παράδειγμα αλλαγές που γίνονται στο XMI μοντέλο μπορούν να διαδοθούν στον κώδικα , έτσι ώστε ο προγραμματιστής να μην χρειάζεται να πειράξει απευθείας τον κώδικα (και το αντίστροφο) .
- Κάθε προγραμματιστής μπορεί να γράψει ένα σενάριο για να εξαγάγει τις πληροφορίες από τα αρχεία XMI και στη συνέχεια να το μετατρέψει σε μορφή εισόδου σε ένα άλλο ιδιόκτητο εργαλείο .

Στόχος λοιπόν αυτού του προτύπου, είναι να επιτραπεί η εύκολη ανταλλαγή μεταδεδομένων μεταξύ των εργαλείων ανάπτυξης εφαρμογών[9][10] μέσω της γλώσσας Extensive Markup Language (XML). Τέτοια εργαλεία συνήθως είναι εργαλεία μοντελοποίησης που βασίζονται στη γλώσσα Unified Modeling Language (UML) καθώς και εργαλεία /εφαρμογές που βασίζονται στο πρότυπο του MOF . Το πρότυπο XMI μπορεί να χρησιμοποιηθεί σε όλα τα μεταμοντέλα τα οποία εκφράζονται μέσω του προτύπου MOF. Έτσι το μοντέλο XMI , το οποίο περιγράφει την σημασιολογία της πληροφορίας, αποτελεί στην ουσία ένα στιγμιότυπο ενός μοντέλου MOF.

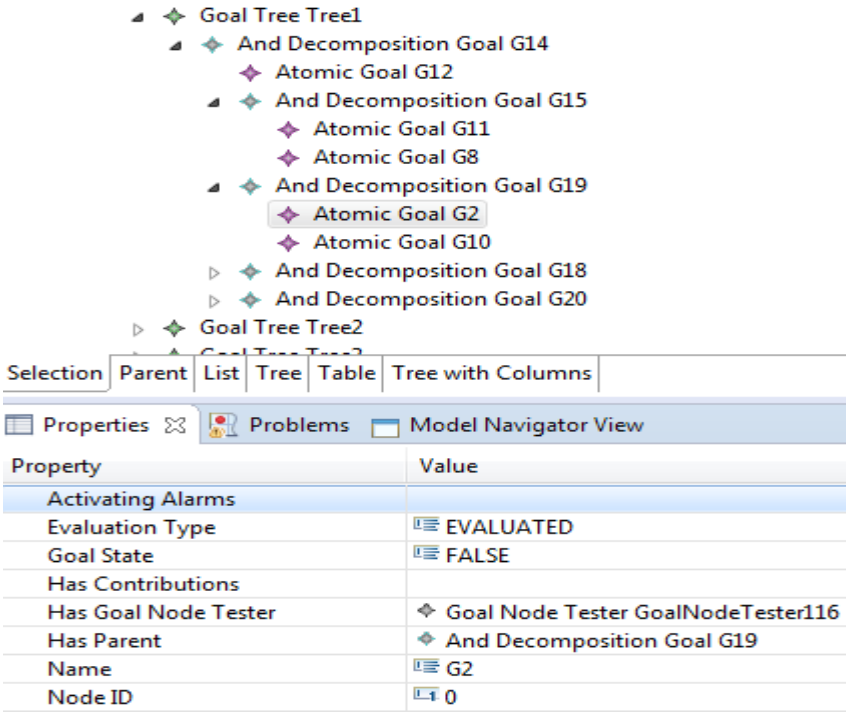
2.2.3 Eclipse Modeling Framework

Το περιβάλλον-πλαίσιο που αναπτύξαμε κατά τη εκπόνηση αυτής της διπλωματικής εργασίας ακολούθησε το αντικειμενοστραφές μοντέλο προγραμματισμού. Για την μοντελοποίηση συστημάτων προγραμματισμού με χρήση αντικειμενοστραφής λογικής απαιτούνται εργαλεία όπως η UML. Βασισμένο στην UML και ένα πολύ χρήσιμο εργαλείο για την μοντελοποίηση συστημάτων λογισμικού είναι το EMOF (Extended Meta-Object Facility) του οργανισμού OMG. Τα παραπάνω αποτελούν τη βάση για το ευρέως διαδεδομένο εργαλείο μοντελοποίησης συστημάτων λογισμικού EMF (Eclipse Modeling Framework) . Το Eclipse Modeling Framework [11] [12] , αποτελεί προσάρτημα λογισμικού (plug-in) του περιβάλλοντος λογισμικού Eclipse . Σκοπός του είναι η διευκόλυνση της ανάπτυξη λογισμικού και εφαρμογών που υπακούουν στην μοντελοκεντρική αρχιτεκτονική. Επιτρέπει την γρήγορη και αποτελεσματική μετατροπή των μοντέλων σε κώδικα. Πιο συγκεκριμένα το EMF δίνει στο χρήστη τη δυνατότητα να σχεδιάσει μοντέλα που υπακούουν στο πρότυπο MOF και περιγράφουν ένα σύστημα λογισμικού [13]. Τα μοντέλα αυτά μπορούν άμεσα να μετατραπούν μέσω του EMF σε πηγαίο κώδικα. Ο χρήστης έχει τη δυνατότητα να προσθέσει απευθείας νέες τάξεις, μεθόδους η μεταβλητές στο κώδικα που έχει που έχει παραχθεί από το EMF και στη συνέχεια να παράξει ξανά κώδικα από τα μοντέλο MOF . Οι αλλαγές του χρήστη θα διατηρηθούν κατά τη διάρκεια της αναγέννησης του κώδικα. Όμως, αν το σημείο στο οποίο ο χρήστης άλλαξε απευθείας τον κώδικας, επηρεάζεται από το μοντέλο, τότε οι αλλαγές του θα επανογραφούν από το κώδικα που θα παραχθεί από το μοντέλο. Μια επίσης σημαντική δυνατότητα του λογισμικού EMF είναι η δυνατότητα παραγωγής στιγμιότυπων λειτουργίας των συστημάτων που περιγράφτηκαν μέσω των MOF μοντέλων. Τα στιγμιότυπα αυτά μπορούν να αποθηκευτούν σε μορφή XMI και να χρησιμοποιηθούν από τον κώδικα που έχει παραχθεί μέσω του μοντέλου.

Στα πλαίσια αυτής της διπλωματικής, το EMF χρησιμοποιήθηκε για την μοντελοποίηση του τροποποιημένου Μοντέλου Δένδρων Στόχων, που χρειάστηκε να ορισθεί για τις ανάγκες του περιβάλλοντος-πλαισίου. Αρχικά δημιουργήθηκε με βάση το MOF, το Μοντέλο του Δένδρου Στόχων, και στην συνέχεια χρησιμοποιήθηκε ο παραγόμενος κώδικα χειρισμού του μοντέλου δέντρου στόχων καθώς και το EMF API, ώστε να οριστούν τα δέντρα στόχων και να υλοποιηθούν οι αλγόριθμοι και δημιουργίας και διαχείρισης των δέντρων στόχων, για τις ανάγκες του περιβάλλοντος πλαισίου. Ένα στιγμιότυπο λειτουργίας του EMF καθώς και ένα τμήμα του μοντέλο βασισμένο στο πρότυπο MOF που χρησιμοποιήθηκε, φαίνονται στις παρακάτω εικόνες .



Σχήμα 2.3: Μοντέλο βασισμένο στο πρότυπο MOF



Σχήμα 2.4: Στιγμιότυπο λειτουργίας του EMF

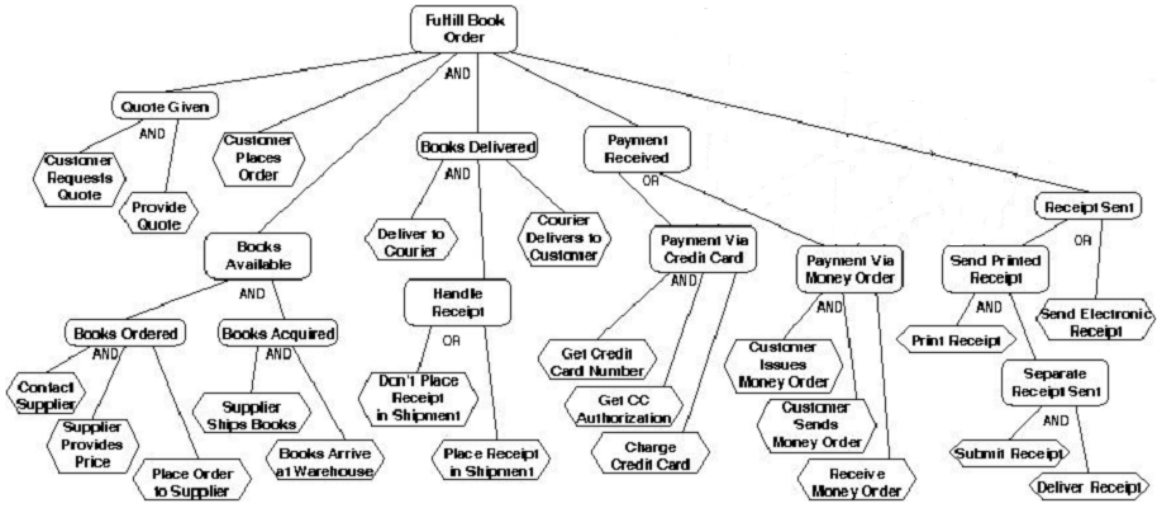
2.3 Δέντρα Στόχων (Goal Trees)

Το κύριο μέτρο της επιτυχίας ενός πληροφοριακού συστήματος είναι ο βαθμός που συναντά τις αρχικές του προδιαγραφές και απαιτήσεις. Επομένως, ο προσδιορισμός αυτών των απαιτήσεων αποτελεί τη σημαντικότερη φάση της ανάπτυξης λογισμικού. Έχει αναγνωριστεί πως ανακριβείς και ημιτελείς απαιτήσεις έχουν σημαντική αρνητική επίδραση στην ποιότητα ενός λογισμικού. Ωστόσο, η Μηχανική Απαιτήσεων (Requirements Engineering) , ένας κλάδος της τεχνολογίας λογισμικού που ασχολείται με την εξαγωγή, ανάλυση και βελτίωση των απαιτήσεων συστημάτων λογισμικού, αποκτά ολοένα και μεγαλύτερο έδαφος στον ακαδημαϊκό χώρο, αλλά και στην βιομηχανία λογισμικού.
Η Μηχανική Απαιτήσεων έχει ως στόχο να προσδιορίσει τους λόγους για τους οποίους ένα σύστημα λογισμικού είναι χρήσιμο, τις λειτουργίες που πρέπει να έχει για να πετύχει τον στόχο του, και τους περιορισμούς σχετικά με το πώς το λογισμικό πρέπει να σχεδιαστεί και να υλοποιηθεί.

Η μοντελοποίηση συστημάτων φαίνεται να είναι ένα βασικό εργαλείο της μηχανικής απαιτήσεων [14]. Το υπάρχον σύστημα λογισμικού συνήθως μοντελοποιείται, χρησιμεύοντας ως βασική κοινή διεπαφή για όλες τις φάσεις της ανάπτυξης του λογισμικού. Υπάρχουν πολλά πλεονεκτήματα της χρήσης της μοντελοποίησης στον τομέα Μηχανικής Απαιτήσεων. Πρώτον, επιτρέπει στον μηχανικό Απαιτήσεων (Requirements Engineer) να έχει μία πιο σφαιρική εικόνα του συστήματος, διευκολύνοντας την βελτίωση των απαιτήσεων. Δεύτερον, τα μοντέλα διευκολύνουν τη γνωστοποίηση των απαιτήσεων στους προγραμματιστές ή ακόμα και στους πελάτες. Τρίτον, τα μοντέλα μπορούν να επαναχρησιμοποιηθούν στην ίδια τεχνολογική περιοχή. Τέλος, αποτελούν τη βάση για την καταγραφή και τεκμηρίωση των απαιτήσεων (Requirements Documentation) και επίσης επιτρέπουν την ακριβή ανάλυση όχι μόνο από τον άνθρωπο αλλά και από εργαλεία λογισμικού.

Μια προσέγγιση στην Μηχανική Απαιτήσεων, με χρήση μοντελοποίησης , είναι η προσανατολισμένη σε στόχους Μηχανική Απαιτήσεων. Σε αυτή τη προσέγγιση, οι απαιτήσεις ενός συστήματος αντιμετωπίζονται σαν στόχοι . Οι στόχοι μπορούν να μοντελοποιηθούν σαν ένα Δένδρο Στόχων, δηλαδή σε μία δενδρική δομή , αποδομώντας κάθε στόχο σε υποστόχους, οι οποίοι αναπαρίστανται σαν στόχοι - παιδιά. Η επαλήθευση των υποστόχων, σε συνάρτηση με την λογική αποσύνθεσης του στόχου για την οποία θα μιλήσουμε παρακάτω, συνεπάγονται την επαλήθευση του αντίστοιχου στόχου [15].

Αναλύοντας ένα μοντέλο στόχων από πάνω προς τα κάτω, παρατηρούμε πως η αποσύνθεση του κάθε στόχου μπορεί να είναι διαφόρων τύπων , και αποτελεί την λογική με την οποία ορίζεται η επαλήθευση του στόχου από τους υποστόχους (παιδιά) [16]. Στα πλαίσια της διπλωματικής εργασίας, θα μιλήσουμε για την αποσύνθεση ενός στόχου μέσω λογικών συζεύξεων και διαζεύξεων. Ένα τέτοιο παράδειγμα δέντρου [17] απεικονίζεται στο επόμενο σχήμα.



Σχήμα 2.5: Παράδειγμα δέντρου στόχων με λογικές συζεύξεις και διαζεύξεις

Η Λογική Σύζευξη συμβολίζεται με τον τύπο ΚΑΙ (AND), ενώ η λογική διάζευξη με τον τύπο Η (OR) , όπως φαίνεται και στο παρακάτω σχήμα (σχήμα 2.6) .

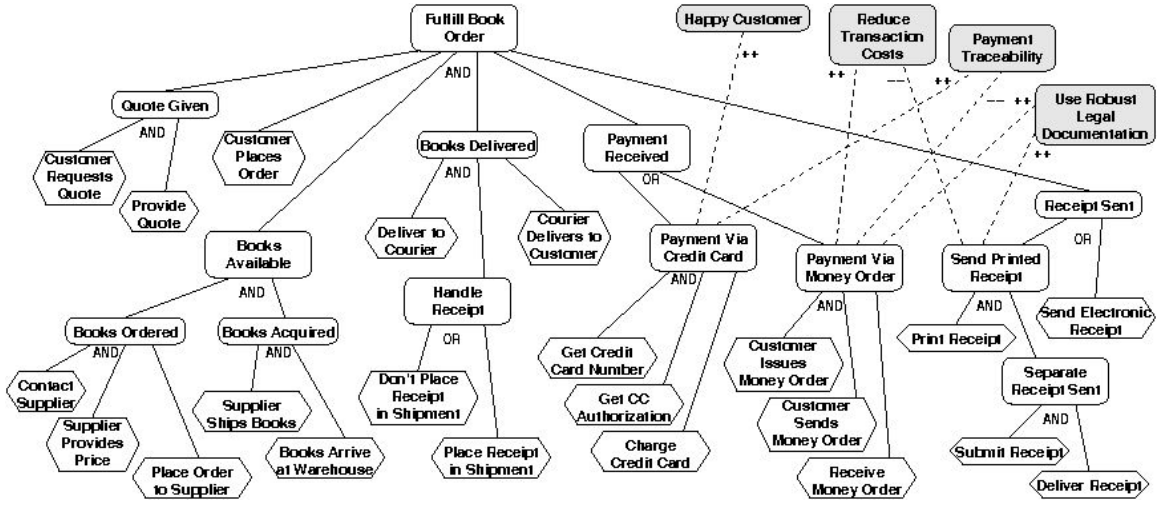
Στο επόμενο σχήμα [18] φαίνεται ο τρόπος επαλήθευσης ενός στόχου από τους στόχους παιδιά. Ένας στόχος που έχει παιδιά τα οποία συνδέονται με τη λογική σύζευξη, επαληθεύεται μόνο εάν επαληθευτούν όλοι οι στόχοι- παιδιά. Ένας στόχος που έχει παιδιά τα οποία συνδέονται με τη λογική διάζευξη , επαληθεύεται εάν επαληθευτεί έστω και ένας στόχος-παιδί.

AND $(G,\{G_1,G_2,\ldots,G_n\})$	– goal G is satisficed when all of $G_1,G_2,\ldots,G_n$ are satisficed and there is no negative evidence against it;
	– goal G is unsatisficed and there is one of $G_1,G_2,\ldots,G_n$ is unsatisficed and there is no positive evidence for it.
OR $(G,\{G_1,G_2,\ldots,G_n\})$	– goal G is satisficed when one of $G_1,G_2,\ldots,G_n$ is satisficed and there is no negative evidence against it;
	– goal G is unsatisficeable if all of $G_1,G_2,\ldots,G_n$ are unsatisfieceable and there is no positive evidence for it.
$+(G_1,G_2)$	– goal G_1 contributes positively to the satisficing of goal G_2.
$-(G_1,G_2)$	– goal G_1 contributes negatively to the satisficing of goal G_2.

Σχήμα 2.6: Ορισμοί λογικών συζεύξεων και διαζεύξεων και συνεισφορών

Τέτοιου είδους στόχοι για τους οποίους υπάρχει μονοσήμαντη σημασιολογία επαλήθευσης του στόχου από το σύνολο στόχων-παιδιών , ονομάζονται hard goals. Στην περίπτωση των hard goals δηλαδή, ένας στόχος είναι ικανοποιημένος απόλυτα όταν και οι υποστόχοι του είναι ικανοποιημένοι . Καθώς όμως η φύση των απαιτήσεων λογισμικού μπορεί να είναι σχετική , και πολλές φορές ασαφής και αβέβαιη, χρειαζόμαστε μία λιγότερο "αυστηρή" έννοια ενός στόχου, ως προς τον τρόπο επαλήθευσής του. Soft goals είναι οι στόχοι που δεν έχουν σαφή κριτήριο για την ικανοποίησή τους. Μπορούμε να πούμε πως τα softgoals ικανοποιούνται όταν υπάρχει επαρκής θετική και ελάχιστα αρνητική ένδειξη αυτού του ισχυρισμού, ενώ δεν ικανοποιούνται όταν υπάρχουν επαρκή αρνητικά στοιχεία και μικρή θετική υποστήριξη για την ικανοποιησιμότητά τους. Μερικές φορές οι αποδείξεις είναι ισχυρές και επαρκείς ώστε να η απόφαση για την ικανοποιησιμότητα ενός στόχου να γίνει αυτόματα, χωρίς την παρέμβαση ανθρώπου. Σε άλλες περιπτώσεις, όταν υπάρχουν ασθενή ή αντικρουόμενα στοιχεία, η απόφαση μπορεί να γίνει με διαδραστικό τρόπο από τα ενδιαφερόμενα μέρη στη διαδικασία ανάλυσης απαιτήσεων.

Στο παρακάτω διάγραμμα (σχήμα 2.7) τα hard goals παρουσιάζονται με τα λευκό χρώμα ενώ τα soft goals με τα γκρι χρώμα [18]. Φαίνεται ξεκάθαρα η δενδρική δομή και οι αποσυνθέσεις των κόμβων - στόχων καθώς επίσης και οι συνεισφορές των στόχων σε κάθε soft goal. Οι θετικές συνεισφορές παρουσιάζονται με το σύμβολο ++ (συν) ενώ οι αρνητικές με το σύμβολο -- (μείον).



Σχήμα 2.7: Παράδειγμα δέντρου στόχων με hard goals και soft goals

Με τις συνεισφορές προσπαθούμε να μοντελοποιήσουμε περίπλοκες σχέσεις μεταξύ στόχων. Μια συνεισφορά μεταξύ δύο στόχων ορίζεται ως η δυνατότητα ενός στόχου να συνεισφέρει θετικά ή αρνητικά στην επιτυχία ή αποτυχία ενός άλλου στόχου. Οι συνεισφορές μεταξύ στόχων

μπορούν να καταταχθούν κάτω από τις επόμενες κατηγορίες [19] [20]:

$$\alpha \rightarrow \beta, \text{ equivalent to } ++S(\alpha, \beta).$$

$$\alpha \rightarrow \neg\beta, \text{ equivalent to } --S(\alpha, \beta)$$

$$\neg\alpha \rightarrow \beta, \text{ equivalent to } --D(\alpha, \beta)$$

$$\neg\alpha \rightarrow \neg\beta, \text{ equivalent to } ++D(\alpha, \beta)$$

Σχήμα 2.8: Τύποι συνεισφορών μεταξύ στόχων

Η συνεισφορά ++S (a, β) εκφράζει ότι η επιτυχία του στόχου α συνεπάγεται την επιτυχία του στόχου β.
Η συνεισφορά --S (α,β) εκφράζει ότι η επιτυχία του στόχου α συνεπάγεται την αποτυχία του στόχου β.
Η συνεισφορά ++ D (α,β) εκφράζει ότι η αποτυχία του στόχου α συνεπάγεται την επιτυχία του στόχου β.
Η συνεισφορά -- D (α,β) εκφράζει ότι η αποτυχία του στόχου α συνεπάγεται την αποτυχία του στόχου β.

Στην παρούσα διπλωματική χρησιμοποιούμε τα δέντρα στόχων για τη μοντελοποίηση των συστημάτων λογισμικού, καθώς και των απαραίτητων πληροφοριών που απαιτούνται για την δημιουργία μίας υπόθεσης σφάλματος (alarms) και την επαλήθευσή της (στρατηγικές) . Οι απαραίτητες πληροφορίες έχουν μοντελοποιηθεί σαν βοηθητικοί κόμβοι πάνω στους κόμβους - στόχους του δέντρου στόχων. Έτσι, η ρίζα του δέντρου στόχου αντιπροσωπεύει μία απαίτηση (requirement) του συστήματος που μοντελοποιείται, η οποία συνδέεται με μία επισημείωση (alarm) υπόθεσης σφάλματος. Δεδομένης μιας υπόθεσης σφάλματος, το προς απόδειξη δέντρο στόχων που ορίσαμε μας δίνει όλους τους δυνατούς τρόπους απόδειξης της υπόθεσης αυτής, που ισοδυναμεί με τους δυνατούς συνδυασμούς υποστόχων που πρέπει να επαληθευτούν ή να διαψευθούν. Τέλος, οι υποστόχοι προς επαλήθευση , είναι συνδεδεμένοι με βοηθητικούς κόμβους- στρατηγικές, οι οποίοι αποτελούν αλγόριθμους απόδειξης της ισχύος των αντίστοιχων υποστόχων.

2.4 Πλεονεκτήματα δέντρων Στόχων

Το περιβάλλον πλαίσιο της παρούσας διπλωματικής, χρησιμοποιεί για την μοντελοποίηση των δεδομένων την αναπαράσταση των στόχων- απαιτήσεων ως δέντρα στόχων, λόγω των παρακάτω πλεονεκτημάτων που προσφέρει:

- Υπάρχει μία άμεση σύνδεση με την άλγεβρα Boole [21] και την λογική πρώτης τάξης [22]. Τα

δέντρα στόχων, αποτελούμενα από λογικές συζεύξεις και διαζεύξεις, μπορούν πολύ εύκολα να μετατραπούν σε ακολουθίες λογικής πρώτης τάξης, καθώς και να αναλυθούν με χρήση της θεωρίας των μαρκοβιανών δικτύων [23] και άλλων μαθηματικών προσεγγίσεων.

- Υπάρχει μία πληθώρα εργαλείων και αλγορίθμων, διαθέσιμα για την επεξεργασία των προβλημάτων της άλγεβρας Boole, γεγονός το οποίο καθιστά ευκολότερη την μελέτη τους.

- Υπάρχει δυνατότητα επέκτασης των δέντρων στόχων με επιπλέον σχέσεις μεταξύ κόμβων - στόχων, διευκολύνοντας την επισύναψη οποιασδήποτε πληροφορίας απαραίτητης για την αναπαράσταση οποιουδήποτε συστήματος [24]. Στο περιβάλλον πλαίσιο που περιγράφεται στη παρούσα διπλωματική εργασία, κάνουμε χρήση των επεκτάσεων αποσύνθεσης decomposition) και συνεισφοράς (contribution) .

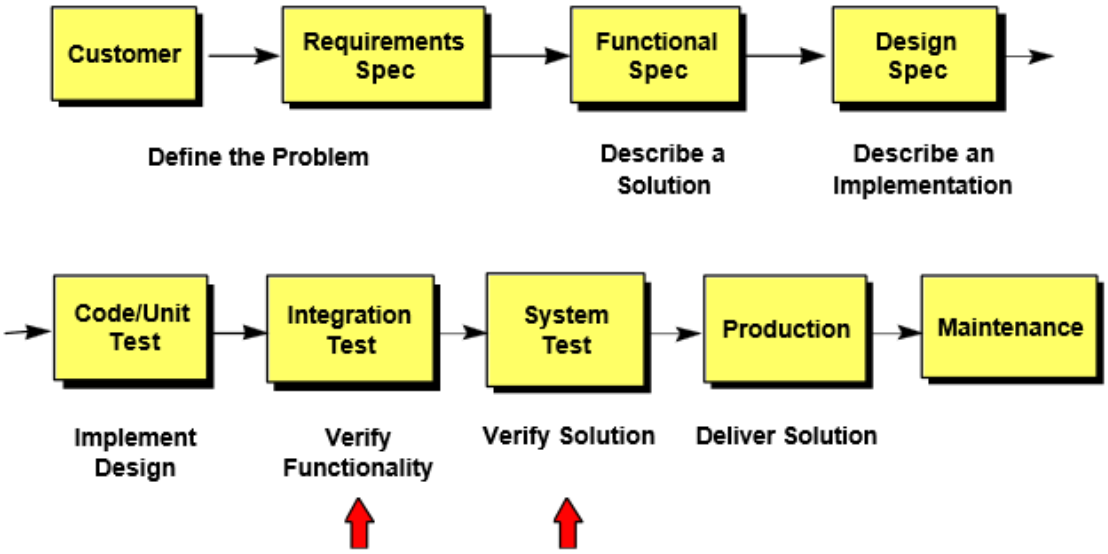
- Τα δέντρα στόχων είναι εύκολα διαχειρίσιμα προγραμματιστικά, λόγω της άμεσης σύνδεσής τους με τις θεωρίες, τις μεθοδολογίες και τα εργαλεία που παρέχουν δυνατότητες επεξεργασίας βασισμένες σε λογικές επαγωγές.

2.5 Model Based Testing

Η χρήση ενός μοντέλου για να περιγράψει την συμπεριφορά ενός συστήματος έχει αποδειχτεί σημαντικό πλεονέκτημα για τις ομάδες ελέγχου σφαλμάτων λογισμικού. Τα μοντέλα μπορούν να χρησιμοποιηθούν με πολλούς τρόπους κατά τη διάρκεια ζωής ενός προϊόντος λογισμικού, για παράδειγμα για τη βελτίωση της ποιότητας των προδιαγραφών και απαιτήσεων του λογισμικού, την παραγωγή κώδικα, την ανάλυση αξιοπιστίας, και την παραγωγή αλγορίθμων δοκιμών. Αυτή η παράγραφος θα επικεντρωθεί στα οφέλη της χρήσης μοντέλων στη διαδικασία ελέγχου λογισμικού και σε πιθανές προκλήσεις που εμπεριέχει [25].
Τα μοντέλα χρησιμοποιούνται σε πολλούς ερευνητικούς κλάδους για την κατανόηση, τον σχεδιασμό και την ανάπτυξη συστημάτων . Από την έρευνα DNA μέχρι τον σχεδιασμό αεροσκαφών, γίνεται ολοένα και μεγαλύτερη χρήση μοντέλων , καθώς διευκολύνουν την κατανόηση σύνθετων συστημάτων και παρέχουν ένα επαναχρησιμοποιήσιμο πλαίσιο για την ανάπτυξη προϊόντων. Στον κλάδο μηχανικής λογισμικού (Software Engineering) , τα μοντέλα είναι πλέον αποδεκτά ως μέρος μίας σύγχρονης αντικειμενοστραφούς ανάλυσης και σχεδιαστικής προσέγγισης (Object Oriented methodology) [26] [27]. Τις δύο τελευταίες δεκαετίες, έχει γραφτεί μία πληθώρα βιβλίων και έργων σχετικά με την εφαρμογή των μοντέλων στην ανάλυση αξιοπιστίας συστημάτων. Παρόλα αυτά, η διαδικασία ελέγχου λογισμικού, με εξαίρεση τις κορυφαίες εταιρείες, συνεχίζει να είναι ως επί το πλείστον μία χειροκίνητη διαδικασία. Στόχος αυτής της διπλωματικής εργασίας είναι να παρουσιάσει μια προσέγγιση για τον έλεγχο ενός συστήματος , βασισμένη σε ένα γραφικό μοντέλο που περιγράφει τη συμπεριφορά του συστήματος προς δοκιμή.
Η μοντελοποίηση είναι ένας πολύ οικονομικός τρόπος για την σύλληψη της γνώσης για ένα σύστημα και στη συνέχεια την επαναχρησιμοποίηση αυτή τη γνώση όπως το σύστημα

μεγαλώνει. Με την κατασκευή ενός μοντέλου που καθορίζει την επιθυμητή συμπεριφορά ενός συστήματος για συγκεκριμένες εισόδους σε αυτό, η υπεύθυνη ομάδα ελέγχου λογισμικού έχει πλέον έναν μηχανισμό για μια δομημένη ανάλυση του συστήματος. Τα σενάρια ελέγχου μπορούν πλέον να περιγραφούν ως μια ακολουθία ενεργειών πάνω στο σύστημα, με τις σωστές απαντήσεις του συστήματος να είναι ήδη καταγεγραμμένες στο μοντέλο [28].
Το μεγαλύτερο όφελος είναι η επαναχρησιμοποίηση. Όλη αυτή η εργασία δεν έχει χαθεί. Ο επόμενος κύκλος δοκιμής μπορεί να ξεκινήσει από εκεί που σταμάτησε. Εάν το προϊόν χρειάζεται νέα χαρακτηριστικά και λειτουργίες, μπορούν να προστεθούν σταδιακά στο μοντέλο. Αν η ποιότητα πρέπει να βελτιωθεί, το μοντέλο μπορεί να βελτιωθεί και οι έλεγχοι να επεκταθούν. Εάν υπάρχουν νέοι άνθρωποι στην ομάδα, μπορούν γρήγορα να κατανοήσουν το σύστημα, μελετώντας το μοντέλο του.

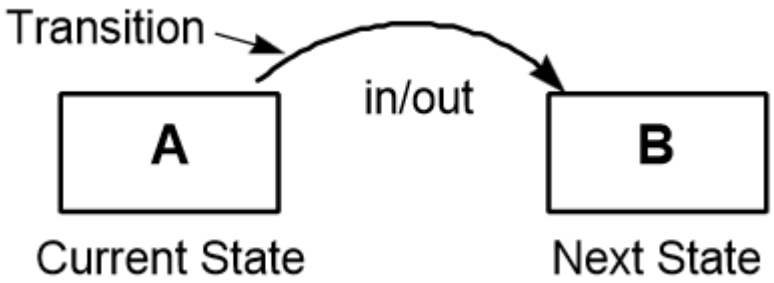
Αυτή η διπλωματική εργασία υποθέτει ότι το λογισμικό που πρόκειται να δοκιμαστεί είναι στη φάση "Integration Test" ή "System Test" της διαδικασίας ανάπτυξης [29] [30] (βλέπε Σχήμα 2.9). Ο στόχος της δοκιμής είναι να εξασφαλιστεί ότι το σύστημα πληροί τις απαιτήσεις του από εξωτερική σκοπιά. Δε θα εστιάσουμε στην εφαρμογή, αλλά στην αξιολόγηση του συστήματος από τους χρήστες. Οι δοκιμές θα μετρήσουν τη συνολική λειτουργική συμμόρφωση του συστήματος με τις αρχικές προδιαγραφές και απαιτήσεις. Οι δοκιμές αυτές είναι γνωστές και ως δοκιμές αποδοχής (Acceptance Tests) και είναι μια παραδοσιακή εφαρμογή δοκιμών "μαύρου κουτιού" (Black Box Testing).



Σχήμα 2.9: Διαδικασία ανάπτυξης λογισμικού

Το πρώτο εμπόδιο που πρέπει να ξεπεραστεί στην ανάπτυξη σεναρίων ελέγχου χρησιμοποιώντας μοντελοποίηση συστήματος, είναι ο προσδιορισμός του στόχου υπό έλεγχο. Είναι λοιπόν βασικό να υπάρχει διαθέσιμη μία αναλυτική περιγραφή της συμπεριφοράς του συστήματος. Ο επίσημος τρόπος προσδιορισμού της συμπεριφοράς του συστήματος είναι η προδιαγραφή των απαιτήσεων (Requirement Specification) [31] . Εάν δεν έχουν προσδιοριστεί πλήρως οι απαιτήσεις του

συστήματος, τότε δεν είναι δυνατή η ανάπτυξη όλων των δυνατών συνδυασμών σεναρίων ελέγχου. Παρόμοιο πρόβλημα προκύπτει εάν η διατύπωση των απαιτήσεων είναι αμφιλεγόμενη , καθώς μπορεί να υπάρξει διαφορετική κατανόηση του σεναρίου ελέγχου από διαφορετικά άτομα. Σε ένα πρόσφατο συνέδριο Μηχανικής απαιτήσεων (Requirements Engineering) , έγγραφα και παρουσιάσεις έδειξαν πως η βασική αιτία του 60-80% όλων των ελαττωμάτων ήταν η ύπαρξη λαθών στις απαιτήσεις προδιαγραφών [32].
Εφαρμόζοντας ένα μοντέλο στο αρχικό επίπεδο της διαδικασίας ανάπτυξης λογισμικού μπορεί να μειώσει δραματικά την ασάφεια - και, ως εκ τούτου, τα σφάλματα. Η μοντελοποίηση σε επίπεδο συμπεριφοράς είναι πολύ παρόμοια με flowcharting, καθώς οι σημαντικότερες λειτουργίες του προϊόντος ορίζονται σε γραφική μορφή, ως μία ακολουθία ενεργειών. Οι περισσότερες τεχνικές μοντελοποίησης υποστηρίζουν την ιδέα ότι υπάρχουν πολλές πιθανές "επόμενες" ενέργειες. Πολλές μεθοδολογίες βασίζονται στην έννοια της μηχανής καταστάσεων (State Machine) (βλέπε σχήμα 2.10), όπου τα βέλη στο διάγραμμα αντιστοιχούν στις ενέργειες, ενώ τα εικονίδια με μια ποικιλία σχημάτων αντιπροσωπεύουν τις διάφορες καταστάσεις του συστήματος [33] [34].



Σχήμα 2.10: Έννοια Μηχανής καταστάσεων

Η ανάπτυξη των απαιτήσεων ενός συστήματος με τη βοήθεια ενός μοντέλου, ακόμη και αν γίνει αργά στη διαδικασία, είναι ένα πολύ αποτελεσματικό μέσο για να 1) ανακαλυφθούν ελαττώματα του συστήματος (πολλά γίνονται ορατά από την προσπάθεια μοντελοποίησης και μόνο), 2) τον άμεσο καθορισμό της βάσης για τα σενάρια ελέγχου της χρήσης του συστήματος, και 3) τη διατήρηση των παραπάνω για μελλοντική ανάπτυξη του συστήματος ή εφαρμογή σε άλλα παρόμοια συστήματα.

Εφόσον οι απαιτήσεις του συστήματος έχουν πλήρως καθοριστεί όπως περιγράψαμε παραπάνω, το επόμενο σημαντικό βήμα είναι ο σχεδιασμός των σεναρίων ελέγχου (Test Scenarios) . Ο στόχος των σεναρίων ελέγχου είναι να βεβαιωθούμε ότι το σύστημα θα συμπεριφερθεί σωστά όταν εκτελείται μία ακολουθία ενεργειών του χρήστη. Τα σενάρια ελέγχου αποτελούνται συνήθως από τις ακόλουθες ενέργειες [35] [36]:

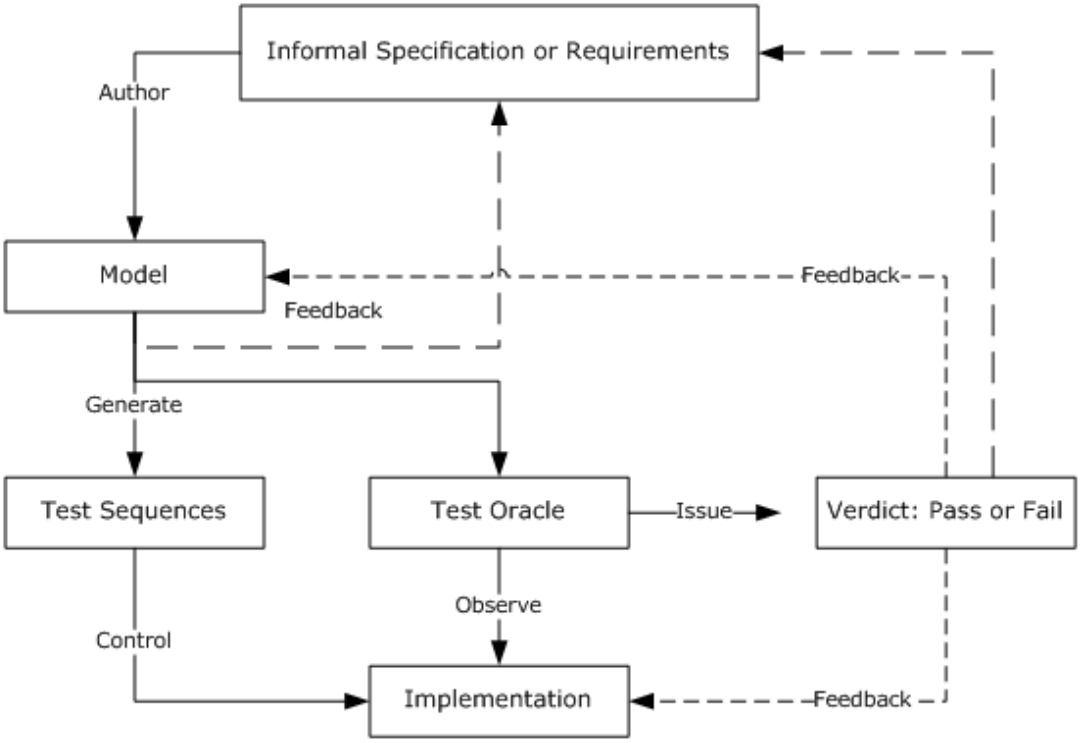
1) Να δοθεί κάποιο ερέθισμα στο σύστημα, ώστε να προκληθεί μετάβαση από την μία κατάσταση του συστήματος σε μία άλλη. Μπορεί να είναι μια ενέργεια του χρήστη, όπως το πάτημα ενός κουμπί σε ένα γραφικό περιβάλλον (GUI), ή την κλήση ενός αριθμού τηλεφώνου . Η ενέργεια μπορεί να ελέγχεται άμεσα από το περιβάλλον εκτέλεσης δοκιμών.

2) Να ελεγχθεί ότι το σύστημα ανταποκρίθηκε σωστά. Η επαλήθευση μπορεί να επιτευχθεί συγκρίνοντας την απόκριση του συστήματος με την αναμενόμενη απόκριση, συμπεραίνοντας αν το σύστημα βρίσκεται ή όχι στην σωστή κατάσταση.

3) Να ρυθμιστεί το περιβάλλον δοκιμών του συστήματος. Οι δοκιμές θα πρέπει συχνά να ελέγχουν το περιβάλλον, έτσι ώστε η επόμενη δράση να ακολουθήσει μια προβλέψιμη πορεία. Σε πολλά περιβάλλοντα, υπάρχουν καταστάσεις που έχουν διάφορα πιθανά αποτελέσματα με την ίδια είσοδο. Για να γίνει αυτή η τελική κατάσταση του συστήματος ντετερμινιστική, το περιβάλλον μπορεί να περιοριστεί προκειμένου να το εξαναγκάσουμε σε μία συγκεκριμένη κατάσταση. Για παράδειγμα, να ρυθμίσουμε μία τηλεφωνική γραμμή ως "απασχολημένη", έτσι ώστε να μπορέσουμε να ελέγξουμε τη λειτουργία προώθησης κλήσεων .

4) Να καταγραφούν τα αποτελέσματα. Ανάλογα με το βαθμό της αυτοματοποίησης κατά την εκτέλεση της δοκιμής, μπορεί να γίνει καταγραφή των αποτελέσματα των δοκιμών σε ένα σύστημα αναφοράς.

Το επόμενο σχήμα (σχήμα 2.11) [37] , δείχνει μία απλουστευμένη μορφή της διαδικασίας δοκιμών με χρήση της μοντελοποίησης συστημάτων, που περιγράψαμε παραπάνω :



Σχήμα 2.11:Διαδικασία δοκιμών με χρήση της μοντελοποίησης συστημάτων

Τα γενικά πλεονεκτήματα του Model Based Testing, μπορούν να συνοψιστούν ως εξής :

1) Οι απαιτήσεις του συστήματος προσδιορίζονται μία φορά και ευκολότερα.

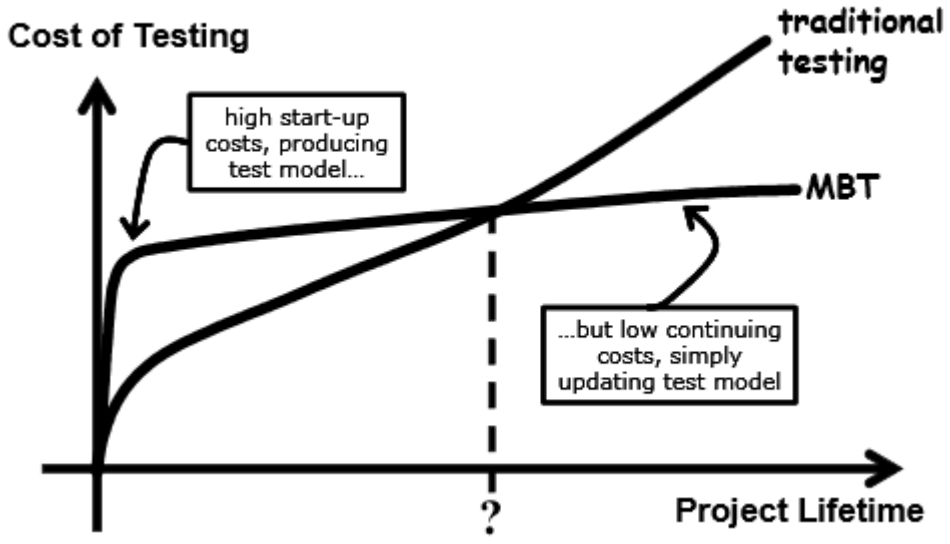
2) Το κόστος συντήρησης είναι χαμηλότερο, καθώς υπάρχει δυνατότητα επαναχρησιμοποίησης των σεναρίων δοκιμών, καθώς και ολόκληρου του μοντέλου.

3) Ο σχεδιασμός είναι επεκτάσιμος, όταν νέα χαρακτηριστικά και λειτουργίες προστίθενται στο σύστημα. Μία αλλαγή μπορεί να εφαρμοστεί σε όλα τα υπάρχοντα σενάρια ελέγχου, χωρίς να χρειαστεί να επανασχεδιαστούν.

4) Ορθότερος σχεδιασμός στην αρχή της ανάπτυξης ενός συστήματος συνεπάγεται μικρότερη πολυπλοκότητα και ποσότητα κώδικα.

5) Υψηλό ποσοστό κάλυψης ελέγχου του συστήματος, καθώς υποστηρίζεται η συνεχής δημιουργία σεναρίων ελέγχου με σκοπό την συνεχή εύρεση σφαλμάτων στο σύστημα.

Πριν ξεκινήσει κανείς τη χρήση μοντέλων για τον έλεγχο ενός συστήματος, πρέπει να είναι βέβαιος ότι η προσέγγιση αυτή είναι κατάλληλη για την περίπτωσή τους. Η δημιουργία των μοντέλων μπορεί να έχει μεγάλο κόστος, αλλά αυτό τις περισσότερες φορές καλύπτεται από το χαμηλότερο κόστος συντήρησης όταν το σύστημα είναι σε λειτουργία [38] (βλέπε σχήμα 2.12).



Σχήμα 2.12: Κόστος Ελέγχου συστημάτων σε συνάρτηση με τον κύκλο ζωής τους

2.6 Root Cause Analysis

Ο μηχανισμός ανάλυσης αιτίας σφάλματος (root cause analysis), είναι μια διαδικασία που χρησιμοποιείται για την αναζήτηση και κατηγοριοποίηση των κύριων αιτιών διαφόρων περιστατικών (events) τα οποία θέτουν σε κίνδυνο ένα προϊόν λογισμικού ή μια εφαρμογή. Με την χρήση της λέξης "περιστατικό" αναφερόμαστε σε οποιοδήποτε συμβάν έχει επιπτώσεις στην ποιότητα , την αξιοπιστία ή την ασφάλεια ενός προϊόντος λογισμικού. Με απλά λόγια η διαδικασία Root Cause Analysis (RCA) είναι το σύνολο των μεθόδων μέσω των οποίων προσδιορίζουμε όχι μόνο το "τι" και το "πως" συνέβη ένα περιστατικό , αλλά και το "γιατί" συνέβη. Η πλήρης κατανόηση της αιτίας ύπαρξης ενός περιστατικού ("γιατί"), οδηγεί όχι μονό στην εύρεση αποτελεσματικών τρόπων επίλυσης του προβλήματος, αλλά και σε αποτελεσματικές προτάσεις αποφυγής παρόμοιων περιστατικών. Έτσι με τη χρήση της διαδικασίας ανάλυσης αιτίας σφάλματος καθ' όλη τη διάρκεια ζωής ενός προϊόντος λογισμικού μπορεί να επιτευχθεί σημαντική βελτίωση σε επίπεδο ποιότητας και αξιοπιστίας του, κάτι που σήμερα στο χώρο της αγοράς λογισμικού αποτελεί ίσως των σημαντικότερο παράγοντα που καθιστά ένα προϊόν λογισμικού ανταγωνιστικό. Ένα επιπλέον όφελος μιας αποτελεσματικής RCA ανάλυσης είναι ότι σε βάθος χρόνου αναγνωρίζονται οι πιο συχνές αιτίες σφαλμάτων, δίνοντας τη δυνατότητα κατηγοριοποίησης τους και περεταίρω βελτίωσης της ποιότητας του λογισμικού.

Προκειμένου να βρούμε την κύρια αιτία ενός σφάλματος είναι αναγκαίο αρχικά να ορίσουμε την έννοια "κύρια αιτία" (" root cause"). Παρακάτω δίνονται οι ορισμοί του όρου "root cause"[44]:

- Κύριες αιτίες , είναι συγκεκριμένες και βαθύτερες αιτίες.
  Στόχος του ερευνητή είναι να εντοπίσει όσο πιο συγκεκριμένα την αιτία του προβλήματος . Όσο πιο συγκεκριμένη είναι η αιτία τόσο πιο συγκεκριμένος θα είναι και ο τρόπος αντιμετώπισης του προβλήματος και η αποτροπή επανεμφάνισης του.
- Κύριες αιτίες , είναι εκείνες οι αιτίες που μπορούν εύλογα να εντοπιστούν.
  Η ανάλυση RCA μπορεί ναι είναι σε βάθος χρόνου οικονομικά επωφελής, ωστόσο δεν είναι πρακτικό και επικερδές η επ' αόριστον αναζήτηση της ρίζας ενός προβλήματος.
- Κύριες αιτίες , είναι εκείνες οι αιτίες οι οποίες μπορούν να αντιμετωπιστούν.

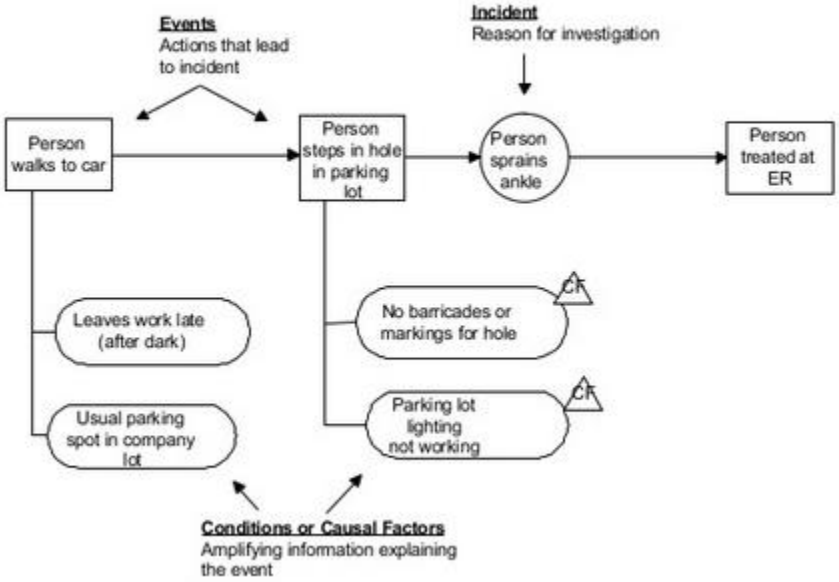
- Κύριες αιτίες, είναι οι αιτίες για τις οποίες μπορούν να εφαρμοστούν μέθοδοι αποτροπής της επανεμφάνισης τους.
  Οι μέθοδοι αποτροπής πρέπει να είναι συγκεκριμένες και να αντιμετωπίζουν τα βαθύτερα αίτια του προβλήματος.

Η ανάλυση RCA είναι μια διαδικασία τεσσάρων βημάτων [45][47]. Τα βήματα αυτά είναι τα εξής:

- Συλλογή δεδομένων

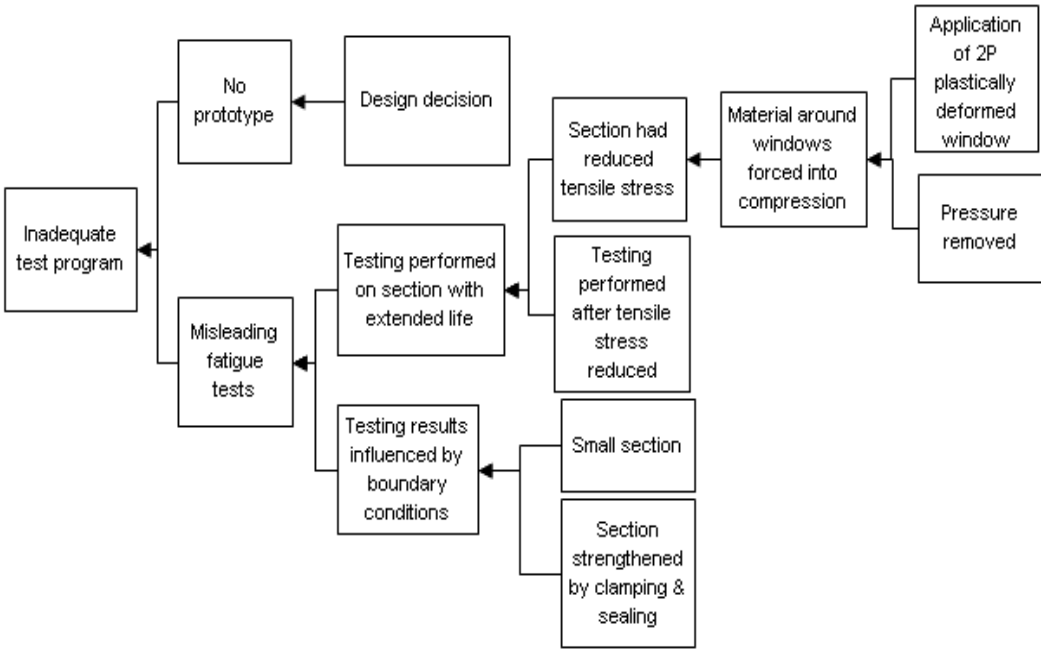
Το πρώτο βήμα της ανάλυσης είναι η συγκέντρωση στοιχείων. Χωρίς πλήρη ενημέρωση και κατανόηση του περιστατικού οι γενικές αιτίες και οι κύριες αιτίες που οδήγησαν στο περιστατικό είναι δύσκολο να εντοπισθούν. Το μεγαλύτερο μέρος του χρόνου που πρέπει να αφιερώνεται στην RCA ανάλυση [69] πρέπει να είναι στη συλλογή των κατάλληλων δεδομένων.

- Γραφήματα αιτιών
  Τα γραφήματα αιτιών δίνουν στους ερευνητές την υποδομή που απαιτείται για την οργάνωση και την ανάλυση των πληροφοριών-αιτιών που έχουν συγκεντρώσει. Το διαγράμματα αιτιών είναι ένα ακολουθιακό διάγραμμα , το οποίο περιέχει τις περιγραφές των περιστατικών καθώς και τις συνθήκες του περιβάλλοντος τη στιγμή που εμφανίστηκε το πρόβλημα (σχήμα 2.13)[46] . Η προετοιμασία του γραφήματος αιτιών θα πρέπει να ξεκινήσει τι στιγμή που οι ερευνητές ξεκινούν να συλλέγουν στοιχεία για την εύρεση της κύριας αιτίας του προβλήματος. Η δημιουργία του γραφήματος ξεκινά με ένα απλό σκελετό του διαγράμματος και ανανεώνεται όσο συνεχίζει η συλλογή δεδομένων. Η διαδικασία αυτή σταματά όταν οι ερευνητές είναι σε θέση να προσδιορίσουν την βασική αιτία του περιστατικού.



Σχήμα 2.13: Γράφημα αιτιών

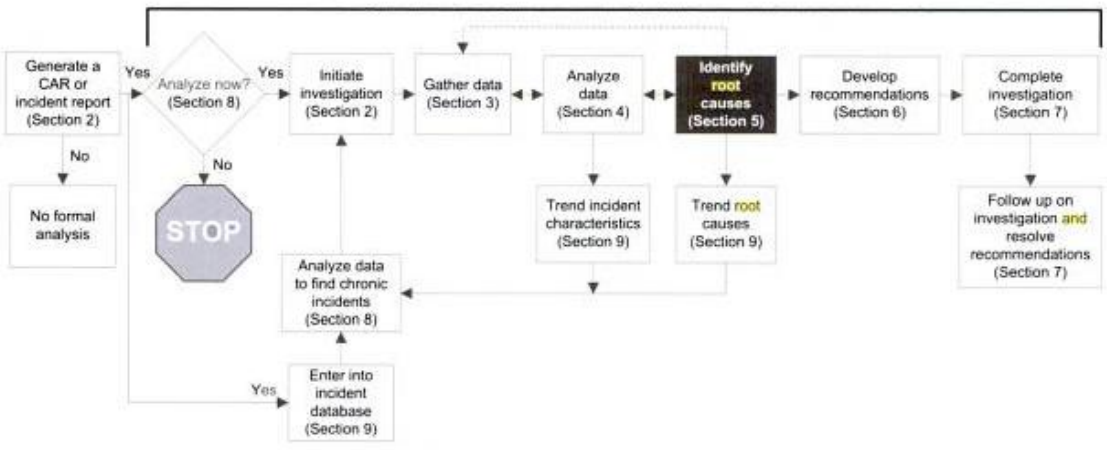
- Προσδιορισμός κύριας αιτίας
  Το στάδιο αυτό περιλαμβάνει τη χρήση ενός διαγράμματος αποφάσεων (Σχήμα 2.14) [46]. Στο διάγραμμα αυτό ο αναλυτής προσδιορίζει τον λόγο για τον οποίο παρουσιάστηκε το πρόβλημα και γιατί δεν είχε εντοπιστεί νωρίτερα.



Σχήμα 2.14 : Root Cause Map

- Δημιουργία και εφαρμογή της μεθόδου αποτροπής επανεμφάνισης του προβλήματος . Στο τελικό αυτό στάδιο προτείνονται οι μέθοδοι αποτροπής της επανεμφάνισης του συμβάντος. Οι μέθοδοι αυτοί πρέπει να είναι υλοποιήσιμοι καθώς και να έχει προσδιοριστεί με αυστηρότητα η ημερομηνία της εφαρμογής του.

Η αποτελεσματικότητα της διαδικασίας RCA είναι άμεσα συνδεδεμένη με αφορμή των προηγούμενων βημάτων. Στο σχήμα 2.15 φαίνονται τα στάδια της διαδικασίας που περιγράφτηκαν παραπάνω.



Σχήμα 2.15: Γράφημα μεθοδολογίας

Βλέπουμε λοιπόν πως η ανάλυση των βαθύτερων αιτίων και η διορθωτική διαδικασία δράσης είναι απολύτως απαραίτητα για τη βελτίωση της ποιότητας του συστήματος διαχείρισης και της ποιότητας του τελικού προϊόντος ή της υπηρεσίας.

Στα πλαίσια αυτής της διπλωματικής θα γίνει μια προσπάθεια εφαρμογής της RCA ανάλυσης με χρήση ενός αυτοματοποιημένου τρόπου. Χρησιμοποιώντας το περιβάλλον-πλαίσιο EMF και των εργαλείων του, θα δημιουργήσουμε στιγμιότυπα του MOF μοντέλου σε μορφή δέντρου όπου οι κόμβοι θα αντιστοιχίζονται στις απαιτήσεις λειτουργίας ενός συστήματος. Έτσι, στην περίπτωση ενός λάθους του συστήματος θα δίνεται άμεσα στο χρήστη η δυνατότητα έλεγχου της ισχύς των απαιτήσεων και η εύρεση της κύριας αιτίας του λάθους. Ο έλεγχος των απαιτήσεων μπορεί να γίνε είτε στατικά είτε δυναμικά , όπως περιγράφεται στις επόμενες δυο υπο-ενότητες. Βλέπουμε λοιπόν πως η ανάλυση των βαθύτερων αιτίων και ή διορθωτική διαδικασία δράσης είναι απολύτως απαραίτητα για τη βελτίωση της ποιότητας του συστήματος διαχείρισης και την και της ποιότητας του τελικού προϊόντος ή υπηρεσίας.

2.7 Στατική ανάλυση (Offline testing)

Στατική ανάλυση (Static testing) καλείται η διαδικασία ελέγχου ενός προϊόντος λογισμικού πριν το λογισμικό εκτελεστεί[48][50]. Η τεχνική αυτή χρησιμοποιείται για να ανιχνευτούν λάθη στον κώδικα του λογισμικού στα αρχικά στάδια κατασκευής του λογισμικού. Συνήθως, στα πρώιμα στάδια του κύκλου ζωής του λογισμικού, το μόνο που υπάρχει διαθέσιμο είναι τα έγγραφα των κατασκευαστικών απαιτήσεων και ο πηγαίος κώδικας τα οποία θα καθορίσουν τη σχεδίαση του

προϊόντος. Αυτά είναι και τα στοιχεία που δοκιμάζονται στην στατική ανάλυση, με στόχο την ανίχνευση ελαττωμάτων στον κώδικα και ασάφειες στα έγγραφα των προδιαγραφών του λογισμικού τα οποία μπορούν να προκαλέσουν λάθη κατά την ανάπτυξη του λογισμικού.

Οι τεχνικές στατικές ανάλυσης περιλαμβάνουν τα στάδια της αξιολόγησης και της στατικής ανάλυσης:

- Αξιολόγηση
  Στο στάδιο της αξιολόγησης, ελέγχεται οποιοδήποτε προϊόν του οποίου η εργασία κατασκευής δε γίνεται με κάποιο αυτοματοποιημένο τρόπο. Τέτοιες εργασίες μπορεί να είναι η κατασκευή εγγράφων προδιαγραφών και απαιτήσεων. Στόχος της αξιολόγησης είναι να διαπιστωθούν τυχόν ελαττώματα όσο το δυνατόν νωρίτερα και να διορθωθούν στο αρχικό αυτό στάδιο για την αποφυγή μεταγενέστερων προβλημάτων.
  Για παράδειγμα, αν ένα προϊόν λογισμικού δεν ανταποκρίνεται στα πρότυπα που έχουν καθοριστεί στα διάφορα έγγραφα, έχει σοβαρές πιθανότητες να παρουσιάσει κάποιο πρόβλημα στη συνέχεια της διαδικασίας ανάπτυξης του λογισμικού. Στο στάδιο της αξιολόγησης το λογισμικό συγκρίνεται με τα πρότυπα αυτά έτσι ώστε σε περίπτωση ασυνέπειας το λογισμικό διορθώνεται. Σε αυτό το στάδιο το κόστος διόρθωσης του λογισμικού είναι πολύ μικρό. Το ίδιο ισχύει και για τις επιπτώσεις στην εταιρία κατασκευής του λογισμικού. Οι επιπτώσεις είναι σαφώς λιγότερες σε σύγκριση με αυτές που θα προέκυπταν αν η ίδια διόρθωση γινόταν μετά την απελευθέρωση του προϊόντος λογισμικού στην αγορά.
- Στατική ανάλυση
  Το στάδιο της στατικής ανάλυσης περιλαμβάνει την εξέταση πηγαίου κώδικα με την χρήση αυτοματοποιημένων εργαλείων που μειώνουν το χρόνο εργασίας και το κόστος του να προσπελαστεί και να ελεγχθεί ο κώδικας σε μικρά τμήματα χειροκίνητα.
  Τα αυτοματοποιημένα εργαλεία αυτά βοηθούν στην ανίχνευση των παραβιάσεων στα πρότυπα προγραμματισμού και στη σύνταξης του κώδικα. Η στατική ανάλυση ελέγχει τον πηγαίο κώδικα του λογισμικού πριν η κατασκευή του συνολικού συστήματος ολοκληρωθεί. Έτσι δε χρειάζεται να περιμένουμε την ολοκλήρωση του συστήματος για τον έλεγχο του πηγαίου κώδικα το οποίο είναι χρονοβόρο και δαπανηρό.

Μια άλλη πτυχή της στατικής ανάλυσης είναι ο έλεγχος των μοντέλων λογισμικού τα οποία ορίζουν το λογισμικό. Αυτά αποτελούν οπτικές παραστάσεις για το πώς τα διάφορα αντικείμενα λογισμικού συνδέονται μεταξύ τους στο σύνολο του λογισμικού. Αναλύοντας τα μοντέλα πριν την εφαρμογή τους βοηθά στην ανεύρεση κενών ή λογικών σφαλμάτων στον τρόπο τον οποίο φαίνεται να σχετίζονται τα αντικείμενα λογισμικού.

Η διαδικασία ανάλυσης του πηγαίου κώδικα λογισμικού και των μοντέλων χειροκίνητα είναι χρονοβόρα και επίπονη. Επιπλέον, είναι επιρρεπής σε παρερμηνείες. Ως εκ τούτου, η στατική ανάλυση αποτελεί τον πιο αποτελεσματικό τρόπο ελέγχου ενός λογισμικού στο αρχικό στάδιο της ανάπτυξης του, αφού χρησιμοποιεί αυτοματοποιημένα εργαλεία τα οποία επιθεωρούν των πηγαίο κώδικα και τα μοντέλα λογισμικού σύμφωνα με τα προκαθορισμένα πρότυπα και ελέγχουν αν υπάρχουν λογικές ή συντακτικές ατέλειες. Ορισμένοι μεταγλωττιστές μπορούν να

θεωρηθούν ως εργαλεία στατικής ανάλυσης, δεδομένου ότι πολλοί από αυτούς κάνουν αναφορά για συντακτικά λάθη ή άλλα ελαττώματα.

Τα εργαλεία στατικής ανάλυσης μπορεί να ανιχνεύσουν κρίσιμα ελαττώματα που υπάρχουν στον κώδικα ή στα μοντέλα λογισμικού , όπως:

- Ακαθόριστες τιμές στις μεταβλητές του κώδικα.
- Απροσπέλαστος ή "νεκρός" κώδικας.
- Λάθη σε επίπεδο διεπαφών.
- Μη ορθή τήρηση των προτύπων.
- Περιπτώσεις ευπαθούς-τρωτού κώδικα.
- Συντακτικά λάθη της γλώσσας προγραμματισμού.

Συμπερασματικά, η στατική ανάλυση είναι μια σημαντική τεχνική στον έλεγχο ενός λογισμικού για την ανίχνευση ελαττωμάτων πριν από το χρόνο εκτέλεσης. Η στατική ανάλυση προτείνεται να χρησιμοποιείται αμέσως μετά την κατασκευή των πρώτων τμημάτων του συστημάτων και κατά την ανάπτυξη των μοντέλων λογισμικού. Μπορούν να ανιχνεύσουν σφάλματα λογικής και ασάφειες σε μια εφαρμογή. Αυτά τα εργαλεία αναλύουν τον πηγαίο κώδικα και ελέγχουν την τήρηση των μετρήσεων κώδικα (code metrics). Η μετρήσεις κώδικα βοηθούν στην κατανόηση της πολυπλοκότητας του κώδικα και τη διατήρηση της ποιότητας του προγράμματος λογισμικού.

2.8 Δυναμική ανάλυση(Dynamic Testing)

Δυναμική ανάλυση (Dynamic testing) καλείται η διαδικασία ελέγχου ενός συστήματος λογισμικού το οποίο απαιτεί την εκτέλεση μιας μονάδας του συστήματος [48] . Χρησιμοποιώντας δυναμικές μεθόδους έλεγχου, το σύστημα λογισμικού εκτελείται χρησιμοποιώντας ως είσοδο ένα συγκεκριμένο σύνολο τιμών. Στη συνέχεια το αποτέλεσμα της εκτέλεσης ελέγχεται και συγκρίνεται με το αναμενόμενο αποτέλεσμα σε κατάσταση ορθής λειτουργίας. Εάν το αποτέλεσμα της σύγκρισης δεν είναι το ίδιο, τότε σημαίνει πως βρέθηκε ένα πρόβλημα στο λογισμικό που ελέγχεται, το οποίο και καταγράφεται. Ο κύριος λόγος εφαρμογής της δυναμικής ανάλυσης είναι η ανεύρεση λαθών σε ένα σύστημα λογισμικού καθώς και η εξασφάλιση και η βελτίωση της ποιότητας του.

Το στάδιο αυτό της δυναμικής ανάλυσης εφαρμόζεται ακριβώς μετά το στάδιο της στατικής ανάλυσης. Ο δυναμικός έλεγχος και οι στατικές δοκιμές αποτελούν συμπληρωματικές μεθόδους καθώς τείνουν να βρίσκουν αποτελεσματικά και αποδοτικά αντίστοιχα, διαφορετικούς τύπους προβλημάτων σε ένα λογισμικό. Οι τύποι των ελαττωμάτων που μπορούν να βρεθούν μέσω της στατικής ανάλυσης είναι κυρίως ελλιπείς προδιαγραφές , αποκλίσεις από τα πρότυπα , σχεδιαστικά ελαττώματα, μη συντηρήσιμος κώδικας, και ασυνέπειες μεταξύ των προδιαγραφών διαφόρων τμημάτων του λογισμικού. Σε αντίθεση με αυτό, η διαδικασίας της δυναμικής

ανάλυσης εντοπίζει κυρίως λάθη και αδυναμίες εκτέλεσης λειτουργίας. Οι δύο αυτές τεχνικές μπορούν να χρησιμοποιηθούν παράλληλα για την επίτευξη κοινών στόχων , όπως είναι η αύξηση της αξιοπιστίας ενός προϊόντος λογισμικού, η εξαγωγή πληροφοριών σχετικά με την ποιότητα του προϊόντος και κυρίως στην εύρεση ελαττωμάτων στο λογισμικό πριν την απελευθέρωση του στην αγορά. Η δυναμική ανάλυση λειτουργεί είτε με την ενσωμάτωση κώδικα στο ελεγχόμενο λογισμικό κατά το χρόνο παρασκευής , ώστε να είναι δυνατή η εμφάνιση συγκεκριμένης πληροφορίας στην περίπτωση λάθους (η πιο κοινή προσέγγιση) , ή παρέχοντας ένα σύστημα προσομοίωσης της εσωτερικής λειτουργίας του υπό έλεγχο λογισμικού [49].

Στο δυναμικό έλεγχο υπάρχουν τρείς διαφορετικές τεχνικές για τον τρόπο σχεδιασμού των σεναρίων έλεγχου του συστήματος. Αυτές είναι :

- Σενάρια έλεγχου βασισμένα στις τεχνικές προδιαγραφές (specification based techniques). Οι τεχνικές αυτές έλεγχου ενός λογισμικού βασίζονται κυρίως στην εξέταση των απαιτήσεων ενός συστήματος αδιαφορώντας για το πως το λογισμικό του συστήματος επιτυγχάνει αυτή την ορθή λειτουργία. Συνηθίζεται για το λόγο αυτό να καλούνται τεχνικές "έλεγχου μαύρου κουτιού" ("black box testing"). Αυτές οι τεχνικές εξετάζουν εάν το αποτέλεσμα του συστήματος ανταποκρίνεται στην αναμενόμενη ορθή λειτουργία , όπως αυτή ορίζεται στις τεχνικές προδιαγραφές του συστήματος ή του μοντελοποιημένου συστήματος.
- Σενάρια έλεγχου βασισμένα στη δομή του συστήματος (structure-based techniques). Οι τεχνικές που ελέγχουν πως λειτουργεί το σύστημα στο εσωτερικό του κομμάτι, ονομάζονται , σενάρια έλεγχου βασισμένα στη δομή του συστήματος. Αυτές οι τεχνικές έρχονται σε αντίθεση με την τεχνική η οποία βασίζεται στις τεχνικές προδιαγραφές , καθώς πλέον εξετάζεται άμεσα ο τρόπος με τον οποίο το λογισμικό του συστήματος επιτυγχάνει την ορθή λειτουργία. Για το λόγο αυτό συνηθίζεται να καλούνται τεχνικές "έλεγχου άσπρου κουτιού" ("white box testing"). Οι τεχνικές δημιουργίας σεναρίων βασισμένα στην δομή ενός συστήματος είναι επίσης γνωστές ως "έλεγχος γυάλινου κουτιού" (glass-box testing) , "έλεγχος καθαρού κουτιού" (clear-box testing) και "έλεγχος μη-αδιαφανους κουτιού" (opaque-box testing) .
- Τεχνικές ελέγχου που βασίζονται στην εμπειρία (experience-based techniques). Στις τεχνικές αυτές η επιλογή του τρόπου κατασκευής των σεναρίων έλεγχου αφήνεται στην κρίση του εκάστοτε έμπειρου μηχανικού. Αυτή η τεχνική στηρίζεται στην γνώση των προγραμματιστών , των χρηστών και όλων των υπόλοιπων ατόμων που εμπλέκονται στην κατασκευή ενός λογισμικού και των σεναρίων ελέγχου του. Η εμπειρία του προσθέτει μια διαφορετική προοπτική στην διαδικασία ελέγχου του λογισμικού η οποία είναι αρκετά χρήσιμη σε περιπτώσεις όπου απαιτείται εξοικονόμηση χρόνου και ανθρώπινου δυναμικού. Εν γένει, οι τεχνικές αυτές χρησιμοποιούνται παράλληλα με τις τεχνικές ελέγχου που βασίζονται στις τεχνικές προδιαγραφές και στη δομή του λογισμικού.

2.9 Σύγκριση Στατικής και Δυναμικής ανάλυσης

Κατά τη διάρκεια της διαδικασίας ελέγχου του λογισμικού μπορούν να χρησιμοποιηθούν διάφορες τεχνικές για τον έλεγχο της ποιότητας και τον περιορισμό εμφάνισης σφαλμάτων που σχετίζονται με ένα προϊόν. Αυτές οι τεχνικές εξαρτώνται σε μεγάλο βαθμό από τον τρόπο με τον οποίο γίνεται ο έλεγχος και από την χρονική στιγμή του κύκλου ζωής του προϊόντος στην οποία εφαρμόζονται. Στα αρχικά στάδια της υλοποίησης ενός λογισμικού , ο έλεγχος εστιάζει στο αν ένα προϊόν πληροί τις απαραίτητες προϋποθέσεις και προδιαγραφές. Αντίθετα, ο έλεγχος στα τελευταία στάδια υλοποίησης γίνεται προκειμένου να ελεγχθεί αν υπάρχουν ελαττώματα κατά διάρκεια εκτέλεσης του λογισμικού. Σύμφωνα λοιπόν με την εκτέλεση ή όχι ενός λογισμικού μπορούμε να χωρίσουμε τις τεχνικές ελέγχου σε δύο μεγάλες κατηγορίες , τη στατική ανάλυση και τη δυναμική ανάλυση.
Όπως αναλύθηκε και στα προηγούμενα κεφάλαια, η στατική ανάλυση εφαρμόζεται στα αρχικά στάδια ανάπτυξης του λογισμικού, ενώ σε αντίθεση με την στατική ανάλυση , η δυναμική ανάλυση εφαρμόζεται στα τελευταία στάδια ανάπτυξης. Επιπλέον, η δυναμική ανάλυση περιλαμβάνει την εκτέλεση του λογισμικού . Το λογισμικό εκτελείται για να βεβαιωθεί ότι πληροί τις απαιτήσεις και ότι ταιριάζει με την έξοδο ορθής λειτουργίας σύμφωνα με τα σχέδια των μελετών. Η δυναμική ανάλυση πραγματοποιείται μόνο αφού το λογισμικό έχει αναπτυχθεί και έτσι εφαρμόζεται μόνο σε κώδικα λογισμικού και άλλα εκτελέσιμα προγράμματα.
Η στατική δοκιμή ανιχνεύει λάθη στα αρχικά στάδια της ανάπτυξης λογισμικού. Ως εκ τούτου, στατικές δοκιμές μπορούν να εφαρμοστούν σε οποιοδήποτε προϊόν σε περιπτώσεις αξιολόγησης των σεναρίων έλεγχου και των έγγραφων. Επειδή η στατική ανάλυση γίνεται νωρίς στη φάση της ανάπτυξης, τυχόν ελαττώματα εντοπίζονται νωρίς στο κύκλο ζωής του προϊόντος. Αυτό εξοικονομεί χρόνο και χρήματα μειώνοντας το μέγεθος της προσπάθειας που απαιτείται για επανεπεξεργασία. Η στατική ανάλυση ελέγχει τον κώδικα σύμφωνα με καθορισμένα πρότυπα πριν από την εκτέλεση του λογισμικού.
Συνοψίζοντας, η στατική ανάλυση διαφέρει από την δυναμική ανάλυση στα παρακάτω σημεία:

- Η στατική ανάλυση πραγματοποιείται στα πρώτα στάδια της ανάπτυξης του λογισμικού, ενώ η δυναμική ανάλυση πραγματοποιείται και τη διάρκεια εκτέλεσης του.
- Η στατική ανάλυση μπορεί να εφαρμοστεί σε οποιοδήποτε προϊόν λογισμικού , ενώ η δυναμική ανάλυση μπορεί να εφαρμοστεί μόνο σε εκτελέσιμα προγράμματα.
- Η στατική ανάλυση είναι πιο οικονομική και ωφέλιμη σε σχέση με την δυναμική ανάλυση.
- Η συμμόρφωση ως προς τα ορισμένα πρότυπα του λογισμικού μπορεί να ελεγχθεί μόνο κατά τη διάρκεια της στατικής ανάλυσης.
- Προβλήματα κατά την εκτέλεση ενός συστήματος λογισμικού μπορούν να εντοπιστούν μόνο και τη διάρκεια της δυναμικής ανάλυσης.

Γίνεται λοιπόν φανερό πως δεν μπορεί να προταθεί η χρήση της μιας τεχνικής έναντι της άλλης. Και οι δύο τεχνικές μπορούν και πρέπει να συνδυαστούν ώστε να επιτευχθεί ο αποτελεσματικός έλεγχος ενός συστήματος λογισμικού.
Στη παρούσα διπλωματική εργασία, γίνεται χρήση και των δύο αυτών τεχνικών ελέγχου. Συγκεκριμένα, στα πλαίσια της στατικής ανάλυσης γίνεται χρήση του πλαισίου ελέγχου μονάδων JUnit, ενώ στα πλαίσια της δυναμικής ανάλυσης πραγματοποιείται ανάλυση δεδομένων και καταγραφών παρακολούθησης (log analysis) του συστήματος, η οποία συγκρίνεται με την κατάσταση ορθής λειτουργίας .

2.10 Έλεγχος μονάδων λογισμικού (Unit Testing)

Η διαδικασία ελέγχου μονάδων (Unit Testing) [39] υπάρχει για πολλά χρόνια. Καθώς ο έλεγχος μονάδων αποτελεί ένα στάδιο ελέγχου που μπορεί να πραγματοποιηθεί σε πολύ αρχικό στάδιο ανάπτυξης λογισμικού, από παλιά έχει θεωρηθεί σημαντικός τρόπος ελέγχου ενός λογισμικού. Ωστόσο, στην πράξη, δεν έχουν όλα τα έργα λογισμικού την πολυτέλεια να μπορούν να χρησιμοποιήσουν αυτή την τεχνική ελέγχου. Αυτό μπορεί να αλλάξει, ειδικά για τα πολύπλοκα συστήματα, καθώς η ζήτηση για την παράδοση υψηλής ποιότητας λογισμικού ολοένα και αυξάνεται. Τα διεθνή πρότυπα, όπως το IEC 61508-3-, ISO / DIS-26262, απαιτούν τον έλεγχο μονάδων ενός συστήματος για τη εγγύηση ενός επιπέδου ασφαλείας που διασφαλίζει τη σωστή λειτουργία των συστημάτων.
Ουσιαστικά , ο έλεγχος μονάδων αφορά την απομόνωση μίας λειτουργίας ή μεθόδου μίας μονάδας του συστήματος , και την κλήση της με ένα συγκεκριμένο σύνολο παραμέτρων. Στη συνέχεια, όταν η εκτέλεση τελειώνει, το αποτέλεσμα αντιπαραβάλλεται με το αναμενόμενο αποτέλεσμα. Ο κώδικας που το επιτυγχάνει αυτό ονομάζεται μια δοκιμή (Test Case) [40]. Ο έλεγχος του αποτελέσματος γίνεται συνήθως με τη μορφή των ισχυρισμών.
Για παράδειγμα, ας υποθέσουμε ότι έχετε την ακόλουθη μέθοδο "foo" [41]:

```
int foo (int a, int b) {
      return b - a -1;
}
```

Σχήμα 2.16: Παράδειγμα μεθόδου προς έλεγχο

Μία δοκιμή θα μπορούσε να είναι:

```
void test_foo ()
{
      int Ret = foo(1,2);
      assertTrue(ret = 0, "Wrong value returned!");
}
```

Σχήμα 2.17: Παράδειγμα ελέγχου μεθόδου

Υπάρχει ένα πλήθος από οφέλη που συνδέονται με τον έλεγχο μονάδας. Όταν εκτελείται ένα σενάριο δοκιμής, ο έλεγχος γίνεται σε πολύ χαμηλό επίπεδο του συστήματος. Έτσι είναι δυνατόν να ελεγχθούν τμήματα του κώδικα που συνήθως δεν καλύπτονται από υψηλού επιπέδου λειτουργικές δοκιμές (functional testing), προσφέροντας μεγαλύτερη κάλυψη ελέγχου του

συστήματος.
Το δεύτερο σημαντικό όφελος προκύπτει από το γεγονός ότι αυτό το είδος ελέγχου λογισμικού αναγκάζει τον προγραμματιστή να γράψει "ελέγξιμο" κώδικα. Αυτό οδηγεί συνήθως σε κώδικα που δεν είναι υπερβολικά περίπλοκος, και είναι καλύτερα σχεδιασμένος.
Ένα άλλο πλεονέκτημα είναι ότι τα σενάρια ελέγχου μονάδας δημιουργούν ένα μεγάλο δίχτυ ασφαλείας για το λογισμικό, καθώς είναι δυνατή η τροποποίηση του κώδικα με τη σιγουριά ότι αν σπάσει κάτι, θα γίνει εμφανές αμέσως. Αυτό μεταφράζεται σε μεγαλύτερη παραγωγικότητα και καλύτερο κώδικα.
Τέλος, ο έλεγχος μονάδων αποκαλύπτει σφάλματα πολύ νωρίς στον κύκλο της ανάπτυξης λογισμικού. Σύμφωνα με πολλές έρευνες, η διόρθωση ενός σφάλματος σε αρχικό στάδιο της ανάπτυξης λογισμικού, κοστίζει πολύ λιγότερο από την διόρθωσή του σε αργότερο στάδιο.
Οι παραπάνω λόγοι οδήγησαν στην εφεύρεση της ανάπτυξης λογισμικού με γνώμονα τον έλεγχο μονάδων (Test Driven Development, TDD). Το TDD υποστηρίζει πως ένας προγραμματιστής θα πρέπει αρχικά να δημιουργήσει ένα σενάριο ελέγχου για κάθε λειτουργία ενός συστήματος και έπειτα να προχωρήσει στην υλοποίησή του [42].

Υπάρχουν αρκετά διαθέσιμα πλαίσια για τις περισσότερες γλώσσες προγραμματισμού. Όσο αφορά την Java, τα πιο δημοφιλή είναι το JUnit και TestNG. Στα πλαίσια αυτής της διπλωματικής εργασίας, θα περιγράψουμε το πλαίσιο JUnit, καθώς και θα το χρησιμοποιήσουμε στην εφαρμογή του περιβάλλοντος πλαισίου μας.

Το JUnit είναι ένα πλαίσιο δοκιμής που χρησιμοποιεί σχολιασμούς (annotations) για τον προσδιορισμό των μεθόδων που ορίζουν μια δοκιμή. Μια δοκιμή ελέγχου JUnit (JUnit Test) [43]είναι μια μέθοδος που περιέχεται σε μια κλάση (java class) η οποία χρησιμοποιείται μόνο για τη δοκιμή. Αυτό ονομάζεται Κλάση Ελέγχου. Για να γράψουμε μια δοκιμή ελέγχου χρησιμοποιώντας το πλαίσιο JUnit , πρέπει να σχολιάσουμε μια μέθοδο με την ένδειξη "@ org.junit.Test" . Σε αυτή τη μέθοδο μπορούμε να χρησιμοποιήσουμε μια μέθοδο "assert" για να ελέγξουμε το αναμενόμενο αποτέλεσμα της εκτέλεσης κώδικα σε σχέση με το πραγματικό αποτέλεσμα. Θα πρέπει να παρέχουμε ουσιαστικά μηνύματα στις παραπάνω μεθόδους έτσι ώστε να είναι ευκολότερο για τον προγραμματιστή να εντοπίσει το πρόβλημα. Αυτό διευκολύνει τη διόρθωση του θέματος ειδικά όταν αυτός που παρακολουθεί τον έλεγχο μονάδων ενός συστήματος, δεν είναι το ίδιο πρόσωπο με εκείνον που έγραψε τον κώδικα υπό έλεγχο ή τον κώδικα ελέγχου.

Ο κώδικας που ακολουθεί δείχνει μια JUnit δοκιμή ελέγχου. Αυτός ο έλεγχος προϋποθέτει ότι υπάρχει η κατηγορία AddClass και η οποία περέχει την addintegers(int, int) μέθοδο [43].

```java
package org.eclipse.e4.core.internal.tests;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class MyTests {

  @Test
  public void multiplicationOfZeroIntegersShouldReturnZero() {

    // MyClass is tested
    MyClass tester = new MyClass();

    // assert statements
    assertEquals("10 x 0 must be 0", 0, tester.multiply(10, 0));
    assertEquals("0 x 10 must be 0", 0, tester.multiply(0, 10));
    assertEquals("0 x 0 must be 0", 0, tester.multiply(0, 0));
  }

}
```

Σχήμα 2.18: Παράδειγμα JUnit δοκιμής ελέγχου

Κεφάλαιο 3: Αρχιτεκτονική και λειτουργία του συστήματος

3.1 Εισαγωγή

Ο σχεδιασμός ενός συστήματος λογισμικού αποτελεί το πιο σημαντικό κομμάτι κατά τη διάρκεια ανάπτυξης του. Στο στάδιο αυτό σχεδιάζεται πώς το σύστημα θα πληροί τις τεχνικές προδιαγραφές με αποτελεσματικό και αποδοτικό τρόπο. Υπάρχουν πολλές πτυχές που πρέπει να λαμβάνονται υπόψη κατά τον σχεδιασμό ενός συστήματος λογισμικού . Κάθε μια θα πρέπει να αντικατοπτρίζει τους στόχους που προσπαθεί να επιτύχει το λογισμικό.
Αυτές στις οποίες εστιάσαμε στα πλαίσια της διπλωματικής είναι οι παρακάτω[51]:

- Συμβατότητα:
  Το λογισμικό πρέπει να είναι σε θέση να λειτουργήσει με άλλα προϊόντα .

- Επεκτασιμότητα:
  Είναι σημαντικό να υπάρχει δυνατότητα προσθήκης νέων λειτουργιών στο λογισμικό χωρίς σημαντικές αλλαγές στη βασική αρχιτεκτονική.

- Δυνατότητα Συντήρησης:
  Αποτελεί ένα μέτρο του βαθμού ευκολίας διόρθωσης σφαλμάτων ή λειτουργικών τροποποιήσεων.

- Αξιοπιστία:
  Το λογισμικό πρέπει να είναι σε θέση να εκτελέσει μια απαιτούμενη λειτουργία υπό καθορισμένες συνθήκες για μια συγκεκριμένη χρονική περίοδο.

- Επαναχρησιμοποίηση:
  Μέρη ή το σύνολο του λογισμικού πρέπει να μπορούν να χρησιμοποιηθούν σε άλλα έργα με καθόλου ή σχεδόν καθόλου τροποποίηση.

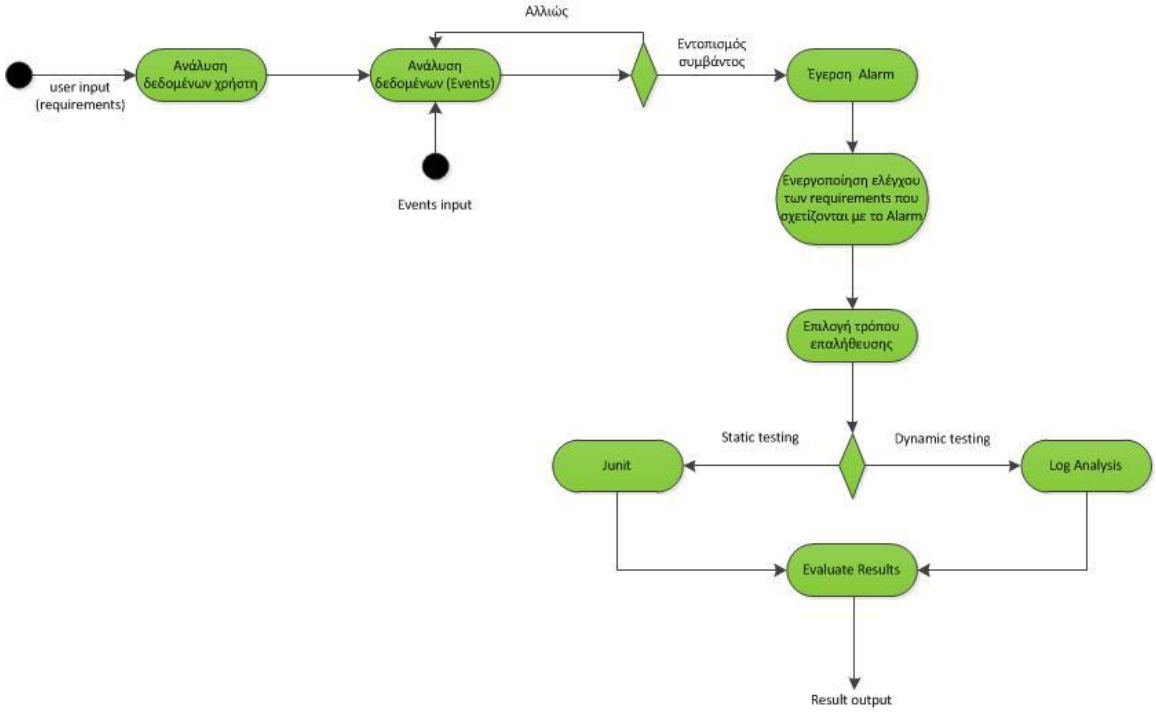
- Ανθεκτικότητα:
  Το λογισμικό πρέπει να είναι σε θέση να λειτουργήσει υπό πίεση ή σε απρόβλεπτες συνθήκες όπως η απρόβλεπτη ή άκυρη είσοδος .

Στο κεφάλαιο αυτό θα παρουσιάσουμε και θα αναλύσουμε την αρχιτεκτονική του περιβάλλοντος πλαισίου που πραγματεύεται η παρούσα διπλωματική εργασία . Θα αναπτύξουμε τις

αρχιτεκτονικές τεχνοτροπίες που χρησιμοποιήθηκαν και σχεδιαστικές αποφάσεις που πάρθηκαν. Επιπλέον θα περιγράψουμε όλα τα δομικά στοιχεία του περιβάλλοντος-πλαισίου και την αρχιτεκτονική διασύνδεσης και επικοινωνίας μεταξύ τους. Τέλος θα παρουσιάσουμε και θα αναλύσουμε τα ακολουθιακά διαγράμματα του συστήματος. Για το σκοπό αυτό θα χρησιμοποιηθούν μπλοκ και ψηφιδικά διαγράμματα της γλώσσας UML. Πριν όμως προχωρήσουμε στην περιγραφή των δομικών στοιχείων του συστήματος θα παρουσιάσουμε το σύνολο των απαιτήσεων που πρέπει να ικανοποιεί το περιβάλλον - πλαίσιο.

Βασικός στόχος του περιβάλλοντος πλαισίου που αναπτύσσεται στην παρούσα διπλωματική εργασία, είναι η εφαρμογή στατικού και δυναμικού ελέγχου σε ένα σύστημα λογισμικού σύμφωνα με το σύνολο των αρχικών προδιαγραφών (requirements), χρησιμοποιώντας μοντελοκεντρική αρχιτεκτονική. Κίνητρο για την εκκίνηση είτε της στατικής είτε της δυναμικής ανάλυσης είναι η υπόθεση ότι κάποια από της αρχικές προδιαγραφές τους ελεγχόμενου συστήματος (requirement) δεν ικανοποιήθηκε. Η υπόθεση αυτή συνεπάγεται την ενεργοποίηση ενός συναγερμού στο σύστημα (alarm) . Με την ενεργοποίηση ενός alarm , ξεκινά ο έλεγχος ισχύος της αρχικής προδιαγραφής που σχετίζεται με το συγκεκριμένο alarm. Για την ολοκλήρωση αυτού του ελέγχου πρέπει να βρεθεί το πλάνο υπολειτουργιών που πρέπει να ελεγχθούν έτσι ώστε να αποφανθεί η ισχύς ή όχι της αρχικής προδιαγραφής. Πρέπει λοιπόν να εκτελεστεί ανάλυση αρχικών αιτιών (root cause analysis) , η οποία επιτυγχάνεται με την βοήθεια δέντρων στόχων που μοντελοποιούν τις αρχικές μας προδιαγραφές. Η αρχική προδιαγραφή τοποθετείται στη ρίζα του δέντρου και οι υπολειτουργίες που απαιτούνται για την ικανοποίηση του , τοποθετούνται στα φύλλα του δέντρου. Η ανάλυση αρχικών αιτιών χρησιμοποιεί το περιβάλλον πλαίσιο SAT4J για την εξαγωγή των ελάχιστων δυνατών κόμβων του δέντρου (υπολειτουργίες) που πρέπει να ελεγχτούν, ώστε να μπορούμε να αποφανθούμε για την ισχύ ή όχι της αρχικής προδιαγραφής. Το περιβάλλον-πλάισιο SAT4J περιγράφεται αναλυτικά στο κεφάλαιο 5.

Οι βασικές λειτουργίες του βρόγχου ελέγχου του περιβάλλοντος-πλαισίου φαίνονται στο παρακάτω διάγραμμα (Σχήμα 3.1):



Σχήμα 3.1: Διάγραμμα Δραστηριότητας Βασικού Βρόχου Εκτέλεσης

Τα δομικά στοιχεία (component) λοιπόν που πρέπει να υλοποιηθούν ώστε να ικανοποιείται το σύνολο των απαιτήσεων που περιγράφθηκε παραπάνω είναι:

- Ένα component που θα είναι υπεύθυνο για την πυροδότηση των alarms και την ενεργοποίηση και απενεργοποίηση των συσχετιζόμενων υποθέσεων. Όπως προαναφέραμε στην περίπτωση που το υπό έλεγχο σύστημα εμφανίσει κάποια ανωμαλία στη λειτουργία του, τότε θα προσθέσει ένα alarm στα δεδομένα παρακολούθησης του. Πρέπει λοιπόν να υπάρχει ένα δομικό στοιχείο στο παραβάλλον-πλαίσιο που θα αναπτύξουμε, το οποία να παρακολουθεί το υπό έλεγχο σύστημα και στην περίπτωση εμφάνισης λάθους να εκκινεί τη διαδικασία ελέγχου πυροδοτώντας και αυτό με τη σειρά του ένα alarm .
- Ένα component υπεύθυνο για την κατάστρωση ενός πλάνου για την απόδειξη μιας υπόθεσης στηριζόμενο στο μοντέλο δένδρων στόχων και στην ανάλυση αρχικών αιτιών με τη βοήθεια του SAT4J.
- Ένα component υπεύθυνο για την επαλήθευση των υποθέσεων και την εκτέλεση των αντίστοιχων τεχνικών ελέγχου.
- Ένα component που θα αναπαριστά τα δέντρα στόχων.
- Ένα component που θα αναπαριστά το σύνολο των alarms, και
- Ένα component για την αναπαράσταση των στρατηγικών ελέγχου των κόμβων.

Επιπλέον αναλύοντας περαιτέρω τις απαιτήσεις που πρέπει να πληροί το υπό σχεδίαση σύστημα αναγνωρίζουμε τις εξής:

- Την δυνατότητα ορισμού νέων υποθέσεων και alarm.
- Την δυνατότητα ορισμού και διαφορετικών στρατηγικών για την επαλήθευση και την παρακολούθηση των υποθέσεων.

Για την ικανοποίηση αυτών των δύο απαιτήσεων, όπως έχουμε προαναφέρει μοντελοποιούμε τις υποθέσεις - προδιαγραφές του συστήματος σε δέντρα στόχων. Μια υπόθεση αντιστοιχεί σε ένα δέντρο στόχων και κάθε υπόθεση συσχετίζεται με ένα ή περισσότερα alarms . Η ρίζα του δέντρου στόχων , δηλαδή η βασική υπόθεση προς επαλήθευση διαχωρίζεται σε επιμέρους υπο-λειτουργίες που αποτελούν τους κόμβους του δέντρου. Οι κόμβοι του δέντρου με τη σειρά τους μπορούν να διασπαστούν σε απλούστερες λειτουργίες, φτάνοντας έτσι στα φύλλα του δέντρου. Με τη χρήση του περιβάλλοντος -πλαισίου SAT4J προσδιορίζονται όλοι οι πιθανοί συνδυασμοί κόμβων που πρέπει να επαληθευτούν προκειμένου να μπορέσουμε να αποφανθούμε για την ισχύ της ρίζας του δέντρου. Πιο συγκεκριμένα επιλέγεται ένας συνδυασμός κόμβων του δέντρου από το πλήθος των αποτελεσμάτων του SAT4J και αποδεικνύεται κάθε κόμβος του συνδυασμού αυτού. Για την απόδειξη των κόμβων εφαρμόζεται τόσο στατική ανάλυση όσο και δυναμική ανάλυση όπως περιγράφεται αναλυτικά στο κεφάλαιο 6. Τα alarms αναπαρίστανται ως ξεχωριστές οντότητες που συσχετίζονται με τις υποθέσεις και ενεργοποιούνται όταν το υπό παρακολούθηση σύστημα εμφανίσει κάποια δυσλειτουργία. Έτσι χρειαζόμαστε ένα component για την αναπαράσταση των δέντρων στόχων, ένα component για την αναπαράσταση των alarms, ένα component για την αναπαράσταση των στρατηγικών επαλήθευσης των κόμβων και τέλος ένα component που θα μας δίνει τη δυνατότητα διαχείρισης των παραπάνω .

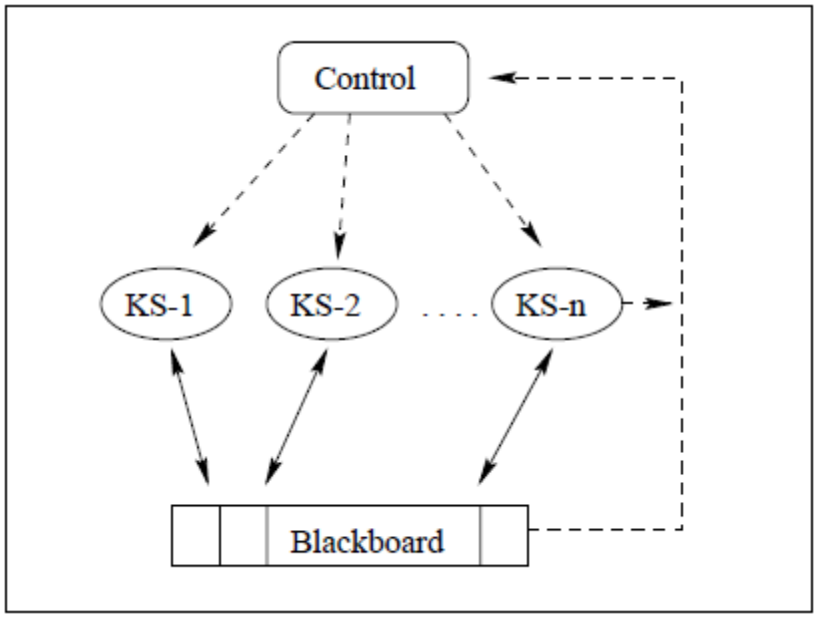
3.2 Επισκόπηση της αρχιτεκτονικής

Στην ενότητα αυτή παρουσιάζεται το σύνολο των δομικών στοιχειών που απαρτίζουν το περιβάλλον πλαίσιο για την εφαρμογή στατικού και δυναμικού έλεγχου. Επιπλέον παρουσιάζεται η αρχιτεκτονική σχεδίασης και ο τρόπος αλληλεπίδρασης των διαφόρων δομικών στοιχείων στο σύνολο του συστήματος.

Το περιβάλλον πλαίσιο που αναπτύχθηκε στα πλαίσια αυτής της διπλωματικής βασίζεται στα αρχιτεκτονικά μοτίβα μαυροπίνακα (blackboard) και Publisher/Subsriber τα οποία παρουσιάζονται περιληπτικά παρακάτω.

Το αρχιτεκτονικό πρότυπο μαυροπίνακα (BlackBoard) παρέχει ένα υπολογιστικό πλαίσιο για το σχεδιασμό και την υλοποίηση των συστημάτων που πρέπει να ενσωματώσουν μεγάλο αριθμό μονάδων και να υλοποιήσουν σύνθετες, μη ντετερμινιστικές στρατηγικές ελέγχου. Αυτό το πρότυπο, όπως περιγράφεται στο [52], ορίζει διάφορα components, από τα οποία το component

ελέγχου είναι το πιο σημαντικό. Αρκετά μοντέλα έχουν προταθεί για την υποστήριξη αυτού του component ελέγχου που είναι το κλειδί για όλες τις ιδιότητες του συστήματος [53]. Το πρότυπο μαυροπίνακα ορίζει ανεξάρτητες μονάδες, τις οποίες ονομάζει πηγές γνώσης (knowledge sources), με στόχο την επίλυση κάποιου προβλήματος. Οι μονάδες αυτές είναι ικανές να επιλύσουν κάποιο υπο-πρόβλημα, αλλά όχι το ενιαίο αρχικό πρόβλημα που καλείται να επιλύσει ο Blackboard. Τα components του μαυροπίνακα , διαβάζουν και γράφουν σχετικά δεδομένα στον μαυροπίνακα, τα οποία λειτουργούν βοηθητικά στην εύρεση της λύσης του αρχικού προβλήματος. Ο μαυροπίνακας μπορεί να θεωρηθεί σαν μία δομημένη μνήμη, πάνω στην οποία συντίθεται σταδιακά η λύση του ενιαίου προβλήματος, με τη συλλογή των δεδομένων από τις πηγές γνώσης. Όταν μια πηγή γνώσης κάνει μια σημαντική αλλαγή στον μαυροπίνακα, τότε λέμε πως δημιουργεί ένα συμβάν (event) . Η γενική οργάνωση ενός συστήματος μαυροπίνακα απεικονίζεται στο επόμενο σχήμα. Οι διακεκομμένες γραμμές παριστάνουν τη ροή ελέγχου, ενώ οι συνεχείς γραμμές δείχνουν τη δυνατότητα καταγραφής δεδομένων στον μαυροπίνακα .

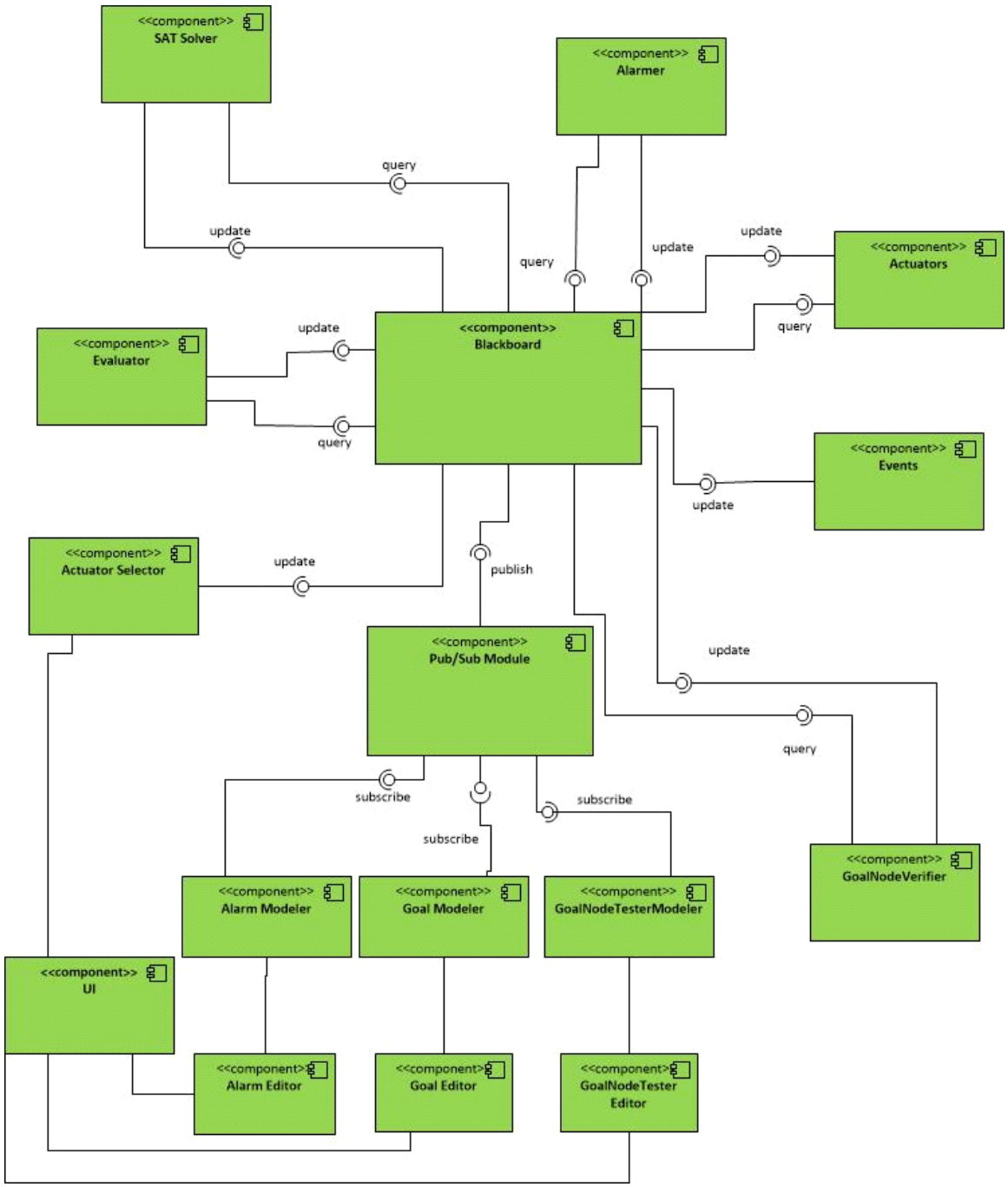


Σχήμα 3.2: Οργάνωση ενός συστήματος μαυροπίνακα

Σε κάθε χρονική στιγμή, πολλές ανταγωνιστικές πηγές γνώσης μπορεί να είναι ενεργοποιημένες ταυτόχρονα . Είναι ο σκοπός του μαυροπίνακα, ως στοιχείο ελέγχου του συστήματος, να επιλέξει το καλύτερο για άμεση εκτέλεση. Οι ενεργοποιήσεις των πηγών γνώσης δεν είναι προγραμματισμένες εκ των προτέρων, αλλά προσδιορίζονται σε κάθε κύκλο ελέγχου ανάλογα με την τρέχουσα κατάσταση. Στο βασικό μοντέλο μαυροπίνακα, οι στρατηγικές ελέγχου είναι καθορισμένες.

Το αρχιτεκτονικό μοτίβο Publisher/Subsriber εντάσσεται στην κατηγορία των μοτίβων συμπεριφοράς (Behavioral Patterns) [54]. Ορίζει μια εξάρτηση ένα-προς-πολλά μεταξύ των αντικειμένων, έτσι ώστε όταν η κατάσταση ενός αντικειμένου αλλάζει όλα τα άλλα αντικείμενα που συμμετέχουν στην σχέση εξάρτησης να μπορούν να ενημερωθούν (αλλάξουν) αυτόματα. Το αρχιτεκτονικό μοτίβο Publisher/Subsriber είναι επίσης γνωστό ως Observer. Στην αρχιτεκτονική η συγκεκριμένη αρχιτεκτονική αποτελεί ένα πρότυπο ανταλλαγής μηνυμάτων, όπου οι αποστολείς των μηνυμάτων που ονομάζονται Publishers δεν επικοινωνούν απευθείας με συγκεκριμένους δέκτες. Αντίθετα, τα μηνύματα αυτά δημοσιεύονται σε κλάσεις χωρίς να υπάρχει η γνώση για το αν υπάρχει κάποιος παραλήπτης η όχι. Όμοια οι παραλήπτες εκφράζουν το ενδιαφέρων τους για μια οι περισσότερες κλάσεις λαμβάνοντας μηνύματα μόνο από αυτές, χωρίς να γνωρίζουν για το πόσοι και ποιοι είναι οι αποστολείς. Αυτό το πρότυπο παρέχει μεγαλύτερη επεκτασιμότητα του δικτύου και μία πιο δυναμική τοπολογία . Οι τρεις παραλλαγές του μοτίβου publish / subscribe που μπορούν να χρησιμοποιηθούν για τη δημιουργία ενός μηχανισμού που στέλνει μηνύματα σε όλους τους ενδιαφερόμενους συνδρομητές είναι [55]: publish / subscribe βασισμένη σε λίστα (List-Based Publish/Subscribe) , publish / subscribe βασισμένη στην μετάδοση (Broadcast-Based Publish/Subscribe) και publish / subscribe βασισμένο στο περιεχόμενο (*Content-Based Publish/Subscribe*).

Ακολουθεί το ψηφιδικό διάγραμμα της αρχιτεκτονικής του συστήματος στο οποίο φαίνεται η υλοποίηση των αρχιτεκτονικών μοτίβων publisher/subscriber και blackboard.



Σχήμα 3.3: Ψηφιδικό διάγραμμα της αρχιτεκτονικής του περιβάλλοντος πλαισίου

Όπως φαίνεται από το παραπάνω διάγραμμα η αρχιτεκτονική αποτελείται από τα εξής βασικά components:

- Blackboard :
  Αποτελεί το βασικό component του περιβάλλοντος πλαισίου και λειτουργεί ως μία δομημένη μνήμη, η οποία συλλέγει όλη την πληροφορία και τα γεγονότα (events) που δημιουργούνται από τις υπόλοιπες πηγές γνώσης, δηλαδή τα υπόλοιπα components του συστήματός μας. Κάθε χρονική στιγμή, ο Blackboard είναι λοιπόν το κεντρικό σημείο που γνωρίζει την κατάσταση του συστήματος , τα γεγονότα και τα ενδιάμεσα αποτελέσματα που παράγονται από όλα τα components του συστήματος , με σκοπό τη σταδιακή επίλυση του προβλήματος που καλείται να λύσει το παρόν περιβάλλον πλαίσιο.

- Pub/Sub Module :
  Όπως αναφέρθηκε και παραπάνω, ο Pub/Sub προσφέρει στα διάφορα components του συστήματος την δυνατότητα εγγραφής για τύπους γεγονότων που έχουν ενδιαφέρον για αυτούς. Επίσης διατηρεί αρχεία για τα εγγεγραμμένα ανά γεγονός components και έχει την δυνατότητα διαγραφής τους. Όταν τα components του συστήματος παράγουν κάποιο event, τότε ενημερώνουν το Pub/Sub και αυτό με την σειρά του ενημερώνει όλα τα components που έχουν εκδηλώσει ενδιαφέρον για τον συγκεκριμένο τύπο γεγονότων. Με τον τρόπο αυτό διευκολύνει την επικοινωνία και ενημέρωση των διαφόρων components προσφέροντάς τους την δυνατότητα αποστολής ενημερώσεων χωρίς να έχουν γνώση των παραληπτών.

- Goal Modeler :
  Αυτό το component είναι υπεύθυνο για την αναπαράσταση του μεταμοντέλου στο οποίο βασίζεται η μοντελοποίηση των απαιτήσεων του συστήματος σε δέντρα στόχους.

- Alarm Modeler :
  Αυτό το component είναι υπεύθυνο για την αναπαράσταση του μεταμοντέλου στο οποίο βασίζεται η μοντελοποίηση των συναγερμών (alarms) του συστήματος.

- GoalNodeTester Modeler:
  Αυτό το component είναι υπεύθυνο για την αναπαράσταση του μεταμοντέλου στο οποίο βασίζεται η μοντελοποίηση του συνόλου στρατηγικών επαλήθευσης των κόμβων του δέντρου στόχων.

- Goal Editor:
  Αυτό το component χρησιμοποιείται για τον ορισμό νέων δέντρων στόχων και την διαγραφή ή την τροποποίηση των ήδη υπαρχόντων. Κάθε νέο μοντέλο δέντρων στόχων πρέπει να είναι σε συμφωνία με το μεταμοντέλο που τα προδιαγράφει. Ο Goal Editor παρέχει λειτουργία ελέγχου του δέντρου στόχων να υπακούει σε αυτούς τους κανόνες, καθώς δεν επιτρέπει την δημιουργία κόμβου που δεν τους ακολουθεί.

- Alarm Editor:
  Ο component Alarm Editor χρησιμοποιείται για τον ορισμό νέων συναγερμών (alarms) και την επεξεργασίας των ήδη υπαρχόντων. Κάθε νέο alarm πρέπει να είναι σε συμφωνία με το μεταμοντέλο που τα προδιαγράφει. Ο Alarm Editor παρέχει λειτουργία ελέγχου

ενός νέου alarm να υπακούει σε αυτούς τους κανόνες, καθώς δεν επιτρέπει την δημιουργία alarm που δεν τους ακολουθεί.

- GoalNodeTester Editor:
  Αυτό το component χρησιμοποιείται για τον ορισμό νέων στρατηγικών ελέγχου κόμβων (GoalNodeTesters) και την διαγραφή ή την τροποποίηση των ήδη υπαρχόντων. Κάθε νέο GoalNodeTester πρέπει να είναι σε συμφωνία με το μεταμοντέλο που τα προδιαγράφει. Ο GoalNodeTester Editor παρέχει λειτουργία ελέγχου μίας νέας στρατηγικής ελέγχου να υπακούει σε αυτούς τους κανόνες, καθώς δεν επιτρέπει την δημιουργία μίας στρατηγικής που δεν τους ακολουθεί.

- Report:
  Αυτό το component χρησιμοποιείται για την ενημέρωση του χρήστη σχετικά με κάθε διεργασία που εκτελείται. Είναι υπεύθυνο για την δήλωση της έναρξης της διαδικασίας ελέγχου ύπαρξης κάπου alarm, την εμφάνιση του alarm που εντοπίστηκε, την εμφάνιση της απαίτησης με την οποία συνδέεται και θα πρέπει να ελεγχθεί, καθώς και ολόκληρη την διαδικασία ελέγχου. Δηλαδή, εμφανίζει στον χρήστη ποιες υπολειτουργίες θα ελεγχθούν, με ποιό τρόπο θα ελεγχθούν και ποιο είναι το αποτέλεσμα ελέγχου. Έτσι ο χρήστης έχει μία πλήρη εικόνα της διαδικασίας ελέγχου του υπό έλεγχο συστήματος, από την έναρξη έως και την ολοκλήρωσή της, καθώς και μία αναφορά αποτελεσμάτων του ελέγχουν, δίνοντας του έτσι την δυνατότητα να εντοπίσει ποιες από τις αρχικές αιτίες είναι υπεύθυνες για την αποτυχία της αντίστοιχης απαίτησης του συστήματος.

- UI :
  Αυτό το component χρησιμοποιείται για την αλληλεπίδραση του χρήστη με το μοντέλο των απαιτήσεων του συστήματος που θέλει να ελέγξει. Στα πλαίσια του παρόντος περιβάλλοντος πλαισίου , ως UI χρησιμοποιήθηκε το πλαίσιο Eclipse Modelling Framework (EMF) , το οποίο περιγράφεται αναλυτικά στο Κεφάλαιο 2. Το EMF δίνει τη δυνατότητα στον χρήστη να μοντελοποιήσει το σύστημα του, να το τροποποιήσει και να το ελέγξει. Επίσης δίνει τη δυνατότητα παραγωγής μεθόδων που διευκολύνουν τη διαχείριση των μοντέλων από το χρήστη.

- GoalSelector :
  Ο component GoalSelector είναι υπεύθυνος για την αντιστοίχιση ενός ενεργοποιημένου alarm με την απαίτηση που πρέπει να επαληθευτεί, και συνεπώς το αντίστοιχο δέντρο στόχων. Ενημερώνεται από τον Blackboard για την ενεργοποίηση κάποιου alarm, και επιστρέφει σε αυτόν τα αντίστοιχα δέντρα στόχων που πρέπει να επαληθευτούν. Η πληροφορία αυτή θα σταλθεί αργότερα στον SAT Solver για την ανάλυση αρχικών αιτιών.

- Alarmer :
  Ο Alarmer είναι υπεύθυνος για την ενεργοποίηση και απενεργοποίηση των alarms. Αναλύει τα δεδομένα που συλλέγονται από το υπο έλεγχο σύστημα και αναζητά την ύπαρξη κάποιας ένδειξης αποτυχίας κάποιας απαιτήσεως του συστήματος. Όταν βρεθεί κάποια τέτοια ένδειξη, τότε πυροδοτεί το κατάλληλο alarm και ενημερώνει τον Blackboard σχετικά με αυτό. Γενικά, είναι ο component που είναι υπεύθυνος για την έναρξη της διαδικασία ελέγχου κάποιας απαίτησης ενός συστήματος υπό παρακολούθηση.

- Events :
  Ο component αυτός παρέχει τα δεδομένα που απαιτούνται για τον έλεγχο του συστήματος. Μπορεί να είναι αρχεία δεδομένων παρακολούθησης (log files) , βάσεις δεδομένων ή ροές δεδομένων (data streams) από κάποιον κόμβο δικτύου .

- SAT Solver :
  Ο συγκεκριμένος component είναι υπεύθυνος για τον προσδιορισμό όλων των πιθανών συνδυασμών που πρέπει να ελεγχθούν για την εύρεση της αιτίας αποτυχίας μίας απαιτήσεως του υπό έλεγχο συστήματος. Ο SAT Solver ενημερώνεται από τον Blackboard για το δέντρο στόχων που πρέπει να επαληθευτεί , και αναλαμβάνει την εκτέλεση της ανάλυσης αρχικών αιτιών (Root Cause Analysis) . Επιστρέφει στον Blackboard όλους τους πιθανούς συνδυασμούς αιτιών στους οποίους μπορεί να οφείλεται η αποτυχία της απαιτήσεως του συστήματος.

- GoalNodeVerifier :
  Το δομικό στοιχείο GoalNodeVerifier είναι υπεύθυνο για την εκτέλεση των δοκιμών ελέγχου για κάθε κόμβο του δέντρου στόχων , με στόχο την εύρεση της αληθοτιμής του. Ενημερώνεται από τον Pub/Sub για τον κόμβο που κάθε φορά πρέπει να επαληθευτεί, και αναζητά τις στρατηγικές που είναι συνδεδεμένες με αυτόν τον κόμβο. Οι στρατηγικές που χρησιμοποιούμε στο περιβάλλον πλαίσιο μας έχουν να κάνουν με ανάλυση δεδομένων παρακολούθησης της εκτέλεσης ενός λογισμικού, είτε με δοκιμές ελέγχου μονάδων του λογισμικού υπό έλεγχο. Μετά την εκτέλεση όλων των στρατηγικών που είναι συνδεδεμένων με έναν κόμβο, ενημερώνει τον Blackboard για την αληθοτιμή αυτού του κόμβου, που αντιστοιχεί στο αποτέλεσμα εκτέλεσης της στρατηγικής. Στο παράδειγμά μας, το αποτέλεσμα αυτό εκφράζεται ως "αληθές" στην περίπτωση που η στρατηγική επιτυγχάνει, και ως "λάθος" στην περίπτωση που η στρατηγική αποτυγχάνει. Αφού ενημερώσει τον Blackboard, συνεχίζει με την επαλήθευση του επόμενου κόμβου.

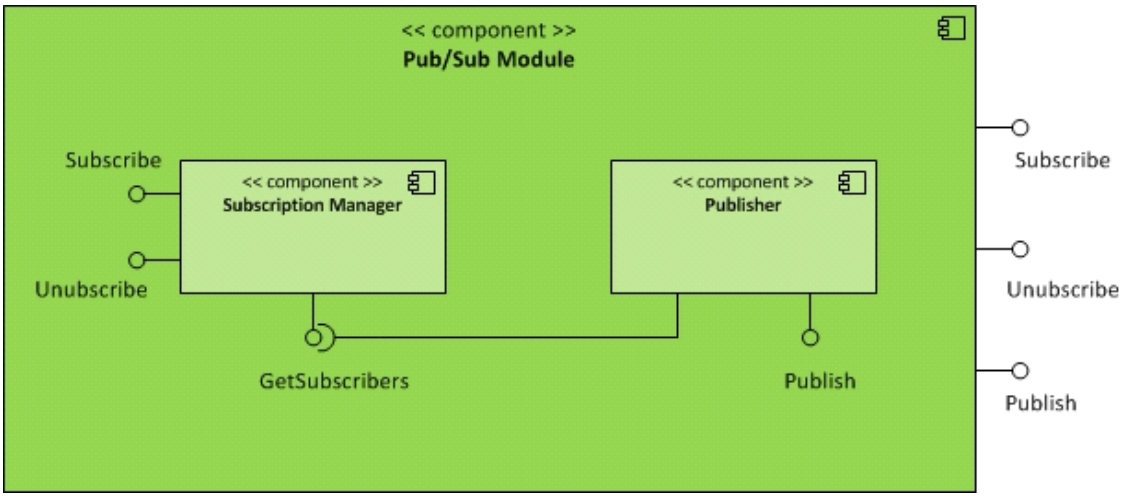
- Evaluator :
  Το δομικό στοιχείο Evaluator είναι υπεύθυνο για την απόφαση της ισχύος της αρχικής απαίτησης, δηλαδή του κόμβου-ρίζα του δέντρου στόχου υπό έλεγχο. Ενημερώνεται από τον Pub/Sub για την απαίτηση που πρέπει να επαληθευτεί, για τους συνδυασμούς των κόμβων που την επαληθεύουν, καθώς και τις αληθοτιμές των κόμβων που είναι ήδη διαθέσιμες. Για την εξαγωγή του τελικού συμπεράσματος, διασταυρώνει εάν οι διαθέσιμες αληθοτιμές των αποδεδειγμένων κόμβων οδηγούν στην επαλήθευση της ισχύος της αρχικής απαίτησης, δηλαδή εάν ικανοποιείται ο συνδυασμός κόμβων που έχει προκύψει από τον SAT Solver.

3.3 Αναλυτική Περιγραφή των Δομικών Στοιχείων

Σε αυτή τη παράγραφο, θα εξετάσουμε πιο αναλυτικά τον κάθε ένα component που αναφέρθηκε στην προηγούμενη παράγραφο. Θα χρησιμοποιήσουμε UML διαγράμματα για να αναπαραστήσουμε τους components και την εσωτερική δομή τους. Τέλος θα αναφερθούμε στις διεπαφές τους και τις αλληλεπιδράσεις τους με άλλους components.

3.3.1 Δομικό στοιχείο Pub/Sub

Ο Pub/Sub αναλαμβάνει, όπως αναφέρθηκε και παραπάνω, την ενημέρωση όλων των components του συστήματος για γεγονότα που τους ενδιαφέρουν. Αυτό το επιτυγχάνει προσφέροντας στα διάφορα components του συστήματος την δυνατότητα εγγραφής για τύπους γεγονότων που έχουν ενδιαφέρον για αυτούς, διατηρώντας αρχεία για τα εγγεγραμμένα ανά γεγονός components . Όταν τα components του συστήματος παράγουν κάποιο event, τότε ενημερώνουν το Pub/Sub και αυτό με την σειρά του ενημερώνει όλα τα components που έχουν εκδηλώσει ενδιαφέρον για τον συγκεκριμένο τύπο γεγονότων. Με τον τρόπο αυτό διευκολύνει την επικοινωνία και ενημέρωση των διαφόρων components προσφέροντάς τους την δυνατότητα αποστολής ενημερώσεων χωρίς να έχουν γνώση των παραληπτών.
Τα παραπάνω επιτυγχάνονται με τη βοήθεια των διεπαφών και των υπολειτουργιών που φαίνονται στο επόμενο σχήμα, και αναλύονται παρακάτω.



Σχήμα 3.4: Ψηφιδικό διάγραμμα του Pub/Sub

Η διεπαφή του Pub/Sub component ορίζει τις εξής συναρτήσεις:

• subscribe: Η συνάρτηση αυτή είναι υπεύθυνη για την εγγραφή των components στον Pub/Sub , δηλώνοντας έτσι πως ενδιαφέρονται να λαμβάνουν ενημερώσεις για κάποια συγκεκριμένα γεγονότα. Επομένως καταγράφει στα αρχεία που διατηρεί ο Pub/Sub την προσθήκη ενός component και ενός ή περισσοτέρων γεγονότων που τον ενδιαφέρουν να λαμβάνει σχετικές ενημερώσεις.

• unsubscribe: Η συνάρτηση αυτή είναι υπεύθυνη για την διαγραφή των components από τον Pub/Sub , δηλώνοντας έτσι πως δεν ενδιαφέρονται πλέον να λαμβάνουν ενημερώσεις για κάποια συγκεκριμένα γεγονότα. Επομένως αφαιρεί από τα αρχεία που διατηρεί ο Pub/Sub τους component που δεν ενδιαφέρονται πλέον να λαμβάνουν σχετικές ενημερώσεις, καθώς και τα αντίστοιχα γεγονότα.

• publish: Η συνάρτηση αυτή είναι υπεύθυνη για την ενημέρωση των components που είναι εγγεγραμμένα στον Pub/Sub , σχετικά με την δήλωση ενός γεγονότος για το οποίο έχουν εκδηλώσει ενδιαφέρον.

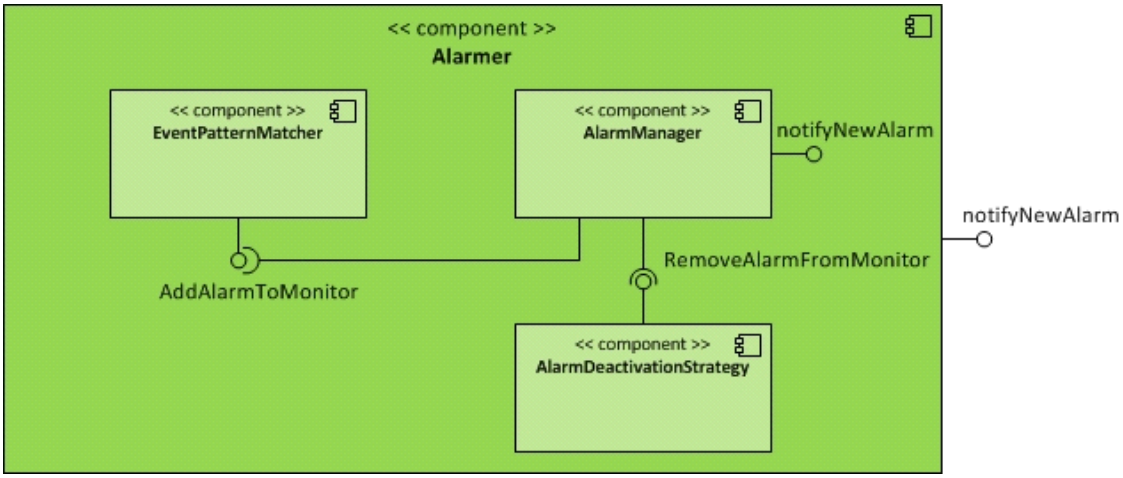
Για να υλοποιήσει τις παραπάνω λειτουργίες περιέχει τις εξής υπολειτουργίες :

• Subscription Manager: Μέσω αυτής της υπολειτουργίας γίνεται η εγγραφή και η διαγραφή των components για την εκδήλωση ή μη ενδιαφέροντος για κάποιο συγκεκριμένο γεγονός. Διατηρεί τα αρχεία του Pub/Sub component, που περιέχουν την πληροφορία εγγραφής και διαγραφής. Είναι επομένως υπεύθυνο για την οργάνωση των συνδρομών (subscriptions) με στόχο την αποδοτική εύρεση των ενδιαφερόμενων παραληπτών των γεγονότων του συστήματος. Η διεπαφή του περιλαμβάνει τις μεθόδους subscribe, unsubscribe και getSubscribers.

• Publisher: Μέσω αυτής της υπολειτουγίας γίνεται η ενημέρωση των ενδιαφερόμενων components για γεγονότα για τα οποία έχουν δηλώσει ενδιαφέρον μέσω του Subscription Manager. Η διεπαφή της περιλαμβάνει τις μεθόδους Publish και getSubscribers . Λαμβάνει έναν νέο γεγονός μέσω της διεπαφής publish, επικοινωνεί μέσω της μεθόδου "getSubscribers" με τον Subscription Manager, ώστε να εντοπίσει τους ενδιαφερόμενους παραλήπτες, και αποστέλλει τα νέα γεγονότα στους παραλήπτες αυτούς.

3.3.2 Δομικό στοιχείο Alarmer

Ο Alarmer είναι υπεύθυνος για την ενεργοποίηση και απενεργοποίηση των alarms. Αναλύει τα δεδομένα που συλλέγονται από το υπό έλεγχο σύστημα και αναζητά την ύπαρξη κάποιας ένδειξης αποτυχίας κάποιας απαιτήσεως του συστήματος. Όταν βρεθεί κάποια τέτοια ένδειξη, τότε πυροδοτεί το κατάλληλο alarm και ενημερώνει τον Blackboard σχετικά με αυτό ώστε να εκκινήσει την διαδικασία ελέγχου κάποιας απαίτησης ενός συστήματος υπό παρακολούθηση. Τα παραπάνω επιτυγχάνονται με τη βοήθεια των διεπαφών και των υπολειτουργιών που φαίνονται στο επόμενο σχήμα, και αναλύονται παρακάτω.



Σχήμα 3.5: Ψηφιδικό διάγραμμα του Alarmer

Η διεπαφή του Alarmer component ορίζει την εξής συνάρτηση:

- notifyNewAlarm :
  Ο Alarmer ενημερώνεται μέσω του Pub/Sub Module για τα alarms που πρέπει να παρακολουθεί μέσω της διεπαφής notifyNewAlarm.

Για να υλοποιήσει τις παραπάνω λειτουργίες του, ο Alarmer περιέχει τις εξής υπολειτουργίες :

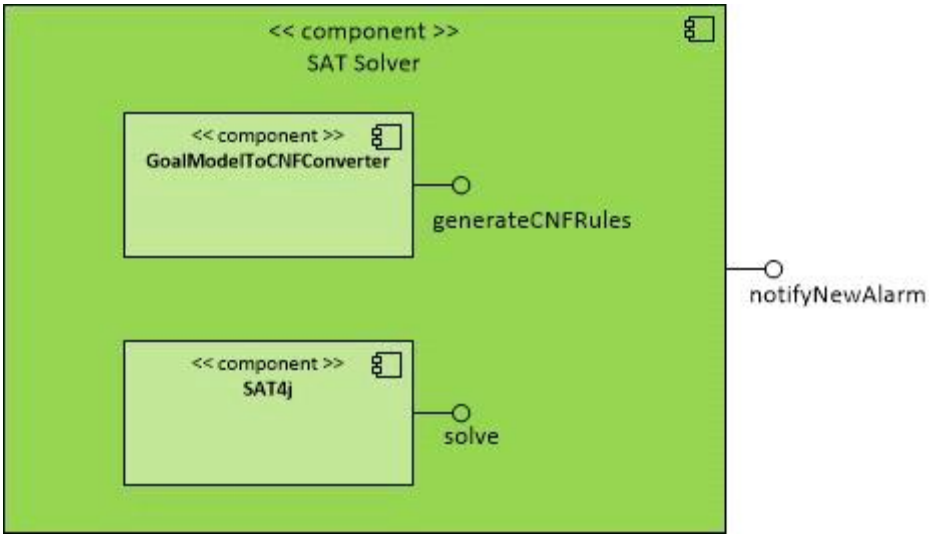
- EventPatternMatcher:
  Ο EventPatternMatcher Αναλύει τα δεδομένα που συλλέγονται από το υπό έλεγχο σύστημα και αναζητά την ύπαρξη κάποιας ένδειξης αποτυχίας κάποιας απαιτήσεως του συστήματος, δηλαδή την ύπαρξη ενός alarm ικανού να εκκινήσει τη διαδικασία έλεγχου και ανάλυσης αρχικών αιτιών.

- AlarmManager:
  Ο AlarmManager διαχειρίζεται την προσθήκη ή την αφαίρεση ενός alarm προς παρακολούθηση, καθώς και την ενεργοποίηση ή απενεργοποίησή τους. Οι ενέργειες αυτές γίνονται με τις αντίστοιχες διεπαφές του Alarm Manager : addAlarmToMonitor, removeAlarm. Η διεπαφή addAlarmToMonitor χρησιμοποιεί τον EventPatternMatcher ενώ η διεπαφή removeAlarm κάνει χρήση του AlarmDeactivationStrategy.

- AlarmDeactivationStrategy: Το component αυτό είναι υπεύθυνο για την απενεργοποίηση κάποιου alarm.

3.3.3 Δομικό στοιχείο SAT Solver

Το δομικό στοιχείο SAT Solver , είναι υπεύθυνο για την παραγωγή όλων των πιθανών συνδυασμών των αρχικών αιτιών (Root Cause Analysis) . Αυτό επιτυγχάνεται μέσω της χρήσης ενός πλαισίου SAT Solver. Στην περίπτωσή μας χρησιμοποιήσαμε το πλαίσιο SAT4j, το οποίο να περιγράψουμε αναλυτικά στο κεφάλαιο 5. Στην χρήση του SAT4j έχουμε ένα περιορισμό, το γεγονός ότι σαν είσοδό του δέχεται εκφράσεις CNF. Ένα πλεονέκτημα των δέντρων στόχων, που προκύπτει από την σχέση τους με την άλγεβρα Boole, είναι η εύκολη μετατροπή τους σε εκφράσεις CNF. Στην ουσία, η ικανοποίηση μίας έκφρασης CNF θα αντιστοιχεί στην ικανοποίηση ενός συνδυασμού κόμβων- αιτιών, άρα και στην ικανοποίηση της αρχικής απαίτησης που θέλουμε να επαληθεύσουμε. Επομένως, ο component SAT Solver , έχει ως αρμοδιότητες την παραγωγή των εκφράσεων CNF, την χρήση του SAT4j και την παροχή των πιθανών συνδυασμών κόμβων που πρέπει να επαληθεύσει το περιβάλλον πλαίσιο που αναπτύσσουμε.
Τα παραπάνω επιτυγχάνονται με τη βοήθεια των διεπαφών και των υπολειτουργιών που φαίνονται στο επόμενο σχήμα, και αναλύονται παρακάτω.



Σχήμα 3.6: Ψηφιδικό διάγραμμα του SAT Solver

Η διεπαφή του SAT Solver component ορίζει την εξής συνάρτηση:

- notifyNewAlarm :
  Ο SAT Solver ενημερώνεται μέσω του Pub/Sub Module για την απαίτηση που πρέπει να επαληθευτεί, και συνεπώς για το δέντρο στόχων που την αναπαριστά, μέσω της διεπαφής notifyNewAlarm.

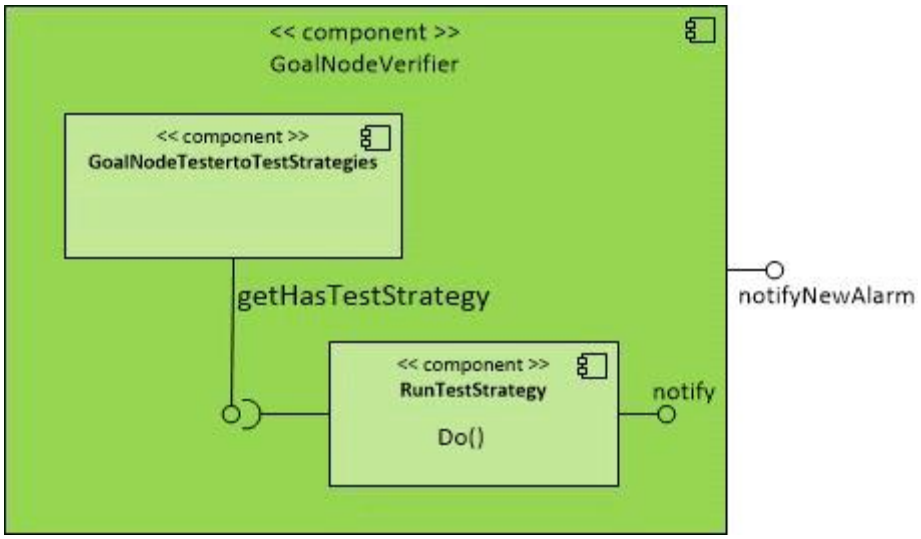
Για να υλοποιήσει τις παραπάνω λειτουργίες του, ο SAT Solver περιέχει τις εξής υπολειτουργίες :

- GoalModelToCNFConverter:
  Παράγει μια έκφραση CNF βάσει του δέντρου στόχων και της ρίζας του δέντρου στόχων που θέλουμε να επαληθεύσουμε.

- SatSolver:
  Υπολογίζει όλους τους δυνατούς τρόπους επαλήθευσης της αρχικής απαίτησης .

3.3.4 Δομικό στοιχείο GoalNodeVerifier

Το δομικό στοιχείο GoalNodeVerifier είναι υπεύθυνο για την εκτέλεση των δοκιμών ελέγχου για κάθε κόμβο του δέντρου στόχων , με στόχο την εύρεση της αληθοτιμής του. Ενημερώνεται μέσω μίας διεπαφής του από τον Pub/Sub για τον κόμβο που κάθε φορά πρέπει να επαληθευτεί, και αναζητά τις στρατηγικές που είναι συνδεδεμένες με αυτόν τον κόμβο. Οι στρατηγικές που χρησιμοποιούμε στο περιβάλλον πλαίσιο μας έχουν να κάνουν με ανάλυση δεδομένων παρακολούθησης της εκτέλεσης ενός λογισμικού, είτε με δοκιμές ελέγχου μονάδων του λογισμικού υπό έλεγχο. Μετά την εκτέλεση όλων των στρατηγικών που είναι συνδεδεμένων με έναν κόμβο, ενημερώνει τον Blackboard για την αληθοτιμή αυτού του κόμβου, που αντιστοιχεί στο αποτέλεσμα εκτέλεσης της στρατηγικής. Στο παράδειγμά μας, το αποτέλεσμα αυτό εκφράζεται ως "αληθές" στην περίπτωση που η στρατηγική επιτυγχάνει, και ως "λάθος" στην περίπτωση που η στρατηγική αποτυγχάνει. Αφού ενημερώσει τον Blackboard, συνεχίζει με την επαλήθευση του επόμενου κόμβου.
Τα παραπάνω επιτυγχάνονται με τη βοήθεια των διεπαφών και των υπολειτουργιών που φαίνονται στο επόμενο σχήμα, και αναλύονται παρακάτω.



Σχήμα 3.7: Ψηφιδικό διάγραμμα του GoalNodeVerifier

Η διεπαφή του GoalNodeVerifier component ορίζει την εξής συνάρτηση:

- notifyNewAlarm :
  Ο GoalNodeVerifier ενημερώνεται μέσω του Pub/Sub Module για για τον κόμβο που κάθε φορά πρέπει να επαληθευτεί μέσω της διεπαφής notifyNewAlarm.

Για να υλοποιήσει τις παραπάνω λειτουργίες του, ο GoalNodeVerifier περιέχει τις εξής υπολειτουργίες :

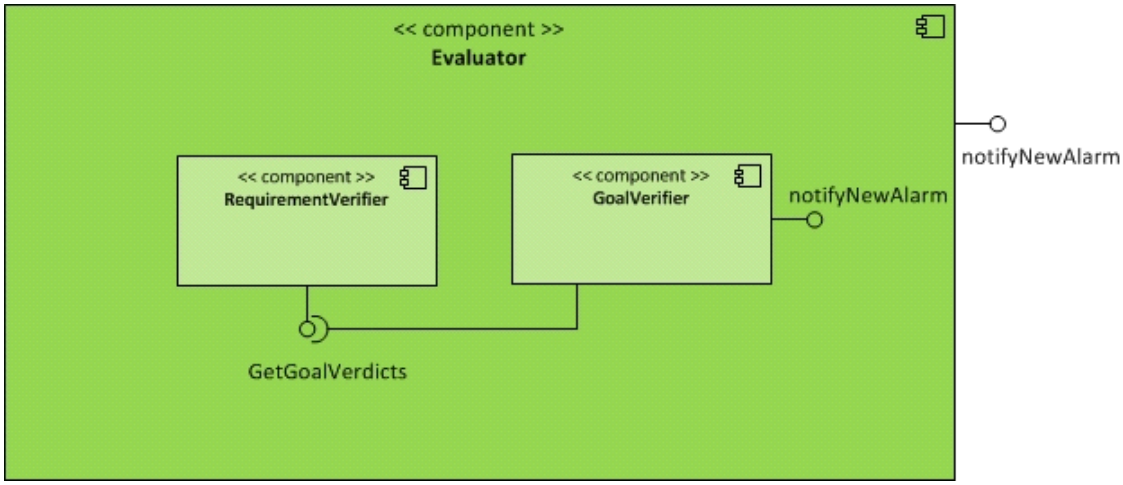
- GoalNodeTestertoTestStrategies :
  Στόχος αυτού του component είναι η εύρεση του GoalNodeTester που είναι συνδεδεμένο με τον κόμβο προς επαλήθευση. Όπως έχουμε ήδη αναφέρει, κάθε GoalNodeTester αποτελείται από μία ή περισσότερες στρατηγικές (Test Strategies). Σκοπός λοιπόν του GoalNodeTestertoTestStrategies είναι να ειδοποιήσει μέσω της συνάρτησης getHasTestStrategy τον component RunTestStrategy, για το ποιές στρατηγικές πρέπει να εκτελεστούν ώστε να μπορέσει να εξαχθεί κάποιο συμπέρασμα για την ισχύ του κόμβου.

- RunTestStrategy :
  Στόχος αυτού του component είναι η εκτέλεση των στρατηγικών που είναι συνδεδεμένες με έναν κόμβο, με στόχο την ενημέρωση του Blackboard για τις αληθοτιμή του. Είναι υπεύθυνος για την εκτέλεση της συνάρτησης DO(), η οποία περιέχει μία δοκιμή ελέγχου ισχύος της υπόθεσης που σχετίζεται με έναν κόμβο, και επιστρέφει την αληθοτιμή αυτού. Στο παράδειγμά μας, η αληθοτιμή εκφράζεται ως "αληθής" στην περίπτωση που η στρατηγική επιτυγχάνει, και ως "λάθος" στην περίπτωση που η στρατηγική αποτυγχάνει.

3.3.5 Δομικό στοιχείο Evaluator

Το δομικό στοιχείο Evaluator είναι υπεύθυνο για την απόφαση της ισχύος της αρχικής απαίτησης, δηλαδή του κόμβου-ρίζα του δέντρου στόχου υπό έλεγχο. Ενημερώνεται μέσω μίας διεπαφής του από τον Pub/Sub για την απαίτηση που πρέπει να επαληθευτεί, για τους συνδυασμούς των κόμβων που την επαληθεύουν, καθώς και τις αληθοτιμές των κόμβων που είναι ήδη διαθέσιμες. Για την εξαγωγή του τελικού συμπεράσματος, διασταυρώνει εάν οι διαθέσιμες αληθοτιμές των αποδεδειγμένων κόμβων οδηγούν στην επαλήθευση της ισχύος της αρχικής απαίτησης, δηλαδή εάν ικανοποιείται ο συνδυασμός κόμβων που έχει προκύψει από τον SAT Solver.

Τα παραπάνω επιτυγχάνονται με τη βοήθεια των διεπαφών και των υπολειτουργιών που φαίνονται στο επόμενο σχήμα, και αναλύονται παρακάτω.



Σχήμα 3.8: Ψηφιδικό διάγραμμα του Evaluator

Η διεπαφή του Evaluator component ορίζει την εξής συνάρτηση:

- notifyNewAlarm :
  Ο Evaluator ενημερώνεται μέσω του Pub/Sub Module για την απαίτηση που πρέπει να επαληθευτεί, το δέντρο στόχων που την αναπαριστά και τις αληθοτιμές που είναι διαθέσιμες στον Blackboard σχετικά με τους κόμβους που έχουν ήδη επαληθευτεί, μέσω της διεπαφής notifyNewAlarm.

Για να υλοποιήσει τις παραπάνω λειτουργίες του, ο Evaluator περιέχει τις εξής υπολειτουργίες :

- GoalVerifier:
  Συλλέγει τις αληθοτιμές των κόμβων όπως προκύπτουν από τον SAT solver και επαληθεύονται από τους GoalNodeTesters. Η διεπαφή του Evaluator ορίζει τη συνάρτηση "GetGoalVerdicts" η οποία ενημερώνει τον RequirementVerifier για τις αληθοτιμές του κάθε κόμβου-στόχου.

- RequirementVerifier:
  Συλλέγει τις αληθοτιμές των κόμβων-στόχων όπως προκύπτουν από τον GoalVerifier και αποφαίνεται για την ισχύ της αρχικής απαίτησης.

3.4 Επικοινωνία μεταξύ των δομικών στοιχείων του συστήματος

Στην ενότητα αυτή θα περιγράψουμε περιγραφικά αλλά και με την χρήση ακολουθιακών διαγραμμάτων κάθε μια από τις λειτουργίες του συστήματος αλλά και τον τρόπο επικοινωνίας των δομικών στοιχείων του.

Όπως έχουμε προαναφέρει το περιβάλλον πλαίσιο που αναπτύχθηκε βασίστηκε στα αρχιτεκτονικά μοτίβα Publisher/subscriber και blackboard. Έτσι η επικοινωνία μεταξύ των διάφορων δομικών στοιχείων του συστήματος είναι άμεσα συνδεδεμένη με την ύπαρξη αποστολέων και παραληπτών που εκδηλώνουν ενδιαφέρον για συγκεκριμένο είδος μηνυμάτων καθώς και ενός δομικού στοιχείου το οποίο θα είναι υπεύθυνο για την συλλογή όλων αυτών τον μηνυμάτων και την ενημέρωση των ενδιαφερόμενων παραληπτών. Το ρόλο αυτό έχει αναλάβει το δομικό στοιχείο blackboard.
Πριν την εκκίνηση του συστήματος απαιτείται η εισαγωγή των δεδομένων που περιγράφουν το υπό έλεγχο σύστημα (requirements) . Τα δεδομένα αυτά μοντελοποιούνται σε μορφή δέντρων στόχων τοποθετώντας στην ρίζα του δέντρου τη βασική απαίτηση (requirement) και στους υπόλοιπους κόμβους τις υπό-απαιτήσεις(υπολειτουργίες) του υπό έλεγχο συστήματος. Κάθε δέντρο στόχων πρέπει να είναι συσχετισμένο με ένα η περισσότερα alarms και κάθε κόμβος του δέντρου μπορεί να έχει μία ή περισσότερες στρατηγικές επαλήθευσης του. Μόνο όταν η πληροφορία αυτή είναι διαθέσιμη το σύστημα μας μπορεί να τεθεί σε λειτουργία.
Το σύστημα ξεκινά όταν ο component Alarmer παρατηρήσει την ύπαρξη κάποιου ενεργοποιημένου alarm είτε στο μοντέλο δέντρων στόχων που δόθηκε, είτε στις πληροφορίες εκτέλεσης του υπό έλεγχο συστήματος. Ο Alarmer ενημερώνει τον blackboard για την εμφάνιση του alarm. Ο blackboard με τη σειρά του ενημερώνει τους ενδιαφερόμενους για την ύπαρξη του alarm . Το δομικό στοιχείο που ακούει σε τέτοιου είδους μηνύματα , είναι το GoalSelector. Αυτό το δομικό στοιχείο βρίσκει τον βασικό κόμβο του μοντέλου στόχων το οποίο συσχετίζεται με το alarm και ενημερώνει τον blackboard. Μέσω του blackboard ο component satSolver ενημερώνεται για την ανάγκη εύρεσης των βέλτιστων μονοπατιών για την απόδειξη αυτού του βασικού κόμβου . Στη συνέχεια ο component Evaluator ενημερώνεται μέσω του blackboard για το σύνολο των προς απόδειξη κόμβων. Εάν ο κόμβος έχει επισημανθεί στο μοντέλο δέντρων ως πάντα αληθής τότε σημειώνεται και στο σύστημα μας ως αληθής και η διαδικασία συνεχίζεται με τον επόμενο κόμβο. Εάν για την απόδειξη του κόμβου απαιτείται να τρέξει κάποια στρατηγική επαλήθευσης , ο Evaluator ενημερώνει το σύστημα για αυτή την ανάγκη . Το δομικό στοιχείο του συστήματος που ακούει σε αυτή την περίπτωση είναι ο GoalNodeVerifier, ο οποίος και ξεκινά την στρατηγική επαλήθευσης. Στο σημείο αυτό αν η επαλήθευση του κόμβου αποτύχει , δοκιμάζεται άλλη λύση του SAT solver. Η διαδικασία αυτή συνεχίζεται μέχρις ότου είτε επαληθευτούν όλοι οι προς απόδειξη κόμβοι , είτε η απόδειξη όλων των λύσεων του SAT solver αποτύχει.
Δεδομένου ότι το μοντέλο δέντρων στόχων που δίνεται ως είσοδος στο σύστημα μπορεί να περιέχει περισσότερα από ένα δέντρα, καθώς και ότι μπορεί να παρουσιαστεί ενεργοποίηση πολλών alarm ταυτόχρονα, γίνεται επιτακτική η ανάγκη παράλληλης εκτέλεσης των παραπάνω λειτουργιών. Αυτό επιτυγχάνεται με την χρήση threads [56] για τα παρακάτω δομικά στοιχεία: goalSelector, goalNodeVerifier, evaluator, alarmer και blackboard.

Με βάση την παραπάνω περιγραφή του συστήματος μπορούμε να χωρίσουμε τις λειτουργίες του συστήματος στις εξής κατηγορίες:

- Εισαγωγή δεδομένων.

- Ενεργοποίηση alarm.
- Επίλυση προβλήματος SAT για το συσχετιζόμενο κόμβο.
- Ενεργοποίηση του πλάνου επαλήθευσης του συνόλου των κόμβων της λύσης SAT.
- Εκτέλεση της αντίστοιχης στρατηγικής έλεγχου.

Στις επόμενες ενότητες ακολουθεί η αναπαράσταση αυτών των βασικών λειτουργιών με τη χρήση ακολουθιακών διαγραμμάτων.

3.4.1 Ακολουθιακό Διάγραμμα Εισαγωγής Δεδομένων

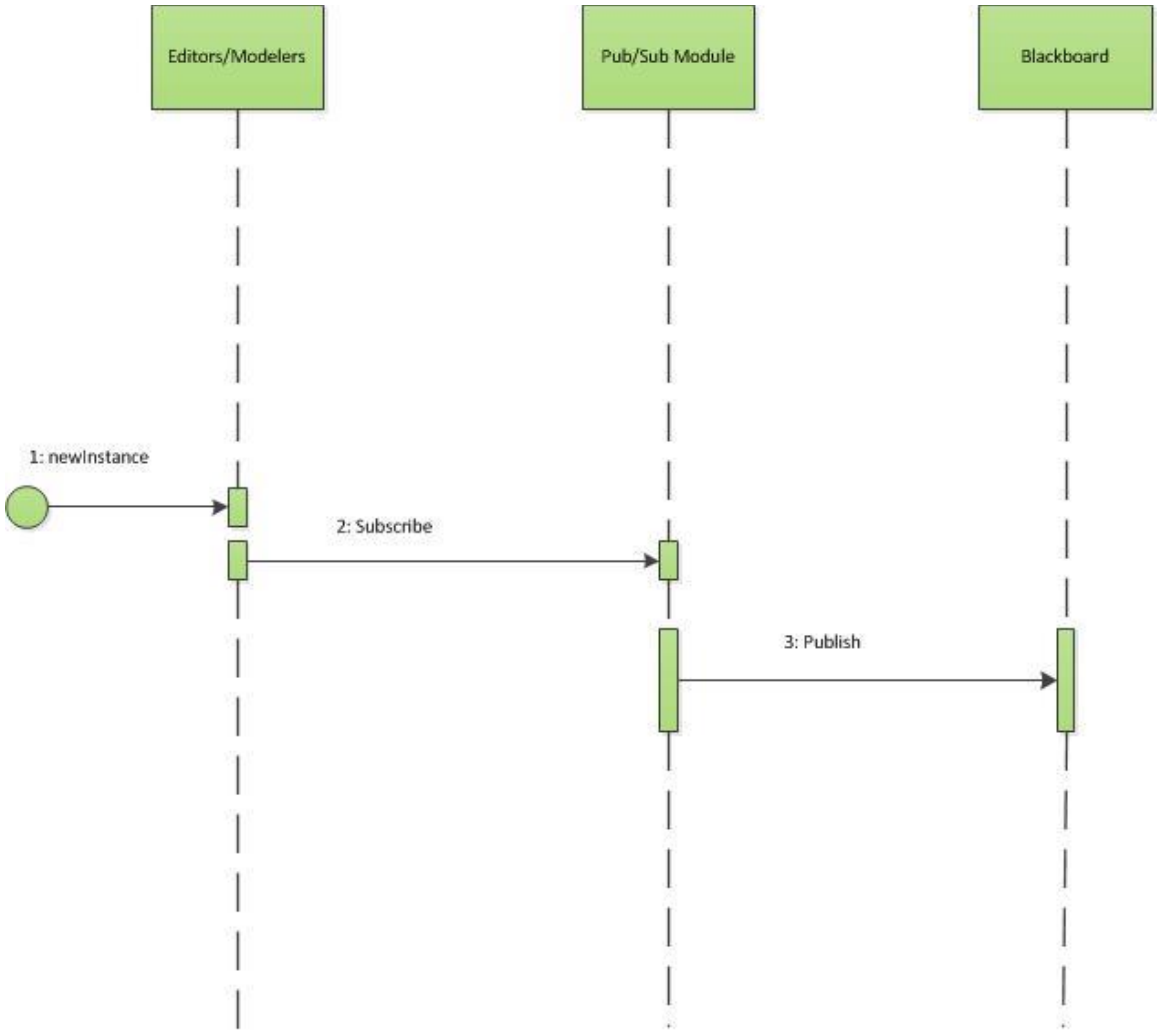
Στην ενότητα αυτή περιγράφεται η διαδικασία εισόδου των δεδομένων από το χρήστη που απατούνται για τη λειτουργία του περιβάλλοντος-πλαισίου που αναπτύχθηκε στην παρούσα διπλωματική εργασία. Τα δεδομένα αυτά είναι το σύνολο των απαιτήσεων του υπό έλεγχο συστήματος τα οποία έχουν μοντελοποιηθεί σε δέντρα στόχων μέσω του Eclipse Modeling Framework. Αναλυτικά τα βήματα εισαγωγής των δέντρων στόχων είναι τα εξής:

1. **newInstance:** Το δέντρο στόχων, το οποίο περιλαμβάνει όλες τις απαραίτητες πληροφορίες σχετικά με τις απαιτήσεις του υπό έλεγχο συστήματος, δημιουργείται από το χρήστη με τη βοήθεια του Eclipse Modeling Framework(Editors/Modelers).

2. **subscribe:** Στο στάδιο αυτό το δέντρο στόχων που περιέχει της πληροφορίες του χρήστη καθώς και το μοντέλο των δομικών στοιχείων του συστήματος που περιγράφει την αλληλεξάρτηση τους, γνωστοποιείται στο σύστημα μέσω του Pub/Sub module.

3. **publish:** Το Pub/Sub module ενημερώνει τον blackboard , παρέχοντας του όλες τις πληροφορίες για το διαθέσιμο δέντρο αποφάσεων και το μοντέλο του συστήματος.

Στο ακολουθιακό διάγραμμα που ακολουθεί παρουσιάζεται η διαδικασία εισόδου του δέντρου στόχων (Σχήμα 3.9).



Σχήμα 3.9:Ακολουθιακό Διάγραμμα Εισαγωγής Δεδομένων

3.4.2 Ακολουθιακό Διάγραμμα ενεργοποίησης Alarm

Στην ενότητα αυτή περιγράφεται η διαδικασία ενεργοποίηση ενός alarm. Ένα alarm ενεργοποιείται από τον Alarmer με δύο τρόπους . Ο πρώτος τρόπος είναι η παρατήρηση των πληροφοριών εκτέλεσης του υπό έλεγχο συστήματος. Όταν το υπό έλεγχο σύστημα παρουσιάσει

κάποια δυσλειτουργία εισάγει στις πληροφορίες εκτέλεσης κάποια επισήμανση για τη δυσλειτουργία αυτή. Ο Alarmer , ο οποίος παρακολουθεί τις πληροφορίες αυτές καταλαβαίνει την ύπαρξη σφάλματος και ξεκινά τη διαδικασία ελέγχου της απαίτησης που σχετίζεται με την επισήμανση που εμφανίστηκε. Ο δεύτερος τρόπος είναι η εισαγωγή από το χρήστη παρόμοιου τύπου επισημάνσεων μέσω του δέντρου στόχων. Ποιό αναλυτικά τα βήματα έγερσης ενός alarm είναι τα εξής:

1. **notifynewAlarm:** Ο alarmer λαμβάνει τις πληροφορίες εκτέλεσης του υπό έλεγχο συστήματος και τις πληροφορίες του δέντρου στόχων που σχετίζονται με τα alarm.

2. **processEvent:** Ο alarmer επεξεργάζεται τα δεδομένα που πήρε με σκοπό να ανακαλύψει κάποια επισήμανση για δυσλειτουργία του υπό έλεγχο συστήματος. Στην περίπτωση που δεν εντοπιστεί κάποια τέτοιου είδους επισήμανση το σύστημα περιμένει μέχρι να έρθουν νέα δεδομένα (ενέργεια 6) .

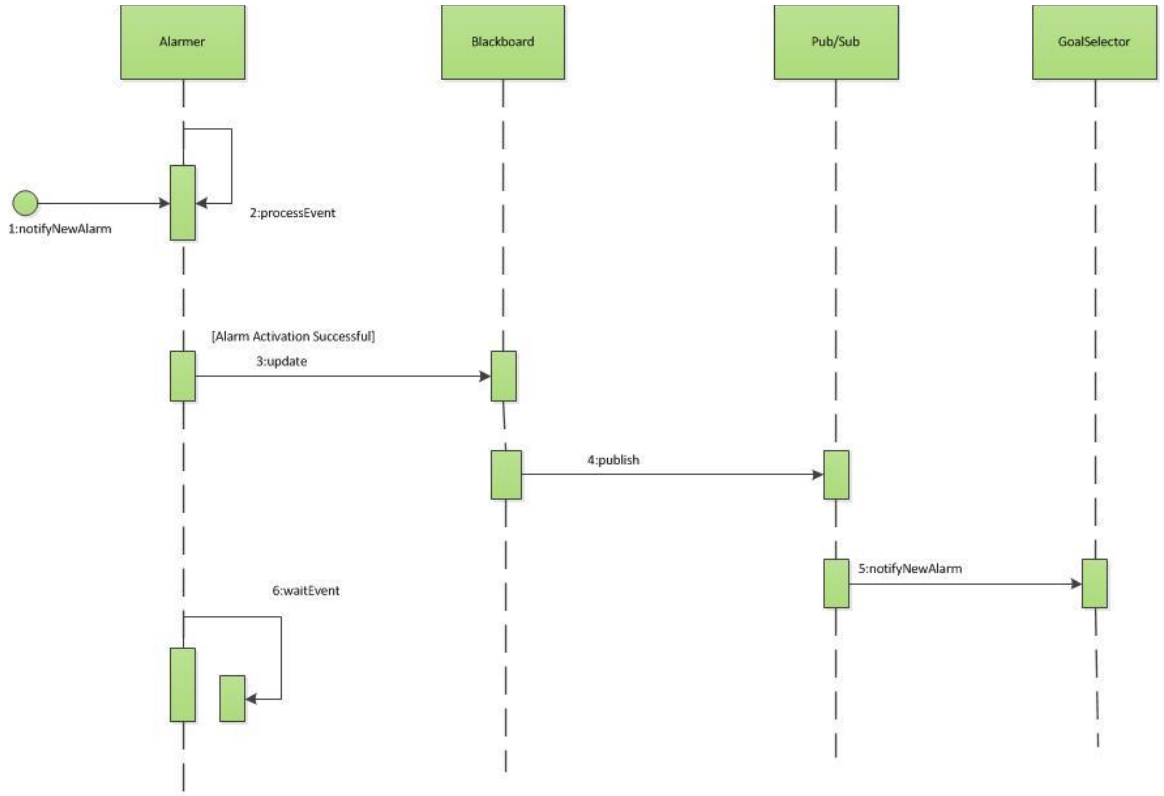
3. **update:** Ο Alarmer ενεργοποιεί το alarm και ενημερώνει τον blackboard για την εμφάνιση του.

4. **publish:** Ο blackboard ενημερώνει το pub/sub module για το alarm που μόλις ενεργοποιήθηκε , ώστε με τη σειρά του να ενημερώσει όλα τα υπόλοιπα δομικά στοιχεία του συστήματος.

5. **notifyNewAlarm**: Ο GoalNodeSelector ενημερώνεται για την ενεργοποίηση του alarm προκειμένου να βρει τον κόμβο του δέντρου στόχων που σχετίζεται με αυτό το alarm.

6. **waitEvent:** Το σύστημα βρίσκεται σε αναμονή , μέχρι την νέα δεδομένα να γίνουν διαθέσιμα.

Το ακολουθιακό διάγραμμα που περιγράφει τη διαδικασία ενεργοποίησης ενός alarm φαίνεται παρακάτω (Σχήμα 3.10) .



Σχήμα 3.10: Ακολουθιακό Διάγραμμα ενεργοποίησης Alarm

3.4.3 Ακολουθιακό Διάγραμμα Επίλυσης προβλήματος SAT για το συσχετιζόμενο κόμβο

Στην ενότητα αυτή περιγράφεται η διαδικασία εύρεσης των συνδυασμών κόμβων που απαιτείται να αποδειχθούν ώστε να μπορούμε να αποφανθούμε για την ισχύ ή όχι του βασικού κόμβου που εμφάνισε το alarm. Αναλυτικά τα βήματα είναι τα εξής:

1. **notifyNewAlarm:** Ο GoalSelector ενημερώνεται για την ενεργοποίηση κάποιου alarm.

2. **matchAlarmWithGoalModel :** Βρίσκει την αντιστοιχία μεταξύ alarm και δέντρου στόχων και υπολογίζει την στρατηγική για την επαλήθευση των υποθέσεων που σχετίζονται με τα alarms.
3. **update:** Οι βασικοί κόμβοι (ρίζες των δέντρων στόχων) που πρέπει να αποδειχτούν προστίθενται στην λίστα του blackboard.

4. **update:** Ο GaolModelToCNFConverter ενημερώνεται για το δέντρο στόχων το όποιο πρέπει να επεξεργαστεί.

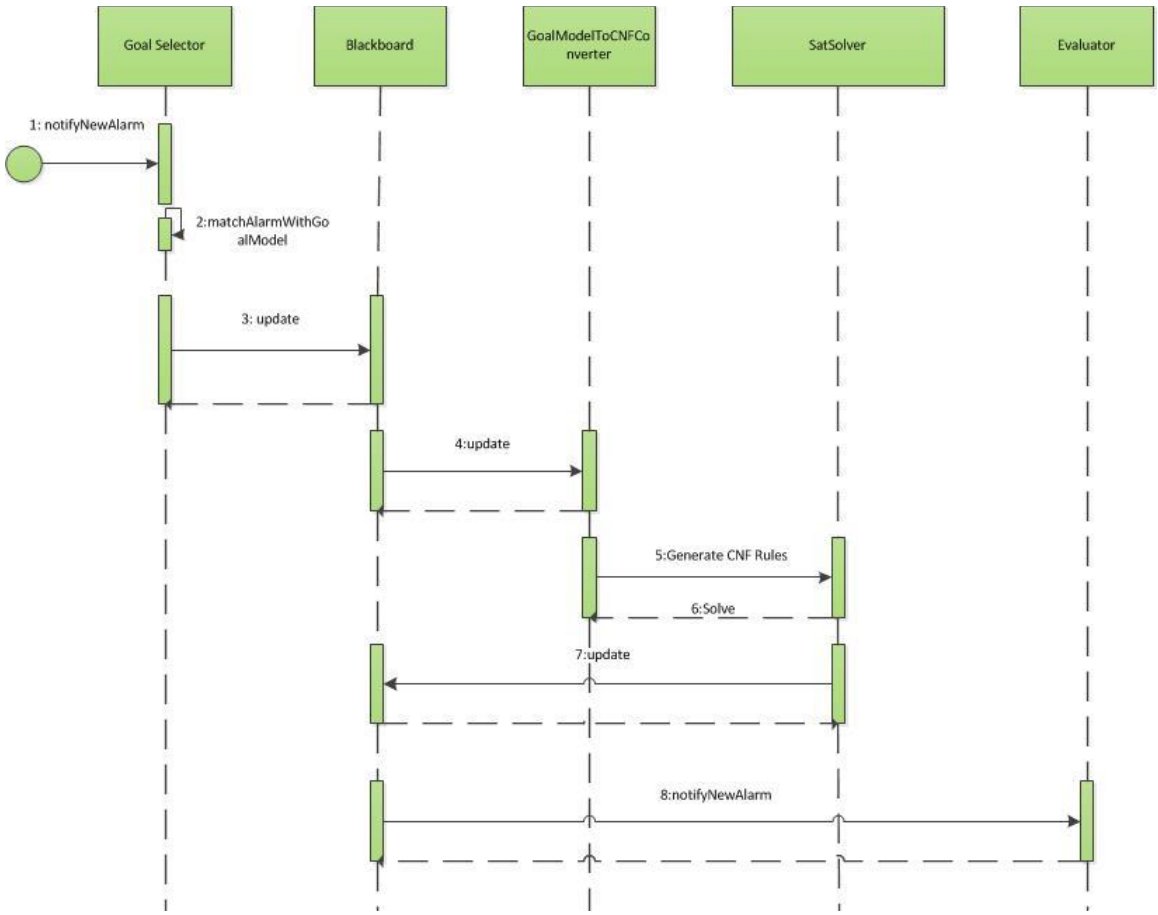
5. **GenerateCNFrules:** στο στάδιο αυτό γίνεται μετατροπή των δέντρων στόχων σε μορφή CNF. Αυτό γίνεται με τη βοήθεια του GaolModelToCNFConverter.

6. **solve:** Ο SatSolver επιλύει το πρόβλημα βρίσκοντας το σύνολο των συνδυασμών κόμβων των οποίων η απόδειξη συνεπάγεται την απόδειξη του κόμβου ρίζα.

7. **update:** Ο blackboard ενημερώνεται για το σύνολο των υπό-κόμβων που πρέπει να ελεγχτούν.

8. **notifyNewAlarm**: Ο blackboard ενημερώνει των Evaluator για την ύπαρξη κόμβων προς απόδειξη.

Ακολουθεί το ακολουθιακό διάγραμμα της επίλυσης του προβλήματος SAT(Σχήμα).



Σχήμα 3.11: Ακολουθιακό Διάγραμμα Επίλυση προβλήματος SAT για το συσχετιζόμενο κόμβο

3.4.4 Ακολουθιακό Διάγραμμα πλάνου επαλήθευσης του συνόλου των κόμβων της λύσης SAT

Στην ενότητα αυτή θα περιγράψουμε το τρόπο με τον οποία γίνεται η επαλήθευση των κόμβων οι οποίοι έχουν επιστραφεί από την επίλυση του προβλήματος SAT. Αναλυτικά η διαδικασία αυτή χωρίζεται στα παρακάτω στάδια:

1. **notifyNewAlarm:** Ο Evaluator ενημερώνεται για τον κόμβο του δέντρου που πρέπει να επαληθευτεί

   1.1 **proveHypothesis:** Ο Evaluator προσπαθεί να αποδείξει τον κόμβο είτε άμεσα, κάνοντας χρήση των πληροφοριών του μοντέλου στόχων (για παράδειγμα ένας κόμβος μπορεί να θεωρείται πάντα αληθής), είτε έμμεσα μέσω της απόδειξης υπό-κόμβων, είτε μέσω της εκτέλεσης στατικού ή δυναμικού ελέγχου. Έτσι , προκύπτουν τρεις διαφορετικές καταστάσεις στις οποίες μπορεί να βρεθεί ο Evaluator , όπως φαίνεται και στο διάγραμμα, η κατάσταση όπου ο κόμβος μπορεί να αποδειχθεί άμεσα , η κατάσταση όπου χρειάζεται να επαληθευτούν υπό-κόμβοι και η κατάσταση όπου χρειάζεται να εκτελεστεί κάποια στρατηγική ελέγχου .

2. **update:** Ο Evaluator ενημερώνει τον blackboard για την επαλήθευση ή όχι του κόμβου.
   2.1 **publish**: Ο Blackboard δημοσιεύει το γεγονός της επαλήθευσης ή της άρνησης του κόμβου αυτού έτσι ώστε να ενημερωθούν αυτόματα όλα τα ενδιαφερόμενα components.

3. **GeneratePseudoAlarm:** Ο Evaluator ενεργοποιεί ένα νέο ψευδο-alarm για το κόμβο-στόχο που θέλει να αποδείξει.

4. **update:** Ο blackboard ενημερώνεται για το νέο αυτό ψευδο-alarm.

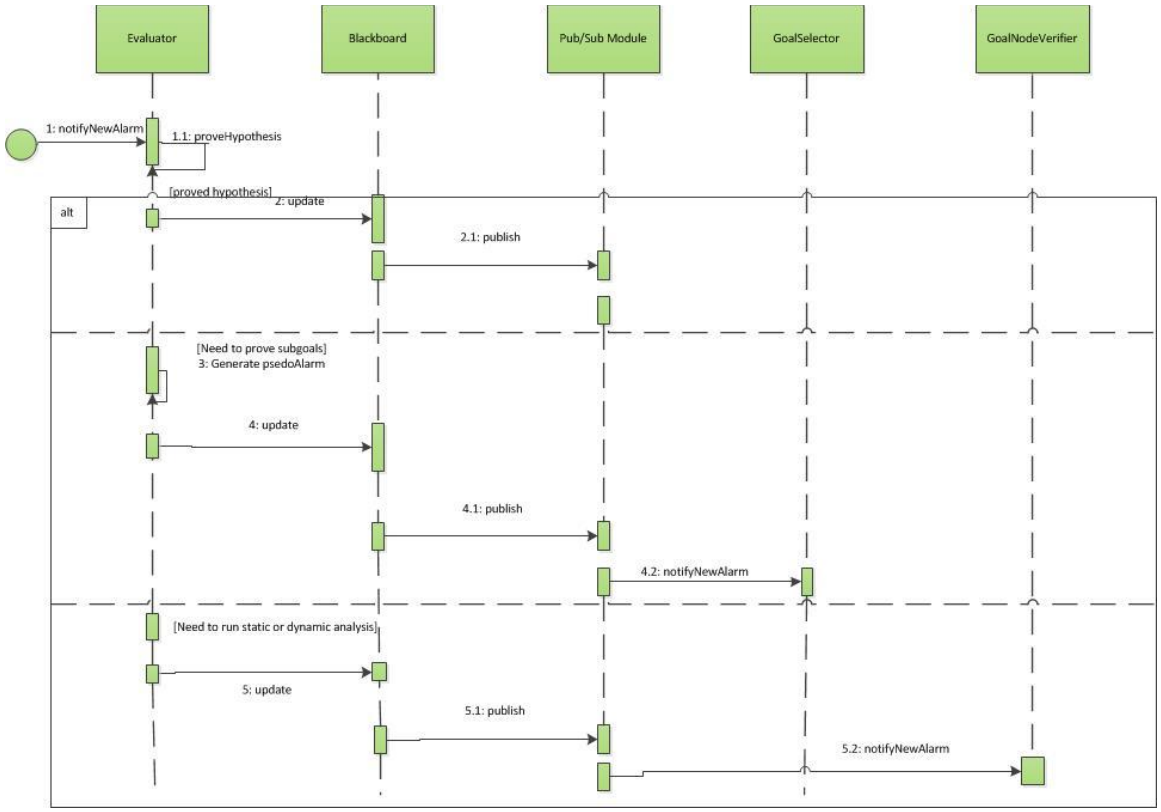
   4.1 **publish:** Ο blackboard ενημερώνει όλα τα ενδιαφερόμενα δομικά στοιχεία του συστήματος για την ύπαρξη του νέου ψευδο-alarm μέσω του pub/sub module.

   4.2 **notifyNewAlarm:** Το pub/sub module ενημερώνει τον goalSelector για το νέο alarm . Η διαδικασία που θα εκκινήσει ο goalSelector λόγω αυτής της ενημέρωσης έχει περιγραφθεί στην ενότητα 3.4.3 .

5. **update:** Ο Evaluator ενημερώνει τον blackboard για την ανάγκη εκτέλεσης δυναμικής ή στατικής ανάλυσης.

   5.1 **publish:** Ο blackboard δημοσιεύει την ανάγκη αυτή στο pub/sub module , ώστε να ενημερωθούν όλα τα ενδιαφερόμενα components.

   5.2 **notifyNew Alarm:** Το Pub/Sub Module ενημερώνει τον GoalNodeVerifier για την ανάγκη εκτέλεσης στατικής ή δυναμικής ανάλυσης.



Σχήμα 3.12 : Ακολουθιακό Διάγραμμα πλάνου επαλήθευσης του συνόλου των κόμβων της λύσης SAT

3.4.5 Ακολουθιακό Διάγραμμα εκτέλεσης στρατηγικής ελέγχου.

Στο παρακάτω διάγραμμα περιγράφεται η διαδικασία εκτέλεσης της στατικής η δυναμικής ανάλυσης. Η διαδικασία αυτή χωρίζεται στα εξής στάδια:

1. **notifyNewAlarm:** Ο GoalNodeVerifier ενημερώνεται για την ανάγκη εκτέλεσης στατικής ή δυναμικής ανάλυσης.

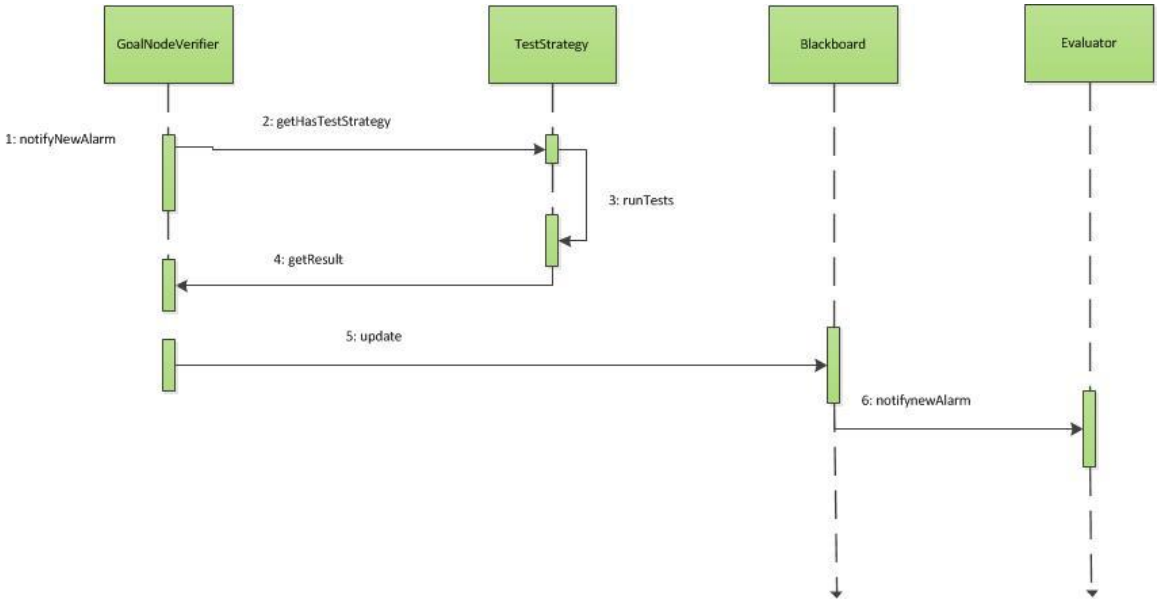
2. **getHasTeststrategy:** Στο στάδιο αυτό επιλέγεται η στρατηγική για την επαλήθευση του κόμβου-στόχου.

3. **runTests:** Στο στάδιο αυτό εκτελείται η στρατηγική επαλήθευσης που είχε επιλεχθεί.

4. **getResult:** Ο GoalNodeVerifier ενημερώνεται για το αποτέλεσμα της στρατηγικής επαλήθευσης.

5. **update:** Το αποτέλεσμα αυτό καταγράφεται στο blackboard.

6. **notifyNewAlarm:** Ο blackboard ενημερώνει το Evaluator για την ισχύ ή όχι του κόμβου-στόχου. Η διαδικασία που θα ακολουθηθεί στη συνέχεια από τον Evaluator περιγράφθηκε στην ενότητα 3.4.4 .



Σχήμα 3.13: Ακολουθιακό Διάγραμμα εκτέλεσης στρατηγικής ελέγχου

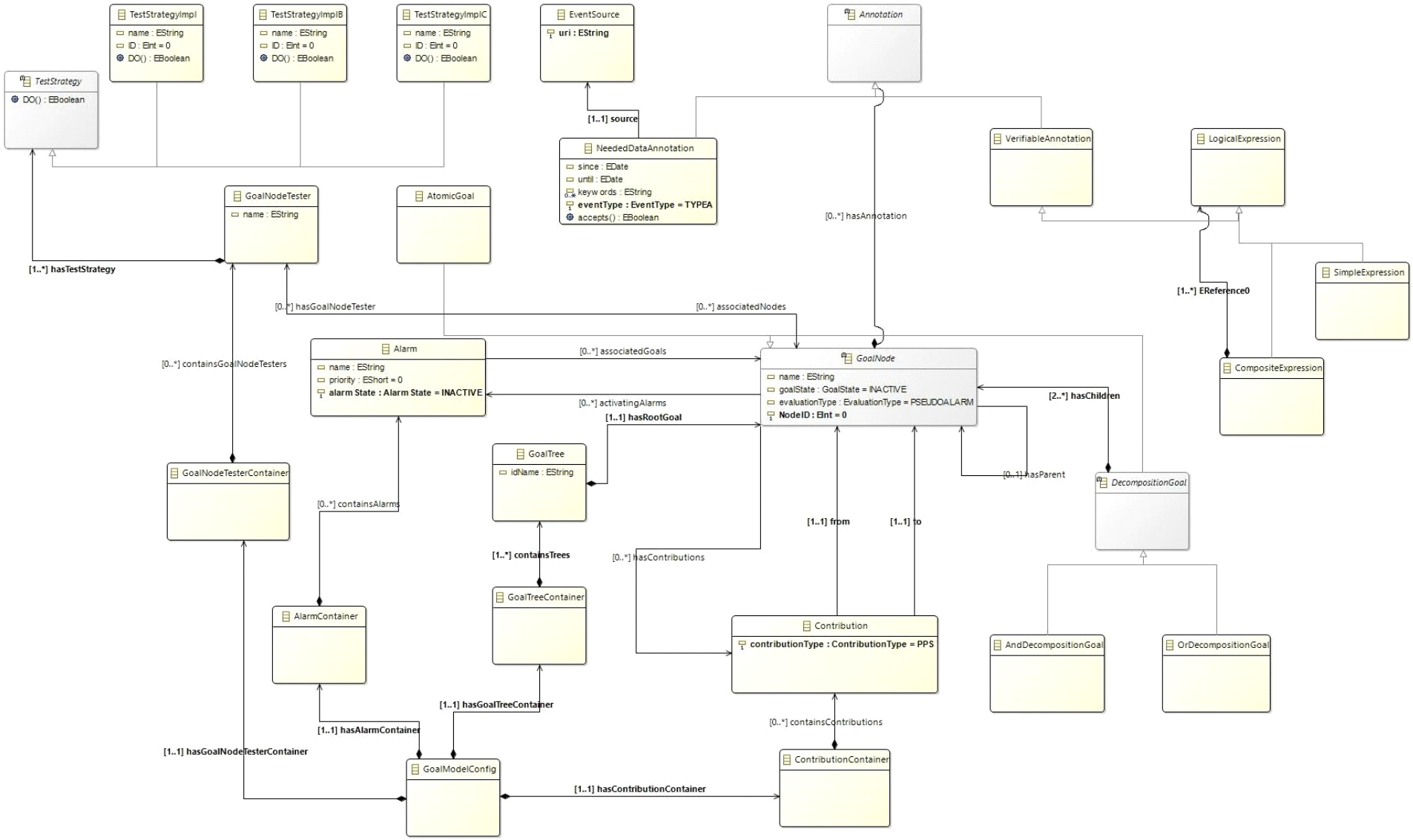
Κεφάλαιο 4: Εννοιολογικό Μοντέλο Δέντρων Στόχων

4.1 Εισαγωγή

Στο κεφάλαιο αυτό θα περιγράψουμε το εννοιολογικό μοντέλο (domain model) των δέντρων στόχων που θα χρησιμοποιηθούν στο παρόν περιβάλλον πλαίσιο. Τα εννοιολογικά μοντέλα είναι ένας τρόπος να περιγράψουμε και να μοντελοποιήσουμε οντότητες καθώς και τις σχέσεις μεταξύ τους, οι οποίες περιγράφουν συλλογικά το χώρο ενός προβλήματος που καλούμαστε να λύσουμε. Προϋποθέτει κατανόηση των απαιτήσεων του συστήματος (System Level Requirements) και παρέχει μία αποδοτική βάση για τον σχεδιασμό συστημάτων με στόχο την συντηρησιμότητα, την σταδιακή ανάπτυξη και την ευκολία ελέγχου τους για σφάλματα. Επειδή συχνά υπάρχει ένα χάσμα μεταξύ της κατανόησης του συστήματος και την ερμηνεία των απαιτήσεων, τα εννοιολογικά μοντέλα είναι μία πρωτεύουσα ιδέα μοντελοποίησης στην Agile μεθοδολογία σε ευρεία κλίμακα. Η ιδέα αυτή συνδέεται εν μέρει με αντικειμενοστραφείς σχεδιαστικές προσεγγίσεις και επιδιώκει τη λύση ενός προβλήματος με τη χρήση ενός συνόλου οντοτήτων που συνεργάζονται για να ικανοποιήσουν τις απαιτήσεις του συστήματος.

4.2 Επισκόπηση Εννοιολογικού Μοντέλου Δέντρων Στόχων

Το εννοιολογικό μοντέλο δέντρων στόχων περιγράφει τις οντότητες από τις οποίες αποτελούνται τα δέντρα στόχων του περιβάλλοντος πλαισίου καθώς επίσης και τις σχέσεις μεταξύ τους. Το EMF βασίζεται στο εννοιολογικό μοντέλο δέντρων στόχων για την παραγωγή του κώδικα του περιβάλλοντος-πλαισίου, και συνεπώς ακολουθεί τις προδιαγραφές MOF.
Το εννοιολογικό μοντέλο δέντρων στόχων υποστηρίζει τις έννοιες της σύζευξης (ΚΑΙ) , απόζευξης (Ή) και συνεισφοράς, οι οποίες παίζουν σημαντικό ρόλο στην περιγραφή των συστημάτων πάνω στο οποίο θα εφαρμοστεί το περιβάλλον πλαίσιο της παρούσας διπλωματικής εργασίας. Επιπλέον υποστηρίζει την μετατροπή των μοντέλων δέντρων στόχων σε εκφράσεις της μορφής CNF, οι οποίες δίνονται σαν είσοδος στον SAT4j για την ανάλυση αρχικών αιτιών , δηλαδή των πιθανών συνδυασμών κόμβων που πρέπει να ελεγχθούν.
Ακολουθεί το διάγραμμα του εννοιολογικού μοντέλου δέντρων, καθώς και μία περιγραφή της κάθε κλάσης που το αποτελεί.


TestStrategyImpl
name : EString
ID : EInt = 0
DO() : EBoolean
TestStrategyImplB
name : EString
ID : EInt = 0
DO() : EBoolean
TestStrategyImplC
name : EString
ID : EInt = 0
DO() : EBoolean
EventSource
uri : EString
Annotation
TestStrategy
DO() : EBoolean
[1..1] source
NeededDataAnnotation
since : EDate
until : EDate
keywords : EString
eventType : EventType = TYPEA
accepts() : EBoolean
VerifiableAnnotation
LogicalExpression
GoalNodeTester
name : EString
AtomicGoal
[0..*] hasAnnotation
[1..*] hasTestStrategy
[1..*] EReference0
SimpleExpression
[0..*] hasGoalNodeTester
[0..*] associatedNodes
Alarm
name : EString
priority : EShort = 0
alarm State : Alarm State = INACTIVE
[0..*] associatedGoals
GoalNode
name : EString
goalState : GoalState = INACTIVE
evaluationType : EvaluationType = PSEUDOALARM
NodeID : EInt = 0
CompositeExpression
[0..*] containsGoalNodeTesters
[0..*] activatingAlarms
[2..*] hasChildren
[1..1] hasRootGoal
GoalTree
idName : EString
GoalNodeTesterContainer
[0..1] hasParent
DecompositionGoal
[0..*] containsAlarms
[1..1] from
[1..1] to
[1..*] containsTrees
[0..*] hasContributions
GoalTreeContainer
AlarmContainer
Contribution
contributionType : ContributionType = PPS
AndDecompositionGoal
OrDecompositionGoal
[1..1] hasGoalTreeContainer
[1..1] hasAlarmContainer
[0..*] containsContributions
[1..1] hasGoalNodeTesterContainer
GoalModelConfig
ContributionContainer
[1..1] hasContributionContainer

Σχήμα 4.1: Εννοιολογικό Μοντέλο Δέντρων Στόχων

4.2.1 Η κλάση GoalTree

Η κλάση GoalTree αντιπροσωπεύει τη βασική δομή του μοντέλου μας, ένα δέντρο στόχων. Στην ουσία αναπαριστά μία απαίτηση του συστήματος (requirement) , χωρισμένη σε υπο-απαιτήσεις με τη βοήθεια δέντρων. Αυτή η κλάση έχει την ιδιότητα name τύπου String που αποτελεί το πρωτεύον κλειδί της κλάσης.

4.2.2 Η κλάση GoalNode

Η κλάση GoalNode αντιπροσωπεύει τους κόμβους του δέντρου στόχων, δηλαδή μία υπο-απαίτηση του συστήματος. Αυτή η κλάση έχει τις ιδιότητες name τύπου String και NodeID τύπου Integer . Η ιδιότητα NodeID αποτελεί το πρωτεύον κλειδί της της κλάσης.
Κάθε κόμβος του δέντρου στόχων, εκτός από τη ρίζα, έχει έναν κόμβο-πατέρα, και δύο κόμβους παιδιά. Εδώ θα πρέπει να διαχωρίσουμε την κλάση GoalNode από την κλάση AtomicGoal, η οποία θα αναλυθεί και παρακάτω. Η διαφορά τους είναι πως ένας κόμβος GoalNode έχει δύο κόμβους παιδιά, ενώ ένας κόμβος AtomicGoal δεν έχει κόμβους παιδιά. Η σχέση ενός κόμβου με τον πατέρα κόμβο, περιγράφεται από την συσχέτιση της κλάσης GoalNode με τον εαυτό της μέσω της ιδιότητας hasParent με πολλαπλότητα 1. Η σχέση ενός κόμβου με τους κόμβους παιδιά του, περιγράφεται από την συσχέτιση της κλάσης GoalNode με την κλάση DecompositionGoal μέσω της ιδιότητας hasChildren με πολλαπλότητα 2..*.
Η σχέση ενός κόμβου με τις συνεισφορές που συνεισφέρουν στον κόμβο αυτό περιγράφεται από την συσχέτιση της κλάσης GoalNode με την κλάση Contribution μέσω της ιδιότητας from με πολλαπλότητα 1. Η σχέση ενός κόμβου με τις συνεισφορές στις οποίες συνεισφέρει ο κόμβος αυτός περιγράφεται από την συσχέτιση της κλάσης GoalNode με την κλάση Contribution μέσω της ιδιότητας to με πολλαπλότητα 1.
Η κλάση GoalNode συσχετίζεται με την κλάση GoalTree μέσω της ιδιότητας hasRootGoal με πολλαπλότητα 1, υποδηλώνοντας την ρίζα του δέντρου στόχου. Επίσης συσχετίζεται με την κλάση Alarm μέσω της ιδιότητας activatingAlarms με πολλαπλότητα 0..*, υποδηλώνοντας πως ένας κόμβος του δέντρου στόχου μπορεί να είναι συνδεδεμένος με κανένα, ένα ή περισσότερα alarms. Μία ακόμη συσχέτιση της κλάσης GoalNode αποτελεί η συσχέτιση με τον GoalNodeTester, μέσω της ιδιότητας hasGoalNodeTester με πολλαπλότητα 0..*, υποδηλώνοντας πως ένας κόμβος του δέντρου στόχου μπορεί να είναι συνδεδεμένος με κανένα, ένα ή περισσότερα GoalNodeTesters. Αυτό σημαίνει πως ένας κόμβος μπορεί να είναι συνδεδεμένος με ένα σύνολο των στρατηγικών που απαιτούνται για τη επαλήθευσή του. Τέλος, η κλάση GoalNode συσχετίζεται με την κλάση Annotation μέσω της ιδιότητας hasAnnotation με πολλαπλότητα 0..*, υποδηλώνοντας πως ένας κόμβος του δέντρου στόχου μπορεί να είναι συνδεδεμένος με κανένα, ένα ή περισσότερα annotations, για την αποθήκευση επιπρόσθετων πληροφοριών που σχετίζονται με έναν κόμβο.

4.2.3 Κλάση AtomicGoal

Η κλάση AtomicGoal είναι μια υποκλάση της κλάσης GoalNode επομένως κληρονομεί τις ιδιότητες της κλάσης GoalNode. Η κλάση AtomicGoal αναπαριστά τους κόμβους που έχουν πατέρα κόμβο, αλλά δεν έχουν κόμβους παιδιά. Αποτελούν δηλαδή τα "φύλλα" του δέντρου. Αυτό στην ουσία σημαίνει πως αποτελούν τις αρχικές αιτίες αποτυχίας μίας απαιτήσεως του υπό έλεγχο συστήματος, που δεν μπορούν να αναλυθούν σε περαιτέρω υπό-απαιτήσεις.

4.2.4 Κλάση DecompositionGoal

Η κλάση DecompositionGoal είναι μια υποκλάση της κλάσης GoalNode επομένως κληρονομεί τις ιδιότητες της κλάσης GoalNode. Αποτελείται από δύο υποκλάσεις , τις AndDecompositionGoal και OrDecompositionGoal . Δύο κόμβοι AndDecompositionGoal αποτελούν δύο κόμβους παιδιά που συνδέονται με την συζευκτική ιδιότητα (ΚΑΙ), ενώ δύο κόμβοι OrDecompositionGoal αποτελούν δύο κόμβους παιδιά που συνδέονται με την διαζευκτική ιδιότητα (Ή). Επομένως , ένας κόμβος DecompositionGoal αποτελείται από δύο παιδιά που μεταξύ τους έχουν μία από τις παραπάνω ιδιότητες. Η συσχέτιση της κλάσης DecompositionGoal με τη κλάση GoalNode, γίνεται μέσω της ιδιότητας hasChildren με πολλαπλότητα 2..* , δηλώνοντας πως για τον ορισμό μίας σχέσης Decomposition, χρειάζονται δύο κόμβοι παιδιά.

4.2.5 Κλάση Contribution

Η κλάση Contribution συμβολίζει τις συνεισφορές στις οποίες συνεισφέρει ένας κόμβος, αλλά και τις συνεισφορές που συνεισφέρουν στον κόμβο αυτόν.
Η σχέση ενός κόμβου με τις συνεισφορές που συνεισφέρουν στον κόμβο αυτό περιγράφεται από την συσχέτιση της κλάσης GoalNode με την κλάση Contribution μέσω της ιδιότητας from με πολλαπλότητα 1. Η σχέση ενός κόμβου με τις συνεισφορές στις οποίες συνεισφέρει ο κόμβος αυτός περιγράφεται από την συσχέτιση της κλάσης GoalNode με την κλάση Contribution μέσω της ιδιότητας to με πολλαπλότητα 1.

4.2.6 Κλάση GoalNodeTester

Η κλάση GoalNodeTester χρησιμοποιείται για την επαλήθευση ενός κόμβου. Χρησιμοποιεί την κλάση TestStrategy, με την οποία συσχετίζεται μέσω της ιδιότητας hasTestStrategy πολλαπλότητας 1..*. Η κλάση GoalNodeTester ισοδυναμεί με το σύνολο των στρατηγικών που επαληθεύει την ισχύ ενός κόμβου-στόχου. Η κλάση GoalNodeTester έχει την ιδιότητα name τύπου String η οποία αποτελεί το πρωτεύον κλειδί της κλάσης.

4.2.7 Κλάση TestStrategy

Η κλάση TestStrategy αποτελεί υποκλάση της GoalNodeTester, οπότε κληρονομεί όλες τις ιδιότητές της. Η κλάση TestStrategy έχει τις υποκλάσεις TestStrategyImpl, TestStrategtImplB , TestStrategtImplC κ.λ.π οι οποίες κληρονομούν τις ιδιότητες της κλάσης TestStrategy. Οι υποκλάσεις TestStrategyImpl ισοδυναμούν με τις στρατηγικές που το σύνολό τους επαληθεύει την ισχύ ενός κόμβου-στόχου. Η κλάση TestStrategy έχει την μέθοδο DO(), η οποία είναι η βασική μέθοδος εκτέλεσης των αλγορίθμων ελέγχου της ισχύος ενός κόμβου στόχου.

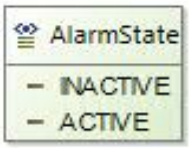
4.2.8 Κλάση Alarm

Η κλάση Alarm αποτελεί μία από τις σημαντικότερες κλάσεις του περιβάλλοντος πλαισίου, καθώς είναι υπεύθυνη για την πυροδότηση της έναρξης ελέγχου μίας απαίτησης. Ένα alarm μπορεί να σχετίζεται με κανέναν, έναν ή περισσότερους κόμβους ρίζες , δηλαδή με κανένα, ένα ή περισσότερα δέντρα στόχων. Αντιπροσωπεύει το σύνολο δεδομένων , ως ένα σύνολο μορφημάτων (pattern) , που είναι απαραίτητα για την ενεργοποίηση ενός alarm συσχετιζόμενου με κάποια απαίτηση του υπό έλεγχο συστήματος. Η ενεργοποίηση ενός alarm συνεπάγεται την αποτυχία μίας απαίτησης του υπό έλεγχο συστήματος, η οποία εκκινεί την διαδικασία ανάλυσης των αρχικών αιτιών. Η εύρεση ενός τέτοιου μορφήματος, επιτυγχάνεται μέσω ανάλυσης δεδομένων παρακολούθησης του υπό έλεγχο συστήματος. Με την ανακάλυψη κάποιου pattern που αντιστοιχεί σε ένα alarm, το alarm αυτό ενεργοποιείται και εκκινεί την διαδικασία ανάλυσης αρχικών αιτιών ξεκινώντας από τον κόμβο- ρίζα στον οποίο αντιστοιχίζεται.
Τα στιγμιότυπα της κλάσης Alarm περιέχονται σε ένα στιγμιότυπο της κλάσης AlarmContainer και ως εκ τούτου, η κλάση αυτή έχει μια επιπλέον ιδιότητα containsAlarms πολλαπλότητας 0..*, καθώς ο AlarmContainer μπορεί να περιέχει κανένα, ένα ή περισσότερα Alarms.
Όπως αναφέραμε, ένα alarm μπορεί να σχετίζεται με κανέναν, έναν ή περισσότερους κόμβους-ρίζες και αυτό ορίζεται με την συσχέτιση της κλάσης Alarm με την κλάση GoalNode μέσω της ιδιότητας associatedGoals πολλαπλότητας 0..* .
Η κλάση Alarm έχει τις ιδιότητες name τύπου String , priority τύπου Short Integer και alarmState τύπου AlarmState , η οποία περιγράφεται παρακάτω και αποτελεί την ένδειξη για τον αν ένα alarm είναι ενεργοποιημένο ή όχι. Η ιδιότητα name αποτελεί το πρωτεύον κλειδί της κλάσης.

4.2.9 Κλάση Annotation

Η κλάση Annotation αποτελεί ένα μέσο επισύναψης πληροφοριών διαφόρων τύπων σε κάθε κόμβο. Για παράδειγμα μπορεί να επισυνάπτει πληροφορία που διευκολύνει την επαλήθευση ενός κόμβου, ή την διαδικασία της ανάλυσης αρχικών αιτιών. Εδώ ως παράδειγμα αναφέρουμε την υποκλάση VerifiableAnnotation η οποία μπορεί να ορίσει δεδομένα ώστε να καταστήσουν τον στόχο αληθή ή ψευδή. Στα πλαίσια αυτής της διπλωματικής εργασίας, δε θα βασιστούμε σε πληροφορίες Annotation για την επαλήθευση των κόμβων των δέντρων στόχων.

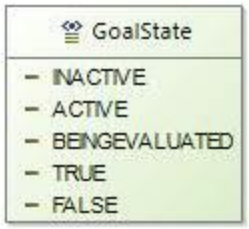
4.2.10 Κλάση AlarmState



Σχήμα 4.2: Η κλάση Alarmstate

Η κλάση Alarmstate ορίζει τις καταστάσεις στις οποίες μπορεί να βρεθεί alarm. Ορίζεται ως μία απαρίθμηση με δυνατές τιμές INACTIVE και ACTIVE. Η κατάσταση Alarmstate.ACTIVE αντιστοιχεί στην περίπτωση που έχει ενεργοποιηθεί κάποιο alarm, ενώ η κατάσταση Alarmstate.INACTIVE στην περίπτωση που δεν έχει ενεργοποιηθεί κάποιο alarm ακόμη. Επομένως, πριν εκτελεστεί το περιβάλλον πλαίσιο μας , όλα τα συνδεδεμένα στους κόμβους alarms έχουν AlarmState με τιμή INACTIVE. Με την αποτυχία κάποιας απαίτησης, το περιβάλλον πλαίσιο ειδοποιείται για την ενεργοποίηση του alarm που σχετίζεται με αυτή, και το ενεργοποιεί αλλάζοντας το AlarmState του σε ACTIVE.

4.2.11 Κλάση GoalState

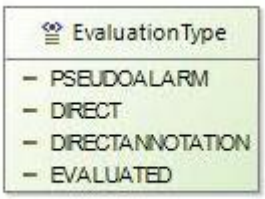


Σχήμα 4.3: Η κλάση GoalState

Η κλάση GoalState εκφράζει τις καταστάσεις στις οποίες μπορεί να βρεθεί ένας κόμβος του

δέντρου στόχων. Αποτελεί μία απαρίθμηση με δυνατές τιμές TRUE, FALSE,BEINGEVALUATED,INACTIVE και ACTIVE. Η κατάσταση GoalState.TRUE υποδηλώνει πως ένα κόμβος-στόχος έχει ήδη επαληθευτεί ως αληθής, η κατάσταση GoalState.FALSE δηλώνει πως ένα κόμβος-στόχος έχει ήδη επαληθευτεί ως ψευδής , η κατάσταση GoalState.BEINGEVALUATED την περίπτωση που ένα κόμβος-στόχος βρίσκεται υπό την διαδικασία της επαλήθευσής του, η κατάσταση GoalState.INACTIVE την περίπτωση που ένα κόμβος-στόχος είναι ανενεργός, οπότε δεν μπορούμε να προχωρήσουμε στην επαλήθευσή του, και τέλος η κατάσταση GoalState.ACTIVE την περίπτωση που ένα κόμβος-στόχος είναι ενεργός.

4.2.12 Κλάση EvaluationType



Σχήμα 4.4: Η κλάση EvaluationType

Η κλάση EvaluationType ορίζει τις καταστάσεις στις οποίες μπορεί να βρεθεί ένας κόμβος, σχετικά με τον τρόπο επαλήθευσής του. Ορίζεται ως μία απαρίθμηση με δυνατές τιμές PSEUDOALARM,DIRECT,DIRECTANNOTATION και EVALUATED. Η κατάσταση EvaluationType.PSEUDOALARM αντιστοιχεί στην περίπτωση που ο κόμβος δεν μπορεί να αποδειχτεί μόνος του αλλά μόνο με την απόδειξη των κόμβων παιδιών του. Η κατάσταση EvaluationType.DIRECT αντιστοιχεί στην περίπτωση που ο κόμβος μπορεί να αποδειχτεί άμεσα μέσω της κλάσης GoalState , η κατάσταση EvaluationType.DIRECTANNOTATION στην περίπτωση που ο κόμβος απαιτεί κάποια στρατηγική να εκτελεστεί ώστε να μπορέσουμε να αποφανθούμε την ισχύ του, και τέλος η κατάσταση EvaluationType.EVALUATED στην περίπτωση που ο κόμβος έχει ήδη αποδειχτεί. Στην τελευταία περίπτωση, την αληθοτιμή του δηλώνει η κλάση GoalState.

Κεφάλαιο 5: Συλλογιστική βασισμένη στα δέντρα στόχων

Στο κεφάλαιο αυτό θα ασχοληθούμε με το πρόβλημα έλεγχου ικανοποιησιμότητας λογικών εκφράσεων με τη χρήση δέντρων στόχων. Αρχικά περιγράφονται οι αρχές που χρησιμοποιούνται στο χώρο της προτασιακής λογικής ενώ στη συνέχεια περιγράφεται ο τρόπος μετατροπής ενός δέντρου στόχων σε λογικές εκφράσεις σε κανονική συζευκτική μορφή . Επιπλέον παρουσιάζεται ο τρόπος επίλυσης του προβλήματος που προκύπτει με τη χρήση της βιβλιοθήκης SAT4j. Τέλος , παρουσιάζεται ο τρόπος επίλυσης του προβλήματος ικανοποίησης περιορισμών όπως αυτός εφαρμόστηκε στο περιβάλλον πλαίσιο της παρούσας διπλωματικής εργασίας.

5.1 Μετασχηματισμός Δέντρων Στόχων σε Κανονική Συζευκτική Μορφή

Τα τελευταία χρόνια εκφράζεται ολοένα και περισσότερο ενδιαφέρον στο πεδίο του ελέγχου ικονοποιησιμότητας λογικών εκφράσεων. Αυτό συμβαίνει διότι πολλά από τα προβλήματα που καλούνται να αντιμετωπίσουν οι σύγχρονοι μηχανικοί μπορούν να αναλυθούν σε προβλήματα λογικών εκφράσεων. Ένα από τα σημαντικότερα προβλήματα που καλείται να αντιμετωπίσει κανείς κατά τη διάρκεια ανάπτυξης και συντήρησης λογισμικού είναι η διατήρηση της ποιότητας του προϊόντος λογισμικού καθώς και η συνέπεια και ικανοποίηση των αρχικών προδιαγραφών. Για την επίλυση του προβλήματος αυτού, η κοινότητα τεχνολογίας λογισμικού οδηγήθηκε στη χρήση μοντέλων-δέντρων στόχων τα οποία αναπαριστούν λειτουργικές και μη λειτουργικές ιδιότητες ενός λογισμικού και περιέχουν καθήκοντα , δράσεις και απόψεις ενδιαφερόμενων μερών. Τα μοντέλα αυτά μπορούν να αναλυθούν σε λογικές εκφράσεις και να επιλυθούν με τη χρήση κάποιας προσέγγισης επίλυσης προβλήματος περιορισμών. Επιπλέον κάθε πρόβλημα ικανοποίησης περιορισμών μπορεί να εκφραστεί διαμέσω μιας έκφρασης ικανοποιησιμότητας σε Κανονική Συζευκτική μορφή (Conjuctive Normal Form -CNF) όπου κάθε διαζευκτική πρόταση (clause) εκφράζει έναν περιορισμό. Στη συνέχεια της ενότητας θα παρουσιαστεί ο τρόπος μετασχηματισμού δέντρων στόχων σε λογικές εκφράσεις σε κανονική συζευκτική μορφή. Για τη μετατροπή σε κανονική συζευκτική μορφή χρησιμοποιείται η προτασιακή λογική.

Η προτασιακή λογική εκφράζεται μέσω ιδιοτήτων, οι οποίες καλούνται προτασιακοί τύποι. Μέσω αυτών των τύπων δημιουργείται ένα μαθηματικό ανάλογο των προτάσεων (ισχυρισμός , δήλωση ή πρόταση) που στην προκειμένη περίπτωση μπορεί να είναι αληθής ή ψευδής. Τα σύμβολα όπως $\leftrightarrow$, $\neg$, $\wedge$, $\vee$, $\rightarrow$ χρησιμοποιούνται για το σχηματισμό προτασιακών τύπων και αποκαλούνται συνδετικά . Οι ονομασίες και η σειρά προτεραιότητας εκτέλεσης των τελεστών φαίνεται στο σχήμα (σχήμα 5.1) .

Συνδετικό	Ονομασία (Ελλ)	Ονομασία (Αγγλ)	Προτεραιότητα
$\top$	Αλήθεια	Verum	
$\bot$	Ψεύδος	Falsum	
$\neg$	Άρνηση	Negation	1
$\wedge$	Σύζευξη	Conjunction	2
$\vee$	Διάζευξη	Disjunction	2
$\rightarrow$	Συνεπαγωγή	Implication	3
$\leftrightarrow$	Ισοδυναμία	Equivalence	4

Σχήμα 5.1: Χρησιμοποιούμενοι τελεστές για τη διαμόρφωση προτασιακών τύπων.

5.1.1 Μετασχηματισμός Αποσυνθέσεων και Συνεισφορών

Η αντιμετώπιση προβλημάτων SAT στα οποία η είσοδος δίνεται σε αυθαίρετη μορφή (χρησιμοποιώντας δηλαδή οποιονδήποτε από τους τελεστές που παρουσιάζονται στο σχήμα 5.1) μπορεί να είναι μια εξαιρετικά επίπονη διαδικασία. Στην πλειοψηφία των συστημάτων αυτόματου συμπερασμού (automated reasoning systems) προτιμάται η χρήση λογικών εκφράσεων που αποτελούνται από σύνολα διαζευκτικών-συζευκτικών προτάσεων κυρίως επειδή:

- Το πρόβλημα της ικανοποιησιμότητας λογικών εκφράσεων για αυθαίρετους προτασιακούς τύπους μπορεί τελικώς να αναχθεί σε πρόβλημα ικανοποιησιμότητας τύπων που αποτελούνται από διαζευκτικές-συζευτικές προτάσεις.
- Συγκριτικά με τους αυθαίρετους προτασιακούς τύπους, οι διαζευκτικές-συζευτικές προτάσεις είναι πολύ πιο απλές και πολύ πιο εύκολες στην υλοποίηση.

Για το λόγο αυτό είναι σημαντική η ανάλυση του προβλήματος που απαιτεί λύση σε κανονική συζευκτική μορφή (CNF) . Στην παρούσα διπλωματική εργασία το πρόβλημα που πρέπει να επιλυθεί είναι το μοντέλο δέντρων στόχων το οποίο περιέχει τις λειτουργικές πληροφορίες - απαιτήσεις του υπό έλεγχο συστήματος καθώς και των AND/OR αλληλεπιδράσεων μεταξύ αυτών. Το δέντρο -στόχων αυτό θα πρέπει να μετατραπεί σε κανονική συζευκτική μορφή με τέτοιο τρόπο ώστε η έκφραση προτασιακής λογικής που θα προκύψει να εκφράζει όλους τους κανόνες που μοντελοποιούν το μοντέλο δέντρων αλλά και τις σχέσεις μεταξύ των στόχων συμπεριλαμβανομένων AND/OR και των συνεισφορών .

Τα δέντρα στόχων AND/OR αποτελούν μια τεχνική μοντελοποίησης που χρησιμοποιείται εκτεταμένα για την μοντελοποίηση των απαιτήσεων ενός συστήματος (requirements) . Η βασική ιδέα αυτής της μοντελοποίησης είναι ο διαχωρισμός των στόχων-απαιτήσεων σε υπό-απαιτήσεις Κάθε υπό-απαίτηση αποσυντίθεται σε άλλες υπό-απαιτήσεις οι οποίες συσχετίζονται με την έννοια τις σύζευξης (AND) ή της διάζευξης (OR). Δανειζόμενοι την σημειογραφία που χρησιμοποιείται στο [57] συμβολίζουμε μία αποσύνθεση AND/OR του στόχου G σε ένα σύνολο των επιμέρους στόχων ως εξής

G $\xrightarrow{AND}$ α και **G** $\xrightarrow{OR}$ α αντίστοιχα

Οι αποσυνθέσεις τους λοιπόν έχουν ως εξής:

• Αποσύνθεση AND

$$\mathbf{G} \xrightarrow{AND} \alpha \Rightarrow g_1 \wedge g_2 \wedge \ldots \wedge g_n \leftrightarrow a \Rightarrow$$

$$\Rightarrow (g_1 \wedge g_2 \wedge \ldots \wedge g_n \rightarrow a) \wedge (a \rightarrow g_1 \wedge g_2 \wedge \ldots \wedge g_n) \Rightarrow$$
$$\Rightarrow (\neg (g_1 \wedge g_2 \wedge \ldots \wedge g_n) \vee a) \wedge (\neg a \vee (g_1 \wedge g_2 \wedge \ldots \wedge g_n)) \Rightarrow$$
$$\Rightarrow (\neg g_1 \vee \neg g_2 \vee \ldots \vee \neg g_n \vee a) \wedge (\neg a \vee g_1) \wedge (\neg a \vee g_2) \wedge \ldots \wedge (\neg a \vee g_n) \qquad (1)$$

• Αποσύνθεση OR

$$\mathbf{G} \xrightarrow{OR} \alpha \Rightarrow g1 \vee g2 \vee \ldots \vee gn \leftrightarrow a \Rightarrow$$

$$\Rightarrow (g_1 \vee g_2 \vee \ldots \vee g_n \rightarrow a) \wedge (a \rightarrow g_1 \vee g_2 \vee \ldots \vee g_n) \Rightarrow$$
$$\Rightarrow (\neg (g_1 \vee g_2 \vee \ldots \vee g_n) \vee a) \wedge (\neg a \vee (g_1 \vee g_2 \vee \ldots \vee g_n)) \Rightarrow$$
$$\Rightarrow ((\neg g_1 \wedge \neg g_2 \wedge \ldots \wedge \neg g_n) \vee a) \wedge (\neg a \vee g_1 \vee g_2 \vee \ldots \vee g_n) \Rightarrow$$
$$\Rightarrow ((\neg g_1 \vee a) \wedge (\neg g_2 \vee a) \wedge \ldots \wedge (\neg g_n \vee a)) \wedge (\neg a \vee g_1 \vee g_2 \vee \ldots \vee g_n) \Rightarrow$$
$$\Rightarrow (\neg g_1 \vee a) \wedge (\neg g_2 \vee a) \wedge \ldots \wedge (\neg g_n \vee a) \wedge (\neg a \vee g_1 \vee g_2 \vee \ldots \vee g_n) \qquad (2)$$

Τέλος σύμφωνα με το [58] θεωρούμε τέσσερα ήδη συνεισφορών . Τις συνεισφορές τύπου ++S(g,g') και --S(g,g') , που σημαίνει ότι ο κόμβος στόχος g' ικανοποιείται (αντίστοιχα δεν ικανοποιείται) όταν ο κόμβος πηγή g ικανοποιείται και τις συνεισφορές ++D(g,g') και --D(g,g') , που σημαίνουν ότι η μη ικανοποίηση του κόμβου πηγή g οδηγεί στην μη ικανοποίηση (αντίστοιχα ικανοποίηση) του κόμβου στόχου g'. Η συμβολική αναπαράσταση των τεσσάρων αυτών συνεισφορών φαίνεται παρακάτω:

++S (g, g') είναι ανάλογο προς $g \rightarrow g'$

−−S (g, g') είναι ανάλογο προς $g \rightarrow \neg g'$

++D (g, g') είναι ανάλογο προς $\neg g \rightarrow \neg g'$

−−D (g,g') είναι ανάλογο προς $\neg g \rightarrow g'$

• Μετασχηματισμός Συνεισφορών

$$g_1 \xrightarrow{++S} g_2 \Rightarrow g_1 \rightarrow g_2 \Rightarrow \neg g_1 \vee g_2 \qquad (3)$$

$$g_1 \xrightarrow{--S} g_2 \Rightarrow g_1 \rightarrow \neg g_2 \Rightarrow \neg g_1 \vee \neg g_2 \qquad (4)$$

$$g_1 \xrightarrow{++D} g_2 \Rightarrow \neg g_1 \rightarrow \neg g_2 \Rightarrow g_1 \vee \neg g_2 \qquad (5)$$

$$g_1 \xrightarrow{--D} g_2 \Rightarrow \neg g_1 \rightarrow g_2 \Rightarrow g_1 \vee g_2 \qquad (6)$$

Όπως ήδη αναφέρθηκε νωρίτερα, κάθε προτασιακός τύπος μπορεί να μετασχηματιστεί σε μια αντίστοιχη έκφραση σε CNF μορφή. Προκειμένου αυτό να καταστεί δυνατό εφαρμόζονται αναληπτικά οι παραπάνω κανόνες μετασχηματισμού των συνεισφορών και των αποσυνθέσεων

AND/OR , αλλά και οι γενικοί κανόνες μετασχηματισμού οι οποίοι φαίνονται παρακάτω:

$A \leftrightarrow B \quad = \quad (\neg A \vee B) \wedge (\neg B \vee A) \qquad (7)$

$A \rightarrow B \quad = \quad \neg A \vee B \qquad (8)$

$\neg (A \vee B) \quad = \quad \neg A \wedge \neg B \qquad (9)$

$\neg (A \wedge B) \quad = \quad \neg A \vee \neg B \qquad (10)$

$\neg A \quad = \quad A \qquad (11)$

$(A \wedge B) \vee C \quad = \quad (A \vee C) \wedge (B \vee C) \qquad (12)$

5.1.2 Δημιουργία τελικής έκφρασης CNF

Στην ενότητα αυτή θα περιγράψουμε τη διαδικασία μετατροπής ενός μοντέλου δέντρου στόχων σε κανονική διαζευκτική μορφή η οποία θα εμπεριέχει όλη την πληροφορία και τη συσχέτιση των κόμβων του δέντρου στόχων έτσι ώστε το πρόβλημα ικανοποίησης του αρχικού κόμβου (αρχικού στόχου του δέντρου) μπορεί να περιοριστεί στην εύρεση μιας λύσης SAT και στην προσπάθεια ικανοποίησης της.

Από τη στιγμή που ένα δέντρο στόχων έχει πλήρως οριστεί , μπορούμε να εξάγουμε από αυτό ένα σύνολο λογικών κανόνων οι οποίοι συμπεριλαμβάνουν όλους τους περιορισμούς που έχουν τεθεί στο δέντρο. Για τη σημασιολογία των κανόνων αυτών αλλά και τις εκφράσεις CNF μπορεί κανείς να ανατρέξει στο [59]. Συνοπτικά παρουσιάζονται στο παρακάτω σχήμα :

$N^{pos}(p)$	$N^{neg}(p)$	$N^{lp}(p)$	Generated AND/OR Rules $p \in N_t$	$p \in N_a$
$= \emptyset$	$= \emptyset$	$= \emptyset$	$p \leftarrow \mathrm{T}(c_1, c_2, \cdots, c_n)$	-
$= \emptyset$	$= \emptyset$	$\{b_1 \cdots b_q\}$	$p \leftarrow \mathrm{AND}(b_1 \cdots b_q, \mathsf{p\_d})$	$p \leftarrow \mathrm{AND}(b_1 \cdots b_q, \mathsf{p\_leaf})$
$= \emptyset$	$\neq \emptyset$	$= \emptyset$	$p \leftarrow \mathrm{AND}(\mathsf{p\_d}, \neg\mathsf{p\_c\_neg})$	-
$= \emptyset$	$\neq \emptyset$	$\{b_1 \cdots b_q\}$	$p \leftarrow \mathrm{AND}(\mathsf{p\_}\ell, \neg\mathsf{p\_c\_neg})$ $\mathsf{p\_}\ell \leftarrow \mathrm{AND}(b_1 \cdots b_q, \mathsf{p\_d})$	-
$\neq \emptyset$	$= \emptyset$	$= \emptyset$	$p \leftarrow \mathrm{OR}(\mathsf{p\_d}, \mathsf{p\_c\_pos})$	-
$\neq \emptyset$	$= \emptyset$	$\{b_1 \cdots b_q\}$	$p \leftarrow \mathrm{OR}(\mathsf{p\_}\ell, \mathsf{p\_c\_pos})$ $\mathsf{p\_}\ell \leftarrow \mathrm{AND}(b_1 \cdots b_q, \mathsf{p\_d})$	-
$\neq \emptyset$	$\neq \emptyset$	$= \emptyset$	$p \leftarrow \mathrm{OR}(\mathsf{p\_d}, \mathsf{p\_c})$ $p \leftarrow \mathrm{AND}(\mathsf{p\_c\_pos}, \neg\mathsf{p\_c\_neg})$	-
$\neq \emptyset$	$\neq \emptyset$	$\{b_1 \cdots b_q\}$	$p \leftarrow \mathrm{OR}(\mathsf{p\_}\ell, \mathsf{p\_c})$ $\mathsf{p\_c} \leftarrow \mathrm{AND}(\mathsf{p\_c\_pos}, \neg\mathsf{p\_c\_neg})$ $\mathsf{p\_}\ell \leftarrow \mathrm{AND}(b_1 \cdots b_q, \mathsf{p\_d})$	-

Σχήμα 5.2 : Κανόνες AND/OR για τη περιγραφή των δραστηριοτήτων των κόμβων.

Boolean Rule	Equivalent Constraints	CNF Clauses
$o \leftarrow \mathrm{AND}(i_1, i_2, \cdots, i_n)$	$\neg i_1 \Rightarrow \neg o, \cdots, \neg i_n \Rightarrow \neg o,$ $i_1 \wedge i_2 \wedge \cdots \wedge i_n \Rightarrow o$	$(i_1 \vee \neg o) \wedge \cdots \wedge (i_n \vee \neg o) \wedge$ $(\neg i_1 \vee \neg i_2 \vee \cdots \vee \neg i_n \vee o)$
$o \leftarrow \mathrm{OR}(i_1, i_2, \cdots, i_n)$	$i_1 \Rightarrow o, \cdots, i_n \Rightarrow o,$ $i_1 \wedge i_2 \wedge \cdots \wedge i_n \Rightarrow o$	$(\neg i_1 \vee o) \wedge \cdots \wedge (\neg i_n \vee o) \wedge$ $(i_1 \vee i_2 \vee \cdots \vee i_n \vee \neg o)$

Σχήμα 5.3: αντιστοίχηση λογικών εκφράσεων σε μορφή CNF

Η δημιουργία των κανόνων προκύπτει από την προσπέλαση κάθε κόμβου p ,που ανήκει στο δέντρο N_t ($p \in N_t$). Για κάθε τέτοιο κόμβο εξάγουμε το σύνολο $N^{lp}(p)$, καθώς και τα δύο παρακάτω σύνολα:

$N^{pos}(p) = N^{++S}(p) \cup N^{--D}(p) = \{e_1, \ldots, e_k\} \cup \{f_1, \ldots, f_l\}$ (13)

$N^{pos}(p) = N^{--S}(p) \cup N^{++D}(p) = \{g_1, \ldots, g_k\} \cup \{h_1, \ldots, h_o\}$ (14)

τα οποία σε συνδυασμό με τον κανόνα αποσύνθεσης $r_d = < T, p \{ c_1, c_2, \ldots, c_n \}$ που αντιστοιχεί στον κόμβο αυτό , καθορίζουν το σύνολο των λογικών κανόνων που θα δημιουργηθούν. Και ανάλογα με το αν αυτά τα σύνολα είναι ορισμένα ή όχι δημιουργούνται διαφορετικά σύνολα από κανόνες , όπως φαίνεται στο προηγούμενο σχήμα.

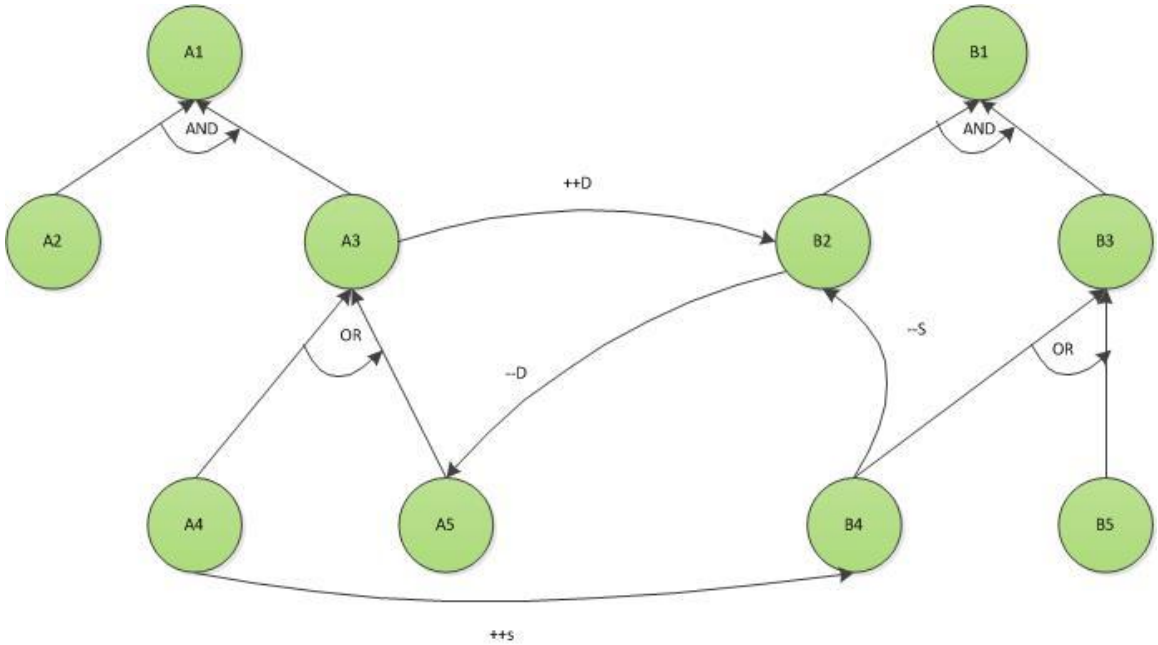
Ας υποθέσουμε πως έχουμε την ενδιάμεση έκφραση CNF, $f=(p_1 \vee p_2 \vee ... \vee p_n) \wedge (p_1^{(2)} \vee p_2^{(2)} \vee ... \vee p_n^{(2)}) \wedge ... \wedge (p_1^{(m)} \vee p_2^{(m)} \vee ... \vee p_n^{(m)})$) που έχει προκύψει από το μετασχηματισμό κάθε δέντρου σε CNF , όπου $p_1, . . . , p_n , p_1^{(1)}, . . . , p_n^{(1)}, p_1^{(m}, . . . , p_n^{(m)}$, ο στόχος ή η άρνηση ενός δέντρου και ++S , μια συνεισφορά από τον κόμβο g_1 στον κόμβο g_2.

Η έκφραση που προκύπτει από την λογική διάζευξη και συνεισφορά είναι η εξής:

$f \vee (\neg g_1 \vee g_2) \Rightarrow$
$\Rightarrow ((p_1 \vee p_2 \vee ... \vee p_n) \wedge (p_1^{(2)} \vee p_2^{(2)} \vee ... \vee p_n^{(2)}) \wedge ... \wedge (p_1^{(m)} \vee p_2^{(m)} \vee ... \vee p_n^{(m)})) \vee (\neg g_1 \vee g_2) \Rightarrow$
(επιμεριστική ιδιότητα)
$\Rightarrow ((p_1 \vee p_2 \vee ... \vee p_n) \vee (\neg g_1 \vee g2)) \wedge ((p_1(2) \vee p_2(2) \vee ... \vee p_n(2)) \vee (\neg g_1 \vee g_2)) \wedge ...$
$... \wedge ((p_1^{(m)} \vee p_2^{(m)} \vee ... \vee p_n^{(m)}) \vee (\neg g_1 \vee g_2))$

Η παραπάνω έκφραση είναι σε CNF μορφή. Για να κατανοήσουμε πλήρως την διαδικασία μετασχηματισμού ενός δέντρου στόχων σε μορφή CNF δίνουμε το παρακάτω παράδειγμα.

Έστω λοιπόν πως έχουμε το μοντέλο στόχων που ακολουθεί:



Σχήμα 5.4: Παράδειγμα Μοντέλου δέντρων στόχων

$((A_2 \wedge A_3 \leftrightarrow A_1) \wedge (A_4 \vee A_5 \leftrightarrow A_2) \wedge (B_2 \wedge B_3 \leftrightarrow B_1) \wedge (B_5 \vee B_6 \leftrightarrow B_3)) \vee ...$
$...\vee (A_4 \xrightarrow{++S} B_4) \vee (B_2 \xrightarrow{--D} A_5) \vee (A_3 \xrightarrow{++D} B_2) \vee (B_4 \xrightarrow{--S} S\ B_2) \Rightarrow$

$\Rightarrow ((\neg A_2 \vee \neg A_3 \vee A_1) \wedge (A_2 \vee \neg A_1) \wedge (A_3 \vee \neg A_1) \wedge ...$
$...\wedge (A_4 \vee A_5 \vee \neg A_2) \wedge (\neg A_4 \vee A_2) \wedge (\neg A_4 \vee A_2) \wedge ...$
$...\wedge (\neg B_2 \vee \neg B_3 \vee B_1) \wedge (B_2 \vee \neg B_1) \wedge (B_3 \vee \neg B_1) \wedge ...$
$...\wedge (B_5 \vee B_4 \vee \neg B_3) \wedge (\neg B_5 \vee B_3) \wedge (\neg B_4 \vee B_3)) \vee ...$
$...\vee ((\neg A_4 \vee B_4) \vee (B_2 \vee A_5) \vee (A_3 \vee \neg B_2) \vee (\neg B_4 \vee B_2)) \Rightarrow$

$\Rightarrow(\neg A_2 \vee \neg A_3 \vee A_1 \vee \neg A_3 \vee B_2 \vee B_5 \vee A_2 \vee A_5 \vee \neg B_5 \vee B_4) \wedge ...$
$... \wedge (A_2 \vee \neg A_1 \vee \neg A_3 \vee B_2 \vee B_5 \vee A_2 \vee A_5 \vee \neg B_5 \vee B_4) \wedge ...$
$...\wedge (A_3 \vee \neg A_1 \vee \neg A_3 \vee B_2 \vee B_5 \vee A_2 \vee A_5 \vee \neg B_5 \vee B_4) \wedge ...$
$...\wedge (A_4 \vee A_5 \vee \neg A_2 \vee \neg A_3 \vee B_2 \vee B_5 \vee A_2 \vee A_5 \vee \neg B_5 \vee B_4) \wedge ...$
$...\wedge (\neg A_4 \vee A_2 \vee \neg A_3 \vee B_2 \vee B_5 \vee A_2 \vee A_5 \vee \neg B_5 \vee B_4) \wedge ...$
$...\wedge (\neg A_4 \vee A_2 \vee \neg A_3 \vee B_2 \vee B_5 \vee A_2 \vee A_5 \vee \neg B_5 \vee B_4) \wedge ...$
$...\wedge (\neg B_2 \vee \neg B_3 \vee B_4 \vee B_1 \vee \neg A_3 \vee B_2 \vee B_5 \vee A_2 \vee A_5 \vee \neg B_5 \vee B_4) \wedge ...$
$...\wedge (B_2 \vee \neg B_1 \vee \neg A_3 \vee B_2 \vee B_5 \vee A_2 \vee A_5 \vee \neg B_5 \vee B_4) \wedge ...$
$...\wedge (B_3 \vee \neg B_1 \vee \neg A_3 \vee B_2 \vee B_5 \vee A_2 \vee A_5 \vee \neg B_5 \vee B_4) \wedge ...$
$...\wedge (B_4 \vee \neg B_1 \vee \neg A_3 \vee B_2 \vee B_5 \vee A_2 \vee A_5 \vee \neg B_5 \vee B_4) \wedge ...$
$...\wedge (B_5 \vee B_4 \vee \neg B_3 \vee \neg A_3 \vee B_2 \vee B_5 \vee A_2 \vee A_5 \vee \neg B_5 \vee B_4) \wedge ...$
$...\wedge (\neg B_4 \vee B_4 \vee \neg A_4 \vee B_2 \vee B_2 \vee A_2 \vee A_3 \vee \neg B_3 \vee \neg B_4 \vee B_2) \wedge ...$
$...\wedge (\neg B_4 \vee B_4 \vee \neg A_4 \vee B_2 \vee B_2 \vee A_5 \vee A_3 \vee \neg B_3 \vee \neg B_4 \vee B_2)$

5.2 Αναγωγή στο πρόβλημα της ικανοποιησιμότητας

Στο κεφάλαιο αυτό αναλύουμε πώς με τη χρήση του SAT4j επιλύουμε το πρόβλημα ικανοποιησιμότητας που προκύπτει από το δέντρο στόχων. Πιο συγκεκριμένα περιγράφουμε την τεχνική δημιουργίας της CNF έκφρασης, τον τρόπο επιλογής των λύσεων που προκύπτουν από την επίλυση του προβλήματος μέσω του SAT4j και τον τρόπο επαλήθευσης των λύσεων αυτών.

5.2.1 Ενέργειες πριν την χρήση του SAT4j

Όπως αναφέραμε και στις προηγούμενες ενότητες το SAT4j, το οποίο αποτελεί μια βιβλιοθήκη της γλώσσας Java, είναι εκείνο που θα επιχειρήσει την επίλυση του προβλήματος ικανοποιησιμότητας . Το πρόβλημα αυτό θα πρέπει να περιγραφτεί σε μορφή τέτοια ώστε η επεξεργασία των δεδομένων του προβλήματος και η επίλυση του να είναι εφικτή μέσω του SAT4j. Η έκδοση του SAT4j που χρησιμοποιήθηκε στην παρούσα διπλωματική εργασία δέχεται ως είσοδο μια έκφραση σε κανονική διαζευκτική μορφή CNF και επιστρέφει ως λύση το σύνολο των συνδυασμών που απαιτείται να ικανοποιηθούν προκειμένου να αποδειχθεί η ικανοποίηση ή όχι της αρχικής έκφρασης. Περιγράφουμε λοιπόν παρακάτω τον τρόπο με τον οποίο δημιουργήθηκε η CNF έκφραση στο περιβάλλον πλαίσιο που αναπτύχθηκε.

Το πρόβλημα που αντιμετωπίζουμε στην παρούσα διπλωματική εργασία είναι ο έλεγχος ικανοποίησης ή όχι ενός συνόλου απαιτήσεων . Όπως περιγράφθηκε και παραπάνω, οι απαιτήσεις αυτές μοντελοποιούνται σε δέντρα στόχων έχοντας τη μορφή που φαίνεται στο σχήμα 5.4. (Σχήμα 5.4 Παράδειγμα Μοντέλου δέντρων στόχων). Έχοντας μοντελοποιήσει τις

απαιτήσεις μας σε δέντρα στόχων , το μόνο στάδιο που μένει πριν την εφαρμογή του SAT4j για την ικανοποίηση του προβλήματος ικανοποιησιμότητας είναι η δημιουργία της έκφρασης CNF που το περιγράφει και εμπεριέχει όλες τις λειτουργίες και αλληλεξαρτήσεις των απαιτήσεων. Στην ενότητα 5.1.2 δόθηκε ένα παράδειγμα δημιουργίας της έκφρασης αυτής. Η τεχνική που χρησιμοποιήθηκε στο παράδειγμα αυτό οδηγεί στην δημιουργία της CNF έκφρασης του συνολικού μοντέλου δέντρων στόχων. Η μέθοδος αυτή δεν είναι αποτελεσματική στην περίπτωση που το μοντέλο δέντρων στόχων αποτελείται από πολλά δέντρα , ή από δέντρα που αποτελούνται από πολλούς υπο-κόμβους. Στην περίπτωση χρήσης της μεθόδου αυτής σε μεγάλα μοντέλα δέντρων, η έκφραση CNF που θα προέκυπτε θα ήταν αρκετά μεγάλη και η επίλυση του προβλήματος μέσω του SAT4j θα καθυστερούσε σημαντικά, κάνοντας το σύστημα μας μη αποδοτικό. Μη αποδοτική θα ήταν επίσης και η προσπάθεια επαλήθευσης της λύσης SAT καθώς κάθε λύση θα περιείχε όλους τους κόμβους προς επαλήθευση όλου του μοντέλου, αυξάνοντας σημαντικά των αριθμό των κόμβων που θα έπρεπε να ελεγχθούν. Προκειμένου λοιπόν να κάνουμε το σύστημα μας πιο αποδοτικό χρησιμοποιήσαμε τη μέθοδο επιλογής υποσυνόλου του μοντέλου δέντρων στόχων. Η επιλογή του υποσυνόλου του μοντέλου δέντρων στόχων που θα επιλεγεί για να σχηματίσει την έκφραση CNF που θα περιλαμβάνει την προς απόδειξη απαίτηση αποτελεί καίριας σημασίας για την επίδοση του αλγορίθμου συλλογιστικής και την επιβεβαίωση μια απαίτησης. Το υποσύνολο του μοντέλου δέντρων στόχων που επιλέγεται στο περιβάλλον πλαίσιο που αναπτύχθηκε, αποτελείται από την προς απόδειξη απαίτηση , το δέντρο στόχων δηλαδή που έχει ως ρίζα την υπόθεση μαζί με όλους του υπό-κόμβους στόχους που το αποτελούν αλλά και από το σύνολο των κόμβων στόχων που συνεισφέρουν στους κόμβους στόχους της προς απόδειξης απαίτησης. Να σημειώσουμε εδώ πως στην CNF έκφραση συμπεριλαμβάνονται μόνο οι κόμβοι στόχοι που συνεισφέρουν στην προς απόδειξη απαίτηση και όχι όλο το δέντρο στόχων στο οποίο ανήκουν . Αυτό γίνεται προκειμένου να αποφευχθεί η αύξηση της CNF έκφρασης αλλά και κύκλοι στην ανάλυσης των ίδιων κόμβων, γεγονός που θα μείωνε σημαντικά την συνολική επίδοση του συστήματος.

Για να γίνει η μέθοδος αυτή πλήρως κατανοητή θα παρουσιάσουμε ένα παράδειγμα εφαρμογής της στο μοντέλου δέντρων στόχων που παρουσιάστηκε στην ενότητα 5.1.2.
Ας υποθέσουμε λοιπόν πως θέλουμε να αποδείξουμε την υπόθεση A_1 του μοντέλου αυτού. Το σύνολο των κόμβων που θα εμπεριέχονται στην περιγραφή του δέντρου μας είναι οι στόχοι A_1 , A_2, A_3, A_4, A_5 αλλά και ο στόχος B_2 που συνεισφέρει στο επιλεγμένο δέντρο. Οι συνεισφορές ++S, ++D και --S και οι κόμβοι του δέντρου B που σχετίζονται με αυτές δεν θα χρησιμοποιηθούν. Η έκφραση περιγραφής του μοντέλου που προκύπτει με βάση αυτή τη μέθοδο είναι:

$$((A_2 \wedge A_3 \leftrightarrow A_1) \wedge (A_4 \vee A_5 \leftrightarrow A_3)) \vee (B_2 \xrightarrow{--D} A_2) \wedge A_2 \wedge A_1$$

ή σε CNF

$$(\neg A_2 \vee \neg A_3 \vee A_1 \vee B_2 \vee A_2) \wedge (A_2 \vee \neg A_1 \vee B_2 \vee A_2) \wedge (A_3 \vee \neg A_1 \vee B_2 \vee A_2) \wedge \ldots$$
$$\ldots \wedge (A_4 \vee A_5 \vee \neg A_2 \vee B_2 \vee A_2) \wedge (\neg A_4 \vee A_2 \vee B_2 \vee A_2) \wedge (\neg A_4 \vee A_2 \vee B_2 \vee A_2) \wedge \ldots$$
$$\ldots \wedge A_3 \wedge A_1$$

5.2.2 Εφαρμογή του SAT4j

Στην ενότητα αυτή θα περιγράψουμε πώς πραγματοποιείται η επίλυση του προβλήματος ικανοποίησης μέσω του SAT4j αλλά και πως επιλέγεται η λύση που θα χρησιμοποιηθεί.

Στην προηγούμενη ενότητα περιγράψαμε τον τρόπο δημιουργίας της έκφρασης CNF του περιβάλλον πλαισίου που αναπτύχθηκε και δώσαμε ένα παράδειγμα υπό την ανάγκη υπόδειξης μιας απαίτησης-υπόθεσης του μοντέλου αυτού. Στην πραγματική λειτουργία όμως του συστήματος δεν περιοριζόμαστε στην απόδειξη μιας υπόθεσης τη φορά. Το σύστημα έχει αναπτυχθεί έτσι ώστε να μπορεί να ικανοποιηθεί η ανάγκη απόδειξης πολλών υποθέσεων του συνολικού μοντέλου δέντρου στόχων ταυτόχρονα. Για να γίνει αυτό εφικτό ακολουθούνται τα παρακάτω βήματα:

- Ταξινόμηση των υποθέσεων των οποίων η ισχύς πρέπει να αποδειχθεί.
  Η λίστα των προς απόδειξη υποθέσεων ταξινομείται με βάση του πεδίου Priority της κλάσης Hypothesis. Τα περιβάλλον πλαίσιο που αναπτύχθηκε βασίστηκε στην αρχιτεκτονική σχεδίασης Strategy μέσω της οποίας γίνεται εφικτή η εφαρμογή νέων στρατηγικών ταξινόμησης των υποθέσεων χωρίς ουσιαστική επίδραση στον υπόλοιπο κώδικα του συστήματος.

- Εύρεση πλάνου επαλήθευσης των υποθέσεων - εφαρμογή του SAT4j.
  Επόμενο στάδιο μετά την ταξινόμηση των προς απόδειξη υποθέσεων είναι η εκτέλεση του SAT4j για όλες τις υποθέσεις αυτές. Ο SAT4j επιστρέφει για κάθε υπόθεση όλους τους συνδυασμούς κόμβων των οποίων η ικανοποίηση θα οδηγήσει στην επαλήθευση της αρχικής υπόθεσης. Τότε η ισχύς κάθε λύσης εξετάζεται, όπως περιγράφεται στην επόμενη ενότητα και ανάλογα με το αν προκύψει αληθής ή ψευδής το σύστημα συνεχίζει με τις επόμενες λειτουργίες ή δοκιμάζει τη επαλήθευση μιας άλλης λύσης του SATSolver. Έτσι λοιπόν, σε περίπτωση που η λύση που έχει επιστραφεί από το SATSolver δεν οδηγεί στην ικανοποίηση της υπόθεσης, επιλέγεται ο έλεγχος της επόμενης λύσης και η διαδικασία αυτή σταματά όταν η αρχική υπόθεση αποδειχτεί αληθής ή όταν καμία από τις επιλεγμένες λύσεις του SATSolver δεν οδηγήσει στην ικανοποίηση της.

- Επιλογή των λύσεων του SAT4j.
  Ο αριθμός των λύσεων που επιστρέφονται από τον SAT4j εξαρτώνται από το μέγεθος του δέντρου της υπόθεσης καθώς και των υπο-στόχων του, έτσι σε μεγάλα δέντρα ο αριθμός των λύσεων μπορεί να είναι αρκετά μεγάλος. Για το λόγο αυτό επιλέγονται πρώτα οι N πρώτες λύσεις. Στην περίπτωση που καμία από της λύσεις αυτές δεν μπορεί να οδηγήσει στην επαλήθευση της υπόθεσης η διαδικασία συνεχίζεται με τις υπόλοιπες λύσεις. Το σημαντικό σε αυτό το σημείο είναι πως κατά τη διάρκεια δοκιμής των αρχικών λύσεων του SAT , πολλοί από τους κόμβους έχουν επισημανθεί ως αληθείς ή ψευδείς. Έτσι στην περίπτωση μη εύρεσης εκείνης της λύσης που επαληθεύει την αρχική μας υπόθεση , ο SATsolver επανεκτελείται κάνοντας χρήση τις γνώσης των αληθοτιμών του κάθε κόμβου. Η γνώση για τις αληθοτιμές των κόμβων-στόχων θέτει επιπλέον περιορισμούς στο SAT4j και έτσι ελαχιστοποιούνται ακόμη περισσότερο η λύσεις που επιστρέφει.

- Επαλήθευση των κόμβων-στόχων που εμπεριέχονται στις λύσεις του SAT4j.
  Στο περιβάλλον πλαίσιο που αναπτύχθηκε, κάθε κόμβος-στόχος του μοντέλου δέντρων

στόχων μπορεί να είναι αληθής ή ψευδής ή να μην είναι γνωστή η αληθοτιμή του. Η πληροφορία για το αν ένας κόμβος είναι εκ των προτέρων αληθής ή ψευδής δίνεται από τον χρήστη του συστήματος, επισημαίνοντας στο μοντέλου δέντρων στόχων που δίνει ως είσοδο στο σύστημα, τους κόμβους - στόχους για τους οποίους γνωρίζει την αληθοτιμή τους, καθώς και την αληθοτιμή αυτών. Στην περίπτωση που η κατάσταση ενός κόμβου δεν είναι γνωστή το σύστημα ενεργοποιεί άλλες στρατηγικές ελέγχου του κόμβου . Στην παρούσα διπλωματική εργασία οι στρατηγικές που χρησιμοποιήθηκαν για τον έλεγχο των κόμβων βασίζονται στις μεθόδους της στατικής και της δυναμικής ανάλυσης. Οι μέθοδοι αυτοί και ο τρόπος εφαρμογής του στο περιβάλλον -πλαίσιο περιγράφονται λεπτομερώς στο κεφάλαιο 6.

5.2.3 Επαλήθευση στόχων και ενέργειες που εκτελούνται μετά την λήψη των αποτελεσμάτων του SAT4J

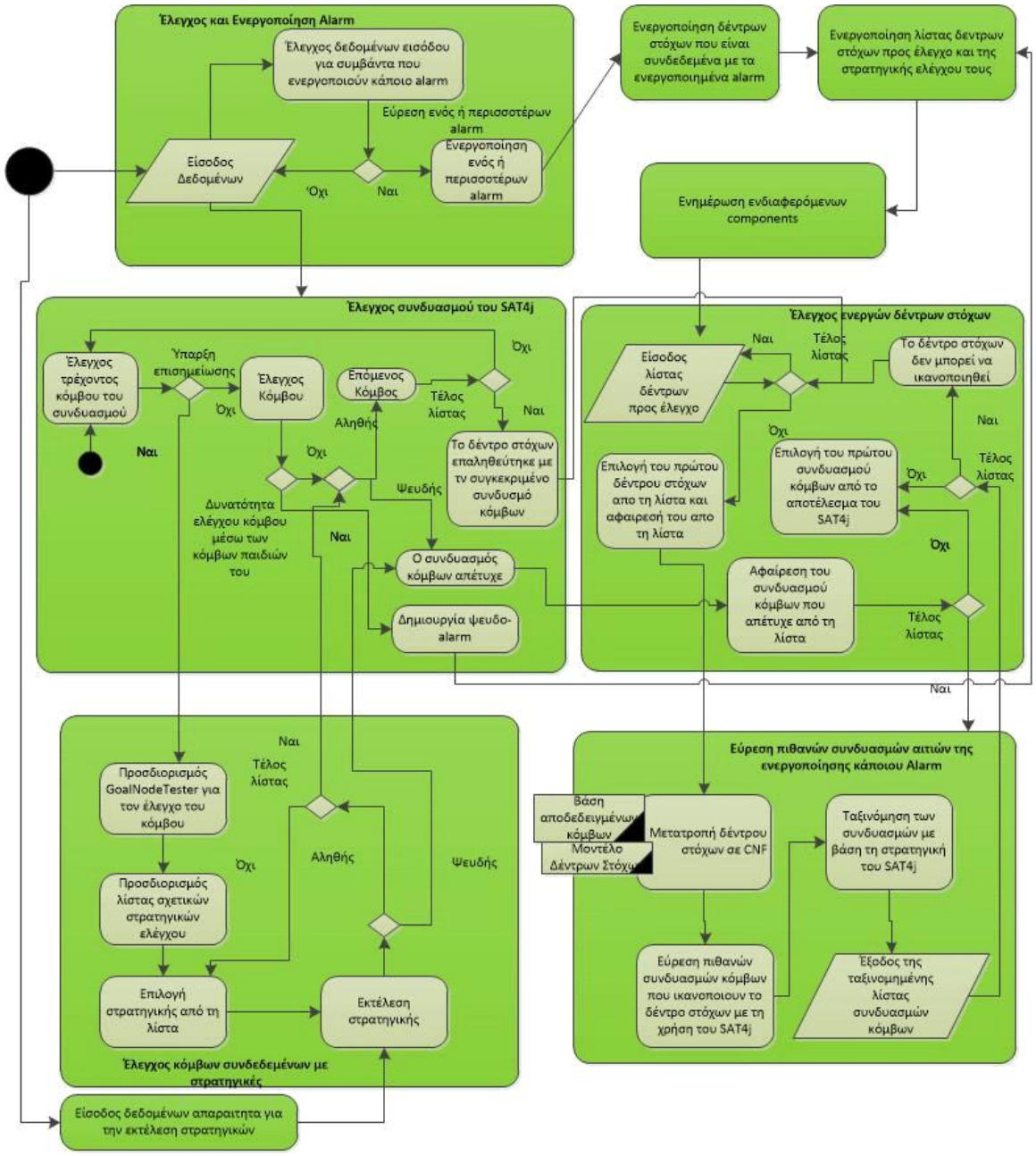
Στην ενότητα αυτή συνοψίζεται η διαδικασία λειτουργίας του SAT4j και παρουσιάζονται συνοπτικά οι λειτουργίες που ενεργοποιούνται μόλις ληφθεί το σύνολο των λύσεων του.

Όπως έχει ήδη αναφερθεί ο SAT4J παίρνει ως είσοδο το μοντέλο δέντρων-στόχων το οποίο έχει περιγραφεί σε υποσύνολα ,σε CNF μορφή. Η μορφή αυτή είναι τέτοια ώστε να εμπεριέχει όλες τις πληροφορίες λειτουργιών και αλληλεπιδράσεων των κόμβων-στόχων. Ως έξοδο της εκτέλεσης του SAT4j έχουμε το σύνολο των συνδυασμών των κόμβων-στόχων, η ικανοποίηση των οποίων οδηγεί στην ικανοποίηση της αρχικής μας υπόθεσης. Μετά την λήψη των λύσεων αυτών βασικό βήμα είναι ο έλεγχος κάθε λύσης, πράγμα το οποίο ισοδυναμεί με τον έλεγχο της αληθοτιμής κάθε κόμβου-στόχου που εμπεριέχεται στη λύση. Αυτό συμβαίνει μόνο στην περίπτωση που η αληθοτιμή αυτή δεν είναι γνωστή ή απαιτούνται παραπάνω στοιχεία για την επιβεβαίωση της. Προκειμένου να ικανοποιήσουμε την απαίτηση ελέγχου της αληθοτιμής ενός κόμβου-στόχου, το σύστημα που αναπτύχθηκε σχεδιάστηκε έτσι ώστε κάθε κόμβος στόχος μιας υπόθεσης να συνδέεται με μια στρατηγική επαλήθευσης. Κάθε στρατηγική επαλήθευση μπορεί να αποτελείται από πολλά σενάρια ελέγχου. Για την επαλήθευση ενός κόμβου απαιτείται η εκτέλεση όλων των σεναρίων ελέγχου που του αντιστοιχούν . Σε περίπτωση που όλα τα σενάρια βρεθούν αληθή ο κόμβος χαρακτηρίζεται ως αληθής και το σύστημα συνεχίζει με τον έλεγχο του επόμενου κόμβου-στόχου της λύσης SAT. Στην περίπτωση που ένα από τα σενάρια ελέγχου δεν ικανοποιηθεί, ο κόμβος χαρακτηρίζεται ως ψευδής και η συγκεκριμένη λύση του SAT4j απορρίπτεται. Το σύστημα έχει σχεδιαστεί έτσι ώστε ο χρήστης να είναι σε θέσει να ορίσει ανεξάρτητα καινούργια σενάρια ελέγχου με σχεδόν καθόλου μεταβολές στον κώδικα του συστήματος. Περισσότερες πληροφορίες για τις στρατηγικές και τα σενάρια ελέγχου καθώς και τον τρόπο υλοποίησης τους παρατίθενται στο κεφάλαιο 6 τις παρούσας διπλωματικής εργασίας.

Κεφάλαιο 6: Ανάλυση του περιβάλλοντος πλαισίου

6.1 Ανάλυση βασικού βρόγχου λειτουργίας

Στην ενότητα αυτή θα περιγράψουμε τον βασικό βρόγχο λειτουργίας του περιβάλλοντος πλαισίου, με στόχο τον έλεγχο ενός μοντελοποιημένου συστήματος λογισμικού. Θα αναλύσουμε τις δραστηριότητες που λαμβάνουν χώρα από τη στιγμή που εισάγονται στο περιβάλλον πλαίσιο τα δεδομένα παρακολούθησης του υπό έλεγχο συστήματος, μέχρι τη στιγμής της απόφασης της τελικής αρχικής αιτίας μίας πιθανής αποτυχίας του. Ακολουθεί το διάγραμμα βασικού βρόγχου λειτουργίας του περιβάλλοντος πλαισίου:



Σχήμα 6.1: Λεπτομερές Διάγραμμα Δραστηριότητας Βασικού Βρόχου Λειτουργίας

Η διαδικασία ξεκινά με την είσοδο δεδομένων, τα οποία προκύπτουν από την παρακολούθηση και καταχώρηση συμβάντων του υπό έλεγχο συστήματος. Στα δεδομένα αυτά αναζητείται η ύπαρξη ενός pattern που συμβολίζει ένα alarm. Η εύρεση ενός τέτοιου alarm, οδηγεί στην ενεργοποίηση του. Εάν βρεθούν περισσότερα από ένα patterns στα δεδομένα εισόδου, τότε αυτό συνεπάγεται την ενεργοποίηση περισσότερων alarms. Εάν ο έλεγχος των δεδομένων δεν έχει σαν αποτέλεσμα την εύρεση κάποιου alarm, τότε το σύστημα αναμένει μέχρι να δοθούν σαν είσοδος

νέα δεδομένα παρακολούθησης.

Όπως έχουν αναφέρει, κάθε alarm είναι συνδεδεμένο με ένα ή περισσότερα δέντρα στόχων. Η ενεργοποίηση κάποιων alarm επομένως, συνεπάγεται την ενεργοποίηση των αντίστοιχων δέντρων στόχων που αναπαριστούν απαιτήσεις του υπό έλεγχο συστήματος και θα πρέπει να ελεγχθεί η σωστή λειτουργία τους. Επίσης θα πρέπει να ενεργοποιηθούν οι αντίστοιχες στρατηγικές που θα πρέπει να ακολουθηθούν για την επίτευξη του ελέγχου των απαιτήσεων. Συνεπώς, συγκροτείται μία λίστα δέντρων στόχων για τα οποία θα πραγματοποιηθεί έλεγχος από το περιβάλλον πλαίσιο. Επίσης , μέσω του Blackboard θα ενημερωθούν οι ενδιαφερόμενοι components για την λίστα δέντρων στόχων που πρέπει να ελεγχθεί.

Σειρά έχει ο έλεγχος των ενεργοποιημένων δέντρων στόχων. Απαραίτητο στοιχείο για την εκκίνηση της διαδικασίας ελέγχου, είναι η είσοδος μιας λίστας η οποία αποτελείται από τα δέντρα στόχων που πρέπει να ελεγχθούν. Εάν η λίστα αυτή είναι κενή, δηλαδή δεν υπάρχουν δέντρα στόχων προς παρακολούθηση, αυτό σημαίνει πως τα alarm που βρέθηκαν δεν αντιστοιχούν σε κανένα υπάρχον δέντρο στόχων. Σε αυτή την περίπτωση, το σύστημα αναμένει την είσοδο της επόμενης λίστας ενεργοποιημένων δέντρων στόχων που θα προκύψει από νέα δεδομένα παρακολούθησης του συστήματος υπό έλεγχο. Σε περίπτωση που η λίστα περιέχει τουλάχιστον ένα δέντρο στόχων, τότε είναι δυνατή η έναρξη της διαδικασίας ελέγχου. Επιλέγεται το πρώτο δέντρο της λίστας των ενεργοποιημένων δέντρων στόχων, το οποίο δίνεται σαν είσοδος στον SAT Solver.

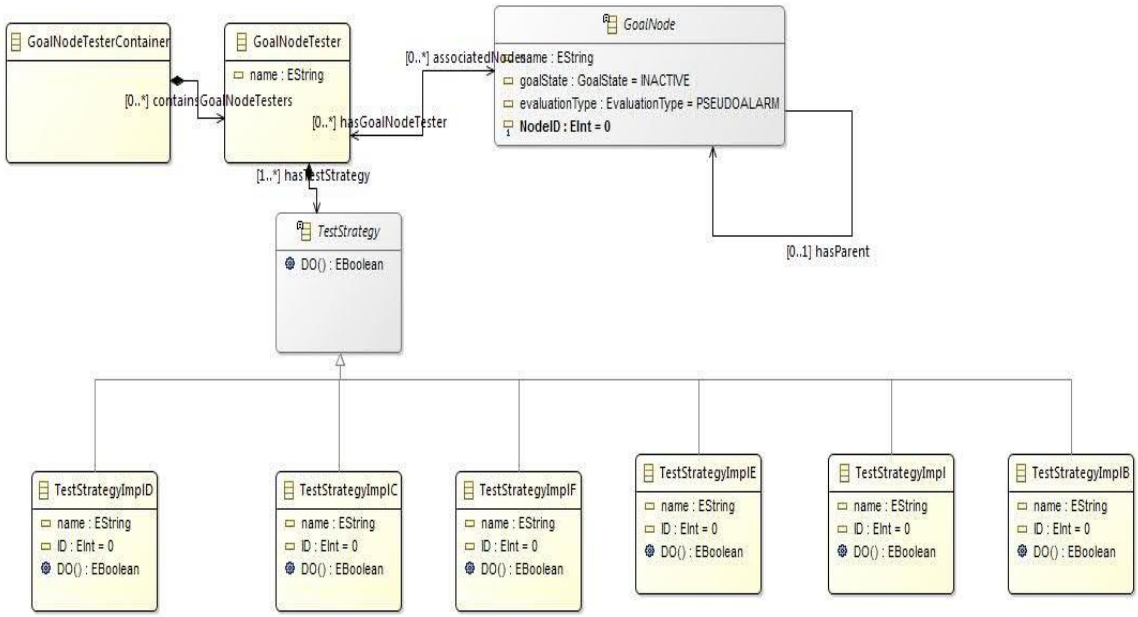
Ακολουθεί η μετατροπή του δέντρου στόχων σε έκφραση CNF, με τον τρόπο που περιγράψαμε αναλυτικά στο κεφάλαιο 5. Για να γίνει αυτό εφικτό, είναι απαραίτητη η πρόσβαση του component SAT Solver στο μοντέλο του δέντρου στόχων που έχει πάρει ως είσοδο. Επίσης είναι απαραίτητη η πρόσβαση στη βάση (Blackboard) που διατηρεί τις αληθοτιμές των ήδη αποδεδειγμένων κόμβων, βάσει των οποίων λαμβάνει αποφάσεις για την παραγωγή συνδυασμών κόμβων που επαληθεύουν το δέντρο, όπως εξηγήσαμε αναλυτικά στο κεφάλαιο 5. Τα δεδομένα αυτά θα ληφθούν υπόψη για την παραγωγή N λύσεων από τον SAT4j, έτσι ώστε να περιοριστεί ο χρόνος εκτέλεσης του SAT4j και να έχουμε μια καλή λύση σε μικρό χρονικό διάστημα. Η έκφραση CNF θα δοθεί στον SAT4j για την εξαγωγή των πιθανών συνδυασμών αρχικών αιτιών , δηλαδή των κόμβων που πρέπει να ελεγχθούν ώστε να αποφανθεί η ισχύς της αρχικής απαίτησης. Έξοδος του SAT4j είναι μία λίστα συνδυασμών κόμβων που ικανοποιούν το δέντρο στόχων. Εάν η λίστα είναι κενή , αυτό σημαίνει πως δεν υπάρχει συνδυασμός κόμβων που ικανοποιούν το δέντρο στόχων. Εάν η λίστα δεν είναι κενή ,τότε ταξινομείται βάσει της στρατηγικής του SAT4j που εξηγήσαμε στο κεφάλαιο 5 και επιχειρείται ο έλεγχος του πρώτου συνδυασμού.
Όπως θα δούμε και παρακάτω, εάν ο συνδυασμός αυτός αποτύχει να επαληθεύσει το δέντρο στόχων, τότε αφαιρείται από την λίστα και συνεχίζουμε με τον επόμενο συνδυασμό που είναι πλέον στην αρχή της λίστας. Αν η λίστα των συνδυασμών του SAT4j εξαντληθεί σημαίνει ότι και οι N συνδυασμοί που επέστρεψε ο SAT4j απότυχαν. Επόμενο βήμα είναι να ξανατρέξουμε τον SAT4j με την νέα βάση γνώσης για να μας δώσει ένα νέο σύνολο λύσεων. Η διαδικασία αυτή σταματάει αν επιτευχθεί η επαλήθευση ενός συνδυασμού κόμβων ή αν δεν επιστραφεί λύση από την SAT4j. Αν κάποιος συνδυασμός κόμβων επαληθευτεί, τότε επαληθεύει και το αντίστοιχο δέντρο στόχων, το οποίο αφαιρείται από την λίστα προτεραιότητας των ενεργών δέντρων στόχων. Ο συνδυασμός κόμβων του δέντρου στόχων που επαλήθευσε την αρχική απαίτηση αποτελεί την διάγνωση των αρχικών αιτιών που οδήγησε στην ενεργοποίηση της υπόθεσης αυτής.
Σε αυτό το σημείο θα περιγράψουμε πιο αναλυτικά την διαδικασία επαλήθευσης ενός συνδυασμού κόμβων, όπως αυτός προκύπτει από τον SAT4j. Όπως αναφέραμε και πριν, ο SAT4j δίνει ως έξοδο μία λίστα συνδυασμών κόμβων η οποία ταξινομείται. Στη συνέχεια επιλέγεται ο

πρώτος συνδυασμός της λίστας αυτής για επαλήθευση. Έπειτα επιλέγεται ο πρώτος κόμβος του συνδυασμού αυτού με στόχο τον έλεγχό του. Ο κόμβος που έχει επιλεχθεί, ελέγχεται για την ύπαρξη επισημείωσης καθώς και για την δυνατότητα επαλήθευσής του μέσω της επαλήθευσης των κόμβων παιδιών του. Έτσι, προκύπτουν οι παρακάτω περιπτώσεις :

- Ο κόμβος δεν περιέχει κάποια επισημείωση και μπορεί να αποδειχτεί άμεσα , χωρίς την επαλήθευση των κόμβων παιδιών του. Σε αυτή τη περίπτωση , η αληθοτιμή του κόμβου προκύπτει από το πεδίο GoalState του κόμβου στο μοντέλο στο οποίο ανήκει.

- Ο κόμβος δεν περιέχει κάποια επισημείωση αλλά δεν μπορεί να αποδειχθεί άμεσα. Αυτό στην ουσία σημαίνει πως ο κόμβος είναι κάποιος ενδιάμεσος κόμβος, η αληθοτιμή του οποίου εξαρτάται από τις αληθοτιμές των κόμβων παιδιών του, οι οποίες σε αυτό το σημείο δεν είναι ακόμα γνωστές. Αυτούς τους κόμβους μπορούμε να τους θεωρήσουμε σαν ξεχωριστά υπο-δέντρα στόχων, και να αφήσουμε τον SAT4j να χειριστεί την επαλήθευσή τους ,όπως ακριβώς κάνει με τα υπόλοιπα δέντρα στόχων. Για να επιτευχθεί αυτό , αρκεί η προσθήκη τους στην λίστα των ενεργοποιημένων δέντρων που δίνεται ως είσοδος στον SAT Solver.

- Ο κόμβος περιέχει κάποια επισημείωση , η οποία υποδεικνύει πως η επαλήθευση του κόμβου αυτού επιτυγχάνεται μόνο μέσω της εκτέλεσης ενός συνόλου στρατηγικών που συνδέονται με αυτόν. Συγκεκριμένα , κάθε κόμβος που περιέχει μία τέτοια υποσημείωση, είναι συνδεδεμένος με έναν GoalNodeTester. Ο GoalNodeTester αποτελεί το σύνολο των στρατηγικών ελέγχου που πρέπει να εκτελεστούν για την απόφαση της ισχύος του συγκεκριμένου κόμβου. Στο παρακάτω σχήμα (σχήμα 6.2)βλέπουμε ένα παράδειγμα στο οποίο ένας GoalNode είναι συνδεδεμένος με έναν GoalNodeTester , που υποδεικνύει ποια στρατηγική πρέπει να ακολουθηθεί για τον έλεγχο του συγκεκριμένου κόμβου. Στο συγκεκριμένο παράδειγμα , υπάρχουν έξι στρατηγικές συνδεδεμένες με τον GoalNodeTester αυτόν. Η εκτέλεση όλων των στρατηγικών απαιτείται για την απόφαση της ισχύος ενός κόμβου. Μόνο εάν όλες οι στρατηγικές επαληθευτούν, τότε ο κόμβος μπορεί να θεωρηθεί αληθής. Εάν μία στρατηγική αποτύχει, τότε ο κόμβος θεωρείται ψευδής και δεν είναι απαραίτητο να συνεχιστεί η διαδικασία εκτέλεσης των υπόλοιπων στρατηγικών.
  Επομένως, για έναν κόμβο με επισημείωση, χρειάζεται να προσδιοριστεί ο αντίστοιχος GoalNodeTester και οι αντίστοιχες στρατηγικές που θα βοηθήσουν στην επαλήθευση του. Από την λίστα στρατηγικών που προκύπτει, επιλέγεται να εκτελεστεί η πρώτη. Εάν το αποτέλεσμα της εκτέλεσης δείχνει πως ο κόμβος είναι ψευδής , τότε η εκτέλεση των υπόλοιπων στρατηγικών της λίστας διακόπτεται. Εάν το αποτέλεσμα της εκτέλεσης δείχνει πως ο κόμβος είναι αληθής , τότε δεν μπορούμε άμεσα να συμπεράνουμε την ισχύ του αντίστοιχου κόμβου, καθώς θα πρέπει να εκτελεστούν όλες οι στρατηγικές. Επομένως επιλέγεται να εκτελεστεί η επόμενη στρατηγική της λίστας. Η ίδια διαδικασία ακολουθείται μέχρι να εκτελεστούν όλες οι στρατηγικές της λίστας, ή μέχρι μία στρατηγική να αποδείξει πως ο κόμβος είναι ψευδής.



Σχήμα 6.2: Σύνδεση GoalNode με έναν GoalNodeTester και πολλά Strategies

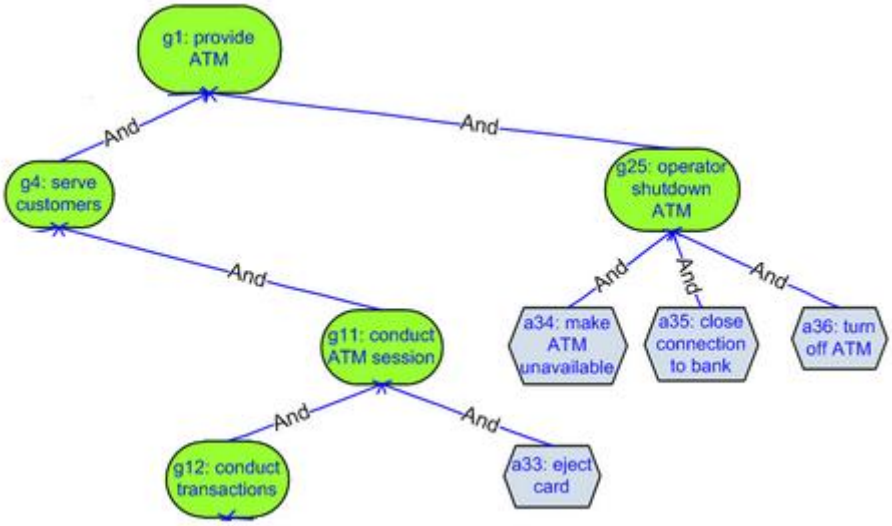
Για κάθε μία από τις παραπάνω περιπτώσεις , η διαδικασία συνεχίζεται με κοινό τρόπο. Εάν το αποτέλεσμα είναι η απόδειξη του κόμβου ως αληθής, τότε η ίδια διαδικασία επαλήθευσης συνεχίζεται για τον επόμενο κόμβο που βρίσκεται στη λίστα συνδυασμών κόμβων που εξετάζεται. Εάν το αποτέλεσμα είναι η απόδειξη του κόμβου ως ψευδής, τότε ο συνδυασμός κόμβων που εξετάζεται , θεωρείται πως έχει αποτύχει. Ο αποτυχημένος συνδυασμός κόμβων αφαιρείται από την λίστα συνδυασμών που προέκυψε από τον SAT4j, και παίρνει σειρά ο επόμενος συνδυασμός για την επανάληψη της διαδικασίας.
Μόλις εξεταστεί μία λίστα κόμβων μέχρι το τέλος της, που σημαίνει πως όλοι οι κόμβοι που την αποτελούν έχουν αποδειχτεί ως αληθείς , το δέντρο στόχων υπό έλεγχο επαληθεύεται. Αυτό σημαίνει πως η αντίστοιχη απαίτηση του υπό έλεγχο συστήματος έχει επαληθευτεί, και οι πιθανές αιτίες μίας πιθανής αποτυχίας της έχουν ελεγχθεί και λειτουργούν ορθά. Επόμενο βήμα είναι η έναρξη της ίδιας διαδικασίας για την επόμενη απαίτηση που πρέπει να εξεταστεί, δηλαδή το επόμενο δέντρο στόχων που βρίσκεται στη λίστα δέντρων στόχων προς έλεγχο. Αυτό γίνεται αφαιρώντας το αποδεδειγμένο δέντρο στόχων από τη λίστα αυτή, και επιλέγοντας πάλι το πρώτο δέντρο της λίστας.

6.2 Ανάλυση μεθόδου δυναμικού ελέγχου

Στην ενότητα αυτή παρουσιάζεται ο δυναμικός τρόπος επαλήθευσης ενός κόμβου, όπως αυτός έχει υλοποιηθεί στο περιβάλλον πλαίσιο που αναπτύχθηκε. Συγκεκριμένα παρουσιάζονται οι διαδικασίες εκκίνησης και εκτέλεσης της δυναμικής ανάλυσης καθώς και ο έλεγχος των αποτελεσμάτων της .

Όπως έχει ήδη ειπωθεί στο κεφάλαιο 2 της παρούσας διπλωματικής εργασίας, η δυναμική ανάλυση ενός συστήματος απαιτεί την εκτέλεση του υπό έλεγχο συστήματος πριν την εφαρμογή της. Τα δεδομένα περιγραφής της εκτέλεσης αυτού του υπό έλεγχο συστήματος θα πρέπει στη συνέχεια να συγκεντρωθούν και να συγκριθούν με το αναμενόμενο αποτέλεσμα ορθής λειτουργίας. Η ίδια μεθοδολογία χρησιμοποιείται και στο περιβάλλον πλαίσιο που αναπτύχθηκε και περιγράφεται αναλυτικότερα στη συνέχεια της ενότητας.

Ας υποθέσουμε, ότι απαιτείται να εξετάσουμε την λειτουργία ενός συστήματος, να εξετάσουμε δηλαδή αν το υπό έλεγχο σύστημα λειτουργεί σύμφωνα με τις τεχνικές προδιαγραφές και τις προκαθορισμένες απαιτήσεις. Για το σκοπό αυτό θα πρέπει σε πρώτο στάδιο να ορίσουμε λεπτομερώς αυτές τις τεχνικές προδιαγραφές και απαιτήσεις του. Η μέθοδος που χρησιμοποιήσαμε για το ορισμό των απαιτήσεων είναι η μέθοδος μοντελοποίησης τους σε δέντρα-στόχων. Ένα δέντρο-στόχων περιλαμβάνει ως κύριο κόμβο-ρίζα την βασική απαίτηση του υπό έλεγχο συστήματος και ως κόμβους του δέντρου τις υπολειτουργίες, η ικανοποίηση των οποίων οδηγεί στην αρχική απαίτηση. Να σημειώσουμε στο σημείο αυτό ότι ένα μοντέλο δέντρων στόχων μπορεί να περιλαμβάνει περισσότερα δέντρα στόχων , καθώς το σύστημα μπορεί να εκφράζεται από ένα σύνολο απαιτήσεων. Ένα παράδειγμα ενός δέντρου-στόχων δίνεται παρακάτω [60][61].



Σχήμα 6.3: Παράδειγμα μοντέλου απαιτήσεων

Έχοντας λοιπόν ορίσει πλήρως τις απαιτήσεις που θα πρέπει να εκτελεί το σύστημα, συνεχίζουμε με το επόμενο στάδιο, το οποίο είναι η χρήση του συστήματος και η παραγωγή πληροφοριών εκτέλεσης. Οι πληροφορίες αυτές θα πρέπει να περιέχουν λεπτομέρειες για όλες τις ενέργειες και καταστάσεις του συστήματος κατά τη διάρκεια λειτουργίας του. Ιδιαίτερη βάση θα πρέπει να δοθεί στις πληροφορίες εκτέλεσης οι οποίες αναφέρονται σε κατάσταση λανθασμένης λειτουργίας του συστήματος. Αυτή η πληροφορία μπορεί να είναι μια επισημείωση (alarm) ή ένα μοτίβο πληροφοριών που χαρακτηρίζουν το λάθος. Στο περιβάλλον πλαίσιο που αναπτύχθηκε στα πλαίσια αυτής της διπλωματικής η δυναμική ανάλυση βασίζεται στον έλεγχο των πληροφοριών εκτέλεσης του υπό-έλεγχο συστήματος και την αναζήτηση επισημειώσεων που

υποδεικνύουν τη λανθασμένη συμπεριφορά του. Όπως περιγράφθηκε και στην προηγούμενη ενότητα η εύρεση μιας τέτοιας επισημείωσης (alarm) εκκινεί τη διαδικασία έλεγχου πληρότητας των απαιτήσεων του συστήματος. Συγκεκριμένα ενεργοποιούνται μόνο τα δέντρα-στόχων του μοντέλου που είναι συνδεδεμένα με τα alarms που βρέθηκαν στον έλεγχο των πληροφοριών εκτέλεσης. Μετά την εύρεση του alarm το περιβάλλον πλαίσιο συνεχίσει με την ανάλυση του μοντέλου σε CNF μορφή και την επίλυση του προβλήματος SAT. Μετά την εφαρμογή του SAT4j το σύστημα έρχεται αντιμέτωπο με το πρόβλημα του ελέγχου της αληθοτιμής των κόμβων που επιστράφηκαν. Όπως περιγράψαμε οι κόμβοι αυτοί αποτελούν υπολειτουργίες του υπό έλεγχου συστήματος. Επιπλέον οι πληροφορίες εκτέλεσης περιλαμβάνουν δεδομένα που αφορούν την επιτυχημένη ή αποτυχημένη λειτουργία αυτών. Το πρόβλημα λοιπόν εύρεσης της αληθοτιμής των κόμβων που επιστράφηκαν από τον SAT4j ανάγεται στη αναζήτηση συγκριμένων δεδομένων στις πληροφορίες εκτέλεσης του υπό-έλεγχο συστήματος οι οποίες μπορούν να υποδείξουν την ορθή ή όχι εκτέλεση της υπολειτουργίας. Η διαδικασία αυτή ελέγχου μιας υπολειτουργίας ενός συστήματος ονομάζεται log analysis . Οι στρατηγικές έλεγχου που αντιστοιχούν σε κάθε κόμβο στην περίπτωση της δυναμικής ανάλυσης περιέχουν σενάρια ελέγχου που βασίζονται στην επεξεργασία των πληροφοριών εκτέλεσης του υπό-έλεγχο συστήματος και στην σύγκριση τους με την ορθή λειτουργία . Ένα παράδειγμα ενός τέτοιου σεναρίου ελέγχου για τον κόμβο a36 του παραδείγματος δέντρου στόχων που δόθηκε παραπάνω (Σχήμα 6.3) θα μπορούσε να είναι η αναζήτηση στις πληροφορίες εκτέλεσης του ATM για γεγονότα που υποδεικνύουν ότι μετά την εκτέλεση της λειτουργίας απενεργοποίησης του ATM, το ATM απενεργοποιήθηκε επιτυχώς, όπως είναι η αναζήτηση της πληροφορίας "ATM was sucessfully turned down , code : 1234" . Η εύρεση της πληροφορίας αυτής θα σήμαινε άμεσα την ικανοποίηση της υπολειτουργίας και περαιτέρω την ικανοποίηση του κόμβου που επιστράφηκε από την λύση του SAT4j, ενώ η μη εύρεση της πληροφορίας αυτής θα οδηγούσε στην απόρριψη του κόμβου και της λύσης του SAT4j που επιστράφηκε .

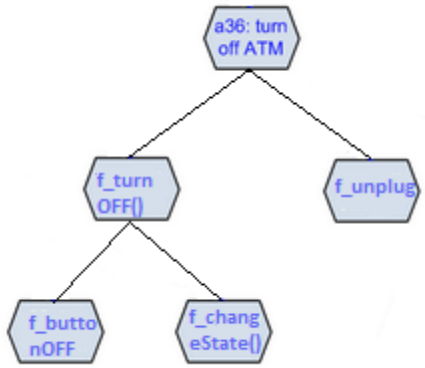
6.3 Ανάλυση μεθόδου στατικού ελέγχου

Στην ενότητα αυτή παρουσιάζεται ο στατικός τρόπος επαλήθευσης ενός κόμβου, όπως αυτός έχει υλοποιηθεί στο περιβάλλον πλαίσιο που αναπτύχθηκε. Συγκεκριμένα παρουσιάζονται οι διαδικασίες εκκίνησης και εκτέλεσης της στατικής ανάλυσης καθώς και ο έλεγχος των αποτελεσμάτων της .

Η στατική ανάλυση, σε αντίθεση με τη δυναμική, δεν απαιτεί την εκτέλεση του υπό έλεγχο συστήματος για την εκτέλεση της. Αυτό σημαίνει ότι ο χρήστης μπορεί οποιαδήποτε στιγμή να εκκινήσει τη διαδικασία στατικής ανάλυσης για τον έλεγχο ενός συγκριμένου συνόλου λειτουργιών του υπό - έλεγχο συστήματος. Η στατική ανάλυση έχει ως στόχο τον έλεγχο τμημάτων του κώδικα του συστήματος και μπορεί να εφαρμοστεί ακόμη και στην περίπτωση όπου δεν υπάρχει πλήρης υλοποίηση του. Στο περιβάλλον- πλαίσιο που αναπτύχθηκε κατά την εφαρμογή της στατικής ανάλυσης εστιάσαμε στην δημιουργία σεναρίων που θα ελέγχουν την

ορθή ή λανθασμένη λειτουργία συγκεκριμένων τμημάτων του κώδικα ενός συστήματος, κάνοντας χρήση του περιβάλλον πλαισίου JUnit.

Όπως και στην περίπτωση της δυναμικής ανάλυσης έτσι και στη στατική ανάλυση απαραίτητη προϋπόθεση για την επιτυχή εφαρμογή της είναι ο ακριβής προσδιορισμός των απαιτήσεων και των τεχνικών προδιαγραφών που πρέπει να ικανοποιούν τα τμήματα κώδικα που θα εξετασθούν. Για το σκοπό αυτό απαιτείται η δημιουργία ενός νέου μοντέλου στόχων το οποίο θα περιγράφει τις λειτουργίες που μπορούν να εξεταστούν μέσω αυτής της τεχνικής. Οι λειτουργίες αυτές μπορούν να περιγραφούν με όμοιο τρόπο με τα μοντέλα δυναμικής ανάλυσης μέσω δέντρων στόχων. Οι διάφορες λειτουργίες του συστήματος αναλύονται σε δέντρα τα οποία δίνονται ως είσοδο στο περιβάλλον πλαίσιο που αναπτύχθηκε. Έχοντας ως αναφορά το παράδειγμα του σχήματος (σχήμα 6.4) , η λειτουργία του κόμβου a36 θα μπορούσε να αναλυθεί σε υπολειτουργίες τμημάτων κώδικα όπως φαίνεται στο παρακάτω σχήμα :



Σχήμα 6.4: μοντέλου απαιτήσεων στατικής ανάλυσης

Η διαδικασία εκκίνησης της διαδικασίας στατικού ελέγχου εξαρτάται από το χρήστη του περιβάλλοντος πλαισίου που αναπτύχθηκε. Ο χρήστης μπορεί να εκκινήσει τι διαδικασία ελέγχου επιλέγοντας συγκεκριμένα δέντρα στόχων ή αλλιώς συγκεκριμένες λειτουργίες. Με την επιλογή των δέντρων στόχων , ενεργοποιούνται τα alarm τα οποία σχετίζονται με αυτά . Η διαδικασία στη συνέχεια είναι παρόμοια με αυτή της δυναμικής ανάλυσης. Το μοντέλο επεξεργάζεται και επιστρέφεται η CNF έκφραση αυτού, η οποία δίνεται ως είσοδο στο SAT4j προκειμένου να πάρουμε τον συνδυασμό εκείνων των λύσεων που θα ικανοποιήσουν τους αρχικούς κόμβους-στόχους. Καταλήγουμε λοιπόν στο πρόβλημα εύρεσης της αληθοτιμής των κόμβων αυτών. Για την εύρεση των αληθοτιμών των κόμβων αυτών γίνεται χρήση των στρατηγικών επαλήθευσης οι οποίες στην περίπτωση της στατικής ανάλυσης αποτελούνται από σενάρια ελέγχου JUnit. Η δημιουργία των σεναρίων JUnit είναι ανεξάρτητη από την υλοποίηση του υπόλοιπου συστήματος του περιβάλλον πλαισίου. Ο χρήστης έχει τη δυνατότητα δημιουργίας νέων σεναρίων χωρίς καθόλου αλλαγές στον βασικό κώδικα του συστήματος. Κάθε κόμβος μπορεί να σχετίζεται με ένα ή περισσότερα σενάρια ελέγχου JUnit. Τα σενάρια αυτά ελέγχουν συγκεκριμένα τμήματα του κώδικα του υπό-έλεγχο συστήματος. Εμπεριέχουν την πληροφορία της ορθής λειτουργίας για συγκεκριμένα στοιχεία εισόδου. Τα στοιχεία εισόδου αυτά δίνονται

στη συνέχεια ως είσοδος στο τμήμα του κώδικα που είναι υπό εξέταση και συγκρίνονται με τα αναμενόμενα αποτελέσματα της κατάστασης ορθής λειτουργίας. Εάν τα αποτελέσματα που προκύψουν ως έξοδο του τμήματος που εξετάζεται είναι ίδια με τα αναμενόμενα αποτελέσματα που προσδιορίστηκαν στο σενάριο ελέγχου JUnit για τα συγκεκριμένα στοιχεία εισόδου, ο κόμβος σημειώνεται ως αληθής , ενώ σε αντίθετη περίπτωση ως ψευδής και η λύση του SAT4j απορρίπτεται.

Η στατική ανάλυση μπορεί να χρησιμοποιηθεί συνδυαστικά με την δυναμική ανάλυση. Για παράδειγμα στην περίπτωση αποτυχίας ορθής επαλήθευσης μιας απαίτησης κατά την δυναμική ανάλυση, ο χρήστης μπορεί να εκτελέσει στατική ανάλυση για όλους τους υπό-κόμβους του δέντρου-στόχων της απαίτησης για την οποία η δυναμική ανάλυση έδειξε πως υπάρχει κάποιο πρόβλημα δυσλειτουργίας επιχειρώντας μια προσπάθεια ανεύρεσης του αιτίου της δυσλειτουργίας αυτής. Η διαδικασία αυτή περιγράφεται λεπτομερώς μέσω ενός παραδείγματος πραγματικής χρήσης του περιβάλλοντος πλαισίου , στο επόμενο κεφάλαιο.

6 .4 Περίπτωση χρήσης

Σε αυτή την ενότητα παρουσιάζουμε ένα παράδειγμα χρήσης του προτεινόμενου περιβάλλοντος-πλαισίου. Ως παράδειγμα επιλέγουμε την περίπτωση μίας προσομοίωσης ενός ATM (automated teller machine). Χρησιμοποιήσαμε μοντελοποίηση κάποιων βασικών λειτουργιών του ATM, και επισυνάψαμε στα μοντέλα που προέκυψαν αντίστοιχες στρατηγικές ελέγχου για την επαλήθευση των λειτουργιών του ATM. Όπως θα εξηγήσουμε αναλυτικά παρακάτω, μελετάμε την περίπτωση που μία απαίτηση του συστήματος ATM αποτυγχάνει. Αναλύουμε λοιπόν με τη βοήθεια του προτεινόμενου περιβάλλοντος πλαισίου τις αρχικές αιτίες που μπορεί να οδήγησαν στην αποτυχία αυτής της απαίτησης. Στη συνέχεια, παράγουμε με τη βοήθεια του SAT4j ένα σύνολο πιθανών συνδυασμών αρχικών αιτιών που πρέπει να ελεγχθούν με σκοπό τη εύρεση της αρχικής αιτίας που είναι υπεύθυνη για την αποτυχία της απαίτησης. Έπειτα ξεκινάμε την διαδικασία ελέγχου των αρχικών αιτιών , με τρόπο που περιγράψαμε στην προηγούμενη ενότητα, μέχρι να επαληθευτεί η αρχική απαίτηση. Χρησιμοποιούμε όλους του τρόπους επαλήθευσης που περιγράψαμε παραπάνω, δηλαδή άμεση επαλήθευση κόμβου, επαλήθευση κόμβου μέσω των κόμβων παιδιών του και επαλήθευση κόμβου με την εκτέλεση στρατηγικών ελέγχου, δίνοντας έμφαση στον τελευταίο τρόπο. Χρησιμοποιήσαμε δύο είδη στρατηγικών, τη δυναμική και τη στατική ανάλυση αρχικών αιτιών. Κατά τη διάρκεια της δυναμικής ανάλυσης , γίνεται ανάλυση των δεδομένων παρακολούθησης του ATM που έχουν δοθεί στο περιβάλλον πλαίσιο, με σκοπό την ανακάλυψη δεδομένων που υποδεικνύουν πως μία αρχική αιτία της απαίτησης έχει αποτύχει. Κατά τη διάρκεια της στατικής ανάλυσης, εκτελούνται αλγόριθμοι ελέγχου πάνω στις μεθόδους του κώδικα του ATM, ώστε να εντοπιστεί η μέθοδος που παρουσιάζει κάποιο πρόβλημα και οδηγεί στην αποτυχία της απαιτήσεως που ελέγχουμε. Παρέχεται η έξοδος του περιβάλλοντος πλαισίου η οποία είναι μία λεπτομερής αναφορά των αποτελεσμάτων έλεγχου κάθε αρχικής αιτίας.

6.4.1 Ενεργοποίηση διαδικασίας ελέγχου

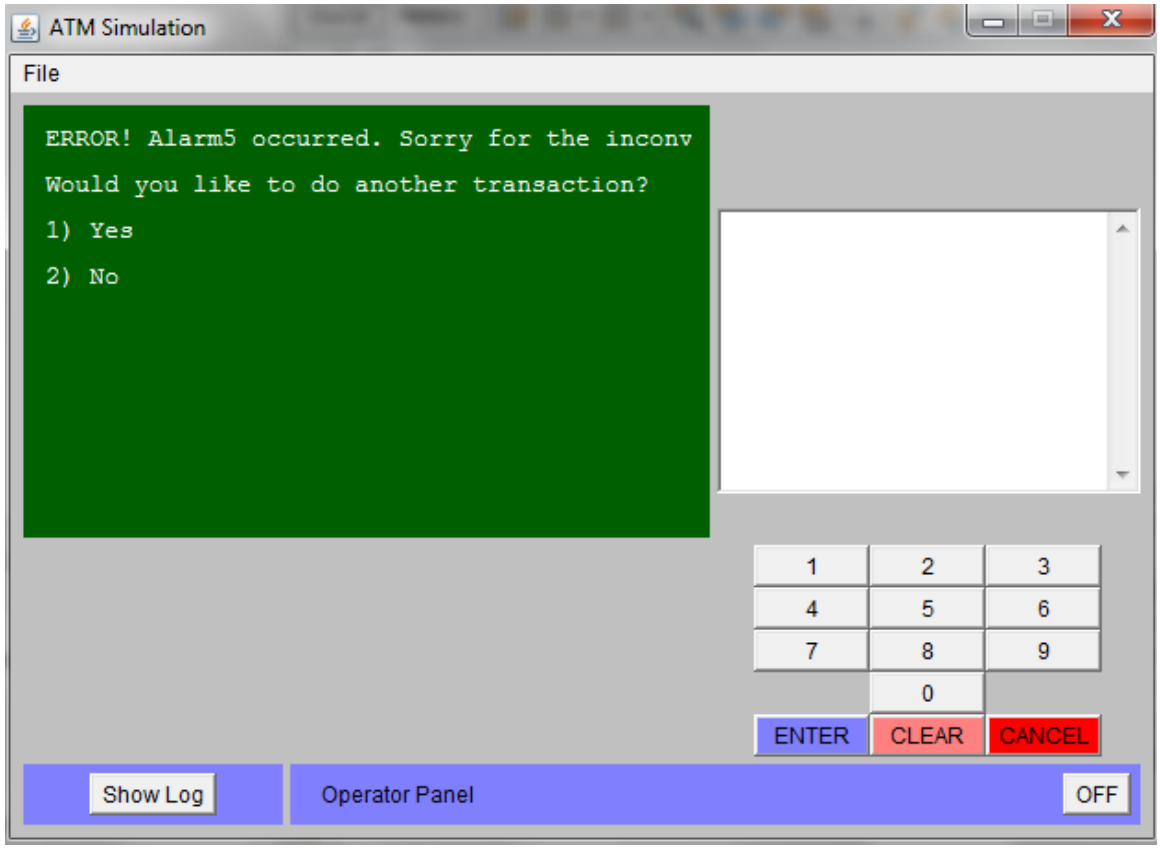
Για τις ανάγκες αυτού του παραδείγματος, χρησιμοποιούμε την προσομοίωση ενός ATM (automated teller machine) [62]. Για τις ανάγκες της παρουσίασης της λειτουργίας του περιβάλλοντος-πλαισίου, δημιουργήθηκε σκόπιμα κάποιο λάθος στον κώδικα του ATM, ώστε να μπορούμε να υποθέσουμε ότι μία απαίτηση του συστήματος δεν ικανοποιείται. Στην περίπτωσή μας, η απαίτηση αυτή είναι η "Επιτυχής ανάληψη χρημάτων" . Δίνονται λοιπόν δύο μοντέλα που αναπαριστούν την απαίτηση αυτή, δηλαδή έχουν ως ρίζα του δέντρου στόχων την απαίτηση "Ανάληψη χρημάτων". Η διαφοροποίηση αυτών των δέντρων στόχων είναι οι υπολειτουργίες που περιέχουν. Στο πρώτο δέντρο στόχων , περιέχονται υπολειτουργίες οι οποίες είναι δυνατόν να ελεγχθούν με ανάλυση δεδομένων παρακολούθησης του ATM. Στο δεύτερο δέντρο στόχων, περιέχονται μέθοδοι του κώδικα του ATM, από τις οποίες εξαρτάται η σωστή λειτουργία της απαίτησης "Ανάληψη χρημάτων". Θα εξετάσουμε αναλυτικά τις δύο αυτές περιπτώσεις στις επόμενες ενότητες.
Ο τρόπος ενεργοποίησης των δέντρων στόχων μπορεί να γίνει με τον παρακάτω τρόπο. Το σύστημα ATM παρακολουθείται και καταγράφονται δεδομένα για κάθε κίνηση που εκτελείται πάνω σε αυτό. Αφού απενεργοποιηθεί το ATM, τα σύνολο των δεδομένων που έχουν καταγραφεί, είναι διαθέσιμα σε ένα παραγόμενο αρχείο (Log). Το αρχείο αυτό, είναι της μορφής που φαίνεται στο παρακάτω σχήμα (σχήμα 6.5), και δείχνει μία σειρά ενεργειών πάνω στο ATM, που ήταν επιτυχείς :

```
INQUIRY  CARD# 2 TRANS# 1 FROM  0 NO TO NO AMOUNT
SUCCESS
TRANSFER CARD# 2 TRANS# 3 FROM  0 TO  0 $40.00
SUCCESS
INIT_DEP CARD# 2 TRANS# 4 NO FROM TO  0 $20.00
SUCCESS
COMP_DEP CARD# 2 TRANS# 4 NO FROM TO  0 $20.00
SUCCESS
WITHDRAW CARD# 2 TRANS# 5 FROM  0 NO TO $50.00
SUCCESS
$50.00
WITHDRAW CARD# 2 TRANS# 5 FROM  0 NO TO $20.00
SUCCESS
$20.00
```

Σχήμα 6.5 : Αρχείο καταγραφής δεδομένων με επιτυχείς ενέργειες

Στην περίπτωση που η απαίτηση του ATM "Επιτυχής Ανάληψη χρημάτων" δεν ικανοποιηθεί, το σύστημα είναι προγραμματισμένο να παράγει ένα Alarm, το οποίο καταγράφεται στα δεδομένα παρακολούθησης . Προκαλέσαμε λοιπόν την αποτυχία της παραπάνω απαίτησης, με αποτέλεσμα

μία ενέργεια ανάληψης χρημάτων να μην ήταν επιτυχής. Αποτέλεσμα αυτής της ενέργειας είναι η παραγωγή του αναμενόμενου Alarm και η καταγραφή του στα δεδομένα παρακολούθησης (Log):



Σχήμα 6.6: Παράδειγμα σφάλματος σε προσομοίωση ATM

```
INQUIRY  CARD# 2 TRANS# 1 FROM  0 NO TO NO AMOUNT
SUCCESS
WITHDRAW CARD# 2 TRANS# 2 FROM  0 NO TO NO AMOUNT
FAILURE ERROR! Alarm5 occurred. Sorry for the inconvenience
```

Σχήμα 6.7: Αρχείο καταγραφής δεδομένων με επιτυχείς ενέργειες

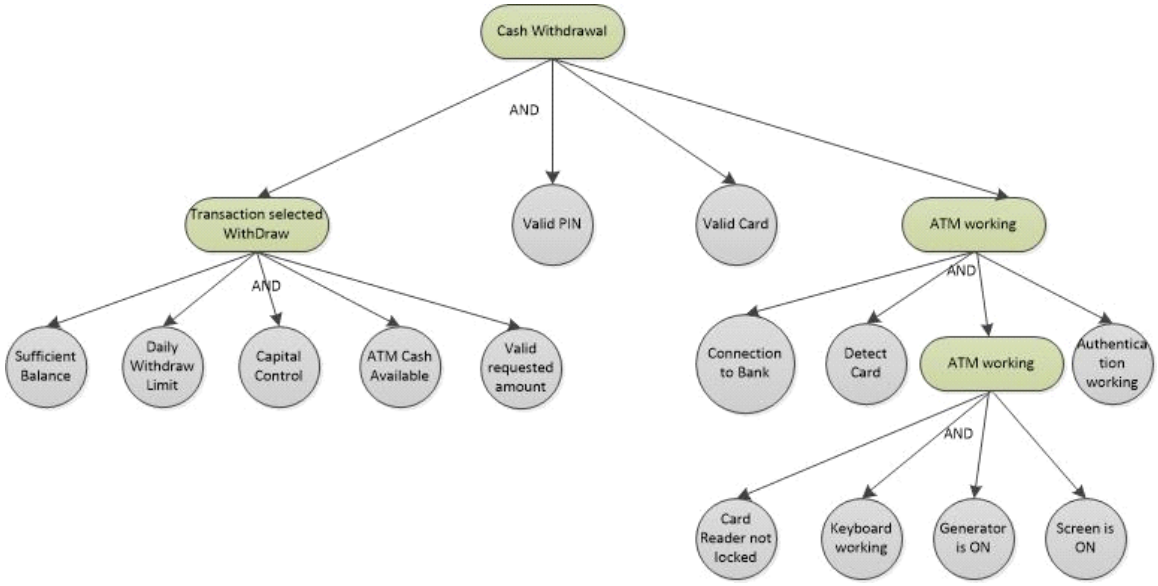
Για τις ανάγκες του παραδείγματος, δίνουμε το παραπάνω αρχείο δεδομένων παρακολούθησης

στο περιβάλλον πλαίσιο μας. Στα δεδομένα αυτά αναζητείται η ύπαρξη ενός pattern που συμβολίζει ένα alarm. Η εύρεση ενός τέτοιου alarm, οδηγεί στην ενεργοποίηση του. Εάν βρεθούν περισσότερα από ένα patterns στα δεδομένα εισόδου, τότε αυτό συνεπάγεται την ενεργοποίηση περισσότερων alarms. Στο παράδειγμα αυτό, το περιβάλλον πλαίσιο θα βρει το Alarm 5 μετά την ανάλυση των δεδομένων εισόδου, και συνεπώς το alarm5 θα ενεργοποιηθεί. Όπως έχουν αναφέρει, κάθε alarm είναι συνδεδεμένο με ένα η περισσότερα δέντρα στόχων. Η ενεργοποίηση κάποιων alarm επομένως, συνεπάγεται την ενεργοποίηση των αντίστοιχων δέντρων στόχων που αναπαριστούν απαιτήσεις του υπό έλεγχο συστήματος και θα πρέπει να ελεγχθεί η σωστή λειτουργία τους. Στο παράδειγμά μας, χρησιμοποιήσαμε δύο δέντρα στόχων που είναι συνδεδεμένα με το συγκεκριμένο alarm. Επομένως , με την ενεργοποίηση του alarm5, ενεργοποιούνται και τα αντίστοιχα δέντρα και εισάγονται στη λίστα ενεργοποιημένων δέντρων στόχων που θα δοθεί σαν είσοδος στον SAT Solver.

Η διαφοροποίησή των δύο δέντρων αφορά τις υπολειτουργίες από τις οποίες αποτελούνται, καθώς και το είδος στρατηγικών που τις επαληθεύουν. Έτσι το πρώτο δέντρο αποτελείται από λειτουργίες οι οποίες μπορούν να ελεγχθούν με δυναμική ανάλυση, καθώς το δεύτερο δέντρο αποτελείται από λειτουργίες οι οποίες μπορούν να ελεγχθούν με στατική ανάλυση. Κατά τη διάρκεια της δυναμικής ανάλυσης , γίνεται ανάλυση των δεδομένων παρακολούθησης του ATM που έχουν δοθεί στο περιβάλλον πλαίσιο, με σκοπό την ανακάλυψη δεδομένων που υποδεικνύουν πως μία αρχική αιτία της απαίτησης έχει αποτύχει. Κατά τη διάρκεια της στατικής ανάλυσης, εκτελούνται αλγόριθμοι ελέγχου πάνω στις μεθόδους του κώδικα του ATM, ώστε να εντοπιστεί η μέθοδος που παρουσιάζει κάποιο πρόβλημα και οδηγεί στην αποτυχία της απαιτήσεως που ελέγχουμε. Θα αναφερθούμε αναλυτικά στη δομή των δέντρων αυτών στις επόμενες ενότητες.

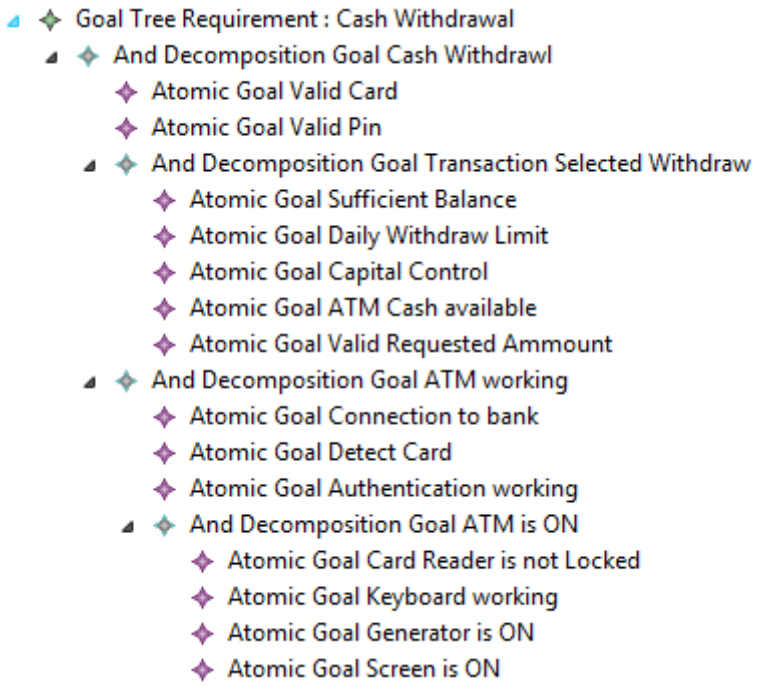
6.4.2 Δυναμικός έλεγχος

Για τις ανάγκες του παραδείγματος αυτής της ενότητας, μοντελοποιήσαμε την απαίτηση "Ανάληψη χρημάτων" ως ένα δέντρο στόχων , που έχει ως ρίζα τον κόμβο "Cash Withdrawal" . Θεωρήσαμε πως για να ικανοποιείται η απαίτηση αυτή, θα πρέπει να ικανοποιούνται οι εξής υπολειτουργίες: η κάρτα να είναι έγκυρη (Valid Card) , ο κωδικός ασφαλείας να έχει εισαχθεί σωστά (Valid PIN), η συναλλαγή που επιλέχθηκε να είναι "Ανάληψη" (Transaction selected Withdraw), και το ATM να είναι σε λειτουργία (ATM working) . Για να είναι επιτυχής η λειτουργία "Transaction selected Withdraw", θα πρέπει να ικανοποιούνται εξής υπολειτουργίες: να υπάρχει επαρκές υπόλοιπο στον λογαριασμό (Sufficient Balance), να ζητηθεί ποσό μικρότερο από το ημερήσιο όριο (Daily Withdrawal Limit), να ζητηθεί ποσό μικρότερο από το όριο κρατικού ελέγχου (Capital Control), να υπάρχουν διαθέσιμα μετρητά στο ATM (ATM Cash Available), να είναι έγκυρο το ποσό που ζητήθηκε (Valid Requested Ammount). Για να είναι επιτυχής η λειτουργία "ATM working", θα πρέπει να ικανοποιούνται εξής υπολειτουργίες : να υπάρχει σύνδεση με την τράπεζα (Connection to Bank), να έχει το ATM την δυνατότητα ανίχνευσης της κάρτας (Detect Card), να λειτουργεί η ταυτοποίηση του χρήστη (Authentication Working) και να είναι το ATM ενεργοποιημένο (ATM is ON). Για να είναι επιτυχής η λειτουργία "ATM is ON", θα πρέπει να ικανοποιούνται εξής υπολειτουργίες : να μην είναι κλειδωμένη η είσοδος κάρτας (Card Reader is not locked), να λειτουργεί το πληκτρολόγιο (Keyboard working), να είναι ενεργοποιημένη η γεννήτρια ρεύματος (Generator is ON), να είναι ενεργοποιημένη η οθόνη του ATM (Screen is ON). Έτσι προκύπτει το παρακάτω δέντρο στόχων, σε UML μορφή .



Σχήμα 6.8: Δέντρο στόχων για την απαίτηση “Cash Withdrawal”

Ακολουθεί η μορφή του δέντρου στόχων, όπως φαίνεται στο EMF.

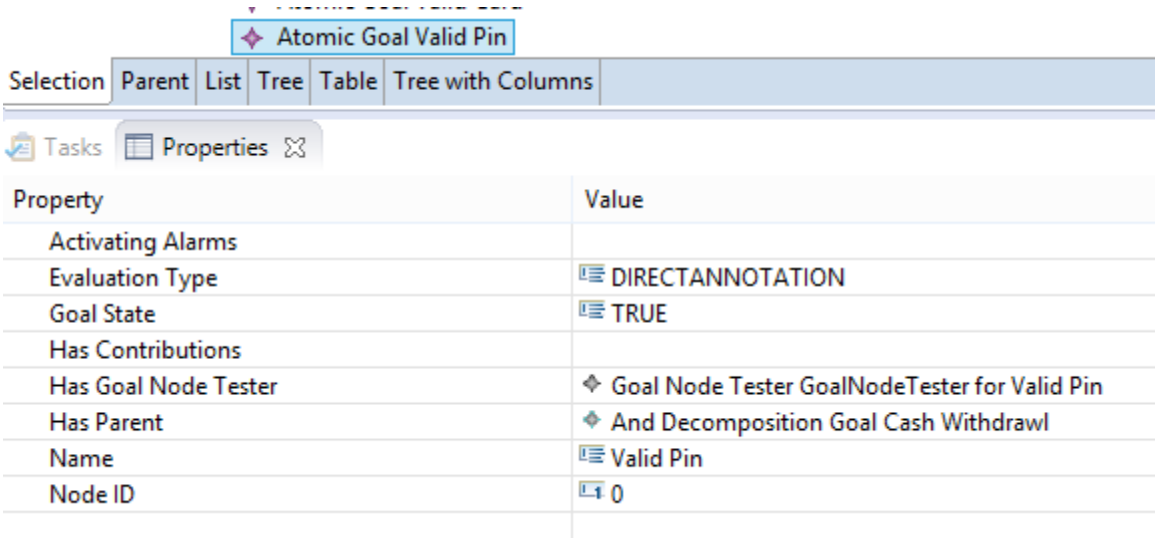


Σχήμα 6.9: Δέντρο στόχων για την απαίτηση “Cash Withdrawal” με χρήση EMF

Όπως αναφέραμε παραπάνω, ένας κόμβος που δεν μπορεί να αποδειχθεί άμεσα είναι κάποιος ενδιάμεσος κόμβος, η αληθοτιμή του οποίου εξαρτάται από τις αληθοτιμές των κόμβων παιδιών του. Στο δέντρο στόχων των παραπάνω σχημάτων λοιπόν, τέτοιου είδους κόμβοι είναι οι "Cash Withdrawal", "Transaction selected Withdraw", "ATM working", καθώς για την ικανοποίησή τους, πρέπει να ικανοποιούνται όλοι οι κόμβοι παιδιά τους. Οι κόμβοι που έχουν αυτή την ιδιότητα, περιέχουν την τιμή "PSEUDOALARM" στο πεδίο "Evaluation type".
Στα πλαίσια αυτού του παραδείγματος, έχουμε θεωρήσει κάποιους κόμβους-φύλλα του δέντρου στόχων ως άμεσα επαληθεύσιμους. Αυτό σημαίνει πως έχουμε αποφασίσει από πριν τις αληθοτιμές των φύλλων ως ψευδείς ή αληθείς. Οι κόμβοι που έχουν αυτή την ιδιότητα, περιέχουν την τιμή "DIRECT" στο πεδίο "Evaluation type". Άμεσα επαληθεύσιμοι κόμβοι στο παράδειγμά μας είναι οι εξής: Valid Card, Daily Withdraw Limit, Capital Control, ATM Cash available, Valid requested Amount, Connection to Bank, Detect Card, Authentication working, Card Reader not locked, Keyboard working, Generator is ON, Screen is ON.

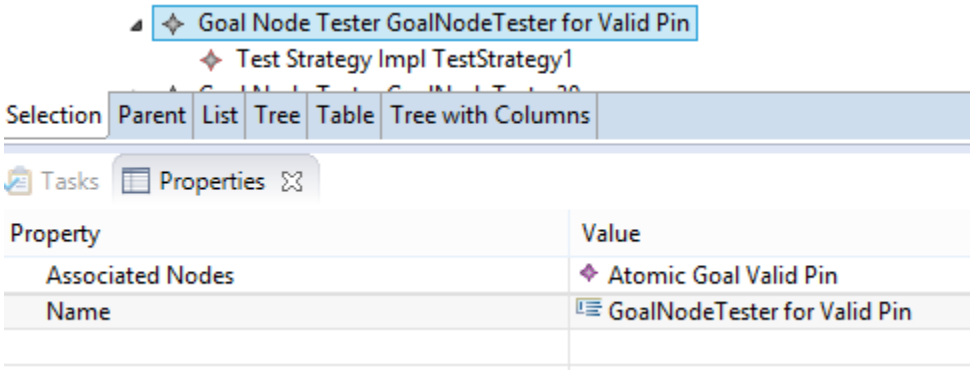
Οι κόμβοι "Sufficient Balance" και "Valid PIN" περιέχουν την τιμή "DIRECTANNOTATION" στο πεδίο "Evaluation type". Αυτό σημαίνει πως οι κόμβοι αυτοί περιέχουν μία επισημείωση που υποδεικνύει πως ο έλεγχος της ισχύος τους απαιτεί την εκτέλεση μίας ή περισσοτέρων στρατηγικών ελέγχου που είναι συνδεδεμένοι μαζί τους.
Στην περίπτωσή μας, οι στρατηγικές ελέγχου αποτελούν μία δυναμική ανάλυση των δεδομένων παρακολούθησης του ATM, για την αναζήτηση στοιχείων που επαληθεύουν ή απορρίπτουν τον κόμβο. Τα δεδομένα αυτά θα πρέπει να περιέχουν λεπτομέρειες για όλες τις ενέργειες και καταστάσεις του συστήματος κατά τη διάρκεια λειτουργίας του. Ιδιαίτερη βάση θα πρέπει να δοθεί στις πληροφορίες εκτέλεσης οι οποίες αναφέρονται σε κατάσταση λανθασμένης λειτουργίας του συστήματος. Τα δεδομένα περιγραφής της εκτέλεσης αυτού του υπό έλεγχο συστήματος θα πρέπει να συγκριθούν με το αναμενόμενο αποτέλεσμα ορθής λειτουργίας. Στην περίπτωση που τα δεδομένα παρακολούθησης περιέχουν ενδείξεις σύμφωνες με το αναμενόμενο αποτέλεσμα, τότε η στρατηγική θεωρείται επιτυχής και ο αντίστοιχος κόμβος λαμβάνει αληθή τιμή. Στην περίπτωση που τα δεδομένα παρακολούθησης περιέχουν ενδείξεις που έρχονται σε αντίθεση με το αναμενόμενο αποτέλεσμα, τότε η το αποτέλεσμα της στρατηγικής που εκτελείται θεωρείται ανεπιτυχές και ο αντίστοιχος κόμβος λαμβάνει ψευδή τιμή.

Ο κόμβος "Valid PIN" είναι συνδεδεμένος με τον GoalNodeTester "GoalNodeTester for Valid PIN", όπως φαίνεται στο σχήμα :



Σχήμα 6.10: Σύνδεση κόμβου «Valid PIN» με τον GoalNodeTester «GoalNodeTester for Valid PIN»

Ο GoalNodeTester "GoalNodeTester for Valid PIN" αποτελείται από μία στρατηγική, την "TestStrategy1", όπως μπορούμε να δούμε στο παρακάτω σχήμα:



Σχήμα 6.11: Σύνδεση GoalNodeTester « GoalNodeTester for Valid PIN » με το Strategy «TestStrategy1»

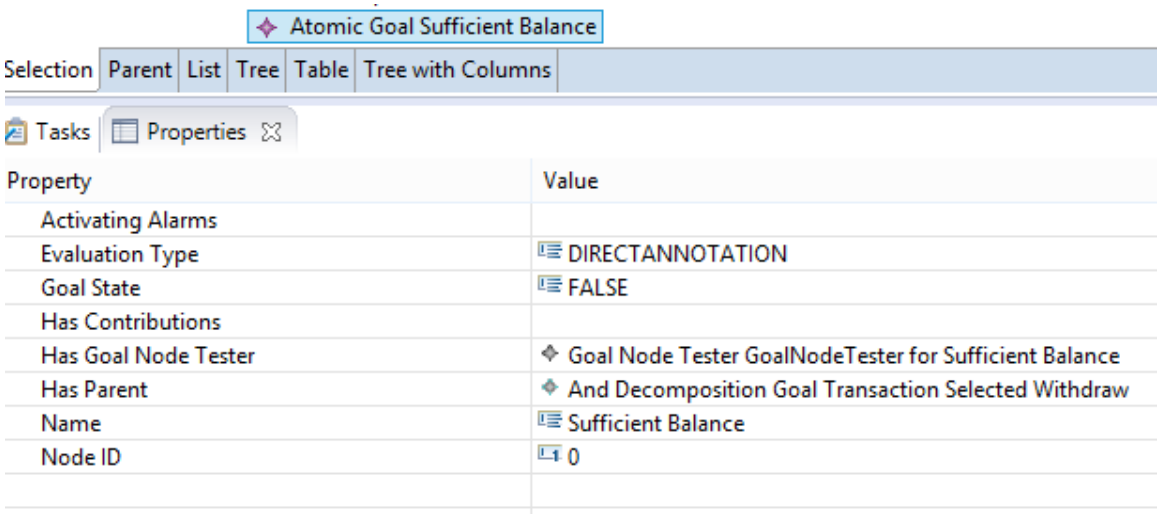
Στόχος αυτής της στρατηγικής, είναι η ανάλυση των δεδομένων παρακολούθησης που έχουν συλλεχθεί κατά τη διάρκεια λειτουργίας του ATM, και η αναζήτηση κάποιου στοιχείου που υποδηλώνει πως ο χρήστης έχει χρησιμοποιήσει λανθασμένο κωδικό. Στην περίπτωση που έχει συμβεί κάτι τέτοιο, αναμένουμε να δούμε στα δεδομένα παρακολούθησης το μήνυμα "Incorrect PIN". Εάν το μήνυμα αυτό βρεθεί μέσα στα δεδομένα παρακολούθησης, τότε θεωρούμε πως ο χρήστης έχει χρησιμοποιήσει λανθασμένο κωδικό, και η στρατηγική αποτυγχάνει. Αυτό σημαίνει πως ο συγκεκριμένος συνδυασμός κόμβων που ελέγχεται, δεν ικανοποιεί την αρχική απαίτηση "Επιτυχής ανάληψη χρημάτων" και η αρχική αιτία είναι το ότι η υπολειτουργία "Valid PIN" δεν

ικανοποιείται.
Την εκτέλεση της στρατηγικής αναλαμβάνει η μέθοδος DO(), όπως έχουμε αναφέρει παραπάνω. Η μέθοδος αυτή επιστρέφει στο περιβάλλον πλαίσιο την αληθοτιμή "true" εάν ικανοποιείται, δηλαδή εάν η υπόθεση "Valid PIN" αποδειχτεί αληθής. Παρομοίως, επιστρέφει την τιμή "false" όταν η υπόθεση "Valid PIN" αποδειχτεί ψευδής, με αποτέλεσμα να σταματάει την διαδικασία ελέγχου του τρέχοντος συνδυασμού κόμβων. Ο αλγόριθμος που εκτελεί η μέθοδος DO() για να ελέγξει τον κόμβο "Valid PIN" φαίνεται στο παρακάτω σχήμα:

```
public boolean DO() {
    boolean result = true;
    File file = new File("C:\\Users\\Xenia\\GP Model - Run SAT - add ID\\satGoalModel\\alarmer\\alarmer

    System.out.println("Checking if PIN was correctly inserted......");
    try {
        Scanner scanner = new Scanner(file);
        //now read the file line by line...
        int lineNum = 0;
        List<String> lines_list = new ArrayList<>();
       // lineNum = 0;
        while (scanner.hasNextLine()) {
            String line = scanner.nextLine();
            lines_list.add(line.trim());
            lineNum++;
            if (line.contains("Incorrect PIN")) {
                System.out.println("Problem found. Incorrect PIN inserted. ");
                result = false;
             }

         }

    } catch(IOException e) {
        // ... handle errors ...
    } finally {
        // ... cleanup that will execute whether or not an error occurred ...
    }
    if (result == true) {
        System.out.println("No Error detected. PIN was correctly inserted.");
     }
    return result;
```

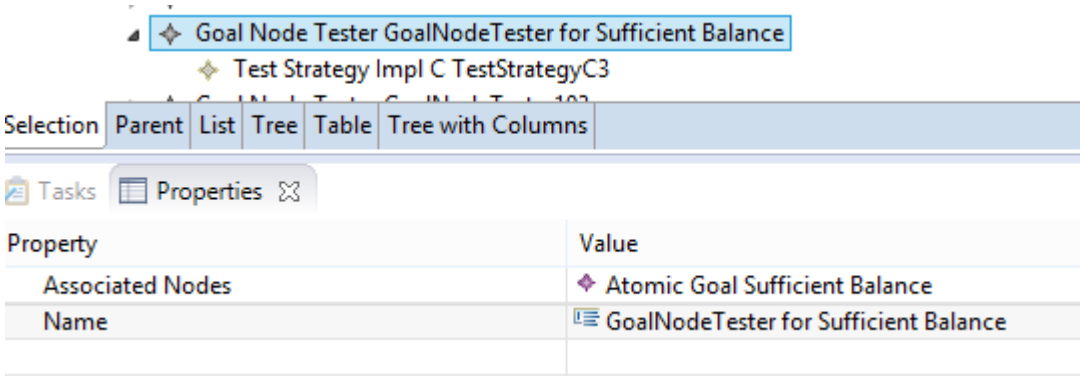
Σχήμα 6.12: Η μέθοδος DO() για τον κόμβο "Valid PIN"

Ο κόμβος "Sufficient Balance" είναι συνδεδεμένος με τον GoalNodeTester "GoalNodeTester for Sufficient Balance", όπως φαίνεται στο σχήμα :



Σχήμα 6.13: Σύνδεση κόμβου « Sufficient Balance » με τον GoalNodeTester «GoalNodeTester for Sufficient Balance »

Ο GoalNodeTester "GoalNodeTester for Sufficient Balance" αποτελείται από μία στρατηγική, την "TestStrategyC3", όπως μπορούμε να δούμε στο παρακάτω σχήμα:



Σχήμα 6.14: Σύνδεση GoalNodeTester « GoalNodeTester for Sufficient Balance » με το Strategy «TestStrategyC3»

Στόχος αυτής της στρατηγικής , είναι η ανάλυση των δεδομένων παρακολούθησης που έχουν συλλεχθεί κατά τη διάρκεια λειτουργίας του ATM, και η αναζήτηση κάποιου στοιχείου που υποδηλώνει πως δεν υπάρχει επαρκές υπόλοιπο λογαριασμού, δηλαδή ο λόγος που η αρχική απαίτηση "Επιτυχής Ανάληψη χρημάτων" έχει αποτύχει είναι το γεγονός ότι ο χρήστης έχει ζητήσει περισσότερα μετρητά από το διαθέσιμο υπόλοιπό του. Στην περίπτωση που έχει συμβεί κάτι τέτοιο, αναμένουμε να δούμε στα δεδομένα παρακολούθησης το μήνυμα "Insufficient available balance". Εάν το μήνυμα αυτό βρεθεί μέσα στα δεδομένα παρακολούθησης, τότε θεωρούμε πως δεν υπάρχει επαρκές υπόλοιπο στον λογαριασμό του χρήστη, και η στρατηγική

αποτυγχάνει. Αυτό σημαίνει πως ο συγκεκριμένος συνδυασμός κόμβων που ελέγχεται, δεν ικανοποιεί την αρχική απαίτηση "Επιτυχής ανάληψη χρημάτων" και η αρχική αιτία είναι το ότι η υπολειτουργία "Sufficient balance" δεν ικανοποιείται.
Την εκτέλεση της στρατηγικής TestStrategyC3 αναλαμβάνει η μέθοδος DO(). Η μέθοδος αυτή επιστρέφει στο περιβάλλον πλαίσιο την αληθοτιμή "true" εάν ικανοποιείται, δηλαδή εάν η υπόθεση "Sufficient balance" αποδειχτεί αληθής. Παρομοίως, επιστρέφει την τιμή "false" όταν η υπόθεση "Sufficient balance" αποδειχτεί ψευδής, με αποτέλεσμα να σταματάει την διαδικασία ελέγχου του τρέχοντος συνδυασμού κόμβων. Ο αλγόριθμος που εκτελεί η μέθοδος DO() για να ελέγξει τον κόμβο "Sufficient balance" φαίνεται στο παρακάτω σχήμα:

```
@Override
public boolean DO() {
    boolean result = true;
    File file = new File("C:\\Users\\Xenia\\GP Model - Run SAT - add ID\\satGoalModel\\alarmer\\alarmer\\ATMLo

    System.out.println("Checking if there is sufficient available balance in your account......");
    try {
        Scanner scanner = new Scanner(file);
        //now read the file line by line...
        int lineNum = 0;
        List<String> lines_list = new ArrayList<>();
        while (scanner.hasNextLine()) {
            String line = scanner.nextLine();
            lines_list.add(line.trim());
            lineNum++;
            if (line.contains("Insufficient available balance")) {

                System.out.println("Problem found. There is Insufficient available balance in your account");
                result = false;
             }

         }
    } catch(IOException e) {
        // ... handle errors ...
    } finally {
        // ... cleanup that will execute whether or not an error occurred ...
    }

    if (result == true) {
        System.out.println("No Error detected. There is sufficient available balance in your account.");
     }
    return result;
} //TestStrategyImplImpl
```

Σχήμα 6.15: Η μέθοδος DO() για τον κόμβο " Sufficient Balance"

Μετά την ενεργοποίηση του Alarm5 που περιγράψαμε στην ενότητα 6.4.1, επιλέγεται το πρώτο δέντρο της λίστας των ενεργοποιημένων δέντρων στόχων, το οποίο είναι το "Cash Withdrawal" και δίνεται σαν είσοδος στον SAT Solver. Αυτό φαίνεται στο παρακάτω σχήμα (σχήμα 6.16)

```
Main Started
Alarm: Alarm5 was found on line: 4: FAILURE ERROR! Alarm5 occurred. Sorry for the inconvenience
Event1: Sun Sep 20 16:27:23 EEST 2015: <<Alarmer>> Alarm Alarm5 was activated - (Associated Goals: Cash Withdrawl)
Event2: Sun Sep 20 16:27:23 EEST 2015: <<GoalSelector>> Generates solutions for goal Cash Withdrawl  using sat solver and updates evaluation queue
```

Σχήμα 6.16: Ενεργοποίηση του alarm “Alarm5” και του δέντρου στόχων “Cash Withdraw”

Ακολουθεί η μετατροπή του δέντρου στόχων σε έκφραση CNF, με τον τρόπο που περιγράψαμε

αναλυτικά στο κεφάλαιο 5. Για να γίνει αυτό εφικτό, είναι απαραίτητη η πρόσβαση του component SAT Solver στο μοντέλο του δέντρου στόχων που έχει πάρει ως είσοδο. Επίσης είναι απαραίτητη η πρόσβαση στη βάση (Blackboard) που διατηρεί τις αληθοτιμές των ήδη αποδεδειγμένων κόμβων, βάση των οποίων λαμβάνει αποφάσεις για την παραγωγή συνδυασμών κόμβων που επαληθεύουν το δέντρο, όπως εξηγήσαμε αναλυτικά στο κεφάλαιο 5. Τα δεδομένα αυτά θα ληφθούν υπόψη για την παραγωγή N λύσεων από τον SAT4j, έτσι ώστε να περιοριστεί ο χρόνος εκτέλεσης του SAT4j και να έχουμε μια καλή λύση σε μικρό χρονικό διάστημα. Η έκφραση CNF θα δοθεί στον SAT4j για την εξαγωγή των πιθανών συνδυασμών αρχικών αιτιών , δηλαδή των κόμβων που πρέπει να ελεγχθούν ώστε να αποφανθεί η ισχύς της αρχικής απαίτησης. Έξοδος του SAT4j είναι μία λίστα συνδυασμών κόμβων που ικανοποιούν το δέντρο στόχων. Εάν η λίστα δεν είναι κενή ,τότε ταξινομείται βάση της στρατηγικής του SAT4j που εξηγήσαμε στο κεφάλαιο 5 και επιχειρείται ο έλεγχος του πρώτου συνδυασμού.
Ο πρώτος συνδυασμός κόμβων που προκύπτει και εμφανίζεται σαν έξοδος του πλαισίου σαν αναφορά, είναι ο εξής :

```
Event2: Sun Sep 20 15:46:28 EEST 2015: <<GoalSelector>> Generates solutions for goal Cash Withdrawl  using sat solver and updates evaluation queue
Event3: Sun Sep 20 15:46:29 EEST 2015: <<Evaluator>> tries to prove goal Cash Withdrawl by proving the following goals:
                                        Capital Control
                                        Screen is ON
                                        ATM Cash available
                                        Detect Card
                                        Generator is ON
                                        Connection to bank
                                        Valid Pin
                                        Valid Requested Ammount
                                        Daily Withdraw Limit
                                        Card Reader is not Locked
                                        Valid Card
                                        Keyboard working
                                        Sufficient Balance
                                        Authentication working
                                        Transaction Selected Withdraw
                                        ATM is ON
                                        ATM working
```

Σχήμα 6.17: Συνδυασμός κόμβων προς έλεγχο

Στη συνέχεια, ο Evaluator θα προσπαθήσει να αποδείξει με τη σειρά κάθε ένα από τους κόμβους του συνδυασμού αυτού. Ξεκινάει λοιπόν με την εξής σειρά:
"Capital Control", "Screen is ON", "ATM Cash available", "Detect Card", "Generator is ON", "Connection to bank". Όπως αναφέραμε και πριν, οι κόμβοι αυτοί είναι άμεσα επαληθεύσιμοι και η κατάστασή τους είναι "TRUE". Επομένως , ο Evaluator αποδεικνύει εύκολα την αληθοτιμή τους όπως φαίνεται στο επόμενο σχήμα, και προχωράει στον επόμενο κόμβο του τρέχοντος συνδυασμού κόμβων που εξετάζει.

```
Event4: Sun Sep 20 15:46:29 EEST 2015: <<Evaluator>> to prove goal Cash Withdrawl proves goal Capital Control true
Event5: Sun Sep 20 15:46:29 EEST 2015: <<Evaluator>> to prove goal Cash Withdrawl proves goal Screen is ON true
Event6: Sun Sep 20 15:46:29 EEST 2015: <<Evaluator>> to prove goal Cash Withdrawl proves goal ATM Cash available true
Event7: Sun Sep 20 15:46:29 EEST 2015: <<Evaluator>> to prove goal Cash Withdrawl proves goal Detect Card true
Event8: Sun Sep 20 15:46:29 EEST 2015: <<Evaluator>> to prove goal Cash Withdrawl proves goal Generator is ON true
Event9: Sun Sep 20 15:46:29 EEST 2015: <<Evaluator>> to prove goal Cash Withdrawl proves goal Connection to bank true
```

Σχήμα 6.18: Έλεγχος άμεσα επαληθεύσιμων κόμβων

Ο επόμενος κόμβος που πρέπει να αποδειχθεί είναι ο "Valid PIN" . Ο συγκεκριμένος κόμβος αποδεικνύεται με την εκτέλεση της στρατηγικής που ανήκει στον "GoalNodeTester for Valid PIN". Το αποτέλεσμα της στρατηγικής αυτής είναι "true", και φαίνεται στο παρακάτω σχήμα:

```
Event10: Sun Sep 20 15:46:29 EEST 2015: <<Evaluator>> requests additional data for Valid Pin
Event11: Sun Sep 20 15:46:29 EEST 2015: <<GoalNodeVerifier>> Running additional tests for goal Valid Pin
Event12: Sun Sep 20 15:46:29 EEST 2015: <<GoalNodeVerifier>> Starting TestDriver for Valid Pin
Running Goal Node Tester :GoalNodeTester for Valid Pin
Running strategygr.ntua.softeng.goalmodel.impl.TestStrategyImplImpl@3311b113 (name: TestStrategy1, ID: 1)
Checking if PIN was correctly inserted......
No Error detected. PIN was correctly inserted.
Event13: Sun Sep 20 15:46:29 EEST 2015: <<Evaluator>> to prove goal Cash Withdrawl proves goal Valid Pin true
```

Σχήμα 6.19: Έλεγχος κόμβου "Valid PIN" συνδεδεμένου με στρατηγική

Ο Evaluator συνεχίζει με την επαλήθευση των επόμενων στόχων , που είναι οι "Valid Requested Ammount", "Daily Withdraw Limit", "Card Reader is not Locked", "Valid Card", "Keyboard working", και είναι άμεσα επαληθεύσιμοι με κατάσταση "true". Η αναφορά αποτελεσμάτων φαίνεται στο επόμενο σχήμα:

```
Event13: Sun Sep 20 15:46:29 EEST 2015: <<Evaluator>> to prove goal Cash Withdrawl proves goal Valid Pin true
Event14: Sun Sep 20 15:46:29 EEST 2015: <<Evaluator>> to prove goal Cash Withdrawl proves goal Valid Requested Ammount true
Event15: Sun Sep 20 15:46:29 EEST 2015: <<Evaluator>> to prove goal Cash Withdrawl proves goal Daily Withdraw Limit true
Event16: Sun Sep 20 15:46:29 EEST 2015: <<Evaluator>> to prove goal Cash Withdrawl proves goal Card Reader is not Locked true
Event17: Sun Sep 20 15:46:30 EEST 2015: <<Evaluator>> to prove goal Cash Withdrawl proves goal Valid Card true
Event18: Sun Sep 20 15:46:30 EEST 2015: <<Evaluator>> to prove goal Cash Withdrawl proves goal Keyboard working true
```

Σχήμα 6.20: Έλεγχος των υπολοίπων άμεσα επαληθεύσιμων κόμβων

Ο επόμενος κόμβος που πρέπει να αποδειχθεί είναι ο "Sufficient Balance" . Ο συγκεκριμένος κόμβος αποδεικνύεται με την εκτέλεση της στρατηγικής που ανήκει στον "GoalNodeTester for Sufficient Balance". Το αποτέλεσμα της στρατηγικής αυτής είναι "false", και φαίνεται στο παρακάτω σχήμα:

```
Event19: Sun Sep 20 15:46:30 EEST 2015: <<Evaluator>> requests additional data for Sufficient Balance
Event20: Sun Sep 20 15:46:30 EEST 2015: <<GoalNodeVerifier>> Running additional tests for goal Sufficient Balance
Event21: Sun Sep 20 15:46:30 EEST 2015: <<GoalNodeVerifier>> Starting TestDriver for Sufficient Balance
Running Goal Node Tester :GoalNodeTester102
Running strategygr.ntua.softeng.goalmodel.impl.TestStrategyImplCImpl@832b065 (name: TestStrategyC3, ID: 3)
Checking if there is sufficient available balance in your account......
Problem found. There is Insufficient available balance in your account
One TestStrategy failed, so the Node Sufficient Balance is false!
Event22: Sun Sep 20 15:46:30 EEST 2015: <<Evaluator>> to prove goal Cash Withdrawl proves goal Sufficient Balance false. Evaluation of Cash Withdrawl fails
```

Σχήμα 6.21: Έλεγχος κόμβου "Sufficient Balance"συνδεδεμένου με στρατηγική

Αυτό σημαίνει πως η στρατηγική για την επαλήθευση του κόμβου "Sufficient Balance" απέτυχε, επομένως η αρχική αιτία της αποτυχίας της απαίτησης "Επιτυχής Ανάληψη χρημάτων" είναι το γεγονός ότι δεν υπάρχει επαρκές υπόλοιπο στο λογαριασμό του χρήστη. Ο συνδυασμός κόμβων που εξετάστηκε έχει αποτύχει και συνεπώς αφαιρείται από την λίστα και συνεχίζουμε με τον επόμενο συνδυασμό που είναι πλέον στην αρχή της λίστας. Στην περίπτωσή μας, η λίστα είναι κενή , δηλαδή υπάρχει μόνο ένας συνδυασμός που μπορεί να επαληθεύσει το δέντρο στόχων. Επόμενο βήμα είναι να ξανατρέξουμε τον SAT4j με την νέα βάση γνώσης για να μας δώσει ένα νέο σύνολο λύσεων. Η διαδικασία σταματάει καθώς δεν επιστρέφεται λύση από τον SAT4j.

Η διαδικασία επαλήθευσης σταματάει με το μήνυμα "Evaluation of Cash Withdrawl fails." . Η απαίτηση έχει αποτύχει και η αρχική αιτία έχει βρεθεί και είναι η "Sufficient Balance", η οποία δεν ικανοποιείται.

6.4.3 Στατικός έλεγχος

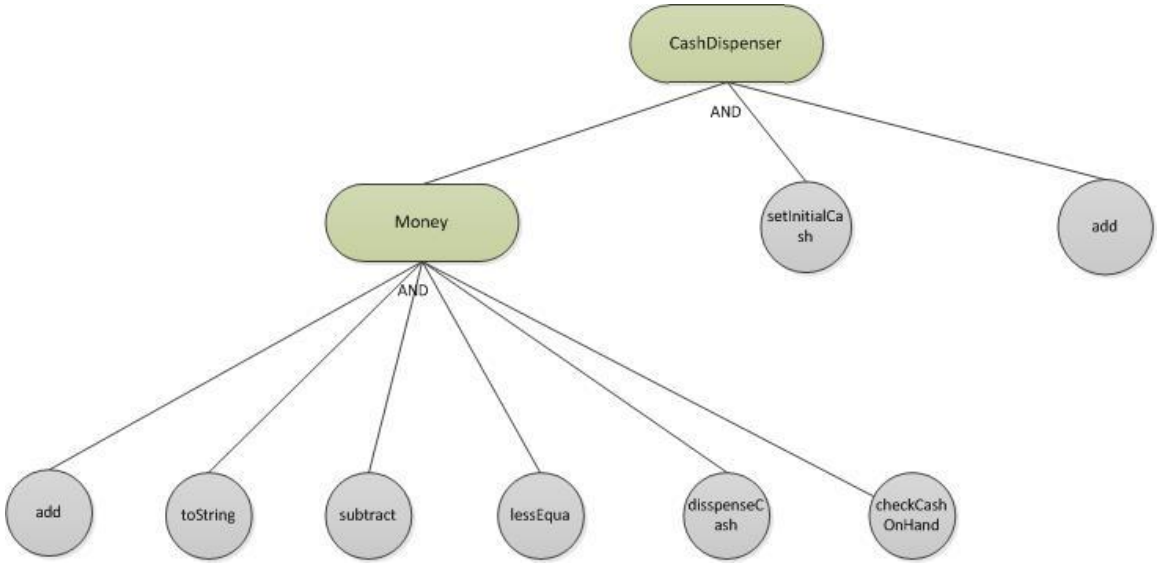
Στην ενότητα αυτή περιγράφεται ο τρόπος ενεργοποίησης και η εφαρμογή της στατικής ανάλυσης στο παράδειγμα του ATM που παρουσιάστηκε παραπάνω.

Για τις ανάγκες τις στατικής ανάλυσης δημιουργούμε ένα νέο μοντέλο δέντρων στόχων το οποίο περιλαμβάνει βασικούς κόμβους-απαιτήσεις που πρέπει να ικανοποιεί η προσομοίωση του ATM. Οι απαιτήσεις που συμπεριλαμβάνονται στο δέντρο αυτό είναι εκείνες που μπορούν να ελεγχθούν με στατικό έλεγχο, δηλαδή με άμεσο έλεγχο των μεθόδων του κώδικα . Τέτοιες απαιτήσεις-λειτουργίες της προσομοίωσης του ATM μπορεί να είναι , ο έλεγχος υπολοίπου , ο έλεγχος ημερήσιου ορίου ανάληψης , ο έλεγχος κατάθεσης κ.α. σε σύγκριση πάντα με το εισαγόμενο ποσό που επιλέγει ο χρήστης. Οι ορθότητα των λειτουργιών αυτών μπορεί άμεσα να ελεγχθεί εξετάζοντας τις μεθόδους του κώδικα του ATM οι οποίες υλοποιούν τις λειτουργίες αυτές. Έτσι για παράδειγμα προκειμένου να εξετάσουμε τη λειτουργία επιτυχούς κατάθεσης , θα έπρεπε να εξετάσουμε όλες τις μεθόδους που πραγματοποιούν σύγκριση , πρόσθεση και έλεγχο υπολοίπου. Η περίπτωση χρήσης που πραγματοποιήθηκε στα πλαίσια της διπλωματικής μας εργασίας εστιάζει στον έλεγχο της διαδικασίας ελέγχου υπολοίπου ("sufficient balance") . Στο σημείο αυτό θα πρέπει να σημειώσουμε πως η λειτουργία αυτή η οποία αποτελεί βασικό στόχο στο δέντρο στατικής ανάλυσης , αποτελούσε μια υπολειτουργία του βασικού στόχου "ανάληψη" (cash withdrawal) του δέντρου δυναμικής ανάλυσης. Η λειτουργία αυτή καθώς και όλες οι υπολειτουργίες του δέντρου δυναμικής ανάλυσης "Cash withdrawal" , οι οποίες αποτελούν βασικούς στόχους στο μοντέλο στατικής ανάλυσης , συνδέονται με το ίδιο Alarm ενεργοποίησης. Έτσι, στην περίπτωση αποτυχίας της υπόθεσης "Cash withdrawal" (η οποία ενεργοποιήθηκε λόγο της εμφάνισης του Alarm5, υπάρχει η δυνατότητα ενεργοποίησης στατικού ελέγχου για όλους τους υπό κόμβους του δέντρου "Cash withdrawal" , απλά ενεργοποιώντας το ίδιο Alarm (Alarm5) για το μοντέλου στατικού ελέγχου. Η διαδικασία αυτή περιγράφεται αναλυτικότερα παρακάτω.

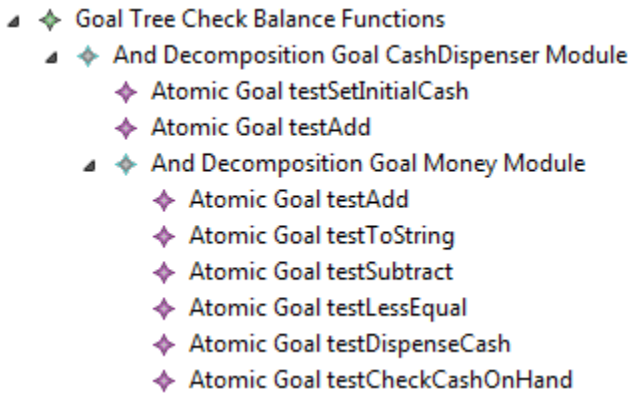
Οι μέθοδοι του κώδικα οι οποίοι θα πρέπει να ελεγχθούν προκειμένου να έχουμε μια σαφή εικόνα για την ορθή λειτουργία η όχι της διαδικασίας του ελέγχου υπολοίπου ("sufficient balance") είναι οι παρακάτω:

- **CashDispenser**
  Αποτελεί την βασική κλάση του κώδικα η οποία αναλαμβάνει των υπολογισμό του υπολοίπου , την αφαίρεση και πρόσθεση χρημάτων καθώς και την σύγκριση ποσών.
- Εξαρτάται από την κλάση Money και τις μεθόδους setInitialCash και add.
- **setInitialCash**
  Είναι η μέθοδος στην οποία ορίζεται το αρχικό υπόλοιπο του χρήστη.
- **add**
  Είναι η μέθοδος η οποία υλοποιεί την πρόσθεση χρημάτων στο διαθέσιμο υπόλοιπο.
- **Money**
  Αποτελεί την κλάση στην οποία γίνεται ο υπολογισμός των χρημάτων . Εξαρτάται από τις μεθόδους : toString, LessEqual, DispenseCash και cashOnHand.
- **toString**
  Στη μέθοδο αυτή μετατρέπεται το ποσό των χρημάτων σε συμβολοσειρά.
- **LessEqual**
  Η μέθοδος αυτή πραγματοποιεί σύγκριση ποσών.
- **DispenseCash**
- Είναι η μέθοδος η οποία υλοποιεί την αφαίρεση χρημάτων από το διαθέσιμο υπόλοιπο.
- **cashOnHand**
  Πραγματοποιεί τον έλεγχο του διαθέσιμου υπολοίπου.

Ένα από τα δέντρα στόχων του μοντέλου δυναμικής ανάλυσης που προκύπτει σύμφωνα με όσα περιγράφηκαν παραπάνω φαίνεται στο σχήμα (σχήμα 6.8) . Η σχηματική αναπαράσταση αυτού στο περιβάλλον EMF του eclipse, φαίνεται στο σχήμα (σχήμα 6.22)



Σχήμα 6.22: Δέντρο στατικής ανάλυσης



Σχήμα 6.23: Στιγμιότυπο δέντρου στατικής ανάλυσης στο περιβάλλον eclipse EMF.

Δεδομένου ότι κατά το αποτέλεσμα της στατικής ανάλυσης έδειξε πως ο υπό-κόμβος "Sufficient Balance" ευθύνεται για την αποτυχία της ανάληψης χρημάτων ενεργοποιούμε στο μοντέλο του στατικού ελέγχου το Alarm το οποίο συνδέεται με τον έλεγχο του κόμβου αυτού. Δίνουμε ως είσοδο λοιπόν στο περιβάλλον πλαίσιο που αναπτύχθηκε το νέο μοντέλο στατικού ελέγχου και το alarm ενεργοποίησης του συγκεκριμένου δέντρου. Η διαδικασία που ακολουθεί είναι η μετατροπή του μοντέλου σε CNF μορφή και η επίλυση του προβλήματος SAT που προκύπτει. Έχουμε σε αυτό το σημείο ένα σύνολο συνδυασμών κόμβων του δέντρου στατικής ανάλυσης για τους οποίους πρέπει να ελεγχθεί η αληθοτιμή τους. Σημειώνουμε στο σημείο αυτό ότι στο μοντέλο στατικής ανάλυσης που ο χρήστης έδωσε ως είσοδο στο σύστημα , η αληθοτιμή των κόμβων setInitialCash, add, toString, subtract και dispenseCash ήταν προκαθορισμένη ως αληθής. Αυτό είναι επιθυμητό καθώς ο χρήστης μπορεί να γνωρίζει την ορθή λειτουργία τμημάτων του κώδικα και να επιθυμεί να αποφύγει τον επανέλεγχο τους. Σε αντίθεση με τους κόμβους αυτούς οι κόμβοι lessEqual και checCashOnHand έχουν χαρακτηριστεί ως κόμβοι οι οποίοι απαιτούν περαιτέρω έλεγχο για την εξακρίβωση ή όχι της ορθής λειτουργίας τους. Στόχος μας λοιπόν στο σημείο αυτό είναι η εύρεση της αληθοτιμής των δύο κόμβων αυτών. Αυτό επιτυγχάνεται με την εκτέλεση των σεναρίων ελέγχου που βρίσκονται στην στρατηγική επαλήθευσης του κάθε κόμβου. Τα σενάρια ελέγχου αυτά αποτελούν σενάρια JUnit.

Ένα σενάριο JUnit καθορίζει ένα σύνολο αρχικών τιμών και ένα αναμενόμενο αποτέλεσμα στην περίπτωση όπου αυτές οι αρχικές τιμές δοθούν ως είσοδο σε μια μέθοδο του κώδικα, υπό την προϋπόθεση πάντα ότι το σύστημα βρίσκεται σε κατάσταση ορθής λειτουργίας. Έτσι προκειμένου να ελεγχθεί για παράδειγμα η μέθοδος lessEqual , το σενάριο JUnit δίνει ως είσοδο στη μέθοδο τις αρχικές τιμές 20 και 10 . Το αναμενόμενο αποτέλεσμα καθορίζεται ως αληθείς καθώς αποτελεί το αποτέλεσμα της σύγκρισης $10 < 20$. Οι αρχικές τιμές αυτές δίνονται στη μέθοδο και εξετάζεται το αποτέλεσμα που επιστράφηκε. Όταν η μέθοδος lessEqual επιστρέψει το αποτέλεσμα για αυτές τις τιμές αληθοτιμών, πραγματοποιείται σύγκριση με το αναμενόμενο

αποτέλεσμα του σεναρίου JUnit. Για παράδειγμα αν η μέθοδος lessEqual επιστρέψει ως αποτέλεσμα πως η έκφραση 10 < 20 είναι λανθασμένη, πραγματοποιείται σύγκριση με το αναμενόμενο αποτέλεσμα, το οποίο προφανώς στην περίπτωση μας είναι ότι η έκφραση αυτή είναι αληθής και αν η τα δύο αυτά αποτελέσματα δεν είναι ίδια , τότε η λειτουργία της μεθόδου χαρακτηρίζεται ως ελαττωματική. Το σενάριο ελέγχου Junit επιστρέφει στο σύστημα μας την πληροφορία αυτή ενημερώνοντας το σύστημα για την δυσλειτουργία της μεθόδου lessEqual.

Το παράδειγμα σεναρίου ελέγχου JUnit που περιγράφθηκε φαίνεται στο παρακάτω σχήμα(σχήμα) :

```
/**
 * Test of lessEqual method, of class Money.
 */
@Test
public void testLessEqual() {
    System.out.println("lessEqual");
    Money compareTo = new Money(20);
    Money instance = new Money(10);
    boolean expResult = true;
    boolean result = instance.lessEqual(compareTo);
    assertEquals(expResult, result);

}
```

Σχήμα 6.24: Σενάριο ελέγχου JUnit για τη μέθοδο lessEqual

Ο κώδικας του σεναρίου ελέγχου με τον οποίο καλέστηκε το παραπάνω σενάριο ελέγχου JUnit φαίνεται στο σχήμα (σχήμα 6.25) :

```
@Override
public boolean DO() {
    // TODO Auto-generated method stub
    System.out.println("Testing function lessEqual with JUnit");

    Class loadedMyClass = null;

    ClassLoader load=this.getClass().getClassLoader();
    try {
     loadedMyClass = load.loadClass("banking.MoneyTest");
    } catch (Exception ex)
    {
     System.out.println(ex.toString());
    }

     Result result = JUnitCore.runClasses(loadedMyClass);
        if (result.wasSuccessful() == true) {
            System.out.println("No Error detected. lessEqual function works correctly.\n");

         }  ;
            if (result.wasSuccessful() == false) {
                System.out.println("ERROR!. lessEqual function is not working correctly.\n");

             }  ;

   return result.wasSuccessful();

}
```

Σχήμα 6.25: Πηγαίος κώδικας κλήσης σεναρίου ελέγχου JUnit

Η έξοδος του παραβάλλον πλαισίου για το παράδειγμα που περιγράψαμε φαίνεται στο σχήμα (σχήμα 6.26):

```
Main Started
Alarm: Alarm5 was found on line: 4: FAILURE ERROR! Alarm5 occurred. Sorry for the inconvenience
Event1: Sun Sep 20 13:38:13 EEST 2015: <<Alarmer>> Alarm Alarm5 was activated - (Associated Goals: CashDispenser Module)
Event2: Sun Sep 20 13:38:13 EEST 2015: <<GoalSelector>> Generates solutions for goal CashDispenser Module  using sat solver
                                                         and updates evaluation queue
Event3: Sun Sep 20 13:38:13 EEST 2015: <<Evaluator>> tries to prove goal CashDispenser Module by proving the following goals:
                                 testAdd
                                 testToString
                                 testLessEqual
                                 testSetInitialCash
                                 testCheckCashOnHand
                                 testSubtract
                                 testDispenseCash
                                 Money Module

Event4: Sun Sep 20 13:38:13 EEST 2015: <<Evaluator>> to prove goal CashDispenser Module proves goal testAdd true
Event5: Sun Sep 20 13:38:13 EEST 2015: <<Evaluator>> to prove goal CashDispenser Module proves goal testToString true
Event6: Sun Sep 20 13:38:13 EEST 2015: <<Evaluator>> requests additional data for testLessEqual
Event7: Sun Sep 20 13:38:13 EEST 2015: <<GoalNodeVerifier>> Running additional tests for goal nodetestLessEqual
Event8: Sun Sep 20 13:38:13 EEST 2015: <<GoalNodeVerifier>> Starting TestDriver for testLessEqual
Event9: Sun Sep 20 13:38:13 EEST 2015: <<GoalNodeVerifier>> Running Goal Node Tester : GoalNodeTester for lessEqual
Testing function lessEqual with JUnit
add
subtract
toString
lessEqual
ERROR!. lessEqual function is not working correctly.

Event10: Sun Sep 20 13:38:13 EEST 2015: <<GoalNodeVerifier>> Goalnode testLessEqual is false
Event11: Sun Sep 20 13:38:13 EEST 2015: <<GoalNodeVerifier>> One TestStrategy failed, so the Node testLessEqual is false!
Event12: Sun Sep 20 13:38:13 EEST 2015: <<Evaluator>> to prove goal CashDispenser Module proves goal testLessEqual false.
                                        Evaluation of CashDispenser Module fails. Evaluator will check another solution
Event13: Sun Sep 20 13:38:13 EEST 2015: <<Evaluator>> all solutions for CashDispenser Module failed. CashDispenser Module
                                                       is false.
Process Completed
542
Max Memory used: 3MB
Proved 4 goals
```

Σχήμα 6.26: Αποτέλεσμα εκτέλεσης στατικής ανάλυσης

Βλέπουμε λοιπόν ότι με την ενεργοποίηση του Alarm5 ξεκίνησε ο έλεγχος της λειτουργίας CashDispenser. Το αποτέλεσμα του SAT4j έδειξε τους κόμβους που πρέπει να ελεγχθούν προκειμένου να αποφανθούμε αν η λειτουργία αυτή του ATM συμφωνεί με τις αρχικές απατήσεις ή όχι. Για τον έλεγχο του κόμβου που αντιστοιχεί στη λειτουργία lessEqual εκτελέστηκαν τα σενάρια ελέγχου Junit (add,subtract και lessEqual) [63][64]. Το αποτέλεσμα των σεναρίων ελέγχου έδειξαν ότι η μέθοδος lessEqual δεν λειτουργεί σωστά. Για το λόγο αυτό όλοι η συνδυασμοί - λύσεις του SAT4j απέτυχαν να ικανοποιήσουν τον αρχικό στόχο ("CashDispenser") . Βρέθηκε λοιπόν ο λόγος για τον οποίο η αρχική προσπάθεια ανάληψης από το προσομοιωμένο ATM απέτυχε είναι η δυσλειτουργία της μεθόδου lessEqual. Η μέθοδος αυτή αποτελεί την αλλαγή που είχαμε πραγματοποιήσει στον κώδικα του προσομοιωμένου ATM με σκοπό την αποτυχία ανάληψης.

Είναι προφανές λοιπόν πως ο συνδυασμός δυναμικής και στατικής ανάλυσης που εφαρμόστηκε, με τη χρήση μοντελοκεντρικής αρχιτεκτονικής, οδήγησε στον εντοπισμό τις λειτουργίας που παρουσιάζει κάποια δυσλειτουργία αλλά και στον εντοπισμό του τμήματος κώδικα το οποίο προκάλεσε τη δυσλειτουργία αυτή.

Κεφάλαιο 7: Επίλογος

Στο κεφάλαιο αυτό αξιολογείται συνοπτικά το περιβάλλον -πλαίσιο που αναπτύχθηκε για τον στατικό και δυναμικό έλεγχο συστημάτων λογισμικού. Η αξιολόγηση αυτή εστιάζει στην εξαγωγή συμπερασμάτων που σχετίζονται με την αρχιτεκτονική και τις λειτουργίες του περιβάλλοντος πλαισίου που χρησιμοποιήθηκαν αλλά και την ικανοποίηση των αρχικών στόχων που τέθηκαν. Επιπλέον παρουσιάζονται προτάσεις για μελλοντικές επεκτάσεις του θέματος που πραγματεύεται η παρούσα διπλωματική εργασία και του περιβάλλοντος πλαισίου που αναπτύχθηκε.

7.1. Σύνοψη και συμπεράσματα

Το περιβάλλον πλαίσιο που αναπτύχθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας εξέτασε το πρόβλημα εφαρμογής στατικού και δυναμικού ελέγχου σε συστήματα λογισμικού. Περιληπτικά το πρόβλημα που εξέτασε ήταν η παρακολούθηση ενός συστήματος λογισμικού με σκοπό τον έλεγχο της λειτουργίας του και το κατά πόσο πληρούνται οι τεχνικές προδιαγραφές και απαιτήσεις του συστήματος. Για το σκοπό αυτό, το σύστημα που αναπτύχθηκε δίνει τη δυνατότητα παρουσίασης των απαιτήσεων ενός συστήματος λογισμικού σε μοντέλα δέντρων-στόχων, παρέχει τη δυνατότητα ανάλυσης των δέντρων αυτών και την εξέταση των υπό-κόμβων στόχων, με την εφαρμογή μεθόδων δυναμικού αλλά και στατικού ελέγχου. Η εφαρμογή των μεθόδων αυτών για τον έλεγχο των απαιτήσεων του υπό έλεγχο συστήματος , μπορεί να οδηγήσει στον εντοπισμό του λανθασμένου τμήματος κώδικα , ενός συστήματος το οποίο παρουσίασε κάποια δυσλειτουργία.

Με το μοντελοκεντρικό αυτό σύστημα ελέγχου που αναπτύχθηκε επιτυγχάνεται άμεσος έλεγχος ικανοποίησης των αρχικών προδιαγραφών αλλά και άμεσος έλεγχος στην περίπτωση δυσλειτουργίας του υπό έλεγχο συστήματος. Η λύση αυτή είναι αποτελεί μια αποδοτική και επεκτάσιμη λύση στο πρόβλημα ελέγχου ενός συστήματος λογισμικού βασισμένη στις απαιτήσεις που το σύστημα πρέπει να πληροί. Η αποδοτικότητα και η επεκτασιμότητα του συστήματος βασίζεται στις αρχές μοντελοποίησης και ανάπτυξης του συστήματος που χρησιμοποιήθηκαν. Η ανάπτυξη του συστήματος βασίστηκε στα αρχιτεκτονικά μορφήματα "Μαυροπίνακα" και "Publisher/subscriber" τα οποία έχουν ως κύριο στόχο τη δημιουργία ευέλικτου και επεκτάσιμου κώδικα σε συστήματα λογισμικού. Η αναπαράσταση των μοντέλων απαιτήσεων πραγματοποιήθηκε μέσω της χρήσης του πλαισίου EMF του Eclipse. Το περιβάλλον πλαίσιο αυτό δίνει τη δυνατότητα στον χρήστη να ορίσει εύκολα πολλά και διαφορετικά μοντέλα στόχων , να προκαθορίσει τη σχέση μεταξύ στόχων και υπό-στόχων αλλά και τη σχέση των στόχων με συγκεκριμένες ενδείξεις δυσλειτουργίας. Επιπλέον για τον έλεγχο των απαιτήσεων

του μοντέλου δέντρου στόχων δεν πραγματοποιήθηκε ένα προς ένα έλεγχος των στόχων και των υπό-στόχων αυτών. Για την βελτιστοποίηση του συστήματος, το δέντρο στόχων μετατρέπεται μέσω του συστήματος σε μορφή τέτοια, ώστε να εμπεριέχει όλη την γνώση επί του μοντέλου και των κόμβων αυτού αλλά και να μπορεί να διεξαχθεί ο ελάχιστος συνδυασμός κόμβων που οδηγούν στην ικανοποίηση των απαιτήσεων του. Αυτό πραγματοποιήθηκε μέσω τις χρήσης της βιβλιοθήκης SAT4j . Τέλος ο έλεγχος κάθε κόμβου πραγματοποιήθηκε με αυτοματοποιημένα σενάρια ελέγχου , τόσο κατά τη διάρκεια της δυναμικής όσο και της στατικής ανάλυσης. Η κατασκευή των σεναρίων ελέγχου και συγκεκριμένα η αλληλεξάρτηση ενός κόμβου με κάποιο σενάριο ελέγχου , βασίστηκε στο αρχιτεκτονικό μόρφημα "Strategy" [68]. Με τη χρήση της στρατηγικής αυτής δίνεται η δυνατότητα στον χρήση να ορίσει μέσω του μοντέλου μια στρατηγική επαλήθευσης του κάθε κόμβου η οποία περιέχει ένα σύνολο διαφορετικών σεναρίων ελέγχου. Με αυτό τον τρόπο η προσθήκη ενός νέου σεναρίου ελέγχου για ένα κόμβο του μοντέλου δεν απαιτεί καθόλου αλλαγές στο κώδικα του περιβάλλοντος πλαισίου που αναπτύχθηκε. Το μόνο που απαιτείται είναι η δημιουργία του αντίστοιχου μοντέλου που θα επισημαίνει την ύπαρξη των σεναρίων ελέγχου . Επιπλέον ο κώδικας του σεναρίου ελέγχου είναι πλήρως ανεξάρτητος από το κώδικα του περιβάλλοντος πλαισίου.

7.2. Μελλοντικές Επεκτάσεις

Το περιβάλλον πλαίσιο της παρούσας διπλωματικής εργασίας έχει σχεδιαστεί έτσι ώστε να είναι επεκτάσιμο και να μπορούν να προστεθούν νέες λειτουργίες και χαρακτηριστικά στο σύστημα. Όπως έχουμε αναφέρει, το κόστος συντήρησης είναι χαμηλό, καθώς υπάρχει δυνατότητα επαναχρησιμοποίησης των σεναρίων δοκιμών, καθώς και ολόκληρου του μοντέλου. Συνεπώς το περιβάλλον πλαίσιο αυτό, προσφέρεται για μελλοντικές επεκτάσεις, προσθήκες και βελτιστοποιήσεις, μερικές από τις οποίες προτείνονται σε αυτή την ενότητα.

Μία σημαντική επέκταση αφορά τον τρόπο επαλήθευσης των κόμβων που είναι συνδεδεμένοι με στρατηγικές. Στην παρούσα εργασία , χρησιμοποιήθηκε η δυναμική ανάλυση (Log Analysis) και η στατική ανάλυση με έλεγχο μονάδων κώδικα (Unit Testing). Ένα πολύ σημαντικό χαρακτηριστικό του σχεδιασμού των στρατηγικών, είναι πως είναι ευέλικτος, δηλαδή μία τροποποίηση μπορεί να εφαρμοστεί σε όλα τα υπάρχοντα σενάρια ελέγχου χωρίς να χρειαστεί να επανασχεδιαστούν. Επομένως προτείνουμε την μελλοντική τροποποίηση των στρατηγικών ώστε να καλύπτουν μεγαλύτερο ποσοστό ελέγχου πάνω στο υπό έλεγχο σύστημα. Αυτό μπορεί να γίνει χρησιμοποιώντας διαφορετικές τεχνικές ελέγχου λογισμικού, που ταιριάζουν καλύτερα στο σύστημα που ελέγχεται κάθε φορά. Ενδεικτικά προτείνεται ο έλεγχος μαύρου κουτιού (BlackBox Testing) [65] , όπου ο έλεγχος βασίζεται στις απαιτήσεις του συστήματος και στη λειτουργία του ως σύνολο, και όχι στον έλεγχο των μονάδων του. Μία άλλη πρόταση είναι έλεγχος προσομοίωσης ακραίων καταστάσεων (Stress Testing) [66], όπου ελέγχεται η απόδοση του συστήματος υπό μεγάλο φορτίο, όπως για παράδειγμα με μεγάλο αριθμό χρηστών την ίδια

στιγμή. Τέλος, προτείνουμε την τροποποίηση των στρατηγικών ώστε να υποστηρίζουν τον έλεγχο συστήματος για εύρεση νεοεισαχθέντων σφαλμάτων (Regression Testing) [67]. Αυτός ο τύπος ελέγχου προϋποθέτει τη συχνή εκτέλεσή του πάνω στο υπό έλεγχο σύστημα, καθώς ελέγχει την εφαρμογή στο σύνολό της για τυχόν τροποποίηση μίας μονάδας ή λειτουργίας του συστήματος, μετά από την είσοδο μίας καινούργιας λειτουργίας ή την διόρθωση ήδη υπάρχουσας.

Μία επιπλέον τροποποίηση που μπορεί να γίνει στο πλαίσιο περιβάλλον της παρούσας διπλωματικής εργασίας, αφορά την λογική της δυναμικής ανάλυσης που ακολουθήσαμε. Στο παράδειγμά μας, το περιβάλλον πλαίσιο αποδεικνύει κάποιες υποθέσεις αναζητώντας συγκεκριμένες καταχωρήσεις στα δεδομένα παρακολούθησης του υπό έλεγχο συστήματος. Μία βελτιστοποίηση που θα μπορούσε να πραγματοποιηθεί, είναι μία πιο πολύπλοκη ανάλυση δεδομένων παρακολούθησης, η οποία θα συνδυάζει πολλές καταγραφές γεγονότων για την εξαγωγή συμπεράσματος. Με τον τρόπο αυτό θα έχουμε μία πιο αξιόπιστη απόφαση ακόμα και για λειτουργίες οι οποίες δεν παράγουν κάποιο δεδομένο ή στην περίπτωση που δεν έχουν καταγραφεί λεπτομερώς τα δεδομένα παρακολούθησης όλων των ενεργειών που εκτελέστηκαν στο σύστημα. Μέσα από τον συνδυασμό γεγονότων, και την σύγκριση χρονικών στιγμιότυπων, θα μπορούμε να βγάλουμε ένα πιο ασφαλές συμπέρασμα ακόμα και για πολύπλοκες λειτουργίες που η σωστή λειτουργία τους αποτελεί συνιστώσα διαφόρων γεγονότων.

Σημαντική βελτίωση μπορεί να αποτελέσει και η μείωση του υπολογιστικού κόστους της επαλήθευσης των κόμβων των δέντρων στόχων. Αυτό μπορεί να επιτευχθεί με ταξινόμηση της λίστας κόμβων, σύμφωνα με κάποια βάρη που επισυνάπτονται σε αυτούς. Τα βάρη αυτά μπορούν να αντιπροσωπεύουν την βαρύτητα που έχει κάποια υπολειτουργία του συστήματος, δηλαδή πόσο μπορεί να επηρεάσει τις σημαντικότερες λειτουργίες του συστήματος, ή την πιθανότητα που έχουν να εμφανίσουν κάποιο σφάλμα. Αυτό προϋποθέτει την διατήρηση κάποιας μορφής ιστορικού από το περιβάλλον πλαίσιο. Συνδέοντας τους κόμβους με βάρη , θα μπορούμε να προσπαθήσουμε να επαληθεύσουμε με προτεραιότητα εκείνους τους κόμβους που είναι πιο πιθανό να παρουσιάσουν κάποιο σφάλμα, και να αποφύγουμε την επαλήθευση των κόμβων με χαμηλότερη προτεραιότητα, βρίσκοντας την αιτία αποτυχίας πιο γρήγορα. Με τον τρόπο αυτό επιτυγχάνουμε μείωση του χρόνου εκτέλεσης του περιβάλλοντος πλαισίου καθώς και του υπολογιστικού του κόστους.

Μία επιπλέον βελτιστοποίηση που μπορεί να υλοποιηθεί μελλοντικά είναι η παραλληλοποίηση της επαλήθευσης των αρχικών αιτιών αποτυχίας του συστήματος, δηλαδή των κόμβων του δέντρου στόχων η οποία αναμένεται να βελτιώσει σημαντικά την επίδοση του περιβάλλοντος-πλαισίου και να μειώσει αισθητά τον χρόνο εκτέλεσής του. Κάτι τέτοιο ωστόσο είναι αρκετά απαιτητικό καθώς τίθενται θέματα συγχρονισμού.

Τέλος, μία μελλοντική επέκταση του παρόντος περιβάλλοντος πλαισίου, θα ήταν η ολοκλήρωσή του με τη χρήση εργαλείων άμεσης παρακολούθησης πραγματικών συστημάτων έτσι ώστε να μπορεί να παρακολουθεί το υπό έλεγχο σύστημα κατά τη διάρκεια εκτέλεσής του και να εκκινεί αυτόματα τη διαδικασία ανάλυσης και επαλήθευσης αρχικών αιτιών μίας πιθανής αποτυχίας του. Με αυτό τον τρόπο δε θα είναι απαραίτητος ο ανθρώπινος παράγοντας για την εισαγωγή των

δεδομένων παρακολούθησης ώστε να εκκινήσει η διαδικασία. Επίσης η διαδικασία ανάλυσης αρχικών αιτιών θα ξεκινά πιο άμεσα, δηλαδή αμέσως μετά την ένδειξη κάποιου σφάλματος, και όχι μετά την ολοκλήρωση μίας σειράς ενεργειών πάνω στο υπό έλεγχο σύστημα.

Κεφάλαιο 8: Βιβλιογραφία

[1] Abhijit A. Sawant,Pranit H. Bari and P. M. Chawan "Software Testing Techniques and Strategies"
Department of Computer Technology, VJTI, University of Mumbai (IJERA) ISSN: 2248-9622

[2] Luis Alejandro Cort´es " Verification and Scheduling Techniques for Real-Time Embedded Systems"
Department of Computer and Information Science Link¨oping University, S-581 83 Link¨oping, Sweden 2005

[3] James J. Rooney and Lee N. Vanden Heuvel "Root Cause Analysis For Beginners" QUALITY PROGRESS JULY 2004

[4] Jan Valdman "Log File Analysis" PhD Report, University of West Bohemia in Pilsen, Department of Computer Science and Engineering, July, 2001

[5] O.M.G., "Meta Object Facility (MOF) Core Specification" , (2015) formal/2015-06-05

[6] Atkinson, C. ; Kuhne, T., " Model-driven development: a metamodeling foundation", 2003

[7] Sridhar Srinivasa Iyengar," Metadata driven system for effecting extensible data interchange based on universal modeling language (UML), meta object facility (MOF) and extensible markup language (XML) standards ", (2005)

[8] Perdita Stevens,"Small-Scale XMI Programming: A Revolution in UML Tool Use?",(2003)

[9] O.M.G., "XML Metadata Interchange (XMI) Specification" , (2015) formal/2015-06-07.

[10] ISO/IEC 19503, Information technology -- XML Metadata Interchange (XMI)", (2005)

[11] E. Foundation. (2015, Sep.) Eclipse Modeling Framework. [Online]. http://www.eclipse.org/modeling/emf/

[12] Wikipedia. (2015, Sep.) Meta Object Facility - Wikipedia. [Online]. http://en.wikipedia.org/wiki/Meta-Object_Facility

[13] Iván García-Magariño, Rubén Fuentes-Fernández, Jorge J Gómez-Sanz,"Guideline for the definition of EMF metamodels using an Entity-Relationship approach", (2009)

[14] Alexei Lapouchnian "Goal-Oriented Requirements Engineering: An Overview of the Current Research" Department of Computer Science University Of Toronto (2005)

[15] Jaelson Castro, Manuel Colp and John Mylopoulos "Towards requirements-driven information systems engineering: the Tropos project" Journal Information Systems - The 13th international conference on advanced information systems engineering (CAiSE*01) Volume 27 Issue 6, September 2002 Pages 365 – 389 (2002)

[16] Evangelia Kavakli and Pericles Loucopoulos, Goal Modelling in Requirements Engineering : Analysis and Critique of Current Methods, Department of Cultural Technology and Communication, University of the Aegean

[17] Zachary J. Oster, Ganesh Ram Santhanam, and Samik Basu, "Automating Analysis of Qualitative Preferences in Goal-Oriented Requirements Engineering"
Published version of paper: 26th IEEE/ACM Intl. Conference on Automated Software Engineering (ASE 2011), pp. 448-451, IEEE CS Press, 2011.

[18] John Mylopoulos,Lawrence Chung, Eric Yu, "From object-oriented to goal-oriented requirements analysis" Magazine Communications of the ACM CACM Homepage archive Volume 42 Issue 1, Jan. 1999 Pages 31-37

[19] Amit K. Chopra,Fabiano Dalpiaz,Paolo Giorgini,John Mylopoulos , "Reasoning about Agents and Protocols via Goals and Commitments" May, 10–14, 2010, Toronto, Canada, pp.457-464

[20] R. Sebastiani, P. Giorgini, and J. Mylopoulos. "Simple and minimum-cost satisfiability for goal models". In Proceedings, CAiSE , Volume 3084 of LNCS, pages 20–35. Springer, 2004

[21] Miko laj Bojanczyk and Thomas Place, University of Warsaw, "Regular Languages of Infinite Trees that are Boolean Combinations of Open Sets" (2004)

[22] Wikipedia (2015, Sep.), "First-order logic", [Online], https://en.wikipedia.org/wiki/First-order_logic

[23] Wikipedia (2015, Sep.), "FMarkov chain", [Online], https://en.wikipedia.org/wiki/Markov_chain

[24] Zinovy Diskin, «Model transformation via pull-backs: algebra vs. Heuristics» School of Computing, Queen's University, Kingston, Ontario, Canada, 2006

[25] Mit (2015, Sep.),Model-based testing (MBT), [Online], http://mit.bme.hu/~micskeiz/pages/modelbased_testing.html

[26] Mark Utting , «Position Paper: Model-Based Testing» , The University of Waikato, New Zealand

[27] Stuart Reid, «Model-Based Testing», Cranfield University ,Royal Military College of Science, Swindon, UK

[28] Weissleder, S. , Schlingloff, H., «An Evaluation of Model-Based Testing in Embedded Applications» Software Testing, Verification and Validation (ICST), 2014 IEEE Seventh International Conference March 31 2014-April 4 2014, Page(s):223 – 232

[29] Wolfgang Prenninger, Alexander Pretschner, “Abstractions for Model-Based Testing” Journal Electronic Notes in Theoretical Computer Science (ENTCS) Volume 116, January, 2005,Pages 59-71

[30] Larry Apfelbaum , John Doyle, “Model Based Testing” , Teradyne Software & Systems Test

[31] Microsoft MSDN (2015, Sep.),"Model-Based Testing", [Online] https://msdn.microsoft.com/en-us/library/ee620469.aspx

[32] Jan Tretmans , “Model-Based Testing and Some Steps towards Test-Based Modelling” Chapter “Formal Methods for Eternal Networked Software Systems”, Volume 6659 of the series Lecture Notes in Computer Science pp 297-326

[33] Zhenyi Jin, “A software architecture-based testing technique” , Doctoral Dissertation,A software architecture-based testing technique, George Mason University Fairfax, VA, USA ©2001

[34] Victor V. Kuliamin, “Model Based Testing of Large-scale Software: How Can Simple Models Help to Test Complex System” (2004) , Proc. of 1-st International Symposium on Leveraging Applications of Formal Methods

[35] School of Computer Science, University of Buenos Aires, "Software Architecture-based Testing and Model-checking", (2005)

[36] 3rd IEEE International Symposium on Requirements Engineering RE97, IEEE Computer Society, 1997

[37] Microsoft MSDN (2015, Sep.),"Model-Based Testing", [Online] https://msdn.microsoft.com/en-us/library/ff742858.aspx

[38] Stuart Reid, «Model-Based Testing», Cranfield University ,Royal Military College of Science, Swindon, UK

[39] Wikipedia (Sep, 2015), "Unit Testing", [Online], https://en.wikipedia.org/wiki/Unit_testing

[40] Laurie Williams, Gunnar Kudrjavets, and Nachiappan Nagappan, “On the Effectiveness of Unit Test Automation at Microsoft”, Proceedings of the IEEE International Symposium on Software Reliability Engineering (ISSRE), November 2009

[41] Parasoft white paper (part1), "Practical Practical Unit Testing for Embedded Embedded Systems Systems", (2015)

[42] Yoonsik Cheon , Gary T. Leavens, “A Simple and Practical Approach to Unit Testing: The JML and JUnit Way” Proceeding ECOOP '02 Proceedings of the 16th European Conference on Object-Oriented Programming , Pages 231-255 Springer-Verlag London, UK ©2002

[43]Vogella (Sep, 2015) , "JUnit tutorials", [Online], http://www.vogella.com/tutorials/JUnit/article.html#junittesting

[44] James J. Rooney and Lee N. Vanden Heuvel,"Root Cause Analysis For Beginners"

[45] ABS Consulting , Lee N. Vanden Heuvel , Donald K. Lorenzo , and James J. Rooney, "Root Cause Analysis Handbook: A Guide to Efficient and Effective Incident Investigation ", 2008

[46] Rootcauseanalysisbook (2015, Sep.), "Root Cause Analysis,", [Online], http://www.rootcauseanalysisbook.com/deHavilland-Comet-aircraft-Accident.aspx

[47] Branislav TOMIĆ,Vesna SPASOJEVIĆ BRKIĆ,"EFFECTIVE ROOT CAUSE ANALYSIS AND CORRECTIVE ACTION PROCESS", (2011)

[48] ISTQB(International Software Testing Qualifications Board), "Standard Glossary of Terms Used in Software Testing" , (Version:3.01)

[49] IBM. (1015, Sep.) Dynamic Analysis Tools. [Online] http://www.ibm.com/developerworks/library/se-static/

[50] Yowcode. (21015, sep.) ISTQB-Static Analysis . [Online] http://www.yowcode.com/

[51] Wikipedia. (2015, Sep.) Systems design - Wikipedia. [Online] https://en.wikipedia.org/wiki/Systems_design

[52] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal," Pattern-oriented Software Architecture: A System of Pattern, Wiley & Sons", 2001

[53] Philippe Lalanda, Two complementary patterns to build multi-expert systems, July 2009

[54] WikiBooks (2015, Sep.), "Introduction to Software Engineering/Architecture/Design Patterns", [Online], https://en.wikibooks.org/wiki/Introduction_to_Software_Engineering/Architecture/Design_Patterns

[55] Microsoft MSDN (2015, Sep.),"Publish/Subscribe", [Online], https://msdn.microsoft.com/en-us/library/ff649664.aspx

[56] Oracle (2015, Sep.), "Processes and Threads", [Online] https://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html

[57] George Chatzikonstantinou1, Kostas Kontogiannis1,and Ioanna-Maria Attarian "A Goal Driven Framework for Software Project Data Analytics", 2013

[58] 5. Chopra, A.K., Dalpiaz, F., Giorgini, P., Mylopoulos, J. "Reasoning about agents and protocols via goals and commitments. In: AAMAS" (2010) 457–464

[59] Velev, M.N, " Efficient translation of boolean formulas to cnf in formal verification of microprocessors. In: Proceedings of the 2004 Asia and South Pacific Design

Automation Conference", ASP-DAC '04 (2004)

[60] Foswiki (2015, Sep.), "Goal-Oriented Conceptual Database Design", [Online], http://foswiki.cs.uu.nl/foswiki/MethodEngineering/Goal-OrientedConceptualDatabaseDesign20122013

[61] Github (2015, Sep.), "How to run the ATM application managed by Zanshin", [Online], https://github.com/sefms-disi-unitn/Zanshin/wiki/How-to-run-the-ATM-application-managed-by-Zanshin

[62] Math-cs.gordon (2015, Sep.), "ATM Example", [Online] , http://www.math-cs.gordon.edu/local/courses/cs211/ATMExample/Intro.html

[63] Alexander Prohorenko and Olexiy Prokhorenko, (2014,April)," Using JUnit With Eclipse IDE", [Online] , http://www.onjava.com/pub/a/onjava/2004/02/04/juie.html

[64] Lars Vogel, (2015, June), "Unit Testing with JUnit - Tutorial", [Online] , http://www.vogella.com/tutorials/JUnit/article.html

[65] Wikipedia (2014, Nov.), " Black-Box-Test ", [Online], https://de.wikipedia.org/wiki/Black-Box-Test

[66] Wikipedia (2015, June), " StressTest", [Online], https://de.wikipedia.org/wiki/Stresstest

[67] Wikipedia (2015, July), " Regression Testing ", [Online], https://en.wikipedia.org/wiki/Regression_testing

[68] Sourcemaking (2014, Nov), " Strategy Design Pattern ", [Online], https://sourcemaking.com/design_patterns/strategy

[69] Hamzeh Zawawy, Serge Mankovskii, Kostas Kontogiannis, John Mylopoulos "Mining Software Logs for Goal-Driven Root Cause Analysis", (2015)

[70] Jan Tretmans, "Model-Based Testing and Some Steps towards Test-Based Modelling", Conference: Formal Methods for Eternal Networked Software Systems , June 13-18, (2011)