Ατομική Διπλωματική Εργασία

ΠΡΟΔΙΑΓΡΑΦΗ ΚΑΙ ΑΞΙΟΛΟΓΗΣΗ ΧΡΟΝΙΣΜΕΝΩΝ ΚΑΤΑΝΕΜΗΜΕΝΩΝ ΑΛΓΟΡΙΘΜΩΝ ΕΚΛΟΓΗΣ ΑΡΧΗΓΟΥ ΜΕ ΤΗ ΧΡΗΣΗ ΤΟΥ ΕΡΓΑΛΕΙΟΥ TEMPO

Χριστίνα Γεωργίου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2013

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΡΟΔΙΑΓΡΑΦΗ ΚΑΙ ΑΞΙΟΛΟΓΗΣΗ ΧΡΟΝΙΣΜΕΝΩΝ ΚΑΤΑΝΕΜΗΜΕΝΩΝ ΑΛΓΟΡΙΘΜΩΝ ΕΚΛΟΓΗΣ ΑΡΧΗΓΟΥ ΜΕ ΤΗ ΧΡΗΣΗ ΤΟΥ ΕΡΓΑΛΕΙΟΥ ΤΕΜΡΟ

Χριστίνα Γεωργίου

Επιβλέπων Καθηγητής
Δρ. Χρύσης Γεωργίου

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

Μάιος 2013

Ευχαριστίες

Θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον επιβλέποντα καθηγητή μου Δρ. Χρύση Γεωργίου, για την ευκαιρία που μου έδωσε να ασχοληθώ με το συγκεκριμένο θέμα και για την εμπιστοσύνη που μου έδειξε στην εκπόνηση της παρούσας διπλωματικής εργασίας.

Ως επιβλέπων καθηγητής, μου έθεσε τις κατευθυντήριες γραμμές και μου πρόσφερε τα απαραίτητα εφόδια για την επιτυχή ολοκλήρωση της εργασίας αυτής. Θα ήθελα να τον ευχαριστήσω για την εξαιρετική συνεργασία, την αμέριστη υποστήριξη, την υπομονή, το ενδιαφέρον και γενικά την πολύτιμη βοήθεια που μου πρόσφερε καθ' όλη τη διάρκεια αυτής της διπλωματικής εργασίας.

Επίσης, θα ήθελα να ευχαριστήσω τους φίλους και συμφοιτητές μου για την υποστήριξη και τη βοήθειά τους. Ιδιαίτερες ευχαριστίες εκφράζω στο φίλο και συμφοιτητή μου Μάριο Μιχαήλ για την πολύτιμη βοήθειά του κατά τη διάρκεια της πειραματικής αξιολόγησης της διπλωματικής μου εργασίας.

Τέλος, θα ήθελα ευχαριστήσω τους γονείς μου, στους οποίους και αφιερώνω την παρούσα εργασία, για την αμέριστη συμπαράσταση και καθοδήγηση που μου προσφέρουν όλα αυτά τα χρόνια.

Περίληψη

Κατά την ανάπτυξη και ολοκλήρωση της ερευνητικής μου εργασίας, ερευνήθηκαν και αναπτύχθηκαν θέματα που σχετίζονται με κατανεμημένους αλγορίθμους και συγκεκριμένα χρονισμένους κατανεμημένους αλγορίθμους που επιλύουν το πρόβλημα εκλογής αρχηγού.

Το πρόβλημα της εκλογής αρχηγού είναι το πρόβλημα της επιλογής μιας διεργασίας μεταξύ πολλών άλλων, προκειμένου να εκτελέσει ένα συγκεκριμένο καθήκον. Υπάρχει ένα ευρύ φάσμα από αλγόριθμους εκλογής αρχηγού, όπου ο κάθε ένας παρουσιάζει διαφορετική χρονική και επικοινωνιακή πολυπλοκότητα . Ποικίλουν ως προς τον τρόπο επικοινωνίας, την τοπολογία του δικτύου, τα κριτήρια εκλογής μιας διεργασίας αρχηγού και διάφορα άλλα στοιχεία. Η εκλογή αρχηγού είναι ένα από τα θεμελιώδη προβλήματα στη θεωρία των κατανεμημένων υπολογισμών και έχει πληθώρα εφαρμογών.

Στόχος της παρούσας διπλωματικής εργασίας είναι η μελέτη, η προδιαγραφή, η υλοποίηση και η αξιολόγηση συγκεκριμένων χρονισμένων κατανεμημένων αλγορίθμων εκλογής αρχηγού, με τη χρήση της γλώσσας TIOA και το εργαλείο Tempo. Η γλώσσα TIOA που βασίζεται στο μοντέλο Χρονισμένων Αυτομάτων Εισόδου/Εξόδου, είναι κατάλληλη για την περιγραφή τέτοιων αλγορίθμων. Βάσει αυτής έχει αναπτυχθεί και το εργαλείο Tempo, το οποίο χρησιμοποιήσαμε για την προσομοίωση των αλγορίθμων. Οι αλγόριθμοι εκλογής αρχηγού που μελετήθηκαν ήταν οι ακόλουθοι: ο αλγόριθμος με τοπολογία δέντρου, ο αλγόριθμος με τοπολογία δακτυλίου, ο αλγόριθμος με τοπολογία ισχυρά συνδεδεμένου γράφου και ένας τροποποιημένος αλγόριθμος με τοπολογία ισχυρά συνδεδεμένου γράφου. Αφού μελετήθηκαν εις βάθος οι συγκεκριμένοι αλγόριθμοι έγινε εκμάθηση της γλώσσα TIOA και του εργαλείου Tempo. Στη συνέχεια έγινε η υλοποίηση των αλγορίθμων χρησιμοποιώντας τα κανάλια επικοινωνίας MPI και TCP. Χρησιμοποιώντας το tempo2java plugin του εργαλείου Tempo, μεταφράσαμε τις προδιαγραφές των αλγορίθμων σε κώδικες της γλώσσας προγραμματισμού Java, οι οποίοι χρησιμοποιήθηκαν για την πειραματική αξιολόγηση των αλγορίθμων. Μέσα από μια σειρά πειραματικών εκτελέσεων έγινε αποσφαλμάτωση των υλοποιήσεών μας και ακολούθησε η αξιολόγησή τους σε μια συστοιχία υπολογιστών στο τοπικό δίκτυο του Πανεπιστημίου Κύπρου, χρησιμοποιώντας τη βιβλιοθήκη MPJ Express. Ένας από τους υλοποιημένους αλγορίθμους, υλοποιημένος με το κανάλι επικοινωνίας TCP, αξιολογήθηκε στο ασύγχρονο και ρεαλιστικό δίκτυο PlanetLab. Η εμπειρική αξιολόγηση των αλγορίθμων οδήγησε στο συμπέρασμα ότι η πειραματική τους

αξιολόγηση συνάδει με τη θεωρητική τους ανάλυση καθώς επίσης και πως η συμπεριφορά των αλγορίθμων εξαρτάται από την ανοχή των σφαλμάτων και τις τοπολογίες στις οποίες υλοποιούνται.

Κατά τη διάρκεια της διπλωματικής αυτής εργασίας, αποκόμισα αρκετές γνώσεις και εμπειρίες γύρω από τον τομέα των κατανεμημένων αλγορίθμων. Επίσης, είχα την ευκαιρία να εξοικειωθώ με τη γλώσσα TIOA, το μοντέλο Χρονισμένων Αυτομάτων Εισόδου/Εξόδου, το εργαλείο Tempo, τα κανάλια επικοινωνίας μεταξύ των διεργασιών MPI και TCP, τη βιβλιοθήκη MPJ Express καθώς επίσης και με το κατανεμημένο δίκτυο PlanetLab.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός και Κίνητρο Διπλωματικής Εργασίας

Ο έλεγχος ορθότητας των κατανεμημένων αλγορίθμων αποτελεί μεγάλη πρόκληση. Για το λόγο αυτό αναπτύχθηκαν τυπικές μεθόδοι για την προδιαγραφή και την απόδειξη της ορθότητάς τους. Το μοντέλο Χρονισμένων Αυτομάτων Εισόδου/Εξόδου [4,7,22] αποτελεί μια δημοφιλή τυπική μέθοδο. Σκοπός της παρούσας διπλωματικής εργασίας είναι η εκμάθηση της γλώσσας TIOA [17] και του εργαλείου Tempo [15] και η χρήση τους για τον προσδιορισμό και την υλοποίηση επιλεγμένων κατανεμημένων αλγορίθμων για το πρόβλημα Εκλογής Αρχηγού (Leader Election) [21].

Τα Κατανεμημένα Συστήματα [20] είναι μια συλλογή από ανεξάρτητους υπολογιστές, οι οποίοι εμφανίζονται στους χρήστες τους ως ένα ενιαίο συνεκτικό σύστημα. Τα συστήματα αυτά στηρίζονται στον Κατανεμημένο Υπολογισμό [20] όπου εκτελούνται συλλογικά υπολογιστικά βήματα από γεωγραφικά απομακρυσμένες συντονισμένες υπολογιστικές οντότητες και το κόστος του υπολογισμού έγκειται κυρίως από τον έλεγχο και την επικοινωνία δεδομένων. Οι αλγόριθμοι που σχεδίαζονται για τα Κατανεμημένα Συστήματα ονομάζονται Κατανεμημένοι Αλγόριθμοι [20] και είναι οι αλγόριθμοι που εκτελούνται ταυτόχρονα από πολλούς υπολογιστικούς κόμβους, οι οποίοι μπορεί να βρίσκονται σε απομακρυσμένες μεταξύ τους γεωγραφικές τοποθεσίες. Η σπουδαιότητα και η χρησιμότητα

των κατανεμημένων αλγορίθμων έγκειται στο γεγονός πως τα κατανεμημένα συστήματα αποτελούν τη βάση των σύγχρονων τεχνολογιών συνεργασίας, διαμοίρασης πόρων και επίτευξης υψηλής απόδοσης επεξεργασίας σε περιβάλλοντα δικτυακής επικοινωνίας.

Η γλώσσα TIOA [17] είναι μια γλώσσα προγραμματισμού και μοντελοποίησης βασισμένη στο μοντέλο Χρονισμένων Αυτομάτων Εισόδου/Εξόδου (TIOA) [4,7,22] και είναι ιδιαίτερα κατάλληλη για την προδιαγραφή και προσομοίωση χρονισμένων κατανεμημένων αλγορίθμων που η λειτουργία τους βασίζεται σε προθεσμίες βάσει χρόνου (timed-based timeouts). Βάσει της γλώσσας αυτής έχει αναπτυχθεί το εργαλείο Tempo [15], το οποίο παρέχει τη δυνατότητα προδιαγραφής, προσομοίωσης και αυτοματοποιημένης υλοποίησης χρονισμένων κατανεμημένων αλγορίθμων.

Το πρόβλημα της Εκλογής Αρχηγού [21] είναι ένα ευρέως διαδεδομένο και μελετημένο πρόβλημα κατανεμημένου υπολογισμού, στο οποίο δεδομένου ενός συνόλου διεργασιών σε ένα δίκτυο, επιλέγεται μία από αυτές ως αρχηγός. Υπάρχει πληθώρα αλγορίθμων που επιλύουν το συγκεκριμένο πρόβλημα, που διαφέρουν κυρίως ως προς την τοπολογία του δικτύου, την ανοχή σφαλμάτων, την κατάρρευση και επαναφορά κόμβων στο δίκτυο. Οι αλγόριθμοι εκλογής αρχηγού που μελετήθηκαν στην εργασία αυτή [21] είναι ο αλγόριθμος με τοπολογία δέντρου, ο αλγόριθμος με τοπολογία δακτυλίου [21], ο αλγόριθμος με τοπολογία ισχυρά συνδεδεμένου γράφου [1] καθώς επίσης και δύο εκδοχές ενός τροποποιημένου αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου [9]. Οι πιο πάνω αλγόριθμοι μελετήθηκαν εις βάθος και υλοποιήθηκαν οι προδιαγραφές τους στο εργαλείο Tempo, χρησιμοποιώντας τη γλώσσα TIOA. Η επικοινωνία μεταξύ των διεργασιών του αλγορίθμου υλοποιείται χρησιμοποιώντας τα κανάλια επικοινωνίας MPI και TCP Χρησιμοποιώντας το tempo2java plugin του εργαλείου Tempo, μεταφράσαμε τις προδιαγραφές σε κώδικα Java τις οποίες χρησιμοποιήσαμε για την πειραματική αξιολόγηση των αλγορίθμων. Μέσα από μια σειρά πειραματικών εκτελέσεων έγινε αποσφαλμάτωση των υλοποιήσεών μας και ακολούθησε η αξιολόγησή τους σε μια συστοιχία υπολογιστών στο τοπικό δίκτυο του Πανεπιστημίου Κύπρου. Ένας από τους υλοποιημένους αλγορίθμους, υλοποιημένος με το κανάλι επικοινωνίας TCP, αξιολογήθηκε στο κατανεμημένο δίκτυο PlanetLab [13,14], το οποίο παρουσιάζει ασυγχρονία, σφάλματα και απώλειες μηνυμάτων.

1.2 Μεθοδολογία

Για την αποπεράτωση της ερευνητικής μου εργασίας ακολουθήθηκε η εξής μεθοδολογία :

Αρχικά, έγινε εκμάθηση του μοντέλου χρονισμένων αυτομάτων Εισόδου/Εξόδου (TIOA), με μελέτη και προδιαγραφή απλών παραδειγμάτων. Για την καλύτερη κατανόηση του μοντέλου αυτού, χρειάστηκε η μελέτη και του μαθηματικού μοντέλου Αυτομάτων Εισόδου/Εξόδου (IOA) [10]. Στη συνέχεια, έγινε η εγκατάσταση και η εκμάθηση του εργαλείου Tempo [23], στο οποίο υλοποιήθηκαν διάφορα παραδείγματα που αποσκοπούσαν στην εξοικείωση του συγκεκριμένου εργαλείου. Στη συνέχεια έγινε μελέτη και κατανόηση συγκεκριμένων κατανεμημένων αλγορίθμων Εκλογής Αρχηγού. Αναγκαία ήταν η απόκτηση θεωρητικού υπόβαθρου για τα κατανεμημένα συστήματα [20], τους κατανεμημένους αλγορίθμους και το πρόβλημα εκλογής αρχηγού. Έπειτα έγινε η προδιαγραφή και η υλοποίηση συγκεκριμένων αλγορίθμων εκλογής αρχηγού στο εργαλείο Tempo. Για την υλοποίηση, χρειάστηκε η κατανόηση των καναλιών επικοινωνίας MPI και TCP [7]. Με το πέρας της υλοποίησης των αλγορίθμων, έγινε η αποσφαλμάτωση και η αξιολόγηση τους καθώς επίσης και η συλλογή των αποτελεσμάτων. Η αξιολόγηση έγινε αρχικά σε τοπικό υπολογιστή, στη συνέχεια σε μια συστοιχία μηχανών (cluster) στο τοπικό δίκτυο του Πανεπιστημίου Κύπρου και αργότερα στο κατανεμημένο δίκτυο PlanetLab. Για την πειραματική αξιολόγηση των αλγορίθμων ήταν αναγκαία η μελέτη της βιβλιοθήκης MPJ Express [11,12], η διερεύνηση της πλατφόρμας του PlanetLab και η εκμάθηση του τρόπου λειτουργίας της [14].

1.3 Δομή Διπλωματικής Εργασίας

Στο Κεφάλαιο 2, που ακολουθεί, παρέχεται ένα γνωστικό υπόβαθρο το οποίο βοηθάει στην καλύτερη κατανόηση του μοντέλου αυτομάτων Εισόδου/Εξόδου, του μοντέλου Χρονισμένων Αυτομάτων Εισόδου/Εξόδου, της γλώσσας TIOA, του εργαλείου Tempo, των καναλιών επικοινωνίας μεταξύ των διεργασιών και του κατανεμημένου δικτύου PlanetLab. Στο Κεφάλαιο 3 παρουσιάζεται το πρόβλημα Εκλογής Αρχηγού, περιγράφονται και αναλύονται οι αλγόριθμοι εκλογής αρχηγού που υλοποιήθηκαν. Στο Κεφάλαιο 4 γίνεται ανάλυση των προδιαγραφών και των υλοποιήσεων των αλγορίθμων που μελετήθηκαν στο εργαλείο Tempo με τη χρήση της γλώσσας TIOA. Στο Κεφάλαιο 5 παρουσιάζεται η πειραματική αξιολόγηση των αλγορίθμων που έγινε από την εκτέλεση του κώδικα σε γλώσσα προγραμματισμού Java που παίρνουμε από τις προδιαγραφές μας. Τέλος, στο Κεφάλαιο 6 αναφέρονται τα συμπεράσματα που εξάγονται από την προδιαγραφή και υλοποίηση των συγκεκριμένων αλγορίθμων που υλοποιήθηκαν, τα προβλήματα που αντιμετωπίστηκαν και πιθανή μελλοντική εργασία που πηγάζει από την εργασία αυτή.

Κεφάλαιο 2

Γνωστικό Υπόβαθρο

2.1 Μοντέλο Αυτομάτων Εισόδου / Εξόδου

Το μοντέλο Αυτομάτων Εισόδου / Εξόδου (Input / Output Automata) είναι ένα γενικό μοντέλο κατάλληλο για τη μοντελοποίηση ασύγχρονων κατανεμημένων συστημάτων, το οποίο αναπτύχθηκε για πρώτη φορά από τους Nancy A. Lynch και Mark R. Tuttle στο Massachusetts Institute of Technology [5,21]. Το μοντέλο αυτό χρησιμοποιείται για την ακριβή περιγραφή και επαλήθευση των συστατικών ενός κατανεμημένου συστήματος, όπως είναι οι διεργασίες και τα κανάλια επικοινωνίας. Είναι μια απλή μηχανή καταστάσεων στην οποία οι μεταβάσεις (transitions) συσχετίζονται με ενέργειες (actions), δηλαδή με την εκτέλεση μιας ενέργεια έχουμε μετάβαση από μια κατάσταση στην άλλη. Το αυτόματο χρησιμοποιεί τις ενέργειες για να επικοινωνεί με το περιβάλλον του, οι οποίες χωρίζονται σε ενέργειες εσόδου (input actions), ενέργειες εξόδου (output actions) και εσωτερικές ενέργειες (internal actions). Συγκεκριμένα, το αυτόματο επικοινωνεί με το περιβάλλον του μέσω των ενεργειών εισόδου και εξόδου που ονομάζονται και εξωτερικές ενέργειες. Οι εσωτερικές ενέργειες είναι ορατές μόνο από το ίδιο το αυτόματο και είναι υπεύθυνες για τις εσωτερικές

του λειτουργίες. Οι ενέργειες εξόδου και οι εσωτερικές ενέργειες είναι υπό τον έλεγχο του ίδιου του αυτομάτου ενώ οι ενέργειες εισόδου όχι.

Ένα αυτόματο Εισόδου / Εξόδου αποτελείται από τα εξής πέντε συστατικά :

- Την υπογραφή (Signature) sig(A), η οποία περιλαμβάνει το σύνολο όλων των ενεργειών του αυτομάτου.
- Ένα σύνολο καταστάσεων (States) states(A), το οποίο περιλαμβάνει όλες τις μεταβλητές καταστάσεων στις οποίες πιθανόν να βρίσκεται το αυτόματο.
- Ένα σύνολο αρχικών καταστάσεων (Start States) start(A), το οποίο περιλαμβάνει όλες τις αρχικές καταστάσεις του αυτομάτου και είναι ένα μη κενό υποσύνολο του συνόλου καταστάσεων.
- Ένα σύνολο μεταβάσεων (transitions relations) steps(A), το οποίο περιλαμβάνει μεταβάσεις ή αλλιώς βήματα (steps) της μορφής (state, action, state).
- Ένα σύνολο εργασιών tasks(A), το οποίο περιλαμβάνει το σύνολο των ενεργειών που ελέγχονται από το αυτόματο, δηλαδή εσωτερικές ενέργειες και ενέργειες εξόδου.

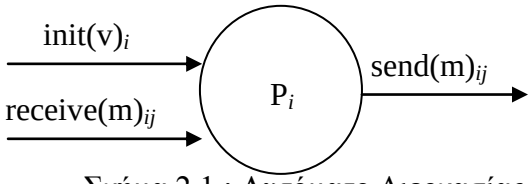
Οι ενέργειες εξόδου και οι εσωτερικές ενέργειες καθορίζονται από τη μορφή προϋπόθεση - συνέπεια (precondition - effect), όπου η προϋπόθεση υποδεικνύει πότε η ενέργεια μπορεί να εκτελεστεί, δηλαδή πότε η ενέργεια είναι ενεργή και η συνέπεια υποδεικνύει το αποτέλεσμα από την εκτέλεση της συγκεκριμένης ενέργειας. Μια ενέργεια π του αυτομάτου θεωρείται ότι είναι ενεργή (enabled) σε μία κατάσταση s, αν ισχύουν οι προϋποθέσεις της. Οι ενέργειες εισόδου είναι ενεργές σε κάθε κατάσταση του αυτομάτου και επομένως το αυτόματο δεν μπορεί να σταματήσει τη λειτουργία τους.

Τα αυτόματα Εισόδου / Εξόδου επιτρέπουν την σύνθεση (composition) διάφορων αυτομάτων που μοντελοποιούν απλούστερα συστήματα, με σκοπό τη δημιουργία ενός μεγαλύτερου αυτομάτου το οποίο μοντελοποιεί ένα πολύπλοκο σύστημα. Στην σύνθεση αυτομάτων, κάθε ενέργεια εξόδου π ενός αυτομάτου ταυτίζεται με την ενέργεια εισόδου π κάθε άλλου αυτομάτου. Δηλαδή, όταν ένα αυτόματο με ενέργεια εξόδου π, εκτελεί την συγκεκριμένη ενέργεια, τότε κάθε αυτόματο που έχει σαν ενέργεια εισόδου την ενέργεια π, την εκτελεί ταυτόχρονα. Βασική προϋπόθεση είναι οι δύο αυτές ενέργειες να έχουν τα ίδια ονόματα. Με αυτόν τον τρόπο επιτυγχάνεται η επικοινωνία μεταξύ αυτομάτων. Κατά τη σύνθεση αυτομάτων, οι εσωτερικές ενέργειες, οι ενέργειες εξόδου και οι ενέργειες εισόδου των επιμέρους αυτομάτων γίνονται εσωτερικές ενέργειες, ενέργειες εξόδου και ενέργειες εισόδου του αυτομάτου σύνθεσης, αντίστοιχα.

Αξίζει να σημειωθεί ότι στην σύνθεση αυτομάτων υπάρχουν κάποιοι περιορισμοί, όπως είναι οι εξής :

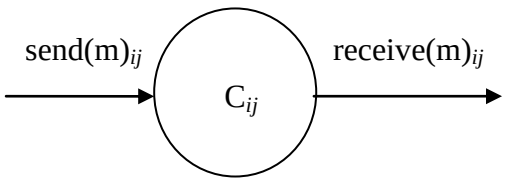
- Εφόσον οι εσωτερικές ενέργειες ενός αυτομάτου Α δεν είναι ορατές από κάποιο άλλο αυτόματο Β, τότε δεν επιτρέπεται η σύνθεση των αυτομάτων Α και Β, εκτός και αν το σύνολο των εσωτερικών ενεργειών του αυτομάτου Α δεν περιέχει ενέργειες που έχει το αυτόματο Β.
- Εφόσον μόνο ένα αυτόματο μπορεί να ελέγχει την εκτέλεση μίας ενέργειας, το σύνολο των εξωτερικών ενεργειών του αυτομάτου Α δεν μπορεί να περιέχει εξωτερικές ενέργειες του αυτομάτου Β και αντίστροφα.
- Κάθε ενέργεια του αυτομάτου σύνθεσης πρέπει να είναι ενέργεια μόνο πεπερασμένου αριθμού του συνόλου των επιμέρους αυτομάτων.

Παραδείγματα Αυτομάτων Εισόδου / Εξόδου: Ένα παράδειγμα ενός απλού αυτομάτου Εισόδου / Εξόδου είναι μια διεργασία σε ένα ασύγχρονο κατανεμημένο σύστημα, όπως φαίνεται στο Σχήμα 2.1. Ο κύκλος αντιπροσωπεύει το αυτόματο της διεργασία P_i , όπου *i* είναι ο δείκτης της διεργασίας. Τα βέλη που εισέρχονται και εξέρχονται του κύκλου, αντιπροσωπεύουν τις ενέργειες εισόδου και εξόδου του αυτομάτου, αντίστοιχα. Οι εσωτερικές ενέργειες του αυτομάτου δεν παρουσιάζονται στο σχήμα. Η διεργασία P_i αποτελείται από μια ενέργεια εισόδου init(v)_i η οποία αντιπροσωπεύει την παραλαβή μίας τιμής εισόδου v, μια ενέργεια εισόδου receive(m)_{ij}, η οποία αντιπροσωπεύει την παραλαβή ενός μηνύματος m από την διεργασία P_j και μια ενέργεια εξόδου send(m)_{ij} η οποία αντιπροσωπεύει την αποστολή ενός μηνύματος m στην διεργασία P_j.



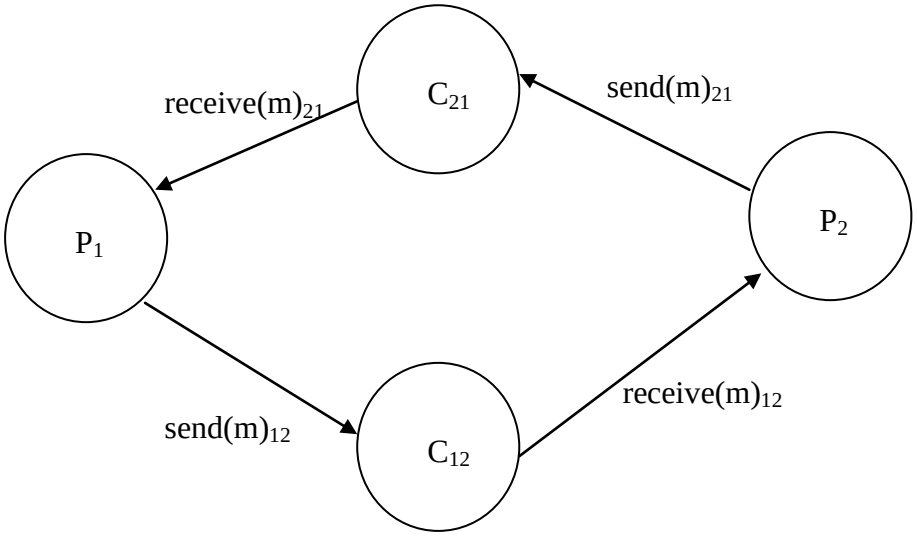
Σχήμα 2.1 : Αυτόματο Διεργασίας

Ακόμη ένα παράδειγμα αυτομάτου Εισόδου / Εξόδου είναι το κανάλι μηνυμάτων, όπως φαίνεται στο Σχήμα 2.2. Ο κύκλος αντιπροσωπεύει το αυτόματο του καναλιού C_{ij}, όπου *i,j* είναι οι δείκτες των διεργασιών που συνδέει το κανάλι. Το κανάλι C_{ij} αποτελείται από: μια ενέργεια εισόδου send(m)_{ij} και μια ενέργεια εξόδου receive(m)_{ij}.



Σχήμα 2.2 : Αυτόματο Καναλιού

Η σύνθεση αυτομάτων Εισόδου / Εξόδου διεργασιών και καναλιών, φαίνεται στο Σχήμα 2.3. Οι ενέργειες εξόδου του ενός αυτομάτου ταυτίζονται με τις ενέργειες εισόδου του άλλου αυτομάτου. Δηλαδή, κάθε ενέργεια εξόδου $send(m)_{ij}$ του αυτομάτου της διεργασίας P_i ταυτίζεται και εκτελείται ταυτόχρονα με μια ενέργεια εισόδου $send(m)_{ij}$ του αυτομάτου καναλιού C_{ij} και κάθε ενέργεια εισόδου $receive(m)_{ij}$ του αυτομάτου της διεργασίας P_i ταυτίζεται και εκτελείται ταυτόχρονα με μια ενέργεια εξόδου $receive(m)_{ij}$ του αυτομάτου καναλιού C_{ij}.



Σχήμα 2.3 : Σύνθεση αυτομάτων διεργασίας και καναλιών

2.2 Μοντέλο Χρονισμένων Αυτομάτων Εισόδου / Εξόδου

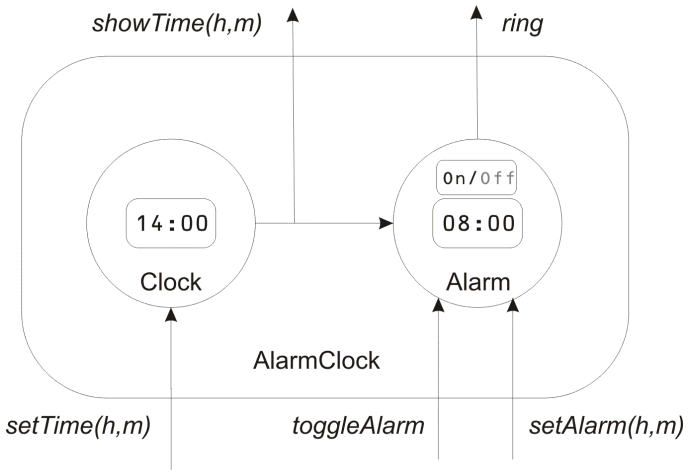
Το μοντέλο Χρονισμένων Αυτομάτων Εισόδου / Εξόδου (Timed Input / Output Automata) [4,7,22] αποτελεί μια εξέλιξη του μοντέλου Αυτομάτων Εισόδου / Εξόδου. Η λειτουργία τους βασίζεται σε προθεσμίες βάσει χρόνου (timed-based timeouts). Είναι συστήματα στα οποία η ορθότητα και η απόδοσή τους εξαρτώνται όχι μόνο από σειρά των γεγονότων αλλά και από τη χρονική στιγμή που συμβαίνει κάθε γεγονός. Χρησιμοποιούνται σε ένα ευρύ φάσμα εφαρμογών όπως στις επικοινωνίες και στα ενσωματωμένα συστήματα, στις οποίες δίνει ασφάλεια, αξιοπιστία και προβλεψιμότητα. Τα Χρονισμένα Αυτόματα Εισόδου / Εξόδου, μοντελοποιούν εκτός από τα συστατικά του συστήματος σε διακριτά βήματα και τη συνεχή εξέλιξη με την πάροδο του χρόνου, η οποία υλοποιείται με τη χρήση των τροχιών (trajectories). Η τροχιά είναι συνάρτηση που περιγράφει την εξέλιξη μιας μεταβλητής κατάστασης ανάμεσα σε διακριτές μεταβάσεις. Επομένως, οι καταστάσεις ενός τέτοιου

αυτομάτου μπορούν να αλλάζουν και με το πέρασμα του χρόνου. Όπως και τα αυτόματα Εισόδου / Εξόδου, τα χρονισμένα αυτόματα Εισόδου / Εξόδου είναι και αυτά μηχανές καταστάσεων και αποτελείται από τα εξής συστατικά :

- Ένα σύνολο εσωτερικών μεταβλητών X.
- Ένα σύνολο καταστάσεων Q.
- Ένα σύνολο αρχικών καταστάσεων Θ, το οποίο είναι ένα μη κενό υποσύνολο του συνόλου καταστάσεων.
- Ένα σύνολο εξωτερικών ενεργειών E και ένα σύνολο εσωτερικών ενεργειών H, τα οποία δεν έχουν κοινά στοιχεία μεταξύ τους.
- Ένα σύνολο διακριτών μεταβάσεων D, το οποίο περιλαμβάνει μεταβάσεις ή αλλιώς βήματα (steps), της μορφής (state, action, state).
- Ένα σύνολο τροχιών (trajectories) T.

Όπως και στα αυτόματα Εισόδου / Εξόδου και στα χρονισμένα αυτόματα Εισόδου / Εξόδου, επιτρέπεται η σύνθεση αυτομάτων. Πολύπλοκα συστήματα μπορούν να αναλυθούν σε υποσυστήματα, η σύνθεση των οποίων δίνει το ενιαίο πολύπλοκο σύστημα. Όλα τα αυτόματα σύνθεσης συμμετέχουν στο σύνολο τροχιών T ταυτόχρονα, με αποτέλεσμα να περνά ιδεατά το ίδιο χρονικό διάστημα. Επίσης, κάθε ενέργεια εξόδου κάθε αυτόματου ταυτίζεται με μιαν ενέργεια εισόδου κάθε άλλου αυτομάτου, όπου οι δύο αυτές ενέργειες έχουν το ίδιο όνομα.

Παράδειγμα Χρονισμένου Αυτομάτου Εισόδου / Εξόδου: Ένα παράδειγμα ενός απλού Χρονισμένου Αυτομάτου Εισόδου / Εξόδου είναι ένα ρολόι αφύπνισης (AlarmClock), το οποίο φαίνεται στο Σχήμα 2.4.



Σχήμα 2.4 : Αυτόματο AlarmClock [2]

Είναι η σύνθεση των Χρονισμένων Αυτομάτων Clock και Alarm, τα οποία είναι υπεύθυνα για την εμφάνιση του χρόνου σε ώρες και λεπτά και την αφύπνιση αντίστοιχα. Το αυτόματο AlarmClock αποτελείται από την ενέργεια εξόδου showTime(h.m) η οποία εμφανίζει τον χρόνο, την ενέργεια εξόδου ring κατά την οποία κτυπά το ρολόι, την ενέργεια εισόδου setTime η οποία καθορίζει τον χρόνο, την ενέργεια εισόδου setAlarm η οποία καθορίζει τον χρόνο αφύπνισης και την ενέργεια εισόδου toggleAlarm η οποία ενεργοποιεί/απενεργοποιεί την αφύπνιση. Αναλυτική περιγραφή του αυτομάτου Clock το οποίο χρησιμοποιεί το αυτόματο AlarmClock, θα γίνει στο επόμενο υποκεφάλαιο όπου θα περιγραφεί η γλώσσα που χρησιμοποιούν τα Χρονισμένα Αυτόματα Εισόδου / Εξόδου.

2.3 Εργαλείο Tempo και Γλώσσα TIOA

Στην ενότητα αυτή περιγράφεται το Εργαλείο Tempo το οποίο θα χρησιμοποιηθεί για την προδιαγραφή των Χρονισμένων Αλγορίθμων Εκλογής Αρχηγού που θα μελετηθούν στην παρούσα Διπλωματική Εργασία. Στη συνέχεια, γίνεται λεπτομερής περιγραφή της γλώσσας TIOA την οποία χρησιμοποιεί το εργαλείο Tempo για την υλοποίηση των Χρονισμένων Μοντέλων Εισόδου/Εξόδου.

2.3.1 ΕργαλείοTempo

Το εργαλείο Tempo αναπτύχθηκε από την Veromodo Inc, διανέμεται δωρεάν και χρησιμοποιείται για τη μοντελοποίηση κατανεμημένων συστημάτων [15]. Περιλαμβάνει εργαλεία για την ανάλυση, την προσομοίωση, τον μοντελοέλεγχο και την απόδειξη της ορθότητας των συστημάτων που περιγράφουν καθώς και την αυτοματοποιημένη μετάφραση του σε κώδικα Java. Τα μοντέλα που περιγράφονται στο εργαλείο αυτό, είναι μοντέλα χρονισμένων Αυτομάτων Εισόδου / Εξόδου και η γλώσσα που χρησιμοποιείται είναι η γλώσσα TIOA [6].

Μπορούμε να προμηθευτούμε το εργαλείο Tempo από την επίσημη σελίδα της Veromondo Inc, στην οποία υπάρχουν πληροφορίες για τη σωστή εγκατάσταση και εκτέλεσή του. Επίσης, υπάρχουν κατάλληλοι οδηγοί που επεξηγούν πλήρως απαιτήσεις, ανάγκες, περιορισμούς και τυχόν προβλήματα του εργαλείου [16,17,18,19]. Το Tempo παρέχει ένα άνετο και εύκολο στη χρήση περιβάλλον προς το χρήστη του. Η διεπαφή χρήστη φαίνεται στο Σχήμα 2.5 και αποτελείται από τα ακόλουθα τμήματα:

- Menu bar : Περιέχει τα μενού File, Edit, Navigate, Window, Tools και Help. Τα μενού File, Edit και Help προβάλουν καθορισμένες λειτουργίες για επεξεργασία κειμένων, όπως είναι η δημιουργία και αποθήκευση αρχείων, η αντιγραφή κειμένου και η βοήθεια. Το Navigate επιτρέπει στους χρήστες να μεταφερθούν σε ένα συγκεκριμένο μέρος του αρχείου. Το Window μενού επιτρέπει τον έλεγχο της εμφάνισης της διεπαφής. Τέλος με το Tools μενού έχουμε πρόσβαση στα διάφορα εργαλεία που υποστηρίζονται στο Tempo.
- Tool bar : Βρίσκεται ακριβώς κάτω από το menu bar και περιέχει τα εικονίδια των εργαλείων που περιέχονται στο εργαλείο Tempo.
- Status bar : Βρίσκεται στο κάτω μέρος της διεπαφής και παρέχει χρήσιμες πληροφορίες για την κατάστασή της, όπως για παράδειγμα τη θέση που βρίσκεται ο δείκτης μέσα σε ένα αρχείο, μια δεδομένη χρονική στιγμή.

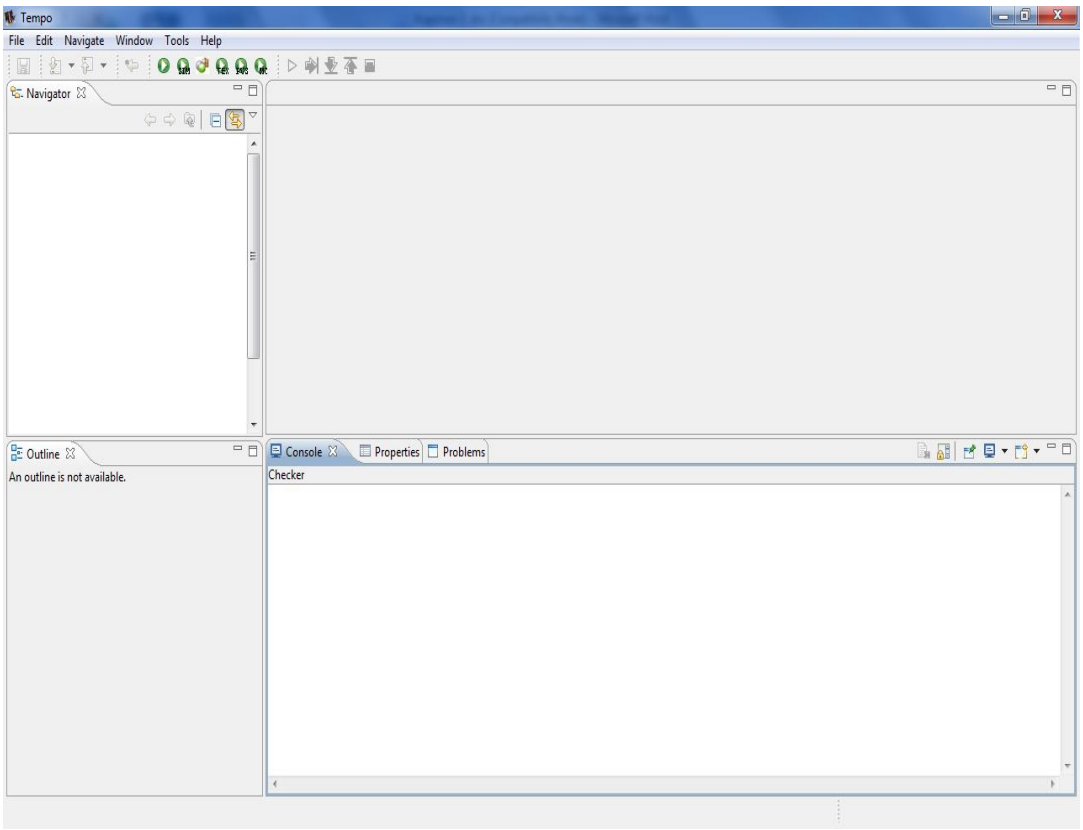
- Navigator : Περιέχει όλες τις εργασίες (projects) που βρίσκονται στο χώρο εργασίας και τα οποία εμφανίζονται με ιεραρχική δομή.
- Outline : Παρουσιάζει διαγραμματικά το περιεχόμενο του αρχείου που βρίσκεται ανοικτό και που εμφανίζεται στον κενό χώρο της διεπαφής (editor). Διακρίνονται οι ενέργειες, οι μεταβλητές, οι τροχιές και γενικά όλα τα τμήματα του αυτομάτου που ορίζεται στο συγκεκριμένο αρχείο. Για αρχεία που περιγράφουν λεξιλόγιο, δεν εμφανίζεται τίποτα.
- Editor : Στο τμήμα αυτό μπορούμε να επεξεργαστούμε τον κώδικα ενός αρχείου.
- Problems : Στο τμήμα αυτό επεξηγούνται τυχόν λάθη που αναγνωρίζονται από τον ελεγκτή του εργαλείου.
- Console : Παρουσιάζει την έξοδο μιας προδιαγραφής και τυχόν λάθη που αναγνωρίζονται κατά την εκτέλεσή της.

Τα εργαλεία που υπάρχουν στο Tempo είναι:

- Ο ελεγκτής (checker) που είναι υπεύθυνος για τον συντακτικό και τον σημασιολογικό έλεγχο της προδιαγραφής του αυτομάτου.
- Ο προσομοιωτής (simulator), ο οποίος χρησιμοποιείται για την προσομοίωση της εκτέλεσης του αυτομάτου.
- Ο μεταφραστής Java (Tempo2Java), ο οποίος μεταγλωττίζει την προδιαγραφή σε γλώσσα JAVA. Τα παραγόμενα αρχεία περιέχονται στο χώρο εργασίας, σε ένα φάκελο με το όνομα ‘Name.java’, όπου Name είναι το όνομα της εργασίας που μεταφράστηκε.

- Ο μεταφραστής LaTex (Tempo2Tex), ο οποίος μεταγλωττίζει την προδιαγραφή σε γλώσσα LaTex. Το παραγόμενο αρχείο περιέχεται στο χώρο εργασίας με το όνομα ‘Name.tex’, όπου Name είναι το όνομα της εργασίας που μεταφράστηκε.
- Το εργαλείο για αποδείξεις (PVS Theorem Prover Tool), το οποίο χρησιμοποιείται για τον έλεγχο της ορθότητας των αλγορίθμων που εκφράζονται σε γλώσσα TIOA.
- Ο μεταφραστής Uppaal (uppaal), ο οποίος μεταγλωττίζει την προδιαγραφή σε γλώσσα Uppaal.

Να σημειώσουμε ότι οι μεταφραστές LaTex, Uppaal και PVS, δεν εκτελούν κάποια λειτουργία στην διεπαφή του εργαλείου Tempo. Επίσης κάθε αρχείο που δημιουργείται στο εργαλείο, πρέπει να έχει την κατάληξη ‘.tioa’.



Σχήμα 2.5 Διεπαφή Χρήστη στο Εργαλείο Tempo (User Interface)

2.3.2 Γλώσσα TIOA

Η γλώσσα που χρησιμοποιήθηκε για την υλοποίηση των αλγορίθμων εκλογής αρχηγού είναι η γλώσσα TIOA [17,23]. Η γλώσσα TIOA βασίζεται στο μοντέλο Χρονισμένων Αυτομάτων Εισόδου / Εξόδου και είναι μια παραλλαγή της γλώσσας IOA που βασίζεται στο μοντέλο αυτομάτων Εισόδου / Εξόδου και που περιγράφει ασύγχρονους κατανεμημένους αλγορίθμους.

Στη συνέχεια θα παρουσιάσουμε όλα τα δομικά στοιχεία της γλώσσας TIOA. Αρχικά θα περιγράψουμε τους τύπους δεδομένων, τις εκφράσεις, τους τελεστές και τις εντολές που υπάρχουν στη συγκεκριμένη γλώσσα. Επίσης, θα περιγραφεί ο τρόπος με τον οποίο ορίζουμε λεξιλόγια, αυτόματα TIOA, υπογραφές αυτομάτων, μεταβλητές καταστάσεων, σχέσεις μεταβάσεων, τροχιές και αυτόματα σύνθεσης που είναι απαραίτητα για την προδιαγραφή αλγορίθμων σε γλώσσα TIOA.

Τύποι Δεδομένων

Η γλώσσα TIOA επιτρέπει στους χρήστες να καθορίσουν τις ενέργειες και τις καταστάσεις των αυτομάτων, χρησιμοποιώντας μαθηματικές εκφράσεις. Οι τύποι δεδομένων είναι στατικοί ή δυναμικοί. Ένας στατικός τύπος για μια μεταβλητή, περιγράφει την τιμή που παίρνει η συγκεκριμένη μεταβλητή, ενώ ένας δυναμικός τύπος για μια μεταβλητή είναι συνάρτηση που περιγράφει την εξέλιξη της συγκεκριμένης μεταβλητής στο χρόνο.

Οι τύποι δεδομένων που υποστηρίζονται στη γλώσσα TIOA είναι οι εξής:

- Boolean (Bool) : Είναι ο βασικότερος αρχέγονος τύπος δεδομένων και περιέχει δύο τιμές, 0 (false) και 1(true) που ονομάζονται boolean ή λογικές τιμές. Η γλώσσα TIOA υποστηρίζει τις ακόλουθες πράξεις σε μια boolean έκφραση: μεγαλύτερο (>), μικρότερο (<), ίσο (=), άνισο (≠), and (/\), or(\/), συνεπαγωγή (=>) και διπλή συνεπαγωγή (<=>). Επίσης, μπορούν να χρησιμοποιηθούν ο καθολικός (∀) και ο υπαρξιακός ποσοδείκτης (∃).
- Natural Numbers (Nat) : Είναι ένας φυσικός αριθμός. Το σύνολο των φυσικών αριθμών περιλαμβάνει όλους τους μη αρνητικούς ακέραιους αριθμούς. Υποστηρίζει τις συναρτήσεις: succ(x), η οποία επιστρέφει τον αμέσως επόμενο φυσικό αριθμό από το x, pred(x), η οποία επιστρέφει τον προηγούμενο φυσικό αριθμό από το x, min(x,y), η οποία επιστρέφει τον μικρότερο φυσικό αριθμό μεταξύ των x και y, max(x,y), η οποία επιστρέφει τον μεγαλύτερο φυσικό αριθμό μεταξύ των x και y, div(x,y), mod(x,y) καθώς επίσης και τις πράξεις: πρόσθεση (+), αφαίρεση (-), πολλαπλασιασμός (*), μεγαλύτερο (>), μεγαλύτερο και ίσο (≥), μικρότερο (<), μικρότερο και ίσο (≤), ίσο (=).
- Integers (Int) : Είναι ένας ακέραιος αριθμός. Υποστηρίζει όλες τις συναρτήσεις και όλες τις πράξεις των φυσικών αριθμών. Επιπλέον υποστηρίζει το αρνητικό πρόσημο (-) και την συνάρτηση abs(x), η οποία επιστρέφει την απόλυτη τιμή του x.
- Real Numbers (Real) : Είναι ένας πραγματικός αριθμός. Υποστηρίζει τις συναρτήσεις: min(x,y), η οποία επιστρέφει τον μικρότερο πραγματικό αριθμό μεταξύ των x και y,

max(x,y), η οποία επιστρέφει τον μεγαλύτερο πραγματικό αριθμό μεταξύ των x και y, abs(x), η οποία επιστρέφει την απόλυτη τιμή του x, floor(x), η οποία επιστρέφει τον μεγαλύτερο ακέραιο αριθμό που είναι μικρότερος ή ίσος με το x και όλες τις πράξεις των φυσικών αριθμών.

- Discrete Real Numbers (DiscreteReal) : Είναι ένας πραγματικός αριθμός, με τη διαφορά ότι δεν αλλάζει μεταξύ των διακριτών βημάτων. Είναι ιδεατός τύπος και χρησιμοποιείται για να ορίζει το φυσικό ρολόι ενός αυτομάτου. Η διαφορά του τύπου DiscreteReal με τον τύπο Real είναι πως ο πρώτος είναι μεταβλητός τύπος ενώ ο δεύτερος στατικός.
- Augmented Real Numbers (AugmentedReal) : Είναι ένας πραγματικός αριθμός συμπεριλαμβανομένων των τιμών $+\infty$ και $-\infty$.
- Characters (Char) : Είναι ένας χαρακτήρας και μπορεί να πάρει τιμές γράμματα και ψηφία. Υποστηρίζει τις συγκρίσεις μεταξύ δύο τιμών ($<,>,\leq,\geq$), μιας και η σύγκριση γίνεται στην τιμή του συγκεκριμένου χαρακτήρα στον πίνακα ASCII.
- Strings (String) : Είναι μια ακολουθία από χαρακτήρες. Μπορεί να αναπαρασταθεί και από τον τύπο Seq[E].
- Arrays (Array[T_1, … , T_n, E]) : Είναι ένας πίνακας. Για κάθε n>0, η δήλωση Array[T_1, … , T_n, E], αποτελεί ένα μονοδιάστατο πίνακα n στοιχείων (T_1, … , T_n) τύπου E.
- Set (Set[E]) : Είναι ένα σύνολο. Τα στοιχεία του Set[E] είναι πεπερασμένα σύνολα στοιχείων τύπου E. Υποστηρίζει τις ακόλουθες πράξεις συνόλων: ένωση ($\cup$), τομή ($\cap$), συμπλήρωμα ($-$), ισότητα (=), υποσύνολο ($\subset$), υποσύνολο ή ίσον ($\subseteq$), ανήκει ($\in$), δεν ανήκει (\notin), κενό σύνολο ($\varnothing$). Επίσης υποστηρίζει τις ακόλουθες συναρτήσεις: size(S), η οποία επιστρέφει το πλήθος των στοιχείων στο σύνολο S, insert(e,S), η οποία εισάγει το στοιχείο e στο σύνολο S και delete(e,S), η οποία διαγράφει το στοιχείο e από το σύνολο S.
- Seq (Seq[E]) : Είναι μια ουρά. Τα στοιχεία του Seq[E] είναι πεπερασμένες ουρές στοιχείων του τύπου E. Υποστηρίζει τις πράξεις: κενή ουρά ($\varnothing$), ανήκει ($\in$), δεν ανήκει (\notin), εισαγωγή στοιχείου (|-), εξαγωγή στοιχείου (-|), και τις συναρτήσεις: head(S), η οποία επιστρέφει το πρώτο στοιχείο της ουράς S, tail(S), η οποία επιστρέφει το τελευταίο στοιχείο της ουράς S, len(S), η οποία επιστρέφει το μήκος της ουράς S και S[n], η οποία επιστρέφει το n-οστό στοιχείο της ουράς S.

Εκφράσεις και Τελεστές

Στη γλώσσα TIOA, οι επιτρεπόμενες εκφράσεις που μπορούν να χρησιμοποιηθούν είναι οι ακόλουθες:

- Enumerations (Enumeration[e_1, … , e_n]) : Ο τύπος αυτός επιτρέπει την δημιουργία ενός νέου τύπου δεδομένου, ο οποίος αποτελείται από το σύνολο των στοιχείων e_1, … , e_n. Ορίζεται ως εξής: **types** Name : **Enumeration**[e_1, … , e_n], όπου Name είναι το όνομα του νέου τύπου δεδομένου και e_1, … , e_n είναι οι τιμές που μπορούν να πάρουν οι μεταβλητές αυτού του τύπου δεδομένων. Να σημειώσουμε ότι για έναν τύπο Name ισχύει η σειρά με την οποία δηλώνονται οι επιτρεπόμενες τιμές. Δηλαδή, εάν μία μεταβλητή c τύπου Name έχει τιμή e_1 και δηλώσω την έκφραση c:=c+1, τότε η τιμή c θα πάρει την τιμή e_2 .
- Unions (Union[f_1 : T_1, … , f_n : T_n]) : Ο τύπος αυτός επιτρέπει στα στοιχεία του να έχουν τιμή της οποίας ο τύπος δεδομένων να είναι ένας από τους τύπους δεδομένων T_1, … , T_n. Για παράδειγμα εάν ορίσουμε:

  OptionValue : Union[s:String, i:Int, n:Nat] και v:OptionValue

  τότε η μεταβλητή v μπορεί να είναι είτε συμβολοσειρά είτε ακέραιος αριθμός είτε φυσικός αριθμός.
- Tuples (Tuple[f_1 : T_1, … , f_n : T_n]) : Ο τύπος αυτός μας επιτρέπει να δηλώνουμε πιο σύνθετες μεταβλητές. Μεταβλητές που έχουν αυτόν τον τύπο δεδομένων είναι πλειάδες με πεδία f_1, … , f_n και τύπο δεδομένων T_1, … , T_n, αντίστοιχα. Για παράδειγμα εάν έχουμε μια μεταβλητή people τύπου Tuple[name:String, age:Int], τότε η μεταβλητή people αποτελείται από τα πεδία name και age, στα οποία έχουμε πρόσβαση μέσω του τελεστή '.'. Δηλαδή, people.name δηλώνει την συμβολοσειρά που συνδέεται με την μεταβλητή people και people.age δηλώνει τον ακέραιο αριθμό που συνδέεται με την μεταβλητή people.
- Συναρτήσεις (functions) : Η γλώσσα TIOA επιτρέπει στον χρήστη να δημιουργήσει τις δικές του συναρτήσεις και να τις χρησιμοποιήσει στις προδιαγραφές. Μία συνάρτηση μπορεί να οριστεί : έξω από το αυτόματο (global function),μέσα στο αυτόματο μετά από τις καταστάσεις του αυτομάτου, στον ορισμό μίας μετάβασης μετά τις τοπικές μεταβλητές και πριν από τις προϋποθέσεις, ή εντός του ορισμού της τροχιάς μετά από το όνομα και πριν από τις παραμέτρους της. Μια συνάρτηση ορίζεται με : τη λέξη κλειδί 'let', το όνομά της, τα ονόματα και τους τύπους των παραμέτρων της, τον τύπο επιστροφής της και τον ορισμό της. Για παράδειγμα, η πιο κάτω συνάρτηση παίρνει δύο φυσικούς αριθμούς hour και minute που εκφράζουν τον

χρόνο σε ώρες και λεπτά και χρησιμοποιείται για να ελέγχει τις τιμές των παραμέτρων της.

let legalTime(hour, minute: Nat) = minute < 60 /\ hour < 24

- Τελεστής Where : Ο τελεστής αυτός χρησιμοποιείται όταν θέλουμε να καθορίσουμε το πεδίο τιμών μιας μεταβλητής. Συνήθως, χρησιμοποιείται σε δηλώσεις ενεργειών και αυτομάτων για να περιορίσουμε τις τιμές των παραμέτρων που δέχονται. Για παράδειγμα εάν δηλώσουμε : **automaton Process (id : Nat) where id >0** , τότε η τιμή που μπορεί να πάρει μια διεργασία σαν παράμετρο πρέπει να είναι μεγαλύτερη από το μηδέν.
- Τελεστής Choose : Ο τελεστής αυτός χρησιμοποιείται για την επιλογή τυχαίων τιμών για τις μεταβλητές και συνδυάζεται με τον τελεστή where. Για παράδειγμα εάν δηλώσουμε: **value : int := choose n (where n≥0 \/ n≤10)** , τότε η μεταβλητή value παίρνει τυχαία μια τιμή μεταξύ του 0 και του 10, συμπεριλαμβανομένου. Εάν δηλώσουμε: **value : int := choose n** , τότε η τιμή μεταβλητή value παίρνει τυχαία μια τιμή από το πεδίο τιμών των ακεραίων αριθμών.

Λεξιλόγιο

Η γλώσσα TIOA επιτρέπει τη δημιουργία ενός λεξιλογίου (Vocabulary), στο οποίο μπορούμε να ορίσουμε δικούς μας τύπους δεδομένων και τελεστές. Ο ορισμός ενός λεξιλογίου αρχίζει με την λέξη κλειδί ‘vocabulary’ και ένα αναγνωριστικό που αντιστοιχεί στο όνομα του λεξιλογίου. Εάν θα συμπεριληφθούν άλλα λεξιλόγια, χρησιμοποιείται η λέξη κλειδί ‘imports’ και το όνομα του λεξιλογίου που θα συμπεριληφθεί. Κάθε λεξιλόγιο μπορεί να συμπεριλαμβάνει ένα ή περισσότερους τύπους (ακολουθώντας τη λέξη κλειδί ‘types’ ή ‘defines’) και μηδέν ή περισσότερους τελεστές (ακολουθώντας τη λέξη κλειδί ‘operators’). Κάθε τελεστής έχει μία υπογραφή, η οποία καθορίζει: τις παραμέτρους του, ακολουθεί το σύμβολο : , τους τύπους δεδομένων των παραμέτρων του, ακολουθεί το σύμβολο → και τον τύπο δεδομένου του αποτελέσματος του. Ο ορισμός ενός λεξιλογίου τελειώνει με την λέξη κλειδί ‘end’. Ακολουθεί ένα μικρό παράδειγμα λεξιλογίου, το οποίο καθορίζει τον τύπο δεδομένων Nat που αντιπροσωπεύει τους φυσικούς αριθμούς.

vocabulary Nat
types Nat
operators
succ, pred: Nat →Nat

+, - , *, **, min, max, div, mod : Nat, Nat → Nat

<, ≤, >, ≥, =, ≠ : Nat, Nat → Bool

end

Εντολές

Οι εντολές που χρησιμοποιούνται στην γλώσσα TIOA είναι οι εξής:

- Εντολή Ανάθεσης : Για να γίνει ανάθεση μιας τιμής σε μια μεταβλητή, χρησιμοποιείται το σύμβολο ‘:=’. Αριστερά και δεξιά του συμβόλου βρίσκεται η μεταβλητή και η τιμή που θα πάρει, αντίστοιχα.
- Εντολή Print : Η εντολή αυτή χρησιμοποιείται για να τυπώνει την τιμή μιας έκφρασης. Ορίζεται με τη λέξη κλειδί print, μία έκφραση και το σύμβολο ‘;’.
- Εντολή If-then-else : Μια τέτοια εντολή αποτελείται από τη λέξη κλειδί ‘if’, ένα λογικό κατηγόρημα με τιμές false και true, τη λέξη κλειδί ‘then’ και ένα σύνολο από εντολές οι οποίες θα εκτελεστούν όταν το λογικό κατηγόρημα που ακολουθεί το ‘if’ είναι αληθές. Στη συνέχεια ακολουθούν μηδέν ή περισσότερα τμήματα elseif, τα οποία ορίζονται από την λέξη κλειδί ‘elseif’, ένα λογικό κατηγόρημα, τη λέξη κλειδί ‘then’, ένα σύνολο εντολών που θα εκτελεστούν όταν το λογικό κατηγόρημα που ακολουθεί το ‘elseif’ είναι αληθές και τη λέξη κλειδί ‘if’. Οι εντολές elseif μπορεί να είναι φωλιασμένες. Εάν υπάρχει τουλάχιστον ένα τμήμα elseif, τότε ακολουθεί το τμήμα else το οποίο ορίζεται με τη λέξη κλειδί ‘else’, το σύνολο των εντολών που θα εκτελεστούν όταν όλα τα προηγούμενα λογικά κατηγορήματα είναι ψευδή και τελειώνει με τη λέξη κλειδί ‘fi’. Ένα παράδειγμα είναι το ακόλουθο, όπου υπολογίζεται ο μεγαλύτερος αριθμός μεταξύ των a και b.

```
if (a>b) then
  max := a;
else
  max := b;
fi;
```

- Εντολή While : Η εντολή αυτή αποτελείται από τη λέξη κλειδί ‘while’, ένα λογικό κατηγόρημα, τη λέξη κλειδί ‘do’, ένα σύνολο από εντολές που θα εκτελούνται μέχρι το λογικό κατηγόρημα γίνει ψευδές και τη λέξη κλειδί ‘od’. Οι εντολές while μπορεί να είναι φωλιασμένες. Ένα παράδειγμα είναι το ακόλουθο όπου τυπώνει τους αριθμούς από το 0 μέχρι και το 10.

```
x:=0;
while (x≤10)do
        print x; x := x + 1;
od
```

- Εντολή For : Η εντολή αυτή αποτελείται από τη λέξη κλειδί ‘for’, το όνομα της μεταβλητής που αποτελεί τον μετρητή, το σύμβολο ‘:’, τον τύπο της μεταβλητής, τη λέξη κλειδί ‘while’, τη συνθήκη επανάληψης, τη λέξη κλειδί ‘do’, ένα σύνολο από εντολές που θα εκτελούνται σε κάθε επανάληψη και τη λέξη κλειδί ‘od’ στο τέλος.
- Εντολή Fire : Η εντολή αυτή χρησιμοποιείται για να πυροδοτήσουμε κάποια μεταβατική ενέργεια. Η εντολή fire αποτελείται από τη λέξη κλειδί ‘fire’, το είδος της ενέργειας που θα πυροδοτηθεί (input, output, internal), το όνομα του αυτομάτου που περιέχει την ενέργεια, το σύμβολο ‘.’, το όνομα της ενέργειας, τις παραμέτρους που παίρνει η ενέργεια σε παρενθέσεις και το σύμβολο ‘;’.
- Εντολή Follow : Η εντολή αυτή χρησιμοποιείται για να εξελίξουμε κάποιες τροχιές. Η εντολή follow αποτελείται από τη λέξη κλειδί ‘follow’, το όνομα του αυτομάτου που περιέχει την τροχιά που θα εξελιχθεί, το σύμβολο ‘.’, το όνομα της τροχιάς, τη λέξη κλειδί ‘duration’ και το χρονικό διάστημα κατά τον οποίο θα εξελίσσεται η τροχιά.

Αυτόματο TIOA

Ορισμός Απλού Αυτομάτου TIOA: Η περιγραφή ενός αυτομάτου στην γλώσσα TIOA, αρχίζει με τη λέξη κλειδί ‘automaton’, ακολουθεί το όνομα του αυτομάτου και οι παράμετροι που παίρνει, εάν υπάρχουν. Επίσης, εάν ένα αυτόματο παίρνει παραμέτρους, τότε μπορεί να χρησιμοποιηθεί ο τελεστής where για να περιορίσουμε το πεδίο τιμών που μπορεί να πάρουν οι παράμετροι. Για παράδειγμα, ένα αυτόματο με τον ακόλουθο ορισμό:

automaton Clock

παίρνει όνομα Clock και δεν έχει καθόλου παραμέτρους.

Το αυτόματο αποτελείται από τα εξής τμήματα :

1. Υπογραφή Αυτομάτου (Action Signature)
2. Μεταβλητές Καταστάσεως (State Variables)
3. Σχέσεις Μεταβάσεων (Transitions Relations)
4. Τροχιές (Trajectories)

Υπογραφή Αυτομάτου: Η υπογραφή αυτομάτου είναι το τμήμα το οποίο περιλαμβάνει τις δηλώσεις όλων των ενεργειών που περιλαμβάνει το αυτόματο και πρέπει να υπάρχει όταν

ορίζουμε ένα αυτόματο. Αρχίζει με την λέξη κλειδί ‘signature’ και ακολουθούν οι εξωτερικές και εσωτερικές ενέργειες του αυτομάτου. Κάθε ενέργεια αποτελείται από το είδος της ενέργειας (input, output ή internal), το όνομα της ενέργειας και οι παράμετροι που παίρνει η συγκεκριμένη ενέργεια, εάν υπάρχουν. Κάθε παράμετρος δηλώνεται από το όνομα και τον τύπο της και χωρίζεται από την επόμενη παράμετρο με κόμμα. Για παράδειγμα, ένα αυτόματο με την ακόλουθη υπογραφή :

Signature

input setTime(hour, minute: Nat) **where** legalTime(hour, minute)

output showTime(hour, minute: Nat) **where** legalTime(hour, minute)

Αποτελείται από μία ενέργεια εξόδου με όνομα showTime και μια ενέργεια εισόδου με όνομα setTime, οι οποίες παίρνουν παραμέτρους δύο φυσικές μεταβλητές hour και minute που αντιστοιχούν στην ώρα και στα λεπτά.

Μεταβλητές Καταστάσεων: Οι μεταβλητές καταστάσεων είναι το τμήμα το οποίο περιλαμβάνει όλες τις μεταβητές του αυτομάτου και ακολουθεί το τμήμα υπογραφή. Όπως και η υπογραφή αυτομάτου και αυτό το τμήμα είναι υποχρεωτικό όταν ορίζουμε ένα αυτόματο. Αρχίζει με τη λέξη κλειδί ‘states’ και ακολουθούν οι δηλώσεις των μεταβλητών καταστάσεων που υπάρχουν στο αυτόματο μαζί με τον τύπο δεδομένων τους και την τιμή με την οποία αρχικοποιούνται. Η αρχικοποίηση μεταβλητών μπορεί να γίνει χρησιμοποιώντας και τους τελεστές where και choose. Ένα παράδειγμα δήλωσης μεταβλητών καταστάσεων είναι το ακόλουθο:

States

whatToShow: Nat := 0,

whenToShow: Nat := 0,

now: Real := 0

στο οποίο υπάρχει η φυσική μεταβλητή wantToShow η οποία αντιπροσωπεύει τον χρόνο που θα εμφανίζεται με την επόμενη εκτέλεση της ενέργειας showTime, η φυσική μεταβλητή whenToShow η οποία αντιπροσωπεύει πότε θα συμβεί η επόμενη ενέργεια showTime και η πραγματική μεταβλητή now η οποία αντιπροσωπεύει τον τρέχον χρόνο στο αυτόματο.

Εάν ο χρήστης θα ορίσει τυχόν συναρτήσεις τότε πρέπει να τις ορίσει αμέσως μετά τις μεταβλητές καταστάσεων και πριν από τις μεταβάσεις.

Σχέσεις Μεταβάσεων: Είναι ένα τμήμα υποχρεωτικό και ακολουθεί το τμήμα καταστάσεων μεταβάσεων. Εδώ ορίζονται οι ενέργειες του αυτομάτου που έχουν δηλωθεί στο τμήμα signature και πρέπει να ταιριάζουν ακριβώς και στο όνομα και στους τύπους δεδομένων.

Δηλαδή, ορίζεται το πως ένα αυτόματο αλλάζει όταν εκτελεστεί μια συγκεκριμένη ενέργεια. Για τον ορισμό των ενεργειών αρχίζουμε με τη δήλωσή της όπως αυτή υπάρχει στο τμήμα signature, χωρίς όμως τους τύπους δεδομένων των παραμέτρων. Στον ορισμό των μεταβάσεων είναι δυνατόν να οριστούν τοπικές μεταβλητές και γίνεται μετά το όνομα και τις παραμέτρους της ενέργειας και πριν τις προϋποθέσεις. Η σύνταξη για τον ορισμό τοπικών μεταβλητών γίνεται με τους ίδιους κανόνες όπως ακριβώς και στο τμήμα μεταβλητών καταστάσεων εκτός από το ότι ξεκινά με τη λέξη κλειδί 'locals' και όχι με τη λέξη κλειδί 'states'. Οι τοπικές ενέργειες που ορίζονται σε μια μετάβαση μπορούν να χρησιμοποιηθούν μόνο στις προϋποθέσεις και τις συνέπειες της συγκεκριμένης μετάβασης. Οι ενέργειες μπορούν να έχουν προϋποθέσεις για να εκτελεστούν. Αυτό δεν συμβαίνει στις ενέργειες εισόδου αφού δε μπορούν να έχουν προϋποθέσεις μιας και είναι πάντα ενεργές. Οι προϋποθέσεις (preconditions) μιας μετάβασης είναι λογικά κατηγορήματα στις μεταβλητές του αυτομάτου και στις τοπικές μεταβλητές και υποδεικνύουν κατά πόσο επιτρέπεται να εκτελεστεί η συγκεκριμένη ενέργεια. Ορίζονται με τη λέξη κλειδί 'pre' και ακολουθούν λογικά κατηγορήματα τα οποία χωρίζονται μεταξύ τους με κόμμα. Εάν δεν υπάρχουν προϋποθέσεις, τότε η ενέργεια είναι πάντα ενεργή. Οι συνέπειες (effects) μιας μετάβασης υποδεικνύουν τις αλλαγές που προκύπτουν από την εκτέλεση της. Οι συνέπειες για να εκτελεστούν πρέπει η ενέργεια να είναι ενεργή, δηλαδή να ισχύουν όλες οι προϋποθέσεις της. Ορίζονται με τη λέξη κλειδί 'eff' και ακολουθούν οι εντολές που εκτελούνται όταν η ενέργεια γίνει ενεργή, οι οποίες χωρίζονται μεταξύ τους με το σύμβολο ';'. Για παράδειγμα ένα αυτόματο με τις πιο κάτω μεταβάσεις:

```
transitions
   input setTime(hour, minute)
    eff whatToShow := (60*hour) + minute;
        whenToShow := floor(now)
   output showTime(hour, minute)
    pre whenToShow = floor(now);
         hour = div(whatToShow, 60);
         minute = mod(whatToShow, 60)
    eff whenToShow := floor(now) + 1;
        whatToShow := mod(whatToShow + 1, 24*60)
```

Ορίζει την ενέργεια εισόδου setTime η οποία καθορίζει στη μεταβλητή wantToShow τον χρόνο που πρέπει να εμφανίζεται στο ρολόι και στη μεταβλητή whenToShow τον χρόνο που χρειάζεται μέχρι να εκτελεστεί η ενέργεια showTime. Επίσης, ορίζει την ενέργεια εξόδου showTime η οποία ενεργοποιείται όταν ο χρόνος έχει σταματήσει και οι τιμές των

παραμέτρων του αντιστοιχούν με την τιμή της μεταβλητής whatToShow. Το αποτέλεσμα της συγκεκριμένης ενέργειας είναι να επαναφέρει τον χρόνο της μεταβλητής whenToShow στον επόμενο χρόνο στον οποίο θα πρέπει να εμφανιστεί ο χρόνος (δηλαδή μετά από ένα λεπτό). Επίσης, πρέπει να θέσει στη μεταβλητή whatToShow την τιμή που θα εμφανιστεί στη συνέχεια.

Τροχιές: Οι τροχιές είναι το τμήμα το οποίο ακολουθεί τις μεταβάσεις, είναι υποχρεωτικό και ορίζει ένα σύνολο από τροχιές. Οι τροχιές είναι εκείνες που υλοποιούν την έννοια του χρόνου στα χρονισμένα αυτόματα Εισόδου / Εξόδου. Ένα αυτόματο μπορεί να έχει πολλές τροχιές οι οποίες εξελίσσονται σε διαφορετικούς ρυθμούς. Όπως οι αλλαγές που γίνονται στις καταστάσεις μεταβλητών υλοποιούνται με τις διακριτές ενέργειες, έτσι και οι τροχιές υλοποιούν τις αλλαγές των καταστάσεων με το πέρασμα του χρόνου. Ο ορισμός των τροχιών ορίζεται με τη λέξη κλειδί 'trajectories'. Κάθε ορισμός μιας τροχιάς αρχίζει με τη λέξη κλειδί 'trajdef' και αποτελείται από τα ακόλουθα στοιχεία : το όνομα της τροχιάς, τυχόν παραμέτρους, τυχόν σταθερές σχέσεις (invariants), τυχόν συνθήκες τερματισμού (stop when) και το ρυθμό εξέλιξής της (evolve). Οι μεταβλητές που εμφανίζονται στις τροχιές εκτός από στατικών τύπων δεδομένων είναι και δυναμικών για το λόγο ότι ελέγχουν την εξέλιξη της τροχιάς στο χρόνο. Οι σταθερές σχέσεις είναι προϋποθέσεις που ορίζονται και οι οποίες πρέπει να είναι αληθείς κατά τη διάρκεια εκτέλεσης του αυτομάτου. Δηλώνονται ως εξής: με τη λέξη κλειδί 'invariant', το όνομα της σταθερής σχέσης, τη λέξη κλειδί 'of', το όνομα του αυτομάτου και τις συνθήκες που θα πρέπει να ισχύουν κατά την εκτέλεση του συγκεκριμένου αυτόματου. Ένα παράδειγμα τροχιάς είναι:

```
trajectories
   trajdef timePassage
    stop when whenToShow = floor(now)
    evolve d(now) = 1
```

Ο ορισμός της τροχιάς timePassage ρυθμίζει την εξέλιξη της τιμής της μεταβλητής now και την επίδρασή της σχετικά με τη λειτουργία του αυτόματου. Η εξέλιξη της περιορίζει την τιμή της μεταβήτής now έτσι ώστε να αυξάνεται με τον ίδιο ρυθμό ενός πραγματικού χρόνου. Η εξέλιξη της τροχιάς θα σταματήσει όταν οι τιμές των μεταβλητών now και whenToShow ισούνται. Ο χρόνος δεν μπορεί να προχωρήσει μέχρι κάποια ενέργεια που θα ενεργοποιηθεί όταν η συνθήκη τερματισμού είναι αληθής, κάνει και πάλι τη συνθήκη τερματισμού ψευδή.

Σύνθεση Αυτομάτων TIOA

Για να δημιουργήσουμε ένα αυτόματο το οποίο θα μοντελοποιεί ένα πολύπλοκο σύστημα χρειάζεται να ορίσουμε μικρά αυτόματα τα οποία να μοντελοποιούν απλούστερα συστήματα και στη συνέχεια να τα συνθέσουμε, έτσι ώστε να επικοινωνούν μεταξύ τους. Η σύνθεση αυτομάτων αποτελείται από τα αυτόματα που συμμετέχουν στην σύνθεση, τυχόν κρυφές ενέργειες και ένα πρόγραμμα μεταβάσεων και τροχιών (schedule). Δεν χρειάζεται να υλοποιηθούν ενέργειες στο αυτόματο σύνθεσης, αφού όλες οι ενέργειες που χρειάζονται θα υλοποιηθούν από τα αυτόματα που συμμετέχουν στην σύνθεση. Αρχικά, όπως και σε ένα απλό αυτόματο έτσι και στο αυτόματο σύνθεσης , η περιγραφή του αρχίζει με τη λέξη κλειδί ‘automaton’, ακολουθεί το όνομα του αυτομάτου σύνθεσης και οι παράμετροι που παίρνει, εάν υπάρχουν. Τα αυτόματα που συμμετέχουν στη σύνθεση δηλώνονται με τη λέξη κλειδί ‘components’. Για κάθε αυτόματο στην σύνθεση: ορίζεται ένα αναγνωριστικό, ακολουθεί το σύμβολο ‘:’, το όνομα του αυτομάτου που αντιπροσωπεύει, οι παράμετροι που παίρνει και το σύμβολο ‘;’. Στη συνέχεια, εάν υπάρχουν κρυφές ενέργειες δηλώνονται με τη λέξη κλειδί ‘hidden’. Ακολουθεί η λέξη κλειδί ‘schedule’, για το χρονοδιάγραμμα που θα ακολουθήσει το αυτόματο σύνθεσης, το οποίο περιλαμβάνει τον ορισμό των μεταβλητών καταστάσεων του αυτομάτου σύνθεσης. Η υλοποίηση του σεναρίου προσομοίωσης μέσω του χρονοδιαγράμματος συμπεριλαμβάνεται μεταξύ των λέξεων κλειδιών ‘do’ και ‘od’, στο οποίο πυροδοτούνται οι ενέργειες και εξελίσσονται οι τροχιές των αυτομάτων που συμμετέχουν στην σύνθεση, χρησιμοποιώντας τις εντολές fire και follow αντίστοιχα. Ένα παράδειγμα αυτομάτου σύνθεσης είναι το ακόλουθο, το οποίο αντιστοιχεί στο αυτόματο που μοντελοποιεί το ρολόι αφύπνισης (AlarmClock).

automaton AlarmClock

components Clock; Alarm

Το ρολόι αφύπνισης ορίζεται ως η σύνθεση των δύο αυτομάτων Clock και Alarm που μοντελοποιούν τη λειτουργία του ρολογιού και της αφύπνισης αντίστοιχα.

2.4 Κανάλια Επικοινωνίας Μεταξύ Διεργασιών

Η επικοινωνία μεταξύ των διεργασιών ενός συστήματος υλοποιείται μέσω κάποιων καναλιών. Το εργαλείο Tempo υποστηρίζει δύο είδη καναλιών επικοινωνίας, το κανάλι MPI (Message Passing Interface) και το κανάλι TCP, τα οποία είναι ήδη υλοποιημένα και μπορεί κανείς να τα βρει στα εγχειρίδια χρήσης του εργαλείου Tempo [7]. Στο Παράρτημα Α,

παραθέτουμε τις προδιαγραφές των αυτομάτων και των λεξιλογίων που μοντελοποιούν κάθε κανάλι.

2.4.1 Κανάλια MPI

Το MPI (Message Passing Interface) [7], είναι ένα σύνολο προδιαγραφών για την ανταλλαγή μηνυμάτων και τη μεταφορά δεδομένων. Χαρακτηρίζεται για την αποτελεσματικότητα, την ευελιξία και την αποδοτικότητά του. Στο μοντέλο TIOA μοντελοποιείται ως ένα σύνθετο αυτόματο, το οποίο αποτελείται από τα αυτόματα: Send Mediator (SendMediator.tioa) και Receive Mediator (ReceiveMediator.tioa). Τα αυτόματα αυτά, χρησιμοποιούν συγκεκριμένο λεξιλόγιο (Vocabulary.tioa), στο οποίο μπορούμε να ορίσουμε νέους τύπους δεδομένων που πιθανώς να χρειαστούμε στις προδιαγραφές. Αρχικά, χρησιμοποιώντας το κανάλι επικοινωνίας MPI, θα πρέπει να ορίσουμε τον αριθμό των διεργασιών , οι οποίες θα εκτελούν το ίδιο πρόγραμμα. Κάθε διεργασία, διακρίνεται με βάση έναν ακέραιο αριθμό (rank) που της αποδίδεται κατά την εκκίνηση του προγράμματος. Το rank κάθε διεργασίας είναι μοναδικό και παίρνει τιμές στο εύρος 0 έως n-1, όπου n είναι το συνολικό πλήθος των διεργασιών και χρησιμοποιείται από τον χρήστη στην επικοινωνία μεταξύ των διεργασιών. Να σημειώσουμε ότι τόσο στα κανάλια MPI, όσο και στα κανάλια TCP που θα περιγράψουμε στη συνέχεια, η αποστολή και η λήψη μηνυμάτων γίνεται ανάμεσα από μόνο δύο διεργασίες, όπου μία διεργασία εκτελεί την αποστολή και μία την λήψη. Στις προδιαγραφές μας, χρησιμοποιούμε δύο βασικές συναρτήσεις του MPI: την MPI_Size(), η οποία επιστρέφει το σύνολο των διεργασιών που δημιουργούνται και την MPI_Rank(), η οποία επιστρέφει για κάθε διεργασία την ταυτότητά της, δηλαδή τον αριθμό με τον οποίο αντιστοιχίζεται όταν δημιουργείται.

2.4.2 Κανάλια TCP

Το κανάλι TCP, μοντελοποιείται στο μοντέλο TIOA ως ένα σύνθετο αυτόματο, που αποτελείται από τα εξής τρία αυτόματα: Send Mediator (TCPSendMed.tioa), Receive Mediator (TCPRecvMed.tioa) και Channel Mediator (TCPChanMed.tioa). Κάθε αυτόματο που χρησιμοποιεί TCP κανάλια επικοινωνίας, απαιτεί συγκεκριμένο λεξιλόγιο, το οποίο ορίζεται στο αρχείο TCPVocabs.tioa.

Το αυτόματο Send Mediator, είναι το πιο απλό από τα τρία αυτόματα που αποτελούν το κανάλι TCP και υλοποιεί την αποστολή των μηνυμάτων. Τα μηνύματα που θα σταλούν από μια διεργασία, αποθηκεύονται προσωρινά στο σύνολο sendBuffer μέχρι να περάσουν στο

αυτόματο του καναλιού. Για να σταλεί ένα μήνυμα, θα πρέπει να δημιουργηθεί μια σύνδεση μεταξύ του αποστολέα και του παραλήπτη, που γίνεται με τη βοήθεια των ενεργειών TCP_senderOpen και το TCP_respSenderOpen. Εάν η σύνδεση δημιουργηθεί επιτυχώς, τότε τα μηνύματα που περιμένουν στο sendBuffer μεταφέρονται στο channel mediator μέσω της ενέργειας TCP_write. Μια σύνδεση που δημιουργείται, μπορεί να κλείσει ανά πάσα στιγμή, μέσω των ενεργειών TCP_senderClose και TCP_respSenderClose.

Το αυτόματο Receive Mediator, υλοποιεί την παράδοση μηνυμάτων σε μια διεργασία. Η ενέργεια RECEIVE, είναι υπεύθυνη για την παραλαβή μηνυμάτων και ενεργοποιείται όταν υπάρχουν μηνύματα που πρέπει να ληφθούν και όταν δημιουργείται σύνδεση μεταξύ της διεργασίας παραλήπτης και της διεργασίας αποστολέας των μηνυμάτων. Μια διεργασία για να παραλάβει ένα μήνυμα πρέπει να ανταποκριθεί με την εγκαθίδρυση ενός JVMServerSocket. Συγκεκριμένα, πρέπει να δημιουργηθεί και να δεσμευτεί μια υποδοχή (socket) για την επικοινωνία μεταξύ δυο διεργασιών, χρησιμοποιώντας τις ενέργειες TCP_bind και TCP_respBind, καθώς επίσης και να γίνει αποδεκτή μέσω των ενεργειών TCP_accept και TCP_respAccept. Ανά πάσα στιγμή, μια διεργασία μπορεί να σταματήσει να ακούει σε κάποιαν υποδοχή και κατά συνέπεια να σταματήσει οποιαδήποτε σύνδεση με κάποια άλλη διεργασία, εκτελώντας τις ενέργειες TCP_stopAccepting και TCP_stopListening. Επίσης οι συνδέσεις που έχουν δημιουργηθεί μεταξύ των διεργασιών κλείνουν με τις ενέργειες TCP_rClose και TCP_rCloseStream. Στην περίπτωση που η διεργασία παραλήπτης εισέλθει σε κατάσταση αποδοχής οποιασδήποτε σύνδεσης, είναι σε θέση να παραλάβει μηνύματα, τα οποία διαβάζονται από το κανάλι μέσω των ενεργειών TCP_read και TCP_respRead και προστίθενται στο σύνολο recvBuffer.

Το αυτόματο Channel Mediator, υλοποιεί το κανάλι μεταξύ των αυτομάτων Send Mediator και Receive Mediator και αλληλεπιδρά με το πρωτόκολλο TCP. Σύμφωνα με τους κανόνες του εργαλείου Tempo, οι ενέργειες των αυτομάτων δεν μπορούν να αποκλειστούν. Επομένως, χρησιμοποιούνται χρονικά όρια (timeouts) για να αποφεύγονται οποιοιδήποτε αποκλεισμοί των αιτημάτων στις υποδοχές (sockets).

2.5 Παράδειγμα Υλοποίησης Αλγορίθμου στο Εργαλείο Tempo

2.5.1 Περιγραφή Αλγορίθμου

Ο αλγόριθμος που θα υλοποιήσουμε είναι ο εξής : “ Έστω ότι υπάρχουν n διεργασίες και κάθε διεργασία έχει τη δική της ταυτότητα. Μία από τις διεργασίες ορίζεται ως αρχηγός και

δίνεται ως παράμετρος στο πρόγραμμα. Κάθε διεργασία (εκτός από την αρχηγό) πρέπει να στείλει την ταυτότητά της στην διεργασία αρχηγό, σε χρονικό διάστημα t_1. Η διεργασία αρχηγός, αφού συλλέξει τις ταυτότητες όλων των διεργασιών και εντοπίσει την διεργασία με τη μεγαλύτερη ταυτότητα, τότε πρέπει να ανακοινώσει στις υπόλοιπες διεργασίες ποια είναι η διεργασία με την μεγαλύτερη ταυτότητα, σε χρονικό διάστημα t_2. Εάν μια διεργασία δεν στείλει την ταυτότητά της στο χρονικό διάστημα που απαιτείται, τότε ξαναστέλνει στην διεργασία αρχηγό την ταυτότητά της. Εάν η διεργασία αρχηγός δεν ανακοινώσει τη διεργασία με τη μεγαλύτερη ταυτότητα στις υπόλοιπες διεργασίας στο χρονικό διάστημα που απαιτείται, τότε την ανακοινώνει ξανά σε όλες τις άλλες διεργασίες. ”

2.5.2 Προδιαγραφή Αλγορίθμου

Για να υλοποιήσουμε τον αλγόριθμο που περιγράφεται πιο πάνω, χρειάζεται να υλοποιήσουμε: το λεξιλόγιο που θα χρησιμοποιείται από τα αυτόματα του αλγορίθμου, το αυτόματο το οποίο θα υλοποιεί την κάθε διεργασία, τα αυτόματα που υλοποιούν το κανάλι επικοινωνίας μεταξύ των διεργασιών και το αυτόματο σύνθεσης. Η επικοινωνία μεταξύ των διεργασιών για το συγκεκριμένο αλγόριθμο θα υλοποιηθεί με το κανάλι επικοινωνίας MPI (Message Passing Interface), οι προδιαγραφές του οποίου έχουν ήδη οριστεί και τις παραθέτουμε στο Παράρτημα Α1.

Το λεξιλόγιο για τον συγκεκριμένο αλγόριθμο φαίνεται στο Σχήμα 2.6 και περιλαμβάνει το λεξιλόγιο που χρησιμοποιούν τα αυτόματα διεργασίες και το λεξιλόγιο που χρειάζεται το κανάλι επικοινωνίας MPI. Ορίζουμε μία πλειάδα Message, η οποία αντιπροσωπεύει ένα τμήμα του μηνύματος που στέλνει η κάθε διεργασία και αποτελείται από δύο φυσικούς αριθμούς οι οποίοι αντιστοιχούν στον τύπο του μηνύματος και στην ταυτότητα της μεγαλύτερης διεργασίας. Ο τύπος του μηνύματος είναι 1 εάν είναι μήνυμα από μια διεργασία προς τη διεργασία αρχηγό και 2 εάν είναι μήνυμα από τη διεργασία αρχηγό προς μια απλή διεργασία. Το τμήμα αυτό μαζί με δύο φυσικούς αριθμούς που δηλώνουν τον αποστολέα και τον παραλήπτη του μηνύματος, ορίζουν τον τύπο του μηνύματος (mpi_message) με το οποίο επικοινωνούν οι διεργασίες μεταξύ τους.

```
%%% :Algorithm vocabs
vocabulary algorithm_voc
	types
		Message : Tuple[type : Nat, maxId : Nat]
end

%%% :MPI Channel vocabs
vocabulary mpi_status_voc
	types mpi_status
	operators
		MPI_Iprobe : Nat -> Null[mpi_status],
		MPI_Test : mpi_status ->Bool
end

vocabulary mpi_message_voc
	imports algorithm_voc, mpi_status_voc
		types mpi_message : Tuple[data:Message,sender : Nat,
destination:Nat]
		operators
			MPI_Irecv: mpi_status, Nat -> mpi_message
end

vocabulary mpi_request_voc
	imports mpi_message_voc
		types mpi_request
		operators
			MPI_Isend: mpi_message, Nat -> Null[mpi_request],
			MPI_Barrier : -> Bool
End

vocabulary mpi_voc
	operators
		MPI_Rank : -> Nat,
		MPI_Size : -> Nat
end
```

Σχήμα 2.6 Λεξιλόγιο (Vocabulary.tioa)

Το Σχήμα 2.7 απεικονίζει την προδιαγραφή του αυτομάτου που αντιπροσωπεύει μία διεργασία στο εργαλείο Tempo. Το αυτόματο αυτό, ονομάζεται Process και παίρνει ως παραμέτρους: το χρονικό διάστημα t1 στο οποίο κάθε διεργασία (εκτός από τη διεργασία αρχηγό) θα πρέπει να στείλει την ταυτότητά της στη διεργασία αρχηγό, το χρονικό διάστημα t2 στο οποίο η διεργασία αρχηγός αφού εντοπίσει τη διεργασία με την μεγαλύτερη ταυτότητα πρέπει να την ανακοινώσει σε όλες τις άλλες διεργασίες, τον αριθμό που αντιστοιχεί στην ταυτότητα της διεργασίας και τον αριθμό που αντιστοιχεί στην ταυτότητα της διεργασίας που ορίζεται ως αρχηγός. Στη συνέχεια δηλώνονται οι ενέργειες μιας διεργασίας για την παραλαβή, αποστολή και προετοιμασία μηνυμάτων ,καθώς επίσης και οι ενέργειες που θα ενεργοποιηθούν από την μη παράδοση των μηνυμάτων στα χρονικά διαστήματα που απαιτούνται. Οι μεταβλητές καταστάσεων του αυτομάτου ορίζονται στο τμήμα states οι οποίες είναι οι ακόλουθες:

- clock : Αντιπροσωπεύει το φυσικό ρολόι της διεργασίας και είναι τύπου AugmentedReal και αρχικοποιείται με 0.
- processClock : Αντιπροσωπεύει το φυσικό ρολόι κάθε διεργασίας (εκτός από τη διεργασία αρχηγό), το οποίο υπολογίζει το χρόνο μέχρι να στείλει την ταυτότητά της στην διεργασία αρχηγό. Είναι τύπου AugmentedReal και αρχικοποιείται με 0.
- coordinatorClock : Αντιπροσωπεύει το φυσικό ρολόι της διεργασίας αρχηγού, το οποίο υπολογίζει το χρόνο μέχρι να ανακοινώσει σε όλες τις άλλες διεργασίες την ταυτότητα της μεγαλύτερης διεργασίας. Είναι τύπου AugmentedReal και αρχικοποιείται με 0.
- msgs : Είναι μία κενή ακολουθία μηνυμάτων mpi_message της κάθε διεργασίας, στην οποία εισέρχονται τα μηνύματα που πρόκειται να στείλει και εξέρχονται τα μηνύματα που στέλνονται επιτυχώς.
- messageCounter : Είναι ένας μετρητής ο οποίος υπολογίζει πόσα μηνύματα λαμβάνει η διεργασία αρχηγός. Είναι ακέραιος αριθμός και αρχικοποιείται με 0.
- maxId : Είναι η φυσική μεταβλητή που αντιπροσωπεύει την ταυτότητα της μεγαλύτερης διεργασίας. Αρχικοποιείται με μηδέν.
- coordinatorAnnounceMaxId : Είναι η boolean μεταβλητή που καθορίζει εάν η διεργασία αρχηγός έχει εντοπίσει την διεργασία με την μεγαλύτερη ταυτότητα. Αρχικοποιείται με false και γίνεται true όταν θα ανακοινώσει η διεργασία αρχηγός την ταυτότητα της μεγαλύτερης διεργασίας.
- processSentId : Είναι η Boolean μεταβλητή που καθορίζει εάν η διεργασία (εκτός από την διεργασία αρχηγό) μπορεί να στείλει την ταυτότητά της στη διεργασία αρχηγός. Αρχικά είναι true γιατί ο αλγόριθμος ξεκινάει στέλλοντας κάθε διεργασία την ταυτότητά της στην διεργασία αρχηγό και ακολούθως γίνεται false.

Οι ενέργειες της διεργασίας που έχουν δηλωθεί στο τμήμα signature, υλοποιούνται στο τμήμα transitions και φαίνονται στο Σχήμα 2.6. Αναλυτικά:

- Με την ενέργεια εισόδου RECEIVE, η διεργασία λαμβάνει ένα μήνυμα m, τύπου mpi_message. Εάν το μήνυμα δεν είναι κενό και ο τύπος του είναι 1, τότε η διεργασία είναι η διεργασία αρχηγός. Αυξάνει τον μετρητή μηνυμάτων που λαμβάνει, βρίσκει την μέγιστη ταυτότητα μεταξύ της maxId και της ταυτότητας που έχει στείλει το συγκεκριμένο μήνυμα και εάν έχει πάρει μηνύματα απ' όλες τις διεργασίας τότε η τιμή της μεταβλητής coordinatorAnnounceMaxId γίνεται true, δηλαδή μπορεί να ανακοινώσει σε όλους ποια είναι η διεργασία με την μεγαλύτερη ταυτότητα. Εάν

όμως το μήνυμα δεν είναι κενό και ο τύπος του μηνύματος είναι 2, τότε η διεργασία δεν είναι η διεργασία αρχηγός αλλά είναι μια απλή διεργασία στην οποία η διεργασία αρχηγός ανακοινώνει ποια είναι η διεργασία με την μεγαλύτερη ταυτότητα. Έτσι αποθηκεύει στην μεταβλητή maxId την τιμή maxId του μηνύματος και κάνει false τη μεταβλητή processSentId αφού δεν υπάρχει λόγος πλέον να στείλει την ταυτότητά της στην αρχηγό. Η εκτέλεση της ενέργειας αυτής, πυροδοτεί όλες τις αντίστοιχες ενέργειες εξόδου RECEIVE.

- Με την ενέργεια εξόδου SEND, η διεργασία στέλλει ένα μήνυμα m. Προυποθέσεις για να εκτελεστεί η ενέργεια αυτή είναι το μήνυμα που θα σταλεί να βρίσκεται πρώτο στην ακολουθία μηνυμάτων msgs και η ακολουθία μηνυμάτων να μη είναι άδεια. Εάν το μήνυμα που θα σταλεί είναι προς τη διεργασία αρχηγό τότε αρχικοποιείται το ρολόι της διεργασίας ενώ αν είναι μήνυμα από την αρχηγό σε κάποια απλή διεργασία αρχικοποιείται το ρολόι της διεργασίας αρχηγού. Η εκτέλεση της ενέργειας αυτής, πυροδοτεί όλες τις αντίστοιχες ενέργειες εισόδου SEND.
- Με την εσωτερική ενέργεια prepMessages, κάθε διεργασία που δεν είναι η διεργασία αρχηγός, για να εκτελέσει την ενέργεια πρέπει η μεταβλητή processSentId να είναι αληθής. Εάν ισχύει η προϋπόθεση, δημιουργεί ένα μήνυμα (με τύπο 1) με την ταυτότητα της προς τη διεργασία αρχηγό και το οποίο εισάγεται στην ακολουθία msgs. Αντίθετα, η διεργασία αρχηγός για να εκτελέσει την ενέργεια, πρέπει η μεταβλητή coordinatorAnnounceMaxId να είναι αληθής. Εάν ισχύει, τότε δημιουργεί ένα μήνυμα (με τύπο 2) με την ταυτότητα της μεγαλύτερης διεργασίας προς όλες τις άλλες διεργασίες και τα οποία εισάγονται στην ακολουθία msgs. Τα μηνύματα της ακολουθίας msgs θα σταλούν όταν εκτελεστεί η ενέργεια εξόδου SEND.
- Με την εσωτερική ενέργεια sendIdTimeout, εάν μια διεργασία που δεν είναι αρχηγός, δεν στείλει την ταυτότητά της στην αρχηγό σε χρονικό διάστημα t1, τότε μηδενίζεται το φυσικό ρολόι της διεργασίας (processClock) και γίνεται true η μεταβλητή prosessSentId για να εκτελεστεί ξανά η ενέργεια prepMessages.
- Με την εσωτερική ενέργεια announceTimeout, εάν η διεργασία αρχηγός, αφού εντοπίσει τη διεργασία με τη μεγαλύτερη ταυτότητα δεν την ανακοινώσει στις υπόλοιπες διεργασίες σε χρονικό διάστημα t2, τότε μηδενίζεται το φυσικό ρολόι της διεργασίας αρχηγού (coordinatorTimeout) και γίνεται true η μεταβλητή coordinatorAnnounceMaxId για να εκτελεστεί ξανά η ενέργεια prepMessages.

Αφού έχουν οριστεί οι ενέργειες του αυτομάτου, μετά το τμήμα transitions ακολουθεί ο ορισμός της τροχιάς Time στο τμήμα trajectories, η οποία εξελίσσει το φυσικό ρολόι clock με ρυθμό 1.

```
automaton Process (t1 : DiscreteReal, t2 : DiscreteReal, id : Nat, coordinatorId
: Nat)

signature
	input RECEIVE(m : Null[mpi_message])
	output SEND(m : Null[mpi_message])
	internal prepMessages, sendIdTimeout, announceTimeout

states
	% The physcal clock of the automaton
	clock : AugmentedReal := 0;
	% The physical clock for the process
	processClock : AugmentedReal := 0;
	% The physical clock for the process which is coordinator
	coordinatorClock : AugmentedReal := 0;
	% An empty sequence of mpi messages
	msgs : Seq[Null[mpi_message]] := {};
	% A counter for the messages that the process coordinator has received
	messageCounter : Int := 0;
	% The max id from all the processes
	maxId : Nat := 0;
	% Indicates that coordinator can announce the prcocess with max id
	coordinatorAnnounceMaxId : Bool := false;
	% Indicates that process can sent its id to the coordinator
	processSentId : Bool := true;

transitions
	input RECEIVE(m)
	eff
		if(m ~= nil())then
			if(val(m).data.type = 1)then
				messageCounter := messageCounter + 1;
				maxId := max(maxId,val(m).sender);
				if(messageCounter = MPI_Size() -1)then
					coordinatorAnnounceMaxId := true;
				fi
			else
				maxId := val(m).data.maxId;
				processSentId := false;
			fi
		fi

	output SEND(m)
	pre
		msgs ~={};
		m = head(msgs);
	eff
		msgs := tail(msgs);
		if(val(m).data.type = 1)then
			processClock := clock;
		else
			coordinatorClock := clock;
		fi

	internal prepMessages
	pre
		if(id ~= coordinatorId)then
			processSentId = true
		else
			coordinatorAnnounceMaxId = true;
```

```
	eff
		if(id ~= coordinatorId)then
			msgs := msgs |- embed([[1,maxId], id, coordinatorId]);
			processSentId := false;
		else
			for n : Nat where(n < MPI_Size()) do
				if(n ~= id)then
					msgs := msgs |- embed([[2, maxId], id, n]);
				fi
			od
			coordinatorAnnounceMaxId := false;
		fi


	internal sendIdTimeout
	pre
		clock > processClock + t1;
		id ~= coordinatorId;

	eff
		processClock := 0;
		processSentId := true;


	internal announceTimeout
	pre
		clock > coordinatorClock + t2;
		id = coordinatorId;
	eff
		coordinatorClock := 0;
		coordinatorAnnounceMaxId := true;

trajectories
	trajdef Time
		evolve d(clock) = 1;
```

Σχήμα 2.7 Αυτόματο διεργασίας (Process.tioa)

Εφόσον έχουμε υλοποιήσει το αυτόματο διεργασίας και τα αυτόματα που υλοποιούν τα κανάλια επικοινωνίας (SendMediator.tioa και ReveiveMediator.tioa) είναι ήδη υλοποιημένα, μπορούμε να υλοποιήσουμε το αυτόματο σύνθεσης Composition. Ξεκινώντας θα πρέπει να συμπεριλάβουμε τα αρχεία στα οποία υλοποιούνται : το λεξιλόγιο του αλγορίθμου, το αυτόματο διεργασίας και τα αυτόματα του καναλιού επικοινωνίας MPI, το οποίο επιτυγχάνεται με την λέξη κλειδί ‘include’. Για το λεξιλόγιο πρέπει να εισάγουμε ρητά, με τη λέξη κλειδί ‘import’, όλα τα λεξιλόγια που ορίζονται στο αρχείο Vocabulary.tioa και τα οποία θα χρησιμοποιήσουμε. Να σημειωθεί ότι συμπεριλάβαμε το λεξιλόγιο μόνο στο αρχείο που υλοποιεί το αυτόματο σύνθεσης, για το λόγο ότι εάν το συμπεριλαμβάναμε και στο αυτόματο διεργασίας, θα είχαμε συντακτικό λάθος για διπλό ορισμό των τύπων που ορίζονται στο λεξιλόγιο. Το αυτόματο αυτό παίρνει 3 παραμέτρους και αυτές αντιστοιχούν στο χρονικό

διάστημα t1 στο οποίο κάθε διεργασία (εκτός από τη διεργασία αρχηγό) θα πρέπει να στείλει την ταυτότητά της στη διεργασία αρχηγό, στο χρονικό διάστημα t2 στο οποίο η διεργασία αρχηγός αφού εντοπίσει τη διεργασία με την μεγαλύτερη ταυτότητα πρέπει να την ανακοινώσει σε όλες τις άλλες διεργασίες και στο αριθμό που αντιστοιχεί στην ταυτότητα της διεργασίας που ορίζεται ως αρχηγός. Το αυτόματο σύνθεσης αποτελείται από: ένα αυτόματο διεργασίας (P), ένα αυτόματο SendMediator (SM) και ένα ReceiveMediator (RM). Στο αυτόματο διεργασίας P, περνιούνται ως παράμετροι οι τρεις πρώτοι παράμετροι του αυτομάτου σύνθεσης. Επίσης δίνεται από τη συνάρτηση MPI_Rank() η ταυτότητα της διεργασίας.

Ακολούθως, δηλώνονται οι μεταβλητές του αυτομάτου σύνθεσης, στο τμήμα states, οι οποίες είναι :

- m : Δηλώνει ένα μήνυμα mpi_message, το οποίο αρχικοποιείται με nil().
- runs : Αντιπροσωπεύει τον αριθμό των επαναλήψεων του αλγόριθμου που θα εκτελεστούν στο πρόγραμμα μεταβάσεων και τροχιών. Αρχικοποιείται με $(MPI\_Size)^2$ που αντιστοιχούν με τον μέγιστο αριθμό μηνυμάτων που μπορούν να σταλούν.

Το πρόγραμμα μεταβάσεων και τροχιών (schedule) αρχίζει με την πυροδότηση της εσωτερικής ενέργειας prepMessages για την προετοιμασία των μηνυμάτων και ακολουθεί ένας βρόγχος στον οποίον: εξελίσσεται η τροχιά του αυτομάτου διεργασίας με ρυθμό 1, πυροδοτούνται οι εσωτερικές ενέργειες sentIdTimeout και announceTimeout, οι οποίες ελέγχουν για τυχόν καθυστερήσεις στις αποστολές των μηνυμάτων, στέλνονται τα μηνύματα που βρίσκονται στο κανάλι και έχουν αποστολέα τη συγκεκριμένη διεργασία της οποίας εκτελείται το πρόγραμμα και ακολούθως λαμβάνονται τυχόν μηνύματα που βρίσκονται στο κανάλι και έχουν ως παραλήπτη τη συγκεκριμένη διεργασία. Η εξέλιξη της τροχιάς και οι πυροδοτήσεις των ενεργειών γίνονται με τις εντολές follow και fire, αντίστοιχα. Το αυτόματο σύνθεσης φαίνεται στο σχήμα που ακολουθεί (Σχήμα 2.8).

```
%%% .: Vocabulary :.
include "Vocabulary.tioa"
imports algorithm_voc, mpi_status_voc, mpi_message_voc, mpi_request_voc,
mpi_voc

%%% .: MPI Mediator Automata :.
include "SendMediator.tioa"
include "ReceiveMediator.tioa"

%%% .: Algorithm Automata :.
include "Process.tioa"

automaton Composition (t1 : DiscreteReal, t2 : DiscreteReal, coordinatorId :
Nat)
      components
```

```
        P : Process(t1, t2, MPI_Rank(), coordinatorId);
        SM : SendMediator;
        RM : ReceiveMediator;

    schedule
        states
            m : Null[mpi_message] := nil();
            runs : Int := MPI_Size() ** 2;
        do
            fire internal P.prepMessages;

            for i: Nat where(i < runs)do
                follow P.Time duration 1;

                fire internal P.sendIdTimeout;
                fire internal P.announceTimeout;

                %% Send Messages
                for n : Nat where(n < MPI_Size())do
                    fire output P.SEND(m);
                od
                follow SM.DELAY duration (MPI_Size() * 10);

                %% Receive Messages
                for n : Nat where(n < MPI_Size())do
                    m := nil();
                    fire input RM.probe(n);
                    fire output RM.RECEIVE(m);
                od
            od
        od
```

Σχήμα 2.8 Αυτόματο σύνθεσης (Composition.tioa)

2.5.3 Μετάφραση Προδιαγραφής Αλγορίθμου σε κώδικα Java

Με την ολοκλήρωση της προδιαγραφής του αλγορίθμου και το συντακτικό της έλεγχο, είμαστε σε θέση μεταφράσουμε τον κώδικα σε γλώσσα προγραμματισμού Java. Αυτό θα γίνει με τον μεταφραστή Java αφού πρώτα καθορίσουμε στο tempo2java applet με ποιο πρωτόκολλο υλοποιείται το κανάλι επικοινωνίας του αλγορίθμου. Στο συγκεκριμένο παράδειγμα, χρησιμοποιούμε το πρωτόκολλο MPI και θα το καθορίσουμε σύμφωνα με τις ακόλουθες οδηγίες:

1. Επιλέγουμε από το menu bar Windows και ακολούθως Preferences.
2. Στο νέο παράθυρο που εμφανίζεται, πηγαίνουμε στο Tempo Plugins και επιλέγουμε το Java Generator.
3. Στη συνέχεια επιλέγουμε το MPI και πατάμε OK.

4. Από την εργαλειοθήκη, επιλέγουμε το εικονίδιο που αντιστοιχεί στον μεταφραστή Java και στην κονσόλα εμφανίζεται ο πηγαίος κώδικας σε Java, ο οποίος παράλληλα αποθηκεύεται στο χώρο εργασίας στο φάκελο 'Composition.java'.

2.6. Κατανεμημένο Σύστημα PlanetLab

Ένας από τους αλγόριθμους εκλογής αρχηγού που θα μελετήσουμε και θα προσομοιώσουμε, θα αξιολογηθεί στο κατανεμημένο σύστημα PlanetLab. Για το λόγο αυτό, θα γίνει μια σύντομη αναφορά στο PlanetLab και στη λειτουργία του [8].

2.6.1. Εισαγωγή στο PlanetLab

Το PlanetLab [13] είναι ένα παγκόσμιο ερευνητικό δίκτυο για την ανάπτυξη και τη χρήση υπηρεσιών παγκόσμιας κλίμακας. Από τις αρχές του 2003, περισσότεροι από 1000 ερευνητές έχουν χρησιμοποιήσει το PlanetLab για ανάπτυξη νέων τεχνολογιών για κατανεμημένη αποθήκευση, χαρτογράφηση δικτύων, peer-to-peer συστήματα, και επεξεργασία ερωτημάτων (query processing).

Το PlanetLab, ως ένα κατανεμημένο σύστημα, περιλαμβάνει κόμβους από όλο τον κόσμο. Τη δεδομένη χρονική στιγμή, το PlanetLab αποτελείται από 1172 κόμβους (nodes), οι οποίοι βρίσκονται σε 552 διαφορετικές τοποθεσίες (sites) [13]. Κάθε τοποθεσία διαθέτει κάποιον αριθμό υπολογιστικών μηχανών και της δίδεται η δυνατότητα να αξιοποιήσει πόρους από όλο το σύστημα, για την εκτέλεση ερευνητικών πειραμάτων μεγάλης κλίμακας. Η αξιοποίηση ενός τόσου μεγάλου σε κλίμακα συστήματος, για την ανάπτυξη εφαρμογής σε κατανεμημένα συστήματα, είναι δύσκολη και αρκετά ενδιαφέρουσα για τα λόγο ότι προσφέρει ένα ρεαλιστικό δίκτυο με την παρουσία συμφόρησης, καταρρεύσεων και ποικίλων συμπεριφορών από τους κόμβους. Το PlanetLab προσφέρει μεγάλες δυνατότητες για την αξιολόγηση της απόδοσης κατανεμημένων εφαρμογών σε πραγματικές συνθήκες.[14]

2.6.2 Λειτουργία του PlanetLab

Κάθε οργανισμός που συμμετέχει στο PlanetLab, διαθέτει δύο ή περισσότερους κόμβους, τους οποίους μπορούν να χρησιμοποιούν τα άλλα μέλη του PlanetLab. Το Πανεπιστήμιο Κύπρου διαθέτει το δικό του site στο PlanetLab Europe, το ευρωπαϊκό τμήμα της διαθέσιμης

πλατφόρμας δοκιμών του PlanetLab και προσφέρει τρεις κόμβους. Οι χρήστες του PlanetLab, διακρίνονται στις εξής τρεις κατηγορίες:

- Principal Investigator (PI) : Ο PI κάθε οργανισμού, είναι υπεύθυνος για τη διαχείριση των μερισμάτων (slices) και των χρηστών του οργανισμού. Είναι το μοναδικό άτομο που μπορεί να ενεργοποιήσει, απενεργοποιήσει και διαγράψει λογαριασμούς χρηστών, να δημιουργήσει και διαγράψει μερίσματα, να αναθέσει μερίσματα στους χρήστες και να κατανέμει πόρους στα μερίσματα του οργανισμού του.
- Technical Contact (Tech Contact) : Κάθε οργανισμός απαιτείται να έχει ένα Technical Contract, ο οποίος είναι υπεύθυνος για την εγκατάσταση, συντήρηση και παρακολούθηση των κόμβων του οργανισμού του.
- User : Απλός χρήστης είναι όποιος αναπτύσσει εφαρμογές στο PlanetLab . Για να είναι κάποιος χρήστης, πρέπει να έχει ένα έγκυρο λογαριασμό και να του έχει ανατεθεί ένα μέρισμα από τον PI του οργανισμού στο οποίο συμμετέχει.

Κάθε οργανισμός μπορεί να δημιουργήσει το πολύ 10 μερίσματα και κάθε μέρισμα έχει διάρκεια ζωής 1 μήνα. Ο χρήστης μπορεί να ανανεώσει το μέρισμά του και σε περίπτωση που αυτό δε γίνει, το μέρισμα λήγει αποδεσμεύοντας τους πόρους του. Κάθε χρήστης μπορεί να χρησιμοποιεί τους κόμβους του μερίσματός του χωρίς να επηρεάζεται από τους άλλους χρήστες που πιθανόν να χρησιμοποιούν τον ίδιο κόμβο. Όμως, η απόδοση του κάθε κόμβου και γενικά του δικτύου επηρεάζεται απ' όλους τους χρήστες. Να σημειώσουμε πως ο κάθε κόμβος μπορεί να καταρρεύσει και να επανεκκινήσει ανά πάσα στιγμή, χωρίς όμως να χάνονται τα δεδομένα, και για το λόγο αυτό γίνονται μη-αξιόπιστοι.

Αρχικά, για να χρησιμοποιήσουμε την πλατφόρμα του PlanetLab, πρέπει να δημιουργήσουμε ένα λογαριασμό. Αυτό γίνεται μέσω της επίσημης σελίδας του PlanetLab, χρησιμοποιώντας την ηλεκτρονική μας διεύθυνση, η οποία θα αποτελεί και το συνθηματικό μας για το PlanetLab. Εισάγοντας τις απαραίτητες πληροφορίες και ως site το όνομα του οργανισμού στο οποίο ανήκουμε, ο PI του οργανισμού μας θα πάρει το αίτημα. Με τη σειρά του, επιβεβαιώνοντας το αίτημα, θα ενεργοποιήσει το λογαριασμό μας και θα μας αναθέσει σε κάποιο μέρισμα.

Έχοντας δημιουργήσει λογαριασμό στο PlanetLab και πριν να αποκτήσουμε πρόσβαση τους κόμβους του PlanetLab, χρειάζεται να δημιουργήσουμε ένα ζεύγος κρυπτογραφημένων κλειδιών SSH για λόγους ασφαλείας. Για να το κάνουμε αυτό, χρησιμοποιούμε το ssh-keygen πρόγραμμα σε οποιοδήποτε ασφαλές σύστημα UNIX και εκτελούμε την εξής εντολή:

ssh-keygen -t rsa -f ~/.ssh/id_rsa

Με την επιλογή -t ορίζουμε τον τύπο του κλειδιού, με την επιλογή -f ορίζουμε που θα αποθηκευτεί το κλειδί με όνομα id_rsa που θα δημιουργήσουμε. Θα μας ζητηθεί να

εισάγουμε έναν κωδικό, τον οποίο θα χρησιμοποιούμε για να ενωθούμε στους κόμβους. Από την εκτέλεση της πιο πάνω εντολής, θα παραχθεί ένα private key id_rsa και ένα public key id_rsa.pub, το οποίο θα ανεβάσουμε στον PlanetLab λογαριασμό μας, χρησιμοποιώντας την ιστοσελίδα του PlanetLab και συγκεκριμένα την σελίδα 'Manage My Keys'.

Γνωστοποιώντας το public key (id_rsa.pub) που δημιουργήσαμε, μπορούμε να εισάγουμε κόμβους (μηχανές) στο μέρισμα μας για να μπορέσουμε να έχουμε πρόσβαση σ' αυτούς. Ο πιο απλός τρόπος για να προσθέσεις κόμβους είναι να τους επιλέγεις ένα ένα από τον λογαριασμό σου στο PlanetLab, όπου έχεις τη δυνατότητα να δεις όλους τους κόμβους που βρίσκονται στο PlanetLab. Για να ενωθούμε σε κάποια μηχανή χρησιμοποιούμε την ακόλουθη εντολή:

ssh -l slice_name -i ~/.ssh/id_rsa hostname

Με την επιλογή -l ορίζουμε το όνομα του χρήστη, όπου στην προκειμένη περίπτωση είναι το όνομα του μερίσματος (slice_name) στο οποίο συμμετέχουμε και με την επιλογή -i προσδιορίζουμε που βρίσκεται το κρυπτογραφημένο κλειδί του χρήστη. Το hostname είναι η διεύθυνση του κόμβου στον οποίο επιθυμούμε να ενωθούμε. Να σημειώσουμε ότι κάθε φορά που θα προσπαθούμε να ενωθούμε σε κάποιο κόμβο, θα μας ζητείται να εισάγουμε τον κρυπτογραφημένο κλειδί μας, για να πιστοποιηθεί η ταυτότητά μας.

Όταν ενωθούμε σε κάποια μηχανή, θα διαπιστώσουμε ότι η μηχανή δεν έχει τίποτα εγκατεστημένο, εκτός από το Fedora Core 8, ένα λειτουργικό σύστημα για Linux. Θα πρέπει να εγκαταστήσουμε τα προγράμματα που θα χρειαστούμε για την εκτέλεση των προγραμμάτων μας. Στην περίπτωσή μας χρειάστηκε α εγκαταστήσουμε τον μεταγλωττιστή της Java. Επίσης, πρέπει να έχουμε υπόψη μας ότι από τη στιγμή που πιθανόν η μηχανή που χρησιμοποιούμε να χρησιμοποιείται και από άλλους χρήστες, ο χώρος μας στη μηχανή είναι διαφορετικός από τους υπόλοιπους χρήστες.

Για να μεταφέρουμε αρχεία από τον υπολογιστή μας σε κάποια μηχανή, χρησιμοποιούμε την πιο κάτω εντολή σε ένα terminal παράθυρο :

rsync -avz -i ~/.ssh/id_rsa fiename1 slice_name@hostname:filename2

Η επιλογή -avz καθορίζει πως μαζί με τα αρχεία, θα μεταφερθούν και τα δικαιώματά τους. Ακολουθεί η επιλογή -i με την διεύθυνση στην οποία βρίσκεται το private key στον υπολογιστή μας. Το filename1 προσδιορίζει την διεύθυνση του αρχείου που θέλουμε να μεταφέρουμε. Εάν θέλουμε να μεταφέρουμε ένα φάκελο, τότε το filename1 είναι η διεύθυνση του συγκεκριμένου φακέλου και εάν επιθυμούμε να μεταφέρουμε μόνο τα περιεχόμενα του φακέλου, τότε προσθέτουμε το χαρακτήρα / στο τέλος του filename1. Το slice_name είναι το όνομα του μερίσματος στο οποίο ανήκουμε, το hostname είναι το όνομα της μηχανής στην οποία θα μεταφέρουμε τα αρχεία. Το filename2, είναι η διεύθυνση στη μηχανή, στην οποία

θα μεταφέρουμε τα αρχεία. Εάν τώρα, επιθυμούμε να μεταφέρουμε αρχεία από κάποια μηχανή στον υπολογιστή μας, τότε εκτελούμε την ίδια εντολή, μεταφέροντας το filename1 στο τέλος.

Περισσότερες πληροφορίες πως μπορεί κάποιος να χρησιμοποιήσει το PlanetLab και τις δυνατότητες που του προσφέρει, υπάρχουν στα εγχειρίδια χρήσης του που είναι διαθέσιμα στην ιστοσελίδα του [13,14].

Κεφάλαιο 3

Περιγραφή Αλγορίθμων Εκλογής Αρχηγού

3.1 Πρόβλημα Εκλογής Αρχηγού

Το πρόβλημα της Εκλογής Αρχηγού (Leader Election) [21] διατυπώθηκε για πρώτη φορά με τον αλγόριθμο LeLann το 1977, από τον Le Lann και αποτυπώνει ένα από τα βασικότερα προβλήματα που εμφανίζονται σε κατανεμημένα συστήματα. Το πρόβλημα αυτό εμφανίζεται σε περιπτώσεις όπου μια διεργασία μεταξύ πολλών άλλων, πρέπει να επιλεγεί προκειμένου να εκτελέσει μια συγκεκριμένη ενέργεια. Μια τέτοια ενέργεια μπορεί να είναι για παράδειγμα η αρχικοποίηση διεργασιών ή κατανεμημένων αλγορίθμων, ο συντονισμός διεργασιών, η εκτέλεση συγκεντρωτικών αλγορίθμων όπου δεν υπάρχει διεργασία που να αναλαμβάνει το ρόλο του εναρκτή του αλγορίθμου, η αρχικοποίηση του δικτύου μετά από αποτυχία ή αδιέξοδο ή η κατασκευή ενός γεννητορικού δέντρου με τη διεργασία αρχηγό ως ρίζα. Πολλοί κατανεμημένοι αλγόριθμοι προϋποθέτουν την ύπαρξη μιας διεργασίας αρχηγού, η οποία όμως δεν μπορεί να καθοριστεί εκ των προτέρων αφού δεν είναι ακριβής από την αρχή ο αριθμός των διεργασιών του δικτύου. Έτσι είναι απαραίτητη η ύπαρξη ενός κατάλληλου αλγορίθμου για την εύρεση της διεργασίας αρχηγού.

Κάθε διεργασία του συστήματος έχει ένα μοναδικό αναγνωριστικό (id) το οποίο είναι γνωστό μόνο στην ίδια και το μεταδίδει στις άλλες διεργασίες μέσω της αποστολής μηνυμάτων. Επίσης, μπορεί να καταλήξει σε δύο τερματικές καταστάσεις: στην κατάσταση αρχηγού, που αναδεικνύει την διεργασία ως αρχηγό (leader) ή στην κατάσταση μη-αρχηγού (ή χαμένου), που αναδεικνύει τη διεργασία ως μη-αρχηγό. Σε κάθε εκτέλεση του αλγορίθμου ικανοποιούνται οι εξής συνθήκες:

- Συνθήκη Ασφαλείας : Μόνο μία διεργασία είναι αρχηγός ενώ όλες οι άλλες διεργασίες βρίσκονται στην κατάσταση του μη-αρχηγού.
- Συνθήκη Ζωτικότητας : Κάθε διεργασία είναι είτε αρχηγός είτε μη-αρχηγός.

Ένας αλγόριθμος, επιλύει το πρόβλημα εκλογής αρχηγού εάν ικανοποιεί τις εξής προϋποθέσεις :

1. Όλες οι διεργασίες εκτελούν τον ίδιο τοπικό αλγόριθμο.
2. Ο αλγόριθμος είναι κατανεμημένος, δηλαδή κάθε υπολογισμός του μπορεί να ξεκινήσει από ένα αυθαίρετο, μη κενό σύνολο διεργασιών, οι οποίες βρίσκονται στην ίδια αρχική κατάσταση.
3. Σε κάθε τελική κατάσταση του αλγορίθμου, υπάρχει ακριβώς μία διεργασία που βρίσκεται στην κατάσταση του αρχηγού και όλες οι άλλες βρίσκονται στην κατάσταση του μη-αρχηγού. Η διεργασία αρχηγός αναλαμβάνει να ενημερώσει όλες τις υπόλοιπες για την εκλογή της.

Τους αλγόριθμους εκλογής αρχηγού μπορούμε να τους κατηγοριοποιήσουμε ως εξής [21]:

- Αλγόριθμοι εύρεσης ακρότατου (extrema-finding algorithm) : Είναι αλγόριθμοι στους οποίους ο αρχηγός δεν αντιπροσωπεύει προτιμήσεις και καθορίζεται ως η διεργασία με μεγαλύτερο (ή μικρότερο) αναγνωριστικό.
- Αλγόριθμοι βασισμένοι σε προτιμήσεις (preferences-based algorithm) : Είναι αλγόριθμοι στους οποίους η εκλογή αρχηγού βασίζεται σε προτιμήσεις των διεργασιών, οι οποίες στηρίζονται σε εκτιμήσεις περί αξιοπιστίας, τοποθεσίας, κ.λπ..
- Πιθανοτικοί αλγόριθμοι (probabilistic algorithms): Είναι αλγόριθμοι οι οποίοι δεν θεωρούν γνωστά τα αναγνωριστικά των διεργασιών.
- Αλγόριθμοι βασισμένοι στην τοπολογία του υποκείμενου δικτύου : Είναι αλγόριθμοι που βασίζονται στην τοπολογία του υποκείμενου δικτύου. Οι τοπολογίες αυτές είναι η τοπολογία δέντρου, δακτυλίου και ισχυρά συνδεδεμένου γράφου, οι οποίες αντιπροσωπεύουν τους αλγόριθμους εκλογής αρχηγού που θα περιγράψουμε και υλοποιήσουμε στη συνέχεια.

Αντιπροσωπευτικά παραδείγματα αλγορίθμων εκλογής αρχηγού είναι : ο αλγόριθμος των LeLann, Chang και Roberts (LCR), ο αλγόριθμος του Peterson (PetersonLE), ο αλγόριθμος

των Hirschberg και Sinclair (HS), ο αλγόριθμος FloodMax, ο αλγόριθμος των Itai και Rodeh (IR) και ο αλγόριθμος TimeSlice.

3.2 Αλγόριθμος με Τοπολογία Δέντρου

Ο πρώτος αλγόριθμος που θα μελετήσουμε είναι ένας αλγόριθμος που βασίζεται σε δίκτυα με τοπολογία δέντρου [21]. Ο αλγόριθμος αυτός μπορεί να εφαρμοστεί και σε δίκτυα με οποιαδήποτε τοπολογία, με την προϋπόθεση ότι υπάρχει και μπορεί να βρεθεί ένα γεννητορικό δέντρο πάνω στο συγκεκριμένο δίκτυο. Στο σημείο αυτό αξίζει να σημειωθεί πως ο αλγόριθμος είναι τέτοιος έτσι ώστε να είναι εύκολη η κατασκευή της δομής του δέντρου που εμείς επιθυμούμε.

3.2.1 Περιγραφή Αλγορίθμου

Ο αλγόριθμος αυτός εκλέγει αρχηγό, τη διεργασία με το μικρότερο αναγνωριστικό. Όλες οι διεργασίες είναι εναρκτές του αλγορίθμου και κάθε διεργασία γνωρίζει μόνο τα αναγνωριστικά των γειτόνων της. Δεν γνωρίζει τα αναγνωριστικά άλλων διεργασιών του δικτύου αλλά ούτε και τον συνολικό αριθμό των διεργασιών. Υποθέτουμε ότι στην υποκείμενη τοπολογία δικτύου υπάρχουν n διεργασίες. Τα μηνύματα που ανταλλάσσονται μεταξύ των διεργασιών μπορεί να είναι είτε μηνύματα < type, id > είτε μηνύματα < tok, u_p >, όπου type είναι ο τύπος του μηνύματος που αποστέλλεται, id το αναγνωριστικό του αποστολέα και u_p το μικρότερο αναγνωριστικό που έχει λάβει η διεργασία. Στη συγκεκριμένη εκδοχή του αλγόριθμου που μελετάμε όλα τα μηνύματα είναι τύπου tok και αντιπροσωπεύουν μηνύματα σκυτάλης. (Σε περίπτωση που δεν ήταν όλες οι διεργασίες εναρκτές, τότε θα χρησιμοποιούσαμε ακόμα ένα τύπο μηνύματος, wake_up, για το ξύπνημα των υπόλοιπων διεργασιών.)

Ο αλγόριθμος με τοπολογία δέντρου, περιγράφεται στο Σχήμα 3.1. Κάθε διεργασία $p_ι$ περιμένει να λάβει μηνύματα < tok, id > από όλους τους γείτονές της εκτός από έναν, έστω p'. Κάθε φορά που η διεργασία λαμβάνει μήνυμα < tok, id > , υπολογίζει το μικρότερο αναγνωριστικό u_p μεταξύ αυτού που έλαβε (id) και του δικού της. Όταν λάβει συνολικά n-1 μηνύματα, όπου n ο αριθμός των γειτόνων της, τότε στέλνει ένα μήνυμα < tok, u_p >, με το μικρότερο αναγνωριστικό από όσα έχει λάβει (u_p) στο γείτονά της p', από τον οποίον δεν έχει λάβει μήνυμα. Η διεργασία περιμένει να λάβει ένα παρόμοιο μήνυμα από αυτόν και το αναγνωριστικό που θα λάβει θα κρίνει ποια διεργασία θα εκλεγεί αρχηγός. Όταν λάβει αυτό

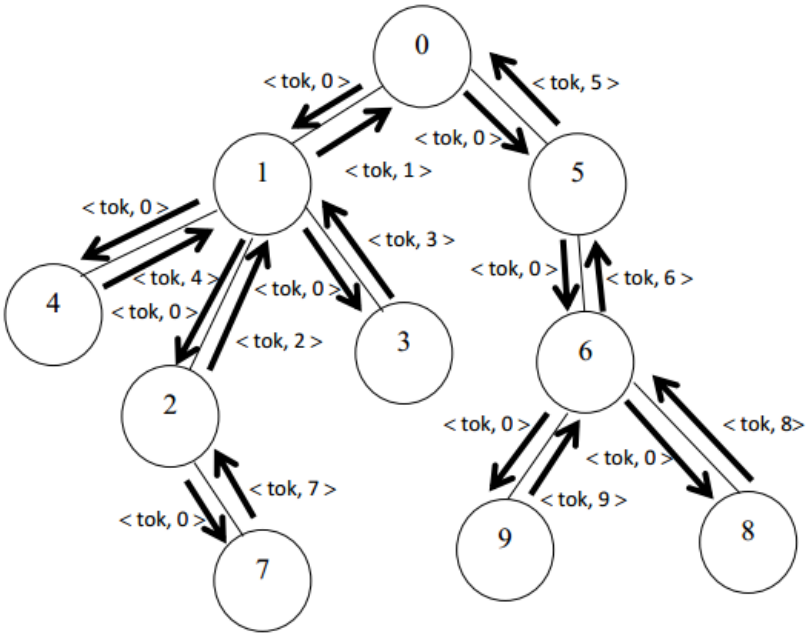
το μήνυμα από τον γείτονά της p' , η διεργασία εντοπίζει και πάλι το μικρότερο αναγνωριστικό (u_p) και εάν είναι το ίδιο με το αναγνωριστικό της, τότε εκλέγεται ως αρχηγός. Η διεργασία, ασχέτως με το αν εκλεγεί ή όχι αρχηγός, στέλλει μηνύματα < tok, u_p >, σε όλους τους γείτονές της εκτός από τον p', με το u_p που έχει υπολογίσει. Με τον τρόπο αυτό, όλες οι διεργασίες ενημερώνονται για το αναγνωριστικό της διεργασίας αρχηγού.

Κάθε διεργασία p_i
  Περιμένει να λάβει μήνυμα < tok, id > από όλους τους γείτονές της, εκτός από έναν, έστω τον p'
  Κάθε φορά που λαμβάνει ένα μήνυμα < tok, id >
    Εντοπίζει το μικρότερο αναγνωριστικό, έστω u_p, ανάμεσα σε αυτά που έχει λάβει και στο δικό της
  Όταν λάβει μηνύματα < tok, id > από όλους τους γείτονές της, εκτός από τον p'
    Στέλνει μήνυμα < tok, u_p > στον p'
    Περιμένει μήνυμα < tok, u_p > από τον p'
    Όταν λάβει μήνυμα < tok, u_p > από τον p'
      Εντοπίζει και πάλι το μικρότερο αναγνωριστικό, έστω u_p, από αυτά που έχει λάβει και το δικό της
      Αν αυτό είναι το δικό της
        Ανακηρύσσεται αρχηγός
      Στέλνει μήνυμα < tok, u_p > σε όλους τους γείτονές της εκτός του p'

Σχήμα 3.1 Αλγόριθμος με τοπολογία δέντρου

Στο Σχήμα 3.2 φαίνεται ένα παράδειγμα εκτέλεσης του παραπάνω αλγορίθμου, για 10 διεργασίες. Αρχικά οι διεργασίες με αναγνωριστικά 3, 4, 7, 8 και 9 στέλλουν μήνυμα τύπου <tok, id>, όπου id είναι το αναγνωριστικό της κάθε διεργασίας, στις διεργασίες 1, 1, 2, 6 και 6, αντίστοιχα. Η διεργασία 2 λαμβάνει το αναγνωριστικό 7, το συγκρίνει με το δικό της και επειδή το δικό της είναι το μικρότερο, στέλνει μήνυμα < tok, 2> στην διεργασία 1. Η διεργασία 6 η οποία λαμβάνει τα αναγνωριστικά 8 και 9, εντοπίζει το δικό της αναγνωριστικό ως το πιο μικρό και το προωθεί στην διεργασία 5. Η διεργασία 1 προωθεί και αυτή το αναγνωριστικό της στη διεργασία 0 αφού τα αναγνωριστικά που λαμβάνει (2, 3 και 4) είναι μικρότερα από αυτό. Το δικό της αναγνωριστικό προωθεί και η διεργασία 5 μιας και το αναγνωριστικό 6 είναι μεγαλύτερο από το δικό της. Η διεργασία 0, η οποία λαμβάνει

μηνύματα με αναγνωριστικά 1 και 5, βρίσκει πως η ίδια έχει το μικρότερο αναγνωριστικό και διαπιστώνει ότι αυτή είναι η αρχηγός. Έτσι στέλνει πρώτα στη διεργασία 1 μήνυμα < tok, 0> και ακολούθως στέλνει μήνυμα < tok, u_p >, όπου u_p = 0 στη διεργασία 5, αφού είναι ο μοναδικός γείτονάς της εκτός από την διεργασία 1. Η διεργασία 5, λαμβάνει το μήνυμα που έχει στείλει η διεργασία 0 και συγκρίνοντάς και πάλι το αναγνωριστικό που έχει λάβει με το δικό της, αντιλαμβάνεται ότι αρχηγός είναι η διεργασία 0, της οποίας το αναγνωριστικό προωθεί σε όλους τους γείτονές της εκτός από τη διεργασία 0. Το ίδιο κάνουν και οι διεργασίες 1, 2 και 6, με αποτέλεσμα να γνωστοποιηθεί σε όλους ότι αρχηγός είναι η διεργασία με αναγνωριστικό 0.



Σχήμα 3.2 Παράδειγμα εκτέλεσης του αλγορίθμου εκλογής
δέντρου σε δίκτυα με τοπολογία δέντρου

3.2.2 Ανάλυση Αλγορίθμου

Η ορθότητα του αλγορίθμου επιβεβαιώνεται από το ότι εκλέγεται ως αρχηγός η διεργασία με το μικρότερο αναγνωριστικό με αποτέλεσμα μόνο μία διεργασία να καταλήγει στην κατάσταση αρχηγού ενώ όλες οι υπόλοιπες διεργασίες να καταλήγουν σε κατάσταση μη-αρχηγού.

Κατά την εκτέλεση του αλγορίθμου, η κάθε διεργασία του δικτύου επικοινωνεί με τους γείτονές της, ανταλλάσσοντας μηνύματα μεταξύ τους. Συγκεκριμένα στέλνονται 2 μηνύματα σε κάθε κανάλι επικοινωνίας μεταξύ δύο διεργασιών και ο συνολικός αριθμός των μηνυμάτων που ανταλλάσσονται στον αλγόριθμο εκλογής αρχηγού με τοπολογία δέντρου

είναι 2∗(n-1). Επομένως η πολυπλοκότητα των μηνυμάτων, καθώς επίσης και η χρονική πολυπλοκότητα του αλγορίθμου με τοπολογία δέντρου είναι της τάξεως O(n).

Ο αλγόριθμος παρουσιάζει τις εξής δύο ασάφειες:

1. Δεν ορίζει τι θα συμβεί στην περίπτωση που μια διεργασία αποτύχει για οποιοδήποτε λόγο, δηλαδή σταματήσει να είναι ενεργή.
2. Δεν διατυπώνει τις ενέργειες μιας διεργασία όταν δεν λάβει τα αναμενόμενα μηνύματα από κάποια διεργασία, δηλαδή χαθούν κάποια από τα μηνύματα καθώς μεταφέρονται στο κανάλι επικοινωνίας.

3.3 Αλγόριθμος με Τοπολογία Δακτυλίου

Ο δεύτερος αλγόριθμος είναι αλγόριθμος σε δίκτυο με τοπολογία δακτυλίου. Ο αλγόριθμος αυτός εφαρμόζεται και σε δίκτυα με οποιαδήποτε τοπολογία, με την προϋπόθεση ότι υπάρχει και μπορεί να βρεθεί ένας επικαλυπτικός δακτύλιος στο συγκεκριμένο δίκτυο. Να σημειώσουμε πως ο αλγόριθμος αυτός, βασίζεται στον αλγόριθμο των LeLann, Chag και Roberts (LCR) [21], τον οποίο και περιγράφουμε με συντομία στο υποκεφάλαιο που ακολουθεί. Επίσης να καθορίσουμε πως όταν αναφερόμαστε σε τοπολογία δακτυλίου, θεωρούμε δακτύλιο μίας κατεύθυνσης και συγκεκριμένα δεξιόστροφης Η σειρά εμφάνισης των διεργασιών στο δακτύλιο είναι τυχαία και γι’ αυτό θα εφαρμόσουμε διαφορετικές εκτελέσεις του ίδιου αλγορίθμου έτσι ώστε να μελετήσουμε πως αυτό επηρεάζει την χρονική και επικοινωνιακή πολυπλοκότητα του αλγορίθμου.

3.3.1 Αλγόριθμος LCR

Περιγραφή Αλγορίθμου

Ο αλγόριθμος LCR είναι ο πρώτος αλγόριθμος που εμφανίστηκε για το πρόβλημα εκλογής αρχηγού κα είναι αλγόριθμος με τοπολογία δακτυλίου. Οφείλεται στον LeLann, και σε βελτιώσεις που είχαν εισηγηθεί οι Chang και Roberts. Ο αλγόριθμος δε γνωρίζει τον ακριβή αριθμό των διεργασιών που υπάρχουν στο δίκτυο και είναι δυνατόν να μην εκτελεστεί ταυτόχρονα απ’ όλες τις διεργασίες αφού οι διεργασίες χωρίζονται σε εναρκτές και μη εναρκτές. Οι εναρκτές είναι οι διεργασίες που εκτελούν τον αλγόριθμο και διεκδικούν την αρχηγία ενώ οι μη εναρκτές είναι οι διεργασίες που δεν μπορούν να εκλεγούν ως αρχηγός και απλά προωθούν τυχόν μηνύματα που λαμβάνουν στην επόμενη διεργασία στο δακτύλιο.

Κάθε διεργασία έχει ένα μοναδικό αναγνωριστικό το οποίο γνωρίζει μόνο η ίδια, στέλνει μηνύματα μόνο στην επόμενη διεργασία και λαμβάνει μηνύματα μόνο από την προηγούμενή της (με τη λέξη ‘επόμενη’ εννοούμε για τη διεργασία i τη διεργασία i+1 και με τη λέξη ‘προηγούμενη’ εννοούμε για τη διεργασία i τη διεργασία i-1). Αρχηγός εκλέγεται η διεργασία με το μικρότερο αναγνωριστικό. Επίσης, τα μηνύματα που ανταλλάσσονται μεταξύ των διεργασιών είναι της μορφής < type, id >, όπου type είναι ο τύπος του μηνύματος (μπορεί να είναι tok για μήνυμα σκυτάλης και leader για μήνυμα αρχηγού) και id το αναγνωριστικό της διεργασίας που στέλνει το μήνυμα.

Στον αρχικό αλγόριθμο του LeLann κάθε διεργασία που είναι εναρκτής, προωθεί στην επόμενή της διεργασία κάθε μήνυμα που λαμβάνει και μία διεργασία εκλέγεται αρχηγός όταν λάβει μήνυμα με αναγνωριστικό ίδιο με το δικό της. Κατά τη διάρκεια του αλγορίθμου αυτού στέλνονταν πάρα πολλά μηνύματα (έχει επικοινωνιακή πολυπλοκότητα τάξεως $O(n^2)$), οι Chang και Roberts έκαναν κάποιες τροποποιήσεις στον αλγόριθμο του LeLann, διατυπώνοντας έτσι τον αλγόριθμο LCR. Στον αλγόριθμο αυτό, μια διεργασία δεν προωθεί τα μηνύματα με id μεγαλύτερο από το δικό της, δηλαδή δεν προωθούνται τα μηνύματα των διεργασιών που δεν μπορούν να γίνουν αρχηγοί. Έτσι μειώνεται η πολυπλοκότητα των μηνυμάτων του αλγορίθμου. Το μήνυμα που στέλνει η διεργασία που πρόκειται να εκλεγεί αρχηγός, είναι το μοναδικό μήνυμα που καταφέρνει να φτάσει πίσω στη διεργασία που το προήγαγε.

Στο Σχήμα 3.3 παρουσιάζεται επιγραμματικά ο αλγόριθμος LCR. Κάθε διεργασία εναρκτής στέλνει μήνυμα < tok, id > στην επόμενή της διεργασία και περιμένει να λάβει αντίστοιχο μήνυμα. Όταν λάβει κάποιο μήνυμα τότε ελέγχει το αναγνωριστικό του μηνύματος με το δικό της. Εάν είναι το ίδιο με το δικό της, τότε ανακηρύσσεται αρχηγός και πρέπει να ενημερώσει τις υπόλοιπες διεργασίες για την εκλογή της. Έτσι, στέλνει leader μήνυμα στην επόμενη της διεργασία, και κάθε διεργασία όταν λάβει τέτοιο μήνυμα ενημερώνεται για το αναγνωριστικό της διεργασίας αρχηγού και το προωθεί στην επόμενη διεργασία. Εάν όμως το αναγνωριστικό που λαμβάνει είναι μικρότερο από το δικό της, προωθεί το μήνυμα στην επόμενή της διεργασία και περιμένει να λάβει τυχόν άλλα μηνύματα. Κάθε διεργασία μη εναρκτής, εάν λάβει κάποιο μήνυμα < tok, id > τότε απλώς το προωθεί στην επόμενη διεργασία.

```
Κάθε διεργασία p_i που είναι εναρκτής
    Στέλνει μήνυμα < tok, id >
    Περιμένει να λάβει μήνυμα < tok, id >
    Όταν λάβει μήνυμα < tok, id >
        Αν το αναγνωριστικό που έλαβε είναι το δικό της
```

Ανακηρύσσεται αρχηγός

Στέλνει μήνυμα < leader, id >

Αν το αναγνωριστικό που έλαβε είναι μικρότερο από το δικό της

Προωθεί το μήνυμα < tok, id >

Περιμένει να λάβει μήνυμα < tok, id >

Όταν λάβει μήνυμα < leader, id >

Προωθεί το μήνυμα < leader, id >

Κάθε διεργασία που δεν είναι εναρκτής

Περιμένει να λάβει μήνυμα < tok, id >

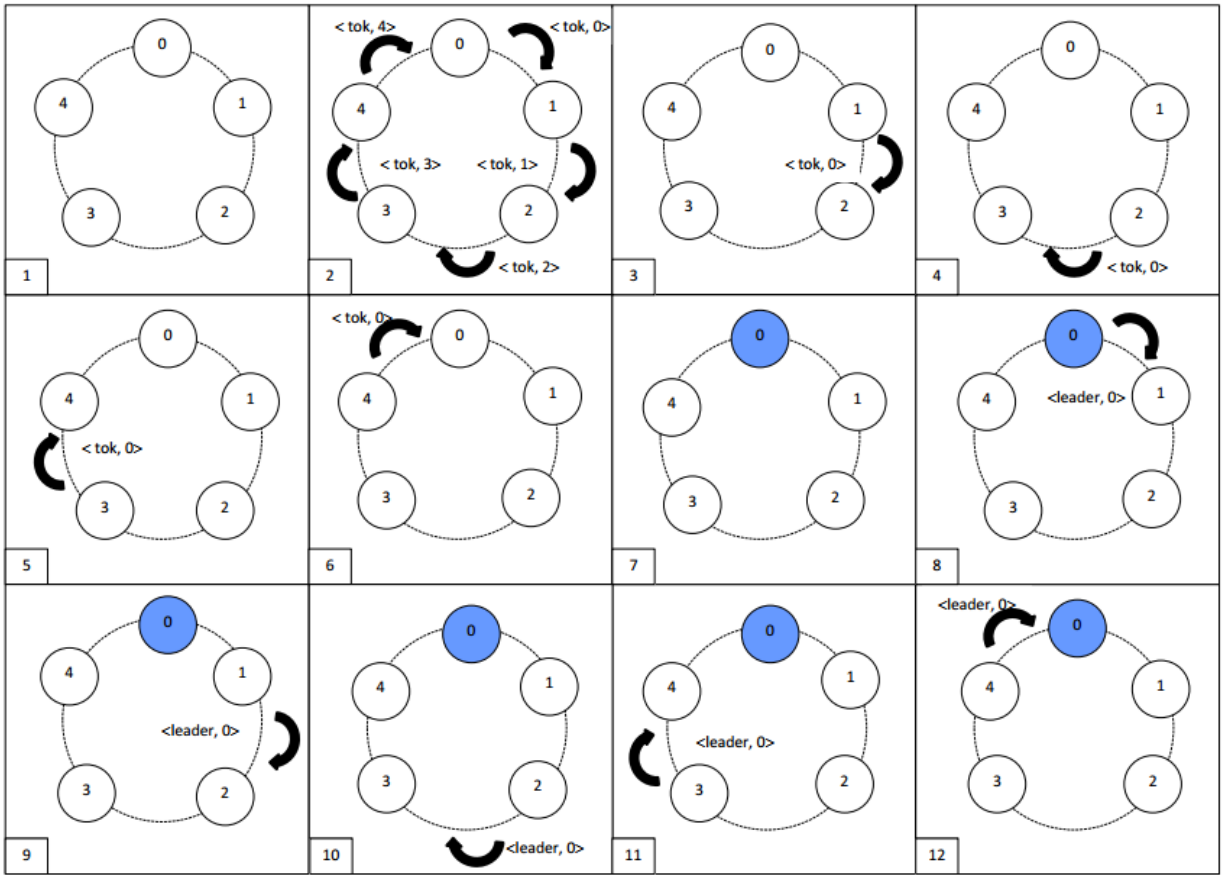
Προωθεί το μήνυμα < tok, id >

Όταν λάβει μήνυμα < leader, id >

Προωθεί το μήνυμα < leader, id >

Σχήμα 3.3 Αλγόριθμος LCR

Στο Σχήμα 3.4 φαίνεται ένα παράδειγμα εκτέλεσης του αλγορίθμου LCR, όπου υπάρχουν 5 διεργασίες και είναι αριθμημένες από 0 έως 4, δεξιόστροφα.



Σχήμα 3.4 Παράδειγμα εκτέλεσης αλγορίθμου LCR

Η κάθε διεργασία έχει μοναδικό αναγνωριστικό και γνωρίζει μόνο το δικό της. Η διάταξη των διεργασιών στον δακτύλιο φαίνεται στο Σχήμα 3.4.(1). Αρχικά, κάθε διεργασία στέλνει μήνυμα < tok, id > στην επόμενή της διεργασία (Σχήμα 3.4.(2)). Στη συνέχεια μόνο η διεργασία 1 θα προωθήσει το μήνυμα που έλαβε γιατί λαμβάνει αναγνωριστικό μικρότερο από το δικό της (Σχήμα 3.4.(3)). Παρόμοια ενεργούν η διεργασίες 2,3 και 4 στους επόμενους γύρους (Σχήματα 3.4.(4), 3.4.(5) και 3.4.(6)). Η διεργασία 0 η οποία λαμβάνει μήνυμα από την διεργασία 4, αντιλαμβάνεται ότι είναι η αρχηγός αφού λαμβάνει αναγνωριστικό ίδιο με το δικό της. Επομένως ανακηρύσσεται αρχηγός (Σχήμα 3.4.(7)) και ενημερώνει όλες τις υπόλοιπες διεργασίες (Σχήματα 3.4.(8) - 3.4.(12)).

Ανάλυση Αλγορίθμου

Ο αλγόριθμος εκλογής αρχηγού LCR είναι ορθός και αυτό αποδεικνύεται από το ότι στο τέλος κάθε εκτέλεσής του, η διεργασία με το μικρότερο αναγνωριστικό εκλέγεται αρχηγός. Αυτό συμβαίνει διότι τα μηνύματα κινούνται μόνο προς μία κατεύθυνση και το μήνυμα με το μικρότερο αναγνωριστικό θα περάσει απ' όλες τις διεργασίες μέχρι να επιστρέψει στην διεργασία που το έστειλε αρχικά, η οποία και θα ανακηρυχθεί ως αρχηγός.

Η επικοινωνιακή πολυπλοκότητα του αλγορίθμου εξαρτάται από τη διάταξη των διεργασιών στο δακτύλιο. Η καλύτερη περίπτωση εκτέλεσης του αλγορίθμου LCR είναι όταν όλες οι διεργασίες είναι εναρκτές και η διάταξή τους πάνω στο δακτύλιο είναι κατά φθίνουσα σειρά προς τη φορά του δακτυλίου, δηλαδή δεξιόστροφα. Η χρονική πολυπλοκότητα του αλγορίθμου σε αυτήν την περίπτωση είναι 2n, δηλαδή της τάξης O(n), αφού θα χρειαστούν ακριβώς n γύροι μέχρι να εκλεγεί η διεργασία με το μικρότερο αναγνωριστικό ως αρχηγός και n γύροι μέχρι να ανακοινωθεί η διεργασία αρχηγός σε όλες τις υπόλοιπες. Ο συνολικός αριθμός μηνυμάτων που στέλνονται στην περίπτωση αυτή είναι n+n-1 = 2n-1, δηλαδή O(n), διότι το μήνυμα της διεργασίας με το μικρότερο αναγνωριστικό θα σταλεί n-1 φορές και κάθε διεργασία θα στείλει μήνυμα με το αναγνωριστικό της 1 φορά. Η χειρότερη περίπτωση εκτέλεσης του αλγορίθμου είναι όταν όλες οι διεργασίες είναι εναρκτές και η διάταξή τους πάνω στο δακτύλιο είναι κατά αύξουσα σειρά προς τη φορά του δακτυλίου. Στην περίπτωση αυτή, για κάθε διεργασία η οποία δεν έχει το μικρότερο αναγνωριστικό θα προωθείται το μήνυμά της μέχρι να καταλήξει στη διεργασία με το μικρότερο αναγνωριστικό. Δηλαδή, η διεργασία με το μεγαλύτερο αναγνωριστικό θα προωθήσει όλα τα μηνύματα που θα λάβει αφού θα περιλαμβάνουν όλα μικρότερα αναγνωριστικά από την ίδια. Επίσης θα χρειαστούν επιπλέον n μηνύματα για την ενημέρωση των διεργασιών για τον αρχηγό. Επομένως, ο συνολικός αριθμός μηνυμάτων που θα σταλούν είναι $n+ (n-1)+ (n-2)+ \ldots + 1 + n = n^2 + n$,

δηλαδή της τάξης του $O(n^2)$. Επίσης, έχει αποδειχθεί πως εάν όλες οι διεργασίες είναι εναρκτές, τότε ο συνολικός αριθμός των μηνυμάτων που στέλνονται στην μέση περίπτωση είναι της τάξης O(nlogn).

3.3.2 Τροποποιημένος Αλγόριθμος

Ο αλγόριθμος που θα μελετήσουμε με τοπολογία δακτυλίου βασίζεται στον αλγόριθμο LRC και επιτυγχάνει επιπλέον μείωση της πολυπλοκότητας των μηνυμάτων. Θεωρούμε ότι υπάρχουν n διεργασίες στο δίκτυο, όλες οι διεργασίες είναι εναρκτές, δηλαδή ο αλγόριθμος εκτελείται ταυτόχρονα απ' όλες τις διεργασίες, και αρχηγός θα εκλεγεί και πάλι η διεργασία με το μικρότερο αναγνωριστικό. Στην παραλλαγή αυτή του αλγορίθμου, όταν μία διεργασία λαμβάνει μήνυμα < tok, id >, υπολογίζει το μικρότερο αναγνωριστικό, έστω u_p, μεταξύ αυτού που έλαβε και του δικού της και προωθεί μόνο τα μηνύματα που έχουν αναγνωριστικό μικρότερο από το u_p.

Το Σχήμα 3.5 περιγράφει την παραλλαγή του αλγορίθμου με τοπολογία δέντρου που μελετάμε.

```
Κάθε διεργασία p_i
    Θέτει το μικρότερο αναγνωριστικό, έστω u_p, ίσο με το αναγνωριστικό της
    Στέλνει μήνυμα < tok, id >
    Περιμένει να λάβει μήνυμα < tok, id >
    Όταν λάβει μήνυμα < tok, id >
        Αν το αναγνωριστικό που έλαβε είναι το δικό της
            Ανακηρύσσεται αρχηγός
            Στέλνει μήνυμα < leader, id >
        Αν το αναγνωριστικό που έλαβε είναι μικρότερο από το μικρότερο αναγνωριστικό u_p
            Προωθεί το μήνυμα < tok, id >
            Περιμένει να λάβει μήνυμα < tok, id >
    Όταν λάβει μήνυμα < leader, id >
        Προωθεί το μήνυμα < leader, id >
```

Σχήμα 3.5 Αλγόριθμος με τοπολογία δακτυλίου

Αρχικά, όλες οι διεργασίες αρχικοποιούν το μικρότερο αναγνωριστικό (u_p) που γνωρίζουν με το αναγνωριστικό της. Κάθε διεργασία p_i στέλνει μήνυμα < tok, id > στη επόμενή του

διεργασία και περιμένει να λάβει παρόμοιο μήνυμα από την προηγούμενη διεργασία. Όταν λάβει μήνυμα < tok, id >, το προωθεί μόνο όταν το αναγνωριστικό που λαμβάνει είναι μικρότερο από το μικρότερο αναγνωριστικό u_p και περιμένει μέχρι να λάβει το επόμενο μήνυμα. Εάν όμως το αναγνωριστικό που λαμβάνει είναι το ίδιο με το δικό της, τότε ανακηρύσσεται αρχηγός και ενημερώνει όλες τις υπόλοιπες διεργασίες.

Από την εκτέλεση του πιο πάνω αλγορίθμου, η διεργασία με το μικρότερο αναγνωριστικό θα εκλεγεί αρχηγός και όλες οι υπόλοιπες θα ενημερωθούν για την εκλογή της, κάτι που έγκειται την ορθότητα του. Όπως και στον αλγόριθμο LCR έτσι και στην παραλλαγή του που περιγράψαμε πιο πάνω, η χρονική πολυπλοκότητα είναι της τάξης O(n), η πολυπλοκότητα των μηνυμάτων στην καλύτερη περίπτωση είναι τάξεως O(n), στη μέση περίπτωση είναι τάξεως O(nlogn) και στη χειρότερη περίπτωση τάξεως $O(n^2)$. Η καλύτερη περίπτωση εκτέλεσης του αλγορίθμου προκύπτει όταν η διάταξη των διεργασιών στον δακτύλιο είναι κατά φθίνουσα σειρά προς τη φορά του δακτυλίου (δεξιόστροφα) και η χειρότερη περίπτωση εκτέλεσης του αλγορίθμου όταν η διάταξη των διεργασιών είναι κατά αύξουσα σειρά. Είναι σαφές ότι στέλνονται λιγότερα μηνύματα σε σχέση με τον αριθμό των μηνυμάτων που στέλνονται με τον αλγόριθμο LCR, γιατί στον αλγόριθμο LCR κάθε διεργασία προωθεί όλα τα μηνύματα με αναγνωριστικό μικρότερο από το δικό της, ενώ στον συγκεκριμένο αλγόριθμο κάθε διεργασία προωθεί κάθε φορά μηνύματα που έχουν αναγνωριστικό μικρότερο από όσα είχε λάβει.

Να σημειώσουμε ότι ο αλγόριθμος με τοπολογία δακτυλίου εμφανίζει τις ίδιες ακριβώς ασάφειες με τον αλγόριθμο σε τοπολογία δέντρου.

3.4 Αλγόριθμος με Τοπολογία Ισχυρά Συνδεδεμένου Γράφου

Ο τρίτος αλγόριθμος θα μελετήσαμε είναι μια μικρή παραλλαγή του αλγορίθμου του ‘νταή’ (bully algorithm) που προτάθηκε το 1982 από τον Garcia Molina και είναι αλγόριθμος σε ισχυρά συνδεδεμένα δίκτυα [21].

3.4.1 Περιγραφή Αλγορίθμου

Ο αλγόριθμος αυτός προϋποθέτει ένα σύγχρονο σύστημα και σε αντίθεση με τους προηγούμενους αλγόριθμους που μελετήσαμε, εκλέγει ως αρχηγό τη διεργασία με το μεγαλύτερο αναγνωριστικό. Κάθε διεργασία έχει ένα μοναδικό αναγνωριστικό , γνωρίζει τα αναγνωριστικά όλων των άλλων διεργασιών, είναι εναρκτής του αλγορίθμου και μπορεί να

αποτυγχάνει και να επανέρχεται οποιαδήποτε στιγμή. Οι διεργασίες δεν γνωρίζουν ποιες άλλες διεργασίες του συστήματος είναι ενεργές και ποιες όχι. Υποθέτουμε ότι στην υποκείμενη τοπολογία δικτύου υπάρχουν n διεργασίες.

Στον συγκεκριμένο αλγόριθμο, υπάρχουν τα ακόλουθα είδη μηνυμάτων:

1. Μήνυμα 'Election', το οποίο στέλνεται από μία διεργασία η οποία αντιλαμβάνεται ότι ο αρχηγός δεν είναι ενεργός και απαιτείται η εκλογή νέου αρχηγού.
2. Μήνυμα 'Reply Election', το οποίο στέλνεται ως απάντηση στο μήνυμα 'Election'.
3. Μήνυμα 'Leader', το οποίο στέλνεται από την διεργασία που εκλέγεται αρχηγός για να ενημερώσει όλες τις υπόλοιπες διεργασίες για την εκλογή της.
4. Μήνυμα 'Check Leader', το οποίο στέλνεται από μια διεργασία η οποία θέλει να ελέγξει εάν η τρέχον διεργασία αρχηγός είναι ενεργή.
5. Μήνυμα 'Reply Check Leader', το οποίο στέλνεται από τη διεργασία αρχηγός ως απάντηση στο μήνυμα 'Reply Check Leader'.

Στο Σχήμα 3.6 φαίνεται ο αλγόριθμος του 'νταή'. Εάν μια διεργασία η οποία είναι ενεργή και έχει το μεγαλύτερο αναγνωριστικό τότε αναγνωρίζει ότι είναι αρχηγός και ενημερώνει όλες τις υπόλοιπες διεργασίες, στέλνοντάς του μήνυμα 'Leader'. Εάν όμως δεν έχει το μεγαλύτερο αναγνωριστικό, τότε στέλνει μήνυμα 'Check Leader' στη διεργασία αρχηγό για να ελέγξει εάν είναι ενεργή και περιμένει για ένα χρονικό διάστημα να πάρει απάντηση. Εάν αντιληφθεί ότι η διεργασία αρχηγός δεν είναι ενεργή, δηλαδή δεν πάρει μήνυμα 'Reply Check Leader' από αυτήν, τότε αντιλαμβάνεται πως πρέπει να γίνει εκλογή νέου αρχηγού. Έτσι στέλνει σε όλες τις διεργασίες με μεγαλύτερο αναγνωριστικό από αυτήν ένα μήνυμα 'Election'. Περιμένει και πάλι για ένα χρονικό διάστημα μέχρι κάποια από αυτές της απαντήσει. Εάν πάρει μήνυμα 'Reply Election' τότε σημαίνει πως μια άλλη διεργασία με μεγαλύτερο αναγνωριστικό αναλαμβάνει τη διαδικασία εκλογής νέου αρχηγού και περιμένει μέχρι να λάβει μήνυμα όπου θα της ανακοινώνεται το αναγνωριστικό της διεργασία αρχηγού που θα εκλεγεί. Εάν όμως εξαντληθεί το χρονικό διάστημα στο οποίο πρέπει να της ανακοινωθεί ο αρχηγός, ξαναστέλνει μηνύματα 'Election' στις μεγαλύτερες από αυτήν διεργασίες. Εάν όμως η διεργασία δεν λάβει καμία απάντηση ως προς τα 'Election' μηνύματα που έχει στείλει τότε αντιλαμβάνεται πως είναι η μεγαλύτερη ενεργή διεργασία του συστήματος, ανακηρύσσεται αρχηγός και ενημερώνει όλες τις άλλες ενεργές διεργασίες για την εκλογή της. Στην περίπτωση που μια διεργασία, έστω p_i, λάβει μήνυμα 'Election' από μιαν άλλη διεργασία, έστω p_i', και έχει μεγαλύτερο αναγνωριστικό από την p_i', τότε της απαντά με μήνυμα 'Reply Election'. Με αυτόν τον τρόπο, η διαδικασία εκλογής αρχηγού μεταβιβάζεται στη διεργασία με το μεγαλύτερο αναγνωριστικό, η οποία θα εκλεγεί ως νέος αρχηγός. Εάν μια διεργασία δεν είναι ενεργή και ξαφνικά επανέρχεται στο σύστημα, τότε εάν

έχει το μεγαλύτερο αναγνωριστικό θα εκλεγεί αμέσως αρχηγός και θα ενημερώσει τις υπόλοιπες διεργασίες. Εάν όμως δεν είναι η μεγαλύτερη ενεργή διεργασία, θα ξεκινήσει της διαδικασία εκλογής νέου αρχηγού, στέλνοντας 'Election' μήνυμα στις διεργασίες με μεγαλύτερο αναγνωριστικό από αυτήν.

Αν μια διεργασία p_i είναι ενεργή
Αν η p_i έχει το μεγαλύτερο δυνατό αναγνωριστικό
Αυτοανακηρύσσεται αρχηγός
Γνωστοποιεί το αναγνωριστικό της σε όλες τις διεργασίες που έχουν μικρότερο αναγνωριστικό στέλνοντάς τους μήνυμα 'Leader'
Αν η p_i δεν έχει το μεγαλύτερο δυνατό αναγνωριστικό
Στέλνει στη διεργασία αρχηγό μήνυμα 'Check Leader'
Περιμένει για κάποιο συγκεκριμένο χρονικό διάστημα ένα μήνυμα 'Reply Check Leader' από τη διεργασία αρχηγό
Αν το χρονικό διάστημα εξαντληθεί και η p_i δεν λάβει κάποιο μήνυμα 'Reply Check Leader'
Στέλνει σε όλες τις διεργασίες με μεγαλύτερο αναγνωριστικό από αυτήν ένα μήνυμα 'Election'
Περιμένει για κάποιο συγκεκριμένο χρονικό διάστημα ένα μήνυμα 'Reply Election' από κάποια από αυτές
Αν η p_i λάβει κάποιο μήνυμα 'Reply Election'
Περιμένει για κάποιο συγκεκριμένο χρονικό διάστημα να λάβει το αναγνωριστικό του αρχηγού
Αν το χρονικό διάστημα εξαντληθεί και δεν έχει λάβει το αναγνωριστικό
Ξαναστέλνει μήνυμα 'Election'
Αν το χρονικό διάστημα εξαντληθεί και η p_i δεν έχει λάβει κανένα μήνυμα 'Reply Election'
Ανακηρύσσεται αρχηγός
Γνωστοποιεί στις διεργασίες που έχουν μικρότερο αναγνωριστικό ότι είναι αρχηγός στέλνοντάς τους μήνυμα 'Leader'
Αν η p_i λάβει ένα μήνυμα 'Election' από μια διεργασία p_i'
Αν έχει μεγαλύτερο αναγνωριστικό από την p_i'
Στέλνει ένα μήνυμα 'Reply Election' στην p_i'

Σχήμα 3.6 Αλγόριθμος με τοπολογία ισχυρά συνδεδεμένου γράφου
(Αλγόριθμος του 'νταή'- bully algorithm)

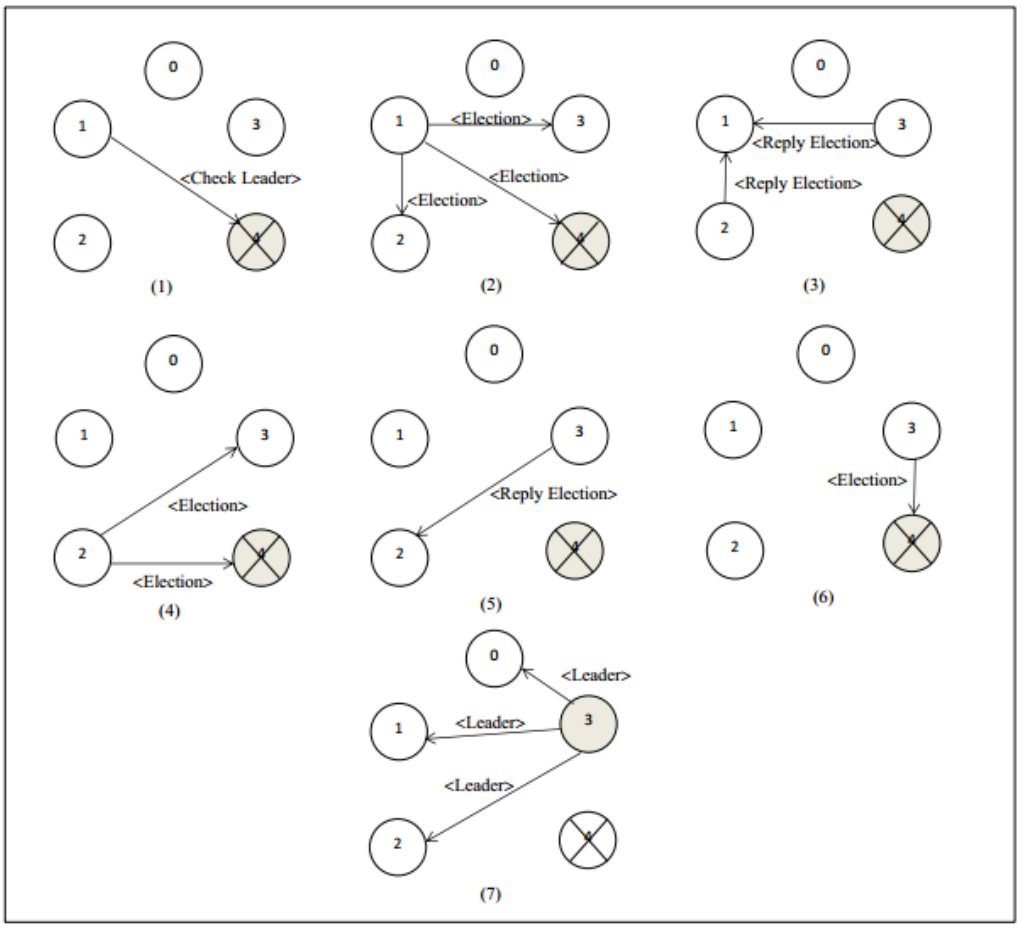
Στο Σχήμα 3.7 φαίνεται ένα παράδειγμα εκτέλεσης του πιο πάνω αλγορίθμου. Αρχηγός είναι η διεργασία με αναγνωριστικό τον αριθμό 4, η οποία όμως δεν είναι ενεργή. Η διεργασία 1 στέλνει μήνυμα 'Check Leader' στην διεργασία αρχηγό, όπου και αντιλαμβάνεται πως έχει αποτύχει αφού δεν παίρνει απάντηση. Τότε στέλνει στις διεργασίες με μεγαλύτερο αναγνωριστικό από αυτήν, δηλαδή στις διεργασίες 2, 3 και 4 μήνυμα 'Election' για να ξεκινήσει η διαδικασία εκλογής νέου αρχηγού. Οι διεργασίες 2 και 3 αφού είναι ενεργές απαντούν στη διεργασία 1 με μήνυμα 'Reply Election'. Στη συνέχεια η διεργασία 2 στέλνει μήνυμα 'Election' στα διεργασίες 3 και 4 και παίρνει απάντηση μόνο από την διεργασία 3, η οποία είναι η μοναδική ενεργή διεργασία με μεγαλύτερο αναγνωριστικό από αυτήν. Με τη σειρά της, η διεργασία 3 στέλνει μήνυμα 'Election' στη διεργασία 4 που είναι μεγαλύτερή της και αφού περάσει κάποιο χρονικό διάστημα χωρίς να πάρει απάντηση, ανακηρύσσεται αρχηγός αφού είναι η μεγαλύτερη ενεργή διεργασία που υπάρχει στο σύστημα τη δεδομένη χρονική στιγμή. Έτσι αρχηγός εκλέγεται η διεργασία 3, η οποία ενημερώνει όλες τις διεργασίες με μικρότερο αναγνωριστικό από αυτήν για την εκλογή της, με την αποστολή μηνύματος 'Leader'.

Ακόμη ένα παράδειγμα είναι αυτό του Σχήματος 3.8 στο οποίο φαίνεται η εκτέλεση του αλγορίθμου όταν μια αποτυχημένη διεργασία (διεργασία 4) επανέλθει στο σύστημα. Η συγκεκριμένη διεργασία, επειδή έχει το μεγαλύτερο αναγνωριστικό από όλες τις υπόλοιπες, ανακηρύσσεται αμέσως αρχηγός και ενημερώνει τις υπόλοιπες για την εκλογή της.

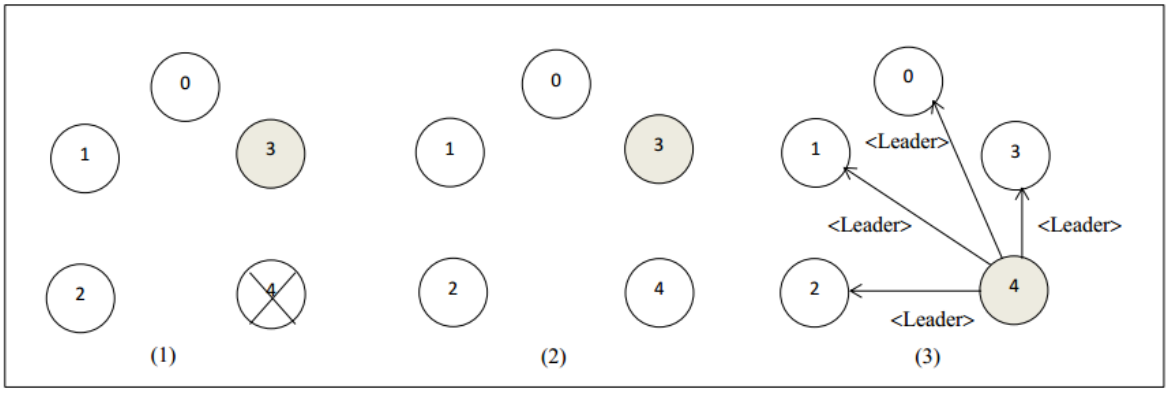
3.4.2 Ανάλυση Αλγορίθμου

Η ορθότητα του παραπάνω αλγορίθμου είναι εμφανής, εφόσον σε κάθε εκτέλεσή του αναμένεται να εκλεγεί αρχηγός, η ενεργή διεργασία με το μεγαλύτερο αναγνωριστικό. Ο αριθμός των μηνυμάτων που στέλνονται κατά την εκτέλεσή του, εξαρτάται από την κατάσταση των διεργασιών στο δίκτυο. Για παράδειγμα, στέλνονται περισσότερα μηνύματα όταν η διεργασία αρχηγός αποτύχει και θα εφαρμοστεί η διαδικασία εκλογής νέου αρχηγού. Έχει αποδειχθεί ότι η χρονική πολυπλοκότητα του αλγορίθμου Garcia Molina είναι της τάξης $O(n^2)$. Ο αριθμός μηνυμάτων που μπορεί να σταλεί κατά τη διάρκεια της εκτέλεσης του υπολογίζεται από τον τύπο $(n-r+1)(n-r) + (n-1)$, όπου n είναι ο συνολικός αριθμός των διεργασιών στο σύστημα και r η διεργασία που θα εκκινήσει τη διαδικασία εκλογής αρχηγού. Η χειρότερη περίπτωση όπου θα σταλεί ο μέγιστος αριθμός μηνυμάτων, είναι η περίπτωση όπου η διεργασία με το μικρότερο αναγνωριστικό, δηλαδή η διεργασία με αναγνωριστικό 0

αντιληφθεί ότι ο αρχηγός δεν είναι ζωντανός και θα ξεκινήσει τη διαδικασία εκλογής. Επομένως, ο αριθμός των μηνυμάτων είναι $n^2 + 2n - 1$, δηλαδή της τάξης $O(n^2)$.



Σχήμα 3.7 Παράδειγμα Εκτέλεσης του αλγορίθμου Garcia Molina



Σχήμα 3.8 Παράδειγμα Εκτέλεσης του αλγορίθμου Garcia Molina, όπου μια διεργασία επανέρχεται στο σύστημα

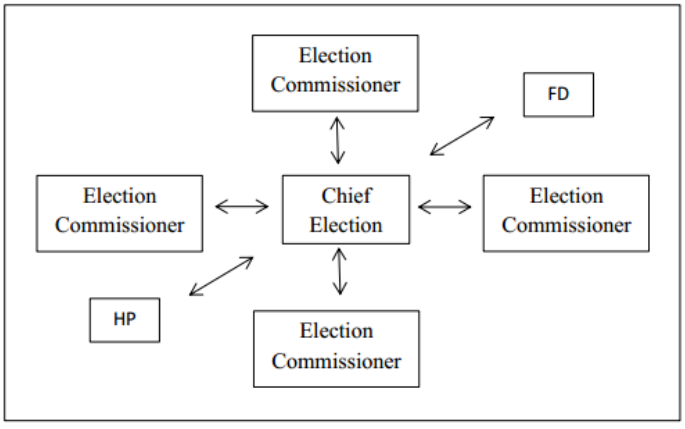
Ο αλγόριθμος αυτός παρουσιάζει μεγάλη επικοινωνιακή πολυπλοκότητα. Επίσης, δεν ελέγχει την παράδοση των μηνυμάτων και δεν διατυπώνει τη πρέπει να γίνει στην περίπτωση

όπου δύο διεργασίες ανακηρύξουν τον εαυτό τους αρχηγό, την ίδια ώρα. Για παράδειγμα, έστω ότι στο σύστημα υπάρχουν οι διεργασίες με αναγνωριστικά 0, 1, 2 και 3, όπου η διεργασία αρχηγός είναι η διεργασία 3. Η διεργασία 2 στέλνει μήνυμα στην διεργασία αρχηγό για να ελέγξει εάν είναι ενεργή και αυτή δεν παίρνει απάντηση στο προκαθορισμένο χρονικό διάστημα για κάποιο λόγο. Τότε η διεργασία 2 θα θεωρήσει πως η αρχηγός δεν είναι ενεργή και θα ανακηρύξει τον εαυτό της αρχηγό. Αυτό δεν πρέπει να συμβεί στο σύστημά μας γιατί τότε ο αλγόριθμος Garcia Molina δεν θα είναι ορθός.

3.5 Τροποποιημένος Αλγόριθμος Garcia Molina

Λόγω των μειονεκτημάτων που εμφανίζει ο αλγόριθμος του Garcia Molina για την εκλογή αρχηγού, διατυπώθηκαν αρκετές τροποποιημένες εκδοχές του. Θα μελετήσουμε μία από αυτές, και συγκεκριμένα τον τροποποιημένο αλγόριθμο Garcia Molina όπως αναφέρεται στο [9], ο οποίος χρησιμοποιεί την ιδέα του Election Commission (EC). Ο αλγόριθμος αυτός όχι μόνο μειώνει τον αριθμό των μηνυμάτων αλλά μειώνει κατά πολύ την κυκλοφορία στο δίκτυο.

Το Election Commission αποτελείται από μια ομάδα ειδικών διεργασιών του συστήματος και χειρίζεται την όλη διαδικασία εκλογής αρχηγού. Έχει ένα μοναδικό αναγνωριστικό μέσω του οποίου οι διεργασίες του συστήματος επικοινωνούν μαζί του και έτσι επιτυγχάνεται η μείωση των μηνυμάτων ‘Election’ μεταξύ των διεργασιών. Η αρχιτεκτονική του Election Commission φαίνεται στο Σχήμα 3.9.



Σχήμα 3.9 Αρχιτεκτονική Election Commission [9]

Αποτελείται από :

- Ένα Chief Election Commissioner (CEC), που είναι η διεργασία με το μεγαλύτερο αναγνωριστικό στο Election Commission.

- Τέσσερις Election Commissioners, όπου ο κάθε Election Commissioner είναι μια ιδιαίτερη διεργασία. Οι Election Commissioners εκτελούν εντολές από τον Chief Election Commissioner για την εκλογή νέου αρχηγού στο σύστημα.
- Ένα Helper (HP), το οποίο γνωρίζει τα αναγνωριστικά όλων των διεργασιών του συστήματος και εντοπίζει την ενεργή διεργασία με το μεγαλύτερο αναγνωριστικό.
- Ένα Failure Detector (FD), το οποίο αναγνωρίζει εάν μια διεργασία είναι η όχι ενεργή.

3.5.1 Περιγραφή Αλγορίθμου

Ο αλγόριθμος αυτός εκλέγει ως αρχηγό τη διεργασία με το μεγαλύτερο αναγνωριστικό. Κάθε διεργασία: έχει ένα μοναδικό αναγνωριστικό , γνωρίζει μόνο το αναγνωριστικό του Election Commission και της διεργασίας αρχηγού, είναι εναρκτής του αλγορίθμου και μπορεί να αποτυγχάνει και να επανέρχεται οποιαδήποτε στιγμή. Όπως και στον αλγόριθμο του Garcia Molina, οι διεργασίες δεν γνωρίζουν ποιες άλλες διεργασίες του συστήματος είναι ενεργές και ποιες όχι. Υποθέτουμε ότι στο σύστημα υπάρχουν n διεργασίες.

Στον συγκεκριμένο αλγόριθμο, υπάρχουν τα ακόλουθα είδη μηνυμάτων:

1. Μήνυμα ‘Election’, το οποίο στέλνεται από μία διεργασία η οποία αντιλαμβάνεται ότι ο αρχηγός δεν είναι ενεργός και απαιτείται η εκλογή νέου αρχηγού.
2. Μήνυμα ‘Reply Election’, το οποίο στέλνεται ως απάντηση στο μήνυμα ‘Election’.
3. Μήνυμα ‘Leader’, το οποίο στέλνεται από το Election Commission σε μια διεργασία για να την ενημερώσει ποια είναι η διεργασία που εκλέγεται ως αρχηγός.
4. Μήνυμα ‘Check Leader’, το οποίο στέλνεται από το Election Commission ή μια διεργασία για να ελέγξουν εάν η τρέχον διεργασία αρχηγός είναι ενεργή.
5. Μήνυμα ‘Reply Check Leader’, το οποίο στέλνεται από τη διεργασία αρχηγός ως απάντηση στο μήνυμα ‘Reply Check Leader’.
6. Μήνυμα ‘Query’, το οποίο στέλνεται από μια διεργασία όταν επανέλθει ξαφνικά στο σύστημα.
7. Μήνυμα ‘Check Process’, το οποίο στέλνεται από το Election Commission σε μια διεργασία για να ελέγξει εάν είναι ενεργή ή όχι.
8. Μήνυμα ‘Reply Check Process’, το οποίο στέλνεται από μια διεργασία ως απάντηση στο μήνυμα ‘Check Process’.

Η περιγραφή του τροποποιημένου αλγορίθμου του Garcia Molina που μελετάμε, φαίνεται στο Σχήμα 3.10. Όταν μία διεργασία p_i αντιληφθεί ότι ο αρχηγός δεν είναι ενεργός, δηλαδή όταν στείλει μήνυμα ‘Check Leader’ στη διεργασία αρχηγό και αυτή δεν απαντήσει σε

συγκεκριμένο χρονικό διάστημα, τότε στέλνει ‘Election’ μήνυμα στο Election Commission (EC). Ο failure detector (FD) του Election Commission επιβεβαιώνει εάν το μήνυμα της p_i είναι αληθές ή όχι. Εάν η διεργασία αρχηγός είναι ενεργή, τότε το Election Commission στέλνει μήνυμα ‘Leader’ στη διεργασία p_i ανακοινώνοντας της τη διεργασία αρχηγό του συστήματος. Εάν όμως η διεργασία αρχηγός δεν είναι ενεργή τότε ελέγχουμε το αναγνωριστικό της διεργασίας p_i. Αν είναι η διεργασία με το μεγαλύτερο αναγνωριστικό τότε εκλέγεται ως αρχηγός και το Election Commission στέλνει μήνυμα ‘Leader’ σε όλες τις διεργασίες του συστήματος με το αναγνωριστικό του νέου αρχηγού. Εάν όμως η διεργασία p_i δεν έχει το μεγαλύτερο αναγνωριστικό, τότε το Election Commission βρίσκει την ενεργή διεργασία με το μεγαλύτερο αναγνωριστικό, χρησιμοποιώντας τον helper (HP), η οποία εκλέγεται αρχηγός και ενημερώνονται όλες οι διεργασίες με ‘Leader’ μήνυμα. Στην περίπτωση όπου μια διεργασία p_i' που δεν ήταν ενεργή και ξαφνικά επανέλθει στο σύστημα, τότε στέλνει στο Election Commission μήνυμα ‘Query’ για να μάθει ποιος είναι αρχηγός του συστήματος. Το Election Commission όταν λάβει αυτό το μήνυμα, τότε ελέγχει εάν το αναγνωριστικό της p_i' είναι μεγαλύτερο από το αναγνωριστικό της διεργασίας αρχηγού. Εάν είναι αληθές, τότε η διεργασία p_i' γίνεται η νέα αρχηγός του συστήματος και ενημερώνονται όλες οι άλλες διεργασίες από το Election Commission για την εκλογή αυτή. Εάν πάλι δεν έχει αναγνωριστικό μεγαλύτερο από τον αρχηγό, το Election Commission στέλνει στην p_i' ‘Leader’ μήνυμα με το αναγνωριστικό της διεργασίας αρχηγού. Επίσης, να επισημάνουμε πως όταν το Election Commission λαμβάνει περισσότερα από ένα ‘Election’ μηνύματα την ίδια ώρα και επιβεβαιώνει ότι η διεργασία αρχηγός δεν είναι ενεργή, τότε λαμβάνει υπόψη το μήνυμα από τη διεργασία με το μεγαλύτερο αναγνωριστικό. Αυτό συμβαίνει για να μειωθεί ο αριθμός των μηνυμάτων που χρειάζονται για τον εντοπισμό της μεγαλύτερης ενεργής διεργασίας από το helper.

Αν μια διεργασία p_i είναι ενεργή
Στέλνει στη διεργασία αρχηγό μήνυμα ‘Check Leader’
Περιμένει για κάποιο συγκεκριμένο χρονικό διάστημα ένα μήνυμα ‘Reply Check Leader’ από τη διεργασία αρχηγό
Αν το χρονικό διάστημα εξαντληθεί και η p_i δεν λάβει κάποιο μήνυμα ‘Reply Check Leader’
Στέλνει στο EC μήνυμα ‘Election’
Αν η p_i λάβει κάποιο μήνυμα ‘Reply Election’
Περιμένει για κάποιο συγκεκριμένο χρονικό διάστημα να λάβει το αναγνωριστικό του αρχηγού

Αν το χρονικό διάστημα εξαντληθεί και δεν έχει λάβει το αναγνωριστικό

Ξαναστέλνει μήνυμα ‘Election’ στο EC

Αν η p_i λάβει κάποιο μήνυμα ‘Check Process Election’

Στέλνει στο EC μήνυμα ‘Reply Check process’

Αν μια διεργασία p_i μόλις έχει επανέλθει στο σύστημα

Στέλνει στο EC μήνυμα ‘Query’

Περιμένει για κάποιο συγκεκριμένο χρονικό διάστημα να λάβει το αναγνωριστικό του αρχηγού

Αν το χρονικό διάστημα εξαντληθεί και δεν έχει λάβει το αναγνωριστικό

Ξαναστέλνει μήνυμα ‘Query’ στο EC

Αν το EC λάβει κάποιο μήνυμα ‘Election’ από μια διεργασία p_i

Στέλνει στη διεργασία αρχηγό μήνυμα ‘Check Leader’

Περιμένει για κάποιο συγκεκριμένο χρονικό διάστημα ένα μήνυμα ‘Reply Check Leader’ από τη διεργασία αρχηγό

Αν το χρονικό διάστημα εξαντληθεί και το EC δεν λάβει κάποιο μήνυμα ‘Reply Check Leader’

Εντοπίζει την ενεργή διεργασία με το μεγαλύτερο αναγνωριστικό.

Αν η p_i είναι η διεργασία με το μεγαλύτερο αναγνωριστικό

Γνωστοποιεί το αναγνωριστικό της p_i σε όλες τις διεργασίες, στέλνοντάς τους μήνυμα Leader

Αν η p_i δεν είναι η διεργασία με το μεγαλύτερο αναγνωριστικό

Γνωστοποιεί το αναγνωριστικό της διεργασίας με το μεγαλύτερο αναγνωριστικό σε όλες τις διεργασίες, στέλνοντάς τους μήνυμα Leader

Αν το EC λάβει κάποιο μήνυμα ‘Query’ από μια διεργασία p_i

Αν η p_i έχει μεγαλύτερο αναγνωριστικό από την διεργασία αρχηγό

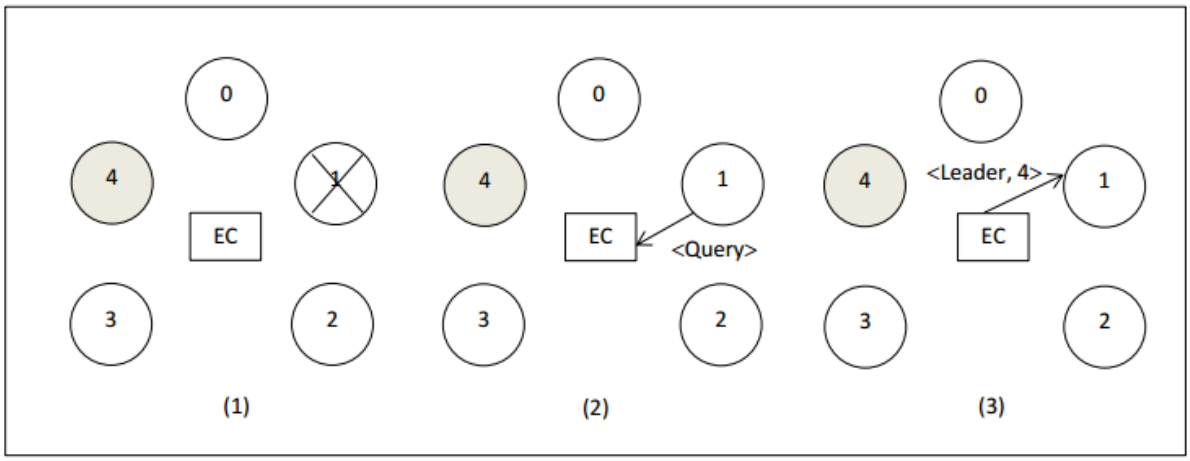
Γνωστοποιεί το αναγνωριστικό της p_i σε όλες τις διεργασίες, στέλνοντάς τους μήνυμα Leader

Αν η p_i δεν έχει μεγαλύτερο αναγνωριστικό από την διεργασία αρχηγό

Γνωστοποιεί το αναγνωριστικό της διεργασίας αρχηγού στη διεργασία p_i, στέλνοντάς της μήνυμα Leader

Σχήμα 3.10 Τροποποιημένος Αλγόριθμος με τοπολογία ισχυρά συνδεδεμένου γράφου

Στο Σχήμα 3.11 παρουσιάζεται ένα παράδειγμα εκτέλεσης του αλγορίθμου, όπου η διεργασία με αναγνωριστικό 1 δεν είναι ενεργή, ξαφνικά επανέρχεται στο σύστημα και στέλνει ‘Query’ μήνυμα στο Election Commission. Το Election Commission ελέγχει το αναγνωριστικό της διεργασίας 1 με το αναγνωριστικό της διεργασίας αρχηγού (διεργασία 4) και έτσι απαντά στη διεργασία 1 με μήνυμα ‘Leader’ και το αναγνωριστικό του αρχηγού.



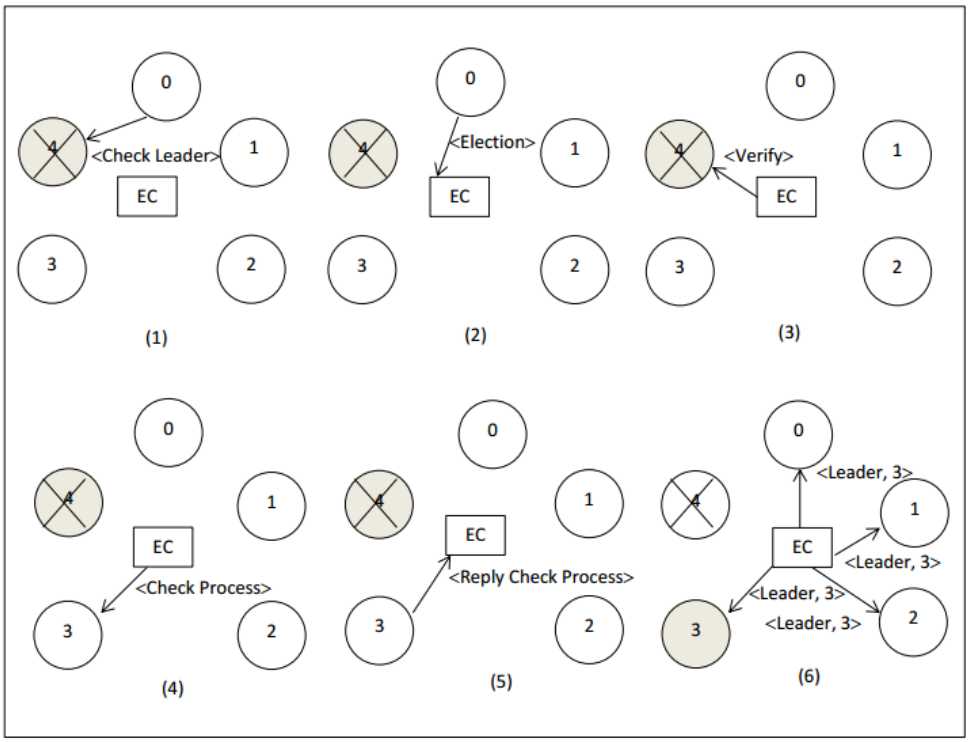
Σχήμα 3.11 Παράδειγμα Εκτέλεσης Τροποποιημένου Αλγορίθμου Garcia Molina

Πιο κάτω, διατυπώνονται δύο εκδοχές του τροποποιημένου αλγορίθμου, οι οποίες διαφέρουν μεταξύ τους στον τρόπο με τον οποίο το Election Commission εντοπίζει την ενεργή διεργασία με το μεγαλύτερο αναγνωριστικό, η οποία θα πρέπει να εκλεγεί ως νέος αρχηγός.

Πρώτη Εκδοχή Τροποποιημένου Αλγορίθμου

Στην πρώτη εκδοχή του τροποποιημένου αλγορίθμου Garcia Molina το Election Commission εντοπίζει την ενεργή διεργασία με το μεγαλύτερο αναγνωριστικό, που θα πρέπει να εκλεγεί ως νέος αρχηγός, στέλνοντας ‘Check Process’ μήνυμα στη διεργασία με το μεγαλύτερο αναγνωριστικό, έστω p, αφού η διεργασία αρχηγός δεν είναι ενεργή. Εάν η διεργασία αυτή είναι ενεργή και απαντήσει με ‘Reply Check Process’ μήνυμα στο Election Commission, τότε αυτή είναι η διεργασία που θα εκλεγεί ως νέος αρχηγός. Εάν όμως δεν απαντήσει, τότε η p δεν είναι ενεργή και θα επαναληφθεί η ίδια διαδικασία με τη μεγαλύτερη διεργασία του συστήματος που είναι μικρότερη από την p. Η διαδικασία αυτή συνεχίζεται μέχρι να εντοπιστεί η μεγαλύτερη ενεργή διεργασία. Στο Σχήμα 3.12 παρουσιάζεται εκτέλεση της πρώτης εκδοχής του τροποποιημένου αλγορίθμου Garcia Molina. Η διεργασία 0 αντιλαμβάνεται ότι η αρχηγός (διεργασία 4) δεν είναι ενεργή και στέλνει ‘Election’ μήνυμα

στο Election Commission. Εκείνο με τη σειρά του στέλνει μήνυμα ‘Verify’ στην διεργασία 4 για να ελέγξει εάν όντως δεν είναι ενεργή. Αφού περάσει κάποιο χρονικό διάστημα και δεν πάρει καμία απάντηση από την διεργασία 4, τότε στέλνει μήνυμα στην διεργασία 3, η οποία είναι η μεγαλύτερη διεργασία πριν την διεργασία 4 για να ελέγξει αν είναι ενεργή. Η διεργασία 3, είναι όντως ενεργή και απαντά στο Election Commission με ένα‘Reply Check Process’ μήνυμα. Τέλος, το Election Commission ανακηρύσσει την διεργασία 3 ως νέα αρχηγό του συστήματος και ενημερώνει τις υπόλοιπες διεργασίες με το κατάλληλο μήνυμα.

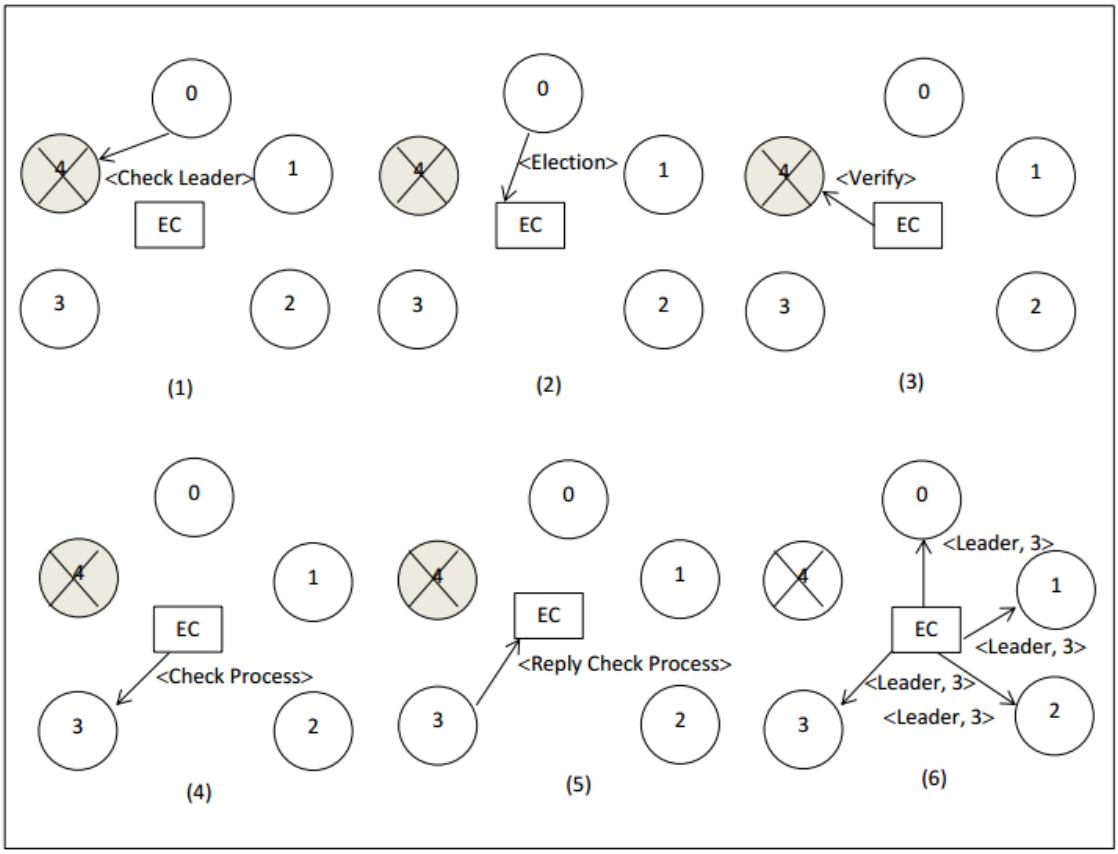


Σχήμα 3.12 Παράδειγμα Εκτέλεσης Πρώτης Εκδοχής Τροποποιημένου Αλγορίθμου Garcia Molina

Δεύτερη Εκδοχή Τροποποιημένου Αλγορίθμου

Στην δεύτερη εκδοχή του τροποποιημένου αλγορίθμου Garcia Molina το Election Commission εντοπίζει τη μεγαλύτερη ενεργή διεργασία που θα εκλεγεί ως νέος αρχηγός, στέλνοντας ‘Check Process’ μήνυμα σε όλες τις διεργασίες του συστήματος και θα περιμένει απάντηση από όλες σε συγκεκριμένο χρονικό διάστημα. Όταν λάβει απ’ όλες μήνυμα ‘Reply Check Process’ ή στην περίπτωση που εξαντληθεί το απαιτούμενο χρονικό διάστημα, ανακηρύσσει αρχηγό τη διεργασία με το μεγαλύτερο αναγνωριστικό από την οποία παίρνει απάντηση. Το Σχήμα 3.13 δείχνει την εκτέλεση της δεύτερη εκδοχής του συγκεκριμένου αλγορίθμου που μελετάμε. Η διεργασία 0 αντιλαμβάνεται ότι η αρχηγός (διεργασία 4) δεν

είναι ενεργή και στέλνει ‘Election’ μήνυμα στο Election Commission. Εκείνο με τη σειρά του στέλνει μήνυμα ‘Verify’ στην διεργασία 4 για να ελέγξει εάν όντως δεν είναι ενεργή. Αφού περάσει κάποιο χρονικό διάστημα και δεν πάρει καμία απάντηση από την διεργασία 4, τότε στέλνει ‘Check Process’ μήνυμα σε όλες τις διεργασίες και περιμένει και πάλι ένα χρονικό διάστημα μέχρι να πάλι απ’ όλες απάντηση. Αφού περάσει το συγκεκριμένο χρονικό διάστημα, αντιλαμβάνεται ότι η μεγαλύτερη ενεργή διεργασία είναι η 3 και ενημερώνει όλες τις διεργασίες του συστήματος για το νέο αρχηγό (διεργασία 3).



Σχήμα 3.13 Παράδειγμα Εκτέλεσης Δεύτερης Εκδοχής Τροποποιημένου Αλγορίθμου Garcia Molina

3.5.2 Ανάλυση Αλγορίθμου

Ο τροποποιημένος αλγόριθμος Garcia Molina, όπως έχει διατυπωθεί από τους Quazi Ehsanul Kabir Mamun, Salahuddin Mohammad Masum και Mohammad Abdur Rahim Mustafa είναι ορθός αφού σε οποιαδήποτε εκτέλεσή του, αρχηγός εκλέγεται η ενεργή διεργασία με το μεγαλύτερο αναγνωριστικό.

Κατά την εκτέλεση του αλγορίθμου, απαιτούνται το πολύ n βήματα για τον εντοπισμό της διεργασίας αρχηγού από το Election Commission και συνεπώς ο τροποποιημένος αλγόριθμος

Garcia Molina παρουσιάζει χρονική πολυπλοκότητα τάξεως O(n). Έχει αποδειχθεί ότι ο αριθμός των μηνυμάτων που στέλνονται κατά την εκτέλεση του συγκεκριμένου αλγορίθμου υπολογίζεται από τον τύπο 2(n-r) + n , όπου n ο αριθμός των διεργασιών στο σύστημα και r το αναγνωριστικό της διεργασίας που εκκινεί τη διαδικασία εκλογής αρχηγού. Η χειρότερη περίπτωση, παρατηρείται όταν η διεργασία με το μικρότερο αναγνωριστικό αντιληφθεί πως ο αρχηγός δεν είναι ενεργός και ξεκινάει εκλογή για νέο αρχηγό. Επομένως, η επικοινωνιακή πολυπλοκότητα του, είναι της τάξεως O(n), και είναι καλύτερος και πιο αποδοτικός από τον κανονικό αλγόριθμο Garcia Molina, στον οποίο ο αριθμός των μηνυμάτων είναι της τάξης $O(n^2)$. Αυτό εύκολα μπορεί να γίνει κατανοητό, εάν μελετήσουμε τις εξής συγκεκριμένες περιπτώσεις:

1. Έστω ότι στο σύστημα υπάρχουν n διεργασίες, οι οποίες είναι όλες ενεργές. Για κάποιο λόγο η διεργασία αρχηγός καθυστερεί να απαντήσει στη μικρότερη ενεργή διεργασία που της έχει στείλει 'Check Leader' μήνυμα. Τότε η διεργασία αντιλαμβάνεται λανθασμένα πως η διεργασία αρχηγός έχει καταρρεύσει. Σύμφωνα με τον κανονικό αλγόριθμο θα στείλει σε όλες τις μεγαλύτερες διεργασίες από αυτήν 'Election'μήνυμα και παρομοίως θα ενεργήσουν όλες οι διεργασίες του συστήματος. Επομένως, θα στείλει (n-1) 'Election' μηνύματα μόνο για να ενημερώσει τις άλλες διεργασίας πως ο αρχηγός δεν είναι ενεργός και πρέπει να γίνει νέα εκλογή. Ενώ, στον τροποποιημένο αλγόριθμο θα στείλει μόνο ένα 'Election' μήνυμα και αυτό προς το Election Commission.
2. Εάν μια διεργασία με αναγνωριστικό μικρότερο από την διεργασία αρχηγό ξαφνικά επανέλθει στο σύστημα, τότε σύμφωνα με τον αλγόριθμο Garcia Molina, θα στείλει 'Election' μήνυμα σε όλες τις διεργασίες με μεγαλύτερο αναγνωριστικό από αυτήν. Αντίθετα, με τον τροποποιημένο αλγόριθμο θα στείλει μόνο ένα 'Query'μήνυμα στο Election Commission, όπου και θα την ενημερώσει το αναγνωριστικό του αρχηγού.

Στον αλγόριθμο αυτό, βασική προϋπόθεση είναι ότι το Election Commission ποτέ δεν αποτυγχάνει και σε περίπτωση που συμβεί, επανέρχεται αμέσως. Αυτό όμως δεν αποκλείει την περίπτωση αποτυχίας του από το σύστημα.

Κεφάλαιο 4

Προδιαγραφή Αλγορίθμων Εκλογής Αρχηγού

Στο Κεφάλαιο αυτό θα γίνει εκτενής περιγραφή των προδιαγραφών των αλγορίθμων Εκλογής Αρχηγού που υλοποιήθηκαν στο εργαλείο Tempo, χρησιμοποιώντας Χρονισμένα Αυτόματα Εισόδου / Εξόδου.

4.1 Προδιαγραφή Αλγορίθμου με Τοπολογία Δέντρου

Η ολοκληρωμένη προδιαγραφή του αλγορίθμου με τοπολογία δέντρου στη γλώσσα TIOA και στο εργαλείο Tempo, όπως αυτός έχει περιγραφεί στο προηγούμενο κεφάλαιο, βρίσκεται στο Παράρτημα Β.1.

Για την υλοποίηση του αλγορίθμου, χρειάζεται να υλοποιήσουμε το λεξιλόγιο που θα χρησιμοποιείται από τα αυτόματα του αλγορίθμου, το αυτόματο το οποίο θα υλοποιεί την κάθε διεργασία, τα αυτόματα που υλοποιούν το κανάλι επικοινωνίας μεταξύ των διεργασιών και το αυτόματο σύνθεσης. Η επικοινωνία μεταξύ των διεργασιών για το συγκεκριμένο αλγόριθμο θα υλοποιηθεί με το κανάλι επικοινωνίας MPI, τις προδιαγραφές του οποίου παραθέτουμε στο Παράρτημα Α.

Το λεξιλόγιο για τον συγκεκριμένο αλγόριθμο φαίνεται στο Σχήμα 4.1 και περιλαμβάνει το λεξιλόγιο που χρησιμοποιούν τα αυτόματα διεργασίες και το λεξιλόγιο που χρειάζεται το κανάλι επικοινωνίας MPI. Ορίζουμε μία πλειάδα Message, η οποία αντιπροσωπεύει ένα τμήμα του μηνύματος που στέλλει η κάθε διεργασία και αποτελείται από δύο φυσικούς αριθμούς οι οποίοι αντιστοιχούν στον τύπο του μηνύματος (type) και στην ταυτότητα της διεργασίας αρχηγού (id). Το τμήμα αυτό μαζί με δύο φυσικούς αριθμούς που δηλώνουν τον αποστολέα και τον παραλήπτη του μηνύματος, ορίζουν τον τύπο του μηνύματος (mpi_message) με το οποίο επικοινωνούν οι διεργασίες μεταξύ τους.

```
%%% :Algorithm vocabs

vocabulary algorithm_voc
   types
      Message : Tuple[type:Nat, id : Nat]
end


%%% :MPI Channel vocabs

vocabulary mpi_status_voc
   types mpi_status
   operators
      MPI_Iprobe : Nat -> Null[mpi_status],
      MPI_Test : mpi_status ->Bool
end

vocabulary mpi_message_voc
   imports algorithm_voc, mpi_status_voc
      types mpi_message : Tuple[data:Message,sender : Nat, destination:Nat]
      operators
         MPI_Irecv: mpi_status, Nat -> mpi_message
end

vocabulary mpi_request_voc
   imports mpi_message_voc
      types mpi_request
      operators
         MPI_Isend: mpi_message, Nat -> Null[mpi_request],
         MPI_Barrier : -> Bool
end
```

Σχήμα 4.1 Λεξιλόγιο αλγορίθμου με τοπολογία δέντρου

Στο Παράρτημα Β1.2 βρίσκεται η προδιαγραφή του αυτομάτου που αντιπροσωπεύει μία διεργασία του αλγορίθμου με τοπολογία δέντρου στο εργαλείο Tempo. Το αυτόματο αυτό, ονομάζεται Algorithm1_Process και παίρνει ως παραμέτρους : το χρονικό διάστημα t1 στο οποίο κάθε διεργασία θα πρέπει να στείλει και να λάβει κάποιο μήνυμα και τον αριθμό που αντιστοιχεί στην ταυτότητα της διεργασίας.

Στο Σχήμα 4.2 παρουσιάζονται οι ενέργειες που μπορεί να εκτελέσει μια διεργασία και που δηλώνονται στο τμήμα signature του αυτομάτου.

```
signature
   input RECEIVE (m:Null[mpi_message])
   output SEND (m:Null[mpi_message])
   internal processTimeout, init, enableCondition
```

Σχήμα 4.2 Δήλωση ενεργειών μιας διεργασίας του αλγορίθμου με τοπολογία δέντρου

Στο Σχήμα 4.3 παρουσιάζονται οι μεταβλητές καταστάσεων του αυτομάτου. Ορίζονται στο τμήμα states και είναι οι ακόλουθες:

- clock : Αντιπροσωπεύει το φυσικό ρολόι της διεργασίας, είναι τύπου DiscreteReal και αρχικοποιείται με 0.
- processTime : Αντιπροσωπεύει το φυσικό ρολόι κάθε διεργασίας, το οποίο υπολογίζει το χρόνο από τη στιγμή που θα στείλει κάποιο μήνυμα μέχρι να λάβει μήνυμα από κάποια γειτονική της διεργασία. Είναι τύπου DiscreteReal και αρχικοποιείται με 0.
- msgs : Είναι μία κενή ακολουθία μηνυμάτων mpi_message της κάθε διεργασίας, στην οποία εισέρχονται τα μηνύματα που πρόκειται να στείλει και εξέρχονται τα μηνύματα που στέλνονται επιτυχώς.
- NbrsNodes : Είναι μια κενή ακολουθία φυσικών αριθμών που αντιπροσωπεύει τις ταυτότητες των γειτονικών διεργασιών της κάθε διεργασίας.
- Nodes : Είναι μια κενή ακολουθία φυσικών αριθμών που αντιπροσωπεύει τις ταυτότητες των γειτονικών διεργασιών της κάθε διεργασίας (όπως και η NbrsNodes), από την οποία όμως θα αφαιρούνται οι ταυτότητες των διεργασιών από τις οποίες η διεργασία έχει λάβει μήνυμα. Η ακολουθία αυτή σε κάθε κύκλο θα δείχνει από πόσες και ποιες διεργασίες περιμένει να λάβει μήνυμα η διεργασία.
- minimunId : Είναι η φυσική μεταβλητή που αντιπροσωπεύει την ταυτότητα της μικρότερης διεργασίας. Αρχικοποιείται με την ταυτότητα της διεργασίας.
- receiveMessagesCounter : Είναι ένας μετρητής ο οποίος υπολογίζει πόσα μηνύματα λαμβάνει η διεργασία. Είναι ακέραιος αριθμός και αρχικοποιείται με 0.
- isLeader : Είναι η boolean μεταβλητή που καθορίζει εάν η διεργασία είναι η μικρότερη διεργασία του συστήματος, και επομένως η διεργασία αρχηγός. Αρχικοποιείται με false.

- processRank : Είναι η φυσική μεταβλητή που αντιπροσωπεύει την ταυτότητα της κάθε διεργασίας. Αρχικοποιείται με την ταυτότητα της διεργασίας που δίνεται ως παράμετρο στο αυτόματο.
- TokenMessage : Αντιπροσωπεύει μήνυμα τύπου 'token', δηλαδή μήνυμα από μια διεργασία προς τη διεργασία αρχηγό.
- LeaderMessage : Αντιπροσωπεύει μήνυμα τύπου 'leader', δηλαδή μήνυμα από τη διεργασία αρχηγό προς μια απλή διεργασία. Με το μήνυμα αυτό, μια διεργασία ενημερώνεται για την ταυτότητα της διεργασίας αρχηγού.

```
states
       % The physcal clock of the automaton
       clock : DiscreteReal :=0;
       % The physical clock for the process
       processTime : DiscreteReal :=0;
       % An empty sequence of mpi messages
       msgs : Seq[Null[mpi_message]] := {};
       % Sequence of all the neighboor processes of this process
       NbrsNodes : Seq[Nat] :={};
       Nodes : Seq[Nat] :={};
       % The minimun id from all the processes
       minimunId : Nat :=rank ;
       % A counter for the messages that the process has received
       receiveMessagesCounter : Int :=0;
       % Indicates that the process is leader
       isLeader : Bool := false;
       % Indicates the process rank
       processRank : Nat := rank;
       % Indicates token message
       TokenMessage : Nat := 1;
       % Indicates leader message
       LeaderMessage:Nat:=2;
```

Σχήμα 4.3 Μεταβλητές Καταστάσεως μιας διεργασίας του αλγορίθμου με τοπολογία δέντρου

Οι ενέργειες της διεργασίας που έχουν δηλωθεί στο τμήμα signature, υλοποιούνται στο τμήμα transitions (Σχήμα 4.4). Αναλυτικά:

- Με την ενέργεια εισόδου RECEIVE, η διεργασία λαμβάνει ένα μήνυμα m, τύπου mpi_message. Εάν το μήνυμα δεν είναι κενό, θέτει στο ρολόι processTime τη συγκεκριμένη χρονική στιγμή που λαμβάνει το μήνυμα. Εάν ο τύπος του μηνύματος είναι TokenMessage, τότε η διεργασία λαμβάνει μήνυμα από κάποια γειτονική της διεργασία και αυξάνει το μετρητή μηνυμάτων που λαμβάνει. Εάν το μήνυμα που λαμβάνει προέρχεται από διεργασία με μικρότερο αναγνωριστικό από το minimunId, τότε το minimunId γίνεται ίσο με το αναγνωριστικό της που περιέχεται στο μήνυμα που λαμβάνει. Στη συνέχεια, εάν υπάρχουν κι' άλλες διεργασίες στην ακολουθία

Nodes, δηλαδή περιμένει να λάβει μηνύματα και από άλλες γειτονικές διεργασίες, αφαιρεί από την ακολουθία αυτή, το αναγνωριστικό της διεργασίας από την οποία μόλις έχει λάβει μήνυμα. Εάν έχει λάβει μηνύματα από όλους τους γείτονές της εκτός από έναν, δηλαδή ο μετρητής receiveMessagesCounter είναι κατά ένα λιγότερος από τον αριθμό των γειτόνων της διεργασίας, τότε στέλνει Token μήνυμα στη γειτονική της διεργασία από την οποία δεν έλαβε ως τώρα μήνυμα, με το αναγνωριστικό της μικρότερης διεργασίας που έχει εντοπίσει (minimunId). Το αναγνωριστικό της γειτονικής διεργασίας από την οποία δεν έχει λάβει μήνυμα, είναι το μοναδικό που βρίσκεται στην ακολουθία Nodes. Όταν λάβει Token μήνυμα και από αυτήν τη διεργασία, τότε συγκρίνει το αναγνωριστικό της με το μικρότερο αναγνωριστικό που έχει υπολογίσει. Εάν είναι τα ίδια, τότε η διεργασία έχει το μικρότερο αναγνωριστικό στο σύστημα και είναι αυτή που θα γίνει η διεργασία αρχηγός. Τότε, στέλνει σε όλες τις υπόλοιπες γειτονικές τις διεργασίες Leader μήνυμα με το αναγνωριστικό της.

Εάν η διεργασία λαμβάνει μήνυμα τύπου LeaderMessage, τότε λαμβάνει μήνυμα από τη διεργασία αρχηγό όπου της ανακοινώνει το αναγνωριστικό της. Έτσι θέτει την τιμή της μεταβλητής minimunId ίσο με το αναγνωριστικό που λαμβάνει (id). Η εκτέλεση της ενέργειας αυτής, πυροδοτεί όλες τις αντίστοιχες ενέργειες εξόδου RECEIVE.

- Με την ενέργεια εξόδου SEND, η διεργασία στέλλει ένα μήνυμα m. Προϋποθέσεις για να εκτελεστεί η ενέργεια αυτή είναι το μήνυμα που θα σταλεί να βρίσκεται πρώτο στην ακολουθία μηνυμάτων msgs και η ακολουθία μηνυμάτων να μη είναι άδεια. Αποτέλεσμα της ενέργειας αυτής είναι να μηδενιστεί το φυσικό ρολόι της διεργασίας που την εκτελεί (processTime), έτσι ώστε να ξεκινάει να μετράει το χρόνο μέχρι να λάβει κάποιο μήνυμα. Η εκτέλεση της ενέργειας αυτής, πυροδοτεί όλες τις αντίστοιχες ενέργειες εισόδου SEND.
- Με την εσωτερική ενέργεια processTimeout, εάν μια διεργασία δεν λάβει κάποιο μήνυμα σε χρονικό διάστημα t1 από τη στιγμή που στείλει κάποιο μήνυμα, τότε μηδενίζεται το φυσικό ρολόι της διεργασίας (processTime) και ξεκινάει ξανά να μετράει ο χρόνος που πρέπει να λάβει κάποιο μήνυμα.
- Με την εσωτερική ενέργεια init, για κάθε διεργασία αρχικοποιούνται οι ακολουθίες με τα αναγνωριστικά των γειτονικών της διεργασιών (Nodes και NbrsNodes). Με την ενέργεια αυτή αρχικοποιείται η δομή του δέντρου που θέλουμε να μελετήσουμε. Στην συγκεκριμένη υλοποίηση, υποθέτουμε πως υπάρχουν 4 διεργασίες και η μορφή του δέντρου είναι η εξής : ρίζα είναι η διεργασία με αναγνωριστικό 0, παιδιά της είναι οι διεργασίες με αναγνωριστικά 1 και 3 και η διεργασία 2 είναι παιδί της διεργασίας 3.

Επομένως, η διεργασία 0 έχει γείτονες τις διεργασίες 1 και 3, η διεργασία 1 έχει γείτονα τη διεργασία 0, η διεργασία 2 έχει γείτονα τη διεργασία 3 και τέλος η διεργασία 3 έχει γείτονες τις διεργασίες 0 και 2.

- Με την εσωτερική ενέργεια enableCondition, εάν μια διεργασία έχει λάβει μηνύματα από όλους τους γείτονές της εκτός από έναν, τότε θα στείλει στη γειτονική της διεργασία από την οποία δεν έχει λάβει μήνυμα, ένα Token μήνυμα με το μικρότερο αναγνωριστικό που έχει υπολογίσει μέχρι εκείνη τη στιγμή. Προϋποθέσεις για την εκτέλεση της συγκεκριμένης ενέργειας είναι ο μετρητής receiveMessagesCounter να ισούται με τον αριθμό των γειτόνων της διεργασίας μείον ένα και στην ακολουθία Nodes να υπάρχει μόνο ένα αναγνωριστικό, που αντιπροσωπεύει το αναγνωριστικό της γειτονικής διεργασίας από την οποία η διεργασία δεν έχει λάβει μήνυμα.

```
transitions
      input RECEIVE(m)
            locals positionOfSender:Nat :=0;
            eff
               if(m ~= nil()) then
                  processTime := clock;

                  if(val(m).data.type = TokenMessage)then
                     receiveMessagesCounter := receiveMessagesCounter +1;

                     if(minimunId > val(m).data.id) then
                        minimunId := val(m).data.id;
                     fi;

                     if(len(Nodes)>1)then
                        positionOfSender :=0;
                        for n:Nat where n < (len(Nodes)) do
                           if(Nodes[n] = (val(m).sender))then
                              positionOfSender:=n;
                           fi;
                        od;
                        Nodes := eject(Nodes,positionOfSender);
                     fi;

                     if(receiveMessagesCounter = (len(NbrsNodes)-1) )then
                        msgs := msgs |- embed([[TokenMessage,
      minimunId],rank,Nodes[0]]);
                     fi;

                     if(receiveMessagesCounter = len(NbrsNodes) /\ val(m).sender
      = Nodes[0]) then
                        if(minimunId = rank)then
                           isLeader := true;
                        fi;
                        for n:Nat where n<len(NbrsNodes) do
                           if(NbrsNodes[n] ~= Nodes[0]) then
                              msgs := msgs |- embed([[LeaderMessage,
      minimunId],rank,NbrsNodes[n]]);
                           fi;
                        od;
```

```
                    fi;
                fi;
                if(val(m).data.type = LeaderMessage)then
                    minimunId := val(m).data.id;
                fi;
            fi;


    output SEND(m)
        pre
            msgs ~= {};
            m = head(msgs);




        eff
            processTime := 0;
            msgs := tail(msgs);


    internal processTimeout
        pre
            processTime >= t1;
        eff
            processTime := 0;


    internal init
        eff

            if (rank = 0)then
                NbrsNodes:= NbrsNodes |- 1;
                NbrsNodes := set(NbrsNodes,0,1);
                NbrsNodes:= NbrsNodes |- 3;
                NbrsNodes := set(NbrsNodes,1,3);
            fi;
            if (rank = 1)then
                NbrsNodes:= NbrsNodes |- 0;
                NbrsNodes := set(NbrsNodes,0,0);
            fi;
            if (rank = 2)then
                NbrsNodes:= NbrsNodes |- 3;
                NbrsNodes := set(NbrsNodes,0,3);
            fi;
            if (rank = 3)then
                NbrsNodes:= NbrsNodes |- 0;
                NbrsNodes := set(NbrsNodes,0,0);
                NbrsNodes:= NbrsNodes |- 2;
                NbrsNodes := set(NbrsNodes,1,2);
            fi;
            Nodes := NbrsNodes;

    internal enableCondition
        pre
            receiveMessagesCounter = len(NbrsNodes) - 1;
            len(Nodes) = 1;
        eff
            msgs := msgs |- embed([[TokenMessage,minimunId],rank,Nodes[0]]);
```

Σχήμα 4.4 Ορισμός ενεργειών μιας διεργασίας του αλγορίθμου με τοπολογία δέντρου

Αφού έχουν οριστεί οι ενέργειες του αυτομάτου, μετά το τμήμα transitions ακολουθεί ο ορισμός της τροχιάς Time στο τμήμα trajectories, η οποία εξελίσσει το φυσικό ρολόι clock με ρυθμό 1, και παρουσιάζεται στο Σχήμα 4.5.

```
trajectories
	trajdef time
		evolve d(clock)=1;
```

Σχήμα 4.5 Ορισμός τροχιάς μιας διεργασίας του αλγορίθμου με τοπολογία δέντρου

Εφόσον έχουμε υλοποιήσει το αυτόματο διεργασίας και τα αυτόματα που υλοποιούν τα κανάλια επικοινωνίας είναι ήδη υλοποιημένα, μπορούμε να υλοποιήσουμε το αυτόματο σύνθεσης (Algorithm1), το οποίο βρίσκεται στο Παράρτημα Β1.3. Ξεκινώντας συμπεριλαμβάνουμε τα αρχεία στα οποία υλοποιούνται : το λεξιλόγιο του αλγορίθμου, το αυτόματο διεργασίας και τα αυτόματα του καναλιού επικοινωνίας MPI (Σχήμα 4.6).

```
%%% .: Vocabulary :.
include "Vocabulary.tioa"
imports algorithm_voc, mpi_status_voc, mpi_message_voc, mpi_request_voc,
mpi_voc

%%% .: MPI Mediator Automata :.
include "SendMediator.tioa"
include "ReceiveMediator.tioa"

%%% .: Algorithm Automata :.
include "Algorithm1_Process.tioa"
```

Σχήμα 4.6 Εισαγωγή Λεξιλογίων και Αυτομάτων στο Αυτόματο Σύνθεσης για τον αλγόριθμο με τοπολογία δέντρου

Το αυτόματο σύνθεσης, όπως φαίνεται στο Σχήμα 4.7 παίρνει ως παράμετρο το χρονικό διάστημα t1 στο οποίο κάθε διεργασία θα πρέπει να λάβει μήνυμα από τη στιγμή που στείλει κάποιο μήνυμα. Το αυτόματο σύνθεσης αποτελείται από: ένα αυτόματο διεργασίας (P), ένα αυτόματο SendMediator (SM) και ένα ReceiveMediator (RM). Στο αυτόματο διεργασίας P, δίδεται ως παράμετρος η παράμετρος του αυτομάτου σύνθεσης καθώς επίσης και η ταυτότητα της διεργασίας μέσω της συνάρτησης MPI_Rank().

```
automaton Algorithm1(t1: DiscreteReal)
components
	P: Algorithm1_Process(t1,MPI_Rank());
	SM: SendMediator;
	RM: ReceiveMediator;
```

Σχήμα 4.7 Αυτόματο Σύνθεσης για τον αλγόριθμο με τοπολογία δέντρου

Οι μεταβλητές του αυτομάτου σύνθεσης, στο τμήμα states, παρουσιάζονται στο επόμενο Σχήμα (Σχήμα 4.8) και είναι οι ακόλουθες :

- m : Δηλώνει ένα μήνυμα mpi_message, το οποίο αρχικοποιείται με nil().
- runs : Αντιπροσωπεύει τον αριθμό των επαναλήψεων του αλγόριθμου που θα εκτελεστούν στο πρόγραμμα μεταβάσεων και τροχιών. Αρχικοποιείται με MPI_Size που αντιστοιχεί στον αριθμό των διεργασιών του συστήματος και είναι ο μέγιστος αριθμός γύρων που χρειάζεται ο αλγόριθμος με τοπολογία δέντρου για την εκλογή αρχηγού.

```
states
      m:Null[mpi_message] :=nil();
      runs : Int :=MPI_Size();
```

Σχήμα 4.8 Μεταβλητές του αυτομάτου σύνθεσης στον αλγόριθμο με τοπολογία δέντρου

Το πρόγραμμα μεταβάσεων και τροχιών (schedule) αρχίζει με την πυροδότηση της εσωτερικής ενέργειας init για την αρχικοποίηση της τοπολογίας δέντρου του αλγορίθμου και συγκεκριμένα των γειτονικών διεργασιών κάθε διεργασίας. Ακολουθεί ένας βρόγχος στον οποίον: εξελίσσεται η τροχιά του αυτομάτου διεργασίας με ρυθμό 1, πυροδοτούνται οι εσωτερικές ενέργειες enableCondition και processTimeout οι οποίες ελέγχουν για την ικανοποίηση της συνθήκης του αλγορίθμου (μια διεργασία λάβει μηνύματα από όλους τους γείτονες της εκτός από έναν) και για τυχόν καθυστερήσεις στην παραλαβή και αποστολή των μηνυμάτων. Επίσης, στέλνονται τα μηνύματα που βρίσκονται στο κανάλι και έχουν αποστολέα τη συγκεκριμένη διεργασία της οποίας εκτελείται το πρόγραμμα και ακολούθως λαμβάνονται τυχόν μηνύματα που βρίσκονται στο κανάλι και έχουν ως παραλήπτη τη συγκεκριμένη διεργασία. Η εξέλιξη της τροχιάς και οι πυροδοτήσεις των ενεργειών γίνονται με τις εντολές follow και fire, αντίστοιχα. Το Σχήμα 4.9 δείχνει τη ροή εκτέλεσης του προγράμματος μεταβάσεων και τροχιών που περιγράψαμε πιο πάνω για τον αλγόριθμο με τοπολογία δέντρου.

```
do
      fire internal P.init;


      for i:Nat where i<runs do
            follow P.time duration 1;

            fire internal P.enableCondition;
            fire internal P.processTimeout;
```

```
            %% Send Messages
            for j:Nat where j<  MPI_Size() do
                  fire output P.SEND(m);
            od

            follow SM.DELAY duration (MPI_Size() * 10);

            %% Receive Messages
            for j:Nat where j<MPI_Size() do
                  m:=nil();
                  fire input RM.probe(j);
                  fire output RM.RECEIVE(m);
            od
      od
od
```

Σχήμα 4.9 Πρόγραμμα μεταβάσεων και τροχιών αυτομάτου σύνθεσης στον αλγόριθμο με τοπολογία δέντρου

4.2 Προδιαγραφή Αλγορίθμου με τοπολογία δακτυλίου

Η ολοκληρωμένη προδιαγραφή του αλγορίθμου με τοπολογία δακτυλίου στη γλώσσα TIOA και στο εργαλείο Tempo, όπως αυτός έχει περιγραφεί στο προηγούμενο κεφάλαιο, βρίσκεται στο Παράρτημα Β.2

Για την υλοποίηση του αλγορίθμου, χρειάζεται να υλοποιήσουμε: το λεξιλόγιο που θα χρησιμοποιείται από τα αυτόματα του αλγορίθμου, το αυτόματο το οποίο θα υλοποιεί την κάθε διεργασία, τα αυτόματα που υλοποιούν το κανάλι επικοινωνίας μεταξύ των διεργασιών και το αυτόματο σύνθεσης. Η επικοινωνία μεταξύ των διεργασιών για το συγκεκριμένο αλγόριθμο θα υλοποιηθεί με το κανάλι επικοινωνίας MPI, τις προδιαγραφές του οποίου παραθέτουμε στο Παράρτημα Α.

Το λεξιλόγιο για το συγκεκριμένο αλγόριθμο φαίνεται στο Σχήμα 4.10 και περιλαμβάνει το λεξιλόγιο που χρησιμοποιούν τα αυτόματα διεργασίες και το λεξιλόγιο που χρειάζεται το κανάλι επικοινωνίας MPI. Ορίζουμε μία πλειάδα Message, η οποία αντιπροσωπεύει ένα τμήμα του μηνύματος που στέλλει η κάθε διεργασία και αποτελείται από δύο φυσικούς αριθμούς οι οποίοι αντιστοιχούν στον τύπο του μηνύματος (type) και στην ταυτότητα της διεργασίας αρχηγού (id). Το τμήμα αυτό μαζί με δύο φυσικούς αριθμούς που δηλώνουν τον αποστολέα και τον παραλήπτη του μηνύματος, ορίζουν τον τύπο του μηνύματος (mpi_message) με το οποίο επικοινωνούν οι διεργασίες μεταξύ τους.

```
%%% :Algorithm vocabs

vocabulary algorithm_voc
```

```
types
            Message : Tuple[type:Nat, id : Nat]
end

%%% :MPI Channel vocabs

vocabulary mpi_status_voc
      types mpi_status
      operators
            MPI_Iprobe : Nat -> Null[mpi_status],
            MPI_Test : mpi_status ->Bool
end

vocabulary mpi_message_voc
      imports algorithm_voc, mpi_status_voc
            types mpi_message : Tuple[data:Message, destination:Nat]
            operators
                  MPI_Irecv: mpi_status, Nat -> mpi_message
end

vocabulary mpi_request_voc
      imports mpi_message_voc
            types mpi_request
            operators
                  MPI_Isend: mpi_message, Nat -> Null[mpi_request],
                  MPI_Barrier : -> Bool
end

vocabulary mpi_voc
      operators
            MPI_Rank : -> Nat,
            MPI_Size : -> Nat
end
```

Σχήμα 4.10 Λεξιλόγιο αλγορίθμου με τοπολογία δακτυλίου

Στο Παράρτημα Β2.2 βρίσκεται η προδιαγραφή του αυτομάτου που αντιπροσωπεύει μία διεργασία του αλγορίθμου με τοπολογία δακτυλίου στο εργαλείο Tempo. Το αυτόματο αυτό, ονομάζεται Algorithm2_Process και παίρνει ως παραμέτρους : το χρονικό διάστημα t1 στο οποίο κάθε διεργασία θα πρέπει να στείλει και να λάβει κάποιο μήνυμα και τον αριθμό που αντιστοιχεί στην ταυτότητα της διεργασίας.

Στο Σχήμα 4.11 παρουσιάζονται οι ενέργειες που μπορεί να εκτελέσει μια διεργασία και που δηλώνονται στο τμήμα signature του αυτομάτου.

```
signature
            input RECEIVE (m:Null[mpi_message])
            output SEND (m:Null[mpi_message])
            internal processTimeout, init, prepMessages
```

Σχήμα 4.11 Δήλωση ενεργειών μιας διεργασίας του αλγορίθμου με τοπολογία δακτυλίου

Στο Σχήμα 4.12 παρουσιάζονται οι μεταβλητές καταστάσεων του αυτομάτου. Ορίζονται στο τμήμα states και είναι οι ακόλουθες:

- clock : Αντιπροσωπεύει το φυσικό ρολόι της διεργασίας, είναι τύπου DiscreteReal και αρχικοποιείται με 0.
- processTime : Αντιπροσωπεύει το φυσικό ρολόι κάθε διεργασίας, το οποίο υπολογίζει το χρόνο από τη στιγμή που θα στείλει κάποιο μήνυμα μέχρι να λάβει μήνυμα από κάποια γειτονική της διεργασία. Είναι τύπου DiscreteReal και αρχικοποιείται με 0.
- msgs : Είναι μία κενή ακολουθία μηνυμάτων mpi_message της κάθε διεργασίας, στην οποία εισέρχονται τα μηνύματα που πρόκειται να στείλει και εξέρχονται τα μηνύματα που στέλνονται επιτυχώς.
- nextNode : Είναι η μεταβλητή που αντιπροσωπεύει το αναγνωριστικό της επόμενης διεργασίας της συγκεκριμένης διεργασίας. Είναι ένας φυσικός αριθμός και αρχικοποιείται με μηδέν. Θα πάρει τιμή με την εκτέλεση της ενέργειας init όπου θα καθοριστεί η θέση της κάθε διεργασίας στο δακτύλιο.
- sendFlag : Είναι boolean μεταβλητή που καθορίζει εάν η διεργασία έχει στείλει κάποιο μήνυμα. Αρχικοποιείται με false.
- isLeader : Είναι η boolean μεταβλητή που καθορίζει εάν η διεργασία είναι η μικρότερη διεργασία του συστήματος, και επομένως η διεργασία αρχηγός. Αρχικοποιείται με false.
- minimunId : Είναι η φυσική μεταβλητή που αντιπροσωπεύει την ταυτότητα της μικρότερης διεργασίας. Αρχικοποιείται με την ταυτότητα της διεργασίας.
- processRank : Είναι η φυσική μεταβλητή που αντιπροσωπεύει την ταυτότητα της κάθε διεργασίας. Αρχικοποιείται με την ταυτότητα της διεργασίας που δίνεται ως παράμετρο στο αυτόματο.
- TokenMessage : Αντιπροσωπεύει μήνυμα τύπου ‘token’, δηλαδή μήνυμα από μια διεργασία προς τη διεργασία αρχηγό.
- LeaderMessage : Αντιπροσωπεύει μήνυμα τύπου ‘leader’, δηλαδή μήνυμα από τη διεργασία αρχηγό προς μια απλή διεργασία. Με το μήνυμα αυτό, μια διεργασία ενημερώνεται για την ταυτότητα της διεργασίας αρχηγού.

```
states
      % The physcal clock of the automaton
      clock : DiscreteReal :=0;
      % The physical clock for the process
      processTime : DiscreteReal :=0;
      % An empty sequence of mpi messages
      msgs : Seq[Null[mpi_message]] := {};
```

```
% Indicates the next node of the process
nextNode : Nat := 0;
% Indicates that the process has sent a message
sendFlag : Bool := false;
% Indicates that the process is leader
isLeader : Bool := false;
% The minimun id from all the processes
minimunId : Nat :=rank ;
% Indicates the rank of this process
processRank : Nat := rank;
%Indicates a token message
TokenMessage : Nat := 1;
% Message which indicates that a leader has found
LeaderMessage : Nat :=2;
```

Σχήμα 4.12 Μεταβλητές Καταστάσεως μιας διεργασίας του αλγορίθμου με τοπολογία δακτυλίου

Οι ενέργειες της διεργασίας που έχουν δηλωθεί στο τμήμα signature, υλοποιούνται στο τμήμα transitions (Σχήμα 4.13). Αναλυτικά:

- Με την ενέργεια εισόδου RECEIVE, η διεργασία λαμβάνει ένα μήνυμα m, τύπου mpi_message. Εάν το μήνυμα δεν είναι κενό, θέτει στο ρολόι processTime τη συγκεκριμένη χρονική στιγμή που λαμβάνει το μήνυμα. Εάν ο τύπος του μηνύματος είναι TokenMessage, τότε η διεργασία λαμβάνει μήνυμα από την προηγούμενή της διεργασία στο δακτύλιο με κάποιο αναγνωριστικό. Εάν το αναγνωριστικό που λαμβάνει είναι μικρότερο από το μικρότερο αναγνωριστικό που γνωρίζει η διεργασία (minimunId), τότε το minimunId γίνεται το αναγνωριστικό που στέλνει η διεργασία που αποστέλλει το μήνυμα. Επίσης, εάν το αναγνωριστικό που λαμβάνει (id) είναι ίδιο με το δικό της αναγνωριστικό, τότε η διεργασία έχει το μικρότερο αναγνωριστικό στο σύστημα και είναι η διεργασία αρχηγός. Γι ‘αυτό, προωθεί Leader μήνυμα με το μικρότερο αναγνωριστικό, που είναι το δικό της στην επόμενη της διεργασία. Εάν ο τύπος του μηνύματος είναι LeaderMessage και η διεργασία δεν είναι η διεργασία αρχηγός, τότε ενημερώνεται από την προηγούμενή της διεργασία για το αναγνωριστικό της διεργασίας αρχηγού. Έτσι θέτει την τιμή της μεταβλητής minimunId ίσο με το αναγνωριστικό που λαμβάνει (id) και προωθεί στην επόμενή της διεργασία μήνυμα Leader με αναγνωριστικό το minimunId, για να της ανακοινώσει τον αρχηγό. Η εκτέλεση της ενέργειας αυτής, πυροδοτεί όλες τις αντίστοιχες ενέργειες εξόδου RECEIVE.
- Με την ενέργεια εξόδου SEND, η διεργασία στέλλει ένα μήνυμα m. Προϋποθέσεις για να εκτελεστεί η ενέργεια αυτή είναι το μήνυμα που θα σταλεί να βρίσκεται πρώτο

στην ακολουθία μηνυμάτων msgs και η ακολουθία μηνυμάτων να μη είναι άδεια. Αποτέλεσμα της ενέργειας αυτής είναι να μηδενιστεί το φυσικό ρολόι της διεργασίας που την εκτελεί (processTime), έτσι ώστε να ξεκινάει να μετράει το χρόνο μέχρι να λάβει κάποιο μήνυμα. Η εκτέλεση της ενέργειας αυτής, πυροδοτεί όλες τις αντίστοιχες ενέργειες εισόδου SEND.

- Με την εσωτερική ενέργεια init, για κάθε διεργασία αρχικοποιείται η επόμενή της διεργασία στο δακτύλιο (nextNode), δηλαδή το αναγνωριστικό της μοναδικής διεργασίας στην οποία μπορεί να στείλει μηνύματα. Με την ενέργεια αυτή, αρχικοποιείται η δομή του δακτυλίου που θέλουμε να μελετήσουμε. Στην συγκεκριμένη υλοποίηση, υποθέτουμε πως υπάρχουν 4 διεργασίες και η σειρά με την οποία εμφανίζονται στον δακτύλιο, με δεξιόστροφή κατεύθυνση είναι η εξής : διεργασία 0 , διεργασία 2, διεργασία 1 και διεργασία 3. Επομένως, η διεργασία 0 στέλνει μηνύματα στη διεργασία 2, η διεργασία 2 στέλνει μηνύματα στην διεργασία 1, η διεργασία 1 στη διεργασία 3 και τέλος η διεργασία 3 στην διεργασία 0.
- Με την εσωτερική ενέργεια prepMessages, ξεκινάει η εκτέλεση του αλγορίθμου με τοπολογία δακτυλίου, όπου κάθε διεργασία στέλνει Token μήνυμα με το αναγνωριστικό της στην επόμενή της διεργασία.
- Με την εσωτερική ενέργεια processTimeout, εάν μια διεργασία δεν λάβει κάποιο μήνυμα σε χρονικό διάστημα t1 από τη στιγμή που στείλει κάποιο μήνυμα, τότε μηδενίζεται το φυσικό ρολόι της διεργασίας (processTime) και ξεκινάει ξανά να μετράει ο χρόνος μέχρι να λάβει κάποιο μήνυμα.

```
transitions
      input RECEIVE(m)
         eff
            if(m ~= nil())then
                processTime := clock;
                if(val(m).data.type = TokenMessage)then
                    if(val(m).data.id < minimunId)then
                       minimunId := val(m).data.id;
                       msgs := msgs |- embed([[TokenMessage,minimunId],nextNode]);
                    fi;
                    if(val(m).data.id = rank)then
                       isLeader := true;
                       msgs := msgs |- embed([[LeaderMessage,minimunId],nextNode]);
                    fi;
                fi;

                if(val(m).data.type = LeaderMessage)then
                   minimunId := val(m).data.id;
                   if(rank ~= minimunId)then
                       msgs := msgs |- embed([[LeaderMessage,minimunId],nextNode]);
                    fi;
                fi;
            fi;
```

```
output SEND(m)
   pre
      msgs ~= {};
       m = head(msgs);
   eff
      processTime := 0;
       msgs :=tail(msgs);

internal init
   eff
      if(rank = 0)then
          nextNode := 2;
       fi;
       if(rank = 1)then
          nextNode := 3;
       fi;
       if(rank = 2)then
          nextNode := 1;
       fi;
       if(rank = 3)then
          nextNode := 0;
       fi;

internal prepMessages
       pre
         len(msgs) = 0;
       eff
          msgs := msgs |- embed([[TokenMessage,rank],nextNode]);


internal processTimeout
       pre
          processTime >= t1;
       eff
          processTime :=0;
```

Σχήμα 4.13 Ορισμός ενεργειών μιας διεργασίας του αλγορίθμου με τοπολογία δακτυλίου

Αφού έχουν οριστεί οι ενέργειες του αυτομάτου, μετά το τμήμα transitions ακολουθεί ο ορισμός της τροχιάς Time στο τμήμα trajectories, η οποία εξελίσσει το φυσικό ρολόι clock με ρυθμό 1, και παρουσιάζεται στο Σχήμα 4.14.

```
trajectories
      trajdef time
            evolve d(clock)=1;
```

Σχήμα 4.14 Ορισμός τροχιάς μιας διεργασίας του αλγορίθμου με τοπολογία δακτυλίου

Εφόσον έχουμε υλοποιήσει το αυτόματο διεργασίας και τα αυτόματα που υλοποιούν τα κανάλια επικοινωνίας είναι ήδη υλοποιημένα, μπορούμε να υλοποιήσουμε το αυτόματο σύνθεσης (Algorithm2), το οποίο βρίσκεται στο Παράρτημα Β2.3. Ξεκινώντας

συμπεριλαμβάνουμε τα αρχεία στα οποία υλοποιούνται : το λεξιλόγιο του αλγορίθμου, το αυτόματο διεργασίας και τα αυτόματα του καναλιού επικοινωνίας MPI (Σχήμα 4.15).

```
%%% .: Vocabulary :.
include "Vocabulary.tioa"
imports algorithm_voc, mpi_status_voc, mpi_message_voc, mpi_request_voc,
mpi_voc

%%% .: MPI Mediator Automata :.
include "SendMediator.tioa"
include "ReceiveMediator.tioa"

%%% .: Algorithm Automata :.
include "Algorithm2_Process.tioa"
```

Σχήμα 4.15 Εισαγωγή Λεξιλογίων και Αυτομάτων στο Αυτόματο Σύνθεσης για τον αλγόριθμο με τοπολογία δακτυλίου

Το αυτόματο σύνθεσης, όπως φαίνεται στο Σχήμα 4.16 παίρνει ως παράμετρο το χρονικό διάστημα t1 στο οποίο κάθε διεργασία θα πρέπει να λάβει μήνυμα από τη στιγμή που στείλει κάποιο μήνυμα. Το αυτόματο σύνθεσης αποτελείται από: ένα αυτόματο διεργασίας (P), ένα αυτόματο SendMediator (SM) και ένα ReceiveMediator (RM). Στο αυτόματο διεργασίας P, δίδεται ως παράμετρος η παράμετρος του αυτομάτου σύνθεσης καθώς επίσης και η ταυτότητα της διεργασίας μέσω της συνάρτησης MPI_Rank().

```
automaton Algorithm2(t1: DiscreteReal)
components
       P: Algorithm2_Process(t1,MPI_Rank());
       SM: SendMediator;
       RM: ReceiveMediator;
```

Σχήμα 4.16 Αυτόματο Σύνθεσης για τον αλγόριθμο με τοπολογία δακτυλίου

Οι μεταβλητές του αυτομάτου σύνθεσης, στο τμήμα states, παρουσιάζονται στο επόμενο Σχήμα (Σχήμα 4.17) και είναι οι ακόλουθες :

- m : Δηλώνει ένα μήνυμα mpi_message, το οποίο αρχικοποιείται με nil().
- runs : Αντιπροσωπεύει τον αριθμό των επαναλήψεων του αλγόριθμου που θα εκτελεστούν στο πρόγραμμα μεταβάσεων και τροχιών. Αρχικοποιείται με $(MPI\_Size)^2$ που είναι ο μέγιστος αριθμός γύρων που χρειάζεται ο αλγόριθμος με τοπολογία δακτυλίου για την εκλογή αρχηγού.

```
states
      m:Null[mpi_message] :=nil();
      runs : Nat :=MPI_Size()**2;
```

Σχήμα 4.17 Μεταβλητές του αυτομάτου σύνθεσης στον αλγόριθμο με τοπολογία δακτυλίου

Το πρόγραμμα μεταβάσεων και τροχιών (schedule) αρχίζει με την πυροδότηση της εσωτερικής ενέργειας init για την αρχικοποίηση της τοπολογίας δακτυλίου του αλγορίθμου και συγκεκριμένα το αναγνωριστικό της επόμενης διεργασίας της κάθε διεργασίας. Ακολουθεί ένας βρόγχος στον οποίον: εξελίσσεται η τροχιά του αυτομάτου διεργασίας με ρυθμό 1, πυροδοτούνται οι εσωτερικές ενέργειες prepMessages και processTimeout οι οποίες εκκινούν τον αλγόριθμο, στέλνοντας κάθε διεργασία μήνυμα Token με το αναγνωριστικό της στην επόμενή της διεργασίας και ελέγχουν για τυχόν καθυστερήσεις στην παραλαβή και αποστολή των μηνυμάτων. Επίσης, στέλνονται τα μηνύματα που βρίσκονται στο κανάλι και έχουν αποστολέα τη συγκεκριμένη διεργασία της οποίας εκτελείται το πρόγραμμα και ακολούθως λαμβάνονται τυχόν μηνύματα που βρίσκονται στο κανάλι και έχουν ως παραλήπτη τη συγκεκριμένη διεργασία. Η εξέλιξη της τροχιάς και οι πυροδοτήσεις των ενεργειών γίνονται με τις εντολές follow και fire, αντίστοιχα. Το Σχήμα 4.18 δείχνει τη ροή εκτέλεσης του προγράμματος μεταβάσεων και τροχιών που περιγράψαμε πιο πάνω για τον αλγόριθμο με τοπολογία δακτυλίου.

```
do
      fire internal P.init;

      for i:Nat where i<runs do
            follow P.time duration 1;

            fire internal P.prepMessages;
            fire internal P.processTimeout;

            % Send Messages
            for j:Nat where j<  MPI_Size() do
                  fire output P.SEND(m);
            od

            follow SM.DELAY duration (MPI_Size() * 10);

            % Receive Messages
            for j:Nat where j<MPI_Size() do
                  fire input RM.probe(j);
                  fire output RM.RECEIVE(m);
            od

      od
 od
```

Σχήμα 4.18 Πρόγραμμα μεταβάσεων και τροχιών αυτομάτου σύνθεσης στον αλγόριθμο με τοπολογία δακτυλίου

4.3 Προδιαγραφή Αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου

Η ολοκληρωμένη προδιαγραφή του αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου στη γλώσσα TIOA και στο εργαλείο Tempo, όπως αυτός έχει περιγραφεί στο προηγούμενο κεφάλαιο, βρίσκεται στο Παράρτημα Β.3.
Για την υλοποίηση του αλγορίθμου, χρειάζεται να υλοποιήσουμε: το λεξιλόγιο που θα χρησιμοποιείται από τα αυτόματα του αλγορίθμου, το αυτόματο το οποίο θα υλοποιεί την κάθε διεργασία, τα αυτόματα που υλοποιούν το κανάλι επικοινωνίας μεταξύ των διεργασιών και το αυτόματο σύνθεσης. Η επικοινωνία μεταξύ των διεργασιών για το συγκεκριμένο αλγόριθμο θα υλοποιηθεί και με το κανάλι επικοινωνίας MPI και με το κανάλι επικοινωνίας TCP, τις προδιαγραφές των οποίων παραθέτουμε στο Παράρτημα Α.

4.3.1 Προδιαγραφή Αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου με το κανάλι επικοινωνίας MPI

Το λεξιλόγιο για τον συγκεκριμένο αλγόριθμο φαίνεται στο Σχήμα 4.19 και περιλαμβάνει το λεξιλόγιο που χρησιμοποιούν τα αυτόματα διεργασίες και το λεξιλόγιο που χρειάζεται το κανάλι επικοινωνίας MPI. Ορίζουμε ένα φυσικό αριθμό για την μεταβλητή messageType, η οποία αντιπροσωπεύει ένα τμήμα του μηνύματος που στέλλει η κάθε διεργασία και που αντιστοιχεί στον τύπο του μηνύματος. Το τμήμα αυτό μαζί με δύο φυσικούς αριθμούς που δηλώνουν τον αποστολέα και τον παραλήπτη του μηνύματος, ορίζουν τον τύπο του μηνύματος (mpi_message) με το οποίο επικοινωνούν οι διεργασίες μεταξύ τους.

```
%%% :Algorithm vocabs

vocabulary algorithm_voc
       types
              messageType : Nat
end


%%% :MPI Channel vocabs

vocabulary mpi_status_voc
       types mpi_status
       operators
              MPI_Iprobe : Nat -> Null[mpi_status],
              MPI_Test : mpi_status ->Bool
end

vocabulary mpi_message_voc
       imports algorithm_voc, mpi_status_voc
       types mpi_message : Tuple[message:messageType, sender:Nat,
destination:Nat]
```

```
        operators
                MPI_Irecv: mpi_status, Nat -> mpi_message
        end

vocabulary mpi_request_voc
        imports mpi_message_voc
                types mpi_request
                operators
                        MPI_Isend: mpi_message, Nat -> Null[mpi_request],
                        MPI_Barrier : -> Bool
end

vocabulary mpi_voc
        operators
                MPI_Rank : -> Nat,
                MPI_Size : -> Nat
end
```

Σχήμα 4.19 Λεξιλόγιο αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένου με το κανάλι επικοινωνίας MPI

Στο Παράρτημα B3.1.2 βρίσκεται η προδιαγραφή του αυτομάτου που αντιπροσωπεύει μία διεργασία του αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένου με το κανάλι επικοινωνίας MPI στο εργαλείο Tempo. Το αυτόματο αυτό, ονομάζεται Algorithm3_Process και παίρνει ως παραμέτρους : το χρονικό διάστημα timeForElection στο οποίο θα πρέπει να ολοκληρωθεί η διαδικασία εκλογής αρχηγού , το χρονικό διάστημα timeForAnnouncementLeader στο οποίο θα πρέπει να ανακοινωθεί το αναγνωριστικό της διεργασίας αρχηγού στη διεργασία που ξεκινάει τη διαδικασία εκλογής αρχηγού, το χρονικό διάστημα timeForCheckleader στο οποίο μια διεργασία θα ελέγχει εάν η διεργασία αρχηγός είναι ζωντανή, την πιθανότητα possibilityToFail που αντιστοιχεί στην πιθανότητα μια διεργασία να καταρρεύσει, την πιθανότητα possibilityToRecover που αντιστοιχεί στην πιθανότητα μια διεργασία που έχει καταρρεύσει να επανέλθει και τον αριθμό που αντιστοιχεί στην ταυτότητα της διεργασίας.

Στο Σχήμα 4.20 παρουσιάζονται οι ενέργειες που μπορεί να εκτελέσει μια διεργασία και που δηλώνονται στο τμήμα signature του αυτομάτου.

```
signature
      input RECEIVE (m:Null[mpi_message])
      output SEND (m:Null[mpi_message])
      internal init,checkLeader
      internal chooseToFail, chooseToRecover
      internal checkLeaderTimeout, electionTimeout, announcementLeaderTimeout
```

Σχήμα 4.20 Δήλωση ενεργειών μιας διεργασίας του αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένο με το κανάλι επικοινωνίας MPI

Στο Σχήμα 4.21 παρουσιάζονται οι μεταβλητές καταστάσεων του αυτομάτου. Ορίζονται στο τμήμα states και είναι οι ακόλουθες:

- clock : Αντιπροσωπεύει το φυσικό ρολόι της διεργασίας, είναι τύπου DiscreteReal και αρχικοποιείται με 0.
- checkLeaderTimer : Αντιπροσωπεύει το φυσικό ρολόι κάθε διεργασίας, το οποίο υπολογίζει το χρονικό διάστημα στο οποίο η διεργασία θα ελέγχει εάν η διεργασία αρχηγός είναι ζωντανή
- electionTimer : Αντιπροσωπεύει το φυσικό ρολόι κάθε διεργασίας, το οποίο υπολογίζει το χρονικό διάστημα στο οποίο η διαδικασία εκλογής αρχηγού θα πρέπει να ολοκληρωθεί
- announcementLeaderTimer : Αντιπροσωπεύει το φυσικό ρολόι κάθε διεργασίας, το οποίο υπολογίζει το χρονικό διάστημα στο οποίο θα πρέπει να ανακοινωθεί στη διεργασία το αναγνωριστικό της διεργασίας αρχηγού
- msgs : Είναι μία κενή ακολουθία μηνυμάτων mpi_message της κάθε διεργασίας, στην οποία εισέρχονται τα μηνύματα που πρόκειται να στείλει και εξέρχονται τα μηνύματα που στέλνονται επιτυχώς.
- messagesCounter : Είναι ένας μετρητής ο οποίος υπολογίζει πόσα μηνύματα λαμβάνει η διεργασία. Είναι ακέραιος αριθμός και αρχικοποιείται με 0.
- processIsActive : Είναι η boolean μεταβλητή που καθορίζει εάν η διεργασία είναι ενεργή ή όχι. Αρχικοποιείται με false.
- processRank : Είναι η φυσική μεταβλητή που αντιπροσωπεύει την ταυτότητα της κάθε διεργασίας. Αρχικοποιείται με την ταυτότητα της διεργασίας που δίνεται ως παράμετρο στο αυτόματο.
- leaderId : Είναι η φυσική μεταβλητή που αντιπροσωπεύει την ταυτότητα της διεργασίας αρχηγού. Αρχικοποιείται με την ταυτότητα της διεργασίας.
- status : Είναι μία πλειάδα, η οποία αντιπροσωπεύει την κατάσταση στην οποία βρίσκεται μια διεργασία και αποτελείται από τρεις boolean μεταβλητές που αντιπροσωπεύουν τις εξής τρεις καταστάσεις : την κατάσταση στην οποία είναι ενημερωμένη για την διεργασία αρχηγό του συστήματος (thereIsLeader), την κατάσταση στην οποία εκτελείται η διαδικασία εκλογής αρχηγού (election) και τη διαδικασία στην οποία περιμένει να της ανακοινωθεί το αναγνωριστικό της διεργασίας αρχηγού (waitForLeaderAnnouncement). Και οι τρεις καταστάσεις αρχικοποιούνται με false και μόνο μία από τις καταστάσεις κάθε φορά θα γίνεται true.

- ElectionMessage : Αντιπροσωπεύει μήνυμα τύπου ‘Election’. Το στέλνει μια διεργασία για την εκκίνηση της διαδικασίας εκλογής αρχηγού, όταν αντιληφθεί πως η διεργασία αρχηγός δεν είναι ζωντανή.
- ReplyElectionMessage : Αντιπροσωπεύει μήνυμα τύπου ‘Reply Election’. Το στέλνει μια διεργασία ως απάντηση προς τη διεργασία που της στέλνει μήνυμα ‘Election’.
- AnnouncementLeaderMessage : Αντιπροσωπεύει μήνυμα τύπου ‘Announcement Leader’. Το στέλνει η διεργασία αρχηγός προς μιαν άλλη διεργασία για να της ανακοινώσει την εκλογή της ως αρχηγός.
- CheckLeaderMessage : Αντιπροσωπεύει μήνυμα τύπου ‘Check Leader’. Το στέλνει μια διεργασία προς τη διεργασία αρχηγό για να ελέγξει εάν είναι ζωντανή.
- ReplyCheckLeaderMessage : Αντιπροσωπεύει μήνυμα τύπου ‘Reply Check leader’. Το στέλνει η διεργασία αρχηγός ως απάντηση προς τη διεργασία που τις στέλνει μήνυμα ‘Reply Check Leader’.

```
states
	% The physcal clock of the automaton
	clock : DiscreteReal :=0;
	% The physical clock to check leader process
	checkLeaderTimer : DiscreteReal :=0;
	% The physical clock to elect leader
	electionTimer : DiscreteReal :=0;
	% The physical clock to announce leader
	announcementLeaderTimer : DiscreteReal :=0;
	% An empty sequence of mpi messages
	msgs :Seq[Null[mpi_message]] :={};
	% A counter for the messages that a process has send
	messagesCounter : Int := 0;
	% A counter for the messages that sent for election
	messagesForElection : Int :=0;
	% Indicates that the process is active
	processIsActive : Bool :=true;
	% Indicates the rank of this process
	processRank : Nat := rank;
	% Indicates the id of the leader process
	leaderId : Nat:=0;
	% Tuple which indicates the possible status of the process
	status : Tuple[thereIsLeader:Bool, election:Bool,
waitForLeaderAnnouncement:Bool] :=[false,false,false];
	% Message to election a leader
	ElectionMessage : Nat :=1;
	% Message to reply in election message
	ReplyElectionMessage : Nat :=2;
	% Message to announce the leader
	AnnouncementLeaderMessage :Nat :=3;
	% Message to check the leader
```

```
CheckLeaderMessage : Nat :=4;
% Message to reply in check leader message
ReplyCheckLeaderMessage : Nat :=5;
```

Σχήμα 4.21 Μεταβλητές Καταστάσεως μιας διεργασίας του αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένο με το κανάλι επικοινωνίας MPI

Οι ενέργειες της διεργασίας που έχουν δηλωθεί στο τμήμα signature, υλοποιούνται στο τμήμα transitions (Σχήμα 4.22). Αναλυτικά:

- Με την ενέργεια εισόδου RECEIVE, η διεργασία λαμβάνει ένα μήνυμα m, τύπου mpi_message. Εάν το μήνυμα δεν είναι κενό και ο τύπος του μηνύματος είναι ElectionMessage, τότε η διεργασία λαμβάνει μήνυμα για την εκκίνηση της διαδικασίας εκλογής αρχηγού. Θέτει την κατάσταση της διεργασίας election με true και απαντά με ReplyElection μήνυμα στη διεργασία που της έχει στείλει το μήνυμα Election. Εάν η διεργασία έχει το μεγαλύτερο αναγνωριστικό απ' όλες τις διεργασίες, τότε είναι η διεργασία που θα εκλεγεί ως αρχηγός, θέτει την μεταβλητή της isLeader αληθές, την κατάστασή της election ψευδές ενώ την κατάσταση της thereIsLeader αληθές. Επίσης, στέλνει Announcement Leader σε όλες τις διεργασίες με μικρότερο αναγνωριστικό από αυτήν, για να τους ανακοινώσει την εκλογή του ως αρχηγός. Εάν η διεργασία λάβει μήνυμα τύπου ReplyElection, τότε παίρνει απάντηση από κάποια διεργασία στην οποία έχει στείλει Election μήνυμα που δηλώνει πως κάποια διεργασία με μεγαλύτερο αναγνωριστικό από αυτήν αναλαμβάνει τη διαδικασία εκλογής αρχηγού. Έτσι, θέτει την μεταβλητή electionTimer ίση με μηδέν, όλες τις καταστάσεις της με false εκτός από την κατάσταση waitForLeaderAnnouncement και ξεκινάει τον μετρητή announcement LeaderTimer περιμένοντας να του ανακοινωθεί το αναγνωριστικό της διεργασίας αρχηγού. Όταν η διεργασία λάβει AnnouncementLeader μήνυμα τότε της ανακοινώνεται από τη διεργασία αρχηγό το αναγνωριστικό της, το οποίο και αποθηκεύει στη μεταβλητή της leaderId. Επίσης θέτει την κατάστασή της thereIsLeader αληθές, ενώ όλες τις υπόλοιπες καταστάσεις της ψευδές και μηδενίζει τον μετρητή announcementLeaderTimer. Εάν μια διεργασία λάβει μήνυμα checkLeader, τότε δηλώνεται πως είναι η τρέχουσα διεργασία αρχηγός και η διεργασία που αποστέλλει το μήνυμα θέλει να ελέγξει εάν είναι ζωντανή. Έτσι, απαντά σ' αυτήν την διεργασία με μήνυμα ReplyCheckLeader. Τέλος, όταν φθάσει μήνυμα ReplyCheckLeader, σημαίνει πως παίρνει απάντηση από τη διεργασία αρχηγό και επομένως είναι ζωντανή. Δεν χρειάζεται να ξεκινήσει τη διαδικασία

εκλογής αρχηγού και μηδενίζει τον μετρητή checkLeaderTimer. Η εκτέλεση της ενέργειας αυτής, πυροδοτεί όλες τις αντίστοιχες ενέργειες εξόδου RECEIVE.

- Με την ενέργεια εξόδου SEND, η διεργασία στέλλει ένα μήνυμα m. Προϋποθέσεις για να εκτελεστεί η ενέργεια αυτή είναι το μήνυμα που θα σταλεί να βρίσκεται πρώτο στην ακολουθία μηνυμάτων msgs, η ακολουθία μηνυμάτων να μη είναι άδεια και η διεργασία που την εκτελεί να είναι ενεργή. Η ενεργοποίηση της ενέργειας αυτής, αυξάνεται ο αριθμός των μηνυμάτων που στέλνει η διεργασία. Εάν το μήνυμα που αποστέλλεται είναι τύπου ElectionMessage, τότε ξεκινάει ο μετρητής electionTimer, για το χρόνο που θα πρέπει να ολοκληρωθεί η διαδικασία εκλογής αρχηγού. Εάν το μήνυμα που αποστέλλεται είναι τύπου CheckLeader, τότε ξεκινάει ο μετρητής checkLeaderTimer για το χρόνο στον οποίο θα πρέπει να πάρει απάντηση από τη διεργασία αρχηγό έτσι ώστε να μην ξεκινήσει τη διαδικασία εκλογής αρχηγού. Η εκτέλεση της ενέργειας αυτής, πυροδοτεί όλες τις αντίστοιχες ενέργειες εισόδου SEND.
- Με την εσωτερική ενέργεια init, κάθε διεργασία αρχικοποιεί την μεταβλητή processIsActive με true, που δηλώνει πως είναι ενεργή, θέτει την κατάσταση thereIsLeader στην οποία βρίσκεται αληθές και αρχικοποιεί την μεταβλητή με το αναγνωριστικό της διεργασίας αρχηγού με το αναγνωριστικό της μεγαλύτερης διεργασίας του συστήματος.
- Με την εσωτερική ενέργεια checkLeader, μια διεργασία η οποία δεν είναι η διεργασία αρχηγός και γνωρίζει πως υπάρχει αρχηγός στο σύστημα, δηλαδή η κατάσταση thereIsLeader είναι αληθές, στέλνει μήνυμα CheckLeader στην διεργασία αρχηγό για να ελέγξει εάν όντως είναι ζωντανή.
- Με την εσωτερική ενέργεια chooseToFail, μια ενεργή διεργασία καταρρέει από το σύστημα, θέτοντας τη μεταβλητή processIsActive ίση με false, όταν η τυχαία μεταβλητή randomValueToFail είναι μικρότερη από την πιθανότητα κατάρρευσης μιας διεργασίας (possibilityToFail).
- Με την εσωτερική ενέργεια chooseToRecover, μια διεργασία μη ενεργή, επιστρέφει στο σύστημα, θέτοντας τη μεταβλητή processIsActive ίση με true, όταν η τυχαία μεταβλητή randomValueToFail είναι μικρότερη από την πιθανότητα επαναφοράς μιας διεργασίας στο σύστημα (possibilityToRecover). Όταν μια διεργασία θα επανέλθει στο σύστημα τότε ελέγχει το αναγνωριστικό της με τα αναγνωριστικό της διεργασίας αρχηγού. Εάν η διεργασία αρχηγός έχει μικρότερο αναγνωριστικό από αυτήν, τότε γίνεται η νέα διεργασία αρχηγός και ενημερώνει με μήνυμα Election όλες τις υπόλοιπες διεργασίες. Αλλιώς, στέλνει μήνυμα Election σε όλες τις διεργασίες με

μεγαλύτερο αναγνωριστικό από αυτήν, ξεκινώντας τη διαδικασία εκλογής νέου αρχηγού.

- Η εσωτερική ενέργεια checkLeaderTimeout, ενεργοποιείται όταν μια διεργασία ενώ γνωρίζει πως υπάρχει διεργασία αρχηγός (η κατάσταση thereIsleader είναι αληθές), δεν παίρνει απάντηση από αυτήν (ReplyCheckLeader μήνυμα) στο CheckLeader μήνυμα που της στέλνει. Στην περίπτωση αυτή, η διεργασία εάν έχει το μεγαλύτερο αναγνωριστικό στο σύστημα, τότε γίνεται η νέα αρχηγός και ενημερώνει όλες τις διεργασίες με μικρότερο αναγνωριστικό από αυτήν, για την εκλογή της. Εάν όμως δεν έχει το μεγαλύτερο αναγνωριστικό ξεκινάει τη διαδικασία εκλογής αρχηγού, στέλνοντας μηνύματα Election σε όλες τις διεργασίες με μεγαλύτερο αναγνωριστικό από αυτήν.
- Η εσωτερική ενέργεια electionTimeout, ενεργοποιείται όταν η διαδικασία εκλογής αρχηγού δεν ολοκληρώνεται στο χρονικό διάστημα timeForElection και όταν η διεργασία βρίσκεται στην κατάσταση election, δηλαδή είναι true και οι καταστάσεις thereIsLeader και waitForLeaderAnnouncement είναι false. Ως αποτέλεσμα της ενέργειας αυτής είναι η επανεκκίνηση της διαδικασίας εκλογής αρχηγού από την συγκεκριμένη διεργασία στέλνοντας μηνύματα Election στις διεργασίες με μεγαλύτερα αναγνωριστικά από αυτήν.
- Η εσωτερική ενέργεια announcementLeaderTimeout, ενεργοποιείται όταν η διεργασία ενώ βρίσκεται στην κατάσταση waitForLeaderAnnouncement δεν ενημερώνεται για το αναγνωριστικό της διεργασίας αρχηγού που εκλέγεται, στο χρονικό διάστημα timeAnouncementLeader που καθορίζεται. Στην περίπτωση αυτή, η διεργασία υποθέτει πως δεν είναι ζωντανή καμία διεργασία με μεγαλύτερο αναγνωριστικό από αυτήν και έτσι γίνεται αυτή η νέα αρχηγός ενημερώνοντας τις διεργασίες με μικρότερο αναγνωριστικό από αυτήν για την εκλογή της.

```
transitions

      input RECEIVE(m)
         locals iAmLeader : Bool := false;
         eff
            if(processIsActive)then
               if(m ~= nil())then

                  if(val(m).message = ElectionMessage)then
                     status := [false,true,false];
                     msgs := msgs |- embed([ReplyElectionMessage, rank,
val(m).sender]);
```

```
                    iAmLeader := true;
                    for n:Nat where n < MPI_Size()do
                           if(n > rank)then
                              msgs := msgs |- embed([ElectionMessage, rank,
n]);
                              iAmLeader := false;
                           fi;
                    od;
                    if(iAmLeader = true)then
                           leaderId := rank;
                           status := [true,false,false];
                           for n:Nat where n < MPI_Size() do
                              if(n < rank)then
                                 msgs := msgs |-
embed([AnnouncementLeaderMessage, rank, n]);
                              fi;
                           od;
                       fi;
                    fi;

                    if(val(m).message = ReplyElectionMessage)then
                       electionTimer := 0;
                       status := [false,false,true];
                       announcementLeaderTimer := clock;
                    fi;

                    if(val(m).message = AnnouncementLeaderMessage)then
                       announcementLeaderTimer := 0;
                       status := [true,false,false];
                       leaderId := val(m).sender;
                    fi;

                    if(val(m).message = CheckLeaderMessage)then
                       msgs := msgs |- embed([ReplyCheckLeaderMessage, rank,
val(m).sender]);
                    fi;


                    if(val(m).message = ReplyCheckLeaderMessage)then
                       checkLeaderTimer :=0;
                    fi;

                 fi;
              fi;


        output SEND (m)
           pre
              processIsActive;
              msgs ~= {};
              m = head(msgs);
           eff
              msgs := tail(msgs);
              messagesCounter := messagesCounter + 1;

              if(val(m).message = ElectionMessage)then
                 electionTimer :=clock;
              fi;
              if(val(m).message = CheckLeaderMessage)then
                 checkLeaderTimer := clock;
              fi;
```

```
      internal init
         eff
            processIsActive := true;
            status := [true,false,false];
             leaderId := rank;
            for n:Nat where n < MPI_Size()do
               if(n > leaderId)then
                  leaderId := n;
               fi;
            od;

      internal checkLeader
         pre
            processIsActive;
            leaderId ~= rank;
            status = [true,false,false];
         eff
            msgs := msgs |- embed([CheckLeaderMessage,rank,leaderId]);


      internal chooseToFail
         locals randomValueToFail : Int := choose n where (n>=1 /\ n <= 100);
         eff
            if(processIsActive)then
               if(randomValueToFail >= 0 /\ randomValueToFail <=
possibilityToFail)then
                  processIsActive := false;
               fi;
            fi;


      internal chooseToRecover
         locals randomValueToRecover : Int := choose n where (n>=1 /\ n <=
100);
         eff
            if(processIsActive = false)then
                if(randomValueToRecover >=0 /\  randomValueToRecover <=
      possibilityToRecover)then
                  processIsActive := true;
                  if(leaderId < rank)then
                     for n:Nat where n < MPI_Size()do
                        if(n ~= rank)then
                           msgs := msgs |-
embed([AnnouncementLeaderMessage,rank,n]);
                        fi;
                     od;
                  else
                     for n:Nat where n < MPI_Size()do
                        if(n > rank)then
                           msgs := msgs |- embed([ElectionMessage,rank,n]);
                        fi;
                     od;
                  fi;
               fi;

            fi;


      internal checkLeaderTimeout
         locals iAmLeader : Bool := false;
         pre
```

```
            clock > (checkLeaderTimer + timeForCheckLeader);
            processIsActive;
            status = [true,false,false];
        eff
            status := [false,true,false];

            iAmLeader := true;
            for n:Nat where n < MPI_Size()do
               if(n > rank)then
                  msgs := msgs |- embed([ElectionMessage, rank, n]);
                  iAmLeader := false;
               fi;
            od;

            if(iAmLeader = true)then
               leaderId := rank;
               status := [true,false,false];
               for n:Nat where n < MPI_Size() do
                  if(n ~= rank)then
                     msgs := msgs |- embed([AnnouncementLeaderMessage, rank,
n]);
                  fi;
               od;
            fi;
            checkLeaderTimer := 0;


      internal electionTimeout
         pre
            clock > (electionTimer + timeForElection);
            processIsActive;
            status = [false,true,false];
         eff
            for n:Nat where n < MPI_Size() do

               if(n > rank)then
                  msgs := msgs |- embed([ElectionMessage, rank, n]);
               fi;
            od;
            electionTimer := 0;


      internal announcementLeaderTimeout
         pre
            clock > (announcementLeaderTimer + timeForAnouncementLeader);
             processIsActive;
            status = [false,false,true];
         eff
            status := [true,false,false];
            leaderId :=rank;
            for n:Nat where n < MPI_Size() do
               if(n < rank)then
                  msgs := msgs |- embed([AnnouncementLeaderMessage, rank,
n]);
               fi;
            od;
            announcementLeaderTimer :=0;
```

Σχήμα 4.22 Ορισμός ενεργειών μιας διεργασίας του αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένου με κανάλι επικοινωνίας MPI

Αφού έχουν οριστεί οι ενέργειες του αυτομάτου, μετά το τμήμα transitions ακολουθεί ο ορισμός της τροχιάς Time στο τμήμα trajectories, η οποία εξελίσσει το φυσικό ρολόι clock με ρυθμό 1, και παρουσιάζεται στο Σχήμα 4.23.

```
trajectories
	trajdef time
		evolve d(clock) = 1;
```

Σχήμα 4.23 Ορισμός τροχιάς μιας διεργασίας του αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένου με κανάλι επικοινωνίας MPI

Εφόσον έχουμε υλοποιήσει το αυτόματο διεργασίας και τα αυτόματα που υλοποιούν τα κανάλια επικοινωνίας είναι ήδη υλοποιημένα, μπορούμε να υλοποιήσουμε το αυτόματο σύνθεσης (Algorithm3), το οποίο βρίσκετα στο Παράρτημα Β3.1.3. Ξεκινώντας συμπεριλαμβάνουμε τα αρχεία στα οποία υλοποιούνται : το λεξιλόγιο του αλγορίθμου, το αυτόματο διεργασίας και τα αυτόματα του καναλιού επικοινωνίας MPI (Σχήμα 4.24).

```
%%% .: Vocabulary :.
include "Vocabulary.tioa"
imports algorithm_voc, mpi_status_voc, mpi_message_voc, mpi_request_voc,
mpi_voc

%%% .: MPI Mediator Automata :.
include "SendMediator.tioa"
include "ReceiveMediator.tioa"

%%% .: Algorithm Automata :.
include "Algorithm3_Process.tioa"
```

Σχήμα 4.24 Εισαγωγή Λεξιλογίων και Αυτομάτων στο Αυτόματο Σύνθεσης για τον αλγόριθμο με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένο με το κανάλι επικοινωνίας MPI

Το αυτόματο σύνθεσης, όπως φαίνεται στο Σχήμα 4.25 παίρνει ως παραμέτρους : το χρονικό διάστημα timeForElection στο οποίο θα πρέπει να ολοκληρωθεί η διαδικασία εκλογής αρχηγού , το χρονικό διάστημα timeForAnnouncementLeader στο οποίο θα πρέπει να ανακοινωθεί το αναγνωριστικό της διεργασίας αρχηγού στη διεργασία που ξεκινάει τη διαδικασία εκλογής αρχηγού, το χρονικό διάστημα timeForCheckleader στο οποίο μια διεργασία θα ελέγχει εάν η διεργασία αρχηγός είναι ζωντανή, την πιθανότητα possibilityToFail που αντιστοιχεί στην πιθανότητα μια διεργασία να καταρρεύσει και την πιθανότητα possibilityToRecover που αντιστοιχεί στην πιθανότητα μια διεργασία που έχει

καταρρεύσει να επανέλθει. Το αυτόματο σύνθεσης αποτελείται από: ένα αυτόματο διεργασίας (P), ένα αυτόματο SendMediator (SM) και ένα ReceiveMediator (RM). Στο αυτόματο διεργασίας P, δίδονται ως παράμετροι οι παράμετροι του αυτομάτου σύνθεσης καθώς επίσης και η ταυτότητα της διεργασίας μέσω της συνάρτησης MPI_Rank().

```
automaton Algorithm3(timeForElection:DiscreteReal,
timeForAnouncementLeader:DiscreteReal,timeForCheckLeader:DiscreteReal,
              possibilityToFail: Int, possibilityToRecover : Int)
components
       P: Algorithm3_Process(timeForElection,timeForAnouncementLeader,
timeForCheckLeader, possibilityToFail, possibilityToRecover,
                     MPI_Rank());
       SM: SendMediator;
       RM: ReceiveMediator;
```

Σχήμα 4.25 Αυτόματο Σύνθεσης για τον αλγόριθμο με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένο με το κανάλι επικοινωνίας MPI

Οι μεταβλητές του αυτομάτου σύνθεσης, στο τμήμα states, παρουσιάζονται στο επόμενο Σχήμα (Σχήμα 4.26) και είναι οι ακόλουθες :

- m : Δηλώνει ένα μήνυμα mpi_message, το οποίο αρχικοποιείται με nil().
- runs : Αντιπροσωπεύει τον αριθμό των επαναλήψεων του αλγόριθμου που θα εκτελεστούν στο πρόγραμμα μεταβάσεων και τροχιών. Αρχικοποιείται με $(MPI\_Size)^2$ που αντιστοιχεί στον μέγιστο αριθμό γύρων που χρειάζεται ο αλγόριθμος με τοπολογία ισχυρά συνδεδεμένου γράφου για την εκλογή αρχηγού.

```
states
       m:Null[mpi_message] :=nil();
       runs : Nat := MPI_Size()**2;
```

Σχήμα 4.26 Μεταβλητές του αυτομάτου σύνθεσης στον αλγόριθμο με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένο με το κανάλι επικοινωνίας MPI

Το πρόγραμμα μεταβάσεων και τροχιών (schedule) αρχίζει με την πυροδότηση της εσωτερικής ενέργειας init για την αρχικοποίηση από κάθε διεργασία το αναγνωριστικό της διεργασίας αρχηγού και την κατάσταση στη οποία βρίσκεται. Ακολουθεί ένας βρόγχος στον οποίον: εξελίσσεται η τροχιά του αυτομάτου διεργασίας με ρυθμό 1, πυροδοτούνται οι εσωτερικές ενέργειες chooseToFail, chooseToRecover, checkLeader, checkLeaderTimeout, electionTimeout και announcementLeaderTimeout. Επίσης, στέλνονται τα μηνύματα που βρίσκονται στο κανάλι και έχουν αποστολέα τη συγκεκριμένη διεργασία της οποίας

εκτελείται το πρόγραμμα και ακολούθως λαμβάνονται τυχόν μηνύματα που βρίσκονται στο κανάλι και έχουν ως παραλήπτη τη συγκεκριμένη διεργασία. Η εξέλιξη της τροχιάς και οι πυροδοτήσεις των ενεργειών γίνονται με τις εντολές follow και fire, αντίστοιχα. Το Σχήμα 4.27 δείχνει τη ροή εκτέλεσης του προγράμματος μεταβάσεων και τροχιών που περιγράψαμε πιο πάνω για τον αλγόριθμο με τοπολογία δέντρου.

```
do
	fire internal P.init;

	for i:Nat where i<runs do
		follow P.time duration 1;

		fire internal P.chooseToFail;
		fire internal P.chooseToRecover;
		fire internal P.checkLeader;
		fire internal P.checkLeaderTimeout;
		fire internal P.electionTimeout;
		fire internal P.announcementLeaderTimeout;

		% Send Messages
		for j:Nat where j<  MPI_Size() do
			fire output P.SEND(m);
		od
		follow SM.DELAY duration (MPI_Size() * 10);

		% Receive Messages
		for j:Nat where j<MPI_Size() do
			fire input RM.probe(j);
			fire output RM.RECEIVE(m);
		od

	od
 od
```

Σχήμα 4.27 Πρόγραμμα μεταβάσεων και τροχιών αυτομάτου σύνθεσης στον αλγόριθμο με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένο με το κανάλι επικοινωνίας MPI

4.3.2 Προδιαγραφή Αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου με το κανάλι επικοινωνίας TCP

Το λεξιλόγιο για τον συγκεκριμένο αλγόριθμο φαίνεται στο Σχήμα 4.28 και περιλαμβάνει το λεξιλόγιο που χρησιμοποιούν τα αυτόματα διεργασίες και το λεξιλόγιο που χρειάζεται το κανάλι επικοινωνίας TCP. Ορίζουμε ένα φυσικό αριθμό για την μεταβλητή messageType, η οποία αντιπροσωπεύει ένα τμήμα του μηνύματος που στέλλει η κάθε διεργασία και που αντιστοιχεί στον τύπο του μηνύματος. Το τμήμα αυτό μαζί με δύο αριθμούς, τύπου Node που

δηλώνουν τις διευθύνσεις IP του αποστολέα και του παραλήπτη του μηνύματος, ορίζουν τον τύπο του μηνύματος (Chan_message) με το οποίο επικοινωνούν οι διεργασίες μεταξύ τους.

```
%%% :Algorithm vocabs

vocabulary alg_specific_voc
       types
          messageType : Nat
end

%%% :TCP Channel vocabs

vocabulary TCPObjectsVoc
       types
          IPv4 :Tuple[one:Nat, two:Nat, three:Nat, four:Nat],
          IPv6 : Tuple[one:Nat, two:Nat, three:Nat, four:Nat, five:Nat,
six:Nat],
          JVMError: String
End

vocabulary TCPNodeVoc
       imports TCPObjectsVoc
       types
          Node : IPv4
       operators
          GT :Node, Node -> Bool,
          EQ :Node, Node -> Bool,
          LT :Node, Node ->Bool
end

vocabulary tcp_specific_voc
       imports alg_specific_voc, TCPObjectsVoc, TCPNodeVoc
       types
          Chan_message : Tuple[message : messageType, sender: Node,
destination: Node],
          Status : Enumeration[closed, notAccepting, opening, emptying,
connecting, reading, rClosing, sConnected, connected, accepting, waiting,
stopping, idle]
end

%% JVM Socket types and operations
vocabulary JVMSocket
       imports TCPObjectsVoc, TCPNodeVoc, tcp_specific_voc
       imports alg_specific_voc
       types JVMSocket
       operators
          JVM_TCPSocketOpen : Node, Nat, Nat -> Null[JVMSocket],
          JVM_TCPSocketClose : JVMSocket -> Null[JVMError],
          JVM_TCPSocketGetLocalIP :Null[JVMSocket] -> Null[Node],
          JVM_TCPSocketGetRemoteIP :Null[JVMSocket] -> Null[Node],
          JVM_read_TCPSocket : Null[JVMSocket] -> Null[Chan_message],
          JVM_write_TCPSocket : JVMSocket, Chan_message -> Null[JVMError],
          JVM_TCPSocketIsConnected :Null[JVMSocket] ->Bool
end


vocabulary JVMServerSocket
       imports TCPObjectsVoc, JVMSocket, TCPNodeVoc
       types JVMServerSocket
```

```
      operators
         JVM_TCPServerSocketOpen : Node, Nat, Nat -> Null[JVMServerSocket],
         JVM_TCPServerSocketClose : JVMServerSocket -> Null[JVMError],
         JVM_TCPServerSocketAccept :JVMServerSocket ->Null[JVMSocket]
end


%% This type provides sugar for the actual types and provides declaration for
types in the specification of the JCP channel.
vocabulary ChannelVoc
      imports JVMSocket, TCPObjectsVoc, tcp_specific_voc
      types
         Channel : Tuple[node:Node, socket:Null[JVMSocket], status:Status,
emptying:Bool, error:Null[JVMError]]
      operators
         empty_channel : ->Channel
end
```

Σχήμα 4.28 Λεξιλόγιο αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένου με το κανάλι επικοινωνίας TCP

Στο Παράρτημα Β3.2.2 βρίσκεται η προδιαγραφή του αυτομάτου που αντιπροσωπεύει μία διεργασία του αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένου με το κανάλι επικοινωνίας TCP στο εργαλείο Tempo. Το αυτόματο αυτό, ονομάζεται Algorithm3_Process και παίρνει ως παραμέτρους : το χρονικό διάστημα timeForElection στο οποίο θα πρέπει να ολοκληρωθεί η διαδικασία εκλογής αρχηγού , το χρονικό διάστημα timeForAnnouncementLeader στο οποίο θα πρέπει να ανακοινωθεί το αναγνωριστικό της διεργασίας αρχηγού στη διεργασία που ξεκινάει τη διαδικασία εκλογής αρχηγού, το χρονικό διάστημα timeForCheckleader στο οποίο μια διεργασία θα ελέγχει εάν η διεργασία αρχηγός είναι ζωντανή, την πιθανότητα possibilityToFail που αντιστοιχεί στην πιθανότητα μια διεργασία να καταρρεύσει, την πιθανότητα possibilityToRecover που αντιστοιχεί στην πιθανότητα μια διεργασία που έχει καταρρεύσει να επανέλθει, την διεύθυνση IP της διεργασίας και τον αριθμό που αντιστοιχεί στην ταυτότητα της διεργασίας.

Στο Σχήμα 4.29 παρουσιάζονται οι ενέργειες που μπορεί να εκτελέσει μια διεργασία και που δηλώνονται στο τμήμα signature του αυτομάτου.

```
signature
             input RECEIVE (m:Null[Chan_message])
             output systemInitialize(w:Seq[Node])
             output SEND (m:Null[Chan_message])
             internal init,checkLeader
             internal chooseToFail, chooseToRecover
             internal checkLeaderTimeout, electionTimeout, announcementLeaderTimeout
```

Σχήμα 4.29 Δήλωση ενεργειών μιας διεργασίας του αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένο με το κανάλι επικοινωνίας TCP

Στο Σχήμα 4.30 παρουσιάζονται οι μεταβλητές καταστάσεων του αυτομάτου. Ορίζονται στο τμήμα states και είναι ίδιες με τις μεταβλητές καταστάσεων του αυτομάτου μιας διεργασίας του αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένου με το κανάλι επικοινωνίας MPI. Επιπλέον, υπάρχουν οι ακόλουθες μεταβλητές :

- processIP : Είναι μια μεταβλητή τύπου Node, που αντιπροσωπεύει την διεύθυνση IP της κάθε διεργασίας. Αρχικοποιείται με τη διεύθυνση της διεργασίας που δίνεται ως παράμετρος στο αυτόματο.
- leaderIP : Είμαι μια μεταβλητή τύπου Node, που αντιπροσωπεύει τη διεύθυνση IP της διεργασίας αρχηγού. Αρχικοποιείται με τη διεύθυνση IP της διεργασίας.
- world : Είναι μία κενή ακολουθία IP διευθύνσεων, στην οποία περιέχονται οι διευθύνσεις όλων των διεργασιών του συστήματος.

```
states
       % The physcal clock of the automaton
       clock : DiscreteReal :=0;
       % The physical clock to check leader process
       checkLeaderTimer : DiscreteReal :=0;
       % The physical clock to elect leader
       electionTimer : DiscreteReal :=0;
       % The physical clock to announce leader
       announcementLeaderTimer : DiscreteReal :=0;
       % An empty sequence of mpi messages
       msgs :Seq[Null[Chan_message]] :={};
       % A counter for the messages that a process has send
       messagesCounter : Int := 0;
       % Indicates that the process is active
       processIsActive : Bool :=true;
       % Indicates the rank of this process
       processRank : Nat := rank;
       % Indicates the IP of this process
       processIP : Node := IP ;
       % Indicates the id of the leader process
       leaderId : Nat := rank;
       % Indicates the IP of the leader process
       leaderIP : Node:=IP;
       % Tuple which indicates the possible status of the process
       status : Tuple[thereIsLeader:Bool,
election:Bool,waitForLeaderAnnouncement:Bool] :=[false,false,false];
       % Message to election a leader
       ElectionMessage : Nat :=1;
       % Message to reply in election message
       ReplyElectionMessage : Nat :=2;
       % Message to announce the leader
       AnnouncementLeaderMessage :Nat :=3;
       % Message to check the leader
       CheckLeaderMessage : Nat :=4;
       % Message to reply in check leader message
       ReplyCheckLeaderMessage : Nat :=5;
       % An empty sequence of IP andresses
       world : Seq[Node] := {};
```

Σχήμα 4.30 Μεταβλητές Καταστάσεως μιας διεργασίας του αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένο με το κανάλι επικοινωνίας TCP

Οι ενέργειες της διεργασίας που έχουν δηλωθεί στο τμήμα signature, υλοποιούνται στο τμήμα transitions (Σχήμα 4.31). Οι ενέργειες της διεργασίας με το κανάλι TCP, στον αλγόριθμο με τοπολογία ισχυρά συνδεδεμένου γράφου, είναι ίδιες με τις ενέργειες του ίδιου αλγορίθμου υλοποιημένου με το κανάλι MPI που περιγράψαμε προηγουμένως. Οι βασικές διαφορές μεταξύ των δύο αυτών υλοποιήσεων, οφείλονται στις δύο κύριες συναρτήσεις του καναλιού MPI τις οποίες το κανάλι TCP δεν υλοποιεί. Συγκεκριμένα, για να βρούμε τον αριθμό των διεργασιών του συστήματος, με το κανάλι MPI χρησιμοποιούσαμε τη συνάρτηση MPI_Size(), ενώ με το κανάλι TCP χρησιμοποιούμε τη συνάρτηση len. Με το len(world), μας επιστρέφεται το μέγεθος της ακολουθίας world και κατά συνέπεια ο αριθμός των διευθύνσεων των διεργασιών που υπάρχουν στο σύστημα. Για να βρούμε το αναγνωριστικό της κάθε διεργασίας, στο κανάλι MPI χρησιμοποιούσαμε τη συνάρτηση MPI_Rank(), ενώ στο κανάλι TCP το περνούμε ως παράμετρο μαζί με την IP διεύθυνσή της κατά την εκτέλεσή της. Στην υλοποίηση του συγκεκριμένου αλγορίθμου με το κανάλι TCP, προστίθεται ακόμη μια ενέργεια, η ενέργεια systemInitialize, η οποία αρχικοποιεί την ακολουθία world με τις IP διευθύνσεις των διεργασιών που υπάρχουν στο σύστημα.

```
transitions
       output systemInitialize(w)
          eff
             world := w;

       input RECEIVE(m)
          locals iAmLeader : Bool := false;
          eff
             if(processIsActive)then
                 if(m ~= nil())then
                    if(val(m).message = ElectionMessage)then
                        status := [false,true,false];
                        msgs := msgs |- embed([ReplyElectionMessage, IP,
val(m).sender]);
                        iAmLeader := true;
                        for n:Nat where n < len(world)do
                           if(n> processRank)then
                              msgs := msgs |- embed([ElectionMessage, IP,
world[n]]);
                              iAmLeader := false;
                           fi;
                        od;

                        if(iAmLeader = true)then
                           leaderIP := IP;
                           leaderId := processRank;
                           status := [true,false,false];
                           for n:Nat where n < len(world) do
                             if(n < processRank )then
                                msgs := msgs |- embed([AnnouncementLeaderMessage,
IP, world[n]]);
                             fi;
                           od;
                        fi;
```

```
                    fi;
                    if(val(m).message = ReplyElectionMessage)then
                        electionTimer := 0;
                        status := [false,false,true];
                        announcementLeaderTimer := clock;
                    fi;

                    if(val(m).message = AnnouncementLeaderMessage)then
                        announcementLeaderTimer := 0;
                        status := [true,false,false];
                        leaderIP := val(m).sender;
                        for n:Nat where n<len(world)do
                            if(EQ(leaderIP,world[n]))then
                                leaderId := n;
                            fi;
                        od
                    fi;

                if(val(m).message = CheckLeaderMessage)then
                    msgs := msgs |- embed([ReplyCheckLeaderMessage, IP,
val(m).sender]);
                fi;

                if(val(m).message = ReplyCheckLeaderMessage)then
                    checkLeaderTimer :=0;
                fi;

            fi;
        fi;

    output SEND (m)
        pre
            processIsActive;
            msgs ~= {};
            m = head(msgs);
        eff
            msgs := tail(msgs);
            messagesCounter := messagesCounter + 1;

            if(val(m).message = ElectionMessage)then
                electionTimer :=clock;
            fi;

            if(val(m).message = CheckLeaderMessage)then
                checkLeaderTimer := clock;
            fi;

    internal init
        eff
            processIsActive := true;
            status := [true,false,false];

            leaderId := 0;
            for n:Nat where n < len(world)do
                if(n>processRank)then
                    leaderIP := world[n];
                    leaderId := n;
                fi;
            od;

    internal checkLeader
        pre
```

```
             processIsActive;
             leaderId ~= processRank;
             status = [true,false,false];
          eff
             msgs := msgs |- embed([CheckLeaderMessage,IP,leaderIP]);


       internal chooseToFail
          locals randomValueToFail : Int := choose n where (n>=1 /\ n <= 100);
          eff
             if(processIsActive)then
                if(randomValueToFail >= 0 /\ randomValueToFail <=
possibilityToFail)then
                   processIsActive := false;
                fi;
             fi;



       internal chooseToRecover
          locals randomValueToRecover : Int := choose n where (n>=1 /\ n <=
100);
          eff
             if(processIsActive = false)then

                if(randomValueToRecover >=0 /\  randomValueToRecover <=
possibilityToRecover)then
                   processIsActive := true;
                   if(leaderId<processRank)then
                      for n:Nat where n < len(world)do
                         if(n~=processRank)then
                            msgs := msgs |-
embed([AnnouncementLeaderMessage,IP,world[n]]);
                         fi;
                      od;
                   else
                      for n:Nat where n < len(world)do
                         if(n>processRank)then
                            msgs := msgs |-
embed([ElectionMessage,IP,world[n]]);
                         fi;
                      od;
                   fi;
                fi;

             fi;

       internal checkLeaderTimeout
          locals iAmLeader : Bool := false;
          pre
             clock > (checkLeaderTimer + timeForCheckLeader);
             processIsActive;
             status = [true,false,false];
          eff
             status := [false,true,false];

             iAmLeader := true;
             for n:Nat where n < len(world)do
                if(n>processRank)then
                   msgs := msgs |- embed([ElectionMessage, IP, world[n]]);
                   iAmLeader := false;
```

```
                fi;
            od;

            if(iAmLeader = true)then
                leaderId := processRank;
                leaderIP := IP;
                status := [true,false,false];
                for n:Nat where n < len(world) do
                    if(n~=processRank)then
                        msgs := msgs |- embed([AnnouncementLeaderMessage, IP,
world[n]]);
                    fi;
                od;
            fi;

            checkLeaderTimer := 0;


        internal electionTimeout
            pre
                clock > (electionTimer + timeForElection);
                processIsActive;
                status = [false,true,false];
            eff
                for n:Nat where n < len(world) do
                    if(n>processRank)then
                        msgs := msgs |- embed([ElectionMessage, IP, world[n]]);
                    fi;
                od;

            electionTimer := 0;

        internal announcementLeaderTimeout
            pre
                clock > (announcementLeaderTimer + timeForAnouncementLeader);
                processIsActive;
                status = [false,false,true];
            eff
                status := [true,false,false];
                leaderId :=processRank;
                leaderIP := IP;
                for n:Nat where n < len(world) do
                    if(n < processRank)then
                        msgs := msgs |- embed([AnnouncementLeaderMessage, IP,
world[n]]);
                    fi;
                od;
                announcementLeaderTimer :=0;
```

Σχήμα 4.31 Ορισμός ενεργειών μιας διεργασίας του αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένου με κανάλι επικοινωνίας TCP

Αφού έχουν οριστεί οι ενέργειες του αυτομάτου, μετά το τμήμα transitions ακολουθεί ο ορισμός της τροχιάς Time στο τμήμα trajectories, η οποία εξελίσσει το φυσικό ρολόι clock με ρυθμό 1, και παρουσιάζεται στο Σχήμα 4.32.

```
trajectories
      trajdef time
            evolve d(clock) = 1;
```

Σχήμα 4.32 Ορισμός τροχιάς μιας διεργασίας του αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένου με κανάλι επικοινωνίας TCP

Εφόσον έχουμε υλοποιήσει το αυτόματο διεργασίας και τα αυτόματα που υλοποιούν τα κανάλια επικοινωνίας είναι ήδη υλοποιημένα, μπορούμε να υλοποιήσουμε το αυτόματο σύνθεσης (Algorithm3), το οποίο βρίσκεται στο Παράρτημα B3.2.3. Ξεκινώντας συμπεριλαμβάνουμε τα αρχεία στα οποία υλοποιούνται : το λεξιλόγιο του αλγορίθμου, το αυτόματο διεργασίας και τα αυτόματα του καναλιού επικοινωνίας TCP (Σχήμα 4.33).

```
%%% .: Vocabulary :.
include "TCPVocabs.tioa"
imports alg_specific_voc, TCPObjectsVoc, TCPNodeVoc, tcp_specific_voc,
JVMSocket, JVMServerSocket, ChannelVoc

%%% .: TCP Mediator Automata :.
include "TCPRecvMed.tioa"
include "TCPSendMed.tioa"
include "TCPChanMed.tioa"

%%% .: Algorithm Automata :.
include "Algorithm3_Process.tioa"
```

Σχήμα 4.33 Εισαγωγή Λεξιλογίων και Αυτομάτων στο Αυτόματο Σύνθεσης για τον αλγόριθμο με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένο με το κανάλι επικοινωνίας TCP

Το αυτόματο σύνθεσης, όπως φαίνεται στο Σχήμα 4.34 παίρνει ως παραμέτρους : τέσσερις φυσικούς αριθμούς που αντιστοιχούν στην διεύθυνση IP της διεργασίας που δημιουργείται, τον αριθμό θύρας port στο οποίο θα γίνει η σύνδεση, το χρονικό διάστημα timeout το οποίο θα πρέπει να δημιουργηθεί η σύνδεση, το χρονικό διάστημα timeForElection στο οποίο θα πρέπει να ολοκληρωθεί η διαδικασία εκλογής αρχηγού , το χρονικό διάστημα timeForAnnouncementLeader στο οποίο θα πρέπει να ανακοινωθεί το αναγνωριστικό της διεργασίας αρχηγού στη διεργασία που ξεκινάει τη διαδικασία εκλογής αρχηγού, το χρονικό διάστημα timeForCheckleader στο οποίο μια διεργασία θα ελέγχει εάν η διεργασία αρχηγός είναι ζωντανή, την πιθανότητα possibilityToFail που αντιστοιχεί στην πιθανότητα μια διεργασία να καταρρεύσει, την πιθανότητα possibilityToRecover που αντιστοιχεί στην πιθανότητα μια διεργασία που έχει καταρρεύσει να επανέλθει και τέλος το αναγνωριστικό της

διεργασία rank που δημιουργείται. Το αυτόματο σύνθεσης αποτελείται από: ένα αυτόματο διεργασίας (P), ένα αυτόματο SendMediator (SM), ένα αυτόματο ReceiveMediator (RM) και ένα ChanMediator(CM). Στο αυτόματο διεργασίας P, δίδονται ως παράμετροι οι παράμετροι του αυτομάτου σύνθεσης, εκτός από τον αριθμό θύρας και το χρονικό διάστημα timeout. Στα υπόλοιπα αυτόματα, δίνονται ως παράμετροι ο αριθμός θύρας και το χρονικό διάστημα timeout.

```
automaton Algorithm3(n1:Nat, n2:Nat, n3:Nat, n4:Nat, port:Nat,
timeout:Nat,timeForElection:DiscreteReal,
timeForAnouncementLeader:DiscreteReal,timeForCheckLeader:DiscreteReal,
              possibilityToFail: Int, possibilityToRecover : Int, processRank
: Nat)
components
       P: Algorithm3_Process(timeForElection,timeForAnouncementLeader,
timeForCheckLeader, possibilityToFail,possibilityToRecover, [n1,n2,n3,n4],
processRank);
       SM: SendMed(port,timeout);
       RM: RecvMed(port,timeout);
       CM: ChanMed(port,timeout);
```

Σχήμα 4.34 Αυτόματο Σύνθεσης για τον αλγόριθμο με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένο με το κανάλι επικοινωνίας TCP

Οι μεταβλητές του αυτομάτου σύνθεσης, στο τμήμα states, παρουσιάζονται στο επόμενο Σχήμα (Σχήμα 4.35) και είναι οι ακόλουθες :

- myIp : : Είμαι μια μεταβλητή τύπου Node, που αντιπροσωπεύει τη διεύθυνση IP της διεργασίας που δίνεται ως παράμετρος.
- world : Είναι μία κενή ακολουθία IP διευθύνσεων, στην οποία θα καταχωρηθούν οι διευθύνσεις όλων των διεργασιών του συστήματος.
- ms : Δηλώνει ένα μήνυμα Chan_message που θα σταλεί από τη διεργασία.
- mr : Δηλώνει ένα μήνυμα Chan_message που θα λάβει η διεργασία.
- dowhile : Είναι boolean μεταβλητή και αρχικοποιείται με true.
- isconn : Είναι boolean μεταβλητή, αρχικοποιείται με false και δείχνει εάν έχει δημιουργηθεί επιτυχώς το κανάλι TCP μεταξύ δύο διεργασιών.
- error : Δηλώνει μήνυμα λάθους στη σύνδεση μεταξύ δύο διεργασιών και είναι τύπου JVMError.
- runs : Αντιπροσωπεύει τον αριθμό των επαναλήψεων του αλγόριθμου που θα εκτελεστούν στο πρόγραμμα μεταβάσεων και τροχιών. Αρχικοποιείται με $(MPI\_Size)^2$ που αντιστοιχεί στον μέγιστο αριθμό γύρων που χρειάζεται ο αλγόριθμος με τοπολογία ισχυρά συνδεδεμένου γράφου για την εκλογή αρχηγού.

```
states
	myIp : Node := [n1,n2,n3,n4];    %IP from parameters
	world : Seq[Node] := {};
	ms:Null[Chan_message] := nil();
	mr:Null[Chan_message] := nil();
	dowhile : Bool :=true;
	isconn : Bool := false;
	error : Null[JVMError] :=nil();
	runs : Nat := MPI_Size()**2;
```

Σχήμα 4.35 Μεταβλητές του αυτομάτου σύνθεσης στον αλγόριθμο με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένο με το κανάλι επικοινωνίας TCP

Το πρόγραμμα μεταβάσεων και τροχιών (schedule) αρχίζει με την εισαγωγή στην ακολουθία world, των διευθύνσεων IP των διεργασιών που θα συμμετέχουν στην συγκεκριμένη εκτέλεση του αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου. Αρχικά, πυροδοτείται η εξωτερική ενέργεια systemInitialize για να ενημερωθεί κάθε διεργασία για την ακολουθία με τις διευθύνσεις όλως των διεργασιών του συστήματος. Στη συνέχεια, κάθε διεργασία δημιουργεί και δεσμεύει μίαν υποδοχή για την επικοινωνίας της με τις άλλες διεργασίες, με τις με τις ενέργειες TCP_bind και TCP_respBind. Επίσης, δημιουργούνται οι συνδέσεις μεταξύ των διεργασιών με την πυροδότηση των ενεργειών TCP_senderOpen, TCP_accept και TCP_respSenderOpen. Ακολουθεί η πυροδότηση της εσωτερικής ενέργειας init για την αρχικοποίηση από κάθε διεργασία το αναγνωριστικό της διεργασίας αρχηγού και την κατάσταση στη οποία βρίσκεται. Ακολουθεί ένας βρόγχος στον οποίον: εξελίσσεται η τροχιά του αυτομάτου διεργασίας με ρυθμό 1, πυροδοτούνται οι εσωτερικές ενέργειες chooseToFail, chooseToRecover, checkLeader, checkLeaderTimeout, electionTimeout και announcementLeaderTimeout. Επίσης, στέλνονται τα μηνύματα που βρίσκονται στο κανάλι και έχουν αποστολέα τη συγκεκριμένη διεργασία της οποίας εκτελείται το πρόγραμμα και ακολούθως λαμβάνονται τυχόν μηνύματα που βρίσκονται στο κανάλι και έχουν ως παραλήπτη τη συγκεκριμένη διεργασία. Η εξέλιξη της τροχιάς και οι πυροδοτήσεις των ενεργειών γίνονται με τις εντολές follow και fire, αντίστοιχα. Το Σχήμα 4.36 δείχνει τη ροή εκτέλεσης του προγράμματος μεταβάσεων και τροχιών που περιγράψαμε πιο πάνω για τον αλγόριθμο με τοπολογία δέντρου.

```
do
	%% INITIALIZE
	world := world |- [163,221,11,71];
	world := world |- [196,219,200,163];
	world := world |- [128,226,103,67];
	world := world |- [142,150,3,77];
	world := world |- [128,195,54,161];
	world := world |- [147,126,66,17];
```

```
       if [n1,n2,n3,n4] \notin world then
             world := world |- [n1,n2,n3,n4];
       fi

       %% triger initialization of all components
       fire output P.systemInitialize(world);

       %% bind to server socket
       %% everyone sets up their server socket
       fire output RM.TCP_bind(myIp);
       fire output CM.TCP_respBind(error,myIp);

       %% connect to each other
       %% create connections to the server
       for n:Nat where n<len(world)do
          fire output SM.TCP_senderOpen(world[n],port);
          follow SM.DELAY duration 200;
          %% accept only on new connections
          fire output RM.TCP_accept;

          %% listen and accept
          fire output CM.TCP_respAccept(error);
          if error ~= nil() then
             print val(error);
          fi
          isconn :=false;
          fire output CM.TCP_respSenderOpen(world[n],port,isconn);
       od

       fire internal P.init;

       for i:Nat where i<runs do
          follow P.time duration 1;

          fire internal P.chooseToFail;
          fire internal P.chooseToRecover;
          fire internal P.checkLeader;
          fire internal P.checkLeaderTimeout;
          fire internal P.electionTimeout;
          fire internal P.announcementLeaderTimeout;

          % -- SEND
          dowhile := true;
          while (dowhile = true)do
             ms := nil();
             fire output P.SEND(ms);
             if(ms ~= nil()) then
                fire output SM.TCP_write(ms, val(ms).sender,
val(ms).destination);
                print val(ms);
             else
                dowhile :=false;
             fi
          od

          follow SM.DELAY duration 200;

          % -- RECEIVE
```

```
fire output RM.TCP_read;
       dowhile :=true;
       while (dowhile = true)do
          mr := nil();
          fire output CM.TCP_respRead(mr);
          if(mr~=nil())then
             fire output RM.RECEIVE(mr);
             print val(mr);
          else
             dowhile :=false;
          fi
       od
    od
od
```

Σχήμα 4.36 Πρόγραμμα μεταβάσεων και τροχιών αυτομάτου σύνθεσης στον αλγόριθμο με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένο με το κανάλι επικοινωνίας TCP

4.4 Προδιαγραφή Τροποποιημένου Αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου

Η ολοκληρωμένη προδιαγραφή του τροποποιημένου αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου στη γλώσσα TIOA και στο εργαλείο Tempo, όπως αυτός έχει περιγραφεί στο προηγούμενο κεφάλαιο, βρίσκεται στο Παράρτημα Β.4. Να σημειώσουμε πως το Election Commission που χρησιμοποιεί ο συγκεκριμένος αλγόριθμος, θα υλοποιηθεί ως μια ξεχωριστή διεργασία από τις υπόλοιπες.

Για την υλοποίηση του αλγορίθμου, χρειάζεται να υλοποιήσουμε: το λεξιλόγιο που θα χρησιμοποιείται από τα αυτόματα του αλγορίθμου, το αυτόματο το οποίο θα υλοποιεί την κάθε διεργασία (εκτός από τη διεργασία Election Commission), το αυτόματο που θα υλοποιεί τη διεργασία Election Commission, τα αυτόματα που υλοποιούν το κανάλι επικοινωνίας μεταξύ των διεργασιών και το αυτόματο σύνθεσης. Η επικοινωνία μεταξύ των διεργασιών για το συγκεκριμένο αλγόριθμο θα υλοποιηθεί και με το κανάλι επικοινωνίας MPI, την προδιαγραφή του οποίου παραθέτουμε στο Παράρτημα Α.

Το λεξιλόγιο για τον συγκεκριμένο αλγόριθμο φαίνεται στο Σχήμα 4.37 και περιλαμβάνει το λεξιλόγιο που χρησιμοποιούν τα αυτόματα διεργασίες και το λεξιλόγιο που χρειάζεται το κανάλι επικοινωνίας MPI. Ορίζουμε μία πλειάδα Message, η οποία αντιπροσωπεύει ένα τμήμα του μηνύματος που στέλλει η κάθε διεργασία και αποτελείται από δύο φυσικούς αριθμούς οι οποίοι αντιστοιχούν στον τύπο του μηνύματος (messageType) και στην ταυτότητα της διεργασίας αρχηγού (leaderId). Το τμήμα αυτό μαζί με δύο φυσικούς αριθμούς

που δηλώνουν τον αποστολέα και τον παραλήπτη του μηνύματος, ορίζουν τον τύπο του μηνύματος (mpi_message) με το οποίο επικοινωνούν οι διεργασίες μεταξύ τους.

```
%%% :Algorithm vocabs

vocabulary algorithm_voc
	types
		Message: Tuple[messageType : Nat,leaderId:Nat]
end


%%% :MPI Channel vocabs

vocabulary mpi_status_voc
	types mpi_status
	operators
		MPI_Iprobe : Nat -> Null[mpi_status],
		MPI_Test : mpi_status ->Bool
end

vocabulary mpi_message_voc
	imports algorithm_voc, mpi_status_voc
		types mpi_message : Tuple[message:Message, sender:Nat,
destination:Nat]
		operators
			MPI_Irecv: mpi_status, Nat -> mpi_message
end


vocabulary mpi_request_voc
	imports mpi_message_voc
		types mpi_request
		operators
			MPI_Isend: mpi_message, Nat -> Null[mpi_request],
			MPI_Barrier : -> Bool
end

vocabulary mpi_voc
	operators
		MPI_Rank : -> Nat,
		MPI_Size : -> Nat
end
```

Σχήμα 4.37 Λεξιλόγιο τροποποιημένου αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένου

Στο Παράρτημα Β4.2 βρίσκεται η προδιαγραφή του αυτομάτου που αντιπροσωπεύει μία διεργασία του τροποποιημένου αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένου. Το αυτόματο αυτό, ονομάζεται Algorithm4_Process και παίρνει ως παραμέτρους : το χρονικό διάστημα timeForElection στο οποίο θα πρέπει να ολοκληρωθεί η διαδικασία εκλογής αρχηγού , το χρονικό διάστημα timeForAnnouncementLeader στο οποίο θα πρέπει να ανακοινωθεί το αναγνωριστικό της διεργασίας αρχηγού στη διεργασία που

ξεκινάει τη διαδικασία εκλογής αρχηγού, το χρονικό διάστημα timeForCheckleader στο οποίο μια διεργασία θα ελέγχει εάν η διεργασία αρχηγός είναι ζωντανή, την πιθανότητα possibilityToFail που αντιστοιχεί στην πιθανότητα μια διεργασία να καταρρεύσει, την πιθανότητα possibilityToRecover που αντιστοιχεί στην πιθανότητα μια διεργασία που έχει καταρρεύσει να επανέλθει και τον αριθμό που αντιστοιχεί στην ταυτότητα της διεργασίας.
Στο Σχήμα 4.38 παρουσιάζονται οι ενέργειες που μπορεί να εκτελέσει μια διεργασία και που δηλώνονται στο τμήμα signature του αυτομάτου.

```
signature
              input RECEIVE (m:Null[mpi_message])
              output SEND (m:Null[mpi_message])
              internal init,checkLeader
              internal chooseToFail, chooseToRecover
              internal checkLeaderTimeout, electionTimeout,
announcementLeaderTimeout
```

Σχήμα 4.38 Δήλωση ενεργειών μιας διεργασίας του τροποποιημένου αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου

Στο Σχήμα 4.39 παρουσιάζονται οι μεταβλητές καταστάσεων του αυτομάτου. Ορίζονται στο τμήμα states και είναι οι ακόλουθες:

- clock : Αντιπροσωπεύει το φυσικό ρολόι της διεργασίας, είναι τύπου DiscreteReal και αρχικοποιείται με 0.
- checkLeaderTimer : Αντιπροσωπεύει το φυσικό ρολόι κάθε διεργασίας, το οποίο υπολογίζει το χρονικό διάστημα στο οποίο η διεργασία θα ελέγχει εάν η διεργασία αρχηγός είναι ζωντανή
- electionTimer : Αντιπροσωπεύει το φυσικό ρολόι κάθε διεργασίας, το οποίο υπολογίζει το χρονικό διάστημα στο οποίο η διαδικασία εκλογής αρχηγού θα πρέπει να ολοκληρωθεί
- announcementLeaderTimer : Αντιπροσωπεύει το φυσικό ρολόι κάθε διεργασίας, το οποίο υπολογίζει το χρονικό διάστημα στο οποίο θα πρέπει να ανακοινωθεί στη διεργασία το αναγνωριστικό της διεργασίας αρχηγού
- msgs : Είναι μία κενή ακολουθία μηνυμάτων mpi_message της κάθε διεργασίας, στην οποία εισέρχονται τα μηνύματα που πρόκειται να στείλει και εξέρχονται τα μηνύματα που στέλνονται επιτυχώς.
- messagesCounter : Είναι ένας μετρητής ο οποίος υπολογίζει πόσα μηνύματα λαμβάνει η διεργασία. Είναι ακέραιος αριθμός και αρχικοποιείται με 0.

- processIsActive : Είναι η boolean μεταβλητή που καθορίζει εάν η διεργασία είναι ενεργή ή όχι. Αρχικοποιείται με false.
- processRank : Είναι η φυσική μεταβλητή που αντιπροσωπεύει την ταυτότητα της κάθε διεργασίας. Αρχικοποιείται με την ταυτότητα της διεργασίας που δίνεται ως παράμετρο στο αυτόματο.
- leaderId : Είναι η φυσική μεταβλητή που αντιπροσωπεύει την ταυτότητα της διεργασίας αρχηγού. Αρχικοποιείται με μηδέν.
- electionCommissionId : Είναι η φυσική μεταβλητή που αντιπροσωπεύει την ταυτότητα της διεργασίας ElectionCommission και αρχικοποιείται με μηδέν.
- electionProcessNumber : Είναι μια φυσική μεταβλητή που αντιπροσωπεύει το αναγνωριστικό της διεργασίας που έχει ξεκινήσει τη διαδικασία εκλογής αρχηγού. Αρχικοποιείται με μηδέν.
- highestProcessNumber : Αντιπροσωπεύει το μεγαλύτερο αναγνωριστικό από όλες τις ενεργές διεργασίες του συστήματος και είναι τύπου Nat. Αρχικοποιείται με μηδέν.
- status : Είναι μία πλειάδα, η οποία αντιπροσωπεύει την κατάσταση στην οποία βρίσκεται μια διεργασία και αποτελείται από τρεις boolean μεταβλητές που αντιπροσωπεύουν τις εξής τρεις καταστάσεις : την κατάσταση στην οποία είναι ενημερωμένη για την διεργασία αρχηγό του συστήματος (thereIsLeader), την κατάσταση στην οποία εκτελείται η διαδικασία εκλογής αρχηγού (election) και τη διαδικασία στην οποία περιμένει να της ανακοινωθεί το αναγνωριστικό της διεργασίας αρχηγού (waitForLeaderAnnouncement). Και οι τρεις καταστάσεις αρχικοποιούνται με false και μόνο μία από τις καταστάσεις κάθε φορά θα γίνεται true.
- ElectionMessage : Αντιπροσωπεύει μήνυμα τύπου ‘Election’. Το στέλνει μια διεργασία στη διεργασία ElectionCommission, για την εκκίνηση της διαδικασίας εκλογής αρχηγού, όταν αντιληφθεί πως η διεργασία αρχηγός δεν είναι ζωντανή.
- ReplyElectionMessage : Αντιπροσωπεύει μήνυμα τύπου ‘Reply Election’. Το στέλνει η διεργασία Election Commission ως απάντηση προς τη διεργασία που της στέλνει μήνυμα ‘Election’.
- AnnouncementLeaderMessage : Αντιπροσωπεύει μήνυμα τύπου ‘Announcement Leader’. Το στέλνει η διεργασία ElectionCommission προς μιαν άλλη διεργασία για να της ανακοινώσει το αναγνωριστικό της διεργασίας που έχει εκλεγεί ως αρχηγός.

- CheckLeaderMessage : Αντιπροσωπεύει μήνυμα τύπου ‘Check Leader’. Το στέλνει μια διεργασία προς τη διεργασία αρχηγό για να ελέγξει εάν είναι ζωντανή.
- ReplyCheckLeaderMessage : Αντιπροσωπεύει μήνυμα τύπου ‘Reply Check leader’. Το στέλνει η διεργασία αρχηγός ως απάντηση προς τη διεργασία που τις στέλνει μήνυμα ‘Reply Check Leader’.
- QueryMessage : Αντιπροσωπεύει μήνυμα τύπου ‘Query’. Το στέλνει μια διεργασία στη διεργασία Election Commission, όταν επανέρχεται στο σύστημα για να ενημερωθεί για το αναγνωριστικό της διεργασίας αρχηγού του συστήματος.
- CheckProcessMessage : Αντιπροσωπεύει μήνυμα τύπου ‘Check Process’. Το στέλνει η διεργασία Election Commission μια διεργασία προς μια διεργασία για να ελέγξει εάν είναι ζωντανή.
- ReplyCheckProcessMessage : Αντιπροσωπεύει μήνυμα τύπου ‘Reply Check Process’. Το στέλνει μια διεργασία ως απάντηση προς τη διεργασία Election Commission που τις στέλνει μήνυμα ‘Reply Check Leader’, επιβεβαιώνοντάς της ότι είναι ζωντανή.

```
states
            % The physcal clock of the automaton
            clock : DiscreteReal := 0;
            % The physical clock to check leader process
            checkLeaderTimer : DiscreteReal := 0
            % The physical clock to elect leader
            electionTimer : DiscreteReal := 0;
            % The physical clock to announce leader
            announcementLeaderTimer : DiscreteReal := 0;
            % An empty sequence of mpi messages
            msgs :Seq[Null[mpi_message]] := {};
            % A counter for the messages that a process has send
            messagesCounter : Int := 0;
            % A counter for the messages that sent for election
            %messagesForElection : Int :=0;
            % Indicates that the process is active
            processIsActive : Bool := true;
            % Indicates the rank of this process
            processRank : Nat := rank;
            % Indicates the id of the leader process
            leaderId : Nat:= 0;
            % Indicates the id of the Election Commission
            electionCommissionId :Nat := 0;
            % Indicates the id of the process which send election message
            electionProcessNumber :Nat := 0;
            % Indicates the highest id of all the processes
            highestProcessNumber :Nat := 0;
            % Tuple which indicates the possible status of the process
            status : Tuple[thereIsLeader:Bool,
election:Bool,waitForLeaderAnnouncement:Bool] :=[false,false,false];
```

```
% Message to election a leader
ElectionMessage : Nat := 1;
% Message to reply in election message
ReplyElectionMessage : Nat := 2;
% Message to announce the leader
AnnouncementLeaderMessage :Nat := 3;
% Message to check the leader
CheckLeaderMessage : Nat := 4;
% Message to reply in check leader message
ReplyCheckLeaderMessage : Nat := 5;
% Indicates query message
QueryMessage : Nat := 6;
% Message to check a process if is alive
CheckProcessMessage : Nat := 7;
% Message to reply in check process message
ReplyCheckProcessMessage : Nat := 8;
```

Σχήμα 4.39 Μεταβλητές Καταστάσεως μιας διεργασίας του τροποποιημένου αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου

Οι ενέργειες της διεργασίας που έχουν δηλωθεί στο τμήμα signature, υλοποιούνται στο τμήμα transitions (Σχήμα 4.40). Αναλυτικά:

- Με την ενέργεια εισόδου RECEIVE, η διεργασία λαμβάνει ένα μήνυμα m, τύπου mpi_message. Εάν το μήνυμα δεν είναι κενό και ο τύπος του μηνύματος είναι ReplyElectionMessage, τότε παίρνει απάντηση από το Election Commission στο οποίο έχει στείλει Election μήνυμα που δηλώνει πως έχει ξεκινήσει την διαδικασία εκλογής αρχηγού. . Έτσι, θέτει την μεταβλητή electionTimer ίση με μηδέν, όλες τις καταστάσεις της με false εκτός από την κατάσταση waitForLeaderAnnouncement και ξεκινάει τον μετρητή announcement LeaderTimer περιμένοντας να του ανακοινωθεί το αναγνωριστικό της διεργασίας αρχηγού. Όταν η διεργασία λάβει AnnouncementLeader μήνυμα τότε της ανακοινώνεται από το Election Commission το αναγνωριστικό της διεργασίας αρχηγού, το οποίο και αποθηκεύει στη μεταβλητή της leaderId. Επίσης θέτει την κατάστασή της thereIsLeader αληθές, ενώ όλες τις υπόλοιπες καταστάσεις της ψευδές και μηδενίζει τον μετρητή announcementLeaderTimer. Εάν μια διεργασία λάβει μήνυμα CheckLeader, τότε δηλώνεται πως είναι η τρέχουσα διεργασία αρχηγός και η διεργασία που αποστέλλει το μήνυμα θέλει να ελέγξει εάν είναι ζωντανή. Έτσι, απαντά σ' αυτήν την διεργασία με μήνυμα ReplyCheckLeader. Όταν φθάσει μήνυμα ReplyCheckLeader, σημαίνει πως παίρνει απάντηση από τη διεργασία αρχηγό και επομένως είναι ζωντανή. Δεν χρειάζεται να ξεκινήσει τη διαδικασία εκλογής αρχηγού και μηδενίζει τον μετρητή checkLeaderTimer. Τέλος, όταν μια διεργασία λάβει μήνυμα CheckProcess, σημαίνει πως το Election Commission της στέλνει μήνυμα για να ελέγξει εάν είναι ζωντανή,

είτε είναι η μεγαλύτερη διεργασία στο σύστημα είτε όχι για να εκλεγεί εκείνη ως αρχηγός. Σε περίπτωση που δεν είναι η μεγαλύτερη διεργασία στο σύστημα και παίρνει το συγκεκριμένο μήνυμα, τότε σημαίνει πως είναι η διεργασία με το μεγαλύτερο αναγνωριστικό που είναι ενεργή. Η εκτέλεση της ενέργειας αυτής, πυροδοτεί όλες τις αντίστοιχες ενέργειες εξόδου RECEIVE.

- Με την ενέργεια εξόδου SEND, η διεργασία στέλλει ένα μήνυμα m. Προϋποθέσεις για να εκτελεστεί η ενέργεια αυτή είναι το μήνυμα που θα σταλεί να βρίσκεται πρώτο στην ακολουθία μηνυμάτων msgs, η ακολουθία μηνυμάτων να μη είναι άδεια, η διεργασία που την εκτελεί να είναι ενεργή και να μην είναι η διεργασία Election Commission. Με την ενεργοποίηση της ενέργειας αυτής, αυξάνεται ο αριθμός των μηνυμάτων που στέλνει η διεργασία. Εάν το μήνυμα που αποστέλλεται είναι τύπου ElectionMessage, τότε ξεκινάει ο μετρητής electionTimer, για το χρόνο που θα πρέπει να ολοκληρωθεί η διαδικασία εκλογής αρχηγού. Εάν το μήνυμα που αποστέλλεται είναι τύπου CheckLeader, τότε ξεκινάει ο μετρητής checkLeaderTimer για το χρόνο στον οποίο θα πρέπει να πάρει απάντηση από τη διεργασία αρχηγό έτσι ώστε να μην ξεκινήσει τη διαδικασία εκλογής αρχηγού. Η εκτέλεση της ενέργειας αυτής, πυροδοτεί όλες τις αντίστοιχες ενέργειες εισόδου SEND.
- Με την εσωτερική ενέργεια init, κάθε διεργασία που είναι ενεργή και δεν είναι η διεργασία Election Commission, αρχικοποιεί την μεταβλητή processIsActive με true, που δηλώνει πως είναι ενεργή και θέτει την κατάσταση thereIsLeader στην οποία βρίσκεται αληθές. Επίσης αρχικοποιεί την μεταβλητή με το αναγνωριστικό της διεργασίας Election Commission με το αναγνωριστικό της μεγαλύτερης διεργασίας του συστήματος, ενώ τη μεταβλητή με το αναγνωριστικό της διεργασίας αρχηγού με το αμέσως μικρότερο αναγνωριστικό από την διεργασία ElectionCommission. Για παράδειγμα εάν υπάρχουν 5 διεργασίες στο σύστημά μας, τότε η διεργασία Election Commission είναι η διεργασία με αναγνωριστικό 4 και η διεργασία αρχηγός η διεργασία με αναγνωριστικό 3.
- Με την εσωτερική ενέργεια checkLeader, μια διεργασία η οποία είναι ενεργή, δεν είναι ούτε η διεργασία αρχηγός ούτε η διεργασία Election Commission και γνωρίζει πως υπάρχει αρχηγός στο σύστημα, δηλαδή η κατάσταση thereIsLeader είναι αληθές, στέλνει μήνυμα CheckLeader στην διεργασία αρχηγό για να ελέγξει εάν όντως είναι ζωντανή.
- Με την εσωτερική ενέργεια chooseToFail, μια ενεργή διεργασία, εκτός από τη διεργασία Election Commission, καταρρέει από το σύστημα, θέτοντας τη μεταβλητή

processIsActive ίση με false, όταν η τυχαία μεταβλητή randomValueToFail είναι μικρότερη από την πιθανότητα κατάρρευσης μιας διεργασίας (possibilityToFail).

- Με την εσωτερική ενέργεια chooseToRecover, μια διεργασία μη ενεργή, , εκτός από τη διεργασία Election Commission, επιστρέφει στο σύστημα, θέτοντας τη μεταβλητή processIsActive ίση με true, όταν η τυχαία μεταβλητή randomValueToFail είναι μικρότερη από την πιθανότητα επαναφοράς μιας διεργασίας στο σύστημα (possibilityToRecover). Όταν μια διεργασία θα επανέλθει στο σύστημα τότε στέλνει μήνυμα Query στην διεργασία Election Commission, για να μάθει το αναγνωριστικό της διεργασίας αρχηγού.
- Η εσωτερική ενέργεια checkLeaderTimeout, ενεργοποιείται όταν μια ενεργή διεργασία, εκτός από τις διεργασίες αρχηγό και Election Commission, ενώ γνωρίζει πως υπάρχει διεργασία αρχηγός (η κατάσταση thereIsleader είναι αληθές), δεν παίρνει απάντηση από αυτήν (ReplyCheckLeader μήνυμα) στο CheckLeader μήνυμα που της στέλνει, μέσα σε συγκεκριμένο χρονικό διάστημα. Στην περίπτωση αυτή, η διεργασία ξεκινάει τη διαδικασία εκλογής αρχηγού, στέλνοντας μήνυμα Election στη διεργασία Election Commission.
- Η εσωτερική ενέργεια electionTimeout, ενεργοποιείται όταν ικανοποιούνται οι εξής προϋποθέσεις: η διεργασία είναι ενεργή, δεν είναι η διεργασία αρχηγός ούτε η διεργασία Election Commission, η διαδικασία εκλογής αρχηγού δεν ολοκληρώνεται στο χρονικό διάστημα timeForElection και η διεργασία βρίσκεται στην κατάσταση election, δηλαδή είναι true και οι καταστάσεις thereIsLeader και waitForLeaderAnnouncement είναι false. Ως αποτέλεσμα της ενέργειας αυτής είναι η επανεκκίνηση της διαδικασίας εκλογής αρχηγού από την συγκεκριμένη διεργασία στέλνοντας μηνύματα Election στις διεργασίες με μεγαλύτερα αναγνωριστικά από αυτήν.
- Η εσωτερική ενέργεια announcementLeaderTimeout, ενεργοποιείται όταν μια ενεργή διεργασία που δεν είναι ούτε η διεργασία αρχηγός ούτε η διεργασία Election Commission, ενώ βρίσκεται στην κατάσταση waitForLeaderAnnouncement δεν ενημερώνεται για το αναγνωριστικό της διεργασίας αρχηγού που εκλέγεται, στο χρονικό διάστημα timeAnouncementLeader που καθορίζεται. Στην περίπτωση αυτή, η διεργασία στέλνει και πάλι Election μήνυμα στην διεργασία Election Commission, για να την ενημερώσει πως θα πρέπει να ξεκινήσει η διαδικασία εκλογής νέου αρχηγού.

```
transitions

       input RECEIVE(m)
          locals iAmLeader : Bool := false;
          eff
             if(processIsActive /\ rank ~= electionCommissionId)then
                if(m ~= nil())then

                   if(val(m).message.messageType = ReplyElectionMessage)then
                      electionTimer := 0;
                      status := [false,false,true];
                      announcementLeaderTimer := clock;
                   fi;

                   if(val(m).message.messageType = AnnouncementLeaderMessage)then
                      announcementLeaderTimer := 0;
                      status := [true,false,false];
                      leaderId := val(m).message.leaderId;
                   fi;

                   if(val(m).message.messageType = CheckLeaderMessage)then
                      msgs := msgs |-
embed([[ReplyCheckLeaderMessage,MPI_Size()], rank, val(m).sender]);
                   fi;

                   if(val(m).message.messageType = ReplyCheckLeaderMessage)then
                      checkLeaderTimer := 0;
                   fi;

                   if(val(m).message.messageType = CheckProcessMessage )then
                      msgs := msgs |-
embed([[ReplyCheckProcessMessage,MPI_Size()], rank, val(m).sender]);
                   fi;

                fi;
             fi;

       output SEND (m)
          pre
             processIsActive;
             msgs ~= {};
             m = head(msgs);
             rank ~= electionCommissionId;
          eff
             msgs := tail(msgs);
             if(val(m).message.messageType = ElectionMessage)then
                electionTimer := clock;
             fi;
             if(val(m).message.messageType = CheckLeaderMessage)then
                checkLeaderTimer := clock;
             fi;
             messagesCounter := messagesCounter + 1;

       internal init
          pre
             rank ~= (MPI_Size()-1);
             processIsActive;
          eff
             processIsActive := true;
             status := [true,false,false];
             leaderId := 0;
```

```
            for n:Nat where n <= (MPI_Size()-1)do
               if(n = (MPI_Size()-1))then
                  electionCommissionId := n;
               else
                  if(n > leaderId)then
                     leaderId := n;
                  fi;
               fi;
            od;

      internal checkLeader
         pre
            processIsActive;
            leaderId ~= rank;
            rank ~= electionCommissionId;
            status = [true,false,false];
         eff
            msgs := msgs |-
embed([[CheckLeaderMessage,MPI_Size()],rank,leaderId]);


      internal chooseToFail
         locals randomValueToFail : Int := choose n where (n>=1 /\ n <= 100);
         pre
            rank ~= electionCommissionId;
         eff
            if(processIsActive )then
               if(randomValueToFail >=0 /\ randomValueToFail <=
possibilityToFail)then
                  processIsActive := false;
               fi;
            fi;


      internal chooseToRecover
         locals randomValueToRecover : Int := choose n where (n>=1 /\ n <= 100);
         pre
            rank ~= electionCommissionId;
         eff
            if(processIsActive = false)then

               if(randomValueToRecover >=0 /\  randomValueToRecover <=
possibilityToRecover)then
                  processIsActive := true;
                  msgs := msgs |-
embed([[QueryMessage,MPI_Size()],rank,electionCommissionId]);
               fi;

            fi;

      internal checkLeaderTimeout
         locals iAmLeader : Bool := false;
         pre
            clock > (checkLeaderTimer + timeForCheckLeader);
            processIsActive;
            rank ~= electionCommissionId;
            rank ~= leaderId;
            status = [true,false,false];
         eff
            status := [false,true,false];
```

```
            msgs := msgs |- embed([[ElectionMessage,MPI_Size()], rank,
electionCommissionId]);
            checkLeaderTimer := 0;

      internal electionTimeout
         pre
            clock > (electionTimer + timeForElection);
            processIsActive;
            rank ~= electionCommissionId;
            status = [false,true,false];
         eff
            msgs := msgs |- embed([[ElectionMessage,MPI_Size()], rank,
electionCommissionId]);
            electionTimer := 0;


      internal announcementLeaderTimeout
         pre
            clock > (announcementLeaderTimer + timeForAnouncementLeader);
            processIsActive;
            rank ~= electionCommissionId;
            status = [false,false,true];
         eff
            msgs := msgs |- embed([[ElectionMessage,MPI_Size()], rank,
electionCommissionId]);
            announcementLeaderTimer := 0;
```

Σχήμα 4.40 Ορισμός ενεργειών μιας διεργασίας του τροποποιημένου αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου

Στο Παράρτημα Β.4.3 βρίσκεται η προδιαγραφή του αυτομάτου που αντιπροσωπεύει τη διεργασία Election Commission της πρώτης εκδοχής του τροποποιημένου αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένου. Η προδιαγραφή της δεύτερης εκδοχής βρίσκεται στο Παράρτημα Β4.4. Το αυτόματο αυτό, ονομάζεται Election Commission και παίρνει ως παραμέτρους : το χρονικό διάστημα timeForVerifyLeader στο οποίο θα ελέγχει εάν η διεργασία αρχηγός είναι ζωντανή, το χρονικό διάστημα timeForReplyAliveProcess στο οποίο θα ελέγχει εάν μια διεργασία του συστήματος είναι ζωντανή και τον αριθμό που αντιστοιχεί στην ταυτότητα της διεργασίας.

Στο Σχήμα 41 παρουσιάζονται οι ενέργειες που μπορεί να εκτελέσει η διεργασία Election Commission και που δηλώνονται στο τμήμα signature του αυτομάτου.

```
signature
      input RECEIVE (m:Null[mpi_message])
      output SEND (m:Null[mpi_message])
      internal init
      internal checkLeaderTimeout, checkProcessTimeout
```

Σχήμα 4.41 Δήλωση ενεργειών της διεργασίας Election Commission του τροποποιημένου αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου

Στο Σχήμα 4.42 παρουσιάζονται οι μεταβλητές καταστάσεων του αυτομάτου Election Commission. Οι καταστάσεις αυτές είναι οι ίδιες με τις καταστάσεις του αυτομάτου μιας διεργασίας του τροποποιημένου αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου, με τη διαφορά πως δεν υπάρχει η μεταβλητή status και προστίθενται οι ακόλουθες καταστάσεις:

- numberOfAliveProcess : Είναι ένας μετρητής ο οποίος υπολογίζει πόσες είναι οι ενεργές διεργασίες του συστήματος και αρχικοποιείται με μηδέν.
- numberOfReplyMessages : Είναι ένας μετρητής ο οποίος υπολογίζει πόσα μηνύματα Reply Check Process λαμβάνει, που δηλώνει τον αριθμό των ενεργών διεργασιών που απαντούν στην διεργασία Election Commission. Αρχικοποιείται με μηδέν.
- thereIsLeader : Αντιπροσωπεύει την κατάσταση στην οποία βρίσκεται η διεργασία Election Commission και στην οποία είναι ενημερωμένη για τη διεργασία αρχηγός του συστήματος και αρχικοποιείται με false.

```
states
       % The physcal clock of the automaton
       clock : DiscreteReal := 0;
       % The physical clock to check leader process
       checkLeaderTimer : DiscreteReal := 0;
       % The physical clock to check process
       checkProcessTimer : DiscreteReal := 0;
       % An empty sequence of mpi messages
       msgs :Seq[Null[mpi_message]] := {};
       % A counter for the messages that a process has send
       messagesCounter : Int := 0;
       % Indicates the id of the leader process
       leaderId : Nat:= 0;
       % Indicates the rank of this process
       processRank : Nat := rank;
       % Indicates the id of the Election Commission
       electionCommissionId :Nat := MPI_Size();
       % Indicates the id of the process which send election message
       electionProcessNumber :Nat := 0;
       % Indicates the highest id of all the processes
       highestProcessNumber :Nat := 0;
       % Indicates the number of alive processes
       numberOfAliveProcesses : Int := 0;
       % Indicates the number of reply messages
       numberOfReplyMessages : Int := 0;
       % Indicates if there is a leader in the system
       thereIsLeader:Bool := true;
       % Indicates that an election message had received
       electionFlag : Bool := false;
       % Indicates that a query message had received
       queryFlag : Bool := false;
       % Message to election a leader
       ElectionMessage : Nat := 1;
       % Message to reply in election message
       ReplyElectionMessage : Nat := 2;
       % Message to announce the leader
       AnnouncementLeaderMessage :Nat := 3;
```

```
% Message to check the leader
CheckLeaderMessage : Nat := 4;
% Message to reply in check leader message
ReplyCheckLeaderMessage : Nat := 5;
% Indicates query message
QueryMessage : Nat := 6;
% Message to check a process if is alive
CheckProcessMessage : Nat := 7;
% Message to reply in check process message
ReplyCheckProcessMessage : Nat := 8;
```

Σχήμα 4.42 Μεταβλητές Καταστάσεως της διεργασίας Election Commission του τροποποιημένου αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου

Οι ενέργειες της διεργασίας Election Commission που έχουν δηλωθεί στο τμήμα signature, υλοποιούνται στο τμήμα transitions. Το Σχήμα 4.43 και το Σχήμα 4.44 περιγράφουν τις ενέργειες όπως αυτές υλοποιούνται στην πρώτη και δεύτερη εκδοχή του τροποποιημένου αλγορίθμου, αντίστοιχα. Αναλυτικά:

- Με την ενέργεια εισόδου RECEIVE, η διεργασία Election Message λαμβάνει ένα μήνυμα m, τύπου mpi_message. Εάν το μήνυμα δεν είναι κενό και ο τύπος του μηνύματος είναι ElectionMessage, τότε λαμβάνει μήνυμα για την εκκίνηση της διαδικασίας εκλογής αρχηγού. Θέτει την μεταβλητή electionFlag με true και την μεταβλητή electionProcessNumber με το αναγνωριστικό του αποστολέα του μηνύματος που έχει λάβει, δηλαδή της διεργασίας που έχει κάνει αίτημα για εκλογή αρχηγού. Επίσης απαντά στην διεργασία που της έχει στείλει το συγκεκριμένο μήνυμα με ReplyElection μήνυμα και στέλνει Check Leader μήνυμα στη διεργασία αρχηγό για να ελέγξει εάν είναι ζωντανή. Εάν λάβει μήνυμα ReplyCheckleader, τότε η διεργασία αρχηγός είναι ζωντανή και δεν χρειάζεται να ξεκινήσει η διαδικασία εκλογής αρχηγού. Επομένως ενημερώνει τη διεργασία που της έχει στείλει μήνυμα Election, το αναγνωριστικό της οποίας βρίσκεται αποθηκευμένο στη μεταβλητή electionProcessNumber, για το αναγνωριστικό της διεργασίας αρχηγού. Όταν η διεργασία Election Commission, πάρει κάποιο Query μήνυμα, τότε αντιλαμβάνεται πως είναι μήνυμα από κάποια διεργασία που μόλις έχει επιστρέψει στο σύστημα. Ελέγχει εάν το αναγνωριστικό της συγκεκριμένης διεργασίας είναι μεγαλύτερο από το αναγνωριστικό της διεργασίας αρχηγού. Εάν αληθεύει τότε γίνεται η νέα αρχηγός του συστήματος και γνωστοποιεί το αναγνωριστικό τις σε όλες τις υπόλοιπες διεργασίες με Announcement Leader μήνυμα. Σε αντίθετη περίπτωση, απλώς ενημερώνει τη διεργασία για το αναγνωριστικό της διεργασίας αρχηγού. Τέλος, εάν

λάβει ReplyCheckProcess υπάρχουν δύο διαφορετικές εκδοχές για το πώς θα ενεργήσει η διεργασία Election Commission. Στην πρώτη εκδοχή του τροποποιημένου αλγορίθμου , η διεργασία Election Commission, υποθέτει πως η διεργασία που της στέλνει το μήνυμα αυτό είναι η μεγαλύτερη ενεργή διεργασία τη δεδομένη χρονική στιγμή και είναι η διεργασία που θα εκλεγεί ως νέος αρχηγός. Αντίθετα, στη δεύτερη εκδοχή του τροποποιημένου αλγορίθμου, για να εντοπίσει τη διεργασία που θα εκλεγεί αρχηγός, θα πρέπει να λάβει Reply Election απ' όλες τις διεργασίες του συστήματος σε ένα χρονικό διάστημα, και η μεγαλύτερη από αυτές να εκλεγεί αρχηγός. Η εκτέλεση της ενέργειας αυτής, πυροδοτεί όλες τις αντίστοιχες ενέργειες εξόδου RECEIVE.

- Με την ενέργεια εξόδου SEND, η διεργασία στέλλει ένα μήνυμα m. Προϋποθέσεις για να εκτελεστεί η ενέργεια αυτή είναι το μήνυμα που θα σταλεί να βρίσκεται πρώτο στην ακολουθία μηνυμάτων msgs και η ακολουθία μηνυμάτων να μη είναι άδεια. Με την ενεργοποίηση της ενέργειας αυτής, αυξάνεται ο αριθμός των μηνυμάτων που στέλνει η διεργασία. Εάν το μήνυμα που αποστέλλεται είναι τύπου CheckLeader, τότε ξεκινάει ο μετρητής checkLeaderTimer για το χρόνο στον οποίο θα πρέπει να πάρει απάντηση από τη διεργασία αρχηγό έτσι ώστε να μην ξεκινήσει τη διαδικασία εκλογής αρχηγού. Εάν το μήνυμα που αποστέλλεται είναι τύπου CheckProcess, τότε ξεκινάει ο μετρητής checkProcssTimer για το χρόνο στον οποίο θα πρέπει να πάρει απάντηση από μια διεργασία για να διαπιστώσει πως είναι ζωντανή. Η εκτέλεση της ενέργειας αυτής, πυροδοτεί όλες τις αντίστοιχες ενέργειες εισόδου SEND.
- Με την εσωτερική ενέργεια init αρχικοποιεί την μεταβλητή με το αναγνωριστικό της διεργασίας Election Commission με το αναγνωριστικό, ενώ τη μεταβλητή με το αναγνωριστικό της διεργασίας αρχηγού με το αμέσως μικρότερο αναγνωριστικό από αυτήν.
- Η εσωτερική ενέργεια checkLeaderTimeout, ενεργοποιείται όταν η διεργασία Election Commission ενώ γνωρίζει πως υπάρχει διεργασία αρχηγός (η κατάσταση thereIsleader είναι αληθές), έχει ξεκινήσει η διαδικασία εκλογής αρχηγού (η κατάσταση electionFlag είναι αληθές), δεν παίρνει απάντηση από αυτήν (ReplyCheckLeader μήνυμα) στο CheckLeader μήνυμα που της στέλνει, μέσα σε συγκεκριμένο χρονικό διάστημα. Στην περίπτωση αυτή, ανάλογα με το ποια εκδοχή του αλγορίθμου υλοποιούμε, η διεργασία Election Commission ενεργεί διαφορετικά. Στην πρώτη εκδοχή, θα στείλει CheckProcess μήνυμα στη διεργασία με αναγνωριστικό αμέσως μικρότερο από τη διεργασία αρχηγό που θεωρείται πως δεν είναι ενεργή. Αντίθετα, στη δεύτερη εκδοχή, θα στείλει σε όλες τις άλλες διεργασίες

του συστήματος CheckProcess μήνυμα και θα περιμένει να λάβει απάντηση απ' όλες μέσα σε κάποιο χρονικό διάστημα και να εντοπίσει την ενεργή διεργασία με το μεγαλύτερο αναγνωριστικό που θα εκλεγεί ως νέος αρχηγός.

- Η εσωτερική ενέργεια checkProcessTimeout, ενεργοποιείται όταν έχει διαπιστώσει πως η διεργασία αρχηγός δεν είναι ζωντανή (η κατάσταση της μεταβλητής queryFlag είναι αληθές) και δεν παίρνει απάντηση από μια διεργασία (ReplyCheckLeader μήνυμα) της οποίας έχει στείλει CheckLeader μήνυμα, μέσα σε συγκεκριμένο χρονικό διάστημα. Στην περίπτωση αυτή, ανάλογα με το ποια εκδοχή του αλγορίθμου υλοποιούμε, η διεργασία Election Commission ενεργεί διαφορετικά. Στην πρώτη εκδοχή, θα στείλει CheckProcess μήνυμα στη διεργασία με αναγνωριστικό αμέσως μικρότερο από τη διεργασία που της έχει στείλει CheckProcess μήνυμα και δεν απάντησε στο καθορισμένο χρονικό διάστημα. Αντίθετα, στη δεύτερη εκδοχή, θα μειώσει τον αριθμό των ενεργών διεργασιών του συστήματος, και θα περιμένει να λάβει τόσα μηνύματα για να αποφασίσει ποια είναι η διεργασία με το μεγαλύτερο αναγνωριστικό και να την εκλέξει αρχηγό.

```
transitions
       input RECEIVE (m)
          eff
             if(rank = electionCommissionId)then
                if(m ~= nil())then

                   if(val(m).message.messageType = ElectionMessage)then
                      electionFlag := true;
                      msgs := msgs |- embed([[ReplyElectionMessage,MPI_Size()],
electionCommissionId,val(m).sender]);
                      msgs := msgs |- embed([[CheckLeaderMessage,MPI_Size()],
electionCommissionId,leaderId]);
                      electionProcessNumber := val(m).sender;
                   fi;

                   if(val(m).message.messageType = ReplyCheckLeaderMessage)then
                      checkLeaderTimer := 0;
                      thereIsLeader := true;
                      electionFlag := false;
                      queryFlag := false;
                      msgs := msgs |-
embed([[AnnouncementLeaderMessage,leaderId],
electionCommissionId,electionProcessNumber]);
                   fi;

                   if(val(m).message.messageType = QueryMessage)then
                      queryFlag := true;
                      if(val(m).sender > leaderId)then
                         leaderId := val(m).sender;
                         for n :Nat where n < (MPI_Size()-1) do
                            msgs := msgs |-
embed([[AnnouncementLeaderMessage,leaderId], electionCommissionId,n]);
                         od;
```

```
                        msgs := msgs |-
embed([[AnnouncementLeaderMessage,leaderId], electionCommissionId,n]);
                        od;
                     else
                        msgs := msgs |-
embed([[AnnouncementLeaderMessage,leaderId],
electionCommissionId,val(m).sender]);
                     fi;
                  fi;

                  if(val(m).message.messageType = ReplyCheckProcessMessage)then
                     checkProcessTimer := 0;
                     highestProcessNumber := val(m).sender;
                     leaderId := highestProcessNumber;
                     queryFlag := false;
                     for n :Nat where n < (MPI_Size()-1) do
                        msgs := msgs |-
embed([[AnnouncementLeaderMessage,leaderId], electionCommissionId,n]);
                     od;

                  fi;
               fi;
            fi;


      output SEND (m)
         pre
            msgs ~= {};
            m = head(msgs);
            rank = electionCommissionId;
         eff
            msgs := tail(msgs);
            if(val(m).message.messageType = CheckLeaderMessage)then
               checkLeaderTimer := clock;
            fi;
            if(val(m).message.messageType = CheckProcessMessage)then
               checkProcessTimer := clock;
            fi;
            messagesCounter := messagesCounter + 1;


      internal init
         pre
            rank = (MPI_Size()-1);
         eff
            leaderId := 0;
            for n:Nat where n < (MPI_Size()-1) do
               if(n > leaderId)then
                  leaderId := n;
               fi;
            od;
            electionCommissionId := rank;

      internal checkLeaderTimeout
         pre
            clock > (checkLeaderTimer + timeForVerifyLeader);
            thereIsLeader;
            rank = electionCommissionId;
            electionFlag = true;
```

```
          eff
             thereIsLeader := false;
             highestProcessNumber := 0;

              for n:Nat where n < leaderId do
                 if(n > highestProcessNumber)then
                    highestProcessNumber := n;
                 fi;
              od;
              msgs := msgs |- embed([[CheckProcessMessage,MPI_Size()],
electionCommissionId,highestProcessNumber]);
              checkLeaderTimer := 0;


       internal checkProcessTimeout
          locals highestProcess : Nat :=0;
          pre
             clock > (checkProcessTimer + timeForReplyAliveProcess);
             thereIsLeader = false;
             rank = electionCommissionId;
             queryFlag = true;
          eff
             for n:Nat where n < (MPI_Size()-1) do
                if(n < highestProcessNumber)then
                   highestProcess := n;
                fi;
             od;
```

Σχήμα 4.43 Ορισμός ενεργειών της διεργασίας Election Commission της πρώτης εκδοχής του τροποποιημένου αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου

```
transitions
       input RECEIVE (m)
          eff
             if(rank = electionCommissionId)then
                if(m ~= nil())then

                   if(val(m).message.messageType = ElectionMessage)then
                      electionFlag := true;
                      if(startElectionTime = 0 /\ startFlag = 0)then
                         startElectionTime := clock;
                         startFlag := 1;
                      fi;
                      msgs := msgs |- embed([[ReplyElectionMessage,MPI_Size()],
electionCommissionId,val(m).sender]);
                      msgs := msgs |- embed([[CheckLeaderMessage,MPI_Size()],
electionCommissionId,leaderId]);
                      electionProcessNumber := val(m).sender;
                   fi;

                   if(val(m).message.messageType = ReplyCheckLeaderMessage)then
                      checkLeaderTimer := 0;
                      thereIsLeader := true;
                      electionFlag := false;
                      queryFlag := false;
```

```
                        msgs := msgs |-
embed([[AnnouncementLeaderMessage,leaderId],
electionCommissionId,electionProcessNumber]);
                    fi;

                    if(val(m).message.messageType = QueryMessage)then
                        queryFlag := true;
                        if(val(m).sender > leaderId)then
                            leaderId := val(m).sender;
                            for n :Nat where n < (MPI_Size()-1) do
                                msgs := msgs |-
embed([[AnnouncementLeaderMessage,leaderId], electionCommissionId,n]);
                            od;
                        else
                            msgs := msgs |-
embed([[AnnouncementLeaderMessage,leaderId],
electionCommissionId,val(m).sender]);
                        fi;
                    fi;

                    if(val(m).message.messageType = ReplyCheckProcessMessage)then
                        checkProcessTimer := 0;
                        numberOfReplyMessages := numberOfReplyMessages + 1;
                        queryFlag := false;

                        if(val(m).sender > highestProcessNumber)then
                            highestProcessNumber :=val(m).sender;
                        fi;

                        if(numberOfReplyMessages = numberOfAliveProcesses )then
                            leaderId := highestProcessNumber;
                            for n :Nat where n<(MPI_Size()-1) do
                                msgs := msgs |-
embed([[AnnouncementLeaderMessage,leaderId], electionCommissionId,n]);
                            od;
                            numberOfReplyMessages :=0;
                        fi;

                    fi;
                fi;
            fi;


        output SEND (m)
            pre
                msgs ~= {};
                m = head(msgs);
                rank = electionCommissionId;
            eff
                msgs := tail(msgs);
                if(val(m).message.messageType = CheckLeaderMessage)then
                    checkLeaderTimer := clock;
                fi;
                if(val(m).message.messageType = CheckProcessMessage)then
                    checkProcessTimer := clock;
                fi;
                messagesCounter := messagesCounter + 1;

        internal init
            pre
```

```
            rank = (MPI_Size()-1);
         eff
            leaderId := 0;
            for n:Nat where n < (MPI_Size()-1) do
               if(n > leaderId)then
                  leaderId := n;
               fi;
            od;
            electionCommissionId := rank;


      internal checkLeaderTimeout
         pre
            clock > (checkLeaderTimer + timeForVerifyLeader);
            thereIsLeader;
            rank = electionCommissionId;
            electionFlag = true;
         eff
            thereIsLeader := false;
            numberOfAliveProcesses :=numberOfAliveProcesses -1;
            checkLeaderTimer := 0;
            for n :Nat where n<(MPI_Size()-1) do
               if(n ~= leaderId)then
                  msgs := msgs |- embed([[CheckProcessMessage,MPI_Size()],
electionCommissionId,n]);
               fi;
            od;


      internal checkProcessTimeout
         locals highestProcess : Nat :=0;
         pre
            clock > (checkProcessTimer + timeForReplyAliveProcess);
            thereIsLeader = false;
            rank = electionCommissionId;
            queryFlag = true;
         eff
            numberOfAliveProcesses :=numberOfAliveProcesses -1;
            checkProcessTimer := 0;
```

Σχήμα 4.44 Ορισμός ενεργειών της διεργασίας Election Commission της δεύτερης εκδοχής του τροποποιημένου αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου

Και στο αυτόματο διεργασίας και στο αυτόματο της διεργασίας Election Commission, μετά το τμήμα transitions ακολουθεί ο ορισμός της τροχιάς Time στο τμήμα trajectories, η οποία εξελίσσει το φυσικό ρολόι clock με ρυθμό 1, και παρουσιάζεται στο Σχήμα 4.45.

```
trajectories
      trajdef time
            evolve d(clock) = 1;
```

Σχήμα 4.45 Ορισμός τροχιάς μιας διεργασίας του τροποποιημένου αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου

Εφόσον έχουμε υλοποιήσει το αυτόματο διεργασίας, το αυτόματο της διεργασίας Election Commission και τα αυτόματα που υλοποιούν τα κανάλια επικοινωνίας είναι ήδη υλοποιημένα, μπορούμε να υλοποιήσουμε το αυτόματο σύνθεσης (Algorithm4), το οποίο βρίσκεται στο Παράρτημα Β4.5. Ξεκινώντας συμπεριλαμβάνουμε τα αρχεία στα οποία υλοποιούνται : το λεξιλόγιο του αλγορίθμου, το αυτόματο διεργασίας, το αυτόματο της διεργασίας Election Commission και τα αυτόματα του καναλιού επικοινωνίας MPI (Σχήμα 4.46).

```
%%% .: Vocabulary :.
include "Vocabulary.tioa"
imports algorithm_voc, mpi_status_voc, mpi_message_voc, mpi_request_voc,
mpi_voc

%%% .: MPI Mediator Automata :.
include "SendMediator.tioa"
include "ReceiveMediator.tioa"

%%% .: Algorithm Automata :.
include "Algorithm4a_Process.tioa"
include "ElectionCommission.tioa"
```

Σχήμα 4.46 Εισαγωγή Λεξιλογίων και Αυτομάτων στο Αυτόματο Σύνθεσης για τον τροποποιημένο αλγόριθμο με τοπολογία ισχυρά συνδεδεμένου γράφου

Το αυτόματο σύνθεσης, όπως φαίνεται στο Σχήμα 4.47 παίρνει ως παραμέτρους : το χρονικό διάστημα timeForElection στο οποίο θα πρέπει να ολοκληρωθεί η διαδικασία εκλογής αρχηγού , το χρονικό διάστημα timeForAnnouncementLeader στο οποίο θα πρέπει να ανακοινωθεί το αναγνωριστικό της διεργασίας αρχηγού στη διεργασία που ξεκινάει τη διαδικασία εκλογής αρχηγού, το χρονικό διάστημα timeForCheckleader στο οποίο μια διεργασία θα ελέγχει εάν η διεργασία αρχηγός είναι ζωντανή, την πιθανότητα possibilityToFail που αντιστοιχεί στην πιθανότητα μια διεργασία να καταρρεύσει, την πιθανότητα possibilityToRecover που αντιστοιχεί στην πιθανότητα μια διεργασία που έχει καταρρεύσει να επανέλθει, το χρονικό διάστημα timeForVerifyLeader στο οποίο θα ελέγχει εάν η διεργασία αρχηγός είναι ζωντανή και το χρονικό διάστημα timeForReplyAliveProcess στο οποίο θα ελέγχει εάν μια διεργασία του συστήματος είναι ζωντανή. Το αυτόματο σύνθεσης αποτελείται από: ένα αυτόματο διεργασίας (P), ένα αυτόματο διεργασίας Election Commission (EC), ένα αυτόματο SendMediator (SM) και ένα ReceiveMediator (RM). Στο αυτόματο διεργασίας P, δίδονται ως παράμετροι πρώτοι πέντε παράμετροι του αυτομάτου σύνθεσης καθώς επίσης και η ταυτότητα της διεργασίας μέσω της συνάρτησης MPI_Rank(). Στο αυτόματο διεργασίας Election Commission EC, δίδονται ως παράμετροι οι δύο

τελευταίοι παράμετροι του αυτομάτου σύνθεσης και η ταυτότητα της διεργασίας μέσω της συνάρτησης MPI_Rank().

```
automaton Algorithm4a(timeForElection:DiscreteReal,
timeForAnouncementLeader:DiscreteReal,timeForCheckLeader:DiscreteReal,
              possibilityToFail: Int, possibilityToRecover : Int,
timeForVerifyLeader: DiscreteReal, timeForReplyAliveProcess : DiscreteReal)
components
       P: Algorithm4a_Process(timeForElection,timeForAnouncementLeader,
timeForCheckLeader, possibilityToFail, possibilityToRecover,
                     MPI_Rank() );
       EC : ElectionCommission(timeForVerifyLeader,
timeForReplyAliveProcess,MPI_Rank());
       SM: SendMediator;
       RM: ReceiveMediator;
```

Σχήμα 4.47 Αυτόματο Σύνθεσης για τον τροποποιημένο αλγόριθμο με τοπολογία ισχυρά συνδεδεμένου γράφου

Οι μεταβλητές του αυτομάτου σύνθεσης, στο τμήμα states, παρουσιάζονται στο επόμενο Σχήμα (Σχήμα 4.48) και είναι οι ακόλουθες :

- m : Δηλώνει ένα μήνυμα mpi_message, το οποίο αρχικοποιείται με nil().
- runs : Αντιπροσωπεύει τον αριθμό των επαναλήψεων του αλγόριθμου που θα εκτελεστούν στο πρόγραμμα μεταβάσεων και τροχιών. Αρχικοποιείται με $(MPI\_Size)^2$ που αντιστοιχεί στον μέγιστο αριθμό γύρων που χρειάζεται ο τροποποιημένος αλγόριθμος με τοπολογία ισχυρά συνδεδεμένου γράφου για την εκλογή αρχηγού.

```
states
       m:Null[mpi_message] :=nil();
       runs : Nat :=MPI_SIZE()**2;
```

Σχήμα 4.48 Μεταβλητές του αυτομάτου σύνθεσης στον τροποποιημένο αλγόριθμο με τοπολογία ισχυρά συνδεδεμένου γράφου

Το πρόγραμμα μεταβάσεων και τροχιών (schedule) αρχίζει με την πυροδότηση της εσωτερικής ενέργειας init κάθε αυτομάτου, για την αρχικοποίηση από κάθε διεργασία το αναγνωριστικό της διεργασίας αρχηγού και την κατάσταση στη οποία βρίσκεται. Ακολουθεί ένας βρόγχος στον οποίον: εξελίσσεται η τροχιά των αυτομάτων ρυθμό 1, πυροδοτούνται οι εσωτερικές ενέργειες τους chooseToFail, chooseToRecover, checkLeader, checkLeaderTimeout, electionTimeout και announcementLeaderTimeout. Επίσης, στέλνονται τα μηνύματα που βρίσκονται στο κανάλι και έχουν αποστολέα τη συγκεκριμένη διεργασία

της οποίας εκτελείται το πρόγραμμα και ακολούθως λαμβάνονται τυχόν μηνύματα που βρίσκονται στο κανάλι και έχουν ως παραλήπτη τη συγκεκριμένη διεργασία. Η εξέλιξη της τροχιάς και οι πυροδοτήσεις των ενεργειών γίνονται με τις εντολές follow και fire, αντίστοιχα. Το Σχήμα 4.49 δείχνει τη ροή εκτέλεσης του προγράμματος μεταβάσεων και τροχιών που περιγράψαμε πιο πάνω για τον αλγόριθμο με τοπολογία δέντρου.

```
do
	fire internal P.init;
	fire internal EC.init;

	for i:Nat where i<runs do
	 follow P.time duration 1;

		fire internal P.chooseToFail;
		fire internal P.chooseToRecover;

		fire internal P.checkLeader;

		fire internal P.checkLeaderTimeout;
		fire internal EC.checkLeaderTimeout;
		fire internal EC.checkProcessTimeout;
		fire internal P.electionTimeout;
		fire internal P.announcementLeaderTimeout;

		% Send Messages
		for j:Nat where j< MPI_Size() do
			fire output P.SEND(m);
			fire output EC.SEND(m);
		od

		follow SM.DELAY duration (MPI_Size() * 10);

		% Receive Messages
		for j:Nat where j<MPI_Size() do
			fire input RM.probe(j);
			fire output RM.RECEIVE(m);
		od
	od
```

Σχήμα 4.49 Πρόγραμμα μεταβάσεων και τροχιών αυτομάτου σύνθεσης στον τροποποιημένο αλγόριθμο με τοπολογία ισχυρά συνδεδεμένου γράφου

Κεφάλαιο 5

Πειραματική Αξιολόγηση

5.1 Προετοιμασία και Μεθοδολογία

Η πειραματική αξιολόγηση των αλγορίθμων εκλογής αρχηγού που υλοποιήθηκαν, διαχωρίστηκε σε δύο μέρη. Στο πρώτο μέρος, αξιολογήθηκαν οι αλγόριθμοι που υλοποιήθηκαν με το κανάλι επικοινωνίας MPI σε τοπικό δίκτυο, χρησιμοποιώντας τη βιβλιοθήκη ανοικτού πηγαίου κώδικα MPJ Express. Στο δεύτερο μέρος αξιολογήθηκε ο αλγόριθμος με τοπολογία ισχυρά συνδεδεμένου γράφου που υλοποιήθηκε με το κανάλι TCP στο κατανεμημένο σύστημα PlanetLab.

Η βιβλιοθήκη MPJ Express διανέμεται δωρεάν υπό την αιγίδα του MIT και πληροφορίες σχετικά με αυτήν, όπως οδηγίες εγκατάστασης και χρήσης της, παρέχονται στην επίσημη ιστοσελίδα της MPJ Express [11,12]. Είναι μια βιβλιοθήκη ανταλλαγής μηνυμάτων, που μπορεί να χρησιμοποιηθεί για την ανάπτυξη και εκτέλεση παράλληλων προγραμμάτων στη γλώσσα Java, με τις διεργασίες που μετέχουν σ’ αυτά να επικοινωνούν μεταξύ τους μέσω της διεπαφής ανταλλαγής μηνυμάτων MPI. Επίσης, μπορεί να χρησιμοποιηθεί με δύο τρόπους: με Multicore Configuration, δηλαδή σε φορητούς και επιτραπέζιους υπολογιστές (laptops και desktops), είτε με Cluster Configuration, δηλαδή σε συστοιχία υπολογιστών (clusters). Στο Παράρτημα Γ, υπάρχουν οδηγίες εγκατάστασης και εκτέλεσης των MPJ Express προγραμμάτων σε Multicore και Cluster Configuration [11].

Αρχικά, για την αξιολόγηση των αλγορίθμων εκλογής αρχηγού που υλοποιήθηκαν με το κανάλι επικοινωνίας MPI, δοκιμάστηκε πειραματικά, η ορθότητα της υλοποίησής τους σε τοπικό υπολογιστή. Μέσα από μια σειρά πειραματικών εκτελέσεων και αλλαγών που έγιναν στην υλοποίηση των αλγορίθμων, πήραμε ενδείξεις για την ορθότητα της υλοποίησής τους. Έτσι προχωρήσαμε στην διεξαγωγή των πειραμάτων σε μια συστοιχία 6 υπολογιστών του Τμήματος Πληροφορικής με χαρακτηριστικά 2-Core AMD Opteron, 2.5 GHz CPU και που έτρεχαν σε λειτουργικό σύστημα Linux CentOS V.5.5. Για τη συλλογή των αποτελεσμάτων, εφαρμόστηκαν διάφορα σενάρια εκτέλεσης των αλγορίθμων για διαφορετικό αριθμό διεργασιών. Για κάθε σενάριο, υπολογιζόταν :

- Ο συνολικός χρόνος εκτέλεσης που χρειάζεται κάθε αλγόριθμος για την εκλογή αρχηγού, υπολογίζοντας το χρονικό διάστημα μεταξύ της χρονικής στιγμής που μια διεργασία αντιλαμβάνεται για πρώτη φορά πως δεν υπάρχει αρχηγός στο σύστημα και εκκινεί τη διαδικασία εκλογής αρχηγού, μέχρι τη στιγμή που ανακοινώνεται και στην τελευταία διεργασία για το αναγνωριστικό της διεργασίας αρχηγού.
- Ο συνολικός αριθμός μηνυμάτων που στέλνονται απ' όλες τις διεργασίες κατά την πρώτη διαδικασία εκλογής αρχηγού που ενδεχομένως να εκκινήσουν.

Για την αξιολόγηση του αλγορίθμου εκλογής αρχηγού με τοπολογία ισχυρά συνδεδεμένου γράφου που υλοποιείται με το κανάλι TCP στο PlanetLab, χρησιμοποιήθηκαν συνολικά 32 μηχανές, οι οποίες παρουσιάζονται στο Σχήμα 5.1.

Όνομα Μηχανής (hostname)	***Διεύθυνση Μηχανής (IP)***	***Γεωγραφική Τοποθεσία***
planet-lab2.cs.ucr.edu	169.235.24.232	ΗΠΑ
planetlab1.cs.uml.edu	129.63.159.102	ΗΠΑ
planetlab2.cs.uml.edu	129.63.159.101	ΗΠΑ
planetlab1.jhu.edu	128.220.251.50	ΗΠΑ
planetlab2.jhu.edu	128.220.251.52	ΗΠΑ
planetab1.cnds.jhu.edu	128.220.231.2	ΗΠΑ
planetlab3.cnds.jhu.edu	128.220.231.4	ΗΠΑ
planetlab4.cnds.jhu.edu	128.220.231.5	ΗΠΑ
planetlab3.hiit.fi	193.167.187.185	Φινλανδία
planetlab4.hiit.fi	193.167.187.186	Φινλανδία
planetlab4.csail.mit.edu	128.31.1.14	ΗΠΑ
planetlab6.csail.mit.edu	128.31.1.16	ΗΠΑ
planetlab1.tamu.edu	165.91.55.8	ΗΠΑ

planetlab2.tamu.edu	165.91.55.9	ΗΠΑ
planetlab1.rd.tut.fi	193.166.167.4	Φινλανδία
planetlab2.rd.tut.fi	193.166.167.5	Φινλανδία
planetlab-03.cs.princeton.edu	128.112.139.97	ΗΠΑ
planetlab-04.cs.princeton.edu	128.112.139.18	ΗΠΑ
planetlab2.aston.ac.uk	134.151.255.181	Ηνωμένο Βασίλειο
plab4.ple.silweb.pl	83.230.127.124	Πολωνία
plab2.ple.silweb.pl	83.230.127.122	Πολωνία
planet-lab1.cs.unibas.ch	192.43.193.71	Ελβετία
ple2.tu.koszalin.pl	62.108.171.76	Πολωνία
prometeusz.we.po.opole.pl	217.173.198.154	Πολωνία
plonk.cs.uwaterloo.ca	129.97.74.14	Καναδά
planetlab1.c3sl.ufpr.br	200.17.202.194	Πορτογαλία
roam2.cs.ou.edu	129.15.78.31	ΗΠΑ
planetlab6.goto.info.waseda.ac.jp	133.9.81.166	Ιαπωνία
mercury.silicon-valley.ru	82.179.176.42	Ρωσία
venus.silicon-valley.ru	82.179.176.44	Ρωσία
roti.mimuw.edu.pl	193.0.109.23	Πολωνία
vn5.cse.wustl.eu	128.252.19.18	ΗΠΑ

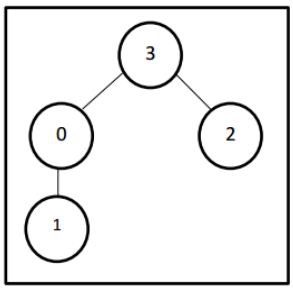
Σχήμα 5.1 Μηχανές του PlanetLab που χρησιμοποιήθηκαν στην πειραματική αξιολόγηση του αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένου με το κανάλι επικοινωνίας TCP

5.2 Σενάρια Προσομοίωσης

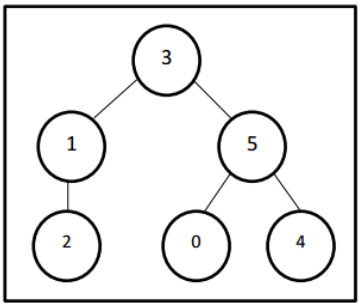
Με απώτερο στόχο τον έλεγχο της ορθότητας της εκτέλεσης των αλγορίθμων που υλοποιήθηκαν, καθώς επίσης και την αξιολόγηση της απόδοσής τους, προετοιμάστηκαν και εκτελέστηκαν επιλεγμένα σενάρια.

Για τον αλγόριθμο εκλογής αρχηγού με τοπολογία δέντρου, εκτελέσαμε τον αλγόριθμο που υλοποιήσαμε σε διάφορες τοπολογίες δέντρου, ανάλογα με τον αριθμό των διεργασιών. Συγκεκριμένα, για αριθμό διεργασιών 4, 6, 8, 12 και 18, εφαρμόσαμε τις τοπολογίες που φαίνονται στα Σχήματα 5.2, 5.3, 5.4, 5.5 και 5.6, αντίστοιχα. Όλες οι εκτελέσεις του συγκεκριμένου αλγορίθμου έγιναν για n επαναλήψεις όπου n ο αριθμός των διεργασιών, με ρυθμό εξέλιξης της τροχιάς ίσο με 1 και με παράμετρο t1 = 100, για να μην σταματήσει η

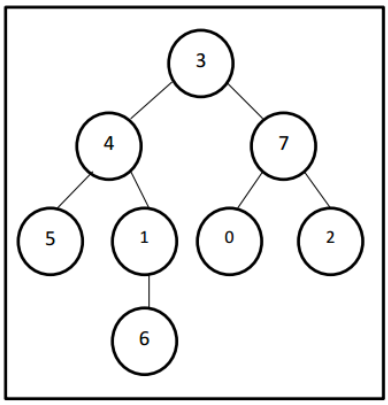
εκτέλεση του αλγορίθμου χωρίς να εκλεγεί κάποια διεργασία αρχηγός.



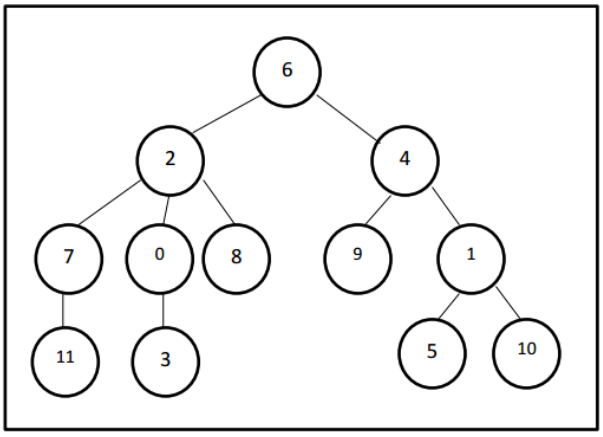
Σχήμα 5.2 : Αλγόριθμος με Τοπολογία Δέντρου για n = 4



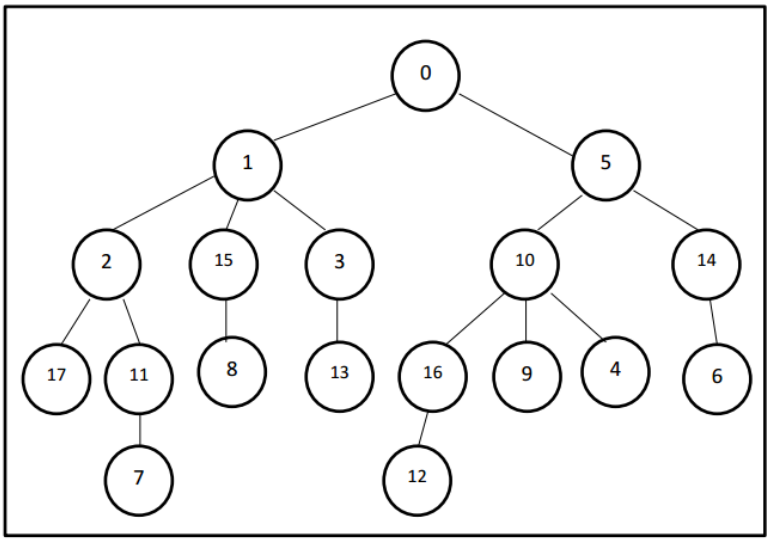
Σχήμα 5.3 : Αλγόριθμος με Τοπολογία Δέντρου για n = 6



Σχήμα 5.4 : Αλγόριθμος με Τοπολογία Δέντρου για n = 8

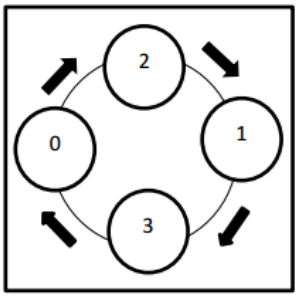


Σχήμα 5.5 : Αλγόριθμος με Τοπολογία Δέντρου για n = 12



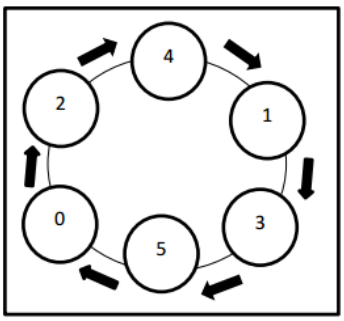
Σχήμα 5.6 : Αλγόριθμος με Τοπολογία Δέντρου για n = 18

Για τον αλγόριθμο εκλογής αρχηγού με τοπολογία δακτυλίου, εκτελέσαμε τον αλγόριθμο που υλοποιήσαμε με διαφορετικό αριθμό διεργασιών. Συγκεκριμένα, για αριθμό διεργασιών 4, 6, 8, 12 και 18, εφαρμόσαμε τις τοπολογίες δακτυλίου που φαίνονται στα Σχήματα 5.7, 5.8, 5.9, 5.10 και 5.11, αντίστοιχα. Επίσης, εκτελέσαμε τον αλγόριθμο για σταθερό αριθμό διεργασιών αλλά με διαφορετική διάταξη πάνω στο δακτύλιο, για να μελετήσουμε πως επηρεάζει η διάταξη των διεργασιών την χρονική και επικοινωνιακή πολυπλοκότητα του αλγορίθμου. Τα Σχήματα 5.7, 5.12 και 5.13 παρουσιάζουν την μέση, την καλύτερη και την χειρότερη εκτέλεση του αλγορίθμου με τοπολογία δακτυλίου με αριθμό διεργασιών 4, αντίστοιχα. Στην μέση εκτέλεση του αλγορίθμου, οι διεργασίες κατατάσσονται με τυχαίο τρόπο στο δακτύλιο ενώ στην καλύτερη και χειρότερη εκτέλεση, οι διεργασίες κατατάσσονται με φθίνουσα και αύξουσα σειρά, αντίστοιχα. Όλες οι εκτελέσεις του αλγορίθμου με τοπολογία δακτυλίου, έγιναν για n^2 επαναλήψεις, όπου n ο αριθμός των διεργασιών, με ρυθμό εξέλιξης της τροχιάς ίσο με 1 και με παράμετρο t1 = 100, για να μην σταματήσει η εκτέλεση του αλγορίθμου χωρίς να εκλεγεί κάποια διεργασία αρχηγός.

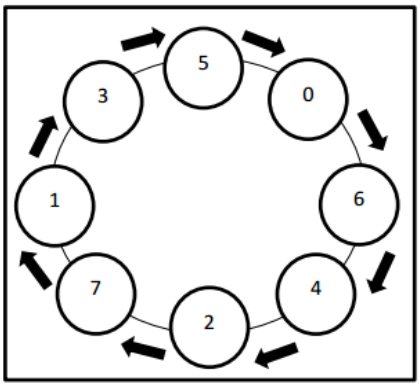


Σχήμα 5.7 : Αλγόριθμος με Τοπολογία Δακτυλίου για n = 4

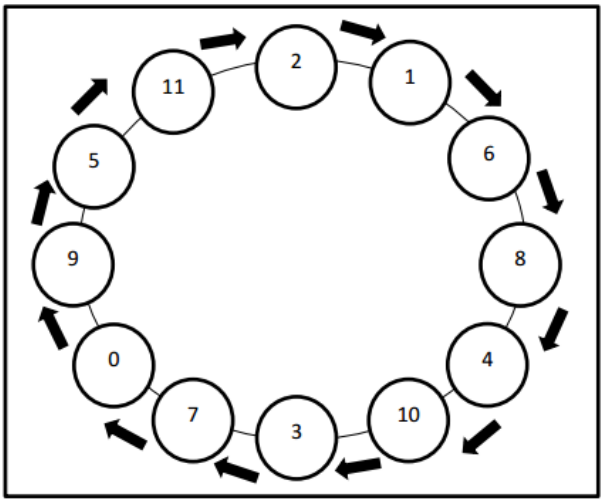
(Μέση Περίπτωση Εκτέλεσης)



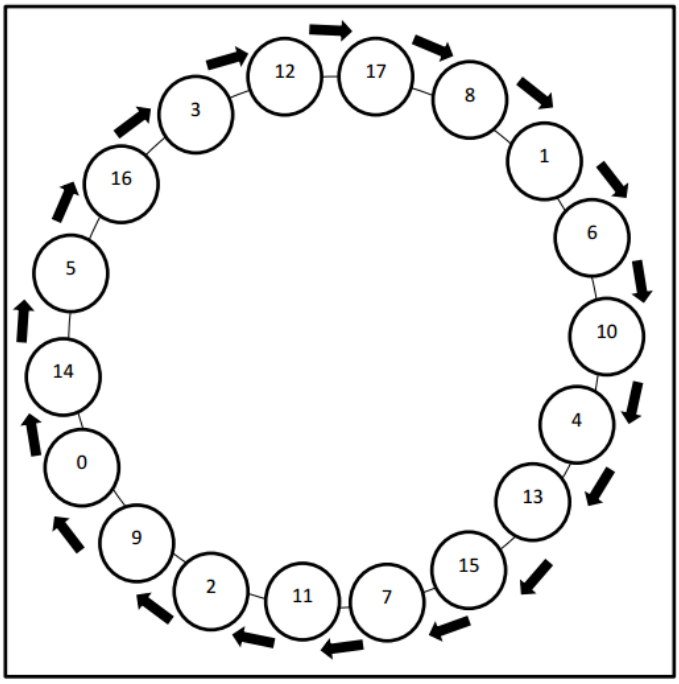
Σχήμα 5.8 : Αλγόριθμος με Τοπολογία Δακτυλίου για n = 6



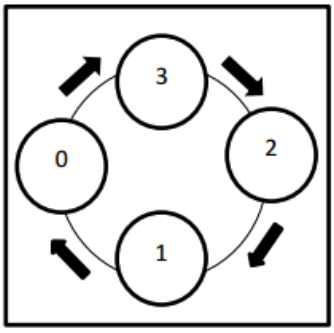
Σχήμα 5.9 : Αλγόριθμος με Τοπολογία Δακτυλίου για n = 8



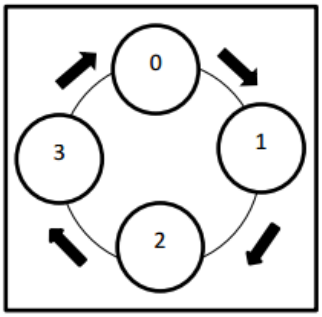
Σχήμα 5.10 : Αλγόριθμος με Τοπολογία Δακτυλίου για n = 12



Σχήμα 5.11 : Αλγόριθμος με Τοπολογία Δακτυλίου για n = 18



Σχήμα 5.12 : Αλγόριθμος με Τοπολογία Δακτυλίου για n = 4
(Καλύτερη Περίπτωση Εκτέλεσης)



Σχήμα 5.13 : Αλγόριθμος με Τοπολογία Δακτυλίου για n = 4
(Χειρότερη Περίπτωση Εκτέλεσης)

Για τον αλγόριθμο εκλογής αρχηγού με τοπολογία ισχυρά συνδεδεμένου γράφου και τις δύο εκδοχές της τροποποιημένης μορφής του, εφαρμόσαμε τα ίδια σενάρια για να ελέγξουμε πως αντιδρούν όταν καταρρέουν και επανέρχονται κόμβοι στο σύστημα. Εφαρμόσαμε τα ακόλουθα σενάρια :

- Σενάριο 1 : Στο σενάριο αυτό, δεν προσομοιώσαμε εσκεμμένα σενάρια για να παρατηρήσουμε την γενική αντίδραση του αλγορίθμου.
- Σενάριο 2 : Στο σενάριο αυτό, κατά την εκκίνηση του αλγορίθμου όλες οι διεργασίες είναι ενεργές. Μετά από 10 κύκλους επανάληψης (clock = 10) καταρρέουμε τη διεργασία με το μεγαλύτερο αναγνωριστικό στο σύστημα, δηλαδή τη διεργασία που θα πρέπει να είναι η διεργασία αρχηγός, χωρίς να την επαναφέρουμε.
- Σενάριο 3 : Στο σενάριο αυτό, κατά την εκκίνηση του αλγορίθμου όλες οι διεργασίες είναι ενεργές εκτός από τη διεργασία αρχηγό, δηλαδή τη διεργασία με το μεγαλύτερο αναγνωριστικό στο σύστημα. Μετά από 10 κύκλους επανάληψης επαναφέρουμε τη συγκεκριμένη διεργασία στο σύστημα.
- Σενάριο 4 : Στο σενάριο αυτό, κατά την εκκίνηση του αλγορίθμου, όλες οι διεργασίες είναι ενεργές εκτός από τη διεργασία με το μικρότερο αναγνωριστικό στο σύστημα, δηλαδή τη διεργασία με αναγνωριστικό 0. Μετά από 10 κύκλους επανάληψης επαναφέρουμε τη συγκεκριμένη διεργασία στο σύστημα.

Με τα πιο πάνω σενάρια επιθυμούμε να ελέγξουμε πως συμπεριφέρονται οι αλγόριθμοι όταν καταρρέει η διεργασία με το μεγαλύτερο αναγνωριστικό, επανέρχεται στο σύστημα η διεργασία με το μεγαλύτερο αναγνωριστικό και πως συμπεριφέρεται ο κάθε αλγόριθμος όταν καταρρέει και επανέρχεται η διεργασία με το μικρότερο αναγνωριστικό στο σύστημα. Συγκεκριμένα να μελετήσουμε πως επηρεάζεται ο χρόνος που χρειάζεται για την εκλογή αρχηγού και ο αριθμό των μηνυμάτων που στέλνονται κατά τη διάρκεια εκλογής αρχηγού. Να σημειώσουμε πως οι πιθανότητες κατάρρευσης και επαναφοράς κάποιας διεργασίας στο σύστημα είναι πιθανοτικές και υλοποιούνται με τυχαίους αριθμούς που επιλέγονται κατά τη διάρκεια εκτέλεσης των αντίστοιχων ενεργειών. Αυτό επηρεάζει τις εκτελέσεις των σεναρίων των αλγορίθμων, γιατί δεν γνωρίζουμε πότε θα καταρρεύσει ή επανέλθει μια διεργασία στο σύστημα. Για την αξιολόγηση του αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου που υλοποιήθηκε με το κανάλι επικοινωνίας TCP εκτελέστηκε και ένα $5^{ο}$ σενάριο. Στο σενάριο αυτό, καταρρέουμε την διεργασία με το μεγαλύτερο αναγνωριστικό στο σύστημα μετά από 10 κύκλους επανάληψης, εάν φυσικά είναι ενεργή, και μετράμε το χρόνο που χρειάζεται να εκλεγεί νέα διεργασία αρχηγός μετά από αυτήν την κατάρρευση. Όλες οι εκτελέσεις των αλγορίθμων με τοπολογία ισχυρά συνδεδεμένων γράφων, έγιναν για n^2 επαναλήψεις, όπου n ο αριθμός των διεργασιών, με ρυθμό εξέλιξης της τροχιάς ίσο με 1 και

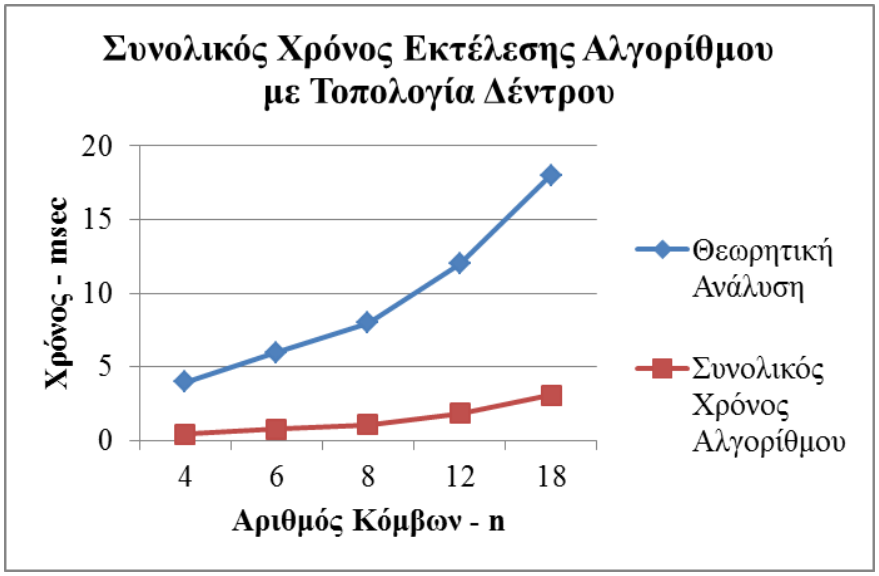
με τις εξής παραμέτρους : το χρονικό διάστημα στο οποίο θα πρέπει να ολοκληρωθεί η διαδικασία εκλογής αρχηγού ίσο με 20 κύκλους επανάληψης (timeForElection = 20), το χρονικό διάστημα στο οποίο θα πρέπει να ανακοινωθεί το αναγνωριστικό της διεργασίας αρχηγού στη διεργασία που ξεκινάει τη διαδικασία εκλογής αρχηγού ίσο με 20 κύκλους επανάληψης (timeForAnnouncementLeader = 20), το χρονικό διάστημα στο οποίο μια διεργασία θα ελέγχει εάν η διεργασία αρχηγός είναι ζωντανή ίσο με 5 κύκλους επανάληψης (timeForCheckleader = 5), την πιθανότητα που αντιστοιχεί στην πιθανότητα μια διεργασία να καταρρεύσει ίση με 2 (possibilityToFail = 2) και την πιθανότητα που αντιστοιχεί στην πιθανότητα μια διεργασία που έχει καταρρεύσει να επανέλθει ίση με 5 (possibilityToRecover = 5). Στις δύο εκδοχές του τροποποιημένου αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου ορίστηκαν επίσης οι εξής παράμετροι : το χρονικό διάστημα στο οποίο το Election Commission θα ελέγχει εάν η διεργασία αρχηγός είναι ζωντανή ίσο με 5 κύκλους επανάληψης (timeForVerifyLeader = 5) και το χρονικό διάστημα στο οποίο το Election Commission θα ελέγχει εάν μια διεργασία του συστήματος είναι ζωντανή ίσο με 5 κύκλους επανάληψης (timeForReplyAliveProcess = 5). Για τον αλγόριθμο που υλοποιήθηκε στο TCP και εκτελέστηκε στο PlanetLab, ορίζαμε : τις παραμέτρους n1, n2, n3 και n4 με τη διεύθυνση IP της μηχανής στην οποία θα έτρεχε μια διεργασία, τον αριθμό θύρας στον οποίο θα γίνει η σύνδεση ίσο με 41903 (port = 41903), το χρονικό διάστημα στο οποίο θα πρέπει να δημιουργηθεί η σύνδεση ίσο με 2000 (timeout = 2000) και το αναγνωριστικό της διεργασίας (rank) ίσο με τον τρέχοντα αριθμό διεργασίας που δημιουργούμε.

5.3 Αποτελέσματα

Τα τελικά αποτελέσματα που συλλέχθηκαν από τις πολλαπλές εκτελέσεις των αλγορίθμων εκλογής που υλοποιήθηκαν, φαίνονται στο Παράρτημα Δ.

Στα Σχήματα 5.14 και 5.15, παρουσιάζεται ο συνολικός χρόνος εκτέλεσης και ο συνολικός αριθμός μηνυμάτων του αλγορίθμου με τοπολογία δέντρου για την εκλογή αρχηγού. Όπως παρατηρείται, ο συνολικός χρόνος εκτέλεσης του αλγορίθμου αυξάνεται γραμμικά με τον αριθμό των κόμβων, όπως επίσης και ο συνολικός αριθμός των μηνυμάτων που ανταλλάσσονται μεταξύ των κόμβων κατά τη διαδικασία εκλογής αρχηγού. Όπως αναμενόταν, η τοπολογία αυτή παρουσίασε καλύτερα αποτελέσματα στο χρόνο εκτέλεσης από τη θεωρητική ανάλυση, αφού στη θεωρητική ανάλυση υπολογίζουμε τη χειρότερη περίπτωση εκτέλεσης του αλγορίθμου. Επίσης, όπως αναμενόταν, ο συνολικός αριθμός των

μηνυμάτων συνάδει με την θεωρητική επικοινωνιακή πολυπλοκότητα του αλγορίθμου, κάτι που επαληθεύει την ορθότητα της υλοποίησης του συγκεκριμένου αλγορίθμου. Συγκεκριμένα, για την εκλογή αρχηγού στέλνονται συνολικά 2n-2 μηνύματα, όπου n ο αριθμός των διεργασιών.

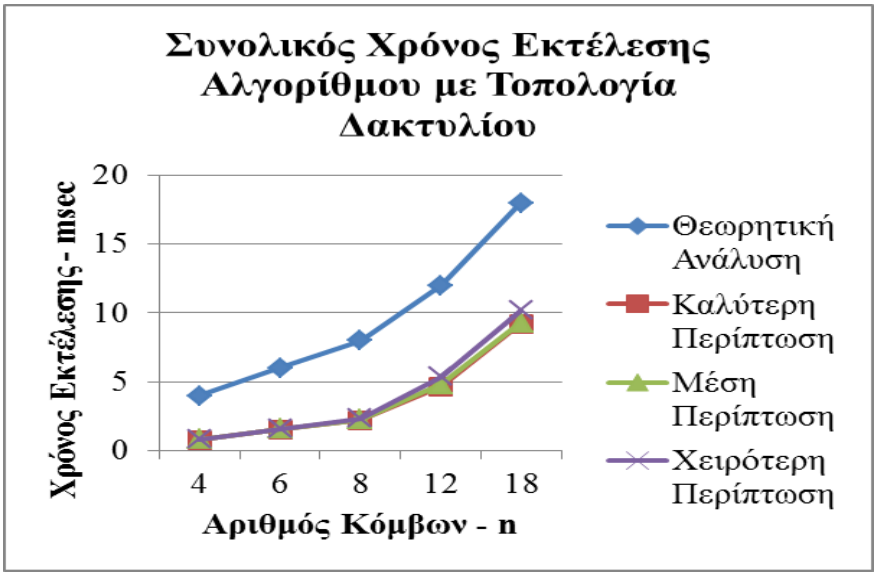


Σχήμα 5.14 : Συνολικός Χρόνος Εκτέλεσης Αλγορίθμου με Τοπολογία Δέντρου

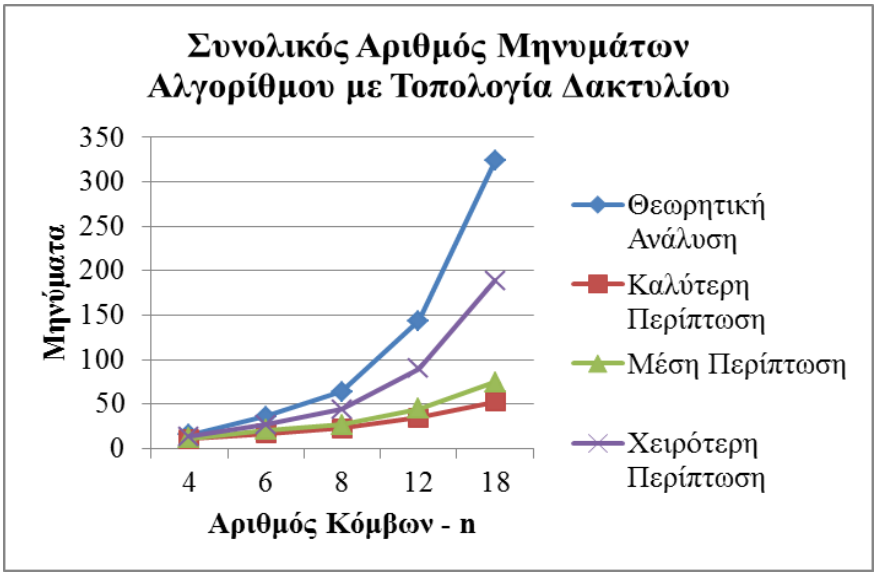
Σχήμα 5.15 : Συνολικός Αριθμός Μηνυμάτων Αλγορίθμου με Τοπολογία Δέντρου

Στα Σχήματα 5.16 και 5.17, παρουσιάζεται ο συνολικός χρόνος εκτέλεσης και ο συνολικός αριθμός μηνυμάτων του αλγορίθμου με τοπολογία δακτυλίου για την εκλογή αρχηγού στην καλύτερη, μέση και χειρότερη περίπτωση εκτέλεσής τους. Όπως παρατηρείται, ο συνολικός χρόνος εκτέλεσης του αλγορίθμου αυξάνεται γραμμικά με τον αριθμό των κόμβων και με παρόμοιο τρόπο και ο συνολικός αριθμός των μηνυμάτων που ανταλλάσσονται μεταξύ των

κόμβων κατά τη διαδικασία εκλογής αρχηγού. Παρατηρούμε πως ο χρόνος που χρειάζεται για να εκλεγεί μια διεργασία αρχηγός είναι ο ίδιος σχεδόν σε όλες τις περιπτώσεις εκτέλεσης του αλγορίθμου της συγκεκριμένης τοπολογίας. Αυτό ήταν και το αποτέλεσμα που αναμενόταν, γιατί σε οποιαδήποτε διάταξη των διεργασιών πάνω στο δακτύλιο, πάντα η διεργασία που θα εκλεγεί αρχηγός είναι η διεργασία που θα ενημερωθεί τελευταία για το αναγνωριστικό του αρχηγού, δηλαδή θα ενημερωθεί για την εκλογή της, τερματίζοντας έτσι την εκτέλεση του αλγορίθμου. Αντιθέτως, όπως παρατηρούμε στο Σχήμα 5.17, ο συνολικός αριθμός που ανταλλάσσονται μεταξύ των διεργασιών διαφέρει ανάλογα με την διάταξη των διεργασιών πάνω στο δακτύλιο. Συγκεκριμένα τα λιγότερα μηνύματα στέλνονται όταν οι διεργασίες είναι διατεταγμένες πάνω στο δακτύλιο σε φθίνουσα σειρά , με δεξιόστροφη διεύθυνση, δηλαδή στην καλύτερη περίπτωση εκτέλεσης του αλγορίθμου. Τα περισσότερα μηνύματα στέλνονται όταν έχουμε την διάταξη των διεργασιών στο δακτύλιο με αύξουσα σειρά, με δεξιόστροφη διεύθυνση, δηλαδή στην χειρότερη περίπτωση εκτέλεσης του αλγορίθμου. Αυτό είναι προφανές, γιατί αφού απαιτούνται n γύροι για την εκλογή αρχηγού, στην καλύτερη περίπτωση μόνο μια διεργασία σε κάθε γύρο θα στέλνει ένα μήνυμα και θα σταλούν συνολικά n μηνύματα. Ενώ στην χειρότερη περίπτωση κάθε διεργασία θα στέλνει μήνυμα σε κάθε γύρο και θα έχουμε συνολικά n^2 μηνύματα, που αποτελούν και την θεωρητική επικοινωνιακή πολυπλοκότητα του αλγορίθμου. Οι εκτελέσεις του αλγορίθμου στη μέση περίπτωση, όπως παρατηρείται και από το σχήμα, δίνουν συνολικό αριθμό μηνυμάτων μεγαλύτερο από αυτόν της καλύτερης περίπτωσης και μικρότερο από αυτόν της χειρότερης περίπτωσης.

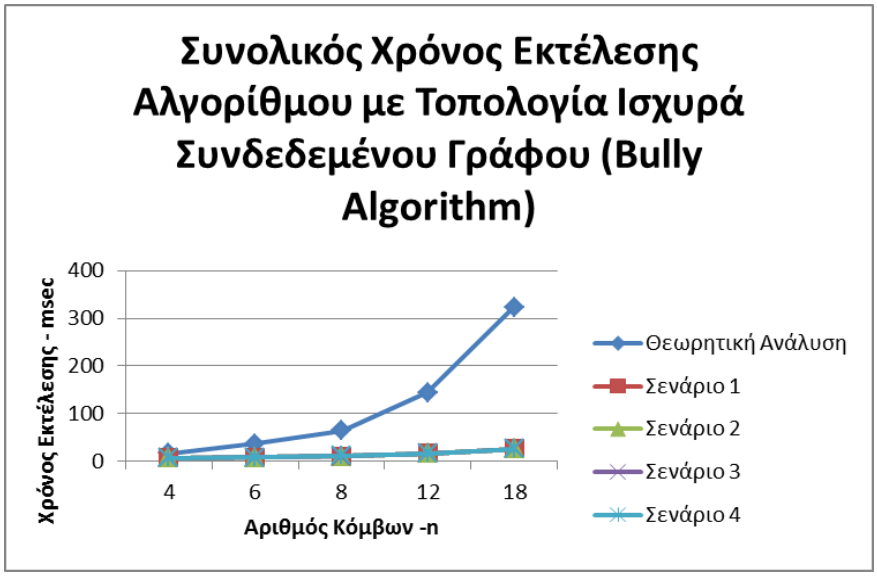


Σχήμα 5.16 : Συνολικός Χρόνος Εκτέλεσης Αλγορίθμου με Τοπολογία Δακτυλίου

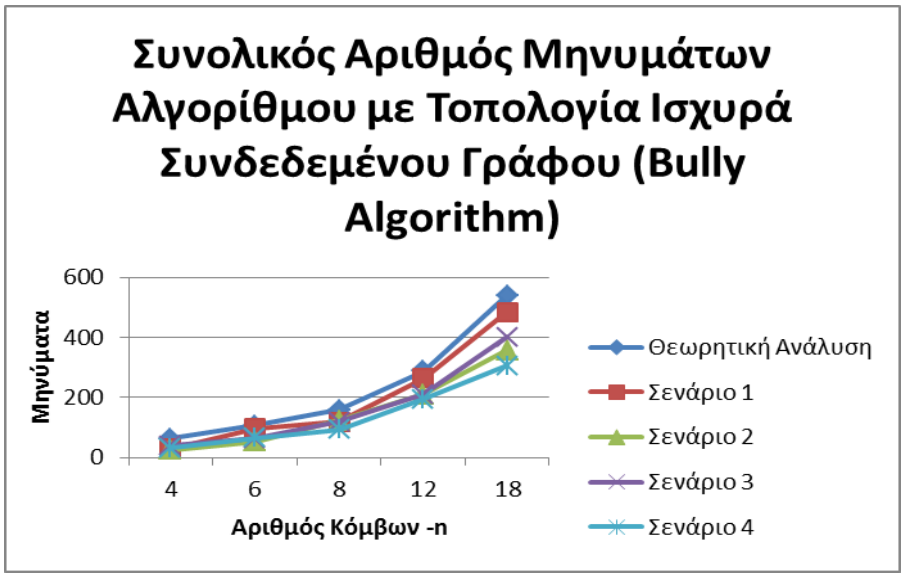


Σχήμα 5.17 : Συνολικός Αριθμός Μηνυμάτων Αλγορίθμου με Τοπολογία Δακτυλίου

Στα Σχήματα 5.18 και 5.19, παρουσιάζεται ο συνολικός χρόνος εκτέλεσης και ο συνολικός αριθμός μηνυμάτων του αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένου με το κανάλι επικοινωνίας MPI, για την εκλογή αρχηγού. Παρατηρούμε ότι ο χρόνος εκτέλεσης των σεναρίων είναι σχεδόν ίδιος και σαφώς μικρότερος από τον θεωρητικό χρόνο εκτέλεσης του συγκεκριμένου αλγορίθμου που είναι της τάξης $O(n^2)$. Αυξάνοντας των αριθμών των κόμβων του συστήματος αυξάνεται γραμμικά ο συνολικός χρόνος εκτέλεσης του αλγορίθμου, καθώς επίσης και ο αριθμός των μηνυμάτων που ανταλλάσσονται κατά τη διάρκεια εκλογής αρχηγού. Αυτό είναι αναμενόμενο αφού κάθε διεργασία θα πρέπει να επικοινωνεί με περισσότερες άλλες διεργασίες ανταλλάσσοντας μηνύματα. Όπως παρατηρούμε από την γραφική παράσταση του συνολικού αριθμού μηνυμάτων, τα λιγότερα μηνύματα αποστέλλονται στο σενάριο 4, γιατί όταν επιστρέφει πίσω στο σύστημα η διεργασία με το μικρότερο αναγνωριστικό και ξεκινήσει τη διαδικασία εκλογής αρχηγού, θα ενημερωθεί από τη διεργασία αρχηγό (διεργασία με το μεγαλύτερο αναγνωριστικό) για την εκλογή της ως αρχηγό. Επίσης, στο σενάριο 3 αποστέλλονται λιγότερα μηνύματα από τα σενάριο 2 για τον εξής λόγο : με την επιστροφή της στο σύστημα, η διεργασία με το μεγαλύτερο αναγνωριστικό, διαπιστώνει πως είναι η διεργασία που πρέπει να εκλεγεί αρχηγός και έτσι ενημερώνει όλες τις υπόλοιπες για την εκλογή της αυτή. Τα παραπλήσια αποτελέσματα των σεναρίων του αλγορίθμου οφείλονται σε μεγάλο βαθμό στην τυχαιότητα με την οποία καταρρέει και επανέρχεται κάθε διεργασία του συστήματος. Στις εκτελέσεις μας, επιλέξαμε μικρές τιμές για τις πιθανότητες κατάρρευσης και επαναφοράς.



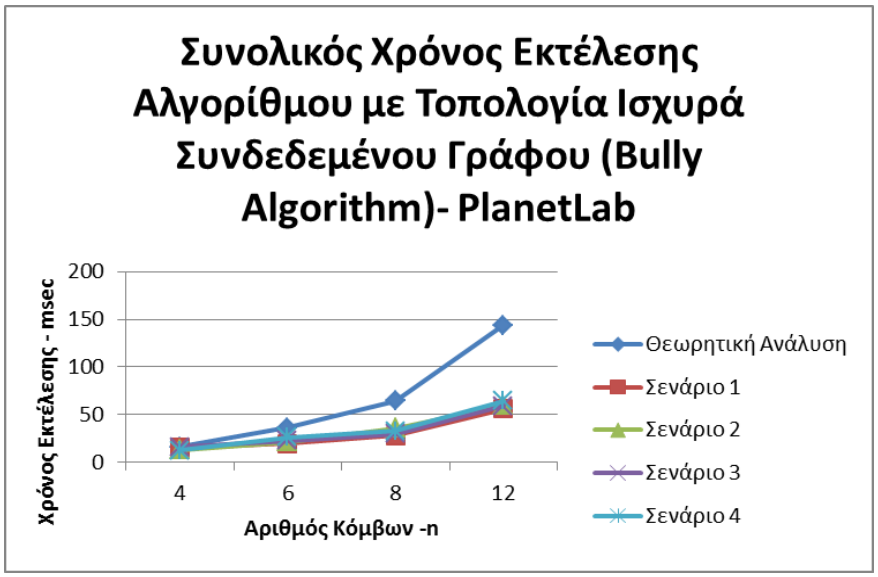
Σχήμα 5.18 : Συνολικός Χρόνος Εκτέλεσης Αλγορίθμου με Τοπολογία Ισχυρά Συνδεδεμένου Γράφου, υλοποιημένο με το κανάλι επικοινωνίας MPI, σε clusters μηχανές



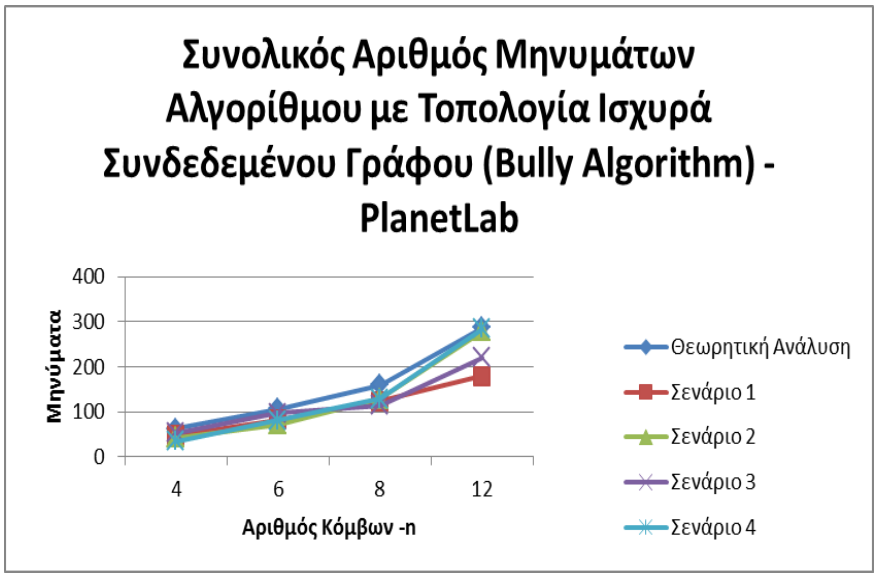
Σχήμα 5.19 : Συνολικός Αριθμός Μηνυμάτων Αλγορίθμου με Τοπολογία Ισχυρά Συνδεδεμένου Γράφου, υλοποιημένο με το κανάλι επικοινωνίας MPI, σε clusters μηχανές

Στα Σχήματα 5.20 και 5.21, παρουσιάζεται ο συνολικός χρόνος εκτέλεσης και ο συνολικός αριθμός μηνυμάτων του αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένου με το κανάλι επικοινωνίας TCP, για την εκλογή αρχηγού. Τα αποτελέσματα αυτού του αλγορίθμου πάρθηκαν τρέχοντάς τον στο κατανεμημένο σύστημα PlanetLab. Παρατηρώντας τις γραφικές παραστάσεις, διαπιστώνουμε πως παρουσιάζει τις ίδιες ακριβώς συμπεριφορές με τον αντίστοιχο αλγόριθμο που υλοποιήθηκε με το κανάλι επικοινωνίας MPI. Η βασική διαφορά είναι το γεγονός πως ο συνολικός χρόνος εκτέλεσής του είναι πολύ

μεγαλύτερος, ενώ ο συνολικός αριθμός μηνυμάτων είναι πολύ μικρότερος. Αυτό είναι αναμενόμενο, γιατί στο PlanetLab μπορεί να παρατηρηθεί συμφόρηση, κατάρρευση ή μη ανταπόκριση κόμβων, απώλεια ή καθυστέρηση παράδοσης μηνυμάτων που επηρεάζουν σημαντικά την εκτέλεση του αλγορίθμου. Όμως, σε γενικές γραμμές τα αποτελέσματα που πήραμε ήταν ικανοποιητικά αφού δεν ξεπερνούσαν την θεωρητική χρονική και επικοινωνιακή πολυπλοκότητα του αλγορίθμου.

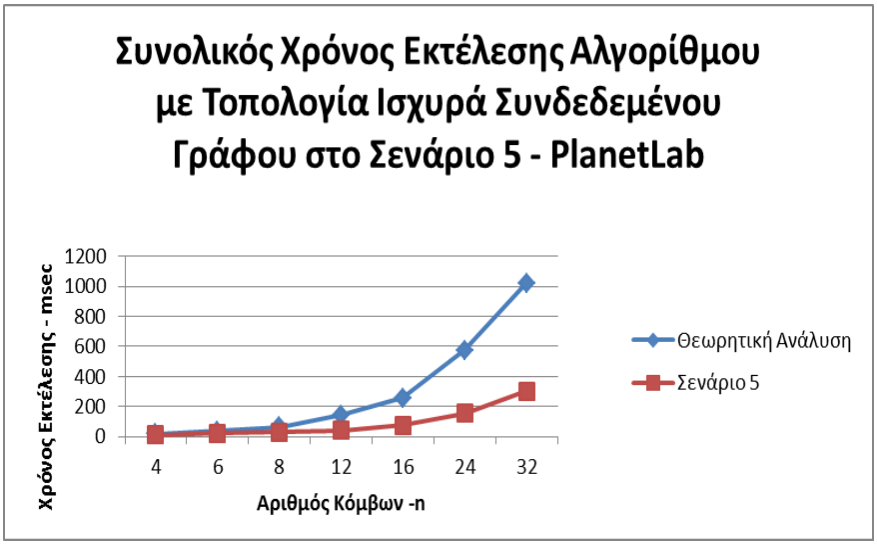


Σχήμα 5.20 : Συνολικός Χρόνος Εκτέλεσης Αλγορίθμου με Τοπολογία Ισχυρά Συνδεδεμένου Γράφου, υλοποιημένο με το κανάλι επικοινωνίας TCP, στο PlanetLab

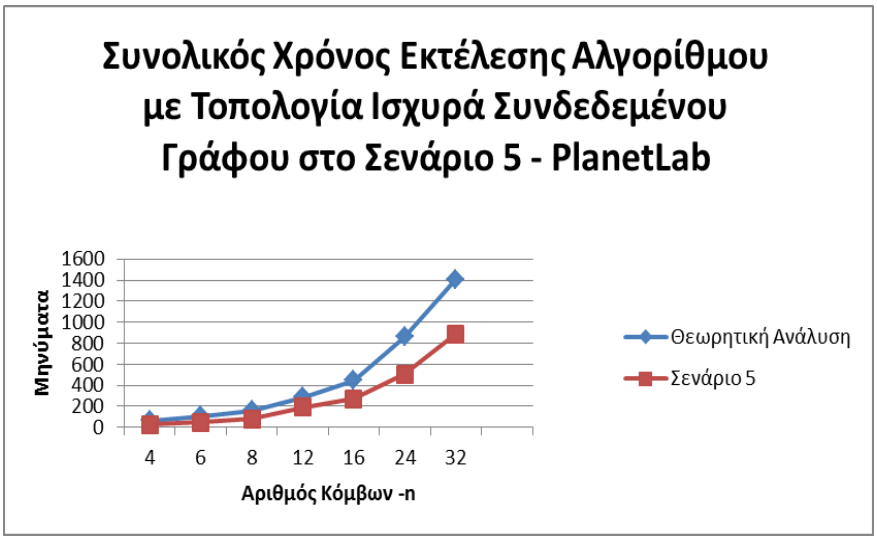


Σχήμα 5.21 : Συνολικός Αριθμός Μηνυμάτων Αλγορίθμου με Τοπολογία Ισχυρά Συνδεδεμένου Γράφου, υλοποιημένο με το κανάλι επικοινωνίας TCP, στο PlanetLab

Στα Σχήματα 5.22 και 5.23, παρουσιάζεται ο συνολικός χρόνος εκτέλεσης και ο συνολικός αριθμός μηνυμάτων του αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένου με το κανάλι επικοινωνίας TCP για το σενάριο 5, στο οποίο υπολογίζουμε τον χρόνο και τον αριθμό μηνυμάτων που χρειάζονται μέχρι να εκλεγεί νέος αρχηγός από τη στιγμή που καταρρέει η διεργασία αρχηγός. Το σενάριο αυτό εκτελέστηκε για διαφορετικό αριθμό διεργασιών, που έφτανε μέχρι και 32 διεργασίες. Όπως παρατηρούμε από τις αντίστοιχες γραφικές παραστάσεις, ο χρόνος μέχρι να εκλεγεί νέος αρχηγός, καθώς επίσης και ο συνολικός αριθμός που αποστέλλονται, εξαρτώνται σημαντικά από τον αριθμό των διεργασιών. Τρέχοντας τον συγκεκριμένο αλγόριθμο, παρατηρήσαμε πως για αριθμό διεργασιών μεγαλύτερο από 8, πολλές διεργασίες δεν ανταποκρίνονταν.

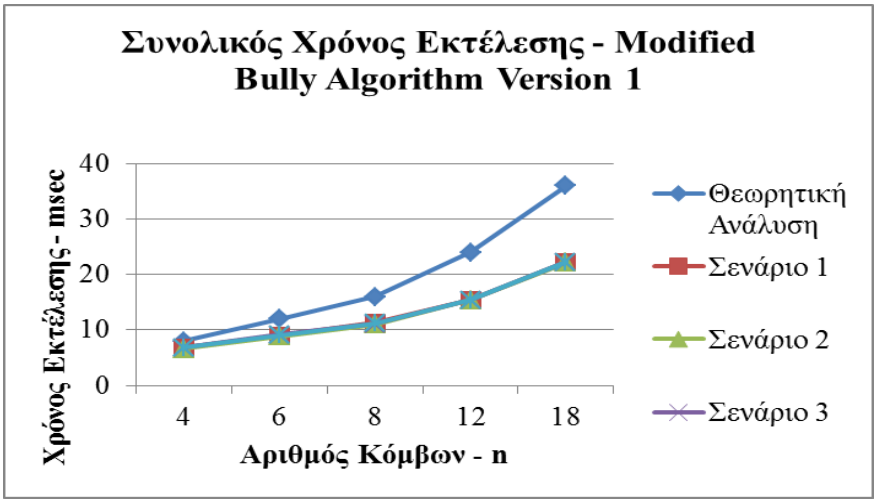


Σχήμα 5.22 : Συνολικός Χρόνος Εκτέλεσης Σεναρίου 5 του Αλγορίθμου με Τοπολογία Ισχυρά Συνδεδεμένου Γράφου, υλοποιημένο με το κανάλι επικοινωνίας TCP, στο PlanetLab

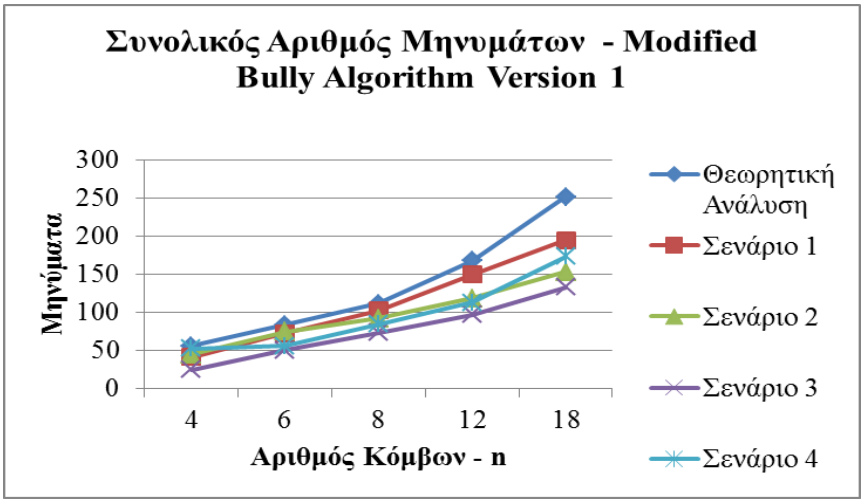


Σχήμα 5.23 : Συνολικός Αριθμός Μηνυμάτων Σεναρίου 5 του Αλγορίθμου με Τοπολογία Ισχυρά Συνδεδεμένου Γράφου, υλοποιημένο με το κανάλι επικοινωνίας TCP, στο PlanetLab

Στα Σχήματα 5.24 και 5.25, παρουσιάζεται ο συνολικός χρόνος εκτέλεσης και ο συνολικός αριθμός μηνυμάτων της πρώτης εκδοχής του τροποποιημένου αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου. Όπως παρατηρείται, ο συνολικός χρόνος εκτέλεσης του αλγορίθμου αυξάνεται γραμμικά με τον αριθμό των κόμβων, όπως επίσης και ο συνολικός αριθμός των μηνυμάτων που ανταλλάσσονται μεταξύ των κόμβων κατά τη διαδικασία εκλογής αρχηγού. Τα διάφορα σενάρια που εκτελέστηκαν, παρουσίασαν παραπλήσιο συνολικό χρόνο εκτέλεσης, με συνολικό χρόνο εκτέλεσης καλύτερο από το χρόνο της θεωρητική ανάλυσης. Επίσης, το ίδιο παρατηρείται και στον συνολικό αριθμό μηνυμάτων. Και πάλι, όπως και στον κανονικό αλγόριθμο με τοπολογία ισχυρά συνδεδεμένου γράφου λιγότερα μηνύματα στέλνονται στο σενάριο 3 όπου επιστρέφει η διεργασία με το μεγαλύτερο αναγνωριστικό στο σύστημα, εκλέγεται αρχηγός και όλες οι άλλες διεργασίες ενημερώνονται για το αναγνωριστικό του νέου αρχηγού.

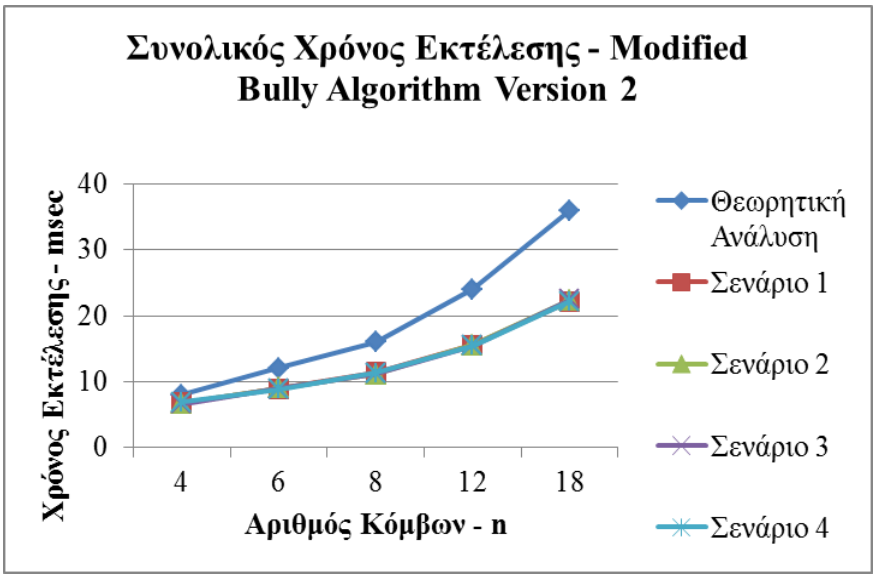


Σχήμα 5.24 : Συνολικός Χρόνος Εκτέλεσης Πρώτης Εκδοχής Τροποποιημένου Αλγορίθμου με Τοπολογία Ισχυρά Συνδεδεμένου Γράφου

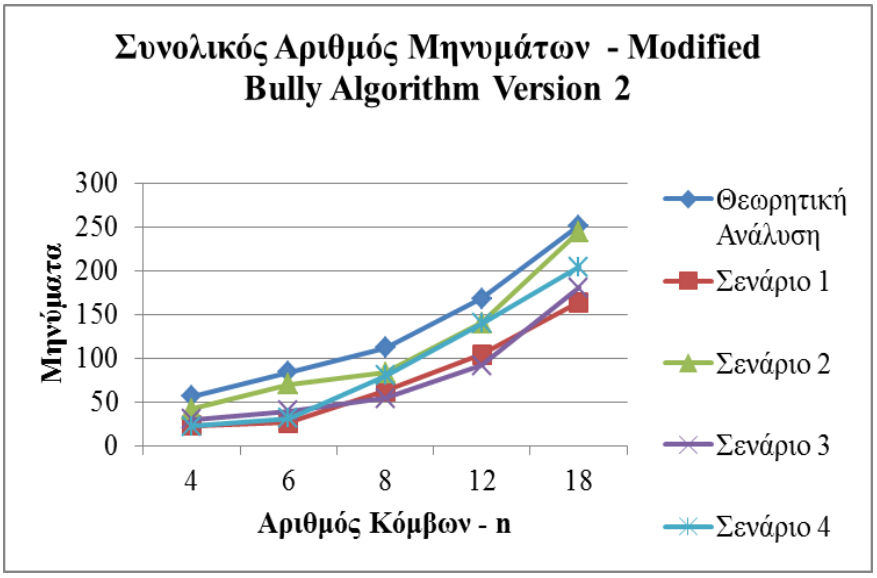


Σχήμα 5.25 : Συνολικός Αριθμός Μηνυμάτων Πρώτης Εκδοχής Τροποποιημένου Αλγορίθμου με Τοπολογία Ισχυρά Συνδεδεμένου Γράφου

Στα Σχήματα 5.26 και 5.27, παρουσιάζεται ο συνολικός χρόνος εκτέλεσης και ο συνολικός αριθμός μηνυμάτων της δεύτερης εκδοχής του τροποποιημένου αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου γράφου. Τα αποτελέσματα είναι αντίστοιχα με εκείνα της πρώτης εκδοχής του τροποποιημένου αλγορίθμου. Οι διαφορές τους σε χρόνο εκτέλεσης και αριθμό μηνυμάτων είναι ελάχιστες και εξαρτώνται από την κατάσταση των διεργασιών στο σύστημα.



Σχήμα 5.26 : Συνολικός Χρόνος Εκτέλεσης Δεύτερης Εκδοχής Τροποποιημένου Αλγορίθμου με Τοπολογία Ισχυρά Συνδεδεμένου Γράφου



Σχήμα 5.27 : Συνολικός Αριθμός Μηνυμάτων Δεύτερης Εκδοχής Τροποποιημένου Αλγορίθμου με Τοπολογία Ισχυρά Συνδεδεμένου Γράφου

Σε περίπτωση που η αμέσως προηγούμενη διεργασία από την διεργασία με το μεγαλύτερο αναγνωριστικό είναι ενεργή, τότε με την πρώτη εκδοχή του τροποποιημένου αλγορίθμου θα σταλούν λιγότερα μηνύματα και σε μικρό χρόνο θα εκλεγεί η νέα αρχηγός του συστήματος. Αντίθετα, με τη δεύτερη εκδοχή θα χρειαστεί περισσότερος χρόνος και θα σταλούν περισσότερα μηνύματα αφού το Election Commission θα πρέπει να επικοινωνήσει με όλες τις διεργασίες του συστήματος. Στην περίπτωση όμως που μόνο η διεργασία με το μικρότερο αναγνωριστικό είναι ενεργή στο σύστημα, τότε τον λιγότερο χρόνο για την εκλογή νέου αρχηγού θα τον δώσει η δεύτερη εκδοχή του τροποποιημένου αλγορίθμου αφού με την πρώτη, το Election Commission θα πρέπει να ελέγξει μία μία τις διεργασίες, ξεκινώντας από τη διεργασία με το μεγαλύτερο αναγνωριστικό και καταλήγοντας στη διεργασία με το μικρότερο αναγνωριστικό. Επομένως, καταλήγουμε στο συμπέρασμα πως τα αποτελέσματα των δύο εκδοχών του τροποποιημένου αλγορίθμου εξαρτώνται σημαντικά από το πόσες και ποιες διεργασίες είναι ενεργές στο σύστημα κάθε χρονική στιγμή.

Κεφάλαιο 6

Συμπεράσματα

6.1 Γενικά Συμπεράσματα

Στην παρούσα διπλωματική εργασία υλοποιήθηκαν και αξιολογήθηκαν διάφοροι αλγόριθμοι εκλογής αρχηγού. Η αξιολόγησή τους έγινε σε μια συστοιχία μηχανών σε τοπικό δίκτυο, εκτός από τον αλγόριθμο με τοπολογία ισχυρά συνδεδεμένου γράφου υλοποιημένου με το κανάλι επικοινωνίας TCP, ο οποίος αξιολογήθηκε και στο ρεαλιστικό και ασύγχρονο δίκτυο PlanetLab. Μέσω της εκτέλεσης των αλγορίθμων για διάφορα σενάρια και διάφορες τοπολογίες, παρατηρήθηκε πως οι αλγόριθμοι έχουν πάντα ως αποτέλεσμα την εκλογή ενός μοναδικού αρχηγού. Η υλοποίηση των αλγορίθμων έγινε όπως περιγράφεται στο [21], όπου οι αλγόριθμοι με τοπολογία δέντρου και δακτυλίου εκλέγουν ως αρχηγό τη διεργασία με το μικρότερο αναγνωριστικό στο σύστημα ενώ οι υπόλοιποι αλγόριθμοι εκλέγουν ως αρχηγό τη διεργασία με το μεγαλύτερο αναγνωριστικό.

Λαμβάνοντας υπόψη τα αποτελέσματα από την πειραματική αξιολόγηση των αλγορίθμων, οι εκτελέσεις των αλγορίθμων με τοπολογία δέντρου και δακτυλίου δίνουν σχετικά μικρό συνολικό χρόνο εκτέλεσης και συνολικό αριθμό μηνυμάτων. Αυτό συμβαίνει επειδή στους συγκεκριμένους αλγόριθμους δεν υπάρχουν σφάλματα και καταρρεύσεις κόμβων. Ο αλγόριθμος με τοπολογία ισχυρά συνδεδεμένου γράφου που είναι υλοποιημένος με το κανάλι επικοινωνίας TCP, παρουσιάζει μεγαλύτερο χρόνο εκτέλεσης από τον αντίστοιχο αλγόριθμο

που είναι υλοποιημένος με το κανάλι επικοινωνίας MPI σε τοπικό δίκτυο. Αυτό οφείλεται στο ρεαλιστικό δίκτυο PlanetLab, στο οποίο παρατηρείται συμφόρηση, κατάρρευση, μη ανταπόκριση και επαναφορά κόμβων, καθώς επίσης και απώλεια μηνυμάτων. Οι δύο εκδοχές του τροποποιημένου αλγορίθμου με τοπολογία ισχυρά συνδεδεμένου αλγορίθμου, παρουσιάζουν σχεδόν τον ίδιο χρόνο εκτέλεσης αλλά διαφέρουν ως προς τα μηνύματα που αποστέλλουν, ανάλογα με την κατάσταση των διεργασιών στο δίκτυο. Αυτό προκύπτει από το ότι σε περίπτωση που αντιληφθεί η διεργασία Election Commission την κατάρρευση της διεργασίας αρχηγού, ενεργούν διαφορετικά για την εκλογή νέας διεργασίας αρχηγού.

Επίσης, όπως προκύπτει από τα αποτελέσματα των εκτελέσεων των αλγορίθμων που υλοποιήθηκαν, τόσο ο χρόνος που απαιτείται για την εκλογή αρχηγού όσο και ο αριθμός των μηνυμάτων που αποστέλλονται στην διαδικασία εκλογής αρχηγού, αυξάνεται καθώς αυξάνεται ο αριθμός των διεργασιών στο σύστημα. Αυτό συμβαίνει γιατί καθώς αυξάνονται οι διεργασίες, η κάθε διεργασία χρειάζεται να επικοινωνήσει με περισσότερες διεργασίες στη διαδικασία εκλογής αρχηγού, με αποτέλεσμα την παραλαβή και προώθηση περισσότερων μηνυμάτων. Ως αποτέλεσμα, δημιουργείται επιπλέον καθυστέρηση στο δίκτυο οδηγώντας στην αύξηση του χρόνου εκτέλεσης και του αριθμού των μηνυμάτων που αποστέλλονται.

Από την αξιολόγηση των αλγορίθμων σε μια συστοιχία μηχανών, καταλήξαμε στο συμπέρασμα πως ο χρόνος εκτέλεσης των αλγορίθμων επηρεάζεται από τον αριθμό των διεργασιών που κατανέμουμε σε κάθε μηχανή. Σε περίπτωση που αυξήσουμε τον αριθμό των διεργασιών ανά μηχανή παρατηρείται επιπλέον αύξηση στον συνολικό χρόνο εκτέλεσης.

6.2 Προβλήματα και Παραδοχές

Κατά τη διάρκεια της υλοποίησης των αλγορίθμων εκλογής αρχηγού, εντοπίστηκαν κάποιες αδυναμίες σχετικά με το πακέτο εργαλείων TIOA και το εργαλείο Tempo, οι οποίες είναι οι εξής :

1. Η εντολή choose .. where δεν λειτουργεί όπως πρέπει και επιστρέφει αυθαίρετους αριθμούς. Αυτό μας δημιουργούσε πρόβλημα όταν προσπαθούσαμε να επιλέγουμε τυχαίες τιμές σε συγκεκριμένο εύρος τιμών για την εκτέλεση των ενεργειών κατάρρευσης και επαναφοράς κόμβων στο σύστημα. Για το λόγο αυτό χρειάστηκε να υλοποιήσουμε τις τυχαίες αυτές τιμές στον κώδικα που παράχθηκε στην γλώσσα προγραμματισμού Java.
2. Η αποσφαλμάτωση της προδιαγραφής κάθε αλγορίθμου ήταν δύσκολη λόγω του ότι τα αποτελέσματα της εκτέλεσης του και τα μηνύματα που εμφανίζονταν στην οθόνη

ήταν πάρα πολλά, τα οποία δεν μπορούσαμε να αποθηκεύσουμε ούτε σε κάποιο αρχείο αλλά ούτε και να τα δούμε στην κονσόλα. Αυτό έκανε δύσκολο τον εντοπισμό τυχόν σφαλμάτων.

Επίσης, κατά τη διάρκεια της πειραματικής αξιολόγησης στο κατανεμημένο και ασύγχρονο δίκτυο PlanetLab, παρουσιάστηκαν συγκεκριμένα προβλήματα. Αρχικά, κατά τη διάρκεια της δημιουργίας του δημοσίου κλειδιού και την εισαγωγή μηχανών στο μέρισμά μας, παρατηρήθηκε καθυστέρηση στη μετάδοση του κλειδιού αυτού σε ορισμένες μηχανές, με αποτέλεσμα να μην επιτρέπεται η χρησιμοποίησή τους. Λόγω του ότι οι μηχανές στο PlanetLab μπορούν να αποτυγχάνουν και να επανέρχονται ανά πάσα χρονική στιγμή, παρατηρήθηκε πως ορισμένες μηχανές χρειάζονταν μέρες για να γίνουν ξανά διαθέσιμες, καθυστερώντας τη αποπεράτωση των πειραμάτων. Επίσης, εκτός από τη μη διαθεσιμότητα των μηχανών, παρατηρήθηκε πως κάποιες μηχανές δεν ήταν προσβάσιμες για διάφορους λόγους. Για κάθε μηχανή του μερίσματος του χρήστη, κατά την πρώτη φορά που έχει πρόσβαση σε αυτήν, αποθηκεύεται το αναγνωριστικό της στο αρχείο known_hosts στον προσωπικό χώρο του χρήστη. Το αναγνωριστικό κάθε μηχανής αλλάζει συνεχώς για λόγους ασφαλείας και έτσι κάθε φορά που αλλάζει θα πρέπει να διαγράφεται από το αρχείο known_hosts το παλιό αναγνωριστικό, για να γίνει επιτυχώς η σύνδεση στη μηχανή. Επίσης, σε μερικές περιπτώσεις όπου κάποια μηχανή επανεκκινούσε, ήταν αδύνατη η σύνδεση με κάποια μηχανή για ορισμένο χρονικό διάστημα και εμφανιζόταν αντίστοιχο μήνυμα. Ακόμη ένα πρόβλημα που παρατηρήθηκε ήταν όταν ο αριθμός των διεργασιών στο σύστημα ήταν πολύ μεγάλος. Στην περίπτωση αυτή, ορισμένες διεργασίες δεν ανταποκρίνονταν. Αυτό πιθανώς να οφειλόταν στο γεγονός ότι η εκτέλεση του προγράμματος χρειαζόταν μεγάλο ποσοστό μνήμης της μηχανής, το οποίο μπορεί να μην ήταν διαθέσιμο εκείνη τη στιγμή.

6.3 Μελλοντική Εργασία

Σε αυτή τη Διπλωματική Εργασία έχουν μελετηθεί, υλοποιηθεί και αξιολογηθεί συγκεκριμένοι αλγόριθμοι εκλογής αρχηγού, τόσο σε συστοιχία τοπικού δικτύου όσο και στο κατανεμημένο δίκτυο PlanetLab. Για σκοπούς περαιτέρω αξιολόγησης της υλοποίησης μας θα μπορούσαν να διεξαχθούν επιπρόσθετα σενάρια με διαφορετικές τοπολογίες και διαφορετικό αριθμό διεργασιών. Από τους αλγόριθμους που μελετήθηκαν, μόνο ο αλγόριθμος με τοπολογία ισχυρά συνδεδεμένου γράφου έχει υλοποιηθεί με το κανάλι επικοινωνίας TCP, για να αξιολογηθεί στο PlanetLab. Ως μελλοντική εργασία, θα μπορούσαν να υλοποιηθούν και οι υπόλοιποι αλγόριθμοι μέσω του καναλιού TCP και να τους

αξιολογήσουμε και εκείνους στην πλατφόρμα του PlanetLab. Ακόμη, το Election Commission του τροποποιημένου αλγορίθμου εκλογής αρχηγού, στη δική μας υλοποίηση έχει μοντελοποιηθεί ως ένα ξεχωριστό αυτόματο σαν μια ειδική διεργασία. Θα μπορούσε να βελτιωθεί αυτή η υλοποίηση και να μοντελοποιηθούν όλα τα μέρη του Election Commission. Επίσης, θα μπορούσαν να υλοποιηθούν επιπρόσθετοι αλγόριθμοι εκλογής αρχηγού μέσω της γλώσσας TIOA και το εργαλείο Tempo και να συγκριθούν με τους αλγόριθμους που μελετήθηκαν και υλοποιήθηκαν στο παρόν έγγραφο. Τέλος, σε μεταγενέστερο στάδιο, θα μπορούσε να δημιουργηθεί κάποια μικρή εφαρμογή με κατάλληλη γραφική διεπαφή, η οποία να μοντελοποιεί κάποιον αλγόριθμο εκλογής αρχηγού και στην οποία ο χρήστης θα μπορεί να εφαρμόζει διάφορα σενάρια και να μελετά τη συμπεριφορά του.

6.4 Οφέλη από τη Διπλωματική Εργασία

Μέσα στα πλαίσια ολοκλήρωσης της διπλωματικής μου εργασίας, εξοικειώθηκα σε μεγάλο βαθμό με την ερευνητική μεθοδολογία και αποκόμισα αρκετές γνώσεις γύρω από διάφορους τομείς της πληροφορικής με τους οποίους δεν είχα την ευκαιρία να διερευνήσω κατά τη διάρκεια των προπτυχιακών μου σπουδών. Αρχικά, είχα την ευκαιρία να αποκομίσω αρκετές γνώσεις γύρω από τον τομέα των κατανεμημένων αλγορίθμων και συστημάτων. Επίσης, έμαθα για τη δομή και τη λειτουργία του μοντέλου αυτομάτων Εισόδου/Εξόδου (Input/Output Automata), αλλά και του μοντέλου χρονισμένων αυτομάτων Εισόδου/Εξόδου (Timed Input/Output Automata). Επιπρόσθετα, μου δόθηκε η δυνατότητα να μελετήσω, να κατανοήσω και να εξοικειωθώ με τη γλώσσα TIOA και το εργαλείο Tempo, υλοποιώντας διάφορα προγράμματα. Μέσα από το συγκεκριμένο θέμα, μελέτησα και κατανόησα συγκεκριμένους κατανεμημένους αλγορίθμους εκλογής αρχηγού τους οποίους είχα τη δυνατότητα να προδιαγράψω στο εργαλείο Tempo χρησιμοποιώντας τη γλώσσα TIOA. Με βοήθησε να κατανοήσω τον τρόπο λειτουργίας των δύο ειδών καναλιών επικοινωνίας, του MPI και του TCP, τα οποία χρησιμοποιήθηκαν στην υλοποίηση των αλγορίθμων για την επικοινωνία μεταξύ των διεργασιών του συστήματος. Τέλος, είχα την δυνατότητα να μάθω για την πλατφόρμα του PlanetLab, να εξοικειωθώ με τον τρόπο λειτουργίας της και να την χρησιμοποιήσω στην αξιολόγηση των αλγορίθμων που υλοποίησα.

Βιβλιογραφία

[1] H. Garcia-Molina. “Elections in a Distributed Computing System”, IEEE Transactions on Computers, vol. C-31, no.1, January 1982, pp 48-59.

[2] S. J. Garland. TIOA User Guide and Reference Manual. Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, September 30, 2005.

[3] S. J. Garland, N. A. Lynch, J. A. Tauber, M.Vaziri. IOA User Guide and Reference Manual. Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, August 14, 2013.

[4] D. K. Kaynar, N. Lynch, R. Segala, F. Vaandrager, “The Theory of Timed I/O Automata”, Morgan & Claypool Publishers, 2011.

[5] N. A. Lynch and M. R. Tuttle, “An Introduction to Input/Output Automata”, Massachusetts Institude of Technology, Cambridge, Mass. 02139, November 18, 1998, pp 4-24.

[6] N. A. Lynch, S. J. Garland, D. Kaynar, L. Michel, A. Shartsman. The Tempo Language User Guide and Reference Manual, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, October 24, 2011.

[7] P. M. Musial. Using Timed Input/Output Automata for Implementing Distrbuted Systems, Computer Science and Artificial Intelligence Laboratory, Mssachusetts Institute of Technology, 2011.

[8] L. Peterson, T. Roscoe. The design Principles of PlanetLab, PlanetLab Design Note PDN-04-021, June 2004.

[9] M. M. Rahman, A. Nahar, “Modified Bully Algorithm using Election Commission” (2010).

[10] IOA Homepage. Available at http://theory.lcs.mit.edu/tds/ioa/ .

[11] MPJ Express : An Implementation of MPI in Java Linux/UNIX/Mac User Guide, April 1, 2010. Available at http://mpj-express.org/docs/guides/linuxguide.pdf .

[12] MPJ Express Homepage. Available at http://mpj-express.org .

[13] PlanetLab Homepage. Available at http://www.planet-lab.org .

[14] PlanetLab User Manual. Available at http://www.planet-lab.org/doc/guides .

[15] Tempo toolkit Homepage. Available at http://www.veromondo.com .

[16] Veromondo, Inc. The TEMPO Simulator How-To, January 14, 2008. Available at http://www.veromondo.com .

[17] Veromondo, Inc. The TIOA Language Version 0.21, May 22, 2005. Available at http://www.veromondo.com .

[18] Veromondo, Inc. TEMPO Release Notes, July 18, 2007. Available at http://www.veromondo.com .

[19] Veromondo, Inc. The TEMPO User Interface, August 24, 2011. Available at http://www.veromondo.com .

[20] Χ. Γεωργίου (2012), Σημειώσεις Μαθήματος ΕΠΛ 601 – Κατανεμημένα Συστήματα, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου.

[21] Ι. Κ. Κάβουρας, Ι. Ζ. Μηλής, Γ. Β. Ξυλωμένος, Α. Α. Ρουκουνάκη. Κατανεμημένα Συστήματα με Java, Συστήματα Υπολογιστών Τόμος ΙΙΙ. Έκδοση 2η, Εκδόσεις Κλειδάριθμος, Αθήνα 2005.

[22] Δ. Κωνσταντίνου (2007). Προσδιορισμός και Αυτοματοποιημένη Υλοποίηση Αλγορίθμου Εκλογής Αρχηγού για Κινητά AD HOC Δίκτυα, Διπλωματική Εργασία, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου.

[23] Μ. Παπαχριστοδούλου (2012). Προδιαγραφή και Προσομοίωση Χρονισμένων Κατανεμημένων Αλγορίθμων Αμοιβαίου Αποκλεισμού χρησιμοποιώντας το εργαλείο Tempo, Διπλωματική Εργασία, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου.

Παράρτημα Α

Σε αυτό το παράρτημα δίνονται οι προδιαγραφές των καναλιών επικοινωνίας MPI και TCP στο εργαλείο Tempo και τη γλώσσα TIOA.

Α.1 Προδιαγραφή του καναλιού επικοινωνίας MPI

Α.1.1 Προδιαγραφή MPI Λεξιλογίου (Vocabulary.tioa)

```
vocabulary mpi_status_voc
   types mpi_status
   operators
      MPI_Iprobe : Nat -> Null[mpi_status],
      MPI_Test : mpi_status ->Bool
end

vocabulary mpi_message_voc
   imports algorithm_voc, mpi_status_voc
   types mpi_message : Tuple[message:messageType, sender:Nat, destination:Nat]
      operators
         MPI_Irecv: mpi_status, Nat -> mpi_message
end

vocabulary mpi_request_voc
   imports mpi_message_voc
      types mpi_request
         operators
            MPI_Isend: mpi_message, Nat -> Null[mpi_request],
            MPI_Barrier : -> Bool
end

vocabulary mpi_voc
   operators
      MPI_Rank : -> Nat,
      MPI_Size : -> Nat
end
```

Α.1.2 Προδιαγραφή MPI Send Mediator (SendMediator.tioa)

```
automaton SendMediator
   signature
      input SEND(m : Null[mpi_message])
   states
      status : Array[Nat, Null[mpi_request]] := constant(nil());
      clock : AugmentedReal := 0;
   transitions
      input SEND(m)
         eff
            if (m ~= nil()) then
```

```
            status[val(m).destination] := MPI_Isend(val(m), val(m).destination);
         fi
   trajectories
      trajdef DELAY
         evolve d(clock) = 1;
```

A.1.3 Προδιαγραφή MPI Receive Mediator (ReceiveMediator.tioa)

```
automaton ReceiveMediator
   signature
      output RECEIVE(m : Null[mpi_message])
      input probe(s:Nat)
   states
      toRecv : Seq[mpi_message] := {};
   transitions
      output RECEIVE(m) where len(toRecv) = 0
         pre
            m = nil();

      output RECEIVE(m) where len(toRecv) ~= 0
         pre
            m = embed(head(toRecv));
         eff
            toRecv := tail(toRecv);

      input probe(s)
         locals
            status : Null[mpi_status] := nil();
         eff
            status := MPI_Iprobe(s);
            if (status ~= nil()) then
               if( MPI_Test(val(status))) then
                  toRecv := toRecv |- MPI_Irecv(val(status), s);
               fi
            fi
```

A.2 Προδιαγραφή του καναλιού επικοινωνίας TCP

A.2.1 Προδιαγραφή TCP Λεξιλογίου (TCPVocabs.tioa)

```
vocabulary TCPObjectsVoc
   types
      IPv4 :Tuple[one:Nat, two:Nat, three:Nat, four:Nat],
      IPv6 : Tuple[one:Nat, two:Nat, three:Nat, four:Nat, five:Nat, six:Nat],
      JVMError: String
end

vocabulary TCPNodeVoc
   imports TCPObjectsVoc
   types
      Node : IPv4
   operators
      GT :Node, Node -> Bool,
      EQ :Node, Node -> Bool,
      LT :Node, Node ->Bool
end
```

```
vocabulary tcp_specific_voc
   imports alg_specific_voc, TCPObjectsVoc, TCPNodeVoc
   types
      Chan_message : Tuple[message : messageType, sender: Node, destination: Node],
      Status : Enumeration[closed, notAccepting, opening, emptying, connecting,
reading, rClosing, sConnected, connected, accepting, waiting, stopping, idle]
end

%% JVM Socket types and operations
vocabulary JVMSocket
   imports TCPObjectsVoc, TCPNodeVoc, tcp_specific_voc
   imports alg_specific_voc
   types JVMSocket
   operators
      JVM_TCPSocketOpen : Node, Nat, Nat -> Null[JVMSocket],
      JVM_TCPSocketClose : JVMSocket -> Null[JVMError],
      JVM_TCPSocketGetLocalIP :Null[JVMSocket] -> Null[Node],
      JVM_TCPSocketGetRemoteIP :Null[JVMSocket] -> Null[Node],
      JVM_read_TCPSocket : Null[JVMSocket] -> Null[Chan_message],
      JVM_write_TCPSocket : JVMSocket, Chan_message -> Null[JVMError],
      JVM_TCPSocketIsConnected :Null[JVMSocket] ->Bool
end

vocabulary JVMServerSocket
   imports TCPObjectsVoc, JVMSocket, TCPNodeVoc
   types JVMServerSocket
   operators
      JVM_TCPServerSocketOpen : Node, Nat, Nat -> Null[JVMServerSocket],
      JVM_TCPServerSocketClose : JVMServerSocket -> Null[JVMError],
      JVM_TCPServerSocketAccept :JVMServerSocket ->Null[JVMSocket]
end

%% This type provides sugar for the actual types and provides declaration for
types in the specification of the JCP channel.
vocabulary ChannelVoc
   imports JVMSocket, TCPObjectsVoc, tcp_specific_voc
   types
      Channel : Tuple[node:Node, socket:Null[JVMSocket], status:Status,
emptying:Bool, error:Null[JVMError]]
   operators
      empty_channel : ->Channel
end
```

A.2.2 Προδιαγραφή TCP Send Mediator (TCPSendMed.tioa)

```
automaton SendMed(port: Nat, timeout:Nat)
signature
   input SEND(m:Null[Chan_message])
   output TCP_senderOpen(remote:Node, port:Nat)
   input TCP_respSenderOpen(remote:Node, port:Nat, resp:Bool)
   output TCP_senderClose(remote: Node)
   input TCP_respSenderClose(remote:Node, resp:Bool)
   output TCP_write(m: Null[Chan_message], s,r :Node)
states
   sendBuffer :Seq[Chan_message] := {};
   remoteStatus : Array[Node, Status] := constant(idle);
   clocks : Array[Node, AugmentedReal] := constant(\infty);
   clock : AugmentedReal :=0;
```

```
let
   getMessage(s,r,index):Node, Node, Nat -> Null[Chan_message] =
   if index = len(sendBuffer) then
      nil() : Null[Chan_message]
   else
      if sendBuffer[index].sender = s /\ sendBuffer[index].destination = r then
         embed(sendBuffer[index])
      else
         getMessage(s,r,index+1);
transitions
   input SEND(m)
      eff
         if m ~= nil() then
            sendBuffer := sendBuffer |- val(m);
         fi;

   output TCP_senderOpen(remote, port)
      pre
         remoteStatus[remote] = closed \/ remoteStatus[remote] = idle;

   input TCP_respSenderOpen(remote,port,resp)
      eff
         if resp then
            remoteStatus[remote] := connected;
         fi;

   output TCP_senderClose(remote)
      pre
         remoteStatus[remote] = connected;

   input TCP_respSenderClose(remote,resp)
      eff
         if resp then
            remoteStatus[remote] := closed;
         fi;

   output TCP_write(m,s,r) where len(sendBuffer) =0
      pre
         m = nil();

   output TCP_write(m,s,r) where len(sendBuffer) ~= 0
      locals
         tempSendBuffer :Seq[Chan_message] := {};
         msg :Null[Chan_message] := nil();
      pre
         m = getMessage(s,r,0);
      eff
         msg := getMessage(s,r,0);
         if msg ~= nil() then
            for n:Nat where n < len(sendBuffer) do
               if sendBuffer[n] = val(msg) then
                  clocks[r]:=0;
               else
                  tempSendBuffer := tempSendBuffer |- sendBuffer[n];
               fi;
            od;
            sendBuffer := tempSendBuffer;
         fi;

%%% Trajectory modeling the delay needed for a message to be delivered to the
remote node
trajectories
```

```
   trajdef DELAY evolve d(clock) = 1;

      trajdef v(n:Node)
         invariant len(sendBuffer) ~= 0;
         stop when clocks[n] >= timeout;
         evolve d(clocks[n]) = 1;
```

A.2.3 Προδιαγραφή TCP Receive Mediator (TCPRecvMed.tioa)

```
automaton RecvMed(port:Nat, timeout:Nat)
signature
   output RECEIVE(m:Null[Chan_message])
   output TCP_read
   input TCP_respRead(m: Null[Chan_message])
   output TCP_bind(local:Node)
   input TCP_respBind(error: Null[JVMError], local:Node)
   output TCP_accept
   input TCP_respAccept(error: Null[JVMError])
   output TCP_stopAccepting
   input TCP_getError(e: Null[JVMError], remote:Node)
   output TCP_stopListening
   output TCP_rCloseStream(remote:Node)
   output TCP_rClose(remote:Node)

states
   recvBuffer : Seq[Chan_message] := {};
   recvErrors : Map[Node, Null[JVMError]] := empty();
   remoteStatus : Map[Node,Status] := empty();
   localStatus : Status :=idle;
   localError : Null[JVMError] := nil();
   noop :Bool :=true;

transitions
   output RECEIVE(m) where len(recvBuffer) = 0
      pre
         m = nil();

   output RECEIVE(m) where len(recvBuffer) ~= 0
      pre
         m = embed(head(recvBuffer));
      eff
         recvBuffer := tail(recvBuffer);

   output TCP_read
      pre
         localStatus ~=idle;
      eff
        recvErrors := empty();

   input TCP_respRead(m)
      eff
         if(m~=nil()) then
            recvBuffer := recvBuffer |- val(m);
          fi;

   output TCP_bind(local)
      pre
        localStatus = idle;
      eff
        localStatus := connecting;
```

```
      localError := nil();

input TCP_respBind(error,local)
   locals
      ftoss : JVMError := "FailesToOpenServerSocket";
   eff
      if(error = nil()) then
         if(localStatus = connecting) then
            localStatus := accepting;
         fi;
      else
         localError := error;
         if(val(error) = ftoss) then
            localStatus := idle;
         fi;
      fi;

output TCP_accept
   pre
      localStatus = accepting;
   eff
      localStatus := waiting;

input TCP_respAccept(error)
   eff
      if(localStatus = waiting) then
         localStatus := accepting;
      fi;
      localError :=error;

output TCP_stopAccepting
   pre
      localStatus = notAccepting;
    eff
      localStatus := idle;

output TCP_stopListening
pre
   localStatus = accepting;
eff
    localStatus := closed;

output TCP_rClose(remote)
pre
   true;
eff
   noop :=true;

output TCP_rCloseStream(remote)
pre
   true;
eff
   noop := true;

input TCP_getError(e,remote)
eff
   if(e ~= nil()) then
      recvErrors := update (recvErrors, remote, e);
    fi;
```

A.2.4 Προδιαγραφή TCP Channel Mediator (TCPChanMed.tioa)

```
automaton ChanMed(port:Nat, timeout:Nat)
signature
   input TCP_read
   output TCP_respRead(m: Null[Chan_message])
   input TCP_bind(local: Node)
   output TCP_respBind(error: Null[JVMError], local:Node)
   input TCP_accept
   output TCP_respAccept(error: Null[JVMError])
   input TCP_stopAccepting
   input TCP_stopListening
   input TCP_rClose(remote: Node)
   input TCP_rCloseStream(remote: Node)
   input TCP_senderOpen(remote :Node, port: Nat)
   output TCP_respSenderOpen(remote:Node, port:Nat, resp:Bool)
   input TCP_senderClose(remote: Node)
   output TCP_respSenderClose(remote: Node, resp: Bool)
   input TCP_write(m: Null[Chan_message], s, r :Node)
   internal TCP_senderClosing(remote: Node)
   output TCP_getError(e: Null[JVMError], remote: Node)

states
   SSocket : Null[JVMServerSocket] := nil();
   acceptStatus : Status :=idle;
   SError : Null[JVMError] := nil();
   AError : Null[JVMError] := nil();
   tcpChannel : Seq[Channel] := {};
   recvBuffer : Seq[Chan_message] := {};

let
   getError(r,index) : Node, Nat -> Null[JVMError] =
   if index = len(tcpChannel) then
      nil() : Null[JVMError]
   else
      if tcpChannel[index].node = r then
         tcpChannel[index].error
      else
         getError(r,index+1);

   isConnected(r,index) :Node, Nat -> Bool =
   if index = len(tcpChannel) then
      false
   else
      if tcpChannel[index].status = connected then
         true
      else
         isConnected(r,index+1);

   isClosed(r,index) : Node, Nat -> Bool =
   if index = len(tcpChannel) then
      false
   else
      if tcpChannel[index].status = closed \/ tcpChannel[index].status = idle then
         true
      else
         isConnected(r,index+1);

transitions
   input TCP_read
```

```
      locals
         msg: Null[Chan_message]:= nil();
       eff
         for n:Nat where n < len(tcpChannel) do
            if tcpChannel[n].socket ~= nil() /\ tcpChannel[n].status = connected
then
                tcpChannel[n].status := reading;
                msg := JVM_read_TCPSocket(tcpChannel[n].socket);
                if msg = nil() then
                   tcpChannel[n].error := embed("TimeoutOnRead");
                else
                    recvBuffer := recvBuffer |- val(msg);
                    tcpChannel[n].error := nil();
                fi;
            fi;
          od

   output TCP_respRead(m) where len(recvBuffer) = 0
       pre
          m = nil();
       eff
          for n:Nat where n < len(tcpChannel) do
              if tcpChannel[n].status = reading then
                 tcpChannel[n].status := connected;
              fi
          od

   output TCP_respRead(m) where len(recvBuffer) ~= 0
      pre
         m = embed(head(recvBuffer));
       eff
         recvBuffer := tail(recvBuffer);
         for n:Nat where n < len(tcpChannel) do
            if tcpChannel[n].status = reading then
               tcpChannel[n].status := connected;
             fi
          od

   input TCP_bind(local)
      eff
         acceptStatus := connecting;
         SSocket := JVM_TCPServerSocketOpen(local, port, timeout);
         if SSocket = nil() then
            SError := embed("FailedToOpenServerSocket");
         fi

   output TCP_respBind(error, local)
      pre
         acceptStatus = connecting;
         error = SError;
       eff
         if SSocket ~= nil() then
            acceptStatus := accepting;
         fi;

   input TCP_accept
      locals
         socket :Null[JVMSocket] := nil();
         found :Bool :=false;
       eff
         if acceptStatus = accepting then
            acceptStatus := waiting;
```

```
            socket := JVM_TCPServerSocketAccept(val(SSocket));
            if socket ~= nil() /\ JVM_TCPSocketIsConnected(socket) then
               for n:Nat where n< len(tcpChannel) /\ ~found do
                  if tcpChannel[n].node = val(JVM_TCPSocketGetRemoteIP(socket))
then
                     found := true;
                     if GT(val(JVM_TCPSocketGetRemoteIP(socket)),
val(JVM_TCPSocketGetLocalIP(socket))) \/ tcpChannel[n].status ~= connected \/
~JVM_TCPSocketIsConnected(tcpChannel[n].socket) then
                        tcpChannel[n].socket := socket;
                        tcpChannel[n].status := connected;
                        tcpChannel[n].emptying := false;
                        tcpChannel[n].error := nil();
                     fi
                  fi
               od
               if found = false then
                  tcpChannel := tcpChannel |-
[val(JVM_TCPSocketGetRemoteIP(socket)), socket, connected, false, nil()];
               fi
               else
                  AError := embed("NoConnectionOnAccept");
               fi
            fi;

   output TCP_respAccept(error)
      pre
         error = AError;
       eff
         if acceptStatus = waiting then
            acceptStatus := accepting;
            AError := nil();
         fi

   input TCP_stopAccepting
      eff
         if acceptStatus = stopping then
            acceptStatus := idle;
            SError := JVM_TCPServerSocketClose(val(SSocket));
            if SError ~= nil() then
               print SError;
            fi
         fi

   input TCP_stopListening
      eff
         if acceptStatus ~= idle then
            acceptStatus := stopping;
         fi

   input TCP_rClose(remote)
      eff
         for y:Nat where y < len(tcpChannel) do
            if tcpChannel[y].node = remote then
               tcpChannel[y].status := rClosing;
            fi
         od

   input TCP_rCloseStream(remote)
      eff
         for y:Nat where y < len(tcpChannel) do
```

```
            if tcpChannel[y].node = remote /\ tcpChannel[y].status = rClosing /\
tcpChannel[y].emptying = false then
                tcpChannel[y].status := closed;
             fi;
          od

   internal TCP_senderClosing(remote)
      pre
          len(tcpChannel) > 0;
       eff
          for y:Nat where y < len(tcpChannel) do
             if tcpChannel[y].status = emptying /\ tcpChannel[y].node = remote
then
                tcpChannel[y].status := closed;
             fi;
           od

   output TCP_getError(e, remote)
      pre
          e = getError(remote, 0);
       eff
          for y:Nat where y < len(tcpChannel) do
             if tcpChannel[y].socket ~= nil() /\ tcpChannel[y].error ~= nil() /\
tcpChannel[y].node = remote /\ tcpChannel[y].status = reading then
                tcpChannel[y].emptying := true;
             fi;
          od

   input TCP_write(m,s,r)
      locals
          error :Null[JVMError] := nil();
          found :Bool :=false;
       eff
          for n:Nat where (n < len(tcpChannel)) /\ ~found do
             if tcpChannel[n].socket = nil() then
                tcpChannel[n].status := closed;
             fi;
             if tcpChannel[n].socket ~= nil() /\ tcpChannel[n].status = connected
/\ tcpChannel[n].node = r then
                found := true;
                error := JVM_write_TCPSocket(val(tcpChannel[n].socket), val(m));
                if error ~= nil() then
                   tcpChannel[n].error := error;
                    print val(error);
                fi
             fi
          od

   input TCP_senderOpen(remote,port)
      locals
          found :Bool :=false;
          index :Nat := 0;
          socket :Null[JVMSocket] := nil();
          error :Null[JVMError] := nil();
      eff
          for n:Nat where n < len(tcpChannel) /\ ~found do
             if tcpChannel[n].node = remote then
                found :=true;
                if tcpChannel[n].socket = nil()     \/ tcpChannel[n].status =
closed then
                   tcpChannel[n].socket := JVM_TCPSocketOpen(remote, port,
timeout);
```

```
                fi
             fi
            od
            if (found = false) then
                socket := JVM_TCPSocketOpen(remote, port, timeout);
                if socket ~= nil() then
                   tcpChannel := tcpChannel |- [remote, socket, connected, false,
nil()];
                else
                   tcpChannel := tcpChannel |- [remote, socket, closed, false,
nil()];
                fi
             fi

   output TCP_respSenderOpen(remote, port, resp)
      pre
          resp = isConnected(remote,0);

   input TCP_senderClose(remote)
      locals
          error :Null[JVMError] := nil();
       eff
          for n:Nat where n < len(tcpChannel)do
             if tcpChannel[n].node = remote then
                tcpChannel[n].emptying :=true;
                error := JVM_TCPSocketClose(val(tcpChannel[n].socket));
                if error ~=nil() then
                   tcpChannel[n].error := error;
                    print val(error);
                fi
             fi
          od

   output TCP_respSenderClose(remote, resp)
      pre
         resp = isClosed(remote,0);
```

Παράρτημα Β

Σε αυτό το παράρτημα δίνονται οι προδιαγραφές των αλγορίθμων που μελετήθηκαν και υλοποιήθηκαν στην παρούσα διπλωματική εργασία.

Β.1 Προδιαγραφή Αλγορίθμου με Τοπολογία Δέντρου

Β.1.1 Προδιαγραφή Λεξιλογίου (Vocabulary.tioa)

```
%%% :Algorithm vocabs

vocabulary algorithm_voc
   types
      Message : Tuple[type:Nat, id : Nat]
end


%%% :MPI Channel vocabs

vocabulary mpi_status_voc
   types mpi_status
   operators
      MPI_Iprobe : Nat -> Null[mpi_status],
      MPI_Test : mpi_status ->Bool
end

vocabulary mpi_message_voc
   imports algorithm_voc, mpi_status_voc
      types mpi_message : Tuple[data:Message,sender : Nat, destination:Nat]
      operators
         MPI_Irecv: mpi_status, Nat -> mpi_message
end

vocabulary mpi_request_voc
   imports mpi_message_voc
      types mpi_request
      operators
         MPI_Isend: mpi_message, Nat -> Null[mpi_request],
         MPI_Barrier : -> Bool
end

vocabulary mpi_voc
       operators
              MPI_Rank : -> Nat,
              MPI_Size : -> Nat
end
```

B.1.2 Προδιαγραφή Αυτομάτου Διεργασίας (Algorithm1_Process)

```
automaton Algorithm1_Process (t1:DiscreteReal, rank :Nat)
signature
   input RECEIVE (m:Null[mpi_message])
   output SEND (m:Null[mpi_message])
   internal processTimeout, init, enableCondition
states
       % The physcal clock of the automaton
       clock : DiscreteReal :=0;
       % The physical clock for the process
       processTime : DiscreteReal :=0;
       % An empty sequence of mpi messages
       msgs : Seq[Null[mpi_message]] := {};
       % Sequence of all the neighboor processes of this process
       NbrsNodes : Seq[Nat] :={};
       Nodes : Seq[Nat] :={};
       % The minimun id from all the processes
       minimunId : Nat :=rank ;
       % A counter for the messages that the process has received
       receiveMessagesCounter : Int :=0;
       % Indicates that the process is leader
       isLeader : Bool := false;
       % Indicates the process rank
       processRank : Nat := rank;
       % Indicates token message
       TokenMessage : Nat := 1;
       % Indicates leader message
       LeaderMessage:Nat:=2;
transitions
       input RECEIVE(m)
              locals positionOfSender:Nat :=0;
              eff
                 if(m ~= nil()) then
                    processTime := clock;

                    if(val(m).data.type = TokenMessage)then
                       receiveMessagesCounter := receiveMessagesCounter +1;

                       if(minimunId > val(m).data.id) then
                          minimunId := val(m).data.id;
                       fi;

                       if(len(Nodes)>1)then
                          positionOfSender :=0;
                          for n:Nat where n < (len(Nodes)) do
                             if(Nodes[n] = (val(m).sender))then
                                positionOfSender:=n;
                             fi;
                          od;
                          Nodes := eject(Nodes,positionOfSender);
                       fi;

                       if(receiveMessagesCounter = (len(NbrsNodes)-1) )then
                          msgs := msgs |- embed([[TokenMessage,
       minimunId],rank,Nodes[0]]);
                       fi;

                       if(receiveMessagesCounter = len(NbrsNodes) /\ val(m).sender
       = Nodes[0]) then
                          if(minimunId = rank)then
                             isLeader := true;
```

```
                    fi;
                    for n:Nat where n<len(NbrsNodes) do
                       if(NbrsNodes[n] ~= Nodes[0]) then
                          msgs := msgs |- embed([[LeaderMessage,
minimunId],rank,NbrsNodes[n]]);
                       fi;
                    od;
                 fi;
              fi;
              if(val(m).data.type = LeaderMessage)then
                 minimunId := val(m).data.id;
              fi;
           fi;


output SEND(m)
       pre
          msgs ~= {};
          m = head(msgs);

       eff
          processTime := 0;
          msgs := tail(msgs);


internal processTimeout
       pre
          processTime >= t1;
       eff
          processTime := 0;


internal init
       eff

          if (rank = 0)then
              NbrsNodes:= NbrsNodes |- 1;
              NbrsNodes := set(NbrsNodes,0,1);
              NbrsNodes:= NbrsNodes |- 3;
              NbrsNodes := set(NbrsNodes,1,3);
          fi;
          if (rank = 1)then
              NbrsNodes:= NbrsNodes |- 0;
              NbrsNodes := set(NbrsNodes,0,0);
          fi;
          if (rank = 2)then
              NbrsNodes:= NbrsNodes |- 3;
              NbrsNodes := set(NbrsNodes,0,3);
          fi;
          if (rank = 3)then
              NbrsNodes:= NbrsNodes |- 0;
              NbrsNodes := set(NbrsNodes,0,0);
              NbrsNodes:= NbrsNodes |- 2;
              NbrsNodes := set(NbrsNodes,1,2);
          fi;
          Nodes := NbrsNodes;

internal enableCondition
       pre
          receiveMessagesCounter = len(NbrsNodes) - 1;
          len(Nodes) = 1;
       eff
```

```
                msgs := msgs |- embed([[TokenMessage,minimunId],rank,Nodes[0]]);
trajectories
       trajdef time
              evolve d(clock)=1;
```

B.1.3 Προδιαγραφή Αυτομάτου Σύνθεσης (Algorithm1.tioa)

```
%%% .: Vocabulary :.
include "Vocabulary.tioa"
imports algorithm_voc, mpi_status_voc, mpi_message_voc, mpi_request_voc, mpi_voc

%%% .: MPI Mediator Automata :.
include "SendMediator.tioa"
include "ReceiveMediator.tioa"

%%% .: Algorithm Automata :.
include "Algorithm1_Process.tioa"

automaton Algorithm1(t1: DiscreteReal)
components
       P: Algorithm1_Process(t1,MPI_Rank());
       SM: SendMediator;
       RM: ReceiveMediator;

states
       m:Null[mpi_message] :=nil();
       runs : Int :=MPI_Size();

do
       fire internal P.init;


       for i:Nat where i<runs do
              follow P.time duration 1;

              fire internal P.enableCondition;
              fire internal P.processTimeout;

              %% Send Messages
              for j:Nat where j<  MPI_Size() do
                     fire output P.SEND(m);
              od

              follow SM.DELAY duration (MPI_Size() * 10);

              %% Receive Messages
              for j:Nat where j<MPI_Size() do
                     m:=nil();
                     fire input RM.probe(j);
                     fire output RM.RECEIVE(m);
              od
       od
 od
```

Β.2 Προδιαγραφή Αλγορίθμου με Τοπολογία Δακτυλίου

Β.2.1 Προδιαγραφή Λεξιλογίου (Vocabulary.tioa)

```
%%% :Algorithm vocabs

vocabulary algorithm_voc
	types
		Message : Tuple[type:Nat, id : Nat]
end


%%% :MPI Channel vocabs

vocabulary mpi_status_voc
	types mpi_status
	operators
		MPI_Iprobe : Nat -> Null[mpi_status],
		MPI_Test : mpi_status ->Bool
end

vocabulary mpi_message_voc
	imports algorithm_voc, mpi_status_voc
		types mpi_message : Tuple[data:Message, destination:Nat]
		operators
			MPI_Irecv: mpi_status, Nat -> mpi_message
end

vocabulary mpi_request_voc
	imports mpi_message_voc
		types mpi_request
		operators
			MPI_Isend: mpi_message, Nat -> Null[mpi_request],
			MPI_Barrier : -> Bool
end

vocabulary mpi_voc
	operators
		MPI_Rank : -> Nat,
		MPI_Size : -> Nat
end
```

Β.2.2 Προδιαγραφή Αυτομάτου Διεργασίας (Algorithm2_Process.tioa)

```
automaton Algorithm2_Process (t1:DiscreteReal,rank :Nat)
signature
		input RECEIVE (m:Null[mpi_message])
		output SEND (m:Null[mpi_message])
		internal processTimeout, init, prepMessages
states
	% The physcal clock of the automaton
	clock : DiscreteReal :=0;
	% The physical clock for the process
	processTime : DiscreteReal :=0;
	% An empty sequence of mpi messages
	msgs : Seq[Null[mpi_message]] := {};
	% Indicates the next node of the process
	nextNode : Nat := 0;
	% Indicates that the process has sent a message
```

```
        sendFlag : Bool := false;
        % Indicates that the process is leader
        isLeader : Bool := false;
        % The minimun id from all the processes
        minimunId : Nat :=rank ;
        % Indicates the rank of this process
        processRank : Nat := rank;
        %Indicates a token message
        TokenMessage : Nat := 1;
        % Message which indicates that a leader has found
        LeaderMessage : Nat :=2;
transitions
        input RECEIVE(m)
           eff
              if(m ~= nil())then
                  processTime := clock;
                  if(val(m).data.type = TokenMessage)then
                      if(val(m).data.id < minimunId)then
                         minimunId := val(m).data.id;
                         msgs := msgs |- embed([[TokenMessage,minimunId],nextNode]);
                      fi;
                      if(val(m).data.id = rank)then
                         isLeader := true;
                         msgs := msgs |-
embed([[LeaderMessage,minimunId],nextNode]);
                      fi;
                  fi;

                  if(val(m).data.type = LeaderMessage)then
                     minimunId := val(m).data.id;
                     if(rank ~= minimunId)then
                         msgs := msgs |-
embed([[LeaderMessage,minimunId],nextNode]);
                      fi;
                  fi;

              fi;

        output SEND(m)
           pre
              msgs ~= {};
              m = head(msgs);
           eff
              processTime := 0;
              msgs :=tail(msgs);
        internal init
           eff
              if(rank = 0)then
                  nextNode := 2;
              fi;
              if(rank = 1)then
                  nextNode := 3;
              fi;
              if(rank = 2)then
                  nextNode := 1;
              fi;
              if(rank = 3)then
                  nextNode := 0;
              fi;


        internal prepMessages
```

```
            pre
               len(msgs) = 0;
            eff
               msgs := msgs |- embed([[TokenMessage,rank],nextNode]);


      internal processTimeout
            pre
               processTime >= t1;
            eff
               processTime :=0;
trajectories
      trajdef time
            evolve d(clock)=1;
```

B.2.3 Προδιαγραφή Αυτομάτου Σύνθεσης (Algorithm2.tioa)

```
%%% .: Vocabulary :.
include "Vocabulary.tioa"
imports algorithm_voc, mpi_status_voc, mpi_message_voc, mpi_request_voc, mpi_voc

%%% .: MPI Mediator Automata :.
include "SendMediator.tioa"
include "ReceiveMediator.tioa"

%%% .: Algorithm Automata :.
include "Algorithm2_Process.tioa"

automaton Algorithm2(t1: DiscreteReal)
components
      P: Algorithm2_Process(t1,MPI_Rank());
      SM: SendMediator;
      RM: ReceiveMediator;

states
      m:Null[mpi_message] :=nil();
      runs : Nat :=MPI_Size()**2;
do
      fire internal P.init;

      for i:Nat where i<runs do
            follow P.time duration 1;

            fire internal P.prepMessages;
            fire internal P.processTimeout;

            % Send Messages
            for j:Nat where j<  MPI_Size() do
                  fire output P.SEND(m);
            od

            follow SM.DELAY duration (MPI_Size() * 10);

            % Receive Messages
            for j:Nat where j<MPI_Size() do
                  fire input RM.probe(j);
                  fire output RM.RECEIVE(m);
            od
```

```
      od
 od
```

B.3 Προδιαγραφή Αλγορίθμου με Τοπολογία Ισχυρά Συνδεδεμένου Γράφου

B.3.1 Προδιαγραφή Αλγορίθμου με Τοπολογία Ισχυρά Συνδεδεμένου Γράφου με το Κανάλι Επικοινωνίας MPI

B.3.1.1 Προδιαγραφή Λεξιλογίου (Vocabulary.tioa)

```
%%% :Algorithm vocabs

vocabulary algorithm_voc
      types
            messageType : Nat
end


%%% :MPI Channel vocabs

vocabulary mpi_status_voc
      types mpi_status
      operators
            MPI_Iprobe : Nat -> Null[mpi_status],
            MPI_Test : mpi_status ->Bool
end

vocabulary mpi_message_voc
      imports algorithm_voc, mpi_status_voc
            types mpi_message : Tuple[message:messageType, sender:Nat,
destination:Nat]
            operators
                  MPI_Irecv: mpi_status, Nat -> mpi_message
end

vocabulary mpi_request_voc
      imports mpi_message_voc
            types mpi_request
            operators
                  MPI_Isend: mpi_message, Nat -> Null[mpi_request],
                  MPI_Barrier : -> Bool
end

vocabulary mpi_voc
      operators
            MPI_Rank : -> Nat,
            MPI_Size : -> Nat
end
```

B.3.1.2 Προδιαγραφή Αυτομάτου Διεργασίας (Alogrithm3_process.tioa)

```
automaton Algorithm3_Process(timeForElection:DiscreteReal,
timeForAnouncementLeader:DiscreteReal, timeForCheckLeader:DiscreteReal,
possibilityToFail: Int, possibilityToRecover : Int, rank :Nat )
signature
	input RECEIVE (m:Null[mpi_message])
	output SEND (m:Null[mpi_message])
	internal init,checkLeader
	internal chooseToFail, chooseToRecover
	internal checkLeaderTimeout, electionTimeout, announcementLeaderTimeout
states
	% The physcal clock of the automaton
	clock : DiscreteReal :=0;
	% The physical clock to check leader process
	checkLeaderTimer : DiscreteReal :=0;
	% The physical clock to elect leader
	electionTimer : DiscreteReal :=0;
	% The physical clock to announce leader
	announcementLeaderTimer : DiscreteReal :=0;
	% An empty sequence of mpi messages
	msgs :Seq[Null[mpi_message]] :={};
	% A counter for the messages that a process has send
	messagesCounter : Int := 0;
	% A counter for the messages that sent for election
	messagesForElection : Int :=0;
	% Indicates that the process is active
	processIsActive : Bool :=true;
	% Indicates the rank of this process
	processRank : Nat := rank;
	% Indicates the id of the leader process
	leaderId : Nat:=0;
	% Tuple which indicates the possible status of the process
	status : Tuple[thereIsLeader:Bool, election:Bool,
waitForLeaderAnnouncement:Bool] :=[false,false,false];
	% Message to election a leader
	ElectionMessage : Nat :=1;
	% Message to reply in election message
	ReplyElectionMessage : Nat :=2;
	% Message to announce the leader
	 AnnouncementLeaderMessage :Nat :=3;
	% Message to check the leader
	CheckLeaderMessage : Nat :=4;
	% Message to reply in check leader message
	ReplyCheckLeaderMessage : Nat :=5;
transitions

	input RECEIVE(m)
	   locals iAmLeader : Bool := false;
	   eff
	      if(processIsActive)then
	         if(m ~= nil())then

	            if(val(m).message = ElectionMessage)then
	               status := [false,true,false];
	               msgs := msgs |- embed([ReplyElectionMessage, rank,
val(m).sender]);
	            iAmLeader := true;
	            for n:Nat where n < MPI_Size()do
	                  if(n > rank)then
```

```
                        msgs := msgs |- embed([ElectionMessage, rank, n]);
                        iAmLeader := false;
                    fi;
                od;
                if(iAmLeader = true)then
                    leaderId := rank;
                    status := [true,false,false];
                    for n:Nat where n < MPI_Size() do
                        if(n < rank)then
                            msgs := msgs |- embed([AnnouncementLeaderMessage,
rank, n]);
                        fi;
                    od;
                fi;
                fi;

                if(val(m).message = ReplyElectionMessage)then
                    electionTimer := 0;
                    status := [false,false,true];
                    announcementLeaderTimer := clock;
                fi;

                if(val(m).message = AnnouncementLeaderMessage)then
                    announcementLeaderTimer := 0;
                    status := [true,false,false];
                    leaderId := val(m).sender;
                fi;

                if(val(m).message = CheckLeaderMessage)then
                    msgs := msgs |- embed([ReplyCheckLeaderMessage, rank,
val(m).sender]);
                fi;


                if(val(m).message = ReplyCheckLeaderMessage)then
                    checkLeaderTimer :=0;
                fi;

            fi;
        fi;


    output SEND (m)
        pre
            processIsActive;
            msgs ~= {};
            m = head(msgs);
        eff
            msgs := tail(msgs);
            messagesCounter := messagesCounter + 1;

            if(val(m).message = ElectionMessage)then
                electionTimer :=clock;
            fi;
            if(val(m).message = CheckLeaderMessage)then
                checkLeaderTimer := clock;
            fi;
    internal init
        eff
            processIsActive := true;
            status := [true,false,false];
            leaderId := rank;
```

```
            for n:Nat where n < MPI_Size()do
               if(n > leaderId)then
                  leaderId := n;
               fi;
            od;


      internal checkLeader
         pre
            processIsActive;
            leaderId ~= rank;
            status = [true,false,false];
         eff
            msgs := msgs |- embed([CheckLeaderMessage,rank,leaderId]);


      internal chooseToFail
         locals randomValueToFail : Int := choose n where (n>=1 /\ n <= 100);
         eff
            if(processIsActive)then
               if(randomValueToFail >= 0 /\ randomValueToFail <=
possibilityToFail)then
                  processIsActive := false;
               fi;
            fi;


      internal chooseToRecover
         locals randomValueToRecover : Int := choose n where (n>=1 /\ n <= 100);
         eff
            if(processIsActive = false)then
                if(randomValueToRecover >=0 /\  randomValueToRecover <=
      possibilityToRecover)then
                  processIsActive := true;
                  if(leaderId < rank)then
                     for n:Nat where n < MPI_Size()do
                        if(n ~= rank)then
                           msgs := msgs |-
embed([AnnouncementLeaderMessage,rank,n]);
                        fi;
                     od;
                  else
                     for n:Nat where n < MPI_Size()do
                        if(n > rank)then
                           msgs := msgs |- embed([ElectionMessage,rank,n]);
                        fi;
                     od;
                  fi;
               fi;

            fi;


      internal checkLeaderTimeout
         locals iAmLeader : Bool := false;
         pre
            clock > (checkLeaderTimer + timeForCheckLeader);
            processIsActive;
            status = [true,false,false];
       eff
            status := [false,true,false];
```

```
            iAmLeader := true;
            for n:Nat where n < MPI_Size()do
               if(n > rank)then
                  msgs := msgs |- embed([ElectionMessage, rank, n]);
                  iAmLeader := false;
               fi;
            od;

            if(iAmLeader = true)then
               leaderId := rank;
               status := [true,false,false];
               for n:Nat where n < MPI_Size() do
                  if(n ~= rank)then
                     msgs := msgs |- embed([AnnouncementLeaderMessage, rank, n]);

                  fi;
               od;
            fi;
            checkLeaderTimer := 0;


      internal electionTimeout
         pre
            clock > (electionTimer + timeForElection);
            processIsActive;
            status = [false,true,false];
         eff
            for n:Nat where n < MPI_Size() do

               if(n > rank)then
                  msgs := msgs |- embed([ElectionMessage, rank, n]);
               fi;
            od;
            electionTimer := 0;


      internal announcementLeaderTimeout
         pre
            clock > (announcementLeaderTimer + timeForAnouncementLeader);
             processIsActive;
            status = [false,false,true];
         eff
            status := [true,false,false];
            leaderId :=rank;
            for n:Nat where n < MPI_Size() do
               if(n < rank)then
                  msgs := msgs |- embed([AnnouncementLeaderMessage, rank, n]);
               fi;
            od;
            announcementLeaderTimer :=0;

trajectories
      trajdef time
             evolve d(clock) = 1;
```

B.3.1.3 Προδιαγραφή Αυτομάτου Σύνθεσης (Algorithm3.tioa)

```
%%% .: Vocabulary :.
```

```
include "Vocabulary.tioa"
imports algorithm_voc, mpi_status_voc, mpi_message_voc, mpi_request_voc, mpi_voc

%%% .: MPI Mediator Automata :.
include "SendMediator.tioa"
include "ReceiveMediator.tioa"

%%% .: Algorithm Automata :.
include "Algorithm3_Process.tioa"
automaton Algorithm3(timeForElection:DiscreteReal,
timeForAnouncementLeader:DiscreteReal,timeForCheckLeader:DiscreteReal,
              possibilityToFail: Int, possibilityToRecover : Int)
components
       P: Algorithm3_Process(timeForElection,timeForAnouncementLeader,
timeForCheckLeader, possibilityToFail, possibilityToRecover,
                     MPI_Rank());
       SM: SendMediator;
       RM: ReceiveMediator;
states
       m:Null[mpi_message] :=nil();
       runs : Nat := MPI_Size()**2;
do
       fire internal P.init;

       for i:Nat where i<runs do
              follow P.time duration 1;

              fire internal P.chooseToFail;
              fire internal P.chooseToRecover;
              fire internal P.checkLeader;
              fire internal P.checkLeaderTimeout;
              fire internal P.electionTimeout;
              fire internal P.announcementLeaderTimeout;

              % Send Messages
              for j:Nat where j<  MPI_Size() do
                     fire output P.SEND(m);
              od
              follow SM.DELAY duration (MPI_Size() * 10);

              % Receive Messages
              for j:Nat where j<MPI_Size() do
                     fire input RM.probe(j);
                     fire output RM.RECEIVE(m);
              od

       od
 od
```

B.3.2 Προδιαγραφή Αλγορίθμου με Τοπολογία Ισχυρά Συνδεδεμένου Γράφου με το Κανάλι Επικοινωνίας TCP

B.3.2.1 Προδιαγραφή Λεξιλογίου (TCPVocabs.tioa)

```
%%% :Algorithm vocabs
```

```
vocabulary alg_specific_voc
       types
          messageType : Nat
end

%%% :TCP Channel vocabs

vocabulary TCPObjectsVoc
       types
          IPv4 :Tuple[one:Nat, two:Nat, three:Nat, four:Nat],
          IPv6 : Tuple[one:Nat, two:Nat, three:Nat, four:Nat, five:Nat, six:Nat],
          JVMError: String
End

vocabulary TCPNodeVoc
       imports TCPObjectsVoc
       types
          Node : IPv4
       operators
          GT :Node, Node -> Bool,
          EQ :Node, Node -> Bool,
          LT :Node, Node ->Bool
end

vocabulary tcp_specific_voc
       imports alg_specific_voc, TCPObjectsVoc, TCPNodeVoc
       types
          Chan_message : Tuple[message : messageType, sender: Node, destination:
Node],
          Status : Enumeration[closed, notAccepting, opening, emptying,
connecting, reading, rClosing, sConnected, connected, accepting, waiting,
stopping, idle]
end

%% JVM Socket types and operations
vocabulary JVMSocket
       imports TCPObjectsVoc, TCPNodeVoc, tcp_specific_voc
       imports alg_specific_voc
       types JVMSocket
       operators
          JVM_TCPSocketOpen : Node, Nat, Nat -> Null[JVMSocket],
          JVM_TCPSocketClose : JVMSocket -> Null[JVMError],
          JVM_TCPSocketGetLocalIP :Null[JVMSocket] -> Null[Node],
          JVM_TCPSocketGetRemoteIP :Null[JVMSocket] -> Null[Node],
          JVM_read_TCPSocket : Null[JVMSocket] -> Null[Chan_message],
          JVM_write_TCPSocket : JVMSocket, Chan_message -> Null[JVMError],
          JVM_TCPSocketIsConnected :Null[JVMSocket] ->Bool
end


vocabulary JVMServerSocket
       imports TCPObjectsVoc, JVMSocket, TCPNodeVoc
       types JVMServerSocket


       operators
          JVM_TCPServerSocketOpen : Node, Nat, Nat -> Null[JVMServerSocket],
          JVM_TCPServerSocketClose : JVMServerSocket -> Null[JVMError],
          JVM_TCPServerSocketAccept :JVMServerSocket ->Null[JVMSocket]
end
```

```
%% This type provides sugar for the actual types and provides declaration for
types in the specification of the JCP channel.
vocabulary ChannelVoc
       imports JVMSocket, TCPObjectsVoc, tcp_specific_voc
       types
          Channel : Tuple[node:Node, socket:Null[JVMSocket], status:Status,
emptying:Bool, error:Null[JVMError]]
       operators
          empty_channel : ->Channel
end
```

B.3.2.2 Προδιαγραφή Αυτομάτου Διεργασίας (Alogrithm3_process.tioa)

```
automaton Algorithm3_Process(timeForElection:DiscreteReal,
timeForAnouncementLeader:DiscreteReal, timeForCheckLeader:DiscreteReal,
possibilityToFail: Int, possibilityToRecover : Int, IP :Node, rank : Nat )
signature
             input RECEIVE (m:Null[Chan_message])
             output systemInitialize(w:Seq[Node])
             output SEND (m:Null[Chan_message])
             internal init,checkLeader
             internal chooseToFail, chooseToRecover
             internal checkLeaderTimeout, electionTimeout,
announcementLeaderTimeout
             states
       % The physcal clock of the automaton
       clock : DiscreteReal :=0;
       % The physical clock to check leader process
       checkLeaderTimer : DiscreteReal :=0;
       % The physical clock to elect leader
       electionTimer : DiscreteReal :=0;
       % The physical clock to announce leader
       announcementLeaderTimer : DiscreteReal :=0;
       % An empty sequence of mpi messages
       msgs :Seq[Null[Chan_message]] :={};
       % A counter for the messages that a process has send
       messagesCounter : Int := 0;
       % Indicates that the process is active
       processIsActive : Bool :=true;
       % Indicates the rank of this process
       processRank : Nat := rank;
       % Indicates the IP of this process
       processIP : Node := IP ;
       % Indicates the id of the leader process
       leaderId : Nat := rank;
       % Indicates the IP of the leader process
       leaderIP : Node:=IP;
       % Tuple which indicates the possible status of the process
       status : Tuple[thereIsLeader:Bool,
election:Bool,waitForLeaderAnnouncement:Bool] :=[false,false,false];
       % Message to election a leader
       ElectionMessage : Nat :=1;
       % Message to reply in election message
       ReplyElectionMessage : Nat :=2;
       % Message to announce the leader
       AnnouncementLeaderMessage :Nat :=3;
       % Message to check the leader
```

```
        CheckLeaderMessage : Nat :=4;
        % Message to reply in check leader message
        ReplyCheckLeaderMessage : Nat :=5;
        % An empty sequence of IP andresses
        world : Seq[Node] := {};
transitions
        output systemInitialize(w)
           eff
              world := w;

        input RECEIVE(m)
           locals iAmLeader : Bool := false;
           eff
              if(processIsActive)then
                  if(m ~= nil())then
                     if(val(m).message = ElectionMessage)then
                         status := [false,true,false];
                         msgs := msgs |- embed([ReplyElectionMessage, IP,
val(m).sender]);
                         iAmLeader := true;
                         for n:Nat where n < len(world)do
                            if(n> processRank)then
                               msgs := msgs |- embed([ElectionMessage, IP,
world[n]]);
                               iAmLeader := false;
                            fi;
                         od;

                         if(iAmLeader = true)then
                            leaderIP := IP;
                            leaderId := processRank;
                            status := [true,false,false];
                            for n:Nat where n < len(world) do
                              if(n < processRank )then
                                 msgs := msgs |- embed([AnnouncementLeaderMessage,
IP, world[n]]);
                              fi;
                            od;
                         fi;
                     fi;
                     if(val(m).message = ReplyElectionMessage)then
                        electionTimer := 0;
                        status := [false,false,true];
                        announcementLeaderTimer := clock;
                     fi;

                     if(val(m).message = AnnouncementLeaderMessage)then
                        announcementLeaderTimer := 0;
                        status := [true,false,false];
                        leaderIP := val(m).sender;
                        for n:Nat where n<len(world)do
                           if(EQ(leaderIP,world[n]))then
                              leaderId := n;
                           fi;
                        od
                     fi;

                  if(val(m).message = CheckLeaderMessage)then
                     msgs := msgs |- embed([ReplyCheckLeaderMessage, IP,
val(m).sender]);
                  fi;
```

```
            if(val(m).message = ReplyCheckLeaderMessage)then
               checkLeaderTimer :=0;
            fi;

         fi;
      fi;

   output SEND (m)
      pre
         processIsActive;
         msgs ~= {};
         m = head(msgs);
      eff
         msgs := tail(msgs);
          messagesCounter := messagesCounter + 1;

          if(val(m).message = ElectionMessage)then
             electionTimer :=clock;
          fi;

         if(val(m).message = CheckLeaderMessage)then
            checkLeaderTimer := clock;
         fi;

   internal init
      eff
         processIsActive := true;
         status := [true,false,false];

         leaderId := 0;
         for n:Nat where n < len(world)do
            if(n>processRank)then
               leaderIP := world[n];
               leaderId := n;
            fi;
         od;

   internal checkLeader
      pre
          processIsActive;
         leaderId ~= processRank;
         status = [true,false,false];
      eff
         msgs := msgs |- embed([CheckLeaderMessage,IP,leaderIP]);


   internal chooseToFail
      locals randomValueToFail : Int := choose n where (n>=1 /\ n <= 100);
      eff
         if(processIsActive)then
            if(randomValueToFail >= 0 /\ randomValueToFail <=
possibilityToFail)then
               processIsActive := false;
            fi;
         fi;



   internal chooseToRecover
      locals randomValueToRecover : Int := choose n where (n>=1 /\ n <= 100);
      eff
         if(processIsActive = false)then
```

```
                if(randomValueToRecover >=0 /\  randomValueToRecover <=
possibilityToRecover)then
                    processIsActive := true;
                    if(leaderId<processRank)then
                       for n:Nat where n < len(world)do
                          if(n~=processRank)then
                             msgs := msgs |-
embed([AnnouncementLeaderMessage,IP,world[n]]);
                          fi;
                       od;
                    else
                       for n:Nat where n < len(world)do
                          if(n>processRank)then
                             msgs := msgs |- embed([ElectionMessage,IP,world[n]]);
                          fi;
                       od;
                    fi;
                fi;

             fi;

       internal checkLeaderTimeout
          locals iAmLeader : Bool := false;
          pre
             clock > (checkLeaderTimer + timeForCheckLeader);
             processIsActive;
             status = [true,false,false];
          eff
             status := [false,true,false];

             iAmLeader := true;
             for n:Nat where n < len(world)do
                if(n>processRank)then
                   msgs := msgs |- embed([ElectionMessage, IP, world[n]]);
                   iAmLeader := false;
                fi;
             od;

             if(iAmLeader = true)then
                leaderId := processRank;
                leaderIP := IP;
                status := [true,false,false];
                for n:Nat where n < len(world) do
                   if(n~=processRank)then
                      msgs := msgs |- embed([AnnouncementLeaderMessage, IP,
world[n]]);
                   fi;
                od;
             fi;

             checkLeaderTimer := 0;


       internal electionTimeout
          pre
             clock > (electionTimer + timeForElection);
             processIsActive;
             status = [false,true,false];
          eff
             for n:Nat where n < len(world) do
                if(n>processRank)then
```

```
                msgs := msgs |- embed([ElectionMessage, IP, world[n]]);
             fi;
          od;

       electionTimer := 0;

    internal announcementLeaderTimeout
       pre
          clock > (announcementLeaderTimer + timeForAnouncementLeader);
          processIsActive;
          status = [false,false,true];
       eff
          status := [true,false,false];
          leaderId :=processRank;
          leaderIP := IP;
          for n:Nat where n < len(world) do
             if(n < processRank)then
                msgs := msgs |- embed([AnnouncementLeaderMessage, IP,
world[n]]);
             fi;
          od;
          announcementLeaderTimer :=0;

trajectories
    trajdef time
          evolve d(clock) = 1;
```

B.3.2.3 Προδιαγραφή Αυτομάτου Σύνθεσης (Algorithm3.tioa)

```
%%% .: Vocabulary :.
include "TCPVocabs.tioa"
imports alg_specific_voc, TCPObjectsVoc, TCPNodeVoc, tcp_specific_voc, JVMSocket,
JVMServerSocket, ChannelVoc

%%% .: TCP Mediator Automata :.
include "TCPRecvMed.tioa"
include "TCPSendMed.tioa"
include "TCPChanMed.tioa"

%%% .: Algorithm Automata :.
include "Algorithm3_Process.tioa"
automaton Algorithm3(n1:Nat, n2:Nat, n3:Nat, n4:Nat, port:Nat,
timeout:Nat,timeForElection:DiscreteReal,
timeForAnouncementLeader:DiscreteReal,timeForCheckLeader:DiscreteReal,
          possibilityToFail: Int, possibilityToRecover : Int, processRank :
Nat)
components
    P: Algorithm3_Process(timeForElection,timeForAnouncementLeader,
timeForCheckLeader, possibilityToFail,possibilityToRecover, [n1,n2,n3,n4],
processRank);
    SM: SendMed(port,timeout);
    RM: RecvMed(port,timeout);
    CM: ChanMed(port,timeout);
states
    myIp : Node := [n1,n2,n3,n4];    %IP from parameters
    world : Seq[Node] := {};
    ms:Null[Chan_message] := nil();
    mr:Null[Chan_message] := nil();
    dowhile : Bool :=true;
    isconn : Bool := false;
```

```
        error : Null[JVMError] :=nil();
        runs : Nat := MPI_Size()**2;
do
        %% INITIALIZE
        world := world |- [163,221,11,71];
        world := world |- [196,219,200,163];
        world := world |- [128,226,103,67];
        world := world |- [142,150,3,77];
        world := world |- [128,195,54,161];
        world := world |- [147,126,66,17];
        if [n1,n2,n3,n4] \notin world then
              world := world |- [n1,n2,n3,n4];
        fi

        %% triger initialization of all components
        fire output P.systemInitialize(world);

        %% bind to server socket
        %% everyone sets up their server socket
        fire output RM.TCP_bind(myIp);
        fire output CM.TCP_respBind(error,myIp);

        %% connect to each other
        %% create connections to the server
        for n:Nat where n<len(world)do
           fire output SM.TCP_senderOpen(world[n],port);
           follow SM.DELAY duration 200;
           %% accept only on new connections
           fire output RM.TCP_accept;

           %% listen and accept
           fire output CM.TCP_respAccept(error);
           if error ~= nil() then
              print val(error);
           fi
           isconn :=false;
           fire output CM.TCP_respSenderOpen(world[n],port,isconn);
        od

        fire internal P.init;

        for i:Nat where i<runs do
           follow P.time duration 1;

           fire internal P.chooseToFail;
           fire internal P.chooseToRecover;
           fire internal P.checkLeader;
           fire internal P.checkLeaderTimeout;
           fire internal P.electionTimeout;
           fire internal P.announcementLeaderTimeout;

           % -- SEND
           dowhile := true;
           while (dowhile = true)do
              ms := nil();
              fire output P.SEND(ms);
              if(ms ~= nil()) then
                 fire output SM.TCP_write(ms, val(ms).sender, val(ms).destination);
                 print val(ms);
              else
                 dowhile :=false;
              fi
```

```
        od

        follow SM.DELAY duration 200;

        % -- RECEIVE

   fire output RM.TCP_read;
        dowhile :=true;
        while (dowhile = true)do
           mr := nil();
           fire output CM.TCP_respRead(mr);
           if(mr~=nil())then
              fire output RM.RECEIVE(mr);
              print val(mr);
           else
              dowhile :=false;
           fi
        od
     od
   od
```

B.4 Προδιαγραφή Τροποποιημένου Αλγορίθμου με Τοπολογία Ισχυρά Συνδεδεμένου Γράφου

B.4.1 Προδιαγραφή Λεξιλογίου (Vocabulary.tioa)

```
%%% :Algorithm vocabs

vocabulary algorithm_voc
       types
              Message: Tuple[messageType : Nat,leaderId:Nat]
end


%%% :MPI Channel vocabs

vocabulary mpi_status_voc
       types mpi_status
       operators
              MPI_Iprobe : Nat -> Null[mpi_status],
              MPI_Test : mpi_status ->Bool
end

vocabulary mpi_message_voc
       imports algorithm_voc, mpi_status_voc
              types mpi_message : Tuple[message:Message, sender:Nat,
destination:Nat]
              operators
                     MPI_Irecv: mpi_status, Nat -> mpi_message
end


vocabulary mpi_request_voc
       imports mpi_message_voc
              types mpi_request
              operators
```

```
                    MPI_Isend: mpi_message, Nat -> Null[mpi_request],
                    MPI_Barrier : -> Bool
end

vocabulary mpi_voc
      operators
            MPI_Rank : -> Nat,
            MPI_Size : -> Nat
end
```

B.4.2 Προδιαγραφή Αυτομάτου Διεργασίας (Algorithm4a_process.tioa)

```
automaton Algorithm4a_Process(timeForElection:DiscreteReal,
timeForAnouncementLeader:DiscreteReal, timeForCheckLeader:DiscreteReal,
possibilityToFail: Int, possibilityToRecover : Int, rank :Nat)
signature
            input RECEIVE (m:Null[mpi_message])
            output SEND (m:Null[mpi_message])
            internal init,checkLeader
            internal chooseToFail, chooseToRecover
            internal checkLeaderTimeout, electionTimeout,
announcementLeaderTimeout
states
            % The physcal clock of the automaton
            clock : DiscreteReal := 0;
            % The physical clock to check leader process
            checkLeaderTimer : DiscreteReal := 0
            % The physical clock to elect leader
            electionTimer : DiscreteReal := 0;
            % The physical clock to announce leader
            announcementLeaderTimer : DiscreteReal := 0;
            % An empty sequence of mpi messages
            msgs :Seq[Null[mpi_message]] := {};
            % A counter for the messages that a process has send
            messagesCounter : Int := 0;
            % A counter for the messages that sent for election
            %messagesForElection : Int :=0;
            % Indicates that the process is active
            processIsActive : Bool := true;
            % Indicates the rank of this process
            processRank : Nat := rank;
            % Indicates the id of the leader process
            leaderId : Nat:= 0;
            % Indicates the id of the Election Commission
            electionCommissionId :Nat := 0;
            % Indicates the id of the process which send election message
            electionProcessNumber :Nat := 0;
            % Indicates the highest id of all the processes
            highestProcessNumber :Nat := 0;
            % Tuple which indicates the possible status of the process
            status : Tuple[thereIsLeader:Bool,
election:Bool,waitForLeaderAnnouncement:Bool] :=[false,false,false];
            % Message to election a leader
            ElectionMessage : Nat := 1;
            % Message to reply in election message
            ReplyElectionMessage : Nat := 2;
            % Message to announce the leader
            AnnouncementLeaderMessage :Nat := 3;
            % Message to check the leader
            CheckLeaderMessage : Nat := 4;
            % Message to reply in check leader message
```

```
            ReplyCheckLeaderMessage : Nat := 5;
            % Indicates query message
            QueryMessage : Nat := 6;
            % Message to check a process if is alive
            CheckProcessMessage : Nat := 7;
            % Message to reply in check process message
            ReplyCheckProcessMessage : Nat := 8;
transitions

      input RECEIVE(m)
         locals iAmLeader : Bool := false;
         eff
            if(processIsActive /\ rank ~= electionCommissionId)then
               if(m ~= nil())then

                  if(val(m).message.messageType = ReplyElectionMessage)then
                  electionTimer := 0;
                     status := [false,false,true];
                     announcementLeaderTimer := clock;
                  fi;

                  if(val(m).message.messageType = AnnouncementLeaderMessage)then
                     announcementLeaderTimer := 0;
                     status := [true,false,false];
                     leaderId := val(m).message.leaderId;
                  fi;

                  if(val(m).message.messageType = CheckLeaderMessage)then
                     msgs := msgs |- embed([[ReplyCheckLeaderMessage,MPI_Size()],
rank, val(m).sender]);                              fi;

                  if(val(m).message.messageType = ReplyCheckLeaderMessage)then
                     checkLeaderTimer := 0;
                  fi;

                  if(val(m).message.messageType = CheckProcessMessage )then
                     msgs := msgs |-
embed([[ReplyCheckProcessMessage,MPI_Size()], rank, val(m).sender]);
                  fi;

               fi;
            fi;


      output SEND (m)
         pre
            processIsActive;
            msgs ~= {};
            m = head(msgs);
            rank ~= electionCommissionId;
         eff
            msgs := tail(msgs);
            if(val(m).message.messageType = ElectionMessage)then
               electionTimer := clock;
            fi;
            if(val(m).message.messageType = CheckLeaderMessage)then
               checkLeaderTimer := clock;
            fi;
            messagesCounter := messagesCounter + 1;

      internal init
         pre
```

```
            rank ~= (MPI_Size()-1);
            processIsActive;
         eff
            processIsActive := true;
            status := [true,false,false];
            leaderId := 0;
            for n:Nat where n <= (MPI_Size()-1)do
               if(n = (MPI_Size()-1))then
                  electionCommissionId := n;
               else
                  if(n > leaderId)then
                     leaderId := n;
                  fi;
               fi;
            od;

      internal checkLeader
         pre
            processIsActive;
            leaderId ~= rank;
            rank ~= electionCommissionId;
            status = [true,false,false];
         eff
            msgs := msgs |-
embed([[CheckLeaderMessage,MPI_Size()],rank,leaderId]);


      internal chooseToFail
         locals randomValueToFail : Int := choose n where (n>=1 /\ n <= 100);
         pre
            rank ~= electionCommissionId;
         eff
            if(processIsActive )then
               if(randomValueToFail >=0 /\ randomValueToFail <=
possibilityToFail)then
                  processIsActive := false;
               fi;
            fi;


      internal chooseToRecover
         locals randomValueToRecover : Int := choose n where (n>=1 /\ n <= 100);
         pre
            rank ~= electionCommissionId;
         eff
            if(processIsActive = false)then

               if(randomValueToRecover >=0 /\  randomValueToRecover <=
possibilityToRecover)then
                  processIsActive := true;
                  msgs := msgs |-
embed([[QueryMessage,MPI_Size()],rank,electionCommissionId]);
               fi;

            fi;

      internal checkLeaderTimeout
         locals iAmLeader : Bool := false;
         pre
            clock > (checkLeaderTimer + timeForCheckLeader);
            processIsActive;
            rank ~= electionCommissionId;
```

```
            rank ~= leaderId;
            status = [true,false,false];
         eff
            status := [false,true,false];
            msgs := msgs |- embed([[ElectionMessage,MPI_Size()], rank,
electionCommissionId]);
            checkLeaderTimer := 0;


      internal electionTimeout
         pre
             clock > (electionTimer + timeForElection);
            processIsActive;
            rank ~= electionCommissionId;
            status = [false,true,false];
         eff
            msgs := msgs |- embed([[ElectionMessage,MPI_Size()], rank,
electionCommissionId]);
            electionTimer := 0;


      internal announcementLeaderTimeout
         pre
            clock > (announcementLeaderTimer + timeForAnouncementLeader);
            processIsActive;
            rank ~= electionCommissionId;
            status = [false,false,true];
         eff
            msgs := msgs |- embed([[ElectionMessage,MPI_Size()], rank,
electionCommissionId]);
            announcementLeaderTimer := 0;
trajectories
      trajdef time
             evolve d(clock) = 1;
```

B.4.3 Προδιαγραφή Αυτομάτου Διεργασίας ElectionCommission στην 1η Εκδοχή (ElectionCommission.tioa)

```
automaton ElectionCommission(timeForVerifyLeader: DiscreteReal,
timeForReplyAliveProcess: DiscreteReal,rank:Nat)
signature
      input RECEIVE (m:Null[mpi_message])
      output SEND (m:Null[mpi_message])
      internal init
      internal checkLeaderTimeout, checkProcessTimeout
states
      % The physcal clock of the automaton
      clock : DiscreteReal := 0;
      % The physical clock to check leader process
      checkLeaderTimer : DiscreteReal := 0;
      % The physical clock to check process
      checkProcessTimer : DiscreteReal := 0;
      % An empty sequence of mpi messages
      msgs :Seq[Null[mpi_message]] := {};
      % A counter for the messages that a process has send
      messagesCounter : Int := 0;
      % Indicates the id of the leader process
      leaderId : Nat:= 0;
      % Indicates the rank of this process
```

```
        processRank : Nat := rank;
        % Indicates the id of the Election Commission
        electionCommissionId :Nat := MPI_Size();
        % Indicates the id of the process which send election message
        electionProcessNumber :Nat := 0;
        % Indicates the highest id of all the processes
        highestProcessNumber :Nat := 0;
        % Indicates the number of alive processes
        numberOfAliveProcesses : Int := 0;
        % Indicates the number of reply messages
        numberOfReplyMessages : Int := 0;
        % Indicates if there is a leader in the system
        thereIsLeader:Bool := true;
        % Indicates that an election message had received
        electionFlag : Bool := false;
        % Indicates that a query message had received
        queryFlag : Bool := false;
        % Message to election a leader
        ElectionMessage : Nat := 1;
        % Message to reply in election message
        ReplyElectionMessage : Nat := 2;
        % Message to announce the leader
        AnnouncementLeaderMessage :Nat := 3;
        % Message to check the leader
        CheckLeaderMessage : Nat := 4;
        % Message to reply in check leader message
        ReplyCheckLeaderMessage : Nat := 5;
        % Indicates query message
        QueryMessage : Nat := 6;
        % Message to check a process if is alive
        CheckProcessMessage : Nat := 7;
        % Message to reply in check process message
        ReplyCheckProcessMessage : Nat := 8;
transitions
        input RECEIVE (m)
           eff
              if(rank = electionCommissionId)then
                 if(m ~= nil())then

                    if(val(m).message.messageType = ElectionMessage)then
                       electionFlag := true;
                       msgs := msgs |- embed([[ReplyElectionMessage,MPI_Size()],
electionCommissionId,val(m).sender]);
                       msgs := msgs |- embed([[CheckLeaderMessage,MPI_Size()],
electionCommissionId,leaderId]);
                       electionProcessNumber := val(m).sender;
                    fi;

                    if(val(m).message.messageType = ReplyCheckLeaderMessage)then
                       checkLeaderTimer := 0;
                       thereIsLeader := true;
                       electionFlag := false;
                       queryFlag := false;
                       msgs := msgs |- embed([[AnnouncementLeaderMessage,leaderId],
electionCommissionId,electionProcessNumber]);
                    fi;

                    if(val(m).message.messageType = QueryMessage)then
                       queryFlag := true;
                       if(val(m).sender > leaderId)then
                          leaderId := val(m).sender;
                          for n :Nat where n < (MPI_Size()-1) do
```

```
                        msgs := msgs |-
embed([[AnnouncementLeaderMessage,leaderId], electionCommissionId,n]);
                    od;

                    msgs := msgs |-
embed([[AnnouncementLeaderMessage,leaderId], electionCommissionId,n]);
                    od;
                else
                    msgs := msgs |-
embed([[AnnouncementLeaderMessage,leaderId], electionCommissionId,val(m).sender]);

                fi;
            fi;

            if(val(m).message.messageType = ReplyCheckProcessMessage)then
                checkProcessTimer := 0;
                highestProcessNumber := val(m).sender;
                leaderId := highestProcessNumber;
                queryFlag := false;
                for n :Nat where n < (MPI_Size()-1) do
                    msgs := msgs |-
embed([[AnnouncementLeaderMessage,leaderId], electionCommissionId,n]);
                od;

            fi;
        fi;
    fi;


output SEND (m)
    pre
        msgs ~= {};
        m = head(msgs);
        rank = electionCommissionId;
    eff
        msgs := tail(msgs);
        if(val(m).message.messageType = CheckLeaderMessage)then
            checkLeaderTimer := clock;
        fi;
        if(val(m).message.messageType = CheckProcessMessage)then
            checkProcessTimer := clock;
        fi;
        messagesCounter := messagesCounter + 1;


internal init
    pre
        rank = (MPI_Size()-1);
    eff
        leaderId := 0;
        for n:Nat where n < (MPI_Size()-1) do
            if(n > leaderId)then
                leaderId := n;
            fi;
        od;
        electionCommissionId := rank;

internal checkLeaderTimeout
    pre
        clock > (checkLeaderTimer + timeForVerifyLeader);
        thereIsLeader;
        rank = electionCommissionId;
```

```
            electionFlag = true;
         eff
            thereIsLeader := false;
            highestProcessNumber := 0;

             for n:Nat where n < leaderId do
               if(n > highestProcessNumber)then
                  highestProcessNumber := n;
               fi;
             od;
             msgs := msgs |- embed([[CheckProcessMessage,MPI_Size()],
electionCommissionId,highestProcessNumber]);
             checkLeaderTimer := 0;


      internal checkProcessTimeout
         locals highestProcess : Nat :=0;
         pre
            clock > (checkProcessTimer + timeForReplyAliveProcess);
            thereIsLeader = false;
            rank = electionCommissionId;
            queryFlag = true;
         eff
            for n:Nat where n < (MPI_Size()-1) do
               if(n < highestProcessNumber)then
                  highestProcess := n;
               fi;
             od;
             msgs := msgs |- embed([[CheckProcessMessage,MPI_Size()],
electionCommissionId,highestProcess]);
             checkProcessTimer := 0;
trajectories
      trajdef time
             evolve d(clock) = 1;
```

B.4.4 Προδιαγραφή Αυτομάτου Διεργασίας ElectionCommission στην 2η Εκδοχή (ElectionCommission.tioa)

```
automaton ElectionCommission(timeForVerifyLeader: DiscreteReal,
timeForReplyAliveProcess: DiscreteReal,rank:Nat)
signature
      input RECEIVE (m:Null[mpi_message])
      output SEND (m:Null[mpi_message])
      internal init
      internal checkLeaderTimeout, checkProcessTimeout
states
      % The physcal clock of the automaton
      clock : DiscreteReal := 0;
      % The physical clock to check leader process
      checkLeaderTimer : DiscreteReal := 0;
      % The physical clock to check process
      checkProcessTimer : DiscreteReal := 0;
      % An empty sequence of mpi messages
      msgs :Seq[Null[mpi_message]] := {};
      % A counter for the messages that a process has send
      messagesCounter : Int := 0;
      % Indicates the id of the leader process
      leaderId : Nat:= 0;
      % Indicates the rank of this process
```

```
        processRank : Nat := rank;
        % Indicates the id of the Election Commission
        electionCommissionId :Nat := MPI_Size();
        % Indicates the id of the process which send election message
        electionProcessNumber :Nat := 0;
        % Indicates the highest id of all the processes
        highestProcessNumber :Nat := 0;
        % Indicates the number of alive processes
        numberOfAliveProcesses : Int := 0;
        % Indicates the number of reply messages
        numberOfReplyMessages : Int := 0;
        % Indicates if there is a leader in the system
        thereIsLeader:Bool := true;
        % Indicates that an election message had received
        electionFlag : Bool := false;
        % Indicates that a query message had received
        queryFlag : Bool := false;
        % Message to election a leader
        ElectionMessage : Nat := 1;
        % Message to reply in election message
        ReplyElectionMessage : Nat := 2;
        % Message to announce the leader
        AnnouncementLeaderMessage :Nat := 3;
        % Message to check the leader
        CheckLeaderMessage : Nat := 4;
        % Message to reply in check leader message
        ReplyCheckLeaderMessage : Nat := 5;
        % Indicates query message
        QueryMessage : Nat := 6;
        % Message to check a process if is alive
        CheckProcessMessage : Nat := 7;
        % Message to reply in check process message
        ReplyCheckProcessMessage : Nat := 8;
transitions
        input RECEIVE (m)
           eff
              if(rank = electionCommissionId)then
                 if(m ~= nil())then

                    if(val(m).message.messageType = ElectionMessage)then
                       electionFlag := true;
                       if(startElectionTime = 0 /\ startFlag = 0)then
                          startElectionTime := clock;
                          startFlag := 1;
                       fi;
                       msgs := msgs |- embed([[ReplyElectionMessage,MPI_Size()],
electionCommissionId,val(m).sender]);
                       msgs := msgs |- embed([[CheckLeaderMessage,MPI_Size()],
electionCommissionId,leaderId]);
                       electionProcessNumber := val(m).sender;
                    fi;

                    if(val(m).message.messageType = ReplyCheckLeaderMessage)then
                       checkLeaderTimer := 0;
                       thereIsLeader := true;
                       electionFlag := false;
                       queryFlag := false;
                       msgs := msgs |- embed([[AnnouncementLeaderMessage,leaderId],
electionCommissionId,electionProcessNumber]);
                    fi;

                    if(val(m).message.messageType = QueryMessage)then
```

```
                        queryFlag := true;
                        if(val(m).sender > leaderId)then
                           leaderId := val(m).sender;
                           for n :Nat where n < (MPI_Size()-1) do
                              msgs := msgs |-
embed([[AnnouncementLeaderMessage,leaderId], electionCommissionId,n]);
                           od;
                        else
                           msgs := msgs |-
embed([[AnnouncementLeaderMessage,leaderId], electionCommissionId,val(m).sender]);

                        fi;
                     fi;

                     if(val(m).message.messageType = ReplyCheckProcessMessage)then
                        checkProcessTimer := 0;
                        numberOfReplyMessages := numberOfReplyMessages + 1;
                        queryFlag := false;

                        if(val(m).sender > highestProcessNumber)then
                           highestProcessNumber :=val(m).sender;
                        fi;

                        if(numberOfReplyMessages = numberOfAliveProcesses )then
                           leaderId := highestProcessNumber;
                           for n :Nat where n<(MPI_Size()-1) do
                              msgs := msgs |-
embed([[AnnouncementLeaderMessage,leaderId], electionCommissionId,n]);
                           od;
                           numberOfReplyMessages :=0;
                        fi;

                     fi;
                  fi;
               fi;


         output SEND (m)
            pre
               msgs ~= {};
               m = head(msgs);
               rank = electionCommissionId;
            eff
               msgs := tail(msgs);
               if(val(m).message.messageType = CheckLeaderMessage)then
                  checkLeaderTimer := clock;
               fi;
               if(val(m).message.messageType = CheckProcessMessage)then
                  checkProcessTimer := clock;
               fi;
               messagesCounter := messagesCounter + 1;


         internal init
            pre
               rank = (MPI_Size()-1);
            eff
               leaderId := 0;
               for n:Nat where n < (MPI_Size()-1) do
                  if(n > leaderId)then
                     leaderId := n;
                  fi;
```

```
            od;
            electionCommissionId := rank;


      internal checkLeaderTimeout
         pre
            clock > (checkLeaderTimer + timeForVerifyLeader);
            thereIsLeader;
            rank = electionCommissionId;
            electionFlag = true;
         eff
            thereIsLeader := false;
            numberOfAliveProcesses :=numberOfAliveProcesses -1;
            checkLeaderTimer := 0;
            for n :Nat where n<(MPI_Size()-1) do
               if(n ~= leaderId)then
                  msgs := msgs |- embed([[CheckProcessMessage,MPI_Size()],
electionCommissionId,n]);
               fi;
            od;


      internal checkProcessTimeout
         locals highestProcess : Nat :=0;
         pre
            clock > (checkProcessTimer + timeForReplyAliveProcess);
            thereIsLeader = false;
            rank = electionCommissionId;
            queryFlag = true;
         eff
            numberOfAliveProcesses :=numberOfAliveProcesses -1;
            checkProcessTimer := 0;
trajectories
      trajdef time
            evolve d(clock) = 1;
```

Β.4.5 Προδιαγραφή Αυτομάτου Σύνθεσης (Algorithm4.tioa)

```
%%% .: Vocabulary :.
include "Vocabulary.tioa"
imports algorithm_voc, mpi_status_voc, mpi_message_voc, mpi_request_voc, mpi_voc

%%% .: MPI Mediator Automata :.
include "SendMediator.tioa"
include "ReceiveMediator.tioa"

%%% .: Algorithm Automata :.
include "Algorithm4a_Process.tioa"
include "ElectionCommission.tioa"
automaton Algorithm4a(timeForElection:DiscreteReal,
timeForAnouncementLeader:DiscreteReal,timeForCheckLeader:DiscreteReal,
             possibilityToFail: Int, possibilityToRecover : Int,
timeForVerifyLeader: DiscreteReal, timeForReplyAliveProcess : DiscreteReal)
components
      P: Algorithm4a_Process(timeForElection,timeForAnouncementLeader,
timeForCheckLeader, possibilityToFail, possibilityToRecover,
                  MPI_Rank() );
      EC : ElectionCommission(timeForVerifyLeader,
timeForReplyAliveProcess,MPI_Rank());
```

```
        SM: SendMediator;
        RM: ReceiveMediator;
states
        m:Null[mpi_message] :=nil();
        runs : Nat :=MPI_SIZE()**2;
do
        fire internal P.init;
        fire internal EC.init;

        for i:Nat where i<runs do
         follow P.time duration 1;

                fire internal P.chooseToFail;
                fire internal P.chooseToRecover;

                fire internal P.checkLeader;

                fire internal P.checkLeaderTimeout;
                fire internal EC.checkLeaderTimeout;
                fire internal EC.checkProcessTimeout;
                fire internal P.electionTimeout;
                fire internal P.announcementLeaderTimeout;

                % Send Messages
                for j:Nat where j< MPI_Size() do
                        fire output P.SEND(m);
                        fire output EC.SEND(m);
                od

                follow SM.DELAY duration (MPI_Size() * 10);

                % Receive Messages
                for j:Nat where j<MPI_Size() do
                        fire input RM.probe(j);
                        fire output RM.RECEIVE(m);
                od
        od

 od
```

Παράρτημα Γ

Στο Παράρτημα αυτό, δίνονται οι οδηγίες εκτέλεσης των προγραμμάτων που χρησιμοποιούν τη βιβλιοθήκη MPJ Express, σε Multicore και Cluster Configuration.

Γ.1 Εκτέλεση MPJ Express προγραμμάτων σε Multicore Configuration

1. Κατεβάζουμε τη βιβλιοθήκη MPJ Express από την επίσημη ιστοσείδα της και την αποσυμπιέζουμε.
2. Θέτουμε τις μεταβλητές MPJ_HOME και PATH, με τις εντολές :
   export MPJ_HOME/path_to_mpj/
   export PATH=$PATH:$MPJ_HOME/bin
3. Γράφουμε το MPJ Express πρόγραμμα και το αποθηκεύουμε (Algorithm1.java) .
4. Compile : javac -cp .:$MPJ_HOME/lib/mpj.jar Algorithm1.java .
5. Execute : mpjrun.sh -np <number_of_processes> Algorithm1.

Γ.2 Εκτέλεση MPJ Express προγραμμάτων σε Cluster Configuration

1. Κατεβάζουμε τη βιβλιοθήκη MPJ Express από την επίσημη ιστοσείδα της και την αποσυμπιέζουμε.
2. Θέτουμε τις μεταβλητές MPJ_HOME και PATH, με τις εντολές :
   export MPJ_HOME/path_to_mpj/
   export PATH=$PATH:$MPJ_HOME/bin
3. Γράφουμε το MPJ Express πρόγραμμα και το αποθηκεύουμε (Algorithm1.java)
4. Γράφουμε ένα αρχείο machines (machines.txt) στο οποίο συμπεριλαμβάνουμε τα ονόματα των μηχανών ή τις IP διευθύνσεις όλων των μηχανών στις οποίες θα τρέξουν οι διεργασίες παράλληλα.
5. Ξεκινούμε τα daemons των μηχανών που θα χρησιμοποιήσουμε και βρίσκονται στο αρχείο machines, με την εντολή : mpjboot machines .
6. Compile : javac -cp .:$MPJ_HOME/lib/mpj.jar Algorithm1.java .
7. Execute : mpjrun.sh -np <number_of_processes> -dev niodev Algorithm1.
8. Σταματάμε τα daemons των μηχανών, με την εντολή : mpjhalt machines .

Σημείωση: Σε περίπτωση που οι διεργασίες που θα εκτελεστούν σε ένα πρόγραμμα είναι n και είναι λιγότερες από τον αριθμό των μηχανών που ορίζονται στο αρχείο machines, τότε θα τρέξει μία διεργασία στις πρώτες n μηχανές. Σε περίπτωση που οι διεργασίες είναι μεγαλύτερες από τον αριθμό των μηχανών, τότε θα εκτελεστεί μία διεργασία σε κάθε μηχανή και οι υπόλοιπες θα ανατεθούν ξανά στις μηχανές ξεκινώντας από την πρώτη μηχανή που βρίσκεται στο αρχείο. Για παράδειγμα εάν υπάρχουν 4 μηχανές με τα ονόματα machine1, machine2, machine3 και machine4, βρίσκονται με αυτή τη σειρά στο αρχείο machines και υπάρχουν 2 διεργασίες στο σύστημα, τότε η μία δεργασία θα τρέξει στο machine1 και η άλλη στο machine2. Εάν υπάρχουν 4 διεργασίες, κάθε machine θα πάρει από μία διεργασία. Ενώ εάν υπάρχουν 6 διεργασίες, οι μηχανές machine3 και machine4 θα πάρουν από 1 διεργασία ενώ οι μηχανές machine1 και machine2 θα πάρουν από 2.

Παράρτημα Δ

Στο Παράρτημα αυτό, παραθέτονται τα αποτελέσματα της πειραματικής αξιολόγησης των αλγορίθμων στο τοπικό δίκτυο του Πανεπιστημίου Κύπρου και στο κατανεμημένο δίκτυο PlanetLab, τα οποία χρησιμοποιήθηκαν στο Κεφάλαιο 5.

Δ.1 Αποτελέσματα Αλγορίθμου με Τοπολογία Δέντρου

Αριθμός Κόμβων	*Συνολικός Χρόνος Εκτέλεσης*	*Συνολικός Αριθμός Μηνυμάτων*
4	0,4615	6
6	0,7775	10
8	1,085	14
12	1,8435	22
18	3,087	33

Δ.2 Αποτελέσματα Αλγορίθμου με Τοπολογία Δακτυλίου

Καλύτερη Περίπτωση

Αριθμός Κόμβων	*Συνολικός Χρόνος Εκτέλεσης*	*Συνολικός Αριθμός Μηνυμάτων*
4	0,7685	11
6	1,5445	17
8	2,2245	23
12	4,656	35
18	9,201	53

Μέση Περίπτωση

Αριθμός Κόμβων	*Συνολικός Χρόνος Εκτέλεσης*	*Συνολικός Αριθμός Μηνυμάτων*

4	0,8195	12
6	1,572	21
8	2,2385	27
12	4,841	45
18	9,31	75

Χειρότερη Περίπτωση

Αριθμός Κόμβων	*Συνολικός Χρόνος Εκτέλεσης*	*Συνολικός Αριθμός Μηνυμάτων*
4	0,838	14
6	1,5795	27
8	2,361	44
12	5,3925	90
18	10,1935	189

Δ.3 Αποτελέσματα Αλγορίθμου με Τοπολογία Ισχυρά Συνδεδεμένου Γράφου, υλοποιημένου με το κανάλι επικοινωνίας MPI

Σενάριο 1

Αριθμός Κόμβων	*Συνολικός Χρόνος Εκτέλεσης*	*Συνολικός Αριθμός Μηνυμάτων*
4	6,48515	29,5
6	8,126	97
8	10,628	119
12	16,2035	263
18	25,26	484

Σενάριο 2

Αριθμός Κόμβων	*Συνολικός Χρόνος Εκτέλεσης*	*Συνολικός Αριθμός Μηνυμάτων*
4	6,40625	24

6	8,088	53
8	10,58475	122
12	16,14025	208,75
18	25,1535	356

Σενάριο 3

Αριθμός Κόμβων	*Συνολικός Χρόνος Εκτέλεσης*	*Συνολικός Αριθμός Μηνυμάτων*
4	6,4945	39,5
6	8,18775	62,5
8	10,708	119
12	16,09425	209,5
18	25,85275	399,5

Σενάριο 4

Αριθμός Κόμβων	*Συνολικός Χρόνος Εκτέλεσης*	*Συνολικός Αριθμός Μηνυμάτων*
4	6,21125	31
6	8,053	64
8	10,8165	91,5
12	16,46025	193
18	25,438	304,5

Δ.4 Αποτελέσματα Αλγορίθμου με Τοπολογία Ισχυρά Συνδεδεμένου Γράφου, υλοποιημένου με το κανάλι επικοινωνίας TCP και αξιολόγηση στο PlanetLab

Σενάριο 1

Αριθμός Κόμβων	*Συνολικός Χρόνος Εκτέλεσης*	*Συνολικός Αριθμός Μηνυμάτων*
4	15,246	47,5
6	19,2	82,5
8	27,7855	123,5

12	55,746	179

Σενάριο 2

Αριθμός Κόμβων	*Συνολικός Χρόνος Εκτέλεσης*	*Συνολικός Αριθμός Μηνυμάτων*
4	12,9215	42
6	21,047	72
8	35,35	129
12	59,687	279

Σενάριο 3

Αριθμός Κόμβων	*Συνολικός Χρόνος Εκτέλεσης*	*Συνολικός Αριθμός Μηνυμάτων*
4	15,7755	53
6	22,579	96,5
8	29,557	115
12	58,66	220

Σενάριο 4

Αριθμός Κόμβων	*Συνολικός Χρόνος Εκτέλεσης*	*Συνολικός Αριθμός Μηνυμάτων*
4	12,469	34,5
6	25,984	81
8	32,3335	129
12	64,5	285

Σενάριο 5

Αριθμός Κόμβων	*Συνολικός Χρόνος Εκτέλεσης*	*Συνολικός Αριθμός Μηνυμάτων*
4	10,4	29
6	20,372	50
8	27,554	81
12	42,357	190

16	75,82	270
24	155,226	505
32	303,635	886

Δ.5 Αποτελέσματα Πρώτης Εκδοχής Τροποποιημένου Αλγορίθμου με Τοπολογία Ισχυρά Συνδεδεμένου Γράφου

Σενάριο 1

Αριθμός Κόμβων	*Συνολικός Χρόνος Εκτέλεσης*	*Συνολικός Αριθμός Μηνυμάτων*
4	6,76525	41,5
6	8,95675	73
8	11,319	103
12	15,43225	150
18	22,21525	195

Σενάριο 2

Αριθμός Κόμβων	*Συνολικός Χρόνος Εκτέλεσης*	*Συνολικός Αριθμός Μηνυμάτων*
4	6,6375	45
6	8,9105	75
8	11,034	92,5
12	15,39375	118,5
18	22,148	153,5

Σενάριο 3

Αριθμός Κόμβων	*Συνολικός Χρόνος Εκτέλεσης*	*Συνολικός Αριθμός Μηνυμάτων*
4	6,815	25
6	9,11575	51
8	11,09825	73,5

12	15,411	97,5
18	22,23975	133

Σενάριο 4

Αριθμός Κόμβων	*Συνολικός Χρόνος Εκτέλεσης*	*Συνολικός Αριθμός Μηνυμάτων*
4	6,78925	53
6	9,03375	56,5
8	11,2285	85
12	15,3665	113
18	22,2725	173,5

Δ.6 Αποτελέσματα Δεύτερης Εκδοχής Τροποποιημένου Αλγορίθμου με Τοπολογία Ισχυρά Συνδεδεμένου Γράφου

Σενάριο 1

Αριθμός Κόμβων	*Συνολικός Χρόνος Εκτέλεσης*	*Συνολικός Αριθμός Μηνυμάτων*
4	6,7765	23
6	8,9365	26,5
8	11,3795	62,5
12	15,5525	104,5
18	22,16325	164

Σενάριο 2

Αριθμός Κόμβων	*Συνολικός Χρόνος Εκτέλεσης*	*Συνολικός Αριθμός Μηνυμάτων*
4	6,72025	42
6	8,96175	70
8	11,08675	83,5
12	15,51175	140,5
18	22,3395	244,5

Σενάριο 3

Αριθμός Κόμβων	*Συνολικός Χρόνος Εκτέλεσης*	*Συνολικός Αριθμός Μηνυμάτων*
4	6,64252	30
6	8,951	39,5
8	11,245	54,5
12	15,46375	91,5
18	22,47425	181

Σενάριο 4

Αριθμός Κόμβων	*Συνολικός Χρόνος Εκτέλεσης*	*Συνολικός Αριθμός Μηνυμάτων*
4	6,8425	22,5
6	8,7985	31
8	11,313	80
12	15,42	140
18	22,20325	204,5