ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Περιβάλλον-Πλαίσιο για την Παρακολούθηση και την Ανάλυση Συστημάτων Λογισμικού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΙΧΑΗΛ Γ. ΑΘΑΝΑΣΟΠΟΥΛΟΣ

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

Αθήνα, Ιούλιος 2008

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Περιβάλλον-Πλαίσιο για την Παρακολούθηση και την Ανάλυση Συστημάτων Λογισμικού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΙΧΑΗΛ Γ. ΑΘΑΝΑΣΟΠΟΥΛΟΣ

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

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 9η Ιουλίου 2008.

.....	.....	.....
Κωνσταντίνος Κοντογιάννης Αναπλ. Καθηγητής Ε.Μ.Π.	Ιωάννης Βασιλείου Καθηγητής Ε.Μ.Π.	Τιμολέων Σελλής Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2008

.....

ΜΙΧΑΗΛ Γ. ΑΘΑΝΑΣΟΠΟΥΛΟΣ

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

Περίληψη

Μια από τις μεθόδους επιβεβαίωσης της ορθής λειτουργίας ενός συστήματος λογισμικού είναι μέσω της παρακολούθησης και ανάλυσης του συστήματος κατά την εκτέλεσή της λειτουργίας του. Η παρακολούθηση λογισμικού (software monitoring) ορίζεται ως η διαδικασία παραγωγής πληροφοριών και εξαγωγής συμπερασμάτων σχετικών με σημαντικά συμβάντα που σημειώνονται κατά τη διάρκεια της λειτουργίας ενός συστήματος λογισμικού. Οι πληροφορίες που εξάγονται από το σύστημα κωδικοποιούνται ως γεγονότα (*events*) και αποθηκεύονται για περαιτέρω ανάλυση και έλεγχο.

Στα μεγάλα όμως, βιομηχανικά συστήματα λογισμικού που αποτελούνται από πολλές μονάδες (δομοστοιχεία) και ακολουθούν συχνά κατανεμημένες αρχιτεκτονικές, παράγεται τεράστιος όγκος πληροφοριών παρακολούθησης, οι οποίες συνήθως είναι δύσκολο να αναλυθούν. Τα κύρια προβλήματα που αφορούν στη διαδικασία παρακολούθησης και ανάλυσης σχετίζονται α) με κατάλληλες τεχνικές υπομνηματισμού και σχολιασμού του λογισμικού για την παραγωγή δεδομένων παρακολούθησης, β) με τη προδιαγραφή σεναρίων ενδιαφέροντος και της έκφρασής τους ως μοτίβα γεγονότων και τέλος γ) με μεθόδους αναγνώρισης των σεναρίων αυτών στις ακολουθίες των γεγονότων που συλλέγονται.

Σκοπός αυτής της εργασίας είναι η δημιουργία ενός περιβάλλοντος-πλαισίου για την αποτελεσματική και αποδοτική παρακολούθηση και ανάλυση της λειτουργίας συστημάτων λογισμικού. Πρώτον, προτείνεται μια τεχνική υπομνηματισμού λογισμικού υλοποιημένου σε Java, χωρίς επέμβαση στον πηγαίο κώδικα (*non-intrusive*) για την παραγωγή πληροφοριών παρακολούθησης. Δεύτερον, προτείνεται η χρήση Διαγραμμάτων Ακολουθίας της Ενοποιημένης Γλώσσας Μοντελοποίησης για τη διατύπωση σεναρίων ορθής-προδιαγραμμένης λειτουργίας του συστήματος, και στη συνέχεια προτείνεται και υλοποιείται ένας μετασχηματισμός των Διαγραμμάτων Ακολουθίας σε Δίκτυα Petri, τα οποία μπορούν να εκτελεστούν και να αναλυθούν με βάση τις πληροφορίες παρακολούθησης. Τρίτον, προτείνεται και υλοποιείται μια τεχνική ανάλυσης των Δικτύων Petri, με βάση τις πληροφορίες παρακολούθησης, η οποία οδηγεί τελικά στην αναγνώριση και επαλήθευση της εμφάνισης, ή μη, των σεναρίων ενδιαφέροντος όπως αυτά έχουν μοντελοποιηθεί από τα αντίστοιχα Δίκτυα Petri. Σε αυτή την εργασία, γίνονται δύο προσεγγίσεις για την ανάλυση των πληροφοριών παρακολούθησης. Στη πρώτη προσέγγιση, τα δεδομένα εισόδου της διαδικασίας προέρχονται από αρχεία καταγραφής (*off-line log file analysis*). Στη δεύτερη προσέγγιση προτείνεται και υλοποιείται μια αρχιτεκτονική επιγραμμικής ανάλυσης των ακολουθιών γεγονότων που τροφοδοτούνται στο εργαλείο ελέγχου και αναλύονται σε πραγματικό χρόνο (*real-time monitoring*).

Οι τεχνικές, τα εργαλεία και οι μέθοδοι παρακολούθησης και ανάλυσης που περιλαμβάνονται στο περιβάλλον-πλαίσιο μπορούν να χρησιμοποιηθούν σε ένα πλήθος περιπτώσεων όπως για τον έλεγχο τήρησης απαιτήσεων για ένα σύστημα, για τον εντοπισμό περιστατικών ασφάλειας και για τη διάγνωση υπαρχόντων ή ενδεχόμενων προβλημάτων.

Λέξεις Κλειδιά: <<παρακολούθηση συστημάτων λογισμικού, μετασχηματισμός μοντέλων, ανάλυση αρχείων καταγραφής, Δίκτυα Petri, ταίριασμα σχηματομορφών, έλεγχος μοντέλου>>

Abstract

One of the techniques for the verification of the correctness and efficiency of a software system is through monitoring and analysis of its execution. Software monitoring is defined as the production of information and the extraction of conclusions related to the run-time behavior of a software system. This information is encoded as events and can be consequently used for analysis and verification.

However, large-scale industrial software systems consist of many components that usually follow distributed architectures, producing a large amount of monitoring data which may be difficult to analyze. The main problems that have to be dealt with regards to monitoring and analysis of industrial software systems are: (a) application of appropriate software instrumentation techniques to monitor execution; (b) expression of scenarios of interest based on events that can be monitored; and (c) definition of methods for matching these different scenarios against the event sequences observed by the monitoring of such systems.

The scope of this thesis is to develop a new framework for effectively and efficiently monitor and analyze the execution of large-scale software systems. First, we utilize a non-intrusive instrumentation technique for software implemented in Java in order to produce monitoring information (log data). Second, we propose the representation of scenarios related to the "correct" or "specified" behavior of the system through Sequence Diagrams of the Unified Modeling Language. A model transformation specification of Sequence Diagrams to Petri Nets can produce equivalent Petri Net models from such Sequence Diagrams. Third, Petri Net models are executed and analyzed against the events collected by the software monitors. The analysis technique allows for the verification or rejection of the original scenarios' occurrences as these are specified by the Petri Net model. This analysis can be performed in two ways. In the first approach, log files are used as input (off-line log file analysis). In the second approach, an online analysis architecture is proposed through which event sequences are collected and analyzed against the specification models in real-time (real-time monitoring).

The techniques, tools and methods of monitoring and analysis included in this framework can be applied into a wide variety of cases such as run-time requirements monitoring, detection of security incidents and diagnosis of existing or imminent problems.

Keywords: <<software systems monitoring, model transformation, log file analysis, Petri Nets, pattern matching, model checking>>

Ευχαριστίες

Θέλω να ευχαριστήσω θερμά τον επιβλέποντα της εργασίας μου κ. Κώστα Κοντογιάννη για το ενδιαφέρον του, για το χρόνο που αφιέρωσε και για την ευκαιρία που μου έδωσε μέσω της διπλωματικής μου εργασίας να επιχειρήσω μια πρώτη, ενδιαφέρουσα γνωριμία με την ερευνητική διάσταση της τεχνολογίας λογισμικού. Ευχαριστώ θερμά και τους καθηγητές κκ. Τίμο Σελλή και Γιάννη Βασιλείου για την τιμή που μου έκαναν να διατελέσουν ως μέλη της επιτροπής για αυτή τη Διπλωματική Εργασία.

Ευχαριστώ επίσης την εταιρία No Magic, Inc. για τη δωρεάν παραχώρηση του προγράμματος MagicDraw UML για ακαδημαϊκή χρήση, όπως και όλους τους επώνυμους και ανώνυμους συνεισφέροντες στα εργαλεία, ανοιχτού ή κλειστού κώδικα, τα οποία χρησιμοποιήθηκαν για την υλοποίηση της εργασίας.

Δε θα μπορούσα να μην ευχαριστήσω τους συμφοιτητές μου στο Πολυτεχνείο, τους συνεργάτες και προϊσταμένους μου στην Εθνική Τράπεζα, τους φίλους μου και τους συγγενείς μου που υποστήριξαν, ο καθένας με το δικό του τρόπο, την προσπάθειά μου να ανταποκριθώ παράλληλα στις απαιτήσεις των σπουδών και της δουλειάς, τόσο κατά το τελευταίο διάστημα εκπόνησης της διπλωματικής εργασίας όσο και συνολικά τα τρία τελευταία χρόνια.

Ευχαριστώ ιδιαιτέρως το θείο μου, Δρ. Κωνσταντίνο Καραγκιόζη για τις πολύτιμες συμβουλές του καθ' όλη τη διάρκεια των σπουδών μου.

Αν και είναι δύσκολο να εκφράσω με λέξεις την ευγνωμοσύνη που νιώθω για την οικογένειά μου, θέλω από τα βάθη της καρδιάς μου να ευχαριστήσω τις αδερφές μου Μαρία και Θεοδώρα και τους γονείς μου Γιώργο και Στέλλα για την υποστήριξη, την εμπιστοσύνη και την αγάπη τους.

Πίνακας περιεχομένων

Κεφάλαιο 1: Εισαγωγή

1.1 Παρακολούθηση, καταγραφή και ανάλυση λειτουργίας συστημάτων λογισμικού

Την τελευταία δεκαετία καθώς εντείνεται η ανάγκη για υψηλότερης ποιότητας, μεγαλύτερης πολυπλοκότητας και μικρότερου κόστους λογισμικό, η επιστημονική και επιχειρηματική κοινότητα της τεχνολογίας λογισμικού έχει προτείνει νέες τεχνολογίες για την προδιαγραφή, τη σχεδίαση, την ανάπτυξη και την παράταξη συστημάτων λογισμικού.

Τεχνικές ανάπτυξης βασισμένες σε δομοστοιχεία και αρχιτεκτονικές προσανατολισμένες σε υπηρεσίες (*Service Oriented Architectures -SOA*) αποτελούν όλο και περισσότερο τις βασικές επιλογές των μηχανικών λογισμικού, ιδιαιτέρως των μεγάλων επιχειρήσεων και οργανισμών, τόσο για τη δημιουργία νέων συστημάτων όσο και για την εξέλιξη των υπαρχόντων. Κατά κανόνα, τα τελευταία ιδιαιτέρως χρόνια, τέτοια μεγάλα βιομηχανικά συστήματα λογισμικού μεταβαίνουν, από δομές της μιας και μοναδικής εφαρμογής εξυπηρετητή (του «κεντρικοποιημένου συστήματος»), σε συστήματα βασισμένα σε αυτόνομα δομοστοιχεία (*components*) τα οποία επικοινωνούν και συνεργάζονται, ενώ ταυτόχρονα είναι κατανεμημένα σε δίκτυα τοπικής ή και ευρείας κλίμακας.

Οι τεχνολογικές αυτές εξελίξεις παρέχουν σημαντικά οφέλη ως προς την ποιότητα του λογισμικού, το χρόνο ανάπτυξης και το κόστος παράταξης και συντήρησης των συστημάτων. Επιπλέον, αυξάνουν τα περιθώρια βελτίωσης της επίδοσης και της διαθεσιμότητας ενός συστήματος και παρέχουν σημαντικά περιθώρια κλιμάκωσης. Παρόλα αυτά, ακριβώς αυτή η τεχνολογική μετάβαση, τόσο λόγω της κατανεμημένης φύσης των συστημάτων που εισηγείται, όσο και της ανεξάρτητης ανάπτυξης και αυτόνομης λειτουργίας των διαφόρων δομοστοιχείων, θέτει μιας ενδιαφέρουσα πρόκληση: Πως μπορούμε να διαχειριστούμε και να απαντήσουμε στους εγγενείς κινδύνους και τις επιπλοκές των νέων αυτών τεχνολογιών;

Η ικανότητα ανάλυσης της λειτουργίας τέτοιων μεγάλων και σύνθετων συστημάτων προϋποθέτει την ύπαρξη κατάλληλης υποδομής για την παρακολούθηση και την καταγραφή γεγονότων ενδιαφέροντος κατά το χρόνο εκτέλεσης. Επιπλέον, εξαρτάται σημαντικά από το πως εκφράζονται οι απαιτήσεις, τα πρότυπα και τα σενάρια ορθής/αποδεκτής (ή εσφαλμένης/καταχρηστικής) λειτουργίας και χρήσης του συστήματος. Τέλος, την ικανότητα αυτή ρυθμίζουν επίσης οι τεχνικές ταιριάσματος σχηματομορφών (*pattern matching*) που επιστρατεύονται για τον έλεγχο εμφάνισης των σεναρίων ενδιαφέροντος όπως και οι δυνατότητες κλιμάκωσης των μεθόδων που χρησιμοποιούνται.

1.2 Κίνητρο

Όπως αναφέρθηκε παραπάνω, η ικανότητα ανάλυσης της συμπεριφοράς ενός συστήματος και των δραστηριοτήτων των εξωτερικών παραγόντων που έρχονται σε αλληλεπίδραση με αυτό αποτελεί κρίσιμο ζήτημα για κάθε σύστημα λογισμικού και ιδιαιτέρως για τα μεγάλης κλίμακας, βιομηχανικά συστήματα. Σε τέτοιου είδους συστήματα, ακόμη και οι μικρότερες αποκλίσεις από την αναμενόμενη λειτουργία τους, όπως και μη αποδεκτές πρακτικές χρηστών και συνεργαζόμενων με αυτά συστημάτων που δεν ελέγχονται, μπορούν άμεσα ή με την πάροδο του χρόνου να προκαλέσουν τεράστια κόστη στους οργανισμούς και στις επιχειρήσεις που εξυπηρετούν.

Η αποτελεσματική παρακολούθηση, καταγραφή και ανάλυση ακολουθιών γεγονότων που συμβαίνουν κατά το χρόνο εκτέλεσης σε ένα τέτοιο βιομηχανικό σύστημα λογισμικού, παρέχει στην επιχείρηση ή στον οργανισμό που το χρησιμοποιεί μια πληθώρα δυνατοτήτων όπως να εντοπίζει τα προβλήματα που εμφανίζονται και τα αίτια που τα προκαλούν, να αναβαθμίζει την αξιοπιστία και την επίδοση του συστήματος, να προστατεύει την ασφάλεια των διαδικασιών και των δεδομένων και γενικότερα να βελτιώνει τόσο την ποιότητα της επιχειρηματικής του λειτουργίας όσο και την αποτελεσματικότητά του.

Η κοινότητα της τεχνολογίας λογισμικού απαντώντας στην ανάγκη αυτή έχει παρουσιάσει ποικίλες λύσεις τόσο σε επίπεδο μεθοδολογιών όσο και εργαλείων. Παρόλα αυτά, λόγω της συνεχούς εξέλιξης των τεχνολογιών προδιαγραφής, σχεδιασμού, ανάπτυξης και παράταξης των συστημάτων λογισμικού, η ίδια κοινότητα καλείται να προτείνει νέες, κατάλληλες λύσεις για τις συνθήκες που διαμορφώνονται. Η αυξημένη πολυπλοκότητα και η κατανεμημένη

φύση των νέων προσεγγίσεων όπως και η ανεξάρτητη πολλές φορές ανάπτυξη των δομοστοιχείων καθιστά την παρακολούθηση, την καταγραφή και τη διάγνωση προβλημάτων των συστημάτων και των εφαρμογών ιδιαίτερα σύνθετη διαδικασία, καθώς τα δομοστοιχεία του συστήματος μπορεί να χρησιμοποιούν ετερογενείς τεχνικές καταγραφής γεγονότων, διαφορετικές πολιτικές παρακολούθησης και διαχείρισης καταγραφών ενώ παράλληλα το πλήθος των γεγονότων που καταγράφονται κι αντίστοιχα το μέγεθος των αρχείων ή άλλων μέσων αποθήκευσης των καταγραφών αυξάνεται πολλαπλασιαστικά.

Επιπλέον, πέρα από τις τεχνολογικές μεταβολές και τις νέες δομές και αρχιτεκτονικές των βιομηχανικών συστημάτων λογισμικού, η υποχρέωση συμμόρφωσης των επιχειρήσεων και των οργανισμών που εξυπηρετούν, σε νομοθετικά και ρυθμιστικά πλαίσια (για παράδειγμα το *Sarbanes-Oxley Act* στις Η.Π.Α.) υπαγορεύει την ανάγκη για ενδελεχή παρακολούθηση και καταγραφή των διαφόρων συμβάντων που εμφανίζονται κατά τη λειτουργία του συστήματος, όπως και για αποτελεσματική ανάλυση των καταγραφών, για λόγους επιθεώρησης και ελέγχων συμμόρφωσης.

Η ανάγκη για ασφάλεια και αξιοπιστία στη λειτουργία των σύγχρονων βιομηχανικών συστημάτων λογισμικού επιβάλλει την αναζήτηση και την εφαρμογή κατάλληλων λύσεων για την παρακολούθηση, την καταγραφή και την ανάλυση της λειτουργίας τους.

1.3 Περιγραφή προβλήματος

Σύμφωνα με τα παραπάνω, οι μηχανικοί λογισμικού ιδιαιτέρως των μεγάλης κλίμακας, βιομηχανικών συστημάτων λογισμικού, καλούνται να απαντήσουν σε ορισμένα σημαντικά ερωτήματα:

1. Πως μπορούμε να διαπιστώσουμε ότι το σύστημα ανταποκρίνεται ορθά και παραδίδει το απαιτούμενο επίπεδο υπηρεσιών στους χρήστες και τα συνεργαζόμενα συστήματα;
2. Σε περίπτωση σφάλματος ή επίθεσης, πως μπορούμε να εντοπίσουμε «*τι πήγε λάθος*» ή «*τι συνέβη*» και ποιο ή ποια δομοστοιχεία αποτελούν την αιτία του προβλήματος;
3. Πέρα από τεχνικές διάγνωσης, ποιες τεχνικές πρόγνωσης και άμεσης αντίδρασης μπορούν να επιστρατευτούν;

1.3.1 Συνεισφορά διπλωματικής εργασίας

Στην παρούσα διπλωματική εργασία προτείνεται ένα περιβάλλον-πλαίσιο για την παρακολούθηση και την ανάλυση της λειτουργίας συστημάτων λογισμικού και παρουσιάζεται μια υλοποίηση του πλαισίου αυτού.

Συγκεκριμένα προτείνεται μια τεχνική ενορχήστρωσης χωρίς επέμβαση στον πηγαίο κώδικα (*non-intrusive*) συστημάτων υλοποιημένων σε Java για τη δημιουργία, τη συλλογή και την καταγραφή των γεγονότων ενδιαφέροντος. Παρουσιάζεται η διατύπωση των απαιτήσεων, των περιορισμών ή άλλων προτύπων αποδεκτής ή μη αποδεκτής λειτουργίας (στη συνέχεια της εργασίας αναφέρονται ως «*σενάρια ενδιαφέροντος*» ή απλώς «*σενάρια*») με τη χρήση Διαγραμμάτων Ακολουθίας της Ενοποιημένης Γλώσσας Μοντελοποίησης 2.0 (UML 2.0 Sequence Diagrams). Οι σημαντικές εκφραστικές δυνατότητες που παρέχουν τα διαγράμματα αυτά, σε συνδυασμό με τη διαδεδομένη αποδοχή και χρήση της UML ως βασικό εργαλείο μοντελοποίησης, καθιστούν τη χρήση τους ως ιδαιτέρως κατάλληλη για την έκφραση τέτοιων σεναρίων. Καθώς όμως οι δυνατότητες τυπικής ανάλυσης των UML μοντέλων είναι περιορισμένες, η ανάλυση αυτών των σεναρίων απαιτεί την ισοδύναμη έκφρασή του σε κάποιο τυπικό μοντέλο. Ορίζεται και υλοποιείται για το λογο αυτό μια γραμματική μετασχηματισμού των Διαγραμμάτων Ακολουθίας σε μια επέκταση των Δικτύων Petri που μπορούν να αναλυθούν και να εκτελεστούν. Στη συνέχεια της εργασίας ορίζεται ένας κατάλληλος αλγόριθμος και υλοποιείται η διαδικασία εκτέλεσης των Δικτύων Petri σύμφωνα:

1. με αρχεία καταγραφής γεγονότων που συλλέχθηκαν κατά τη λειτουργία (*log files*) για την εξαγωγή συμπερασμάτων ως προς την επαλήθευση των σεναρίων ή όχι
2. με αλληλουχίες γεγονότων που συλλέγονται στα πλαίσια μιας επιγραμμικής (*online*) αρχιτεκτονικής η οποία επιτρέπει την άμεση ανάλυση των αλληλουχιών αυτών και την ενημέρωση στην περίπτωση επαλήθευσης ή μη των σεναρίων σε πραγματικό χρόνο ή σε χρόνο κοντά στον πραγματικό

Η επισκόπηση της μεθόδου που προτείνεται, σε υψηλό, αφαιρετικό επίπεδο, δίνεται στο Σχήμα 1.1.

Σχήμα 1.1: Επισκόπηση της συνολική διαδικασίας που περιγράφεται στην παρούσα εργασία

Η συνεισφορά της εργασίας μπορεί να κωδικοποιηθεί στα εξής σημεία:

1. *Σύστημα παρακολούθησης και καταγραφής*: Δεδομένης κάποιας πολιτικής που καθορίζει το τι πρέπει να παρακολουθηθεί, χρησιμοποιείται μια τεχνική για την ενορχήστρωση λογισμικού υλοποιημένου σε Java χωρίς την επέμβαση σε κώδικα. Ορίζονται για το λόγο αυτό κατάλληλα *probes* μέσω των οποίων ρυθμίζεται το επίπεδο αφαίρεσης, ο μορφότυπος, το είδος των καταγραφών και οι τρόποι εξαγωγής των δεδομένων που συλλέγονται.

2. *Μετασχηματισμός Διαγραμμάτων Ακολουθίας σε Δίκτυα Petri*: Ορίζεται και υλοποιείται μια μέθοδος μετασχηματισμού των Διαγραμμάτων Ακολουθίας που μοντελοποιούν σενάρια ενδιαφέροντος για το σύστημα, σε Δίκτυα Petri που μπορούν να εκτελεστούν και να αναλυθούν τυπικά.

3. *Έλεγχος ταύτισης σεναρίων έναντι αρχείων καταγραφής*: Προτείνεται μια τεχνική ανάλυσης δεδομένων καταγραφής για την αναγνώριση εμφάνισης των σεναρίων ενδιαφέροντος κατά τη λειτουργία του συστήματος. Για το λόγο αυτό περιγράφεται και υλοποιείται μια διαδικασίας εκτέλεσης των Δικτύων Petri που έχουν προκύψει, με βάση τις πληροφορίες των αρχείων καταγραφής.

4. *Έλεγχος ταύτισης σεναρίων έναντι επιγραμμικής τροφοδότησης ακολουθιών γεγονότων*: Προτείνεται και υλοποιείται μια αρχιτεκτονική και μια αντίστοιχη με την παραπάνω τεχνική, για την ανάλυση ακολουθιών γεγονότων σε πραγματικό χρόνο η οποία μπορεί να χρησιμοποιηθεί για τον έγκαιρο εντοπισμό παρεκκλίσεων, παραβιάσεων και προβλημάτων και την άμεση αντίδραση στα φαινόμενα αυτά.

1.4 Οργάνωση κειμένου

Στο 2^{ο} κεφάλαιο παρουσιάζονται εργασίες σχετικές με το αντικείμενο της διπλωματικής και των τεχνολογιών που χρησιμοποιούνται στις περιοχές της παρακολούθησης και της ανάλυσης

συστημάτων λογισμικού, της επιτήρησης των απαιτήσεων του συστήματος, του εντοπισμού περιστατικών ασφάλειας και παρακολούθησης της επίδοσης και της αξιοπιστίας των συστημάτων. Επιπλέον, εξετάζονται εργασίες σχετικές με τον μετασχηματισμό Διαγραμμάτων Ακολουθίας της UML σε διαφόρους τύπους Δικτύων Petri. Το κεφάλαιο κλείνει με μια περιγραφή της επιστημονικής περιοχής του ελέγχου μοντέλου (*model checking*), γίνεται μια τοποθέτηση της διαδικασία ανάλυσης που προτείνεται στην παρούσα εργασία ως προς την παραπάνω περιοχή και εξετάζονται ορισμένες σχετικές εργασίες.

Στο 3^{ο} κεφάλαιο παρουσιάζονται βασικές έννοιες, πρότυπα και τεχνολογίες σχετικές με το πλαίσιο που περιγράφεται, απαραίτητες για την κατανόηση του περιβάλλοντος και της υποδομής στην οποία στηρίζονται οι λειτουργίες του πλαισίου.

Στο 4^{ο} κεφάλαιο παρουσιάζεται η μέθοδος έκφρασης και μετασχηματισμού των σεναρίων που μοντελοποιούν απαιτήσεις, περιορισμούς ή άλλα πρότυπα ενδιαφέροντος για το σύστημα. Συγκεκριμένα, παρουσιάζονται τα βασικά χαρακτηριστικά του μεταμοντέλου των Διαγραμμάτων Ακολουθίας (τμήμα του μεταμοντέλου της UML), περιγράφεται η προτεινόμενη επέκταση των Δικτύων Petri με το αντίστοιχο μεταμοντέλο, παρουσιάζεται η γραμματική του μετασχηματισμού, περιγράφεται η διαδικασία στο σύνολό της και συζητούνται σχετικά όρια και περιορισμοί.

Στο 5^{ο} κεφάλαιο παρουσιάζεται η μέθοδος παρακολούθησης, καταγραφής και ανάλυσης των ακολουθιών γεγονότων και συγκεκριμένα παρατίθεται ο μορφότυπος των λημμάτων των καταγραφών, η μέθοδος παραγωγής και συλλογής των γεγονότων ενδιαφέροντος χωρίς επέμβαση στον πηγαίο κώδικα (για λογισμικό σε Java), η διαδικασία ταιριάσματος σχηματομορφών με εκτέλεση των Δικτύων Petri σύμφωνα με τις ακολουθίες των γεγονότων που έχουν συλλεχθεί, οι επιλογές ανάλυσης αρχείων καταγραφής και επιγραμμικής ανάλυσης ακολουθιών γεγονότων και, τέλος, δίνονται ορισμένα παραδείγματα για τον έλεγχο και την αποτίμηση της μεθόδου.

Στο 6^{ο} κεφάλαιο παρουσιάζεται διαγραμματικά η μεθοδολογία και η αρχιτεκτονική του περιβάλλοντος, δίνονται τεχνικές λεπτομέρειες ως προς την υλοποίηση του πλαισίου, μελετάται η συμπεριφορά του αλγορίθμου εκτέλεσης σε συνάρτηση με την πολυπλοκότητα των σεναρίων και περιλαμβάνονται ορισμένες λήψεις οθόνης των εργαλείων που δημιουργήθηκαν ή/και χρησιμοποιούνται.

Η εργασία ολοκληρώνεται στο 7^{ο} κεφάλαιο όπου καταγράφονται τα συμπεράσματα που προέκυψαν, τα σημεία των τεθέντων στόχων που επιτεύχθηκαν και σε ποιο βαθμό, και κάποιες ιδέες για μελλοντική επέκταση και εξέλιξη του περιβάλλοντος-πλαισίου.

Κεφάλαιο 2: Σχετικές εργασίες

Στο κεφάλαιο αυτό παρουσιάζονται σχετικές εργασίες με τον παρόντα τόμο. Αρχικά παρουσιάζονται εργασίες που αφορούν σε μεθόδους και συστήματα παρακολούθησης και καταγραφής συστημάτων λογισμικού. Στη συνέχεια, παρουσιάζονται εφαρμογές της ανάλυσης ακολουθιών γεγονότων και συγκεκριμένα περιγράφονται μέθοδοι, περιβάλλοντα-πλαίσια και εργαλεία, που έχουν προταθεί από επιστημονικές ομάδες για: (α) την επαλήθευση των απαιτήσεων ενός συστήματος ή τον εντοπισμού αποκλίσεων από την προδιαγεγραμμένη συμπεριφορά, (β) τον έλεγχο της επίδοσης και της αξιοπιστίας συστημάτων λογισμικού, (γ) την έκφραση και την ανάλυση εντοπισμού εισβολών και περιστατικών ασφάλειας.

Κατόπιν, αναφέρονται εργασίες που περιλαμβάνουν κανόνες και διαδικασίες μετασχηματισμών Διαγραμμάτων Ακολουθίας (διαφόρων εκδόσεων) σε διαφόρους τύπους Δικτύων Petri και σχολιάζονται σε σχέση με τον προτεινόμενο μετασχηματισμό.

Το κεφάλαιο κλείνει κάνοντας μια σύντομη αναφορά στις μεθόδους ελέγχου μοντέλων και ανάλυσης Δικτύων Petri, για την επαλήθευση ή την απόρριψη ιδιοτήτων.

2.1 Παρακολούθηση συστημάτων και ανάλυση ακολουθιών γεγονότων

Η παρακολούθηση (*monitoring*) συστημάτων λογισμικού είναι η διαδικασία παρατήρησης γεγονότων και διεργασιών ενός συστήματος σύμφωνα με κάποια καθορισμένη πολιτική παρακολούθησης. Η ανάλυση των ακολουθιών των καταγεγραμμένων γεγονότων μπορεί να χρησιμοποιηθεί για την κατανόηση και την αποσφαλμάτωση των εφαρμογών, για τη διασφάλιση της επίδοσης και της ποιότητας των συστημάτων, για τον εντοπισμό προβλημάτων και την ειδοποίηση των διαχειριστών, για τον εντοπισμό παραβιάσεων των απαιτήσεων όπως και για την ενίσχυση της ασφάλειας των εφαρμογών και των συστημάτων.

Για την επίτευξη των παραπάνω στόχων έχουν προταθεί διάφορες μέθοδοι, τεχνικές και συστήματα.

Ως προς την παρακολούθηση εκτέλεσης προγράμματος, στην έρευνα που διεξήγαγαν το 1981 οι Plattner και Nievergelt [PN81] παρουσίασαν τις ως τότε εξελίξεις στον τομέα, αναδεικνύοντας τη σπουδαιότητά της παρακολούθησης εκτέλεσης προγράμματος ως εργαλείο τόσο για την ανάπτυξη όσο και για τη συντήρηση του λογισμικού. Προέβλεψαν ότι η κρισιμότητα του ρόλου της παρακολούθησης δε θα μειώνονταν καθώς η κατανόηση που παρέχει δε μπορεί να αντικατασταθεί από τη στατιστική ανάλυση του κειμένου του προγράμματος. Σημείωσαν επίσης ότι το εμπορικό λογισμικό της εποχής σπάνια υποστήριζε ικανοποιητικά την παρακολούθηση κατά το χρόνο εκτέλεσης και επισήμαναν την ανάγκη για δραστικές αλλαγές που έπρεπε να γίνουν ως προς τις τότε πρακτικές για υλοποιήσεις μεταγλωττιστών και υλικού οι οποίες θα παρείχαν δυνατότητες αναλυτικής παρακολούθησης. Τέλος, ανέφεραν ότι η παρακολούθηση εκτέλεσης δεν είχε ως τότε λάβει από την ερευνητική κοινότητα την προσοχή που αναλογεί στην πρακτική της σημασία, καθώς στη διαθέσιμη βιβλιογραφία περιλαμβάνονταν πολλές μελέτες περιπτώσεων αλλά χωρίς επαρκή συζήτηση ως προς τις θεμελιώδεις έννοιες, τους στόχους και τους περιορισμούς.

Τα επόμενα χρόνια η έρευνα στο συγκεκριμένο χώρο στράφηκε στη δημιουργία συστημάτων παρακολούθησης βασισμένων είτε στην ενορχήστρωση του πηγαίου κώδικα των προγραμμάτων, είτε στην ενορχήστρωση κάποιου διερμηνευτή ώστε να περιληφθεί η δραστηριότητα παρακολούθησης. Η ενορχήστρωση κώδικα ήταν και συνεχίζει να είναι μια από τις βασικές τεχνικές που χρησιμοποιούνται για την παρακολούθηση καθώς οι προγραμματιστές των εφαρμογών είναι εκείνοι που συνήθως ξέρουν καλύτερα τι χρειάζεται να παρακολουθηθεί ώστε από τα δεδομένα που συλλέγονται να μπορούν να βγούνε συμπεράσματα. Παρόλα αυτά, η ενορχήστρωση κώδικα προϋποθέτει την πρόσβαση στον πηγαίο κώδικα της εφαρμογής ενώ ταυτόχρονα απαιτεί σημαντική προσπάθεια από τον προγραμματιστή και συχνά βαθιά γνώση του συστήματος που ενορχηστρώνεται ώστε να γίνεται αποτελεσματικά. Επιπλέον, τέτοιες προσεγγίσεις εμφάνιζαν συχνά περιορισμούς που αφορούσαν στην αλληλεπίδραση τους τόσο με την κανονική εκτέλεση του προγράμματος όσο και με άλλες διαδικασίες παρακολούθησης. Τέλος, η τυπική τεκμηρίωση των μεθόδων αυτών ήταν ελάχιστη και συνήθως βασίζονταν σε «χειροποίητες» τεχνικές [KHC91].

Το 1991 οι Koshi, Hudak και Consel παρουσιάζουν στην εργασία τους [KHC91] ένα τυπικό πλαίσιο για τον προσδιορισμό, την υλοποίηση και την τεκμηρίωση επιτηρητών (*monitor*) εκτέλεσης. Συγκεκριμένα εισάγουν τον όρο «*σημασιολογία παρακολούθησης*» (*monitoring semantics*) και τον ορίζουν ως μια συντηρητική επέκταση της βασικής δηλωτικής σημασιολογίας μιας γλώσσας, η οποία είναι παραμετροποιημένη σύμφωνα με προδιαγραφές

κάποιας δραστηριότητας παρακολούθησης. Το πλαίσιο που εισηγούνται είναι γενικό υπό την έννοια ότι μια σημασιολογία παρακολούθησης μπορεί αυτόματα να εξαχθεί από κάθε δηλωτική σημασιολογία συνέχισης (*denotational continuation semantics*). Η σημασιολογία αυτή μπορεί να περικλείσει κάθε ακολουθιακή, ντετερμινιστική δραστηριότητα παρακολούθησης, είναι συνθετική και μπορεί να διατηρήσει τη στάνταρ σημασιολογία. Πρακτικά μέσω μετασχηματισμού «*κώδικα σε κώδικα*», χρησιμοποιώντας τμηματική αποτίμηση (*partial evaluation*), μπορούν να παραχθούν ενορχηστρωμένοι διερμηνευτές όπως και ενορχηστρωμένα προγράμματα από μια σημασιολογία παρακολούθησης σε μια ενιαία βάση.

Μια σχετική προσέγγιση στην παρακολούθηση προγραμμάτων ακολουθεί και ο Thiemann στην εργασία του [Thi03] στην οποία χρησιμοποιείται τη μερική αποτίμηση ως εργαλείο για τον μετασχηματισμό διερμηνευτών εφοδιασμένων με κώδικα παρακολούθησης σε μη πρότυπους (non standard) μεταφραστές. Οι μεταφραστές αυτοί παράγουν τελικά κώδικα εφαρμογής που ενσωματώνει κώδικα παρακολούθησης. Ο μετασχηματισμός τρέχει σε γραμμικό χρόνο και παράγει κώδικα γραμμικώς ανάλογο ως προς το μέγεθος του αρχικού προγράμματος.

Οι Jahier και Ducasse στην εργασία τους [JD02] προτείνουν μια εναλλακτική στην επέμβαση τόσο σε επίπεδο κώδικα όσο και σε επίπεδο μεταγλωττιστή για την εξυπηρέτηση της παρακολούθησης κι επιστρατεύουν την ανάλυση ιχνών εκτέλεσης όπως κλήσεων διαδικασιών. Αρχικά περιγράφουν ένα αρχέτυπο, υψηλού επιπέδου χειριστή ιχνών, μέσω του οποίου ο χρήστης μπορεί να καθορίσει τι θέλει να παρακολουθείται. Το πλαίσιο που προτείνουν στηρίζεται στην ύπαρξη ενός ανιχνευτή χαμηλού επιπέδου (*tracer*) και στον πλήρη διαχωρισμό της διαδικασία ορισμού των monitors από τον πηγαίο κώδικα και από τον μεταγλωττιστή της γλώσσας.

Ο Hofmann και οι συνεργάτες του, στην εργασία τους [HKM+94] παρουσιάζουν μια μέθοδο για την ανάλυση της λειτουργικής συμπεριφοράς και της επίδοσης των προγραμμάτων σε κατανεμημένα συστήματα. Χρησιμοποιούν μια τεχνική υβριδικής παρακολούθησης που συνδυάζει πλεονεκτήματα της παρακολούθησης λογισμικού και της παρακολούθησης υλικού. Πιο συγκεκριμένα, παρουσιάζεται το πακέτο ZK4/SIMPLE το οποίο χρησιμοποιείται για την υβριδική παρακολούθηση και για την ανάλυση των ακολουθιών των γεγονότων που συλλέγονται. Στην υβριδική παρακολούθηση η ενορχήστρωση των προγραμμάτων γίνεται με εντολές που γράφουν γεγονότα σε μια διεπαφή υλικού, συνδυάζοντας έτσι τη δήλωση και τον προσδιορισμό των γεγονότων σε επίπεδο κώδικα -δηλαδή με τρόπο φιλικό προς τον προγραμματιστή- και την μικρή επίδραση της διαδικασίας στη συμπεριφορά του συστήματος, καθώς τα γεγονότα δεν γράφονται σε περιοχή μνήμης του υπό παρακολούθηση συστήματος.

Η ανάλυση ακολουθιών γεγονότων βρίσκει σημαντική εφαρμογή στη μηχανική απαιτήσεων και συγκεκριμένα στον τομέα της παρακολούθησης απαιτήσεων κατά το χρόνο εκτέλεσης (*run-time requirements monitoring*). Η παρακολούθηση απαιτήσεων κατά το χρόνο εκτέλεσης είναι η διαδικασία επιτήρησης της συμπεριφοράς ενός συστήματος κατά το χρόνο εκτέλεσης κατά την οποία επισημαίνονται οι περιπτώσεις κατά τις οποίες το σύστημα παρεκκλίνει από απαιτήσεις που έχουν τεθεί για αυτό. Μπορεί μάλιστα να θεωρηθεί ως μια γέφυρα που επιχειρεί να καλύψει το κενό μεταξύ των μεθόδων τυπικής επαλήθευσης, οι οποίες διασφαλίζουν την ορθότητα περισσότερο ενός σχεδιασμού παρά μιας υλοποίησης, και του ελέγχου (*testing*) ο οποίος δεν παρέχει τυπικές εγγυήσεις ως προς την ορθότητα του συστήματος. Για την υποστήριξη της ανάλυσης γίνονται υποθέσεις, ως τμήμα της διαδικασίας ορισμού των απαιτήσεων.

Μια από τις πρώτες εργασίες εστιασμένη στην περιγραφή της παρακολούθησης απαιτήσεων κατά το χρόνο εκτέλεσης είναι η [FF95] όπου οι Fickas και Feather απευθύνουν και στη συνέχεια εξετάζουν ορισμένα θεμελιώδη ερωτήματα που δίνουν τη βάση και κίνητρα για την διερεύνηση της περιοχής (σε ελεύθερη μετάφραση):

«Πρώτον, πως μπορούμε να ξέρουμε πότε το σύστημά μας χρειάζεται να εξελιχθεί; Μπορούν οι υποθέσεις πόρων και λειτουργικών αναγκών που έγιναν κατά το χρόνο σχεδιασμού να «μεταφερθούν» και ώστε να ελεγχθούν κατά το χρόνο εκτέλεσης, κι αν ναι πως; Πως μπορούμε δηλαδή να αναγνωρίζουμε το αν οι υποθέσεις που έγιναν κατά το σχεδιασμό του ενός συστήματος καθίστανται άκυρες όσο το σύστημα βρίσκεται σε λειτουργία.

Δεύτερον, υποθέτοντας ότι μπορούμε να εντοπίσουμε διαφοροποιήσεις στο περιβάλλον, οι οποίες μάλιστα επιτάσσουν την εξέλιξη του συστήματός μας, πως μπορούμε να χρησιμοποιήσουμε τη διαθέσιμη αυτή πληροφορία για να οργανώσουμε ορθότερα την εξέλιξη; Σε μια ιδανική περίπτωση αυτή η διαδικασία θα ήταν αυτοματοποιημένη: οι πληροφορίες παρακολούθησης θα καταναλώνονταν από το σύστημα καθαυτό, το οποίο θα προσάρμοζε την ίδια του τη δομή ή τη λειτουργικότητα. Εναλλακτικά και περισσότερο πρακτικά, οι πληροφορίες αυτές θα μπορούσαν να παρασχεθούν στους σχεδιαστές ή στους διαχειριστές του συστήματος βοηθώντας κι ενίοτε καθοδηγώντας τους στο έργο της εξέλιξης του συστήματός τους.»

Στα πλαίσια αυτής της περιοχής της μηχανικής απαιτήσεων, η έρευνα που έχει γίνει χωρίζεται σε τέσσερις βασικές κατευθύνσεις (σύμφωνα με την εργασία [SM04]), οι οποίες είναι:

1. ο τρόπος καθορισμού των απαιτήσεων και ο μετασχηματισμός αυτών σε γεγονότα τα οποία μπορούν να παρακολουθηθούν κατά το χρόνο εκτέλεσης.
2. η ανάπτυξη μηχανισμών παρακολούθησης γεγονότων.
3. η ανάπτυξη μηχανισμών για τη γένεση γεγονότων συστήματος που μπορούν να χρησιμοποιηθούν για την παρακολούθηση

4. η ανάπτυξη μηχανισμών για την προσαρμογή των συστημάτων ούτως ώστε να χειρίζονται παρεκκλίσεις από τις απαιτήσεις κατά τον χρόνο εκτέλεσης.

Οι Fickas και Feather πρότειναν μια αρχιτεκτονική που επιτρέπει τη δυναμική διαμόρφωση συστημάτων βασισμένων σε στοιχεία λογισμικού, ανάλογα με την ικανοποίηση ή μη των απαιτήσεων κατά τη διάρκεια της εκτέλεσης [FF95]. Στην εργασία αυτή εφαρμόζουν τη παρακολούθηση ορισμένων απαιτήσεων που έχουν τεθεί για έναν εξυπηρετητή αδειών λογισμικού. Όταν ο εξυπηρετητής δε καταφέρνει να ανταποκριθεί σε αυτές τις απαιτήσεις (όπως για παράδειγμα στην απαίτηση σε ένα χρήστη να αποδίδεται άδεια στο 90% των αιτημάτων του) εξαιτίας μιας αλλαγής στο περιβάλλον του συστήματος, το σύστημα ειδοποιεί τον διαχειριστή.

Οι ίδιοι ερευνητές μαζί με τους Cohen και Narayanaswamy παρουσιάζουν ένα σύστημα [CFN+97] με το οποίο είναι δυνατή η παρακολούθηση απαιτήσεων και υποθέσεων κατά την εκτέλεση, το *AMOS*. Συγκεκριμένα, για τη συλλογή δεδομένων χρησιμοποιείται ένας μηχανισμός που λειτουργεί μεταξύ του συστήματος που παρακολουθείται και του περιβάλλοντός του συγκεντρώνοντας, για παράδειγμα μηνύματα που ανταλλάσσει το σύστημα με το περιβάλλον του. Στο AMOS ο χρήστης δηλώνει τις απαιτήσεις και τις υποθέσεις του για το σύστημα, στην εξειδικευμένου σκοπού γλώσσα *FLEA* (*Formal Language for Expressing Assumptions*) και ο μεταγλωττιστής της γλώσσας μετατρέπει τις εκφράσεις σε κώδικα παρακολούθησης. Ο μηχανισμός που χρησιμοποιείται για την υλοποίηση του monitor είναι μια ενεργή βάση δεδομένων, δηλαδή μια βάση δεδομένων με triggers η οποία εξετάζει κάθε συναλλαγή για να καθορίσει αν και πότε οι συνθήκες triggering επαληθεύονται.

Οι Σπανουδάκης και Mahbub παρουσιάζουν ένα περιβάλλον-πλαίσιο για την παρακολούθηση απαιτήσεων κατά το χρόνο εκτέλεσης, για συστήματα προσανατολισμένα σε υπηρεσίες [SM04]. Οι απαιτήσεις στο πλαίσιο αυτό καθορίζονται μέσω του λογισμού γεγονότων (*event calculus*) και παίρνουν τη μορφή ιδιοτήτων συμπεριφοράς και υποθέσεων τόσο ως προς τα συστήματα καθαυτά όσο και ως προς τις ατομικές υπηρεσίες και τους πράκτορες που αλληλεπιδρούν με αυτά. Η εξαγωγή των ιδιοτήτων συμπεριφοράς προκύπτει από την προδιαγραφή της διαδικασίας σύνθεσης ενός συστήματος βασισμένου σε υπηρεσίες. Αυτό πρακτικά σημαίνει ότι οι ιδιότητες διατυπώνονται σε όρους γεγονότων που συμβαίνουν κατά την αλληλεπίδραση της διαδικασίας σύνθεσης και των συστατικών υπηρεσιών του συστήματος, τα οποία γεγονότα περιλαμβάνονται στις καταγραφές τους συστήματος. Οι υποθέσεις προς παρακολούθηση, ορίζονται ανάλογα σε όρους αναγνωρίσιμων γεγονότων σε επίπεδο συστήματος.

Στην εργασία [Rob02] ο Robinson παρουσιάζει ένα πλαίσιο για την παρακολούθηση των απαιτήσεων λογισμικού καθώς αυτό εκτελείται, μέσω της ενορχήστρωσης κώδικα. Η

υλοποίηση του πλαισίου ονομάζεται *ReqMon* και η διαδικασία για την παραγωγή ενός monitor και τον έλεγχο για παραβίαση των απαιτήσεων είναι η εξής: Αρχικά καθορίζονται οι απαιτήσεις από τις οποίες παράγεται ένα UML μοντέλο. Το UML μοντέλο χρησιμοποιείται για την εξαγωγή Java κώδικα, ο οποίος ενορχηστρώνεται για να παράγει γεγονότα που μπορεί να επεξεργαστεί το monitor. Έπειτα, περιοδικά, από το κώδικα του προγράμματος και την ακολουθία των γεγονότων, παράγεται ένα μοντέλο monitor (*monitor model*) το οποίο ελέγχεται για την αποτυχία ύποπτων συνθηκών και απαιτήσεων, χρησιμοποιώντας το εργαλείο ελέγχου μοντέλων *Spin*.

Μια ακόμη προσέγγιση γίνεται στην εργασία [MMK+04] όπου παρουσιάζεται το Java-MaC περιγράφεται το οποίο αποτελεί μια υλοποίηση της Monitoring and Checking Architecture (MaC) για προγράμματα Java. Οι Moonzoo, Mahesh, Kannan, Lee kai Sokolsky προτείνουν τη MaC αρχιτεκτονική ως μια «ελαφρά» (lightweight) τυπική μέθοδο σε αντίθεση με τις παραδοσιακές τυπικές μεθόδους που υπάρχουν. Στη MaC οι διαδικασίες ενορχήστρωσης, παρακολούθησης και ελέγχου εφαρμόζονται πλήρως αυτοματοποιημένα και χωρίς την ανθρώπινη καθοδήγηση, πράγμα που αυξάνει την ακρίβεια της ανάλυσης. Ένα δεύτερο χαρακτηριστικό της μεθόδου είναι ο ξεκάθαρος διαχωρισμός μεταξύ της παρακολούθησης των εξαρτώμενων από την υλοποίηση, χαμηλού επιπέδου συμπεριφορών και του ελέγχου των υψηλού επιπέδου συμπεριφορών. Το γεγονός αυτό επιτρέπει την επαναχρησιμοποίηση των υψηλού επιπέδου περιγραφών απαιτήσεων ακόμα κι αν η υλοποίηση του υπό παρακολούθηση προγράμματος αλλάζει.

Πέρα από την μηχανική απαιτήσεων η ανάλυση καταγραφών χρησιμοποιείται και για την αναβάθμιση της ασφάλειας των συστημάτων μέσω τον εντοπισμό ύποπτων ή μη αποδεκτών συμπεριφορών, προσπαθειών εισβολής (*intrusion detection*) και γενικότερα περιστατικών ασφαλείας (*security incidents*). Οι Kemmerer και Vigna κάνουν μια παρουσίαση της περιοχής στο άρθρο τους [KV02]. Ένα κρίσιμο ζήτημα για τα συστήματα εντοπισμού ύποπτων συμπεριφορών και παραβιάσεων αφορά στην αξιοπιστία και στην πληρότητα των δεδομένων που συλλέγουν και αναλύουν. Τα περισσότερα λειτουργικά συστήματα παρέχουν καταγραφές των λειτουργιών των διαφόρων χρηστών αλλά πολλές φορές είτε δε περιλαμβάνονται αρκετές πληροφορίες σχετικές με την ασφάλεια, όπως για παράδειγμα αποτυχημένες προσπάθειες εισόδου στο σύστημα, είτε περιλαμβάνουν πλήρεις αναφορές για κάθε κλήση του συστήματος από κάθε διεργασία. Καταγραφές ακολουθιών γεγονότων παρέχουν επίσης και στοιχεία δικτύου όπως δρομολογητές (*routers*) και τείχη προστασίας (*firewalls*). Οι καταγραφές αυτές σχετίζονται με τη δικτυακή δραστηριότητα και περιλαμβάνουν συνήθως τα ανοίγματα και τα κλεισίματα των συνδέσεων κι ενίοτε πλήρεις καταγραφές των πακέτων που διαχειρίζονται. Η απόφαση του ποια γεγονότα θα καταγραφούν για να αναλυθούν και ποια

όχι κρίνει σε σημαντικό βαθμό την αποτελεσματικότητα των εργαλείων αυτών ως προς τα μοντέλα απειλών (*threat models*) που αφορούν ένα σύστημα και ταυτόχρονα το επίπεδο λεπτομέρειας των καταγραφών επηρεάζει συχνά την επίδοση του συστήματος και τις ανάγκες του σε πόρους (ιδιαιτέρως σε αποθηκευτικούς χώρους). Οι κύριες διαγνωστικές τεχνικές είναι ο εντοπισμός ανωμαλιών (*anomaly detection*) και ο εντοπισμός καταχρήσεων (*misuse detection*). Στη πρώτη τεχνική χρησιμοποιούνται μοντέλα που περιγράφουν την «*κανονική*» συμπεριφορά των χρηστών και των εφαρμογών και οι αποκλίσεις από αυτή τη συμπεριφορά μεταφράζονται ως «*πρόβλημα*». Η προσέγγιση αυτή στηρίζεται στην υπόθεση ότι οι επιθέσεις αφήνουν διαφορετικά ίχνη από την αποδεκτή χρήση και αποδεικνύεται αποτελεσματική ιδιαιτέρως σε είδη επιθέσεων που δεν ήταν προηγουμένως γνωστά. Παρόλα αυτά, η εκπαίδευση τέτοιων συστημάτων σε συστήματα δυναμικών περιβαλλόντων είναι αρκετά δύσκολη και συχνά τα ποσοστά λανθασμένων ειδοποιήσεων είναι υψηλά. Στην τεχνική εντοπισμού καταχρήσεων ορίζονται οι περιγραφές των επιθέσεων, ή αλλιώς οι «*υπογραφές*» (*signatures*), ώστε να συγκριθούν με τα δεδομένα που συλλέγονται και να εντοπιστούν περιστατικά επιθέσεων γνωστού τύπου. Η δεύτερη αυτή τεχνική αν και εμφανίζει χαμηλότερα ποσοστά λανθασμένων ειδοποιήσεων σε σχέση με την τεχνική εντοπισμού ανωμαλιών, δε μπορεί να ανιχνεύσει επιθέσεις νέων τύπων. Αν παρόλα αυτά οι νέοι τύποι επιθέσεων αναγνωριστούν με άλλους τρόπους, αφού μοντελοποιηθούν μπορούν να εισαχθούν στη βάση περιγραφών ενός τέτοιου συστήματος.

Τα ανοιχτά ζητήματα στην περιοχή αφορούν κυρίως: (α) την αποτελεσματικότητα των συστημάτων, η οποία στην ιδανική περίπτωση θα σήμαινε εντοπισμό όλων των επιθέσεων και περιορισμό στο ελάχιστο των λανθασμένων ειδοποιήσεων, (β) την επίδοση, το επίπεδο της οποίας είναι δύσκολο να διατηρηθεί όσο αυξάνονται οι ταχύτητες των δικτύων, η επεξεργαστική ικανότητα των κόμβων και το μέγεθος των καταγραφών, και (γ) τη δυνατότητα ανάλυσης σε όλο το εύρος του δικτύου η οποία απαιτεί τη συσχέτιση γεγονότων και το συνδυασμό ακολουθιών γεγονότων που συμβαίνουν σε διαφορετικούς κόμβους κι αποτελούν ενδεχομένως στάδια-βήματα μιας γενικότερης επίθεσης.

Μια σχετική εργασία είναι αυτή των Kumar και Spafford [KS94] η οποία προσεγγίζει το πρόβλημα του εντοπισμού καταχρήσεων μέσω μιας μεθόδου ταύτισης σχηματομορφών. Συγκεκριμένα προτείνεται ένα γενικό μοντέλο ταύτισης που βασίζεται σε Έγχρωμα Δίκτυα Petri (*Colored Petri Nets - CPN*) το οποίο μπορεί να προσαρμοσθεί σε αρκετά πεδία προβλημάτων. Στην εργασία προτείνεται η χρήση ιχνών ελέγχου (*audit trails*) του UNIX ως παράδειγμα για την ανάδειξη της χρησιμότητας και του εφαρμοστέου της προσέγγισης.

Στην ίδια περιοχή κινείται και η εργασία των Ho, Frinke και Tobin [HFT98]. Οι συγγραφείς προτείνουν μια αρχιτεκτονική εντοπισμού εισβολής που συνδυάζει τον μερικής διάταξης σχεδιασμό (*planning*) και εκτελέσιμα Δίκτυα Petri για να εντοπίσει επιθέσεις. Η προσέγγιση

μπορεί να χρησιμοποιηθεί για τον εντοπισμό καταχρήσεων και η τεχνική ανάλυσης που χρησιμοποιείται ονομάζεται *POSTAT* (*Partial Order State Transition Analysis Technique*). Οι συγγραφείς παρουσιάζουν επίσης τη δυνατότητα χρήσης της μεθόδου και για τον εντοπισμό ανωμαλιών, στην οποία όμως δεν περιλαμβάνεται κάποιος μηχανισμός μάθησης για τον καθορισμό της αποδεκτής συμπεριφοράς. Η διαδικασία εντοπισμού εισβολών έχει ως εξής: Αρχικά δηλώνονται τα πρότυπα επίθεσης τα οποία εκφράζονται χρησιμοποιώντας μια περιγραφή λογικής πρώτης τάξης (*first-order logic*) των γνωστών δραστηριοτήτων και στόχων του εισβολέα. Η περιγραφή αυτή ονομάζεται «*πλάνο*», γνωστή από το χώρο της τεχνητής νοημοσύνης [RN95]. Στη συνέχεια δημιουργείται μια απεικόνιση των σεναρίων αυτών σε Δίκτυα Petri και τα οποία τελικά εκτελούνται με βάση ακολουθίες ιχνών που έχουν συλλεχθεί κατά τη λειτουργία του συστήματος. Κάθε Δίκτυο Petri ξεκινά από μια αρχική κατάσταση και υπάρχουν συγκεκριμένες τελικές καταστάσεις που αναπαριστούν την εμφάνιση του σεναρίου. Η τεχνική αυτή αυξάνει την ευελιξία της παραδοσιακής προσέγγισης ανάλυσης καταστάσεων, προσφέροντας τη δυνατότητα δήλωσης γεγονότων που δε βρίσκονται σε διάταξη στην ακολουθία ενεργειών που χαρακτηρίζει μια υπογραφή επίθεσης.

Οι τελευταίες δύο εργασίες που περιγράφηκαν εμφανίζουν ορισμένα κοινά σημεία με τη προσέγγιση που γίνεται στην παρούσα εργασία ως προς τη χρήση Δικτύων Petri για τη ταύτιση σχηματομορφών. Στην [KS94] αν και η προσέγγιση που γίνεται είναι ενδιαφέρουσα σε θεωρητικό επίπεδο δεν παρουσιάζεται κάποια υλοποίηση και όπως αναγνωρίζουν οι συγγραφείς, η ουσιαστική αποτίμηση μπορεί να γίνει μόνο αν ένα υλοποιημένο σύστημα δοκιμαστεί υπό πραγματικές συνθήκες εντοπισμού εισβολής. Επίσης, τόσο ως προς την [KS94] όσο και ως προς τη [HFT98], η χρήση Διαγραμμάτων Ακολουθίας, παρουσιάζει πλεονεκτήματα σε σχέση με την ευκολία διατύπωσης σεναρίων. Τέλος, μια παράπλευρη διαφορά είναι ότι η γραμματική που υλοποιείται για τον μετασχηματισμό των σεναρίων σε Δίκτυα Petri διασφαλίζει ότι έχουμε μια και μοναδική αρχική κατάσταση για το δίκτυο - πράγμα που δεν ισχύει για τα μερικής διάταξης δίκτυα που συντίθενται και στις δύο εργασίες.

Μια άλλη ιδιαίτερα διαδεδομένη εφαρμογή των τεχνικών καταγραφής και ανάλυσης ακολουθιών γεγονότων είναι η εξόρυξη δεδομένων χρήσης ιστού (*web usage mining*). Αρχεία ή βάσεις δεδομένων καταγραφής τηρούνται από τους εξυπηρετητές ιστού, στα οποία καταγράφονται οι αιτήσεις που δέχονται οι εξυπηρετητές με στοιχεία όπως ο χρόνος, τα στοιχεία του καλούντα, το περιεχόμενο της αίτησης και άλλα. Εφαρμόζοντας τεχνικές της περιοχής εξόρυξης δεδομένων και μηχανισμών μάθησης είναι δυνατή η εξαγωγή συμπερασμάτων για τα πρότυπα χρήσης-πλοήγησης και ενδείξεις προτιμήσεων των χρηστών. Τα πρότυπα αυτά μπορούν στη συνέχεια να χρησιμοποιηθούν μέσω τεχνικών ταύτισης σχηματομορφών για να προσφέρουν στις επιχειρήσεις και τους οργανισμούς σημαντικές

δυνατότητες βελτίωσης του επιπέδου και της αποτελεσματικότητας των υπηρεσιών που παρέχουν (για παράδειγμα μέσω τεχνικών προσωποποίησης, συστημάτων προτάσεων κλπ). Μια βιβλιογραφική μελέτη στην περιοχή του web usage mining (και του web mining, γενικότερα) υπάρχει στο [Ath07].

2.2 Μετασχηματισμοί μοντέλων

Ο μετασχηματισμός Διαγραμμάτων Ακολουθίας σε Δίκτυα Petri αποτέλεσε αντικείμενο αρκετών εργασιών, κυρίως λόγω της έλλειψης τυπικής σημασιολογίας της UML που έχει ως συνέπεια να μην είναι συνήθως δυνατή η απευθείας εφαρμογή μαθηματικών τεχνικών ανάλυσης των μοντέλων. Κάθε προσέγγιση είχε συνήθως διαφορετική οπτική και ζητούμενα. Στις επόμενες παραγράφους παρουσιάζονται σχετικές εργασίες που βρέθηκαν στη βιβλιογραφία και τελικά σχολιάζεται το πως ο μετασχηματισμός που προτείνεται στην παρούσα εργασία εξυπηρετεί τους στόχους της μοντελοποίησης σεναρίων για την παρακολούθηση και την ανάλυση της συμπεριφοράς συστημάτων λογισμικού.

Μια από τις πρώτες σχετικές εργασίες είναι η [EK98] των Elkoutbi και Keller οι οποίοι παρουσίασαν ένα τρόπο για την προδιαγραφή της συμπεριφοράς ενός συστήματος χρησιμοποιώντας Ιεραρχικά Έγχρωμα Δίκτυα Petri (*Hierarchical Colored Petri Nets - HCPN*). Η προσέγγισή τους γίνεται σε επίπεδο συμπεριφοράς ολόκληρου του συστήματος και όχι των στοιχείων που το απαρτίζουν. Συγκεκριμένα χρησιμοποιούν διαγράμματα περιπτώσεων χρήσης (*Use Case Diagrams*) που απεικονίζουν τις περιπτώσεις χρήσεις του συστήματος και εξάγουν από αυτά ένα απλό Δίκτυο Petri. Στο δεύτερο στάδιο της διαδικασίας αποσαφηνίζονται οι σχέσεις μεταξύ των χρήσεων και ενημερώνεται το αρχικό δίκτυο. Ακολουθεί η μοντελοποίηση των σεναρίων για κάθε περίπτωση χρήσης (ενδεχομένως περισσότερα του ενός για κάθε χρήση) μέσω «*διαγραμμάτων σεναρίων*» -τα οποία διαθέτουν ένα υποσύνολο των χαρακτηριστικών των Διαγραμμάτων Ακολουθίας. Τέλος, γίνεται η ενσωμάτωση αυτών των σεναρίων για κάθε περίπτωση χρήσης, μέσω του μετασχηματισμού τους σε Ιεραρχικά Έγχρωμα Δίκτυα Petri.

Το 2002 οι Bernardi, Donatelli και Merseguer μελέτησαν τη χρήση Διαγραμμάτων Ακολουθίας και Διαγραμμάτων Καταστάσεων (*Statechart Diagrams*) της UML για την επαλήθευση και την αποτίμηση επίδοσης των συστημάτων [BDM02]. Η μελέτη αφορά σε μοντέλα που περιγράφονταν από την τότε τρέχουσα έκδοση της UML που ήταν η 1.4. Η διαδικασία που πρότειναν ήταν η συνδυαστική χρήση των διαγραμμάτων αυτών για την περιγραφή του συστήματος. Τα Διαγράμματα Κατάστασης χρησιμοποιήθηκαν για τη μοντελοποίηση της συμπεριφοράς κάθε κλάσης και τα Διαγράμματα Ακολουθίας για την

απόδοση έμφασης σε συγκεκριμένα πρότυπα αλληλεπιδράσεων μεταξύ των Διαγραμμάτων Κατάστασης. Οι συγγραφείς επισημαίνουν ότι η μοντελοποίηση γίνεται σε επίπεδο κλάσεων και όχι επίπεδο αντικειμένων (περιστάσεων). Στη συνέχεια τόσο τα Διαγράμματα Κατάστασης όσο και τα Διαγράμματα Ακολουθίας μετασχηματίζονται σε Επισημασμένα Γενικευμένα Στοχαστικά Δίκτυα Petri (*Labeled Generalized Stochastic Petri Nets - LGSPN*). Τελικά ο συνδυασμός των δικτύων αυτών παρέχει ένα επαρκές μοντέλο συμπεριφοράς για το σύστημα και επιτρέπει την εφαρμογή τεχνικών επαλήθευσης και ανάλυσης επίδοσης. Πρέπει να σημειωθεί ότι στόχος του Bernardi και των συνεργατών του ήταν η στατική ανάλυση, δηλαδή δε λαμβάνονται υπόψη μεταβλητές και οι «*φύλακες*» (*guards*) των Διαγραμμάτων Κατάστασης αντιμετωπίζονται πιθανοκρατικά.

Μια σχετική με την προηγούμενη προσέγγιση έγινε στην εργασία [SS00] από τους Saldhana και Shatz. Στόχος ήταν να καταστεί δυνατή η τυπική επαλήθευση των προδιαγραφών ενός συστήματος που έχουν εκφραστεί σε UML. Στην εργασία περιγράφεται μια μέθοδο για την ανάπτυξη ενός Δικτύου Petri μοντέλου ενός συστήματος, εξάγοντας μια μορφή *Object Petri Net* (βασίζονται στα Έγχρωμα Δίκτυα Petri και περιγράφονται στην [Lak94]) από Διαγράμματα Κατάστασης σε σύνδεση με Συνεργατικά Διαγράμματα (*Collaboration Diagrams*), η ονομασία των οποίων από την έκδοση 2.0 της UML κι έπειτα είναι Διαγράμματα Επικοινωνίας – *Communication Diagrams* και μαζί με τα Διαγράμματα Ακολουθίας αποτελούν τα Διαγράμματα Αλληλεπίδρασης (*Interaction Diagrams*) της UML. Το τελικό μοντέλο που προκύπτει από τη μέθοδο μπορεί να υποβληθεί σε τεχνικές ανάλυσης και προσομοίωσης για τον έλεγχο των προδιαγραφών συμπεριφοράς του συστήματος και τον εντοπισμό επιπλοκών σχετικών με σύνδρομα χαρακτηριστικά -όπως για παράδειγμα αδιέξοδα.

Στο σημείο αυτό μπορεί να σημειωθεί ότι τα Διαγράμματα Ακολουθίας που περιλαμβάνονται στην έκδοση 1.4 της UML δε περιλαμβάνουν *συνδυασμένα τμήματα* (*Combined Fragments*) όπως συμβαίνει στις εκδόσεις από τη 2.0 κι έπειτα. Η προσθήκη των συνδυασμένων τμημάτων ήταν μια κίνηση που εμπλούτισε σημαντικά την εκφραστικότητα των Διαγραμμάτων Ακολουθίας κι έδωσε τη δυνατότητα για τη συμπυκνωμένη αποτύπωση σύνθετων σεναρίων.

Στην [RF06] οι Ribeiro και Fernandes παρουσιάζουν ένα σύνολο κανόνων για τον μετασχηματισμό Διαγραμμάτων Ακολουθίας της έκδοσης 2.0 της UML σε Έγχρωμα Δίκτυα Petri (CPNs). Προσεγγίζουν το πρόβλημα με σκοπό να καταστεί δυνατή η δημιουργία κινούμενων απεικονίσεων (*animations*) βασισμένων σε ένα CPN για την αναπαραγωγή και επίδειξη των μοντελοποιημένων σεναρίων. Στόχος τους είναι να δοθεί η δυνατότητα στους πελάτες/χρήστες ενός συστήματος που δε διαθέτουν τεχνικές γνώσεις να αντιληφθούν και να συζητήσουν τις απαιτήσεις όπως αυτές έχουν αποτυπωθεί. Στην εργασία περιγράφονται

γραφικά κανόνες μετασχηματισμού των χειριστών *seq, strict, par, loop* και *alt* και παρουσιάζεται μια περίπτωση χρήσης. Οι ίδιοι ερευνητές μαζί με τους Tjell και Jorgensen προχωρούν, χρησιμοποιώντας μια μελέτη περίπτωσης των προδιαγραφών ενός ελεγκτή ανελκυστήρα, στην περιγραφή ενός πλήρους μετασχηματισμού Διαγραμμάτων Περιπτώσεων Χρήσης σε συνδυασμό με Διαγράμματα Ακολουθίας σε Έγχρωμα Δίκτυα Petri (CPNs) [FTJ+07]. Η προσέγγιση αναλύεται σε διαφορετικά επίπεδα. Στο πρώτο βήμα καταστρώνεται ένα CPN από τις περιπτώσεις χρήσεις λαμβάνοντας υπόψη μόνο τις πρωτεύουσες περιπτώσεις, δηλαδή όπως είναι συνδεδεμένες με κάποιον *δράστη* (*actor*). Οι «*συνυπολογισμοί*» («*include*») μιας περίπτωσης χρήσης μοντελοποιούνται μέσω του τμήματος αλληλεπίδρασης *ref* (*ref Interaction Fragment*). Το *ref* ορίζει ότι ένα Διάγραμμα Ακολουθίας μπορεί να αναφέρεται και να χρησιμοποιεί ένα άλλο Διάγραμμα Ακολουθίας - χαρακτηριστικό ιδιαίτερα χρήσιμο καθώς προσθέτει δυνατότητες αφαίρεσης και επαναχρησιμοποίησης. Στο CPN που προκύπτει περιλαμβάνεται μια μετάβαση υποκατάστασης για κάθε περίπτωση χρήσης. Στο δεύτερο βήμα καταστρώνονται τα CPNs που αντιστοιχούν στα Διαγράμματα Ακολουθίας των περιπτώσεων χρήσης. Η γραμματική που χρησιμοποιείται για το μετασχηματισμό αυτό περιλαμβάνει συνδυασμένα τμήματα με χειριστές *opt, alt, par, loop* και το πλαίσιο με χειριστή *ref*. Οι κανόνες για τους χειριστές *alt* και *par* είναι επεκτάσεις αυτών στην [RF06]. Ο χειριστής *opt* αντιμετωπίζεται ως ειδική περίπτωση του *alt*. Τέλος ο κανόνας για τον χειριστή *loop* έχει εξελιχθεί ώστε να λαμβάνει υπόψη πέρα από το φύλακα, τη δυνατότητα καθορισμού ελαχίστου και μεγίστου πλήθους επαναλήψεων -όπως ορίζεται στη προδιαγραφή της UML 2.0.

Μια άλλη ενδιαφέρουσα προσέγγιση γίνεται στην [EFM+05] όπου ο Eichner και οι συνεργάτες του εισάγουν μια τυπική σημασιολογία για τις περισσότερες από τις έννοιες των Διαγραμμάτων Ακολουθίας της έκδοσης 2.0 της UML χρησιμοποιώντας Δίκτυα Petri ως ένα τυπικό μοντέλο. Η σημασιολογία παράγεται με μια διαδικασία που στηρίζεται στις βασικές συνθετικές λειτουργίες των Δικτύων Petri. Λόγω των χαρακτηριστικών των Δικτύων Petri είναι δυνατή η έκφραση της μερικώς διατεταγμένης (*partially ordered*) και σύνδρομη (*concurrent*) συμπεριφοράς που περιγράφουν διαγραμμάτων. Το παραγόμενο μοντέλο είναι Έγχρωμα Δίκτυα Petri, τα οποία δίνουν τη δυνατότητα για μια αναλυτική και αποδοτική δομή για τύπους δεδομένων και στοιχεία ελέγχου. Παρουσιάζονται κανόνες μετασχηματισμού σχεδόν για όλα τα συνδυασμένα τμήματα και τους αντίστοιχους χειριστές. Μια θεμελιώδης διαφορά αυτής της εργασίας με τις [RF06] και [FTJ+07] είναι ότι εδώ τα μηνύματα μοντελοποιούνται ως *θέσεις* (*places*) και όχι ως *μεταβάσεις* (*transitions*) -όπως συμβαίνει στις προαναφερθείσες εργασίες.

Ενδιαφέρον παρουσιάζει επίσης, η [JLD05] στην οποία παρουσιάζεται ένας πολυώνυμος αλγόριθμος μέσω του οποίου μπορεί να αποφασιστεί αν ένα σενάριο, προσδιορισμένο ως

Επισημασμένη Μερική Διάταξη (*Labeled Partial Order - LPO*) είναι εκτελέσιμο σε ένα *Place/Transition Net*. Ο αλγόριθμος διατηρεί την συνδρομή (*concurrency*) του σεναρίου χωρίς να προσθέτει αιτιότητα. Όταν το σενάριο μπορεί να εκτελεστεί από το Δίκτυο Petri τότε ο αλγόριθμος επιστρέφει ένα δίκτυο διεργασίας (*process net*) που τηρεί τη συνδρομή που εκφράζεται στο σενάριο. Στην εργασία επίσης παρουσιάζεται κι ένας πολυώνυμος έλεγχος με τον οποίο μπορεί να αποφασιστεί αν μια LPO είναι αυστηρά εκτελέσιμη (δηλαδή εφόσον το δεδομένο ύψος συνδρομής είναι μέγιστο, ή συμπληρωματικά, εφόσον το ύψος της αιτιότητας είναι ελάχιστο). Ο αλγόριθμος περιορίζεται σε απλού τύπου Δίκτυα Petri και δεν μπορεί να εφαρμοστεί σε υψηλού επιπέδου δίκτυα όπως τα CPNs ή τα *Predicate/Transition Nets* (*Pr/T Nets*).

Ορισμένες από τις εργασίες που παρουσιάστηκαν παραπάνω αφορούν παλαιότερες εκδόσεις της γλώσσας UML ([EK98], [BDM02]) με αποτέλεσμα οι κανόνες μετασχηματισμού που προτείνονται να μη καλύπτουν το εύρος των στοιχείων που έχουν προστεθεί στα Διαγράμματα Ακολουθίας από την έκδοση 2.0 και μετά. Πιο κοντά στην προσέγγιση που γίνεται στην παρούσα εργασία βρίσκονται οι [EFM+05], [FTJ+07] και [RF06] που αφορούν Διαγράμματα Ακολουθίας της UML 2.0. Ως προς την [EFM+05] η λογική που ακολουθείται διαφέρει ριζικά καθώς τα γεγονότα αναπαριστώνται από μεταβάσεις και όχι από θέσεις – παρόλα αυτά, από τη γραμματική του μετασχηματισμού εξασφαλίζεται ότι σε κάθε μετάβαση που αντιστοιχεί σε εμφάνιση μηνύματος αντιστοιχεί μια μοναδική θέση εξόδου. Οι κανόνες που παρουσιάζονται στην [FTJ+07] όπως και εκείνοι στην [RF06] εμφανίζουν σε ορισμένους από τους χειριστές αλληλεπίδρασης κοινά σημεία ως προς τη δομή, με τη γραμματική που προτείνεται στην παρούσα εργασία. Παρόλα αυτά υπάρχουν αρκετές διαφοροποιήσεις τόσο δομικά όσο και στη λογική του μετασχηματισμού καθώς, αφενός διαφέρει ο τύπος των Δικτύων Petri που χρησιμοποιείται, αφετέρου οι ιδιαίτερες ανάγκες που σχετίζονται με τη χρήση των προκυπτόντων δικτύων για το ταίριασμα σεναρίων έναντι καταγραφών απαιτούν οι μεταβάσεις να φέρουν επιπλέον πληροφορίες που εξάγονται από το αρχικό μοντέλο. Στο 4^{ο} κεφάλαιο της εργασίας γίνεται διεξοδική περιγραφή του μετασχηματισμού.

2.3 Έλεγχος μοντέλου

Ο έλεγχος μοντέλου (*model checking*) είναι μια αυτόματη τεχνική για την επαλήθευση ιδιοτήτων ορθότητας σε κρίσιμα ως προς την ασφάλεια αντιδραστικά συστήματα (*reactive systems*). Αντιδραστικά αποκαλούνται τα συστήματα αποτελούμενα από δομοστοιχεία τα οποία αλληλεπιδρούν μεταξύ τους και με το περιβάλλον του συστήματος [Cla00]. Η

επιστημονική αυτή περιοχή που αναπτύχθηκε και εξελίχθηκε ιδιαιτέρως τις τελευταίες τρεις δεκαετίες έχει εφαρμοσθεί με επιτυχία για τον εντοπισμό δυσδιάκριτων λαθών σε σύνθετα βιομηχανικά συστήματα (όπως ακολουθιακά κυκλώματα, πρωτόκολλα επικοινωνίας, ψηφιακούς ελεγκτές και αλλού). Στους σημαντικότερους εκπροσώπους της έρευνας πάνω στο model checking συγκαταλέγονται οι Clarke, Emerson και Σηφάκης -στους οποίους μάλιστα απονεμήθηκε το Turing Award το 2007 για την προσφορά τους στην επιστημονική αυτή περιοχή.

Πρέπει να σημειωθεί ότι στο πλαίσιο που προτείνεται με την παρούσα εργασία, η προσέγγισή και η χρήση του έλεγχο μοντέλου εμφανίζει κοινά χαρακτηριστικά σε θεωρητικό επίπεδο, με την προσέγγιση που γίνεται στην [Rob02] (βλ. 2.1). Πιο συγκεκριμένα, ως μοντέλο θεωρείται το προκύπτων (μετά δηλαδή από το μετασχηματισμό) Δίκτυο Petri σε συνδυασμό με την ακολουθία γεγονότων, είτε σε μορφή αρχείου καταγραφών, είτε στην ειδική δομή που χρησιμοποιείται κατά την επιγραμμική ανάλυση. Υπό αυτή την οπτική, το μοντέλο ελέγχεται ως προς τη δυνατότητα μετάβασης από μια δεδομένη αρχική κατάσταση, σε μια τελική κατάσταση -αν αυτό συμβεί τότε έχουμε επαλήθευση του σεναρίου.

Μέσω του ελέγχου μοντέλου, επιχειρείται τυπική επαλήθευση σύνδρομων συστημάτων πεπερασμένων καταστάσεων. Οι προδιαγραφές του συστήματος εκφράζονται ως εξισώσεις χρονικής (*temporal*) λογικής και στη συνέχεια επιστρατεύονται αλγόριθμοι (συμβολικοί ή άλλοι) οι οποίοι διατρέχουν το μοντέλο που ορίζεται από το σύστημα και ελέγχουν αν οι προδιαγραφές ισχύουν ή όχι. Η βασική πρόκληση στον έλεγχο μοντέλου είναι ο χειρισμός του προβλήματος της «έκρηξης» του χώρου καταστάσεων. Δηλαδή, σε συστήματα με πολλά δομοστοιχεία που αλληλεπιδρούν μεταξύ τους, ή σε συστήματα με δομές δεδομένων για τις οποίες μπορούν να υποθέσεις πολλών διαφορετικών τιμών ο αριθμός των καταστάσεων για το σύστημα μπορεί να είναι τεράστιος. Παρόλα αυτά, η πρόοδος που έχει γίνει στην περιοχή επιτρέπει τον έλεγχο αρκετά μεγάλων χώρων καταστάσεων σε λίγο χρόνο.

Αν και η μέθοδος του ελέγχου μοντέλων βρίσκει εφαρμογή συνήθως σε συστήματα υλικού η χρήση του για το λογισμικό είναι αυξανόμενη. Δυστυχώς όμως, η πολυπλοκότητα των γλωσσών προγραμματισμού (σε σχέση με τις γλώσσες περιγραφής υλικού) και άρα η δυσκολία μοντελοποίησης των προγραμμάτων όπως και οι δυσκολίες στο να καθορισθούν ιδιότητες που έχουν σημασία για το σύστημα χρησιμοποιώντας τους συνηθισμένους φορμαλισμούς χρονικής λογικής του model checking, καθιστούν τη διαδικασία αυτή ιδιαίτερα σύνθετη. Επιπλέον λαμβάνοντας υπόψη το πάγιο πρόβλημα της έκρηξης του χώρου καταστάσεων, η πολυπλοκότητα επαλήθευσης μιας υλοποίησης έναντι μιας προδιαγραφής γίνεται συχνά απαγορευτική [CCO+04].

Στην [BH04] οι Bell και Haverkort παρουσιάζουν ακολουθιακούς αλλά και κατανεμημένους αλγόριθμους για τον έλεγχο μοντέλου υπολογιστικής λογικής δέντρου (*Computational tree logic – CTL*) έναντι συστημάτων πεπερασμένων καταστάσεων που περιγράφονται από Δίκτυα Petri. Ένα σημαντικό στοιχείο των αλγορίθμων που παρουσιάζουν είναι ότι αν και βασίζονται στην διεξοδική αναπαράσταση του χώρου των καταστάσεων του συστήματος, η σχέση μετάβασης (*transition relation*) υπολογίζεται εκ νέου, κάθε φορά που αυτό είναι απαραίτητο. Αυτό επιτρέπει τον έλεγχο μοντέλων πολύ μεγάλων συστημάτων με εκατοντάδες εκατομμύρια καταστάσεων, με ένα γρήγορο και αποδοτικό τρόπο. Τα αποτελέσματα των πειραμάτων που διεξήγαγαν δείχνουν ότι οι κατανεμημένοι αλγόριθμοι έχουν καλή κλιμάκωση (η αποδοτικότητά τους κινήθηκε στη περιοχή του 60% με 95% ανάλογα με την περίπτωση).

Τέλος, μια πρόσφατη εργασία σχετικά με την πολυπλοκότητα της τεχνικής ελέγχου μοντέλου βασισμένης σε ίχνη μερικής διάταξης είναι η [MMV08]. Συγκεκριμένα μελετάται ο έλεγχος μοντέλου για CTL, CTL* και LTL (*linear temporal logic*) σύμφωνα με ακολουθίες μερικώς διατεταγμένων γεγονότων και αποδεικνύεται ότι οι δύο πρώτες περιπτώσεις είναι *PSPACE-complete* ενώ στην τρίτη είναι *coNP-complete*.

Κεφάλαιο 3: Βασικές έννοιες, πρότυπα και τεχνολογίες

Για την καλύτερη κατανόηση του πλαισίου που προτείνεται στην παρούσα εργασία παρουσιάζονται στο κεφάλαιο αυτό βασικές έννοιες, πρότυπα και τεχνολογίες που χρησιμοποιούνται ή που σχετίζονται άμεσα με την προσέγγιση του γίνεται. Αρχικά παρουσιάζονται τεχνικές για τον ορισμό, τη συλλογή και την καταγραφή γεγονότων κατά το χρόνο εκτέλεσης του λογισμικού. Ακολουθεί μια σύντομη περιγραφή των συνηθέστερων μορφότυπων των λημμάτων των καταγραφών και παρουσιάζονται συνοπτικά ορισμένες τεχνολογίες που προσφέρουν τη δυνατότητα προσαρμογής και μετασχηματισμών μεταξύ μορφότυπων. Κατόπιν, συζητούνται ορισμένες αρχιτεκτονικές και συστήματα που χρησιμοποιούνται στη βιομηχανία για την παρακολούθηση συστημάτων λογισμικού.

Στη συνέχεια του κεφαλαίου, περιγράφονται πρότυπα και τεχνολογίες που εντάσσονται στην περιοχή της ανάπτυξης οδηγούμενης από μοντέλα (*Model Driven Development*) και αντίστοιχα εργαλεία και υλοποιήσεις του περιβάλλοντος ανάπτυξης που χρησιμοποιήθηκε (*Eclipse*) που εξυπηρετούν τον ορισμό, την έκφραση, την αποθήκευση, τη δυνατότητα μεταφοράς και διαλειτουργικότητας και τον μετασχηματισμό των μοντέλων που χειρίζεται η προτεινόμενη μεθοδολογία. Επίσης, γίνεται μια σύντομη αναφορά στη γλώσσα μετασχηματισμού μοντέλων *Atlas Transformation Language (ATL)* η οποία χρησιμοποιήθηκε για την υλοποίηση του μετασχηματισμού.

3.1 Παρακολούθηση και καταγραφή

Η τήρηση καταγραφών (*logging*) σε επίπεδο εφαρμογών, λειτουργικού συστήματος και στοιχείων δικτύου είναι η διαδικασία κατά την οποία συλλέγονται και καταγράφονται γεγονότα που συμβαίνουν κατά τη λειτουργία και την αλληλεπίδραση των στοιχείων και των χρηστών ενός συστήματος λογισμικού. Το περιεχόμενο, το επίπεδο αφαίρεσης και ο

μορφότυπος (*format*) των καταγραφών όπως και οι τεχνικές συλλογής και διαχείρισης προκύπτουν βάσει πολιτικών παρακολούθησης και καταγραφής.

Ένα σύστημα καταγραφής αποτελείται από τρία βασικά στοιχεία [RK08]. Το πρώτο αφορά σε ένα σχήμα που δηλώνει το είδος, το επίπεδο αφαίρεσης και τα χαρακτηριστικά των ατομικών γεγονότων που καταγράφονται, στα οποία συνήθως περιλαμβάνεται ο χρόνος και η προέλευση. Το δεύτερο σχετίζεται με την ενορχήστρωση *probe* τα οποία γεννούν και εξάγουν τα γεγονότα προς καταγραφή. Οι τεχνικές *probing* έχουν να κάνουν με το είδος των γεγονότων που θέλουμε να εξάγουμε και από τη διαθεσιμότητα ή όχι του πηγαίου κώδικα του συστήματος. Συγκεκριμένα, υπάρχουν probe που αποτελούνται από κώδικα ο οποίος εισάγεται στο λογισμικό και άρα η ενορχήστρωση προϋποθέτει πρόσβαση και δυνατότητα επέμβασης στον πηγαίο κώδικα (*intrusive*), probe που ενορχηστρώνονται σε ενδιάμεσο επίπεδο όπως για παράδειγμα σε εφαρμογές υλοποιημένες σε Java στο *bytecode* (*bytecode instrumentation - BCI*) αλλά και probe που λειτουργούν ως ανεξάρτητες διεργασίες και μεσολαβούν μεταξύ των εφαρμογών και του περιβάλλοντος εκτέλεσής τους και συλλέγουν δεδομένα αλληλεπίδρασης και επικοινωνίας των διαφόρων στοιχείων λογισμικού σε διαφορετικά επίπεδα (*non-intrusive*). Επίσης, όπως αναφέρθηκε και σε προηγούμενη ενότητα, στη βιβλιογραφία έχουν προταθεί υβριδικές τεχνικές συλλογής δεδομένων με τη χρήση τόσο λογισμικού όσο και υλικού. Το τρίτο βασικό στοιχείο ενός συστήματος καταγραφής αφορά στον τρόπο οργάνωσης και διατήρησης των γεγονότων που καταγράφονται, δηλαδή αφορά σε μια βιβλιοθήκη αποθήκευσης (*repository*) που περιλαμβάνει τις καταγραφές και που κατά κανόνα, πρακτικά υλοποιείται είτε ως συλλογές αρχείων καταγραφής (*log files*), είτε από εξειδικευμένες βάσεις δεδομένων.

Ένα σημαντικό πρόβλημα που οι διαχειριστές των συστημάτων και οι μηχανικοί λογισμικού αντιμετωπίζουν ως προς την καταγραφή είναι οι ετερογενείς μορφότυποι των λημμάτων που περιγράφουν τα ατομικά γεγονότα. Πολλές φορές ακόμα και η μορφή των «*χρονοσφραγίδων*» (*timestamps*) διαφέρει και μάλιστα ενίοτε όχι μόνο μεταξύ των εφαρμογών αλλά και μεταξύ διαφορετικών εκδόσεων των ίδιων εφαρμογών.

Το γεγονός ότι οι μορφότυποι των λημμάτων πολύ συχνά ακολουθούν ιδιωτικά πρότυπα σχετίζεται με το ότι οι μορφότυποι αυτοί συνήθως έχουν να κάνουν με εξειδικευμένες ανάγκες αποτύπωσης ανά εφαρμογή κι εξαρτώνται από τις εκτιμήσεις των προγραμματιστών για τη χρήση των καταγραφών. Για να καταστεί δυνατή η χρήση εργαλείων και η εφαρμογή τεχνικών ανάλυσης σε μια κοινή βάση έχουν προταθεί και χρησιμοποιούνται γενικοί - συνήθως σε επίπεδο τύπου εφαρμογών- και συχνά προτυποποιημένοι μορφότυποι λημμάτων εκ των οποίων οι σημαντικότεροι παρουσιάζονται στη συνέχεια. Βέβαια συνήθως η επικράτηση κάποιου μορφότυπου δεν έχει να κάνει τόσο με διαδικασίες προτυποποίησης όσο

με τη χρησιμοποίησή του σε ευρέως διαδεδομένες εφαρμογές (για παράδειγμα ο εξυπηρετητής ιστού *Apache HTTP Server*). Επιπλέον, και πάλι για λόγους διαλειτουργικότητας, έχουν αναπτυχθεί εργαλεία μετασχηματισμού μορφότυπων δίνοντας τη δυνατότητα αναδιαμόρφωσης των καταγραφών που ακολουθούν κάποιο μορφότυπου σε ένα δεύτερο, κατάλληλο για χρήση σε εξειδικευμένα, μη ευέλικτα, εργαλεία ανάλυσης. Στο τελευταίο μέρος της ενότητας περιλαμβάνεται μια σύντομη περιγραφή τεχνολογιών που διευκολύνουν το μετασχηματισμό μορφότυπων λημμάτων.

Το μοντέλο *Common Base Event (CBE)* [CBE] περιγράφει ένα μορφότυπο λημμάτων ιδιαιτέρως κατάλληλο για γεγονότα που συμβαίνουν μεταξύ των διαφορετικών τύπων κατανεμημένων εφαρμογών. Στόχοι του είναι η διασφάλιση της πληρότητας των καταγραφών και της ακρίβειας των δεδομένων όπως και η βελτίωση του επιπέδου λεπτομέρειας κάθε εγγραφής. Η βάση στην οποία τα σύγχρονα κατανεμημένα συστήματα επικοινωνούν θεωρείται το γεγονός (*event*) το οποίο ενθυλακώνει δεδομένα μηνύματος. Τα δεδομένα μηνύματος στέλνονται ως αποτέλεσμα μιας εξέλιξης ή κάποιας κατάστασης. Οι πληροφορίες που καταγράφονται σε κάθε εγγραφή μπορούν να θεωρηθούν ως μια «*τριπλέτα*» (*3-tuple*). Τα στοιχεία της τριπλέτας περιλαμβάνουν το αναγνωριστικό του δομοστοιχείου που αναφέρει την κατάσταση, το αναγνωριστικό του δομοστοιχείου που επηρεάζεται από την κατάσταση (ενδεχομένως ίδιο με εκείνο που αναφέρει την κατάσταση) και την κατάσταση καθαυτή. Το μοντέλο CBE είναι μια συνεισφορά της εταιρίας IBM και βρίσκει εφαρμογή στα πλαίσια του *Autonomic Computing Toolkit* [ACT] (της ίδιας εταιρίας) το οποίο είναι ένα σύνολο εργαλείων για την καινοτόμα ιδέα της αυτοδιαχείρισης εφαρμογών, συστημάτων ή και ολόκληρων δικτύων. Ως αυτοδιαχείριση η IBM ορίζει τέσσερις λειτουργίες: αυτορύθμιση (self-configure), αυτοΐαση (self-heal), αυτο-βελτιστοποίηση (self-optimize) και αυτοπροστασία (self-protect).

Αρκετά διαδεδομένοι είναι οι μορφότυποι που χρησιμοποιούνται από την εφαρμογή *Apache HTTP Server* και καταγράφουν τα στοιχεία πρόσβασης για αιτήσεις που δέχεται ο εξυπηρετητής: *Common Log Format* και *Combined Log Format* [ALF]. Βέβαια πρακτικά ο μορφότυπος των λημμάτων μπορεί να τροποποιηθεί ανάλογα με τις ανάγκες και τις επιλογές των διαχειριστών. Τυπικά όμως στα περιεχόμενα του Common Log Format περιλαμβάνονται η IP διεύθυνση, το στοιχείο «*identd*» του αιτούντα όταν είναι διαθέσιμο, το όνομα του χρήστη όταν υπάρχει, η χρονική στιγμή κατά την οποία παρελήφθη η αίτηση σε συγκεκριμένη μορφή, το περιεχόμενο της αίτησης (η μέθοδος, ο συγκεκριμένος πόρος και το πρωτόκολλο), ο κωδικός χειρισμού της αίτησης από τη πλευρά του εξυπηρετητή και το μέγεθος του αντικειμένου που επιστρέφει στον αιτούντα. Στο Combined Log Format πέρα

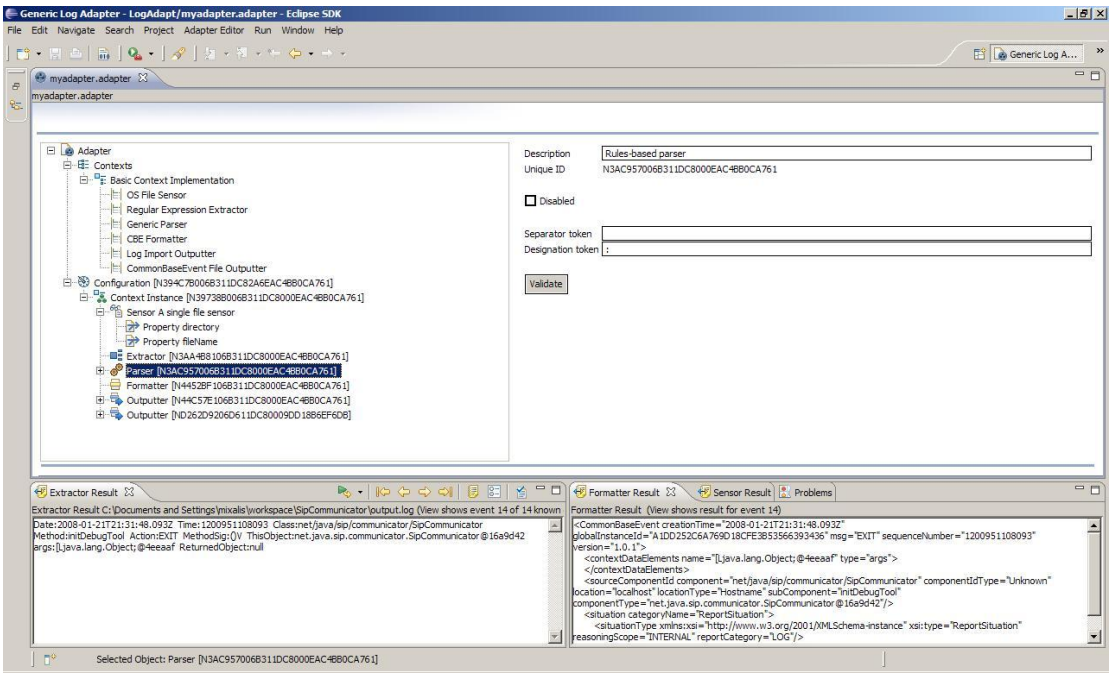
από τα παραπάνω στοιχεία περιλαμβάνονται οι τιμές των ετικετών κεφαλής της HTTP αίτησης «*Referer*» και «*User-Agent*».

Τέλος, δύο επίσης σημαντικοί και ευρέως χρησιμοποιούμενοι μορφότυποι είναι εκείνοι που ορίζονται στη βιβλιοθήκη *java.util.logging* της Java [JLOG]: ο απλός (μέσω του *SimpleFormatter*) και εκφρασμένος σε XML μορφότυπος λημμάτων (μέσω του *XMLFormatter*). Η παραπάνω βιβλιοθήκη της Java προσφέρει τη δυνατότητα να γίνονται κλήσεις καταγραφής από τις εφαρμογές σε αντικείμενα «*καταγραφείς*» *Logger*. Ένα αντικείμενο Logger για κάθε κλήση που δέχεται δημιουργεί ένα αντικείμενο εγγραφής, *LogRecord*, το οποίο τελικά διαβιβάζεται σε ένα αντικείμενο χειριστή, *Handler*. Το αντικείμενο Handler αναλαμβάνει τη δημοσιοποίηση των εγγραφών σε κάποιο μέσο εξόδου (*file*/*memory*/*socket*/*stream*) κι έχει τη δυνατότητα να χρησιμοποιήσει ένα αντικείμενο διαμορφωτή *Formatter* για να αποδώσει κάποια συγκεκριμένη μορφή στο μήνυμα που θα δημοσιοποιηθεί. Παράλληλα υπάρχει δυνατότητα να χρησιμοποιηθούν φίλτρα (*Filter*) και επίπεδα (*Level*) τόσο από τα αντικείμενα Logger, όσο και από τα Handler. Στον XML μορφότυπο κάθε εγγραφή περιλαμβάνει τη χρονική στιγμή, την έκφραση της χρονικής στιγμής σε χιλιοστά του δευτερολέπτου, έναν αριθμό ακολουθίας, ένα αναγνωριστικό του ποιος κάνει την καταγραφή (ποιο αντικείμενο Logger), το επίπεδο από το οποίο χαρακτηρίζεται η κρισιμότητας της εγγραφής, η κλάση από την οποία προήλθε, το όνομα της μεθόδου, ο αριθμός του νήματος και το μήνυμα. Στον απλό μορφότυπο κάθε εγγραφή καταλαμβάνει δύο σειρές και περιλαμβάνει τη χρονική στιγμή, την κλάση, τη μέθοδο στη πρώτη γραμμή και το επίπεδο και το μήνυμα στη δεύτερη.

Όπως αναφέρθηκε παραπάνω η δυνατότητα μετασχηματισμού και εναλλακτικής χρήσης μορφότυπων παίζει σημαντικό ρόλο στην αντιμετώπιση της ανομοιογένειας των καταγραφών που συλλέγονται από διάφορες εφαρμογές, συστήματα ή στοιχεία δικτύου. Πολύ συχνά, η ανομοιογένεια αυτή σε συνδυασμό με την έλλειψη κατάλληλων εργαλείων για τη διαχείριση των βιβλιοθηκών αποθήκευσης των καταγραφών, είναι τα μεγαλύτερα εμπόδια για τον ορισμό κατάλληλων και αποτελεσματικών «*μηχανισμών συγχώνευσης*» και δυσχεραίνοντας τελικά την αποδοτική ανάλυση των καταγραφών.

Οι προσεγγίσεις στο παραπάνω πρόβλημα είναι συνήθως δύο: είτε γράφεται κάποιο εξειδικευμένο πρόγραμμα ή τμήματα κώδικα που αναλαμβάνουν το μετασχηματισμό από κάποιο μορφότυπο σε κάποιον άλλο, είτε γράφονται κάποια σενάρια (*scripts*) που μπορούν να χρησιμοποιηθούν στα πλαίσια διαδεδομένων εργαλείων σεναρίων (π.χ. *Perl*). Αφού οριστούν οι μετασχηματισμοί, τα εργαλεία ή τα τμήματα κώδικα που δημιουργούνται ενσωματώνονται συνήθως στις διαδικασίες ανάλυσης.

Όταν ο ζητούμενος μορφότυπος είναι εκείνος που ορίζεται από το *Common Base Event* μοντέλο, τότε μια εναλλακτική λύση που επιταχύνει τη διαδικασία παρέχεται από το *Generic Log Adaptor (GLA)* [GLA]. Το GLA είναι ένα πλαίσιο που δίνει τη δυνατότητα επεξεργασίας και μετασχηματισμού αρχείων καταγραφής διαφόρων μορφότυπων στο μορφότυπο CBE, μέσω κανόνων που διατυπώνονται σε Java ή σεναρίων που περιλαμβάνουν τυπικές εκφράσεις για την περιγραφή της αντιστοίχισης του περιεχομένου του αρχείου καταγραφής στα στοιχεία του CBE. Στο πλαίσιο GLA περιλαμβάνεται το εργαλείο επεξεργασίας *Adaptor Configuration Editor (ACE)* μέσω του οποίου είναι δυνατή η διατύπωση κανόνων σε φιλικό προς τον χρήστη περιβάλλον. Συγκεκριμένα, η διαδικασία μετασχηματισμού μέσω του ACE είναι η εξής: προσδιορίζονται ο/οι *Sensors* των δεδομένων εισόδου (όπως για παράδειγμα το αρχείο με τις πρωτογενείς καταγραφές), ο *Extractor* μέσω του οποίου δηλώνονται τα όρια των εγγραφών με χρήση τυπικών εκφράσεων, ο *Parser* όπου γίνεται η γραμματική ανάλυση (*parsing*) κάθε εγγραφής και η αντιστοίχηση των δεδομένων με στοιχεία του CBE, ο/οι *Formatters* (στο *GLA* περιλαμβάνονται οι *CBE* και *Serialized CBE Formatters*) και τέλος ο/οι *Outputters* μέσω των οποίων καθορίζεται το μέσο εξόδου των μετασχηματισμένων εγγραφών -όπως για παράδειγμα κάποιο αρχείο εξόδου ή κάποιος πράκτορας για την απευθείας ανάλυση στα πλαίσια του *Test & Performance Tool Platform (TPTP)* [TPTP], κ.ά.). Μια λήψη οθόνης του Adaptor Configuration Editor δίνεται στο Σχήμα 3.1.



Σχήμα 3.1: Adaptor Configuration Editor, Generic Log Adaptor, Eclipse

Μια πιο ειδική αλλά και απλούστερη λύση μπορεί να εφαρμοσθεί στις περιπτώσεις που ο αρχικός μορφότυπος είναι κάποιος από τους: *Apache HTTP Server (CLF), Apache HTTP Server error, Java Logging XML* και *Microsoft Windows Application/Security/System log formats*. Τότε, μπορεί να χρησιμοποιηθεί απευθείας το πλαίσιο ανάλυσης *Log and Trace Analyzer* (του έργου TPTP), το οποίο ως παράπλευρη δυνατότητα, επιτρέπει την εξαγωγή σε μορφή CBE οποιουδήποτε αρχείου εισάγεται εφόσον ο μορφότυπος του είναι κάποιος από τους παραπάνω. Επίσης, το *Log and Trace Analyzer* δίνει τη δυνατότητα για την εφαρμογή ορισμένων σημαντικών τεχνικών ανάλυσης σε αρχεία καταγραφής, όπως τη δημιουργία και εφαρμογή συσχετίσεων (*correlations*) μεταξύ αυτών και τον εντοπισμό συμπτωμάτων μέσω του ορισμού και της εφαρμογής καταλόγων συμπτωμάτων (*symptom catalogs*).

Οι εταιρίες λογισμικού αναγνωρίζοντας τις αυξημένες και συχνά εξεζητημένες ανάγκες παρακολούθησης και καταγραφής των σύγχρονων βιομηχανικών συστημάτων έχουν προτείνει και προωθούν διάφορα εμπορικά πακέτα.

Η Sun Microsystems παρέχει στα πλαίσια του *Sun Java™ Enterprise System 5 (Java ES)* δυνατότητες παρακολούθησης μέσω του *Sun Java System Monitoring Framework 2.0* και του *Sun Java System Monitoring Console 1.0* [SUNMON]. Το Monitoring Framework παρέχει την υποδομή ώστε τα δομοστοιχεία να ενορχηστρώνονται και να εκθέτουν τα προς παρακολούθηση χαρακτηριστικά τους. Στη διαδικασία χρησιμοποιείται το μοντέλο *Common Monitoring Model* το οποίο διασφαλίζει ότι όλα τα Java ES δομοστοιχεία εκθέτουν ομοιόμορφα αντικείμενα και τιμές για συγκρίσιμα χαρακτηριστικά. Ανά κόμβο ορίζεται ένας πράκτορας («*node agent*») ο οποίος συνθέτει μια εικόνα όλων των παρακολουθούμενων δομοστοιχείων του κόμβου, συγκεντρώνει στατιστικά λειτουργίας και αποστέλλει ειδοποιήσεις στις περιπτώσεις υπέρβασης των κατωφλιών που έχουν τεθεί από τους διαχειριστές. Το Monitoring Console αποτελεί τη γραφική διεπαφή μέσω της οποίας παρακολουθείται η συμπεριφορά των δομοστοιχείων. Περιλαμβάνει ένα κεντρικό πράκτορα («*master agent*») ο οποίος συνδέεται με όλους του πράκτορες κόμβου και υλοποιείται ως εφαρμογή Ιστού ώστε να είναι προσβάσιμη από παντού μέσω του HTTP πρωτοκόλλου. Τέλος, πέρα από την ενημέρωση για τις ειδοποιήσεις και την κατάσταση των δομοστοιχείων, το Monitor Console προσφέρει τη δυνατότητα δημιουργίας νέων κανόνων ειδοποιήσεων. Η Sun Microsystems προσφέρει επίσης ένα πλαίσιο καταγραφής ιχνών για το *Solaris Operating Environment* που το ονομάζει *DTrace*. Μέσω του DTrace συλλέγονται καταγράφονται αναλυτικά στοιχεία (ίχνη) από τη λειτουργία του λειτουργικού συστήματος και των προγραμμάτων τα οποία προσφέρονται για ανάλυση.

Η IBM προσφέρει το προϊόν *Tivoli Monitoring* για την παρακολούθηση της πληροφοριακής υποδομής μιας επιχείρησης ή ενός οργανισμού συμπεριλαμβανομένων λειτουργικών

συστημάτων, βάσεων δεδομένων και εξυπηρετητών σε κατανεμημένα και μη περιβάλλοντα, με στόχο την βελτιστοποίηση της επίδοσης και της διαθεσιμότητας. Μέσω του Tivoli Monitoring μπορούν να εντοπιστούν στενώσεις και πιθανά προβλήματα σε βασικούς πόρους του συστήματος, ενώ παρέχεται η δυνατότητα αυτόματης επαναφοράς από κρίσιμες καταστάσεις ώστε να διασφαλίζεται η απρόσκοπτη λειτουργία των κρίσιμων εφαρμογών για μια επιχείρηση ή οργανισμό.

Το προϊόν της Microsoft που επιχειρεί να καλύψει την ανάγκη για υπηρεσίες απ' άκρου εις άκρου παρακολούθησης σε ένα βιομηχανικό περιβάλλον λογισμικού είναι το *System Center Operations Manager 2007*. Στις δυνατότητες του Operations Manager περιλαμβάνονται η παρακολούθησης κατανεμημένων εφαρμογών, η συλλογή καταγραφών ελέγχου, η δήλωση κανόνων, η παροχή ειδοποιήσεων κ.ά. Ένα χαρακτηριστικό του πακέτου αυτού είναι η δυνατότητα επέκταση των λειτουργιών του Operations Manager μέσω εξειδικευμένων πακέτων διαχείρισης ανά λειτουργικό σύστημα, εφαρμογή ή άλλο στοιχείο (*Management Packs*) τα οποία χρησιμοποιώντας τα ιδιαίτερα χαρακτηριστικά κάθε στοιχείου, όπως και μοντέλα υγιούς λειτουργίας, αυξάνουν τις δυνατότητες διαχείρισης, παρακολούθησης, εντοπισμού προβλημάτων και ανάλυσης της επίδοσης, της διαθεσιμότητας και της ασφάλειας.

3.2 Μετασχηματισμοί μοντέλων

Στη μηχανική λογισμικού τα μοντέλα χρησιμοποιούνται συνήθως ως αφαιρέσεις των συστημάτων ή των περιβαλλόντων στα οποία τα συστήματα λειτουργούν, ή και των δύο, οι οποίες επιτρέπουν στους σχεδιαστές, στους προγραμματιστές, στους διαχειριστές και στους χρήστες να διατυπώνουν απόψεις και ερωτήματα ως προς χαρακτηριστικά του συστήματος ή ως προς την εφαρμογή κάποιας αλλαγής σε αυτό, και να εξάγουν συμπεράσματα. Στο χώρο της ανάπτυξης οδηγούμενης από μοντέλα (*Model-Driven Development - MDD*) σπουδαίο ρόλο παίζουν οι μετασχηματισμοί μεταξύ μοντέλων καθώς εφαρμόζονται σε ένα πλήθος περιπτώσεων. Συγκεκριμένα, τέτοιοι μετασχηματισμοί χρησιμοποιούνται στην παραγωγή χαμηλού επιπέδου μοντέλων και τελικά κώδικα από μοντέλα υψηλού επιπέδου, στην αντιστοίχιση και στο συγχρονισμό μεταξύ μοντέλων του ίδιου ή διαφορετικών επιπέδων αφαίρεσης, στη δημιουργία όψεων για το σύστημα βασισμένων σε ερωτήματα, όπως και σε εργασίες σχετικές με την εξέλιξη των μοντέλων ή την εφαρμογή αντίστροφης μηχανικής (*reverse engineering*) για την εξαγωγή μοντέλων υψηλότερου επιπέδου από μοντέλα χαμηλού επιπέδου ή κώδικα [CH06].

Η μέθοδος που προτείνεται στην εργασία αυτή περιλαμβάνει το μετασχηματισμό Διαγραμμάτων Ακολουθίας της UML σε Δίκτυα Petri. Για το μετασχηματισμό χρησιμοποιούνται πρότυπα και τεχνολογίες της μηχανικής οδηγούμενης από μοντέλα και για το λόγο αυτό ακολουθεί μια σύντομη περιγραφή ορισμένων βασικών εννοιών και εργαλείων της περιοχής.

Σπουδαίο ρόλο στο χώρο της μοντελοποίησης και των μετασχηματισμών μοντέλων παίζει το *Object Management Group, Inc (OMG)* [OMG] το οποίο είναι μια διεθνής, ανοιχτής συμμετοχής, μη κερδοσκοπική κοινοπραξία (*consortium*) της βιομηχανίας υπολογιστών. Οι ομάδες εργασίας του OMG αναπτύσσουν πρότυπα (*standards*) για μιας ευρεία περιοχή τεχνολογιών και για μια ακόμα πιο ευρεία περιοχή βιομηχανιών. Μια από τις πιο γνωστές και σημαντικές συνεισφορές του OMG είναι η προδιαγραφή της UML η χρήση της οποίας κυριαρχεί πλέον στη βιομηχανία λογισμικού για την προδιαγραφή και τον σχεδιασμό συστημάτων. Τα μοντέλα που ορίζονται μέσω των διαγραμμάτων της UML μπορούν να περιγράψουν τη δομή και τη συμπεριφορά των συστημάτων. Σύντομα συνειδητοποιήθηκε από τους μετέχοντες στο OMG, η ανάγκη για μια αρχιτεκτονική «*μεταμοντελοποίησης*» (*metamodeling*) που θα καθιστούσε δυνατό τον καθορισμό της UML, δηλαδή για την εισαγωγή ενός μεταμοντέλου στο οποίο η UML έπρεπε να συμμορφώνεται. Έτσι δημιουργήθηκε το *Meta-Object Facility (MOF)* που είναι ένα πρότυπο στην περιοχή της μηχανικής οδηγούμενης από μοντέλα και σύμφωνα με το OMG «*είναι η βάση του πρωτυποποιημένου περιβάλλοντος του OMG στο οποίο μοντέλα μπορούν να εξαχθούν από μια εφαρμογή, να εισαχθούν σε μια άλλη, να μεταφερθούν μέσω ενός δικτύου, να αποθηκευτούν σε μια βιβλιοθήκη αποθήκευσης (repository) κι έπειτα να ανακτηθούν, να αποδοθούν σε διαφορετικές μορφές, να μετασχηματισθούν και να χρησιμοποιηθούν για να παραχθεί κώδικα εφαρμογής.*» [MOF].

Το MOF σχεδιάστηκε ως μια αρχιτεκτονική τεσσάρων επιπέδων. Στο κορυφαίο επίπεδο, που ονομάζεται *M3*, υπάρχει ένα μεταμεταμοντέλο (*metametamodel - MMM*). Το MMM αυτό αποτελεί τη γλώσσα που χρησιμοποιείται από το MOF για τη δημιουργία μεταμοντέλων. Τα μεταμοντέλα αυτά ονομάζονται *M2-μοντέλα* και ένα τέτοιο μεταμοντέλο είναι αυτό της UML. Τα *M2-μοντέλα* περιγράφουν στοιχεία του *M1-επιπέδου* κι αυτά με τη σειρά τους χρησιμοποιούνται στον καθορισμό *M1-μοντέλων*. Χαρακτηριστικό παράδειγμα M1-μοντέλων είναι τα μοντέλα που δημιουργούνται μέσω της UML. Τέλος, ορίζεται το επίπεδο M0 ή αλλιώς επίπεδο δεδομένων, το οποίο χρησιμοποιείται για να περιγράψει αντικείμενα του πραγματικού κόσμου.

Το MOF είναι μια «κλειστή» αρχιτεκτονική μεταμοντελοποίησης, δηλαδή καθορίζει ένα M3-μοντέλο το οποίο συμμορφώνεται στον εαυτό του. Για την καλύτερη κατανόηση του MOF,

κάποιος θα μπορούσε να πει ότι ως προς τον καθορισμό μεταμοντέλων το MOF παίζει το ρόλο που παίζει το EBNF για τον καθορισμό των γραμματικών των γλωσσών προγραμματισμού. Επιπλέον το MOF μπορεί να χρησιμοποιηθεί για να ορίσει αντικειμενοστρεφή μεταμοντέλα (όπως της UML) αλλά και για μη αντικειμενοστραφή μεταμοντέλα (όπως ενός Δικτύου Petri). Αν και επισήμως η τρέχουσα έκδοση (Ιούνιος 2008) του MOF παραμένει η 1.4, μια νέα έκδοση που ανταποκρίνεται στη UML 2.0 υπολογίζεται ότι θα υιοθετηθεί σε μικρό χρονικό διάστημα (8-9/08) [MOF].

Μια στενά συνδεδεμένη τεχνολογία με το MOF είναι ο μορφότυπος *XML Metadata Interexchange (XMI)*. Συγκεκριμένα το XMI είναι ένα πρότυπο του OMG για την ανταλλαγή μεταδεδομένων μέσω της *eXtensible Markup Language (XML)*. Αν τα μεταδεδομένα μπορούν να εκφραστούν ως μοντέλα στο MOF τότε μπορούν να αποδοθούν σε XMI μορφή. Η προδιαγραφή της MOF/XMI αντιστοίχισης ορίζεται στο [XMI]. Η έκφραση των μοντέλων σε XMI επιτρέπει την ανταλλαγή μεταδεδομένων μεταξύ εργαλείων μοντελοποίησης και μετασχηματισμού μοντέλων. Η ποιο ευρεία εφαρμογή που βρίσκει το XMI είναι η αποτύπωση UML μοντέλων καθώς η υιοθέτηση και υποστήριξή του από τα πολλά εργαλεία μοντελοποίησης λογισμικού, διευκολύνει τη μεταφορά των μοντέλων. Επίσης, συνήθως μοντέλα αποτυπωμένα σε XMI αποτελούν δεδομένα εισόδου εργαλείων παραγωγής κώδικα. Ένα ανταγωνιστικό XML πρότυπο για την αναπαράσταση μεταδεδομένων είναι το *Web Ontology Language (OWL)* που έχει δημιουργηθεί στηριζόμενο στο *Resource Description Framework (RDF)*.

Στο χώρο της μοντελοποίησης με χρήση κυρίως της UML αλλά όχι μόνο, μια πολύ σημαντική γλώσσα είναι η *Object Constraint Language (OCL)* [OCL]. Η OCL είναι μια δηλωτική γλώσσα που χρησιμοποιείται για την περιγραφή κανόνων που εφαρμόζονται σε μοντέλα. Αναπτύχθηκε από την IBM αλλά στη συνέχεια ενσωματώθηκε στο πρότυπο της UML και αποτελεί πρότυπο του OMG. Αρχικά η OCL αποτελούσε μια επέκταση της UML για τη διατύπωση περιορισμών αλλά πλέον μπορεί να χρησιμοποιηθεί σε οποιοδήποτε MOF μοντέλο ή μεταμοντέλο. Μέσω της OCL μπορούν να διατυπωθούν με τυπικό τρόπο περιορισμοί και ερωτήματα τα οποία δε μπορούν να εκφραστούν διαγραμματικά. Οι δηλώσεις της γλώσσας αποτελούνται από τέσσερα τμήματα: το *context* που αφορά στην εμβέλεια της δήλωσης (για παράδειγμα μια κλάση), το *property* που σχετίζεται με κάποιο χαρακτηριστικό του context, το *operation* που χειρίζεται ή αποτιμά ένα property και λέξεις κλειδιά (π.χ. *if, then, else, and, or*) που χρησιμοποιούνται για τη σύνθεση εκφράσεων συνθήκης.

Ένα ευρέως χρησιμοποιούμενο περιβάλλον-πλαίσιο μοντελοποίησης είναι το *Eclipse Modeling Framework (EMF)* το οποίο λειτουργεί πάνω στην πλατφόρμα του Eclipse. Μέσω του EMF διευκολύνεται η παραγωγή κώδικά για τη δημιουργία εργαλείων και άλλων εφαρμογών που βασίζονται σε ένα δομημένο μοντέλο δεδομένων. Τα μοντέλα που χειρίζεται το EMF ακολουθούν τον XMI μορφότυπο και μπορούν να εισαχθούν σε αυτό όντας ήδη αποτυπωμένα σε XMI από κάποιο εργαλείο μοντελοποίησης, αλλά και να δημιουργηθούν από επισημασμένες κλάσεις Java ή να εισαχθούν σε XML μορφή.

Το EMF ξεκίνησε ως μια υλοποίηση της προδιαγραφής του MOF. Εξελίχθηκε όμως, βασισμένο στην εμπειρία που η κοινότητα ανάπτυξης απέκτησε υλοποιώντας ένα μεγάλο πλήθος εργαλείων που το χρησιμοποιούν. Το EMF μπορεί να θεωρηθεί ως μια υψηλής αποδοτικότητας υλοποίηση σε Java ενός υποσυνόλου πυρήνα του MOF API [EMF]. Για να αποφευχθεί όμως οποιαδήποτε σύγχυση, το μεταμοντέλο πυρήνα του EMF που είναι σαν το MOF καλείται *Ecore*. Μάλιστα, στην πρόταση για το MOF 2.0 ορίζεται ένα παρόμοιο υποσύνολο του MOF –το οποίο αποκαλείται *EMOF* (*Essential MOF*)- με το οποίο το Ecore εμφανίζει μικρές, ονομαστικές διαφορές..

Τέλος, στο EMF βασίζονται δύο σημαντικές υλοποιήσεις: το *XML Schema Infoset (XSD)* και το *Service Data Objects (SDO)*. Το XSD παρέχει ένα μοντέλο κι ένα API για το χειρισμό των στοιχείων ενός σχήματος XML, με πρόσβαση στην υποκείμενη *DOM (Document Object Model)* αναπαράσταση του εγγράφου σχήματος. Επίσης στα πλαίσια του EMF έχουν αναπτυχθεί έργα όπως τα *Model Query, Model Transaction* και *Validation Framework.*

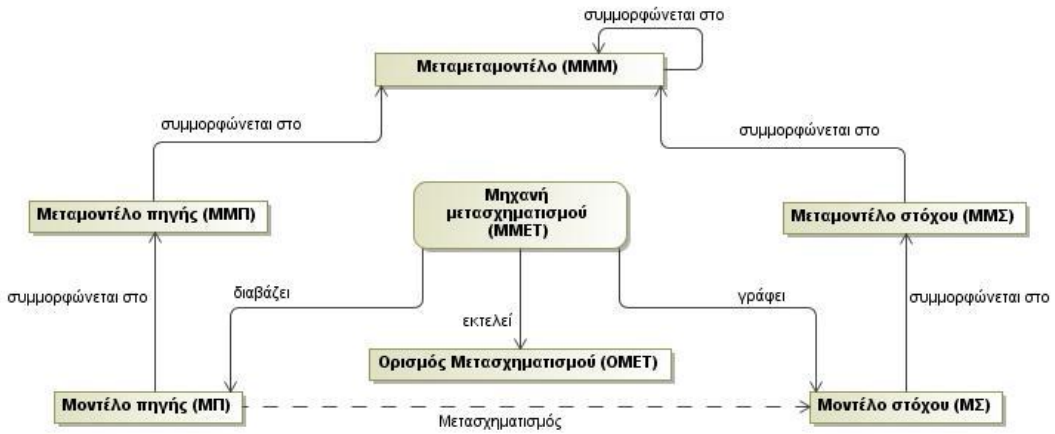
Έως τώρα παρουσιάστηκαν πρότυπα και τεχνολογίες μέσω των οποίων είναι δυνατός ο καθορισμός, η επεξεργασία, η αποθήκευση και η ανταλλαγή μοντέλων. Στη συνέχεια της ενότητας σχολιάζεται συνοπτικά η έννοια των μοντέλων και του μετασχηματισμού μοντέλων στο χώρο της μηχανικής λογισμικού, δίνεται μια εποπτική εικόνα της διαδικασίας μετασχηματισμού μοντέλων και παρουσιάζονται περιβάλλοντα και γλώσσες που έχουν προταθεί για τη διαδικασία αυτή.

Μια από τις χρήσεις των μοντέλων στη μηχανική λογισμικού είναι η έκφραση αφαιρέσεων σε επίπεδα υψηλότερα από αυτό του πηγαίου κώδικα, όπως απαιτήσεις και προδιαγραφές σχεδιασμού. Ενίοτε ακόμα και ο κώδικας προγράμματος μπορεί να θεωρηθεί μοντέλο (ως αφαίρεση του κώδικα μηχανής που παράγεται από τον μεταγλωττιστή). Το πως εκφράζονται τα μοντέλα έχει να κάνει με το είδος των εφαρμογών που αναπαριστούν, ή με κάποια συγκεκριμένη διάσταση μιας εφαρμογής. Συνήθως είναι διαθέσιμη κάποια γραφική αναπαράσταση αλλά χωρίς αυτό να είναι απαραίτητο.

Θεωρώντας ότι και ο κώδικας αποτελεί μοντέλο τότε ο μετασχηματισμός μοντέλων είναι ένας όρος που καλύπτει τον μετασχηματισμό προγράμματος χωρίς να περιορίζεται σε αυτόν.

Επειδή η αντίληψη των μοντέλων είναι πιο γενική από εκείνη των προγραμμάτων, οι μετασχηματισμοί των πρώτων ενεργούν σε ένα πιο ποικίλο σύνολο στοιχείων. Στη βιβλιογραφία ορίζονται μετασχηματισμοί μοντέλων που αφορούν την ανάπτυξη λογισμικού σε ένα ευρύ φάσμα περιοχών, όπως μεταξύ UML μοντέλων, προδιαγραφών διεπαφής, σχημάτων δεδομένων, περιγραφών δομοστοιχείων και βέβαια κώδικα προγράμματος [CH06].

Στη διαδικασία μετασχηματισμού μοντέλων έχουμε τυπικά ένα *μοντέλο πηγή* (*source model*) ΜΠ κι ένα *μοντέλο στόχο* (*target model*) ΜΣ. Το κάθε ένα από αυτά συμμορφώνεται σε κάποιο μεταμοντέλο, ΜΜΠ και ΜΜΣ αντίστοιχα. Αν τα μεταμοντέλα αυτά ταυτίζονται έχουμε «*ενδογενή*» μετασχηματισμό, διαφορετικά «*ετερογενή*». Η μετάβαση από το μοντέλο πηγή στο μοντέλο στόχο υλοποιείται από μια *μηχανή μετασχηματισμού* (ΜΜΕΤ) η οποία εκτελεί τον ορισμό του μετασχηματισμού (ΟΜΕΤ), δηλαδή τους *κανόνες αντιστοίχησης* του ΜΜΠ στο ΜΜΣ. Τα μεταμοντέλα συμμορφώνονται με τη σειρά τους σε ένα (κοινό) μεταμεταμοντέλο (ΜΜΜ). Το ΜΜΜ συμμορφώνεται στον εαυτό του. Μια απεικόνιση της διαδικασίας και των σχέσεων μεταξύ των στοιχείων δίνεται στο Σχήμα 3.2.



Σχήμα 3.2: Μετασχηματισμός μοντέλων

Το OMG αναγνώρισε την ανάγκη για τη δημιουργία ενός προτύπου ως προς τον ορισμό μετασχηματισμών, συμβατό με τα υπόλοιπα πρότυπα του OMG στην περιοχή της Αρχιτεκτονικής Οδηγούμενης από Μοντέλα (*Model Driven Architecture - MDA*). Τον Απρίλιο του 2002 κατέθεσε ένα *Request for Proposal (RFP)* πάνω σε MOF Ερωτήματα/Όψεις/Μετασχηματισμοί (*Query/Views/Transformations*) το οποίο, μετά από τρία χρόνια οδήγησε στην υιοθέτηση και δημοσίευση του *QVT* προτύπου. Σύμφωνα με το πρότυπο αυτό τα μοντέλα που συμμετέχουν στον μετασχηματισμό όπως και τα μεταμοντέλα τους πρέπει να συμμορφώνονται στο MOF μεταμοντέλο. Το ίδιο πρέπει να ισχύει για το

πρόγραμμα μετασχηματισμού καθαυτό. Κατά συνέπεια το αφηρημένο συντακτικό (*abstract syntax*) του QVT πρέπει να συμμορφώνεται στο MOF μεταμοντέλο.

Η γλώσσα QVT ενσωματώνει το πρότυπο της OCL και το επεκτείνει σε μια προστατική μορφή της OCL. Επίσης το QVT ορίζει τρεις γλώσσες εξειδικευμένου πεδίου (*domain-specific languages)* που ονομάζονται *Relations*, *Core* και *Operational Mappings*, οργανωμένες σε μια αρχιτεκτονική επιπέδων. Οι γλώσσες Relations και Core είναι δηλωτικές, σε δύο διαφορετικά επίπεδα αφαίρεσης. Η Relations έχει γραφικό συντακτικό ενώ υπάρχει μετασχηματισμός Relations σε Core. Η γλώσσα Operational Mappings είναι προστακτικού τύπου παρέχοντας δομές όπως loops, συνθήκες κλπ, ενώ ταυτόχρονα επεκτείνει τόσο τη Relations όσο και την Core. Επιπλέον, στο QVT πρότυπο περιγράφεται ένας μηχανισμός για την κλήση μετασχηματισμών που έχουν γραφτεί σε άλλες γλώσσες (*BlackBox*). Τέλος, το πρότυπο χειρίζεται μόνο μετασχηματισμούς μοντέλου σε μοντέλο, χωρίς δηλαδή να υποστηρίζει μετασχηματισμούς του τύπου μοντέλο σε κείμενο, κείμενο σε μοντέλο κλπ.

Το μεταμοντέλο που χρησιμοποιήθηκε στην υλοποίηση του μετασχηματισμού είναι το EMF UML2. Το UML2 είναι μια υλοποίηση του μεταμοντέλου της UML 2.x του OMG, βασισμένο στο EMF, για την πλατφόρμα του Eclipse. Οι στόχοι του UML2 έργου όπως καταγράφονται στο [UML2] είναι να παρέχει μια υλοποίηση του μεταμοντέλου της UML που μπορεί να χρησιμοποιηθεί για την ανάπτυξη εργαλείων, ένα κοινό σχήμα XMI που θα διευκολύνει την ανταλλαγή μεταξύ σημασιολογικών μοντέλων, περιπτώσεις ελέγχου ως μέσο επικύρωσης της προδιαγραφής, κανόνες επικύρωσης ως ένα μέσο ορισμού και εφαρμογής επιπέδων συμμόρφωσης.

Μια από τις απαντήσεις στο παραπάνω RFP του OMG υπήρξε η γλώσσα μετασχηματισμού μοντέλων ATL, της ερευνητικής ομάδας *ATLAS INRIA & LINA*. Η γλώσσα ATL έχει ορισθεί τόσο ως μεταμοντέλο όσο και ως συμπαγές συντακτικό κειμένου και έχει υβριδική φύση: δηλωτική και προστακτική. Το στυλ συγγραφής των μετασχηματισμών που προτιμάται είναι το δηλωτικό αλλά παρόλα αυτά, στις περιπτώσεις κατά τις οποίες ο δηλωτικός χειρισμός των προτύπων αντιστοίχισης είναι ιδιαίτερα σύνθετος μπορούν να επιστρατευθούν προστατικές δομές. Όταν τα σύνθετα αυτά πρότυπα αντιστοίχησης αναγνωριστούν τότε μπορούν να προστεθούν δηλωτικές δομές ώστε να απλοποιήσουν τη συγγραφή του μετασχηματισμού [ATL].

Η ATL χρησιμοποιείται συνήθως στο περιβάλλον του Eclipse και είναι ενταγμένη στο έργο M2M [M2M] ως δομοστοιχείο που παρέχει ένα σύνολο εργαλείων για μετασχηματισμούς μοντέλων.

Ο μετασχηματισμός Διαγραμμάτων Ακολουθίας σε Δίκτυα Petri που ορίζεται σε αυτή την εργασία έχει υλοποιηθεί στη γλώσσα ATL. Μια περιγραφή των χαρακτηριστικών, του συντακτικού και των τρόπων χρήσης της γλώσσας μπορεί να βρεθεί στο [ATL UM].

3.3 Ορισμός και ανάλυση Δικτύων Petri

Τα Δίκτυα Petri είναι γραφικά και μαθηματικά εργαλεία μοντελοποίησης τα οποία μπορούν να εφαρμοσθούν σε ένα πλήθος συστημάτων και εμφανίζουν ιδιαίτερα προτερήματα για την περιγραφή και την μελέτη συστημάτων επεξεργασίας πληροφοριών που χαρακτηρίζονται ως σύνδρομα, ασύγχρονα, κατανεμημένα, παράλληλα, μη ντετερμινιστικά και/ή στοχαστικά [Mur89].

Ο τυπικός ορισμός ενός Δικτύου Petri είναι ο ακόλουθος [Mur89]:

Ένα Δίκτυο Petri είναι μια πεντάδα $PN = (P, T, F, W, M_0)$ όπου:

$P = \{p_1, p_2, \ldots, p_m\}$ είναι ένα πεπερασμένο σύνολο θέσεων,

$T = \{t_1, t_2, \ldots, t_n\}$ είναι ένα πεπερασμένο σύνολο μεταβάσεων,

$F \subseteq (P \times T) \cup (T \times P)$ είναι ένα σύνολο διανυσμάτων (σχέση ροής),

$W: F \rightarrow \{1, 2, 3, \ldots\}$ είναι μια συνάρτηση βάρους,

$M_0: P \rightarrow \{0, 1, 2, 3, \ldots\}$ είναι ο αρχικός χαρακτηρισμός,

$P \cap T = \varnothing$ και $P \cup T \neq \varnothing$

Μια δομή Δικτύου Petri $N = (P, T, F, W)$ χωρίς αρχικό χαρακτηρισμό συμβολίζεται ως N.

Ένα Δίκτυο Petri με δομή N και αρχικό χαρακτηρισμό M_0 συμβολίζεται ως (N, M_0).

Ένα Δίκτυο Petri είναι ένα συγκεκριμένο είδος κατευθυνόμενου γράφου μαζί με μια αρχική κατάσταση που καλείται αρχικός χαρακτηρισμός (*initial marking*). Αποτελείται από δύο είδη κόμβων, τις *θέσεις* (*places*) και τις *μεταβάσεις* (*transitions*) και από *διανύσματα* (*arcs*) μεταξύ των κόμβων. Τα διανύσματα μπορούν να είναι είτε *διανύσματα εισόδου* (*input arcs*) τα οποία ξεκινούν από θέσεις και καταλήγουν σε μεταβάσεις, είτε *διανύσματα εξόδου* (*output arcs*) τα οποία έχουν κατεύθυνση από τις μεταβάσεις προς τις θέσεις. Επίσης τα διανύσματα μπορούν να φέρουν βάρη (θετικούς ακεραίους), όπου ένα βέλος με βάρος *k* μπορεί να μεταφραστεί ως *k* παράλληλα διανύσματα. Πέρα από τα παραπάνω υπάρχουν διανύσματα άλλων τύπων όπως τα *inhibitor arcs* για την έκφραση ειδικών λειτουργιών.

Σε συμφωνία με τα διανύσματα εισόδου και εξόδου, ορίζονται για κάθε μετάβαση *θέσεις εισόδου* και *θέσεις εξόδου*. Δηλαδή θέσεις εισόδου (και αντίστοιχα εξόδου) για μια μετάβαση

θεωρούνται οι θέσεις που συνδέονται μαζί της με διανύσματα εισόδου (εξόδου). Επιπλέον, οι θέσεις μπορούν να περιέχουν *token* (σημεία/αδειοδοτικά). Τα token ανατίθενται στις θέσεις ενός Δικτύου Petri και είτε μένουν εκεί ή είτε «καταναλώνονται». Μέσω των token ορίζεται η εκτέλεση ενός Δικτύου Petri και τόσο ο αριθμός όσο και η θέση τους μπορεί να μεταβάλλονται κατά τη διάρκεια της εκτέλεσης. Ο αρχικός χαρακτηρισμός ενός Δικτύου Petri καθορίζεται από τον αρχικό αριθμό -και τον τύπο αν πρόκειται για τύπο δικτύου στον οποίο τα token μπορούν να διακριθούν- των token σε κάθε θέση, πριν από οποιαδήποτε εκτέλεση μετάβασης. Κάθε κατάσταση που προκύπτει κατά τη διάρκεια της εκτέλεσης ονομάζεται χαρακτηρισμός (*marking)*.

Οι μεταβάσεις είναι ενεργά στοιχεία και μοντελοποιούν δράσεις. Μια μετάβαση μπορεί να εκτελεστεί (*fire*) και να αλλάξει το χαρακτηρισμό του Δικτύου Petri, όπως μια δράση μπορεί να λάβει χώρα μεταβάλλοντας την κατάσταση ενός συστήματος. Για να εκτελεστεί μια μετάβαση πρέπει να είναι ενεργοποιημένη (*enabled*), το οποίο σύμφωνα με την προηγούμενη αναλογία σημαίνει ότι πρέπει να πληρούνται όλες οι προϋποθέσεις για τη δράση που μοντελοποιεί. Στα Δίκτυα Petri το ρόλο των συνθηκών παίζουν οι θέσεις. Μια μετάβαση είναι ενεργοποιημένη όταν υπάρχουν τα αρκετά token διαθέσιμα στις θέσεις εισόδου – παρόλα αυτά μια ενεργοποιημένη μετάβαση, δεν εκτελείται απαραίτητα. Όταν η μετάβαση εκτελείται τότε απομακρύνει («καταναλώνει») token από τις θέσεις εισόδου και προσθέτει token στις θέσεις εξόδου. Ο αριθμός των token που θα καταναλωθούν και θα προστεθούν εξαρτάται από το βάρος του κάθε διανύσματος που συνδέεται με τη μετάβαση (είτε ως πηγή, είτε ως προορισμός).

Ένας συχνά χρησιμοποιούμενος όρος είναι ο «*token game*» και αναφέρεται στη διαδραστική εκτέλεση των μεταβάσεων και στη σειρά των διαδοχικών markings που προκύπτουν.

Γραφικά, οι μεταβάσεις αναπαριστώνται ως μπάρες ή κουτιά, οι θέσεις ως κύκλοι, τα διανύσματα ως βέλη (τα οποία ενδεχομένως να φέρουν κάποια ετικέτα βάρους) και τα (απλά) token ως τελείες.

Η ιδέα των Δικτύων Petri εισήχθη από τον Carl Adam Petri το 1962 στη διδακτορική του διατριβή [Pet62]. Από τότε έχουν προταθεί μια σειρά από επεκτάσεις και διαφορετικούς τύπους Δικτύων Petri. Η γενική κατηγοριοποίηση των Δικτύων Petri που χρησιμοποιείται ορίζει τρία βασικά επίπεδα δικτύων. Τα Δίκτυα Petri *επιπέδου 1* χαρακτηρίζονται από «*boolean tokens*», δηλαδή οι θέσεις περιέχουν το πολύ ένα (απλό) token. Στο *επίπεδο 2* κατατάσσονται τα Δίκτυα Petri των οποίων οι θέσεις ενδεχομένως να περιέχουν περισσότερα του ενός (απλά) token και χαρακτηρίζονται από «*integer tokens*». Τα Δίκτυα Petri του *επιπέδου 3* χαρακτηρίζονται από «*high-level tokens*», δηλαδή οι θέσεις περιέχουν δομημένα (σύνθετα) tokens τα οποία φέρουν πληροφορία.

Η περιγραφή που προηγήθηκε αφορά κυρίως στο βασικό τύπο των Δικτύων Petri που εναλλακτικά καλούνται και *Place/Transition Nets* (*P/T nets*). Πέρα όμως από τα P/T nets έχει προταθεί ένα πλήθος τύπων όπως τα Έγχρωμα Δίκτυα Petri (*Colored Petri Nets*), τα Predicate/Transition Nets, τα Timed Petri Nets, τα Stochastic Petri Nets, τα Modular Petri Nets, τα Object Petri Nets, τα M-Nets, και άλλοι. Οι δύο πρώτοι τύποι καλούνται και *υψηλού επιπέδου Δίκτυα Petri* (High-Level Petri Nets) και ανήκουν στο επίπεδο 3, της παραπάνω κατηγοριοποίησης. Τα Timed Petri Nets ενσωματώνουν την έννοια του χρόνου στο μοντέλο: μια μετάβαση πρέπει να περιμένει συγκεκριμένο χρονικό διάστημα για να ενεργοποιηθεί. Στα δίκτυα αυτά μπορούν να αναγνωριστούν διαφορετικά είδη μεταβάσεων ανάλογα με τη κατανομή του χρόνου που πρέπει να περιμένουν.

Πριν την περιγραφή τεχνικών και εργαλείων ανάλυσης, δίνεται ένα απλό παράδειγμα μοντελοποίησης μιας διαδικασίας με ένα Δίκτυο Petri και περιγράφεται σύντομα η εκτέλεσή του. Ο έμπειρος, ως προς τα Δίκτυα Petri, αναγνώστης μπορεί να το προσπεράσει.

Η παράθεση ενός απλού αλλά ολοκληρωμένου παραδείγματος κρίνεται σημαντική, καθώς στοχεύει στην ανάδειξη των δυνατοτήτων μοντελοποίησης που παρέχουν τα Δίκτυα Petri και της καταλληλότητάς τους για την έκφραση παράλληλων λειτουργιών και διεργασιών με πολλαπλά μονοπάτια εκτέλεσης, ιδιαιτέρως στον αναγνώστη που δεν είναι εξοικειωμένος με το μαθηματικό και γραφικό αυτό εργαλείο.

Ο τίτλος που μπορεί να δοθεί στη διαδικασία που μοντελοποιείται στο παράδειγμα είναι «*Απόκτηση Διπλώματος ΗΜΜΥ ΕΜΠ*». Μέσω ενός Δικτύου Petri επιπέδου 2, θα εκφραστούν τις απαιτήσεις για την απόκτηση διπλώματος που αφορούν την εισαγωγή, την εκπλήρωση των μαθημάτων, την εκπόνηση της διπλωματικής εργασίας και την απονομή του διπλώματος. Πρόκειται βέβαια για μια χοντρική προσέγγιση που όμως αποτυπώνει τις βασικές προϋποθέσεις που θέτει ο τρέχων οδηγός σπουδών και το πλαίσιο λειτουργίας της σχολής.

Συγκεκριμένα:

- Ένας υποψήφιος μπορεί να εγγραφεί στη σχολή μέσω τριών οδών: πανελλήνιες εξετάσεις, κατατακτήριες εξετάσεις, μετεγγραφή/ειδικές κατηγορίες.

- Αφού αποκτήσει τη φοιτητική ιδιότητα, για την απόκτηση του διπλώματος πρέπει να εξεταστεί επιτυχώς σε 60 τουλάχιστον μαθήματα. Προαιρετικά μπορεί να αναλάβει και να εξεταστεί επιτυχώς σε ως και 4 επιπλέον μαθήματα, τα οποία όμως δε λαμβάνονται υπόψη στον υπολογισμό του βαθμού.

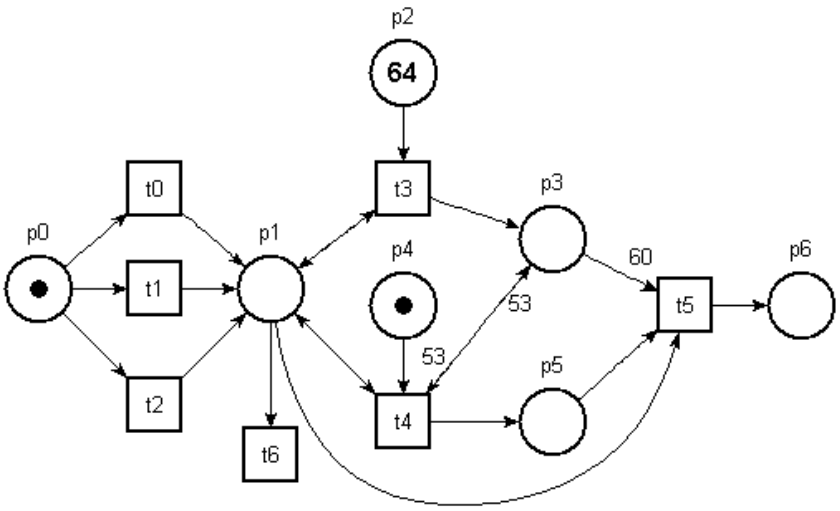
- Για την απόκτηση του διπλώματος απαιτείται επίσης η εκπόνηση και η επιτυχής εξέταση σε μια διπλωματική εργασία. Η εργασία μπορεί να αναληφθεί και να εκπονηθεί

μόνο αν ο φοιτητής οφείλει 7 ή λιγότερα μαθήματα από τα απαιτούμενα για τη λήψη του διπλώματος (δηλαδή πρέπει να έχει ολοκληρώσει τουλάχιστον 53 μαθήματα).

- Μετά την ολοκλήρωση των μαθημάτων και της διπλωματικής, ο φοιτητής πρέπει να καταθέσει την εργασία του στη βιβλιοθήκη και να κινήσει τη διαδικασία αποφοίτησης μέσω της γραμματείας (αίτηση αποφοίτησης, έκδοση πιστοποιητικών κλπ).

- Αν η φοιτητική ιδιότητα χαθεί σε οποιαδήποτε φάση της διαδικασίας για άλλους λόγους, όπως για παράδειγμα λόγω εγγραφής του φοιτητή σε άλλη σχολή ή υπέρβασης του ανωτάτου χρονικού ορίου φοίτησης κλπ, τότε η διαδικασία απόκτησης διπλώματος ΗΜΜΥ ΕΜΠ οδηγείται σε αδιέξοδο.

Ένα Δίκτυο Petri που μοντελοποιεί την παραπάνω διαδικασία, λαμβάνοντας υπόψη τους περιορισμούς και τις προϋποθέσεις που διατυπώθηκαν, απεικονίζεται στο Σχήμα 3.3.



Σχήμα 3.3: «Απόκτηση Διπλώματος ΗΜΜΥ ΕΜΠ»

Ακολουθεί η επεξήγηση των στοιχείων του μοντέλου:

Μεταβάσεις:

t0, t1, t2: αντιστοιχούν στους τρεις τρόπου εισαγωγής στη σχολή,

t6: απώλεια της φοιτητικής ιδιότητας για άλλους λόγους,

t3: ολοκλήρωση μαθήματος (εγγραφή και επιτυχής εξέταση),

t4: ολοκλήρωση διπλωματικής εργασίας (ανάληψη, εκπόνηση και επιτυχής εξέταση),

t5: γραφειοκρατική διαδικασία αποφοίτησης

Θέσεις:

p0: ενδιαφέρον απόκτησης Διπλώματος ΗΜΜΥ ΕΜΠ

p1: φοιτητική ιδιότητα

p2: υπόλοιπο μαθημάτων προς ολοκλήρωση

p3: μαθήματα που εξετάστηκαν με επιτυχία

p4: υποχρέωση εκπόνησης διπλωματικής εργασίας

p5: επιτυχία εξέτασης διπλωματικής εργασίας

p6: κατοχή Διπλώματος ΗΜΜΥ ΕΜΠ

Η εκτέλεση του δικτύου μπορεί να ακολουθήσει την εξής διαδικασία: Ο ενδιαφερόμενος (p0) υποψήφιος εισάγεται με έναν από τους τρεις τρόπους (t0, t1, t2) στη σχολή και αποκτά την φοιτητική ιδιότητα (p1). Όσο διατηρεί τη φοιτητική ιδιότητα μπορεί να συμμετέχει στη διαδικασία ολοκλήρωσης μαθημάτων (t3). Από τη στιγμή που ολοκληρώσει 53 (p3, βάρη διανυσμάτων t4→p3, p3→t4) μαθήματα και εφόσον διατηρεί τη φοιτητική ιδιότητα μπορεί να αναλάβει διπλωματική εργασία, να την εκπονήσει και να εξεταστεί επιτυχώς (t4). Όταν ολοκληρώσει 60 (p3, βάρος διανύσματος p3→t5) μαθήματα και τη διπλωματική (p5) μπορεί να κινήσει τη γραφειοκρατική διαδικασία αποφοίτησης (t5) ώστε να καταστεί διπλωματούχος (p6) χάνοντας ταυτόχρονα τη φοιτητική ιδιότητα (p1). Αν οποιαδήποτε στιγμή μετά την εισαγωγή στη σχολή χαθεί η φοιτητική ιδιότητα για άλλους λόγους (t6), τότε χάνεται άμεσα η δυνατότητα συμμετοχής στις λειτουργίες της σχολής και η προοπτική απόκτησης Διπλώματος.

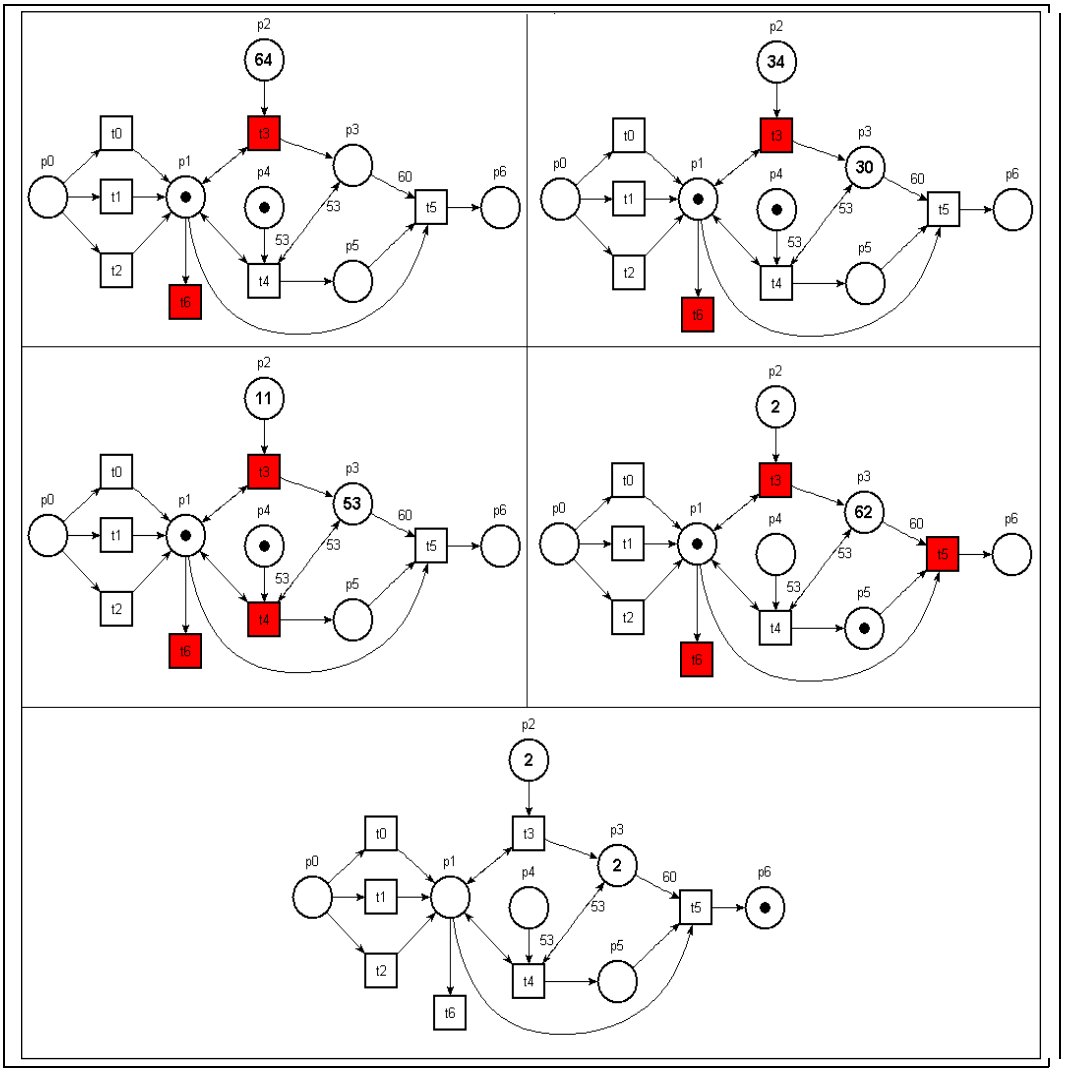
Αν και το παράδειγμα που περιγράφηκε είναι απλό, οι δυνατές καταστάσεις (δηλαδή τα διαφορετικά markings) στις οποίες μπορεί να βρεθεί το μοντέλο είναι 160. Αν αποφασίσουμε να μη μοντελοποιήσουμε την απώλεια της φοιτητικής ιδιότητας για άλλους λόγους (t6) και άρα θεωρήσουμε ότι μετά την εισαγωγή, ο φοιτητής αναμένεται να αποφοιτήσει και μόνο, τότε ο χώρος καταστάσεων για το μοντέλο μειώνεται σχεδόν στο μισό (83). Αυτό συμβαίνει γιατί στο αρχικό μοντέλο η απώλεια της φοιτητικής ιδιότητας για άλλους λόγους μπορεί να συμβεί οποιαδήποτε στιγμή μετά την εισαγωγή στη σχολή (και πριν τη αποφοίτηση), δηλαδή ασύγχρονα και ασυσχέτιστα με οποιαδήποτε άλλη δραστηριότητα στα πλαίσια της σχολής. Επίσης, επειδή κάποιος φοιτητής μπορεί να αναλάβει και να εκπονήσει διπλωματική οποιαδήποτε στιγμή μετά την ολοκλήρωση τουλάχιστον 53 μαθημάτων, η ολοκλήρωση μαθημάτων μετά τα πρώτα 53 και η ολοκλήρωση διπλωματικής είναι δράσεις που μπορούν να συμβούν παράλληλα. Αυτές οι συνθήκες όπως και το γεγονός ότι κάποιος φοιτητής μπορεί να ολοκληρώσει ως και 4 περισσότερα μαθήματα από τα απαιτούμενα δρουν αυξητικά στον

χώρο των καταστάσεων του συστήματος καθιστώντας ταυτόχρονα την ανάλυση και τον έλεγχο του συστήματος πιο απαιτητικό.

Στο Σχήμα 3.4 περιλαμβάνονται ορισμένες χαρακτηριστικές καταστάσεις στις οποίες μπορεί να βρεθεί που δίκτυο (με τις εκάστοτε ενεργοποιημένες μεταβάσεις να είναι χρωματισμένες). Διαβάζοντας το σχήμα από τα αριστερά προς τα δεξιά και από πάνω προς τα κάτω, οι καταστάσεις μπορούν να θεωρηθούν στιγμιότυπα μιας πορείας εκτέλεσης από την αρχική σε μια τελική κατάσταση –όχι βέβαια συνεχόμενα. Τελική κατάσταση θεωρείται η κατοχή διπλώματος, δηλαδή η ύπαρξη token στη θέση p6. Έχουμε όμως 5 τελικές καταστάσεις, τα markings των οποίων διαφέρουν ως προς τον συνδυασμό των token που απομένουν στις θέσεις p2 και p3, και από το αν και από το πόσα επιπλέον μαθήματα ολοκλήρωσε ο φοιτητής.

Μια τελευταία παρατήρηση που μπορεί να γίνει είναι ότι στο μοντέλο που χρησιμοποιήθηκε δε διαχωρίστηκαν τα μαθήματα μεταξύ τους ή η ολοκλήρωση του καθενός από αυτά ως ειδική πράξη. Σε μια τέτοια περίπτωση το δίκτυο θα γιγαντώνονταν κι αυτό σχετίζεται με την εγγενή τάση των (χαμηλού επιπέδου) Δικτύων Petri να γίνονται πολύ μεγάλα σε μέγεθος ακόμα και για σχετικά απλά συστήματα.

Η παρουσίαση του παραδείγματος στοχεύει στην ανάδειξη ορισμένων δυνατοτήτων που προσφέρουν τα Δίκτυα Petri ως εργαλείο μοντελοποίησης αλλά και στο να δοθεί μια πρώτη, ποιοτική αίσθηση της πολυπλοκότητας που η μοντελοποίηση και η ανάλυση σύνδρομων, ασύγχρονων και μη ντετερμινιστικών συστημάτων εμπεριέχει.



Σχήμα 3.4: Χαρακτηριστικές καταστάσεις μοντέλου

Στη συνέχεια της ενότητας γίνεται μια καταγραφή των ιδιοτήτων που μπορούν να μελετηθούν και των μεθόδων ανάλυσης που μπορούν να εφαρμοσθούν σε ένα Δίκτυο Petri. Έπειτα, παρουσιάζονται ορισμένα εργαλεία δημιουργίας, επεξεργασίας, ανάλυσης και ελέγχου Δικτύων Petri. Η ενότητα κλείνει με μια σύντομη αναφορά στη γλώσσα *PNML*, που είναι μια προσπάθεια προτυποποίησης της έκφρασης των διαφόρων τύπων Δικτύων Petri σε XML μορφή.

Ένας δείκτης επιτυχίας ή αποτυχίας κάποιας γλώσσας μοντελοποίησης είναι το πόσες, ποιες και πόσο σημαντικές ιδιότητες των μοντέλων που η γλώσσα προσδιορίζει μπορούν να μελετηθούν. Ένα από τα μειονεκτήματα της UML είναι ότι αν και διαθέτει πλούσια εκφραστικά μέσα, δε διαθέτει τυπικά χαρακτηριστικά και οι ιδιότητες των μοντέλων δεν μπορούν να εξετασθούν με τυπικές μεθόδους. Αυτό σημαίνει ότι τα (μαθηματικά) ερωτήματα που μπορούν να τεθούν και τα συμπεράσματα που μπορούν να εξαχθούν για το σύστημα μέσω του μοντέλου είναι περιορισμένα. Η δυνατότητα μελέτης των ιδιοτήτων, οι

θεμελιωμένες τεχνικές ανάλυσης και οι ενιαίοι και απλοί κανόνες εκτέλεσης των Δικτύων Petri, αποτέλεσαν βασικό κίνητρο για την υλοποίηση του μετασχηματισμού των Διαγραμμάτων Ακολουθίας σε Δίκτυα Petri, τόσο για την παρούσα εργασία, όσο και για σχετικές προσεγγίσεις διαφόρων ερευνητών (βλ. 2.2).

Τα Δίκτυα Petri επιτρέπουν τη μελέτη ενός πλήθους ιδιοτήτων οι οποίες μπορούν να χωριστούν σε ιδιότητες εξαρτώμενες από το initial marking και σε ανεξάρτητες από αυτό. Οι πρώτες ονομάζονται ιδιότητες συμπεριφοράς (*behavioral properties*), ενώ οι δεύτερες, δομικές (*structural properties*). Στις ιδιότητες συμπεριφοράς περιλαμβάνονται: η προσβασιμότητα (*reachability*), η ικανότητα φραγμού (*boundedness*), η ζωτικότητα (*liveness*), η ασφάλεια (*safeness*), η δυνατότητα επιστροφής (*reversibility*) είτε στην αρχική κατάσταση είτε σε κάποια άλλη καθορισμένη κατάσταση (*home state*), η δυνατότητα κάλυψης (*coverability*), η διατήρηση (*persistence*), η συγχρονική απόσταση (*synchronic distance*) και η δικαιοσύνη (*fairness*). Στις δομικές ιδιότητες ενός Δικτύου Petri περιλαμβάνονται: η δομική ζωτικότητα (*structural liveness*), η δυνατότητα ελέγχου (*controllability*), η δομική ικανότητα φραγμού (*structural boundedness*), η απόρροια (*corollary*), η συντηρητικότητα (*conservativeness*), η ικανότητα επανάληψης (*repetitiveness*), η συνέπεια (*consistency*), οι *S*- και *T-invariants* και η δομική *B-fairness*.

Οι μέθοδοι ανάλυσης μπορούν να χωριστούν σε τρεις κατηγορίες: Η πρώτη περιλαμβάνει απαραίτητα την απαρίθμηση όλων των markings στα οποία υπάρχει πρόσβαση ή μπορούν να καλυφθούν. Η μέθοδος αυτή μπορεί να θεωρητικά να εφαρμοστεί σε οποιοδήποτε μοντέλο αλλά λόγω της πολυπλοκότητας που εμπεριέχει η έκρηξη του χώρου καταστάσεων πρέπει να εφαρμόζεται μόνο σε σχετικά περιορισμένα δίκτυα. Η δεύτερη κατηγορία μεθόδων ανάλυσης περιλαμβάνει εξισώσεις πινάκων που προκύπτουν από τη δομή και το marking των μοντέλων και η τρίτη κατηγορία αφορά σε επαγωγικές τεχνικές και τεχνικές αποσύνθεσης.

Η διεξοδική περιγραφή των παραπάνω ιδιοτήτων και τεχνικών ξεφεύγει από τους στόχους της παρούσας εργασίας. Περαιτέρω πληροφορίες μπορούν να αναζητηθούν στις εργασίες [Mur89][Pet77] και τυπικά σε οποιοδήποτε σύγγραμμα σχετικό με τα Δίκτυα Petri.

Για τον σχεδιασμός και την ανάλυση μοντέλων Δικτύων Petri διαφόρων τύπων έχει υλοποιηθεί και προσφέρεται μια πληθώρα εργαλείων. Οι δυνατότητες που προσφέρουν είναι συνήθως μια γραφική διεπαφή για την σχεδίαση των δικτύων, λειτουργίες εκτέλεσης/προσομοίωσης (με ή χωρίς κινούμενες εικόνες), υπολογισμό του χώρου καταστάσεων, δομική ανάλυση, ανάλυση επίδοσης (για *Timed Petri Nets*) και άλλες. Στη συνέχεια θα περιγραφούν συνοπτικά δύο εργαλεία που παρέχουν ένα πλήθος δυνατοτήτων,

υποστηρίζουν χαρακτηριστικά διαλειτουργικότητας και διατίθενται δωρεάν. Μια ενημερωμένη λίστα με περιγραφές δεκάδων εργαλείων υπάρχει στον ιστοσελίδα [PTTOOL].

Το εργαλείο *TINA* [TINA] υποστηρίζει Place/Transition Nets, Timed Petri Nets και γενικότερα Δίκτυα Petri με χαρακτηριστικά χρόνου, έχει έναν ιδιαίτερα εύχρηστο γραφικό επεξεργαστή και παρέχει δυνατότητες εκτέλεσης (διαδραστικής, τυχαίας αλλά και βάση κάποιου αρχείου με ακολουθίες μεταβάσεων) και ανάλυσης (χώρου καταστάσεων, state και transition invariants, κ.ά.). Ένα από τα σημαντικά χαρακτηριστικά του εργαλείου είναι ότι αν και χρησιμοποιεί μια ιδιωτική γλώσσα για τον ορισμό των μοντέλων, υποστηρίζει την εισαγωγή από και την εξαγωγή σε άλλες γλώσσες έκφρασης Δικτύων Petri (όπως η PNML). Επιπλέον στο TINA έχουν ενσωματωθεί εργαλεία που επιτρέπουν την αυτόματη γραφική σχεδίαση των δικτύων που δίνονται σε μορφή κειμένου. Τα εργαλεία αυτά προέρχονται από το πακέτο *Graphviz* της AT&T [GRAATT] και υλοποιούν αλγορίθμους αυτόματης σχεδίασης βάσει των συσχετίσεων των κόμβων των δικτύων που δίνονται. Εκδόσεις του TINA υπάρχουν για αρκετά λειτουργικά συστήματα (Windows, Linux, SunOS και MacOS X). Παρόλα αυτά το εργαλείο αυτό δεν είναι ανοιχτού κώδικα και αν και η δημιουργία επεκτάσεων είναι δυνατή μέσω *Tcl* δε προσφέρει μεγάλα περιθώρια επέμβασης.

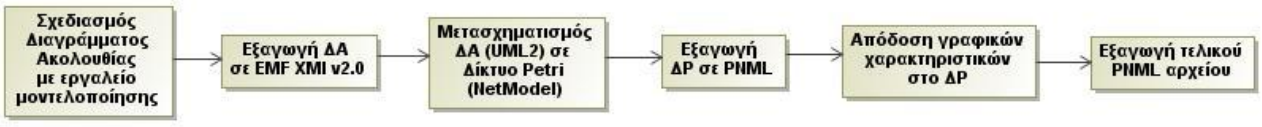
Ένα δεύτερο εργαλείο που εμφανίζει ενδιαφέροντα χαρακτηριστικά είναι το *Platform Independent Petri net Editor 2.5* [PIPE2]. Το εργαλείο αυτό χειρίζεται Place/Transition Nets και Timed Petri Nets και προσφέρει ένα πλήθος λειτουργιών και τεχνικών ανάλυσης (token game/τυχαία εκτέλεση με/χωρίς γραφική αναπαράσταση, ανάλυση χώρου καταστάσεων, state και transition invariants, δομική ανάλυση και άλλα). Επιπλέον, έχει υλοποιηθεί σε Java οπότε μπορεί να εκτελεστεί σε οποιαδήποτε πλατφόρμα με υποστήριξη Java και διαθέτει κατάλληλη υποδομή για την εύκολη προσθήκη επεκτάσεων (*modules*). Επίσης, εμφανίζει σημαντικά χαρακτηριστικά διαλειτουργικότητας υποστηρίζοντας αρχεία PNML και παρέχοντας τη δυνατότητα επέκτασης του μορφότυπου του αρχείου μέσω *XSLT*. Το Platform Independent Petri Net Editor είναι ανοιχτού κώδικα και είναι διαθέσιμο για τροποποιήσεις και επεκτάσεις πέρα από την προσθήκη module.

Η γλώσσα *Petri Net Markup Language (PNML)* αποτελεί έναν μορφότυπο βασισμένο σε XML για την έκφραση Δικτύων Petri. Αποτελεί μια πρόταση στη γενικότερη διαδικασία καθορισμού ενός γενικά αποδεκτού μορφότυπου για την έκφραση Δικτύων Petri και υποστηρίζεται από ένα πλήθος σχετικών εργαλείων. Ένα βασικό χαρακτηριστικό της PNML είναι ότι είναι ανοιχτή σε επεκτάσεις, στοιχείο ιδιαίτερα χρήσιμο για ένα είδος μοντέλων όπως τα Δίκτυα Petri που ακολουθούν πολλούς και διαφορετικούς τύπους, όπως αναφέρθηκε και παραπάνω. Για κάθε τύπο Δικτύων Petri μπορεί να δημιουργηθεί ένας ορισμός μέσω των *Petri Net Type Definitions* τα οποία ορίζουν τα χαρακτηριστικά κάθε τύπου μέσω XML.

Εναλλακτικά, τα χαρακτηριστικά κάποιου τύπου Δικτύου Petri μπορεί να προκύπτουν από μια σύνθεση τέτοιων ορισμών. Περισσότερες πληροφορίες για τη γλώσσα PNML και τα χαρακτηριστικά της μπορούν να αναζητηθούν στον ιστότοπο [PNML].

Κεφάλαιο 4: Μετασχηματισμός Διαγραμμάτων Ακολουθίας σε Δίκτυα Petri

Στο κεφάλαιο αυτό αναλύεται ο μετασχηματισμός των Διαγραμμάτων Ακολουθίας της UML (εκδόσεων 2.x) σε Δίκτυα Petri. Για το σκοπό αυτό παρουσιάζεται το υποσύνολο των χαρακτηριστικών των Διαγραμμάτων Ακολουθίας που ενδιαφέρουν, όπως έχουν οριστεί στην προδιαγραφή της UML και περιγράφεται η χρήση τους για την έκφραση σεναρίων εκτέλεσης σε συστήματα λογισμικού. Κατόπιν αυτού, γίνεται μια αντίστοιχη περιγραφή του μεταμοντέλου και των χαρακτηριστικών του δικτύου που θα χρησιμοποιηθεί. Έπειτα παρουσιάζονται σε γραφική μορφή και σχολιάζονται οι κανόνες αντιστοίχησης των στοιχείων των μεταμοντέλων. Η μετάβαση από ένα Διάγραμμα Ακολουθίας που δημιουργείται σε κάποιο εργαλείο μοντελοποίησης σε ένα Δίκτυο Petri αναγνώσιμο από ένα εργαλείο ελέγχου και ανάλυσης γίνεται μέσω μιας αλληλουχίας βημάτων. Η διαδικασία αυτή περιγράφεται αναλυτικά σε αντίστοιχη ενότητα του κεφαλαίου ενώ μια πρώτη, εποπτική εικόνα της δίνεται στο Σχήμα 4.1. Στο τέλος του κεφαλαίου αναφέρονται και σχολιάζονται όρια και περιορισμοί που τίθενται για και από τον προτεινόμενο μετασχηματισμό.



Σχήμα 4.1: Διαδικασία μετασχηματισμού Διαγραμμάτων Ακολουθίας σε Δίκτυα Petri

4.1 Μοντέλο Πεδίου Διαγραμμάτων Ακολουθίας

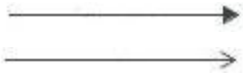
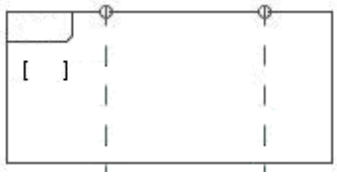
Τα Διαγράμματα Ακολουθίας (ΔΑ) (*Sequence Diagrams*) ανήκουν στην κατηγορία των Διαγραμμάτων Αλληλεπίδρασης (*Interaction Diagrams*) της UML και χρησιμοποιούνται για να αποτυπώσουν τις αλληλεπιδράσεις μεταξύ αντικειμένων / οντοτήτων αναδεικνύοντας τη

σειρά με την οποία οι αλληλεπιδράσεις αυτές λαμβάνουν χώρα. Με τον όρο αλληλεπίδραση εννοείται η επικοινωνία μεταξύ των στοιχείων ενός συστήματος μέσω ανταλλαγής μηνυμάτων. Μηνύματα είναι είτε κλήσεις μεθόδων (*method invocations*) είτε διεργασίες που επηρεάζουν ένα στοιχείο. Τέλος, στα Διαγράμματα Ακολουθίας η αύξηση του χρόνου ορίζεται από πάνω προς τα κάτω και τα μηνύματα που αποτυπώνονται μπορούν να είναι μερικώς διατεταγμένα (*partially ordered*).

Ένα από τα βασικά προβλήματα που εμφανίζονταν στις εκδόσεις 1.x της UML ήταν η δυσκολία και ενίοτε η αδυναμία μοντελοποίησης της λογικής ροής στα Διαγράμματα Ακολουθίας. Οι αλλαγές όμως και οι προσθήκες που εισήχθησαν στην περιγραφή των Διαγραμμάτων Ακολουθίας στην έκδοση 2.0 της UML [UML] επέφεραν σημαντική βελτίωση στη δυνατότητα μοντελοποίησης της λογικής σε ακολουθίες και παράλληλα ο ορισμός πλαισίων αναφοράς *ref* έδωσε τη δυνατότητα για την ιεραρχική δόμηση και την επαναχρησιμοποίηση των διαγραμμάτων.

Η αλληλεπίδραση είναι μια έννοια που μπορεί να γίνει αντιληπτή σε διάφορα επίπεδα αφαίρεσης. Για παράδειγμα οι σχεδιαστές των συστημάτων λογισμικού συνήθως παράγουν αλληλεπιδράσεις σε επίπεδο ίχνους εκτέλεσης (*trace*). Αντίθετα, οι αναλυτές επιχειρησιακών διαδικασιών μπορεί να χρησιμοποιήσουν αλληλεπιδράσεις για να περιγράψουν συστήματα και χρήστες, με τα μηνύματα που ανταλλάσσονται να μοντελοποιούν λειτουργίες σε υψηλό επίπεδο. Αυτή η δυνατότητα ευρείας χρήσης των αλληλεπιδράσεων όπως και η δυνατότητα μοντελοποίησης κάποιας συγκεκριμένης λογικής ροής που διαθέτουν τα Διαγράμματα Ακολουθίας καθιστά τον τύπο αυτό διαγραμμάτων κατάλληλο για την έκφραση σεναρίων μερικής διάταξης διαφόρων τύπων, αφαιρετικών επιπέδων και πεδίων εφαρμογής.

Ο Πίνακας 4.1 περιλαμβάνει ορισμένα βασικά γραφικά στοιχεία ενός Διαγράμματος Ακολουθίας, τα οποία θα αποτελέσουν αντικείμενο του μετασχηματισμού. Παρόλα αυτά η περιγραφή του μετασχηματισμού στηρίζεται και σε στοιχεία (κλάσεις) της Superstructure προδιαγραφής της UML των οποίων η γραφική απεικόνιση ενδεχομένως να μην είναι τόσο ξεκάθαρη όσο των παραπάνω. Τα στοιχεία αυτά παρουσιάζονται στη συνέχεια χωρίς όμως να αναλύονται διεξοδικά. Η διεξοδική περιγραφή τους, όπως και στοιχείων του μεταμοντέλου που δε λαμβάνονται υπόψη στο μετασχηματισμό, ξεφεύγει από τα πλαίσια του παρόντος τόμου και μπορεί να αναζητηθεί στο [UML].

Σύμβολο	Τίτλος	Περιγραφή
	αντικείμενο : κλάση / οντότητα	Συμβολίζει το στοιχείο που συμμετέχει στην αλληλεπίδραση.
	γραμμή ζωής (lifeline)	Συμβολίζει τον άξονα του χρόνου ζωής κάθε αντικειμένου
	μήνυμα (σύγχρονο / ασύγχρονο)	Υλοποιεί την επικοινωνία
	συνδυασμένο τμήμα	Μέσο περιγραφής ενός αριθμού ιχνών με ένα συμπαγή και συνοπτικό τρόπο. Χαρακτηρίζεται από το χειριστή και τους τελεστέους αλληλεπίδρασης.

Πίνακας 4.1: Ορισμένα βασικά σύμβολα των Διαγραμμάτων Ακολουθίας

Μια σημαντική κλάση του μεταμοντέλου της UML είναι η *InteractionFragment* (τμήμα αλληλεπίδρασης) η οποία είναι αφηρημένη έννοια της πιο γενικής μονάδας αλληλεπίδρασης.

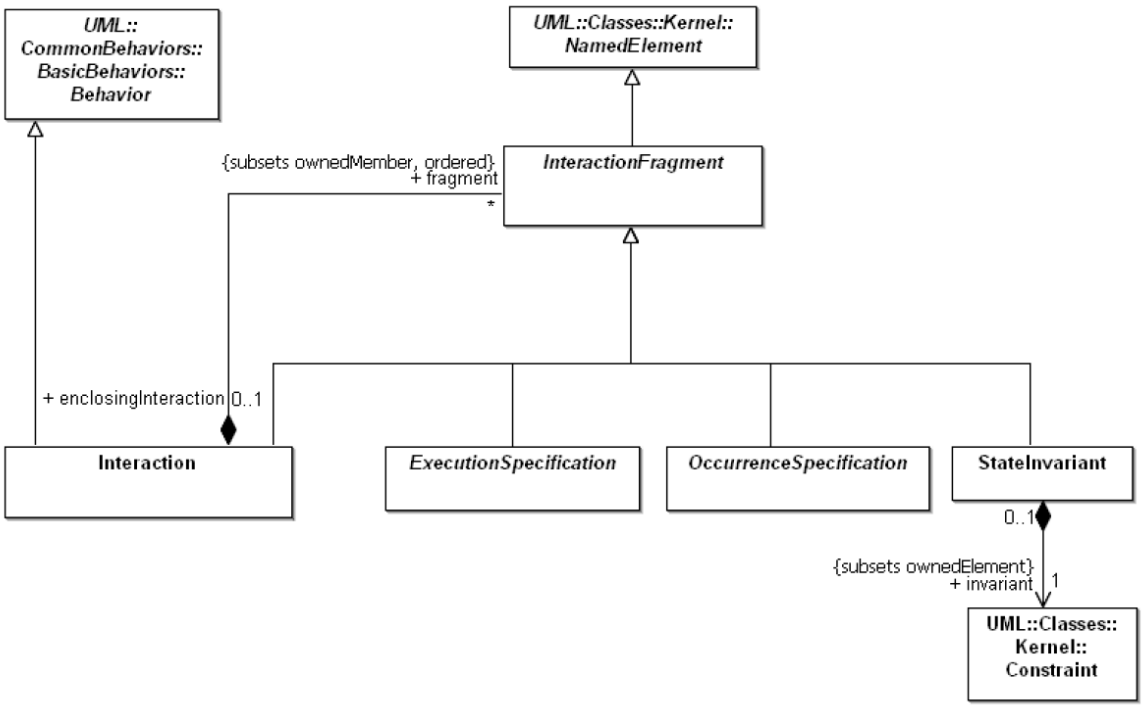
Μια από τις εξειδικεύσεις της κλάσης αυτής είναι η *ExecutionSpecification*. Μια κλάση ExecutionSpecification αποτελεί την προδιαγραφή της εκτέλεσης μιας μονάδας συμπεριφοράς ή δράσης σε μια γραμμή ζωής.

Όταν σε ένα Διάγραμμα Ακολουθίας μοντελοποιείται η μετάδοση ενός μηνύματος, εκτός από μια περίσταση της κλάσης *Message*, υπάρχουν δύο περιστάσεις της κλάσης *MessageOccurrenceSpecification* που συμβολίζουν τα «άκρα» ενός μηνύματος, δηλαδή την πρόκληση και την αποδοχή της κλήσης (και σχετίζονται με την κλάση Message). Αυτό ισχύει τόσο για σύγχρονες κλήσεις λειτουργιών, όσο και για ασύγχρονα σήματα. Οι εμφανίσεις αυτές «παραλαμβάνονται» από μια σχετική υπόσταση εκτέλεσης (εξειδίκευση της ExecutionSpecification).

Η εκτέλεση μιας συμπεριφοράς αναπαριστάται με την κλάση *BehaviorExecutionSpecification* που είναι ένα είδος ExecutionSpecification. Πρακτικά, για κάποια συμπεριφορά, έχουμε μια υπόσταση της BehaviorExecutionSpecification για κάθε μετάδοση μηνύματος. Η

παρατήρηση αυτή είναι βασική για την υλοποίηση των κανόνων αντιστοίχισης που προτείνονται.

Στο Σχήμα 4.2 παρατίθεται το διάγραμμα κλάσεων για τις αλληλεπιδράσεις όπως υπάρχει στο [UML].

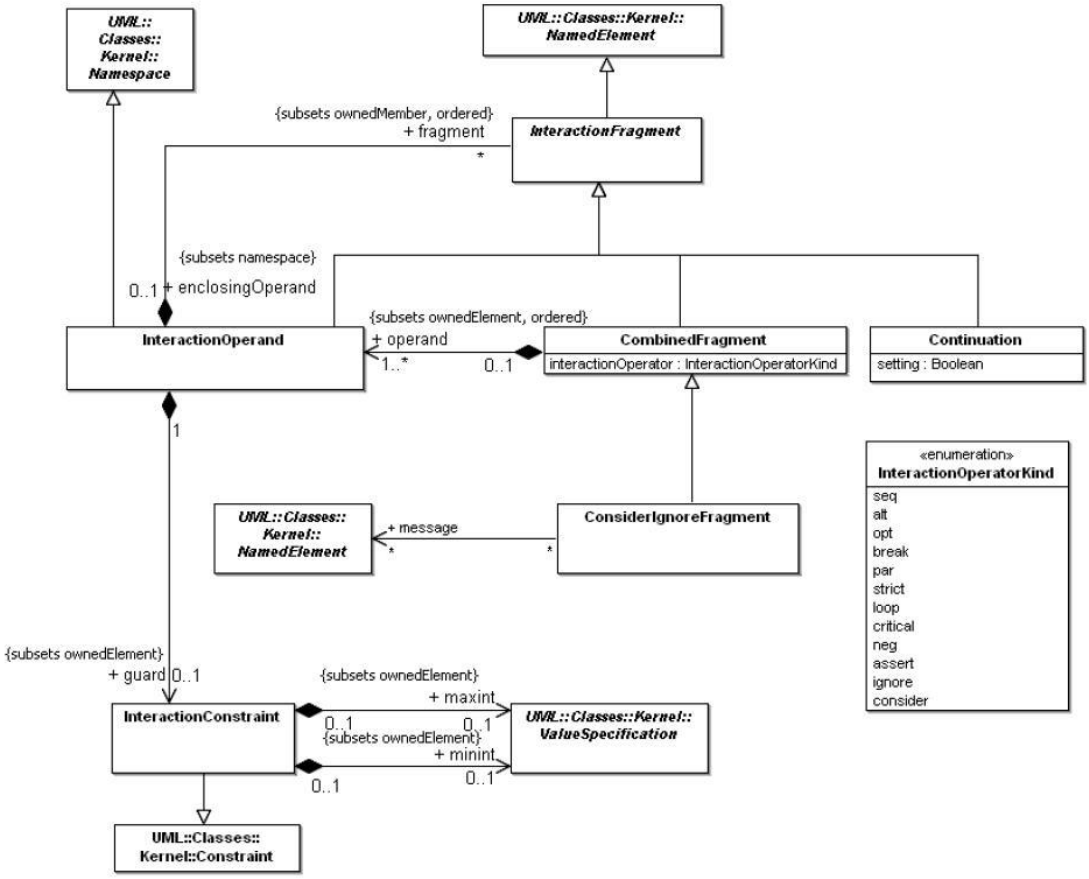


Σχήμα 4.2: Αλληλεπιδράσεις (Copyright © OMG, Inc)

Η μοντελοποίηση της λογικής ροής αλλά και άλλων χαρακτηριστικών γίνεται μέσω των *συνδυασμένων τμημάτων*. Υπάρχει για το λόγο αυτό μια κλάση, η *CombinedFragment*, η οποία αποτελεί επίσης εξειδίκευση της αφηρημένης InteractionFragment και ορίζει την έκφραση τμημάτων αλληλεπίδρασης μέσω των οποίων είναι δυνατή η αποτύπωση ενός αριθμού ιχνών (ακολουθιών γεγονότων) με συμπαγή και συνοπτικό τρόπο. Ένα συνδυασμένο τμήμα ορίζεται από ένα *χειριστή αλληλεπίδρασης* (*interaction operator*) και έναν ή περισσότερους *τελεστέους αλληλεπίδρασης* (*interaction operands*). Ο χειριστής αλληλεπίδρασης είναι ένα ιδιοχαρακτηριστικό (*attribute*) της κλάσης CombinedFragment η τιμή του οποίου προσδιορίζει τη σημασιολογία του συνδυασμένου τμήματος και μπορεί να λάβει δώδεκα τιμές -οι τιμές αυτές και η σημασιολογία που αποδίδουν περιγράφονται στη συνέχεια της ενότητας. Επίσης, κάθε CombinedFragment περιλαμβάνει ένα σύνολο κλάσεων *InteractionOperand* που αποτελούν τους τελεστέους αλληλεπίδρασης του συνδυασμένου τμήματος. Ένας τελεστέος αλληλεπίδρασης είναι κι αυτός ένα τμήμα αλληλεπίδρασης συνοδευόμενο (προαιρετικά) από μια *έκφραση φύλαξης* (*guard expression*). Η έκφραση αυτή αναπαριστάται από μια κλάση *InteractionConstraint* που είναι μια έκφραση Μπούλ,

εξειδίκευση της κλάσης «πυρήνα» του μεταμοντέλου της UML, *Constraint*. Οι εκφράσεις φύλαξης μπορούν να διατυπώνονται σε διάφορες γλώσσες. Στην υλοποίηση του πλαισίου θεωρείται ότι οι εκφράσεις αυτές είναι διατυπωμένες είτε ως OCL περιορισμοί, είτε σε φυσική γλώσσα, είτε ακολουθώντας το συντακτικό της Java/C/C++ κ.ά. για τη διατύπωση εκφράσεων Μπούλ.

Ένας τελεστέος αλληλεπίδρασης μπορεί να περιλαμβάνει ένα ταξινομημένο σύνολο από τμήματα αλληλεπίδρασης. Η ταξινόμηση των τμημάτων γίνεται βάση της γεωμετρικής θέσης που καταλαμβάνουν στον κάθετο άξονα. Το διάγραμμα κλάσεων για τα συνδυασμένα τμήματα, όπως περιλαμβάνεται στο [UML] παρατίθεται στο Σχήμα 4.3.



Σχήμα 4.3: Συνδυασμένα Τμήματα (Copyright © OMG, Inc)

Ο ρυθμιστικός παράγοντας για τη σημασιολογία ενός συνδυασμένου τμήματος είναι η τιμή του χειριστή αλληλεπίδρασης. Στη συνέχεια παρατίθενται οι δώδεκα δυνατές τιμές με μια σύντομη περιγραφή της σημασιολογίας του προσδίδουν στα συνδυασμένα τμήματα. Η περιγραφή βασίζεται στον ορισμό της σημασιολογίας του κάθε τμήματος όπως διατυπώνεται στο [UML].

alt: δηλώνει ότι το συνδυασμένο τμήμα αναπαριστά μια επιλογή συμπεριφοράς μεταξύ εναλλακτικών. Το πολύ ένας τελεστέος επιλέγεται προς εκτέλεση, του οποίου η έκφραση φύλακα πρέπει να επαληθεύεται. Αν η έκφραση αυτή είναι else τότε η συνθήκη ταυτίζεται με την άρνηση της διάζευξης όλων των εκφράσεων φύλαξης που ανήκουν στους υπόλοιπους τελεστέους του CombinedFragment.

opt: δηλώνει ότι το συνδυασμένο τμήμα αναπαριστά μια επιλογή συμπεριφοράς στην οποία είτε ο μοναδικός τελεστέος συμβαίνει, είτε δε συμβαίνει τίποτα. Μεταξύ του opt και του alt μπορεί να οριστεί μια σημασιολογική ισοδυναμία: το opt είναι ένα alt στο οποίο υπάρχει ένα τελεστέος με το ίδιο περιεχόμενο και έκφραση φύλαξης με τα αντίστοιχα του opt κι ένας δεύτερος χωρίς περιεχόμενο.

par: δηλώνει ότι το συνδυασμένο τμήμα αποτελεί μια έκφραση συγχώνευσης παράλληλων συμπεριφορών που περιλαμβάνονται στους τελεστέους του. Η εμφανίσεις των συμβάντων μπορούν να διαπλέκονται με οποιαδήποτε τρόπο αρκεί να διατηρείται η διάταξη ανά τελεστέο.

loop: δηλώνει ότι το συνδυασμένο τμήμα αναπαριστά ένα βρόχο. Περιλαμβάνει ένα τελεστέο και μια έκφραση φύλαξης. Η έκφραση αυτή, ειδικά για συνδυασμένα τμήματα με χειριστή loop μπορεί να περιλαμβάνει πέρα από την έκφραση Μπουλ, ένα κατώτατο (*minint*) κι ένα ανώτατο (*maxint*) όριο επαναλήψεων.

seq: δηλώνει ότι το συνδυασμένο τμήμα αναπαριστά μια ασθενή διάταξη μεταξύ των συμπεριφορών των τελεστέων. Συγκεκριμένα ισχύουν τρεις κανόνες για τα ίχνη: οι εμφανίσεις σε κάθε τελεστέό διατηρούν τη σειρά που προσδιορίζει ο κατακόρυφος άξονας, οι εμφανίσεις που αναφέρονται σε διαφορετικές γραμμές ζωής μπορεί να έχουν οποιαδήποτε σειρά και οι εμφανίσεις που αναφέρονται στην ίδια γραμμή και βρίσκονται σε διαφορετικούς τελεστέους πρέπει να διατηρούν τη διάταξη που ορίζει η γεωμετρική σειρά των τελεστέων.

critical: δηλώνει ότι το συνδυασμένο τμήμα αναπαριστά μια κρίσιμη περιοχή, δηλαδή μια περιοχή της οποίας τα ίχνη δε μπορούν να διαπλέκονται. Αν για παράδειγμα σε κάποιο τελεστέο ενός par οριστεί μια κρίσιμη περιοχή τότε αυτή πρέπει να εκτελεστεί ατομικά, χωρίς δηλαδή την παρεμβολή άλλων γεγονότων.

break: δηλώνει ότι το συνδυασμένο τμήμα αναπαριστά ένα σενάριο διακοπής. Περιλαμβάνει ένα μοναδικό τελεστέο αλληλεπίδρασης ο οποίος εκτελείται όταν επαληθεύεται η έκφραση φύλακα που περιλαμβάνει και αντικαθιστά τα τμήματα αλληλεπίδρασης που υπολείπονται μέσα στο InteractionFragment που περιλαμβάνει και το break. Μπορεί και πάλι να οριστεί μια σημασιολογική ισοδυναμία μεταξύ του break και του alt: ένα break είναι ένα alt με έναν τελεστέο που φέρει το φύλακα και το περιεχόμενο του break κι ένα δεύτερο τελεστέο με έκφραση φύλακα else και το περιεχόμενο του υπολοίπου του InteractionFragment.

strict: δηλώνει ότι το συνδυασμένο τμήμα αναπαριστά μια αυστηρή διάταξη μεταξύ των συμπεριφορών των τελεστέων. Η εμβέλεια αυτής της διάταξης των τελεστέων περιορίζεται σε πρώτο επίπεδο. Αυτό σημαίνει ότι εμφανίσεις γεγονότων σε κάποιο συνδυασμένο τμήμα που εσωκλείνεται δε συγκρίνονται απευθείας με τις εμφανίσεις των γεγονότων του (εξωτερικού) συνδυασμένου τμήματος.

neg: δηλώνει ότι το συνδυασμένο τμήμα αναπαριστά ίχνη τα οποία δεν είναι έγκυρα και δε πρέπει να εμφανίζονται.

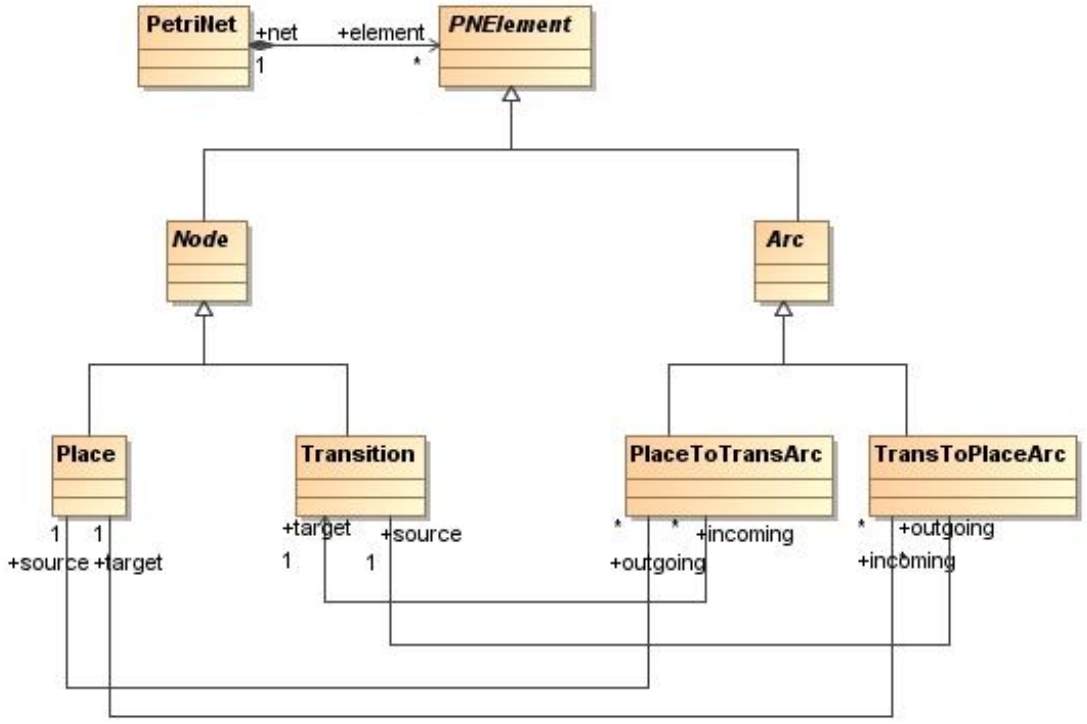
ignore / **consider**: δηλώνει ότι στο συνδυασμένο τμήμα υπάρχουν κάποιοι τύποι μηνυμάτων που δεν πρέπει να λαμβάνονται υπόψη (ignore) ή αντίστροφα, τύποι μηνυμάτων που πρέπει να λαμβάνονται υπόψη (consider). Τα συνδυασμένα τμήματα αυτά δηλώνονται από την κλάση *ConsiderIgnoreFragment* η οποία είναι εξειδίκευση της CombinedFragment και στη δήλωσή τους περιλαμβάνεται ένα σύνολο μηνυμάτων για τα οποία ισχύει.

assertion: δηλώνει ότι το συνδυασμένο τμήμα αναπαριστά μια διαβεβαίωση. Οι ακολουθίες του τελεστέου ενός assertion είναι οι μόνες έγκυρες αλληλουχίες. Συνδυασμένα τμήματα με χειριστή assertion συνήθως χρησιμοποιούνται σε συνδυασμό με συνδυασμένα τμήματα με χειριστή ignore/consider.

Η περιγραφή που προηγήθηκε στην ενότητα αυτή αφορά μόνο σε ένα υποσύνολο των χαρακτηριστικών Διαγραμμάτων Ακολουθίας και συγκεκριμένα σε όσα είναι απαραίτητα για την παρουσίαση της γραμματικής του μετασχηματισμού που προτείνεται.

4.2 Μοντέλο Πεδίου Δικτύων Petri

Από τον ορισμό των συνήθων (απλών) Δικτύων Petri, που υπάρχει και στην ενότητα 3.2, μπορούμε να προσδιορίσουμε ένα μεταμοντέλο στο οποίο κάθε δίκτυο πρέπει να συμμορφώνεται. Ένα τέτοιο μεταμοντέλο σε επίπεδο κλάσεων και σχέσεων απεικονίζεται στο Σχήμα 4.4.

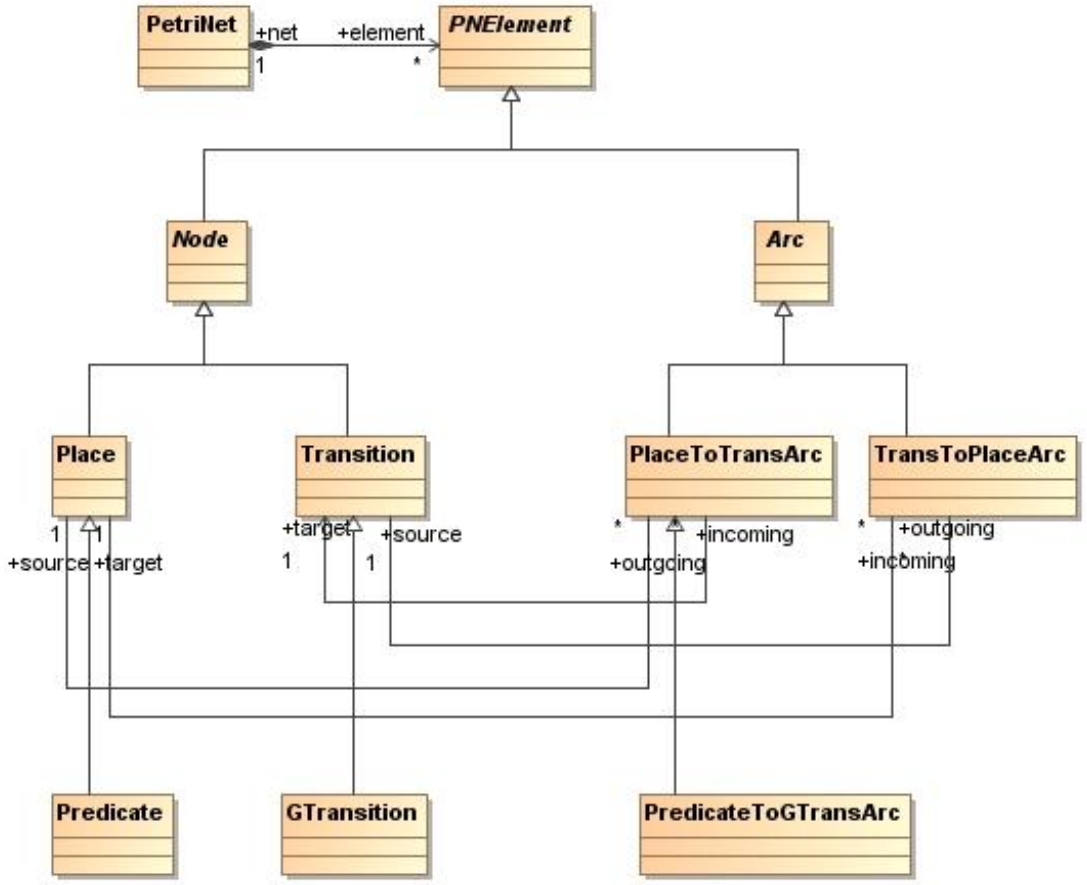


Σχήμα 4.4: Μεταμοντέλο συνήθων Δικτύων Petri

Το μεταμοντέλο στο Σχήματος 4.3 καλύπτει τον ορισμό οποιουδήποτε συνήθους Δικτύου Petri. Παρόλα αυτά, ένα από τα ζητούμενα του μετασχηματισμού είναι η εμφάνιση των εκφράσεων φύλαξης στους τελεστέους των διαφόρων συνδυασμένων τμημάτων στο προκύπτων μοντέλο έτσι ώστε να μπορεί να ελεγχθεί και η λογική ροή βάση, πέρα από τα γεγονότα (εμφανίσεις μηνυμάτων). Θεωρούμε ότι οι φύλακες μπορούν να εκφραστούν με μια επέκταση των απλών μεταβάσεων, τις «*φυλασσόμενες μεταβάσεις*» στις οποίες θα αποδίδεται ως συνθήκη δυνατότητας εκτέλεσης η έκφραση Μπουλ που φέρει ο φύλακας. Παράλληλα όμως, για τη θεωρητική πληρότητα του μοντέλου, απαιτούνται και ειδικές θέσεις που θα περιλαμβάνουν τις τιμές των μεταβλητών που περιλαμβάνονται στις λογικές εκφράσεις. Η ανάγκη αυτή οδηγεί στην επέκταση του βασικού μοντέλου και στην προσέγγιση του μοντέλου των Predicate/Transition Nets [Gen87] μέσω της προσθήκης ειδικού τύπου μεταβάσεων, θέσεων και διανυσμάτων. Τα Predicate/Transition Nets είναι Δίκτυα Petri υψηλού επιπέδου (επίπεδο 3) όπου οι θέσεις τους ονομάζονται *predicates* (*κατηγορήματα*) και

τα περιεχόμενά τους φέρουν πληροφορία -δηλαδή τα token μπορούν να διακριθούν μεταξύ τους.

Πιο συγκεκριμένα μπορεί να προστεθεί ένας τύπος μετάβασης, η φυλασσόμενη μετάβαση (*GTransition*) στην οποία αποδίδεται μια συνθήκη και ενεργοποιείται μόνο αν η συνθήκη αυτή ικανοποιείται. Μια τέτοια συνθήκη θα μπορούσε να περιλαμβάνει την έκφραση κάποιου φύλακα του Διαγράμματος Ακολουθίας αλλά και όχι μόνο. Επίσης, για την πληρότητα του μοντέλου θα πρέπει να προστεθεί ένας τύπος θέσεων (*Predicate*) ο οποίος φέρει τις τιμές των μεταβλητών που περιλαμβάνονται στις συνθήκες και ένας ειδικός τύπος διανυσμάτων *PlaceToTransArc*, τα διανύσματα *PredicateToGTransArc*. Ονομάζουμε το νέο μοντέλο που προκύπτει *PrTModel*, το οποίο σύμφωνα με τα παραπάνω έχει τη μορφή που φαίνεται στο Σχήμα 4.5.



Σχήμα 4.5: Μεταμοντέλο PrTModel – επέκταση συνήθων Δικτύων Petri

Πρέπει να σημειωθεί ότι το PrTModel που παρουσιάζεται στο Σχήμα 4.5 περιλαμβάνει ένα υποσύνολο μόνο των χαρακτηριστικών των Predicate/Transitions Nets.

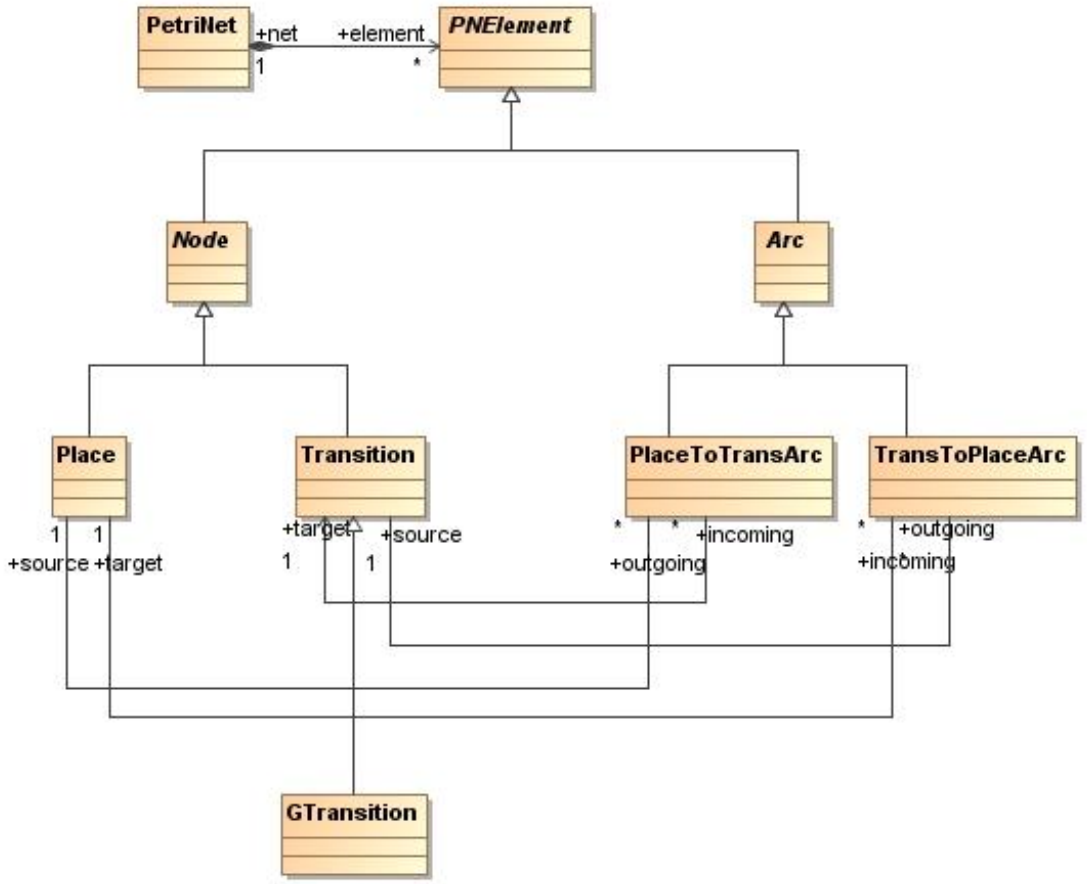
Ένα Δίκτυο Petri, όπως αναφέρεται στον ορισμό που δίνεται στην ενότητα 3.2, αποτελείται από ένα δίκτυο N κι από έναν αρχικό χαρακτηρισμό (initial marking) M_0. Αυτό σημαίνει ότι

ακόμα κι αν δομικά μπορεί να δημιουργηθεί ένα μοντέλο που να υλοποιεί τη ροή λογικής που εκφράζεται σε ένα Διάγραμμα Ακολουθίας, στο μοντέλο αυτό δε υπάρχει δυνατότητα να αποδοθεί περιεχόμενο στα predicates, καθώς οι τιμές των μεταβλητών που περιλαμβάνονται στις εκφράσεις φύλαξης συνήθως δεν είναι διαθέσιμες στο αρχικό μοντέλο. Συγκεκριμένα, οι τιμές αυτές κατά κανόνα γίνονται διαθέσιμες κατά τον χρόνο παρακολούθησης/καταγραφής (χρόνος εκτέλεσης).

Για την υπέρβαση του παραπάνω ζητήματος θα μπορούσαν να προταθούν διάφοροι διέξοδοι. Μια από αυτές θα ήταν η υλοποίηση του μετασχηματισμού να ακολουθεί το χρόνο παρακολούθησης/καταγραφής ώστε να λαμβάνει υπόψη τις τιμές των μεταβλητών όπως έχουν συλλεχθεί (και άρα το μοντέλο «πηγή» να ήταν το Διάγραμμα Ακολουθίας σε συνδυασμό με τις καταγραφές που αφορούν τιμές μεταβλητών). Η διέξοδος όμως αυτή δημιουργεί περιορισμούς ως προς την επιγραμμική ανάλυση. Στην περίπτωση της επιγραμμικής ανάλυσης τα predicate θα πρέπει να τροφοδοτούνται με τιμές κατά το χρόνο ανάλυσης και στη συνέχεια να εξετάζονται οι φυλασσόμενες μεταβάσεις. Υπό αυτές τις προϋποθέσεις το μοντέλο στόχος του μετασχηματισμού μπορεί να είναι ένα Pr/T net.

Μια δεύτερη ισοδύναμη εναλλακτική, η οποία είναι και αυτή που ακολουθείται, είναι το τελικό μοντέλου που ελέγχεται να είναι ο συνδυασμός του Δικτύου Petri που προκύπτει από το μετασχηματισμό και του συνόλου των δεδομένων παρακολούθησης (που αφορούν σε γεγονότα και σε τιμές μεταβλητών). Συγκεκριμένα, αρχικά γίνεται ο μετασχηματισμός των Διαγραμμάτων Ακολουθίας και η εξαγωγή της λογικής ροής μέσω της προσθήκης φυλασσόμενων μεταβάσεων. Ο μετασχηματισμός αφορά στα δομικά χαρακτηριστικά του μοντέλου και αποδίδει έναν γενικό αρχικό χαρακτηρισμό. Έπειτα, κατά το χρόνο ανάλυσης, δηλαδή εκτέλεσης του δικτύου σύμφωνα με τις ακολουθίες των καταγεγραμμένων γεγονότων, οι φυλασσόμενες μεταβάσεις εφόσον ενεργοποιηθούν σε επίπεδο δικτύου, λαμβάνουν ειδικό χειρισμό: αναζητούνται οι τιμές των μεταβλητών (είτε από τις καταγραφές, είτε από τη διαρρύθμιση του συστήματος, είτε μέσω άλλων οδών) κι έπειτα αποτιμώνται οι εκφράσεις. Η προσέγγιση αυτή αποτρέπει την ύπαρξη ειδικών θέσεων και διανυσμάτων και μπορεί να υλοποιηθεί ευκολότερα, με επέκταση οποιουδήποτε εργαλείου απλών Δικτύων Petri.

Σύμφωνα με τα παραπάνω, το δίκτυο που ορίζεται τελικά είναι ένα ειδικού τύπου Δίκτυο Petri, στο οποίο τα token που υπάρχουν στις θέσεις είναι τα συνήθη (*black tokens*) αλλά η δυνατότητα εκτέλεσης των φυλασσόμενων μεταβάσεων εξαρτάται και από εξωγενείς ως προς το δίκτυο παράγοντες -δηλαδή από τις τιμές που υπάρχουν στις καταγραφές ή μέσω άλλων μεθόδων αποτίμησης. Τον ειδικό αυτό χειρισμό, τον αναλαμβάνει το εργαλείο που χρησιμοποιείται για τον έλεγχο του τελικού (συνδυασμένου) μοντέλου. Το μεταμοντέλο του ειδικού αυτού τύπου Δικτύων Petri ονομάζεται *NetModel* και δίνεται στο Σχήμα 4.6.



Σχήμα 4.6: NetModel

Στο μεταμοντέλο του Σχήματος 4.6 συμμορφώνονται τα Δίκτυα Petri που προκύπτουν στο τέλος της διαδικασίας του μετασχηματισμού. Παρόλα αυτά, για λόγους εξυπηρέτησης του μετασχηματισμού -οι οποίοι εξηγούνται αναλυτικά στην επόμενη ενότητα- το NetModel επεκτείνεται με κλάσεις που αναπαριστούν συγκεκριμένες δομές αντιστοιχίας με τα Διαγράμματα Ακολουθίας. Σε κάθε περίπτωση όμως, τα δίκτυο που εισάγεται στη φάση της ανάλυσης αποτελείται από τα στοιχεία του Σχήματος 4.6 και μόνο.

4.3 Γραμματική μετασχηματισμού

Η γραμματική του μετασχηματισμού ενός Ακολουθιακού Διαγράμματος σε ένα Δίκτυο Petri που συμμορφώνεται στο NetModel περιλαμβάνει κατάλληλους κανόνες αντιστοίχισης των στοιχείων και των χαρακτηριστικών του μεταμοντέλου της UML σε στοιχεία και χαρακτηριστικά του NetModel.

Οι κανόνες που παρουσιάζονται στη συνέχεια μέσω βασικών παραδειγμάτων αντιστοίχησης αφορούν τα απλά μηνύματα και συνδυασμένα τμήματα με χειριστές αλληλεπίδρασης: *alt, opt, par, loop, seq, critical* και *break*.

Από τους παραπάνω χειριστές, στην υλοποίηση του μετασχηματισμού με ATL περιλαμβάνονται οι alt, opt, par και loop και πέρα από τη γραφική αντιστοιχία παρατίθεται και μια σύντομη περιγραφή για την κάθε περίπτωση. Η εκφραστικότητα που παρέχεται με αυτούς τους χειριστές αλληλεπίδρασης δίνει τη δυνατότητα να οριστούν σύνθετα σενάρια με εύκολο και συμπαγή τρόπο. Συχνά η εκφραστικότητα αντίστοιχων γλωσσών είναι σαφώς πιο περιορισμένη.

Μετά την παρουσίαση των κανόνων σχολιάζονται οι χειριστές για τους οποίους δε δίνεται κάποια περιγραφή μετασχηματισμού και το τμήμα *ref* που χρησιμοποιείται για να δηλώσει την αναφορά σε άλλο Διάγραμμα Ακολουθίας.

Η ενότητα κλείνει με μια αναφορά στις δυνατότητες χειρισμού του «*θορύβου*» στο επίπεδο του μετασχηματισμού. Ο όρος *θόρυβος* χρησιμοποιείται για να προσδιορίσει τις καταγραφές μηνυμάτων τα οποία δεν περιλαμβάνονται στα σενάρια.

Πριν την αναλυτική παρουσίαση των κανόνων είναι σκόπιμο να επισημανθούν ορισμένες γενικές αρχές και ζητούμενα που διέπουν τον προτεινόμενο μετασχηματισμό και σχετίζονται με επιθυμητά χαρακτηριστικά του παραγόμενου δικτύου για τη διευκόλυνση της ανάλυσης.

Για κάθε Διάγραμμα Ακολουθίας δημιουργείται μια θέση που περιέχει ένα token η οποία είναι η *αρχική θέση* του δικτύου. Κάθε Δίκτυο Petri που δημιουργείται έχει επίσης μια μοναδική *τελική θέση*, η οποία ταυτίζεται με τη θέση εξόδου που αντιστοιχεί στο τελευταίο τμήμα του διαγράμματος και περιγράφεται στη συνέχεια.

Στο Δίκτυο Petri που παράγεται δεν υπάρχουν διανύσματα με βάρος μεγαλύτερο της μονάδας.

Οι θέσεις μπορεί να φέρουν το πολύ ένα token. Δηλαδή το δίκτυο που παράγεται είναι *safe* ή *1-bounded* (βλ. 3.3).

Για κάθε τμήμα αλληλεπίδρασης που είναι μήνυμα ή συνδυασμένο τμήμα του Διαγράμματος Ακολουθίας, πρέπει να υπάρχει στο Δίκτυο Petri μια *θέση εισόδου* και μια *θέση εξόδου* που να αντιστοιχούν σε αυτό. Η θέση εισόδου είναι είτε η θέση εξόδου που αντιστοιχεί στο προηγούμενο τμήμα, είτε η εσωτερική αρχική θέση ενός τελεστέου αλληλεπίδρασης. Για το πρώτο τμήμα του διαγράμματος, θέση εισόδου είναι η αρχική θέση του δικτύου που ορίστηκε παραπάνω. Με το τρόπο αυτό εξασφαλίζεται η τήρηση της (μερικής) διάταξης των αλληλεπιδράσεων όπως εκφράζεται στα Διαγράμματα Ακολουθίας. Επιπλέον, ο ορισμός μιας μόνο τελικής και μιας μόνο αρχικής θέσης για κάθε τμήμα (όχι μόνο για κάθε γεγονός)

αποτρέπει τελικά την «έκρηξη» του πλήθους των θέσεων -όπως θα εμφανίζονταν για παράδειγμα σε αλληλουχίες συνδυασμένων τμημάτων με χειριστή par. Για να επιτευχθεί όμως η απλοποίηση αυτή απαιτείται η χρήση κατάλληλων ενδιάμεσων μεταβάσεων.

Στο σημείο αυτό είναι σκόπιμο να σημειωθεί ότι η επέκταση του βασικού μεταμοντέλου των Δικτύων Petri με την προσθήκη φυλασσόμενων μεταβάσεων, όπως περιγράφηκε στην προηγούμενη ενότητα, πέρα από τη έκφραση των φυλάκων μπορεί να χρησιμοποιηθεί για τη σύνθεση απλούστερων κανόνων μετασχηματισμού χρησιμοποιώντας μια εκφυλισμένη μορφή φυλασσόμενων μεταβάσεων με προκαθορισμένη αληθή συνθήκη (*True-By-Default Transitions – TBD Transitions*). Ο χειρισμός αυτών των μεταβάσεων ως προς την εκτέλεση θα περιγραφεί στο επόμενο κεφάλαιο.

Τέλος, για τη διευκόλυνση των περιγραφών ορίζουμε τους παρακάτω συμβολισμούς και συμβάσεις:

ΔΑ: Διάγραμμα Ακολουθίας

ΔΡ: Δίκτυο Petri

ΣΤ: Συνδυασμένο Τμήμα

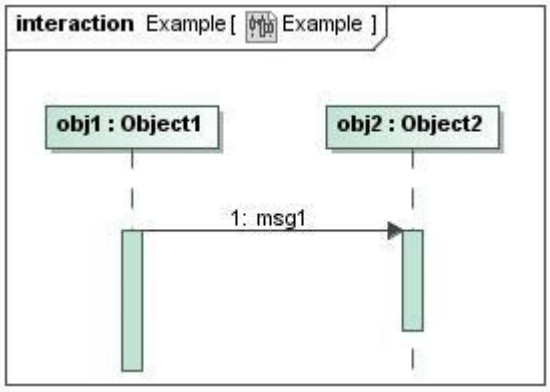
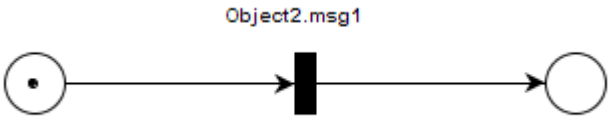
PtTarc: Διάνυσμα εισόδου -από θέση σε μετάβαση (Place to Transition arc)

TtParc: Διάνυσμα εξόδου -από μετάβαση σε θέση (Transition to Place arc)

Επίσης, ο όρος «γεγονός» χρησιμοποιείται για να εκφράσει την εμφάνιση και εκτέλεση ενός μηνύματος (αναφέρεται δηλαδή στην κλάση BehaviorExecutionSpecification του μεταμοντέλου της UML).

Μια σημαντική σύμβαση που ακολουθείται είναι ότι ο όρος «τμήμα» αναφέρεται σε τμήματα αλληλεπίδρασης (InteractionFragment) που είναι είτε γεγονότα και άρα εκτελέσεις συμπεριφορών (BehaviorExecutionSpecification), είτε συνδυασμένα τμήματα (CombinedFragment). Παρόλα αυτά, τμήματα αλληλεπίδρασης είναι τόσο οι τελεστέοι αλληλεπίδρασης, όσο και άλλα στοιχεία του μεταμοντέλου.

Κανόνας 1: Events / Γεγονότα

Διάγραμμα Ακολουθίας	
Δίκτυο Petri	

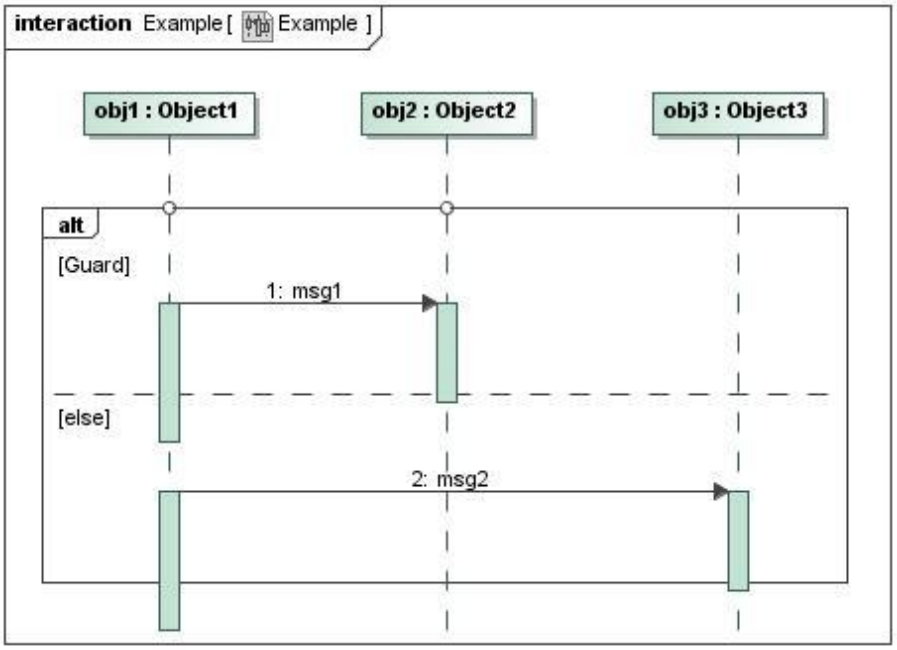
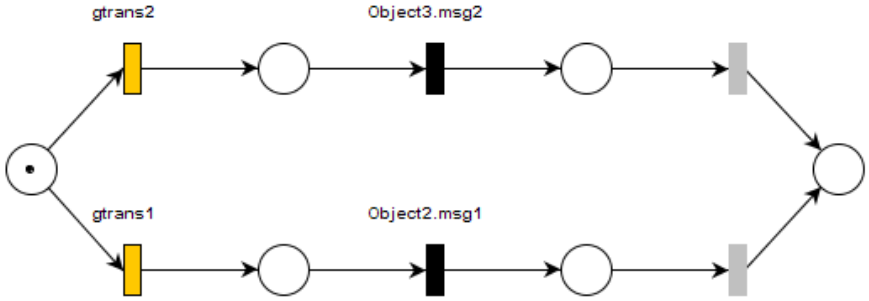
Περιγραφή: Κάθε γεγονός αντιστοιχείται σε μια μετάβαση και σε μια θέση εξόδου για τη μετάβαση αυτή. Επίσης, σε κάθε γεγονός αντιστοιχείται ένα PtTarc που ξεκινά από τη θέση εξόδου του προηγουμένου τμήματος (όταν είναι το πρώτο τμήμα στο ΔΑ, χρησιμοποιείται η αρχική θέση του δικτύου) ή την εσωτερική αρχική θέση ενός τελεστέου και καταλήγει στη μετάβαση, όπως κι ένα TtParc που ξεκινά από τη μετάβαση και καταλήγει στη θέση εξόδου.

Ένα από τα ζητούμενα του μετασχηματισμού είναι να χειρίζεται κατάλληλα διαφορετικές εμφανίσεις ίδιων μηνυμάτων. Έτσι κάθε μετάβαση αποδίδεται ένα μοναδικό αναγνωριστικό κι ένα όνομα. Το αναγνωριστικό παράγεται κατά τον μετασχηματισμό και απαιτείται για την ικανότητα διάκρισης μεταξύ των μεταβάσεων στο ΔΡ. Το όνομα έχει τη μορφή: `κλάση.μέθοδος`. Η μορφή αυτή ακολουθείται κατά την υλοποίηση αλλά μπορεί πολύ εύκολα να τροποποιηθεί για να καλύψει ενδεχομένως διαφορετικές ανάγκες.

Ένα τρίτο ιδιοχαρακτηριστικό της μετάβασης που ορίζεται είναι η ετικέτα η οποία παίρνει τιμή από το όνομα του αντικειμένου που κάνει την κλήση και το όνομα του αντικειμένου παραλαμβάνει την κλήση, στη μορφή `αποστολέας->παραλήπτης`. Η πληροφορία που περιλαμβάνει η ετικέτα είναι απαραίτητη για την ανάλυση των καταγραφών.

Σύμφωνα με τα παραπάνω, στο παράδειγμα που αναπαριστάται γραφικά θα είχαμε για τη μετάβαση, αναγνωριστικό: `EVT1` (δεν απεικονίζεται), όνομα: `Object2.msg1` και ετικέτα: `obj1->obj2`.

Κανόνας 2: Alternatives / Εναλλακτικές Ροές, *alt*

Διάγραμμα Ακολουθίας	
Δίκτυο Petri	

<u>Περιγραφή</u>: Τόσο η περιγραφή όσο και τα βήματα του μετασχηματισμού για τα ΣΤ μπορούν να χωριστούν σε δύο επίπεδα: επίπεδο ΣΤ, επίπεδο τελεστέου αλληλεπίδρασης. Για τα ΣΤ με χειριστή αλληλεπίδρασης alt έχουμε:

Επίπεδο ΣΤ: Η θέση εισόδου για το ΣΤ είναι η θέση εξόδου του προηγούμενου τμήματος ή η εσωτερική αρχική θέση αν είναι το πρώτο τμήμα σε τελεστή αλληλεπίδρασης. Επίσης για το ΣΤ, ορίζεται μια θέση εξόδου.

Επίπεδο τελεστέου: Σε κάθε τελεστέο αλληλεπίδρασης αντιστοιχείται μια φυλασσόμενη μετάβαση που φέρει τη συνθήκη του φύλακα. Επιπλέον αντιστοιχείται μια θέση η οποία παίζει ανάλογο ρόλο με την αρχική θέση του δικτύου αλλά για το εσωτερικό του τελεστέου (*εσωτερική αρχική θέση*) χωρίς βέβαια να φέρει token. Η θέση αυτή, στην περίπτωση του alt είναι επίσης και θέση εξόδου για τη φυλασσόμενη μετάβαση του τελεστέου.

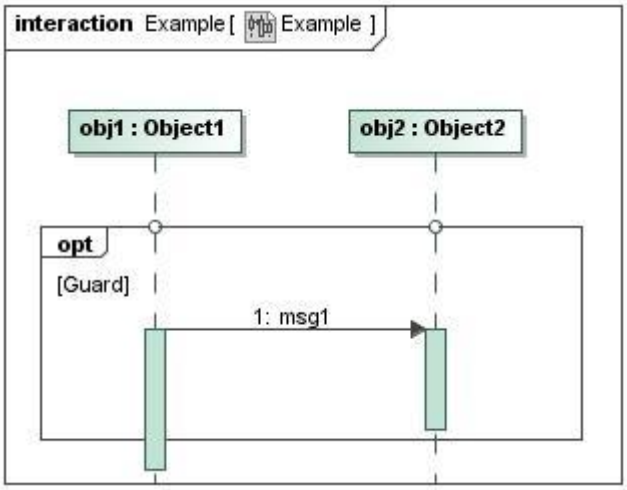
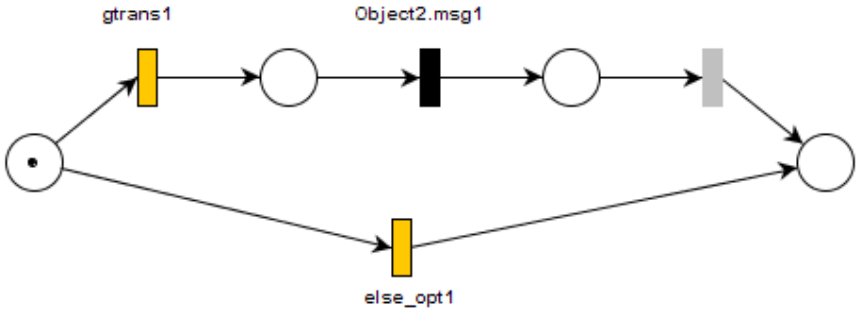
Επιπλέον, σε κάθε φυλασσόμενη μετάβαση που αναπαριστά τη συνθήκη του φύλακα καταλήγει ένα PtTarc το οποίο ξεκινά από την αρχική θέση του ΣΤ. Με αυτόν τον τρόπο

υλοποιείται η εναλλακτική επιλογή, δηλαδή μόνο μια από τις φυλασσόμενες μεταβάσεις μπορεί να εκτελεστεί.

Τέλος, στο επίπεδο τελεστέου, αφού μετασχηματισθεί το εσωτερικό του (μηνύματα ή φωλιασμένα ΣΤ) ορίζεται μια εξορισμού αληθής φυλασσόμενη μετάβαση (TBD Transition). Σε αυτή τη μετάβαση καταλήγει ένα PtTarc που ξεκινά από τη θέση εξόδου του τελευταίου τμήματος κάθε τελεστέου και επίσης, ξεκινά ένα TtParc το οποίο καταλήγει στη θέση εξόδου του ΣΤ. Με τη χρήση των TBD μεταβάσεων μειώνεται η πολυπλοκότητα των αντιστοιχίσεων καθώς δεν υπάρχει ειδικός χειρισμός για το τελευταίο τμήμα κάθε τελεστέου διότι διατηρείται η αρχή της μιας (ξεχωριστής) τελικής θέσης αν τμήμα.

Πρέπει να σημειωθεί ότι ο κανόνας που προτείνεται και υλοποιείται για τα συνδυασμένα τμήματα με χειριστή alt προϋποθέτει την ύπαρξη ενός τελεστέου με έκφραση φύλαξης else. Αυτό όμως δεν είναι απαραίτητο να συμβαίνει γενικά στα Διαγράμματα Ακολουθίας. Περισσότερες πληροφορίες για τον περιορισμό αυτό υπάρχουν στην αντίστοιχη ενότητα (βλ. 4.4).

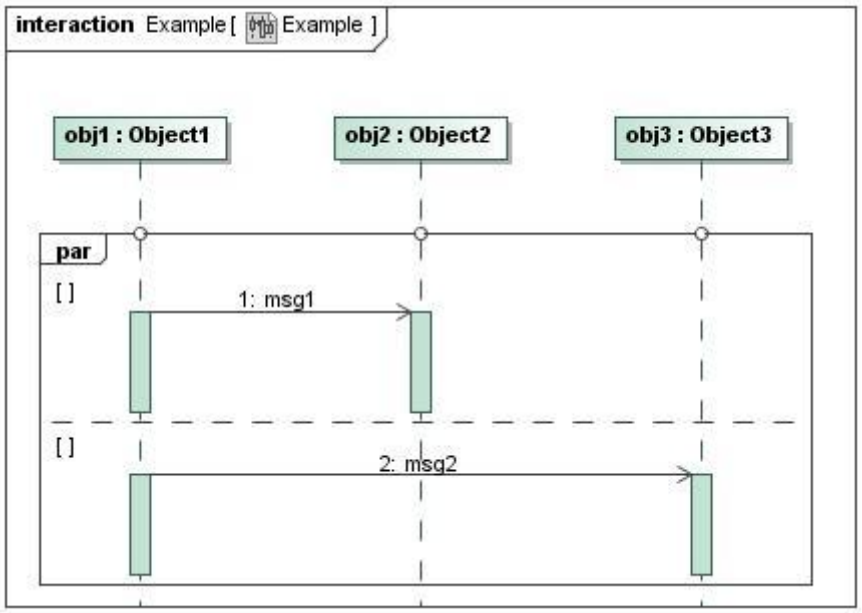
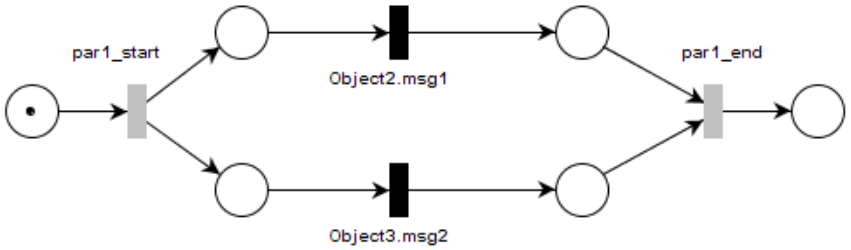
Κανόνας 3: Option / Προαιρετική Ροή, *opt*

Διάγραμμα Ακολουθίας	
Δίκτυο Petri	

<u>Περιγραφή</u>: Ο κανόνας για το opt ακολουθεί δομικά όσα ορίζονται για το ΣΤ alt με τη διαφορά ότι περιλαμβάνει ένα και μοναδικό τελεστέο.

Επιπλέον, ορίζεται στο επίπεδο ΣΤ μια δεύτερη φυλασσόμενη μετάβαση που αναπαριστά την άρνηση της συνθήκης του τελεστέου (else). Στη φυλασσόμενη αυτή μετάβαση καταλήγει ένα PtTarc από την θέση εισόδου του ΣΤ και ξεκινά ένα TtParc που καταλήγει στη θέση εξόδου του ΣΤ.

Κανόνας 4: Parallel / Παραλληλία, *par*

Διάγραμμα Ακολουθίας	
Δίκτυο Petri	

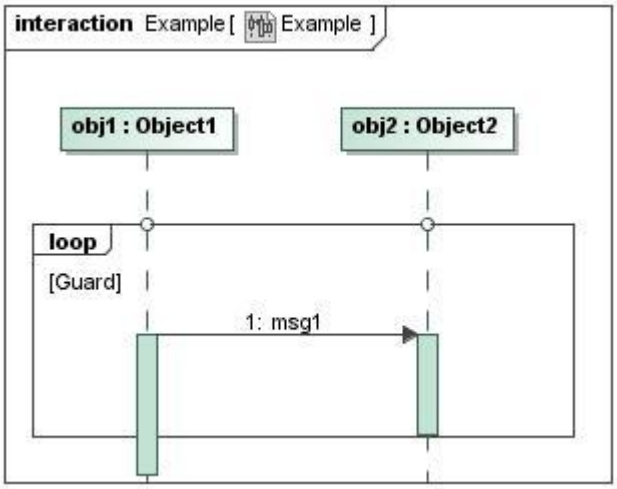
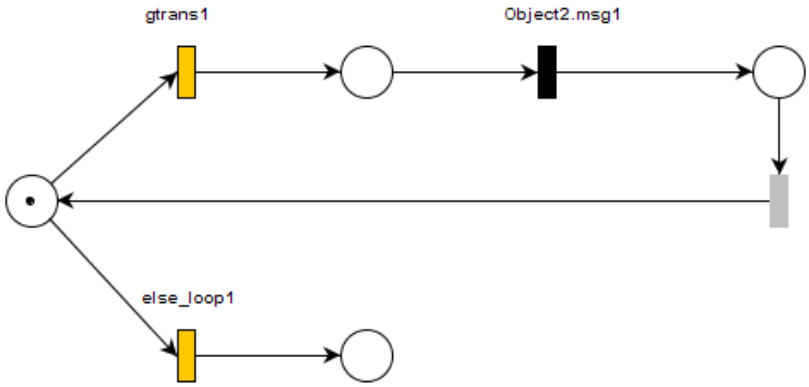
<u>Περιγραφή</u>: Η περιγραφή του ΣΤ par μπορεί να χωριστεί σε δύο επίπεδα, όπως παραπάνω.

Επίπεδο ΣΤ: Η θέση εισόδου του ΣΤ προσδιορίζεται όπως και στα προηγούμενα είδη ΣΤ και ορίζεται μια θέση εξόδου. Επίσης, στο επίπεδο ΣΤ ορίζεται μια TBD μετάβαση «*έναρξης*» που έχει ως θέση εισόδου, τη θέση εισόδου του ΣΤ και θέσεις εξόδου τις εσωτερικές αρχικές θέσεις που ορίζονται σε κάθε τελεστέο. Στο επίπεδο του ΣΤ, ορίζεται επίσης μια TBD μετάβαση «*λήξης*» που έχει ως θέσεις εισόδου τις θέσεις εξόδου των τελευταίων τμημάτων κάθε τελεστέου και θέση εξόδου τη τελική θέση του ΣΤ. Οι ορισμοί όμως των διανυσμάτων μεταξύ των παραπάνω μεταβάσεων και θέσεων ορίζονται σε επίπεδο τελεστέου.

Με τη δομή αυτή, όταν η εκτέλεση φτάσει στο ΣΤ par, τότε εκτελείται άμεσα η TBD μετάβαση έναρξης με αποτέλεσμα να ενεργοποιηθούν τα πρώτα τμήματα κάθε τελεστέου. Κατά αναλογία, η TBD μετάβαση λήξης ενεργοποιείται (και εκτελείται άμεσα) μόνο όταν εκτελεστούν τα τελικά τμήματα του κάθε τελεστέου.

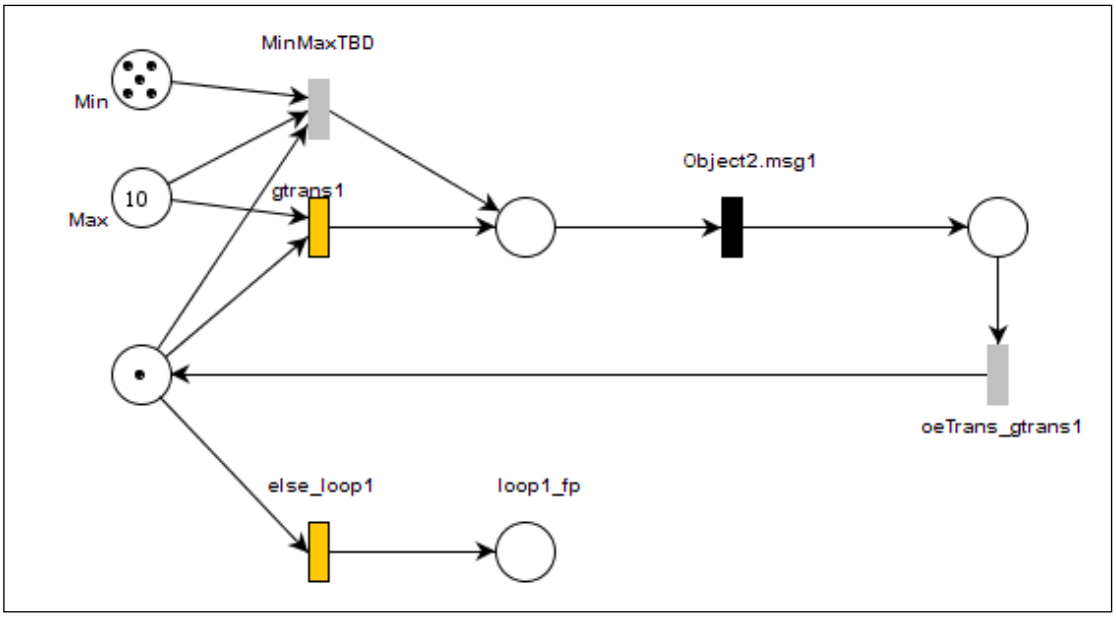
Επίπεδο τελεστέου: Όπως ήδη αναφέρθηκε, στο επίπεδο αυτό ορίζεται μια εσωτερική αρχική θέση (ανά τελεστέο). Επιπλέον, ορίζεται ένα TtParc από την μετάβαση έναρξης του ΣΤ προς την εσωτερική αρχική θέση κι ένα PtTarc από τη θέση εξόδου του τελευταίου τμήματος του τελεστή προς τη μετάβαση λήξης.

Κανόνας 5: Loop / Βρόχος, *loop*

Διάγραμμα Ακολουθίας	
Δίκτυο Petri	

<u>Περιγραφή</u>: Ο κανόνας για τον μετασχηματισμό ΣΤ loop εμφανίζει κοινά σημεία με εκείνο που ακολουθείται για ΣΤ opt. Η θεμελιώδης διαφορά είναι ότι το TtParc που ξεκινά από την TBD μετάβαση του τελεστέου καταλήγει στη θέση εισόδου του ΣΤ (επανάληψη).

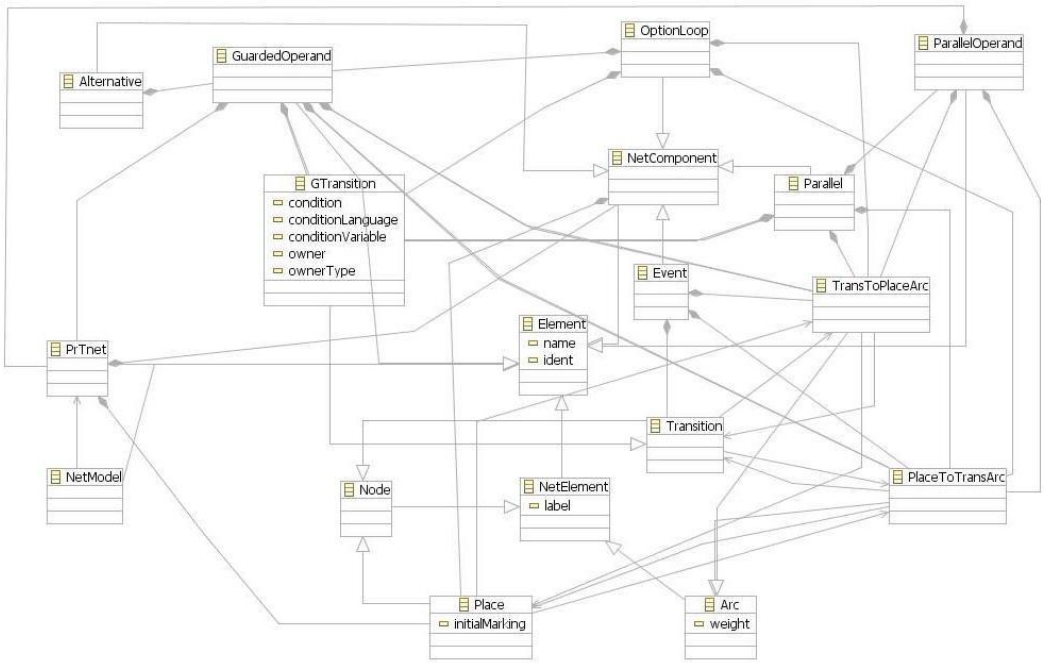
Πρέπει να σημειωθεί ότι σε ένα ΣΤ loop, στη συνθήκη του φύλακα μπορούν προαιρετικά να περιληφθούν ένας μέγιστος κι ένας ελάχιστος αριθμός επαναλήψεων. Αυτή η δυνατότητα, δε λαμβάνεται υπόψη στον κανόνα που περιγράφηκε παραπάνω. Παρόλα αυτά, για λόγους πληρότητας δίνεται στη συνέχεια ένας τρόπος έκφρασης των επιπλέον αυτών συνθηκών με την χρήση μιας TDB μετάβασης (MinMaxTBD) και θέσεων με tokens περισσότερα του ενός (το δίκτυο παύει να είναι safe/1-bounded).



ΔP: loop, με minint = 5 και maxint = 10

Στο παράδειγμα που απεικονίζει το παραπάνω ΔΡ, είναι minint = 5, maxint = 10. Η ύπαρξη της *MinMaxTBD* μετάβασης και των θέσεων *Min* και *Max* εξασφαλίζει ότι ο τελεστέος θα εκτελεστεί τουλάχιστον 5 φορές και όχι περισσότερες από 10. Για να είναι έγκυρη η αντιστοιχία αυτή, πρέπει να δοθεί προτεραιότητα εκτέλεσης των TBD μεταβάσεων έναντι των υπολοίπων φυλασσόμενων μεταβάσεων. Πρακτικά αυτό σημαίνει ότι για minint επαναλήψεις θα εκτελεστεί οπωσδήποτε η MinMaxTBD χωρίς να αποτιμηθεί η συνθήκη της *grtrans1* (και άρα και της *else_loop1*).

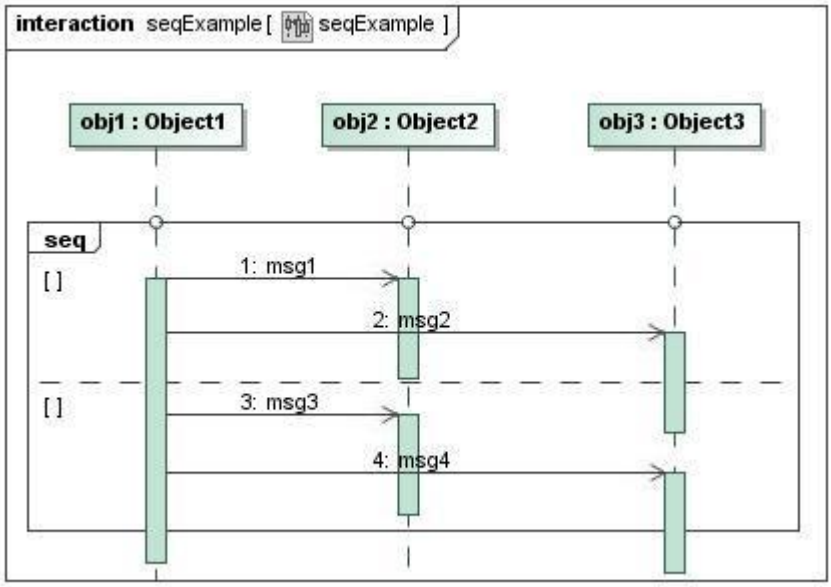
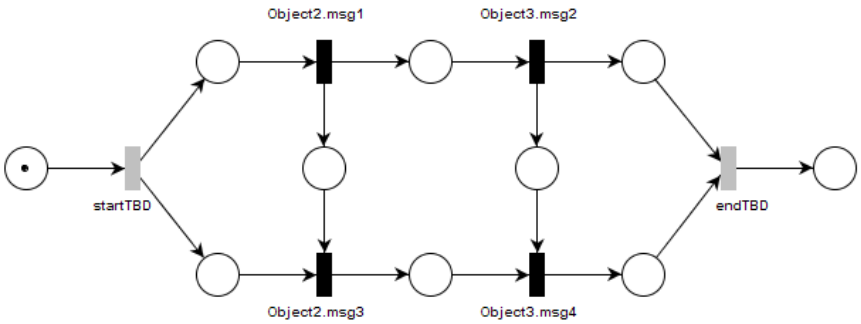
Όπως μπορεί να φανεί από τις περιγραφές των κανόνων για τα ΣΤ, χωρίζοντας τις αντιστοιχίσεις σε δύο επίπεδα (επίπεδο ΣΤ, επίπεδο τελεστέου) ο μετασχηματισμός γίνεται ευκολότερα αντιληπτός. Το ίδιο μπορεί να ειπωθεί και για την υλοποίηση των κανόνων. Για το λόγο αυτό, το NetModel που παρουσιάστηκε στην προηγούμενη ενότητα επεκτείνεται ώστε να συμπεριλάβει συνδυασμένα τμήματα ανάλογα με το χειριστή αλληλεπίδρασης που τα χαρακτηρίζει και τελεστέους αλληλεπίδρασης ανάλογα με το είδος του ΣΤ στο οποίο περιλαμβάνονται. Το τελικό μεταμοντέλο (*Extended NetModel*) των δικτύων που δημιουργούνται στη φάση αυτή του μετασχηματισμού δίνεται στο Σχήμα 4.7. Όπως αναφέρθηκε και προηγουμένως το δίκτυο που παράγεται τελικά περιλαμβάνει στοιχεία μόνο του αρχικού NetModel. Παρόλα αυτά, η χρήση ενός ενδιάμεσου μεταμοντέλου γίνεται για την σαφέστερη διατύπωση των κανόνων μετασχηματισμού σε ATL (ή και σε οποιαδήποτε άλλη γλώσσα μετασχηματισμού μοντέλων). Στη συνέχεια του κεφαλαίου και συγκεκριμένα στην ενότητα που περιγράφεται η διαδικασία του μετασχηματισμού παρατίθενται τόσο η τυπική περιγραφή (σε KM3 μορφή) του (Extended) NetModel όσο και οι υλοποιημένοι κανόνες σε ATL.



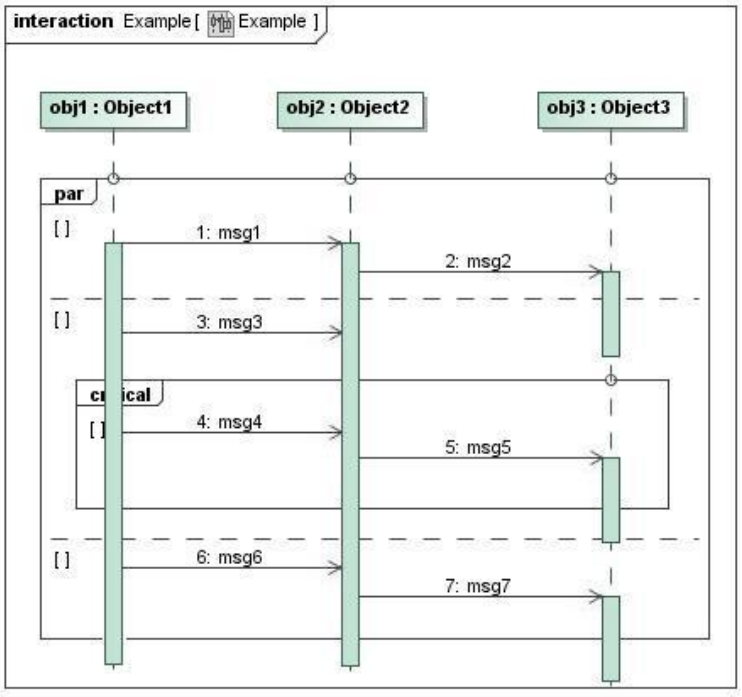
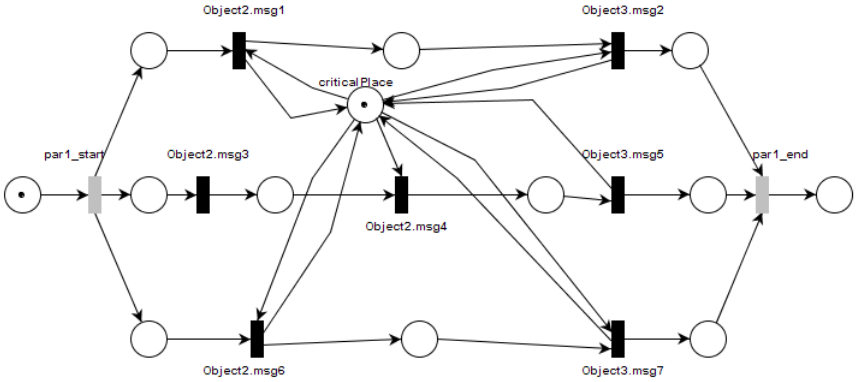
Σχήμα 4.7: Extended NetModel

Οι κανόνες που παρουσιάστηκαν παραπάνω αφορούν τους σημαντικότερους χειριστές αλληλεπίδρασης ως προς τη λογική ροή και τη διάταξη και έχουν υλοποιηθεί σε γλώσσα ATL. Παρόλα αυτά η εκφραστικότητα των Διαγραμμάτων Ακολουθίας δεν περιορίζεται σε αυτούς. Στη συνέχεια παρατίθενται γραφικά, μέσω απλών παραδειγμάτων, κανόνες μετασχηματισμού για ορισμένους ακόμη τύπους συνδυασμένων τμημάτων οι οποίοι θα μπορούσαν να αποτελέσουν τη βάση για τον τυπικό τους ορισμό σε κάποια γλώσσα μετασχηματισμών. Επιπλέον, σχολιάζονται οι χειριστές αλληλεπίδρασης για τους οποίους δε παρουσιάζεται κάποιος κανόνας.

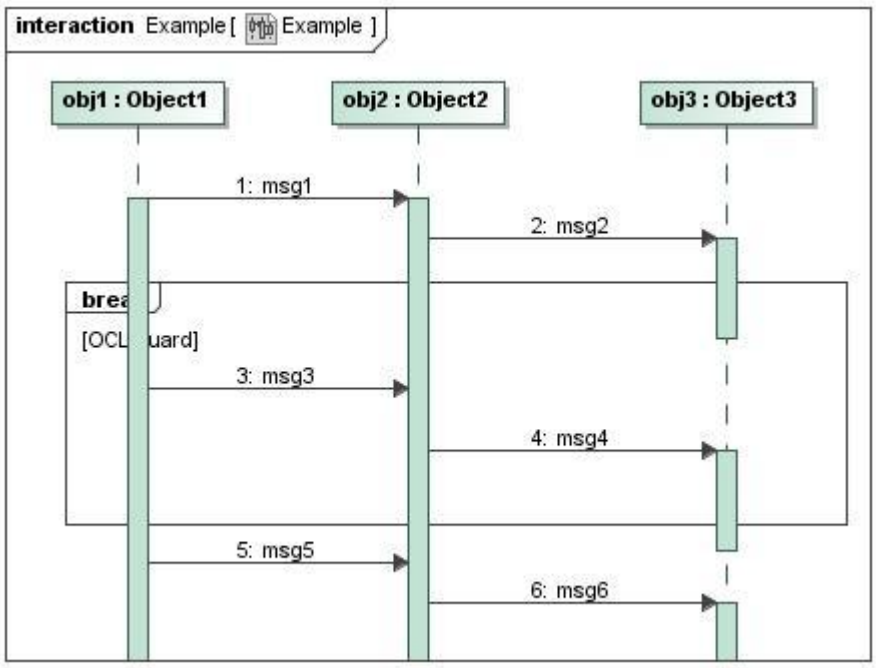
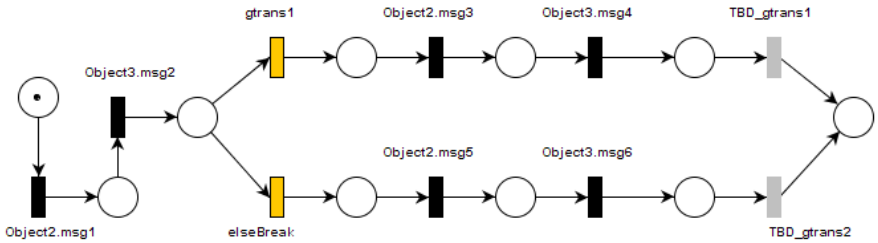
Κανόνας 6: Weak Sequencing / Ασθενής Διάταξη, *seq*

Διάγραμμα Ακολουθίας	
Δίκτυο Petri	

Κανόνας 7: Critical Region / Κρίσιμη Περιοχή, *critical*

Διάγραμμα Ακολουθίας	
Δίκτυο Petri	

Κανόνας 8: Break / Διακοπή, *break*

Διάγραμμα Ακολουθίας	
Δίκτυο Petri	

Ακολουθεί ο σχολιασμός των χειριστών αλληλεπίδρασης για τους οποίους δε παρουσιάστηκε κάποιος κανόνας μετασχηματισμού:

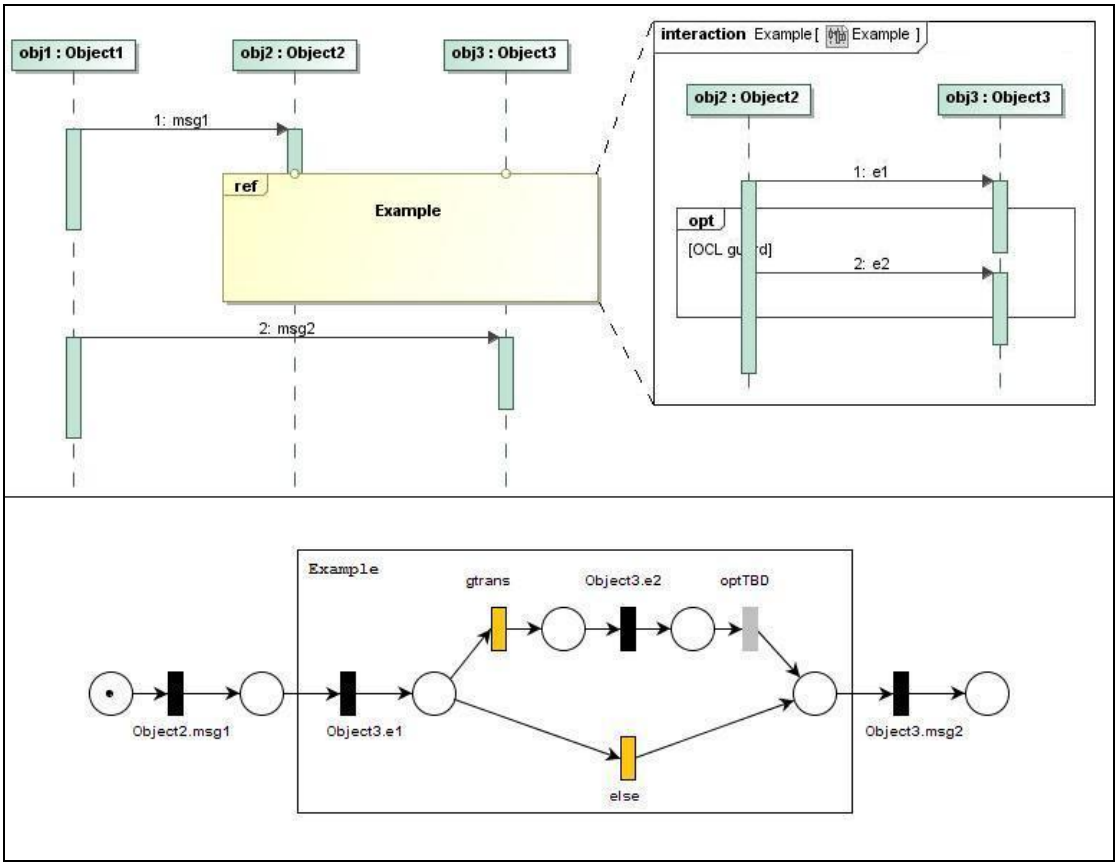
strict: η αυστηρή διάταξη όπως ορίζεται από το χειριστή strict δε παρουσιάζει ιδιαίτερο ενδιαφέρον ως προς την έκφραση σεναρίων του είδους που πραγματεύεται η εργασία.

neg: ο χειριστής αυτός μπορεί να χρησιμοποιηθεί για να «περικλείσει» τα σενάρια που δε θέλουμε να επαληθεύονται από τις καταγραφές. Η πληροφορία δηλαδή που φέρει μπορεί ενδεχομένως να χρησιμοποιηθεί σε υψηλότερο επίπεδο από αυτό του μετασχηματισμού.

ignore / **consider** και **assertion**: ένας μετασχηματισμός με συνδυασμένα τμήματα αυτού του είδους θα αγνοούσε / λάμβανε υπόψη ορισμένα γεγονότα κι αντίστοιχα δηλώσεις. Παρόλα αυτά το ενδιαφέρον που παρουσιάζουν για το μετασχηματισμό τους σε ΔΡ είναι περιορισμένο, εκτός από την περίπτωση που χρησιμοποιούνται για να ορίσουν τον θόρυβο. Το ζήτημα του θορύβου σχολιάζεται στη συνέχεια.

Ο χειριστής *ref* και το αντίστοιχο πλαίσιο παρουσιάζει ιδιαίτερο ενδιαφέρον καθώς μπορεί να χρησιμοποιηθεί για την έκφραση σεναρίων που συντίθενται από άλλα. Μέσω του ref δηλαδή είναι δυνατός ο ορισμός ιεραρχιών σεναρίων και η επαναχρησιμοποίηση διαγραμμάτων μέσα σε άλλα. Δομικά ο κανόνας μετασχηματισμού για ένα πλαίσιο ref θα μπορούσε να λάβει διάφορες μορφές. Συγκεκριμένα, ένα πλαίσιο ref θα μπορούσε να αντιστοιχηθεί σε μια φυλασσόμενη μετάβαση (μαζί βέβαια με μια θέση εξόδου και τα απαιτούμενα διανύσματα), η συνθήκη της οποίας είναι το αν το σενάριο στο οποίο αναφέρεται το πλαίσιο επαληθεύεται. Για να απαντήσει όμως κανείς στο αν η συνθήκη επαληθεύεται και άρα η συνθήκη μπορεί να εκτελεστεί, ή όχι, πρέπει να έχει στη διάθεσή του και να μπορεί να εξετάσει το Δίκτυο Petri που αντιστοιχεί στο Διάγραμμα Ακολουθίας της αναφοράς ως προς αντίστοιχες καταγραφές. Δηλαδή, η ιεραρχία των σεναρίων διατηρείται και στη φάση της ανάλυσης και σε κάθε Διάγραμμα Ακολουθίας αντιστοιχεί ένα (ατομικό) Δίκτυο Petri.

Εναλλακτικά, θα μπορούσε να οριστεί μια διαδικασία ενσωμάτωσης των δικτύων. Δηλαδή, αντί για τη φυλασσόμενη μετάβαση που προτάθηκε παραπάνω, θα μπορούσε να ενσωματωθεί το Δίκτυο Petri που αντιστοιχεί στο διάγραμμα αναφοράς ως *υποδίκτυο*. Συγκεκριμένα, η γραμματική του μετασχηματισμού παράγει δίκτυα που έχουν μια αρχική θέση και μια τελική. Για την ενσωμάτωση ενός υποδικτύου σε κάποιο σημείο ενός άλλου μπορούμε να ακολουθήσουμε διάφορες οδούς. Για παράδειγμα, μπορούμε να αφαιρέσουμε την αρχική θέση του υποδικτύου και να χρησιμοποιήσουμε ως αρχική θέση για αυτό την τελική θέση που αντιστοιχεί στο προηγούμενο τμήμα ή την εσωτερική αρχική θέση ενός τελεστέου αν το πλαίσιο ref περιλαμβάνεται και είναι το πρώτο σε αυτόν. Αν θέλουμε βέβαια μπορούμε να αποφύγουμε την παραπάνω αφαίρεση και να προσθέσουμε απλώς ένα PtTarc που να ξεκινά από αρχική θέση και να καταλήγει στις μεταβάσεις που αντιστοιχεί στο πρώτο τμήμα του διαγράμματος αναφοράς. Στο Σχήμα 4.8 παρουσιάζεται ένα απλό παράδειγμα διαχείρισης του πλαισίου ref μέσω της ενσωμάτωσης, χρησιμοποιώντας την πρώτη τεχνική.

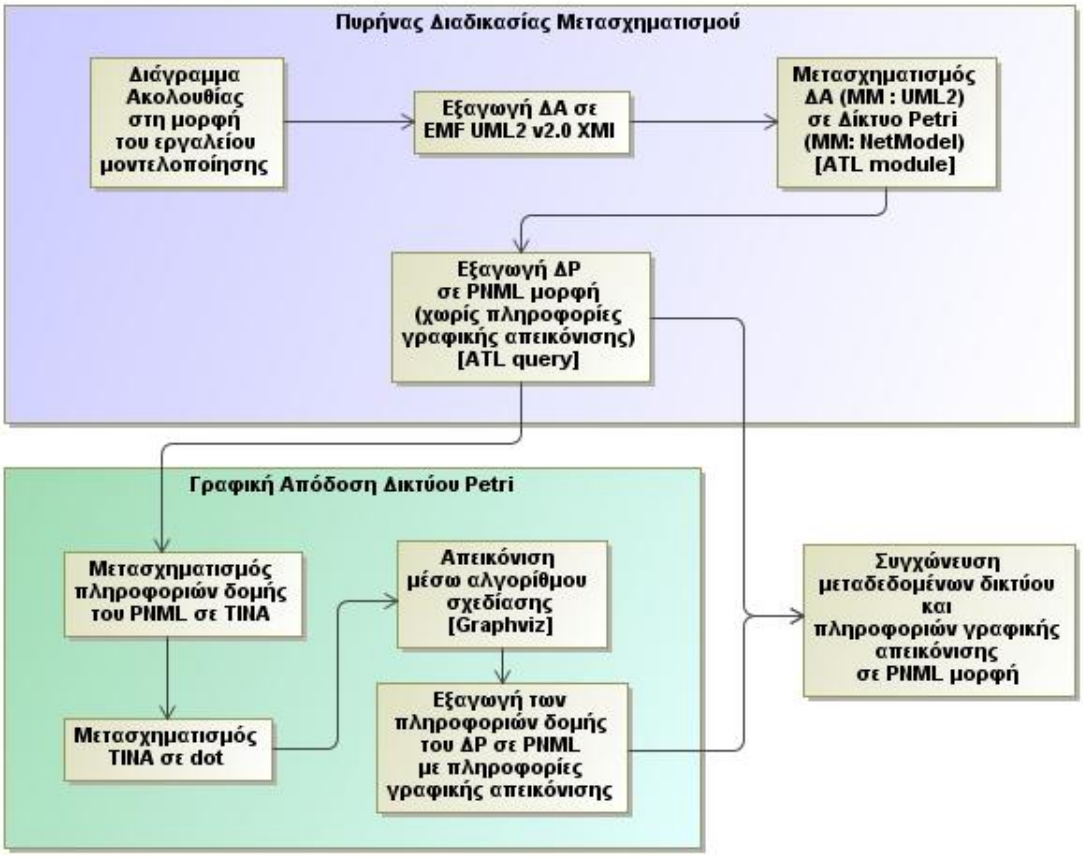


Σχήμα 4.8: Παράδειγμα ενσωμάτωσης υποδικτύου

Ο «θόρυβος», δηλαδή γεγονότα των καταγραφών που δεν ενδιαφέρουν επειδή δε περιλαμβάνονται στα σενάρια, μπορεί να είναι πολύ έντονος. Ο χειρισμός του θορύβου αυτού μπορεί να γίνει σε διαφορετικές φάσεις της διαδικασίας που παρουσιάζεται στην παρούσα εργασία. Για παράδειγμα θα μπορούσε να αντιμετωπιστεί μέσω φίλτρων κατά τη φάση της παρακολούθησης/καταγραφής, είτε μέσω ειδικού χειρισμού κατά τη φάση της ανάλυσης. Ένας τρίτος τρόπος χειρισμού του θορύβου είναι στο επίπεδο του μοντέλου, εφόσον είναι γνωστό το σύνολο των πιθανών γεγονότων. Πρακτικά, για κάθε πιθανό γεγονός που δεν περιλαμβάνεται στο σενάριο μπορεί να ορισθεί μια μετάβαση. Όλες οι μεταβάσεις αυτές συνδέονται αμφίδρομα με μια (κοινή) θέση που φέρει ένα token. Η δομή αυτή είναι ανεξάρτητη του υπολοίπου δικτύου και οι μεταβάσεις παραμένουν ενεργοποιημένες καθ' όλη τη διάρκεια της εκτέλεσης. Μοναδική εξαίρεση αποτελεί η ύπαρξη μιας ή περισσότερων κρίσιμων περιοχών. Στην περίπτωση αυτή δεν ορίζεται μια νέα θέση αλλά οι μεταβάσεις θορύβου συνδέονται αμφίδρομα με όλες τις θέσεις που έχουν ορισθεί για τις κρίσιμες περιοχές.

4.4 Διαδικασία μετασχηματισμού

Στην αρχή του κεφαλαίου παρουσιάστηκε μέσω του Σχήματος 4.1 μια επισκόπηση της διαδικασίας μετασχηματισμού. Στην ενότητα αυτή δίνεται μια αναλυτική περιγραφή των βημάτων που απαιτούνται έτσι ώστε ξεκινώντας από ένα Διάγραμμα Ακολουθίας που ορίζεται σε κάποιο εργαλείο UML να παραχθεί ένα αρχείο PNML που περιλαμβάνει το δίκτυο που παράγουμε βάσει των κανόνων της προηγούμενης ενότητας και που είναι αναγνώσιμο από εργαλεία επεξεργασίας, ανάλυσης και ελέγχου Δικτύων Petri. Στο Σχήμα 4.9 παρουσιάζεται αναλυτικά η διαδικασία του μετασχηματισμού.



Σχήμα 4.9: Διαδικασία παραγωγής Δικτύου Petri

Στη συνέχεια της ενότητας επεξηγούνται τα βήματα που απεικονίζονται στο Σχήμα 4.9. Επίσης, με τους όρους «γραφική απόδοση» και «απεικόνιση» εννοείται η απόδοση γραφικών χαρακτηριστικών σε επίπεδο δικτύου και όχι στοιχείων δικτύου. Πρέπει να σημειωθεί ότι η υποδιαδικασία αυτή δεν είναι απαραίτητη για την εξαγωγή ενός δικτύου που μπορεί να αναλυθεί αλλά χρησιμοποιείται για την απεικόνιση του μοντέλου σε αποδεκτή μορφή. Η

εύστοχη οπτικοποίηση επιτρέπει την καλύτερη κατανόηση και παρατήρηση των χαρακτηριστικών του δικτύου.

Πυρήνας Διαδικασίας Μετασχηματισμού

Τα βήματα που περιλαμβάνονται στο πυρήνα της διαδικασίας αρκούν ώστε να παραχθεί ένα Δίκτυο Petri κατάλληλο για ανάλυση από αντίστοιχα εργαλεία. Τα δίκτυο αυτό εκφράζεται σε PNML μορφή χωρίς όμως να φέρει πληροφορίες γραφικής απεικόνισης.

Διάγραμμα Ακολουθίας στη μορφή του εργαλείου μοντελοποίησης

Διαγράμματα Ακολουθίας μπορούν να δημιουργηθούν με μια πληθώρα εργαλείων, εμπορικών ή μη. Στο βήμα αυτό, δημιουργούνται τα διαγράμματα που αποτυπώνουν τα σενάρια ενδιαφέροντος και αποθηκεύονται στη μορφή που κάθε εργαλείο υποστηρίζει (συχνά και σε XMI). Στην υλοποίηση χρησιμοποιήθηκε το εργαλείο *MagicDraw UML Personal Edition 15.1*. Βασική παράμετρος στην επιλογή του εργαλείου έπαιξε η δυνατότητα εξαγωγής των μοντέλων που δημιουργούνται σε XMI ώστε να μπορούν αυτά να εισάγονται στο περιβάλλον του μετασχηματισμού. Επίσης, ένα ενδιαφέρον χαρακτηριστικό του παραπάνω εργαλείου είναι ότι δίνεται επιπλέον η δυνατότητα προσδιορισμού του προτύπου (*template*) σύμφωνα με το οποίο θα γίνει η εξαγωγή.

Εξαγωγή Διαγράμματος Ακολουθίας σε EMF UML2 v2.0 XMI μορφή που συμμορφώνεται στο UML2 Metamodel.

Αρκετά εργαλεία σχεδιασμού παρέχουν τη δυνατότητα εξαγωγής σε XMI. Η μορφή όμως που είναι πλήρως συμβατή με το περιβάλλον στο οποίο ορίζονται και υλοποιούνται οι κανόνες του μετασχηματισμού (Eclipse, EMF, ATL) είναι η EMF UML2 v2.0 XMI. Το UML2, όπως περιγράφηκε σε προηγούμενη ενότητα (βλ. 3.2) είναι μια υλοποίηση του μεταμοντέλου της UML 2.x βασισμένο στο EMF. Το MagicDraw UML PE 15.1, όπως και αρκετά άλλα εργαλεία, παρέχει επιλογή αυτόματης εξαγωγής σε αυτή τη μορφή. Εναλλακτικά, θα μπορούσε ενδεχομένως να οριστεί ένας μετασχηματισμός της ιδιωτικής μορφής αποθήκευσης του εργαλείου στο επιθυμητό σχήμα.

Μετασχηματισμός Διαγράμματος Ακολουθίας (Μεταμοντέλο: UML2) σε Δίκτυο Petri (Μεταμοντέλο: NetModel) – ATL module.

Στο βήμα αυτό γίνεται ουσιαστικά η μετάβαση από το ένα μοντέλο στο άλλο. Συγκεκριμένα, εισάγεται στο περιβάλλον (Eclipse, EMF, UML2, ATL) το Διάγραμμα Ακολουθίας και αναγνωρίζεται ως αποδεκτό UML μοντέλο. Ορίζεται επίσης το μεταμοντέλο NetModel σε

KM3 μορφή –η KM3 γλώσσα είναι μια domain specific language (DSL) για την περιγραφή μεταμοντέλων και παρέχεται με το πακέτο της ATL. Η περιγραφή αυτή δίνεται στη συνέχεια:

```
package NetModel {

	abstract class Element {
		attribute name : String;
		attribute ident: String;
	}

	class NetModel extends Element {
		reference prtNet : PrTnet;
	}

	class PrTnet extends Element {
		reference startPlace container : Place;
		reference  subnet[*] container : NetComponent;
	}

	abstract class NetComponent extends Element {
		reference finalPlace container : Place;
	}

	class Event extends NetComponent {
		reference initialPlaceToEventTransArc container : PlaceToTransArc;
		reference eventTrans container : Transition;
		reference eventTransToFinalPlaceArc container : TransToPlaceArc;
	}

	-- CF level for alt
	class Alternative extends NetComponent {
		reference guardedOperands[2-*] container : GuardedOperand;
	}

	-- CF level for option, loop
	class OptionLoop extends NetComponent {
		reference guardedOperand container : GuardedOperand;
		reference elseGuardedTransition container : GTransition;
		reference initialPlaceToElseGTransArc container : PlaceToTransArc;
		reference elseGTransToFinalPlaceArc container : TransToPlaceArc;
	}

	-- operand level for CFs  with guarded operands
	class GuardedOperand extends Element {
		reference guardedTrans container : GTransition;
		reference initialPlaceToGTransArc container : PlaceToTransArc;
		reference gTransToSubnetStartPlaceArc container : TransToPlaceArc;
		reference subPrTnet container : PrTnet;
		reference operEndTrans container : GTransition;
		reference operEndTransToFinalPlaceArc container : TransToPlaceArc;
		reference subLastPlaceToOperEndTransArc container : PlaceToTransArc;
	}
```

```
-- CF level for par
class Parallel extends NetComponent {
        reference startGTransition container: GTransition;
        reference initialPlaceToStartGTransArc container: PlaceToTransArc;
        reference parallelOperands [1-*] container: ParallelOperand;
        reference endGTransition container: GTransition;
        reference endGTransToFinalPlaceArc container: TransToPlaceArc;
}

-- operand level for par
class ParallelOperand extends Element {
        reference startGTransToSubnetStartPlaceArc container: TransToPlaceArc;
        reference subPrTnet container: PrTnet;
        reference subLastPlaceToEndTransArc container: PlaceToTransArc;
}


-- low level Petri net elements

class NetElement extends Element {
        attribute label : String;
}

abstract class Node extends NetElement { }

class Place extends Node {
        attribute initialMarking : Integer;
        reference incoming[*] : TransToPlaceArc oppositeOf target;
        reference outgoing[*] : PlaceToTransArc oppositeOf source;
}

class Transition extends Node {
        reference incoming[1-*] : PlaceToTransArc oppositeOf target;
        reference outgoing[1-*] : TransToPlaceArc oppositeOf source;
}

class GTransition extends Transition {
        attribute condition : String;
        attribute conditionLanguage : String;
        attribute conditionVariable : String;
        attribute owner : String; -- Combined Fragment ID
        attribute ownerType : String; -- interaction operator of the CF
}

abstract class Arc extends NetElement {
        attribute weight : Integer;
}

class PlaceToTransArc extends Arc {
        reference source : Place oppositeOf outgoing;
        reference target : Transition oppositeOf incoming;
}
```

```
        class TransToPlaceArc extends Arc {
                reference source : Transition oppositeOf outgoing;
                reference target : Place oppositeOf incoming;
        }
}

package PrimitiveTypes {
        datatype String;
        datatype Integer;
}
```

Από την περιγραφή σε KM3 παράγεται τελικά το NetModel σε «Ecore» μορφή. Στο σημείο αυτό υπάρχουν τα δύο μεταμοντέλα (UML2, NetModel), το μοντέλο πηγή (Διάγραμμα Ακολουθίας) και η μηχανή μετασχηματισμού (ATL transformation engine). Σύμφωνα με το Σχήμα 3.2, εκείνο που απομένει είναι ο ορισμός του μετασχηματισμού, δηλαδή η διατύπωση των κανόνων αντιστοίχισης των στοιχείων του ενός μεταμοντέλου στο δεύτερο. Ο ορισμός αυτός γίνεται με ένα ATL module στο οποίο διατυπώνονται οι κανόνες που περιγράφηκαν στην προηγούμενη ενότητα -κάποιοι σε συνεπτυγμένη αλλά ισοδύναμη μορφή- και ορίζονται κάποιοι βοηθοί (*helpers*) για την εξυπηρέτηση του μετασχηματισμού. Οι βοηθοί στην ATL έχουν ανάλογο ρόλο με αυτό των μεθόδων/συναρτήσεων στις συνήθεις γλώσσες προγραμματισμού.

Στη συνέχεια παρατίθενται διατυπωμένοι σε ATL οι κανόνες για τα απλά μηνύματα (*Event*) και για τα συνδυασμένα τμήματα alt, opt, par, loop. Όπως περιγράφηκε στην αντίστοιχη ενότητα, για τα συνδυασμένα τμήματα γίνεται διαχωρισμός σε δύο επίπεδα: επίπεδο ΣΤ, επίπεδο τελεστέου. Για τα επίπεδα ΣΤ ορίζονται τρεις κανόνες: *AlternativeCombinedFragment* (alt), *ParallelCombinedFragment* (par), *OptionLoop* (opt, loop). Για τα opt και loop ορίζεται ένας κανόνας καθώς στο επίπεδο ΣΤ οι αντιστοιχήσεις είναι οι ίδιες. Για τα επίπεδα τελεστέων ορίζονται δύο κανόνες, οι *GuardedOperand* (alt, opt, loop) και *ParallelOperand* (par). Ο κανόνας GuardedOperand καλύπτει τους τελεστέους για alt, opt και loop με τον ίδιο τρόπο εκτός από το TtParc που ξεκινά από την (τελική) TBD μετάβαση και που στην περίπτωση του loop καταλήγει στη θέση εισόδου του συνδυασμένου τμήματος, ενώ στις περιπτώσεις των alt και opt καταλήγει στη θέση εξόδου του συνδυασμένου τμήματος. Ο χειρισμός αυτός γίνεται στο εσωτερικό του κανόνα. Τέλος, σε διάφορα σημεία των κανόνων χρησιμοποιούνται κάποιοι βοηθοί οι οποίοι παρατίθενται στη συνέχεια της ενότητας.

```
rule Event {
        from
                bes : UML2!BehaviorExecutionSpecification (
                        thisModule.allInterFrag.includes(bes)
                --      and bes.start.message.messageSort <> #reply
                )
        to
                net_component : NetModel!Event (
                                ident <- 'ENC' + thisModule.allInterFrag
                                        ->select(e | e.oclIsTypeOf(UML2!BehaviorExecutionSpecification))
                                        ->asSequence()->indexOf(bes).toString(),
                                name <- bes.start.covered->first().represents.type.name + '.'
                                                                + bes.start.message.name,
                                initialPlaceToEventTransArc <- ipArc,
                                eventTrans <- eTrans,
                                eventTransToFinalPlaceArc <- fpArc,
                                finalPlace <- fPlace
                        ),
                eTrans : NetModel!Transition (
                                ident <- 'EVT' + thisModule.allInterFrag
                                        ->select(e | e.oclIsTypeOf(UML2!BehaviorExecutionSpecification))
                                        ->asSequence()->indexOf(bes).toString(),
                                 name <- net_component.name,
                                 label <- bes.start.message.sendEvent.covered
                                          ->first().represents.name + '->' + bes.start.covered
                                          ->first().represents.name
                        ),
                ipArc : NetModel!PlaceToTransArc (
                                name <- 'PtTArc_' + eTrans.name + thisModule.allInterFrag
                                        ->select(e | e.oclIsTypeOf(UML2!BehaviorExecutionSpecification))
                                        ->asSequence()->indexOf(bes).toString(),
                                target <- eTrans,
                                source <- if bes = thisModule.root   then
                                                thisModule.resolveTemp(UML2!Interaction.allInstances()->first(),
                                                                                'net_model').prtNet.startPlace
                                          else
                                                bes.getInitialPlace()
                                          endif
                        ),
                fpArc : NetModel!TransToPlaceArc (
                                name <- 'TtPArc_' + eTrans.name + thisModule.allInterFrag
                                        ->select(e | e.oclIsTypeOf(UML2!BehaviorExecutionSpecification))
                                        ->asSequence()->indexOf(bes).toString(),
                                source <- eTrans,
                                target <- net_component.finalPlace
                ),
                fPlace : NetModel!Place (
                                ident <- 'EVFP' + thisModule.allInterFrag
                                        ->select(e | e.oclIsTypeOf(UML2!BehaviorExecutionSpecification))
                                        ->asSequence()->indexOf(bes).toString(),
                                name <- net_component.name + '_fp',
                                initialMarking <- 0
                        )
}


rule AltCombinedFragment {
        from
                c_f : UML2!CombinedFragment(
                        thisModule.allInterFrag.includes(c_f)
                        and
                        c_f.interactionOperator = #alt
                )
        to
                net_component : NetModel!Alternative (
                                ident <- 'ANC' + thisModule.allInterFrag
                                        ->select(e | e.oclIsTypeOf(UML2!CombinedFragment))
                                        ->asSequence()->indexOf(c_f).toString(),
                                name <- c_f.interactionOperator.toString() + thisModule.allInterFrag
                                        ->asSet()->select(q | q.oclIsTypeOf(UML2!CombinedFragment))
                                        ->select(qq | qq.interactionOperator = c_f.interactionOperator)
```

```
                                        ->asSequence()->indexOf(c_f).toString(),
                                guardedOperands <- Set{
                                                thisModule.allGuardedInterOper
                                                ->select(q | q.refImmediateComposite() = c_f)
                                                ->collect(e | thisModule.resolveTemp(e, 'guarded_oper'))
                                                },
                                finalPlace <- fPlace
                        ),
                fPlace : NetModel!Place (
                                ident <- 'AFP' + thisModule.allInterFrag
                                        ->select(e | e.oclIsTypeOf(UML2!CombinedFragment))
                                        ->asSequence()->indexOf(c_f).toString(),
                                name <- net_component.name + '_fp',
                                initialMarking <- 0
                        )
}


rule ParCombinedFragment {
        from
                c_f : UML2!CombinedFragment(
                        thisModule.allInterFrag.includes(c_f)
                        and
                        c_f.interactionOperator = #par
                )
        to
                net_component : NetModel!Parallel (
                                ident <- 'PNC' + thisModule.allInterFrag
                                        ->select(e | e.oclIsTypeOf(UML2!CombinedFragment))
                                        ->asSequence()->indexOf(c_f).toString(),
                                name <- c_f.interactionOperator.toString() + thisModule.allInterFrag
                                        ->asSet()->select(q | q.oclIsTypeOf(UML2!CombinedFragment))
                                        ->select(qq | qq.interactionOperator = c_f.interactionOperator)
                                        ->asSequence()->indexOf(c_f).toString(),
                                parallelOperands <- Set{
                                                thisModule.allParallelInterOper
                                                ->select(q | q.refImmediateComposite() = c_f)
                                                ->collect(e | thisModule.resolveTemp(e, 'parallel_oper'))
                                                },
                                startGTransition <- sTrans,
                                initialPlaceToStartGTransArc <- ipstArc,
                                endGTransition <- eTrans,
                                endGTransToFinalPlaceArc <- etfpArc,
                                finalPlace <- fPlace
                        ),
                sTrans : NetModel!GTransition (
                                ident <- 'PST' + thisModule.allInterFrag
                                        ->select(e | e.oclIsTypeOf(UML2!CombinedFragment))
                                        ->asSequence()->indexOf(c_f).toString(),
                                name <- net_component.name + '_start',
                                condition <- 'true',
                                conditionLanguage <- 'default',
                                conditionVariable <- '-1',
                            owner <- net_component.ident,
                            ownerType <- c_f.interactionOperator.toString()

                        ),
                ipstArc : NetModel!PlaceToTransArc (
                                name <- 'IPtoST_' + net_component.name,
                                target <- sTrans,
                                source <- if c_f = thisModule.root  then
                                                thisModule.resolveTemp(UML2!Interaction.allInstances()->first(),
                                                                                'net_model').prtNet.startPlace
                                        else
                                                c_f.getInitialPlace()
                                        endif
                        ),
                eTrans : NetModel!GTransition (
                                ident <- 'PET' + thisModule.allInterFrag
                                        ->select(e | e.oclIsTypeOf(UML2!CombinedFragment))
                                        ->asSequence()->indexOf(c_f).toString(),
```

```
                                        name <- net_component.name + '_end',
                                        condition <- 'true',
                                        conditionLanguage <- 'default',
                                        conditionVariable <- '-1',
                                    owner <- net_component.ident,
                                    ownerType <- c_f.interactionOperator.toString()
                                ),
                        fPlace : NetModel!Place (
                                        ident <- 'PFP' + thisModule.allInterFrag
                                                ->select(e | e.oclIsTypeOf(UML2!CombinedFragment))
                                                ->asSequence()->indexOf(c_f).toString(),
                                        name <- net_component.name + '_fp',
                                        initialMarking <- 0
                                ),
                        etfpArc : NetModel!TransToPlaceArc (
                                        name <- 'ETtoFP_' + net_component.name,
                                        source <- eTrans,
                                        target <- fPlace
                                )
}


rule OptLoopCombinedFragment {
        from
                        c_f : UML2!CombinedFragment(
                                thisModule.allInterFrag.includes(c_f)
                                and
                                (c_f.interactionOperator = #opt or c_f.interactionOperator = #loop)
                        )
        to
                        net_component : NetModel!OptionLoop (
                                        ident <- 'OLNC' + thisModule.allInterFrag
                                                ->select(e | e.oclIsTypeOf(UML2!CombinedFragment))
                                                ->asSequence()->indexOf(c_f).toString(),
                                        name <- c_f.interactionOperator.toString() + thisModule.allInterFrag
                                                ->asSet()->select(q | q.oclIsTypeOf(UML2!CombinedFragment))
                                                ->select(qq | qq.interactionOperator = c_f.interactionOperator)
                                                ->asSequence()->indexOf(c_f).toString(),
                                        guardedOperand <- thisModule.resolveTemp(
                                                        thisModule.allGuardedInterOper
                                                        ->select(q | q.refImmediateComposite() = c_f)
                                                        ->first(), 'guarded_oper'),
                                        finalPlace <- fPlace,
                                        elseGuardedTransition <- egTrans,
                                        initialPlaceToElseGTransArc <- ipegArc,
                                        elseGTransToFinalPlaceArc <- egfpArc
                                ),
                        fPlace : NetModel!Place (
                                        ident <- 'OLFP' + thisModule.allInterFrag
                                                ->select(e | e.oclIsTypeOf(UML2!CombinedFragment))
                                                ->asSequence()->indexOf(c_f).toString(),
                                        name <- net_component.name + '_fp',
                                        initialMarking <- 0
                                ),
                        egTrans : NetModel!GTransition (
                                        ident <- 'OLEGT' + thisModule.allInterFrag
                                                ->select(e | e.oclIsTypeOf(UML2!CombinedFragment))
                                                ->asSequence()->indexOf(c_f).toString(),
                                        name <- 'else_' + net_component.name,
                                        condition <- 'else',
                                        conditionLanguage <- 'default',
                                        conditionVariable <- thisModule.allInterFrag
                                                        ->select(q | q.oclIsTypeOf(UML2!CombinedFragment))
                                                        ->select(w |  w.interactionOperator = #alt
                                                                        or w.interactionOperator = #opt
                                                                        or w.interactionOperator = #loop)
                                                        ->asSequence()->indexOf(c_f).toString(),
                                    owner <- net_component.ident,
                                    ownerType <- c_f.interactionOperator.toString()
                                ),
                        ipegArc : NetModel!PlaceToTransArc (
```

```
                                        name <- 'IPtoEGarc_' + egTrans.name,
                                        target <- egTrans,
                                        source <- if c_f = thisModule.root then
                                                thisModule.resolveTemp(UML2!Interaction.allInstances()
                                                        ->first(), 'net_model').prtNet.startPlace
                                                else
                                                        c_f.getInitialPlace()
                                                endif
                                ),
                        egfpArc : NetModel!TransToPlaceArc (
                                        name <- 'EGtoFParc_' + egTrans.name,
                                        source <- egTrans,
                                        target <- fPlace
                                )
}


rule GuardedOperand {
        from
                inter_oper : UML2!InteractionOperand (
                                        thisModule.allGuardedInterOper->includes(inter_oper)
                                )
        to
                guarded_oper : NetModel!GuardedOperand (
                                        ident <- 'GO' + UML2!InteractionOperand.allInstances()
                                                ->asSequence()->indexOf(inter_oper).toString(),
                                        guardedTrans <- gTrans,
                                        initialPlaceToGTransArc <- ipArc,
                                        gTransToSubnetStartPlaceArc <- subArc,
                                        subPrTnet <- sPrTnet,
                                        operEndTrans <- oeTrans,
                                        subLastPlaceToOperEndTransArc <- sloeArc,
                                        operEndTransToFinalPlaceArc <- oefpArc
                                ),
                        gTrans : NetModel!GTransition (
                                        ident <- 'GOGT' + UML2!InteractionOperand.allInstances()
                                                ->asSequence()->indexOf(inter_oper).toString(),
                                        name <- 'gtrans' + thisModule.allGuardedInterOper
                                                ->indexOf(inter_oper).toString(),
                                        condition <- if inter_oper.guard.oclIsUndefined() then
                                                        'else' -- 'true' σύμφωνα με το [UML], βλ. 4.4
                                                else if
                                                inter_oper.guard.specification.oclIsTypeOf(UML2!OpaqueExpression)
                                                        then
                                                                inter_oper.guard.specification.body->first().toString()
                                                        else
                                                                inter_oper.guard.specification.value.toString()
                                                        endif
                                                endif,
                                        conditionLanguage <-  if inter_oper.guard.oclIsUndefined() then
                                                                'default'
                                                        else if
                                                inter_oper.guard.specification.oclIsTypeOf(UML2!OpaqueExpression)
                                                                then
                                                                        inter_oper.guard.specification.language
                                                                        ->first().toString()
                                                                else
                                                                        'English'
                                                                endif
                                                        endif,
                                        conditionVariable <- thisModule.allInterFrag
                                                        ->select(q | q.oclIsTypeOf(UML2!CombinedFragment))
                                                        ->select(w |   w.interactionOperator = #alt
                                                                        or w.interactionOperator = #opt
                                                                        or w.interactionOperator = #loop)
                                                        ->asSequence()
                                                        ->indexOf(inter_oper.refImmediateComposite()).toString(),
                                        owner <- thisModule.resolveTemp(inter_oper.refImmediateComposite(),
                                                                        'net_component').ident,
                                        ownerType <- inter_oper.refImmediateComposite().interactionOperator.toString()
                                ),
```

```
                ipArc : NetModel!PlaceToTransArc (
                        name <- 'PtTArc_' + gTrans.name,
                        target <- gTrans,
                        source <- if inter_oper.refImmediateComposite() = thisModule.root  then
                                thisModule.resolveTemp(UML2!Interaction.allInstances()
                                ->first(), 'net_model').prtNet.startPlace
                            else
                                inter_oper.refImmediateComposite().getInitialPlace()
                            endif
                ),
                sPrTnet : NetModel!PrTnet (
                        ident <- 'GOSN' +  UML2!InteractionOperand.allInstances()
                            ->asSequence()->indexOf(inter_oper).toString(),
                        startPlace <- sPlace,
                        subnet <- Set{
                            thisModule.allInterFrag
                            ->select(q | q.refImmediateComposite() = inter_oper)
                            ->collect(e | thisModule.resolveTemp(e, 'net_component'))
                        }
                ),
                sPlace : NetModel!Place (
                        ident <- 'GOSP' +  UML2!InteractionOperand.allInstances()
                            ->asSequence()->indexOf(inter_oper).toString(),
                        name <- gTrans.name + '_sp',
                        initialMarking <- 0
                ),
                subArc : NetModel!TransToPlaceArc (
                        name <- 'GTtSPArc_' + gTrans.name,
                        source <- gTrans,
                        target <- sPlace
                ),
                oeTrans : NetModel!GTransition (
                        ident <- 'GOEGT' +  UML2!InteractionOperand.allInstances()
                            ->asSequence()->indexOf(inter_oper).toString(),
                        name <- 'TBD_' + gTrans.name,
                        condition <- 'true',
                        conditionLanguage <- 'default',
                        conditionVariable <- '-1',
                        owner <- thisModule.resolveTemp( inter_oper.refImmediateComposite(),
                                                                'net_component').ident,
                        ownerType <- inter_oper.refImmediateComposite().interactionOperator.
                                                                toString()
                ),
                sloeArc : NetModel!PlaceToTransArc (
                        name <- oeTrans.name + '_SLPtoOEArc',
                        target <- oeTrans,
                        source <- thisModule.resolveTemp(
                                thisModule.allInterFrag
                                ->select(q | q.refImmediateComposite() = inter_oper)
                                ->asSequence()->last(), 'fPlace')
                ),
                oefpArc : NetModel!TransToPlaceArc (
                        name <- oeTrans.name + '_OEtoFPArc',
                        source <- oeTrans,
                        target <- if
                                inter_oper.refImmediateComposite().interactionOperator = #loop
                            then
                              if
                                inter_oper.refImmediateComposite() = thisModule.root
                              then
                                thisModule.resolveTemp(UML2!Interaction.allInstances()
                                                ->first(), 'net_model').prtNet.startPlace
                              else
                                inter_oper.refImmediateComposite().getInitialPlace()
                              endif
                            else
                          thisModule.resolveTemp(inter_oper.refImmediateComposite(),'fPlace')
                              endif
                )
}
```

```
rule ParallelOperand {
        from
                inter_oper : UML2!InteractionOperand (
                                thisModule.allParallelInterOper->includes(inter_oper)
                        )
        to
                parallel_oper : NetModel!ParallelOperand (
                                ident <- 'PO' + UML2!InteractionOperand.allInstances()
                                        ->asSequence()->indexOf(inter_oper).toString(),
                                startGTransToSubnetStartPlaceArc <- stspArc,
                                subPrTnet <- sPrTnet,
                                subLastPlaceToEndTransArc <- slpetArc
                        ),
                        sPrTnet : NetModel!PrTnet (
                                ident <- 'POSN' + UML2!InteractionOperand.allInstances()
                                ->asSequence()->indexOf(inter_oper).toString(),
                                startPlace <- sPlace,
                                subnet <- Set{
                                                thisModule.allInterFrag
                                                ->select(q | q.refImmediateComposite() = inter_oper)
                                                ->collect(e | thisModule.resolveTemp(e, 'net_component'))
                                               }
                        ),
                        sPlace : NetModel!Place (
                                ident <- 'POSP' + UML2!InteractionOperand.allInstances()
                                        ->asSequence()->indexOf(inter_oper).toString(),
                                name <- 'startTrans' + thisModule.allParallelInterOper
                                        ->indexOf(inter_oper).toString() + '_sp',
                                initialMarking <- 0
                        ),
                        stspArc : NetModel!TransToPlaceArc (
                                name <- 'STtoSP_' + thisModule.allParallelInterOper
                                        ->indexOf(inter_oper).toString(),
                                target <- sPlace,
                                source <- thisModule.resolveTemp(inter_oper.refImmediateComposite(),
                                                                        'net_component').startGTransition
                        ),
                        slpetArc : NetModel!PlaceToTransArc (
                                name <- 'SLPtoET_' + thisModule.allParallelInterOper
                                        ->indexOf(inter_oper).toString(),
                                source <- thisModule.resolveTemp(thisModule.allInterFrag
                                        ->select(q | q.refImmediateComposite() = inter_oper)
                                        ->asSequence()->last(), 'fPlace'),
                                target <- thisModule.resolveTemp(inter_oper.refImmediateComposite(),
                                                                        'net_component').endGTransition
                        )
}
```

Στους κανόνες και ειδικά στην παράγωγή των μοναδικών αναγνωριστικών θα μπορούσαν να χρησιμοποιηθούν επιπλέον βοηθοί ή προστακτικός κώδικας, για τη βελτίωση της επίδοσης. Προτιμήθηκε όμως ο όσο το δυνατόν περισσότερο εξατομικευμένος χειρισμός των κανόνων.

Λόγω κυρίως της συχνής τους χρήσης ορίστηκαν οι παρακάτω βοηθοί:

```
helper def: allInterFrag : Sequence(UML2!InteractionFragment) = UML2!InteractionFragment.allInstances()
                        ->select(w |   (w.oclIsTypeOf(UML2!BehaviorExecutionSpecification)
                                        or w.oclIsTypeOf(UML2!CombinedFragment)))
                        ->asSequence();

helper def: allGuardedInterOper : Sequence(UML2!InteractionOperand) = UML2!InteractionOperand.allInstances()
                                ->select(q | q.refImmediateComposite().interactionOperator = #alt
                                                or q.refImmediateComposite().interactionOperator = #opt
                                                or q.refImmediateComposite().interactionOperator = #loop)
                                ->asSequence();
```

```
helper def: allParallelInterOper : Sequence(UML2!InteractionOperand) = UML2!InteractionOperand.allInstances()
                                    ->select(q | q.refImmediateComposite().interactionOperator = #par)
                                    ->asSequence();

helper def: root : UML2!InteractionFragment = thisModule.allInterFrag->first();
```

Με το βοηθό *allInterFrag* υπολογίζεται μια ακολουθία με όλα τα τμήματα αλληλεπίδρασης που ενδιαφέρουν, δηλαδή τα τμήματα που θα μετασχηματιστούν. Με το βοηθό *allGuardedInterOper* υπολογίζεται μια ακολουθία με όλους τους τελεστέους που ανήκουν σε συνδυασμένα τμήματα με χειριστές alt, opt ή loop. Ανάλογος υπολογισμός γίνεται και με το βοηθό allParallelInterOper αλλά για τελεστέους συνδυασμένων τμημάτων par.

Οι παραπάνω βοηθοί ανήκουν στην κατηγορία των *attributes* της ATL, καθώς δεν έχουν παραμέτρους και αρκεί ο εφάπαξ στατικός υπολογισμός τους που γίνεται στην αρχή της εκτέλεσης του μετασχηματισμού διότι είναι ορισμένοι στο context του module.

Πέρα όμως από τους παραπάνω βοηθούς ορίζονται δύο ακόμη που είναι και οι σημαντικότεροι καθώς από την υλοποίησή τους εξαρτάται η πιστότητα μετασχηματισμού της πληροφορίας διάταξης των τμημάτων στη δομή του Δικτύου Petri που προκύπτει. Συγκεκριμένα ο πρώτος καλείται για τον προσδιορισμό της θέσης εισόδου κάθε τμήματος (*getInitialPlace*). Η θέση αυτή, όπως έχει ήδη αναφερθεί είναι είτε η θέση εξόδου που αντιστοιχεί στο προηγούμενο (βάση του σεναρίου) τμήματος, είτε η εσωτερική αρχική θέση κάποιου τελεστέου αλληλεπίδρασης. Παρόλα αυτά, στο επίπεδο της υλοποίησης η εύρεση του προηγούμενου τμήματος δεν απλή καθώς τα τμήματα σειριοποιούνται -αυτό μπορεί να γίνει σε μια από τους δύο τύπους συλλογών δεδομένων που φέρουν πληροφορία διάταξης της ATL (Sequence, OrdedSet)- και στη σειριοποιημένη μορφή το προηγούμενο τμήμα μπορεί να βρίσκεται σε οποιαδήποτε θέση πριν τη θέση του τμήματος που εξετάζεται.

Ο δεύτερος βοηθός (*getSameLevelCF*) καλείται από τον πρώτο, υπό περίπτωση, για να προσδιορίσει το τμήμα που προηγείται, βρίσκεται στο ίδιο επίπεδο και απέχει τη μικρότερη απόσταση (στη σειριοποιημένη ακολουθία) από το υπό εξέταση τμήμα.

Οι ορισμοί των παραπάνω βοηθών παρατίθενται στη συνέχεια:

```
helper context UML2!InteractionFragment
def: getInitialPlace() : NetModel!Place =
        let i : Integer = thisModule.allInterFrag->indexOf(self) in
        let x : UML2!InteractionFragment = thisModule.allInterFrag->at(i-1) in
        if self.refImmediateComposite() = x.refImmediateComposite() then
        -- αν είναι στο ίδιο επίπεδο επέστρεψε τη τελική θέση του προηγουμένου
            thisModule.resolveTemp(x, 'fPlace')
        else
        -- αν δεν είναι στο ίδιο επίπεδο:
          if self = thisModule.allInterFrag->select(w | w.refImmediateComposite()
                                                = self.refImmediateComposite())->asSequence()->first() then
            -- αν περιλαμβάνεται σε τελεστέο και είναι το πρώτο επέστρεψε την εσωτερική αρχική θέση
                thisModule.resolveTemp(self.refImmediateComposite(), 'sPlace')
          else
            -- διαφορετικά επέστρεψε την τελική θέση του τμήματος που περιλαμβάνει το προηγούμενο στη
σειριοποιημένη διάταξη και είναι του ίδιο επιπέδου
                thisModule.resolveTemp(self.getSameLevelCF(x), 'fPlace')
```

```
            endif
        endif;


helper context UML2!InteractionFragment
def: getSameLevelCF(x: UML2!InteractionFragment) : UML2!CombinedFragment =
        if self.refImmediateComposite() <>
                        x.refImmediateComposite().refImmediateComposite().refImmediateComposite() then
          -- αν δε περιλαμβάνονται στο ίδιο συνδυασμένο τμήμα ή πρώτο επίπεδο, έλεγξε το επόμενο επίπεδο
                self.getSameLevelCF(x.refImmediateComposite().refImmediateComposite())
        else
          -- διαφορετικά επέστρεψε το συνδυασμένο τμήμα που περιλαμβάνει το x
                x.refImmediateComposite().refImmediateComposite()
        endif;
```

Εξαγωγή Δικτύου Petri σε PNML μορφή (χωρίς πληροφορίες γραφικής απεικόνισης) – ATL query.

Στο βήμα αυτό έχει πλέον παραχθεί το δίκτυο το οποίο είναι εκφρασμένο σε Ecore XMI και συμμορφώνεται στο NetModel. Όπως αναφέρθηκε προηγουμένως η επεκταμένη μορφή του NetModel περιλαμβάνει κλάσεις *containers* για την σαφέστερη διατύπωση των αντιστοιχίσεων των στοιχείων του μεταμοντέλου της UML. Τα στοιχεία όμως του NetModel που ενδιαφέρουν τελικά είναι μόνο οι μεταβάσεις (φυλασσόμενες ή μη), οι θέσεις και τα διανύσματα. Έτσι ορίζουμε έναν απλό κανόνα εξαγωγής στον οποίο τα παραπάνω στοιχεία εκφράζονται σύμφωνα με τους συντακτικού κανόνες της PNML. Μάλιστα, για την έκφραση ιδιαίτερων μεταδεδομένων του δικτύου που παράγεται (εκφράσεις φύλαξης, ετικέτες αντικειμένων, δομής κλπ) τα οποία δε μπορούν να περιληφθούν στη βασική PNML μορφή, ορίζεται μια επέκταση του μορφότυπου της PNML και τελικά το δίκτυο εξάγεται στη μορφή αυτή. Το ATL query της εξαγωγής σε PNML δίνεται στη συνέχεια:

```
query QNetModelToPNML = NetModel!PrTnet.allInstances()
                        ->asSequence()->first().toString2().writeTo('C:\\queryOut.pnml'); -- Query Template

helper context NetModel!Element def: toString2() : String =
        let na : Sequence(NetModel!NetElement) = NetModel!NetElement.allInstances()
                        ->select(e | e.oclIsKindOf(NetModel!Place)
                                or e.oclIsKindOf(NetModel!Transition)
                                or e.oclIsKindOf(NetModel!Arc)) in
'<?xml version="1.0" encoding="iso-8859-1"?>' +
'<pnml>' +
'<net id="net" type="P/T net">' +
        if na->size() > 0 then
                na->iterate(e; acc : String = '' |
                        acc +
                                if e.oclIsTypeOf(NetModel!Transition) then
                                        e.toString2()
                                else
                                        if e.oclIsTypeOf(NetModel!GTransition) then
                                                e.toString2()
                                        else
                                                if e.oclIsTypeOf(NetModel!PlaceToTransArc) then
                                                        e.toString2()
                                                else
                                                        if e.oclIsTypeOf(NetModel!TransToPlaceArc) then
                                                                e.toString2()
```

```
                                                    else ''
                                                    endif
                                                endif
                                            endif
                                        endif
                                    + '')
            else
                        ''
            endif
            +
'</net>' +
'</pnml>';

helper context NetModel!Place def: toString2() : String =
                        '<place id="' + self.ident + '">' +
                        '<name><value>'+ self.name +'</value></name>' +
                        '<initialMarking><value>' + self.initialMarking.toString() + '</value></initialMarking>' +
                        '<capacity><value>1</value></capacity>' +
                        '</place>';

helper context NetModel!Transition def: toString2() : String =
                        '<transition id="'+ self.ident +'">' +
                        '<name><value>'+ self.name +'</value></name>'+
                        '<orientation><value>0</value></orientation>' +
                        '<rate><value>1.0</value></rate>' +
                        '<timed><value>false</value></timed>' +
                        '<object><value>' + self.label + '</value></object>' +
                        '<guarded><value>false</value><condition/><owner/></guarded>' +
                        '<infiniteServer><value>false</value></infiniteServer>' +
                        '<priority><value>1</value></priority>' +
                        '</transition>';

helper context NetModel!GTransition def: toString2() : String =
                        '<transition id="'+ self.ident +'">' +
                        '<name><value>'+ self.name +'</value></name>'+
                        '<orientation><value>0</value></orientation>' +
                        '<rate><value>1.0</value></rate>' +
                        '<timed><value>false</value></timed>' +
                        '<object><value>null</value></object>' +
                        '<guarded><value>true</value>' +
                        '<condition><value>'+ self.condition + '</value>' +
                                    '<language>' + self.conditionLanguage + '</language>' +
                                    '<variable>' + self.conditionVariable + '</variable></condition>' +
                        '<owner><id>' + self.owner +'</id>' +
                                '<type>' + self.ownerType + '</type></owner></guarded>' +
                        '<infiniteServer><value>false</value></infiniteServer>' +
                        '<priority><value>1</value></priority>' +
                        '</transition>';

helper context NetModel!PlaceToTransArc def: toString2() : String =
'<arc id="'+ self.name + '" source="' + self.source.ident +'" target="' + self.target.ident + '"></arc>';

helper context NetModel!TransToPlaceArc def: toString2() : String =
'<arc id="'+ self.name + '" source="' + self.source.ident +'" target="' + self.target.ident + '"></arc>';
```

Γραφική Απόδοση Δικτύου Petri.

Όπως αναφέρθηκε παραπάνω το τμήμα αυτό μπορεί να παραληφθεί αν στοχεύουμε στην ανάλυση των μοντέλων χωρίς να ενδιαφέρει η οπτική αναπαράστασή τους. Παρόλα αυτά, η φιλική προς τον χρήστη απεικόνιση που επιτρέπουν τα Δίκτυα Petri θεωρείται ένα από τα σημαντικότερα πλεονεκτήματα του τύπου αυτού μοντέλων, μαζί με τη σαφή μαθηματική τους διατύπωση και τη δυνατότητα εκτέλεσης και αλληλεπίδρασης. Για το λόγο αυτό θεωρήθηκε σημαντική η επιστράτευση κάποιας τεχνικής για τη γραφική απόδοση του δικτύου σε αποδεκτή μορφή.

Οι πληροφορίες γραφικής απεικόνισης σίγουρα δεν μπορούν να εξαχθούν από το αρχικό μοντέλο. Για την αντιμετώπιση αυτού του ζητήματος χρησιμοποιήθηκαν υλοποιήσεις αλγορίθμων αυτόματης απόδοσης γράφων που περιλαμβάνονται στο πακέτο *Graphviz*. Για να είναι δυνατή η εκτέλεση των εργαλείων αυτών πρέπει η δομή του δικτύου να αναπαρασταθεί στην ειδικού σκοπού γλώσσα *dot*. Άρα προκύπτει η ανάγκη τουλάχιστον ενός μετασχηματισμού από το αρχικό δίκτυο σε dot κι ενός από dot με πληροφορίες απεικόνισης σε PNML. Υπάρχουν διάφορες λύσεις για το θέμα αυτό καθώς κάποια εργαλεία υλοποιούν ήδη τους παραπάνω αλλά και ανάλογους μετασχηματισμούς. Ένας τέτοιο εργαλείο είναι το TINA που παρουσιάστηκε σε προηγούμενη ενότητα (βλ. 3.3).

Μετασχηματισμός πληροφοριών δομής του PNML σε TINA.

Το TINA είναι ένα εργαλείο για την επεξεργασία και την ανάλυση συνήθων και χρονικών Δικτύων Petri. Για τον προσδιορισμό των δικτύων αυτών χρησιμοποιεί μια ιδιωτική γλώσσα, στην οποία αν και μπορούν να εκφραστούν τα δομικά χαρακτηριστικά του δικτύου (κόμβοι, διανύσματα) δεν υπάρχει δυνατότητα να εκφραστούν οι άλλες πληροφορίες που φέρουν τα στοιχεία που παράγονται από το μετασχηματισμό και οι οποίες είναι απαραίτητες για την ανάλυση. Παρόλα αυτά, στο σημείο αυτό ενδιαφέρει η γραφική αναπαράσταση του δικτύου κι έτσι αρκεί η εισαγωγή των πληροφοριών δομής. Μάλιστα η δυνατότητα εισαγωγής δικτύων σε PNML μορφή που προσφέρει το TINA και ο αυτόματος μετασχηματισμός τους στην ιδιωτική γλώσσα του TINA απλοποιεί τη διαδικασία. Είναι όμως σαφές ότι το δίκτυο που εισάγεται δεν είναι πλήρες καθώς έχουν αποκοπεί δεδομένα τα οποία το TINA δε μπορεί να μεταφράσει/διαχειριστεί.

Για την υλοποίηση του βήματος αυτού δημιουργήθηκε ένα ATL query παρόμοιο με αυτό που παρουσιάστηκε προηγουμένως, με τη διαφορά ότι περιελάμβανε μόνο τα βασικά δομικά χαρακτηριστικά του δικτύου.

Μετασχηματισμός TINA σε dot

Αφού εκφραστεί ένα δίκτυο στην ιδιωτική γλώσσα του TINA, εφόσον ο χρήστης επιλέξει την απεικόνιση του δικτύου, το εργαλείο υλοποιεί σε μη ορατό επίπεδο για το χρήστη ένα μετασχηματισμό / προσαρμογής της ιδιωτικής γλώσσας σε dot.

Απεικόνιση μέσω αλγορίθμου σχεδίασης – Graphviz

Στο σημείο αυτό, δίνεται στο χρήστη η δυνατότητα επιλογής του αλγορίθμου απόδοσης και καλείται το αντίστοιχο εργαλείο του πακέτου Graphviz [GRAATT] για την εκτέλεση του αλγορίθμου. Από τις επιλογές που παρέχονται στο TINA, η επιλογή «neato» αποδίδει συνήθως απεικονίσεις που είναι πιο εύκολα αναγνώσιμες ειδικά για δίκτυα με περισσότερους από 15 κόμβους περίπου. Το neato υλοποιεί τον αλγόριθμο Kamada-Kawai για τη σχεδίαση γράφων.

Εξαγωγή των πληροφοριών δομής του Δικτύου Petri σε PNML με πληροφορίες γραφικής απεικόνισης.

Τελικά το δίκτυο απεικονίζεται σε μια αποδεκτή, φιλική προς τον χρήστη μορφή. Το TINA επιτρέπει την εξαγωγή ενός δικτύου σε PNML μορφή, στο οποίο πλέον περιλαμβάνονται οι πληροφορίες γραφικής απεικόνισης.

Συγχώνευση μεταδεδομένων δικτύου και πληροφοριών απεικόνισης σε μορφή PNML.

Στο τελευταίο αυτό βήμα γίνεται η συγχώνευση των πληροφοριών απεικόνισης με τις απαιτούμενες για την ανάλυση πληροφορίες/μεταδεδομένα του δικτύου. Για τη συγχώνευση αυτή δημιουργήθηκε ένα εργαλείο σε Java (*Merge*) στο οποίο εισάγονται τα δύο αρχεία σε PNML μορφή και εξάγεται ένα που συγχωνεύει τα περιεχόμενά τους και πάλι τηρώντας το PNML συντακτικό.

Τα βήματα που περιγράφηκαν παραπάνω αφορούν στη διαδικασία μετασχηματισμού ενός Διαγράμματος Ακολουθίας σε Δίκτυο Petri και απόδοσης γραφικών χαρακτηριστικών στο δίκτυο αυτό, όπως υλοποιείται στο περιβάλλον-πλαίσιο που παρουσιάζεται στην παρούσα εργασία. Παρόλα αυτά, μπορούν να χρησιμοποιηθούν διάφορες εναλλακτικές τόσο για την παραγωγή και την έκφραση του τελικού μοντέλου, όσο και για την αναπαράστασή του. Οι επιλογές που έγιναν και στα παραπάνω βήματα συμβαδίζουν με τρία ζητούμενα: τη χρήση γενικώς αποδεκτών και διαδεδομένων προτύπων (XMI, EMF UML2, PNML), την απόδοση (χρήση της ειδικού σκοπού γλώσσας ATL για τη διατύπωση του μετασχηματισμού) και την απλότητα (χρησιμοποίηση του εργαλείου TINA που υλοποιεί ήδη την απόδοση γραφικών

χαρακτηριστικών ενσωματώνοντας εργαλεία του Graphviz). Τελικά τα εργαλεία που δημιουργήθηκαν ήταν: ο ορισμός του μετασχηματισμού σε ATL (ATL module), η εξαγωγή από το ενδιάμεσο μοντέλο σε δίκτυο εκφρασμένο σε PNML (ATL query) και ένα πρόγραμμα συγχώνευσης των PNML αρχείων (Java).

4.5 Όρια και περιορισμοί

Στην ενότητα αυτή παρουσιάζονται τα όρια που εμφανίζει και οι περιορισμοί που θέτουν οι κανόνες και η διαδικασία του μετασχηματισμού που περιγράφηκαν παραπάνω.

Αρχικά, πρέπει να σημειωθεί ότι ο προτεινόμενος μετασχηματισμός δεν είναι πλήρης, δηλαδή δε μετασχηματίζονται όλα τα δυνατά στοιχεία που μπορούν να περιληφθούν σε κάποιο Διάγραμμα Ακολουθίας, σε στοιχεία ή δομές Δικτύου Petri. Στην εργασία αυτή μας ενδιαφέρει ο μετασχηματισμός της βασικής πληροφορίας που φέρει ένα Διάγραμμα Ακολουθίας που είναι η μερική διάταξη των γεγονότων και η ροή ελέγχου που περιλαμβάνεται σε κάθε σενάριο. Η μέριμνα για το μετασχηματισμό χαρακτηριστικών απολύτου χρόνου, επιπλέον τύπων συνδυασμένων τμημάτων, πυλών (Gates) ή περιορισμών (OCL constraints) πέρα όσων εκφράζονται με φύλακες θα μπορούσε σίγουρα να αποτελέσει αντικείμενο επέκτασης.

Επίσης, όπως αναφέρεται στο UML 2.x Superstructure specification [UML] οι αλληλεπιδράσεις δίνουν έμφαση στην αλληλουχία των μηνυμάτων η οποία είναι πολύ σημαντική για την κατανόηση μιας κατάστασης. Παρόλα αυτά, οι αλληλεπιδράσεις δεν επικεντρώνονται στον χειρισμό των δεδομένων που τα μηνύματα μπορεί να φέρουν (ή που οι γραμμές ζωής αποθηκεύουν) και ενδεχομένως να είναι εξίσου σημαντικά -αν και τα δεδομένα μπορεί να χρησιμοποιούνται για την «διακόσμηση» των διαγραμμάτων. Η παρατήρηση αυτή σχετίζεται με την επιλογή που γίνεται ώστε ο μετασχηματισμός να στοχεύει σε ένα Δίκτυο Petri χαμηλού επιπέδου. Παρόλα αυτά, η χρήση φυλασσόμενων μεταβάσεων και η αποτίμησή τους, που προκύπτει από «εξωτερικά» ως προς το δίκτυο δεδομένα, το απομακρύνει από το επίπεδο 1.

Επιπλέον, η χρήση φυλασσόμενων μεταβάσεων με εξορισμού αληθή συνθήκη (TBD) αν και απλοποιεί τους κανόνες και προσδίδει ομοιογένεια για κάθε τμήμα που μετασχηματίζεται, επιβαρύνει το δίκτυο με κόμβους και διανύσματα που θα μπορούσαν ενδεχομένως να παραληφθούν. Η επιλογή χρήσης αυτών των μεταβάσεων προήλθε έπειτα από τη στάθμιση των παραπάνω πλεονεκτημάτων και της πολυπλοκότητας τόσο του μετασχηματισμού όσο και του προκύπτοντος δικτύου χωρίς τη χρήση TBD μεταβάσεων (π.χ. αλληλουχία συνδυασμένων τμημάτων με χειριστή par).

Όπως ήδη αναφέρθηκε η υλοποίηση του μετασχηματισμού σε ATL περιλαμβάνει κανόνες μόνο για ορισμένα βασικά τμήματα των Διαγραμμάτων Ακολουθίας και συγκεκριμένα για τα γεγονότα και για τα συνδυασμένα τμήματα με χειριστές: alt, opt, par και loop. Για το χειριστή loop επίσης δε λαμβάνεται υπόψη η δυνατότητα ορισμού ελάχιστων και μέγιστων επαναλήψεων. Επιλέον, για το χειριστή par δε λαμβάνεται υπόψη η δυνατότητα ορισμού φυλάκων στους τελεστέους, αν και μια ισοδύναμη έκφραση είναι το φώλιασμα ενός συνδυασμένου τμήματος με χειριστή opt μέσα στον κάθε τελεστέο που απαιτεί φύλακα.

Για τα συνδυασμένα τμήματα με χειριστή alt όπως σημειώθηκε και παραπάνω, πρέπει να ορίζεται ένας τελεστέος με έκφραση φύλαξης else. Αυτό αποτελεί μια σύμβαση για την καλή λειτουργία του μετασχηματισμού και της διαδικασίας ανάλυσης που θα περιγραφεί στη συνέχεια της εργασίας. Επίσης, στο [UML] αναφέρεται ότι στην περίπτωση που ο φύλακας αφήνεται κενός σε κάποιο τελεστή συνδυασμένου τμήματος με χειριστή alt τότε υπονοείται ότι η έκφραση φύλαξης είναι true. Παρόλα αυτά, στο MagicDraw UML στα alt συνδυασμένα τμήματα που δημιουργούνται όταν στο διάγραμμα εμφανίζεται ως φύλακας το else, η έκφραση φύλαξης είναι κενή. Το γεγονός αυτό αντιμετωπίζεται από τον αντίστοιχο κανόνα.

Παράλληλα, μια σύμβαση που έχει γίνει είναι ότι τα αντικείμενα που συμμετέχουν στα Διαγράμματα Ακολουθίας έχουν προσδιοριστεί στο μοντέλο με κάποια αντίστοιχη κλάση, ώστε να εισάγεται το όνομά της στο πρώτο τμήμα του ονόματος κάθε μετάβασης. Οι κλάσεις αυτές πρέπει να περιλαμβάνονται στο XMI αρχείο που χρησιμοποιείται στην εκτέλεση του ATL module.

Επιπλέον, το μεταμοντέλο που ορίζεται για τα Δίκτυα Petri (NetModel στην επεκταμένη του μορφή) ορίζεται με τρόπο που εξυπηρετεί τη διατύπωση των κανόνων αντιστοίχησης. Ο προσδιορισμός του δηλαδή είναι γενικά αυθαίρετος και απέχει από τα ελάχιστα δυνατά στοιχεία που απαιτούνται.

Αν και η φιλοσοφία πίσω από τους κανόνες και τη διαδικασία που παρουσιάζεται μπορεί να θεωρηθεί πάγια, η υλοποίηση εξαρτάται άμεσα από τις εκδόσεις των διαφόρων προτύπων και ιδιαίτερα του μεταμοντέλου της UML. Είναι δηλαδή αναμενόμενο ότι η καλή λειτουργία των κανόνων αντιστοίχησης σε επίπεδο υλοποίησης, απαιτεί το συγχρονισμό τους με τις διάφορες εκδόσεις. Η υλοποίηση των κανόνων, με εξαιρέσεις τους περιορισμούς που αναφέρθηκαν παραπάνω, βρίσκεται σε συμφωνία με το UML 2.1 μεταμοντέλο.

Κεφάλαιο 5: Συλλογή και ανάλυση καταγραφών

Στο κεφάλαιο αυτό αναλύεται η διαδικασία της συλλογής (δημιουργίας, συγκέντρωσης, αποθήκευσης) και ανάλυσης των καταγραφών για τον έλεγχο εμφάνισης σεναρίων ενδιαφέροντος. Αρχικά παρουσιάζονται τα βασικά στοιχεία του συστήματος καταγραφής κι έπειτα περιγράφεται η τεχνική ενορχήστρωσης χωρίς επέμβαση στον πηγαίο κώδικα που χρησιμοποιείται στην υλοποίηση ενώ αναφέρονται και ορισμένες εναλλακτικές. Κατόπιν, συζητείται ο ορισμός του μοντέλο που ελέγχεται (Δίκτυα Petri και καταγραφές) και περιγράφεται η διαδικασία ανάλυσης και η εφαρμογή της τόσο ως προς αρχεία καταγραφής όσο και ως προς ακολουθίες γεγονότων που αναλύονται επιγραμμικά.

5.1 Σύστημα καταγραφής

Σύμφωνα με το διαχωρισμό των στοιχείων που αναλύθηκε σε προηγούμενη ενότητα (βλ. 3.1), περιγράφεται στη συνέχεια το σύστημα καταγραφής που χρησιμοποιείται στην υλοποίηση του πλαισίου:

Πρώτο στοιχείο: Σχήμα εγγραφών και επίπεδο καταγραφής

Γεγονότα ενδιαφέροντος θεωρούνται οι κλήσεις και οι επιστροφές μεθόδων. Στα βασικά χαρακτηριστικά κάθε εγγραφής περιλαμβάνεται η χρονική στιγμή που συνέβη το γεγονός (*timestamp*), ο τύπος του (*EVENT, INSTANCE*), το περιεχόμενο του μηνύματος (όνομα της μεθόδου και της κλάσης, τιμές/υποστάσεις μεταβλητών), το/α αντικείμενο/α που σχετίζεται με την εγγραφή, άλλες παράμετροι που αφορούν την εγγραφή και ενδεχομένως τη διεύθυνση IP (ανάλογα με τις ανάγκες). Ο μορφότυπος των εγγραφών που χρησιμοποιούνται στην υλοποίηση είναι ο εξής:

```
Time:<ms από 01/01/1970> Type:<τύπος> Object:<αντικείμενο>
Message:<μήνυμα> Arguments:<παράμετροι> [IP:<IP address>]
```

Ανάλογα με την οπτική και το επίπεδο της ανάλυσης που θέλουμε να πραγματοποιήσουμε καθορίζουμε το ποιες μέθοδοι, κλάσεις, δομοστοιχεία ή μεταβλητές παρακολουθούνται. Για παράδειγμα, αν εξετάζονται σενάρια που συμβαίνουν μεταξύ διαφορετικών δομοστοιχείων λογισμικού τότε σίγουρα πρέπει να καταγραφούν οι κλήσεις και οι επιστροφές των μεθόδων διεπαφής και ενδεχομένως όχι η εσωτερική, ανά δομοστοιχείο, επικοινωνία –που η σημασιολογία της μπορεί να μην είναι καν γνωστή. Αντίστοιχα, αν το ζητούμενο της ανάλυσης είναι ο εντοπισμός ενός εσωτερικού προβλήματος που εμφανίζει κάποιο δομοστοιχείο ή ακόμα και κάποια κλάση, τότε η καταγραφή πρέπει ανάγεται σε «χαμηλότερο» επίπεδο, στοχεύοντας στις κλήσεις των μεθόδων εντός του δομοστοιχείου ή της κλάσης.

Δεύτερο στοιχείο: Παραγωγή και συλλογή δεδομένων παρακολούθησης

Για την παραγωγή των δεδομένων που χαρακτηρίζουν γεγονότα ενδιαφέροντος επιλέχθηκε η τεχνική ενορχήστρωσης του Java bytecode (*ByteCode Instrumentation - BCI*). Η επιλογή της BCI τεχνικής έγινε γιατί μέσω αυτής καταφέρνουμε να ενορχηστρώσουμε λογισμικό γραμμένο σε Java με ενιαίο τρόπο, σύμφωνα με πολιτικές καταγραφής και παρακολούθησης που έχουμε ορίσει και χωρίς να επέμβουμε στον πηγαίο κώδικα των κλάσεων -ο οποίος ενδεχομένως να μην είναι καν διαθέσιμος. Πληροφορίες σχετικές με το θεωρητικό κομμάτι της BCI τεχνικής παρέχονται στο άρθρο [Aar05]. Στη συνέχεια παρουσιάζεται η τεχνική και το εργαλείο δημιουργίας κατάλληλων probe για την παρακολούθηση εφαρμογών λογισμικού.

Το TPTP έργο για την πλατφόρμα του Eclipse προσφέρει μια πληθώρα επιλογών ως προς την έλεγχο και την παρακολούθηση επίδοσης των συστημάτων. Στα πλαίσια του TPTP εντάσσεται το Probekit που είναι μια μηχανή BCI η οποία διαθέτει μια φιλική προς το χρήστη γραφική διεπαφή. Μέσω της γραφικής αυτής διεπαφής είναι δυνατός ο προσδιορισμός του τύπου, των τμημάτων κώδικα και των φίλτρων ενός probe. Όπως έχει αναφερθεί και προηγουμένως (βλ. 3.1) τα probe είναι επαναχρησιμοποιήσιμα τμήματα κώδικα (Java στην προκειμένη περίπτωση) που γράφονται για τη συλλογή λεπτομερών πληροφοριών κατά το χρόνο εκτέλεσης σχετικών με τα αντικείμενα ενός προγράμματος, τις τιμές/υποστάσεις μεταβλητών όπως και για παραμέτρους και εξαιρέσεις. Το Probekit πρακτικά παρέχει ένα πλαίσιο στην πλατφόρμα του Eclipse για τη δημιουργία και τη χρήση των probe.

Τα probe που μπορούν να δημιουργηθούν στο Probekit χωρίζονται σε δύο βασικές κατηγορίες: probe μεθόδου (*method probe*) και probe σημείου κλήσης (*callsite probe*). Τα

probe μεθόδου εισάγονται στο σώμα της στοχευόμενης μεθόδου ενώ τα probes σημείου κλήσης εισάγονται σε κάθε μέθοδο που καλεί άλλες μεθόδους. Το είδος ενός probe ρυθμίζεται από τον τύπο των τμημάτων που περιλαμβάνει. Για τους σκοπούς της παρακολούθησης, όπως προσεγγίζονται από την παρούσα εργασία, μπορούν να χρησιμοποιηθούν τόσο probe μεθόδου όσο και probe σημείου κλήσης και η επιλογή εξαρτάται από το είδος της ανάλυσης που θέλει κάποιος να πραγματοποιήσει.

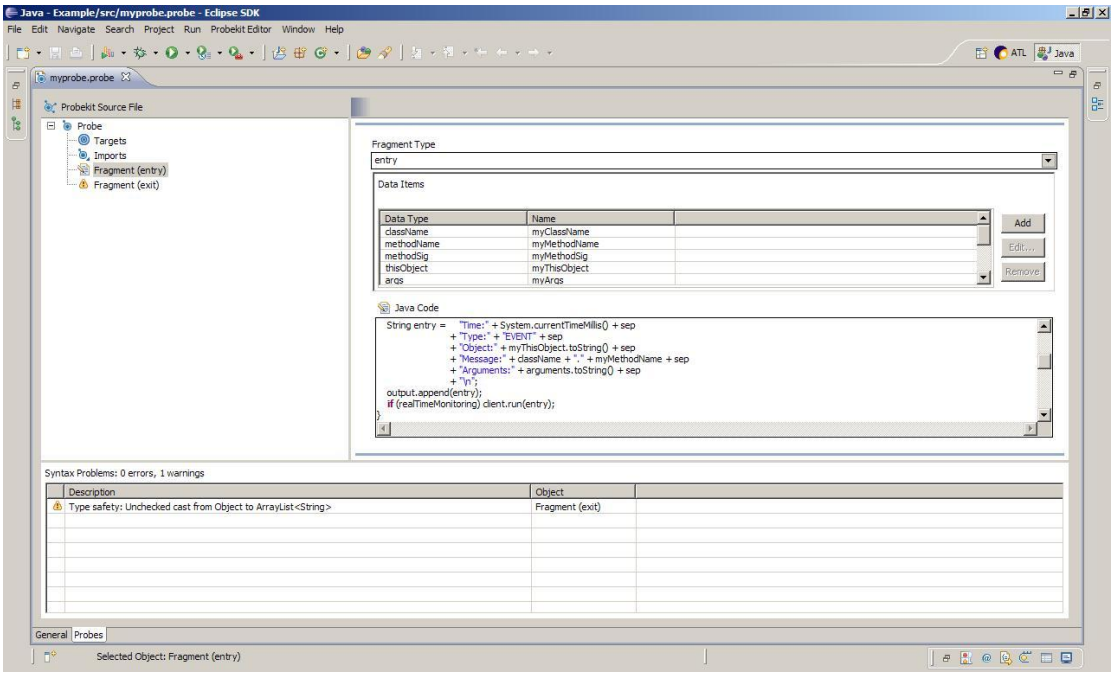
Τα τμήματα κώδικα ενός probe περιλαμβάνουν:

1. Μια περιγραφή του τύπου του τμήματος. Για παράδειγμα ένα τμήμα «εισόδου», τρέχει τη στιγμή εισόδου σε μια μέθοδο. Άλλοι τύποι τμημάτων τρέχουν τη στιγμή εξόδου από μέθοδο, τη στιγμή χειρισμού εξαίρεσης, πριν από τον πρωτογενή κώδικα στον στατικό initializer της κλάσης, τη στιγμή της κλήσης μιας κλάσης (στο σημείο κλήσης –callsite probe) και σε ορισμένες ακόμη περιπτώσεις.
2. Στοιχεία δεδομένων στα οποία έχει πρόσβαση το τμήμα. Τέτοια στοιχεία είναι το όνομα πακέτου, κλάσης, μεθόδου, αναγνωριστικό αντικειμένου, παράμεροι κλήσης, τιμές επιστροφής και άλλα.
3. Ένα (σχετικά μικρό) κομμάτι κώδικα Java που ορίζει τη λογική του probe. Για παράδειγμα εκτύπωση ενός μηνύματος, αποθήκευση σε κάποιο αρχείο κ.ά.

Επίσης, μια επιλογή που προσφέρεται από τον επεξεργαστή του Probekit είναι ο ορισμός φίλτρων, μέσω του οποίου ο χρήστης μπορεί να αποκλείσει (ή αντίστοιχα να συμπεριλάβει) μεθόδους ή κλάσεις ή και συνδυασμούς αυτών.

Στο Σχήμα 5.1 παρουσιάζεται ένα probe που ορίστηκε στα πλαίσια της υλοποίησης. Συγκεκριμένα, στο πεδίο «*Fragment Type*» έχει δοθεί η τιμή «*entry*» και άρα το τμήμα που εμφανίζεται είναι τμήμα εισόδου. Στο πεδίο «*Data Items*» έχουν δηλωθεί τα στοιχεία δεδομένων που είναι διαθέσιμα στο τμήμα κώδικα και στο πεδίο «*Java code*» δηλώνεται ο κώδικας Java που εκτελείται κατά την είσοδο σε μεθόδους. Πρέπει επίσης να σημειωθεί με την επιλογή «*Probe*» του αριστερού μενού εμφανίζεται ένα πεδίο στο οποίο μπορεί να περιληφθεί κώδικας Java με σκοπιά κλάσης («*Java code for Fragment At Class Scope*»). Στο πεδίο αυτό μπορούν να δηλωθούν (static) μέθοδοι, ιδιοχαρακτηριστικά ή και κλάσεις ώστε να χρησιμοποιούνται από τα τμήματα.

Η εισαγωγή των probes, μπορεί να γίνει είτε στατικά (μόνιμα), είτε δυναμικά κατά την εκτέλεση στα πλαίσια του TPTP. Περισσότερες πληροφορίες για τον ορισμό probe και τη χρήση του Probekit υπάρχουν στο [Probe].



Σχήμα 5.1: Δημιουργία probe στον επεξεργαστή του Probekit

Αν και στην υλοποίηση χρησιμοποιήθηκε η BCI τεχνική για Java εφαρμογές, η προσέγγιση ανάλυσης της λειτουργίας συστημάτων λογισμικού που γίνεται στον παρόντα τόμο είναι γενική και μπορεί να χρησιμοποιηθεί για συστήματα, εφαρμογές ή δομοστοιχεία λογισμικού ανεξαρτήτως της γλώσσας υλοποίησης, εφόσον χρησιμοποιηθεί κατάλληλη τεχνική συλλογής γεγονότων που σχετίζονται με την επικοινωνία των δομοστοιχείων ενός συστήματος. Συγκεκριμένα επειδή τα γεγονότα που ενδιαφέρουν είναι οι κλήσεις (και ενδεχομένως οι επιστροφές μεθόδων μέσω των οποίων μπορούν να συλλεχθούν τιμές/υποστάσεις μεταβλητών), για λογισμικό που δεν είναι γραμμένο σε Java μπορεί να χρησιμοποιηθεί μια ποικιλία τεχνικών, πολλές από τις οποίες περιγράφτηκαν στην ενότητα 2.2. Αναφορικά στις τεχνικές αυτές εντάσσονται η ενορχήστρωση κώδικα παρακολούθησης η οποία στη συγκεκριμένη περίπτωση μπορεί να αυτοματοποιηθεί, η ενορχήστρωση σε επίπεδο κώδικα μηχανής, η χρήση κατάλληλου μεταγλωττιστή και η χρήση κάποιου tracer που πιθανόν να είναι ήδη διαθέσιμος από το περιβάλλον.

Στο σημείο αυτό, αξίζει να σημειωθεί ότι σύμφωνα με το [TPTP] σύντομα θα προστεθεί η δυνατότητα συλλογής ιχνών εκτέλεσης (*tracing*) και για λογισμικό υλοποιημένο σε C/C++.

Τρίτο στοιχείο: Αποθήκευση / διαχείριση δεδομένων καταγραφής

Τα γεγονότα αφού συλλεχθούν, είτε αποθηκεύονται σε αρχεία καταγραφής (*log/trace files*) για μελλοντική επεξεργασία είτε προωθούνται με σύνδρομο (*online*) τρόπο στον εξυπηρετητή του εργαλείου ελέγχου που υλοποιήθηκε για την ανάλυση των ακολουθιών σε πραγματικό

χρόνο. Περισσότερες λεπτομέρειες για την προώθηση των γεγονότων θα δοθούν στην επόμενη ενότητα.

Σύμφωνα μετα παραπάνω, μπορεί να προταθεί ένα σύνολο δράσεων για την αποτελεσματική παρακολούθηση της συνολική συμπεριφοράς ενός συστήματος λογισμικού. Αναφορικά, ξεκινώντας από την κατηγοριοποίηση των σεναρίων την εμφάνιση των οποίων θέλουμε να ελέγξουμε, σε επίπεδα κρισιμότητας, μπορούμε να ρυθμίσουμε ανάλογα το επίπεδο καταγραφής των δομοστοιχείων και των κλάσεων που συμμετέχουν σε αυτά. Όταν, για παράδειγμα, το σενάριο που θέλουμε να ελέγξουμε είναι ιδιαίτερα κρίσιμο για την αξιοπιστία, την ασφάλεια ή την επίδοση του συστήματος, τότε μπορούμε να χρησιμοποιήσουμε την επιγραμμική προσέγγιση ώστε η ανάλυση και τελικά η ενημέρωση να είναι -όσο το δυνατόν- πιο έγκαιρη. Τέτοια αντιμετώπιση θα μπορούσαν να έχουν κάποια σενάρια πρόληψης ώστε μέσω κατάλληλων ειδοποιήσεων τελικά να ελέγχονται οι συνθήκες εκτέλεσης και να αποτρέπονται προβλήματα που θα μπορούσαν να έχουν καταστροφικές συνέπειες για το σύστημα. Αντίστοιχα, όταν θέλουμε να ελέγξουμε, για παράδειγμα, σενάρια για την επιθεώρηση (*audit*) και γενικότερα για την εξέταση λειτουργικών χαρακτηριστικών του συστήματος, μπορούμε να χρησιμοποιήσουμε τα αρχεία καταγραφής.

5.2 Τεχνική ανάλυσης ακολουθιών γεγονότων

Στην ενότητα αυτή θα περιγραφεί η τεχνική που ακολουθείται για την ανάλυση των δεδομένων που έχουν συλλεχθεί κατά την παρακολούθηση. Συγκεκριμένα θα οριστεί το μοντέλο που ελέγχεται και θα παρουσιαστεί η διαδικασία και ο αλγόριθμος με τον οποίο διαπιστώνεται η εμφάνιση ή η απόρριψη ενός σεναρίου βάση ενός συνόλου καταγραφών. Θα παρουσιαστεί επίσης η εφαρμογή της διαδικασίας για την ανάλυση αρχείων καταγραφών (*log file analysis*) η οποία χρησιμοποιείται συνήθως ως αντιδραστική (*reactive*) τακτική για τον εντοπισμό προβλημάτων, παρεκκλίσεων κλπ. Τέλος, θα προταθεί και θα περιγραφεί η υλοποίηση μιας αρχιτεκτονικής για την επιγραμμική ανάλυση ακολουθιών γεγονότων η οποία μπορεί τελικά να χρησιμοποιηθεί για την αποστολή άμεσων (προ)ειδοποιήσεων ή/και την ενεργοποίηση (προληπτικών ή μη) δράσεων.

Στη φάση αυτή θεωρούμε ότι έχει πλέον μετασχηματιστεί ένα Διάγραμμα Ακολουθίας που αποτυπώνει κάποιο σενάριο ενδιαφέροντος, σε ένα Δίκτυο Petri και το τελευταίο έχει εκφραστεί σε PNML μορφή. Επίσης είναι διαθέσιμες οι καταγραφές που συλλέχθηκαν κατά τη λειτουργία του συστήματος οι οποίες συμμορφώνονται σε ένα συγκεκριμένο μορφότυπο.

Το ερώτημα που τίθεται είναι αν υπάρχει περίπτωση το Δίκτυο Petri ξεκινώντας, από μια δεδομένη αρχική κατάσταση και εκτελούμενο βάση των καταγραφών να βρεθεί σε τελική κατάσταση. Αν αυτό συμβεί τότε η εμφάνιση του σεναρίου επαληθεύεται. Σύμφωνα με τους κανόνες μετασχηματισμού που υλοποιήθηκαν, για το Δίκτυο Petri υπάρχει μόνο ένας χαρακτηρισμός (marking) που να αντιστοιχεί σε τελική κατάσταση και είναι η κατάσταση στην οποία η τελική θέση του δικτύου φέρει ένα token ενώ όλες οι υπόλοιπες θέσεις είναι κενές –και βέβαια, κατά συνέπεια, δεν υπάρχουν ενεργοποιημένες μεταβάσεις. Ως προς την ακολουθία των γεγονότων, η τελική κατάσταση μπορεί να οριστεί ανάλογα με την ζητούμενη «αυστηρότητα» της ταύτισης. Για παράδειγμα, ενδεχομένως η ακολουθία των γεγονότων που τροφοδοτείται να πρέπει να ολοκληρώνεται με την ταύτιση του τελευταίου τμήματος του σεναρίου –στην περίπτωση αυτή τελική κατάσταση για την ακολουθία είναι το να μην υπάρχει επόμενο γεγονός προς ανάγνωση.

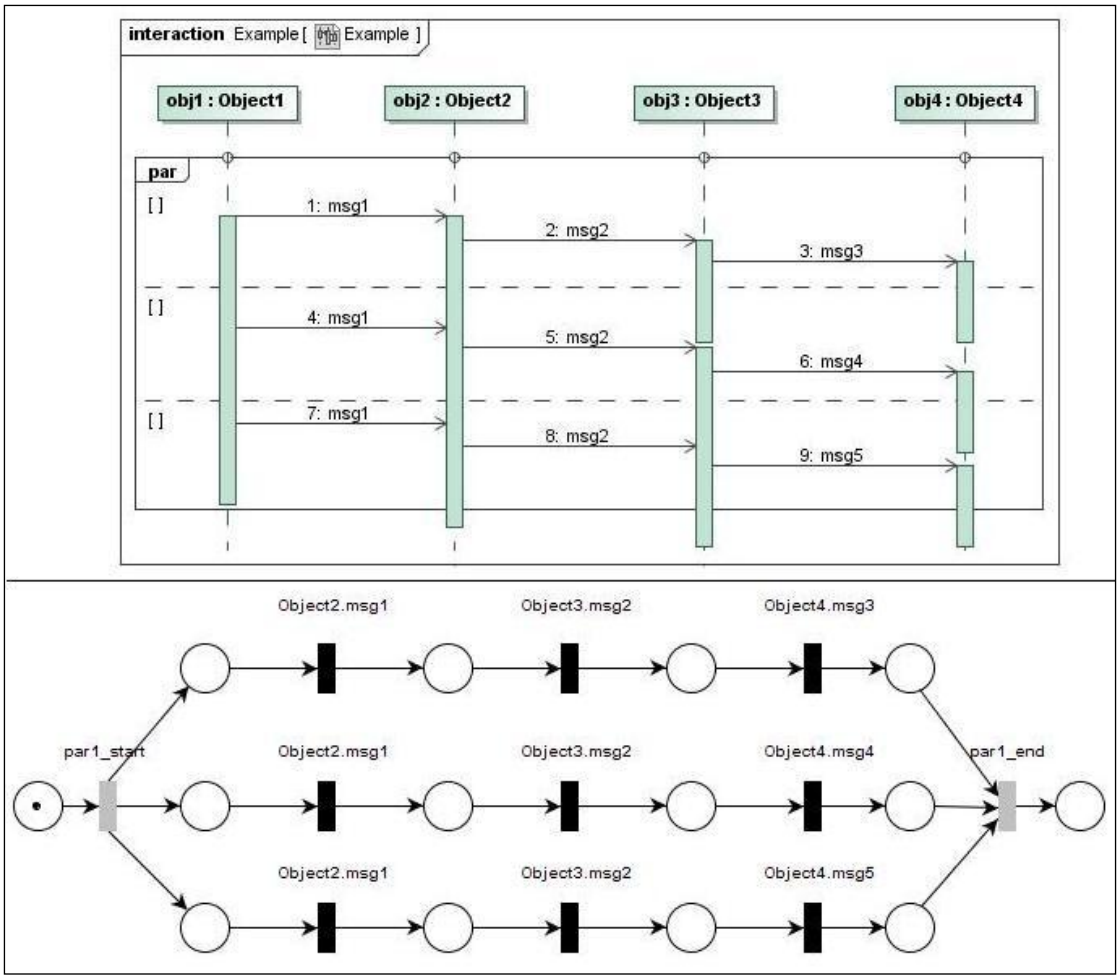
Γενικά, μπορούμε να ορίσουμε ένα μοντέλο *M* που αποτελείται από ένα Δίκτυο Petri (N, M_0) και μια αλληλουχία μερικώς διατεταγμένων γεγονότων *L* το οποίο μπορεί να βρεθεί σε ένα σύνολο καταστάσεων *S* μέσω συγκεκριμένων μεταβάσεων *T*. Για το μοντέλο αυτό έστω ότι τίθεται η συνθήκη *p* που είναι η απόρριψη του σεναρίου. Η τεχνική ελέγχου καλείται να διασχίσει τον χώρο καταστάσεων αναζητώντας μια κατάσταση στην οποία ισχύει η $\neg p$.

Σύμφωνα με όσα έχουν αναφερθεί παραπάνω, η διαδικασία ανάλυσης των καταγραφών στηρίζεται στην εκτέλεση του Δικτύου Petri βάσει της αλληλουχίας των καταγεγραμμένων γεγονότων. Η εκτέλεση ακολουθεί ορισμένους βασικούς κανόνες:

Για να εκτελεστεί μια (απλή) μετάβαση πρέπει να είναι ενεργοποιημένη στο επίπεδο του δικτύου (δηλαδή να καλύπτονται οι προϋποθέσεις της) και επιπλέον να επαληθεύεται η συνθήκη ταύτισης που στη γενική περίπτωση είναι η ταύτιση της τρέχουσας εγγραφής με το μήνυμα που έχει αντιστοιχηθεί στη μετάβαση. Στη συνέχεια της ενότητας συζητούνται και ορισμένες άλλες συνθήκες ταύτισης που μπορούν να χρησιμοποιηθούν.

Για να εκτελεστεί μια φυλασσόμενη μετάβαση πρέπει να είναι ενεργοποιημένη στο επίπεδο του δικτύου και επιπλέον η αποτίμηση της συνθήκης που φέρει να επαληθεύεται. Για να είναι δυνατή η αποτίμηση αυτή απαιτούνται οι τιμές/υποστάσεις των μεταβλητών που περιλαμβάνονται στην έκφραση φύλαξης. Οι τιμές αυτές μπορούν να αποκτηθούν είτε μέσω καταγραφών, παρακολουθώντας δηλαδή πέρα από κλήσεις, συγκεκριμένες μεταβλητές, είτε διαδραστικά από το χρήστη (π.χ. για συνθήκες σε φυσική γλώσσα), είτε μέσω εναλλακτικών οδών που συζητούνται στη συνέχεια της ενότητας. Όπως αναμένεται, η εκτέλεση των φυλασσόμενων μεταβάσεων με εξορισμού αληθή συνθήκη (TBD μεταβάσεις) είναι άμεση, εφόσον ενεργοποιηθούν στο επίπεδο του δικτύου.

Όπως έχει περιγραφεί σε προηγούμενο κεφάλαιο, κάθε μετάβαση του Δικτύου Petri αντιστοιχεί σε ένα γεγονός, δηλαδή σε μια εμφάνιση μηνύματος. Αυτό σημαίνει ότι σε ένα σενάριο είναι αποδεκτό να υπάρχουν διαφορετικές εμφανίσεις του ίδιου, σε περιεχόμενο, μηνύματος (δηλαδή κλήση της ίδιας μεθόδου του ίδιου αντικειμένου), ή αλλιώς είναι αποδεκτό να υπάρχουν γεγονότα που φέρουν τις ίδιες συνθήκες ταύτισης. Το γεγονός αυτό, σε συνδυασμό με τη μερική διάταξη των εμφανίσεων, δηλαδή τη δυνατότητα ταυτόχρονης ενεργοποίησης και διαπλοκής των γεγονότων μέσω συνδυασμένων τμημάτων με χειριστή par (σε ότι αφορά τους κανόνες που έχουν υλοποιηθεί), προκαλεί μια έκρηξη του πλήθους των «μονοπατιών» εκτέλεσης που πρέπει να εξεταστούν. Για κάθε βήμα της εκτέλεσης στο οποίο υπάρχουν n μεταβάσεις που είναι ενεργοποιημένες και για τις οποίες επαληθεύεται η συνθήκη ταύτισης (δηλαδή, στη γενική περίπτωση, αφορούν στο ίδιο όνομα μηνύματος) υπάρχουν *n!* διατάξεις ως προς την εκτέλεση των *n* αυτών. Για την επιλογή της μετάβασης που ενδεχομένως οδηγεί σε επαλήθευση του σεναρίου, πρέπει να εξεταστούν εξαντλητικά οι δυνατές αλληλουχίες εκτέλεσης του δικτύου που ξεκινούν από κάθε μετάβαση έως ότου, είτε εκπληρωθεί η συνθήκη επαλήθευσης, είτε εξεταστούν όλες οι αλληλουχίες χωρίς το μοντέλο να βρεθεί σε τελική κατάσταση. Όπως γίνεται σαφές, το φώλιασμα αντίστοιχων συνθηκών στο εσωτερικό των επιμέρους εκτελέσεων («μονοπατιών») που εξετάζονται δρα αυξητικά στην πολυπλοκότητα της εκτέλεσης. Στο Σχήμα 5.2 παρουσιάζεται με ένα απλό παράδειγμα το παραπάνω πρόβλημα.



Σχήμα 5.2: Παράδειγμα αύξησης χώρου καταστάσεων λόγω διαπλοκής γεγονότων

Ο έλεγχος εμφάνισης του σεναρίου που παρουσιάζεται στο Σχήμα 5.2 βάσει μιας ακολουθίας γεγονότων που αναδεικνύει το παραπάνω ζήτημα, θα παρουσιαστεί στην ενότητα των παραδειγμάτων (βλ. 5.3).

Η συνθήκη ταύτισης, όπως ήδη αναφέρθηκε, στη γενική περίπτωση αφορά στο όνομα της μετάβασης και μόνο. Με αυτό το τρόπο αυξάνεται η γενικότητα της διαδικασίας και διατηρείται σε χαμηλό επίπεδο η απαίτηση πληροφοριών από τις καταγραφές.

Όταν τηρούνται στις εγγραφές πληροφορίες που αφορούν στην υπόσταση του αντικειμένου που έκανε την κλήση και στην υπόσταση του αντικειμένου που έλαβε την κλήση, το ιδιοχαρακτηριστικό «ετικέτα» που αποδίδεται κατά το μετασχηματισμό στις μεταβάσεις, μπορεί να χρησιμοποιηθεί για να γίνει πιστή ανάλυση ως προς τις πληροφορίες που περιλαμβάνει από το Διάγραμμα Ακολουθίας για το σενάριο αλλά και για να βελτιωθεί η επίδοση της ανάλυσης. Η ετικέτα των μεταβάσεων περιλαμβάνει τα ονόματα των αντικειμένων που επικοινωνούν (όπως προσδιορίστηκαν κατά το σχεδιασμό, για παράδειγμα «obj1->obj2»). Στην περίπτωση αυτή, πρέπει να δημιουργηθεί μια 1-1 αντιστοίχιση μεταξύ

αυτών των ονομάτων και των αναγνωριστικών των υποστάσεων που υπάρχουν στις καταγραφές. Με το τρόπο αυτό λαμβάνονται υπόψη μόνο οι υποστάσεις που ενδιαφέρουν. Τελικά η συνθήκη ταύτισης πέρα από το όνομα του μηνύματος, απαιτεί για την επαλήθευσή της την ταύτιση, μέσω των αντιστοιχίσεων, και των αντικειμένων αποστολής και λήψης. Αυτό σημαίνει ότι «ομώνυμα» γεγονότα ενδέχεται να μην προκαλούν αύξηση των «μονοπατιών» που πρέπει να εξεταστούν. Βέβαια, αν οι υποστάσεις που ενδιαφέρουν δεν είναι συγκεκριμένες τότε ο έλεγχος μπορεί να γίνει σε επίπεδο κλάσεων, χρησιμοποιώντας ως επιπλέον στοιχείο ταύτισης το όνομα της κλάσης που αποστέλλει το μήνυμα.

Η επιλογή της συνθήκης ταύτισης έχει να κάνει με τις διαθέσιμες πληροφορίες των καταγραφών και με το πόσο αυστηρή πρέπει να είναι η ταύτιση καθαυτή. Αν δηλαδή δεν έχει σημασία ποια υπόσταση εκτελεί μια λειτουργία, ή δεν υπάρχει καν υπόσταση αντικειμένου (π.χ. κλήση μιας static μεθόδου κάποια κλάσεις) τότε δε πρέπει να ληφθεί υπόψη η αντιστοίχιση πληροφορία.

Στην υλοποίηση, η συνθήκη ταύτισης περιλαμβάνει το μήνυμα και αντικείμενο λήψης μιας κλήσης, ως συμβιβασμός των παραπάνω. Παρόλα αυτά, η τροποποίηση της συνθήκης μπορεί να γίνει με απλό τρόπο. Εκείνο που πρέπει να σημειωθεί είναι ότι όταν είναι επιθυμητό να λαμβάνεται υπόψη το αντικείμενο αποστολής στη συνθήκη ταύτισης, τότε πρέπει να χρησιμοποιείται probe σημείου κλήσης (callsite probe).

Στη συνέχεια παρουσιάζεται ο αλγόριθμος που χρησιμοποιείται για τον έλεγχο του μοντέλου που συζητήθηκε προηγουμένως. Κατόπιν, αποτυπώνεται σε ψευδοκώδικα και σχολιάζεται η υλοποίησή του ως επέκταση του εργαλείου που επιλέχθηκε.

Η αρχική κατάσταση του Δικτύου Petri, σύμφωνα με τους κανόνες αντιστοίχησης που έχουν υλοποιηθεί, περιλαμβάνει ένα token στην αρχική θέση του δικτύου και όλες οι υπόλοιπες θέσεις είναι κενές. Μάλιστα ο μόνος από τους υπόλοιπους κανόνες που απαιτεί την ύπαρξη «αρχικών συνθηκών» είναι εκείνος του loop με κατώτατο ή/και ανώτατο όριο επαναλήψεων, όπου και πάλι βέβαια ορίζεται μια μοναδική αρχική κατάσταση για το δίκτυο.

Η αρχική κατάσταση του μοντέλου *M* συντίθεται από την αρχική κατάσταση του Δικτύου Petri και την αρχική θέση ανάγνωσης στην ακολουθία των γεγονότων που είναι πριν το πρώτο γεγονός.

Για τον έλεγχο του μοντέλου ορίζουμε μια μέθοδο με παραμέτρους εισόδου: το Δίκτυο Petri στην τρέχουσα κατάστασή του, το σημείο επόμενης ανάγνωσης από τις καταγραφές και μια μετάβαση προς εκτέλεση που έχει αντιστοιχηθεί σε καταγεγραμμένο σε γεγονός. Με άλλα λόγια, σε κάθε κλήση της μεθόδου (βήμα) εκτελείται η μετάβαση που επιλέχθηκε από το προηγούμενο βήμα και επιλέγεται η επόμενη. Κατά συνέπεια στην πρώτη κλήση εισάγεται η αρχική κατάσταση του μοντέλου και η παράμετρος της μετάβασης προς εκτέλεση είναι κενή.

Αν κατά την πορεία εκτέλεσης το δίκτυο βρεθεί σε τελική κατάσταση τότε η μέθοδος επιστρέφει τιμή «true», διαφορετικά επιστρέφει «false». Η μέθοδος καλείται αναδρομικά έτσι ώστε να εξετάζεται εξαντλητικά το ενδεχόμενο επαλήθευσης του σεναρίου.

Συγκεκριμένα, αρχικά εισάγεται η τρέχουσα κατάσταση σε μια στοίβα. Αν η παράμετρος της προς εκτέλεση μετάβασης δεν είναι κενή, τότε η μετάβαση εκτελείται.

Στη συνέχεια εξετάζονται οι ενεργοποιημένες μεταβάσεις. Όσο υπάρχουν ενεργοποιημένες φυλασσόμενες μεταβάσεις, αρχικά εκτελούνται οι TBD κι έπειτα ελέγχονται οι υπόλοιπες φυλασσόμενες μεταβάσεις ανά δομή επιλογής. Ο έλεγχος ανά δομή επιλογής, δηλαδή ανά δομή του Δικτύου Petri που προήλθε από συνδυασμένο τμήμα με χειριστή alt, opt, ή loop διευκολύνεται με τη χρήση ενός ιδιοχαρακτηριστικού («*owner*») που έχει αποδοθεί στις φυλασσόμενες μεταβάσεις κατά το μετασχηματισμό και φέρει ένα μοναδικό αναγνωριστικό της δομής στην οποία μετέχουν. Κατά τον έλεγχο αποτιμώνται οι συνθήκες φύλαξης αναζητώντας πληροφορίες για τις τιμές των μεταβλητών που συμμετέχουν στις εκφράσεις (είτε από τις καταγραφές, είτε με εναλλακτικούς τρόπους που παρουσιάζονται στη συνέχεια). Τελικά για κάθε δομή εκτελείται μια φυλασσόμενη μετάβαση. Οι εκτελέσεις φυλασσόμενων μεταβάσεων διαφορετικών δομών είναι ανεξάρτητες, δηλαδή δεν έχει σημασία η σειρά με την οποία συμβαίνουν. Κατόπιν, ελέγχεται η συνθήκη τερματισμού και αν ισχύει τότε το σενάριο επαληθεύεται από τις καταγραφές και η μέθοδος επιστρέφει true.

Διαφορετικά διαβάζεται η επόμενη εγγραφή από την ακολουθία των γεγονότων σύμφωνα με τον δείκτη ανάγνωσης στην ακολουθία γεγονότων. Αν οι καταγραφές περιλαμβάνουν θόρυβο τότε η ακολουθία διατρέχεται έως ότι βρεθεί γεγονός ενδιαφέροντος –ο χειρισμός του θορύβου μπορεί να είναι και διαφορετικός, ανάλογα με το πόσο αυστηρή πρέπει να είναι η ταύτιση των καταγραφών ως προς τα σενάρια. Στην περίπτωση που δε βρεθεί γεγονός ενδιαφέροντος η μέθοδος επιστρέφει false. Ακολούθως, ελέγχεται κάθε μια από τις ενεργοποιημένες μεταβάσεις ως προς τη συνθήκη ταύτισης. Για κάθε ενεργοποιημένη μετάβαση που πληροί τη συνθήκη ταύτισης, διαδοχικά (ή και με τυχαίο τρόπο), καλείται αναδρομικά η μέθοδος με είσοδο την τρέχουσα κατάσταση, την επόμενη θέση ανάγνωσης και την εκάστοτε μετάβαση προς εκτέλεση. Η διαδοχή συνεχίζει όσο η εκτέλεση για κάθε μετάβαση επιστρέφει false. Στην περίπτωση που επιστραφεί true τότε υπάρχει επαλήθευση του σεναρίου οπότε η διαδοχή διακόπτεται και η μέθοδος επιστρέφει true.

Η διαδικασία που περιγράφηκε παραπάνω μπορεί αποτυπωθεί σε ψευδοκώδικα ως εξής:

```
BOOLEAN METHOD checkModel(το ΔΡ με το τρέχον marking, δείκτης ανάγνωσης στην
ακολουθίας, μετάβαση προς εκτέλεση)
```

```
PUSH ένα checkpoint της τρέχουσας κατάστασης στη στοίβα
IF η μετάβαση προς εκτέλεση δεν είναι κενή THEN
       Εκτέλεση μετάβασης
       Ενημέρωση του marking του ΔΡ
ENDIF

WHILE υπάρχουν ενεργοποιημένες φυλασσόμενες μεταβάσεις
       FOR κάθε ενεργοποιημένη TBD φυλασσόμενη μετάβαση
              Εκτέλεση μετάβασης
              Ενημέρωση του marking του ΔΡ
       ENDFOR

       FOR κάθε δομή επιλογής
              FOR κάθε ενεργοποιημένη φυλασσόμενη μετάβαση της δομής
                     IF η συνθήκη φύλαξης επαληθεύεται THEN
                            Εκτέλεση της μετάβασης
                            Διακοπή της επανάληψης (πρώτου επιπέδου)
                     ENDIF
              ENDFOR
       ENDFOR
ENDWHILE

IF η συνθήκη επαλήθευσης ισχύει THEN
       RETURN true
ENDIF

READ επόμενο γεγονός από την ακολουθίας βάση του δείκτη ανάγνωσης
FOR κάθε ενεργοποιημένη μετάβαση
       IF η συνθήκη ταύτισης με το γεγονός ισχύει THEN
                     result =  checkModel(ΔΡ με τρέχον marking,
                                          επόμενη θέση δείκτη,
                                          τρέχουσα  μετάβαση)
                     IF result = true THEN
                            RETURN true;
                     ENDIF

                     POP ένα checkpoint από τη στοίβα καταστάσεων
                     Επαναφορά του μοντέλου
```

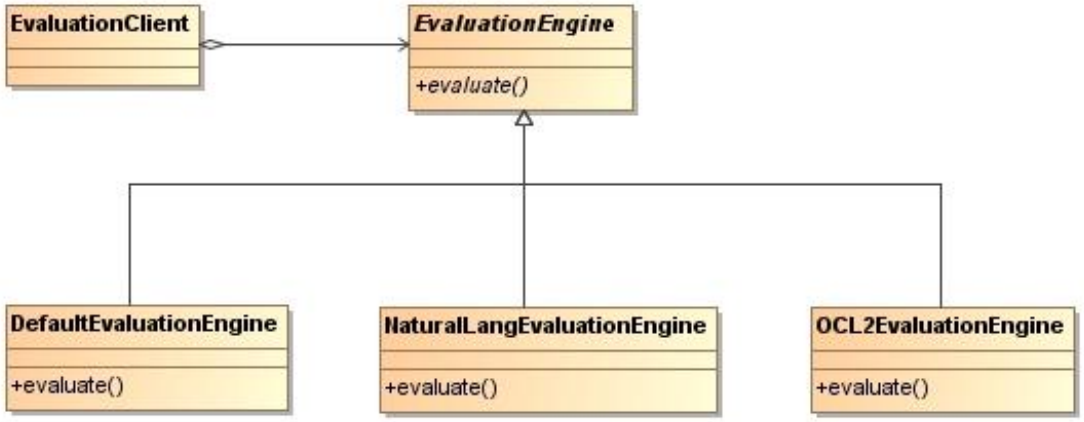
```
                        RETURN false;
                    ENDIF
        ENDFOR
        POP ένα checkpoint από τη στοίβα καταστάσεων
        Επαναφορά του μοντέλου
ENDMETHOD
```

Ο παραπάνω ψευδοκώδικας αποτυπώνει σε απλοποιημένη μορφή τη λογική του αλγορίθμου ταύτισης του προτύπου που ορίζει το Δίκτυο Petri και των ακολουθιών γεγονότων. Όπως φαίνεται παραπάνω η επαλήθευση μπορεί να προκύψει τόσο μετά από εκτέλεση φυλασσόμενης μετάβασης, όσο κι έπειτα από εκτέλεση μετάβασης που αντιστοιχεί σε γεγονός. Επίσης, θεωρείται ότι οι παράμετροι περνάνε στη μέθοδο με το μηχανισμό που ακολουθείται στη Java (κατά τιμή) και ότι η πρώτη (ΔΡ με τρέχον marking) και η τρίτη παράμετρος (μετάβαση) αφορούν σε αντικείμενα και άρα περνάνε σε κάθε κλήση αναφορές σε αυτά. Για την επαναφορά του μοντέλου χρησιμοποιείται η στοίβα των καταστάσεων στην οποία κάθε checkpoint είναι μια δομή που περιέχει τον εκάστοτε χαρακτηρισμό του ΔΡ και τον δείκτη ανάγνωσης. Κάθε φορά που η διαδικασία πρόκειται να επιστρέψει false, το μοντέλο επαναφέρεται για τη συνέχιση της ανάλυσης μέχρι την εξάντληση του ενδεχομένου επαλήθευσης. Τέλος, κάτι που δεν εμφανίζεται στην παραπάνω περιγραφή αλλά υλοποιείται είναι η συλλογή, σε μια δομή, του «μονοπατιού» που οδήγησε στην επαλήθευση.

Η υλοποίηση της τεχνικής που παρουσιάστηκε προηγουμένως έγινε επεκτείνοντας το εργαλείο *Platform Independent Petri Net Editor 2.5 (PIPE)* που παρουσιάστηκε σε προηγούμενη ενότητα (βλ. 3.2). Το εργαλείο αυτό είναι ανοιχτού κώδικα, υλοποιημένο σε Java, αναπτύχθηκε και επεκτείνεται κυρίως από μεταπτυχιακούς σπουδαστές του Imperial College στο Λονδίνο και χρησιμοποιείται από μια ευρεία ενεργή κοινότητα χρηστών και εκτός του Imperial College.

Αρχικά επεκτάθηκε ο μορφότυπος των αρχείων PNML που δέχεται το εργαλείο (προσδιορίζεται σε XSLT) έτσι ώστε να περιληφθούν οι πρόσθετες πληροφορίες που φέρουν τα Δίκτυα Petri που παράγονται από τη διαδικασία του μετασχηματισμού. Παράλληλα προστέθηκε η δυνατότητα ορισμού και χειρισμού του είδος των φυλασσόμενων μεταβάσεων. Επίσης επεκτάθηκαν οι απλές («*Immediate*») μεταβάσεις ώστε να περιλαμβάνουν τα ιδιοχαρακτηριστικά των γεγονότων που ανατίθενται κατά το μετασχηματισμό. Στο εργαλείο, τέλος, δημιουργήθηκε η υποδομή για την εναλλακτική χρήση μηχανών αποτίμησης των εκφράσεων φύλαξης των φυλασσόμενων μεταβάσεων. Η υποδομή αυτή στηρίχθηκε στην

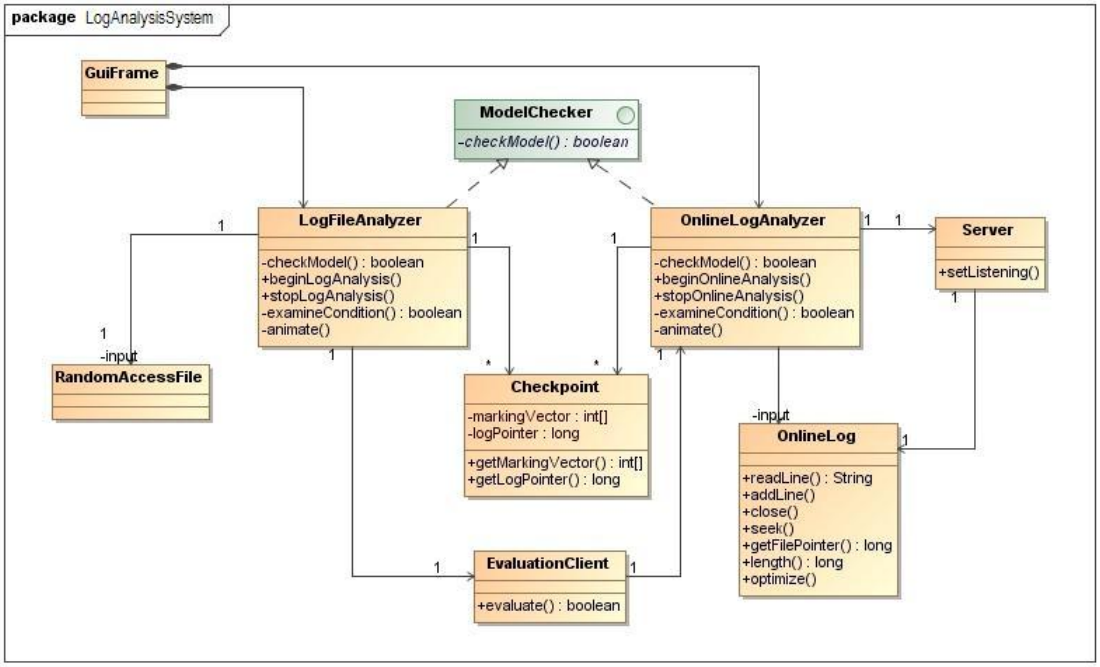
υλοποίηση του σχεδιαστικού μορφήματος (*design pattern*) *Strategy*. Το Σχήμα 5.3 απεικονίζει το διάγραμμα κλάσεων για την αποτίμηση των εκφράσεων φύλαξης.



Σχήμα 5.3: Αποτίμηση εκφράσεων φύλαξης (σχεδιαστικό μόρφημα Strategy)

Οι δυνατότητες εκτέλεσης που διαθέτει το PIPE πέρα από τη διαδραστική εκτέλεση (*token game*), είναι η τυχαία εκτέλεση κάποιας (ενεργοποιημένης) μετάβασης και η τυχαία εκτέλεση συγκεκριμένου αριθμού μεταβάσεων με ταυτόχρονη γραφική απεικόνιση. Όπως είναι σαφές, η πιο σημαντική προσθήκη που έπρεπε να γίνει, ήταν η υλοποίηση του αλγόριθμου εκτέλεσης που παρουσιάστηκε παραπάνω και που προσφέρει τη δυνατότητα εκτέλεσης των δικτύων βάσει ακολουθιών γεγονότων που τροφοδοτούνται στο εργαλείο. Δημιουργήθηκαν δύο υλοποιήσεις της διαδικασίας που διαφοροποιούνται κυρίως ως προς το πως εισάγονται τα δεδομένα προς ανάλυση στο εργαλείο ενώ ο πυρήνας του αλγορίθμου είναι ο ίδιος. Συγκεκριμένα, το εργαλείο μπορεί να χρησιμοποιηθεί τόσο για την ανάλυση ακολουθιών γεγονότων από αρχεία καταγραφών, όσο και για την επιγραμμική ανάλυση σε πραγματικό χρόνο ή σε χρόνο κοντά στον πραγματικό. Οι δύο αυτές υλοποιήσεις θα συζητηθούν στις υποενότητες που ακολουθούν.

Στο Σχήμα 5.4 παρουσιάζεται ένα Διάγραμμα Κλάσεων που απεικονίζει τη δομή του συστήματος ανάλυσης ακολουθιών γεγονότων που προστέθηκε στο PIPE. Πέρα από τις κλάσεις αυτές, προστέθηκαν νέες ή επεκτάθηκαν κλάσεις του PIPE που σχετίζονταν με την εισαγωγή και την επεξεργασία των δικτύων που χρησιμοποιούνται.



Σχήμα 5.4: Διάγραμμα κλάσεων συστήματος ανάλυσης ακολουθιών γεγονότων

5.2.1 Ανάλυση αποθηκευμένων καταγραφών

Τα γεγονότα που καταγράφονται κατά την εκτέλεση του λογισμικού συνήθως αποθηκεύονται είτε με μορφή αρχείων καταγραφής είτε σε εξειδικευμένες βάσεις δεδομένων. Κάθε μια από τις επιλογές αυτές εμφανίζει διαφορετικές απαιτήσεις πόρων για το σύστημα. Στην περίπτωση των αρχείων καταγραφής η διαδικασία της αποθήκευσης είναι απλή, χωρίς να απαιτείται κάποιο ιδιαίτερο σύστημα όπως απαιτείται στην περίπτωση των βάσεων δεδομένων. Οι βάσεις δεδομένων καταγραφής όμως προσφέρουν σημαντικές διευκολύνσεις. Για παράδειγμα, πολλά φαινόμενα/προβλήματα κατά την εκτέλεση, μπορούν να εντοπιστούν και να εξεταστούν με απλές υποβολές SQL ερωτημάτων στη βάση. Η επιλογή που κάνει ο σχεδιαστής ή ο διαχειριστής ενός συστήματος αποτελεί συμβιβασμό των παραπάνω χαρακτηριστικών αλλά και άλλων παραγόντων που σχετίζονται με εξειδικευμένο τρόπο με το σύστημα που διαχειρίζεται (μέγεθος, πλήθος καταγραφών προς καταγραφή, διαθέσιμοι πόροι, ανάγκες διαχείρισης και ανάλυσης καταγραφών).

Η χρήση των αποθηκευμένων καταγραφών είναι «αντιδραστική» (*reactive*). Επιστρατεύονται δηλαδή για τη διερεύνηση της συμπεριφοράς που εμφάνισε το σύστημα κατά τη λειτουργία του και μπορεί να αφορά στον εντοπισμό ενός προβλήματος και της αιτίας που το προκαλεί, στην τήρηση συγκεκριμένων απαιτήσεων ή στον εντοπισμό εισβολών/επιθέσεων. Όπως είναι σαφές, η δυνατότητα αυτή είναι κρίσιμη για την επιθεώρηση, την τεκμηρίωση και τη βελτίωση των λειτουργιών οποιουδήποτε οργανισμού ή εταιρίας.

Στην υλοποίηση του πλαισίου χρησιμοποιήθηκαν αρχεία καταγραφής ως μέσο αποθήκευσης. Παρόλα αυτά, με ορισμένες τροποποιήσεις ως προς τη μέθοδο ανάκτησης των γεγονότων, η ανάλυση μπορεί να υλοποιηθεί και για δεδομένα καταγραφής προερχόμενα από βάσεις δεδομένων.

Όταν χρησιμοποιούνται αρχεία καταγραφής, ένα πρώτο ζήτημα είναι το που δημιουργούνται και το που τηρούνται τα αρχεία αυτά. Ειδικά στα κατανεμημένα συστήματα, το ζήτημα αυτό παρουσιάζει ιδιαίτερο ενδιαφέρον. Για παράδειγμα, τα γεγονότα θα μπορούσαν μετά τη γέννησή τους να αποστέλλονται σε ένα συγκεκριμένο σημείο συγκέντρωσης το οποίο θα αναλάμβανε την ταξινόμησή τους σε ένα ή περισσότερα αρχεία. Αυτό σαφώς έχει κόστος σε πόρους και ειδικά σε δικτυακό φορτίο. Εναλλακτικά, θα μπορούσαν τα αρχεία να αποθηκεύονται προσωρινά τοπικά, ενδεχομένως ανά μηχάνημα ή ανά εφαρμογή και μια batch διεργασία θα μπορούσε να αναλαμβάνει την περισυλλογή και την οργάνωσή τους, συγκεκριμένες ώρες όπου οι πόροι του συστήματος βρίσκονται σε μεγαλύτερη διαθεσιμότητα.

Στην ανάλυση θεωρείται ότι τα παραπάνω ζητήματα έχουν λυθεί και ότι υπάρχει πλέον ένα ενιαίο αρχείο που περιέχει την ακολουθία των γεγονότων που ενδιαφέρει, ταξινομημένη με αύξοντα χρόνο. Σε επίπεδο υλοποίησης, ορίστηκε για κάθε δομοστοιχείο ένα τοπικό αρχείο στο probe με το οποίο ενορχηστρώνονται οι εφαρμογές. Τα αρχεία αυτά, συγκεντρώνονται και το περιεχόμενό τους ταξινομείται βάσει της χρονοσφραγίδας κάθε εγγραφής.

Σε επίπεδο υλοποίησης, στις επιλογές εκτέλεσης του PIPE προστέθηκε η εκτέλεση σύμφωνα με αρχείο καταγραφής. Αφού εισαχθεί το Δίκτυο Petri, ο χρήστης κάνοντας την επιλογή αυτή καλείται να προσδιορίσει το αρχείο καταγραφής βάσει του οποίου επιθυμεί να ελεγχθεί το σενάριο. Για το αρχείο αυτό δημιουργείται ένα αντικείμενο *RandomAccessFile* (*java.io*) το οποίο επιτρέπει την τυχαία προσπέλαση του αρχείου βάσει ενός δείκτη (*filePointer*). Ο έλεγχος εκτελείται σύμφωνα με τον αλγόριθμο που περιγράφτηκε παραπάνω και αναλόγως με τις επιλογές των συνθηκών φύλαξης και τις μηχανές αποτίμησης, ενδεχομένως ο χρήστης να κληθεί να επιλέξει μια απάντηση (για παράδειγμα για εκφράσεις σε φυσική γλώσσα και για μηχανή που υλοποιεί την αποτίμηση ως ερώτημα στο χρήστη). Η εκτέλεση του αλγορίθμου επαναλαμβάνεται έως ότου είτε επαληθευτεί η εμφάνιση του σεναρίου, είτε διατρεχθεί πλήρως το αρχείο.

Μια ενδιαφέρουσα περίπτωση είναι να υπάρχουν στις καταγραφές γεγονότα που θα μπορούσαν να εμφανιστούν στο «μονοπάτι» επαλήθευσης, αλλά η διάταξή τους δεν επαληθεύει κάτι τέτοιο. Τότε είναι δυνατόν τα γεγονότα αυτά να αγνοούνται και η διερεύνηση να συνεχίζεται. Στην περίπτωση αυτή όμως, ο πλήρης έλεγχος επαλήθευσης προϋποθέτει ότι μετά από κάθε αποτυχία, ο αλγόριθμος επανεκκινείται με τον δείκτη

ανάγνωσης να βρίσκεται στο επόμενο κατά σειρά γεγονός που είναι δυνατόν να ξεκινά μια ακολουθία επαλήθευσης –και όχι στο γεγονός που ακολουθεί το τελευταίο που διαβάστηκε.

Για παράδειγμα έστω ότι η ακολουθία γεγονότων στο αρχείο καταγραφής είναι η εξής: *a*;*b*;*c*;*a*;*b*;*c*;*a*;*c*. Και έστω ότι το σενάριο είναι: *a*;*b*;*c*;*a*;*c*. Ξεκινώντας από την πρώτη θέση, ο αλγόριθμος θα επιστρέψει false αφού διαβάσει το δεύτερο «*b*» της ακολουθίας. Αν επανεκκινήσει την ανάγνωση από την αμέσως επόμενη θέση θα διαβάσει το «*c*» που θα το αγνοήσει/αποτύχει κι έπειτα το τρίτο «a», οπότε θα αποτύχει εκ νέου. Παρόλα αυτά, όπως μπορεί εύκολα να διαπιστωθεί, το σενάριο εμφανίζεται στην ακολουθία και ο έλεγχος πρέπει να ξεκινήσει από το δεύτερο «a». Αυτό σημαίνει ότι πρέπει να δημιουργεί μια δομή ουράς (*queue*) η οποία θα περιλαμβάνει όλες τις θέσεις των εν δυνάμει «εναρκτήριων» –για το σενάριο- γεγονότων. Τα γεγονότα αυτά μπορεί να είναι διαφορετικού είδους καθώς το πρώτο τμήμα του σεναρίου μπορεί να είναι ένα συνδυασμένο τμήμα par (φωλιασμένο σε άλλο συνδυασμένο τμήμα ή μη). Για τις περιπτώσεις των συνδυασμένων τμημάτων με φυλασσόμενους τελεστέους, πρέπει να αναζητηθούν και να αποτιμηθούν οι -πιο πρόσφατες- τιμές/υποστάσεις των μεταβλητών που περιλαμβάνονται στις συνθήκες. Για τη βελτίωση της επίδοσης η συμπλήρωση της ουράς αυτής μπορεί να γίνεται κατά τη διαδικασία εκτέλεσης του αλγορίθμου, αφού προηγηθεί η αναγνώριση των «εναρκτήριων» γεγονότων.

Κατά τη διάρκεια της ανάλυσης ενημερώνεται περιοδικά η απεικόνιση του Δικτύου Petri αλλά συνήθως λόγω της ταχύτητας δεν είναι εμφανές το «μονοπάτι» που ακολουθείται. Για το λόγο αυτό, μετά το πέρας της ανάλυσης και στην περίπτωση που το σενάριο που μοντελοποιείται από το δίκτυο επαληθευτεί, προσφέρεται στο χρήστη η δυνατότητα γραφικής αναπαραγωγής της εκτέλεσης (με δυνατότητα καθορισμού του χρονικού διαστήματος μεταξύ των βημάτων). Σε όρους της μεθόδου ελέγχου μοντέλου, δίνεται η δυνατότητα παράθεσης ενός αντιπαραδείγματος (*counterexample*).

5.2.2 Επιγραμμική ανάλυση ακολουθιών γεγονότων

Αν και η ανάλυση αποθηκευμένων καταγραφών καλύπτει ένα σημαντικό σύνολο αναγκών, όπως αναφέρθηκε και προηγουμένως η χρήση του συνήθως είναι αντιδραστική. Σε κάθε περίπτωση απαιτεί τη συγκέντρωση των δεδομένων καταγραφής σε αρχεία κι έπειτα την εισαγωγή αυτών των αρχείων στο εργαλείο ανάλυσης. Ενδεχομένως όμως να υπάρχουν σενάρια πρόληψης ή υπογραφές επιθέσεων που ο έλεγχος εμφάνισής τους είναι κρίσιμο να γίνεται άμεσα ώστε να αποτρέπονται επιπλοκές, προβλήματα ή παραβιάσεις.

Ο αλγόριθμος εκτέλεσης που παρουσιάστηκε παραπάνω μπορεί να χρησιμοποιηθεί αποτελεσματικά για αυτού του είδους την ανάλυση, εφόσον οριστεί μια κατάλληλη

αρχιτεκτονική για την αποστολή των γεγονότων κατά την εκτέλεση του λογισμικού, την περισυλλογή τους από έναν εξυπηρετητή και την τροφοδότησή τους στο εργαλείο ανάλυσης.

Η ανάλυση μπορεί να γίνει σε χρόνο κοντά στον πραγματικό. Το επίρρημα «κοντά» αφορά κατά κύριο λόγο τα κατανεμημένα συστήματα και σχετίζεται με τρεις βασικές χρονικές απαιτήσεις:

1. απαιτείται ένα «παράθυρο» χρόνου τουλάχιστον ίσο με τη χρονική απόσταση του εξυπηρετητή από το πιο απομακρυσμένο δομοστοιχείο. Το παράθυρο μπορεί να μεταβάλλεται καθώς η χρονική αυτή απόσταση είναι μεταβλητή και μπορεί να επηρεάζεται από την εμφάνιση συνθηκών συμφόρησης για το δίκτυο.
2. η εξαντλητική εξέταση των εναλλακτικών εκτελέσεων στις περιπτώσεις μεταβάσεων που είναι ενεργοποιημένες ταυτόχρονα και παράλληλα πληρούν τη συνθήκη ταύτισης, ανάλογα με την ακολουθία που τροφοδοτείται πιθανόν να εμφανίσει μεγαλύτερους χρόνους εκτέλεσης.
3. η αποτίμηση των φυλασσόμενων μεταβλητών ενδεχομένως να απαιτεί επιπλέον χρόνο ή αλληλεπίδραση με εξωτερικούς παράγοντες (τον χρήστη, άλλα συστήματα κλπ) και άρα η διαδικασία να χρειαστεί να αναμείνει.

Παρόλα αυτά οι καθυστερήσεις αυτές αντισταθμίζονται σε ένα βαθμό από το χρόνο που απαιτεί η εκτέλεση κάθε μεθόδου -ο οποίος βέβαια μπορεί να ποικίλει.

Η αρχιτεκτονική που προτείνεται ακολουθεί το μοντέλο του πελάτη-εξυπηρετητή. Συγκεκριμένα στο probe ενσωματώνεται μια πολύ απλή υλοποίηση πελάτη ο οποίος παραλαμβάνει τα γεγονότα αμέσως μετά τη γέννησή τους και τα αποστέλλει (μέσω TCP/IP) σε έναν εξυπηρετητή. Πρακτικά, κάθε δομοστοιχείο μέσω του probe αναφέρει τη δραστηριότητα που αναπτύσσει ένα τέτοιο πελάτη.

Στο σημείο της ανάλυσης, ο εξυπηρετητής τρέχει παράλληλα (ξεχωριστό νήμα) με τον ελεγκτή του μοντέλου και αναλαμβάνει να παραλάβει τα γεγονότα από τους πελάτες. Ταυτόχρονα φροντίζει να εξασφαλίσει την ορθή τροφοδότησή τους σε μια δομή (*OnlineLog*) που, πρακτικά, προσομοιώνει και επεκτείνει τη λειτουργία ενός αρχείου τυχαίας προσπέλασης. Συγκεκριμένα, το OnlineLog διαθέτει ένα δείκτη και παρέχει μια μέθοδο ανάγνωσης που επιστρέφει μια εγγραφή ανάλογα με την τρέχουσα θέση του δείκτη. Οι αναγνώσεις πραγματοποιούνται από τον ελεγκτή του μοντέλου. Αν δεν υπάρχει επόμενη εγγραφή προς ανάγνωση τότε η μέθοδος αναμένει. Ανάλογα, η κλάση OnlineLog περιλαμβάνει μια μέθοδο εγγραφής η οποία καλείται από τον εξυπηρετητή για την προσθήκη γεγονότων. Οι δύο αυτές μέθοδοι είναι συγχρονισμένες και κάθε προσθήκη ολοκληρώνεται με ειδοποίηση (*notify*) της διεργασίας που ενδεχομένως αναμένει.

Επιπλέον, η OnlineLog διαθέτει ένα μέγιστο δείκτη που έχει το ρόλο του «μεγέθους». Το μέγεθος της ακολουθίας γεγονότων που πρέπει να διατηρεί στη μνήμη το OnlineLog πρέπει να είναι τουλάχιστον με το βάθος των αναδρομικών κλήσεων. Στη περίπτωση που οι τιμές/περιστάσεις των μεταβλητών που περιλαμβάνονται σε συνθήκες φύλαξης συλλέγονται με τον ίδιο τρόπο, το ελάχιστο μέγεθος της ακολουθίας αυξάνεται ανάλογα. Σαφώς, ο θόρυβος δε χρειάζεται να συγκρατείται και επίσης, μετά από κάθε αποτυχία επαλήθευσης η δομή αρχικοποιείται.

Η παραπάνω πρόταση για επιγραμμική ανάλυση εμφανίζει σίγουρα ορισμένες αδυναμίες. Πρώτον, το λογισμικό που ενορχηστρώνεται επιβαρύνεται από την επικοινωνία που επιχειρεί ο πελάτης με τον εξυπηρετητή. Μια δεύτερη επιβάρυνση από την επικοινωνία αυτή αφορά στο φορτίο του δικτύου. Τρίτον, η μέγιστη χρονική απόσταση του εξυπηρετητή από τα ενορχηστρωμένα δομοστοιχεία ορίζει μια «καθυστέρηση» στη διαδικασία που δε μπορεί να προσπεραστεί -τουλάχιστον με τον αλγόριθμο ανάλυσης που παρουσιάστηκε παραπάνω.

Παρόλα αυτά η επιγραμμική ανάλυση μπορεί να αποδειχθεί ιδιαίτερα αποτελεσματική σε μια σειρά ρεαλιστικών χρήσεων. Οι προϋποθέσεις που πρέπει να τηρούνται στις περιπτώσεις αυτές είναι τα σενάρια να μη φέρουν πολλαπλές μεταβάσεις που μπορούν να ενεργοποιηθούν ταυτόχρονα και να πληρούν τη συνθήκη ταύτισης, η αποτίμηση των εκφράσεων φύλαξης να γίνεται αυτοματοποιημένα και όσο το δυνατόν πιο σύντομα και βέβαια τα δομοστοιχεία του υπό εξέταση συστήματος να έχουν «αποδεκτή» χρονική απόσταση ως προς τον εξυπηρετητή. Υπό αυτές τις συνθήκες τα αποτελέσματα της ανάλυσης είναι άμεσα και με την επαλήθευση ενός σεναρίου θα μπορούσε άμεσα να κινηθεί μια διαδικασία αντιμετώπισης ενός ενδεχόμενου ή πρόσφατου προβλήματος.

Είναι σαφές ότι αυτού του είδους η ανάλυση έχει μεγαλύτερο συνολικό κόστος για το σύστημα. Παρόλα αυτά θα μπορούσε να κριθεί ως κατάλληλη ειδικά για συστήματα λογισμικού κρίσιμα ως προς την ασφάλεια, στα οποία η δυνατότητα άμεσης ανάλυσης της συμπεριφοράς και έγκαιρου εντοπισμού παρεκκλίσεων ή προβλημάτων είναι ζήτημα ιδιαιτέρως σημαντικό.

5.3 Παραδείγματα

Παράδειγμα μερικής διάταξης

Μια από τις ακολουθίες γεγονότων που επαληθεύουν το σενάριο του Σχήματος 5.2 είναι η επόμενη:

1	Time:1212422124281 Type:EVENT Object:Object2@c17164 Message:Object2.msg1 Arguments:
2	Time:1212422124375 Type:EVENT Object:Object3@1fb8ee3 Message:Object3.msg2 Arguments:
3	Time:1212422124390 Type:EVENT Object:Object2@c17164 Message:Object2.msg1 Arguments:
4	Time:1212422124406 Type:EVENT Object:Object2@c17164 Message:Object2.msg1 Arguments:
5	Time:1212422124421 Type:EVENT Object:Object4@61de33 Message:Object4.msg5 Arguments:
6	Time:1212422124453 Type:EVENT Object:Object3@1fb8ee3 Message:Object3.msg2 Arguments:
7	Time:1212422124468 Type:EVENT Object:Object4@61de33 Message:Object4.msg4 Arguments:
8	Time:1212422124484 Type:EVENT Object:Object3@1fb8ee3 Message:Object3.msg2 Arguments:
9	Time:1212422124500 Type:EVENT Object:Object4@61de33 Message:Object4.msg3 Arguments:

Για να γίνει κατανοητή η πορεία εκτέλεσης στην παραπάνω περίπτωση απαιτείται η χρήση των (μοναδικών) αναγνωριστικών των μεταβάσεων που αντιστοιχούν σε γεγονότα. Αριθμώντας με αύξοντα τρόπο τους τελεστέους αλληλεπίδρασης ως προς το άξονα του χρόνου, η αντιστοίχιση είναι η εξής:

Τελεστέος 1	Object2.msg1 ← EVT1	Object3.msg2 ← EVT2	Object4.msg3 ← EVT3
Τελεστέος 2	Object2.msg1 ← EVT4	Object3.msg2 ← EVT5	Object4.msg4 ← EVT6
Τελεστέος 3	Object2.msg1 ← EVT7	Object3.msg2 ← EVT8	Object4.msg5 ← EVT9

Ακολουθεί το ίχνος της διαδικασίας επαλήθευσης σύμφωνα με τον αλγόριθμο που παρουσιάστηκε στην προηγούμενη ενότητα:

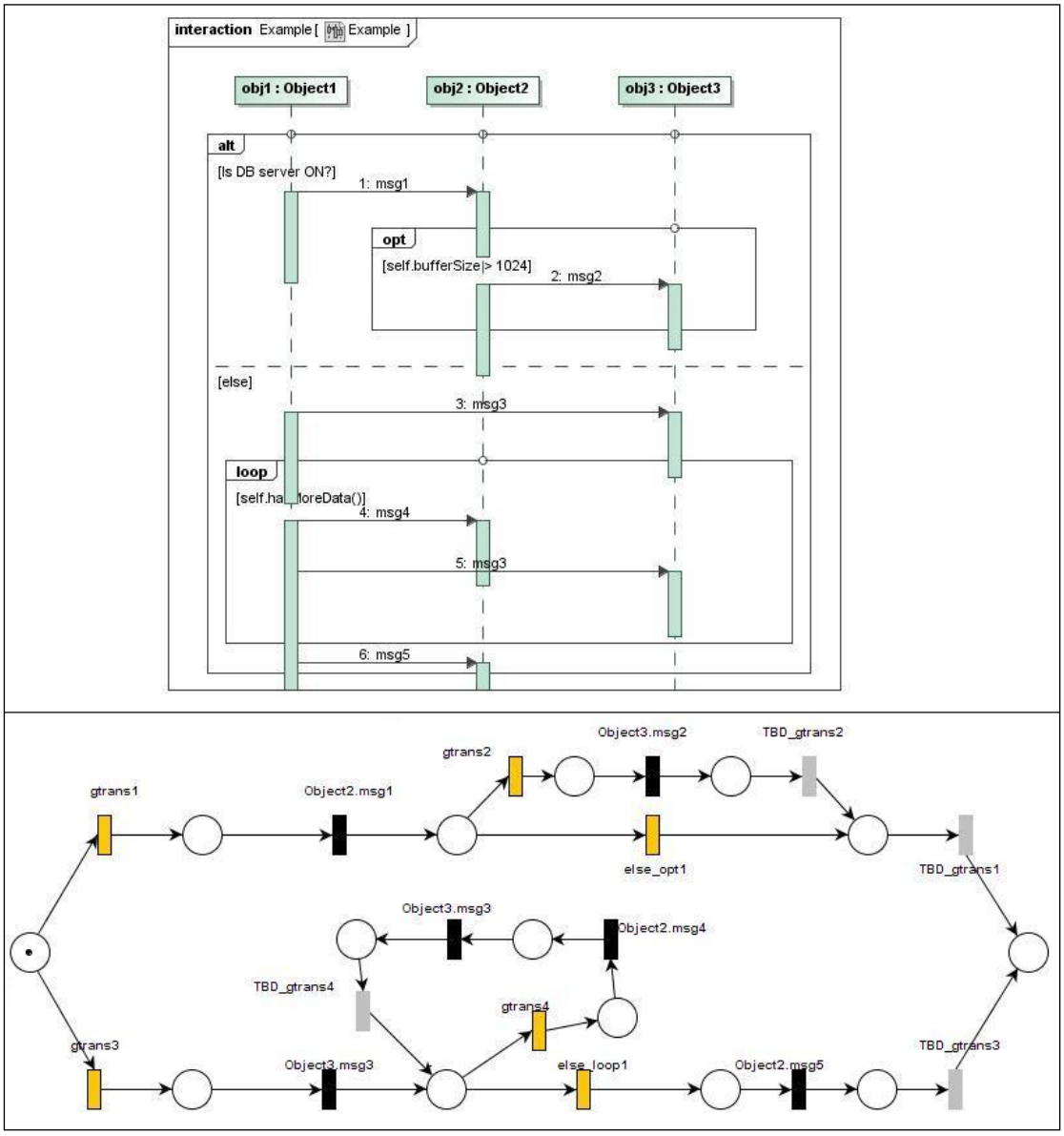
Ίχνος διαδικασίας επαλήθευσης:

1	Checkpoint stack EMPTY	31	Checkpoint PUSH
2	Checkpoint PUSH	32	Transition fired id: EVT7, name: Object2.msg1
3	Checkpoint PUSH	33	Checkpoint PUSH
4	Transition fired id: EVT1, name: Object2.msg1	34	Transition fired id: EVT1, name: Object2.msg1
5	Checkpoint PUSH	35	Checkpoint POP
6	Transition fired id: EVT2, name: Object3.msg2	36	Checkpoint POP
7	Checkpoint PUSH	37	Checkpoint POP
8	Transition fired id: EVT4, name: Object2.msg1	38	Checkpoint POP
9	Checkpoint PUSH	39	Checkpoint PUSH
10	Transition fired id: EVT7, name: Object2.msg1	40	Transition fired id: EVT7, name: Object2.msg1
11	Checkpoint POP	41	Checkpoint PUSH
12	Checkpoint POP	42	Transition fired id: EVT8, name: Object3.msg2
13	Checkpoint PUSH	43	Checkpoint PUSH
14	Transition fired id: EVT7, name: Object2.msg1	44	Transition fired id: EVT1, name: Object2.msg1
15	Checkpoint PUSH	45	Checkpoint PUSH
16	Transition fired id: EVT4, name: Object2.msg1	46	Transition fired id: EVT4, name: Object2.msg1
17	Checkpoint POP	47	Checkpoint PUSH
18	Checkpoint POP	48	Transition fired id: EVT9, name: Object4.msg5
19	Checkpoint POP	49	Checkpoint PUSH
20	Checkpoint POP	50	Transition fired id: EVT2, name: Object3.msg2
21	Checkpoint PUSH	51	Checkpoint POP
22	Transition fired id: EVT4, name: Object2.msg1	52	Checkpoint PUSH
23	Checkpoint PUSH	53	Transition fired id: EVT5, name: Object3.msg2
24	Transition fired id: EVT5, name: Object3.msg2	54	Checkpoint PUSH
25	Checkpoint PUSH	55	Transition fired id: EVT6, name: Object4.msg4
26	Transition fired id: EVT1, name: Object2.msg1	56	Checkpoint PUSH
27	Checkpoint PUSH	57	Transition fired id: EVT2, name: Object3.msg2
28	Transition fired id: EVT7, name: Object2.msg1	58	Checkpoint PUSH
29	Checkpoint POP	59	Transition fired id: EVT3, name: Object4.msg3
30	Checkpoint POP	60	Checkpoint stack CLEARED

Όπως φαίνεται από τις παραπάνω εγγραφές, κάθε φορά που η εκτέλεση καταλήγει σε αδιέξοδο, το μοντέλο επιστρέφει στην κατάσταση που βρισκόταν πριν την πιο πρόσφατη επιλογή (Checkpoint POP) και γίνεται διαφορετική επιλογή. Η δομή της στοίβας (*Last In First Out - LIFO*) διευκολύνει τον εξαντλητικό έλεγχο καθώς το μοντέλο επιστρέφει μετά από κάθε αδιέξοδο στην κατάσταση της πιο πρόσφατης επιλογής.

Παράδειγμα δικτύου με φυλασσόμενες μεταβάσεις

Στο Σχήμα 5.5 απεικονίζεται ένα σενάριο που περιλαμβάνει εκφράσεις φύλαξης σε συνδυασμένα τμήματα (alt, opt, loop) με το αντίστοιχο δίκτυο με φυλασσόμενες μεταβάσεις.



Σχήμα 5.5: Παράδειγμα δικτύου με φυλασσόμενες μεταβάσεις

Ακολουθούν ορισμένα παραδείγματα ακολουθιών γεγονότων που επαληθεύουν το σενάριο. Στις ακολουθίες αυτές περιλαμβάνονται και καταγραφές με τις τιμές των μεταβλητών που περιλαμβάνονται στις εκφράσεις φύλαξης. Επιπλέον, για κάθε μια από αυτές δίνεται το ίχνος της διαδικασίας επαλήθευσης.

Περίπτωση 1: (DB server is ON) = true, ([OCL 2.0 ...] self.bufferSize > 1024) = true

Καταγραφές:

1	Time:1212439967093 Type:INSTANCE Object:null Message:true,null,null Arguments:[Ljava.lang.Object;@61de33
2	Time:1212439967421 Type:EVENT Object:Object2@14318bb Message:Object2.msg1 Arguments:
3	Time:1212439967437 Type:INSTANCE Object:null Message:true,1660,null Arguments:[Ljava.lang.Object;@ca0b6
4	Time:1212439967438 Type:EVENT Object:Object3@10b30a7 Message:Object3.msg2 Arguments:

Ίχνος διαδικασίας επαλήθευσης:

1	Checkpoint stack EMPTY
2	Checkpoint PUSH
3	Guarded Transition: gtrans1, condition: Is DB server ON?, evaluation: true
4	Guarded Transition fired: gtrans1
5	Checkpoint PUSH
6	Transition fired id: EVT1, name: Object2.msg1
7	Guarded Transition: gtrans2, condition: self.bufferSize > 1024, evaluation: true
8	Guarded Transition fired: gtrans2
9	Checkpoint PUSH
10	Transition fired id: EVT2, name: Object3.msg2
11	TBD Transition fired: TBD_gtrans2
12	TBD Transition fired: TBD_gtrans1
13	Checkpoint stack CLEARED

Περίπτωση 2: (DB server is ON) = false, ([OCL 2.0 ...] self.hasMoreData() = true) = true (2 loops)

Καταγραφές:

1	Time:1212443431109 Type:INSTANCE Object:null Message:false,null,null Arguments:[Ljava.lang.Object;@61de33
2	Time:1212443431125 Type:EVENT Object:Object3@14318bb Message:Object3.msg3 Arguments:
3	Time:1212443431156 Type:INSTANCE Object:null Message:false,null,true Arguments:[Ljava.lang.Object;@ca0b6
4	Time:1212443431156 Type:EVENT Object:Object2@10b30a7 Message:Object2.msg4 Arguments:
5	Time:1212443431171 Type:EVENT Object:Object3@14318bb Message:Object3.msg3 Arguments:
6	Time:1212443431187 Type:INSTANCE Object:null Message:false,null,true Arguments:[Ljava.lang.Object;@1a758c
7	Time:1212443431187 Type:EVENT Object:Object2@10b30a7 Message:Object2.msg4 Arguments:
8	Time:1212443431203 Type:EVENT Object:Object3@14318bb Message:Object3.msg3 Arguments:
9	Time:1212443431218 Type:INSTANCE Object:null Message:false,null,false Arguments:[Ljava.lang.Object;@1b67f
10	Time:1212443431234 Type:EVENT Object:Object2@10b30a7 Message:Object2.msg5 Arguments:

Ίχνος διαδικασίας επαλήθευσης:

1	Checkpoint stack EMPTY	14	Guarded Transition: gtrans4, condition: self.hasMoreData() = true, evaluation: true
2	Checkpoint PUSH	15	Guarded Transition fired: gtrans4
3	Else transition: gtrans3, evaluation: true	16	Checkpoint PUSH
4	Guarded Transition fired: gtrans3	17	Transition fired id: EVT4, name: Object2.msg4
5	Checkpoint PUSH	18	Checkpoint PUSH
6	Transition fired id: EVT3, name: Object3.msg3	19	Transition fired id: EVT5, name: Object3.msg3
7	Guarded Transition: gtrans4, condition: self.hasMoreData() = true, evaluation: true	20	TBD Transition fired: TBD_gtrans4
8	Guarded Transition fired: gtrans4	21	Else transition: else_loop1, evaluation: true
9	Checkpoint PUSH	22	Guarded Transition fired: else_loop1
10	Transition fired id: EVT4, name: Object2.msg4	23	Checkpoint PUSH
11	Checkpoint PUSH	24	Transition fired id: EVT6, name: Object2.msg5
12	Transition fired id: EVT5, name: Object3.msg3	25	TBD Transition fired: TBD_gtrans3
13	TBD Transition fired: TBD_gtrans4	26	Checkpoint stack CLEARED

Η αποτίμηση των εκφράσεων φύλαξης στο δίκτυο του Σχήματος 5.5 γίνεται με χρήση των τιμών των μεταβλητών που υπάρχουν στις καταγραφές. Γίνεται σαφές ότι στην περίπτωση αυτή είναι απαραίτητο, πέρα από τις κλήσεις μεθόδων, να παρακολουθούνται και τιμές που φέρουν οι μεταβλητές αυτές κατά την εκτέλεση. Κάτι τέτοιο σίγουρα μπορεί να γίνει με ενορχήστρωση του πηγαίου κώδικα, αν και ένας από τους στόχους που έχουν τεθεί είναι η αποφυγή επέμβασης στον πηγαίο κώδικα.

Για την αντιμετώπιση αυτού του προβλήματος μπορεί να προταθεί μια τεχνική συλλογής των τιμών των μεταβλητών και πάλι μέσω της ενορχήστρωσης bytecode. Πρέπει όμως να σημειωθεί ότι η εφαρμογή της τεχνικής προϋποθέτει ότι οι μεταβλητές που χρησιμοποιούνται σε εκφράσεις φύλαξης, εφόσον αναφέρονται σε κάποια κλάση είτε να είναι δημόσιες (*public*) ή, στην περίπτωση που είναι ιδιωτικές (*private*) ή προστατευμένες (*protected*) να υπάρχουν αντίστοιχες μέθοδοι που επιστρέφουν τις τιμές τους («*getters*»). Όταν η παραπάνω προϋπόθεση ισχύει, μπορεί να εφαρμοσθεί μια μέθοδος «*poling*». Συγκεκριμένα, στον ορισμό του probe λαμβάνονται υπόψη οι μεταβλητές που ενδιαφέρουν με τις αντίστοιχες κλάσεις. Μάλιστα, καθώς κατά κανόνα οι εκφράσεις αυτές όταν περιλαμβάνουν μεταβλητές κλάσεων εκφράζονται σε κάποιο τυπικό συντακτικό, η διαδικασία αναγνώρισης μπορεί να αυτοματοποιηθεί. Στο τμήμα του probe που παρακολουθεί τις εισόδους σε μεθόδους γίνεται *casting* του αντικειμένου που δέχεται την κλήση. Εφόσον αναγνωριστεί η κλάση του αντικειμένου, τότε λαμβάνονται οι τρέχουσες τιμές των μεταβλητών (ιδιοχαρακτηριστικών) που ενδιαφέρουν για το συγκεκριμένο αντικείμενο και συγκρίνονται με τις τελευταίες τιμές που έχουν καταγραφεί. Η ανάκτηση των τιμών μπορεί να γίνει είτε απ' ευθείας, είτε μέσω κάποιας getter μεθόδου, ανάλογα με την εμβέλεια της εκάστοτε μεταβλητής όπως εξηγήθηκε

προηγουμένως. Εφόσον υπάρχουν μεταβολές, τότε στις καταγραφές προστίθενται αντίστοιχες εγγραφές *INSTANCE* που φέρουν τις νέες τιμές. Βέβαια, για να μη διαταραχθεί η λειτουργία του probe από τις κλήσεις σε μεθόδους που ενδεχομένως να γίνονται, πρέπει οι κλήσεις αυτές να περιλαμβάνονται στο φίλτρο του probe. Αυτό όμως μπορεί να γίνει μόνο αν οι μέθοδοι αυτές δεν εμφανίζονται στα σενάρια ενδιαφέροντος. Παρόλα αυτά, ακόμα και στην περίπτωση αυτή όμως, μπορούν να αναζητηθούν διάφορες διέξοδοι για τη συλλογή των τιμών που ενδιαφέρουν.

Σε κάθε περίπτωση η χρήση τιμών από καταγραφές είναι μια από τις δυνατότητες, καθώς μπορούν να χρησιμοποιηθούν διάφορες εναλλακτικές ανάλογα με τις ανάγκες και τις δυνατότητες που υπάρχουν. Για παράδειγμα, οι εκφράσεις φύλαξης ενδεχομένως να αφορούν το *configuration* του συστήματος ή της εφαρμογής, μεταβλητές περιβάλλοντος που τηρούνται σε διαφορετικές ροές καταγραφής, χρονικές εξαρτήσεις μεταξύ των γεγονότων, αλληλεπίδραση και αποφάσεις από το χρήστη ή κάποιο σύστημα και άλλα.

Κεφάλαιο 6: Λεπτομέρειες υλοποίησης και μελέτες περίπτωσης

Στο κεφάλαιο αυτό παρουσιάζεται η αρχιτεκτονική του πλαισίου, γίνεται μια συγκέντρωση και ανακεφαλαίωση των τεχνικών λεπτομερειών και περιορισμών της υλοποίησης, τόσο ως προς τα εργαλεία που δημιουργήθηκαν ή όσο και ως προς αυτά που χρησιμοποιούνται. Επιπλεόν, αξιολογείται η διαδικασία ελέγχου εμφάνισης των σεναρίων που προτάθηκε μέσω μετρήσεων του απαιτούμενου χρόνου ως προς την πολυπλοκότητα των σεναρίων και το πλήθος των εγγραφών που αναλύονται. Στα πλαίσια της αξιολόγησης αυτής προτείνεται, επίσης, μια απλή μετρική για την πολυπλοκότητα των Διαγραμμάτων Ακολουθίας. Στην τελευταία ενότητα, παρατίθενται χαρακτηριστικές λήψης οθόνης ορισμένων εκ των εργαλείων που περιλαμβάνονται στο πλαίσιο.

6.1 Αρχιτεκτονική πλαισίου

Μια εποπτική εικόνα της υλοποίησης του περιβάλλοντος-πλαίσιο που προτείνεται για την παρακολούθηση και την ανάλυση της εκτέλεσης συστημάτων λογισμικού φαίνεται στο Σχήμα 6.1

Σχήμα 6.1: Εποπτική εικόνα υλοποίησης περιβάλλοντος-πλαισίου

Το πλαίσιο προϋποθέτει τον προσδιορισμό κάποιας πολιτικής παρακολούθησης και καταγραφής. Η πολιτική αυτή περιλαμβάνει το ποια συστήματα λογισμικού θα παρακολουθηθούν και ως προς τι (απαιτήσεις, υπογραφές εισβολών, χαρακτηριστικά επίδοσης κλπ). Από την πολιτική που προσδιορίζεται προκύπτουν τα σενάρια προς παρακολούθηση, τα δομοστοιχεία προς ενορχήστρωση και το επιθυμητό επίπεδο καταγραφής για το καθένα, με τα αντίστοιχα probes.

Το πλαίσιο μπορεί να χωριστεί σε τρία τμήματα. Το πρώτο αφορά τη διαδικασία παραγωγής των Δικτύων Petri μέσω του μετασχηματισμού των Διαγραμμάτων Ακολουθίας που αποτυπώνουν τα σενάρια. Το δεύτερο αφορά στην παρακολούθηση της εκτέλεσης του λογισμικού και στη συλλογή και εξαγωγή των γεγονότων που ενδιαφέρουν μέσω του προσδιορισμού κατάλληλων probe. Το τρίτο τμήμα αφορά στην εκτέλεση των Δικτύων Petri βάση των καταγραφών που έχουν συλλεχθεί για την επαλήθευση ή την απόρριψη των σεναρίων. Στο τμήμα αυτό δίνεται η δυνατότητα τόσο της ανάλυσης αρχείων καταγραφής όσο και της επιγραμμικής ανάλυσης, σε χρόνο κοντά στον πραγματικό.

Λεπτομέρειες υλοποίησης πρώτου τμήματος:

Χρησιμοποιήθηκε το εργαλείο MagicDraw UML Personal Edition v15.1 για τον σχεδιασμό των Διαγραμμάτων Ακολουθίας και την εξαγωγή τους σε EMF UML 2.0 XMI.

Στο περιβάλλον του Eclipse και του Eclipse Modeling Framework, χρησιμοποιήθηκε η ATL ως γλώσσα έκφρασης των κανόνων αντιστοίχησης και η αντίστοιχη μηχανή της ATL για εκτέλεση του μετασχηματισμού. Το μεταμοντέλο του μοντέλου στόχου προσδιορίστηκε στη γλώσσα KM3 και εξήχθη με αυτοματοποιημένο τρόπο σε EMF XMI. Για τους σκοπούς του μετασχηματισμού δημιουργήθηκε ένα ATL module με τους κανόνες αντιστοίχησης και κατάλληλους βοηθούς. Το module παράγει ένα μοντέλο σε XMI που εισάγεται σε δύο ATL queries για την εξαγωγή δύο αρχείων σε PNML μορφή, ένα για το εργαλείο TINA κι ένα πλήρες ως προς τα μεταδεδομένα/πληροφορίες του δικτύου. Εκτελείται έπειτα μέσω του TINA μια διαδικασία απόδοσης γραφικών χαρακτηριστικών στο δίκτυο και τελικά μέσω ενός προγράμματος που δημιουργήθηκε (*Merge*), τα αρχεία που έχουν παραχθεί συγχωνεύονται ώστε να εξαχθεί η τελική μορφή του αρχείου PNML που αντιστοιχεί στο δίκτυο. Η παραπάνω διαδικασία περιγράφεται αναλυτικά στο κεφάλαιο του μετασχηματισμού (βλ. 4.4).

Λεπτομέρειες υλοποίησης δεύτερου τμήματος:

Στο περιβάλλον του Eclipse και πάλι, χρησιμοποιείται το Probekit του TPTP για τη δημιουργία κατάλληλων probe. Τα probe ρυθμίζονται ώστε να εκτελούν τα τμήματα κώδικα σε κάθε είσοδο σε μέθοδο. Επίσης, παρακολουθούνται και τιμές μεταβλητών που περιλαμβάνονται σε εκφράσεις φύλαξης σύμφωνα με μια τεχνική που προσδιορίστηκε σε προηγούμενο κεφάλαιο (βλ. 5.3). Επιπλέον στο probe προσδιορίζεται ένα αρχείο εξόδου για την αποθήκευση των γεγονότων υπό το συγκεκριμένο μορφότυπο (βλ. 5.1) ή ενσωματώνεται (εσωτερική κλάση) ένας απλός TCP/IP πελάτης ο αποστέλλει τις εγγραφές (και πάλι υπό τον ίδιο μορφότυπο) σε έναν εξυπηρετητή του εργαλείου ανάλυσης. Το TPTP παρέχει κατάλληλη υποδομή ώστε τα probe να ενορχηστρώνονται δυναμικά κατά την εκτέλεση αλλά και ένα εργαλείο για την στατική ενορχήστρωσή τους στο bytecode των κλάσεων Java. Για λόγους διαλειτουργικότητας έχει προσδιοριστεί επίσης ένας κανόνας προσαρμογής των αρχείων καταγραφής που παράγονται, από το συγκεκριμένο μορφότυπο που ορίζεται, στη μορφή Common Base Event με τη βοήθεια του Generic Log Adapter (βλ. 3.1).

Λεπτομέρειες υλοποίησης τρίτου τμήματος:

Επεκτάθηκε το εργαλείο Platform Independent Petri net Editor v2.5 ώστε να χειρίζεται Δίκτυα Petri υπό τη μορφή που ορίστηκε στο κεφάλαιο του μετασχηματισμού (βλ. 4.2). Επεκτάθηκε ο μορφότυπος των PNML αρχείων που το εργαλείο έκανε αποδεκτά και

επεκτάθηκε ο γραφικός επεξεργαστής του εργαλείου ώστε να απεικονίζει και να τροποποιεί τα ιδιοχαρακτηριστικά των μεταβάσεις (φυλασσόμενων και μη).

Προστέθηκαν δύο επιλογές, μια για την ανάλυση αρχείων καταγραφής, μέσω της κλάσης *LogFileAnalyzer* και μια για την επιγραμμική ανάλυση μέσω της κλάσης *OnlineLogAnalyzer*. Επίσης υλοποιήθηκε ένας εξυπηρετητής (κλάση *Server*) για την παραλαβή των γεγονότων κατά την επιγραμμική ανάλυση και μια κατάλληλη δομή για τη διατήρηση της αλληλουχίας των γεγονότων στη μνήμη, προσομοιώνοντας πρακτικά και επεκτείνοντας τα χαρακτηριστικά ενός αρχείου τυχαίας πρόσβασης (*OnlineLog*). Επιπλέον, υλοποιήθηκε μια υποδομή για την αποτίμηση των εκφράσεων φύλαξης βάση του σχεδιαστικού μορφήματος Strategy και δημιουργήθηκαν ορισμένες υποτυπώδεις μηχανές αποτίμησης, για την εξυπηρέτηση του ελέγχου (*DefaultEvaluationEngine, NaturalLanguageEvaluationEngine, OCL2EvaluationEngine*). Τέλος, προστέθηκαν ορισμένες κλάσεις για την παραγωγή δεδομένων αξιολόγησης της τεχνικής ανάλυσης των καταγραφών και της πολυπλοκότητας των δικτύων που εξετάζονται.

Όταν από τον έλεγχο προκύπτει ότι το σενάριο επαληθεύεται από τις καταγραφές τότε δίνεται η δυνατότητα να παρακολουθήσει ο χρήστης με γραφικό τρόπο την εκτέλεση του δικτύου. Σε κάθε περίπτωση υπάρχει η δυνατότητα εξαγωγής του ιστορικού της εκτέλεσης.

Περιορισμοί υλοποίησης:

Όπως σημειώθηκε και προηγουμένως ένας βασικός περιορισμός της παραπάνω υλοποίησης είναι ότι ο μετασχηματισμός των Διαγραμμάτων Ακολουθίας σε Διαγράμματα Petri αφορά μόνο ένα υποσύνολο των στοιχείων των διαγραμμάτων. Όπως θα συζητηθεί και στον επίλογο της εργασίας μια δυνατή επέκταση του μετασχηματισμού θα ήταν ο χειρισμός χαρακτηριστικών απολύτου χρόνου. Στην περίπτωση αυτή, η χρησιμοποίηση του PIPE παρέχει ήδη την υποδομή για την χρήση μεταβάσεων χρόνου όπως και αντίστοιχα εργαλεία.

Ένας δεύτερος επίσης βασικός περιορισμός είναι ότι οι εφαρμογές που ενορχηστρώνονται μέσω του Probekit πρέπει να είναι γραμμένες σε Java κι αυτό γίνεται για να αποφευχθεί η ενορχήστρωση του κώδικα των εφαρμογών.

Τέλος ο χειρισμός του θορύβου που περιλαμβάνεται στις καταγραφών γίνεται κατά την εκτέλεση της ανάλυσης. Εναλλακτικά θα μπορούσαν να εφαρμοστούν φίλτρα, σε διάφορα ενδεχομένως στάδια πριν την τελική τροφοδότηση της ακολουθίας των γεγονότων ή ο χειρισμός του σε επίπεδο μοντέλου (βλ. 4.3).

6.2 Αξιολόγηση προσέγγισης

Μια παράμετρος αξιολόγησης του πλαισίου είναι η επίδοση της τεχνικής της ανάλυσης ως προς την πολυπλοκότητα των σεναρίων -και άρα και των δικτύων που παράγονται από τη διαδικασία μετασχηματισμού.

Ως επίδοση μπορεί να θεωρηθεί ο χρόνος που διαρκεί η εκτέλεση της ανάλυσης. Μια τέτοια θεώρηση γίνεται για να αποτυπωθεί ποιοτικά η σχέση καθώς ο απόλυτος χρόνος σαφώς, εξαρτάται και από πολλούς εξωγενείς παράγοντες.

Μια μέτρηση της πολυπλοκότητα ενός Διαγράμματος Ακολουθίας ως προς τη ροή ελέγχου, θα μπορούσε να γίνει μέσω της κυκλωματική πολυπλοκότητα (*cyclomatic complexity – CC*) του McCabe. Η κυκλωματική πολυπλοκότητα μπορεί να υπολογιστεί από το γράφο ροής ελέγχου (*control flow graph*) που προκύπτει από το σενάριο και ισούται με:

$$CC = E – N + 2P$$

όπου E είναι το πλήθος των ακμών, N το πλήθος των κόμβων και P το πλήθος των ισχυρώς συνδεδεμένων στοιχείων (σε ένα πρόγραμμα οι υπορουτίνες και οι συναρτήσεις). Στις περιπτώσεις των διαγραμμάτων χωρίς πλαίσια με χειριστή ref, όπως αυτά που θα εξετασθούν το P ισούται με τη μονάδα. Επίσης, μια ισοδύναμη έκφραση της παραπάνω σχέσης είναι η:

$$CC = q + 1$$

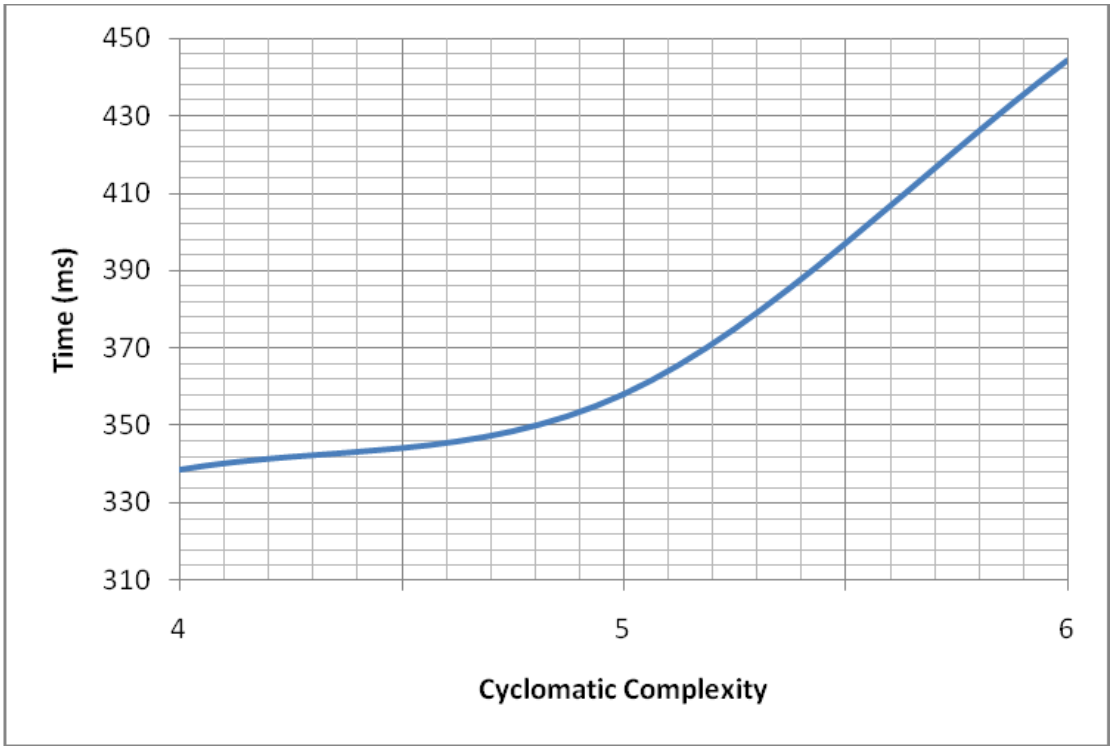
όπου q τα σημεία απόφασης. Η κυκλωματική πολυπλοκότητα δίνει έμφαση στη ροή ελέγχου ενός προγράμματος (και σε ευθεία αντιστοιχία ενός σεναρίου) αλλά δεν αποδίδει αυξημένο βάρος σε φωλιασμένες δομές ελέγχου. Επιπλέον, η δυνατότητα χρήσης συνδυασμένων τμημάτων με χειριστή par για την έκφραση διαδικασιών που συμβαίνουν παράλληλα είναι κάτι που η κυκλωματικής πολυπλοκότητα δεν αποτυπώνει καθώς χρησιμοποιείται κατά κανόνα για ακολουθιακά προγράμματα.

Μια μετρική που επεκτείνει την κυκλωματική πολυπλοκότητα λαμβάνοντας υπόψη σύνδρομα χαρακτηριστικά καλείται σύνδρομη πολυπλοκότητα (*concurrent complexity*) και προτάθηκε από τους de Paoli και Morasca στην εργασία [PM90]. Μάλιστα στην εργασία αυτή η μετρική αυτή έγινε με τη βοήθεια Δικτύων Petri και συγκεκριμένα δίνεται από τον τύπο:

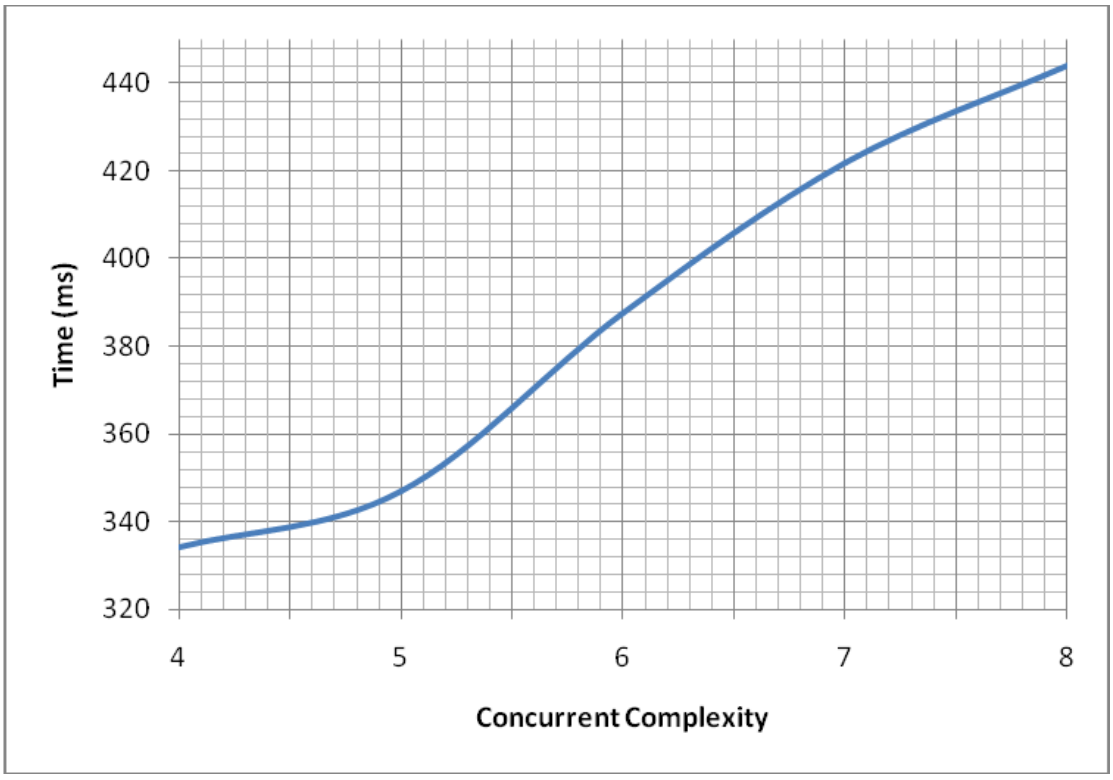
$$V(PN) = |F| - |T| - |P| + 2$$

όπου |F| το πλήθος των διανυσμάτων στο δίκτυο, |T| το πλήθος των μεταβάσεων και |P| το πλήθος των θέσεων. Αν και η προσέγγιση αναπαράστασης της λογικής ροής σε Δίκτυα Petri που ακολουθείται στην εργασία είναι ελαφρώς διαφορετική (χρησιμοποιούνται απλά Δίκτυα Petri όπου οι επιλογές γίνονται μη ντετερμινιστικά) ο παραπάνω τύπος θα μπορούσε να χρησιμοποιηθεί για την ποιοτική αξιολόγηση της τεχνικής της ανάλυσης.

Στο Σχήμα 6.2 και στο Σχήμα 6.3 παρατίθενται γραφήματα που απεικονίζουν το μέσο χρόνο εκτέλεσης της διαδικασίας επαλήθευσης για δίκτυα διαφορετικής κυκλωματικής και σύνδρομης πολυπλοκότητας. Χρησιμοποιήθηκαν δίκτυα πέντε δίκτυα, διαφορετικής σύνθεσης (ως προς τις δομές που αντιστοιχούν σε συνδυασμένα τμήματα) και μεγέθους. Το κάθε δίκτυο εκτελέστηκε δέκα φορές μέσω ακολουθιών γεγονότων που το επαλήθευαν και παράγονταν μέσω τυχαίας εκτέλεσης των ενορχηστρωμένων εφαρμογών. Οι μετρήσεις πραγματοποιήθηκαν σε περιβάλλον θορύβου, ο οποίος κυμαίνονταν με τυχαίο τρόπο στην περιοχή του 30-70% των συνολικών καταγραφών. Για τις εκτελέσεις κάθε δικτύου υπολογίστηκε ο μέσος όρος του χρόνου ανάλυσης.



Σχήμα 6.2: Χρόνος εκτέλεσης ως προς την κυκλωματική πολυπλοκότητα του δικτύου



Σχήμα 6.3: *Χρόνος εκτέλεσης ως προς την κυκλωματική πολυπλοκότητα του δικτύου*

Οι μετρήσεις έγιναν χρησιμοποιώντας την τεχνική της επιγραμμικής ανάλυσης στην οποία τόσο οι εφαρμογές όσο και το εργαλείο ελέγχου τρέχανε στον ίδιο σε φορητό υπολογιστή με επεξεργαστή Intel Core Duo 1.66 GHz και μνήμη RAM 1 GB στο περιβάλλον των Windows XP. Αντίστοιχες μετρήσεις με χρήση όμως αρχείων καταγραφής έδωσαν μικρότερους χρόνους, σχεδόν κατά 40%. Η διαφορά αυτή δικαιολογείται από τις χρονικές επιβαρύνσεις που συζητήθηκαν στο προηγούμενο κεφάλαιο.

Αν και τα γραφήματα στα Σχήματα 6.2 και 6.3 παρέχουν μια εικόνα για τη σχέση επίδοσης και πολυπλοκότητας σεναρίων/δικτύων, δεν αποτυπώνουν τη συμπεριφορά του αλγορίθμου ταύτισης όταν στο δίκτυο υπάρχουν δομές που αντιστοιχούν σε συνδυασμένα τμήματα με χειριστή par και οι οποίες περιλαμβάνουν γεγονότα που ενδέχεται να ικανοποιούν ταυτόχρονα τις συνθήκες ταύτισης («ταυτόσημα γεγονότα»). Επίσης καμία από τις δύο μετρικές δε συνυπολογίζει το μέγεθος του υπό διερεύνηση σεναρίου.

Δυστυχώς οι μετρικές που βρέθηκαν στη βιβλιογραφία δεν λαμβάνουν υπόψη τα παραπάνω ζητήματα. Μπορούμε παρόλα αυτά να ορίσουμε μια απλή μετρική *M* που να σταθμίζει με διαφορετικά βάρη τα διάφορα συνδυασμένα ή μη, τμήματα.

Στη μετρική ορίζουμε ότι ένα απλό γεγονός έχει κόστος ίσο με τη μονάδα. Για κάθε alt υπολογίζεται το άθροισμα των κόστων των τελεστέων και αυτό διαιρείται με το πλήθος τους. Για κάθε opt υπολογίζεται το κόστος του τελεστέου και πολλαπλασιάζεται με βάρος O=0,5.

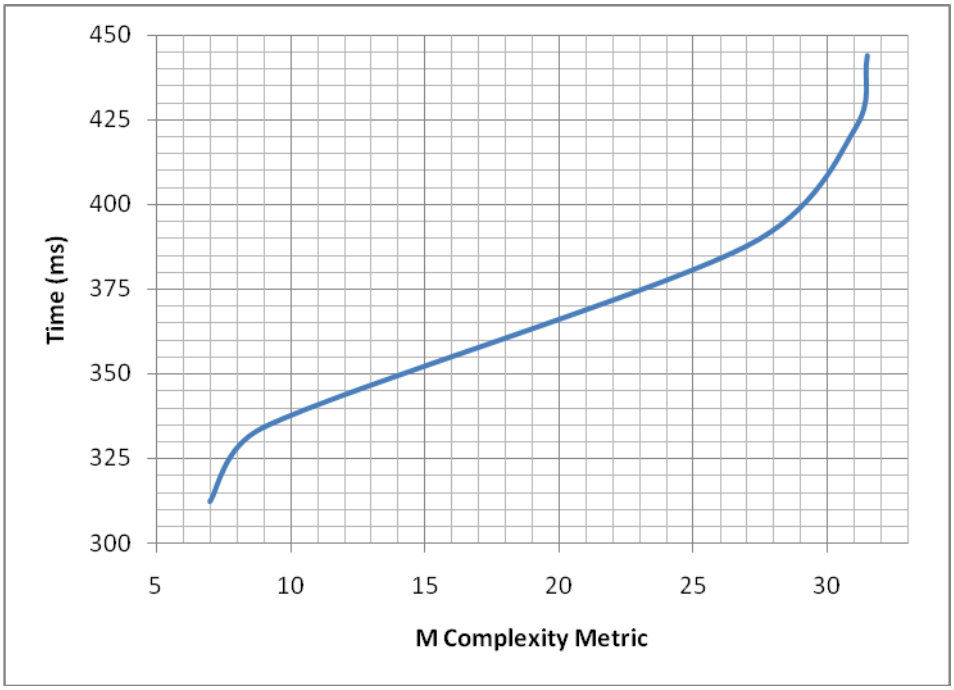
Για κάθε loop υπολογίζεται το κόστος του τελεστέου και πολλαπλασιάζεται με βάρος L=2. Για κάθε par υπολογίζεται το άθροισμα των κόστων των τελεστέων και πολλαπλασιάζεται με βάρος P=4. Με τον τρόπο αυτό λαμβάνεται υπόψη το πλήθος των γεγονότων στο σενάριο, το ενδεχόμενο φωλιασμένων συνδυασμένων τμημάτων και μέσω της απόδοσης βαρών σταθμίζεται ανάλογα κάθε τύπος τμήματος. Επίσης, το γεγονός ότι η μετρική αναφέρεται στο αρχικό μοντέλο (Διαγράμματα Ακολουθίας) μπορεί να χρησιμοποιηθεί για την αξιολόγηση σε επίπεδο επίδοσης, του συνόλου της διαδικασία -συμπεριλαμβανομένης δηλαδή και της γραμματικής του μετασχηματισμού. Παρόλα αυτά, η χρήση των συγκεκριμένων τιμών για τα βάρη δεν ακολουθεί κάποιο κανόνα και η προσέγγιση είναι εμπειρική. Για τη μετρική, M έχουμε:

$$M = \sum_{i=1}^{n} C_i \, , i = 1 \dots n \; \tau\alpha \; \gamma\varepsilon\gamma o\nu \acute{o}\tau\alpha \; \acute{\eta} \; \tau\alpha \; \sigma\upsilon\nu\delta\upsilon\alpha\sigma\mu \acute{\varepsilon}\nu\alpha \; \tau\mu \acute{\eta}\mu\alpha\tau\alpha \; \pi\rho\acute{\omega}\tau o\upsilon \; \varepsilon\pi\iota\pi \acute{\varepsilon}\delta o\upsilon$$

$$C_i = \begin{cases} 1 & \alpha\nu \; i \; \varepsilon \acute{\iota}\nu\alpha\iota \; \gamma\varepsilon\gamma o\nu \acute{o}\varsigma \\ \dfrac{\sum_{j=1}^{m} M_j}{m} & \alpha\nu \; i \; \varepsilon \acute{\iota}\nu\alpha\iota \; alt \; \Sigma T, \;\; j = 1 \dots m \; o\iota \; \tau\varepsilon\lambda\varepsilon\sigma\tau \acute{\varepsilon}o\iota \\ O \cdot M_{op} & \alpha\nu \; i \; \varepsilon \acute{\iota}\nu\alpha\iota \; opt \; \Sigma T, \;\; op \; o \; \tau\varepsilon\lambda\varepsilon\sigma\tau \acute{\varepsilon}o\varsigma \\ L \cdot M_{op} & \alpha\nu \; i \; \varepsilon \acute{\iota}\nu\alpha\iota \; loop \; \Sigma T, \;\; op \; o \; \tau\varepsilon\lambda\varepsilon\sigma\tau \acute{\varepsilon}o\varsigma \\ P \cdot M_{op} & \alpha\nu \; i \; \varepsilon \acute{\iota}\nu\alpha\iota \; par \; \Sigma T, \;\; op \; o \; \tau\varepsilon\lambda\varepsilon\sigma\tau \acute{\varepsilon}o\varsigma \end{cases}$$

Θεωρούμε O=0,5, L=2 και P=4.

Σύμφωνα με τα παραπάνω, για παρόμοιες με τις παραπάνω μετρήσεις, η σχέση μεταξύ χρόνου και μετρικής *M* αποτυπώνεται στο Σχήμα 6.4.

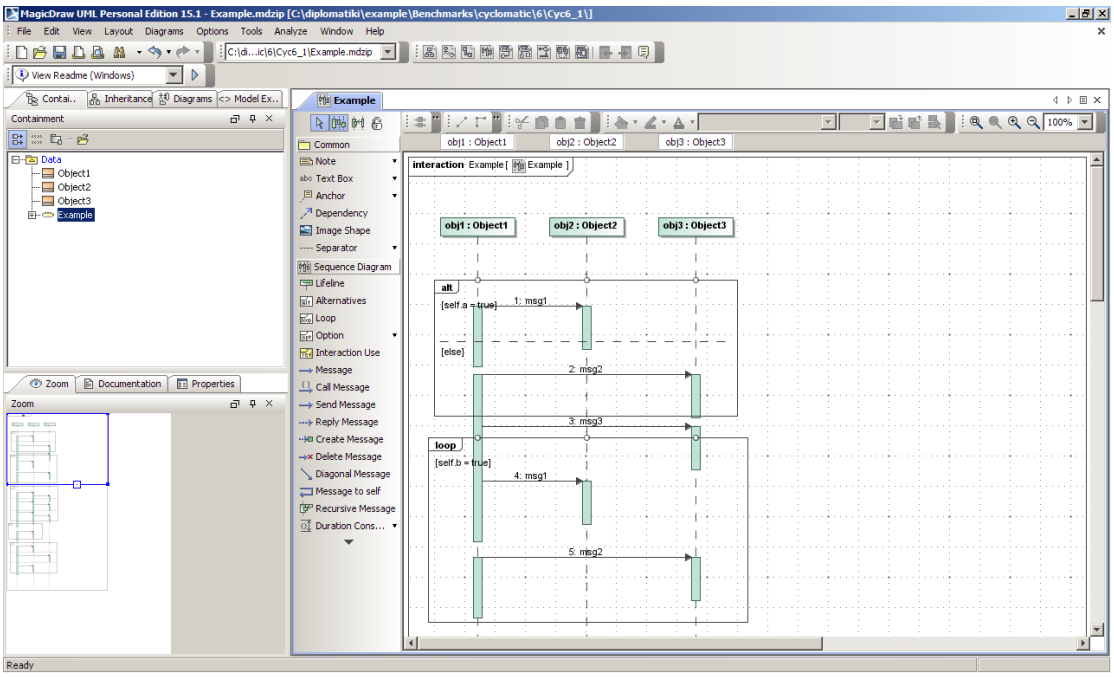


Σχήμα 6.4: Χρόνος εκτέλεσης ως προς τη μετρική M

6.3 Λήψεις οθόνης εργαλείων

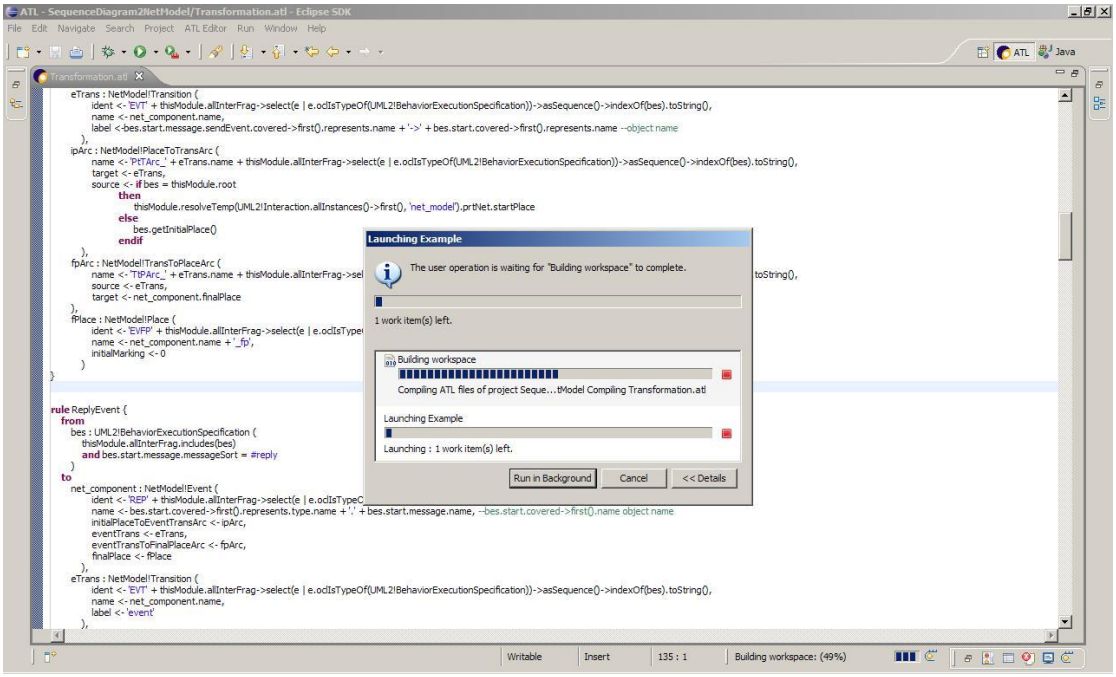
Σχεδιασμός Διαγράμματος Ακολουθίας

Εργαλείο: MagicDraw UML Personal Edition (v15.1)



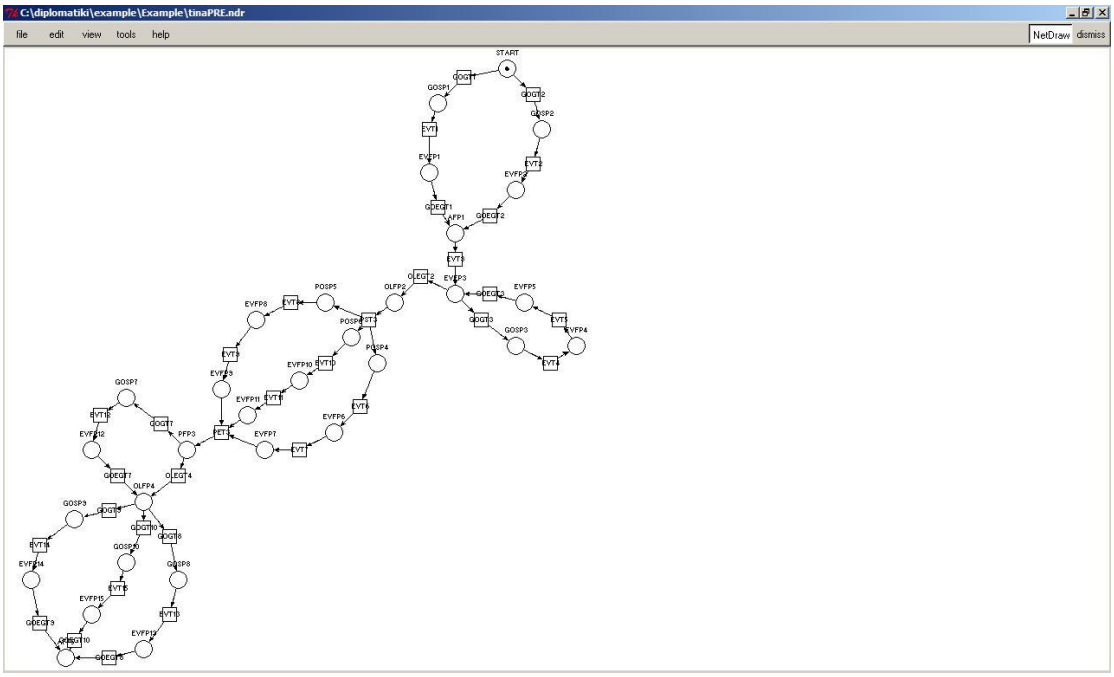
Εκτέλεση Μετασχηματισμού

Εργαλείο: ATL Perspective, Eclipse



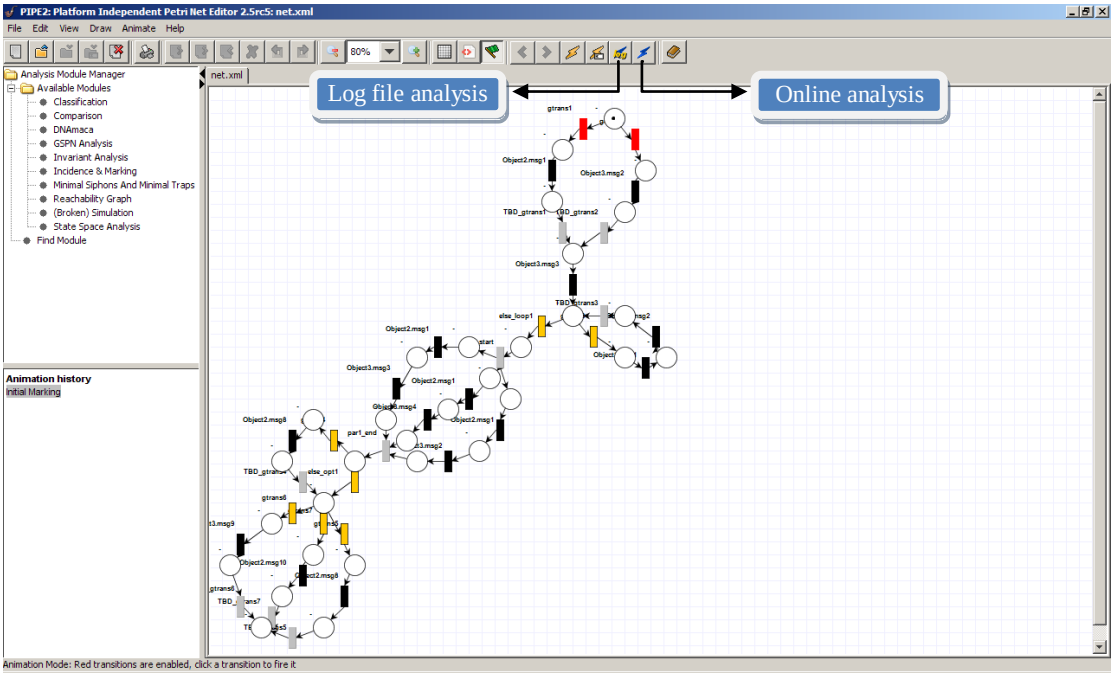
Απόδοση γραφικών χαρακτηριστικών

Εργαλείο: TINA (v2.9)



Έλεγχος Μοντέλου

Εργαλείο: Platform Independent Petri Net Editor (v2.5)



Κεφάλαιο 7: Επίλογος

Στο κεφάλαιο αυτό γίνεται μια σύνοψη του περιβάλλοντος-πλαισίου που παρουσιάστηκε στον παρόντα τόμο και συζητούνται τα συμπεράσματα που εξήχθησαν ως προς την κάλυψη των τεθέντων στόχων. Το κεφάλαιο κλείνει με μια αναφορά σε δυνατές μελλοντικές επεκτάσεις του πλαισίου και των τεχνικών που αναπτύχθηκαν.

7.1 Σύνοψη και συμπεράσματα

Στην παρούσα εργασία συζητήθηκε η ανάγκη για παρακολούθηση, καταγραφή και ανάλυση της λειτουργίας των συστημάτων λογισμικού και των συμπεριφορών που αναπτύσσονται κατά τη διάρκεια της εκτέλεσής τους και επισημάνθηκε η σπουδαιότητα της διαδικασία αυτής ως προς τον έλεγχο της αξιοπιστίας, της επίδοσης και της ασφάλειας των συστημάτων. Μάλιστα, καθώς τα συστήματα ακολουθούν όλο και περισσότερο κατανεμημένες αρχιτεκτονικές οι απαιτήσεις για την κάλυψη της ανάγκης αυτής αυξάνονται. Στο περιβάλλον-πλαίσιο που παρουσιάστηκε, προτάθηκε ως τρόπος περιγραφής αποδεκτών ή μη αποδεκτών λειτουργιών και συμπεριφορών η χρήση των Ακολουθιακών Διαγραμμάτων της UML, λόγω των πλούσιων εκφραστικών μέσων που διαθέτουν και της μεγάλης απήχησης που έχουν στους μηχανικούς λογισμικού όσο και στους αναλυτές συστημάτων ή διαδικασιών. Μέσω των Διαγραμμάτων Ακολουθίας μπορούν να εκφραστούν σενάρια δηλαδή σχηματομορφές ακολουθιών γεγονότων που μπορούν να συμβούν κατά την εκτέλεση του συστήματος. Επειδή όμως τα Διαγράμματα Ακολουθίας δε διαθέτουν ισχυρά τυπικά χαρακτηριστικά και οι δυνατότητες ανάλυσης είναι περιορισμένες, ορίστηκαν και υλοποιήθηκαν στα πλαίσια της εργασίας κανόνες μετασχηματισμού τους σε Δίκτυα Petri, τα οποία μπορούν να εκτελεστούν και για τα οποία υπάρχει μια ισχυρή μαθηματική υποδομή. Η διαδικασία αυτή άντλησε πολλά πρότυπα και τεχνολογίες μοντελοποίησης και μετασχηματισμών από το χώρο της ανάπτυξης οδηγούμενης από μοντέλα. Στα πλαίσια της διαδικασίας παραγωγής Δικτύων Petri, παρουσιάστηκε μια τεχνική για την απόδοση

γραφικών χαρακτηριστικών στα παραγόμενα δίκτυα. Παράλληλα, για τη συλλογή των δεδομένων κατά την εκτέλεση εφαρμογών γραμμένων σε Java χρησιμοποιήθηκε η ενορχήστρωση του bytecode ως εναλλακτική στην ενορχήστρωση κώδικα. Για το λόγο αυτό ορίστηκαν κατάλληλα probe που ενορχηστρώθηκαν στις εφαρμογές για την εξαγωγή δεδομένων παρακολούθησης συγκεκριμένου μορφότυπου. Η ανάλυση των δεδομένων αυτών ως προς τα σενάρια ενδιαφέροντος έγινε με χρήση των Δικτύων Petri που παρήχθησαν από τα Διαγράμματα Ακολουθίας και συγκεκριμένα μέσω της εκτέλεση των δικτύων βάσει των καταγεγραμμένων γεγονότων. Η ανάλυση αυτή υλοποιήθηκε τόσο ως προς αρχεία καταγραφής (log file analysis) όσο και μέσω μιας επιγραμμική αρχιτεκτονικής όπου οι ακολουθίες γεγονότων που συμβαίνουν και συλλέγονται από το υπό εξέταση σύστημα αναλύονται άμεσα και η απάντηση δίνεται σε χρόνο «κοντά» στον πραγματικό. Ο αλγόριθμος της εκτέλεσης αντιμετωπίζει το ενδεχόμενο υποακολουθιών γεγονότων που συμβαίνουν παράλληλα και περιλαμβάνουν «ταυτόσημα» γεγονότα εξετάζοντας εξαντλητικά το ενδεχόμενο αναγνώρισης των σεναρίων. Επίσης, η ανάλυση που γίνεται είναι «δυναμική», δηλαδή οι συνθήκες που ορίζονται για γεγονότα ή για ακολουθίες γεγονότων αποτιμώνται κατά την εκτέλεση του αλγορίθμου και για το λόγο αυτό δημιουργήθηκε κατάλληλη υποδομή που διευκολύνει την προσθήκη μηχανών αποτίμησης για την κάλυψη διαφορετικών αναγκών και περιπτώσεων εφαρμογής.

Συμπερασματικά, η χρήση Διαγραμμάτων Ακολουθίας για την έκφραση των σεναρίων απαλλάσσει από την ανάγκη για γνώση ιδιωτικών και εξειδικευμένων γλωσσών και εργαλείων για την έκφραση σχηματομορφών όπως συνήθως χρησιμοποιείται σε αντίστοιχα πλαίσια. Ο μετασχηματισμός που ορίστηκε μεταβιβάζει σε ικανοποιητικό βαθμό τις πληροφορίες που φέρουν τα Διαγράμματα Ακολουθίας σε ένα τυπικό μοντέλο όπως τα Δίκτυα Petri το οποίο είναι διαθέσιμο για ανάλυση και έλεγχο. Ο μετασχηματισμός αυτός θα μπορούσε να βρει εφαρμογή και ένα σύνολο περιπτώσεων πέρα από την ταύτιση σχηματομορφών για την οποία χρησιμοποιήθηκε στην παρούσα εργασία, όπως ο έλεγχος και η προσομοίωση εκτέλεσης των μοντέλων, κατά το σχεδιασμό. Παρόλα αυτά, οι κανόνες που έχουν υλοποιηθεί αφορούν ένα μόνο υποσύνολο των εκφραστικών μέσων που διαθέτουν τα Διαγράμματα Ακολουθίας, ενώ για την ενσωμάτωση χαρακτηριστικών απολύτου χρόνου θα πρέπει να επεκταθεί περαιτέρω το μεταμοντέλο των Δικτύων Petri που ορίστηκε. Επίσης, αντί του ορισμού μιας επέκτασης του βασικού μοντέλου Δικτύων Petri θα μπορούσε ενδεχομένως να χρησιμοποιηθεί κάποιος τύπος Δικτύων Petri υψηλού επιπέδου, όπως Έγχρωμα Δίκτυα Petri ή Predicate/Transition Nets για τα οποία, και ειδικά για τα Έγχρωμα Δίκτυα Petri, υπάρχει διαθέσιμο ένα πλήθος εργαλείων ανάλυσης.

Η διαδικασία της ανάλυσης και η ικανότητα προσαρμογής της τόσο για ανάλυση αρχείων όσο και για επιγραμμική ανάλυση, δίνει τη δυνατότητα προσδιορισμού ενός συνολικού

αποτελεσματικού «πρωτοκόλλου» ανάλυσης κάποιου συστήματος μέσω του συνδυασμού των μεθόδων αυτών. Επίσης, η ενορχήστρωση Java bytecode και όχι πηγαίου κώδικα, επιτρέπει τη χρήση του πλαισίου και σε περιπτώσεις που ο πηγαίος κώδικας δεν είναι διαθέσιμος. Από την άλλη το μειονέκτημα της τεχνικής αυτής είναι ότι δε μπορεί να χρησιμοποιηθεί για δομοστοιχεία που έχουν υλοποιηθεί σε άλλες γλώσσες και στις περιπτώσεις αυτές πρέπει να επιστρατεύεται κάποια άλλη μέθοδος ενορχήστρωσης. Τέλος, η «δυναμική» ανάλυση που επιλέχθηκε, απαιτεί την είσοδο στη διαδικασία δεδομένων ή πληροφοριών που μπορούν να χρησιμοποιηθούν για την αποτίμηση των εκφράσεων φύλαξης. Παρόλα αυτά, θα μπορούσε το ενδεχόμενο ταύτισης να εξετάζεται εξαντλητικά και ως προς τα σημεία επιλογής (πέρα δηλαδή από τις περιπτώσεις συνδρόμων ταυτόσημων γεγονότων) ή στοχαστικά. Μάλιστα για την πρώτη περίπτωση, το πλήθος των μονοπατιών εκτέλεσης και άρα το μέγιστο πλήθος επαναλήψεων ως προς τα σημεία αυτά δίνεται από την κυκλωματική πολυπλοκότητα.

Τελικά, μέσω του περιβάλλοντος-πλαισίου που υλοποιήθηκε δόθηκε η δυνατότητα παρακολούθησης και ανάλυσης της συμπεριφοράς ενός συστήματος, δίνοντας έμφαση στις αλληλεπιδράσεις μεταξύ αντικειμένων, ανεξαρτήτως του αν αυτά συναποτελούν κάποιο κεντρικό σύστημα, ή αν κατανέμονται και λειτουργούν αυτόνομα. Δόθηκε επίσης η δυνατότητα εντοπισμού προβλημάτων σε ένα σύστημα σε επίπεδο δομοστοιχείου αλλά και σε επίπεδο συγκεκριμένης μεθόδου ή συνόλων δομοστοιχείων και μεθόδων. Επίσης, η επιγραμμική ανάλυση μπορεί να αποτελέσει την υποδομή για την εφαρμογή τεχνικών πρόβλεψης ή αποτροπής (π.χ. σε περιπτώσεις επιθέσεων) μέσω της χρήσης κατάλληλων σεναρίων. Η επαλήθευση ή η απόρριψη κάποιου σεναρίου μπορεί να συνδεθεί άμεσα με κατάλληλες ενέργειες ειδοποίησης ή επαναφοράς/αποκατάστασης.

7.2 Μελλοντικές επεκτάσεις

Στα πλαίσια της παρακολούθησης και της ανάλυσης συστημάτων μια πρώτη επέκταση του πλαισίου, όπως αναφέρθηκε και προηγουμένως θα ήταν η υποστήριξη από το μετασχηματισμό επιπλέον εκφραστικών μέσων των Διαγραμμάτων Ακολουθίας, όπως συνδυασμένα τμήματα με τους υπόλοιπους χειριστές πέρα των alt, opt, par και loop, το πλαίσιο αναφοράς σε άλλα Διαγράμματα Ακολουθίας με χειριστή ref, χαρακτηριστικά απολύτου χρόνου και άλλα. Μια επέκταση που θα περιλαμβάνει χαρακτηριστικά απολύτου χρόνου προϋποθέτει επίσης την επέκταση του μεταμοντέλου των Δικτύων Petri για την προσθήκη μεταβάσεων χρόνου. Επίσης, μια πρακτική εξέλιξη θα ήταν η υποστήριξη τόσο σε επίπεδο probing όσο και στο εργαλείο ανάλυση, του μοντέλου CBE ως μορφότυπο των λημμάτων καταγραφής.

Επιπλέον, μια πρώτη βελτίωση του αλγόριθμου ελέγχου του μοντέλου ως προς τη δυνατότητα επαλήθευσης του σεναρίου θα μπορούσε να είναι η χρήση της στοίβα των σημείων επαναφοράς μόνο στις περιπτώσεις «ταυτόσημων» γεγονότων. Ακόμη, μια σημαντική βελτίωση θα ήταν η τροποποίηση του αλγορίθμου ώστε κατά την επιγραμμική ανάλυση να μειώνει στο ελάχιστο τις εγγραφές που τηρούνται στη δομή που χρησιμοποιείται.

Με ορισμένες τροποποιήσεις το πλαίσιο μπορεί να χρησιμοποιηθεί για την αναγνώριση διαφόρων τύπων μοτίβων σε ακολουθίες καταγραφής, χωρίς απαραίτητα στα σενάρια να μοντελοποιούνται αλληλεπιδράσεις.

Τέλος, μια ιδιαίτερα ενδιαφέρουσα επέκταση του πλαισίου θα ήταν η ενσωμάτωσή του σε κάποιο εργαλείο για τον έλεγχο τήρησης συμβολαίων επιπέδου υπηρεσίας (*Service Level Agreement*) που συνάπτονται μεταξύ παροχέων υπηρεσιών και πελατών που τις χρησιμοποιούν. Όσο η επέκταση των αρχιτεκτονικών προσανατολισμένων σε υπηρεσίες συνεχίζεται με ραγδαίους ρυθμούς, η ανάγκη τέτοιων ελεγκτών θα γίνεται όλο και εντονότερη.

Κεφάλαιο 8: Βιβλιογραφία

[PN81] Plattner B., Nievergelt J., “Monitoring Program Execution: A Survey”. Computer, vol. 14, pages 76-93, November 1981.

[KHC91] Kishon A., Hudak P., C. Consel C., “Monitoring Semantics: A Formal Framework for Specifying, Implementing, and Reasoning about Execution Monitors”, Proceedings of the ACM SIGPLAN conference on Programming language design and implementation, 1991.

[Thi03] Thiemann P., “Program specialization for execution monitoring”. Journal of Functional Programming, v.13 n.3, p.573-600, May 2003.

[JD02] Jahier E., Ducassé M., “Generic Program Monitoring by Trace Analysis”. Theory and Practice of Logic Programming. Volume 2, Issue 4-5, pp. 611 – 643, July 2002.

[HKM+94] Hofmann R., Klar R., Mohr B., Quick A., Siegle M., “Distributed Performance Monitoring: Methods, Tools and Applications”. IEEE Transactions on Parallel and Distributed Systems, 1994.

[FF95] Feather M., Fickas S., "Requirements Monitoring in Dynamic Environments". Proc. of Int. Conf. on Requirements Engineering, 1995.

[SM04] Spanoudakis G., Mahbub K., "Requirements Monitoring for Service-Based Systems: Towards a framework based on Event Calculus". Proceedings of the 19th International Conference on Automated Software Engineering, 2004.

[CFN+97] Cohen D., Feather M., Narayanaswamy K., Fickas S., "Automatic Monitoring of Software Requirements". 1997.

[Rob02] Robinson W., "Monitoring Software Requirements using Instrumented Code". Proc. of the Hawaii Int. Conf. on Systems Sciences, 2002.

[MMK+04] MoonZoo K., Mahesh V., Sampath K., Insup L., Sokolsky o., "Java-MaC: A Run-Time Assurance Approach for Java Programs", Formal Methods in System Design, Volume 24, Number 2, pages 129-155, March, 2004.

[KV02] Kemmerer R., Vigna G., "Intrusion Detection: A Brief History and Overview". Security & Privacy, 2002.

[KS94] Kumar S., Spafford E., "A Pattern Matching Model for Misuse Intrustion Detection". Proceedings of the National Computer Security Conference, pp. 11-21, Octomber 1994.

[HFT98] Ho Y., Frincke D., Tobin D., "Planning, Petri Nets, and Intrusion Detection", 21st Proc. National Information Systems Security Conference, 1998.

[RN95] Russell S., Norvig P., "Artificial Intelligence - A Modern Approach". Prentice Hall, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1995.

[Ath07] Αθανασόπουλος Μ., "Web mining: μια βιβλιογραφική μελέτη", http://users.ntua.gr/el02134/db2/webmining.pdf

[EK98] Elkoutbi M., Keller R., "Modeling Interactive Systems with Hierarchical Colored Petri Nets". Proc. of the Conference on High Performance Computing, Boston, April 1998

[BDM02] Bernardi S., Donatelli S., Merseguer J., "From UML Sequence Diagrams and Statecharts to analysable Petri Net models". 2002.

[SS00] Saldhana J., Shatz S., "UML Diagrams to Object Petri Net Models: An Approach for Modeling and Analysis". 2000.

[Lak94] Lakos C., "Object Petri Nets – Definition and relationship to colored nets". Tech Report TR 94-3, Computer Science Dept, University of Tasmania.

[RF06] Ribeiro O., Fernandes J., "Some Rules to Transform Sequence Diagrams into Coloured Petri Nets", 7th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, pages 237–56, 2006.

[FTJ+07] Fernandes J., Tjell S., Jorgensen J., Ribeiro O., "Designing Tool Support for Translating Use Cases and UML 2.0 Sequence Diagrams into a Coloured Petri Net", Sixth International Workshop on Scenarios and State Machines (SCESM'07), 2007.

[EFM+05] Eichner C.,Fleischhack H., Meyer R., Schrimpf U., Stehno C., "Compositional Semantics for UML 2.0 Sequence Diagrams Using Petri Nets". SDL 2005: Model Driven Systems Design, volume 3530 of Lecture Notes in Computer Science, pages 133–48, Springer, 2005.

[JLD05] Juhs G., Lorenz R., Desel J., "Can I Execute My Scenario in Your Net?". G. Ciardo and P. Darondeau (Eds.): ICATPN 2005, LNCS 3536, pp. 289–308. Springer-Verlag Berlin Heidelberg 2005.

[Cla00] Clarke E., "Model Checking", Handbook of Automated Reasoning, 2000.

[CCO+04] Chaki S., Clarke E., Ouaknine J., Sharygina N., Sinha N., "State/Event-Based Software Model Checking". E. Boiten, J. Derrick, G. Smith (Eds.): IFM 2004, LNCS 2999, pp. 128–147, Springer-Verlag Berlin Heidelberg, 2004.

[BH04] Bell A., Haverkort B., "Sequential and distributed model checking of Petri nets". Int J Softw Tools Technol Transfer (2005) 7: 43–60. Springer-Verlag 2004

[MMV08] Massart T., Meuter C., Van Begin L, "On the Complexity of Partial Order Trace Model Checking". Information Processing Letters 106(3): 120-126 (2008)

[RK08] Razavi A., Kontogiannis K., "Pattern Matching Framework for Policy Driven Software Monitoring", 2008.

[Aar05] Aarniala J., "Instrumenting Java bytecode", Seminar work for the Compilers-course, Department of Computer Science, University of Helsinki, Finland, Spring 2005.
http://www.cs.helsinki.fi/u/pohjalai/k05/okk/seminar/Aarniala-instrumenting.pdf

[CH06] Czarnecki K., Helsen S., "Feature-based survey of model transformation approaches". IBM Systems Journal, vol. 45, No. 3, 2006.

[Mur89] Murata T., "Petri Nets: Properties, Analysis and Applications", Proceedings of the IEEE, vol. 77, No. 4, April 1989

[Pet62] Petri C. A., "Kommunikation mit Automaten", Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962

[Pet77] Peterson J., "Petri Nets", Computing Surveys, Vol. 9, No 3, September 1977

[Gen87] Genrich H. J., "Predicate/Transition Nets", LNCS vol. 254, Springer Verlag, 1987

[PM90] de Paoli F., Morasca S., Proceedings of Computer Software and Applications Conference, 1990 (COMPSAC 90), pp 414-419, 31 Oct-2 Nov 1990, Chicago, USA

[CBE] Common Base Event specification
http://www.ibm.com/developerworks/library/specification/ws-cbe/

[ACT] Autonomic Computing Toolkit
http://www.ibm.com/developerworks/autonomic/overview.html

[ALF] Apache HTTP Server Log Formats
http://httpd.apache.org/docs/2.0/logs.html

[JLOG] Java Logging API
http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/

[GLA] Generic Log Adapter
http://help.eclipse.org/help31/index.jsp?topic=/org.eclipse.tptp.monitoring.doc.user/concepts/cintro.html

[TPTP] Test & Performance Toolkit Platform
http://www.eclipse.org/tptp/

[SUNMON] Sun Java Enterprise System 5 Monitoring Guide
http://docs.sun.com/app/docs/doc/819-5081

[OMG] Object Management Group, Inc.
http://www.omg.org/

[MOF] Meta-Object Facility specification
http://www.omg.org/technology/documents/modeling_spec_catalog.htm#MOF

[XMI] MOF / XMI Mapping specification
http://www.omg.org/technology/documents/modeling_spec_catalog.htm#XMI

[OCL] Object Constraint Language specification
http://www.omg.org/technology/documents/modeling_spec_catalog.htm#OCL

[EMF] Eclipse Modeling Framework
http://www.eclipse.org/modeling/emf/

[UML2] EMF UML2 project
http://www.eclipse.org/modeling/mdt/?project=uml2

[QVT] MOF Query / Views / Transformations
http://www.omg.org/technology/documents/modeling_spec_catalog.htm#MOF_QVT

[ATL] ATL – Atlas Transformation Language
http://www.eclipse.org/m2m/atl/

[ATL UM] ATL User Manual
http://www.eclipse.org/m2m/atl/doc/ATL_User_Manual%5Bv0.7%5D.pdf

[M2M] M2M Project
http://www.eclipse.org/m2m/

[PTTOOL] Petri Nets World: All Records in Petri Nets Tool Database
http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/complete_db.html

[TINA] Tina, Time Petri Net Analyzer
http://www.laas.fr/tina/

[PIPE2] Platform Independent Petri net Editor 2
http://pipe2.sourceforge.net/

[PNML] Petri Net Markup Language

http://www2.informatik.hu-berlin.de/top/pnml/about.html

[GRAATT] Graphviz, AT&T

http://www.graphviz.org/

[UML] UML Infrastructure & Superstructure specifications

http://www.omg.org/spec/UML/2.1.2/

[PROBE] TPTP Probekit

http://help.eclipse.org/help31/index.jsp?topic=/org.eclipse.hyades.probekit.doc.user/topics/c_pk_intro.htm