Ατομική Διπλωματική Εργασία

ΠΡΟΔΙΑΓΡΑΦΗ ΚΑΙ ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΧΡΟΝΙΣΜΕΝΟΥ ΚΑΤΑΝΕΜΗΜΕΝΟΥ ΑΛΓΟΡΙΘΜΟΥ ΔΙΑΧΕΙΡΙΣΗΣ ΑΡΧΕΙΩΝ ΜΕ ΤΗ ΧΡΗΣΗ ΤΟΥ ΕΡΓΑΛΕΙΟΥ TEMPO

Σωτηρούλα Ιωάννου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2014

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Προδιαγραφή και Πειραματική Αξιολόγηση Χρονισμενου Κατανεμημένου Αλγόριθμου Διαχείρισης Αρχείων με τη Χρήση του Εργαλείου TEMPO

Σωτηρούλα Ιωάννου

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 2014

Ευχαριστίες

Θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες, στον επιβλέποντα καθηγητή μου Δρ. Χρύση Γεωργίου για την καθοδήγηση και την υποστήριξη που μου έδειξε καθ' όλη τη διάρκεια της διπλωματικής μου εργασίας. Επίσης, θα ήθελα να τον ευχαριστήσω για την ευκαιρία που μου έδωσε και την εμπιστοσύνη που έδειξε στο πρόσωπό μου, αλλά και για την άψογη συνεργασία που είχαμε όλο αυτό το διάστημα.

Ακολούθως, θα ήθελα να εκφράσω τις ευχαριστίες μου στο Χρήστο Πλουτάρχου για τη βοήθεια του σε κρίσιμο σημείο της διπλωματικής εργασίας.

Επίσης θα ήθελα να ευχαριστήσω την οικογένεια μου για την αγάπη και την κατανόηση που μου έδειξαν αυτή τη δύσκολη περίοδο των σπουδών μου.

Τέλος, θα ήθελα να πω ένα μεγάλο ευχαριστώ στους φίλους μου, και ιδιαίτερα στο Γιάννη, για τη συμπαράσταση και την υπομονή που έδειχναν σε όλη τη διάρκεια της διπλωματικής μου εργασίας.

Περίληψη

Στην παρούσα Διπλωματική Εργασία, έχει γίνει η προδιαγραφή και η πειραματική αξιολόγηση του χρονισμένου κατανεμημένου αλγόριθμου διαχείρισης αρχείων Layered Data Replication (LDR). Η προδιαγραφή του αλγορίθμου έγινε στη γλώσσα TIOA (Timed Input/Output Automata) με τη χρήση του εργαλείου TEMPO το οποίο μοντελοποιεί αυτή τη γλώσσα. Ο αλγόριθμος αυτός μας δίνει τη δυνατότητα αντιγραφής μεγάλων δεδομένων, όπως για παράδειγμα αντιγραφή αρχείων. Η κύρια ιδέα του αλγόριθμου, είναι ότι διατηρεί τα αντίγραφα των αρχείων σε ξεχωριστό χώρο, στους αντιγραφείς (replicas), απ' ότι τις πληροφορίες που αφορούν την τοποθεσία των ενημερωμένων (up-to-date) αντιγράφων οι οποίες αποθηκεύονται στα ευρετήρια (directories). Για αυτό το λόγο, παρατηρούμε ότι εκτελεί τις περισσότερες λειτουργίες του, χρησιμοποιώντας τα μεταδεδομένα (metadata), παρά τα πραγματικά δεδομένα κάτι το οποίο μειώνει το κόστος επικοινωνίας. Επίσης, ο αλγόριθμος κάνει χρήση μόνο ορισμένων από τα πραγματικά δεδομένα και πιο συγκεκριμένα μόνο ένα αντίγραφο, κατά τη διάρκεια της ανάγνωσης και μόνο f+1 αντίγραφα, κατά τη διάρκεια της εγγραφής, όπου το f είναι ένα άνω όριο για τον αριθμό των αντιγράφων που μπορεί να είναι εσφαλμένοι. Ο αλγόριθμος δουλεύει σε οποιοδήποτε ασύγχρονο και αξιόπιστο δίκτυο ανταλλαγής μηνυμάτων, χωρίς να κάνει χρήση άλλων υψηλού επιπέδου συναρτήσεων, όπως για παράδειγμα κατανεμημένο κλείδωμα ή ομαδική επικοινωνία.

Η υλοποίηση μας είναι κατάλληλη για LAN, αλλά και WAN και δίνει στο χρήστη τη δυνατότητα να γράψει όλα τα κοινά είδη αρχείων, συμπεριλαμβανομένων .docx, .txt, .pdf και .jpg. Δίνονται εγγυήσεις για αποδοτικότητα και συνέπεια, καθώς επίσης την εγγύηση στους χρήστες ότι θα διαβάζουν την πιο ενημερωμένη εκδοχή του αρχείου. Η πειραματική αξιολόγηση πραγματοποιήθηκε στο ασύγχρονο και ρεαλιστικό δίκτυο PlanetLab, χρησιμοποιώντας μηνύματα των 2048 bytes ως default block size και αρχεία μεγέθους μέχρι 1 MB.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός και Κίνητρο

Η ορθότητα και η αποδοτικότητα των κατανεμημένων αλγόριθμων αποτελεί μεγάλη πρόκληση στις μέρες μας. Έτσι, αναπτύχθηκαν τυπικές μεθόδοι για την προδιαγραφή και την απόδειξη της ορθότητας ενός αλγόριθμου. Το μοντέλο Χρονισμένων Αυτόματων Εισόδου/Εξόδου (TIOA) [2,3,4,8] αποτελεί μια από τις πιο δημοφιλής μεθόδους για την προδιαγραφή και επαλήθευση κατανεμημένων αλγόριθμων. Σκοπός της παρούσας Διπλωματικής Εργασίας, είναι η εκμάθηση της γλώσσας TIOA [12,17] και του εργαλείου TEMPO [1,3,10] για να ορίσουμε την προδιαγραφή του αλγόριθμου διαχείρισης αρχείων LDR. Ακολούθως, επιθυμούμε να τροποποιήσουμε την προδιαγραφή αυτή έτσι ώστε να διαχειρίζεται πραγματικά αρχεία (.docx, .pdf, .txt, .jpg) και τέλος να αξιολογήσουμε την υλοποίησή μας στο ρεαλιστικό και ασύγχρονο περιβάλλον PlanetLab [13,14].

Τα Κατανεμημένα Συστήματα [22,23], κατά τον Tanenbaum, van Steen [22], είναι μια συλλογή από ανεξάρτητες υπολογιστικές μονάδες που παρουσιάζονται στο χρήστη ως μια μοναδική οντότητα. Οι αλγόριθμοι που σχεδιάζονται για Κατανεμημένα Συστήματα,

ονομάζονται Κατανεμημένοι Αλγόριθμοι [22,23]. Πιο συγκεκριμένα οι αλγόριθμοι αυτοί είναι σχεδιασμένοι να τρέχουν σε πολλούς υπολογιστικούς κόμβους, οι οποίοι μπορεί να βρίσκονται σε απομακρυσμένες μεταξύ τους γεωγραφικές τοποθεσίες, με την απουσία κάποιου κεντρικού ελέγχου. Αντιθέτως, αυτοί οι υπολογιστικοί κόμβοι επικοινωνούν μεταξύ τους, είτε μέσω μιας κατανεμημένης κοινόχρηστης μνήμης, είτε μέσω ανταλλαγής μηνυμάτων και οι αποφάσεις που παίρνουν γίνονται τοπικά. Θεωρούνται πιο δύσκολοι τόσο στη σχεδίαση, όσο και στην κατανόηση τους σε σχέση με τους σειριακούς αλγόριθμους, εφόσον σκοπός μας είναι η σωστή συνεργασία των κόμβων αυτών, για την ορθή επίλυση του προβλήματος.

Η κατανεμημένη αποθήκευση αρχείων είναι πολύ σημαντική στις μέρες μας, εφόσον προσφέρει πλεονεκτήματα, όπως αξιοπιστία, διαθεσιμότητα και ανοχής σφαλμάτων. Τα κατανεμημένα συστήματα αποθήκευσης, έρχονται αντιμέτωπα με κάποιες προκλήσεις, όπως για παράδειγμα αυτήν της συνέπειας (consistency), δηλαδή από ποια αποθηκευτική οντότητα πρέπει να επιλέξει για να διαβάσει, έτσι ώστε να διαβάσει την πιο πρόσφατη τιμή. Αυτό συμβαίνει ειδικά σε περιπτώσεις ταυτοχρονισμού.

1.2 Μεθοδολογία

Για την ολοκλήρωση της παρούσας Διπλωματικής Εργασίας ακολουθήσαμε μια πολύ ξεκάθαρη μεθοδολογία από τα πρώιμα στάδια της. Αρχικά, έγινε εκτενής μελέτη για την κατανόηση των Χρονισμένων Αυτομάτων Εισόδου/Εξόδου (TIOA) [8,12], η οποία μας οδήγησε και στη μελέτη των Αυτομάτων Εισόδου/Εξόδου (IOA) [9,23] για την καλύτερη κατανόηση του πρώτου μοντέλου. Ακολούθως, έγινε εγκατάσταση του εργαλείου TEMPO [1] το οποίο είναι το εργαλείο που χρησιμοποιήσαμε για την προδιαγραφή του αλγορίθμου μας. Επίσης, μελετήθηκαν θέματα που σχετίζονται με τους χρονισμένους κατανεμημένους αλγόριθμους. Ένας από τους κυριότερους στόχους μας ήταν η εκμάθηση του μοντέλου TIOA και αυτό το πετύχαμε μέσα από τη μελέτη και τη προδιαγραφή παραδειγμάτων, τα οποία στη συνέχεια υλοποιήσαμε στο εργαλείο TEMPO. Αυτό συνέβαλε στην κατανόηση της λειτουργίας του εργαλείου. Για την προδιαγραφή των παραδειγμάτων ήταν αναγκαία και η μελέτη των καναλιών επικοινωνίας MPI [6] και TCP [18]. Στη συνέχεια, έγινε εκτενής μελέτη του αλγορίθμου Layered Data Replication (LDR) [15,16] και η προδιαγραφή του σε γλώσσα TIOA, χρησιμοποιώντας το εργαλείο TEMPO [10]. Όταν

ολοκληρώθηκε η προδιαγραφή του αλγορίθμου χρειάστηκε να γίνει αποσφαλμάτωση και κάποιες αλλαγές στον κώδικα Java που παράγει το εργαλείο, έτσι ώστε οι λειτουργίες Read και Write να γίνονται με πραγματικά αρχεία και όχι με ακέραιες τιμές, όπως είχαμε στην προδιαγραφή του αλγορίθμου. Στο τέλος, όσον αφορά την αξιολόγηση του αλγορίθμου, έγινε αρχικά σε συστοιχία υπολογιστών (cluster) στο τοπικό δίκτυο του Πανεπιστημίου Κύπρου και ακολούθως στο κατανεμημένο δίκτυο PlanetLab [6,13,14,24], για το οποίο χρειάστηκε να γίνει μελέτη των λειτουργιών που παρέχει.

1.3 Δομή Διπλωματικής Εργασίας

Στο Κεφάλαιο 2, περιγράφεται εκτενώς το γνωστικό υπόβαθρο το οποίο μας βοηθάει στην καλύτερη κατανόηση του μοντέλου αυτομάτων Εισόδου/Εξόδου, του μοντέλου χρονισμένων αυτομάτων Εισόδου/Εξόδου, του εργαλείο TEMPO [1,10], αλλά και της γλώσσας TIOA την οποία χρησιμοποιήσαμε για να γράψουμε τις προδιαγραφές του αλγορίθμου. Ακολούθως γίνεται αναφορά για τα κανάλια επικοινωνίας μεταξύ των διεργασιών, υπάρχει ένα παράδειγμα χρονισμένου αυτομάτου Εισόδου/Εξόδου και περιγραφή του κατανεμημένου δίκτυου PlanetLab. Τέλος, γίνεται μια μικρή αναφορά στον ορισμό της ατομικότητας, των συστημάτων απαρτίας και του αλγόριθμου ABD [5]. Στο Κεφάλαιο 3, γίνεται αναλυτική περιγραφή του αλγόριθμου LDR και στο Κεφάλαιο 4, η προδιαγραφή του αλγόριθμου στη γλώσσα TIOA [2,21] χρησιμοποιώντας το εργαλείο TEMPO. Ακολούθως, στο Κεφάλαιο 5 περιγράφεται η πειραματική αξιολόγηση του αλγορίθμου στο κατανεμημένο δίκτυο PlanetLab και πιο συγκεκριμένα τα σενάρια των πειραμάτων που εκτελέσαμε και τα αποτελέσματα που εξάγαμε. Τέλος, στο Κεφάλαιο 6 γίνεται αναφορά στα συμπεράσματα που εξάγονται από την προδιαγραφή και την πειραματική αξιολόγηση του αλγόριθμου LDR [15,16], τα προβλήματα που αντιμετωπίσαμε και πιθανή μελλοντική εργασία που πηγάζει από την παρούσα Διπλωματική Εργασία.

Κεφάλαιο 2

Γνωστικό Υπόβαθρο

2.1 Μοντέλο Αυτομάτων Εισόδου / Εξόδου

Το μοντέλο Αυτομάτων Εισόδου / Εξόδου (Input / Output Automata – IOA) [9,23], είναι κατάλληλο για τη μοντελοποίηση ασύγχρονων κατανεμημένων συστημάτων και αναπτύχθηκε από τους Nancy A. Lynch και Mark R. Tuttle, στο Massachusetts Institute of

Technology. Το μοντέλο αυτό, είναι κατάλληλο για συστήματα μοντελοποίησης στα οποία τα συστατικά λειτουργούν ασύγχρονα και χρησιμοποιείται για την ακριβή περιγραφή και επαλήθευση των συστατικών ενός κατανεμημένου συστήματος, όπως είναι οι διεργασίες και τα κανάλια επικοινωνίας. Το κάθε συστατικό του συστήματος μοντελοποείται ως ένα Input/Output αυτόματο και έχει τη δυνατότητα να αλληλεπιδρά με τα υπόλοιπα συστατικά του συστήματος. Είναι μια απλή μηχανή καταστάσεων στην οποία οι μεταβάσεις (transitions) συσχετίζονται με τις ενέργειες (actions), πιο συγκεκριμένα με την εκτέλεση μιας ενέργεια έχουμε μετάβαση από μια κατάσταση σε μια άλλη.

Μια θεμελιώδης ιδιότητα του μοντέλου αυτού είναι ότι υπάρχει σαφής διαχωρισμός ανάμεσα στις ενέργειες. Σε αυτές που η εκτέλεσή τους είναι υπό τον έλεγχο του αυτόματου και στις άλλες που είναι υπό τον έλεγχο του περιβάλλοντός του αυτόματου. Πιο συγκεκριμένα, ένα IOA αποτελείται από ένα σύνολο ενεργειών (actions) οι οποίες κατατάσσονται στις εξής τρεις υπό - κατηγορίες:

- Ενέργειες εισόδου (Input).
- Ενέργειες εξόδου (Output).
- Εσωτερικές ενέργειες (Internal).

Ένα αυτόματο δημιουργεί output και internal actions αυτόνομα και διαδίδει την έξοδο στιγμιαία στο περιβάλλον του, ενώ τα input actions δημιουργούνται από το περιβάλλον και διαδίδονται στιγμιαία στο αυτόματο. Ουσιαστικά ο διαχωρισμός αυτός βασίζεται στο ποιος καθορίζει τη χρονική στιγμή που θα εκτελεστεί μια ενέργεια. Το αυτόματο μπορεί να θέσει περιορισμούς για το πότε θα εκτελεστεί ένα output ή ένα internal action, αλλά δεν μπορεί να περιορίσει το πότε θα εκτελεστεί ένα input action.

Πιο συγκεκριμένα, 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 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 για αυτό το λόγο το αυτόματο δεν μπορεί να σταματήσει τη λειτουργία τους.

Οι ενέργειες εισόδου και εξόδου ονομάζονται εξωτερικές ενέργειες και χάρη σε αυτές το αυτόματο μπορεί να επικοινωνήσει με τα υπόλοιπα αυτόματα του συστήματος. Ενώ, οι εσωτερικές ενέργειες είναι ορατές μόνο από αυτόματο στο οποίο ανήκουν και εκτελούν τις εσωτερικές λειτουργίες του.

Ένα input/output αυτόματο A απαρτίζεται από πέντε συστατικά:

- Την ***Υπογραφή*** (Signature), που συμβολίζεται ως sig(A) και αποτελείται από το σύνολο όλων των ενεργειών (actions) του αυτομάτου A.
- Ένα ***σύνολο καταστάσεων*** (States), που συμβολίζεται ως states(A) και αποτελείται από όλες τις πιθανές καταστάσεις στις οποίες μπορεί να βρίσκεται το αυτόματο A.
- Ένα ***σύνολο αρχικών καταστάσεων*** (Start States), που συμβολίζεται ως start(A), είναι ένα μη κενό υποσύνολο start(A) ⊆ states(A) του συνόλου των καταστάσεων του A.
- Μια ***σχέση μεταβάσεων*** (transition relations), που συμβολίζεται ως steps(A) και περιλαμβάνει τις μεταβάσεις η οποίες έχουν τη μορφή (state, actions, states).
- Ένα ***σύνολο εργασιών*** (tasks), το οποίο περιλαμβάνει το σύνολο των ενεργειών που ελέγχονται τοπικά, δηλαδή τις ενέργειες εξόδου και τις εσωτερικές ενέργειες του αυτόματου A.

Τα μοντέλα IOA επιτρέπουν τη σύνθεση αυτομάτων, τα οποία μοντελοποιούν απλούστερα συστήματα με σκοπό τη δημιουργία πιο σύνθετων συστημάτων. Η σύνθεση αυτομάτων μας εγγυάται ότι εάν ένα αυτόματο έχει ένα output action π, τότε το π είναι ένα input action στα υπόλοιπα αυτόματα που έχουν το π σαν ενέργειά τους. Επομένως, εάν ένα αυτόματο εκτελέσει μια εξωτερική του ενέργεια, τότε στιγμιαία αυτό το θα μεταδοθεί ταυτόχρονα σε όλα τα υπόλοιπα αυτόματα που έχουν την ίδια ενέργεια σαν εσωτερική. 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 για τη σύνθεση δύο αυτομάτων Α και Β, υπάρχουν κάποιοι περιορισμοί οι οποίοι πρέπει να ισχύουν για να μπορεί να γίνει η σύνθεσή τους. Ορίζουμε ότι μια μετρήσιμη συλλογή υπογραφών $\{S_i\}_{i \in I}$ είναι *συμβατή* αν για όλα τα $i,j \in I$, $i \neq j$ ισχύουν τα ακόλουθα:

- $int(S_i) \cap acts(S_j) = \varnothing$

Εφόσον, οι εσωτερικές ενέργειες ενός αυτόματου 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.

- $out(S_i) \cap out(S_j) = \varnothing$

Εφόσον, μόνο ένα αυτόματο μπορεί να ελέγχει την εκτέλεση μίας ενέργειας, δεν επιτρέπεται η σύνθεση των Α και Β αν τα σύνολα των ενεργειών εξόδου τους δεν είναι ανεξάρτητα.

- *Καμία ενέργεια δεν περιέχεται σε άπειρα πολλά σύνολα* $acts(S_i)$.

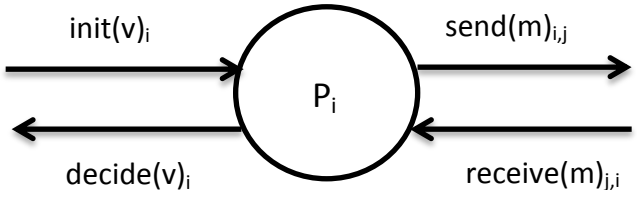
Κάθε ενέργεια του αυτομάτου σύνθεσης, πρέπει να είναι ενέργεια μόνο πεπερασμένου αριθμού του συνόλου των επιμέρους αυτομάτων.

Λέμε ότι μια συλλογή από αυτόματα είναι συμβατή αν οι υπογραφές τους είναι συμβατές.

Παράδειγμα αυτομάτου Εισόδου / Εξόδου:

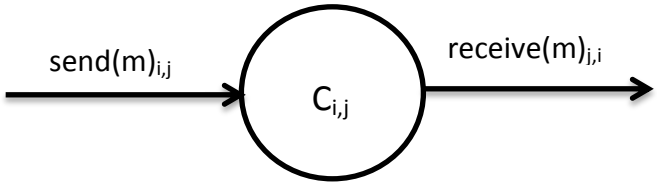
Ένα παράδειγμα ενός πολύ απλού αυτομάτου είναι μια διεργασία σε ένα ασύγχρονο περιβάλλον, όπως φαίνεται στο Σχήμα 2.1 [23,24]. Το αυτόματο P_i, όπου το i είναι ο δείκτης που αντιπροσωπεύει τη διεργασία αυτή, σχεδιάζεται ως ένας κύκλος, με εισερχόμενα και εξερχόμενα βέλη. Τα εισερχόμενα βέλη αντιπροσωπεύουν τις ενέργειες εισόδου, ενώ τα εξερχόμενα τις ενέργειες εξόδου. Οι εσωτερικές ενέργειες του αυτομάτου δεν εμφανίζονται στο σχήμα. Στο παράδειγμά μας το αυτόματο αποτελείται από μια ενέργεια εισόδου $init(v)_i$, η οποία αναπαριστά τη λήψη μιας τιμής εισόδου v. Το αυτόματο αυτό επικοινωνεί με άλλες διεργασίες, κάνοντας χρήση της ενέργειας εξόδου $send(m)_{i,j}$,

που ουσιαστικά εκτελεί αποστολή μηνυμάτων από τη διεργασία P_i στην P_j και με τη χρήση της ενέργειας εισόδου receive$(m)_{j,i}$, με την οποία γίνεται παραλαβή μηνυμάτων από την διεργασία P_j στην P_i . Τέλος, το αυτόματο μέσα από την επικοινωνία με τα άλλα αυτόματα παίρνει μια απόφαση και στέλνει εξόδους της μορφής decide$(v)_i$, που αναπαριστούν μια απόφαση v.



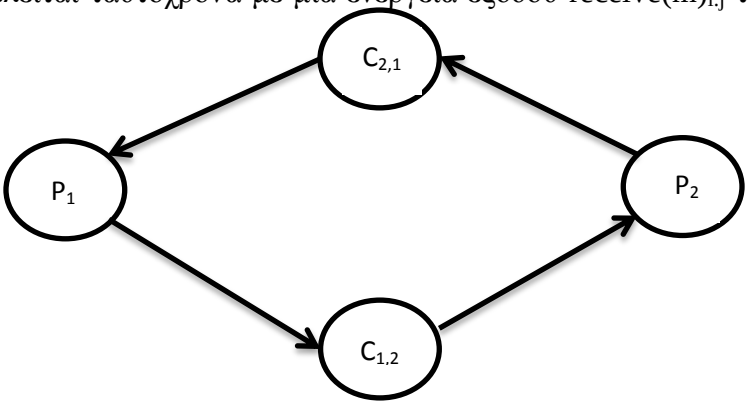
Σχήμα 2.1: Αυτόματο Διεργασίας.

Ένα άλλο παράδειγμα αυτομάτου μπορεί να θεωρηθεί ένα FIFO κανάλι επικοινωνίας, όπως φαίνεται στο Σχήμα 2.2. Ένα κανάλι επικοινωνίας συμβολίζεται με $C_{i,j}$ όπου i και j συμβολίζουν τους δείκτες των δύο διεργασιών που συνδέει το κανάλι αυτό και αποτελείται από δυο ενέργειες μια ενέργεια εισόδου send$(m)_{i,j}$ και μια ενέργεια εξόδου receive$(m)_{j,i}$.



Σχήμα 2.2: Αυτόματο Καναλιού.

Η σύνθεση των δυο αυτομάτων Εισόδου / Εξόδου και δυο 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$(m)_{i,j}$ του αυτομάτου της διεργασίας P_i ταυτίζεται και εκτελείται ταυτόχρονα με μια ενέργεια εισόδου send$(m)_{i,j}$ του αυτομάτου καναλιού $C_{i,j}$ και κάθε ενέργεια εισόδου receive$(m)_{i,j}$ του αυτομάτου της διεργασίας P_i ταυτίζεται και εκτελείται ταυτόχρονα με μια ενέργεια εξόδου receive$(m)_{i.j}$ 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 (TIOA) [2,4,12], αποτελεί εξέλιξη του Μοντέλου Αυτομάτων Εισόδου/Εξόδου που περιγράψαμε στην προηγούμενη ενότητα. Είναι μια μη ντετερμινιστική μηχανή καταστάσεων, της οποίας η λειτουργία βασίζεται σε προθεσμίες βάση χρόνου. Η απόδοση και η ορθότητα των αυτομάτων αυτών δεν βασίζεται μόνο στη σειρά των γεγονότων, αλλά στη χρονική στιγμή που συμβαίνει το κάθε γεγονός. Το μοντέλο αυτό δεν περιορίζεται στα διακριτά βήματα, αλλά μπορεί να υπάρχει και συνεχής εξέλιξη με την πάροδο του χρόνου. Η εξέλιξη αυτή υλοποιείται με τη χρήση των τροχιών (trajectories). Επομένως, η κατάσταση ενός TIOA μπορεί να αλλάξει στιγμιαία, είτε λόγω της ύπαρξης μιας διακριτής μετάβασης, είτε λόγω μιας τροχιάς, η οποία είναι μια συνάρτηση που περιγράφει την εξέλιξη μιας μεταβλητής κατάστασης ανάμεσα σε διακριτές μεταβάσεις.

Ένα timed Input/Output αυτόματο A είναι μια πλειάδα (X, Q, Θ, E, H, D, T). Πιο συγκεκριμένα αποτελείται από τα εξής επτά συστατικά:

- Ένα ***σύνολο εσωτερικών μεταβλητών*** X.
- Ένα ***σύνολο καταστάσεων*** Q, όπου $Q \subseteq val(X)$.
- Ένα μη κενό ***σύνολο αρχικών καταστάσεων*** Θ, όπου $\Theta \subseteq Q$.
- Ένα ***σύνολο εξωτερικών λειτουργιών*** E, το οποίο ουσιαστικά είναι η ένωση των ενεργειών εισόδου και εξόδου.
- Ένα ***σύνολο εσωτερικών λειτουργιών*** H, όπου $E \cap H = ø$.
- Ένα ***σύνολο διακριτών καταστάσεων*** D, όπου $D \subseteq Q \times A \times Q$ και περιλαμβάνει τις μεταβάσεις του τύπου (state, action, state).
- Ένα ***σύνολο τροχιών*** (trajectories) T, όπου $T \subseteq trajs(Q)$.

Όπως και στα αυτόματα Εισόδου/Εξόδου, έτσι και σε αυτά τα αυτόματα υπάρχει η έννοια της σύνθεσης. Πολύπλοκα συστήματα μπορούν να αναλυθούν σε υποσυστήματα, η σύνθεση των οποίων θα μας δώσει ένα ενιαίο πολύπλοκο σύστημα. Όλα τα TIOAs που λαμβάνουν μέρος στη σύνθεση, συμμετέχουν στο σύνολο των trajectories ταυτόχρονα με αποτέλεσμα να περνά ιδεατά το ίδιο χρονικό διάστημα. Επίσης, κάθε ενέργεια εξόδου του

κάθε αυτόματου ταυτίζεται με μια ενέργεια εισόδου κάθε άλλου αυτομάτου, όπου οι δύο αυτές ενέργειες έχουν το ίδιο όνομα.

2.3 Μοντέλα Επικοινωνίας στα Κατανεμημένα Συστήματα

Τα Κατανεμημένα Συστήματα παρέχουν δυο μοντέλα επικοινωνίας [22]. Η επικοινωνία δυο διεργασιών μπορεί να γίνει είτε μέσω κοινόχρηστης μνήμης, είτε με ανταλλαγή μηνυμάτων. Το μοντέλο επικοινωνίας αποτελεί μια από τις παραμέτρους που ο δημιουργός κάποιου αλγόριθμου πρέπει να γνωρίζει πριν αρχίσει την ανάπτυξή του.

2.3.1 Κατανεμημένη Κοινόχρηστη Μνήμη (ΚΚΜ)

Οι επεξεργαστές στο μοντέλο αυτό επικοινωνούν μέσω μιας κοινής μνήμης, η οποία περιλαμβάνει ένα σύνολο από κοινές μεταβλητές. Μας παρέχει δυο τύπους μεταβλητών, ανάγνωσης και γραφής (*read/write*) και ανάγνωσης-μετατροπής-γραφής (*read-modify-write*). Κάποιοι ειδικοί τύποι των μεταβλητών ανάγνωσης-μετατροπής-γραφής είναι οι τύποι μεταβλητών ελέγχου και αλλαγής (*test&set*) και σύγκρισης και αντιμετάθεσης (*compare&swap*). Τα μοντέλα ταυτοχρονισμού που παρέχει η ΚΚΜ είναι:

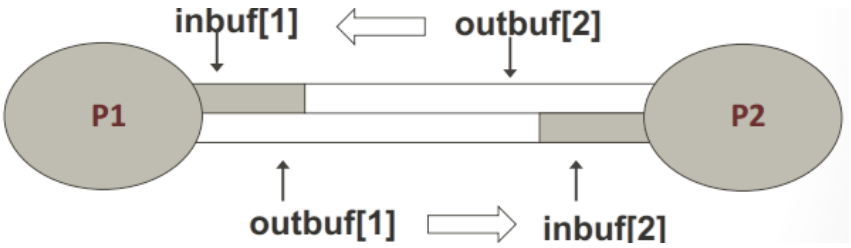
- *Μοντέλο ενός εγγραφέα και ενός αναγνώστη (SWSR):* μια εγγραφή μπορεί να είναι ταυτόχρονη με μια ανάγνωση πάνω σε μια μεταβλητή Χ της ΚΚΜ.
- *Μοντέλο ενός εγγραφέα και πολλών αναγνωστών (SWMR):* μια εγγραφή ταυτόχρονη με πολλές ταυτόχρονες αναγνώσεις πάνω σε μια μεταβλητή Χ της ΚΚΜ.
- *Μοντέλο πολλών εγγραφών και πολλών αναγνωστών (MWMR):* πολλές ταυτόχρονες εγγραφές ταυτόχρονες με πολλές ταυτόχρονες αναγνώσεις πάνω σε μια μεταβλητή Χ της ΚΚΜ.

2.3.2 Ανταλλαγή Μηνυμάτων

Οι επεξεργαστές σε αυτό το μοντέλο επικοινωνούν με μηνύματα που στέλνονται στο δίκτυο, στα οποία περιέχονται κάποιες πληροφορίες. Ο κάθε επεξεργαστής διαθέτει:

- μια λίστα εισερχόμενων μηνυμάτων.
- μια λίστα εξερχόμενων μηνυμάτων.

Πιο συγκεκριμένα, όπως βλέπουμε και στο Σχήμα 2.4, τα μηνύματα που θέλει να αποστείλει η διεργασία P_2 στη διεργασία P_1, μεταφέρονται από τη λίστα των εξερχόμενων μηνυμάτων, outbuf[2], της διεργασίας αυτής, στην λίστα των εισερχόμενων μηνυμάτων, inbuf[1], της διεργασίας P_1 έτσι ώστε να τα παραλάβει η διεργασία 1. Παρομοίως, γίνεται και κατά τη διαδικασία της αποστολής μηνυμάτων από την διεργασία P_1 στη P_2.



Σχήμα 2.4: Επικοινωνία μέσω ανταλλαγής μηνυμάτων [21].

Εμείς στην προδιαγραφή μας, χρησιμοποιήσαμε ως μοντέλο επικοινωνίας την ανταλλαγή μηνυμάτων.

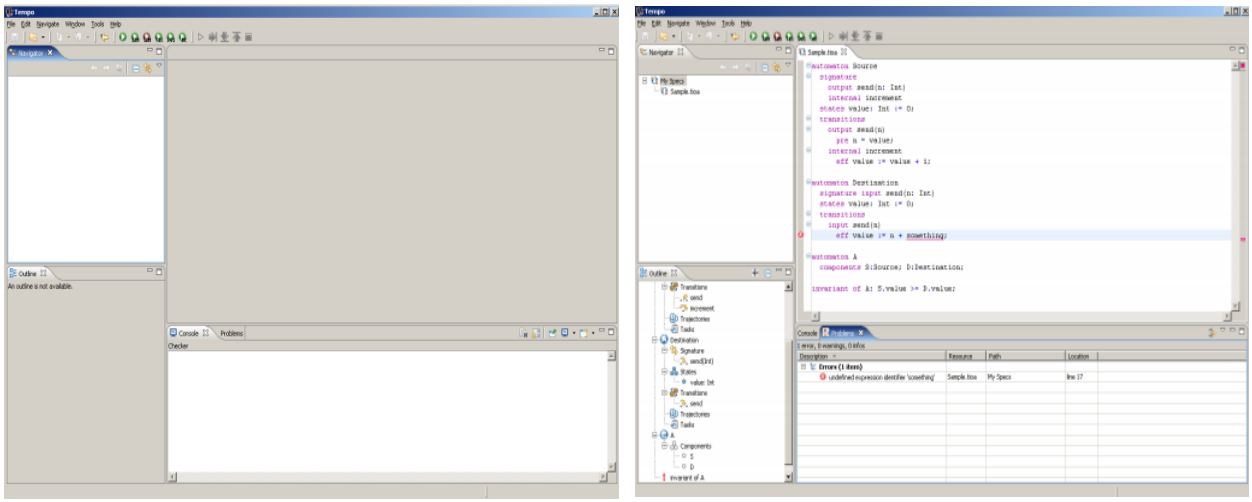
2.4 Εργαλείο TEMPO και Γλώσσα TIOA

Στην ενότητα αυτή περιγράφεται το εργαλείο TEMPO [10,17] το οποίο χρησιμοποιήθηκε για την προδιαγραφή του αλγόριθμου LDR, που μελετήθηκε στην παρούσα Διπλωματική Εργασία. Στη συνέχεια θα γίνει λεπτομερής περιγραφή της Γλώσσας TIOA που χρησιμοποιεί το εργαλείο TEMPO για την υλοποίηση Χρονισμένων Μοντέλων Εισόδου/Εξόδου.

2.4.1 Εργαλείο TEMPO

Το εργαλείο Tempo αναπτύχθηκε από την VeroModo Inc. [19,20] και διανέμεται δωρεάν. Χρησιμοποιείται για την εγγραφή προδιαγραφών και παρέχει εργαλεία για την ανάλυση τους. Πιο συγκεκριμένα, περιλαμβάνει εργαλεία για την ανάλυση, την προσομοίωση, το 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.

Μπορούμε να προμηθευτούμε το εργαλείο Tempo από την επίσημη σελίδα της VeroModo Inc, στην οποία υπάρχουν πληροφορίες για τη σωστή εγκατάσταση. Επίσης, υπάρχουν κατάλληλοι οδηγοί που επεξηγούν πλήρως απαιτήσεις, ανάγκες, περιορισμούς και τυχόν προβλήματα του εργαλείου. Το εργαλείο Tempo παρέχει ένα άνετο και εύκολο στη χρήση περιβάλλον προς το χρήστη. Η διεπαφή χρήστη φαίνεται στο Σχήμα 2.5. Την πρώτη φορά που θα χρησιμοποιήσει κάποιος τη διεπαφή θα του εμφανιστεί το άδειο παράθυρο στα αριστερά. Κάποιες φορές ίσως να διαφέρει από πλατφόρμα σε πλατφόρμα, αλλά η βασική διεπαφή παραμένει η ίδια. Αργότερα, με την αλληλεπίδραση του χρήστη με το εργαλείο θα δούμε ότι η διεπαφή θα έχει παρόμοια μορφή με το παράθυρο στα δεξιά.



Σχήμα 2.5: Διεπαφή χρήστη στο Εργαλείο TEMPO.

Η διεπαφή αποτελείται από τα ακόλουθα τμήματα:

- **Menu Bar:** αποτελείται από έξι pull-down μενού. Το *File*, *Edit* και *Help* τα οποία παρέχουν κάποιες κύριες λειτουργίες για την επεξεργασία των κειμένων για παράδειγμα δημιουργία, αποθήκευση, αποκοπή και αντιγραφή κειμένου και για βοήθεια. Ακολούθως έχουμε το *Navigate*, το οποίο επιτρέπει στο χρήστη την εναλλαγή του σε ένα συγκεκριμένο σημείο ενός αρχείου. Το *Windows* που επιτρέπει τον έλεγχο της εμφάνισης της διεπαφής του χρήστη και επιτρέπει την πρόσβαση στα preferences του εργαλείου. Και τέλος, το *Tools* που επιτρέπει την πρόσβαση στα εργαλεία που παρέχει το TEMPO.

- **Toolbar:** βρίσκεται ακριβώς κάτω από το Menu Bar και επιτρέπει στους χρήστες να εφαρμόσουν κάποιες λειτουργίες γρήγορα, πατώντας απλά ένα εικονίδιο.
- **Status Bar:** βρίσκεται στο κάτω μέρος της διεπαφής και παρέχει πληροφορίες για την κατάσταση, όπως για παράδειγμα σε ποια γραμμή βρίσκεται ο δείκτης μέσα στο αρχείο μια συγκεκριμένη στιγμή.
- **Editor:** στο τμήμα αυτό μπορούμε να επεξεργαστούμε την προδιαγραφή ενός συγκεκριμένου αρχείου.
- **Navigator:** παρέχει μια ιεραρχική δομή όλων των εργασιών (projects) που βρίσκονται στο χώρο εργασίας.
- **Outline:** παρουσιάζει διαγραμματικά το περιεχόμενο ενός .tioa αρχείου που είναι ανοικτό στον editor. Διακρίνονται οι ενέργειες, οι καταστάσεις, οι τροχιές και γενικά όλα τα τμήματα που περιγράφονται στο αυτόματο του συγκεκριμένου αρχείου.
- **Problems:** εδώ εμφανίζονται όλα τα errors που ανιχνεύτηκαν από τον ελεγκτή του εργαλείου.
- **Console:** εμφανίζει την εξόδου μιας προδιαγραφής κατά την εκτέλεση κάποιου από τα εργαλεία που παρέχει.

Τα εργαλεία που παρέχει το εργαλείο TEMPO είναι:

- Ο **ελεγκτής (checker)**, ο οποίος ελέγχει τη σύνταξη και εφαρμόζει στατική σημασιολογική ανάλυση στην προδιαγραφή του αυτομάτου.
- Ο **προσομοιωτής (simulator)**, ο οποίος είναι υπεύθυνος για την προσομοίωση της εκτέλεσης του συστήματος σύμφωνα με το πρόγραμμα (scheduler).
- Ο **μεταφραστής JAVA (tempo2java)**, ο οποίος μεταγλωττίζει την προδιαγραφή σε γλώσσα προγραμματισμού JAVA.
- Ο **μεταφραστής LaTex (tempo2tex)**, ο οποίος μεταγλωττίζει την προδιαγραφή σε γλώσσα LaTex.
- Το **εργαλείο για αποδείξεις (PVS interactive theorem proover)**, το οποίο χρησιμοποιείται για τον έλεγχο της ορθότητας των μοντέλων που γράφονται σε γλώσσα TIOA.

- Ο **μεταφραστής Uppaal**, ο οποίος μεταγλωττίζει το μοντέλο σε Uppaal γλώσσα.

Για την πραγματοποίηση της παρούσας Διπλωματικής Εργασίας, χρησιμοποιήθηκαν ο ελεγκτής (checker) για να ελέγξουμε ότι οι προδιαγραφές μας ήταν ορθές συντακτικά και σημασιολογικά και ο μεταφραστής tempo2java για να αξιολογήσουμε τον αλγόριθμό μας.

2.4.2 Γλώσσα TIOA

Η γλώσσα που χρησιμοποιήθηκε για την προδιαγραφή του αλγόριθμου που αφορά την παρούσα Διπλωματική εργασία είναι η γλώσσα TIOA [3,17]. Έγινε χρήση της γλώσσας TIOA, διότι επιθυμούμε να έχουμε τη δυνατότητα χρήσης time-outs στην προδιαγραφή του αλγόριθμου. Η γλώσσα αυτή χρησιμοποιείται ευρέως για την προδιαγραφή Χρονισμένων Αυτομάτων Εισόδου/Εξόδου και υποστηρίζεται από το εργαλείο TEMPO. Η γλώσσα αυτή είναι πολύ όμοια με κάποιες διαφορές με τη γλώσσα IOA, που χρησιμοποιείται για την προδιαγραφή ασύγχρονων κατανεμημένων αλγόριθμων. Ακολούθως, θα περιγράψουμε όλα τα δομικά στοιχεία που είναι απαραίτητα για την προδιαγραφή ενός σύγχρονου κατανεμημένου αλγόριθμου.

Κάθε μεταβλητή έχει ένα στατικό τμήμα το οποίο καθορίζει τη τιμή που έχει μια μεταβλητή και έχει και ένα δυναμικό τμήμα το οποίο είναι η συνάρτηση που περιγράφει την εξέλιξη της μεταβλητής με το πέρασμα του χρόνου. Η γλώσσα TIOA παρέχει τους πιο κάτω τύπους δεδομένων:

Τύποι Δεδομένων:

Booleans (Bool): είναι ο αρχέγονος τύπος δεδομένων, ο οποίος μπορεί να πάρει μόνο δυο τιμές, 1(true) και 0(false), οι οποίες ονομάζονται Boolean ή λογικές τιμές. Η γλώσσα TIOA υποστηρίζει τις ακόλουθες πράξεις σε μια Boolean έκφραση: άρνηση (¬), λογικό AND (/\), λογικό OR (\/), συνεπαγωγή (=>), λογική ισοδυναμία (<=>), ισότητα (=) και ανισότητα (~=). Επίσης μπορούμε να χρησιμοποιήσουμε τον καθολικό ποσοδείκτη (∀), αλλά και τον υπαρξιακό ποσοδείκτη (∃).

Natural Numbers (Nat): αυτός ο τύπος δηλώνει στοιχεία τα οποία είναι μη αρνητικοί ακέραιοι αριθμοί (0, 1, 2, ...). Υποστηρίζει τις συναρτήσεις succ(x), η οποία επιστρέφει τον

αμέσως επόμενο φυσικό αριθμό, pred(x), η οποία επιστρέφει τον προηγούμενο φυσικό αριθμό, min(x,y), η οποία επιστρέφει το μικρότερο από τους δύο αριθμούς x και y, max(x,y), η οποία επιστρέφει το μεγαλύτερο από τους δύο αριθμούς x και y, div(x,y), mod(x,y), καθώς και τις πράξεις πρόσθεση (+), αφαίρεση(-), πολλαπλασιασμός (*), εκθετική δύναμη (**). Τέλος υποστηρίζει τους συγκριτικούς τελεστές μεγαλύτερο (>), μικρότερο (<), μεγαλύτερο και ίσο (≥), μικρότερο και ίσο (≤), ίσο (=) και άνισο (~=).

Integers (Int): αυτός ο τύπος δηλώνει στοιχεία ακέραιους αριθμούς. Υποστηρίζει όλες τις συναρτήσεις και τις πράξεις των φυσικών αριθμών, καθώς επίσης το αρνητικό πρόσημο (-) και την απόλυτη τιμή abs(x).

Real Numbers (Real): αυτός ο τύπος δηλώνει πραγματικούς αριθμούς. Υποστηρίζει τις συναρτήσεις min(x,y), η οποία επιστρέφει το μικρότερο από τους δυο αριθμούς, max(x,y), οποία επιστρέφει το μεγαλύτερο από τους δυο αριθμούς, abs(x), η οποία επιστρέφει την απόλυτη τιμή του x και floor(x), η οποία επιστρέφει το μεγαλύτερο ακέραιο αριθμό ο οποίος είναι ίσος ή μικρότερος από το x. Καθώς επίσης και όλες τις πράξεις και τους συγκριτικούς τελεστές που υποστηρίζουν οι ακέραιοι αριθμοί.

Discrete Real Numbers (DiscreteReal): είναι ένας πραγματικός αριθμός με τη διαφορά ότι δεν αλλάξει μεταξύ διακριτών βημάτων. Είναι ιδεατός τύπος και χρησιμοποιείται για να ορίζουμε το φυσικό ρολόι ενός αυτόματου. Η μόνη διαφορά μεταξύ του DiscreteReal και του Real, είναι ότι ο πρώτος τύπος δεδομένων είναι μεταβλητός, ενώ ο δεύτερος στατικός.

Augmented Real Numbers (AugmentedReal): είναι ένας πραγματικός αριθμός ο οποίος μπορεί να πάρει δυο επιπλέον τιμές, τις τιμές -∞ και +∞. Σύμφωνα με την ιεραρχία του TEMPO οι Real αριθμοί είναι υποσύνολο των AugmentedReal αριθμών.

Characters (Char): αυτός ο τύπος δεδομένων δηλώνει ένα χαρακτήρα ο οποίος μπορεί να πάρει ως τιμές γράμματα και ψηφία. Ένας χαρακτήρας περιέχεται μέσα από μονούς αποστρόφους ('a', 'A', '0'). Υποστηρίζει τους συγκριτικούς τελεστές μεγαλύτερο (>), μικρότερο (<), μεγαλύτερο και ίσο (≥), μικρότερο και ίσο (≤) και συγκρίνει δυο χαρακτήρες ανάλογα με αλφαβητική τους σειρά.

Strings (String): αυτός ο τύπος δεδομένων δηλώνει μια ακολουθία από χαρακτήρες. Μπορεί να αναπαρασταθεί επίσης και ως Seq[Char]. Ένα String μπορεί να αναπαρασταθεί από ένα σύνολο χαρακτήρων μέσα σε διπλές αποστρόφους ("hello").

Arrays (Array[T_1, …, T_n, E]): αυτός ο τύπος δεδομένων δηλώνει ένα πίνακα. Για κάθε n>0, η δήλωση Array[T_1, …, T_n, E] δηλώνει ένα n-διάστατο πίνακα με στοιχεία του τύπου E. Για παράδειγμα ένας δυσδιάστατος πίνακας ακεραίων θα πρέπει να δηλωθεί ως Array[Int, Int, Int] Υποστηρίζει τη συνάρτηση constant(e), η οποία αρχικοποιεί όλα τα στοιχεία του πίνακα με e.

Set (Set[E]): αυτός ο τύπος δεδομένων δηλώνει ένα πεπερασμένο σύνολο στοιχείων τύπου E. Υποστηρίζει τις ακόλουθες συναρτήσεις insert(e,s), η οποία προσθέτει το στοιχείο e στο σύνολο στοιχείων s, delete(e,s), η οποία διαγράφει το στοιχείο e από το σύνολο στοιχείων s και size(s), η οποία επιστρέφει το συνολικό πλήθος στοιχείων που περιέχει το σύνολο s. Καθώς επίσης και τις ακόλουθες πράξεις συνόλων: ένωση (\U), τομή (\I), συμπλήρωμα (-), ανήκει (\in), δεν ανήκει (\notin), υποσύνολο (\subset), πλήρες υποσύνολο (\subseteq), υπερσύνολο (\supset) και πλήρες υπερσύνολο (\supseteq).

Sequences (Seq[E]): αυτός ο τύπος δεδομένων δηλώνει μια πεπερασμένη ακολουθία στοιχείων τύπου E. Υποστηρίζει τις ακόλουθες συναρτήσεις, (όπου το n έχει τύπο Nat), head(s), η οποία μας επιστρέφει το πρώτο στοιχείο της ακολουθίας, last(s), η οποία μας επιστρέφει το τελευταίο στοιχείο της ακολουθίας, len(s), η οποία επιστρέφει το μέγεθος της ακολουθίας, get(s,n), η οποία επιστρέφει το n-οστό στοιχείο της ακολουθίας s, set(s,n,e), η οποία ενημερώνει το n-οστό στοιχείο της ακολουθίας s με το στοιχείο e, inject(s,n,e), η οποία προσθέτει το στοιχείο e στην n-οστή θέση της ακολουθίας s και eject(s,n), η οποία διαγράφει το n-οστό στοιχείο της ακολουθίας s. Επίσης υποστηρίζει τις πράξεις κενή ακολουθία ({}), ανήκει (\in), δεν ανήκει (\notin), εισαγωγή στοιχείου στο τέλος της ακολουθίας (|-), εισαγωγή στοιχείου στο τέλος της ακολουθίας (-|) και συνένωση δυο ακολουθιών (||).

Enumerations (Enumeration[e_1,…,e_n]): αυτός ο τύπος δεδομένων δηλώνει ένα πεπερασμένο σύνολο που αποτελείται από τα στοιχεία e_1,…,e_n. Ορίζεται με το εξής τρόπο **types** *Name*:***Enumeration***[e_1,…, e_n], όπου Name είναι το όνομα το enumeration και

$e_1,\ldots,e_n$ οι τιμές που μπορεί να πάρει αυτή η μεταβλητή. Υποστηρίζει τη συνάρτηση succ(e), η οποία μας επιστρέφει την τιμή που ακολουθεί την τιμή e στον ορισμό του enumeration.

Tuples (Tuple[f_1:T_1, … , f_n:T_n]): αυτός ο τύπος δεδομένων μας επιτρέπει να δηλώσουμε μια πλειάδα, δηλαδή έχουμε τη δυνατότητα να ορίσουμε ένα πιο σύνθετο τύπο δεδομένων. Μια τέτοια μεταβλητή θα έχει $f_1,\ldots,f_n$ πεδία, που το κάθε ένα θα είναι μεταβλητή τύπου $T_1,\ldots,T_n$ αντίστοιχα. Για παράδειγμα εάν έχουμε μια μεταβλητή person του τύπου Tuple[name:String, age:Int], τότε μπορούμε να έχουμε πρόσβαση στο πεδίο name της μεταβλητής person με τον τελεστή **'.'**, για παράδειγμα person.name = "John".

Unions (Union[f_1:T_1, … , f_n:T_n]): αυτός ο τύπος δεδομένων μας επιτρέπει να δηλώσουμε μια μεταβλητή της οποίας ο τύπος της μπορεί να είναι ένας από τους τύπους $T_1,\ldots,T_n$. Για παράδειγμα αν ορίσουμε OptionValue : Union[s:String, i:Int, n:Nat] και v : OptionValue, τότε η μεταβλητή v μπορεί να έχει οποιοδήποτε από τους τρεις τύπους String, Int και Nat. Μπορούμε να αρχικοποιήσουμε τη μεταβλητή v χρησιμοποιώντας τον constructor s με τον εξής τρόπο v := s("hello").

Τελεστές:

Where: ο τελεστής where μας δίνει τη δυνατότητα να περιορίσουμε το πεδίο τιμών που μπορεί να πάρει μια μεταβλητή. Ορίζεται με τον εξής τρόπο **where** *predicate*, για παράδειγμα **output** receive (m:Msgs,j:Int) **where** j>2. Δηλαδή εδώ το j επιτρέπεται να είναι οποιαδήποτε ακέραια τιμή μεγαλύτερη του 2.

Choose: ο τελεστής choose μας δίνει τη δυνατότητα να διαλέξουμε μη-ντετερμινιστικά την τιμή μιας μεταβλητής. Ο τελεστής αυτός μπορεί να συνδυαστεί με τον τελεστή where. Ορίζεται με τον εξής τρόπο **choose** *variable* **where** *predicate*, το οποίο θα μας επιστρέψει μια τιμή έτσι ώστε να ικανοποιείται το predicate. Για παράδειγμα num : **Int** := **choose** n **where** $1\leq n$ /\ $n\leq 3$, επομένως η μεταβλητή num θα αρχικοποιηθεί μη-ντετερμινιστικά με μια από τις τιμές 1 ή 2 ή 3. Εάν δηλώσουμε num : **Int** := **choose** n, τότε τυχαία θα μας επιστρέψει οποιαδήποτε τιμή από το πεδίο τιμών που έχει ο τύπος δεδομένων Int.

Λεξιλόγιο

Η γλώσσα TIOA, δίνει στο χρήστης τη δυνατότητα να ορίσει δικό του λεξιλόγιο το οποίο μπορούμε να ορίσουμε δικούς μας αφαιρετικούς τύπους δεδομένων και τελεστές. Ο ορισμός ενός λεξιλογίου αρχίζει με τη λέξη κλειδί **vocabulary** και ένα αναγνωριστικό το οποίο αντιπροσωπεύει το όνομα του λεξιλογίου. Εάν θα συμπεριληφθούν άλλα λεξιλόγια πρέπει να χρησιμοποιήσουμε τη λέξη κλειδί **imports** ακολουθημένο από το όνομα του λεξιλογίου που θέλουμε να προσθέσουμε. Κάθε vocabulary, μπορεί να περιέχει τον ορισμό ενός ή περισσότερων τύπων δεδομένων, μετά από τις λέξεις **types** και **defines** και τον ορισμό μηδέν ή περισσότερων τελεστών, μετά από τη λέξη **operators**. Κάθε τελεστής έχει μια υπογραφή η οποία καθορίζει τα ονόματα των παραμέτρων του, ακολουθεί το σύμβολο ':', ο τύπος των παραμέτρων του, ακολουθεί το σύμβολο → (->) και τον τύπο του αποτελέσματος. Ο ορισμός ενός vocabulary ολοκληρώνεται με τη λέξη **end.** Στον Πίνακα 2.1 συνοψίζονται οι τρόποι με τους οποίους μπορούμε να ορίσουμε τελεστές, καθώς και κάποια παραδείγματα χρήσης τους.

Παράδειγμα Ορισμού Τελεστή	**Τύπος Τελεστή**	**Παράδειγμα Χρήσης**
f : Int -> Int	functional	f(i)
min : Int, Int -> Int	functional	min(i,j)
_ _ < _ _ : Int, Int -> Bool	infix	i < j
-_ _ : Int -> Int	prefix	-i
_ _! : Int -> Int	postfix	i!
_ _[_ _] : A, Int -> Int	mixfix	a[i]
{_ _} : E -> Seq[E]	mixfix	{x}
if _ _ then _ _ else _ _ : Bool, S, S -> S	mixfix	if x<0 then –x else x

Πίνακας 2.1: Ορισμός Τελεστών σε Λεξιλόγιο.

Τώρα είμαστε σε θέση να ορίσουμε ορθά ένα απλό παράδειγμα λεξιλογίου το οποίο ορίζει τελεστές για ταξινόμηση.

vocabulary Ordering (T : type)

types T

operators

_ _<_ _ , _ _≤_ _ , _ _>_ _ , _ _≥_ _ , _ _=_ _ , _ _~=_ _ : T, T -> Bool

end

Συναρτήσεις (functions): Η γλώσσα TIOA επιτρέπει στο χρήστη να ορίσει τις δικές του συναρτήσεις και να τις χρησιμοποιήσει στις προδιαγραφές του. Ο ορισμός μιας συνάρτησης ξεκινά με τη λέξη **let**, ακολουθεί το όνομα της συνάρτησης, μια αριστερή παρένθεση και μηδέν ή περισσότερα ονόματα παραμέτρων. Ακολουθεί μια δεξιά παρένθεση μαζί με άνω και κάτω τελεία και ο τύπος υπογραφής της συνάρτησης. Τέλος, ακολουθεί το σύμβολο = και ο ορισμός της συνάρτησης. Για παράδειγμα,

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

Εντολή ανάθεσης (:=): για να γίνει ανάθεση μιας τιμής σε μια μεταβλητή χρησιμοποιείται το σύμβολο **':= '.** Στα αριστερά του συμβόλου αυτού γράφεται το όνομα της μεταβλητής, ενώ στα δεξιά η τιμή που θέλουμε να έχει η μεταβλητή μας μετά την εκτέλεση της ανάθεσης.

Εντολή τυπώματος (print): για να τυπώσουμε την τιμή μιας μεταβλητής χρησιμοποιούμε τη λέξη κλειδί **print** και το όνομα της μεταβλητής που επιθυμούμε να τυπωθεί.

Εντολή if-then-else: μια τέτοια εντολή ξεκινάει με τη λέξη κλειδί **if**, μετά ακολουθεί μια λογική έκφραση που μπορεί να αποτιμηθεί σε true ή false, ακολουθεί η λέξη κλειδί **then** και μετά ένα σύνολο από εντολές οι οποίες θα εκτελεστούν εάν η λογική έκφραση του if είναι αληθής. Στη συνέχεια μπορούν να προστεθούν μηδέν ή περισσότερα **elseif** ή **else**. Εάν ακολουθεί elseif, τότε πρέπει μετά να υπάρχει μια λογική έκφραση, η λέξη κλειδί then και ένα σύνολο εντολών οι οποίες θα εκτελεστούν εάν η λογική έκφραση αποτιμηθεί με true. Διαφορετικά εάν ακολουθεί else, μετά πρέπει να υπάρχει ένα σύνολο εντολών το οποίο θα εκτελεστεί εάν η λογική έκφραση κάποιο if ή elseif αποτιμηθεί με false. Σε κάθε περίπτωση στο τέλος προσθέτουμε τη λέξη κλειδί **fi**. Οι εντολές αυτές μπορεί να είναι

φωλιασμένες. Ακολουθεί παράδειγμα που βρίσκει το μεγαλύτερο αριθμό ανάμεσα σε a και b, ενώ αν οι δυο αριθμοί ισούνται κάνει την Boolean μεταβλητή ίση με true.

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

Εντολή while: μια τέτοια εντολή αρχίζει με τη λέξη κλειδί **while,** ακολουθεί μια λογική έκφραση η οποία αποτιμάται σε true ή false, ένα σύνολο εντολών οι οποίες θα εκτελεστούν εάν η λογική έκφραση αποτιμηθεί με true και τέλος ακολουθεί η λέξη κλειδί **od**. Οι εντολές αυτές μπορεί να είναι φωλιασμένες. Ακολουθεί παράδειγμα το οποίο τυπώνει τους αριθμούς από 0 μέχρι 5

```
x := 0;
while (x <=5)
        print x;
        x := x + 1;
od
```

Εντολή for: μια τέτοια εντολή αρχίζει με τη λέξη κλειδί **for,** το όνομα της μεταβλητής που θα είναι ο μετρητής, ο χαρακτήρας **':',** ο τύπος του μετρητή, ακολουθεί η λέξη κλειδί **where,** μια λογική έκφραση που έχει να κάνει με τον αριθμό των επαναλήψεων του for statement και η λέξη κλειδί **do.** Στη συνέχεια υπάρχει ένα σύνολο εντολών οι οποίες θα εκτελεστούν μέχρις ότου η λογική έκφραση να αποτιμάται με true και τέλος έχουμε τη λέξη κλειδί **od**. Ακολουθεί παράδειγμα που τυπώνει το μετρητή k.

for k:Nat where k<5 do

print k;

od

Εντολή Fire: η εντολή αυτή χρησιμοποιείται για να πυροδοτήσουμε μια ενέργεια στο αυτόματο σύνθεσης. Μια τέτοια εντολή αρχίζει με τη λέξη κλειδί **fire**, ακολουθεί το είδος της ενέργειας που θα πυροδοτηθεί (**internal**/**output**/**input**), το όνομα του αυτόματου στο οποίο ανήκει η ενέργεια αυτή, ο χαρακτήρας **'.'** και το όνομα της ενέργειας μαζί με τις παραμέτρους που τυχόν να δέχεται η ενέργεια μέσα σε παρενθέσεις. Για παράδειγμα:

fire output C.send(m);

όπου C το αυτόματο που περιέχει την ενέργεια εξωτερική ενέργεια send.

Εντολή Follow: η εντολή αυτή χρησιμοποιείται για την εξέλιξη των τροχιών στο αυτόματο σύνθεσης. Μια τέτοια εντολή αρχίζει με τη λέξη κλειδί **follow**, στη συνέχεια το όνομα του αυτόματου στο οποίο ανήκει η συγκεκριμένη τροχιά, ο χαρακτήρας **'.'**, το όνομα της τροχιάς, η λέξη κλειδί **duration** και το χρονικό διάστημα κατά το οποίο θα εξελίσσεται η τροχιά.

Αυτόματο TIOA

Ο ορισμός ενός αυτόματου στη γλώσσα TIOA ξεκινάει με τη λέξη κλειδί **automaton** και μετά ακολουθεί το όνομα του αυτόματου. Μετά το όνομα του αυτόματου μπορεί να ακολουθούν μηδέν ή περισσότερες παράμετροι μέσα σε παρένθεση, οι οποίες πρέπει να είναι της μορφής όνομα παραμέτρου, χαρακτήρας **':'** και ο τύπος της παραμέτρου (για παράδειγμα id : Nat). Η κάθε παράμετρος διαχωρίζεται με το χαρακτήρα **','** από την επόμενη παράμετρο. Ακολούθως, εάν θέλουμε να ενσωματώσουμε κάποιο λεξιλόγιο που έχουμε δημιουργήσει σε κάποιο άλλο αρχείο (Vocabulary.tioa) πρέπει στην πρώτη γραμμή του αρχείου μας να τοποθετήσουμε τη λέξη κλειδί **include** και το όνομα του αρχείου που βρίσκεται το λεξιλόγιο μέσα σε διπλά εισαγωγικά (include "Vocabulary.tioa"). Στη συνέχεια με τη λέξη κλειδί **import**, πρέπει να γίνει ρητή αναφορά σε κάθε λεξιλόγιο, που

βρίσκεται μέσα στο αρχείο Vocabulary, το οποίο θέλουμε να χρησιμοποιήσουμε στο αυτόματο (import tcp_voc, algorithm_voc).

automaton ClockSync (u,r : Real, i : Index)

Ένα αυτόματο αποτελείται από τα εξής τμήματα:

- Υπογραφή Αυτομάτου (Action Signature).
- Μεταβλητές Καταστάσεων (States Variables).
- Σχέσεις Μεταβάσεων (Transition Relations).
- Τροχιές. (Trajectories).

Υπογραφή Αυτόματου: στο τμήμα αυτό γίνεται η δήλωση των ενεργειών του αυτομάτου. Επομένως, ξεκινάει με τη λέξη κλειδί **signature** και ακολουθούν οι δηλώσεις των ενεργειών. Για τη δήλωση της κάθε ενέργειας αρχίζουμε με τη λέξη κλειδί που χαρακτηρίζει σε ποια κατηγορία ανήκει η συγκεκριμένη ενέργεια, **input**, εάν είναι ενέργεια εισόδου, **output**, εάν είναι ενέργεια εξόδου και **internal**, εάν είναι εσωτερική ενέργεια. Μετά τοποθετούμε το όνομα της ενέργειας και πιθανόν μια λίστα παραμέτρων που επιθυμούμε να έχει η ενέργεια αυτή μέσα σε παρενθέσεις. Οι παράμετροι πρέπει να έχουν την εξής μορφή: όνομα παραμέτρου, χαρακτήρας **':'** και ο τύπος της παραμέτρου. Η κάθε παράμετρος διαχωρίζεται με το χαρακτήρα **','** από την επόμενη παράμετρο. Επίσης μπορούμε να περιορίσουμε το πεδίο τιμών που μπορεί να πάρει μια παράμετρος κάνοντας χρήση της λέξης κλειδί **where**.

signature

input RECEIVE (m : Real, j : Index, i : Index) where j ~= i

output SEND (m : Real, i : Index)

Μεταβλητές καταστάσεων: μετά από την υπογραφή του αυτομάτου ακολουθούν οι δηλώσεις των μεταβλητών καταστάσεων που θα έχει το αυτόματό μας. Ξεκινάει με τη λέξη κλειδί **states** και μετά ακολουθεί μια λίστα από δηλώσεις οι οποίες διαχωρίζονται με το χαρακτήρα **';'**. Τώρα η δήλωση μιας μεταβλητής θα πρέπει να έχει την εξής μορφή: όνομα παραμέτρου, χαρακτήρας **':'**, τύπος μεταβλητής, εάν επιθυμούμε να αρχικοποιήσουμε τη

μεταβλητή μας μπορούμε πολύ εύκολα να προσθέσουμε τους χαρακτήρες **':='** και την τιμή που θέλουμε να έχει η μεταβλητή. Δικαιούμαστε επίσης, να αρχικοποιήσουμε τη μεταβλητή με μη-ντετερμινιστικό τρόπο κάνοντας χρήση των εντολών choose και where. Επιπρόσθετα μπορούμε να περιορίσουμε το πεδίο τιμών όλων ή κάποιων από των μεταβλητών χρησιμοποιώντας μόνο μια εντολή με τη λέξη κλειδί **initially**.

```
states
        nextSend : Real := 0;
        maxOther : Real := 0;
        physClock : Real := 0;
        initially u > 0 /\ (0 <= r < l);
```

Σχέσεις Μεταβάσεων: το τμήμα αυτό περιέχει τους ορισμούς των ενεργειών που δηλώσαμε στο τμήμα signatures, δηλαδή το πώς θα αλλάζει η κατάσταση του αυτομάτου με την εκτέλεση μιας ενέργειας. Ξεκινάει με τη λέξη κλειδί **transitions** και ακολουθούν οι ορισμοί των ενεργειών. Ο ορισμός μιας ενέργειας αρχίζει με τη δήλωση της ενέργειας όπως αυτή δηλώθηκε στην υπογραφή του αυτομάτου, χωρίς όμως τους τύπους δεδομένων των παραμέτρων. Αμέσως μετά μπορούμε να δηλώσουμε τοπικές μεταβλητές χρησιμοποιώντας τη λέξη κλειδί **locals** και ακολούθως τη δήλωση μιας μεταβλητής καταστάσεων με τον ίδιο τρόπο όπως στο τμήμα states. Οι μεταβλητές που θα δηλωθούν στο τμήμα locals μπορούν να χρησιμοποιηθούν μόνο από τη συγκεκριμένη ενέργεια και συνήθως δηλώνονται όταν επιθυμούμε να έχουμε μια προσωρινή μεταβλητή. Επίσης, αν επιθυμούμε μπορούμε να ορίσουμε ένα σύνολο από preconditions, τα οποία δηλώνονται με τη λέξη κλειδί **pre** και μετά ακολουθεί ένα σύνολο λογικών εκφράσεων που μπορούν να αποτιμηθούν είτε με true, είτε με false. Ουσιαστικά εάν έχουμε preconditions πρέπει να αποτιμηθούν με true για να καταφέρει η ενέργεια να εκτελεστεί, διαφορετικά δεν εκτελείται. Στις εσωτερικές ενέργειες δεν μπορούν να οριστούν preconditions. Ακολούθως, μπορούμε να προσθέσουμε τη λέξη κλειδί **eff**, εδώ θα γίνεται ανάθεση νέων τιμών στις μεταβλητές καταστάσεων, οι οποίες θα οδηγήσουν το αυτόματό μας σε άλλη κατάσταση. Εάν δεν υπάρχει το τμήμα effects, τότε το αυτόματο παραμένει στην ίδια κατάσταση.

transitions

input RECEIVE (m, j, i)

eff maxOther := max(maxOther, m);

output SEND(m, i)

pre m = physclock ∧ physclock = nextSend;

eff nextsend := nextsend + u;

Τροχιές: οι τροχιές είναι αυτές που υλοποιούν την έννοια του χρόνου στα χρονισμένα αυτόματα Εισόδου/Εξόδου. Ξεκινάει με τη λέξη κλειδί **trajectories** και επειδή ένα αυτόματο μπορεί να έχει περισσότερες από μια τροχιές, οι οποίες να εξελίσσονται με διαφορετικό τρόπο, για κάθε νέα τροχιά πρέπει να προσθέσουμε τη λέξη κλειδί **trajdef** και το όνομα της τροχιάς. Μετά ακολουθούν τυχόν σταθερές σχέσεις με τη λέξη κλειδί **invariants**, τυχόν συνθήκες τερματισμού με τη λέξη κλειδί **stop when** και το ρυθμό εξέλιξης με τη λέξη κλειδί **evolve**. Οι σταθερές σχέσεις είναι κάποιες προϋποθέσεις οι οποίες πρέπει να αποτιμούνται πάντοτε με true.

trajectories

trajdef always

stop when physClock = nextSend

evolve (1 −r) ≤ d (physClock) ≤ (1 + r);

Σύνθεση Αυτομάτων TIOA

Για τη δημιουργία ενός πιο περίπλοκου συστήματος θα πρέπει να διασπάσουμε τις λειτουργίες τους σε απλούστερα και λιγότερο περίπλοκα αυτόματα και στη συνέχεια να τα συνθέσουμε για να αλληλεπιδρούν μεταξύ τους. Για τη σύνθεση των αυτομάτων πρέπει σε ένα καινούργιο αρχείο να ενσωματώσουμε τυχόν λεξιλόγια που ορίσαμε και ακολούθως να ενσωματώσουμε ένα-ένα τα αυτόματα τα οποία θα λάβουν μέρος στη σύνθεση. Αυτό γίνεται με τη λέξη κλειδί **include** και το όνομα του αρχείου σε διπλούς αποστρόφους (include "Process.tioa"). Μετά ακολουθεί το όνομα του αυτομάτου και οι παράμετροι που τυχόν να έχει μέσα σε παρενθέσεις, ενώ μετά τοποθετούμε τη λέξη κλειδί **components**. Για

κάθε αυτόματο που λαμβάνει μέρος στη σύνθεση δηλώνουμε ένα αναγνωριστικό, μετά το χαρακτήρα **':'**, το όνομα του συγκεκριμένου αυτομάτου, τα ονόματα των παραμέτρων που δέχεται χωρίς τους τύπους δεδομένων τους και το χαρακτήρα **';'**. Στη συνέχεια εάν υπάρχουν κρυφές ενέργειες δηλώνονται με τη λέξη κλειδί **hidden**. Ακολουθεί η λέξη κλειδί **schedule** στο οποίο καθορίζεται το χρονοδιάγραμμα που θα ακολουθήσει το αυτόματο σύνθεσης. Μπορούμε εδώ να δηλώσουμε τις μεταβλητές καταστάσεων του αυτομάτου σύνθεσης με τον ίδιο τρόπο όπως στα αυτόματα. Η υλοποίηση του σεναρίου προσομοίωσης μέσω του χρονοδιαγράμματος συμπεριλαμβάνεται ανάμεσα στις λέξεις κλειδιά **do** και **od**. Μέσα σε αυτό το τμήμα πυροδοτούνται οι ενέργειες κάνοντας χρήση της εντολής **fire** και εξελίσσονται οι τροχιές των αυτομάτων που λαμβάνουν μέρος στη σύνθεση με την εντολή **follow**.

2.5 Κανάλια Επικοινωνίας Μεταξύ Διεργασιών

Η επικοινωνία μεταξύ των διεργασιών ενός συστήματος είναι εφικτή μέσω των καναλιών επικοινωνίας. Το εργαλείο TEMPO υποστηρίζει δύο είδη καναλιών, το κανάλι επικοινωνίας MPI (Message Passing Interface) και το κανάλι TCP (Transmission Control Protocol). Τα δύο αυτά κανάλια είναι ήδη υλοποιημένα και μπορείς να τα βρεις στα εγχειρίδια χρήσης που παρέχει το εργαλείο. Στο Παράρτημα Α παραθέτουμε τις προδιαγραφές των καναλιών αυτών και των λεξιλογίων που χρησιμοποιούν. Να σημειώσουμε ότι τόσο στα κανάλια MPI, όσο και στα κανάλια TCP, η αποστολή και η λήψη μηνυμάτων γίνεται ανάμεσα σε μόνο δύο διεργασίες, όπου μία διεργασία εκτελεί την αποστολή και η άλλη τη λήψη ενός μηνύματος.

2.5.1 Κανάλια MPI

Το MPI (Message Passing Interface) [6], είναι ένα σύνολο προδιαγραφών για την επικοινωνία και τη μεταφορά δεδομένων. Χαρακτηρίζεται για την αποτελεσματικότητα και την ευελιξία του, ως επίσης και για την αποδοτικότητα που παρέχει. Στο μοντέλο TIOA, αυτό το κανάλι επικοινωνίας αποτελείται από το αυτόματο Send Mediator (SendMediator.tioa), το αυτόματο Receive Mediator (ReceiveMediator.tioa). Επίσης, αυτά τα δύο αυτόματα χρησιμοποιούν ένα συγκεκριμένο λεξιλόγιο (Vocabulary.tioa), στο οποίο μπορούμε να ορίσουμε νέους τύπους δεδομένων που θα χρησιμοποιήσουμε στις

προδιαγραφές. Αρχικά, στο MPI πρέπει να ορίσουμε τον αριθμό των διεργασιών που επιθυμούμε να έχουμε, οι οποίες θα εκτελούν το ίδιο πρόγραμμα. Κάθε διεργασία, κατά την εκκίνηση της εκτέλεσης χαρακτηρίζεται από ένα διακριτό ακέραιο αριθμό rank από το πεδίο 0 – (n-1), όπου n το πλήθος των διεργασιών. Δύο πολύ βασικές συναρτήσεις του MPI που χρησιμοποιούμε στις προδιαγραφές πιο κάτω είναι οι συναρτήσεις MPI_SIZE() και MPI_RANK(). Η πρώτη μας επιστρέφει το πλήθος των διεργασιών που δημιουργούνται και η δεύτερη μας επιστρέφει την ταυτότητα μιας συγκεκριμένης διεργασίας.

2.5.2 Κανάλια TCP

Το TCP (Transmission Control Protocol), στο μοντέλο TIOA, μοντελοποιείται ως ένα σύνθετο αυτόματο που αποτελείται από το αυτόματο Send Mediator (TCPSendMed.tioa), το αυτόματο Receive Mediator (TCPRecvMed.tioa) και το αυτόματο Channel Mediator (TCPChanMed.tioa). Το κάθε ένα από αυτά τα τρια αυτόματα χρησιμοποιεί και ένα συγκεκριμένο λεξιλόγιο το οποίο ορίζεται στο αρχείο TCPVocabs.tioa.

Στο αυτόματο Send Mediator, υπάρχει η προδιαγραφή για την αποστολή των μηνυμάτων από τον αποστολέα στον παραλήπτη. Για να είναι δυνατή η αποστολή των μηνυμάτων ανάμεσα σε δυο διεργασίες θα πρέπει πρώτα να εγκαθιδρυθεί μια σύνδεση μεταξύ τους με τις ενέργειες TCP_senderOpen και TCP_respSenderOpen. Τα μηνύματα που θα στείλει μια διεργασία αποθηκεύονται προσωρινά σε ένα buffer, που ονομάζεται sendBuffer, μέχρι να περάσουν στο αυτόματο του καναλιού. Εάν η σύνδεση μεταξύ δυο διεργασιών δημιουργηθεί επιτυχώς, τότε τα μηνύματα που βρίσκονται στο sendBuffer μεταφέρονται στο recvBuffer του Channel Mediator μέσω της ενέργειας TCP_write. Μια σύνδεση μπορεί να τερματιστεί ανά πάσα στιγμή με τις ενέργειες TCP_senderClose και TCP_respSenderOpen.

Στο αυτόματο Receive Mediator, υπάρχει η προδιαγραφή για την παράδοση μηνυμάτων στον παραλήπτη. Η ενέργεια RECEIVE, είναι υπεύθυνη για την παραλαβή μηνυμάτων και εκτελείται όταν υπάρχουν μηνύματα που πρέπει να παραληφθούν. Μια διεργασία για να μπορεί να παραλάβει ένα μήνυμα μόνο όταν ήδη έχει εγκαθιδρύσει μια σύνδεση μέσω του JVMServerSocket.Πιο συγκεκριμένα, είναι απαραίτητο να δημιουργηθεί και να δεσμευτεί ένα TCP socket για την επικοινωνία δύο διεργασιών, με τις ενέργειες TCP_bind και

TCP_respBind και ακολούθως να γίνει αποδοχή μέσω των ενεργειών TCP_accept και TCP_respAccept. Από τη στιγμή που μια διεργασία παραλήπτης αποδεχτεί οποιαδήποτε σύνδεση, τότε είναι σε θέση να παραλάβει τα μηνύματα της διαβάζοντάς τα από το κανάλι χρησιμοποιώντας τις ενέργειες TCP_read και TCP_respRead. Όσα μηνύματα παραλάβει προστίθενται στο recvBuffer της διεργασίας. Μια σύνδεση μπορεί να σταματήσει να ακούει σε κάποιο από τα sockets της και αυτό γίνεται κατορθωτό με τις ενέργειες TCP_stopAccepting και TCP_stopListening. Επίσης, οι συνδέσεις που έχουν δημιουργηθεί μεταξύ των διεργασιών είναι δυνατό να κλείσουν με τις ενέργειες TCP_rClose και TCP_rCloseStream.

Στο αυτόματο Channel Mediator, υπάρχει η προδιαγραφή για το κανάλι επικοινωνίας μεταξύ των αυτομάτων Send Mediator και Receive Mediator και αλληλεπιδρά με το πρωτόκολλο TCP. Ουσιαστικά, υπάρχουν οι αντίστοιχες ενέργειες που υπάρχουν και στα δυο αυτά αυτόματα και εκτελούνται ταυτόχρονα κατά την εκτέλεση των αντίστοιχων ενεργειών στα άλλα δυο αυτόματα εκτελώντας διαφορετικές λειτουργίες. Λόγω του ότι το εργαλείο TEMPO δεν μπορεί να αποκλείσει τις ενέργειες των αυτομάτων, το αυτόματο Channel Mediator χρησιμοποιεί timeouts για να αποφεύγει οποιουσδήποτε αποκλεισμούς των αιτημάτων στα sockets.

2.6 Παράδειγμα Υλοποίησης Απλού Αλγορίθμου στο Εργαλείο Tempo

Στο σημείο αυτό, θα παραθέσουμε την περιγραφή και την προδιαγραφή ενός απλού αλγόριθμου στο εργαλείο TEMPO, έτσι ώστε να γίνει καλύτερη κατανόηση του εργαλείου.

2.6.1 Περιγραφή Αλγόριθμου

Ο αλγόριθμος που θα υλοποιήσουμε είναι ο εξής: Έστω ότι υπάρχουν n διεργασίες και κάθε διεργασία έχει τη δική της ταυτότητα. Η κάθε διεργασία παίρνει μια τιμή x ως παράμετρο εισόδου και έχει ένα χρονικό διάστημα d_1 να στείλει αυτή τη τιμή σε όλους της τους γείτονες. Η κάθε διεργασία αφού συλλέξει όλες τις τιμές x από τους γείτονές της έχει ένα χρονικό διάστημα d_2 να αποφασίσει αν είναι η αρχηγός. Αρχηγός ανακηρύσσεται η διεργασία με την πιο μεγάλη τιμή x. Αν δύο τιμές x ισούνται, τότε αρχηγός εκλέγεται η διεργασία με τη μικρότερη ταυτότητα. Εφόσον, μια διεργασία είναι η αρχηγός θα πρέπει να

το ανακοινώσει και στις υπόλοιπες διεργασίες και τέλος να το τυπώσει στο χρήστη. Εάν μια διεργασία δεν στείλει την τιμή x στους γείτονες της στο χρονικό διάστημα που απαιτείται, τότε ξαναστέλνει την τιμή της. Εάν η διεργασία που έχει τη μεγαλύτερη τιμή x δεν ανακοινώσει στο χρονικό διάστημα που απαιτείται στις άλλες διεργασίες ότι είναι η αρχηγός , τότε το ανακοινώνει ξανά σε όλες τις διεργασίες.

2.6.2 Προδιαγραφή αλγορίθμου

Για να υλοποιήσουμε τον αλγόριθμο που περιγράψαμε πιο πάνω χρειάζεται να υλοποιήσουμε το λεξιλόγιο που χρησιμοποιούν τα αυτόματα του αλγορίθμου, το αυτόματο το οποίο θα αντιπροσωπεύει την κάθε διεργασία, τα αυτόματα που υλοποιούν το κανάλι επικοινωνίας μεταξύ των διεργασιών και τέλος το αυτόματο σύνθεσης. Η επικοινωνία μεταξύ των διεργασιών στο συγκεκριμένο παράδειγμα θα υλοποιηθεί με κανάλια επικοινωνίας MPI (Messege Passing Interface), οι προδιαγραφές του οποίου έχουν οριστεί στο Παράρτημα Α.

Το λεξιλόγιο του αλγορίθμου αυτού φαίνεται στο Σχήμα 2.6 και περιλαμβάνει το λεξιλόγιο που χρησιμοποιεί μια διεργασία, αλλά και το λεξιλόγιο που χρησιμοποιούν τα κανάλια επικοινωνίας MPI. Ορίζουμε ένα φυσικό αριθμό message, ο οποίος περιέχει την τιμή x της κάθε διεργασίας. Ακολούθως, ορίζουμε την πλειάδα mpi_message, η οποία αποτελείται από μια μεταβλητή του τύπου message (η τιμή x) και τρεις φυσικούς αριθμούς και καθορίζουν τον τρόπο με τον οποίο επικοινωνούν οι διεργασίες. Ο πρώτος φυσικός αριθμός, καθορίζει τον τύπο του μηνύματος, ο δεύτερος την ταυτότητα της διεργασίας που αποστέλλει το μήνυμα και ο τρίτος την ταυτότητα της διεργασίας που παραλαμβάνει το μήνυμα. Ο τύπος του μηνύματος είναι 1 εάν είναι το αρχικό μήνυμα που στέλλουν όλες οι διεργασίες τον αριθμό x σε όλους τους γείτονές τους και 2 εάν είναι το μήνυμα που στέλλει η διεργασία αρχηγός σε όλες τις άλλες για να τις ενημερώσει ότι είναι η αρχηγός.

```
%%% Algorithm vocabulary
vocabulary algorithm_voc
	types message : Nat
end

%%% MPI Channel vocabulary
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[value:message, type:Nat, 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 μιας διεργασίας. Το αυτόματο αυτό ονομάζεται TimedProcess και παίρνει ως παραμέτρους μια τιμή x, τον αριθμό της ταυτότητάς της id, το χρονικό διάστημα που έχει μέχρι να στείλει την τιμή x στους γείτονες της d_1 και το χρονικό διάστημα που έχει η διεργασία αρχηγός να στείλει στους γείτονές της ότι είναι η αρχηγός d_2. Ακολούθως, ορίζονται οι ενέργειες του συγκεκριμένου αυτομάτου για την παραλαβή (*RECEIVE*), την αποστολή (*SEND*) και την ετοιμασία των μηνυμάτων (*prepMessages*), για την αρχικοποίηση της συγκεκριμένης διεργασίας (*init*), για την ανακοίνωση του αρχηγού (*announce*) και κάποιες ενέργειες που θα ενεργοποιηθούν από τη μη έγκαιρη παράδοση των μηνυμάτων (*announceTimeout* και *sendIdTimeout*). Οι

μεταβλητές καταστάσεων του αυτομάτου αυτού ορίζονται στο τμήμα states και είναι οι ακόλουθες:

- ***clock:*** αντιπροσωπεύει το φυσικό ρολόι της διεργασίας και είναι τύπου AugmentedReal και αρχικοποιείται με 0.
- ***processClock:*** αντιπροσωπεύει το φυσικό ρολόι της κάθε διεργασίας, το οποίο υπολογίζει το χρόνο μέχρι να στείλει η κάθε διεργασία την τιμή της x σε όλες τις άλλες. Είναι τύπου AugmentedReal και αρχικοποιείται με 0.
- ***leaderClock:*** αντιπροσωπεύει το φυσικό ρολόι της κάθε διεργασίας, το οποίο υπολογίζει το χρόνο μέχρι να ανακοινώσει η διεργασία αρχηγός ότι είναι η αρχηγός σε όλες τις άλλες. Είναι τύπου AugmentedReal και αρχικοποιείται με 0.
- ***maxValue:*** είναι η φυσική μεταβλητή που αντιπροσωπεύει τη μεγαλύτερη τιμή x από τις τιμές x που έχει παραλάβει μια διεργασία και αρχικοποιείται με 0.
- ***maxId:*** είναι η φυσική μεταβλητή αντιπροσωπεύει την ταυτότητα της διεργασίας με τη μεγαλύτερη τιμή x και αρχικοποιείται με 0.
- ***acceptValue:*** είναι η Boolean μεταβλητή που καθορίζει εάν μια διεργασία μπορεί να δεχθεί μια νέα τιμή x, εφόσον στον αλγόριθμό μας επιθυμούμε μόνο μια τιμή να μπορεί να επεξεργάζεται ανά πάσα στιγμή. Αρχικοποιείται με false και γίνεται true όταν η διεργασία λάβει μια τιμή x**.**
- ***msgs:*** είναι μια κενή ακολουθία από μηνύματα του τύπου mpi_message. Σε αυτή την ακολουθία εισέρχονται τα μηνύματα που πρόκειται να στείλει η συγκεκριμένη διεργασία και εξέρχονται τα μηνύματα που στέλλονται επιτυχώς προς άλλες διεργασίες.
- ***leaderAnnounce:*** είναι η Boolean μεταβλητή που καθορίζει εάν η διεργασία αυτή είναι ο αρχηγός και αν είναι στέλλει το μήνυμα για να ενημερώσει και τις υπόλοιπες διεργασίες. Αρχικοποιείται με false και γίνεται true όταν η διεργασία αυτή έχει την μεγαλύτερη τιμή x και επομένως εκλέγεται ως αρχηγός.
- ***sentValue:*** είναι η Boolean μεταβλητή που καθορίζει εάν η διεργασία αυτή μπορεί να στείλει την τιμή τις στις άλλες διεργασίες. Αρχικοποιείται με false και γίνεται true όταν λάβει μια νέα τιμή από το χρήστη.

- ***messageCounter:*** είναι η ακέραια μεταβλητή που μετράει πόσα μηνύματα τύπου 1 έχει λάβει η συγκεκριμένη διεργασία από τους γείτονές της. Αρχικοποιείται με 0.
- ***printVar:*** είναι η Boolean μεταβλητή που καθορίζει εάν η διεργασία αυτή είναι η αρχηγός και πρέπει να ενημερώσει το χρήστη για αυτό. Αρχικοποιείται με false και γίνεται true όταν ανακηρυχτεί μια διεργασία ως αρχηγός.

Οι προδιαγραφή των ενεργειών που δηλώθηκαν στο τμήμα states βρίσκεται στο τμήμα signatures του αυτομάτου. Πιο συγκεκριμένα:

- Με την ενέργεια εισόδου ***init***, η διεργασία εάν δεν επεξεργάζεται κάποια τιμή ήδη, αρχικοποιεί τη μεταβλητή maxValue με την τιμή που έδωσε ο χρήστης και τη μεταβλητή maxId ίση με την ταυτότητά της. Ακολούθως κάνει τη μεταβλητή acceptValue ίση με false, έτσι ώστε να μην δέχεται νέες τιμές από το χρήστη μέχρι να τελειώσει με την επεξεργασία αυτής της τιμής. Τέλος, θέτει τη μεταβλητή sentValue ίση με true για να στείλει το μήνυμα τύπου 1 σε όλες τις άλλες διεργασίες.
- Με την ενέργεια εισόδου ***RECEIVE***, η διεργασία λαμβάνει ένα μήνυμα m, τύπου mpi_message. Εάν το μήνυμα αυτό δεν είναι κενό και ο τύπος του μηνύματος είναι ίσος με 1, τότε η διεργασία ελέγχει εάν η τιμή που παρέλαβε είναι μεγαλύτερη από τη μεταβλητή maxValue. Εάν είναι, θέτει τις μεταβλητές maxValue και maxId ίσες με τη μεταβλητή value και sender του μηνύματος m αντίστοιχα. Εάν όμως η τιμή που παρέλαβε είναι ίση με τη μεταβλητή maxValue, θέτει το maxId ίσο με το μικρότερο από τις δύο ταυτότητες. Ακολούθως, αυξάνει το μετρητή που μετράει τα μηνύματα τύπου 1 που έχει παραλάβει η διεργασία. Τώρα, αν ο τύπος του μηνύματος είναι ίσος με 2, τότε σημαίνει ότι η διαδικασία αυτή ολοκληρώθηκε, έχει ανακηρυχθεί ο αρχηγός και αρχικοποιούμε τις μεταβλητές μας για να μπορεί η διεργασία να λάβει μια νέα τιμή x από το χρήστη.
- Με τη διεργασία εξόδου ***SEND***, η διεργασία στέλλει ένα μήνυμα m με τις προϋποθέσεις ότι το μήνυμα m που θα σταλεί βρίσκεται πρώτο στην ακολουθία μηνυμάτων msgs και ότι η ακολουθία αυτή δεν είναι άδεια. Εάν το μήνυμα που θα σταλεί είναι τύπου 2, τότε το ρολόι leaderClock με την τιμή του clock και αν η

τιμή της μεταβλητής printVar είναι ίση με false, γίνεται true έτσι ώστε να ενημερωθεί ο χρήστης ότι αυτή η διεργασία είναι ο αρχηγός. Ενώ, αν το μήνυμα είναι τύπου 1, τότε το ρολόι processClock αρχικοποιείται με την τιμή του clock.

- Με την ενέργεια εξόδου ***printLeader***, η διεργασία η οποία εκλέγετε αρχηγός θα ενημερώσει το χρήστη ότι είναι ο αρχηγός και στο τέλος θα αρχικοποιήσει τις μεταβλητές για να μπορεί η διεργασία να λάβει μια νέα τιμή x από το χρήστη.
- Με την εσωτερική ενέργεια ***prepMessages***, κάθε διεργασία για να μπορέσει να εκτελέσει την ενέργεια αυτή θα πρέπει είτε η μεταβλητή sentValue, είτε η μεταβλητή leaderAnnounce να είναι αληθής. Αν η μεταβλητή sentValue είναι αληθής, τότε δημιουργεί ένα μήνυμα τύπου 1, που είναι ίσο με την πλειάδα [x,1,sender,destination] και εισάγεται στην ακολουθία msgs. Διαφορετικά, εάν η μεταβλητή leaderAnnounce είναι αληθής, τότε δημιουργεί ένα μήνυμα τύπου 2 που είναι ίσο με την πλειάδα [maxValue,2,sender,destination].
- Με την εσωτερική ενέργεια ***announce***, η διεργασία η οποία έλαβε μηνύματα από τις υπόλοιπες διεργασίες και το id της είναι ίσο με το maxId θα καταφέρει να εκτελέσει αυτή την ενέργεια και θα θέσει τη μεταβλητή leaderAnnounce true.
- Με την εσωτερική ενέργεια ***sendIdTimeout***, η διεργασία η οποία δεν έστειλε την τιμή της στις άλλες διεργασίες μέσα στο επιτρεπτό χρονικό διάστημα θα καταφέρει να εκτελέσει την ενέργεια αυτή. Θα θέσει τη μεταβλητή processClock ίση με 0 και θα ξαναστείλει την τιμή της σε όλες τις διεργασίες.
- Με την εσωτερική ενέργεια ***announceTimeout***, η διεργασία αρχηγός αν δεν στείλει το μήνυμα ανακοίνωσης στις άλλες διεργασίες μέσα στο επιτρεπτό χρονικό διάστημα θα καταφέρει να εκτελέσει την ενέργεια αυτή. Θα θέσει τη μεταβλητή leaderClock ίση με 0 και θα ξαναστείλει το μήνυμα σε όλες τις διεργασίες.

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(x:message, id:Nat, d1:DiscreteReal, d2:DiscreteReal)
signature
       input RECEIVE(m:Null[mpi message]), init
       output SEND(m:Null[mpi message]), printLeader
       internal prepMessages, announce, announceTimeout, sendIdTimeout
```

```
states
	% the physical clock of the automaton
	clock : AugmentedReal := 0;
	% the physical clock for the process
	processClock : AugmentedReal := 0;
	% the physical clock for the process which is leader
	leaderClock : AugmentedReal := 0;
	% the maximun value
	maxValue : Nat := 0;
	% the id that have the max value
	maxId : Nat := 0;
	% indicates that this process can accept values
	acceptValue : Bool := true;
	% an empty sequence of mpi messages
	msgs : Seq[Null[mpi_message]] := {};
	% indicates that process can announce that is the leader
	leaderAnnounce : Bool := false;
	% indicates that process can sent the value to the other processes
	sentValue : Bool := false;
	% a counter for the messages that the process has received
	messageCounter : Int := 0;
	% indicates that the process can print the value
	printVar : Bool := false;
transitions
	input init
		eff
			if (acceptValue = true) then
				if (maxValue < x) then
					maxValue := x;
					maxId := id;
				fi
				if (maxValue = x) then
					maxId := min(id, maxId);
				fi
				acceptValue := false;
				sentValue := true;
			fi
	input RECEIVE(m)
		eff
			if (m ~= nil) then
				if (val(m).type = 1) then
					if (maxValue < val(m).value) then
						maxValue := val(m).value;
						maxId := val(m).sender;
					fi
					if (maxValue = val(m).value) then
						maxId := min(maxId, val(m).sender);
					fi
					messageCounter := messageCounter + 1;
				fi
				if (val(m).type = 2) then
					acceptValue := true;
					maxId := 0;
					maxValue := 0;
					messageCounter := 0;
					processClock := 0;
					leaderClock := 0;
				fi
			fi
```

```
	output SEND(m)
		pre
			msgs ~= {};
			m = head(msgs);
		eff
			msgs := tail(msgs);
			if (val(m).type = 2 ) then
				leaderClock := clock;
				if (printVar = false) then
					printVar := true;
				fi
			else
				processClock := clock;
			fi
	internal announce
	pre
		messageCounter = MPI_Size() - 1;
		maxId = id;
	eff
		leaderAnnounce := true;
	internal prepMessages
		pre
			sentValue = true \/ leaderAnnounce = true;
		eff
			for n:Nat where(n < MPI_Size()) do
				if (n ~= id) then
					if (sentValue = true) then
						msgs := msgs |- embed([x, 1, id, n]);
					else
						msgs := msgs |- embed([maxValue, 2,
id, n]);
					fi
				fi
			od
			if (sentValue = true) then
				sentValue := false;
			else
				leaderAnnounce := false;
			fi
	output printLeader
		pre
			printVar = true;
		eff
			printVar := false;
			maxId := 0;
			maxValue := 0;
			messageCounter := 0;
			acceptValue := true;
			processClock := 0;
			leaderClock := 0;
	internal announceTimeout
		pre
			clock > leaderClock + d2;
			id = maxId;
		eff
			leaderClock := 0;
			leaderAnnounce := true;
```

```
internal sendIdTimeout
        pre
                clock > processClock + d1;
        eff
                processClock := 0;
                sentValue := true;
trajectories
trajdef Time
evolve d(clock) = 1;
```

Σχήμα 2.7: Διεργασία (Process.tioa).

Η προδιαγραφή της διεργασίας έχει ολοκληρωθεί και εφόσον τα κανάλια επικοινωνίας SendMediator και ReceiveMediador είναι ήδη υλοποιημένα αυτό που μας λείπει είναι το αυτόματο σύνθεσης. Ξεκινώντας θα πρέπει να ενσωματώσουμε τα αρχεία τα αρχεία που θα χρησιμοποιήσουμε στο αυτόματο σύνθεσης, δηλαδή το αρχείο του λεξιλογίου, της διεργασίας και των καναλιών επικοινωνίας. Αυτό γίνεται κατορθωτό με τη λέξη κλειδί **include**. Εδώ αξίζει να σημειωθεί ότι συμπεριλάβαμε το αρχείο Vocabulary.tioa μόνο στο αυτόματο σύνθεσης γιατί διαφορετικά αν το συμπεριλαμβάναμε και στο αρχείο της διεργασίας θα είχαμε συντακτικό λάθος για το διπλό ορισμό των τύπων που είναι ορισμένα στο λεξιλόγιο. Το αυτόματο σύνθεσης παίρνει ως παραμέτρους την τιμή x, την οποία θα πρέπει η κάθε διεργασία να τη στείλει σε όλες τις άλλες, το χρονικό διάστημα d_1, στο οποίο η κάθε διεργασία πρέπει να στείλει το μήνυμα με την τιμή x σε όλες τις άλλες και το χρονικό διάστημα d_2, στο οποίο η διεργασία αρχηγός πρέπει να ενημερώσει όλες τις άλλες ότι είναι η αρχηγός. Το αυτόματο σύνθεσης αποτελείται από ένα αυτόματο Process (P), ένα αυτόματο SendMediator (SM) και ένα αυτόματο ReceiveMediator (RM). Στο αυτόματο της διεργασίας περνιούνται ως παράμετροι οι τρεις παράμετροι του αυτομάτου σύνθεσης και η τιμή που μας επιστρέφει η συνάρτηση MPI_RANK(), η οποία αντιστοιχεί στη μοναδική της ταυτότητα. Ακολούθως, ορίζονται οι μεταβλητές καταστάσεων του αυτομάτου σύνθεσης:

- m: δηλώνει ένα μήνυμα του τύπου mpi_message, το οποίο αρχικοποιείται με nil().
- runs: δηλώνει τον αριθμό των επαναλήψεων του αλγορίθμου μας και αρχικοποιείται με MPI_SIZE()**2, όπου το MPI_SIZE() μας επιστρέφει τον αριθμό των διεργασιών.

Το πρόγραμμα μεταβάσεων και τροχιών (schedule) ξεκινάει με την πυροδότηση της ενέργειας ενέργειας εισόδου *init* του αυτομάτου Process και μετά με την πυροδότηση της εσωτερικής ενέργειας *prepMessages* για την προετοιμασία των μηνυμάτων. Ακολούθως, ξεκινάει το κομμάτι της επανάληψης στο οποίο αρχικά εξελίσσεται η τροχιά *Time* του αυτομάτου Process με ρυθμό 1, πυροδοτείται η ενέργεια *sendIdTimeout*, η οποία ελέγχει για τυχόν καθυστέρηση στην αποστολή των μηνυμάτων, στέλλονται τα μηνύματα που βρίσκονται στο buffer μιας διεργασίας και ακολούθως λαμβάνονται τα μηνύματα αυτά από τους παραλήπτες τους. Μετά πυροδοτείται η εσωτερική ενέργεια *announce* και η διεργασίας αρχηγός θα καταφέρει να ενημερώσει με μήνυμα τις άλλες διεργασίες ότι είναι η αρχηγός. Τέλος, θα ενημερώσει και το χρήστη ότι είναι αρχηγός.

```
%%% .: Vocabulary :.
include "Vocabulary.tioa"
imports algorithm_voc, mpi_status_voc, mpi_message_voc, mpi_request_voc, mpi_voc
%%% .: MPI mediator automata :.
include "ReceiveMediator.tioa"
include "SendMediator.tioa"
%%% Algorithm Automata
include "Process.tioa"
automaton TimedComposition(x:message, d1:DiscreteReal, d2:DiscreteReal)
	components
		P : Process(x, MPI_Rank, d1, d2);
		SM : SendMediator;
		RM : ReceiveMediator;
	schedule
		states
			m : Null[mpi_message] := nil();
			runs : Int := MPI_Size() ** 2;
	do
		fire input P.init;
		fire internal P.prepMessages;
		for i: Nat where(i < runs)do
			follow P.Time duration 1;
			fire internal P.sendIdTimeout;
			%% 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
```

```
                fire internal P.announce;
                if (P.leaderAnnounce = true) then
                        fire internal P.prepMessages;
                        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
                        fire output P.printLeader;
                fi
        od
```

Σχήμα 2.8: Αυτόματο Σύνθεσης (Composition.tioa).

2.6.3 Μετάφραση προδιαγραφής σε κώδικα JAVA

Εφόσον ολοκληρώσαμε την προδιαγραφή του αυτομάτου και έγινε συντακτικός έλεγχος της προδιαγραφής με το plugin checker που μας παρέχει το TEMPO, ήρθε η στιγμή να μεταφράσουμε την προδιαγραφή μας σε γλώσσα προγραμματισμού JAVA. Αυτό θα γίνει με τη χρήση του plugin tempo2java, αφού πρώτα καθορίσουμε πιο κανάλι επικοινωνίας θα χρησιμοποιήσουμε. Στο συγκεκριμένο παράδειγμα εφόσον κάνουμε χρήση MPI θα πρέπει να το επιλέξουμε αυτό. Επομένως είναι αναγκαίο να ακολουθήσουμε κάποια βήματα:

1. Επιλέγουμε από το Menu Bar το Windows και μετά Preferences.
2. Επιλέγουμε το TEMPO plugins και ακολούθως το Java generator.
3. Επιλέγουμε το MPI και πατάμε OK.
4. Επιλέγουμε από το ToolBar το εικονίδιο για το plugin tempo2java και ο κώδικας JAVA που αντιστοιχεί στην προδιαγραφή μας θα εμφανιστεί στην κονσόλα και θα αποθηκευτεί στο workspace.

2.7 Κατανεμημένο Σύστημα PlanetLab

Ο αλγόριθμος που μελετήθηκε στην παρούσα Διπλωματική Εργασία, θα αξιολογηθεί στο κατανεμημένο σύστημα PlanetLab [13,14]. Για αυτό θα γίνει μια σύντομη αναφορά και περιγραφή των λειτουργιών του PlanetLab.

2.7.1 Εισαγωγή στο PlanetLab

Το PlanetLab είναι ένα παγκόσμιο ερευνητικό δίκτυο το οποίο απαρτίζεται από μια συλλογή μηχανών που βρίσκονται σε όλο το κόσμο. Οι περισσότερες μηχανές ανήκουν σε ερευνητικά προγράμματα, ενώ κάποιες βρίσκονται σε co-location και σε κέντρα δρομολόγησης. Χρησιμοποιώντας τη πλατφόρμα αυτή, οι ερευνητές έχουν τη δυνατότητα να έχουν πρόσβαση σε ένα μεγάλο σύνολο από γεωγραφικά κατανεμημένες μηχανές οι οποίες προσφέρουν ρεαλιστικό περιβάλλον, εφόσον οι μηχανές βιώνουν συμφόρηση, σφάλματα κατάρρευσης και ρεαλιστικό φόρτο εργασίας. Αυτή τη στιγμή υπάρχουν περισσότερα από 600 ερευνητικά προγράμματα που τρέχουν σε μηχανές του PlanetLab.

2.7.2 Λειτουργίες PlanetLab

Κάθε οργανισμός που συμμετέχει στο PlanetLab πρέπει να δεσμευτεί ότι μπορεί να αφιερώσει τουλάχιστο δύο μηχανές (nodes), οι οποίες να χρησιμοποιούνται αποκλειστικά από το PlanetLab. Το Πανεπιστήμιο Κύπρου διαθέτει το δικό του site στο PlanetLab Europe και προσφέρει τρεις κόμβους. Το κάθε site που συμμετέχει στο PlanetLab χρειάζεται να απαρτίζεται από τα εξής άτομα:

- *Principal Investigator (PI):* είναι υπεύθυνος για τη διαχείριση των μερισμάτων (slices) και είναι υπεύθυνος για τη συμπεριφορά των slices που έχει δημιουργήσει. Πιο συγκεκριμένα είναι το μοναδικό άτομο που μπορεί να ενεργοποιήσει και να απενεργοποιήσει λογαριασμούς, αλλά και να δημιουργήσει και να αναθέσει μερίσματα στους χρήστες.
- *Technical Contact (Tech Contact):* είναι υπεύθυνος για την εγκατάσταση, τη συντήρηση και την παρακολούθηση των κόμβων ενός site.
- *User:* είναι οποιοδήποτε άτομο που αναπτύσσει εφαρμογές στο PlanetLab. Χρήστης είναι αυτός που έχει ένα έγκυρο λογαριασμό και του έχει ανατεθεί ένα μέρισμα από τον PI του οργανισμού στον οποίο συμμετέχει.
- *Administrative Contact:* είναι υπεύθυνος για να διαχειρίζεται τα συμβόλαια.
- *Authorized Official:* είναι υπεύθυνος να δεσμεύει τον οργανισμό νομικά.

Ο κάθε χρήστης σε μια μηχανή έχει το δικό του αποκλειστικό χώρο και δεν μπορεί οποιοσδήποτε άλλος χρήστης να έχει πρόσβαση σε αυτόν. Όμως, η απόδοση της μηχανής επηρεάζεται από όλους τους χρήστες που τη χρησιμοποιούν. Όποιος επιθυμεί να χρησιμοποιήσει την πλατφόρμα PlanetLab θα πρέπει να δημιουργήσει λογαριασμό. Σαν Πανεπιστήμιο Κύπρου επειδή συμμετέχουμε στο PlanetLab Europe (one-plus) πρέπει να κάνουμε την αίτηση μας για λογαριασμό εκεί. Εφόσον δημιουργήσουμε το λογαριασμό, περιμένουμε μέχρι ο PI του site μας να λάβει το αίτημά μας, να μας αποδεχτεί και να μας αναθέσει ένα site. Εφόσον τώρα έχουμε το δικό μας site μας δίνεται η δυνατότητα να χρησιμοποιήσουμε γύρω στις 150 μηχανές. Για να έχουμε πρόσβαση στις μηχανές αυτές θα πρέπει να δημιουργήσουμε ένα ζεύγος SSH key. Το κλειδί αυτό χωρίζεται σε δύο κατηγορίες, το public key και το private key. Εμείς θα πρέπει να ανεβάσουμε το public shh key στην ιστοσελίδα του PlanetLab. Για να δημιουργήσουμε το κλειδί αυτό χρειαζόμαστε πρόσβαση σε μια μηχανή με λειτουργικό σύστημα UNIX για να εκτελέσουμε την εντολή:

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

Η εντολή αυτή θα μας ζητήσει συνθηματικό, το οποίο θα χρησιμοποιούμε για να ενωθούμε στις μηχανές. Εφόσον είμαστε έτοιμοι να προσθέσουμε κόμβους στο slice μας ο χρήστης πρέπει να είναι προσεκτικός ποιες μηχανές διαλέγει γιατί υπάρχουν και μηχανές που έχουν καταρρεύσει. Ο χρήστης πρέπει να προσθέτει στο slice του μόνο κόμβους που έχουν ως status του το boot. Μέχρι το PlanetLab να μετάδοση το κλειδί σε όλες τις μηχανές του slice μας χρειάζεται κάποιος χρόνος, για αυτό αν κάποιος προσπαθήσει να ενωθεί μπορεί να του εμφανιστεί κάποιο μήνυμα λάθους. Για να ενωθούμε σε κάποια μηχανή θα πρέπει να εκτελέσουμε την εντολή:

ssh –l slice_name –i ~/.ssh/id_rsa host_name

όπου το slice_name είναι το όνομα του δικού μας slice και host_name το όνομα της μηχανής στην οποία επιθυμούμε να ενωθούμε. Ενώ για να ανεβάσουμε ένα αρχείο από τον τοπικό μας υπολογιστή σε μια μηχανή του PlanetLab θα πρέπει να χρησιμοποιήσουμε την εντολή:

scp –i ~/ .ssh/id_rsa file.ex slice_name@host_name:

όπου το file.ex είναι το όνομα του αρχείου που θέλουμε να ανεβάσουμε στη μηχανή του PlanetLab. Επειδή, οι μηχανές δεν έχουν κάτι εγκατεστημένο παρά μόνο το λειτουργικό Fedora Core 8, ότι χρειαζόμαστε θα πρέπει να το εγκαταστήσουμε. Για παράδειγμα, για την πειραματική αξιολόγηση του αλγόριθμου Efficient Replication of Large Data Object επειδή πρέπει να τρέξουμε εκτελέσιμο κώδικα Java, θα χρειαστούμε να εγκαταστήσουμε το πακέτο Java for RPM based Linux Platforms στις μηχανές μας. Θα πρέπει να ανεβάσουμε στις μηχανές το πακέτο αυτό και να το εγκαταστήσουμε. Για να μας επιτραπεί η άδεια να εγκαταστήσουμε το πακέτο, πρέπει να μπορούμε να λειτουργούμε σαν super user και αυτό γίνεται κατορθωτό με την ***su***. Επομένως τώρα με την εντολή αυτή θα εγκαταστήσουμε το πακέτο:

rpm -ivh package_name.rpm

Για να γίνουμε και πάλι απλοί users εκτελούμε την εντολή ***exit***. Επίσης, μπορούμε να εγκαταστήσουμε και τον emacs editor, με την εντολή:

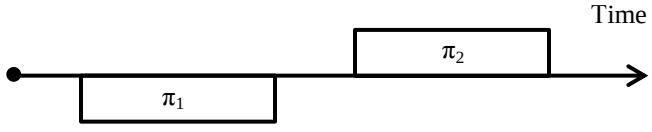
yum install emacs

Γενικά, κάποιος μπορεί να βρει περισσότερες πληροφορίες στην ιστοσελίδα του PlanetLab Europe.

2.8 Ορισμός Ατομικότητας

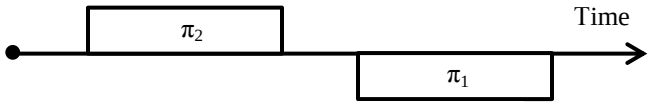
Αρχικά ας εξηγήσουμε τη σχέση προτεραιότητας που μπορούν να έχουν δύο λειτουργίες π_1 και π_2 για να μας βοηθήσει να ορίσουμε την έννοια της ατομικότητας [22,23]:

- Η π_1 ***προηγείται*** της π_2 εάν η απάντηση της π_1 συμβαίνει χρονικά πριν από την αίτηση της π_2:



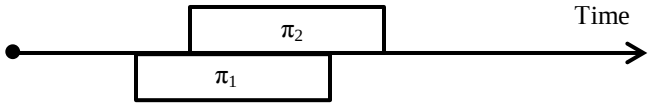
Σχήμα 2.9: Σχέση Προηγείται.

- Η π_1 ***έπεται*** της π_2 εάν η αίτηση της π_1 συμβαίνει μετά από την απάντηση της π_2:



Σχήμα 2.10: Σχέση Έπεται.

- Η π_1 είναι ***ταυτόχρονη*** με την π_2 εάν η π_1 ούτε προηγείται και ούτε έπεται της π_2:



Σχήμα 2.11: Σχέση Ταυτοχρονίας.

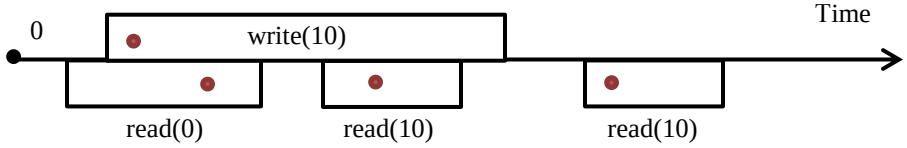
Οι λειτουργίες που μπορούν να εκτελεστούν είνα η ανάγνωση, *read(X)* και η εγγραφή, *write(X,v)*, όπου X το όνομα της μεταβλητής και v η καινούρια τιμή που θα έχει. Οι λειτουργίες αυτές να πάρουν ως απάντηση *return(X,v)* και *ack(X)*, αντίστοιχα.

Για να ισχύει η συνθήκη της ατομικότητας θα πρέπει να ικανοποιούνται οι εξής προτάσεις:

- Εάν μια λειτουργία ανάγνωσης δεν είναι ταυτόχρονη με μια λειτουργίας εγγραφής, τότε η ανάγνωση επιστρέφει την τιμή που γράφτηκε από την τελευταία εγγραφή που προηγείται.
- Εάν μια ανάγνωση είναι ταυτόχρονη με μια εγγραφή, τότε η ανάγνωση επιστρέφει είτε την τιμή της αμέσως προηγούμενης εγγραφής, είτε την τιμή που αναγράφεται από την ταυτόχρονη εγγραφή.

Επίσης, μπορούμε να συρρικνώσουμε κάθε λειτουργία σε ένα χρονικό σημείο, έτσι ώστε να δίνουμε την ψευδαίσθηση μιας ακολουθιακής εκτέλεσης, όπως φαίνεται στο Σχήμα 2.12.

Για παράδειγμα,



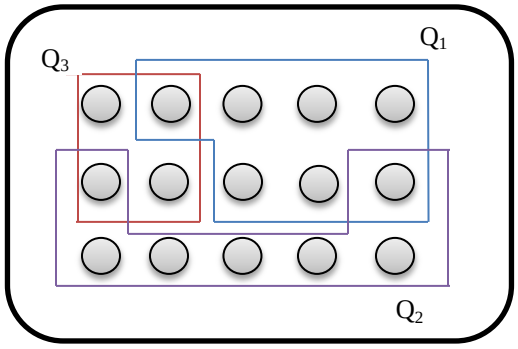
Σχήμα 2.12: Συνθήκη Ατομικότητας.

2.9 Σύστημα Απαρτίας (Quorum Systems)

Η απαρτία [22] ως λέξη, δηλώνει τον ελάχιστο αριθμό μελών που χρειάζονται για να παρθεί μια απόφαση που αφορά την ομάδα. Για παράδειγμα η πλειοψηφία (majority) είναι ένα σύστημα απαρτίας στο οποίο αναμένουμε ότι αν έχουμε n στοιχειά θα πρέπει να λάβουν μέρος στην απόφαση (n/2 + 1) - μέλη. **Ορισμός:** έστω S είναι ένα σύνολο και (Q_1,Q_2) είναι ένα ζευγάρι του συστήματος απαρτίας πάνω στο S εάν ισχύουν ότι:

- $Q_1, Q_2 \subseteq 2^S$.
- $\forall\, q_1 \in Q_1 \,\forall\, q_2 \in Q_2 : Q_1 \cap Q_2 \neq \emptyset$.

Στο Σχήμα 2.13, έχουμε το σύστημα απαρτίας S που αποτελείται από το σύνολο $\{Q_1, Q_2, Q_3\}$, όπου τα στοιχεία του συνόλου αυτά είναι απαρτίες. Η ιδιότητα που έχει αυτό το σύνολο είναι ότι κάθε ζεύγος απαρτιών τέμνονται.



Σχήμα 2.13: Σύστημα Απαρτίας S.

2.10 Αλγόριθμος ABD

Ο αλγόριθμος αυτός παρουσιάστηκε από τους Attiya, Bar-Noy και Dolev [22] και αφορούσε την προσομοίωση μιας single-writer/multi-reader κοινόχρηση μνήμης μέσα σε ένα δίκτυο ανταλλαγής μηνυμάτων. Ο αλγόριθμος αυτός επεκτάθηκε αργότερα από τους Lynch και Shvartman [11], σε μοντέλο multi-writer/multi-read και στο υπόλοιπο της παρούσας διπλωματικής εργασίας θα αναφερόμαστε στον MWMR αλγόριθμο ως ABD. Ο αλγόριθμος αυτός εκτελεί αντιγραφές βασισμένες σε απαρτίες, οι οποίες δεν χρειάζονται οποιοδήποτε επιπλέον μηχανισμό για έλεγχο ταυτοχρονίας, αντίθετα με άλλους

αλγόριθμους αντιγραφής. Αυτό μειώνει την πολυπλοκότητα του αλγόριθμου και βελτιώνει την ανεκτικότητα που έχει στα σφάλματα, αλλά και την αποδοτικότητα του.

Περιγραφή Αλγόριθμου:

Το σύστημα αποτελείται από δυο σύνολα οντοτήτων τους πελάτες (clients), οι οποίοι εκτελούν τις δυο λειτουργίες, ανάγνωση (read) και εγγραφή (write) και τους εξυπηρετητές οι οποίοι αποθηκεύουν τα δεδομένα. Κάθε εγγραφή χαρακτηρίζεται από μια χρονοσφραγίδα (tag), η οποία απαρτίζεται από ένα φυσικό αριθμό και το ID του συγκεκριμένου χρήστη που έκανε την εγγραφή. Το σύνολο των χρονοσφραγίδων ταξινομείται λεξικογραφικά. Ο κάθε εξυπηρετητής αποθηκεύει στα δεδομένα του την τιμή (value) και την χρονοσφραγίδα μιας εγγραφής. Για την εκτέλεση μιας εγγραφής, ο εγγραφέας αρχικά πρέπει να διαβάσει τη χρονοσφραγίδα από μια απαρτία εξυπηρετητών και ακολούθως να επιλέξει από αυτά τη μεγαλύτερη χρονοσφραγίδα και να αυξήσει την τιμή αυτή κατά ένα. Τέλος, ολοκληρώνει την εγγραφή του με το να γράψει τα δεδομένα αυτά σε μια απαρτία εξυπηρετητών.

Η λειτουργία της ανάγνωσης αποτελείται από δυο φάσεις, τη φάση επερώτησης (query phase) και τη φάση μετάδοσης (propagation phase). Κατά την εκτέλεση την πρώτης φάσης, ο αναγνώστης διαβάζει τις χρονοσφραγίδες (tag) και τις τιμές (values) από μια απαρτία εξυπηρετητών. Ακολούθως, στη δεύτερη φάση ο αναγνώστης επιλέγει την τιμή (value) με τη μεγαλύτερη χρονοσφραγίδα και γράφει την τιμή αυτή μαζί με τη χρονοσφραγίδα της σε μια απαρτία εξυπηρετητών. Με την ολοκλήρωση της εγγραφής αυτή, ο αναγνώστης επιστρέφει την τιμή που επέλεξε.

Ο εξυπηρετητής, αν λάβει μήνυμα write(<timestamp,v>) και η τοπική του χρονοσφραγίδα είναι μικρότερη από τη χρονοσφραγίδα του μηνύματος τότε αλλάζει τη τιμή του αντιγράφου του σε v και την τοπική χρονοσφραγίδα σε timestamp. Τέλος, απαντάει σε όλα τα μηνύματα που παραλαμβάνει με reply(<timestamp,v>), όπου v η τιμή του αντιγράφου.

Κεφάλαιο 3

Περιγραφή Αλγόριθμου Διαχείρισης Αρχείων LDR

3.1 Επισκόπηση Αλγόριθμου

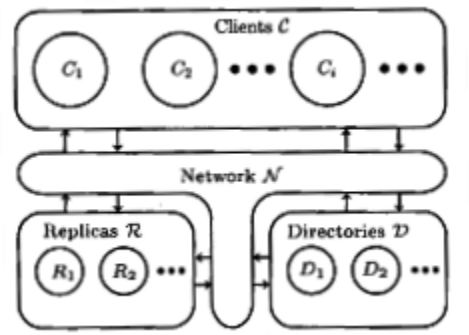
Η κύρια ιδέα του αγλόριθμου Layred Data Replication (LDR) [6,15,16], είναι ότι η αντιγραφή των δεδομένων γίνεται σε αυθαίρετες τοποθεσίες και μετά γίνεται χρήση των ευρητηρίων για να βρεθούν οι ενημερωμένοι αντιγραφείς. Ο αλγόριθμος αυτός βασίζεται πάνω στον αλγόριθμο ABD που εξηγήσαμε στο προηγούμενο κεφάλαιο, με τη διαφορά ότι εδώ η λειτουργία της ανάγνωσης είναι πιο αποδοτική, ενώ η λειτουργία της εγγραφής παραμένει το ίδιο αποδοτική με πριν. Ο αλγόριθμος είναι βασισμένος στο μοντέλο πελάτη-εξυπηρετητή, όπου η κάθε οντότητα αντιπροσωπεύεται από ένα αυτόματο Εισόδου/Εξόδου και η επικοινωνία τους γίνεται μέσω FIFO ασύγχρονου δικτύου. Με τη σειρά τους οι εξυπηρετητές χωρίζονται σε δυο μη-κενά σύνολα αυτομάτων, τα ευρετήρια και τους αντιγραφείς. Οι πελάτες είναι αυτοί που στέλλουν τα αιτήματα για ανάγνωση και εγγραφή

των τιμών αυτών. Ο αλγόριθμος ονομάζεται LDR, γιατί έχει δυο layers από εξυπηρετητές, τα ευρετήρια στα οποία αποθηκεύονται οι αντιγραφείς, που έχουν την πιο ενημερωμένη τιμή, μαζί με τη χρονοσφραγίδα και τους αντιγραφείς οι οποίοι αποθηκεύουν τα πραγματικά δεδομένα, για παράδειγμα ένα αρχείο.

Ο αλγόριθμος μπορεί να αντιμετωπίσει σφάλματα κατάρρευσης, δηλαδή να σταματήσει από κάποια χρονική στιγμή και μετά να εκτελεί βήματα. Για το λόγο αυτό κάθε i ϵ *C* U *D* U *R*, έχει μια εσωτερική ενέργεια $fail_i$ η οποία με την εκτέλεση της δηλώνει ότι η διεργασία i έχει καταρρεύσει.

3.2 Αρχιτεκτονική

Το μοντέλο δικτύου του αλγόριθμου όπως αναφερθήκαμε και πιο πάνω αποτελείται από τους πελάτες C, και τους servers S. Οι serves διαχωρίζονται σε ευρητήρια D και αντιγραφείς R, όπου S = R U D. Για κάθε i ϵ R, υπάρχει ένα αυτόματο R_i, για κάθε i ϵ C, υπάρχει ένα αυτόματο C_i και για κάθε i ϵ D, υπάρχει ένα αυτόματο D_i. Η αρχιτεκτονική του αλγόριθμου LDR φαίνεται στο Σχήμα 3.1.



Σχήμα 3.1: LDR Αρχιτεκτονική [15].

3.3 Ορισμοί

Σε αυτό το σημείο θα δώσουμε λεπτομερή περιγραφή κάποιων ορισμών για να είναι πιο εύκολη η κατανόηση του αλγόριθμου:

- *Χρονοσφραγίδα (tag):* χρησιμοποιείται από τους πελάτες για να χαρακτηρίσουν την εγγραφή τους. Έστω ότι, T = $\mathbb{N}$ x C όπου $\mathbb{N}$ το σύνολο των φυσικών αριθμών και C

το σύνολο των πελατών. Εφαρμόζεται λεξικογραφική ταξινόμηση πάνω στις χρονοσφραγίδες. Αρχικοποιείται με μια αυθαίρετη τιμή t_0, όπου $t_0 < t \epsilon T$.

- *Σύστημα απαρτίας:* εδώ το σύστημα απαρτίας αποτελείται από τα δύο σύνολα $\{Q_R, Q_W\}$ και εφαρμόζεται πάνω στα ευρετήρια. Ισχύει ότι, $Q_R, Q_W \subseteq 2^S$ και $Q_1 \epsilon Q_R$ και $Q_2 \epsilon Q_W$, τότε $Q_1 \cap Q_2 \neq \emptyset$.
- *Σφάλματα κατάρρευσης:* εδώ ο αριθμός των σφαλμάτων που μπορούν να συμβούν συμβολίζεται με f και ισχύει ότι $f < |R|$.
- *Αντικείμενο x:* Θέτουμε ότι x είναι το αντικείμενο στο οποίο θα εκτελούμε τις λειτουργίες ανάγνωσης και εγγραφής. Παίρνει τιμές από ένα σύνολο V και είναι αρχικοποιημένο με την τιμή v_0. Εάν επιθυμούμε να έχουμε περισσότερα από ένα αντικείμενα, τότε πρέπει να εκτελέσουμε πολλαπλές φορές τον αλγόριθμο.

3.4 Περιγραφή λειτουργίας ενός Πελάτη

Η υπογραφή του αυτόματου του Client αποτελείται από τις ενέργειες εισόδου $read_i$, $write(v)_i$ όπου $v \epsilon V$, $recv(m)_{i,j}$ και $fail_i$ και τις ενέργειες εξόδου $read\text{-}ok(v)_i$ όπου $v \epsilon V$, $write\text{-}ok_i$ και $send(m)_{i,j}$. Οι μεταβλητές καταστάσεων του αυτόματου είναι: η μεταβλητή *acc*, η οποία αποθηκεύει τα ids των διεργασιών που ανταποκρίνονται σε ένα μήνυμα του πελάτη, τη μεταβλητή *phase*, στην οποία αποθηκεύεται η φάση στην οποία βρίσκεται η διεργασία μια συγκεκριμένη στιγμή, η μεταβλητή *tag*, στην οποία αποθηκεύεται η πιο ενημερωμένη χρονοσφραγίδα, η μεταβλητή *val* στην οποία αποθηκεύεται η ενημερωμένη τιμή της x και ένας φυσικός αριθμός *mid* που χαρακτηρίζει τα μηνύματα που στέλλει η διεργασία. Επίσης, έχουμε μια ακολουθία μηνυμάτων *msg* στην οποία αποθηκεύονται προσωρινά τα μηνύματα που θα στείλει ο πελάτης.

Ο πελάτης δέχεται από το χρήστη εντολές είτε για να διαβάσει την πιο ενημερωμένη τιμή του x, είτε για να γράψει μια νέα τιμή σε αυτή. Μαρκάρει το κάθε μήνυμα που στέλλει με ένα φυσικό αριθμό mid, ο οποίος του επιτρέπει να γνωρίζει πόσο πρόσφατες είναι οι απαντήσεις που λαμβάνει. Το φυσικό αυτό αριθμό συμπεριλαμβάνουν στα μηνύματά τους, στο πεδίο id και τα ευρετήρια και οι αντιγραφείς, όταν απαντούν σε ένα μήνυμα προς τον

client. Εάν το id < mid, τότε ο client αγνοεί τη συγκεκριμένη απάντηση, εφόσον είναι πιο παλιά.

Στο Σχήμα 3.2**,** υπάρχουν οι προδιαγραφές των μεταβάσεων του αυτόματου Client.

3.4.1 Λειτουργία Ανάγνωσης

Ένας πελάτης C_i κατά τη εκτέλεση μιας ανάγνωσης ακολουθεί τις εξής τέσσερις φάσεις: read-directories-read *(rdr),* read-directories-write *(rdw),* read-replicas-read *(rrr)* και read-ok *(rok).* Η κάθε φάση αντιπροσωπεύει και ένα επικοινωνιακό γύρο. Αρχικά ο C_i, παραλαμβάνει ένα αίτημα για ανάγνωση $read_i$, στέλλει ένα μήνυμα <*rread,mid*> σε όλα τα ευρετήρια και εισέρχεται στη φάση rdr. Ο πελάτης τώρα ψάχνει να βρει το σύνολο των αντιγραφέων με την τελευταία τιμή του x. Με τη σειρά τους τα ευρετήρια θα απαντήσουν πίσω στους πελάτες με ένα μήνυμα <*rread-ok,S,t,mid*>, όπου S είναι ένα σύνολο από αντιγραφείς και το t είναι μια χρονοσφραγίδα. Ο πελάτης C_i, περιμένει απάντηση από μια απαρτία Q_R και τότε επιλέγει το ζευγάρι (S,t) με τη μεγαλύτερη τιμή χρονοσφραγίδας. Ο C_i θέτει το ζευγάρι (utd,tag) ίσο με το (S,t), στέλλει σε όλα τα ευρετήρια το μήνυμα <*rwrite,utd,tag,mid*> για να γράψει σε αυτούς το ζευγάρι (utd,tag) και εισέρχεται στη φάση rdw (το utd αντιπροσωπεύει το σύνολο των αντιγραφέων από τα οποία ο πελάτης C_i θα διαβάσει την τιμή του x).Ο C_i περιμένει απάντηση με το μήνυμα <*rwrite-ok,mid*> από μια απαρτία των αντιγραφέων. Ακολούθως, ο C_i στέλλει το μήνυμα <*read,tag,mid*> σε ένα replica που ανήκει στο σύνολο utd για να διαβάσει τη τιμή του x και εισέρχεται στη φάση rrr. Όταν ο C_i πάρει απάντηση με το μήνυμα <*read-ok,v,t,id*> από τον αντιγραφέα και ανταποκρίνεται στο χρήστη με την ενέργεια read-ok$(v)_i$ και εισέρχεται στη φάση rok.

3.4.2 Λειτουργία Εγγραφής

Ένας πελάτης C_i κατά τη εκτέλεση μιας εγγραφής ακολουθεί τις εξής τέσσερις φάσεις: write-directories-read (*wdr*), write-replicas-write (*wrw*), write-directories-write (*wdr*) και write-ok (*wok*). Οι τρεις πρώτες φάσεις αντιπροσωπεύουν ένα επικοινωνιακό γύρο, ενώ η τελευταία φάση αντιπροσωπεύει μισό επικοινωνιακό γύρο από τους πελάτες στους αντιγραφείς. Αρχικά ο C_i, παραλαμβάνει ένα αίτημα εγγραφής write$(v)_i$, για να ξεκινήσει η εγγραφή της τιμής v και στέλλει το μήνυμα <*wread,mid*> σε όλα τα ευρετήρια. Σε αυτό το

σημείο είναι που ψάχνει να βρει τη μεγαλύτερη χρονοσφραγίδα (tag) για να μαρκάρει τη δική του εγγραφή και εισέρχεται στη φάση wdr. Ακολούθως, περιμένει απάντηση από μια απαρτία των ευρετηρίων με το μήνυμα <*wread-ok,t,id*>, όπου το t αντιστοιχεί στη χρονοσφραγίδα και διαλέγει το μεγαλύτερο t=(n,i') από αυτά. Στη συνέχεια θέτει το δικό του tag=(n+1,i') και στέλλει το μήνυμα <*wwrite,val,tag,mid*> σε όλους τους αντιγραφείς για να γράψει την τιμή (val,tag) στα δεδομένα των αντιγραφέων και εισέρχεται στη φάση wrw. Ο C_i, αναμένει απάντηση από τουλάχιστον f+1 αντιγραφείς για να του επιβεβαιώσουν ότι η εγγραφή θα επιζήσει εάν καταρρεύσουν f αντιγραφείς. Όταν λάβει τις απαντήσεις από τους αντιγραφείς στέλλει το μήνυμα <*wwrtie,acc,tag,mid*> σε όλα τα ευρετήρια για να τα ενημερώσει για το νέο ζευγάρι (val,tag) και εισέρχεται στη φάση wdw. Ο C_i, περιμένει μήνυμα <*wwrite-ok,id*> από μια απαρτία των ευρετηρίων και στέλλει το μήνυμα <*secure,tag,mid*> σε όλους τους αντιγραφείς που έχει το σύνολο acc του C_i. Τέλος, ανταποκρίνεται στο χρήση με την εκτέλεση της εξωτερικής ενέργειας write-ok_i. Με το μήνυμα secure που στέλλει ο C_i στους αντιγραφείς τους ενημερώνει για το γεγονός ότι η νέα εγγραφή βρίσκεται ήδη σε μια απαρτία των ευρετηρίων και για το λόγο αυτό καμία ανάγνωση από εδώ και πέρα δεν πρέπει να επιστρέψει μια τιμή του x με μικρότερη χρονοσφραγίδα από αυτό που συμπεριλαμβάνει το μήνυμα.

```
input read_i
Effect:
    mid ← mid + 1
    for all j ∈ D do msg[j] ← ⟨read, mid⟩
    phase ← rdr

input write(v)_i
Effect:
    val ← v; mid ← mid + 1
    for all j ∈ D do msg[j] ← ⟨read, mid⟩
    phase ← wdr

input fail_i
Effect:
    stop taking locally-controlled steps

output read-ok(v)_i
Precondition:
    (val = v) ∧ (phase = rok)
Effect:
    phase ← idle

output write-ok_i
Precondition:
    phase = wok
Effect:
    phase ← idle

output send(m)_{i,j}
Precondition:
    msg[j] = m
Effect:
    msg[j] ← ⊥

input recv(m)_{j,i} where (m = ⟨read-ok, S, t, id⟩)
Effect:
    if (phase = rdr) ∧ (id = mid) then
      acc ← acc ∪ {j}
      if (t > tag) then
        tag ← t; utd ← S
      if (∃Q ∈ Q_R : Q ⊆ acc) then
        mid ← mid + 1
        for all j ∈ D do msg[j] ← ⟨write, utd, tag, mid⟩
        acc ← ∅; phase ← rdw

input recv(m)_{j,i} where (m = ⟨write-ok, id⟩)
Effect:
    if (phase = rdw) ∧ (id = mid) then
      acc ← acc ∪ {j}
      if (∃Q ∈ Q_W : Q ⊆ acc) then
        mid ← mid + 1
        for all j ∈ utd do msg[j] ← ⟨read, tag, mid⟩
        acc ← ∅; phase ← rrr

input recv(m)_{j,i} where (m = ⟨read-ok, v, t, id⟩)
Effect:
    if (phase = rrr) ∧ (id = mid) then
      val ← v; tag ← t; phase ← rok

input recv(m)_{j,i} where (m = ⟨read-ok, t, id⟩)
Effect:
    if (phase = wdr) ∧ (id = mid) then
      acc ← acc ∪ {j}
      if (t > tag) then
        tag ← t                    //tag = (n, i')
      if (∃Q ∈ Q_R : Q ⊆ acc) then
        mid ← mid + 1; tag ← (n + 1, i)
        for all j ∈ R do msg[j] ← ⟨write, val, tag, mid⟩
        acc ← ∅; phase ← wrw

input recv(m)_{j,i} where (m = ⟨write-ok, id⟩)
Effect:
    if (phase = wrw) ∧ (id = mid) then
      acc ← acc ∪ {j}
      if (|acc| > f) then
        mid ← mid + 1
        for all j ∈ D do msg[j] ← ⟨write, acc, tag, mid⟩
        acc ← ∅; phase ← wdw
    else if (phase = wdw) ∧ (id = mid) then
      acc ← acc ∪ {j}
      if (∃Q ∈ Q_W : Q ⊆ acc) then
        mid ← mid + 1
        for all j ∈ R do msg[j] ← ⟨secure, tag, mid⟩
        acc ← ∅; phase ← wok
```

Σχήμα 3.2: Σχέσεις Μεταβάσεων του αυτόματου Client.[15]

3.5 Περιγραφή λειτουργίας ενός Αντιγραφέα

Η υπογραφή του αυτόματου του Replica αποτελείται από την ενέργεια εισόδου $recv(m)_{i,j}$ και $fail_i$, την ενέργεια εξόδου $send(m)_{i,j}$ και τις εσωτερικές ενέργειες $gossip_i$ και gc_i. Οι μεταβλητές καταστάσεων του αυτόματου είναι η μεταβλητή *data* η οποία είναι μια πλειάδα της μορφής [value,tag,bit], όπου value είναι η τιμή του x, tag η χρονοσφραγίδα που μας δείχνει πόσο πρόσφατη είναι η τιμή x και bit, το bit ασφαλείας (security bit) που παίρνει τιμές 0 και 1 και μας δείχνει εάν η εγγραφή έγινε επιτυχώς. Άρα εάν έχουμε [*,t,0] σημαίνει ότι η εγγραφή δεν έχει ολοκληρωθεί ακόμη, ενώ αν έχουμε [*,t,1] υποδεικνύει ότι η εγγραφή ολοκληρώθηκε και είναι ασφαλής. Επίσης, έχουμε μια ακολουθία μηνυμάτων *msg* στην οποία αποθηκεύονται προσωρινά τα μηνύματα που θα στείλει ο αντιγραφέας. Επίσης, εφαρμόζει συλλογή παλιότερων τιμών με τη εσωτερική ενέργεια gc_i (gcgarbage-collect) και διάδοση νέων τιμών με την εσωτερική ενέργεια $gossip_i$.

3.5.1 Λειτουργία Ανάγνωσης

Όταν λάβει το μήνυμα <*read,t,mid*> από κάποιο πελάτη που ζητάει να διαβάσει την τιμή με χρονοσφραγίδα t, τότε ελέγχει εάν υπάρχει στη μεταβλητή data η πλειάδα [v,t,*] και την επιστρέφει. Διαφορετικά, σημαίνει ότι ο garbage-collector έχει διαγράψει τη τιμή αυτή και ο αντιγραφέας πρέπει να επιστρέψει τη μεγαλύτερη ασφαλής τιμή, maxst(data) μαζί με τη χρονοσφραγίδα της. Η τιμή maxst(data) ορίζεται ως εξής:

$$\text{maxst}(data) = \begin{cases} (v,t) & ((v,t,1) \in data) \wedge ((v',t',1) \in data \Rightarrow t \geq t') \\ (v_0,t_0) & \nexists v,t : (v,t,1) \in data \end{cases}$$

3.5.2 Λειτουργία Εγγραφής

Όταν παραλάβει το μήνυμα <*write,v,t,mid*> από κάποιο client που θέλει να γράψει μια νέα τιμή, ο αντιγραφέας προσθέτει στη μεταβλητή data την πλειάδα [v,t,0].

3.5.3 Λειτουργία Διάδοσης

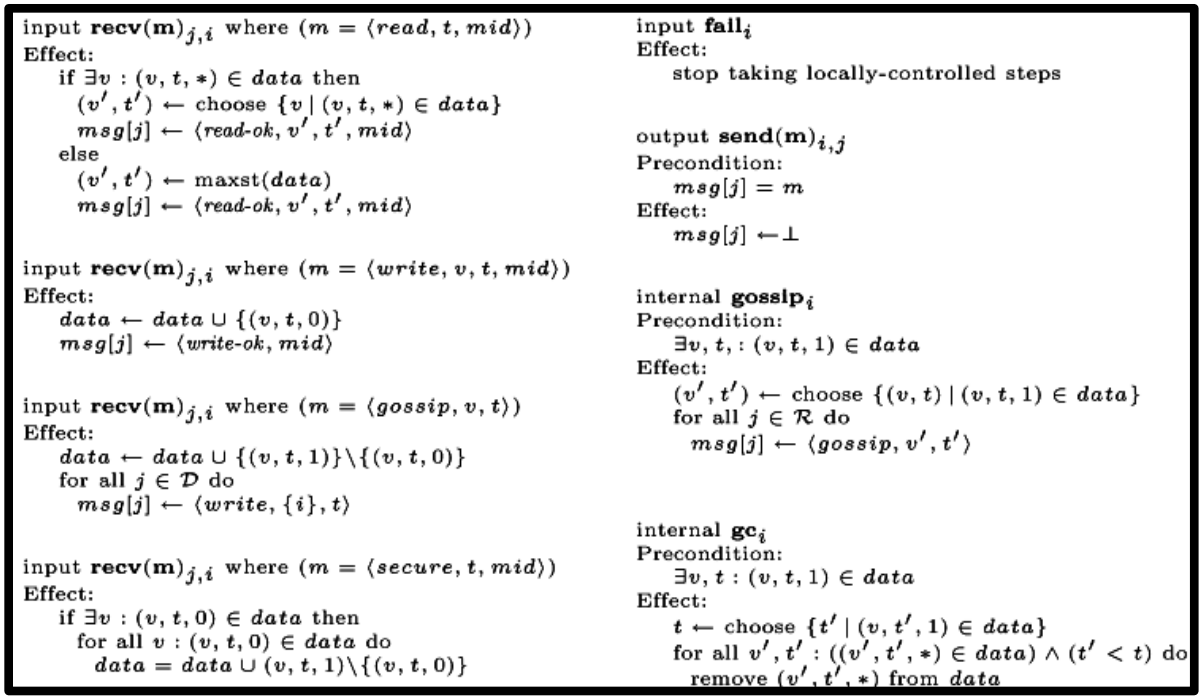
Κατά τη διάρκεια της διάδοσης νέων τιμών (gossip), ουσιαστικά οι αντιγραφείς διαδίδουν secured τιμές του x σε άλλους αντιγραφείς. Ο R_i, θα διαλέξει ένα secured ζεύγος [v,t] και θα στείλει το μήνυμα <*gossip,v,t*> σε όλους τους αντιγραφείς. Αντίθετα, όταν ένας αντιγραφέας παραλάβει ένα μήνυμα gossip προσθέτει στη μεταβλητή data την πλειάδα

[v,t,1] και στέλλει στα ευρετήρια το μήνυμα <*wwrite,{i},t*> για να τα ενημερώσει ότι έχει το ζεύγος [v,t] στα δεδομένα του.

3.5.4 Λειτουργία Περισυλλογής

Κατά τη διάρκεια της συλλογής παλιών τιμών (garbage-collect) ο garbage collector διαγράφει τιμές οι οποίες είναι πλέον παλιές. Όπως είπαμε πιο πάνω όταν ολοκληρωθεί μια εγγραφή και ο πελάτης εισέρχεται στη φάση wok στέλλει ένα μήνυμα <*secure,t,mid*> για να ενημερώσει τους αντιγραφείς. Τότε οι αντιγραφείς εάν έχουν στα δεδομένα τους τιμή με χρονοσφραγίδα t διαγράφουν την πλειάδα [v,t,0], εάν υπάρχει, και προσθέτουν την πλειάδα [v,t,1]. Επομένως, όταν ένας αντιγραφέας παραλάβει ένα secure μήνυμα γνωρίζει ότι δεν θα χρειαστεί ξανά να επιστρέψει τιμή με μικρότερη χρονοσφραγίδα. Για το λόγο αυτό διαγράφει τις τιμές με μικρότερη χρονοσφραγίδα.

Στο Σχήμα 3.3**,** υπάρχουν οι προδιαγραφές των μεταβάσεων του αυτόματου Replica.



Σχήμα 3.3: Σχέσεις Μεταβάσεων του αυτόματου Replica. [15]

3.6 Περιγραφή λειτουργίας ενός Ευρετηρίου:

Η υπογραφή του αυτόματου του Directory αποτελείται από την ενέργεια εισόδου $recv(m)_{i,j}$ και $fail_i$, την ενέργεια εξόδου $send(m)_{i,j}$. Οι μεταβλητές καταστάσεων του αυτόματου είναι

η μεταβλητή *utd*, στην οποία βρίσκονται οι ενημερωμένοι αντιγραφείς, η μεταβλητή *tag*, στην οποία βρίσκεται η ενημερωμένη χρονοσφραγίδα και μια ακολουθία μηνυμάτων *msg* στην οποία αποθηκεύονται προσωρινά τα μηνύματα που θα στείλει το ευρετήριο.

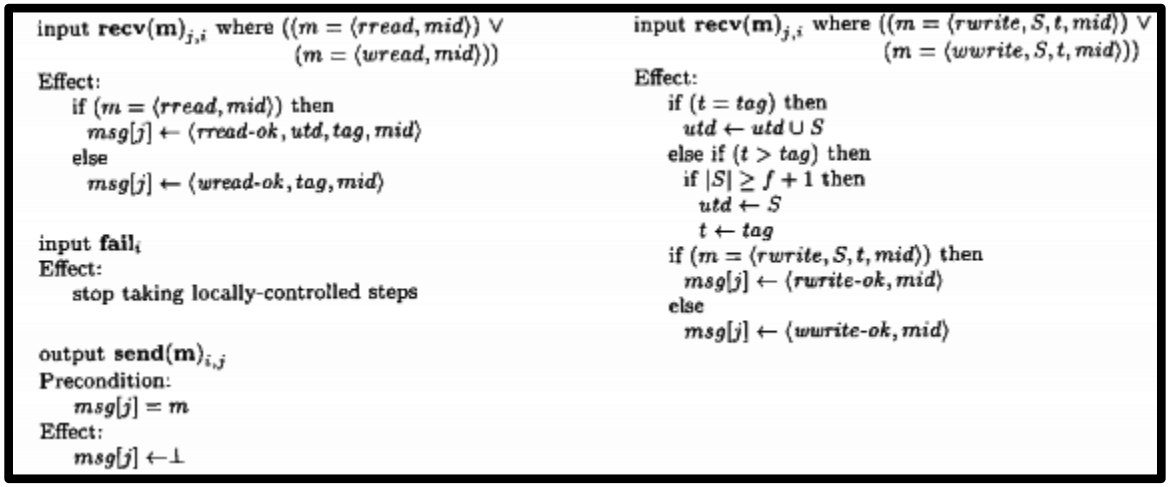
3.6.1 Λειτουργία Ανάγνωσης

Εδώ είναι που αποθηκεύονται ποιοι αντιγραφείς έχουν την τελευταία τιμή του x και τη χρονοσφραγίδα για τη τιμή αυτή. Όταν το ευρετήριο D_i, παραλάβει το μήνυμα <*rread,mid*> ανταποκρίνεται με το μήνυμα <*rread-ok,utd,tag,mid*>. Ενώ, όταν παραλάβει το μήνυμα <*wread,mid*> ανταποκρίνεται με το μήνυμα <*wread-ok,tag,mid*>.

3.6.2 Λειτουργία Εγγραφής

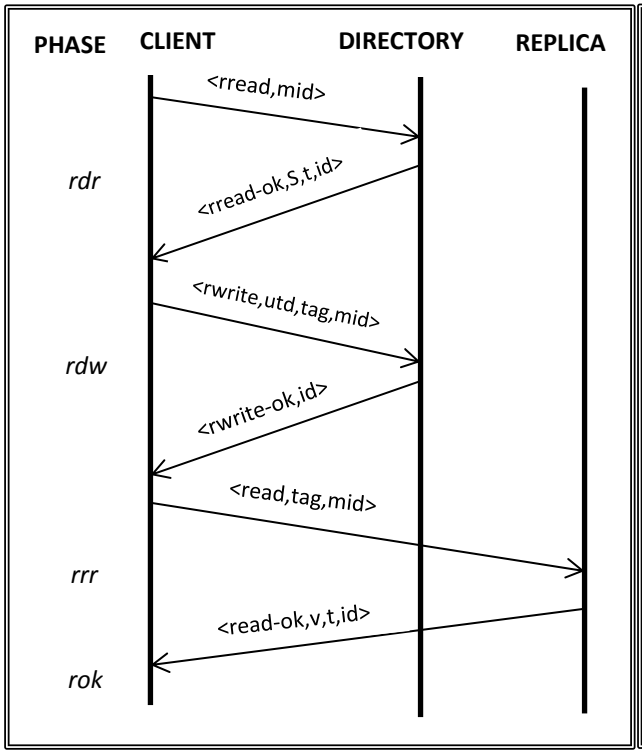
Ακολούθως, όταν παραλάβει το μήνυμα <*rwrite,S,t,mid*>, ελέγχει εάν το t < tag, τότε το write είναι παλιό και δεν εκτελεί καμία λειτουργία. Ενώ, αν το t = tag, τότε το S είναι ένα σύνολο από αντιγραφείς που έχουν την νέα τιμή του x και προσθέτει στο utd το S. Και τέλος, εάν το t > tag, τότε έχουμε μια νέα τιμή x και αν το πλήθος του S είναι μεγαλύτερο από f ανανεώνουμε τη μεταβλητή utd με το S και τη χρονοσφραγίδα με το t. Εάν |S| < f, αγνοεί το μήνυμα. Σε όλες τις περιπτώσεις όμως, το ευρετήριο απαντά στον πελάτη με το μήνυμα <*rwrite-ok,mid*>. Την ίδια διαδικασία ακολουθεί και όταν παραλάβει το μήνυμα <*wwrite,S,t,mid*>, αλλά απαντά στον πελάτη με το μήνυμα <*wwrite-ok,id*>.

Στο Σχήμα 3.4**,** υπάρχουν οι προδιαγραφές των μεταβάσεων του αυτόματου Directory.

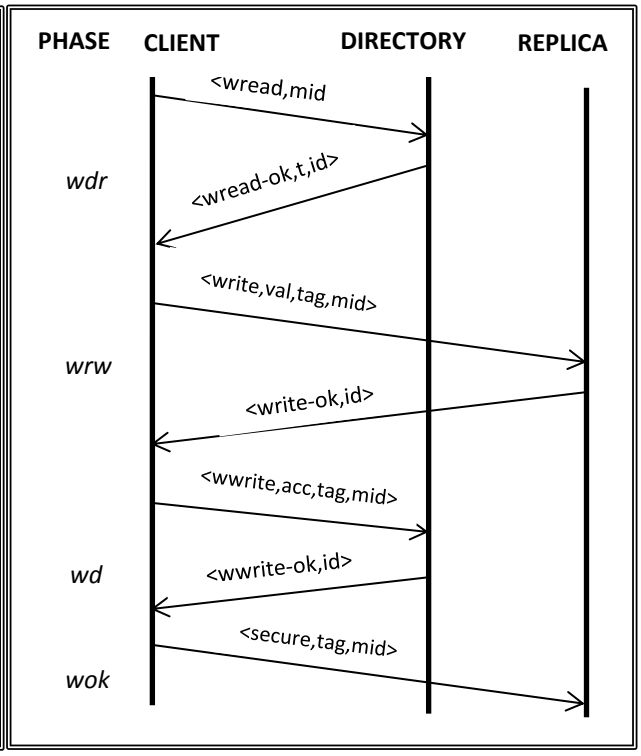


Σχήμα 3.4: Σχέσεις Μεταβάσεων του αυτόματου Directory [15].

Στο Σχήμα 3.5, μπορούμε να δούμε την αλληλεπίδραση που έχει ένας αναγνώστης κατά τη διάρκεια μιας ανάγνωσης και στο Σχήμα 3.6, μπορούμε να δούμε την αλληλεπίδραση που έχει ένας εγγραφέας κατά τη διάρκεια μιας εγγραφής. Τα δύο αυτά σχήματα απεικονίζουν τις λειτουργίες που περιγράψαμε πιο αναλυτικά στα πιο πάνω υποκεφάλαια.



Σχήμα 3.5: Λειτουργία Ανάγνωσης.



Σχήμα 3.6: Λειτουργία Εγγραφής.

Κεφάλαιο 4

Προδιαγραφή Αλγόριθμου Διαχείρισης Αρχείων LDR

Στο κεφάλαιο αυτό θα γίνει εκτενής περιγραφή των προδιαγραφών του αλγόριθμου LDR στο εργαλείο TEMPO χρησιμοποιώντας Χρονισμένα Αυτόματα Εισόδου/Εξόδου και τη γλώσσα TIOA. Πιο συγκεκριμένα θα παραθέσουμε την προδιαγραφή του αυτόματου Λεξιλογίου (Vocabulary.tioa), του αυτόματου Client (Client.tioa), του αυτόματου Directory (Directory.tioa), του αυτόματου Replica (Replica.tioa) και του αυτόματου Σύνθεσης (Composition.tioa). Επίσης, χρειαζόμαστε τα αυτόματα για τα κανάλια επικοινωνίας TCP, δηλαδή τα αυτόματα SendMed (TCPSendMed.tioa), RecvMed (TCPRecvMed.tioa) και ChanMed (TCPChanMed.tioa).

4.1 Λεξιλόγιο

Το λεξιλόγιο του αλγόριθμου φαίνεται στο Σχήμα 4.1 και περιλαμβάνει το λεξιλόγιο που χρησιμοποιούν τα αυτόματα Client, Directory και Replica, καθώς επίσης και το λεξιλόγιο

που χρειάζεται το κανάλι επικοινωνίας TCP. Εμείς ορίσαμε τους εξής νέους τύπους δεδομένων:

- *tag:* είναι μια πλειάδα δυο φυσικών αριθμών [num,id] και χρησιμοποιείται για να ορίσουμε τις χρονοσφραγίδες που χρησιμοποιεί ο αλγόριθμος. Όπου num είναι το πεδίο που μας δείχνει πόσο πρόσφατη είναι μια εγγραφή και το id αντιστοιχεί στον αριθμό της διεργασίας που έκανε τη συγκεκριμένη εγγραφή.
- *message:* είναι μια πλειάδα που αποτελείται από έξι στοιχεία [type,S,val,t,mid,rank], η οποία περιέχει τα δεδομένα που ανταλλάσσονται μεταξύ των διεργασιών. Πιο συγκεκριμένα αποτελείται από τα στοιχεία: *type*, που είναι τύπου String και αντιπροσωπεύει το όνομα της συγκεκριμένης ενέργειας που θα εκτελεστεί και S, που είναι μια ακολουθία τύπου Node στην οποία αποθηκεύονται οι IP διευθύνσεις των ενημερωμένων αντιγραφέων. Ακολούθως, έχουμε το στοιχείο val, το οποίο είναι μια πλειάδα από δυο ακολουθίες χαρακτήρων που αντιπροσωπεύουν τα πραγματικά δεδομένα (actual data) και τα meta data που ανταλλάσουν οι διεργασίες. Στη συνέχεια το στοιχείο t τύπου tag, το οποίο αντιπροσωπεύει τη χρονοσφραγίδα, το στοιχείο mid που είναι τύπου Nat και μας βοηθά να γνωρίζουμε πόσο πρόσφατη είναι μια απάντηση και τέλος το στοιχείο rank που είναι τύπου Nat και αντιπροσωπεύει την ταυτότητα της διεργασίας.
- *ChanMessage:* ο τύπος αυτός είναι μια πλειάδα [query,sender,destination], όπου το στοιχείο query είναι τύπου message, ο sender είναι τύπου Node και αντιπροσωπεύει την IP διεύθυνση του αποστολέα και τέλος το destination είναι τύπου Node και αντιπροσωπεύει την IP διεύθυνση του παραλήπτη.

```
%%% TCPObjectsVoc
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
```

```
%%% Algorithm vocabs
vocabulary algorithm_voc
        imports TCPNodeVoc
        types tag : Tuple[num:Nat, id:Nat]
        types message : Tuple[type:String, S:Seq[Node], val: Tuple[actualData : Seq[Char],
metaData : Seq[Char]], t:tag, mid:Nat, rank : Nat]
end
vocabulary tcp_specific_voc
        imports algorithm_voc, TCPObjectsVoc, TCPNodeVoc
        types
                Chan_message : Tuple[query : message, 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 algorithm_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 4.1: Λεξιλόγιο (Vocabulary.tioa).

4.2 Αυτόματο Directory

Το αυτόματο αυτό ονομάζεται Directory και παίρνει ως παράμετρο ένα φυσικό αριθμό f, είναι ένα άνω όριο για τον αριθμό των αντιγράφων που μπορεί να είναι εσφαλμένοι [`automaton Directory (f:Nat)`]. Στο Σχήμα 4.2 φαίνονται οι ενέργειες που μπορεί να εκτελέσει το αυτόματο αυτό οι οποίες δηλώνονται στο τμήμα signatures.

```
signature
   input RECEIVE(m : Null[Chan_message])
   output SEND(m : Null[Chan_message])
   internal prep_messages
   output systemInitialize(C:Seq[Node], R:Seq[Node], D:Seq[Node], ip:Node,  r:Nat)
```

Σχήμα 4.2: Υπογραφή Αυτόματου Directory.

Στο Σχήμα 4.3 φαίνονται οι μεταβλητές καταστάσεων του αυτόματου οι οποίες ορίζονται στο τμήμα states. Έχουμε τις εξής μεταβλητές:

- *clients:* είναι μια κενή ακολουθία τύπου Node που αντιπροσωπεύει τις IP διευθύνσεις των πελατών του συστήματος μας.
- *directories:* είναι μια κενή ακολουθία τύπου Node που αντιπροσωπεύει τις IP διευθύνσεις των ευρετηρίων του συστήματος μας.
- *replicas:* είναι μια κενή ακολουθία τύπου Node που αντιπροσωπεύει τις IP διευθύνσεις των αντιγραφέων του συστήματος μας.
- *IP:* είναι μια μεταβλητή τύπου Node που αντιπροσωπεύει την IP διεύθυνση της συγκεκριμένης διεργασίας. Αρχικοποιείται με τη διεύθυνση [0,0,0,0].
- *rank:* είναι μια φυσική μεταβλητή που αντιπροσωπεύει την ταυτότητα της συγκεκριμένης διεργασίας. Αρχικοποιείται με 0.
- *msgs:* είναι μια κενή ακολουθία μηνυμάτων Chan_message της κάθε διεργασίας στην οποία εισέρχονται τα μηνύματα που πρόκειται να στείλει και εξέρχονται τα μηνύματα που στέλλονται επιτυχώς.
- *tag:* είναι μια μεταβλητή τύπου tag που αντιπροσωπεύει μια χρονοσφραγίδα. Αρχικοποιείται με [0,0].
- *utd:* είναι μια κενή ακολουθία τύπου Node που αντιπροσωπεύει τις IP διευθύνσεις των ενημερωμένων αντιγραφέων.
- *sent_rread:* είναι μια boolean μεταβλητή που καθορίζει εάν πρέπει να ετοιμαστούν μηνύματα τύπου rread. Αρχικοποιείται με false.
- *sent_rreadOK:* είναι μια boolean μεταβλητή που καθορίζει εάν πρέπει να ετοιμαστούν μηνύματα τύπου rreadOK. Αρχικοποιείται με false.

- *sent_wreadOK:* είναι μια boolean μεταβλητή που καθορίζει εάν πρέπει να ετοιμαστούν μηνύματα τύπου wreadOK. Αρχικοποιείται με false.
- *sent_rwriteOK:* είναι μια boolean μεταβλητή που καθορίζει εάν πρέπει να ετοιμαστούν μηνύματα τύπου rwriteOK. Αρχικοποιείται με false.
- *sent_wwriteOK:* είναι μια boolean μεταβλητή που καθορίζει εάν πρέπει να ετοιμαστούν μηνύματα τύπου wwriteOK. Αρχικοποιείται με false.
- *arrayWrite:* είναι μια κενή ακολουθία από φυσικούς αριθμούς που χρησιμοποιούμε προσωρινά για να αποθηκεύουμε τα rank των διεργασιών που στέλλουν μήνυμα στη συγκεκριμένη διεργασία κατά τη λειτουργία μιας εγγραφής.
- *arrayRead:* είναι μια κενή ακολουθία από φυσικούς αριθμούς που χρησιμοποιούμε προσωρινά για να αποθηκεύουμε τα rank των διεργασιών που στέλλουν μήνυμα στη συγκεκριμένη διεργασία κατά τη λειτουργία μιας ανάγνωσης.
- *clock:* είναι μια μεταβλητή τύπου AugmentedReal που αντιπροσωπεύει το φυσικό ρολόι της διεργασίας. Αρχικοποιείται με 0.
- *midWrite:* είναι μια κενή ακολουθία από φυσικούς αριθμούς που χρησιμοποιούμε προσωρινά για να αποθηκεύουμε τα mid των διεργασιών που στέλλουν μήνυμα στη συγκεκριμένη διεργασία κατά τη λειτουργία μιας εγγραφής.
- *midRead:* είναι μια κενή ακολουθία από φυσικούς αριθμούς που χρησιμοποιούμε προσωρινά για να αποθηκεύουμε τα rank των διεργασιών που στέλλουν μήνυμα στη συγκεκριμένη διεργασία κατά τη λειτουργία μιας ανάγνωσης.

```
states
% indicates an empty sequence of clients IP addresses
clients : Seq[Node] := {};
% indicates an empty sequence of directories IP addresses
directories : Seq[Node] := {};
% indicates an empty sequence of replicas IP addresses
replicas : Seq[Node] := {};
% indicates the IP of this directory
IP : Node := [0,0,0,0];
% indicates the rank of this directory
rank : Nat := 0;
% indicates an empty sequence of chan messages
msgs : Seq[Null[Chan_message]] := {};
% indicates the tag associated with the latest value
tag : tag := [0,0];
% indicates a set of replicas with the latest value
utd : Seq[Node] := {};
% indicates that client can sent message <rread,mid>
sent_rread : Bool := false;
```

```
% indicates that client can sent message <secure,tag,mid>
sent_secure : Bool := false;
% indicates that directory can sent message <rread-ok,S,t,mid>
sent_rreadOK : Bool := false;
% indicates that directory can sent message <wread-ok,t,mid>
sent_wreadOK : Bool := false;
% indicates that directory can sent message <rwrite-ok,mid>
sent_rwriteOK : Bool := false;
% indicates that directory can sent message <wwrite-ok,mid>
sent_wwriteOK : Bool := false;
% indicates a temp seq with ranks for write operations
arrayWrite : Seq[Nat] := {};
% indicates a temp seq with ranks for read operations
arrayRead : Seq[Nat] := {};
% The physcal clock of the automaton
clock : AugmentedReal := 0;
% indicates a temp seq with mids for write operations
midWrite : Seq[Nat] := {};
% indicates a temp seq with mids for read operations
midRead : Seq[Nat] := {};
```

Σχήμα 4.3: Μεταβλητές Καταστάσεων Αυτόματου Directory.

Στο Σχήμα 4.4 φαίνονται οι σχέσεις μεταβάσεων, εκεί γίνεται ο ορισμός των ενεργειών που δηλώσαμε στην υπογραφή του αυτόματου. Πιο αναλυτικά:

- Με την ενέργεια εξόδου *systemInitialize*, το αυτόματο δέχεται πέντε παραμέτρους C, R, D, ip, r με τις οποίες αρχικοποιεί κάποιες από τις μεταβλητές του. Πιο συγκεκριμένα, με τις τρεις πρώτες αρχικοποιούμε τις ακολουθίες clients, replicas και directories αντίστοιχα. Ακολούθως, αρχικοποιούμε τη μεταβλητή IP με τη διεύθυνση της συγκεκριμένης μηχανής (ip) και τη μεταβλητή rank ανάλογα με την κατάταξη της στην ακολουθία (r).
- Με την ενέργεια εισόδου *RECEIVE*, το αυτόματο παραλαμβάνει ένα μήνυμα m. Εάν το μήνυμα δεν είναι κενό και είναι του τύπου *rread*, τότε προσθέτει στην ακολουθία midRead το πεδίο mid του μηνύματος m, στην ακολουθία arrayRead το πεδίο rank του μηνύματος και αλλάζει τη τιμή της Boolean μεταβλητής sent_rreadOK σε true. Εάν είναι τύπου wread, τότε προσθέτει στην ακολουθία midWrite το πεδίο mid του μηνύματος m, στην ακολουθία arrayWrite το πεδίο rank του μηνύματος και αλλάζει τη τιμή της Boolean μεταβλητής sent_wreadOK σε true.

Ακολούθως, εάν ο τύπος του μηνύματος είναι rwrite ή wwrite, τότε ελέγχει εάν το πεδίο tag του μηνύματος m είναι ίσο με τη χρονοσφραγίδα της συγκεκριμένης διεργασίας. Εάν ισχύει, προσθέτει στην ακολουθία utd, τα στοιχεία του συνόλου S που δεν υπάρχουν ήδη στην ακολουθία αυτή. Εάν ισχύει ότι το πεδίο tag του μηνύματος m είναι μεγαλύτερο από τη τοπική χρονοσφραγίδα, τότε ελέγχει εάν το πλήθος του συνόλου S είναι μεγαλύτερο από f+1 και αν ισχύει θέτει τη τοπική μεταβλητή utd ίση με το σύνολο S και ανανεώνει τη τοπική χρονοσφραγίδα με αυτή του μηνύματος m. Στη συνέχεια ανάλογα με τον τύπο του μηνύματος, αν ήταν rwrite ή wwrite, προσθέτονται οι τιμές στις ακολουθίες midRead ή midWrite και arrayRead ή arrayWrite αντίστοιχα. Επιπρόσθετα, εάν ήταν rwrite τότε θέτουμε τη τιμή της μεταβλητής sent_rwriteOK με true, ενώ αν ήταν wwrite θέτουμε τη τιμή της μεταβλητής sent_wwriteOK με true. Η εκτέλεση της ενέργειας αυτής, πυροδοτεί όλες τις αντίστοιχες ενέργειες εξόδου RECEIVE.

- Με την ενέργεια εξόδου *SEND*, το αυτόματο στέλλει ένα μήνυμα m. Οι προυποθέσεις για να εκτελεστεί η ενέργεια αυτή, είναι η ακολουθία msgs να μην είναι κενή και το μήνυμα m να βρίσκεται πρώτο στην ακολουθία. Η εκτέλεση της ενέργειας αυτής, πυροδοτεί όλες τις αντίστοιχες ενέργειες εισόδου SEND.
- Με την εσωτερική ενέργεια *prep_messages*, το αυτόματο ετοιμάζει τα μηνύματα που θα στείλει η συγκεκριμένη διεργασία σε άλλες. Οι προϋποθέσεις για να εκτελεστεί η ενέργεια αυτή είναι μια από τις μεταβλητές sent_rreadOK, sent_wreadOK, sent_rwriteOK και sent_wwriteOK να είναι αληθής. Εάν η μεταβλητή sent_rreadOK είναι αληθής, τότε ετοιμάζουμε μηνύματα του τύπου <["rreadOK", utd, [{},{}], tag, midRead[n], rank], IP, clients[t]> και θέτουμε τη μεταβλητή sent_rreadOK ίση με false. Εάν η μεταβλητής sent_wreadOK είναι αληθής, τότε ετοιμάζουμε μηνύματα του τύπου <["wreadOK", utd, [{},{}], tag, midWrite[n], rank], IP, clients[t]> και θέτουμε τη μεταβλητή sent_wreadOK ίση με false. Ακολούθως, εάν η μεταβλητή sent_rwriteOK είναι αληθής, τότε ετοιμάζουμε μηνύματα του τύπου <["rwriteOK", {}, [{},{}], [0,0], midRead[n], rank], IP, clients[t]> και θέτουμε τη μεταβλητή sent_ rwriteOK ίση με false. Τέλος, εάν η μεταβλητή sent_wwriteOK είναι αληθής, τότε ετοιμάζουμε μηνύματα του τύπου

<["wwriteOK", {}, [{},{}], [0,0], midWrite[n], rank], IP, clients[t]> και θέτουμε τη μεταβλητή sent_wwriteOK ίση με false.

```
transitions
	output systemInitialize(C,R,D,ip,r)
	eff
		clients := C;
		directories := D;
		replicas := R;
		IP := ip;
		utd := replicas;
		rank := r;
	input RECEIVE(m)
	eff
	   if (m ~= nil()) then
		if (val(m).query.type = "rread") then
		   midRead := midRead |- val(m).query.mid;
		   arrayRead := arrayRead |- val(m).query.rank;
		   sent_rreadOK := true;
		else if (val(m).query.type = "wread") then
			midWrite := midWrite |- val(m).query.mid;
			arrayWrite := arrayWrite |- val(m).query.rank;
			sent_wreadOK := true;
		    else if (val(m).query.type="rwrite"\/val(m).query.type="wwrite")
then
				if (val(m).query.t = tag) then
				   for n:Nat where n<len(val(m).query.S) do
					if (val(m).query.S[n] \notin utd) then
						utd := utd |- val(m).query.S[n];
					fi
				   od
				else if (val(m).query.t.num > tag.num \/
(val(m).query.t.num = tag.num /\ val(m).query.t.id >= tag.id)) then
					if (len(val(m).query.S) >= f + 1) then
					   utd := val(m).query.S;
					   tag := val(m).query.t;
					fi
				    fi
				fi
				if (val(m).query.type = "rwrite") then
				   midRead := midRead |- val(m).query.mid;
				   arrayRead := arrayRead |- val(m).query.rank;
				   sent_rwriteOK := true;
				else if (val(m).sender \in clients) then
					midWrite := midWrite |- val(m).query.mid;
					arrayWrite := arrayWrite |- val(m).query.rank;
					sent_wwriteOK := true;
				  fi
				fi
			fi
		fi
	    fi
	fi
	output SEND(m)
	pre
		msgs ~={};
		m = head(msgs);
	eff
		msgs:= tail(msgs);
```

```
        % prepare the messages
        internal prep_messages
        locals
                t : Nat := 0;
        pre
                sent_rreadOK = true \/ sent_wreadOK = true \/ sent_rwriteOK = true \/
sent_wwriteOK = true;
        eff
                if (sent_rreadOK = true) then
                        for n:Nat where n < len(arrayRead) do
                                t := arrayRead[n];
                                msgs := msgs |- embed([["rreadOK", utd, [{},{}], tag,
midRead[n], rank], IP, clients[t]]);
                        od
                        arrayRead := {};
                        midRead := {};
                        sent_rreadOK := false;
                fi
                if (sent_wreadOK = true) then
                        for n:Nat where n < len(arrayWrite) do
                                t := arrayWrite[n];
                                msgs := msgs |- embed([["wreadOK", {}, [{},{}], tag,
midWrite[n],rank], IP, clients[t]]);
                        od
                        arrayWrite := {};
                        midWrite := {};
                        sent_wreadOK := false;
                fi
                if (sent_rwriteOK = true) then
                        for n:Nat where n < len(arrayRead) do
                                t := arrayRead[n];
                                msgs := msgs |- embed([["rwriteOK", {}, [{},{}], [0,0],
midRead[n], rank], IP, clients[t]]);
                        od
                        arrayRead := {};
                        midRead := {};
                        sent_rwriteOK := false;
                fi
                if (sent_wwriteOK = true) then
                        for n:Nat where n < len(arrayWrite) do
                                t := arrayWrite[n];
                                msgs := msgs |- embed([["wwriteOK", {}, [{},{}], [0,0],
midWrite[n], rank], IP, clients[t]]);
                        od
                        arrayWrite := {};
                        midWrite := {};
                        sent_wwriteOK := false;
                fi
```

Σχήμα 4.4: Σχέσεις Μεταβάσεων Αυτόματου Directory.

Εφόσον ορίσαμε και τις σχέσεις μεταβάσεων, το μόνο που μας μένει είναι ο ορισμός της τροχιάς Time, που ορίζεται στο τμήμα trajectories στο Σχήμα 4.5 και εξελίσσει το φυσικό ρολόι clock με ρυθμό 1.

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

Σχήμα 4.5: Τροχιές Αυτόματου Directory.

4.3 Αυτόματο Client

Το αυτόματο αυτό ονομάζεται Client και παίρνει ως παράμετρο ένα φυσικό αριθμό f, ο οποίος είναι ένα άνω όριο για τον αριθμό των αντιγράφων που μπορεί να είναι εσφαλμένοι και ένα DiscreteReal αριθμό t1 που αντιπροσωπεύει το χρόνο που περιμένει ένας client για να πάρει απάντηση από ένα replica όταν του στείλει το μήνυμα read, `[automaton Client (f:Nat, t1:DiscreteReal)]`. Στο Σχήμα 4.6 φαίνονται οι ενέργειες που μπορεί να εκτελέσει το αυτόματο αυτό οι οποίες δηλώνονται στο τμήμα signatures.

```
signature
   input RECEIVE(m : Null[Chan message])
   output SEND(m : Null[Chan message])
   input read, write(v : Seq[Char])
   internal prep_messages, readTimeout
   output read_ok(v : Seq[Char]), write_ok
   output systemInitialize(C:Seq[Node], R:Seq[Node], D:Seq[Node], ip:Node, r:Nat)
```

Σχήμα 4.6: Υπογραφή Αυτόματου Client.

Στο Σχήμα 4.7 φαίνονται οι μεταβλητές καταστάσεων του αυτόματου οι οποίες ορίζονται στο τμήμα states. Κάποιες μεταβλητές είναι κοινές με αυτές του αυτόματου Directory, για αυτό εδώ θα γίνει επεξήγηση μόνο για τις νέες μεταβλητές καταστάσεων:

- *acc:* είναι μια κενή ακολουθία τύπου Node που αντιπροσωπεύει τις IP διευθύνσεις των μηχανών που απάντησαν σε ένα μήνυμα που έστειλε ο client.
- *phase:* είναι μια ακέραια τιμή που αντιπροσωπεύει τη φάση στην οποία βρίσκεται μια λειτουργία ανάγνωσης ή εγγραφής. Αρχικοποιείται με 0 και παίρνει τιμές 1-4 για τις φάσεις μιας ανάγνωσης rdr, rdw, rrr και rok, αντίστοιχα, και 5-8 για τις φάσεις μιας εγγραφής wdr, wrw, wdw και wok, αντίστοιχα.
- *val:* είναι μια κενή πλειάδα από δυο πεδία τύπου Char που αντιπροσωπεύουν τα πραγματικά δεδομένα και τα μεταδεδομένα της ενημερωμένης τιμής.

- *sent_rread:* είναι μια boolean μεταβλητή που καθορίζει εάν πρέπει να ετοιμαστούν μηνύματα τύπου rread. Αρχικοποιείται με false.
- *sent_wread:* είναι μια boolean μεταβλητή που καθορίζει εάν πρέπει να ετοιμαστούν μηνύματα τύπου wread. Αρχικοποιείται με false.
- *sent_rwrite:* είναι μια boolean μεταβλητή που καθορίζει εάν πρέπει να ετοιμαστούν μηνύματα τύπου rwrite. Αρχικοποιείται με false.
- *sent_wwrite:* είναι μια boolean μεταβλητή που καθορίζει εάν πρέπει να ετοιμαστούν μηνύματα τύπου wwrite. Αρχικοποιείται με false.
- *sent_write:* είναι μια boolean μεταβλητή που καθορίζει εάν πρέπει να ετοιμαστούν μηνύματα τύπου write. Αρχικοποιείται με false.
- *sent_read:* είναι μια boolean μεταβλητή που καθορίζει εάν πρέπει να ετοιμαστούν μηνύματα τύπου read. Αρχικοποιείται με false.
- *sent_secure:* είναι μια boolean μεταβλητή που καθορίζει εάν πρέπει να ετοιμαστούν μηνύματα τύπου secure. Αρχικοποιείται με false.
- *senderIP:* είναι μια μεταβλητή τύπου Node που αντιπροσωπεύει την IP διεύθυνση του αποστολέα ενός μηνύματος που παραλαμβάνει η συγκεκριμένη διεργασία. Αρχικοποιείται με [0,0,0,0].
- *maxTag:* είναι μια μεταβλητή τύπου tag, που μας βοηθάει να βρούμε τη μεγαλύτερη χρονοσφραγίδα, ανάμεσα σε ένα αριθμό μηνυμάτων. Αρχικοποιείται με [0,0].
- *readClock:* είναι μια μεταβλητή τύπου AugmentedReal που αντιπροσωπεύει το φυσικό ρολόι της διεργασίας που υπολογίζει το χρόνο από τη στιγμή που στάληκε ένα μήνυμα τύπου read. Αρχικοποιείται με 0.

```
states
       % indicates an empty sequence of clients IP addresses
       clients : Seq[Node] := {};
       % indicates an empty sequence of directories IP addresses
       directories : Seq[Node] := {};
       % indicates an empty sequence of replicas IP addresses
       replicas : Seq[Node] := {};
       % indicates the IP of this client
       IP : Node := [0,0,0,0];
       % indicates the rank of this client
       rank : Nat := 0;
       % indicates an empty sequence of chan messages
       msgs : Seq[Null[Chan_message]] := {};
       % indicates the id of message that client sends with the request
       mid : Nat := 0;
```

```
% indicates the tag associated with the latest value
tag : tag := [0,0] ;
% indicates a set of replicas with the latest value
utd : Seq[Node] := {};
% indicates a set of replicas which replied to client
acc : Seq[Node] := {};
% indicates in which phase the client is
phase : Int := 0;
% indicates the up-to-date value
val: Tuple[actualData : Seq[Char], metaData : Seq[Char]] := [{}, {}];
% indicates that client can sent message <rread,mid>
sent_rread : Bool := false;
% indicates that client can sent message <wread,mid>
sent_wread : Bool := false;
% indicates that client can sent message <rwrite,utd,tag,mid>
sent_rwrite : Bool := false;
% indicates that client can sent message <wwriteC,acc,tag,mid>
sent_wwrite : Bool := false;
% indicates that client can sent message <write,val,tag,mid>
sent_write : Bool := false;
% indicates that client can sent message <read,tag,mid>
sent_read : Bool := false;
% indicates that client can sent message <secure,tag,mid>
sent_secure : Bool := false;
% indicates the IP of the sender
senderIP : Node := [0,0,0,0];
% indicates a temp variable for a tag
maxTag : tag := [0,0];
% The physcal clock of the automaton
clock : AugmentedReal := 0;
% The physical clock for the process
readClock : AugmentedReal := 0;
```

Σχήμα 4.7: Μεταβλητές Καταστάσεων Αυτόματου Client.

Στο Σχήμα 4.8 φαίνονται οι σχέσεις μεταβάσεων, εκεί γίνεται ο ορισμός των ενεργειών που δηλώσαμε στην υπογραφή του αυτόματου. Πιο αναλυτικά:

- Με την ενέργεια εξόδου *systemInitialize*, το αυτόματο δέχεται πέντε παραμέτρους C, R, D, ip, r με τις οποίες αρχικοποιεί κάποιες από τις μεταβλητές του. Πιο συγκεκριμένα, με τις τρεις πρώτες αρχικοποιούμε τις ακολουθίες clients, replicas και directories αντίστοιχα. Ακολούθως, αρχικοποιούμε τη μεταβλητή IP με τη διεύθυνση της συγκεκριμένης μηχανής (ip) και τη μεταβλητή rank ανάλογα με την κατάταξη της στην ακολουθία (r).
- Με την ενέργεια εισόδου *read*, ο client ξεκινάει μια νέα λειτουργία ανάγνωσης. Αυξάνει τη μεταβλητή mid κατά ένα και θέτει τη τιμή της Boolean μεταβλητής sent_rread true.

- Με την ενέργεια εισόδου *write*, ο client παίρνει ως παράμετρο μια τιμή v και ξεκινάει μια νέα λειτουργία εγγραφής. Αποθηκεύει την τιμή v στα πραγματικά δεδομένα της μεταβλητής val, αυξάνει τη μεταβλητή mid κατά ένα και θέτει τη τιμή της Boolean μεταβλητής sent_wread true.
- Με την ενέργεια εξόδου *read_ok*, παίρνει ως παράμετρο μια τιμή v και πρέπει να ελέγξουμε εάν ολοκληρώθηκε η λειτουργία της ανάγνωσης. Οι προϋποθέσεις που πρέπει να ισχύουν είναι η τιμή v να είναι ίση με τα πραγματικά δεδομένα της μεταβλητής val και η μεταβλητή phase να είναι ίση με 4 δηλαδή να είναι στη φάση rok. Εάν ισχύουν, τότε θέτουμε τη τιμή της μεταβλητής phase ίση με 0.
- Με την ενέργεια εξόδου *write_ok*, ελέγχουμε εάν ολοκληρώθηκε η λειτουργία της εγγραφής. Η προϋπόθεση για να εκτελεστεί η ενέργεια είναι το αυτόματο να βρίσκεται στη phase 8, δηλαδή στη φάση wok. Κατά την εκτέλεση της θέτει τη τιμή της μεταβλητής phase ίση με 0.
- Με την ενέργεια εξόδου *SEND*, το αυτόματο στέλλει ένα μήνυμα m. Οι προϋποθέσεις για να εκτελεστεί η ενέργεια αυτή, είναι η ακολουθία msgs να μην είναι κενή και το μήνυμα m να βρίσκεται πρώτο στην ακολουθία. Η εκτέλεση της ενέργειας αυτής, πυροδοτεί όλες τις αντίστοιχες ενέργειες εισόδου SEND και εάν ο τύπος του μηνύματος που θα σταλεί είναι rread, τότε αρχικοποιεί το ρολόι readClock με την τιμή του clock.
- Με την εσωτερική ενέργεια *readTimeout,* ελέγχουμε εάν πέρασε χρόνος t_1 από τη χρονική στιγμή που έστειλε το μήνυμα read για να δει αν χρειάζεται να το ξαναστείλει. Οι προϋποθέσεις για να εκτελεστεί η ενέργεια αυτή είναι το ρολόι readClock να είναι άνισο του 0, που σημαίνει ότι έστειλε το μήνυμα read και περιμένει απάντηση και η τιμή readClock+t_1 να είναι μεγαλύτερη από την τιμή του clock, που σημαίνει ότι πέρασε ο χρόνος t_1 από τη χρονική στιγμή που έστειλε το μήνυμα read. Εάν ισχύουν, τότε θα διαλέξει ένα replica από το σύνολο utd και θα ετοιμάσει ένα μήνυμα τύπου read.
- Με την ενέργεια εισόδου *RECEIVE*, το αυτόματο παραλαμβάνει ένα μήνυμα m. Εάν το μήνυμα δεν είναι κενό θέτουμε τη μεταβλητής mid, ίση με το πεδίο mid του μηνύματος m και τη μεταβλητή senderIP, ίση με το πεδίο sender του μηνύματος m.

Εάν το μήνυμα είναι τύπου rreadOK και η μεταβλητή phase είναι ίση με 1, γίνεται έλεγχος εάν ο αποστολέας ανήκει στο σύνολο acc. Αν δεν ανήκει τότε τον προσθέτουμε. Εάν η χρονοσφραγίδα του μηνύματος είναι μεγαλύτερη από την τοπική, τότε ανανεώνουμε την τοπική χρονοσφραγίδα με αυτή του μηνύματος m και θέτουμε το σύνολο utd ίσο με το σύνολο S του μηνύματος. Μετά γίνεται έλεγχος εάν το πλήθος του συνόλου acc είναι μεγαλύτερο από την πλειοψηφία των ευρετηρίων (majority = directories/2 + 1) και αν ισχύει αυξάνουμε την τιμή του mid κατά ένα και θέτουμε τη τιμή του sent_rwrite ίση με true. Παρόμοια διαδικασία εκτελείται και όταν το μήνυμα είναι του τύπου rwriteOK, με τη διαφορά ότι εδώ πρέπει η μεταβλητή phase να είναι ίση με 2 και εάν έχουμε απάντηση από πλειοψηφία των ευρετηρίων θέτουμε τη τιμή του sent_read ίση με true. Εάν το μήνυμα είναι τύπου rreadOK και η μεταβλητή phase είναι ίση με 3, τότε αποθηκεύουμε τη τιμή που μας επέστρεψε ο replica στη μεταβλητή val και ανανεώνουμε και τη χρονοσφραγίδα. Επίσης, θέτουμε τη τιμή της μεταβλητής phase ίση με 4 και το ρολόι readClock ίσο με 0. Ακολούθως, εάν το μήνυμα είναι τύπου wreadOK και η μεταβλητή phase είναι ίση με 1, γίνεται έλεγχος εάν ο αποστολέας ανήκει στο σύνολο acc. Αν δεν ανήκει τότε τον προσθέτουμε. Κάνουμε κάποιους ελέγχους εφόσον θέλουμε να βρούμε τη μεγαλύτερη χρονοσφραγίδα στις απαντήσεις που θα πάρει και αν το πλήθος του συνόλου acc είναι μεγαλύτερο από την πλεοψηφεία των ευρετηρίων, αυξάνουμε την τιμή του mid κατά ένα και θέτουμε τη τιμή του sent_write ίση με true. Παρόμοια διαδικασία εκτελείται και όταν το μήνυμα είναι του τύπου wwriteOK, με τη διαφορά ότι εδώ πρέπει η μεταβλητή phase να είναι ίση με 7 και εάν έχουμε απάντηση από πλειοψηφία των ευρετηρίων θέτουμε τη τιμή του sent_secure ίση με true. Τέλος, εάν το μήνυμα είναι τύπου writeOK και η μεταβλητή phase είναι ίση με 6, γίνεται έλεγχος εάν ο αποστολέας ανήκει στο σύνολο acc. Αν δεν ανήκει τότε τον προσθέτουμε. Μετά γίνεται έλεγχος εάν το πλήθος του συνόλου acc είναι μεγαλύτερο από f, τότε αυξάνουμε την τιμή του mid κατά ένα και θέτουμε τη τιμή του sent_wwrite ίση με true.

- Με την εσωτερική ενέργεια *prep_messages*, το αυτόματο ετοιμάζει τα μηνύματα που θα στείλει η συγκεκριμένη διεργασία σε άλλες. Οι προϋποθέσεις για να εκτελεστεί η ενέργεια αυτή είναι μια από τις μεταβλητές sent_rread, sent_wread,

sent_rwrite, sent_wwrite, sent_write, sent_read και sent_secure να είναι αληθής. Εάν η μεταβλητή sent_rread είναι αληθής, τότε ετοιμάζουμε μηνύματα του τύπου <["rread", {}, [{},{}], [0,0], mid, rank], IP, directories[n]> με παραλήπτες το σύνολο των ευρετηρίων, θέτουμε τη μεταβλητή sent_rread ίση με false και τη μεταβλητή phase ίση με 1 (rdr). Εάν η μεταβλητής sent_wread είναι αληθής, τότε ετοιμάζουμε μηνύματα του τύπου <["wread", {}, [{},{}], [0,0], mid, rank], IP, directories[n]> με παραλήπτες το σύνολο των ευρετηρίων, θέτουμε τη μεταβλητή sent_wread ίση με false και τη μεταβλητή phase ίση με 5 (wdr). Ακολούθως, εάν η μεταβλητή sent_rwrite είναι αληθής, τότε ετοιμάζουμε μηνύματα του τύπου <["rwrite", utd, [{},{}], tag, mid, rank], IP, directories[n]> με παραλήπτες το σύνολο των ευρετηρίων, θέτουμε τη μεταβλητή sent_rwrite ίση με false και τη μεταβλητή phase ίση με 2 (rdw). Εάν η μεταβλητή sent_wwrite είναι αληθής, τότε ετοιμάζουμε μηνύματα του τύπου <["wwrite", acc, [{},{}], tag, mid, rank], IP, directories[n]> με παραλήπτες το σύνολο των ευρετηρίων, θέτουμε τη μεταβλητή sent_wwrite ίση με false και τη μεταβλητή phase ίση με 7 (wdw).. Στη συνέχεια, εάν η μεταβλητή sent_write είναι αληθής, τότε ετοιμάζουμε μηνύματα του τύπου <["write", {}, [{},{}], tag, mid, rank], IP, replicas[n]> με παραλήπτες το σύνολο των αντιγραφέων, θέτουμε τη μεταβλητή sent_write ίση με false και τη μεταβλητή phase ίση με 6 (wrw). Εάν η μεταβλητή sent_read είναι αληθής, τότε διαλέγουμε τυχαία ένα replica από το σύνολο utd, ετοιμάζουμε ένα μήνυμα του τύπου <["read", {}, [{},{}], tag, mid, rank], IP, utd[tempRank]>, θέτουμε τη μεταβλητή sent_read ίση με false και τη μεταβλητή phase ίση με 3 (rrr). Τέλος, εάν η μεταβλητή sent_secure είναι αληθής, τότε ετοιμάζουμε ένα μήνυμα του τύπου <["secure", {}, [{},{}], tag, mid, rank], IP, replicas[n]> με παραλήπτες το σύνολο των αντιγραφέων, θέτουμε τη μεταβλητή sent_secure ίση με false και τη μεταβλητή phase ίση με 8 (wok).

```
transitions
        output systemInitialize(C,R,D,ip,r)
        eff
                clients := C;
                directories := D;
                replicas := R;
                IP := ip;
                rank := r;
        input read
        eff
                mid := mid + 1;
                sent_rread := true;
```

```
        input write(v)
        eff
                val.actualData := v;
                mid := mid + 1;
                sent_wread := true;
        output read_ok(v)
        pre
                v = val.actualData /\ phase = 4;
        eff
                phase := 0
        output write_ok
        pre
                phase = 8;
        eff
                phase := 0;
        output SEND(m)
        pre
                msgs ~={};
                m = head(msgs);
        eff
                msgs:= tail(msgs);
                if (val(m).query.type = "read") then
                        readClock := clock;
                fi
        internal readTimeout
        locals
                tempRank : Nat := 0;
        pre
                 readClock ~= 0 /\ clock > readClock + t1;
        eff
                if (len(utd) > 0) then
                   tempRank := choose t where (t>=len(utd) /\ t<=len(utd));
                   msgs := msgs |- embed([["read", {}, [{}, {}], tag, mid, rank],IP,
utd[tempRank]]);
                fi
                readClock := 0;
        input RECEIVE(m)
        eff
                if (m ~= nil()) then
                        mid := val(m).query.mid;
                        senderIP := val(m).sender;
                        if (val(m).query.type = "rreadOK") then
                                if ((phase = 1) /\ (mid = val(m).query.mid)) then
                                        if (val(m).sender \notin acc) then
                                                acc := acc |- val(m).sender;
                                        fi
                                        if (val(m).query.t.num > tag.num \/
(val(m).query.t.num = tag.num /\ val(m).query.t.id >= tag.id)) then
                                                tag := val(m).query.t;
                                                utd := val(m).query.S;
                                        fi
                                        if (len(acc) >= (div(len(directories),2) + 1)) then
                                                mid := mid + 1;
                                                sent_rwrite := true;
                                        fi
                                fi
                        fi
```

```
                        if (val(m).query.type = "rwriteOK") then
                                if ((phase = 2) /\ (mid = val(m).query.mid)) then
                                        if (val(m).sender \notin acc) then
                                                acc := acc |- val(m).sender;
                                        fi
                                        if (len(acc) >= (div(len(directories),2) + 1)) then
                                                mid := mid + 1;
                                                sent_read := true;
                                        fi
                                fi
                        fi
                        if (val(m).query.type = "readOK") then
                                if ((phase = 3) /\ (mid = val(m).query.mid)) then
                                        val := val(m).query.val;
                                        tag := val(m).query.t;
                                        phase := 4;
                                        readClock := 0;
                                fi
                        fi
                        if (val(m).query.type = "wreadOK") then
                                if ((phase = 5) /\ (mid = val(m).query.mid)) then
                                        if (val(m).sender \notin acc) then
                                                acc := acc |- val(m).sender;
                                        fi
                                        maxTag := tag;
                                        if (val(m).query.t.num > maxTag.num \/
(val(m).query.t.num = maxTag.num /\ val(m).query.t.id >= tag.id)) then
                                                maxTag := val(m).query.t;

                                        fi
                                        if (len(acc) >= (div(len(directories),2) + 1)) then
                                                mid := mid + 1;
                                                sent_write := true;
                                        fi
                                fi
                        fi
                        if (val(m).query.type = "writeOK") then
                                if ((phase = 6) /\ (mid = val(m).query.mid)) then
                                        if (val(m).sender \notin acc) then
                                                acc := acc |- val(m).sender;
                                        fi
                                        if (len(acc) > f) then
                                                mid := mid + 1;
                                                sent_wwrite := true;
                                        fi
                                fi
                        fi
                        if (val(m).query.type = "wwriteOK") then
                                if ((phase = 7) /\ (mid = val(m).query.mid)) then
                                        if (val(m).sender \notin acc) then
                                                acc := acc |- val(m).sender;
                                        fi
                                        if (len(acc) >= (div(len(directories),2) + 1)) then
                                                mid := mid +1;
                                                sent_secure := true;
                                        fi
                                fi
                        fi
                fi
```

```
        internal prep_messages
        locals
                tempRank : Nat := 0;
        pre
                sent_rread = true \/ sent_wread = true  \/ sent_rwrite = true \/
sent_wwrite = true \/ sent_write = true \/ sent_read = true \/ sent_secure = true;
        eff
                if (sent_rread = true) then
                        for n:Nat where(n < len(directories)) do
                           msgs := msgs |- embed([["rread", {}, [{}, {}], [0,0], mid,
rank],IP, directories[n]]);
                        od
                        sent_rread := false;
                        phase := 1;
                fi
                if (sent_wread = true) then
                        for n:Nat where(n < len(directories)) do
                                msgs := msgs |- embed([["wread", {}, [{}, {}], [0,0],
mid,rank], IP, directories[n]]);
                        od
                        sent_wread := false;
                        phase := 5;
                fi
                if (sent_rwrite = true) then
                        for n:Nat where(n < len(directories)) do
                                msgs := msgs |- embed([["rwrite", utd, [{}, {}], tag, mid,
rank],IP, directories[n]]);
                        od
                        acc := {};
                        sent_rwrite := false;
                        phase := 2;
                fi
                if (sent_wwrite = true) then
                        for n:Nat where(n < len(directories)) do
                                msgs := msgs |- embed([["wwrite", acc, [{}, {}], tag, mid,
rank],IP, directories[n]]);
                        od
                        acc := {};
                        sent_wwrite := false;
                        phase := 7;
                fi
                if (sent_write = true) then
                        if (len(acc) >= (div(len(directories),2) + 1)) then
                                tag.num := maxTag.num + 1;
                                tag.id := rank;
                        fi
                        for n:Nat where(n < len(replicas)) do
                                msgs := msgs |- embed([["write", {}, val, tag, mid,
rank],IP, replicas[n]]);
                        od
                        acc := {};
                        maxTag := [0,0];
                        sent_write := false;
                        phase := 6;
                fi
```

```
if (sent_read = true) then
                        if (len(utd) > 0) then
                                tempRank := choose t where (t>=len(utd) /\ t<=len(utd));
                                msgs := msgs |- embed([["read", {}, [{}, {}], tag, mid,
rank],IP, utd[tempRank]]);
                                acc := {};
                                sent_read := false;
                                phase := 3;
                        fi
                fi
                if (sent_secure = true) then
                        for n:Nat where(n < len(replicas)) do
                                msgs := msgs |- embed([["secure", {}, [{}, {}], tag, mid,
rank],IP, replicas[n]]);
                        od
                        acc := {};
                        sent_secure:= false;
                        phase := 8;
                fi
```

Σχήμα 4.8: Σχέσεις Μεταβάσεων Αυτόματου Client.

Εφόσον ορίσαμε και τις σχέσεις μεταβάσεων, το μόνο που μας μένει είναι ο ορισμός της τροχιάς Time, που ορίζεται στο τμήμα trajectories στο Σχήμα 4.9 και εξελίσσει το φυσικό ρολόι clock με ρυθμό 1.

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

Σχήμα 4.9: Τροχιές Αυτόματου Client.

4.4 Αυτόματο Replica

Το αυτόματο αυτό ονομάζεται Replica και παίρνει ως παράμετρο ένα φυσικό αριθμό f, ο οποίος είναι ένα άνω όριο για τον αριθμό των αντιγράφων που μπορεί να είναι εσφαλμένοι [automaton Replica (f:Nat)]. Στο Σχήμα 4.10 φαίνονται οι ενέργειες που μπορεί να εκτελέσει το αυτόματο αυτό οι οποίες δηλώνονται στο τμήμα signatures.

```
signature
   input RECEIVE(m : Null[Chan message])
   output SEND(m : Null[Chan message])
   internal prep_messages
   output systemInitialize(C:Seq[Node], R:Seq[Node], D:Seq[Node], ip:Node, r :Nat)
   internal gossip, gc
```

Σχήμα 4.10: Υπογραφή Αυτόματου Replica.

Στο Σχήμα 4.11 φαίνονται οι μεταβλητές καταστάσεων του αυτόματου οι οποίες ορίζονται στο τμήμα states. Κάποιες μεταβλητές είναι κοινές με αυτές των αυτόματων Directory και Client, για αυτό εδώ θα γίνει επεξήγηση μόνο για τις νέες μεταβλητές καταστάσεων:

- *data:* είναι μια κενή ακολουθία που αποτελείται από μια πλειάδα τριών στοιχείων και αντιπροσωπεύει τα δεδομένα που αποθηκεύει ένας replica. Το πρώτο στοιχείο είναι μια πλειάδα από δυο ακολουθίες που αντιπροσωπεύουν τα πραγματικά δεδομένα και τα μεταδεδομένα, το δεύτερο στοιχείο είναι μια μεταβλητή τύπου tag που αντιπροσωπεύει τη χρονοσφραγίδα και το τρίτο στοιχείο είναι ένας φυσικός αριθμός που αντιπροσωπεύει το security bit.
- *sent_writeOK*: είναι μια boolean μεταβλητή που καθορίζει εάν πρέπει να ετοιμαστούν μηνύματα τύπου writeOK. Αρχικοποιείται με false.
- *sent_readOK:* είναι μια boolean μεταβλητή που καθορίζει εάν πρέπει να ετοιμαστούν μηνύματα τύπου readOK. Αρχικοποιείται με false.
- *sent_gossip:* είναι μια boolean μεταβλητή που καθορίζει εάν πρέπει να ετοιμαστούν μηνύματα τύπου gossip. Αρχικοποιείται με false.
- *sent_wwrite:* είναι μια boolean μεταβλητή που καθορίζει εάν πρέπει να ετοιμαστούν μηνύματα τύπου wwrite. Αρχικοποιείται με false.
- *start_gossip:* είναι μια boolean μεταβλητή που καθορίζει πότε πρέπει να ξεκινήσει η διαδικασία διάδοσης μιας νέας τιμής. Αρχικοποιείται με false.
- *tempData:* είναι μια μεταβλητή με τον ίδιο τύπο δεδομένων που περιέχεται στην ακολουθία data και τη χρησιμοποιούμε για να αποθηκεύσουμε προσωρινά μια τιμή.
- *tempDataRead:* είναι μια μεταβλητή με τον ίδιο τύπο δεδομένων που περιέχεται στην ακολουθία data και τη χρησιμοποιούμε για να αποθηκεύσουμε προσωρινά μια τιμή κατά τη λειτουργία της ανάγνωσης.
- *tempTag:* είναι μια μεταβλητή τύπου tag, που μας βοηθάει να αποθηκεύσουμε προσωρινά μια χρονοσφραγίδα. Αρχικοποιείται με [0,0].

```
states
        % indicates an empty sequence of clients IP addresses
        clients : Seq[Node] := {};
        % indicates an empty sequence of directories IP addresses
        directories : Seq[Node] := {};
        % indicates an empty sequence of replicas IP addresses
        replicas : Seq[Node] := {};
        % indicates the IP of this replica
        IP : Node := [0,0,0,0];
        % indicates the rank of this replica
        rank : Nat := 0;
        % an empty sequence of chan messages
        msgs : Seq[Null[Chan_message]] := {};
        % indicates the set of tuple[value-tag-bit] that stores a replica
        data : Seq[Tuple[value: Tuple[actualData : Seq[Char], metaData : Seq[Char]],
tag : tag, bit : Nat]] := {};
        % indicates that replica can sent message <write-ok,mid>
        sent_writeOK : Bool := false;
        % indicates that replica can sent message type=12 <read-ok,v,t,mid>
        sent_readOK : Bool := false;
        % indicates that replica can sent message type=13 <gossip,v,t>
        sent_gossip : Bool := false;
        % indicates that replica can sent message type=14 <wwriteR,{i},t>
        sent_wwrite : Bool := false;
        % indicates temp data
        tempData : Tuple[value: Tuple[actualData : Seq[Char], metaData : Seq[Char]],
tag : tag] := [[{},{}], [0,0]];
        tempDataRead : Tuple[value: Tuple[actualData : Seq[Char], metaData :
Seq[Char]], tag : tag] := [[{},{}],[0,0]];
        % indicates temp data
        tempTag : tag := [0,0];
        % indicates a temp table for write operations
        arrayWrite : Seq[Nat] := {};
        % indicates a temp table for read operations
        arrayRead : Seq[Nat] := {};
        % The physcal clock of the automaton
        clock : AugmentedReal := 0;   % indicates when to start gossip operation
        start_gossip : Bool := false;
        % indicates a temp seq with mids for write operations
        midWrite : Seq[Nat] := {};
        % indicates a temp seq with mids for read operations
        midRead : Seq[Nat] := {};
```

Σχήμα 4.11: Μεταβλητές Καταστάσεων Αυτόματου Replica.

Στο Σχήμα 4.12 φαίνονται οι σχέσεις μεταβάσεων, εκεί γίνεται ο ορισμός των ενεργειών που δηλώσαμε στην υπογραφή του αυτόματου. Πιο αναλυτικά:

- Με την ενέργεια εξόδου *systemInitialize*, το αυτόματο δέχεται πέντε παραμέτρους C, R, D, ip, r με τις οποίες αρχικοποιεί κάποιες από τις μεταβλητές του. Πιο συγκεκριμένα, με τις τρεις πρώτες αρχικοποιούμε τις ακολουθίες clients, replicas

και directories αντίστοιχα. Ακολούθως, αρχικοποιούμε τη μεταβλητή IP με τη διεύθυνση της συγκεκριμένης μηχανής (ip) και τη μεταβλητή rank ανάλογα με την κατάταξη της στην ακολουθία (r). Επίσης, αρχικοποιούμε τη δομή data με τη τιμή [[0],{}],[0,0],1].

- Με την ενέργεια εξόδου *SEND*, το αυτόματο στέλλει ένα μήνυμα m. Οι προϋποθέσεις για να εκτελεστεί η ενέργεια αυτή, είναι η ακολουθία msgs να μην είναι κενή και το μήνυμα m να βρίσκεται πρώτο στην ακολουθία. Η εκτέλεση της ενέργειας αυτής, πυροδοτεί όλες τις αντίστοιχες ενέργειες εισόδου SEND.
- Με την ενέργεια εισόδου *RECEIVE*, το αυτόματο παραλαμβάνει ένα μήνυμα m. Εάν το μήνυμα m είναι τύπου read, τότε προσθέτει στην ακολουθία midRead το πεδίο mid του μηνύματος m, στην ακολουθία arrayRead το πεδίο rank του μηνύματος και ψάχνει να βρει τη χρονοσφραγίδα του μηνύματος μέσα στα δεδομένα (data) της. Εάν το βρει, τότε θα στείλει την τιμή αυτή στον client, διαφορετικά θα βρει τη τιμή με τη μεγαλύτερη χρονοσφραγίδα και θα τη στείλει αυτή. Τέλος, θέτει τη τιμή της μεταβλητής sent_readOK true. Ακολούθως, εάν το μήνυμα m είναι τύπου write, γίνονται κάποιοι έλεγχοι για να δούμε αν η συγκεκριμένη τιμή του μηνύματος πρέπει να γραφτεί στο data και αν πρέπει θέτει τη τιμή της μεταβλητής sent_writeOK ίση με true. Στη συνέχεια, εάν το μήνυμα m είναι τύπου gossip, τότε ελέγχει εάν η τιμή και η χρονοσφραγίδα του μηνύματος υπάρχουν ήδη στο data. Εάν υπάρχουν, τότε διαγράφει τη συγκεκριμένη καταχώρηση. Είτε υπάρχουν, είτε δεν υπάρχουν προσθέτει τη τιμή και τη χρονοσφραγίδα του μηνύματος στα δεδομένα του με security bit ίσο με 1. Τέλος, εάν παραλάβει μήνυμα τύπου secure, τότε ελέγχει εάν έχει στο data του τη χρονοσφραγίδα του μηνύματος m. Αν την έχει αλλάζει το security bit από 0 σε 1.
- Με την εσωτερική ενέργεια *prep_messages*, το αυτόματο ετοιμάζει τα μηνύματα που θα στείλει η συγκεκριμένη διεργασία σε άλλες. Η προϋπόθεση για να εκτελεστεί η ενέργεια αυτή είναι μια από τις μεταβλητές sent_writeOK, sent_readOK, sent_wwrite και sent_gossip να είναι αληθής. Εάν η μεταβλητή sent_writeOK είναι αληθής, τότε ετοιμάζουμε μηνύματα του τύπου <["writeOK", {}, [{},{}], [0,0], midWrite[n], rank], IP, clients[t]>, θέτουμε τη μεταβλητή

sent_writeOK ίση με false και τη μεταβλητή start_gossip ίση με true. Εάν η μεταβλητής sent_readOK είναι αληθής, τότε ετοιμάζουμε μηνύματα του τύπου <["readOK", {}, tempDataRead.value, tempDataRead.tag, midRead[n], rank], IP, clients[t]> και θέτουμε τη μεταβλητή sent_readOK ίση με false. Ακολούθως, εάν η μεταβλητής sent_wwrite είναι αληθής, τότε ετοιμάζουμε μηνύματα του τύπου <["wwrite", tempSeq, [{},{}], tempTag, 0, rank], IP, directories[n]> με παραλήπτες το σύνολο των ευρετηρίων και θέτουμε τη μεταβλητή sent_wwrite ίση με false. Τέλος, εάν η μεταβλητής sent_gossip είναι αληθής, τότε ετοιμάζουμε μηνύματα του τύπου <["gossip", {}, tempData.value, tempData.tag, 0, rank], IP, replicas[n]> με παραλήπτες το σύνολο των replica και θέτουμε τη μεταβλητή sent_gossip ίση με false.

- Με την εσωτερική ενέργεια *gossip* γίνεται η διάδοση νέων τιμών. Οι προϋποθέσεις για να εκτελεστεί η ενέργεια αυτή είναι η ακολουθία data να μην είναι κενή και η μεταβλητή start_gossip να είναι true. Τυχαία διαλέγουμε μια καταχώρηση του data που να είναι secured και τότε θέτουμε τη τιμή της μεταβλητής sent_gossip ίση με true.
- Με την εσωτερική ενέργεια *gc* γίνεται η περισυλλογή παλαιότερων τιμών. Η προυπόθεση για να εκτελεστεί η ενέργεια αυτή είναι η ακολουθία data να μην είναι κενή. Τυχαία διαλέγουμε μια καταχώρηση του data που είναι secured και διαγράφουμε όλες τις καταχωρήσεις που έχουν μικρότερη χρονοσφραγίδα από αυτή.

```
transitions
        output systemInitialize(C,R,D,ip,r)
        locals
                temp : Seq[Char] := {};
        eff
                clients := C;
                directories := D;
                replicas := R;
                temp := temp |- '0';
                data := data |- [[temp,{}],[0,0],1];
                IP := ip;
                rank := r;
        % send messages
        output SEND(m)
        pre
                msgs ~={};
                m = head(msgs);
        eff
                msgs:= tail(msgs);
```

```
input RECEIVE(m)
        locals
                f : Bool := false;
                go : Int := 0;
                s1 : Bool := false;
                s2 : Bool := false;
                s3 : Bool := false;
                maxTag : tag := [0,0];
                maxValue : Seq[Char] := {};
                position : Nat := 0;
                found : Array[Int,Bool] := constant(false:Bool);
                temp : Seq[Tuple[value : Tuple[actualData : Seq[Char], metaData :
Seq[Char]], tag : tag, bit : Nat]] := {};
        eff
                if (m ~= nil()) then
                   if (val(m).query.type = "read") then
                        arrayRead := arrayRead |- val(m).query.rank;
                        midRead := midRead |- val(m).query.mid;
                        for b:Nat where b < len(data) do
                           if (data[b].tag.num = val(m).query.t.num /\ data[b].tag.id =
val(m).query.t.id /\ f = false) then
                             tempDataRead := [[data[b].value.actualData,{}], data[b].tag];
                             f := true;
                           fi
                        od
                        if (f = false) then
                            for k:Nat where k < len(data) do
                                    if (data[k].tag.num > maxTag.num /\ data[k].tag.id
> maxTag.id) then
                                            maxTag := data[k].tag;
                                            maxValue := data[k].value.actualData;
                                    fi
                            od
                            tempDataRead := [[maxValue,{}], maxTag];
                        else
                                f := false;
                           fi
                           sent_readOK := true;
                        f
                    if (val(m).query.type = "gossip") then
                        for t:Nat where t<len(data) do
                          if (data[t].value.actualData= val(m).query.val.actualData /\
data[t].tag = val(m).query.t) then
                                            position := t;
                                            if (data[t].bit = 0) then
                                                    go := 1;
                                            else
                                                    go := 2;
                                            fi
                                    fi
                            od
                            if (go = 1) then
                                    data := eject(data,position);
                            fi
                            if (go ~= 2) then
                               data := [val(m).query.val, val(m).query.t, 1] -| data;
                               tempTag := data[0].tag;
                               go := 0;
                               position := 0;
                               sent_wwrite := true;
                            fi
                        fi
```

```
        if (val(m).query.type = "write") then
                                for k:Nat where k<len(data) do
                                        if (data[k].tag.num = val(m).query.t.num /\
data[k].tag.id < val(m).query.t.id /\ data[k].bit = 0) then
                                                found[k] := true;
                                                s1 := true;
                                        else if (data[k].tag.num = val(m).query.t.num /\
data[k].tag.id > val(m).query.t.id) then
                                                        if (data[k].bit = 0) then
                                                                s3 := true;
                                                        else
                                                                s2 := false;
                                                        fi
                                                else if (data[k].tag.num <
val(m).query.t.num ) then
                                                                s2 := true;
                                                        fi
                                                fi
                                        fi
                                od
                                if (s1 = true) then
                                        for k:Nat where k<len(data) do
                                                if (found[k] = false) then
                                                        temp := temp |- data[k];
                                                fi
                                        od
                                        data := {};

                                        for k:Nat where k<len(temp) do
                                                data := data |- temp[k];
                                        od
                                fi
                                if (s1 = true \/ s2 = true /\ s3 = false ) then
                                        data := data |- [val(m).query.val, val(m).query.t,
0];

                                        arrayWrite := arrayWrite |- val(m).query.rank;
                                        midWrite := midWrite |- val(m).query.mid;
                                        sent_writeOK := true;
                                fi
                                s1 := false;
                                s2 := false;
                                s3 := false;
                                found := constant(false:Bool);
                                f := false;
                        fi
        if (val(m).query.type = "secure") then
                        for b:Nat where b < len(data) do
                                if (data[b].tag = val(m).query.t /\ data[b].bit = 0) then
                                        data[b].bit := 1;
                                fi
                        od
        fi
fi
```

Σχήμα 4.12: Σχέσεις Μεταβάσεων Αυτόματου Replica.

Εφόσον ορίσαμε και τις σχέσεις μεταβάσεων, το μόνο που μας μένει είναι ο ορισμός της τροχιάς Time, που ορίζεται στο τμήμα trajectories στο Σχήμα 4.13 και εξελίσσει το φυσικό ρολόι clock με ρυθμό 1.

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

Σχήμα 4.13: Τροχιές Αυτόματου Replica.

4.5 Αυτόματο Σύνθεσης

Εφόσον ολοκληρώσαμε τις προδιαγραφές των αυτόματων Directory, Client και Replica και τα αυτόματα που υλοποιούν το κανάλι επικοινωνίας TCP είναι ήδη υλοποιημένα, μπορούμε να συνεχίσουμε με την προδιαγραφή του αυτόματου σύνθεσης (Composition). Όπως φαίνεται και στο Σχήμα 4.14 αρχικά χρειάζεται να συμπεριλάβουμε τα αρχεία στα οποία υπάρχουν οι προδιαγραφές: του Λεξιλόγιο, των αυτόματων Directory, Client και Replica και των αυτόματων του καναλιού επικοινωνία TCP.

```
%%% .: Vocabulary :.
include "Vocabulary.tioa"
imports algorithm_voc, TCPObjectsVoc, TCPNodeVoc, tcp_specific_voc, JVMSocket,
JVMServerSocket, ChannelVoc
%%% .: TCP Mediator Automata :.
include "TCPRecvMed.tioa"
include "TCPSendMed.tioa"
include "TCPChanMed.tioa"
%%% .: Algorithm Automata :.
include "Client.tioa"
include "Directory.tioa"
include "Replica.tioa"
```

Σχήμα 4.14: Εισαγωγή Λεξιλόγιων και Αυτόματων στο Αυτόματο Σύνθεσης.

Όπως φαίνεται και στο Σχήμα 4.15 το αυτόματο σύνθεσης δέχεται ως παραμέτρους τους τέσσερις φυσικούς αριθμούς n1, n2, n3 και n4 που αντιστοιχούν στη IP διεύθυνση της διεργασίας, τον αριθμό θύρας port στον οποίο γίνεται η σύνδεση, το χρονικό διάστημα timeout στο οποίο πρέπει να γίνει η σύνδεση, το χρονικό διάστημα t1 στο οποίο πρέπει να πάρει απάντηση στο μήνυμα read, ένας client από ένα replica κατά τη λειτουργία της ανάγνωσης. Καθώς επίσης, το φυσικό αριθμό f που αντιστοιχεί στο μέγιστο αριθμό αντιγραφέων που μπορούν να καταρρεύσουν, το φυσικό αριθμό operation το οποίο καθορίζει ποια λειτουργία θα εκτελεστεί (1 για εγγραφή, 0 για ανάγνωση) και τέλος μια

ακολουθία από χαρακτήρες που αντιπροσωπεύουν τη τιμή που επιθυμούμε να γράψουμε στους αντιγραφείς. Ακολούθως, πρέπει να ορίσουμε από ποια αυτόματα αποτελείται το αυτόματο σύνθεσης στη δική μας περίπτωση αποτελείται από ένα αυτόματο Client (C), ένα αυτόματο Replica (R), ένα αυτόματο Directory (D), ένα αυτόματο SendMediator (SM), ένα αυτόματο ReceiveMediator (RM) και ένα αυτόματο ChannelMediator (CM). Επίσης, εδώ είναι και το σημείο που θα περάσουμε τις παραμέτρους που χρειάζεται το κάθε αυτόματο όπως εξηγήσαμε και πιο πάνω.

```
automaton Composition(n1:Nat, n2:Nat, n3:Nat, n4:Nat, port:Nat, timeout:Nat,
t1:DiscreteReal, f:Nat, operation:Nat, writeNum:Seq[Char])
components
	C : Client(f, t1);
	D : Directory(f);
	R : Replica(f);
	SM : SendMed(port, timeout);
	RM : RecvMed(port, timeout);
	CM : ChanMed(port, timeout);
```

Σχήμα 4.15: Ορισμός Παραμέτρων του Αυτόματου Σύνθεσης και των Αυτόματων που το απαρτίζουν.

Οι μεταβλητές καταστάσεων του αυτόματου σύνθεσης που βρίσκονται στο τμήμα states, φαίνονται στο Σχήμα 4.16. Πιο αναλυτικά:

- *IP:* είναι μια μεταβλητή τύπου Node που αντιπροσωπεύει την IP διεύθυνση της συγκεκριμένης διεργασίας. Αρχικοποιείται με τις πρώτες τέσσερις παραμέτρους του αυτόματου [n1,n2,n3,n4].
- *clients:* είναι μια κενή ακολουθία IP διευθύνσεων που αντιπροσωπεύει το σύνολο των πελατών που βρίσκονται στο σύστημα μας.
- *replicas:* είναι μια κενή ακολουθία IP διευθύνσεων που αντιπροσωπεύει το σύνολο των αντιγραφέων που βρίσκονται στο σύστημα μας.
- *directories:* είναι μια κενή ακολουθία IP διευθύνσεων που αντιπροσωπεύει το σύνολο των ευρετηρίων που βρίσκονται στο σύστημα μας.
- *ms:* δηλώνει ένα μήνυμα τύπου Chan_message που θα σταλεί από τη συγκεκριμένη διεργασία.
- *mr:* δηλώνει ένα μήνυμα τύπου Chan_message που θα λάβει η συγκεκριμένη διεργασία.

- *dowhile:* είναι μια boolean μεταβλητή που χρησιμοποιείται κατά την αποστολή και παραλαβή μηνυμάτων. Αρχικοποιείται με false.
- *isConn:* είναι μια boolean μεταβλητή που χρησιμοποιείται για να δούμε εάν δημιουργήθηκε επιτυχώς το κανάλι επικοινωνίας TCP ανάμεσα σε δυο διεργασίες. Αρχικοποιείται με false.
- *error:* είναι τύπου JVMError και αντιπροσωπεύει κάποιο μήνυμα λάθους που μπορεί να υπάρξει στη σύνδεση μεταξύ δυο διεργασιών.
- *runs:* είναι ένας φυσικός αριθμός που αντιπροσωπεύει τον αριθμό των επαναλήψεων του αλγόριθμου που θα εκτελεστεί στο πρόγραμμα μεταβάσεων. Αρχικοποιείται με 0.
- *world:* είναι μια κενή ακολουθία IP διευθύνσεων που αντιπροσωπεύει το σύνολο των πελατών, των ευρετηρίων και των αντιγραφέων που βρίσκονται στο σύστημα μας.
- *val:* είναι μια κενή ακολουθία χαρακτήρων η οποία αντιπροσωπεύει την τιμή που θα γράψει ένας client σε ένα replica κατά τη διαδικασία μιας εγγραφής.

```
schedule
        states
                IP : Node := [n1,n2,n3,n4];
                clients : Seq[Node] := {};
                replicas : Seq[Node] := {};
                directories : Seq[Node] := {};
                ms : Null[Chan_message] := nil();
                mr : Null[Chan_message] := nil();
                dowhile : Bool := true;
                isConn : Bool := false;
                error : Null[JVMError] := nil();
                runs : Nat := 0;
                world : Seq[Node] := {};
                val : Seq[Char] := {};
```

Σχήμα 4.16: Μεταβλητές Καταστάσεων του Αυτόματου Σύνθεσης.

Το πρόγραμμα μεταβάσεων (schedule) στο Σχήμα 4.17 αρχίζει με την εισαγωγή των IP διευθύνσεων των πελατών, των αντιγραφέων και των ευρετηρίων στις αντίστοιχες ακολουθίες και μετά ενώνουμε τις τρεις αυτές ακολουθίες στην ακολουθία world. Ακολούθως, γίνεται έλεγχος για να αναγνωρίσουμε σε ποια από τις τρεις κατηγορίες ανήκει η συγκεκριμένη διεργασία και ανάλογα πυροδοτούμε την εξωτερική ενέργεια systemInitialize που της αντιστοιχεί. Στη συνέχεια κάθε διεργασία δημιουργεί και δεσμεύει

μια υποδοχή για την επικοινωνία της με τις άλλες διεργασίες του συστήματος, κάνοντας χρήση των ενεργειών TCP_bind και TCP_respBind. Οι συνδέσεις μεταξύ των διεργασιών εγκαθιστονται με τις ενέργειες TCP_SenderOpen, TCP_accept, TCP_respAccept και TCP_respSenderOpen. Μετά γίνεται έλεγχος εάν το IP της διεργασίας ανήκει στους πελάτες και ανάλογα με την τιμή της μεταβλητής operation, εάν είναι 0 εκτελείται η λειτουργία της ανάγνωσης, διαφορετικά της εγγραφής. Στη συνέχεια εκτελείται ένας βρόγχος στον οποίο αρχικά εξελίσσεται η αντίστοιχη τροχιά κατά 1 ανάλογα σε ποια κατηγορία ανήκει η διεργασία αυτή, αν ανήκει στους πελάτες πυροδοτείται η εσωτερική ενέργεια readTimeout και ακολούθως πυροδοτείται η αντίστοιχη εσωτερική ενέργεια prep_messages. Ακολούθως, στέλλονται τα μηνύματα που βρίσκονται στο κανάλι και έχουν αποστολέα τη συγκεκριμένη διεργασία και παραλαμβάνονται μηνύματα που έχουν τη συγκεκριμένη διεργασία ως παραλήπτη. Στο τέλος, εάν η τιμή της μεταβλητής operation είναι ίση με 0, τότε πυροδοτείται η εξωτερική ενέργεια read_ok, διαφορετικά η εξωτερική ενέργεια write_ok και αν η διεργασία ανήκει στους αντιγραφείς τότε πυροδοτούνται και οι εσωτερικές ενέργειες gc και gossip.

```
do
        %% INITIALIZE
        clients := clients |- [10,16,12,230];
        clients := clients |- [10,16,12,144];
        clients := clients |- [10,16,12,227];
        directories := directories |- [10,16,12,116];
        directories := directories |- [10,16,12,118];
        directories := directories |- [10,16,12,240];
        replicas := replicas |- [10,16,12,213];
        replicas := replicas |- [10,16,12,119];
        replicas := replicas |- [10,16,12,169];
        world := world || clients ;
        world := world || directories;
        world := world || replicas;
        runs := 70;
        %% triger initialization of all component
        for k:Nat where k < len(clients) do
           if (IP = clients[k]) then
               fire output C.systemInitialize(clients, replicas, directories,
clients[k],k);
           fi
        od
        for k:Nat where k < len(directories) do
           if (IP = directories[k]) then
               fire output D.systemInitialize(clients, replicas, directories,
directories[k],k);
           fi
        od
```

```
        for k:Nat where k < len(replicas) do
           if (IP = replicas[k]) then
                fire output R.systemInitialize(clients, replicas, directories,
replicas[k],k);
           fi
        od
for k:Nat where k < len(replicas) do
           if (IP = replicas[k]) then
                fire output R.systemInitialize(clients, replicas, directories,
replicas[k],k);
           fi
        od
        %% bind to server socket
        %% everyone sets up their server socket
        fire output RM.TCP_bind(IP);
        fire output CM.TCP_respBind(error, IP);
        %% 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 len(world)*100;
                %% 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      %end of for
        if (IP \in clients) then
                if (operation = 0) then
                        fire input C.read;
                else
                        fire input C.write(writeNum);
                fi
        fi
        for i : Nat where i < runs do
                if (IP \in replicas) then
                        follow R.Time duration 1;
                fi
                if (IP \in clients) then
                        follow C.Time duration 1;
                fi
                if (IP \in directories) then
                        follow D.Time duration 1;
                fi
                if (IP \in clients) then
                        fire internal C.readTimeout;
                fi
        if (IP \in clients) then
                fire internal C.prep_messages;
        else if (IP \in directories) then
                        fire internal D.prep_messages;
                else if (IP \in replicas) then
                          fire internal R.prep_messages;
                     fi
                fi
        fi
```

```
        % SEND
        for j : Nat where j < len(world) do
                ms := nil();
                if (IP \in clients) then
                        fire output C.SEND(ms);
                else if (IP \in directories) then
                                fire output D.SEND(ms);
                        else if (IP \in replicas) then
                                        fire output R.SEND(ms);
                                fi
                        fi
                fi
                if (ms ~= nil()) then
                        fire output SM.TCP_write(ms, val(ms).sender,
val(ms).destination);
                        print val(ms).sender;
                fi
          od
          follow SM.DELAY duration 200;
          % -- extract messages from channel if there are any
          fire output RM.TCP_read;
          for j : Nat where j < len(world) do
                dowhile := true;
                while( dowhile = true ) do
                        fire output CM.TCP_respRead( ms );
                        if (ms ~= nil()) then
                                fire output RM.RECEIVE( ms );
                                print val(ms);
                        else
                                dowhile := false;
                        fi
                od
          od
          if (operation = 0) then
                fire output C.read_ok(val);
                print val;
          else
                fire output C.write_ok;
                if (IP \in replicas) then
                        fire internal R.gc;
                        fire internal R.gossip;
                fi
           fi
        od %end of for
od
```

Σχήμα 4.17: Πρόγραμμα μεταβάσεων και τροχιών του Αυτόματου Σύνθεσης.

Εφόσον ολοκληρώθηκε η προδιαγραφή του αυτόματου σύνθεσης, έχει σειρά η μετάφραση των προδιαγραφών σε κώδικα Java με το εργαλείο tempo2java που μας παρέχει το εργαλείο TEMPO, με τον τρόπο που δείξαμε στο Κεφάλαιο 2. Ακολούθως, στον κώδικα

Java που παράχθηκε χρειάζονταν κάποιες αλλαγές, καθώς επίσης και να προσθέσουμε κάποιες επιπλέον λειτουργίες. Αρχικά, διορθώσαμε κάποια κομμάτια κώδικα στα οποία δεν έγινε ορθή μετάφραση από το εργαλείο και στη συνέχεια υλοποιήσαμε τη συνάρτηση choose η οποία δεν λειτουργούσε σωστά. Πιο συγκεκριμένα η συνάρτηση αυτή επέστρεφε τυχαίες τιμές οι οποίες δεν περιλαμβάνονταν στα όρια που θέταμε. Το πρόβλημα αυτό το επιλύσαμε με τη χρήση for loops. Στο τέλος, έπρεπε να προσθέσουμε κάποιες επιπλέον λειτουργίες που αφορούσαν τη λειτουργία του αλγόριθμου με αρχεία. Επομένως, έπρεπε να γίνεται ανάγνωση/εγγραφή από/σε αρχεία, σπάσιμο των αρχείων σε πακέτα των 2000 bytes και εφαρμογή ελέγχου (checksum) με τον αλγόριθμο SHA για να ελέγχουμε εάν παραλήφθηκαν σωστά τα δεδομένα είτε στο replica, είτε στον client. Με αυτές τις αλλαγές καταλήξαμε στην τελική υλοποίηση μας, την οποία χρησιμοποιήσαμε στην αξιολόγηση του αλγόριθμου.

Κεφάλαιο 5

Πειραματική Αξιολόγηση

5.1 Πλατφόρμα Υλοποίησης

Για τους σκοπούς της πειραματικής αξιολόγησης του αλγόριθμου, εκτελέσαμε έξι διαφορετικά σενάρια που το καθένα μας βοήθησε να αντλήσουμε διαφορετικά συμπεράσματα. Τροποποιήσαμε τον κώδικα Java που πήραμε από το εργαλείο TEMPO, έτσι ώστε να υπολογίσουμε το χρονικό διάστημα (latency) που χρειάζεται η κάθε λειτουργία μέχρι να ολοκληρωθεί. Το χρονικό αυτό διάστημα, ξεκινάει να μετράει το χρόνο, από τη στιγμή που ο χρήστης στέλλει την αίτηση του, είτε για εγγραφή, είτε για ανάγνωση και σταματάει όταν ολοκληρωθεί η λειτουργία αυτή, δηλαδή όταν ισχύουν οι προϋποθέσεις των εξωτερικών ενεργειών write-ok$(v)_i$ και read-ok$(v)_i$ αντίστοιχα, του αυτόματου Client. Με το τέλος της κάθε λειτουργίας ο πελάτης περιμένει τυχαία ένα χρονικό διάστημα από 1 έως 10 δευτερόλεπτα, μέχρι να ξεκινήσει μια νέα λειτουργία. Το κάθε σενάριο εκτελεί 15 τέτοιες επαναλήψεις και τρέξαμε το κάθε σενάριο 3 φορές. Ακολούθως, πήραμε το μέσο όρο αυτών των τιμών που μας επιστρέφονταν και φτιάξαμε τα γραφήματα. Για το λόγο ότι οι παραμέτροι αλλάζουν σε κάθε σενάριο, θα εξηγήσουμε στο επόμενο υποκεφάλαιο τις παραμέτρους του κάθε σεναρίου λεπτομερώς. Επίσης, τα πειραματικά σενάρια που θα περιγράψουμε στο επόμενο υποκεφάλαιο, αρχικά τα

εκτελέσαμε κάνοντας χρήση της λειτουργίας gossip των αντιγραφέων. Αλλά, επειδή παρατηρήσαμε μεγάλες καθυστερήσεις στις λειτουργίες ανάγνωσης και εγγραφής, υποψιαστήκαμε ότι η λειτουργία gossip, ίσως να επιβάρυνε το σύστημα με περιττά μηνύματα και καθυστερήσεις. Γι' αυτό εκτελέσαμε τα πειραματικά σενάρια και χωρίς να κάνουμε χρήση της λειτουργίας gossip, για να ελέγξουμε πως επηρεάζει τελικά το χρόνο ολοκλήρωσης των λειτουργιών ανάγνωσης και εγγραφής. Στο σημείο αυτό να αναφέρουμε ότι η λειτουργία του gossip εκτελείται στους αντιγραφείς, κάθε φορά που θα ολοκληρωθεί μια νέα εγγραφή για να ενημερώσει τους άλλους αντιγραφείς για το νέο αρχείο. Επομένως, στέλλονται μηνύματα που περιέχουν το πραγματικό αρχείο διαχωρισμένο σε πακέτα. Κατά τη διάρκεια των σεναρίων χρησιμοποιήσαμε 2048 bytes ως default block size. Η επιλογή αυτού του μεγέθους block έγινε διότι επιθυμούμε να έχουμε ένα αρχείο μικρού μεγέθους για τα πρώτα σενάρια, δεν μπορούσαμε να χρησιμοποιήσουμε .docx αρχείο, γι' αυτό κάνουμε χρήση ενός .txt αρχείου. Όμως, σε επόμενο σενάριο θα δείξουμε ότι ο αλγόριθμος λειτουργεί και με άλλους τύπους αρχείων.

5.2 Σενάρια Υλοποίησης

Σενάριο 1 – Αναγνωστών

Στο σενάριο Αναγνωστών, εκτελέσαμε τον αλγόριθμο για 10, 20 και 30 ταυτόχρονους αναγνώστες. Ο αριθμός των ευρετηρίων και των αντιγραφέων παρέμενε σταθερός και ίσος με 3 και το αρχείο, τύπου .txt ίσο με 2KB. Ο παράγοντας f ήταν ίσος με 1. Σκοπός του σεναρίου αυτού, ήταν να ελέγξουμε πως επηρεάζεται η συνέπεια και η απόδοση του συστήματος, αλλά και το χρονικό διάστημα για την ολοκλήρωση της λειτουργίας της ανάγνωσης με την αύξηση του αριθμού των αναγνωστών. (Αρχικά οι αντιγραφείς δεν έχουν αποθηκευμένο στα δεδομένα τους κάποιο αρχείο για να εκτελείται η λειτουργία της ανάγνωσης. Για αυτό τρέξαμε μια λειτουργία εγγραφής, με επανάληψη μόνο μια φορά για να γράψει ένα αρχείο.)

Σενάριο 2 – Εγγραφέων

Στο σενάριο Εγγραφέων, εκτελέσαμε τον αλγόριθμο για 10, 20 και 30 ταυτόχρονους εγγραφείς. Ο αριθμός των ευρετηρίων και των αντιγραφέων παρέμενε σταθερός και ίσος με 3 και το αρχείο, τύπου .txt ίσο με 2KB. Ο παράγοντας f ήταν ίσος με 1. Σκοπός του

σεναρίου αυτού, ήταν να ελέγξουμε πως επηρεάζεται η συνέπεια και η απόδοση του συστήματος, αλλά και το χρονικό διάστημα για την ολοκλήρωση της λειτουργίας της εγγραφής, με την αύξηση του αριθμού των εγγραφέων.

Σενάριο 3 – Αντιγραφέων

Στο σενάριο Αντιγραφέων, χρησιμοποιούμε 10 ταυτόχρονους εγγραφείς, μαζί με 20 ταυτόχρονους αναγνώστες, με σταθερό αριθμό ευρετηρίων και ίσο με 3 και το μέγεθος του αρχείου παρέμενε σταθερό, τύπου .txt ίσο με 2KB. Εκτελέσαμε το σενάριο αυτό για 3, 6 και 9 αντιγραφείς και θέταμε τον παράγοντα f ίσο με 1, 2 και 3 αντίστοιχα. Σκοπός του σεναρίου αυτού, ήταν να δούμε ποιος αριθμός των αντιγραφέων μας δίνει καλύτερη απόδοση και πως επηρεάζει η αύξηση των αντιγραφέων και του f, τη συνέπεια του συστήματος.

Σενάριο 4 – Ευρετηρίων

Στο σενάριο Ευρετηρίων, χρησιμοποιούμε 10 ταυτόχρονους εγγραφείς, μαζί με 20 ταυτόχρονους αναγνώστες, με σταθερό αριθμό αντιγραφέων και ίσο με 3 και το μέγεθος του αρχείου παρέμενε σταθερό, τύπου .txt και ίσο με 2KB. Εκτελέσαμε το σενάριο αυτό για 3, 6 και 9 ευρετήρια. Ο παράγοντας f ήταν ίσος με 1. Σκοπός του σεναρίου αυτού, ήταν να δούμε ποιος αριθμός των ευρετηρίων μας δίνει καλύτερη απόδοση και πως επηρεάζει η αύξηση των ευρετηρίων τη συνέπεια του συστήματος.

Σενάριο 5 – Μέγεθος Αρχείου

Στο σενάριο Μέγεθος Αρχείου, χρησιμοποιούμε 5 ταυτόχρονους εγγραφείς, μαζί με 10 ταυτόχρονους αναγνώστες, με σταθερό αριθμό αντιγραφέων και ευρετηρίων, ίσο με 3. Ο παράγοντας f ήταν ίσος με 1. Επίσης, κρατάμε σταθερό τον τύπο του αρχείου σε .docx. Επειδή επιθυμούμε να τρέξουμε το σενάριο για ποιο μεγάλο αρχείο, αλλάξαμε το default block size των πακέτων από 2000 bytes σε 64000 bytes για να μειώσουμε τα πακέτα που θα υπάρχουν στο σύστημα, έτσι ώστε να μπορούν να αντεπεξέρχονται οι αντιγραφείς αποδοτικά. Εκτελέσαμε το σενάριο αυτό για αρχεία μεγέθους 125KB, 250KB και 500KB. Ενώ, για το πείραμα που δεν κάνει χρήση gossip, χρησιμοποιήσαμε και αρχείο μεγέθους 1

MB. Σκοπός του σεναρίου αυτού, ήταν να ελέγξουμε πως το μέγεθος του αρχείου επηρεάζει την απόδοση και τη συνέπεια του συστήματος.

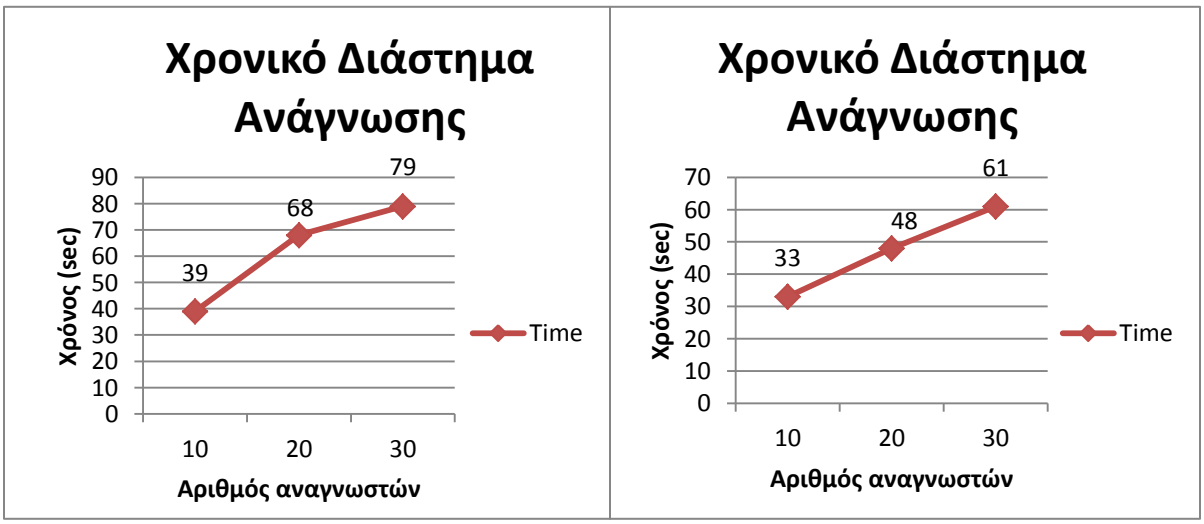
Σενάριο 6 – Τύπος Αρχείου

Στο σενάριο Τύπος Αρχείου, χρησιμοποιούμε 5 ταυτόχρονους εγγραφείς, μαζί με 10 ταυτόχρονους αναγνώστες, με σταθερό αριθμό αντιγραφέων και ευρετηρίων, ίσο με 3 και το μέγεθος του αρχείου παρέμενε σταθερό και ίσο με 13KB. Ο παράγοντας f ήταν ίσος με 1. Εκτελέσαμε το σενάριο αυτό για τέσσερις τύπους αρχείων .docx, .txt, .pdf και .jpg. Σκοπός του σεναρίου αυτού, ήταν να ελέγξουμε πως ο τύπος του αρχείου επηρεάζει την απόδοση και τη συνέπεια του συστήματος.

5.3 Αποτελέσματα

Σενάριο Αναγνωστών

Αναμένουμε ότι ο χρόνος ολοκλήρωσης της λειτουργίας ανάγνωσης θα αυξηθεί κατά την αύξηση των ταυτόχρονων αναγνωστών, λόγω του ότι θα αυξηθούν και τα μηνύματα που υπάρχουν στο σύστημα, με αποτέλεσμα οι εξυπηρετητές να καθυστερούν στις απαντήσεις τους. Στο Σχήμα 5.1 παρουσιάζεται ο μέσος χρόνος που χρειάζεται για να ολοκληρωθεί μια λειτουργία ανάγνωσης, στα αριστερά κάνοντας χρήση της λειτουργίας gossip, ενώ στα δεξιά χωρίς τη χρήση της λειτουργίας gossip. Στον οριζόντιο άξονα βλέπουμε τον αριθμό των αναγνωστών που λαμβάνουν μέρος κάθε φορά και στον κάθετο άξονα βλέπουμε το χρόνο που χρειάζεται σε δευτερόλεπτα.



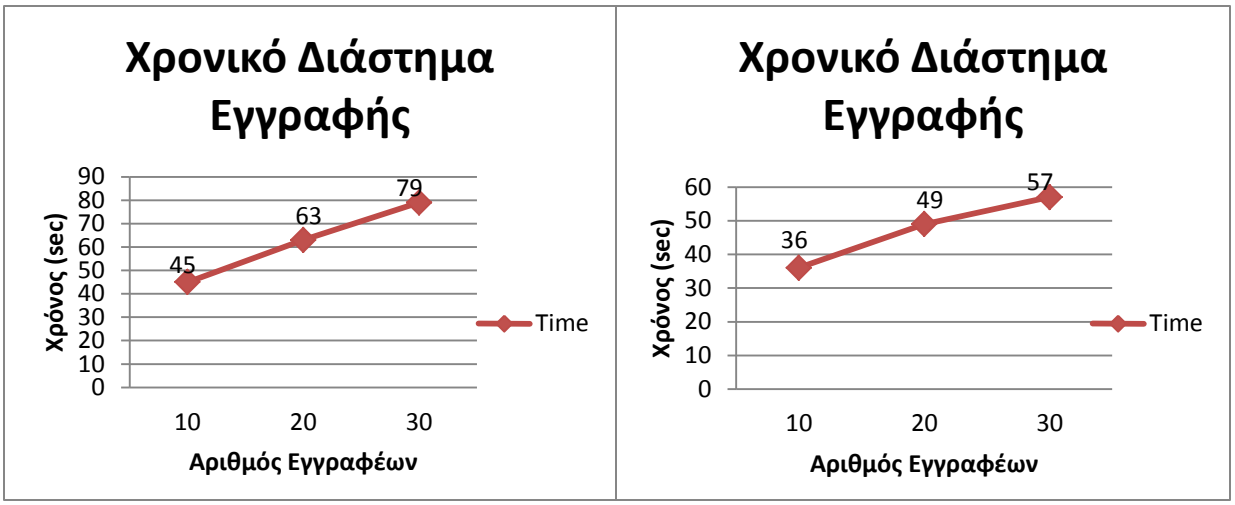
Σχήμα 5.1: Αριστερά - Χρονικό Διάστημα Ολοκλήρωσης της Λειτουργίας Ανάγνωσης με τη χρήση gossip.

Δεξιά – Χρονικό Διάστημα Ολοκλήρωσης της Λειτουργίας Ανάγνωσης χωρίς τη χρήση gossip.

Από τις δύο γραφικές παραστάσεις στο Σχήμα 5.1, παρατηρούμε ότι ο χρόνος που χρειάζεται για να ολοκληρωθεί μια λειτουργία ανάγνωσης, επηρεάζεται άμεσα από τον αριθμό των ταυτόχρονων αναγνωστών που υπάρχουν στο σύστημα μας. Βλέπουμε ότι καθώς αυξάνονταν οι αναγνώστες, αυξανόταν και ο χρόνος που χρειαζόταν για να ολοκληρωθεί η λειτουργία ανάγνωσης. Αυτό είναι αναμενόμενο, εφόσον όσο αυξάνεται ο αριθμός των αναγνωστών φορτώνονται περισσότερο και οι εξυπηρετητές, με αποτέλεσμα να καθυστερούν περισσότερο να στείλουν τις απαντήσεις στους αναγνώστες. Ακολούθως, όσον αφορά την σύγκριση των χρόνων των δυο γραφικών παραστάσεων είναι εμφανές ότι στη γραφική παράσταση στα δεξιά, μειώνεται ο χρόνος που χρειάζεται η λειτουργία ανάγνωσης. Όπως ανέφερα πιο πάνω, επειδή αρχικά δεν υπάρχει κάποιο αρχείο στα δεδομένα των αντιγραφέων, εκτελούμε εγγραφή για μια φορά. Επομένως, παρ' όλο που βρισκόμαστε στη λειτουργία της ανάγνωσης στην γραφική παράσταση στα αριστερά εκτελείται η λειτουργία gossip, γιατί υπάρχει αρχείο αποθηκευμένο στους αντιγραφείς. Άρα, επιβεβαιώνεται η αρχική μας σκέψη ότι η λειτουργία gossip προσθέτει καθυστερήσεις.

Σενάριο Εγγραφέων

Αναμένουμε ότι ο χρόνος ολοκλήρωσης της λειτουργίας εγγραφής θα αυξηθεί κατά την αύξηση των ταυτόχρονων εγγραφέων λόγω του ότι θα αυξηθούν και τα μηνύματα που υπάρχουν στο σύστημα, με αποτέλεσμα οι εξυπηρετητές να καθυστερούν στις απαντήσεις τους. Στο Σχήμα 5.2 παρουσιάζεται ο μέσος χρόνος που χρειάζεται για να ολοκληρωθεί μια λειτουργία εγγραφής, στα αριστερά κάνοντας χρήση της λειτουργίας gossip, ενώ στα δεξιά χωρίς τη χρήση της λειτουργίας gossip. Στον οριζόντιο άξονα βλέπουμε τον αριθμό των εγγραφέων που λαμβάνουν μέρος κάθε φορά και στον κάθετο άξονα βλέπουμε το χρόνο που χρειάζεται σε δευτερόλεπτα.



Σχήμα 5.2:

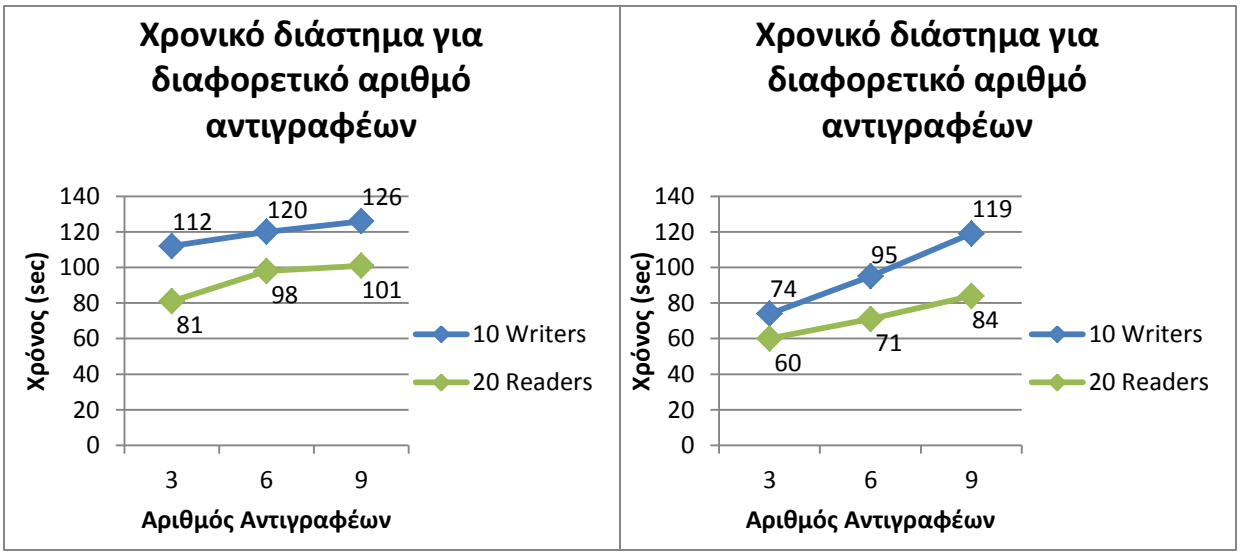
Αριστερά - Χρονικό Διάστημα Ολοκλήρωσης της Λειτουργίας Εγγραφής με τη χρήση gossip.

Δεξιά – Χρονικό Διάστημα Ολοκλήρωσης της Λειτουργίας Εγγραφής χωρίς τη χρήση gossip.

Από τις δυο γραφικές παραστάσεις στο Σχήμα 5.2, παρατηρούμε ότι ο χρόνος που χρειάζεται για να ολοκληρωθεί μια λειτουργία εγγραφής, επηρεάζεται άμεσα από τον αριθμό των ταυτόχρονων εγγραφέων που υπάρχουν στο σύστημα μας. Βλέπουμε ότι καθώς αυξάνονταν οι εγγραφείς, αυξανόταν και ο χρόνος που χρειαζόταν για να ολοκληρωθεί η λειτουργία εγγραφής. Αυτό είναι αναμενόμενο, εφόσον όσο αυξάνεται ο αριθμός των εγγραφέων φορτώνονται περισσότερο και οι εξυπηρετητές, με αποτέλεσμα να καθυστερούν περισσότερο να στείλουν τις απαντήσεις στους εγγραφείς. Ακολούθως, όσον αφορά την σύγκριση των χρόνων των δυο γραφικών παραστάσεων είναι εμφανές ότι στη γραφική παράσταση στα δεξιά μειώνεται ο χρόνος που χρειάζεται η λειτουργία εγγραφής. Αυτό συμβαίνει επειδή απενεργοποιήσαμε τη λειτουργία του gossip, επομένως δεν επιβαρύνονται οι αντιγραφείς με αυτή τη λειτουργία. Άρα, επιβεβαιώνεται η αρχική μας σκέψη ότι η λειτουργία gossip προσθέτει καθυστερήσεις.

Σενάριο Αντιγραφέων

Αναμένουμε ότι ο χρόνος που χρειάζεται να ολοκληρωθεί η λειτουργία εγγραφής θα αυξάνεται καθώς αυξάνεται ο αριθμός των αντιγραφέων, γιατί οι πελάτες θα στέλλουν περισσότερα μηνύματα κατά την εκτέλεση της λειτουργίας, εφόσον θα στέλλουν μήνυμα σε όλους τους αντιγραφείς και θα περιμένουν απάντηση από f+1. Ενώ, για τη λειτουργία της ανάγνωσης δεν αναμένουμε μεγάλες διαφορές στο χρόνο που χρειάζεται, επειδή η λειτουργία της ανάγνωσης δεν σχετίζεται με τον παράγοντα f. Στο Σχήμα 5.3, παρουσιάζεται ο μέσος χρόνος που χρειάζεται για να ολοκληρωθεί μια λειτουργία εγγραφής ή ανάγνωσης, στα αριστερά κάνοντας χρήση της λειτουργίας gossip, ενώ στα δεξιά χωρίς τη χρήση της λειτουργίας gossip. Στον οριζόντιο άξονα βλέπουμε τον αριθμό των αντιγραφέων που λαμβάνουν μέρος κάθε φορά και στον κάθετο άξονα βλέπουμε το χρόνο που χρειάζεται σε δευτερόλεπτα. Η μπλε γραμμή αντιπροσωπεύει το χρονικό διάστημα που χρειάζονται οι 10 εγγραφείς, κατά τη λειτουργία της εγγραφής, ενώ η πράσινη γραμμή αντιπροσωπεύει το χρονικό διάστημα που χρειάζονται οι 20 αναγνώστες, κατά τη λειτουργία της ανάγνωσης.



Σχήμα 5.3:

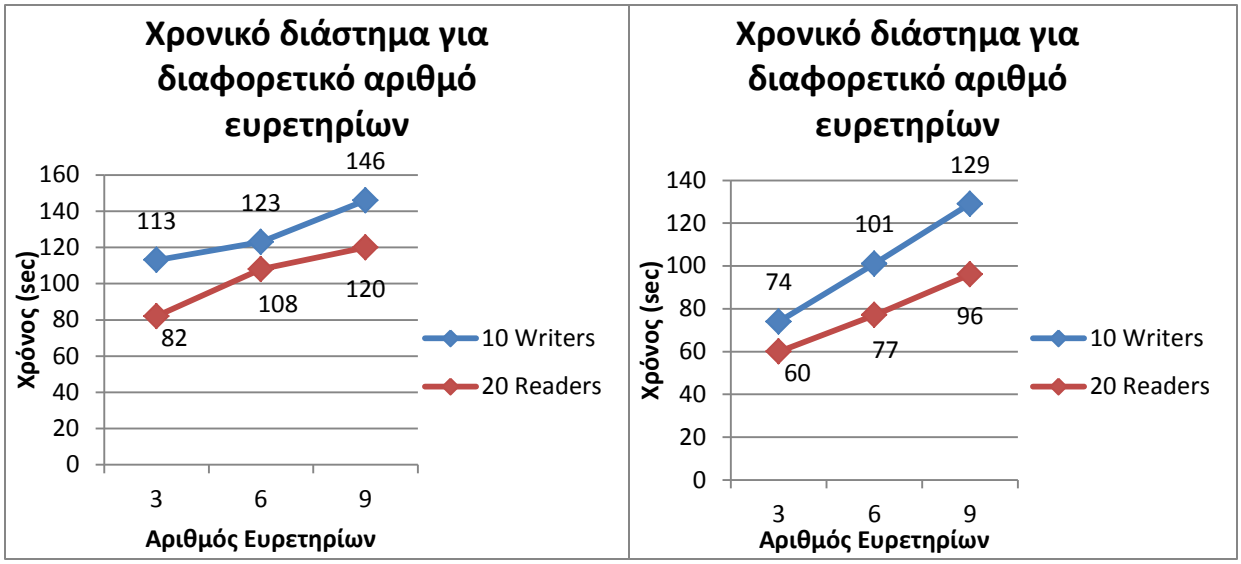
Αριστερά - Χρονικό Διάστημα ολοκλήρωσης για διαφορετικό αριθμό αντιγραφέων με τη χρήση gossip.

Δεξιά - Χρονικό Διάστημα ολοκλήρωσης για διαφορετικό αριθμό αντιγραφέων χωρίς τη χρήση gossip.

Από τις δυο γραφικές παραστάσεις στο Σχήμα 5.3, παρατηρούμε ότι αυξάνοντας τον αριθμό των αντιγραφέων και τον παράγοντα f, δεν βελτιώνεται η απόδοση του συστήματος, αντίθετα χειροτερεύει. Ακόμη, παρατηρούμε ότι ο χρόνος των αναγνωστών αυξάνεται με μικρότερο ρυθμό αύξησης από τον χρόνο που χρειάζονται οι εγγραφείς. Αυτό ίσως να οφείλεται στο λόγο που εξηγήσαμε πιο πάνω. Επίσης, παρατηρούμε ότι οι εγγραφείς χρειάζονται περισσότερο χρόνο από τους αναγνώστες για να ολοκληρώσουν τη λειτουργία τους. Αυτό οφείλεται στο ότι οι εγγραφείς χρειάζεται να επικοινωνήσουν ακόμη μια φορά με τους αντιγραφείς για να στείλουν μήνυμα secure και αυτό ίσως να έχει ως αποτέλεσμα επιπλέον καθυστέρηση. Ακολούθως, όσον αφορά την σύγκριση των χρόνων των δυο γραφικών παραστάσεων είναι εμφανές ότι στη γραφική παράσταση στα δεξιά μειώνεται ο χρόνος που χρειάζονται οι δυο λειτουργίες. Αυτό συμβαίνει επειδή απενεργοποιήσαμε τη λειτουργία του gossip, επομένως δεν επιβαρύνονται οι αντιγραφείς με αυτή τη λειτουργία. Άρα, επιβεβαιώνεται η αρχική μας σκέψη ότι η λειτουργία gossip προσθέτει καθυστερήσεις.

Σενάριο Ευρετηρίων

Αναμένουμε ότι ο χρόνος ολοκλήρωσης και των δυο λειτουργιών, τόσο της ανάγνωσης, όσο και της εγγραφής θα αυξάνεται καθώς αυξάνεται ο αριθμός των ευρετηρίων. Αυτό αναμένουμε να συμβεί, λόγο του ότι οι εγγραφείς και οι αναγνώστες θα στέλλουν περισσότερα μηνύματα, εφόσον πρέπει να στείλουν μήνυμα σε όλα τα ευρετήρια και θα περιμένουν απάντηση από πλειοψηφία. Στο Σχήμα 5.4, παρουσιάζεται ο μέσος χρόνος που χρειάζεται για να ολοκληρωθεί μια λειτουργία εγγραφής ή ανάγνωσης, στα αριστερά κάνοντας χρήση της λειτουργίας gossip, ενώ στα δεξιά χωρίς τη χρήση της λειτουργίας gossip. Στον οριζόντιο άξονα βλέπουμε τον αριθμό των ευρετηρίων που λαμβάνουν μέρος κάθε φορά και στον κάθετο άξονα βλέπουμε το χρόνο που χρειάζεται σε δευτερόλεπτα. Η μπλε γραμμή αντιπροσωπεύει το χρονικό διάστημα που χρειάζονται οι 10 εγγραφείς, κατά τη λειτουργία της εγγραφής, ενώ η κόκκινη γραμμή αντιπροσωπεύει το χρονικό διάστημα που χρειάζονται οι 20 αναγνώστες, κατά τη λειτουργία της ανάγνωσης.



Σχήμα 5.4:

Αριστερά - Χρονικό Διάστημα ολοκλήρωσης για διαφορετικό αριθμό ευρετηρίων με τη χρήση gossip.

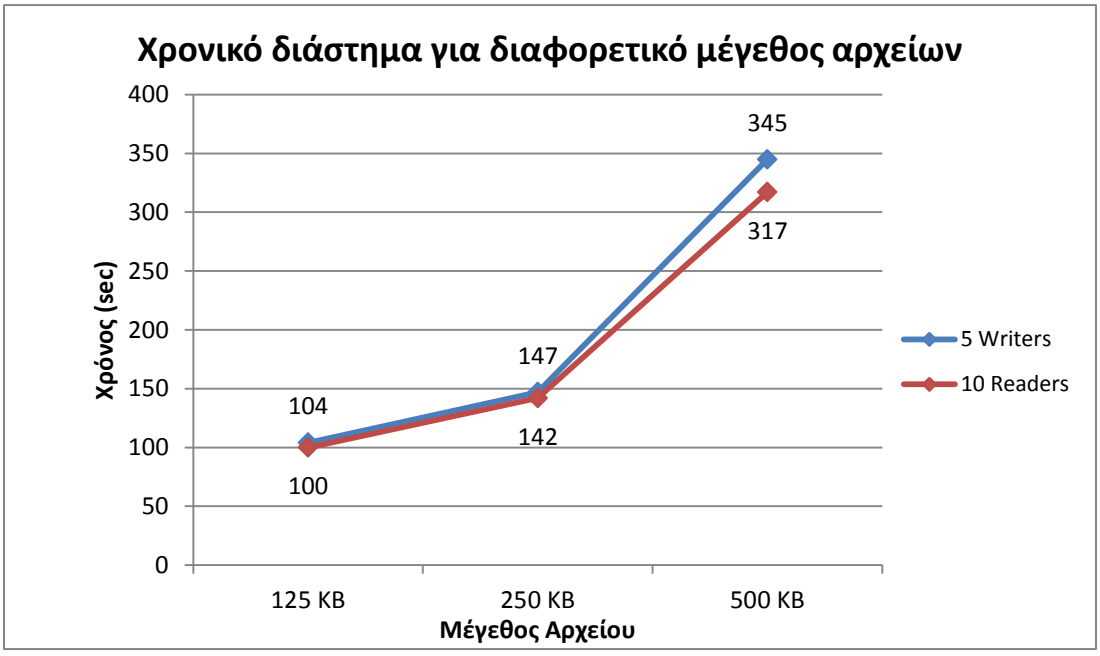
Δεξιά - Χρονικό Διάστημα ολοκλήρωσης για διαφορετικό αριθμό ευρετηρίων χωρίς τη χρήση gossip.

Από τις δυο γραφικές παραστάσεις στο Σχήμα 5.4, παρατηρούμε ότι αυξάνοντας τον αριθμό των ευρετηρίων δεν βελτιώνεται η απόδοση του συστήματος, αντίθετα χειροτερεύει. Αυτό ίσως να οφείλεται στο λόγο που εξηγήσαμε πιο πάνω. Επίσης, παρατηρούμε ότι οι εγγρφαφείς χρειάζονται περισσότερο χρόνο από τους αναγνώστες για να ολοκληρώσουν τη λειτουργία τους. Αυτό οφείλεται στο ότι οι εγγραφείς πρέπει να επικοινωνήσουν ακόμη μια φορά με τους αντιγραφείς για να στείλουν το μήνυμα secure και αυτό έχει ως αποτέλεσμα επιπλέον καθυστέρηση. Επίσης, είναι ξεκάθαρο ότι ο αριθμός των ευρετηρίων ευθύνεται για την αύξηση της καθυστέρησης, εφόσον η επικοινωνία με αυτούς δεν περιέχει πραγματικά δεδομένα. Περιέχει τα μεταδεδομένα που έχουν σταθερό μέγεθος και επίσης το χρονικό διάστημα μιας λειτουργίας δεν επηρεάζεται από τη λειτουργία που εκτελείται. Ακολούθως, όσον αφορά την σύγκριση των χρόνων των δυο γραφικών παραστάσεων είναι εμφανές ότι στη γραφική παράσταση στα δεξιά μειώνεται ο χρόνος που χρειάζονται οι δυο λειτουργίες. Αυτό συμβαίνει επειδή απενεργοποιήσαμε τη λειτουργία του gossip, επομένως δεν επιβαρύνονται οι αντιγραφείς με αυτή τη λειτουργία.

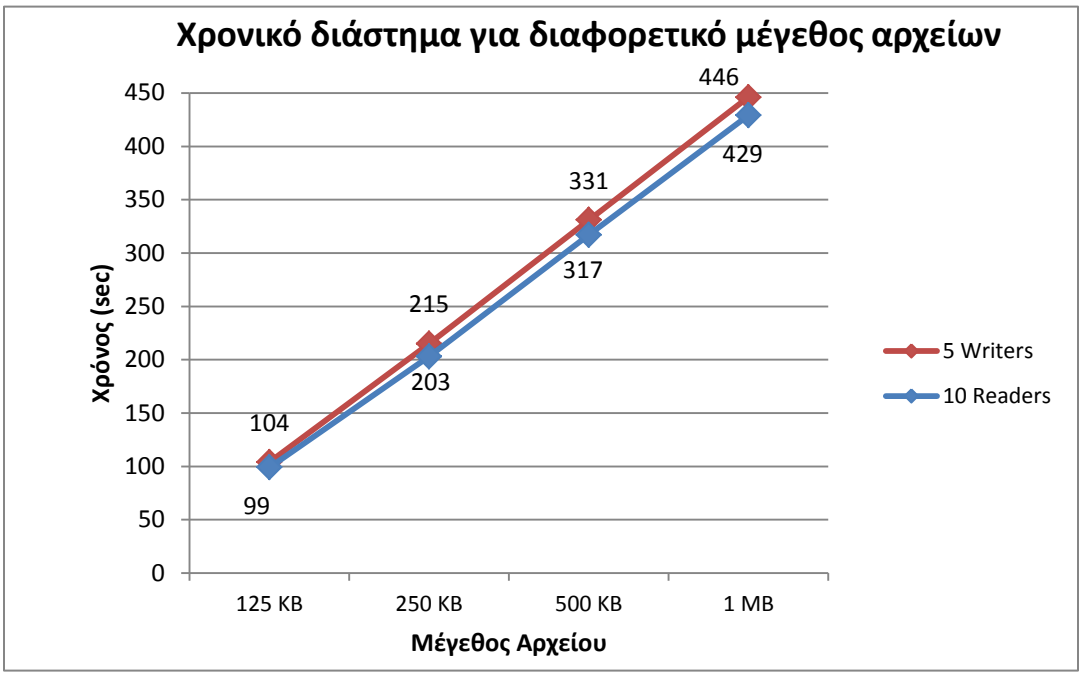
Άρα, επιβεβαιώνεται η αρχική μας σκέψη ότι η λειτουργία gossip προσθέτει καθυστερήσεις.

Σενάριο Μέγεθος Αρχείου

Αναμένουμε ότι το χρονικό διάστημα που χρειάζεται για την ολοκλήρωση μιας λειτουργίας εγγραφής ή ανάγνωσης θα αυξάνεται καθώς αυξάνεται και το μέγεθος του αρχείου. Αυτό οφείλεται στο ότι αυξάνονται τα πακέτα που πρέπει να στείλουν οι πελάτες προς τους αντιγραφείς, κατά τη διάρκεια της εγγραφής και τα πακέτα που πρέπει να στείλουν οι αντιγραφείς στους πελάτες κατά τη διάρκεια της ανάγνωσης. Στο Σχήμα 5.5, παρουσιάζεται ο μέσος χρόνος που χρειάζεται για να ολοκληρωθεί μια λειτουργία εγγραφής ή ανάγνωσης κάνοντας χρήση της λειτουργίας gossip. Ενώ, στο Σχήμα 5.6, παρουσιάζεται ο μέσος χρόνος που χρειάζεται για να ολοκληρωθεί μια λειτουργία εγγραφής ή ανάγνωσης χωρίς τη χρήση της λειτουργίας gossip. Στον οριζόντιο άξονα των δυο σχημάτων βλέπουμε το μέγεθος των αρχείων που χρησιμοποιήσαμε σε αυτό το σενάριο και στον κάθετο άξονα βλέπουμε το χρόνο που χρειάζεται σε δευτερόλεπτα. Η κόκκινη γραμμή αντιπροσωπεύει το χρονικό διάστημα που χρειάζονται οι 5 εγγραφείς, κατά τη λειτουργία της εγγραφής, ενώ η μπλε γραμμή αντιπροσωπεύει το χρονικό διάστημα που χρειάζονται οι 10 αναγνώστες, κατά τη λειτουργία της ανάγνωσης.



Σχήμα 5.5: Χρονικό Διάστημα ολοκλήρωσης για διαφορετικό μέγεθος αρχείων με τη χρήση gossip.

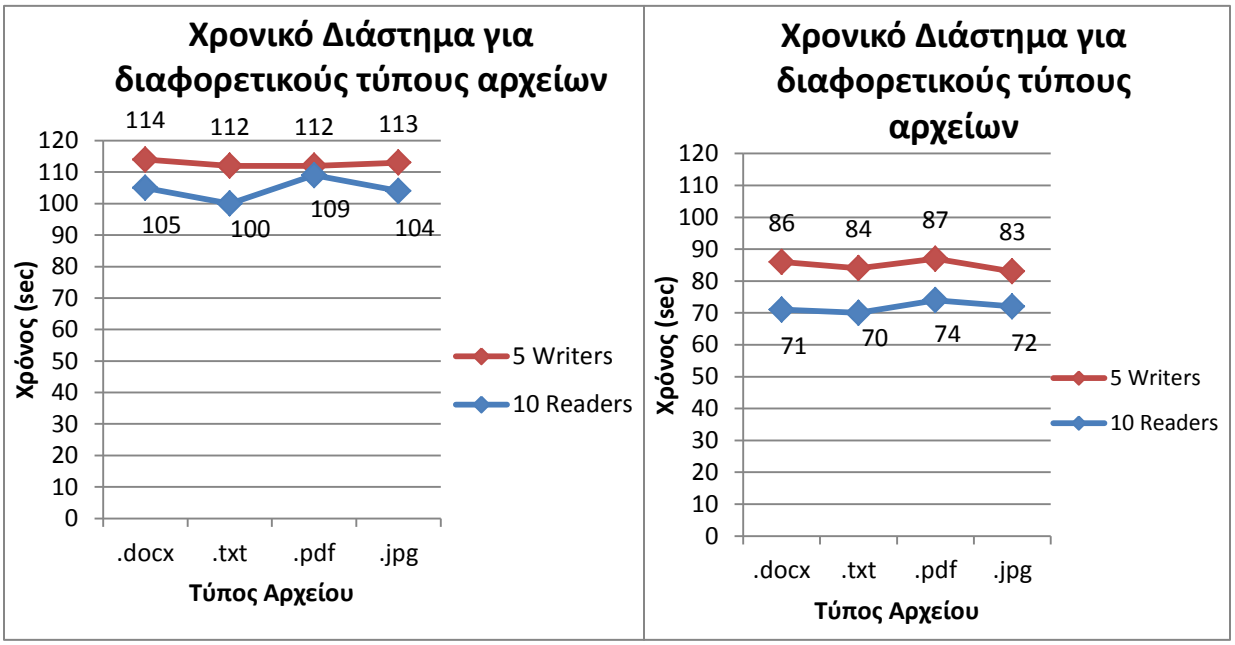


Σχήμα 5.6: Χρονικό Διάστημα ολοκλήρωσης για διαφορετικό μέγεθος αρχείων χωρίς τη χρήση gossip.

Από τις δυο γραφικές παραστάσεις στο Σχήμα 5.5 και στο Σχήμα 5.6, παρατηρούμε ότι ο χρόνος που χρειάζονται οι λειτουργίες για να ολοκληρωθούν αυξάνεται, καθώς αυξάνεται και το μέγεθος του αρχείου. Αυτό ίσως να οφείλεται στο λόγο που εξηγήσαμε πιο πάνω. Επίσης, βλέπουμε οι εγγραφείς χρειάζονται περισσότερο χρόνο από τους αναγνώστες για να ολοκληρώσουν τη λειτουργία τους. Αυτό οφείλεται στο ότι οι εγγραφείς πρέπει να επικοινωνήσουν ακόμη μια φορά με τους αντιγραφείς για να στείλουν το μήνυμα secure και αυτό έχει ως αποτέλεσμα επιπλέον καθυστέρηση. Επομένως, παρατηρούμε ότι το μέγεθος του αρχείου διαδραματίζει σημαντικό ρόλο στο χρόνο που χρειάζεται μια λειτουργία για να ολοκληρωθεί. Ακολούθως, όσον αφορά την σύγκριση των χρόνων των δυο γραφικών παραστάσεων είναι εμφανές ότι στη γραφική παράσταση στο Σχήμα 5.6, μειώνεται ο χρόνος που χρειάζονται οι δυο λειτουργίες, σε σχέση με το Σχήμα 5.5. Αυτό συμβαίνει επειδή απενεργοποιήσαμε τη λειτουργία του gossip, επομένως δεν επιβαρύνονται οι αντιγραφείς με αυτή τη λειτουργία. Άρα, επιβεβαιώνεται η αρχική μας σκέψη ότι η λειτουργία gossip προσθέτει καθυστερήσεις.

Σενάριο Τύπος Αρχείου

Αναμένουμε ότι ο τύπος του αρχείου δεν θα επηρεάσει τον χρόνο ολοκλήρωσης οποιασδήποτε λειτουργίας, επειδή κρατούμε σταθερός το μέγεθος των αρχείων, καθώς και επειδή στην υλοποίηση μας διαβάζουμε διφυολέξεις (bytes). Στο Σχήμα 5.7, παρουσιάζεται ο μέσος χρόνος που χρειάζεται για να ολοκληρωθεί μια λειτουργία εγγραφής ή ανάγνωσης, στα αριστερά κάνοντας χρήση της λειτουργίας gossip, ενώ στα δεξιά χωρίς τη χρήση της λειτουργίας gossip. Στον οριζόντιο άξονα βλέπουμε τους τύπους των αρχείων που χρησιμοποιήσαμε σε αυτό το σενάριο και στον κάθετο άξονα βλέπουμε το χρόνο που χρειάζεται σε δευτερόλεπτα. Η κόκκινη γραμμή αντιπροσωπεύει το χρονικό διάστημα που χρειάζονται οι 5 εγγραφείς, κατά τη λειτουργία της εγγραφής, ενώ η μπλε γραμμή αντιπροσωπεύει το χρονικό διάστημα που χρειάζονται οι 10 αναγνώστες, κατά τη λειτουργία της ανάγνωσης.



Σχήμα 5.7:

Αριστερά - Χρονικό Διάστημα ολοκλήρωσης για διαφορετικό τύπο αρχείου με τη χρήση gossip.

Δεξιά - Χρονικό Διάστημα ολοκλήρωσης για διαφορετικό τύπο αρχείου χωρίς τη χρήση gossip.

Από τις δυο γραφικές παραστάσεις στο Σχήμα 5.7, δεν παρατηρούμε μεγάλες αλλαγές στο χρόνο που χρειάζεται να ολοκληρωθεί η κάθε λειτουργία ανάλογα με τον τύπο του

αρχείου. Πιο συγκεκριμένα, βλέπουμε ότι το χρονικό διάστημα ολοκλήρωσης της εγγραφής και της ανάγνωσης παραμένει σχεδόν το ίδιο. Αυτό ίσως να οφείλεται στο λόγο που εξηγήσαμε πιο πάνω. Όπως βλέπουμε οι εγγραφείς χρειάζονται περισσότερο χρόνο από τους αναγνώστες για να ολοκληρώσουν τη λειτουργία τους. Αυτό οφείλεται στο ότι οι εγγραφείς πρέπει να επικοινωνήσουν ακόμη μια φορά με τους αντιγραφείς για να στείλουν το μήνυμα secure και αυτό έχει ως αποτέλεσμα επιπλέον καθυστέρηση. Επομένως, παρατηρούμε ότι ο τύπος του αρχείου δεν διαδραματίζει σημαντικό ρόλο στο χρόνο που χρειάζεται μια λειτουργία για να ολοκληρωθεί. Ακολούθως, όσον αφορά την σύγκριση των χρόνων των δυο γραφικών παραστάσεων είναι εμφανές ότι στη γραφική παράσταση στα δεξιά μειώνεται ο χρόνος που χρειάζονται οι δυο λειτουργίες. Αυτό συμβαίνει επειδή απενεργοποιήσαμε τη λειτουργία του gossip, επομένως δεν επιβαρύνονται οι αντιγραφείς με αυτή τη λειτουργία. Άρα, επιβεβαιώνεται η αρχική μας σκέψη ότι η λειτουργία gossip προσθέτει καθυστερήσεις.

Κεφάλαιο 6

Συμπεράσματα

6.1 Γενικά Συμπεράσματα

Στην παρούσα Διπλωματική Εργασία έγινε η προδιαγραφή και η υλοποίηση του αλγόριθμου LDR, καθώς και η πειραματική αξιολόγηση του στο ρεαλιστικό και ασύγχρονο δίκτυο PlanetLab. Μέσα από την πειραματική αξιολόγηση και τα διαφορετικά σενάρια που τρέξαμε, παρατηρήσαμε ότι οι servers ανταποκρίνονται σχεδόν σε όλα τα αιτήματα των πελατών και ότι πάντοτε ισχύει η ατομικότητα. Πάντα κατά τη λειτουργία της ανάγνωσης επιστρέφουν στον client την τελευταία τιμή που γράφτηκε, χωρίς να παραβιάζουν τη συνθήκη ατομικότητας. Επομένως, ο αλγόριθμος LDR επιτυγχάνει τους τρεις κύριους στόχους που ο κάθε κατανεμημένος αλγόριθμος για διαχείριση αρχείων θα επιθυμούσε να έχει, δηλαδή αντιγραφή, αποδοτικότητα και ατομικότητα. Το κλειδί για την απόδοση του συστήματος είναι ότι διαχωρίζουμε τα μεταδεδομένα από τα πραγματικά δεδομένα και χρησιμοποιούμε τα πρώτα για να εκτελούμε τις περισσότερες λειτουργίες. Κάποιες καθυστερήσεις που παρατηρούμε, οφείλονται στο ότι υπάρχουν σφάλματα κατάρρευσης, οι

κόμβοι μερικές φορές δεν ανταποκρίνονται, καθώς και στην απώλεια κάποιων μηνυμάτων. Επίσης, από την αξιολόγηση της προδιαγραφής, παρατηρήσαμε ότι ο χρόνος για να ολοκληρωθεί μια λειτουργία εξαρτάται από τον αριθμό των ευρετηρίων και των αντιγραφέων που υπάρχουν στο σύστημα, τον παράγοντα f, καθώς και το μέγεθος του πακέτου (default block size) και το μέγεθος του αρχείου. Ο τύπος του αρχείου δεν διαδραματίζει τόσο σημαντικό ρόλο στον χρόνο που χρειάζεται μια λειτουργία για να ολοκληρωθεί. Τέλος, καταλήξαμε στο συμπέρασμα ότι η λειτουργία gossip, προσθέτει καθυστερήσεις στο χρόνο ολοκλήρωσης των λειτουργιών ανάγνωσης και εγγραφής.

6.2 Προβλήματα και Παραδοχές

Κατά την εκπόνηση της παρούσας Διπλωματικής Εργασίας παρουσιάστηκαν κάποια προβλήματα όσον αφορά το εργαλείο TEMPO. Πιο συγκεκριμένα:

- Η εντολή *choose ... where ...* δεν λειτουργεί όπως πρέπει και επιστρέφει αυθαίρετες τιμές. Για αυτό έπρεπε να υλοποιήσουμε εμείς τη λειτουργία αυτή, στον κώδικα Java που παράχθηκε από το εργαλείο TEMPO.
- Το σύμβολο του υπαρξιακού ποσοδείκτη (∃) δεν λειτουργούσε και δεν υπήρχε άλλος τρόπος για να γίνει ο έλεγχος αυτός σε ένα σύνολο (Set). Έτσι, χρειάστηκε να αντικαταστήσουμε όλα τα σύνολα (Set) με ακολουθία (Seq). Επίσης, για τον ίδιο λόγο οι εσωτερικές λειτουργίες gossip και gc του αυτόματου Replica, στη δική μας υλοποίηση εκτελούνται εάν η ακολουθία data δεν είναι κενή και αυτό προσθέτει καθυστέρηση στην υλοποίηση μας.
- Επειδή χρησιμοποιήσαμε κανάλια επικοινωνίας TCP δεν μπορούσαμε να τρέξουμε τον κώδικα μας τοπικά για αποσφαλμάτωση, αλλά από τα πρώτα στάδια χρειαστήκαμε συστοιχία υπολογιστών για να τρέχουμε εκεί τον κώδικα.
- Κάποια if statements δεν μεταφράζονταν σωστά και εκτελείτο ο κώδικας του if ενώ κανονικά δεν έπρεπε. Χρειάστηκε να εντοπίσουμε τα κομμάτια αυτά και να τα διορθώσουμε. Ένα τέτοιο if statement ήταν ο έλεγχος:

  ```
  if ((phase = 1) /\ (mid = val(m).query.mid)) then … fi
  ```

  Στο αυτόματο Client υπήρχαν έξι τέτοιοι ελέγχοι, όπου κάθε φορά άλλαζε ο αριθμός στον έλεγχο ισότητας της μεταβλητής phase. Επίσης, επειδή ο αντίστοιχος

κώδικας Java που πήραμε από το εργαλείο TEMPO ήταν περίπλοκος, δεν μπορούσε εύκολα να γίνει αποσφαλμάτωση σε ποιο σημείο του if statement υπήρχε το λάθος.

- Η αποσφαλμάτωση της προδιαγραφής ήταν δύσκολη διότι το αποτέλεσμα της εκτέλεσης ήταν αρκετά μεγάλο σε μέγεθος και έκανε τον εντοπισμό των σφαλμάτων αρκετά δύσκολο. Για το λόγο αυτό χρειάστηκε να το αποθηκεύουμε σε ένα αρχείο, για να είναι πιο εύκολο το ψάξιμο και να αφαιρέσουμε κάποια περιττά τυπώματα.

Όσον αφορά το κατανεμημένο δίκτυο PlanetLab, αντιμετωπίσαμε και εκεί κάποια προβλήματα, λόγω του ότι υπήρχε η πιθανότητα να καταρρεύσουν οι μηχανές. Επομένως, εάν λάμβανε χώρο μια κατάρρευση στη μηχανή *m*, για κάποιο χρονικό διάστημα ήταν αδύνατο να συνδεθούμε στη συγκεκριμένη μηχανή. Τέλος, σε κάποιες περιπτώσεις λόγω του ότι χρειαζόμασταν μεγάλο ποσοστό μνήμης, το οποίο μπορεί να μην ήταν διαθέσιμο εκείνη τη στιγμή, η διεργασία δεν ανταποκρινόταν πλέον και έπρεπε να αντικαταστήσουμε τη συγκεκριμένη μηχανή με κάποια άλλη.

6.3 Μελλοντική Εργασία

Σε αυτή τη Διπλωματική Εργασία έχει μελετηθεί και αξιολογηθεί ο αλγόριθμος LDR. Για περαιτέρω αξιολόγηση θα μπορούσαν να διεξαχθούν περισσότερα σενάρια με διαφορετικές παραμέτρους για να μας δοθεί η δυνατότητα να ελέγξουμε πως αντιδρά η υλοποίηση μας και σε άλλες περιπτώσεις. Επίσης, σε επόμενο στάδιο θα μπορούσε να δημιουργηθεί μια γραφική διεπαφή για να μπορεί ο χρήστης να εφαρμόζει από μόνος του κάποια σενάρια που επιθυμεί και να διαλέγει από τον τοπικό του υπολογιστή ποιο αρχείο επιθυμεί να γράψει, αλλά και να διαλέγει την τοποθεσία που θέλει να αποθηκευτεί το αρχείο που διαβάζει. Ακόμη, θα ήταν καλό να τυπώνονται τα αποτελέσματα, έτσι ώστε να είναι πιο εύκολο για εκείνο να βλέπει τη συμπεριφορά του αλγόριθμου. Στη συνέχεια, οι servers (ευρετήρια και αντιγραφείς) είναι stateless, δηλαδή δεν αποθηκεύουν τις αλληλεπιδράσεις με τους χρήστες και σε περίπτωση κατάρρευσης δεν μπορούν να ανακτήσουν τα δεδομένα τους. Επομένως, θα ήταν καλό να έχουμε ένα log file στο οποίο θα μπορεί να ανατρέξει για να μπει στην ίδια κατάσταση που ήταν πριν καταρρεύσει.

6.4 Οφέλη από τη Διπλωματική Εργασία

Με την ολοκλήρωση της παρούσας Διπλωματικής Εργασίας έχω αποκομίσει πολλά θετικά στοιχεία. Αρχικά, εξοικειώθηκα σε μεγάλο βαθμό με την ερευνητική μεθοδολογία και κατά δεύτερο απέκτησα πολλές γνώσεις σε τομείς της πληροφορικής που δεν είχα την ευκαιρία να μελετήσω μέχρι τώρα. Έγινε μελέτη σε θέματα που αφορούν τους κατανεμημένους αλγόριθμους και τα κατανεμημένα συστήματα, αλλά της λειτουργίας των Αυτομάτων Εισόδου/Εξόδου (IOA) και Χρονισμένων Αυτόματων Εισόδου/Εξόδου (TIOA). Πιο συγκεκριμένα μελετήθηκε η γλώσσα TIOA και το εργαλείο TEMPO, βάση των οποίων έγινε και η προδιαγραφή του αλγόριθμου. Μελετήθηκαν επίσης, θέματα που αφορούν τη κατανεμημένη διαχείριση αρχείων και ο αλγόριθμος LDR, τον οποίο υλοποιήσαμε στη παρούσα Διπλωματική Εργασία. Τέλος, μου δόθηκε η ευκαιρία να χρησιμοποιήσω το κατανεμημένο δίκτυο PlanetLab για την αξιολόγηση του αλγόριθμου.

Βιβλιογραφία

[1] Chryssis Georgiou, Peter M. Musial, and Christos Ploutarchou, Tempo-toolkit: Tempo to Java Translation Module, in Proc. of the 12th IEEE International Symposium on Network Computing and Applications (NCA 2013), accepted, Cambridge, MA, 2013.

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

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

[4] D. K. Kaynar, N. Lynch, R. Segala, F. Vaandrager. Timed I/O Automata: A Mathematical Framework for Modeling and Analyzing Real – Time Systems, 2003.

[5] Hagit Attiya, Amotz Bar-Noy, Danny Dolev. Sharing memory robustly in message-passing stystems. Journal of the ACM, 42(1):124-142, January 1995.

[6] Message Passing Interface, MPI Org. Available at http://mpj-express.org/.

[7] M. Demeti. Implementation and Experimental Evaluation of The Layered Data Replication Algorithm. Master Thesis, Computer Science Department, University of Cyprus. January 2013.

[8] N. Lynch, L. Michel, A. A. Shvartsman. Tempo: A Toolkit for The Timed Input/Output Automata Formalism. First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems, March 2009.

[9] N. Lynch, M. R. Tuttle. An Introduction to Input/Output Automata. Massachusetts Institute of Technology. November 18, 1988.

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

[11] Nancy Lynch, Alex Shvartsman. Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts. In Twenty-Seventh Annual International

Symposium on Fault-Tolerant Computing (FTCS'97), pages 272-281, Seattle, Washinghton, USA, June 1997, IEEE.

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

[13] PlanetLab Europe. Available at https://www.planet-lab.eu.

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

[15] R. Fan. Efficient Replication of Large Data Objects. Master Thesis, Computer Science and Engineering, Massachussetts Institute of Technology. February 2003.

[16] R. Fan, N. Lynch. Efficient Replication of Large Data Objects. MIT Computer Science and Artificial Intelligence Laboratory. 2003.

[17] Stephen J. Garland, Dilsum Kaynar, Nancy A. Lynch, Joshua A. Tauber, Mandana Vaziri. TIOA Tutorial. Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, MA, May 22, 2005.

[18] Transmission Control Protocol (TCP). http://www.ietf.org/rfc/rfc793.txt

[19] VeroModo, Inc. //http://www.veromodo.com.

[20] VeroModo, Inc. The Tempo User Interface. August 24, 2011.

[21] Λ. Παπαδόπουλος. Υλοποίηση και Πειραματική Αξιολόγηση Αλγόριθμου Κατανεμημένης Αποθήκευσης υπό Βυζαντινών Σφαλμάτων. Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου. Δεκέμβρης 2012.

[22] Ν. Νικολάου. Σημειώσεις Μαθήματος ΕΠΛ 432 – Κατανεμημένοι Αλγόριθμοι. Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου, 2013.

[23] Ιωάννης Χατζηγιαννάκης, Παύλος Σπυράκης, Χρήστος Ζαρολιάγκης. Σημειώσεις Μαθήματος Κατανεμημένα Συστήματα, Τμήμα Πληροφορικής, Πανεπιστήμιο Πάτρας, 2013.

[24] Χριστίνα Γεωργίου. Προδιαγραφή και Προσομοίωση Χρονισμένων Κατανεμημένων Αλγορίθμων Εκλογής Αρχηγού χρησιμοποιώντας το εργαλείο Tempo. Διπλωματική Εργασία, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου. 2013

Παράρτημα Α

Σε αυτό το παράρτημα υπάρχουν οι προδιαγραφές των καναλιών επικοινωνίας MPI και TCP στο εργαλείο TEMPO και στη 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 paxos_voc, mpi_status_voc
        types mpi_message : Tuple[data:Data,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 (SendMed.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;
```

Α.1.3 Προδιαγραφή MPI Receive Mediator (RecvMed.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 (Vocabulary.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
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
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 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);
```

A.2.3 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);
			print(msg);
			print(msg);
			print(msg);
			print(msg);
			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;
	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.2.4 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;
```

Παράρτημα Β

Σε αυτό το παράρτημα υπάρχουν οι προδιαγραφές των αυτόματων Vocabulary, Directory, Client, Replica και Composition που υλοποιήθηκαν στην παρούσα Διπλωματική Εργασία.

Β. 1 Προδιαγραφή αλγόριθμου LDR

Β.1.1 Προδιαγραφή Λεξιλόγιου (Vocabulary.tioa)

```
%%% TCPObjectsVoc
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
%%% Algorithm vocabs
vocabulary algorithm_voc
	imports TCPNodeVoc
	types tag : Tuple[num:Nat, id:Nat]
	types message : Tuple[type:String, S:Seq[Node], val: Tuple[actualData : Seq[Char],
metaData : Seq[Char]], t:tag, mid:Nat, rank : Nat]
end
vocabulary tcp_specific_voc
	imports algorithm_voc, TCPObjectsVoc, TCPNodeVoc
	types
		Chan_message : Tuple[query : message, 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 algorithm_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.1.2 Προδιαγραφή Directory (Directory.tioa)

```
automaton Directory (f:Nat)
signature
        input RECEIVE(m : Null[Chan_message])
        output SEND(m : Null[Chan_message])
        internal prep_messages
        output systemInitialize(C:Seq[Node],R:Seq[Node],D:Seq[Node],ip:Node,r:Nat)
states
        % indicates an empty sequence of clients IP addresses
        clients : Seq[Node] := {};
        % indicates an empty sequence of directories IP addresses
        directories : Seq[Node] := {};
        % indicates an empty sequence of replicas IP addresses
        replicas : Seq[Node] := {};
        % indicates the IP of this directory
        IP : Node := [0,0,0,0];
        % indicates the rank of this directory
        rank : Nat := 0;
        % indicates an empty sequence of chan messages
        msgs : Seq[Null[Chan_message]] := {};
        % indicates the tag associated with the latest value
        tag : tag := [0,0];
        % indicates a set of replicas with the latest value
        utd : Seq[Node] := {};
        % indicates that client can sent message type1=<rread,mid>
        sent_rread : Bool := false;
        % indicates that directory can sent message <rread-ok,S,t,mid>
        sent_rreadOK : Bool := false;
        % indicates that directory can sent message <wread-ok,t,mid>
        sent_wreadOK : Bool := false;
        % indicates that directory can sent message <rwrite-ok,mid>
        sent_rwriteOK : Bool := false;
        % indicates that directory can sent message <wwrite-ok,mid>
        sent_wwriteOK : Bool := false;
        % indicates a temp table for write operations
        arrayWrite : Seq[Nat] := {};
        % indicates a temp table for read operations
        arrayRead : Seq[Nat] := {};
        % The physcal clock of the automaton
        clock : AugmentedReal := 0;
        % indicates a temp seq with mids for write operations
        midWrite : Seq[Nat] := {};
        % indicates a temp seq with mids for read operations
        midRead : Seq[Nat] := {};
transitions
        output systemInitialize(C,R,D,ip,r)
        eff
```

```
		clients := C;
		directories := D;
		replicas := R;
		IP := ip;
		utd := replicas;
		rank := r;
	% receive message
	input RECEIVE(m)
	eff
		if (m ~= nil()) then
			if (val(m).query.type = "rread") then
				midRead := midRead |- val(m).query.mid;
				arrayRead := arrayRead |- val(m).query.rank;
				sent_rreadOK := true;
			else if (val(m).query.type = "wread") then
					midWrite := midWrite |- val(m).query.mid;
					arrayWrite := arrayWrite |- val(m).query.rank;
					sent_wreadOK := true;
				else if (val(m).query.type = "rwrite" \/ val(m).query.type =
"wwrite") then
						if (val(m).query.t = tag) then
							for n:Nat where n<len(val(m).query.S) do
								if (val(m).query.S[n] \notin
utd) then
									utd := utd |-
val(m).query.S[n];
								fi
							od
						else if (val(m).query.t.num > tag.num \/
(val(m).query.t.num = tag.num /\ val(m).query.t.id >= tag.id)) then
								if (len(val(m).query.S) >= f +
1) then
									utd := val(m).query.S;
									tag := val(m).query.t;
								fi
							fi
						fi
						if (val(m).query.type = "rwrite") then
							midRead := midRead |- val(m).query.mid;
							arrayRead := arrayRead |-
val(m).query.rank;
							sent_rwriteOK := true;
						else if (val(m).sender \in clients) then

							midWrite := midWrite |-
val(m).query.mid;
							arrayWrite := arrayWrite |-
val(m).query.rank;
							sent_wwriteOK := true;
							fi
						fi
					fi
				fi
			fi
		fi
	% send messages
	output SEND(m)
	pre
		msgs ~={};
		m = head(msgs);
	eff
		msgs:= tail(msgs);
	% prepare the messages
```

```
        internal prep_messages
        locals
                        t : Nat := 0;
        pre
                sent_rreadOK = true \/ sent_wreadOK = true \/ sent_rwriteOK = true \/
sent_wwriteOK = true;
        eff
                if (sent_rreadOK = true) then
                        for n:Nat where n < len(arrayRead) do
                                t := arrayRead[n];
                                msgs := msgs |- embed([["rreadOK", utd, [{},{}], tag,
midRead[n], rank], IP, clients[t]]);
                        od
                        arrayRead := {};
                        midRead := {};
                        sent_rreadOK := false;
                fi
                if (sent_wreadOK = true) then
                        for n:Nat where n < len(arrayWrite) do
                                t := arrayWrite[n];
                                msgs := msgs |- embed([["wreadOK", {}, [{},{}], tag,
midWrite[n],rank], IP, clients[t]]);
                        od
                        arrayWrite := {};
                        midWrite := {};
                        sent_wreadOK := false;
                fi
                if (sent_rwriteOK = true) then
                        for n:Nat where n < len(arrayRead) do
                                t := arrayRead[n];
                                msgs := msgs |- embed([["rwriteOK", {}, [{},{}], [0,0],
midRead[n], rank], IP, clients[t]]);
                        od
                        arrayRead := {};
                        midRead := {};
                        sent_rwriteOK := false;
                fi
                if (sent_wwriteOK = true) then
                        for n:Nat where n < len(arrayWrite) do
                                t := arrayWrite[n];
                                msgs := msgs |- embed([["wwriteOK", {}, [{},{}], [0,0],
midWrite[n], rank], IP, clients[t]]);
                        od
                        arrayWrite := {};
                        midWrite := {};
                        sent_wwriteOK := false;
                fi
        trajectories
        trajdef Time
                evolve d(clock) = 1;
```

B.1.3 Προδιαγραφή Client (Client.tioa)

```
automaton Client (f:Nat, t1:DiscreteReal)
signature
        input RECEIVE(m : Null[Chan_message])
        output SEND(m : Null[Chan_message])
        input read, write(v : Seq[Char])
        internal prep_messages, readTimeout
        output read_ok(v : Seq[Char]), write_ok
        output systemInitialize(C : Seq[Node], R : Seq[Node], D : Seq[Node], ip : Node,
r:Nat)
```

```
states
        % indicates an empty sequence of clients IP addresses
        clients : Seq[Node] := {};
        % indicates an empty sequence of directories IP addresses
        directories : Seq[Node] := {};
        % indicates an empty sequence of replicas IP addresses
        replicas : Seq[Node] := {};
        % indicates the IP of this client
        IP : Node := [0,0,0,0];
        % indicates the rank of this client
        rank : Nat := 0;
        % indicates an empty sequence of chan messages
        msgs : Seq[Null[Chan_message]] := {};
        % indicates the id of message that client sends with the request
        mid : Nat := 0;
        % indicates the tag associated with the latest value
        tag : tag := [0,0] ;
        % indicates a set of replicas with the latest value
        utd : Seq[Node] := {};
        acc : Seq[Node] := {};
        % indicates in which phase the client is
        phase : Int := 0;
        % indicates the value
        val: Tuple[actualData : Seq[Char], metaData : Seq[Char]] := [{}, {}];
        % indicates that client can sent message type1=<rread,mid>
        sent_rread : Bool := false;
        % indicates that client can sent message type2=<wread,mid>
        sent_wread : Bool := false;
        % indicates that client can sent message type3=<rwrite,utd,tag,mid>
        sent_rwrite : Bool := false;
        % indicates that client can sent message type4=<wwriteC,acc,tag,mid>
        sent_wwrite : Bool := false;
        % indicates that client can sent message type9=<write,val,tag,mid>
        sent_write : Bool := false;
        % indicates that client can sent message type11=<read,tag,mid>
        sent_read : Bool := false;
        % indicates that client can sent message type15=<secure,tag,mid>
        sent_secure : Bool := false;
        % indicates the IP of the sender
        senderIP : Node := [0,0,0,0];
        maxTag : tag := [0,0];
        % The physcal clock of the automaton
        clock : AugmentedReal := 0;
        % The physical clock for the process
        readClock : AugmentedReal := 0;
transitions
        output systemInitialize(C,R,D,ip,r)
        eff
                clients := C;
                directories := D;
                replicas := R;
                IP := ip;
                rank := r;
        % read invokation
        input read
        eff
                mid := mid + 1;
                sent_rread := true;
        % write invokation
        input write(v)
        eff
                val.actualData := v;
                mid := mid + 1;
```

```
                sent_wread := true;
        % the read operation finished
        output read_ok(v)
        pre
                v = val.actualData /\ phase = 4;
        eff
                phase := 0;
        % the write operation finished
        output write_ok
        pre
                phase = 8;
        eff
                phase := 0;
        % send messages
        output SEND(m)
        pre
                msgs ~={};
                m = head(msgs);
        eff
                msgs:= tail(msgs);
                if (val(m).query.type = "read") then
                        readClock := clock;
                fi

        % internal read timeout
        internal readTimeout
        locals
                tempRank : Nat := 0;
        pre
                 readClock ~= 0 /\ clock > readClock + t1;
        eff
                if (len(utd) > 0) then
                        tempRank := choose t where (t>=len(utd) /\ t<=len(utd));
                        msgs := msgs |- embed([["read", {}, [{}, {}], tag, mid, rank],IP,
utd[tempRank]]);
                fi
                readClock := 0;

        % receive message
        input RECEIVE(m)
        eff
                if (m ~= nil()) then
                        mid := val(m).query.mid;
                        senderIP := val(m).sender;
                        if (val(m).query.type = "rreadOK") then
                                if ((phase = 1) /\ (mid = val(m).query.mid)) then
                                        if (val(m).sender \notin acc) then
                                                acc := acc |- val(m).sender;
                                        fi
                                        if (val(m).query.t.num > tag.num \/ (val(m).query.t.num
= tag.num /\ val(m).query.t.id >= tag.id)) then
                                                tag := val(m).query.t;
                                                utd := val(m).query.S;
                                        fi
                                        if (len(acc) >= (div(len(directories),2) + 1)) then
                                                mid := mid + 1;
                                                sent_rwrite := true;
                                        fi
                                fi
                        fi
                        if (val(m).query.type = "rwriteOK") then
                                if ((phase = 2) /\ (mid = val(m).query.mid)) then
                                        if (val(m).sender \notin acc) then
```

```
                                        acc := acc |- val(m).sender;
                                fi
                                if (len(acc) >= (div(len(directories),2) + 1)) then
                                        mid := mid + 1;
                                        sent_read := true;
                                fi
                        fi
                fi
                if (val(m).query.type = "readOK") then
                        if ((phase = 3) /\ (mid = val(m).query.mid)) then
                                val := val(m).query.val;
                                tag := val(m).query.t;
                                phase := 4;
                                readClock := 0;
                        fi
                fi
                if (val(m).query.type = "wreadOK") then
                        if ((phase = 5) /\ (mid = val(m).query.mid)) then
                                if (val(m).sender \notin acc) then
                                        acc := acc |- val(m).sender;
                                fi
                                maxTag := tag;
                                if (val(m).query.t.num > maxTag.num \/
(val(m).query.t.num = maxTag.num /\ val(m).query.t.id >= tag.id)) then
                                        maxTag := val(m).query.t;

                                fi
                                if (len(acc) >= (div(len(directories),2) + 1)) then
                                        mid := mid + 1;
                                        sent_write := true;
                                fi
                        fi
                fi
                if (val(m).query.type = "writeOK") then
                        if ((phase = 6) /\ (mid = val(m).query.mid)) then
                                if (val(m).sender \notin acc) then
                                        acc := acc |- val(m).sender;
                                fi
                                if (len(acc) > f) then
                                        mid := mid + 1;
                                        sent_wwrite := true;
                                fi
                        fi
                fi
                if (val(m).query.type = "wwriteOK") then
                        if ((phase = 7) /\ (mid = val(m).query.mid)) then
                                if (val(m).sender \notin acc) then
                                        acc := acc |- val(m).sender;
                                fi
                                if (len(acc) >= (div(len(directories),2) + 1)) then
                                        mid := mid +1;
                                        sent_secure := true;
                                fi
                        fi
                fi
        fi
    % prepare the messages
    internal prep_messages
    locals
        tempRank : Nat := 0;
    pre
        sent_rread = true \/ sent_wread = true  \/ sent_rwrite = true \/ sent_wwrite =
true \/ sent_write = true \/ sent_read = true \/ sent_secure = true;
```

```
        eff
                if (sent_rread = true) then
                        for n:Nat where(n < len(directories)) do
                                msgs := msgs |- embed([["rread", {}, [{}, {}], [0,0], mid,
rank],IP, directories[n]]);
                        od
                        sent_rread := false;
                        phase := 1;
                fi
                if (sent_wread = true) then
                        for n:Nat where(n < len(directories)) do
                                msgs := msgs |- embed([["wread", {}, [{}, {}], [0,0],
mid,rank], IP, directories[n]]);
                        od
                        sent_wread := false;
                        phase := 5;
                fi
                if (sent_rwrite = true) then
                        for n:Nat where(n < len(directories)) do
                                msgs := msgs |- embed([["rwrite", utd, [{}, {}], tag, mid,
rank],IP, directories[n]]);
                        od
                        acc := {};
                        sent_rwrite := false;
                        phase := 2;
                fi
                if (sent_wwrite = true) then
                        for n:Nat where(n < len(directories)) do
                                msgs := msgs |- embed([["wwrite", acc, [{}, {}], tag, mid,
rank],IP, directories[n]]);
                        od
                        acc := {};
                        sent_wwrite := false;
                        phase := 7;
                fi
                if (sent_write = true) then
                        if (len(acc) >= (div(len(directories),2) + 1)) then
                                tag.num := maxTag.num + 1;
                                tag.id := rank;
                        fi
                        for n:Nat where(n < len(replicas)) do
                                msgs := msgs |- embed([["write", {}, val, tag, mid, rank],IP,
replicas[n]]);
                        od
                        acc := {};
                        maxTag := [0,0];
                        sent_write := false;
                        phase := 6;
                fi
                if (sent_read = true) then
                        if (len(utd) > 0) then
                                tempRank := choose t where (t>=len(utd) /\ t<=len(utd));
                                msgs := msgs |- embed([["read", {}, [{}, {}], tag, mid,
rank],IP, utd[tempRank]]);
                                acc := {};
                                sent_read := false;
                                phase := 3;
                        fi
                fi
                if (sent_secure = true) then
                        for n:Nat where(n < len(replicas)) do
                                msgs := msgs |- embed([["secure", {}, [{}, {}], tag, mid,
rank],IP, replicas[n]]);
```

```
                od
                acc := {};
                sent_secure:= false;
                phase := 8;
        fi
    trajectories
    trajdef Time
        evolve d(clock) = 1;
```

B.1.3 Προδιαγραφή Replica (Replica.tioa)

```
automaton Replica (f:Nat)
signature
    input RECEIVE(m : Null[Chan_message])
    output SEND(m : Null[Chan_message])
    internal prep_messages
    output systemInitialize(C : Seq[Node], R : Seq[Node], D : Seq[Node], ip : Node, r
:Nat)
    internal gossip, gc
states
    % indicates an empty sequence of clients IP addresses
    clients : Seq[Node] := {};
    % indicates an empty sequence of directories IP addresses
    directories : Seq[Node] := {};
    % indicates an empty sequence of replicas IP addresses
    replicas : Seq[Node] := {};
    % indicates the IP of this replica
    IP : Node := [0,0,0,0];
    % indicates the rank of this replica
    rank : Nat := 0;
    % an empty sequence of chan messages
    msgs : Seq[Null[Chan_message]] := {};
    % indicates the set of tuple[value-tag-bit] that stores a replica
    data : Seq[Tuple[value: Tuple[actualData : Seq[Char], metaData : Seq[Char]], tag :
tag, bit : Nat]] := {};
    % indicates that replica can sent message <write-ok,mid>
    sent_writeOK : Bool := false;
    % indicates that replica can sent message type=12 <read-ok,v,t,mid>
    sent_readOK : Bool := false;
    % indicates that replica can sent message type=13 <gossip,v,t>
    sent_gossip : Bool := false;
    % indicates that replica can sent message type=14 <wwriteR,{i},t>
    sent_wwrite : Bool := false;
    % indicates temp data
    tempData : Tuple[value: Tuple[actualData : Seq[Char], metaData : Seq[Char]], tag :
tag] := [[{},{}], [0,0]];
    tempDataRead : Tuple[value: Tuple[actualData : Seq[Char], metaData : Seq[Char]], tag :
tag] := [[{},{}],[0,0]];
    % indicates temp data
    tempTag : tag := [0,0];
    % indicates a temp table for write operations
    arrayWrite : Seq[Nat] := {};
    % indicates a temp table for read operations
    arrayRead : Seq[Nat] := {};
    % The physcal clock of the automaton
    clock : AugmentedReal := 0;
    start_gossip : Bool := false;
    midWrite : Seq[Nat] := {};
    midRead : Seq[Nat] := {};
transitions
    output systemInitialize(C,R,D,ip,r)
    locals
```

```
		temp : Seq[Char] := {};
	eff
		clients := C;
		directories := D;
		replicas := R;
		temp := temp |- '0';
		data := data |- [[temp,{}],[0,0],1];
		IP := ip;
		rank := r;
	% send messages
	output SEND(m)
	pre
		msgs ~={};
		m = head(msgs);
	eff
		msgs:= tail(msgs);

	input RECEIVE(m)
	locals
		f : Bool := false;
		go : Int := 0;
		s1 : Bool := false;
		s2 : Bool := false;
		s3 : Bool := false;
		maxTag : tag := [0,0];
		maxValue : Seq[Char] := {};
		position : Nat := 0;
		found : Array[Int,Bool] := constant(false:Bool);
		temp : Seq[Tuple[value : Tuple[actualData : Seq[Char], metaData : Seq[Char]],
tag : tag, bit : Nat]] := {};
	eff
		if (m ~= nil()) then
			if (val(m).query.type = "read") then
				arrayRead := arrayRead |- val(m).query.rank;
				midRead := midRead |- val(m).query.mid;
				for b:Nat where b < len(data) do
					if (data[b].tag.num = val(m).query.t.num /\
data[b].tag.id = val(m).query.t.id /\ f = false) then
						tempDataRead := [[data[b].value.actualData,{}],
data[b].tag];
						f := true;
					fi
				od
				if (f = false) then
					for k:Nat where k < len(data) do
						if (data[k].tag.num > maxTag.num /\
data[k].tag.id > maxTag.id) then
							maxTag := data[k].tag;
							maxValue := data[k].value.actualData;
						fi
					od
					tempDataRead := [[maxValue,{}], maxTag];
				else
					f := false;
				fi
				sent_readOK := true;
			fi
			if (val(m).query.type = "write") then
				for k:Nat where k<len(data) do
					if (data[k].tag.num = val(m).query.t.num /\
data[k].tag.id < val(m).query.t.id /\ data[k].bit = 0) then
						found[k] := true;
						s1 := true;
```

```
                                        else if (data[k].tag.num = val(m).query.t.num /\
data[k].tag.id > val(m).query.t.id) then
                                                if (data[k].bit = 0) then
                                                        s3 := true;
                                                else
                                                        s2 := false;
                                                fi
                                        else if (data[k].tag.num < val(m).query.t.num )
then
                                                        s2 := true;
                                                fi
                                        fi
                                fi
                        od
                        if (s1 = true) then
                                for k:Nat where k<len(data) do
                                        if (found[k] = false) then
                                                temp := temp |- data[k];
                                        fi
                                od
                                data := {};

                                for k:Nat where k<len(temp) do
                                        data := data |- temp[k];
                                od
                        fi
                        if (s1 = true \/ s2 = true /\ s3 = false ) then
                                data := data |- [val(m).query.val, val(m).query.t, 0];
                                arrayWrite := arrayWrite |- val(m).query.rank;
                                midWrite := midWrite |- val(m).query.mid;
                                sent_writeOK := true;
                        fi
                        s1 := false;
                        s2 := false;
                        s3 := false;
                        found := constant(false:Bool);
                        f := false;
                fi
                if (val(m).query.type = "gossip") then
                        for t:Nat where t<len(data) do
                                if (data[t].value.actualData =
val(m).query.val.actualData /\ data[t].tag = val(m).query.t) then
                                        position := t;
                                        if (data[t].bit = 0) then
                                                go := 1;
                                        else
                                                go := 2;
                                        fi
                                fi
                        od
                        if (go = 1) then
                                data := eject(data,position);
                        fi
                        if (go ~= 2) then
                                data := [val(m).query.val, val(m).query.t, 1] -| data;
                                tempTag := data[0].tag;
                                go := 0;
                                position := 0;
                                sent_wwrite := true;
                        fi
                fi
                if (val(m).query.type = "secure") then
                        for b:Nat where b < len(data) do
```

```
                                if (data[b].tag = val(m).query.t /\ data[b].bit = 0) then
                                        data[b].bit := 1;
                                fi
                        od
                fi
        fi
    % prepare the messages which directory sends
    internal prep_messages
    locals
        t : Nat := 0;
        tempSeq : Seq[Node] := {};
    pre
        sent_writeOK = true \/ sent_readOK = true \/ sent_wwrite = true \/ sent_gossip
= true;
    eff
        if (sent_writeOK = true) then
            for n:Nat where n < len(arrayWrite) do
                t := arrayWrite[n];
                msgs := msgs |- embed([["writeOK", {}, [{},{}],[0,0],
midWrite[n],rank], IP, clients[t]]);
            od
            arrayWrite := {};
            midWrite := {};
            sent_writeOK := false;
            start_gossip := true;
        fi
        if (sent_readOK = true) then
            for n:Nat where n < len(arrayRead) do
                t := arrayRead[n];
                msgs := msgs |- embed([["readOK", {}, tempDataRead.value,
tempDataRead.tag, midRead[n], rank],IP, clients[t]]);
            od
            arrayRead := {};
            midRead := {};
            tempDataRead :=  [[{},{}],[0,0]];
            sent_readOK := false;
        fi
        if (sent_wwrite = true) then
            tempSeq := tempSeq |- IP;
            for n:Nat where(n < len(directories)) do
                msgs := msgs |- embed([["wwrite", tempSeq , [{},{}], tempTag,
0,rank], IP, directories[n]]);
            od
            tempTag := [0,0];
            tempSeq := {};
            sent_wwrite := false;
        fi
        if (sent_gossip = true) then
            for n:Nat where(n < len(replicas)) do
                msgs := msgs |- embed([["gossip", {}, tempData.value,
tempData.tag, 0, rank],IP, replicas[n]]);
            od
            tempData := [[{},{}],[0,0]];
            sent_gossip := false;
        fi
    internal gossip
    locals
        found : Bool := false;
    pre
        data ~= {} /\ start_gossip = true;
    eff
        for k:Nat where k<len(data) do
            if([data[k].value, data[k].tag, 1] \in data /\ found = false) then
```

```
                        tempData := choose l where([l.value, l.tag,1] \in data);
                        found := true;
                fi
        od
        if (found = true) then
                sent_gossip := true;
                found := false;
        fi
        start_gossip := false;
internal gc
locals
        found : Bool := false;
pre
        data ~= {};
eff
        for k:Nat where k<len(data) do
                if([data[k].value, data[k].tag, 1] \in data /\ found = false) then
                        tempData := choose l where([l.value, l.tag,1] \in data);
                        print(found);
                        found := true;
                fi
        od
        if (found = true) then
                print(found);
                for k:Nat where k<len(data) do
                        if (data[k].tag.num < tempData.tag.num) then
                                data := eject(data,k);
                        fi
                od
                tempData := [[{},{}],[0,0]];
                found := false;
        fi
trajectories
trajdef Time
        evolve d(clock) = 1;
```

B.1.4 Προδιαγραφή Composition (Composition.tioa)

```
%%% .: Vocabulary :.
include "Vocabulary.tioa"
imports algorithm_voc, TCPObjectsVoc, TCPNodeVoc, tcp_specific_voc, JVMSocket,
JVMServerSocket, ChannelVoc
%%% .: TCP Mediator Automata :.
include "TCPRecvMed.tioa"
include "TCPSendMed.tioa"
include "TCPChanMed.tioa"
include "Client.tioa"
include "Directory.tioa"
include "Replica.tioa"
automaton Composition(n1:Nat, n2:Nat, n3:Nat, n4:Nat, port:Nat, timeout:Nat, t1:DiscreteReal,
f:Nat, operation:Nat, writeNum:Seq[Char], numberOfOperations:Int)
components
        C : Client(f, t1);
        D : Directory(f);
        R : Replica(f);
        SM : SendMed(port, timeout);
        RM : RecvMed(port, timeout);
        CM : ChanMed(port, timeout);
schedule
        states
                IP : Node := [n1,n2,n3,n4];
                clients : Seq[Node] := {};
```

```
                replicas : Seq[Node] := {};
                directories : Seq[Node] := {};
                ms : Null[Chan_message] := nil();
                mr : Null[Chan_message] := nil();
                dowhile : Bool := true;
                isConn : Bool := false;
                error : Null[JVMError] := nil();
                runs : Nat := 0;
                world : Seq[Node] := {};
                val : Seq[Char] := {};
        do
                %% INITIALIZE
                clients := clients |- [10,16,12,230];
                clients := clients |- [10,16,12,144];
                clients := clients |- [10,16,12,227];
                directories := directories |- [10,16,12,116];
                directories := directories |- [10,16,12,118];
                directories := directories |- [10,16,12,240];
                replicas := replicas |- [10,16,12,213];
                replicas := replicas |- [10,16,12,119];
                replicas := replicas |- [10,16,12,169];
                world := world || clients ;
                world := world || directories;
                world := world || replicas;
                runs := 10;
                %% triger initialization of all component
                for k:Nat where k < len(clients) do
                        if (IP = clients[k]) then
                                fire output C.systemInitialize(clients, replicas, directories,
clients[k],k);
                        fi
                od
                for k:Nat where k < len(directories) do
                        if (IP = directories[k]) then
                                fire output D.systemInitialize(clients, replicas, directories,
directories[k],k);
                        fi
                od
                for k:Nat where k < len(replicas) do
                        if (IP = replicas[k]) then
                                fire output R.systemInitialize(clients, replicas, directories,
replicas[k],k);
                        fi
                od
                %% bind to server socket
                %% everyone sets up their server socket
                fire output RM.TCP_bind(IP);
                fire output CM.TCP_respBind(error, IP);
                %% 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 len(world)*100;
                        %% 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      %end of for
if (IP \in clients) then
        if (operation = 0) then
                fire input C.read;
        else
                fire input C.write(writeNum);
        fi
fi
for i : Nat where i < runs do
        if (IP \in replicas) then
                follow R.Time duration 1;
        fi
        if (IP \in clients) then
                follow C.Time duration 1;
        fi
        if (IP \in directories) then
                follow D.Time duration 1;
        fi

        if (IP \in clients) then
                fire internal C.readTimeout;
        fi

        if (IP \in clients) then
                fire internal C.prep_messages;
        else if (IP \in directories) then
                        fire internal D.prep_messages;
                else if (IP \in replicas) then
                                fire internal R.prep_messages;

                        fi
                fi
        fi
        % SEND
        for j : Nat where j < len(world) do
                ms := nil();
                if (IP \in clients) then
                        fire output C.SEND(ms);
                else if (IP \in directories) then
                                fire output D.SEND(ms);
                        else if (IP \in replicas) then
                                        fire output R.SEND(ms);
                                fi
                        fi
                fi
                if (ms ~= nil()) then
        fire output SM.TCP_write(ms,val(ms).sender,val(ms).destination);
                        print val(ms).sender;
                fi
        od
        follow SM.DELAY duration 200;
% -- extract messages from channel if there are any
        fire output RM.TCP_read;
        for j : Nat where j < len(world) do
                dowhile := true;
        while( dowhile = true ) do
                fire output CM.TCP_respRead( ms );
                if (ms ~= nil()) then
                        fire output RM.RECEIVE( ms );
                        print val(ms);
                else
```

```
                                        dowhile := false;
                                         fi
                                od
                        od
                        if (operation = 0) then
                                fire output C.read_ok(val);
                                print val;
                        else
                                fire output C.write_ok;
                                if (IP \in replicas) then
                                        fire internal R.gc;
                                        fire internal R.gossip;
                                fi
                        fi
                od %end of for
od
```