Ατομική Διπλωματική Εργασία

ΠΡΟΔΙΑΓΡΑΦΗ ΚΑΙ ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΤΗΣ ΚΑΤΑΝΕΜΗΜΕΝΗΣ ΥΠΗΡΕΣΙΑΣ ΓΡΑΦΗΣ/ΑΝΑΓΝΩΣΗΣ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ DYNASTORE ΜΕ ΤΗ ΧΡΗΣΗ ΤΟΥ ΕΡΓΑΛΕΙΟΥ TEMPO

Ανδρέας Παπανδρέου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2015

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΡΟΔΙΑΓΡΑΦΗ ΚΑΙ ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΤΗΣ ΚΑΤΑΝΕΜΗΜΕΝΗΣ ΥΠΗΡΕΣΙΑΣ ΓΡΑΦΗΣ/ΑΝΑΓΝΩΣΗΣ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ DYNASTORE ΜΕ ΤΗ ΧΡΗΣΗ ΤΟΥ ΕΡΓΑΛΕΙΟΥ TEMPO

Ανδρέας Παπανδρέου

Επιβλέπων Καθηγητής

Χρύσης Γεωργίου

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

Μάιος 2015

Ευχαριστίες

Θα ήθελα να ευχαριστήσω όλους όσους συνέλαβαν με οποιονδήποτε τρόπο στην ολοκλήρωση αυτής της διπλωματικής εργασίας. Θερμές ευχαριστίες στον καθηγητή μου Δρ. Χρύση Γεωργίου για την επίβλεψη αυτής της διπλωματικής εργασίας, και την ευκαιρία που μου έδωσε να ασχοληθώ με το συγκεκριμένο θέμα. Πάντα ήταν πρόθυμος και διαθέσιμος να μου προσφέρει τις γνώσεις του, καθώς και την βοήθεια του σε κάθε απορία μου σχετικά με την εργασία αυτή.

Τέλος, το μεγαλύτερο ευχαριστώ το οφείλω στους γονείς μου, στους οποίους και αφιερώνω την παρούσα εργασία, για την πίστη στις δυνατότητές μου και τη συμπαράσταση που μου προσφέρουν όλα αυτά τα χρόνια.

Περίληψη

Ένα Κατανεμημένο Σύστημα Αποθήκευσης διασφαλίζει την βιωσιμότητα των δεδομένων του με την χρήση της μεθόδου αναπαραγωγής. Συγκεκριμένα, αντίγραφα των δεδομένων διατηρούνται σε πολλούς εξυπηρετητές του συστήματος και η συνέπεια τους επιτυγχάνεται με την ανταλλαγή μηνυμάτων. Ωστόσο, η εφαρμογή αυτής της μεθόδου αποτελεί πρόκληση σε ένα σύστημα το οποίο χαρακτηρίζεται από συχνές αλλαγές στους εξυπηρετητές του, λόγω καταρρεύσεων. Σε ένα τέτοιο σύστημα, το σύνολο των εξυπηρετητών πρέπει να αναδιαμορφώνεται, χωρίς όμως να χάνεται η συνέπεια των δεδομένων του. Το σύστημα Dynastore, παρέχει μια λύση στο πρόβλημα διατήρησης της συνέπειας των δεδομένων παράλληλα με την αναδιαμόρφωση. Αποτελείται από δύο κύριες υπηρεσίες. Την υπηρεσία γραφής/ανάγνωσης κατανεμημένων ατομικών αντικειμένων και την υπηρεσία αναδιαμόρφωσης των εξυπηρετητών που διατηρούν αντίγραφο των δεδομένων. Στην παρούσα διπλωματική εργασία θα ασχοληθούμε με την υπηρεσία γραφής/ανάγνωσης κατανεμημένων αντικειμένων. Στόχος ήταν η προδιαγραφή μέσω του εργαλείου TEMPO το οποίο παρέχει αυτοματοποιημένο τρόπο υλοποίησης, διατηρώντας την ορθότητα της. Έγινε αξιολόγηση της υλοποίησης σε τοπικό δίκτυο του Πανεπιστημίου Κύπρου και έπειτα στο δίκτυο PlanetLab.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός και Κίνητρο

Ένα Κατανεμημένο Σύστημα Αποθήκευσης [16] αποτελεί μια συλλογή από ανεξάρτητους υπολογιστές οι οποίοι δίνουν στους χρήστες την εντύπωση ενός ενιαίου συνεκτικού συστήματος. Σε ένα τέτοιο σύστημα η βιωσιμότητα των δεδομένων διασφαλίζεται με την χρήση της μεθόδου αναπαραγωγής. Συγκεκριμένα, αντίγραφο των δεδομένων του συστήματος διατηρείται σε πολλούς εξυπηρετητές και η συνέπεια του επιτυγχάνεται με την ανταλλαγή μηνυμάτων. Ωστόσο, η εφαρμογή αυτής της μεθόδου αποτελεί πρόκληση σε ένα δυναμικό σύστημα το οποίο χαρακτηρίζεται από συχνές αλλαγές στους εξυπηρετητές του, λόγω καταρρεύσεων. Σε ένα τέτοιο σύστημα, το σύνολο των εξυπηρετητών πρέπει να αναδιαμορφώνεται, χωρίς όμως να χάνεται η συνέπεια των δεδομένων του. Το σύστημα Dynastore [7, 8], παρέχει μια λύση στο πρόβλημα διατήρησης της συνέπειας των δεδομένων παράλληλα με την αναδιαμόρφωση. Το σύστημα αποτελείται από την υπηρεσία γραφής/ανάγνωσης κατανεμημένων ατομικών αντικειμένων και την υπηρεσία αναδιαμόρφωσης των εξυπηρετητών που διατηρούν αντίγραφο των δεδομένων. Σκοπός της παρούσας διπλωματικής εργασίας είναι η μελέτη, προδιαγραφή, υλοποίηση και αξιολόγηση της υπηρεσίας γραφής/ανάγνωσης κατανεμημένων αντικειμένων και συγκεκριμένα τον αλγόριθμο Reader-Writer-Recon. Η προδιαγραφή έγινε με την χρήση του μοντέλου Χρονισμένων Αυτομάτων Εισόδου/Εξόδου [2, 3, 5, 16] το οποίο αποτελεί μία από τις δημοφιλέστερες μεθόδους προδιαγραφής κατανεμημένων αλγόριθμων. Έγινε χρήση της γλώσσας TIOA [2, 3, 5] και του εργαλείου TEMPO [2, 3, 5], το οποίο παρέχει αυτοματοποιημένο τρόπο υλοποίησης διατηρώντας την ορθότητα της προδιαγραφής [1]. Η αξιολόγηση της υλοποίησης έγινε σε τοπικό δίκτυο του Πανεπιστημίου Κύπρου

και έπειτα στο ρεαλιστικό κατανεμημένο δίκτυο PlanetLab [10, 12] για μελέτη της ευρωστίας του αλγόριθμου σε δυσμενείς συνθήκες αλλά και την διαπίστωση της ορθότητας της υλοποίησης.

1.2 Μεθοδολογία Υλοποίησης

Για την ολοκλήρωση της Διπλωματικής Εργασίας ακολουθήθηκε η εξής μεθοδολογία: Αρχικά, έγινε εκμάθηση του γενικού μοντέλου Αυτόματων Εισόδου/Εξόδου (IOA) [6, 9] με ιδιαίτερη έμφαση σε ένα συγκεκριμένο τύπο IOA, τα Χρονισμένα Αυτόματα Εισόδου/Εξόδου (TIOA) [2, 3, 5]. Έπειτα από μελέτη και προδιαγραφή απλών παραδειγμάτων για καλύτερη κατανόηση του μοντέλου ακολούθησε η εγκατάσταση και εκμάθηση του εργαλείου TEMPO [1, 4, 13]. Η εξοικείωση με το εργαλείο επιτεύχθηκε μέσω της υλοποίησης διάφορων παραδειγμάτων. Ωστόσο, κρίθηκε αναγκαία και η απόκτηση θεωρητικού υπόβαθρου για τα κατανεμημένα συστήματα και κατανεμημένους αλγόριθμους. Αποκτώντας λοιπόν τις απαραίτητες βάσεις, ακολούθησε η εκτενής μελέτη και κατανόηση του κατανεμημένου αλγόριθμου Dynastore και διαχωρισμός του σε δύο κύρια μέρη, WeakSnapShot-ABD και Reader-Writer-Recon [7, 8]. Η παρούσα Διπλωματική Εργασία αφορά αλγόριθμο Reader-Writer-Recon που αποτελεί το μέρος του αλγόριθμου που προσφέρει την κατανεμημένη υπηρεσία γραφής/ανάγνωσης ατομικών αντικειμένων. Η προδιαγραφή έγινε σε γλώσσα TIOA [2, 3, 5] μέσω του εργαλείου TEMPO. Παράλληλα, χρειάστηκε η κατανόηση των καναλιών επικοινωνίας MPI [11] και TCP [17], απαραίτητα για την υλοποίηση του αλγόριθμου αλλά και των παραδειγμάτων που μελετήθηκαν. Την υλοποίηση του αλγορίθμου ακολούθησε η αποσφαλμάτωση και πειραματική αξιολόγηση και συλλογή αποτελεσμάτων αρχικά σε συστοιχία μηχανών (cluster) στο τοπικό δίκτυο του Πανεπιστημίου Κύπρου και σε μετέπειτα στάδιο στο κατανεμημένο δίκτυο PlanetLab. Απαιτήθηκε όμως η μελέτη της πλατφόρμας του PlanetLab για την ορθή χρήση των υπηρεσιών που παρέχει [10, 12].

1.3 Δομή Διπλωματικής Εργασίας

Στο Κεφάλαιο 2 που ακολουθεί παρουσιάζεται το γνωστικό υπόβαθρο, απαραίτητο για την κατανόηση των αυτόματων IOA και TIOA καθώς και του εργαλείου TEMPO [1, 4,

13] και της γλώσσας TIOA [2, 3, 5]. Παρουσιάζονται τα κανάλια επικοινωνίας TCP και MPI και γίνεται αναφορά στους όρους ατομικότητα, συστήματα απαρτίας, συστήματα κατανεμημένης κοινόχρηστης μνήμης και καθυστέρηση λειτουργίας. Επιπλέον περιγράφεται το κατανεμημένο δίκτυο PlanetLab[10, 12] και διαδικασία σύνδεσης σε αυτό. Στο Κεφάλαιο 3 γίνεται αναλυτική περιγραφή του αλγόριθμου Reader-Writer-Recon και της λειτουργίας του. Η προδιαγραφή του αλγόριθμου σε γλώσσα TIOA [2] παρατίθεται στο Κεφάλαιο 4. Ακολουθεί η πειραματική αξιολόγηση του αλγόριθμου στο κατανεμημένο δίκτυο PlanetLab, Κεφάλαιο 5. Τέλος, στο Κεφάλαιο 6 γίνεται αναφορά στα συμπεράσματα από την πειραματική αξιολόγηση, τυχόν προβλήματα που έχουν αντιμετωπιστεί κατά τη διάρκεια της Διπλωματικής Εργασίας καθώς και πιθανή μελλοντική εργασία που πηγάζει από την εργασία αυτή.

Κεφάλαιο 2

Γνωστικό Υπόβαθρο

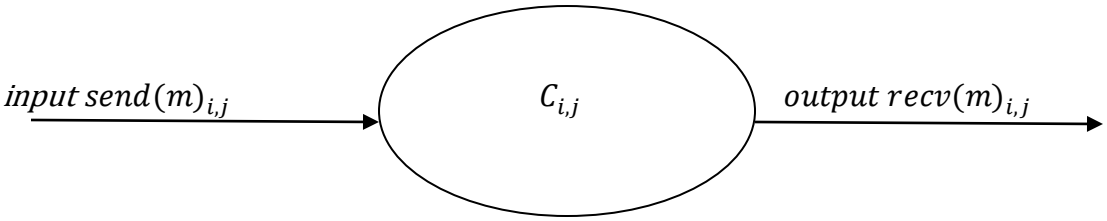
2.1 Μοντέλο Αυτομάτων Εισόδου/ Εξόδου

Το μοντέλο Αυτομάτων Εισόδου / Εξόδου (Input / Output Automata – IOA) [6], είναι κατάλληλο για τη μοντελοποίηση ασύγχρονων κατανεμημένων συστημάτων. Αναπτύχθηκε για πρώτη φορά από τους Nancy A. Lynch και Mark R. Tuttle, στο Massachusetts Institute of Technology [6]. Το μοντέλο είναι κατάλληλο για τη μοντελοποίηση συστημάτων των οποίων τα συστατικά λειτουργούν ασύγχρονα, όπως αλγόριθμοι επικοινωνίας, αλγόριθμοι κατανομής πόρων σε δίκτυο και διαχείριση κοινόχρηστων ατομικών αντικειμένων. Συγκεκριμένα, ένα σύστημα διασπάται σε συστατικά τα οποία μοντελοποιούνται ξεχωριστά ως Input/Output αυτόματα και έχουν την δυνατότητα να αλληλεπιδρούν μεταξύ τους. Το μοντέλο, αποτελεί μια απλή μηχανή καταστάσεων (states) όπου κάθε ενέργεια (action) ακολουθείται από μία μετάβαση (transition) από την παρούσα κατάσταση σε επόμενη.

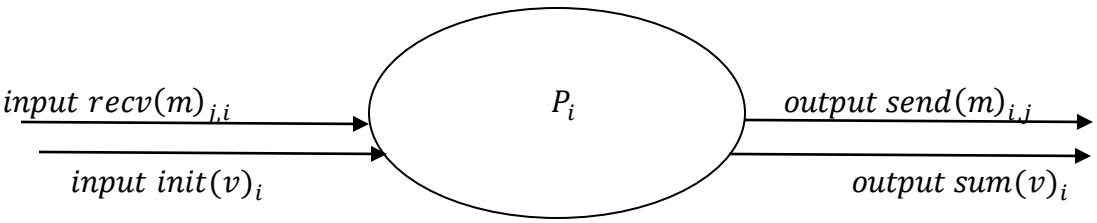
Μια θεμελιώδης ιδιότητα του μοντέλου αποτελεί ο σαφής διαχωρισμός ανάμεσα στις ενέργειες του μοντέλου ανάμεσα σε αυτές που η εκτέλεση τους είναι υπό τον έλεγχο του αυτόματου και σε εκείνες που ελέγχονται από το περιβάλλον του. Συγκεκριμένα οι ενέργειες ενός IOA[6] κατατάσσονται σε τρεις κατηγορίες:

- Ενέργειες Εισόδου (Input)
- Ενέργειες Εξόδου (Output)
- Εσωτερικές Ενέργειες (Internal)

Ένα αυτόματο δημιουργεί εσωτερικές ενέργειες και ενέργειες εξόδου αυτόνομα, και διαδίδει την έξοδο του στιγμιαία στο περιβάλλον. Παρόμοια, η είσοδος ενός αυτόματου δημιουργείται από το περιβάλλον και μεταδίδεται στιγμιαία στο αυτόματο. Ο διαχωρισμός των ενεργειών εισόδου και των υπολοίπων βασίζεται στο 'ποιος' αποφασίζει πότε μια ενέργεια θα εκτελεστεί. Ένα αυτόματο μπορεί να θέσει περιορισμούς ή προϋποθέσεις (preconditions) στην εκτέλεση ενεργειών εξόδου και εσωτερικών αλλά σε καμία περίπτωση δεν μπορεί να αποτρέψει την εκτέλεση μιας εσωτερικής ενέργειας. Πιο απλά, οι ενέργειες εξόδου και εσωτερικές ενέργειες δηλώνονται στη μορφή προϋπόθεση-συνέπεια (precondition-effect) όπου η προϋπόθεση υποδεικνύει πότε η ενέργεια μπορεί να εκτελεστεί και η συνέπεια το αποτέλεσμα της εκτέλεσης. Μια ενέργεια π ενός αυτομάτου θεωρείται 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 εκτέλεση τους.



Σχήμα 2.1 Παράδειγμα Αυτόματου Εισόδου/Εξόδου (Αυτόματο Καναλιού). Το περιβάλλον του αυτόματου είναι οι κόμβοι i, j και η επικοινωνία επιτυγχάνεται μέσω των ενεργειών $send(m)_{i,j}, recv(m)_{i,j}$



Σχήμα 2.2 Παράδειγμα Αυτόματου Εισόδου/Εξόδου (Αυτόματο Διεργασίας) υπολογισμού αθροίσματος.

Το σύνολο των ενεργειών ενός αυτόματου και ο διαχωρισμός τους καθορίζει μια διεπαφή ανάμεσα στο αυτόματο και το περιβάλλον του. Η διεπαφή αυτή αναφέρεται ως υπογραφή του αυτόματου(action signature S). Συγκεκριμένα, μια υπογραφή S είναι ο διαχωρισμός ενός συνόλου ενεργειών act(S) σε τρία ανεξάρτητα σύνολα ενεργειών in(S), out(S), int(S). Οι εντολές εισόδου και εξόδου στο σύνολο τους εκφράζονται ως εξωτερικές ενέργειες (external trace), ext(S)= in(S) U out(S), και είναι ορατές στο περιβάλλον του αυτόματου ενώ οι εσωτερικές ενέργειες είναι ορατές μόνο στο αυτόματο.

Ωστόσο, παράλληλα με την υπογραφή, ένα input/output αυτόματο A απαρτίζεται από πέντε συστατικά [6, 9] :

- 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 οι 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.

Channel Automaton $C_{i,j}$

Signature:

input: $\text{send}(m)_{i,j}$

output: $\text{recv}(m)_{i,j}$

State:

MSG, a set of elements in M, initially $\emptyset$

Transitions:

Input $\text{send}(m)_{i,j}$	Output $\text{recv}(m)_{i,j}$
Effect:	Precondition:
MSG ← MSG ∪ {m}	m ∈ MSG
	Effect
	MSG ← MSG – {M}

Tasks:

$\{\text{recv}(m)_{i,j} : m \in M\}$

Σχήμα 2.3 Παράδειγμα προδιαγραφής Αυτόματου Καναλιού.

Κάθε στοιχείο στις σχέσεις μεταβάσεων ενός αυτόματου αναπαριστά ένα πιθανό βήμα κατά τον υπολογισμό του συστήματος. Για παράδειγμα, έστω το στοιχείο (s', π, s) ανήκει στο σύνολο steps(A) του αυτόματου A. Τότε η ενέργεια π θεωρείται ενεργή κατά την κατάσταση s'. «Εκτέλεση» ενός αυτόματου ονομάζεται μια εναλλασσόμενη πεπερασμένη ή μη ακολουθία από καταστάσεις και ενέργειες όπου $(s_i, \pi_{i+1}, s_{i+1})$ ϵ trans(A).

- Πεπερασμένη

  $\alpha = s_0, \pi_1 \ s_1 \ s_1, \pi_2 \ s_2, \ldots, \pi_i \ s_i$
- Μη πεπερασμένη ακολουθία

  $\alpha = s_0, \pi_1 \ s_1 \ s_1, \pi_2 \ s_2, \ldots, \pi_i \ s_i \ldots$

Για παράδειγμα έχουμε τις παρακάτω πιθανές εκτελέσεις του παραδείγματος στο Σχήμα 2.3 με αλφάβητο μηνυμάτων M={α,β } :

1. $[\emptyset], send(a)_{i,j}, [a], send(b)_{i,j}, [ab], recv(b)_{i,j}, [a]$
2. $[\emptyset], send(a)_{i,j}, [a]. send(a)_{i,j}, [aa], send(a)_{i,j}, [aaa] \ldots$

Τα μοντέλα IOA επιτρέπουν τη σύνθεση, η οποία συμβολίζεται με το σύμβολο ° , αυτόματων που μοντελοποιούν απλούστερα συστήματα με σκοπό τη δημιουργία πιο

σύνθετων συστημάτων. Η έννοια της σύνθεσης αυτομάτων είναι σχετικά απλή. Έστω ένα σύνολο αυτομάτων στο οποίο ένα αυτόματο έχει μία ενέργεια εξόδου π, τότε το π είναι ένα μία ενέργεια εισόδου στα υπόλοιπα αυτόματα που έχουν το π σαν ενέργειά τους. Επομένως, εάν ένα αυτόματο εκτελέσει μια εξωτερική του ενέργεια π, τότε όλα τα αυτόματα που έχουν την π, εκτελούν την π παράλληλα. Βασική προϋπόθεση είναι οι δύο αυτές ενέργειες να έχουν τα ίδια ονόματα αφού με αυτόν τον τρόπο επιτυγχάνεται η επικοινωνία μεταξύ αυτομάτων.

Ωστόσο, για να είναι επιτρεπτή η σύνθεση δύο αυτομάτων πρέπει να ισχύουν κάποιοι περιορισμοί. Συγκεκριμένα, έστω το πεπερασμένο σύνολο αυτομάτων $\{S_i\}, i \in I$. Τότε $\forall\, i, j \in I\,, i \neq j$ ισχύει :

- Αφού το σύνολο τον εσωτερικών ενεργειών ενός αυτόματου δεν είναι ορατό σε άλλα αυτόματα τότε δεν επιτρέπεται η σύνθεση αυτομάτων παρά μόνο εάν δεν έχουν κοινές εσωτερικές διεργασίες. Σε αντίθετη περίπτωση, μια εσωτερική ενέργεια του ενός αυτόματου μπορεί να ενεργοποιήσει παράλληλα και μία ενέργεια ενός άλλου.

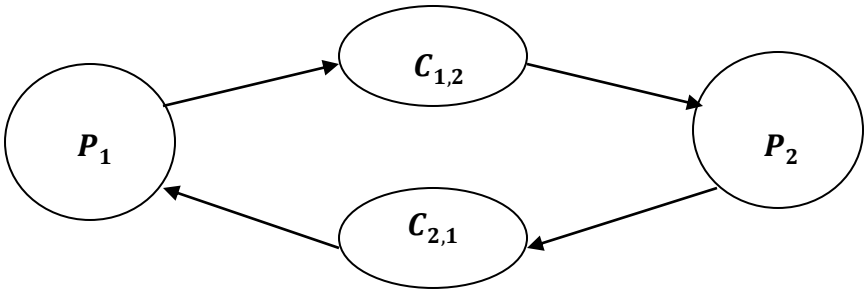
$$\mathbf{init}(\mathbf{S_i}) \cap \mathbf{acts}(\mathbf{S_j}) = \emptyset$$

- Εφόσον μόνο ένα αυτόματό μπορεί να ελέγχει την εκτέλεση μίας ενέργειας, δεν επιτρέπεται η σύνθεση αυτομάτων που έχουν κοινές ενέργειες εξόδου.

$$\mathbf{out}(\mathbf{S_i}) \cap \mathbf{out}(\mathbf{S_j}) = \emptyset$$

- Κάθε ενέργεια μπορεί να είναι ενέργεια μόνο ενός πεπερασμένου συνόλου ενεργειών των συστατικών μιας σύνθεσης.

Κατά την σύνθεση των αυτόματων, καναλιού και διεργασιών που φαίνονται στα Σχήματα 2.1 και 2.2, τα οποία είναι συμβατά, θα έχουμε ένα αυτόματο Α όπου θα αποτελείται από όλες τις μεταβάσεις των αυτομάτων.



Σχήμα 2.4 Γραφική αναπαράσταση σύνθεσης δύο αυτόματων διεργασιών και δύο καναλιών.

2.2 Τύποι Μοντέλων Αυτομάτων

Το μοντέλο αυτομάτων που αναφέραμε στην προηγούμενη ενότητα χωρίζεται σε υποκατηγορίες ανάλογα με τον τρόπο εξέλιξής του κατά την εκτέλεση.

2.2.1 Μοντέλο Χρονισμένων Αυτομάτων Εισόδου/Εξόδου (TIOA)

Το Μοντέλο Χρονισμένων Αυτομάτων Εισόδου/Εξόδου (TIOA) [2, 3, 5] είναι ίσως η δημοφιλέστερη εκδοχή αυτομάτων. Αποτελεί μια μη ντετερμινιστική μηχανή καταστάσεων, της οποίας η λειτουργία βασίζεται σε προθεσμίες βάση χρόνου. Η απόδοση και η ορθότητα αυτής της κατηγορίας αυτομάτων δεν βασίζεται μόνο στη σειρά των γεγονότων , αλλά και στη χρονική στιγμή που συμβαίνει το κάθε γεγονός. Δεν περιορίζεται σε διακριτά βήματα αλλά μπορεί να υπάρχει συνεχής εξέλιξη με την πάροδο του χρόνου η οποία υλοποιείται με την βοήθεια των τροχιών. Συνεπώς, η κατάσταση ενός TIOA μπορεί να αλλάξει είτε λόγω της ύπαρξης μια διακριτής μετάβασης, είτε λόγω μίας τροχιάς. Μία τροχιά ορίζεται ως μια συνάρτηση που περιγράφει την εξέλιξη μίας μεταβλητής ανάμεσα σε δύο διακριτές μεταβάσεις. Ένα χρονισμένο αυτόματο εισόδου/εξόδου μπορεί να έχει περισσότερες από μία τροχιές.

2.2.2 Υβριδικά Αυτόματα

Αποτελούν μια επέκταση των TIOA όπου έχουν εσωτερικές αλλά και εξωτερικές καταστάσεις. Οι εσωτερικές καταστάσεις, όπως και τα TIOA, αντιπροσωπεύουν τις καταστάσεις του αυτόματου ενώ οι εξωτερικές χρησιμοποιούνται παράλληλα με τις εξωτερικές ενέργειες του αυτόματου για επικοινωνία με το περιβάλλον[16].

2.2.3 Αυτόματα Πιθανοτήτων

Αποτελούν επέκταση του βασικού μοντέλου αυτόματων όπου οι μεταβάσεις εκτελούνται με βάση κάποιας πιθανότητας. Συγκεκριμένα, η μετάβαση μίας ενέργειας π δεν οδηγεί στην κατάσταση s αλλά σε ένα σύνολο καταστάσεων όπου η επιλογή μιας κατάστασης γίνεται βάση πιθανοτήτων[16].

2.2.4 Δυναμικά Αυτόματα

Η συγκεκριμένη κατηγορία αυτομάτων[16] επεκτείνει το βασικό μοντέλο αυτομάτων με δύο τρόπους :

- Κατά την εκτέλεση, έχουν την δυνατότητα δημιουργίας επιπλέον αυτόματων εάν δεν υπάρχουν,
- Υπάρχει η δυνατότητα να τροποποιηθεί η υπογραφή κατά την εκτέλεση. Μια κενή υπογραφή δηλώνει την κατάργηση ενός αυτόματου.

2.3 Γλώσσα TIOA και Εργαλείο TEMPO

2.3.1 Γλώσσα TIOA

Η γλώσσα TIOA [2, 3, 5] αποτελεί μια γλώσσα προδιαγραφής των χρονισμένων αυτομάτων εισόδου/εξόδου και των ιδιοτήτων τους και η γλώσσα που χρησιμοποιήθηκε για την προδιαγραφή του αλγορίθμου που αφορά τη παρούσα Διπλωματική εργασία. Η γλώσσα TIOA έχει αρκετές ομοιότητες με την επίσημη γλώσσα IOA [6] και χρησιμοποιείται για την προδιαγραφή ασύγχρονων κατανεμημένων αλγόριθμων.
Ακολουθούν τα δομικά στοιχεία της γλώσσας TIOA όπως οι τύποι δεδομένων, εκφράσεις, τελεστές και εντολές. Έπειτα, ακολουθεί η ακριβής διαδικασία για τον ορισμό των συστατικών που απαιτούνται για την προδιαγραφή αλγορίθμων σε γλώσσα TIOA.

Τύποι δεδομένων

Οι τύποι δεδομένων μπορεί να είναι στατικοί ή δυναμικοί. Ο στατικός τύπος για μία μεταβλητή περιγράφει την τιμή που παίρνει ενώ ένας δυναμικός τύπος αποτελεί την συνάρτηση που περιγράφει την εξέλιξη της συγκεκριμένης μεταβλητής στον χρόνο. Οι βασικοί τύποι δεδομένων που υποστηρίζονται στη γλώσσα TIOA είναι οι εξής: ***Boolean (Bool)***: Αρχέγονος τύπος δεδομένων ο οποίος μπορεί να πάρει δύο τιμές, 0(false) και 1(true) που ονομάζονται boolean ή λογικές τιμές. Η γλώσσα TIOA υποστηρίζει τις ακόλουθες πράξεις σε μια boolean έκφραση: άρνηση (~), λογικό AND

(/\), λογικό OR(\/), συνεπαγωγή (=>) και διπλή συνεπαγωγή (<=>). Επίσης, μπορούν να χρησιμοποιηθούν ο καθολικός (∀) και ο υπαρξιακός ποσοδείκτης (∃).

Natural Numbers (Nat): Το σύνολο των μη αρνητικών ακέραιων αριθμών. Υποστηρίζονται οι συναρτήσεις: succ(*x*), η οποία επιστρέφει τον αμέσως επόμενο φυσικό αριθμό από το *x*, pred(*x*), η οποία επιστρέφει τον προηγούμενο φυσικό αριθμό από το x, min(*x*,*y*), η οποία επιστρέφει τον μικρότερο φυσικό αριθμό μεταξύ των x και y, max(x,y), η οποία επιστρέφει τον μεγαλύτερο φυσικό αριθμό μεταξύ των x και y, div(*x*,*y*), mod(*x*,*y*) και τις πράξεις: πρόσθεση (+), αφαίρεση (-), πολλαπλασιασμός (*), μεγαλύτερο (>), μεγαλύτερο και ίσο (≥), μικρότερο (<), μικρότερο και ίσο (≤), ίσο (=) και άνισο (~=).

Integers (Int): Είναι ένας ακέραιος αριθμός. Υποστηρίζει όλες τις συναρτήσεις και όλες τις πράξεις των φυσικών αριθμών και επιπρόσθετα το αρνητικό πρόσημο (-) και την συνάρτηση abs(x), η οποία επιστρέφει την απόλυτη τιμή του *x*.

Augmented Real Numbers (AugmentedReal): Είναι ένας πραγματικός αριθμός συμπεριλαμβανομένων των τιμών + ∞ και -∞.

Real Numbers (Real): Είναι ένας πραγματικός αριθμός, υποσύνολο των αριθμών AugmentedReal. Υποστηρίζει όλες τις πράξεις των φυσικών αριθμών καθώς και τις συναρτήσεις: min(*x*,*y*), η οποία επιστρέφει τον μικρότερο πραγματικό αριθμό μεταξύ των x και y, max(*x*,*y*), η οποία επιστρέφει τον μεγαλύτερο πραγματικό αριθμό μεταξύ των x και y, abs(*x*), η οποία επιστρέφει την απόλυτη τιμή του *x*, floor(*x*), η οποία επιστρέφει τον μεγαλύτερο ακέραιο αριθμό που είναι μικρότερος από τον *x*.

Discrete Real Numbers (DiscreteReal): Είναι ένας πραγματικός αριθμός, με τη διαφορά ότι δεν αλλάζει μεταξύ των διακριτών βημάτων. Είναι ιδεατός τύπος και χρησιμοποιείται για να ορίζει το φυσικό ρολόι ενός αυτομάτου. Η διαφορά μεταξύ DiscreteReal και Real είναι πως ο DiscreteReal είναι μεταβλητός τύπος ενώ ο Real στατικός.

Characters (Char): Αποτελεί ένα χαρακτήρα. Υποστηρίζει τις συγκρίσεις μεταξύ δύο τιμών (<,>,≥ ,≤) και η σύγκριση γίνεται στην τιμή του συγκεκριμένου χαρακτήρα στον πίνακα ASCII.

Ωστόσο, υπάρχουν και οι πιο σύνθετοι τύποι δεδομένων αλλά και οι εκφράσεις και τελεστές όπως :

***Strings*:** Είναι μια ακολουθία από χαρακτήρες η οποία μπορεί να αναπαρασταθεί και από τον τύπο Seq [E].

***Arrays (Array [$T_1, \dots T_n, E$])*:** Για κάθε n>0, η δήλωση Array [$T_1, \dots T_n, E$], αποτελεί ένα μονοδιάστατο πίνακα n στοιχείων [$T_1, \dots T_n, E$] τύπου E.

Set (Set[E]): Αποτελεί ένα σύνολο πεπερασμένου αριθμού στοιχείων τύπου E. Υποστηρίζει τις ακόλουθες πράξεις συνόλων: ένωση (∪), τομή (∩), συμπλήρωμα (—), ισότητα (=), υποσύνολο (⊂), υποσύνολο ή ίσον (⊆), ανήκει (∈), δεν ανήκει (∉), κενό σύνολο (∅). Επίσης υποστηρίζει τις ακόλουθες συναρτήσεις: size(S), η οποία επιστρέφει το πλήθος των στοιχείων στο σύνολο S, insert(e,S), η οποία εισάγει το στοιχείο e στο σύνολο S και delete(e,S), η οποία διαγράφει το στοιχείο e από το σύνολο S.

Seq (Seq[E]): Αποτελεί μια ακολουθία πεπερασμένου αριθμού στοιχείων του τύπου E. Υποστηρίζει τις πράξεις: κενό (∅), ανήκει (∈), δεν ανήκει (∉), εισαγωγή στοιχείου (|-), εξαγωγή στοιχείου (-|), και τις συναρτήσεις: head(S), η οποία επιστρέφει το πρώτο στοιχείο της ακολουθίας S, tail(S), η οποία επιστρέφει το τελευταίο στοιχείο της ακολουθίας S, len(S), η οποία επιστρέφει το μήκος της ακολουθίας S και S[n], η οποία επιστρέφει το n-οστό στοιχείο της.

***Enumerations (Enumeration [$e_1, \dots, e_n$])*:** υποδηλώνει ένα πεπερασμένο πεδίο τιμών $e_1, \dots, e_n$ το οποίο ορίζεται ως εξής: Name: Enumeration [$e_1, \dots, e_n$], όπου Name είναι το όνομα αυτής της μεταβλητής και $e_1, \dots, e_n$ είναι το πεδίο τιμών που μπορεί να έχει. Να σημειώσουμε ότι για έναν τύπο Name ισχύει η σειρά με την οποία δηλώνονται οι επιτρεπόμενες τιμές. Δηλαδή, εάν μία μεταβλητή c τύπου Name έχει τιμή e_1, η έκφραση c:=c+1 έχει ως αποτέλεσμα την αλλαγή της c με επόμενη διαθέσιμη τιμή e_2 .

Unions (Union[$f_1: T_1, \dots, f_n: T_n$]) : Ο τύπος αυτός επιτρέπει στα στοιχεία του να έχουν μια τιμή $f_1, \dots, f_n$ της οποίας ο τύπος δεδομένων να είναι ένας από τους τύπους δεδομένων *T1, … , Tn*. Για παράδειγμα εάν ορίσουμε:

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

τότε η μεταβλητή v μπορεί να είναι είτε συμβολοσειρά είτε ακέραιος αριθμός είτε φυσικός αριθμός.

Tuples (Tuple[$f_1: T_1, \dots, f_n: T_n$]) : Η 'πλειάδα' αποτελεί ένα νέο σύνθετο τύπο δεδομένων όπου κάθε πεδίο $f_1, \dots, f_n$ είναι τύπου $T_1, \dots, T_n$, αντίστοιχα. Για παράδειγμα εάν έχουμε μια μεταβλητή action τύπου Tuple[type:String, id:Int], τότε η μεταβλητή action αποτελείται από τα πεδία type και id, στα οποία έχουμε πρόσβαση μέσω του τελεστή '.'. Δηλαδή, η action.type παρέχει πρόσβαση στην συμβολοσειρά type που συνδέεται με την μεταβλητή action.

Συναρτήσεις (functions) **:** Η γλώσσα TIOA επιτρέπει την δημιουργία ειδικών συναρτήσεων των οποίων η χρήση μπορεί να γίνει κατά την προδιαγραφή ενός αυτόματου. Μία συνάρτηση μπορεί να οριστεί : έξω από το αυτόματο (global function), μέσα στο αυτόματο μετά από τις καταστάσεις του αυτομάτου, στον ορισμό μίας μετάβασης μετά τις τοπικές μεταβλητές και πριν από τις προϋποθέσεις, ή εντός του ορισμού της τροχιάς μετά από το όνομα και πριν από τις παραμέτρους της. Ο ορισμός μίας συνάρτησης αποτελείται από : τη λέξη κλειδί 'let', το όνομά της, τα ονόματα και τους τύπους των παραμέτρων της, τον τύπο επιστροφής της και τον ορισμό της. Για παράδειγμα, η πιο κάτω συνάρτηση παίρνει δύο φυσικούς αριθμούς hour και minute που εκφράζουν τον χρόνο σε ώρες και λεπτά και χρησιμοποιείται για να ελέγχει τις τιμές των παραμέτρων της.

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

Τελεστής Where : Χρησιμοποιείται για τον καθορισμό του πεδίο τιμών μιας μεταβλητής. Συνήθως, χρησιμοποιείται σε δηλώσεις ενεργειών και αυτομάτων για να περιορισμό του εύρους τιμών των παραμέτρων. Για παράδειγμα: automaton Process (id : Nat) where id >0 , δηλώνει ότι η παράμετρος id πρέπει να είναι μεγαλύτερη από το μηδέν.

Τελεστής Choose : Η επιλογή τυχαίων τιμών για τις μεταβλητές κάνει χρήση του τελεστή choose και συνδυάζεται με τον τελεστή where. Για παράδειγμα: value : int := choose *n* (where $n \geq 0 \vee n \leq 10$) , τότε η μεταβλητή value παίρνει τυχαία μια τιμή μεταξύ του 0 και του 10, συμπεριλαμβανομένων. Εάν δηλωθεί: value : int := choose *n* , τότε η τιμή μεταβλητή *value* παίρνει τυχαία μια τιμή από το πεδίο τιμών των ακεραίων αριθμών.

Λεξιλόγιο

Η γλώσσα TIOA παρέχει στον χρήστη τη δυνατότητα ορισμού του δικού του λεξιλογίου. Σε ένα λεξιλόγιο μπορεί να οριστούν αφαιρετικοί τύπου δεδομένων και τελεστές ανάλογα με τις ανάγκες του χρήστη. Ο ορισμός ενός λεξιλογίου αρχίζει με τη λέξη κλειδί 'vocabulary' και ένα αναγνωριστικό το οποίο αντιπροσωπεύει το όνομα του. Σε περίπτωση χρήσης τύπων δεδομένων ή τελεστών από άλλα λεξιλόγια θα πρέπει να γίνει εισαγωγή τους στο παρών λεξιλόγιο για να αναγνωρίζονται. Αυτό γίνεται με την χρήση της λέξης 'import' και το όνομα του λεξιλογίου που θα προστεθεί. Κάθε λεξιλόγιο μπορεί να συμπεριλαμβάνει ένα η περισσότερους τύπους δεδομένων

(ακολουθώντας την λέξη ‘types’ ή ‘defines’) και μηδέν ή περισσότερους τελεστές (ακολουθώντας την λέξη ‘operators’). Κάθε τελεστής αποτελείται από μία υπογραφή, η οποία καθορίζει : τις παραμέτρους του , ακολουθώντας το σύμβολο ‘:’ , τους τύπους δεδομένων των παραμέτρων του, ακολουθώντας το σύμβολο → και τον τύπο του αποτελέσματος. Ο ορισμός ενός λεξιλογίου τελειώνει με την λέξη κλειδί ‘end’. Στο ακόλουθο παράδειγμα ορίζεται ένα απλό λεξιλόγιο το οποίο καθορίζει τις πράξεις μεταξύ φυσικών αριθμών :

```
vocabulary Equation
types
Nat
operators
        +, - , *, **: Nat, Nat → Nat
end
```

Εντολές

Εντολή Ανάθεσης : Για να γίνει ανάθεση μιας τιμής σε μια μεταβλητή, χρησιμοποιείται το σύμβολο ‘:=’. Στο αριστερό μέλος της ανάθεσης βρίσκεται η μεταβλητή και δεξιά η τιμή ανάθεσης.

Εντολή Print : Η εντολή αυτή χρησιμοποιείται για να τυπώνει την τιμή μιας έκφρασης. Ορίζεται με τη λέξη κλειδί print, μία έκφραση και το σύμβολο ‘;’.

Εντολή If-then-else : Αποτελείται από τη λέξη κλειδί ‘if’, ένα λογικό κατηγόρημα στο οποίο αποτιμάτε η τιμή true ή false, τη λέξη κλειδί ‘then’ και ένα σύνολο από εντολές οι οποίες θα εκτελεστούν όταν το λογικό κατηγόρημα που ακολουθεί το ‘if’ είναι αληθές. Στη συνέχεια ακολουθούν μηδέν ή περισσότερα τμήματα elseif, τα οποία ορίζονται με τον ίδιο τρόπο. Οι εντολές elseif μπορεί να είναι φωλιασμένες. Ακολουθεί το τμήμα else εάν υπάρχει το οποίο ορίζεται με τη λέξη κλειδί ‘else’, το σύνολο των εντολών που θα εκτελεστούν όταν όλα τα προηγούμενα λογικά κατηγορήματα είναι ψευδή και τελειώνει με τη λέξη κλειδί ‘fi’. Ένα παράδειγμα είναι το ακόλουθο, όπου τυπώνεται ο αριθμός a ή failed ανάλογα με το αν ισχύει η συνθήκη:

```
if (a>5) then
        print a;
else
        print “failed”;
fi;
```

Εντολή While : Η εντολή αυτή αποτελείται από τη λέξη κλειδί 'while', ένα λογικό κατηγόρημα, τη λέξη κλειδί 'do', ένα σύνολο από εντολές που θα εκτελούνται μέχρι το λογικό κατηγόρημα γίνει ψευδές. Ολοκληρώνεται με τη λέξη κλειδί 'od'. Οι εντολές while μπορεί να είναι φωλιασμένες. Ένα παράδειγμα είναι το ακόλουθο όπου τυπώνει τους αριθμούς από το 0 μέχρι και το 9.

```
a:=0;
while (a<10)do
        print a;
        a := a + 1;
od
```

Εντολή For : Αποτελείται από τη λέξη κλειδί 'for', το όνομα της μεταβλητής που αποτελεί τον μετρητή, το σύμβολο ':', τον τύπο της μεταβλητής, τη λέξη κλειδί 'while', τη συνθήκη επανάληψης, τη λέξη κλειδί 'do', ένα σύνολο από εντολές που θα εκτελούνται σε κάθε επανάληψη. Ολοκληρώνεται με τη λέξη κλειδί 'od' στο τέλος.

Εντολή Fire : Χρησιμοποιείται για την πυροδότηση μίας μεταβατικής ενέργειας. Η εντολή fire αποτελείται από τη λέξη κλειδί 'fire', το είδος της ενέργειας που θα πυροδοτηθεί (input, output, internal), το όνομα του αυτομάτου που περιέχει την ενέργεια, το σύμβολο '.', το όνομα της ενέργειας, τις παραμέτρους που παίρνει η ενέργεια σε παρενθέσεις και το σύμβολο ';'. Για παράδειγμα : *fire input P.init(v)*

Εντολή Follow : Χρησιμοποιείται για την εξέλιξη μιας τροχιάς. Η εντολή follow αποτελείται από τη λέξη κλειδί 'follow', το όνομα του αυτομάτου που περιέχει την τροχιά που θα εξελιχθεί, το σύμβολο '.', το όνομα της τροχιάς, τη λέξη κλειδί 'duration' και το χρονικό διάστημα κατά τον οποίο θα εξελίσσεται η τροχιά.

Αυτόματα TIOA

Ο ορισμός ενός απλού αυτόματου στη γλώσσα TIOA αρχίζει με την λέξη κλειδί 'automaton' ακολουθούμενη από το όνομα του αυτομάτου και τυχόν παραμέτρους εάν υπάρχουν. Στην περίπτωση που γίνεται χρήση παραμέτρων μπορεί να χρησιμοποιηθεί ο τελεστής *where*, όπως έχει οριστεί πιο πάνω για τον καθορισμό του πεδίου τιμών τους. Για παράδειγμα το ακόλουθο αυτόματο:

```
automaton Process(id:Nat, d1:DiscreteReal) where id ≥ 0
```

έχει το όνομα Process και παραμέτρους id και d1 με τύπο Nat και DiscreteReal αντίστοιχα. Επιπλέον, τίθεται περιορισμός στην μεταβλητή id η οποία δεν μπορεί να πάρει αρνητικές τιμές.

Ο ορισμός ενός απλού αυτόματου απαρτίζεται από τα εξής τμήματα :

- Υπογραφή Αυτόματου (Action Signature)
- Μεταβλητές Καταστάσεων (State Variables)
- Σχέσεις Μεταβάσεων (Transition Relations)
- Τροχιές (Trajectories)

Υπογραφή Αυτόματου : Η υπογραφή ενός αυτόματου όπως αναφέρθηκε και σε προηγούμενη υποενότητα, περιλαμβάνει το σύνολο των ενεργειών ενός αυτόματου και είναι απαραίτητη κατά τον ορισμό του. Αρχίζει με την λέξη κλειδί 'signature' και ακολουθούν οι ενέργειες του αυτόματου. Κάθε ενέργεια δηλώνεται με το είδος της (input , output, internal) και όπως και στον ορισμό του αυτόματου, ακολουθείτε από το όνομα και τις παραμέτρους, με παρόμοιο τρόπο. Για παράδειγμα :

signature

input write(v : Nat) where v ≥ 0

output printVal

internal init

το πιο πάνω αυτόματο αποτελείται από μια εσωτερική ενέργεια init, μια ενέργεια εισόδου *write* με περιορισμό $v \geq 0$ και μία εξόδου, την printVal.

Μεταβλητές Καταστάσεων : Σε αυτό το τμήμα, το οποίο είναι επίσης υποχρεωτικό, περιλαμβάνονται όλες οι μεταβλητές του αυτομάτου και ακολουθεί πάντα το τμήμα υπογραφής. Αρχίζει με την λέξη κλειδί 'states' και ακολουθούν οι δηλώσεις των μεταβλητών μαζί με τον τύπο και την αρχική τιμή τους. Η αρχικοποίηση μεταβλητών μπορεί να γίνει και με την χρήση των τελεστών where και choose. Στο παρακάτω παράδειγμα μεταβλητών καταστάσεων :

states

initialized: Bool := false;

value: Nat:=0;

msgscounter: Nat := 0;

Υπάρχει η μεταβλητή *i*nitialized τύπου Boolean και αρχικοποιημένη σε false και οι μεταβλητές value, msgcounter τύπου Nat, αρχικοποιημένες με 0.

Τυχόν συναρτήσεις πρέπει να οριστούν σε αυτό το σημείο, αμέσως μετά τις μεταβλητές καταστάσεων και πριν τις σχέσεις μεταβάσεων.

Σχέσεις μεταβάσεων: Ένα επίσης υποχρεωτικό τμήμα κατά τον ορισμό ενός αυτόματου. Ορίζεται το πώς ένα αυτόματο μεταβάλλεται όταν εκτελεστεί μια συγκεκριμένη ενέργεια. Είναι σημαντικό οι ενέργειες που έχουν δηλωθεί στην υπογραφή του αυτόματου να είναι ακριβής αντιστοίχηση των ενεργειών σε αυτό το τμήμα (είδος, όνομα, αριθμός παραμέτρων) χωρίς τον τύπο των παραμέτρων. Επιπρόσθετες μεταβλητές μπορούν να οριστούν εντός της ενέργειας μετά το όνομα της ενέργειας αλλά πριν από τις προϋποθέσεις. Οι μεταβλητές αυτές ονομάζονται τοπικές και η διαδικασία δήλωσης είναι η ίδια με των μεταβλητών καταστάσεων με την διαφορά ότι χρησιμοποιείται η λέξη κλειδί ‘locals’ έναντι της λέξης ‘states’. Η ορατότητα των συγκεκριμένων μεταβλητών περιορίζεται εντός της ενέργειας σε αντίθεση με των μεταβλητών καταστάσεων. Επιπλέον, όπως έχει προαναφερθεί, οι ενέργειες (εκτός των ενεργειών τύπου input) μπορεί να έχουν κάποιες προϋποθέσεις πριν την εκτέλεση τους. Η προϋποθέσεις (preconditions) αυτές δηλώνονται αμέσως μετά, στην μορφή

```
prec
        A
        B
eff
```

όπου A και B είναι λογικά κατηγορήματα στις μεταβλητές του αυτόματου (τοπικές και καθολικές). Στην απουσία προϋποθέσεων η ενέργεια θεωρείται πάντοτε ενεργή. Οι συνέπειες μιας ενέργειας ορίζονται στο επόμενο στάδιο μετά την λέξη κλειδί ‘eff’ και πριν από τις εντολές που εκτελούνται όταν η ενέργεια γίνει ενεργή. Κάθε εντολή τερματίζεται με το σύμβολο ‘;’ . Για παράδειγμα :

```
transitions
        Input init
        eff
                if (initialized=false) then
                        initialized:=true;
                        value:= x * x ;
                        maxVal:=value;
                fi
```

Ορίζεται η ενέργεια εισόδου init η οποία εάν ισχύει initialized = false τότε τροποποιεί τις τιμές των μεταβλητών initialized, value, maxVal. Βλέπουμε ότι αν και οι ενέργειες εισόδου είναι πάντοτε ενεργές, μπορούμε να επιλέξουμε εάν θα εκτελεστεί το συγκεκριμένο κομμάτι κώδικα.

Τροχιές : Σε αντίθεση με τα υπόλοιπα τμήματα του αυτόματου, η δήλωση μιας τροχιάς δεν είναι υποχρεωτική σε περίπτωση που το αυτόματο δεν κάνει χρήση του χρόνου κατά οποιαδήποτε έννοια. Οι τροχιές υλοποιούν την έννοια του χρόνου στα χρονισμένα αυτόματα Εισόδου/Εξόδου . Η λέξη κλειδί 'trajectories' υποδηλώνει την έναρξη του συγκεκριμένου τμήματος κατά τον ορισμό του αυτόματου . Ο ορισμός μιας τροχιάς αρχίζει με την λέξη κλειδί 'trajdef' και ακολουθείται από το όνομα της. Τυχόν σταθερές σχέσεις δηλώνονται ως : 'invaraiants' , το όνομα της σταθερής σχέσης, τη λέξη κλειδί 'of' το όνομα του αυτόματου καθώς και τις συνθήκες που πρέπει να ισχύουν κατά την εκτέλεση του συγκεκριμένου αυτόματου.

Για παράδειγμα :

```
trajectories
    trajdef T
    evolve d(physclock) = 1;
```

Ορίζεται η τροχιά T η οποία ρυθμίζει την εξέλιξη της μεταβλητής *now* με ρυθμό πραγματικού χρόνου.

Σύνθεση Αυτομάτων TIOA

Όπως προαναφέρθηκε, η σύνθεση δύο η περισσότερο αυτομάτων είναι εφικτή. Η σύνθεση αυτόματων βρίσκει χρήση στην διάσπαση πολύπλοκων συστημάτων σε μικρότερα και απλούστερα τμήματα, αυτόματα. Το αυτόματο σύνθεσης παρέχει την δυνατότητα επικοινωνίας μεταξύ των τμημάτων αυτών εάν είναι απαραίτητη για την ολοκλήρωση των διεργασιών. Εφόσον ένα αυτόματο σύνθεσης αποτελείται από την ένωση άλλων, ήδη υλοποιημένων αυτόματων, δεν απαιτείται η υλοποίηση ενεργειών παρά μόνο η εισαγωγή των αυτομάτων και των λεξιλογίων τους και η σειρά εκτέλεσης των ενεργειών η οποία αναφέρεται ως πρόγραμμα μεταβάσεων και τροχιών (schedule) του αυτόματου σύνθεσης. Υπάρχει ωστόσο, η δυνατότητα δήλωσης κρυφών ενεργειών. Ο ορισμός του αυτόματου σύνθεσης αρχίζει με την λέξη κλειδί 'automaton', το όνομα του αυτόματου και τις παραμέτρους που δέχεται όπως ένα κανονικό αυτόματο. Στην συνέχεια, μετά την λέξη κλειδί 'components' ακολουθεί η δήλωση των αυτόματων που

αποτελούν την σύνθεση, σε παρόμοια μορφή με αυτή των μεταβλητών καταστάσεων. Ορίζεται δηλαδή ένα αναγνωριστικό και μετά το σύμβολο ‘:’ οι παράμετροι που παίρνει και το σύμβολο ‘;’. Εάν υπάρχουν κρυφές ενέργειες ορίζονται στο σημείο αυτό μετά την λέξη κλειδί ‘hidden’. Σε αντίθετη περίπτωση ακολουθεί η λέξη κλειδί ‘schedule’ για το χρονοδιάγραμμα που θα ακολουθήσει το αυτόματο σύνθεσης, το οποίο περιλαμβάνει τον ορισμό των μεταβλητών καταστάσεων. Το σενάριο προσομοίωσης, στο οποίο πυροδοτούνται οι ενέργειες και εξελίσσονται οι τροχιές των αυτομάτων περικλείεται εντός των λέξεων ‘do … od’ . Για την πυροδότηση μιας ενέργειας χρησιμοποιείται η εντολή fire, το όνομα και οι παράμετροι της ενώ για την εξέλιξη μίας τροχιάς η εντολή follow και το όνομα, και η διάρκεια εξέλιξης μετά την λέξη κλειδί ‘duration’. Ένα παράδειγμα σύνθεσης είναι το ακόλουθο :

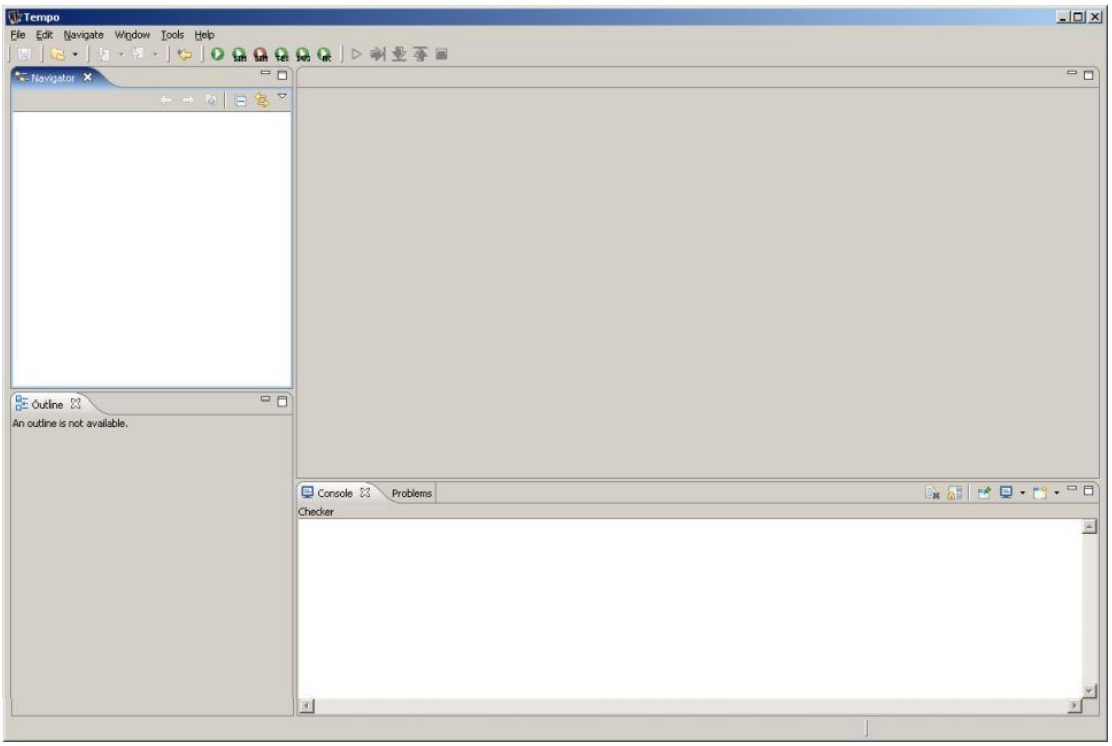
automaton FindMaxCompostion()
components
PR : Process;
SM : SendMediator;
RM : ReceiveMediator;

Το αυτόματο example αποτελείται από την σύνθεση τριών αυτομάτων, PR, SM, και RM

2.3.2 Εργαλείο TEMPO

Το εργαλείο TEMPO αποτελεί ένα εργαλείο μοντελοποίησης κατανεμημένων συστημάτων και διανέμεται δωρεάν από την Veromodo Inc. [13, 14, 15]. Περιλαμβάνονται εργαλεία για προσομοίωσης, ανάλυσης και απόδειξης της ορθότητας των συστημάτων καθώς και αυτοματοποιημένη μετάφραση σε Java, LaTex, Uppaal. Η γλώσσα που χρησιμοποιείται είναι η TIOA (με επέκταση αρχείου '.tioa'), την οποία έχουμε προαναφέρει, και συνεπώς το μοντέλο αυτόματου που μοντελοποιείται με τη χρήση του εργαλείου είναι τα Χρονισμένα Αυτόματα Εισόδου Εξόδου.

Στην ιστοσελίδα της Veromondo, πέραν του εργαλείου ο χρήστης μπορεί να προμηθευτεί πλήρης εγχειρίδια για την εγκατάσταση, λειτουργία αλλά και για τυχόν προβλήματα ή περιορισμούς του εργαλείου [13, 14, 15]. Η επιφάνεια χρήστη φαίνεται στο Σχήμα 2.5 που ακολουθεί και αποτελείται από τα εξής τμήματα.



Σχήμα 2.5 Επιφάνεια Χρήστη στο εργαλείο TEMPO

Menu bar: Περιέχει τα μενού File, Edit, Navigate, Window, Tools και Help. Τα μενού File, Edit και Help προσφέρουν τις λειτουργίες ενός επεξεργαστή κειμένου, όπως είναι η δημιουργία και αποθήκευση αρχείων, η αντιγραφή κειμένου και η βοήθεια. Μέσω του Navigate ο χρήστης μπορεί να μεταφερθεί σε ένα συγκεκριμένο μέρος του αρχείου. Το Window μενού επιτρέπει τον έλεγχο της εμφάνισης της διεπαφής. Τέλος, με το Tools μενού παρέχεται πρόσβαση στα διάφορα εργαλεία που υποστηρίζονται στο TEMPO και θα αναλυθούν στην συνέχεια.

Tool bar: Περιέχει τα εικονίδια των εργαλείων που περιέχονται στο εργαλείο TEMPO.

Status bar: Βρίσκεται στο κάτω μέρος της διεπαφής και παρέχει χρήσιμες πληροφορίες για την κατάστασή της, όπως για παράδειγμα τη θέση που βρίσκεται ο δείκτης μέσα σε ένα αρχείο, γραμμή και στήλη, μια δεδομένη χρονική στιγμή.

Navigator: Περιέχει όλες τις εργασίες (projects) που βρίσκονται στο χώρο εργασίας και τα οποία εμφανίζονται με ιεραρχική δομή.

Outline: Παρουσιάζει διαγραμματικά το περιεχόμενο του αρχείου που βρίσκεται ανοικτό και που εμφανίζεται στον κενό χώρο της διεπαφής (editor). Διακρίνονται οι ενέργειες, οι μεταβλητές, οι τροχιές και γενικά όλα τα τμήματα του αυτομάτου που

ορίζονται στο συγκεκριμένο αρχείο. Για αρχεία που περιγράφουν λεξιλόγιο, δεν εμφανίζεται τίποτα.

Editor: Παρέχεται η δυνατότητα επεξεργασίας ενός αρχείου.

Problems: Στο τμήμα αυτό επεξηγούνται τυχόν προβλήματα που αναγνωρίζονται από τον ελεγκτή του εργαλείου.

Console: Παρουσιάζει την έξοδο μιας προδιαγραφής και τυχόν λάθη που αναγνωρίζονται κατά την εκτέλεσή της.

Το εργαλεία που προσφέρονται από το TEMPO και είναι προσβάσιμα από το menu είναι :

- Ο **ελεγκτής** (checker) που είναι υπεύθυνος για τον έλεγχο της σύνταξης αλλά και τον σημασιολογικό έλεγχο της προδιαγραφής ενός αυτομάτου.
- Ο **προσομοιωτής** (simulator), υπεύθυνος για την προσομοίωση της εκτέλεσης του αυτομάτου.
- Ο **μεταφραστής Java** (Tempo2Java), ο οποίος μεταγλωττίζει την προδιαγραφή σε γλώσσα JAVA. Τα παραγόμενα αρχεία περιέχονται στο χώρο εργασίας, σε ένα φάκελο με το όνομα 'Name.java', όπου Name είναι το όνομα της εργασίας που μεταφράστηκε.
- Ο **μεταφραστής LaTex** (Tempo2Tex), ο οποίος μεταγλωττίζει την προδιαγραφή σε γλώσσα LaTex. Το παραγόμενο αρχείο περιέχεται στο χώρο εργασίας με το όνομα 'Name.tex', όπου Name είναι το όνομα της εργασίας που μεταφράστηκε.
- Το **εργαλείο για αποδείξεις** (PVS Theorem Prover Tool), το οποίο χρησιμοποιείται για τον έλεγχο της ορθότητας των αλγορίθμων που εκφράζονται σε γλώσσα TIOA.
- Ο **μεταφραστής Uppaal** (uppaal), ο οποίος μεταγλωττίζει την προδιαγραφή σε γλώσσα Uppaal.

Να σημειώσουμε ότι οι μεταφραστές LaTex, Uppaal και PVS, δεν εκτελούν κάποια λειτουργία στην διεπαφή του εργαλείου Tempo.

2.4 Κανάλια Επικοινωνίας Διεργασιών

Η επίτευξη της επικοινωνίας μεταξύ των διεργασιών ενός συστήματος πραγματοποιείται μέσω κάποιων καναλιών. Υπάρχουν δυο είδη καναλιών επικοινωνίας, τα οποία υποστηρίζονται από το εργαλείο Tempo, το MPI (Message Passing Interface) και το TPC. Στα εγχειρίδια χρήσης του εργαλείου Tempo [1, 4, 13] βρίσκεται η υλοποίηση των συγκεκριμένων καναλιών, των οποίων οι προδιαγραφές των αυτομάτων και των λεξιλογίων που τα μοντελοποιούν παρατίθενται στο Παράρτημα Α.

2.4.1 Κανάλια MPI

Το MPI (Message Passing Interface) [11], ορίζεται ως το σύνολο προδιαγραφών ανταλλαγής μηνυμάτων και τη μεταφορά δεδομένων. Κύρια χαρακτηριστικά του είναι η αποτελεσματικότητα, η ευελιξία και η αποδοτικότητά του. Έχει την δυνατότητα μοντελοποίησης ως ένα σύνθετο αυτόματο, στο μοντέλο TIOA, το οποίο αποτελείται από τα αυτόματα: Send Mediator (SendMediator.tioa) και Receive Mediator(ReceiveMediator.tioa). Για την ορθή λειτουργία των αυτόματων αυτών γίνεται αναφορά σε συγκεκριμένο λεξιλόγιο (Vocabulary.tioa). Επιτρέπεται ωστόσο η δήλωση νέου λεξιλογίου εάν θεωρηθεί απαραίτητο από την προδιαγραφή ενός αλγορίθμου, χωρίς όμως να τροποποιείται το τμήμα που αφορά τα κανάλια και την λειτουργία τους. Αρχικά, πρέπει να οριστεί ο αριθμός των διεργασιών που θα εκτελούν τον ίδιο αλγόριθμο. Κατά την έναρξη του προγράμματος σε κάθε διεργασία αποδίδεται ένα διακριτικό με βάση έναν ακέραιο αριθμό (rank) , ο οποίος είναι μοναδικός για κάθε διεργασία. Το εύρος τιμών όπου κυμαίνεται το rank είναι από 0 έως n-1, με n να ορίζεται το πλήθος των διεργασιών. Να σημειωθεί ότι τόσο στο κανάλι MPI όσο και στο TPC, η αποστολή όπως και η λήψη μηνυμάτων γίνεται ανάμεσα σε μόνο δύο διεργασίες, με τη μια διεργασία να εκτελεί την αποστολή και η άλλη την λήψη. Στις προδιαγραφές μας, γίνεται χρήση δύο βασικών συναρτήσεων του MPI: την MPI_Size(), η οποία επιστρέφει το σύνολο των διεργασιών που δημιουργούνται και την MPI_Rank(), η οποία επιστρέφει το αναγνωριστικό μίας διεργασίας, δηλαδή τον αριθμό με τον οποίο αντιστοιχίζεται όταν δημιουργείται.

2.4.2 Κανάλι TCP

Το κανάλι TCP, μοντελοποιείται στο μοντέλο TIOA ως ένα σύνθετο αυτόματο, το οποίο απαρτίζεται από τα αυτόματα Send Mediator (TCPSendMed.tioa), Receive Mediator (TCPRecvMed.tioa) και Channel Mediator (TCPChanMed.tioa). Όπως προαναφέρθηκε και στο κανάλι MPI κάθε αυτόματο που χρησιμοποιείτε στο κανάλι TCP απαιτεί συγκεκριμένο λεξιλόγιο, TCPVocabs.

Το αυτόματο Send Mediator TCP υλοποιεί την αποστολή των μηνυμάτων. Γίνεται μια προσωρινή αποθήκευση στην ακολουθία sendBuffer των μηνυμάτων που αποστέλλονται από μία διεργασία, μέχρι να περάσουν στο αυτόματο του καναλιού. Για την αποστολή μηνύματος αναγκαία είναι η δημιουργία μιας σύνδεσης μεταξύ των δύο μελών, του αποστολέα και του παραλήπτη. Αυτό επιτυγχάνεται με τις ενέργειες TCP_senderOpen και το TCP_respSenderOpen. Ακολούθως, με την επιτυχία εγκατάστασης μίας σύνδεσης όσα μηνύματα υπάρχουν στο sendBuffer μεταφέρονται στο channel mediator μέσω της ενέργειας TCP_write. Μέσα από τις ενέργειες TCP_senderClose και TCP_respSenderClose, μπορεί ανά πάσα στιγμή να τερματιστεί μία σύνδεση.

Το αυτόματο Receive Mediator, πραγματοποιεί την παράδοση μηνυμάτων σε μια διεργασία. Η ενέργεια η οποία είναι υπεύθυνη για την παραλαβή μηνυμάτων είναι η RECEIVE, η οποία ενεργοποιείται όταν υπάρχουν μηνύματα που πρέπει να ληφθούν και όταν δημιουργείται μια σύνδεση μεταξύ της διεργασίας παραλήπτης και της διεργασίας αποστολέας. Για την διαδικασία παραλαβής ενός μηνύματος, μια διεργασία πρέπει να ανταποκριθεί με την εγκαθίδρυση ενός JVMServerSocket. Συγκεκριμένα, με την χρήση των ενεργειών TCP_bind και TCP_respBind, πρέπει να δημιουργηθεί και να δεσμευτεί μια υποδοχή (socket) για την επικοινωνία μεταξύ δυο διεργασιών, καθώς επίσης και να γίνει αποδεκτή μέσω των ενεργειών TCP_accept και TCP_respAccept. Η αποδέσμευση μιας διεργασίας από μία υποδοχή έχει ως συνέπεια τη διακοπή των συνδέσεων προς άλλες διεργασίες Αυτό γίνεται με την χρήση των εντολών TCP_stopAccepting και TCP_stopListening. Επιπλέον, για το κλείσιμο των συνδέσεων που έχουν δημιουργηθεί μεταξύ των διεργασιών χρησιμοποιούνται οι ενέργειες TCP_rClose και TCP_rCloseStream. Μέσω των ενεργειών TCP_read και TCP_respRead γίνεται η παραλαβή μηνυμάτων, εφόσον βέβαια η διεργασία

παραλήπτης βρίσκεται σε κατάσταση υποδοχής. Τα μηνύματα τα οποία διαβάζονται προστίθενται στην ακολουθία recvBuffer.

Το αυτόματο Channel Mediator, υλοποιεί το κανάλι μεταξύ των αυτομάτων Send Mediator και Receive Mediator με βάση το πρωτόκολλο TCP. Οι ενέργειες των αυτομάτων δεν μπορούν να αποκλειστούν. Επομένως, χρησιμοποιούνται χρονικά όρια (timeouts) για αποφυγή οποιονδήποτε αποκλεισμών των αιτημάτων στις υποδοχές (sockets).

2.5 Παράδειγμα υλοποίησης αλγορίθμου στο εργαλείο Tempo

2.5.1 Περιγραφή Αλγόριθμου

Στο σημείο αυτό θα γίνει η προδιαγραφή και υλοποίηση ενός απλού αλγορίθμου για καλύτερη κατανόηση των όσων έχουν ειπωθεί μέχρι στιγμής σε αυτή την ενότητα. Η γενική ιδέα του αλγορίθμου, έστω αλγόριθμος FindMax, είναι η συλλογή *n* τιμών από *n* διαφορετικές διεργασίες και εύρεση της μέγιστης τιμής από το σύνολο των τιμών αυτών. Η τιμή που έχει αποθηκευμένη η κάθε διεργασία στην τοπική της μνήμη αποτελεί το τετράγωνο της τιμής που δίνει ο χρήστης κατά την αρχικοποίηση της διεργασίας. Πιο απλά, έστω value η τιμή που έχει δώσει ο χρήστης στη διεργασία FindMax_i, τότε η διεργασία αποθηκεύει την τιμή *value=value*value* στην τοπική της μνήμη. Επιπλέον, έστω time η χρονική στιγμή που έγινε η αρχικοποίηση της μεταβλητής value. Κάθε διεργασία αρχίζει την μετάδοση της τιμής της ακριβώς την χρονική στιγμή *time + t* όπου *t* αποτελεί ένα πραγματικό αριθμό, όμοιο για όλες τις διεργασίες, και εισάγετε από τον χρήστη.

2.5.2 Προδιαγραφή Αλγόριθμου

Για την υλοποίηση του αλγόριθμου FindMax απαιτείται η υλοποίηση του αντίστοιχου αυτόματου, των αυτόματων που υλοποιούν τα κανάλια επικοινωνίας για μεταφορά των τιμών , του λεξιλογίου και το αυτόματο σύνθεσης. Για την επικοινωνία μεταξύ των διεργασιών χρησιμοποιείται το πρωτόκολλο MPI.

Το λεξιλόγιο που θα χρησιμοποιηθεί, Σχήμα 2.6, αποτελείται από το λεξιλόγιο του αλγόριθμου και των καναλιών MPI. Ορίζεται ο τύπος message ο οποίος είναι ο φυσικός αριθμός που θα ανταλλάσσεται μεταξύ των διεργασιών ενώ το λεξιλόγιο των καναλιών παραμένει ως έχει.

```
%% 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[data: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 του αλγόριθμου FindMax

Το αυτόματο του αλγορίθμου ονομάζεται Process και απεικονίζεται στο Σχήμα 2.7. Δέχεται ως παραμέτρους τις μεταβλητές x, id, t_1. Ο φυσικός αριθμός x είναι η τιμή που εισάγεται από τον χρήστη και ο φυσικός αριθμός id αντιστοιχεί στην ταυτότητα της διεργασίας. Ο πραγματικός αριθμός t_1 είναι το χρονικό διάστημα που πρέπει να περάσει προτού η διεργασία ανακοινώσει την μέγιστη τιμή που κατέχει. Η υπογραφή της διεργασίας αποτελείται από τις ενέργειες RECEIVE, SEND, init, prep_messages, collect και announceTimeout. Οι μεταβλητές καταστάσεων του αυτόματου στο τμήμα states ορίζονται ως ακολούθως:

- maxVal : Αντιπροσωπεύει την μέγιστη τιμή που έχει λάβει η διεργασία τύπου Nat και αρχικοποιείται με την αρχική τιμή της διεργασίας
- msgs: Είναι μια κενή αρχικά ακολουθία μηνυμάτων τύπου MPI_Message τα οποία θα αποσταλούν στις υπόλοιπες διεργασίες.
- initialized: Είναι μια μεταβλητή τύπου Boolean η οποία υποδεικνύει κατά πόσο μια διεργασία έχει αρχικοποιηθεί ή όχι. Αρχικοποιείται με false. Με την χρήση της, αποφεύγεται η εκτέλεση της ενέργειας init περισσότερες από μία φορές.
- value: Αντιπροσωπεύει την αρχική τιμή της διεργασίας τύπου Nat. Αρχικοποιείται από την τιμή που δίνεται ως το τετράγωνο της παραμέτρου x.
- msgcounter: Μεταβλητή τύπου Nat και αντιπροσωπεύει τον αριθμό των μηνυμάτων που έχουν ληφθεί από την διεργασία. Η αρχική τιμή είναι ίση με 0.
- clock: Είναι μεταβλητή τύπου AugmentedReal και αντιπροσωπεύει το φυσικό ρολόι του αυτόματου και αρχικοποιείται με 0.
- processClock: Αντιπροσωπεύει το φυσικό ρολόι τύπου AugmentedReal της διεργασίας το οποίο απαιτείται στον υπολογισμό του χρονικού διαστήματος μετά την λήψη μηνυμάτων από όλες τις διεργασίες. Αρχικοποιείται με την τιμή 0.
- collected: Είναι μεταβλητή τύπου Boolean και καθορίζει κατά πόσο έχουν ληφθεί όλα τα μηνύματα από τις διεργασίες. Αρχικοποιείται με false και μετατρέπεται σε true όταν ληφθούν όλα τα μηνύματα. Ο αριθμός των μηνυμάτων είναι ίσως με τον αριθμό των διεργασιών.

Τις μεταβλητές καταστάσεων ακολουθεί το τμήμα των σχέσεων μεταβάσεων (transitions). Το τμήμα αυτό αποτελείται από τις εξής μεταβάσεις :

- Η ενέργεια εισόδου init αρχικοποιεί την μεταβλητή value με το τετράγωνο της τιμής x που δίνεται ως παράμετρο στο αυτόματο. Εφόσον είναι η πρώτη τιμή που γίνεται γνωστή στο αυτόματο, η μεταβλητή maxVal είναι επίσης ίση με την τιμή αυτή. Αφού τελειώσει η αρχικοποίηση, η μεταβλητή initialized μεταβάλλεται σε true ούτως ώστε να μην επιτρέπεται η αρχικοποίηση της διεργασίας.
- Η ενέργεια εισόδου RECEIVE παίρνει ως είσοδο ένα μήνυμα m το οποίο εάν δεν είναι κενό και έχει ως παραλήπτη την συγκεκριμένη διεργασία, αυξάνεται η τιμή msgcounter κατά ένα. Ακολούθως, γίνεται έλεγχος εάν η μεταβλητή maxVal είναι μικρότερη από την τιμή που έχει ληφθεί. Σε μία τέτοια περίπτωση, γίνεται αντικατάσταση της τιμής με αυτή που έχει ληφθεί.
- Η εσωτερική ενέργεια prep_messages είναι υπεύθυνη για την συμπλήρωση της ακολουθίας msgs με τα απαραίτητα μηνύματα για αποστολή σε όλες τις υπόλοιπες διεργασίες. Κάθε μήνυμα αποτελείται από την μέγιστη τιμή, την ταυτότητα της διεργασίας αποστολέα και παραλήπτη. Προϋπόθεση για την εκτέλεση είναι η αρχικοποίηση της μεταβλητής initialized.
- Η εξωτερική ενέργεια SEND αποστέλλει το μήνυμα m από την ακολουθία msgs. Προϋπόθεση για την εκτέλεση είναι το μήνυμα να είναι πρώτο στην ακολουθία και η ακολουθία να μην είναι κενή. Ακολούθως, αφαιρείται το μήνυμα.
- Η εσωτερική ενέργεια collect ενημερώνει την μεταβλητή collected αλλάζοντας την σε true και εκκινεί την χρονομέτρηση για τύπωση της μέγιστης τιμής, μέσω της μεταβλητής processClock η οποία γίνεται ίση με τη μεταβλητή clock. Προϋπόθεση της ενέργειας είναι να έχουν ληφθεί μηνύματα από όλες τις διεργασίες και να είναι η πρώτη εκτέλεση της, δηλαδή η μεταβλητή collected να είναι ίση με false. Σε περίπτωση που επιτρεπόταν η επανάληψη τότε η μεταβλητή clock δεν θα ξεπερνούσε ποτέ την μεταβλητή processClock κατά d_1 μονάδες χρόνου και η τιμή maxVal δεν θα τυπωνόταν ποτέ.
- Η εσωτερική ενέργεια announceTimeout ενεργοποιείται εφόσον ισχύει clock > processClock + d_1 και collected == true. Δηλαδή, έχει περάσει χρόνος ίσος με

d_1 από την εκτέλεση της εσωτερικής ενέργειας collect και συνεπώς τυπώνεται η μεταβλητή maxVal.

Ο ορισμός του αυτόματου ολοκληρώνεται με την τροχιά Time στο τμήμα trajectories η οποία εξελίσσει το φυσικό ρολόι clock με ρυθμό ίσο με 1.

```
automaton Process(x:message, id:Nat, d1:DiscreteReal)
signature
	input RECEIVE(m:Null[mpi_message]), init
	output SEND(m:Null[mpi_message])
	internal prepMessages,collect,announceTimeout
states
	% max value
	maxVal : Nat := 0;
	% empty mpi message sequence
	msgs : Seq[Null[mpi_message]] := {} ;
	% automaton initialized??
	initialized : Bool := false;
	% user value^2
	value : Nat:=0;
	% counter for the messages received
	msgcounter : Nat := 0;
	% clock of the automaton
	clock: AugmentedReal := 0;
	% clock of the process
	processClock : AugmentedReal := 0;
	% all values collected ??
	collected : Bool := true;
transitions
	input init
		eff
			if(initialized=false) then
				initialized:=true;
				value:= x * x ;
				maxVal:=value;
			fi

	input RECEIVE(m)
		eff
			if(m~=nil) then

				if(val(m).destination=id) then
					msgcounter := msgcounter + 1 ;
					if( maxVal< val(m).data) then
						maxVal:=val(m).data;
					fi
				fi
			fi
	internal prepMessages
		pre
			initialized = true ;
```

```
            eff
                  for n:Nat where (n < MPI_Size()) do
                        if ( n~=id) then
                              msgs := msgs |- embed([maxVal,id,n]);
                        fi
                  od
      output SEND(m)
            pre
                  msgs~= {} ;
                  m= head(msgs);
            eff
                  msgs := tail(msgs);
      internal collect
            pre
                  msgcounter = MPI_Size() - 1;
                  collected=false;
            eff
                  collected := true ;
                  processClock:=clock;
      internal announceTimeout
            pre
                  clock > processClock + d1;
                  collected = true;
            eff
                        print maxVal;

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

Σχήμα 2.7 Αυτόματο διεργασίας (Process.tioa)

Η υλοποίηση του αλγόριθμου ολοκληρώνεται με τη δημιουργία ενός αυτόματου σύνθεσης Composition. Στο αυτόματο σύνθεσης συμπεριλαμβάνονται τα αυτόματα με την υλοποίηση των καναλιών επικοινωνίας SendMediator.tioa και ReceiveMediator.tioa, το αυτόματο διεργασίας και το λεξιλόγιο που έχουν αναφερθεί μέσω της λέξης κλειδί 'include'. Για την ορθή εισαγωγή του λεξιλογίου απαιτείτε η ρητή εισαγωγή όλων των λεξιλογίων που το απαρτίζουν με την λέξη κλειδί 'import' και ακολούθως τα ονόματα των λεξιλογίων. Κάθε λεξιλόγιο δηλώνεται μόνο στο αυτόματο σύνθεσης για αποφυγή διπλού ορισμού των τύπων δεδομένων που δηλώνονται . Το αυτόματο σύνθεσης δέχεται ως είσοδο δύο παραμέτρους. Ο φυσικός αριθμός x που αντιπροσωπεύει την τιμή της διεργασίας και ο πραγματικός αριθμός d_1, το χρονικό διάστημα δηλαδή που απαιτείται προτού αρχίσει η αποστολή της μέγιστης τιμής. Οι τιμές αυτές εισάγονται και στο αυτόματο διεργασίας ως παράμετροι μαζί με τον αριθμό

της διεργασίας ο οποίος επιστρέφεται από την συνάρτηση MPI_Rank() . Παράλληλα ορίζονται και τα αυτόματα SendMediator (SM) και ReceiveMediator (RM).

Έπειτα, δηλώνονται οι μεταβλητές καταστάσεων :

- m: Κενό μήνυμα τύπου mpi_message. Χρησιμοποιείται ως είσοδο στις ενέργειες RECEIVE και SEND.
- runs: Ακέραιος αριθμός που υποδεικνύει τον αριθμό επαναλήψεων του χρονοδιαγράμματος ενός αυτόματου σύνθεσης. Αρχικοποιείται με $\text{MPI\_Size}()^2$ αφού ο μέγιστος αριθμός μηνυμάτων που θα υπάρξουν κατά την ανταλλαγή μηνυμάτων μεταξύ n διεργασιών είναι φραγμένος από το n^2 (Κάθε μία από τις n διεργασίες στέλνει μέχρι και n-1 μηνύματα).

Το πρόγραμμα μεταβάσεων και τροχιών (schedule) αρχίζει με την πυροδότηση της εσωτερικής ενέργειας init για αρχικοποίηση του αυτόματου διεργασίας. Στη συνέχεια γίνεται προετοιμασία των μηνυμάτων που θα αποσταλούν μέσω της εσωτερικής ενέργειας prep_messages. Στον βρόγχο που ακολουθεί εκτελούνται με σειρά η εξέλιξη της τροχιάς του αυτόματου διεργασίας κατά μία μονάδα και αποστολή μηνυμάτων προς όλες τις διεργασίες. Έπειτα γίνεται λήψη των μηνυμάτων που έχουν σταλεί με παραλήπτη την συγκεκριμένη διεργασία. Κάθε επανάληψη τελειώνει με την εκτέλεση των ενεργειών collect, announceTimeout οι οποίες θα εκτελεστούν εάν είναι ενεργές τη συγκεκριμένη χρονική στιγμή. Μετά από την αποστολή μηνυμάτων ακολουθεί μια καθυστέρηση στο αυτόματο SM ούτως ώστε να υπάρχει χρόνος για μεταφορά των μηνυμάτων στο κανάλι. Η διάρκεια καθυστέρησης είναι ενδεικτική. Στο Σχήμα 2.8 που ακολουθεί φαίνεται η υλοποίηση του αυτόματου.

Όπως αναφέρθηκε στον ορισμό της σύνθεσης αυτομάτων, με κάθε πυροδότηση μιας ενέργειας εκτελούνται οι αντίστοιχες ενέργειες στα υπόλοιπα αυτόματα όπως για παράδειγμα η ενέργεια RECEIVE από το αυτόματο καναλιού η οποία πυροδοτεί την αντίστοιχη του αυτόματου διεργασίας.

```
%%% .: 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)
components
	PR : Process(x, MPI_Rank(), d1);
	SM : SendMediator;
	RM : ReceiveMediator;
schedule
states
	m : Null[mpi_message] := nil();
	runs : Int := MPI_Size() **2 ;
do
	fire input PR.init;
	fire internal PR.prepMessages;
	for i: Nat where(i < runs)do
		follow PR.Time duration 1;
		for n : Nat where(n < MPI_Size())do
			fire output PR.SEND(m);
		od
		follow SM.DELAY duration (MPI_Size() * 10);
		for n : Nat where(n < MPI_Size())do
			m := nil();
			fire input RM.probe(n);
			fire output RM.RECEIVE(m);
		od
		fire internal PR.collect;
		fire internal PR.announceTimeout;
	od
od
```

Σχήμα 2.8 Αυτόματο σύνθεσης (Composition.tioa)

2.5.3 Μετάφραση TIOA σε κώδικα Java

Εφόσον έχει ολοκληρωθεί η υλοποίηση του αλγόριθμου και ο συντακτικός έλεγχος το μόνο που απομένει είναι η μετάφραση του αλγόριθμου από γλώσσα TIOA σε Java. Να σημειωθεί ότι, κατά τον συντακτικό έλεγχο οι μεταβλητές και τύποι που ορίζονται σε άλλα αυτόματα ή λεξιλόγια πιθανόν να θεωρούνται ως λανθασμένες από το εργαλείο TEMPO εμφανίζοντας το μήνυμα λάθους 'undefined'. Αυτό είναι φυσικό και καθόλου ανησυχητικό αφού το κάθε αυτόματο, εκτός του αυτόματου σύνθεσης, δεν γνωρίζει την ύπαρξη άλλων αυτόματων. Με την επιλογή 'Build Together' που εμφανίζεται επιλέγοντας εργασία μας από τον navigator επιλύονται τέτοιου είδους προβλήματα. Συνεχίζοντας, για την ορθή μετάφραση πρέπει πρώτα να επιλεχθεί το πρωτόκολλο επικοινωνίας που χρησιμοποιείται (MPI/TCP) ως ακολούθως:

1. Επιλέγουμε από το menu bar Windows και ακολούθως Preferences.
2. Στο νέο παράθυρο που εμφανίζεται, στην κατηγορία Tempo Plugins επιλέγουμε το Java Generator.
3. Στη συνέχεια επιλέγουμε το MPI ή TCP (ανάλογα με τις ανάγκες του αυτόματου) και πατάμε OK.

Τέλος, από την εργαλειοθήκη επιλέγουμε το εικονίδιο tempo2java και στην κονσόλα εμφανίζεται ο πηγαίος κώδικας σε Java. Παράλληλα, δημιουργείται ο φάκελος Composition.java ο οποίος περιέχει όλα τα αρχεία που έχουν δημιουργηθεί.
Έχει αποδειχθεί [1] ότι η μετάφραση διατηρεί την ορθότητα της πηγαίας προδιαγραφής.

2.6 Κατανεμημένο Σύστημα PlanetLab

Η αξιολόγηση του αλγόριθμου Reader-Writer-Recon [7, 8] έχει γίνει στο κατανεμημένο δίκτυο PlanetLab και επομένως ακολουθεί μια σύντομη αναφορά σε αυτό και του τρόπου λειτουργίας του.

2.6.1 Εισαγωγή στο PlanetLab

Το PlanetLab [10, 12] είναι ένα διεθνές ερευνητικό δίκτυο το οποίο χρησιμοποιείται για την ανάπτυξη και τη χρήση υπηρεσιών παγκόσμιας κλίμακας. Έχει χρησιμοποιηθεί για

ανάπτυξη νέων τεχνολογιών για κατανεμημένη αποθήκευση, χαρτογράφηση δικτύων, peer-to-peer συστήματα, και επεξεργασία ερωτημάτων (query processing).

Το PlanetLab περιλαμβάνει κόμβους από όλο τον κόσμο και δίνεται η δυνατότητα αξιοποίησης πόρων από όλο το σύστημα με σκοπό την εκτέλεση ερευνητικών πειραμάτων μεγάλης κλίμακας. Η χρήση ενός τόσου μεγάλου σε κλίμακα συστήματος είναι σημαντική, αφού παρέχει ένα ρεαλιστικό δίκτυο με την παρουσία συμφόρησης, καταρρεύσεων και ποικίλων συμπεριφορών από τους κόμβους [12].

2.6.2 Λειτουργία του PlanetLab

Κάθε οργανισμός που λαμβάνει μέρος στο PlanetLab, δεσμεύεται με την παροχή δύο ή περισσότερων κόμβων, οι οποίοι είναι προσβάσιμοι στα υπόλοιπα μέλη του PlanetLab. Το Πανεπιστήμιο Κύπρου έχει το δικό του site στο ευρωπαϊκό τμήμα της διαθέσιμης πλατφόρμας δοκιμών του PlanetLab (PlanetLab Europe) το οποίο προσφέρει τρεις κόμβους.

Οι χρήστες του PlanetLab, διακρίνονται στις τρεις πιο κάτω κατηγορίες:

- Principal Investigator (PI): Ο PI είναι το άτομο που είναι υπεύθυνο για την διαχείριση των μερισμάτων (slices) όπως και των χρηστών του οργανισμού. Είναι το μοναδικό άτομο στο οποίο παρέχεται η δυνατότητα ενεργοποίησης, απενεργοποίησης και διαγραφής λογαριασμών χρηστών, δημιουργίας και διαγραφής μερισμάτων, καθώς και ανάθεση μερισμάτων στους χρήστες .
- Technical Contact (Tech Contact): Το άτομο όπου απαιτείται να υπάρχει σε κάθε οργανισμό και είναι υπεύθυνο για την εγκατάσταση, συντήρηση και παρακολούθηση των κόμβων του οργανισμού του.
- User: Οποιοδήποτε άτομο το οποίο δημιουργεί εφαρμογές στο PlanetLab, θεωρείται απλός χρήστης. Οι χρήστες του PlanetLab κατέχουν έναν έγκυρο λογαριασμό και τους έχει παραχωρηθεί ένα μέρισμα από τον PI του οργανισμού.

Κάθε οργανισμός μπορεί να δημιουργήσει μέχρι και 10 μερίσματα με διάρκεια ζωής ενός μήνα το καθένα με την δυνατότητα ανανέωσης. Αποτυχία ανανέωσης σηματοδοτεί την λήξη του μερίσματος και αποδέσμευση των πόρων. Ένας κόμβος γίνεται διαθέσιμος σε περισσότερους από ένα χρήστες με επιπτώσεις στην απόδοσή

του. Οι κόμβοι θεωρούνται μη-αξιόπιστοι αφού υπάρχει η πιθανότητα κατάρρευσης και απρόσμενης επανεκκίνησης, χωρίς όμως να χάνονται τα δεδομένα.

Η χρήση της πλατφόρμας του PlanetLab προαπαιτεί την δημιουργία λογαριασμού μέσω της επίσημης σελίδας του PlanetLab [10] με την ηλεκτρονική διεύθυνση , η οποία αποτελεί το συνθηματικό για την πρόσβαση στο PlanetLab. Με την εισαγωγή των απαραίτητων πληροφοριών και το όνομα του οργανισμού όπου ανήκει ο χρήστης ως το όνομα του site, αποστέλλεται αίτημα στο PI του οργανισμού του, ο οποίος αναθέτει στον χρήστη κάποιο μέρισμα, αφού επιβεβαιώσει το αίτημα και ενεργοποιήσει τον λογαριασμό του.

Για λόγους ασφαλείας, η δημιουργία ενός ζεύγους κρυπτογραφημένων κλειδιών SSH για την πρόσβαση στους κόμβους του PlanetLab τίθεται απαραίτητη. Αυτό γίνεται εφικτό με την χρήση του προγράμματος ssh-keygen σε ένα ασφαλές σύστημα UNIX εκτελώντας την εξής εντολή:

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

όπου id_rsa είναι το όνομα του κλειδιού που δημιουργείται.. Έπειτα ζητείται η εισαγωγή ενός συνθηματικού, το οποίο θα χρησιμοποιείται για την πρόσβαση στους κόμβους. Με την εκτέλεση της εντολής δημιουργείται το ζεύγος public-private κλειδιών. Το public_key id_rsa.pub πρέπει να ανεβεί στον λογαριασμό PlanetLab, με την χρήση της ιστοσελίδας του, κάτω από την επιλογή My Account→One Key.

Με την δημοσίευση του public key στην ιστοσελίδα, μεταφέρεται το κλειδί σε όλες τις μηχανές που υπάρχουν στο μέρισμα του χρήστη και καθίσταστε εφικτή η σύνδεση σε αυτές. Η πρόσθεση ή αφαίρεση κόμβων από ένα μέρισμα γίνεται και πάλι μέσω της ιστοσελίδας του PlanetLab κάτω από την επιλογή My Slices→Add Nodes. Για την σύνδεση σε μία μηχανή εκτελείται η εντολή

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

όπου slice_name είναι το όνομα του μερίσματος στο οποίο συμμετέχει ο χρήστης και με την επιλογή -i προσδιορίζουμε την τοποθεσία του κρυπτογραφημένο κλειδιού του χρήστη. Το hostname είναι η διεύθυνση του κόμβου στον οποίο επιδιώκεται η σύνδεση. Ακολούθως ζητείται το συνθηματικό που έχει ορίσει ο χρήστης κατά τη δημιουργία του κλειδιού και ολοκληρώνεται η διαδικασία σύνδεσης.

Οι μηχανές έχουν εγκατεστημένο το Fedora Core 8, ένα λειτουργικό σύστημα για Linux, και κανένα άλλο πρόγραμμα. Για την εκτέλεση του αλγορίθμου Reader-Writer-

Recon [7, 8] χρειάστηκε η εγκατάσταση του μεταγλωττιστή της Java με την διαδικασία που περιγράφετε στο Παράρτημα Γ.

Επιπλέον, με την χρήση της ακόλουθης εντολής παρέχεται η δυνατότητα μεταφοράς αρχείων προς τις μηχανές του μερίσματος :

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

Η μεταβλητή file.ex αποτελεί το όνομα του αρχείου και η μεταβλητή slice_name@host_name το όνομα του μερίσματος και μηχανής όπου θα γίνει η μεταφορά.

Περισσότερες πληροφορίες για την χρήση του PlanetLab και τις δυνατότητες που μπορεί να προσφέρει , υπάρχουν στα εγχειρίδια χρήσης του που είναι διαθέσιμα στην ιστοσελίδα του [10].

2.7 Επιπρόσθετοι Ορισμοί

2.7.1 Συστήματα Κατανεμημένης Κοινόχρηστης μνήμης

Οι συμμετέχοντες (διεργασίες) σε ένα τέτοιο σύστημα επικοινωνούν μέσω εικονικής κοινόχρηστης μνήμης, η οποία αποτελείται από εικονικό χώρο διευθύνσεων από τις εικονικές μνήμες του κάθε ενός. Οι εικονικές αυτές διευθύνσεις μπορούν να οριστούν ως Κατανεμημένα Ατομικά Αντικείμενα. Τα αντικείμενα αυτά είναι κατανεμημένα σε διαφορετικές τοποθεσίες και η συνέπεια τους εξασφαλίζεται μέσω της ατομικής πρόσβασης τους. Η πρόσβαση στα αντικείμενα γίνεται με βάση τους περιορισμούς κάποιου μοντέλου ταυτοχρονισμού. Τα μοντέλα αυτά ορίζονται ως:

- Μοντέλο ενός γραφέα και ενός αναγνώστη (SWSR) : δεν επιτρέπεται ταυτόχρονη λειτουργία γραφής ή ανάγνωσης σε ένα αντικείμενο
- Μοντέλο ενός γραφέα και πολλών αναγνωστών (SWMR) : επιτρέπεται ταυτόχρονη λειτουργία ανάγνωσης αλλά όχι γραφής
- Μοντέλο πολλών γραφέων και πολλών αναγνωστών (MWMR): επιτρέπεται ταυτόχρονη λειτουργία γραφής ή ανάγνωσης σε ένα αντικείμενο.

2.7.2 Έννοια της Ατομικότητας

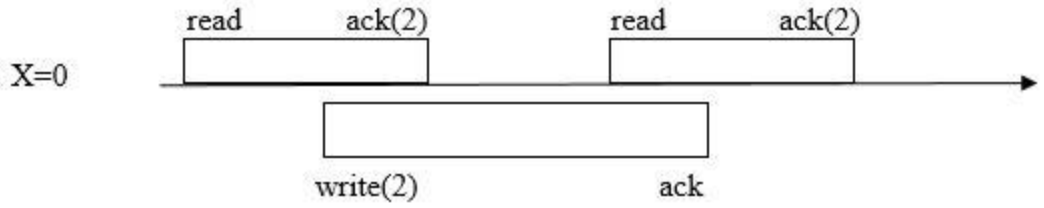
Στην παρούσα Διπλωματική Εργασία γίνεται μελέτη αλγόριθμου ο οποίος προσφέρει τις λειτουργίες MWMR σε κατανεμημένα αντικείμενα εντός ενός δυναμικού Κατανεμημένου Συστήματος. Η έννοια του δυναμικού συστήματος αναλύεται στην επόμενη υποενότητα.

Μια λειτουργία ανάγνωσης read επιστρέφει την πιο πρόσφατη τιμή ενός τέτοιου αντικειμένου ενώ μια λειτουργία γραφής write(v) αντικαθιστά την τιμή ενός αντικειμένου με την v και επιστρέφει ένα μήνυμα το οποίο υποδεικνύει την επιτυχία γραφής. Κάθε λειτουργία έχει σημείο επίκλησης και σημείο επιστροφής, δηλαδή ένα σημείο έναρξης και ένα ολοκλήρωσης. Με βάση αυτό, μπορεί να τεθεί μια σχέση προτεραιότητας μεταξύ δύο λειτουργιών α, β όπου η α προηγείται της β εάν ολοκληρωθεί πριν το σημείο επίκλησης της β. Δύο λειτουργίες θεωρούνται ταυτόχρονες ανν καμία δεν προηγείται της άλλης.

Κάθε λειτουργία, γραφής ή ανάγνωσης συμβαίνει σε κάποιο σημείο μεταξύ της επίκλησης και της επιστροφής της. Προχωρώντας στη έννοια της ατομικότητας, έχοντας υπόψη τα πιο πάνω, μπορούν να οριστούν οι παρακάτω συνθήκες ατομικής πρόσβασης :

- Μια λειτουργία ανάγνωσης σε ένα αντικείμενο επιστρέφει την τιμή της τελευταίας ολοκληρωμένης γραφής στο αντικείμενο ή την τιμή μιας γραφής ταυτόχρονης με την ανάγνωση.
- Αν μια λειτουργία ανάγνωσης α_1 επιστρέψει την τιμή που γράφτηκε από μία λειτουργία γραφής ε_1 και μία λειτουργία ανάγνωσης α_2 επιστρέψει την τιμή που γράφτηκε από την λειτουργία γραφής ε_2 και η α_1 δεν προηγείται της α_2 τότε η ε_2 προηγείται της ε_1.

Η διατήρηση των συνθηκών ατομικής πρόσβασης αποτελεί πρόκληση των αλγορίθμων που αφορούν κατανεμημένα ατομικά αντικείμενα.



Σχήμα 2.9 Στο παράδειγμα οι ενέργειες ανάγνωσης επιστρέφουν την τιμή μίας ταυτόχρονης ενέργειας γραφής διατηρώντας τις συνθήκες ατομικότητας

2.7.3 Καθυστέρηση λειτουργιών

Για την διατήρηση των συνθηκών ατομικής πρόσβασης, σε αρκετούς κατανεμημένους αλγόριθμους οι διεργασίες επικοινωνούν μεταξύ τους προτού εκτελεστεί μια λειτουργία ανάγνωσης ή γραφής. Στο σημείο αυτό αναδύεται ο όρος καθυστέρηση λειτουργιών (operation latency). Μια διεργασία εκτελεί ένα *γύρο επικοινωνίας* για μία λειτουργία σε ένα αντικείμενο όταν

- Στέλνει ένα μήνυμα *m* σχετικό με τη λειτουργία σε ένα υποσύνολο διεργασιών (πιθανόν και όλες)
- Όποια διεργασία παραλάβει το *m*, απαντά στον αποστολέα με τις απαραίτητες πληροφορίες
- Η διεργασία συλλέγει 'αρκετές' απαντήσεις προτού προχωρήσει με την λειτουργία

Η *καθυστέρηση λειτουργιών* αποτελεί τον αριθμό των γύρων επικοινωνίας που χρειάζεται για την ολοκλήρωση μιας λειτουργίας, όπου θεωρείται και ως το κόστος εκτέλεσης της.

2.7.4 Συστήματα Απαρτίας

Όπως αναφέρθηκε, μια διεργασία συλλέγει 'αρκετές' απαντήσεις προτού προχωρήσει με μία λειτουργία. Οι απαντήσεις που απαιτούνται είναι ανάλογες του συστήματος απαρτίας που χρησιμοποιείται. Με τον όρο απαρτία εννοούμε τον ελάχιστο αριθμό μελών ενός συνόλου που απαιτούνται για να παρθεί μια απόφαση σχετικά με την ομάδα. Για παράδειγμα, η πλειοψηφία (majority) αποτελεί ένα σύστημα απαρτίας και

συγκεκριμένα γίνεται χρήση της στον αλγόριθμο Dynastore [7, 8]. Στο σύστημα πλειοψηφίας αναμένεται από ένα σύνολο *n* μελών ανταπόκριση από τουλάχιστον *n*/2+1 μέλη προτού ληφθεί μία απόφαση. Με την χρήση των συστημάτων απαρτίας επιτυγχάνεται η συνέπεια. Όσο πιο μικρό είναι το μέγεθος της απαρτίας που χρησιμοποιείται σε ένα σύστημα τόσο πιο μικρό το κόστος επικοινωνίας αλλά γίνεται πιο επιρρεπές σε σφάλματα.

2.7.5 Κατανεμημένα Συστήματα

Τα κατανεμημένα συστήματα μπορούν να χωριστούν σε δύο κατηγορίες, στατικά και δυναμικά. Ένα στατικό σύστημα χαρακτηρίζεται από την αμετάβλητη τοπολογία του δικτύου επικοινωνίας. Σε ένα τέτοιο σύστημα γίνεται χρήση στατικών συστημάτων απαρτίας. Ένα παράδειγμα πρωτόκολλου υλοποίησης SWMR/MWMR Ατομικών αντικειμένων σε Στατικό Κατανεμημένο Σύστημα είναι ο αλγόριθμος ABD [18]. Κάθε λειτουργία γραφής στον αλγόριθμο MWMR ABD έχει κόστος δύο γύρους επικοινωνίας όπως και κάθε λειτουργία ανάγνωσης με σκοπό την διατήρηση των συνθηκών ατομικής πρόσβασης. Ο αλγόριθμος κάνει χρήση συστημάτων απαρτίας.
Το ίδιο όμως δεν συμβαίνει στα δυναμικά κατανεμημένα συστήματα. Τα συστήματα αυτά χαρακτηρίζονται από τις αλλαγές στην τοπολογία του δικτύου, κινητικότητα κόμβων/διεργασιών, μεταβαλλόμενα σύνολα ενεργών κόμβων/διεργασιών, ευρύ φάσμα αποτυχιών/σφαλμάτων και χρονικές διαφοροποιήσεις. Η διατήρηση της συνέπειας λοιπόν κατά την διάρκεια αναδιαμόρφωσης αλλά και της αναδιαμόρφωσης χωρίς την ακύρωση η μπλοκαρίσματος των λειτουργιών σε εξέλιξη αποτελεί πρόκληση.

Κεφάλαιο 3

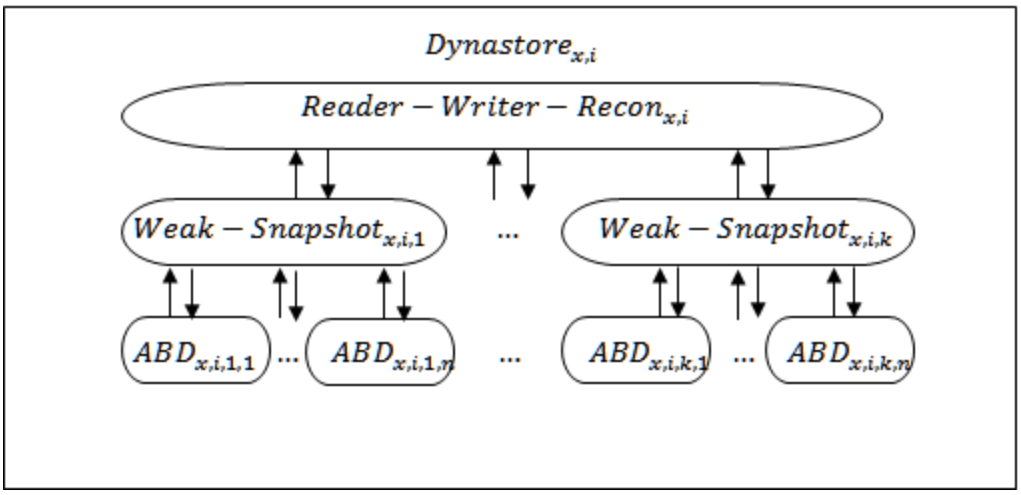
Περιγραφή Αλγόριθμου Reader-Writer-Recon

3.1 Επισκόπηση Αλγορίθμου Reader-Writer-Recon

Στο σημείο αυτό θα παρουσιάσουμε την αναδιαμόρφωση της απαρτίας ως ένα μέρος της λειτουργίας ανάγνωσης και γραφής. Ο αλγόριθμος αποτελεί ένα ολοκληρωμένο σύστημα αναδιαμόρφωσης της απαρτίας το οποίο δέχεται ως είσοδο ένα σύνολο από αλλαγές. Η νέα σύνθεση της απαρτίας μπορεί να αποτελείται από πολλαπλά σύνολα αλλαγών τα οποία συγκεντρώθηκαν μαζί. Σε αντίθεση με τον αλγόριθμο RAMBO [19] όπου η αναδιαμόρφωση αποφασιζόταν μέσα από ένα σύνολο από διαμορφώσεις που προτείνονταν παράλληλα, η τελική σύνθεση της απαρτίας εξαρτάτε από την παρούσα σύνθεση του συστήματος και όλων των αλλαγών που έχουν προταθεί από τα μέλη της απαρτίας.

Ένα σημαντικό πλεονέκτημα αυτής της αυξητικής αναδιαμόρφωσης είναι ότι αναιρεί την έννοια της ομοφωνίας. Δεν είναι απαραίτητο από τις διεργασίες να συμφωνούν σε μια κοινή σύνθεση της απαρτίας. Ωστόσο απαιτείται προσοχή για να διασφαλιστεί ότι η μερική σύνθεση που κατέχει η κάθε διεργασία βγάζει νόημα. Να σημειωθεί ότι ο αλγόριθμος ανέχεται μόνο σφάλματα κατάρρευσης και όχι άλλου είδους κακόβουλων σφαλμάτων.

Ο Dynastore [7, 8] απαρτίζεται από τον αλγόριθμο Reader-Writer-Recon ο οποίος επικοινωνεί με όσες διεργασίες WeakSnapshot όσες και οι παράλληλες ενεργές συνθέσεις του συστήματος.



Σχήμα 3.1 : Η αρχιτεκτονική του Dynastore για ένα αντικείμενο x σε μία διεργασία i αποτελείται από τα αυτόματα Reader-Writer-Recon, WeakSnapshot και ABD. Τα κανάλια επικοινωνίας μεταξύ των αυτόματων παραλείπονται για λόγους απλότητας.

3.2 Ορισμοί

Όπως προαναφέρθηκε, ο αλγόριθμος Dynastore [7, 8] υποστηρίζει την αυξητική αναδιαμόρφωση τους σύνθεσης του συστήματος με την προσθήκη ή αφαίρεση μελών από την παρούσα σύνθεση κατά την μετάβαση στη νέα σύνθεση. Μια τέτοια αναδιαμόρφωση παρουσιάζεται ως ακολούθως :

Ορίζεται το σύνολο αλλαγών changes ως $\{+,-\}$ x I όπου I είναι το σύνολο των μελών (διεργασιών) και μία αλλαγή παρουσιάζεται ως $\{+,i\}$ ή $\{-,j\}$ με την έννοια ότι εισάγεται το μέλος με αναγνωριστικό i και αφαιρείται το μέλος j . Με τον όρο σύνθεση (view) ή διαμόρφωση (configuration) αναφερόμαστε σε ένα σύνολο από αλλαγές. Μια αλλαγή w εντάσσεται στην εκτέλεση εάν μια ενέργεια $\text{reconfiguration}_i(c)$ εκτελεστεί σε μία διεργασία i και ισχύει $w \in c$.

Μια αυξητική αναδιαμόρφωση παίρνει ως παραμέτρους ένα σύνολο από αλλαγές. Υποθέτουμε ότι μια μη-κενή σύνθεση είναι γνωστή κατά την αρχικοποίηση μίας διεργασίας $i \in I$. Μια διεργασία i είναι ενεργή εάν δεν έχει καταρρεύσει κατά την διάρκεια εργασίας της και δεν έχει προταθεί κάποια αναδιαμόρφωση κατά την οποία η συγκεκριμένη διεργασία έχει αφαιρεθεί , $\{-,i\}$. Σε περίπτωση που είναι ενεργή, μια διεργασία μπορεί να εκτελέσει άπειρα βήματα.

Επιπλέον, στην περιγραφή του αλγορίθμου γίνεται αναφορά σε αρκετούς νέους όρους μερικοί εκ των οποίων είναι οι εξής :

Χρονοσφραγίδα (tag): Μια χρονοσφραγίδα ορίζεται ως $T = \mathbb{N} \times I$ όπου $\mathbb{N}$ το σύνολο των φυσικών αριθμών και I το σύνολο των μελών του συστήματος. Εφαρμόζεται λεξικογραφική ταξινόμηση πάνω στις χρονοσφραγίδες. Η αρχική της τιμή είναι 0 για κάθε διεργασία RWR (Reader-Writer-Recon)

Τύπος λειτουργίας (op-type) : Δηλώνει την ενέργεια που θα εκτελέσει μια διεργασία RWR και ισχύει op-type ϵ {read, write, recon, none}.

Στάδιο (stage) : Δηλώνει το στάδιο στο οποίο βρίσκεται η συγκεκριμένη διεργασία στην προσπάθεια εκτέλεσης της ενέργειας της. Stage ϵ { idle, traverse, traverse_rec1, traverse_rec2, traverse_fix, pick_new_ts, update, update_done, scan, scan_done, contactq_start, contactq, notifyq, notifyq_done } και αρχικοποιείται σε idle.

Φάση (phase): Η φάση στην οποία βρίσκεται η διεργασία. phase ϵ {query, propagate, none} και αρχικοποιείται σε query.

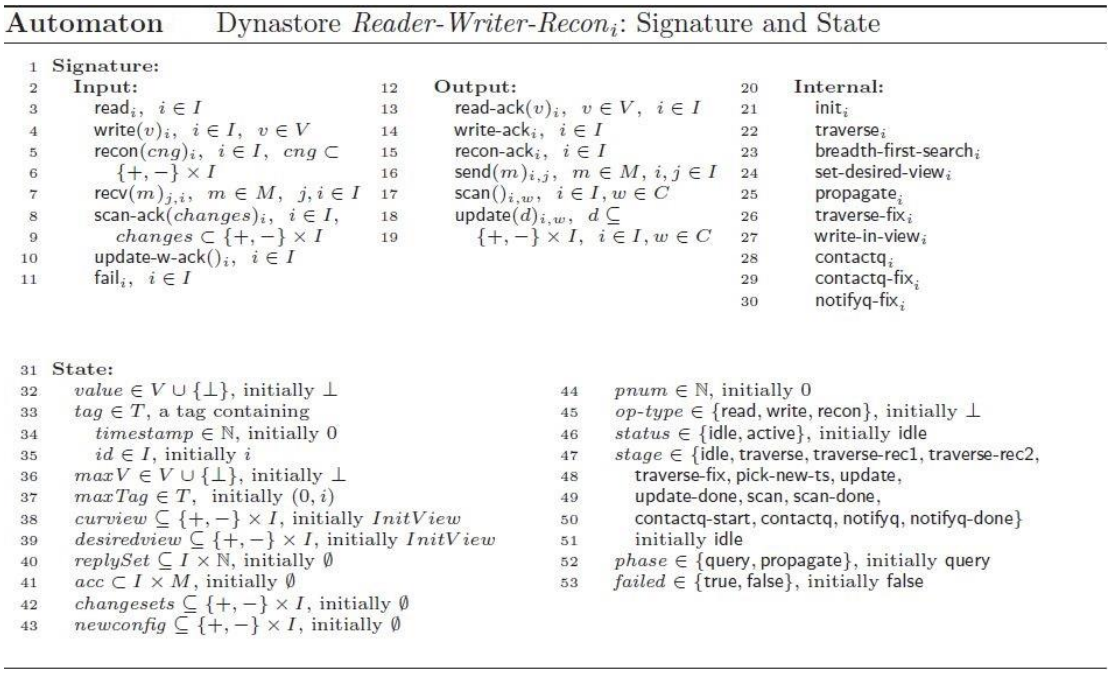
Πλειοψηφία (majority) : Έστω ότι N το σύνολο των διεργασιών/μελών ενός συστήματος. Τότε πλειοψηφία χαρακτηρίζεται ως $M \geq N/2 + 1$.

3.3 Περιγραφή λειτουργίας ενός Reader-Writer- Recon

Μια διεργασία Reader-Writer-Recon της οποία η υπογραφή παρατίθεται στο Σχήμα 3.2 είναι υπεύθυνη για τις λειτουργίες υψηλού επιπέδου του αλγόριθμου Dynastore και υλοποιεί λειτουργία multi-writer/multi-reader σε ένα δυναμικό σύστημα. Συγκεκριμένα, η υπογραφή του αυτόματου περιέχει τις ενέργειες read, write, και recon και αποθηκεύει μία τιμή $v_0 \, \epsilon \, V$. Όταν ενεργοποιηθεί μια ενέργεια read ή write(v) το σύστημα ανταποκρίνεται με τις ενέργειες read-ack(v) και write-ack() αντίστοιχα που σηματοδοτούν την ολοκλήρωση της λειτουργίας.

3.3.1 Μερικώς διατεταγμένες διαμορφώσεις

Κάθε λειτουργία read/write περνά από τις δύο φάσεις query και prop. Σε κάθε φάση η διεργασία επικοινωνεί με μία πλειοψηφία διεργασιών στο σύστημα μέσω της ενέργειας



Automaton Dynastore *Reader-Writer-Recon*$_i$: Signature and State

Signature:

Input:
- read$_i$, $i \in I$
- write$(v)_i$, $i \in I$, $v \in V$
- recon$(cng)_i$, $i \in I$, $cng \subset \{+,-\} \times I$
- recv$(m)_{j,i}$, $m \in M$, $j, i \in I$
- scan-ack$(changes)_i$, $i \in I$, $changes \subset \{+,-\} \times I$
- update-w-ack$()_i$, $i \in I$
- fail$_i$, $i \in I$

Output:
- read-ack$(v)_i$, $v \in V$, $i \in I$
- write-ack$_i$, $i \in I$
- recon-ack$_i$, $i \in I$
- send$(m)_{i,j}$, $m \in M$, $i, j \in I$
- scan$()_{i,w}$, $i \in I, w \in C$
- update$(d)_{i,w}$, $d \subseteq \{+,-\} \times I$, $i \in I, w \in C$

Internal:
- init$_i$
- traverse$_i$
- breadth-first-search$_i$
- set-desired-view$_i$
- propagate$_i$
- traverse-fix$_i$
- write-in-view$_i$
- contactq$_i$
- contactq-fix$_i$
- notifyq-fix$_i$

State:
- $value \in V \cup \{\perp\}$, initially $\perp$
- $tag \in T$, a tag containing
  - $timestamp \in \mathbb{N}$, initially 0
  - $id \in I$, initially i
- $maxV \in V \cup \{\perp\}$, initially $\perp$
- $maxTag \in T$, initially $(0, i)$
- $curview \subseteq \{+,-\} \times I$, initially $InitView$
- $desiredview \subseteq \{+,-\} \times I$, initially $InitView$
- $replySet \subseteq I \times \mathbb{N}$, initially $\emptyset$
- $acc \subset I \times M$, initially $\emptyset$
- $changesets \subseteq \{+,-\} \times I$, initially $\emptyset$
- $newconfig \subseteq \{+,-\} \times I$, initially $\emptyset$
- $pnum \in \mathbb{N}$, initially 0
- $op\text{-}type \in$ {read, write, recon}, initially $\perp$
- $status \in$ {idle, active}, initially idle
- $stage \in$ {idle, traverse, traverse-rec1, traverse-rec2, traverse-fix, pick-new-ts, update, update-done, scan, scan-done, contactq-start, contactq, notifyq, notifyq-done} initially idle
- $phase \in$ {query, propagate}, initially query
- $failed \in$ {true, false}, initially false

Σχήμα 3.2 Υπογραφή και μεταβλητές καταστάσεων Reader-Writer-Recon [7]

contactq ούτως ώστε να αποκτήσει την πιο πρόσφατη σύνθεση του συστήματος. Εφόσον, όπως έχει αναφερθεί, η πιο πρόσφατη σύνθεση του συστήματος αποτελείται από την αρχική και τυχόν αναδιαμορφώσεις που έχουν προκύψει τότε μπορεί να παρουσιαστεί ως ένας άκυκλος κατευθυνόμενος γράφος στον οποίο οι κόμβοι αναπαριστούν μια σύνθεση και οι ακμές τις αναδιαμορφώσεις που απαιτούνται για την μετάβαση σε αυτή την σύνθεση.

Κάθε λειτουργία read/write(v)/recon(cnf) διασχίζει τον γράφο με την βοήθεια της κατά πλάτος αναζήτησης και καταλήγει στην πιο πρόσφατη σύνθεση όταν διασχίσει όλους τους κόμβους. Εξού και η ενέργειες traverse, bread-first-search, set-desired-view, propagate, traverse-fix μέσω των οποίων εκτελείται αυτή η επαναλαμβανόμενη διάσχιση του γράφου μέχρι την απόκτηση της τελικής σύνθεσης του συστήματος . Η ρίζα του γράφου ονομάζεται *front* και η επιθυμητή σύνθεση του συστήματος desiredview περιέχει την πιο πρόσφατη σύνθεση. Ας σημειωθεί ότι η *desiredview* είναι διαφορετική από την *curview* (παρούσα σύνθεση) μόνο εάν η διάσχιση του γράφου αποτελεί μέρος της λειτουργίας αναδιαμόρφωσης recon λόγω του ότι η *desiredview* θα αποτελεί ένωση της *curview* και *newconfig*(νέα σύνθεση). Υπόψη, ότι μία διεργασία μπορεί να διασχίσει τον γράφο μόνο εάν αποτελεί μέλος της παρούσας σύνθεσης. Η

διάσχιση του γράφου αρχίζει με την πιο κοντινή σύνθεση του συστήματος στην παρούσα, δηλαδή την σύνθεση που απαιτεί τις λιγότερες αλλαγές.

3.3.2 Προσθήκη νέας σύνθεσης

Σε περίπτωση που η παρούσα σύνθεση του συστήματος δεν περιέχει όλες τις αλλαγές του συστήματος, με άλλα λόγια $w \neq desiredview$, τότε οι αλλαγές αυτές τοποθετούνται στον γράφο ως μια νέα ακμή από την παρούσα σύνθεση μέσω της ενέργειας update. Ωστόσο η προσθήκη περισσότερων αλλαγών δεν μπορεί να επιτευχθεί μέσω της ενέργειας αυτής αλλά είναι εφικτή κατά την διάρκεια της διάσχισης του γράφου (traverse). Σε αντίθετη περίπτωση, δηλαδή όταν η διεργασία κατέχει την πιο πρόσφατη σύνθεση, ξεκινά η φάση query κατά την οποία μια διεργασία αναζητά αλλαγές στο σύστημα μέσω του αυτόματου Weak Snapshot, η ανάλυση του οποίου δεν αποτελεί μέρος αυτής της Ατομικής Διπλωματικής Εργασίας. Η διαδικασία επαναλαμβάνεται μέχρι να μην υπάρχουν περαιτέρω αναδιαμορφώσεις. Στο σημείο αυτό, το αυτόματο προχωρεί στην επόμενη φάση, propagate, κατά την οποία ενημερώνει το σύστημα για την πιο πρόσφατη σύνθεση που κατέχει. Συγκρίνοντας τις 2 φάσεις του αυτόματου παρατηρούμε ότι η φάση query αναζητά καινούργιες διαμορφώσεις στο σύστημα προτού έρθει σε επαφή με τα μέλη του συστήματος ενώ με τη φάση propagate έρχεται σε επαφή με το σύστημα και αναζήτηση για αλλαγές αφού επιλέξει μια νέα χρονοσφραγίδα. Η διαφορετική προσέγγιση που ακολουθεί η φάση propagate πετυχαίνει την μεταφορά της τιμής που θα γραφεί στην πιο πρόσφατη σύνθεση του συστήματος και διατήρηση της συνθήκης ατομικότητας.

3.3.3 Επικοινωνία με απαρτία

Με την εκτέλεση της ενέργειας contactq γίνεται αρχικοποίηση του πεδίου acc, το στάδιο εκτέλεσης αλλάζει σε contactq και ο αριθμός της φάσης phase num αυξάνεται κατά ένα ως ένδειξη ότι η λειτουργία προχώρησε σε επόμενη φάση. Με την παραλαβή μηνύματος με την αντίστοιχη φάση και αριθμό φάσης από την ενέργεια recv αποθηκεύεται ο αποστολέας στο πεδίο acc. Αφού ληφθεί απάντηση από την πλειοψηφία ενεργοποιείται η ενέργεια contactq-fix για ενημέρωση της χρονοσφραγίδας του αυτόματου με μία πιο πρόσφατη εάν υπάρχει. Ο τερματισμός της ενέργειας contact-

fix σηματοδοτεί τον τερματισμό της φάσης στην οποία βρίσκεται η διεργασία. Σε περίπτωση που δεν βρίσκεται στην φάση query και έχουν παρατηρηθεί αλλαγές στην σύνθεση του συστήματος κατά τη ενέργεια scan τότε επαναλαμβάνεται η ενέργεια traverse.

Automaton Dynastore *Reader-Writer-Recon*$_i$: Transitions 1

Internal init$_i$
  Precondition:
    $\neg failed \wedge status =$ active
    $i \in InitView.members$
  Effect:
    $status \leftarrow$ active

Input fail$_i$
  Effect:
    $failed \leftarrow$ true

Input read$_i$
  Effect:
    if $\neg failed \wedge status =$ active **then**
      $op\text{-}type \leftarrow$ read
      $newconfig \leftarrow \emptyset$
      $stage \leftarrow$ traverse

Output read-ack$(v)_i$
  Precondition:
    $\neg failed \wedge status =$ active
    $op\text{-}type =$ read
    $stage =$ notifyq-done
    $v = maxV$
  Effect:
    $stage \leftarrow$ idle

Input write$(v)_i$
  Effect:
    if $\neg failed \wedge status =$ active **then**
      $value \leftarrow v$
      $op\text{-}type \leftarrow$ write
      $newconfig \leftarrow \emptyset$
      $stage \leftarrow$ traverse

Output write-ack$_i$
  Precondition:
    $\neg failed \wedge status =$ active
    $op\text{-}type =$ write
    $stage =$ notifyq-done
  Effect:
    $stage \leftarrow$ idle

Input recon$(cnf)_i$
  Effect:
    if $\neg failed \wedge status =$ active **then**
      $op\text{-}type \leftarrow$ recon
      $newconfig \leftarrow cng$
      $stage \leftarrow$ traverse

Output recon-ack$()_i$
  Precondition:
    $\neg failed \wedge status =$ active
    $op\text{-}type =$ recon
    $stage =$ notifyq-done
  Effect:
    $stage \leftarrow$ idle

Output send$(\langle mtype, ph, view, pn, v, t\rangle)_{i,j}$
  Precondition:
    $\neg failed \wedge status =$ active
    $[stage =$ contactq $\wedge\ j \in w.members \wedge$
      $\langle mtype, ph, view, pn, v, t\rangle =$
      $\langle$ REQUEST $, phase, \perp, pnum, maxV, maxTag\rangle]$
        // send read/write request
    $[\langle j, num\rangle \in replySet \wedge$
      $\langle mtype, ph, view, pn, v, t\rangle =$
      $\langle$ REPLY $, \perp, \perp, num, value, tag\rangle]$
        // reply to read/write request
    $[notify\text{-}v \in notifySet \wedge j \in notify\text{-}v.members \wedge$
      $\langle mtype, ph, view, pn, v, t\rangle =$
      $\langle$ NOTIFY $, \perp, notify\text{-}v, \perp, \perp, \perp\rangle]$
        // send notify for a view
  Effect:
    None

Input recv$(\langle mtype, ph, view, pn, v, t\rangle)_{j,i}$
  Effect:
    if $\neg failed \wedge status =$ active **then**
      if $mtype =$REPLY **then**
        $acc \leftarrow acc \cup \{j, \langle mtype, ph, pn, v, t\rangle\}$
      if $mtype =$REQUEST **then**
        if $ph =$ propagate $\wedge\ t > tag$ **then**
          $(value, tag) \leftarrow (v, t)$
        $replySet \leftarrow replySet \cup \{(j, pn)\}$
      if $mtype =$NOTIFY **then**
        if $stage =$ notifyq **then** *// collect the replies*
          $rcvd\text{-}notify \leftarrow rcvd\text{-}notify \cup \{(j, view)\}$
        else *// received notify while stage $\neq$ notifyq*
          if $view \notin notifySet$ **then**
            // add the view the first time we receive it
            $notifySet \leftarrow notifySet \cup \{view\}$
            if $curView \subset view$ **then**
              $curView \leftarrow view$
              if $i \in view.join$ **then**
                $status \leftarrow$ active
              $stage \leftarrow$ traverse *// restart traverse* [1]

Internal notifyq-fix$_i$
  Precondition:
    $\neg failed \wedge status =$ active
    $stage =$ notifyq
    $rcvdP = \{j : (j, curview) \in rcvd\text{-}notify, j \in I\}$
    $majority(curview.members) \subseteq rcvdP$
  Effect:
    $stage \leftarrow$ notifyq-done

Σχήμα 3.3 Σχέσεις μεταβάσεων αυτόματου Reader-writer-Recon [7]

Automaton Dynastore *Reader-Writer-Recon*$_i$: Transitions 2

```
97  Internal traverse()_i
98    Precondition:
99      ¬failed ∧ status = active
100     stage = traverse
101   Effect:
102     front ← curview
103     desiredview ← curview ∪ newconfig
104     stage ← traverse-rec1

105 Internal breadth-first-search()_i
106   Precondition:
107     ¬failed ∧ status = active
108     stage = traverse-rec1
109     w = ℓ s.t. |ℓ| = min_{ℓ'∈front}{|ℓ'|}
110   Effect:
111     if i ∉ w.members then
112       status ← idle   // halt if not a member of the view
113     else
114       if w ≠ desiredview then
115         to-be-updated ← ⟨w, desiredview \ w⟩
116         stage ← update
117       else
118         phase ← query
119         stage ← scan

120 Internal set-desired-view()_i
121   Precondition:
122     ¬failed ∧ status = active
123     stage = traverse-rec2
124     changesets ≠ ∅
125   Effect:
126     front ← front \ {w}
127     for all c ∈ changesets do
128       desiredview ← desiredview ∪ c
129       front ← front ∪ {w ∪ c}
130     stage ← traverse-rec1

131 Internal propagate()_i
132   Precondition:
133     ¬failed ∧ status = active
134     stage = traverse-rec2
135     changesets = ∅
136   Effect:
137     phase ← propagate
138     if (op-type = write) then stage ← pick-new-ts
139     else stage ← contactq-start      // same ts for reads

140 Internal traverse-fix()_i
141   Precondition:
142     ¬failed ∧ status = active
143     stage = traverse-fix
144   Effect:
145     curview ← desiredview
146     stage ← notifyq

147 Output scan()_{i,w}
148   Precondition:
149     ¬failed ∧ status = active
150     stage = scan
151   Effect: None

152 Input scan-ack(changes)_{i,w}
153   Effect:
154     if ¬failed ∧ status = active then
155       changesets ← changes
156       if phase = query then
157         stage ← contactq-start
158       else                          // phase = propagate
159         if changesets ≠ ∅ then
160           stage ← traverse-rec1
161         else
162           stage ← traverse-fix

163 Internal pick-new-ts()_i
164   Precondition:
165     ¬failed ∧ status = active
166     stage = pick-new-ts
167     changesets = ∅
168   Effect:
169     maxtag ← ⟨maxtag.timestamp + 1, i⟩
170     maxv ← value
171     stage ← contactq-start

172 Internal contactq_i
173   Precondition:
174     ¬failed ∧ status = active
175     stage = contactq-start
176   Effect:
177     acc ← ∅
178     pnum ← pnum + 1
179     stage ← contactq

180 Internal contactq-fix_i
181   Precondition:
182     ¬failed ∧ status = active
183     stage = contactq
184     repliedP = {j : ⟨j, m⟩ ∈ acc ∧ m.pn = pnum}
185     majority(w.members) ⊆ repliedP
186   Effect:
187     if op-type = read then
188       t ← max_{⟨j,m⟩∈acc}{m.tag}
189       v ← v' : ⟨t, v'⟩ ∈ acc
190       if t > maxtag then
191         ⟨maxV, maxtag⟩ ← ⟨t, v⟩
192     if phase = query then
193       stage ← traverse-rec2
194     else                            // phase = propagate
195       stage ← scan

196 Output update(d)_{i,w}
197   Precondition:
198     ¬failed ∧ status = active
199     ⟨w, d⟩ = to-be-updated
200     stage = update
201   Effect:
202     to-be-updated ← ⊥

203 Input update-ack()_{i,w}
204   Effect:
205     if ¬failed ∧ status = active then
206       phase ← query
207       stage ← scan
```

Σχήμα 3.4 Σχέσεις μεταβάσεων αυτόματου Reader-writer-Recon [7] (συνέχεια)

Κεφάλαιο 4

Προδιαγραφή Αλγορίθμου Reader-Writer-Recon στο εργαλείο TEMPO

Στο κεφάλαιο αυτό θα γίνει εκτενής περιγραφή των προδιαγραφών του αυτόματου Reader-Writer-Recon, μέρος του αλγορίθμου Dynastore, στο εργαλείο TEMPO χρησιμοποιώντας Χρονισμένα Αυτόματα Εισόδου/Εξόδου. Ακολουθεί η προδιαγραφή του λεξιλογίου (Vocabulary.tioa) και του αυτόματου Reader-Writer-Recon (ReaderWriterRecon.tioa). Η ολοκληρωμένη προδιαγραφή του αλγόριθμου βρίσκεται στο Παράρτημα Β.

4.1 Αυτόματο Λεξιλόγιο

Το λεξιλόγιο του αλγορίθμου φαίνεται στο Σχήμα 4.1. Πέραν των βασικών τύπων δεδομένων που χρησιμοποιούνται από το κανάλι επικοινωνίας TCP έχουν χρησιμοποιηθεί και οι παρακάτω νέοι τύποι :

Tag: Πλειάδα δύο στοιχείων [timestamp,IP]. Το timestamp, δηλαδή την χρονοσφραγίδα, είναι ένας φυσικός αριθμός και η διεύθυνση IP αποτελεί την διεύθυνση του κόμβου στον οποίο αντιστοιχεί. Μεγαλύτερο timestamp σημαίνει πιο πρόσφατο tag.

Reply: Πλειάδα δύο στοιχείων [IP, num] και αντιστοιχεί σε μία απάντηση σε αίτηση για γραφή/ανάγνωση από ένα κόμβο με διεύθυνση IP και σε αριθμό φάσης num.

Change: Αποτελεί μια πλειάδα δύο στοιχείων [Nat, IP]. Το Change είναι μια αλλαγή στη σύνθεση του συστήματος όπου η διεύθυνση IP αφορά μία διεργασία. Για παράδειγμα, {0,0.0.0.0} αντιστοιχεί στην αφαίρεση της διεργασίας με διεύθυνση IP στην παρούσα σύνθεση του συστήματος. Μια διεργασία με IP= 0.0.0.0 θεωρείται ενεργή στην σύνθεση w εάν $\{0,0.0.0.0\} \notin w \wedge \{1,0.0.0.0\} \in w$.

View: Αποτελεί μια ακολουθία από αλλαγές Seq[Change] και αναπαριστά μια σύνθεση του συστήματος.

Notify: Μια πλειάδα δύο στοιχείων [IP,View] όπου View αποτελεί την πιο πρόσφατη σύνθεση που κατέχει η διεργασία με διεύθυνση IP. Χρησιμοποιείται κατά την μετάδοση της πιο πρόσφατης σύνθεσης στα μέλη του συστήματος.

D_Status: Αναπαριστά την κατάσταση στην οποία βρίσκεται μία διεργασία. Έχει πεδίο τιμών το σύνολο {idle,active}. Μια διεργασία μπορεί να εκτελέσει ενέργειες μόνο εάν βρίσκεται σε κατάσταση idle. Σύμφωνα με την προδιαγραφή του αλγορίθμου κατά την πρώτη εκτέλεση μιας διεργασίας περνά από την αρχική της κατάσταση 'idle' σε active εάν ληφθεί μήνυμα τύπου Notify από κάποιο μέλος του συστήματος και {1,IP} ∈ Notify.View όπου IP είναι το IP της διεργασίας. Ωστόσο, για ευκολία αξιολόγησης του αλγορίθμου, αρχική κατάσταση των διεργασιών ορίζεται η 'active'.

Op_Type: Ορίζει ένα σύνολο τιμών {read, write, recon, none} όπου συμβολίζει την λειτουργία που εκτελεί μία διεργασία και αρχική τιμή είναι η 'none'.

Phase: Αποτελεί ένα σύνολο τιμών {query, propagate, none} όπου η κάθε μία συμβολίζει την φάση στην οποία βρίσκεται μια διεργασία. Αρχική φάση ορίζεται η 'none'.

Acc: Πλειάδα τιμών [IP, m] η οποία αποθηκεύεται αφού παραληφθεί μήνυμα τύπου 'REPLY' ούτως ώστε να γνωρίζει μία διεργασία κατά πόσο απάντησε η πλειοψηφία του συστήματος σε ένα αίτημα γραφής ή ανάγνωσης. Η τιμή IP είναι την διεύθυνση της διεργασίας που απάντησε και m αποτελεί μια πλειάδα τιμών η οποία περιέχει την σύνθεση του συστήματος, φάση εκτέλεσης, τιμή και χρονοσφραγίδα που έχει αποθηκευμένα.

Stage: Αποτελεί ένα σύνολο τιμών {idle, traverse, traverse_rec1, traverse_rec2, traverse_fix, pick_new_ts, update, update_done, scan, scan_done, contactq_start, contactq, notifyq, notifyq_done} όπου η κάθε μία αναπαριστά ένα στάδιο στο οποίο μπορεί να βρίσκεται μια διεργασία κατά την εκτέλεση μίας λειτουργίας

D_message: Ο τύπος μηνύματος D_message είναι μια πλειάδα τιμών [mtype, Phase,View, pn, v, Tag] στην οποία mtype είναι μια συμβολοσειρά με τον τύπο μηνύματος {REQUEST, REPLY, NOTIFY} και την φάση, σύνθεση, αριθμό φάσης pn, τιμή v, και χρονοσφραγίδα που έχει αποθηκευμένη η διεργασία αποστολέας κατά την συγκεκριμένη χρονική στιγμή.

Update: Αποτελείται από δύο συνθέσεις [x,y] όπου η y συμβολίζει την αλλαγή που θα αποθηκευτεί στην x.

ChanMessage: ο τύπος αυτός είναι μια πλειάδα [D_message,sender,destination], όπου το στοιχείο D_message αποτελεί τις πληροφορίες του μηνύματος, ο sender και destination αντιπροσωπεύει την IP διεύθυνση του αποστολέα και παραλήπτη αντίστοιχα.

```
%%% TCPObjects Voc
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 dynastore
    imports TCPNodeVoc
    types
        Tag          : Tuple[timestamp:Nat,IP:Node],
        Reply        : Tuple[IP:Node,num:Nat],
        change       : Tuple[act: Nat,IP:Node], % <<+,->,IP>
        View         : Seq[change],
        Notify       : Tuple[sender:Node,view : View],
        D_Status     : Enumeration [idle,active],
        Op_Type      : Enumeration [read,write,recon,none],
        Phase        : Enumeration [query,propagate,none],
        Acc                                    :    Tuple[IP:Node,m:    Tuple
[mtype:String,ph:Phase,pn:Nat,v:Nat,t:Tag]],
        Stage                                          :      Enumeration
[idle,traverse,traverse_rec1,traverse_rec2,traverse_fix,pick_new_ts,update,up
date_done,scan,scan_done,contactq_start,contactq,notifyq,notifyq_done],
```

```
        D_message       : Tuple[mtype:String, ph:Phase,view : View, pn:Nat,
v:Nat, t:Tag, type:String, obj:Node, VT:val_tag_obj, C:Nat, rank : Nat],
        Update          : Tuple[x :View,y : change]
end
vocabulary tcp_specific_voc
    imports  TCPObjectsVoc, TCPNodeVoc ,dynastore%%alg_specific_voc,
    types
        Chan_message : Tuple[x : D_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
    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
```

Σχήμα 4.1 Λεξιλόγιο του αλγορίθμου Reader-Writer-Recon

4.2 Αυτόματο Reader-Writer-Recon

Το αυτόματο Reader-Writer-Recon αποτελεί τον κύριο αλγόριθμο των διεργασιών που θα χρησιμοποιηθούν. Κατά την αρχικοποίηση, το αυτόματο δέχεται ως είσοδο μια διεύθυνση IP η οποία απαιτείται για σκοπούς επικοινωνίας μέσω καναλιών TCP, μεταξύ των διεργασιών του συστήματος.

```
automaton ReaderWriterRecon(IP: Node)
```

Στην συνέχεια, Σχήμα 4.2, παραθέτονται οι ενέργειες που μπορεί να εκτελέσει ένα αυτόματο και αποτελούν την υπογραφή του.

Ακολούθως, Σχήμα 4.3, φαίνονται οι μεταβλητές καταστάσεων του αυτόματου που ορίζονται στο τμήμα states :

- value: Φυσικός αριθμός που αποθηκεύει μια ειδική τιμή του αυτόματου.
- tag: Προσωπική χρονοσφραγίδα του αυτόματου.
- maxV: Μέγιστη τιμή που παρατηρήθηκε από τα μηνύματα μεταξύ των διεργασιών του συστήματος.
- maxTag: Μέγιστη χρονοσφραγίδα που παρατηρήθηκε από τα μηνύματα μεταξύ των διεργασιών του συστήματος.
- ReplySet : Ακολουθία από ‘Reply’, δηλαδή διεργασιών που βρίσκονται σε μία συγκεκριμένη φάση με αριθμό Reply.num, διεύθυνση Reply.IP και αναμένουν απάντηση σε μια αίτηση για γραφή/ανάγνωση.
- acc: Ακολουθία από δεδομένα τύπου acc που αντιπροσωπεύουν τις διεργασίες που έχουν απαντήσει σε ένα αίτημα για γραφή/ανάγνωση. Σε περίπτωση που το πλήθος στοιχείων της ακολουθίας είναι ίσο με την πλειοψηφία του συστήματος τότε το αυτόματο προχωράει στο επόμενο στάδιο.

```
signature
    input RWRread
    input RWRwrite(v : Nat)
    input recon(cnf : View)
    input RECEIVE(m : Null[Chan_message])
    input scan_ack(changes : View)
    input update_ack
    input fail
    input init_view(view : View)

    output RWRread_ack(v : Nat)
    output RWRwrite_ack
    output recon_ack
    output SEND(m : Null[Chan_message])
    output scan
    output update(cng : change)

    internal init
    internal traverse
    internal breadth_first_search
    internal set_desired_view
    internal propagate
    internal traverse_fix

    internal contactq
    internal contactq_fix
    internal notifyq_fix
    internal pick_new_ts

    internal prep_contactq
    internal prep_replies
    internal prep_notify
```

Σχήμα 4.2 Υπογραφή του αυτόματου Reader-Writer-Recon

- changesets: Σύνολο από συνθέσεις διαφορετικές από αυτές του αυτόματου και έχουν παρατηρηθεί κατά την ενέργεια scan. Σκοπός είναι η συλλογή τυχόν πιο

πρόσφατων συνθέσεων που έχουν παρατηρηθεί από τα αυτόματα WeakSnapshot για αναβάθμιση της παρούσας σύνθεσης του αυτόματου.

- newconfig: Μια νέα σύνθεση που προτείνεται στο αυτόματο μέσω της ενέργειας recon.
- pnum: Φυσικός αριθμός που αντιπροσωπεύει τον αριθμό φάσης που βρίσκεται το αυτόματο. Αρχικοποιείται με 0 και αλλάζει μετά από κάθε επικοινωνία με την πλειοψηφία του συστήματος μέσω της ενέργειας contactq.
- op_type: Αντιπροσωπεύει την λειτουργία που εκτελεί το αυτόματο.
- status: Κατάσταση στην οποία βρίσκεται το αυτόματο
- stage: Αντιπροσωπεύει το στάδιο στο οποίο βρίσκεται η λειτουργία που εκτελεί το αυτόματο.
- phase : Αντιπροσωπεύει το στάδιο στο οποίο βρίσκεται η λειτουργία που εκτελεί το αυτόματο.
- failed : Μεταβλητή τύπου Boolean η οποία υποδεικνύει κατά πόσο η διεργασία έχει καταρρεύσει ή όχι. Σε περίπτωση κατάρρευσης, καμία ενέργεια δεν μπορεί να εκτελεστεί αφού η συνθήκη failed = false αποτελεί προϋπόθεση σε όλες τις ενέργειες.
- initView : Αποτελεί την αρχική σύνθεση του συστήματος.
- curview : Αποτελεί την παρούσα σύνθεση που είναι ενεργή στο αυτόματο. Ίσως να μην είναι η πιο πρόσφατη σύνθεση του συστήματος.
- desiredview : Η επιθυμητή σύνθεση στην οποία πρέπει να καταλήξει το curview μετά από την ενημέρωση του.
- notifySet: Το σύνολο των συνθέσεων του συστήματος για τις οποίες έχει ήδη γίνει ενημέρωση (notify) από κάποια διεργασία του συστήματος. Δεν αποστέλλονται μηνύματα τύπου notify που αφορούν μια σύνθεση που περιέχεται στην συγκεκριμένη ακολουθία.
- rcvd_notify: Αποτελεί το σύνολο των διεργασιών που έχουν απαντήσει σε ένα μήνυμα τύπου 'NOTIFY' καθώς και την πιο πρόσφατη σύνθεση που γνωρίζουν. Ένα αυτόματο στο στάδιο notify μπορεί να προχωρήσει στο επόμενο στάδιο εάν το σύνολο των στοιχείων στην ακολουθία rcvd_notify είναι ίσο με την πλειοψηφία του συστήματος.

- front: Αποτελείται από το σύνολο όλων των συνθέσεων, διαφορετικών από την παρούσα, που έχουν παρατηρηθεί από τα αυτόματα WeakSnapshot.
- to_be_updated: Αντιπροσωπεύει μία σύνθεση x η οποία θα ενημερωθεί με τις αλλαγές y.
- msgs: Μια ακολουθία με τα μηνύματα τύπου Chan_Message τα οποία θα αποστείλει η διεργασία μέσω της επόμενης ενέργειας send().
- success: Ο αριθμός αυτός αυξάνεται κατά μία μονάδα σε κάθε επιτυχή τερματισμό μίας λειτουργίας, δηλαδή την εκτέλεση μίας από τις ενέργειες RWRread_ack/RWRwrite_ack/recon_ack.
- forwardnotify: Η μεταβλητή αυτή, τύπου Nat, γίνεται ίση με 1 σε περίπτωση που έχει ληφθεί ένα μήνυμα τύπου NOTIFY και απαιτείται η προώθηση του.

Στο Σχήμα 4.4 φαίνονται οι σχέσεις μεταβάσεων του αυτόματου όπως έχουν οριστεί στην υπογραφή του :

- Μέσω της ενέργειας init_view(view) αρχικοποιούνται οι μεταβλητές InitView, curview οι οποίες είναι απαραίτητες κατά την πρώτη εκτέλεση του αυτόματου.
- Η ενέργεια prep_replies είναι υπεύθυνη για την δημιουργία των μηνυμάτων τύπου REPLY ανάλογα με τα στοιχεία που περιέχονται στην ακολουθία ReplySet. Ένα μήνυμα αυτού του τύπου αποτελεί απάντηση σε μήνυμα τύπου REQUEST. Επιπλέον, γίνεται προώθηση κάποιου μηνύματος NOTIFY που έχει ληφθεί σε περίπτωση που η μεταβλητή forwardnotify είναι ίση με 1.
- Η ενέργεια prep_notify είναι υπεύθυνη για την αποστολή μηνυμάτων τύπου NOTIFY σε περίπτωση που η λειτουργία βρίσκεται στο στάδιο notifyq και δεν έχει ληφθεί ή αποσταλεί μέχρι στιγμής τέτοιου τύπου μήνυμα για την παρούσα σύνθεση.. Τα μηνύματα αυτού του τύπου εξυπηρετούν τρεις σκοπούς : ενημέρωση νέων μελών για την είσοδο τους στη σύνθεση, ενημέρωση μελών ότι δεν αποτελούν πλέον μέρος της σύνθεσης και τέλος, τα μέλη της παρούσας σύνθεσης ενημερώνονται για την πιο πρόσφατη σύνθεση που κατέχει η συγκεκριμένη διεργασία.
- Η ενέργεια prep_contactq είναι υπεύθυνη για την δημιουργία των μηνυμάτων τύπου REQUEST σε όλα τα ενεργά μέλη της σύνθεσης curview και θεωρείται αίτηση για γραφή ή ανάγνωση.

```
states
    %% Value of the automaton
    value       : Nat := 0;
    %% Tag of the automaton
    tag         : Tag := [0,[0,0,0,0]];
    %% Max value received from curview.members
    maxV        : Nat := 0;
    %% Max tag received from curview.members
    maxTag      : Tag := [0,IP];
    %% Curview.members who are waiting for reply
    ReplySet    : Seq[Reply] := {};
    %% Curview.members who replied
    acc         : Seq[Acc] := {};
    %% Set of changes collected by WeakSnapshot automata
    changesets  : Seq[View] := {};
    %% New configuration
    newconfig   : View := {};
    %% Phase number
    pnum        : Nat := 0;
    %% Operation type
    op_type     : Op_Type:=none;
    %% Status of the automato
    status      : D_Status := idle;
    %% Stage of the automato
    stage       : Stage := idle;
    %% Phase of the automato
    phase       : Phase := query;
    %% True = failed, False = notfailed
    failed      : Bool  := false;
    %% Initial View
    InitView    : View := {};
    %% Current View
    curview     : View := {};
    %% Desired View
    desiredview : View := {};
    %% Set to be notified about a view
    NotifySet   : Seq[View] := {};
    %% Set who replied on a NOTIFY message
    rcvd_notify : Seq[Notify] := {};
    %% Set of nodes/views in the DAG
    front       : Seq[View] := {};
    %% tuple [view to be updated,update]
    to_be_updated: Update := [{},[0,[0,0,0,0]]];
    clock : AugmentedReal := 0;
    msgs : Seq[Null[Chan_message]] := {};
    success : Nat := 0;
    forwardnotify :Nat := 0;
```

Σχήμα 4.3 Μεταβλητές καταστάσεων αυτόματου Reader-Writer-Recon

- Μέσω της ενέργειας init ενεργοποιείται το αυτόματο αλλάζοντας την μεταβλητή status από idle σε active.

- Η ενέργεια fail αναγκάζει το αυτόματο να καταρρεύσει αλλάζοντας την τιμή της μεταβλητής failed. Το αυτόματο δεν μπορεί να εκτελέσει καμία άλλη λειτουργία που ακολουθεί την fail.
- Η ενέργειες RWRread, RWRwrite(v), recon(cnf) αρχικοποιούν τις μεταβλητές op_type , newconfig, stage ούτως ώστε να αρχίσει η λειτουργία ανάγνωσης, γραφής, αναδιαμόρφωσης αντίστοιχα. Η είσοδος της ενέργειας write είναι η μεταβλητή η οποία θα αντικαταστήσει την τιμή value στο αυτόματο ενώ η είσοδος cnf στην ενέργεια recon αποτελεί την αλλαγή στη σύνθεση του συστήματος .
- Η ενέργεια RWRread_ack(v) σηματοδοτεί τον επιτυχή τερματισμό της λειτουργίας read και επιστρέφει την τιμή v στον χρήστη ενώ οι ενέργειες RWRwrite_ack, recon_ack τον επιτυχή τερματισμό των λειτουργιών write και recon αντίστοιχα.
- Μέσω της ενέργειας SEND αποστέλλονται τα μηνύματα που περιέχονται στην ακολουθία msgs.
- Η ενέργεια RECEIVE είναι υπεύθυνη για την παραλαβή, εξαγωγή των απαραίτητων πληροφοριών από ένα μήνυμα και τοποθέτηση τους στην ανάλογη ακολουθία {rcvd_notify, acc, ReplySet}. Ο διαχωρισμός των μηνυμάτων γίνεται με βάση τον τύπο του μηνύματος. Επιπλέον, σε περίπτωση που έχει ληφθεί μήνυμα τύπου ‘’REQUEST’’με τον αποστολέα σε φάση propagate τότε γίνεται έλεγχος της χρονοσφραγίδας του μηνύματος για αναβάθμιση της υπάρχουσας εάν θεωρηθεί απαραίτητο. Σε περίπτωση μηνύματος NOTIFY η διεργασία ελέγχει κατά πόσο έχει εισαχθεί ή αφαιρεθεί από τη σύνθεση του συστήματος ή εάν έχει ληφθεί μια ενημερωμένη σύνθεση του συστήματος η οποία θα αντικαταστήσει την παρούσα. Σε περίπτωση ενημερωμένης σύνθεσης επαναλαμβάνεται η διαδικασία διάσχισης του γράφου για ανανέωση του σύνθεσης. Αρχικοποιείται η μεταβλητή forwardnotify για προώθηση του μηνύματος στα υπόλοιπα μέλη του συστήματος.
- Με την εσωτερική ενέργεια contactq αρχικοποιείται η ακολουθία acc και αυξάνεται ο αριθμός φάσης του αυτόματου υποδεικνύοντας ότι η λειτουργία έχει περάσει σε μία νέα φάση. Με το τέλος της contactq το νέο στάδιο του αυτόματου είναι το contact, όπου επιχειρείται επικοινωνία με την πλειοψηφία του συστήματος μέσω της ενέργειας SEND. Τα απαραίτητα μηνύματα για αυτή

την επικοινωνία προστίθενται στην ακολουθία msg μέσω της ενέργειας prep_contact.

- Η εσωτερική ενέργεια contactq_fix αναμένει την ανταπόκριση της πλειοψηφίας στο αίτημα το οποίο έχει σταλεί στο στάδιο contact. Αφού ληφθεί ο απαραίτητος αριθμός απαντήσεων η ενέργεια προχωράει με τον έλεγχο των χρονοσφραγίδων που έχουν αποθηκευτεί στην ακολουθία acc. Σε περίπτωση που βρεθεί πιο πρόσφατη χρονοσφραγίδα από την υπάρχουσα, τότε ενημερώνεται η τιμή maxTag αλλά και η maxV με την τιμή v που είναι αποθηκευμένη παράλληλα με τη μέγιστη χρονοσφραγίδα στη acc. Έπειτα, η λειτουργία προχωράει στο επόμενο στάδιο traverse_rec2 εάν βρίσκεται στη φάση query ή στο στάδιο scan σε αντίθετη περίπτωση.
- Με την εσωτερική ενέργεια traverse αρχικοποιείται η μεταβλητή front με την παρούσα σύνθεση του συστήματος curview και η μεταβλητή desiredview με την ένωση των curview και newconfig.
- Κατά την εσωτερική ενέργεια breadth_first_search επιλέγεται η μικρότερη σε πλήθος σύνθεση από την ακολουθία front η οποία, εφόσον είναι η μικρότερη, περιέχει τις λιγότερες αλλαγές από τις υπόλοιπες σε σχέση με την παρούσα σύνθεση. Σε περίπτωση που το αυτόματο δεν περιέχεται στην σύνθεση αυτή τερματίζει τη λειτουργία του και εισέρχεται σε κατάσταση αδράνειας (idle). Σε αντίθετη περίπτωση ελέγχεται κατά πόσο η σύνθεση αυτή είναι διαφορετική από την επιθυμητή (desiredview). Εάν ισχύει, γίνεται αρχικοποίηση της μεταβλητής to_be_updated με την σύνθεση αυτή και τις αλλαγές που απαιτούνται ώστε να ισούται με την επιθυμητή σύνθεση και επόμενο στάδιο ορίζεται το update. Η ενέργεια αυτή συμβολίζεται ως to_be_updated = < w, desiredview \ w> . Εάν είναι ίσα τότε η επόμενη φάση είναι η είναι η query στάδιο το scan.
- Η εσωτερική ενέργεια set_desired_view ενεργοποιείται στο στάδιο traverse_rec2 έναντι της propagate εάν κατά το στάδιο scan και φάση query η ισχύει changesets $\neq \emptyset$. Με την εκτέλεση της ενέργειας αφαιρείται η παρούσα σύνθεση από το σύνολο front και προστίθενται στην επιθυμητή σύνθεση όλες οι αλλαγές που έχουν παρατηρηθεί από τα αυτόματα WeakSnapshot και στο front όλες οι νέες συνθέσεις που προκύπτουν από την ένωση της παρούσας και κάθε σύνολο αλλαγών ξεχωριστά. Την ενέργεια και traverse_rec2 ακολουθούν η

breadth_first_search και traverse_rec1 όπου και σηματοδοτούν την επανεκκίνηση της διάσχισης του γράφου με τις συνθέσεις.

- Η εσωτερική ενέργεια propagate ακολουθεί το στάδιο contactq και είναι υπεύθυνη για την αλλαγή της φάσης λειτουργίας από query σε propagate. Επιπλέον, το στάδιο λειτουργίας αλλάζει σε pick_new_ts ή contactq_start εάν η λειτουργία είναι γραφή ή ανάγνωση αντίστοιχα.
- Η εσωτερική ενέργεια traverse_fix σηματοδοτεί τον τερματισμό τις διάσχισης του γράφου (traverse) και την πλήρης ενημέρωση της σύνθεσης του συστήματος, δηλαδή curview = desiredview. Συνεπώς ενέργεια ενημερώνει την curview με την τιμή της desiredview και αλλάζει το στάδιο λειτουργίας σε notifyq για ενημέρωση της πλειοψηφίας την τελική αλλαγή της σύνθεσης αφού προστεθούν τα απαραίτητα μηνύματα στην ακολουθία msgs μέσω της prep_notify.
- Η εξωτερική ενέργεια scan πυροδοτεί την αντίστοιχη ενέργεια στα αυτόματα WeakSnapshot για αναζήτηση αλλαγών στο σύστημα.
- Η ενέργεια εισόδου scan_ack πυροδοτείται από το WeakSnapshot και έχει ως είσοδο το σύνολο αλλαγών που έχουν παρατηρηθεί. Οι αλλαγές αυτές αποθηκεύονται στη μεταβλητή changesets. Εάν η φάση λειτουργίας είναι η query τότε το στάδιο που ακολουθεί είναι το contactq-start. Εάν η φάση είναι η propagate τότε σε περίπτωση που δεν υπάρχουν αλλαγές, changesets $\neq \emptyset$, το αυτόματο συνεχίζει με το στάδιο traverse_rec1 αλλιώς το επόμενο στάδιο είναι το traverse-fix αφού αυτό υποδεικνύει ότι το αυτόματο κατέχει την πιο πρόσφατη σύνθεση του συστήματος.
- Με την εσωτερική ενέργεια pick_new_ts επιλέγεται μια νέα χρονοσφραγίδα, timestamp+1, η μεταβλητή maxV γίνεται ίση με την τιμή value και το επόμενο στάδιο είναι το contactq-start. Η συγκεκριμένη ενέργεια εκτελείται μόνο κατά τη λειτουργία γραφής
- Η εσωτερική ενέργεια notifyq_fix εκτελείται αφού ληφθεί απάντηση από την πλειοψηφία του συστήματος στο μήνυμα με τύπο NOTIFY. Το στάδιο που ακολουθεί είναι το notifyq-done που σηματοδοτεί τον επιτυχή τερματισμό μίας λειτουργίας.
- Μέσω της εξωτερικής ενέργειας update(cng) πυροδοτείτε η αντίστοιχη ενέργεια των αυτομάτων WeakSnapshot για ενημέρωση της σύνθεσης με τις αλλαγές που

περιέχονται στη μεταβλητή to_be_updated. Έπειτα η μεταβλητή γίνεται ίση με το κενό.

- Η εσωτερική ενέργεια update_ack πυροδοτείτε από τα αυτόματα WeakSnapshot και επαναφέρει το αυτόματο στη φάση query και στάδιο scan.

```
transitions
    input init_view(view)
    eff
        InitView := view;
        curview  := view;


    internal prep_replies
    locals
        tview : View := {};
    pre
       ~failed /\ status = active;
    eff
        %Reply to requests
        for i : Nat where i< len(ReplySet) do
            msgs := msgs |-
embed([["REPLY",phase,{},ReplySet[i].num,value,tag,"",
[0,0,0,0],[[[0,[0,0,0,0]],0],[0,0,0,0]], 0,0],IP,ReplySet[i].IP]);
        od
        ReplySet   := {};
        %forward any notify
        if(forwardnotify=1) then
            forwardnotify:=0;
            for i : Nat where i < len(NotifySet) do
                tview := NotifySet[i];
                for j : Nat where j< len(tview)do
                    if (tview[j].act = 1 /\ tview[j].IP ~=IP ) then
                        msgs := msgs |-
embed([["NOTIFY",phase,tview,0,0,[0,[0,0,0,0]],"",
[0,0,0,0],[[[0,[0,0,0,0]],0],[0,0,0,0]], 0,0],IP,tview[j].IP]);
                    fi
                od
            od
        fi

   internal prep_notify
   locals
        tview : View := {};
    pre
       ~failed /\ status = active;
       stage = notifyq;
    eff
        %Send notify to view members. View members are denoted as [1,IP] in a
view
        if(curview \notin NotifySet) then
              for i : Nat where i< len(curview) do
                  if (curview[i].act = 1 /\ curview[i].IP ~=IP) then
                      msgs := msgs |-
embed([["NOTIFY",phase,curview,0,0,[0,[0,0,0,0]],"",
[0,0,0,0],[[[0,[0,0,0,0]],0],[0,0,0,0]], 0,0],IP,curview[i].IP]);
                  fi
```

```
                od
        else
            stage := notifyq_done;
        fi



    internal prep_contactq
    pre
       ~failed /\ status = active;
    eff
        %Send read/write requests
        if (stage = contactq) then
            for i : Nat where i< len(curview) do
                if (curview[i].act = 1 /\ curview[i].IP ~=IP) then
                    msgs := msgs |-
embed([["REQUEST",phase,{},pnum,maxV,maxTag,"",
[0,0,0,0],[[[0,[0,0,0,0]],0],[0,0,0,0]], 0,0],IP,curview[i].IP]);
                fi
            od
        fi


    internal init
    pre
        ~failed /\ status = idle; %% switched from "status = active" to
"status = idle" for testing purposes
        [1,IP] \in InitView /\ [0,IP] \notin InitView;
    eff
        status := active;

    input fail
    eff
        failed := true ;

    input RWRread
    eff
        if (~failed /\ status = active) then
            op_type := read;
            newconfig := {};
            stage := traverse;
        fi
    output RWRread_ack(v)
    pre
        ~failed /\ status = active;
        op_type = read;
        stage = notifyq_done ;
        v=maxV ;
   eff
        stage := idle;
        success := success + 1;



   input RWRwrite(v)
   eff
        if  (~failed /\ status = active) then
            value := v;
```

```
            op_type := write;
            newconfig := {};
            stage := traverse;
        fi
    output RWRwrite_ack
    pre
        ~failed /\ status=active ;
        op_type=write;
        stage=notifyq_done;
    eff
        stage:=idle;
        success := success + 1;

    input recon(cnf)
    eff
        if  (~failed /\ status= active) then
            op_type:=recon;
            newconfig:=cnf;
            stage:=traverse;
        fi

    output recon_ack
    pre
        ~failed /\ status= active;
        op_type= recon;
        stage = notifyq_done;
    eff
        stage := idle;
        success := success + 1;

    output SEND(m)
    pre
        msgs ~= {};
        m = head(msgs);
        ~failed /\ status = active;

    eff
        msgs := tail(msgs);

    input RECEIVE(m)
    locals
        reply: String := "REPLY";
        request: String := "REQUEST";
        notify: String := "NOTIFY";
        tview : View := {};
        total : Nat :=0;
    eff
        if (~failed /\ status=active) then
            if (val(m).x.mtype =reply) then
                acc := acc |-
[val(m).sender,[val(m).x.mtype,val(m).x.ph,val(m).x.pn,val(m).x.v,val(m).x.t]
];
            fi
            if (val(m).x.mtype =request) then
                if (val(m).x.ph=propagate /\
val(m).x.t.timestamp>tag.timestamp) then
                    value := val(m).x.v;
                    tag   := val(m).x.t;
```

```
                fi
                if([val(m).sender,val(m).x.pn] \notin ReplySet) then
                    ReplySet := ReplySet |- [val(m).sender,val(m).x.pn];
                fi
            fi
            if (val(m).x.mtype =notify) then
                if (stage=notifyq) then %% collect replies
                    rcvd_notify := rcvd_notify |-
[val(m).sender,val(m).x.view];
                else
                    if (val(m).x.view \notin NotifySet) then
                        NotifySet := NotifySet |- val(m).x.view;
                        forwardnotify:=1;
                        tview := val(m).x.view;
                        for i:Nat where i<len(curview) do
                            if(curview[i] \in tview) then
                                total :=total +1;
                            fi
                        od
                        if(len(curview) < len(tview)) then
                                 if (total = len(curview)) then
                                    if ([1,IP] \in tview) then
                                            status := active;
                                    fi
                                    curview := tview ;
                                    stage := traverse; %%restart traverse
                                 fi
                        fi
                    fi
                fi
            fi
        fi

    internal notifyq_fix
    locals
        curview_members : Nat :=0;
        rcvdnot_members : Nat :=0;
    pre
        ~failed /\ status = active;
        stage = notifyq;
    eff
        for i:Nat where i< len(rcvd_notify) do
            if (rcvd_notify[i].view = curview) then
                rcvdnot_members := rcvdnot_members + 1;
            fi
        od
        for i:Nat where i< len(curview) do

            if (curview[i].act = 1 /\ [0,curview[i].IP] \notin curview) then
                curview_members := curview_members + 1 ;
            fi
        od
        if(rcvdnot_members >= (div(curview_members,2)+1)) then
            stage:=notifyq_done;
            rcvd_notify:={};
        fi

    internal traverse
```

```
    pre
        ~failed /\ status = active ;
        stage = traverse ;
    eff
        front := {};
        front := front |- curview;
        desiredview := curview;
        for i:Nat where i < len(newconfig) do
            if (newconfig[i] \notin desiredview) then
                desiredview := desiredview |- newconfig[i];
            fi
        od
        stage := traverse_rec1;

    internal breadth_first_search
    locals
        minlen  : Nat := 0;
        minpos  : Nat := 0;
        updated : change := [0,[0,0,0,0]];
    pre
        ~failed /\ status = active;
        stage=traverse_rec1;
    eff
        minlen :=len(front[0]);

        for i:Nat where i < len(front) do
            if (minlen > len(front[i])) then
                minlen := len(front[i]);
                minpos := i;
            fi
        od
        curview := front[minpos];

        if ([1,IP] \notin curview) then
            status := idle; % halt if not a members of the view
        else
            if curview ~= desiredview then
                for i:Nat where i < len(desiredview) do
                    if (desiredview[i] \notin curview) then
                        updated := desiredview[i];
                    fi
                od
                to_be_updated.x := curview;
                to_be_updated.y := updated;
                stage:= update;
            else
                phase:=query;
                stage:=scan;
            fi
        fi


internal set_desired_view
locals
        tfront : Seq[View] := {}; %front := front \ w;
        tview  : View :={};
        c      : View :={};
```

```
pre
     ~failed /\ status = active;
     stage = traverse_rec2;
     changesets ~= {};
eff
     %front := front \ w;
     for i:Nat where i < len(front) do
        if (front[i] ~= curview) then
             tfront := tfront |- front[i];
         fi
     od
     front:=tfront;
     desiredview := curview ;
     for  i:Nat where i< len(changesets) do
         tview:=curview;
         c := changesets[i];
         for k:Nat where k<len(c) do
             if(c[k] \notin desiredview) then
                 desiredview := desiredview |- c[k];
             fi
             if(c[k] \notin tview) then
                 tview := tview |- c[k];
             fi
         od
         if (tview \notin front) then
             front := front |- tview;
         fi
     od
     stage := traverse_rec1;

internal propagate
pre
     ~failed /\ status = active;
     stage = traverse_rec2;
     changesets = {};
eff
     phase :=propagate;
     if (op_type = write) then
         stage := pick_new_ts;
     else
         stage := contactq_start;
     fi
 internal traverse_fix
 pre
     ~failed /\ status = active;
     stage = traverse_fix;
 eff
     curview := desiredview;
     stage:= notifyq;


 output scan
 pre
     ~failed /\ status = active;
     stage = scan;

 input scan_ack(changes)
 eff
```

```
        if (~failed /\ status = active) then
            if(changes ~= {}) then
                changesets := changesets |- changes;
            fi
            if (phase = query) then
                stage := contactq_start;
            else % phase = propagate
                if (changesets ~= {}) then
                    stage := traverse_rec1;
                else
                    stage := traverse_fix;
                fi
            fi
        fi

    internal pick_new_ts
    pre
        ~failed /\ status = active;
        stage = pick_new_ts;
        changesets = {};
    eff
        maxTag := [maxTag.timestamp+1,IP];
        maxV:=value;
        stage:= contactq_start;

    internal contactq
    pre
        ~failed /\ status = active;
        stage = contactq_start;
    eff
        acc := {};
        pnum := pnum+1;
        stage:= contactq;

    output update(cng)
    pre
        ~failed /\ status = active;
        [curview,cng] = to_be_updated;
        stage = update;
    eff
        to_be_updated := [{},[0,[0,0,0,0]]];

    input update_ack
    eff
        if (~failed /\ status = active) then
            phase := query;
            stage := scan;
        fi



    internal contactq_fix
    locals
        max_tag : Tag := [0,[0,0,0,0]]; % max tag in acc
        tag_pos : Nat := 0;             % max tag position in acc
        t       : Tag := [0,[0,0,0,0]];
        v       : Nat := 0;
        replied : Nat := 0; % number of replies
```

```
        members : Nat := 0; % total curview.members
    pre
        ~failed /\ status = active;
        stage = contactq;
    eff
        for i:Nat where i< len(acc) do
            if (acc[i].m.pn = pnum) then
                replied := replied + 1;
            fi
        od
        for i:Nat where i< len(curview) do
            %if an IP exist and is not to be 0d
            if (curview[i].act = 1 /\ [0,curview[i].IP] \notin curview) then
                members := members + 1 ;
            fi
        od
        if(replied >= (div(members,2)+1)) then
            if (op_type = read) then

                max_tag :=  acc[0].m.t;
                tag_pos :=  0;
                for i:Nat where i<len(acc) do
                    if (acc[i].m.t.timestamp>max_tag.timestamp) then
                      max_tag :=  acc[i].m.t;
                      tag_pos :=  i;
                    fi
                od
                t := max_tag;
                v := acc[tag_pos].m.v;
                if (t.timestamp > maxTag.timestamp) then

                    maxV := v;
                    maxTag :=t;
                fi
            fi
            if (phase = query) then
                stage := traverse_rec2;
            else
                stage := scan; %phase = propagate
            fi
        fi

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

Σχήμα 4.4 Σχέσεις μεταβάσεων αλγόριθμου Reader-Writer-Recon

4.3 Αυτόματο Σύνθεσης

Αφού έχει τελειώσει η υλοποίηση του αυτόματου διεργασία καθώς και των καναλιών επικοινωνίας, τα οποία είναι ήδη υλοποιημένα, μπορεί να γίνει η υλοποίηση του αυτόματου σύνθεσης (Composition). Ξεκινώντας, όπως φαίνεται στο Σχήμα 4.5,

συμπεριλαμβάνονται τα απαραίτητα αρχεία για την εκτέλεση του αλγορίθμου όπως το λεξιλόγιο, το αυτόματο διεργασίας (Reader-Writer-Recon) και τα αυτόματα TCPRecvMed, TCPSendMed, TCPChanMed που υλοποιούν την επικοινωνία μέσω πρωτοκόλλου TCP.

```
%%% .: Vocabulary :.
include "Vocabulary.tioa"
imports dynastore, TCPObjectsVoc, TCPNodeVoc, tcp_specific_voc, JVMSocket,
JVMServerSocket, ChannelVoc
%%% .: TCP Mediator Automata :.
include "TCPRecvMed.tioa"
include "TCPSendMed.tioa"
include "TCPChanMed.tioa"
%%% .: Algorithm Automata :.
include "ReaderWriterRecon.tioa"
```

Σχήμα 4.5 Εισαγωγή αρχείων στο αυτόματου σύνθεσης Composition.tioa

Το αυτόματο σύνθεσης, Σχήμα 4.6, δέχεται ως παραμέτρους επτά φυσικούς αριθμούς οι οποίοι αναπαριστούν μια διεύθυνση IP(4), την υποδοχή(port) μέσω της οποίας θα γίνεται η επικοινωνία, τον μέγιστο χρόνο αναμονής στο κανάλι TCP(timeout) και τον τύπο λειτουργίας(OP) που θα εκτελεστεί. Το πεδίο τιμών του τύπου λειτουργίας είναι {0,1,2} όπου αντιστοιχεί σε {read,write,recon}. Για παράδειγμα, με είσοδο 192 168 10 1 10000 1000 1 δημιουργείται ένα αυτόματο σύνθεσης με διεύθυνση 192.168.10.1 του οποίου τα κανάλια επικοινωνίας χρησιμοποιούν το port 10000 και έχουν timeout ίσο με 1 δευτερόλεπτο (1000 ms = 1 s). Η λειτουργία που θα εκτελεστεί είναι η write. Επιπλέον, το αυτόματο σύνθεσης αποτελείται από ένα αυτόματο RWR(Reader-Writer-Recon) με είσοδο την διεύθυνση IP, ένα αυτόματο SM(SendMed), RM(RecvMed), CM(ChanMed) με είσοδο το port και το timeout το κάθε ένα.

Οι μεταβλητές που χρησιμοποιεί το αυτόματο σύνθεσης στο τμήμα states φαίνονται πιο κάτω:

- IP : αντιπροσωπεύει την διεύθυνση IP του αυτόματου
- DYNodes: Περιέχει το σύνολο των διευθύνσεων των μηχανών όπου είναι ενεργή μια διεργασία του αλγορίθμου κατά την εκκίνηση του αλγόριθμου.

```
automaton Composition(n1:Nat, n2:Nat, n3:Nat, n4:Nat, port:Nat, timeout:Nat,
OP : Nat)
components
    RWR : ReaderWriterRecon([n1,n2,n3,n4]);
    SM  : SendMed(port, timeout);
    RM  : RecvMed(port, timeout);
    CM  : ChanMed(port, timeout);
```

Σχήμα 4.6 Εισαγωγή συστατικών του αυτόματου σύνθεσης (Composition.tioa)

- ms: Δηλώνει ένα μήνυμα τύπου Chan_message το οποίο θα σταλεί από την διεργασία.
- mr: Δηλώνει ένα μήνυμα τύπου Chan_message το οποίο θα ληφθεί από την διεργασία.
- isConn: Boolean μεταβλητή η οποία χρησιμοποιείται από το αυτόματο καναλιού CM κατά την εγκατάσταση καναλιού μεταξύ δύο διεργασιών. Αληθής(true) τιμή υποδηλώνει την επιτυχής εγκατάσταση του καναλιού ενώ ψευδής(false) το αντίθετο.
- error : τύπου JVMError, χρησιμοποιείται από το αυτόματο καναλιού CM για την αναφορά προβλημάτων κατά την δημιουργία ενός καναλιού.
- runs : δηλώνει τον αριθμό επαναλήψεων του προγράμματος μεταβάσεων
- InitView : Δηλώνει την αρχική σύνθεση του συστήματος
- recon_r: Δηλώνει μια νέα σύνθεση του συστήματος η οποία εισάγεται μέσω της ενέργειας recon.
- scan_cnf: Είσοδο στην ενέργεια scan_ack και δηλώνει τις αλλαγές που έχουν παρατηρηθεί.
- val: Σε περίπτωση γραφής, δηλώνει τη τιμή που θα γραφτεί μέσω της write(v).
- new_msg: μεταβλητή τύπου Boolean η οποία χρησιμεύει στην επανάληψη των ενεργειών SEND, RECEIVE μέχρι να μην υπάρχουν άλλα μηνύματα. Όταν σταλούν ή ληφθούν όλα τα μηνύματα η μεταβλητή γίνεται false και τερματίζεται η επανάληψη.

```
states
%IP of this Node
IP      : Node              := [n1,n2,n3,n4];
%IP of nodes in view
DYNodes : Seq[Node]         := {};
%Message to send
ms      : Null[Chan_message]:= nil();
%Message to receive
mr      : Null[Chan_message]:= nil();

isConn  : Bool              := false;

error   : Null[JVMError]    := nil();
%Number of runs
runs    : Nat               := 0;
%Initial View of the system.
InitView: View       :={};
% Reconfiguration
recon_r : View       :={};
% Scan Results
scan_cnf: View  :={};
% Value
val     :Nat                :=5;
% wait for messages
new_msg  :Bool               := true;
```

Σχήμα 4.7 Μεταβλητές καταστάσεων αυτόματου σύνθεσης (Composition.tioa)

Το τμήμα των μεταβλητών καταστάσεων ακολουθεί το πρόγραμμα μεταβάσεων (schedule). Το πρόγραμμα αρχίζει με την τοποθέτηση όλων των διευθύνσεων των ενεργών διεργασιών στην ακολουθία DYNodes και η αρχικοποίηση της αρχικής σύνθεσης του συστήματος InitView.

Έπειτα εκτελείτε η αρχικοποίηση του αυτομάτου με την ενέργεια init_view και είσοδο την InitView. Ακολούθως, δημιουργείται και δεσμεύεται η υποδοχή για απαραίτητη για την εγκατάσταση καναλιών με άλλες διεργασίες μέσω των ενεργειών TCP_bind, TCP_respBind. Η εγκατάσταση των καναλιών προς όλες τις υπόλοιπες διεργασίες γίνεται με την εκτέλεση των ενεργειών TCP_SenderOpen, TCP_accept,

TCP_respAccept, TCP_respSender. Ανάλογα με την λειτουργία (OP) που θα εκτελεστεί ακολουθεί η ενέργεια read και write και 'runs' επαναλήψεις του κύριου μέρους του προγράμματος μεταβάσεων.

Η εκτέλεση των λειτουργιών γραφής, ανάγνωσης και αναδιαμόρφωσης ακολουθεί παρόμοια σειρά ενεργειών σε κάθε περίπτωση με ελάχιστες διαφορές τις οποίες θα τονίσουμε κατά την περιγραφή της.

Σε περίπτωση όπου η λειτουργία είναι αναδιαμόρφωση, εκτελείται πρώτα η ενέργεια recon με είσοδο την μεταβλητή recon_r και έπειτα η ενέργεια traverse. Οι ενέργειες breadth_first_search, scan, scan_ack, contactq, prep_contactq, SEND, RECEIVE, prep_replies, SEND, RECEIVE, contactq_fix, set_desired_view εκτελούνται σε επανάληψη μέχρι να μην παρατηρούνται νέες αλλαγές από την ενέργεια scan, όπως ακριβώς και αναφέρθηκε κατά την ανάλυση του αλγορίθμου στο Κεφάλαιο 3. Μεταξύ των ενεργειών BFS και scan εκτελείται η ενέργεια update, update_ack στην περίπτωση όπου το στάδιο εκτέλεσης είναι το update. Επιπλέον οι ενέργειες SEND, RECEIVE επαναλαμβάνονται μέχρι να μην υπάρχουν να εξαντληθούν όλα τα μηνύματα με την χρήση while loop και της μεταβλητής new_msg. Η δεύτερη επανάληψη των ενεργειών SEND, RECEIVE που ακολουθεί την ενέργεια prep_replies οφείλεται στην ανάγκη ανταπόκρισης στα μηνύματα που έχουν ληφθεί και λήψη της αντίστοιχης ανταπόκρισης από άλλες διεργασίες. Ακολουθεί η ενέργεια propagate και εάν η λειτουργία που εκτελείται είναι γραφή τότε εκτελείται η ενέργεια pick_new_ts. Στη συνέχεια, μέσω των contactq, prep_contactq, SEND, RECEIVE, prep_replies, SEND, RECEIVE, γίνεται επικοινωνία με την πλειοψηφία του συστήματος η οποία τελειώνει επιτυχώς με την εκτέλεση της contact_fix. Η ολοκλήρωση της ενέργειας traverse_fix σηματοδοτεί την ολοκλήρωση των βασικών ενεργειών της λειτουργίας που εκτελείται. Το αυτόματο στη συνέχεια μέσω των ενεργειών prep_notify, SEND, RECEIVE, prep_replies, SEND, RECEIVE, notfiyq_fix ενημερώνει τις διεργασίες του συστήματος για την παρούσα σύνθεση που έχει στην κατοχή του. Αφού ληφθεί απάντηση από την πλειοψηφία του συστήματος τότε εκτελείται η notify_fix η οποία τερματίζει την λειτουργία προχωρώντας το αυτόματο στο επόμενο στάδιο, notfiyq-done. Οι ενέργειες RWRread_ack, RWRwrite_ack και recon_ack εκτελούνται στο τέλος του προγράμματος μεταβάσεων τυπώνοντας και την τιμή που έχει διαβαστεί εάν η λειτουργία ήταν ανάγνωση.

```
  do
        DYNodes := DYNodes |- [150,244,58,161];
        DYNodes := DYNodes |- [194,29,178,14];
        DYNodes := DYNodes |- [129,242,19,197];
        DYNodes := DYNodes |- [130,206,158,138];
        runs    := 1 ;
        InitView:= InitView |- [1,[150,244,58,161]];
        InitView:= InitView |- [1,[194,29,178,14]];
        InitView:= InitView |- [1,[129,242,19,197]];
        InitView:= InitView |- [1,[130,206,158,138]];
        %Initialize
        fire input RWR.init_view(InitView);
        %% 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(DYNodes) do
            if(IP ~= DYNodes[n]) then
                fire output SM.TCP_senderOpen(DYNodes[n], port);
                follow SM.DELAY duration len(DYNodes)*200;
                %% accept only on new connection
                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(DYNodes[n], port, isConn);
            fi
        od %end of for
        if (error =nil()) then
            follow RWR.Time duration timeout;
        fi
        fire internal RWR.init;
        %% Read
        if (OP = 0) then
            fire input RWR.RWRread;
        fi
        %% Write
        if (OP = 1) then
            fire input RWR.RWRwrite(val);
        fi

        for i: Nat where i < runs do
            follow RWR.Time duration 1;
            %% Recon
            if (OP = 2) then
                fire input RWR.recon(recon_r);
            fi
            fire internal RWR.traverse;
            while (RWR.changesets~={} ) do   % Restart traverse if changesets
=/= {}
```

```
                fire internal RWR.breadth_first_search;
                if( RWR.stage = update ) then
                    fire output RWR.update(RWR.to_be_updated.y);
                    fire input  RWR.update_ack;
                fi
                fire output RWR.scan;
                fire input RWR.scan_ack(scan_cnf);
                fire internal RWR.contactq;
                % prepare messages to contact quorum
                fire internal RWR.prep_contactq;
                %SEND
                new_msg:=true;
                while (new_msg = true) do
                    ms:=nil();
                    fire output RWR.SEND(ms);
                    if(ms~=nil()) then
                        fire output SM.TCP_write(ms,
val(ms).sender,val(ms).destination);
                    else
                        new_msg := false;
                    fi
                od
                follow SM.DELAY duration 200;
                % extract messages from channel if there are any
                fire output RM.TCP_read;
                new_msg := true;
                while (new_msg = true) do
                    fire output CM.TCP_respRead(mr);
                    if( mr ~= nil()) then
                        fire output RM.RECEIVE (mr);
                        print val(mr);
                    else
                        new_msg := false;
                    fi
                od
                % prepare replies if any
                fire internal RWR.prep_replies;
                %SEND
                new_msg:=true;
                while (new_msg = true) do
                    ms:=nil();
                    fire output RWR.SEND(ms);
                    if(ms~=nil()) then
                        fire output SM.TCP_write(ms,
val(ms).sender,val(ms).destination);
                    else
                        new_msg := false;
                    fi
                od
                follow SM.DELAY duration 200;
                % extract messages from channel if there are any
                fire output RM.TCP_read;
                new_msg := true;
                while (new_msg = true) do
                    fire output CM.TCP_respRead(mr);
                    if( mr ~= nil()) then
                        fire output RM.RECEIVE (mr);
                        print val(mr);
```

```
                    else
                        new_msg := false;
                    fi
                od
                fire internal RWR.contactq_fix;
                fire internal RWR.set_desired_view;
            od

            fire internal RWR.propagate;
            if (OP = 1) then
                fire internal RWR.pick_new_ts;
            fi
            fire internal RWR.contactq;
            % prepare messages to contact quorum
            fire internal RWR.prep_contactq;
            %SEND
            new_msg:=true;
            while (new_msg = true) do
                ms:=nil();
                fire output RWR.SEND(ms);
                if(ms~=nil()) then
                    fire output SM.TCP_write(ms,
val(ms).sender,val(ms).destination);
                else
                    new_msg := false;
                fi
            od
            follow SM.DELAY duration 200;
            % extract messages from channel if there are any
            fire output RM.TCP_read;
            new_msg := true;
            while (new_msg = true) do
                fire output CM.TCP_respRead(mr);
                if( mr ~= nil()) then
                    fire output RM.RECEIVE (mr);
                    print val(mr);
                else
                    new_msg := false;
                fi
            od
            fire internal RWR.prep_replies;
            %SEND
                new_msg:=true;
                while (new_msg = true) do
                    ms:=nil();
                    fire output RWR.SEND(ms);
                    if(ms~=nil()) then
                        fire output SM.TCP_write(ms,
val(ms).sender,val(ms).destination);
                    else
                        new_msg := false;
                    fi
                od
            follow SM.DELAY duration 200;
            % extract messages from channel if there are any
            fire output RM.TCP_read;

                new_msg := true;
```

```
                while (new_msg = true) do
                    fire output CM.TCP_respRead(mr);
                    if( mr ~= nil()) then
                        fire output RM.RECEIVE (mr);
                        print val(mr);
                    else
                        new_msg := false;
                    fi
                od
            fire internal RWR.contactq_fix;
            fire output RWR.scan;
            fire input RWR.scan_ack(scan_cnf);
            fire internal RWR.traverse_fix;
             % prepare messages to notify quorum
            fire internal RWR.prep_notify;
            %SEND
            new_msg:=true;
            while (new_msg = true) do
                ms:=nil();
                fire output RWR.SEND(ms);
                if(ms~=nil()) then
                    fire output SM.TCP_write(ms,
val(ms).sender,val(ms).destination);
                else
                    new_msg := false;
                fi
            od

            follow SM.DELAY duration 200;
            % extract messages from channel if there are any
            fire output RM.TCP_read;
            new_msg := true;
            while (new_msg = true) do
                fire output CM.TCP_respRead(mr);
                if( mr ~= nil()) then
                    fire output RM.RECEIVE (mr);
                    print val(mr);
                else
                    new_msg := false;
                fi
            od
            fire internal RWR.prep_replies;
            %SEND
            new_msg:=true;
            while (new_msg = true) do
                ms:=nil();
                fire output RWR.SEND(ms);
                if(ms~=nil()) then
                    fire output SM.TCP_write(ms,
val(ms).sender,val(ms).destination);
                else
                    new_msg := false;
                fi
            od
            follow SM.DELAY duration 200;
            % extract messages from channel if there are any
            fire output RM.TCP_read;
            new_msg := true;
```

```
            while (new_msg = true) do
                fire output CM.TCP_respRead(mr);
                if( mr ~= nil()) then
                    fire output RM.RECEIVE (mr);
                    print val(mr);
                else
                    new_msg := false;
                fi
            od
            fire internal RWR.notifyq_fix;
            if (OP = 0 ) then
                fire output RWR.RWRread_ack(val);
                print val;
            fi
            if (OP = 1 ) then
                fire output RWR.RWRwrite_ack;
            fi
            if (OP = 2 ) then
               fire output RWR.recon_ack;
            fi
        od
    od
```

Σχήμα 4.8 Πρόγραμμα μεταβάσεων αυτόματου σύνθεσης (Composition.tioa)

Στο σημείο αυτό, να υπενθυμίσουμε ότι η ορθότητα προδιαγραφής διατηρείται από τον μεταφραστή tempo2java. [1].

Κεφάλαιο 5

Πειραματική αξιολόγηση

Η αξιολόγηση έγινε με σκοπό τη διαπίστωση της ορθότητας της υλοποίησης μας αλλά και τη μελέτη της ευρωστίας της κατανεμημένης υπηρεσίας γραφής/ανάγνωσης του συστήματος Dynastore, δηλαδή του ότι όντως λειτουργεί ορθά σε δυσμενής συνθήκες.

5.1 Μεθοδολογία και Πλατφόρμα Υλοποίησης

Για την αξιολόγηση του αλγόριθμου χρησιμοποιήθηκε το κατανεμημένο δίκτυο PlanetLab το οποίο παρέχει πληθώρα μηχανών σε ένα ρεαλιστικό περιβάλλον. Πιο κάτω παρατίθεται ένας πίνακας με τις μηχανές που χρησιμοποιήθηκαν στα πειράματα.

Όνομα Μηχανής	*Χώρα*	*Διεύθυνση IP*
planetlab2.utt.fr	Γαλλία	194.254.215.12
anateus.ipv6.lip6.fr	Γαλλία	132.227.62.85
planetlab2.wiwi.hu-berlin.de	Γερμανία	141.20.103.211
planetlab1.informatik.uni-erlangen.de	Γερμανία	131.188.44.100
mars.planetlab.haw-hamburg.de	Γερμανία	141.22.213.35
host2.planetlab.informatik.tu-darmstadt.de	Γερμανία	130.83.166.245
icnalplabs2.epfl.ch	Ελβετία	192.33.193.18
planetlab3.xeno.cl.cam.ac.uk	HB	128.232.103.203
planetlab3.cs.st-andrews.ac.uk	HB	138.251.214.77
planetlab2.xeno.cl.cam.ac.uk	HB	128.232.103.202
utet.ii.uam.es	Ισπανία	150.244.58.161
planetlab1.tlm.unavarra.es	Ισπανία	130.206.158.138
planetlab2.cs.uit.no	Νορβηγία	129.242.19.197
planetlab4.mini.pw.edu.pl	Πολωνία	194.29.178.14
planetlab-1.tagus.ist.utl.pt	Πορτογαλία	193.136.166.66

Πίνακας 5.1 Λίστα μηχανών που χρησιμοποιήθηκαν κατά την διάρκεια των πειραμάτων

Τα παραδείγματα μας εκτελούνται σε ένα περιβάλλον με ένα σύνολο από γραφείς και αναγνώστες χωρίς όμως την χρήση αναδιαμορφώσεων εφόσον απαιτείται η χρήση των αυτομάτων WeakSnapshot και ABD για την ολοκλήρωση μίας τέτοιας λειτουργίας. Στα σημεία όπου απαιτείται η επικοινωνία μαζί με τα συγκεκριμένα αυτόματα (scan_ack, update_ack), έχουν πυροδοτηθεί οι συγκεκριμένες ενέργειες μέσω του αυτόματου σύνθεσης υποθέτοντας ότι δεν έχει ανακαλυφθεί κάποια αλλαγή της σύνθεσης του συστήματος.

Ο αλγόριθμος εκτέλεσης είναι διατυπωμένος σε γλώσσα Java όπως ακριβώς μεταφράζεται από το εργαλείο, χωρίς επιπρόσθετες βελτιστοποιήσεις πέραν των αλλαγών που απαιτούνται ώστε να θεωρείται εκτελέσιμος. Συνεπώς κάνει χρήση των Java Sockets για την υλοποίηση του πρωτοκόλλου TCP/IP. Σε κάθε μηχανή μπορεί να τρέχει μόνο μία διεργασία η οποία εκτελεί μία μόνο λειτουργία. Αυτό οφείλεται στους περιορισμούς του πρωτόκολλου TCP που χρησιμοποιεί το εργαλείο TEMPO.

Κατά την χρήση του δικτύου PlanetLab παρατηρήθηκαν κάποιες δυσκολίες όσο αφορά την δειγματοληψία. Αυτό οφείλεται στις συνθήκες περιβάλλοντος που επικρατούν όπου οι μηχανές μπορούν να καταρρεύσουν ανά πάσα στιγμή, ο χρόνος επικοινωνίας μέσα στο δίκτυο αυξομειώνεται και γενικά η επικοινωνία μεταξύ των διεργασιών τυγχάνει παρεμβολών. Συγκεκριμένα, οι συνθήκες που επικρατούν στο δίκτυο κατά την εκτέλεση κάθε σεναρίου είναι διαφορετικές, προκαλώντας ‘παρεμβολές’ κατά τον υπολογισμό της καθυστέρησης λειτουργίας.

Παράλληλα όμως, οι συνθήκες αυτές εμπόδιζαν τον ορθό τερματισμό κάποιων λειτουργιών. Συγκεκριμένα, παρατηρήθηκε ότι κάποιοι κόμβοι ‘έτρεχαν γρηγορότερα’ από κάποιους άλλους με αποτέλεσμα ελάχιστοι ‘αργοί’ κόμβοι να μένουν πίσω στην λειτουργία τους. Εφόσον χρησιμοποιείτε η πλειοψηφία για ανταλλαγή μηνυμάτων, μία διεργασία όταν λάβει ‘αρκετά’ μηνύματα προχωρεί στο επόμενο βήμα στο πρόγραμμα μεταβάσεων. Μια αργή διεργασία όμως, η οποία βρίσκεται σε προηγούμενο βήμα δεν θα καταφέρει να επικοινωνήσει με την πλειοψηφία εάν περισσότερες από τις μισές διεργασίες έχουν περάσει ήδη στο επόμενο βήμα. Αποτέλεσμα, η διεργασία αποτυγχάνει κατά την επικοινωνία και συνεπώς ολοκλήρωση της ενέργειας της. Αποτυχία ολοκλήρωσης μίας ενέργειας σηματοδοτεί και την αποτυχία εκτέλεσης της συγκεκριμένης επανάληψης του προγράμματος μεταβάσεων εφόσον οι προϋποθέσεις των ενεργειών που ακολουθούν δεν θα τηρούνται. Για την επίλυση αυτού του προβλήματος, απαιτήθηκε η αύξηση του αριθμού επαναλήψεων παράλληλα με τον

αριθμό των διεργασιών. Μετά από δοκιμές (2,4,8,16 κόμβων) υπολογίστηκε ότι χρειάζεται επανάληψη του προγράμματος όσες και ο αριθμός των κόμβων (π.χ. 8 επαναλήψεις για 8 κόμβους), ούτως ώστε να επιτυγχάνετε η επικοινωνία και ολοκλήρωση όλων διεργασιών (μη προβληματικών εκτελέσεων). Η εύρεση μη προβληματικών μηχανών ήταν δύσκολη εφόσον η απόδοση κάθε μηχανής μεταβαλλόταν με την πάροδο του χρόνου, οδηγώντας σε κάποιες περιπτώσεις στην κατάρρευση της.

5.2 Σενάρια Αξιολόγησης

Τα σενάρια που έχουν δημιουργηθεί αφορούν τις επιπτώσεις που έχει η αύξηση των μελών ενός συστήματος στον χρόνο ολοκλήρωσης μίας λειτουργίας αλλά και την σύγκριση της καθυστέρησης των λειτουργιών γραφής και ανάγνωσης.

Τα σενάρια αποτελούνται από 4, 6, 8, 10, 12, 16 γραφείς και αναγνώστες και γίνεται μέτρηση του μέσου χρόνου γραφής και ανάγνωσης.

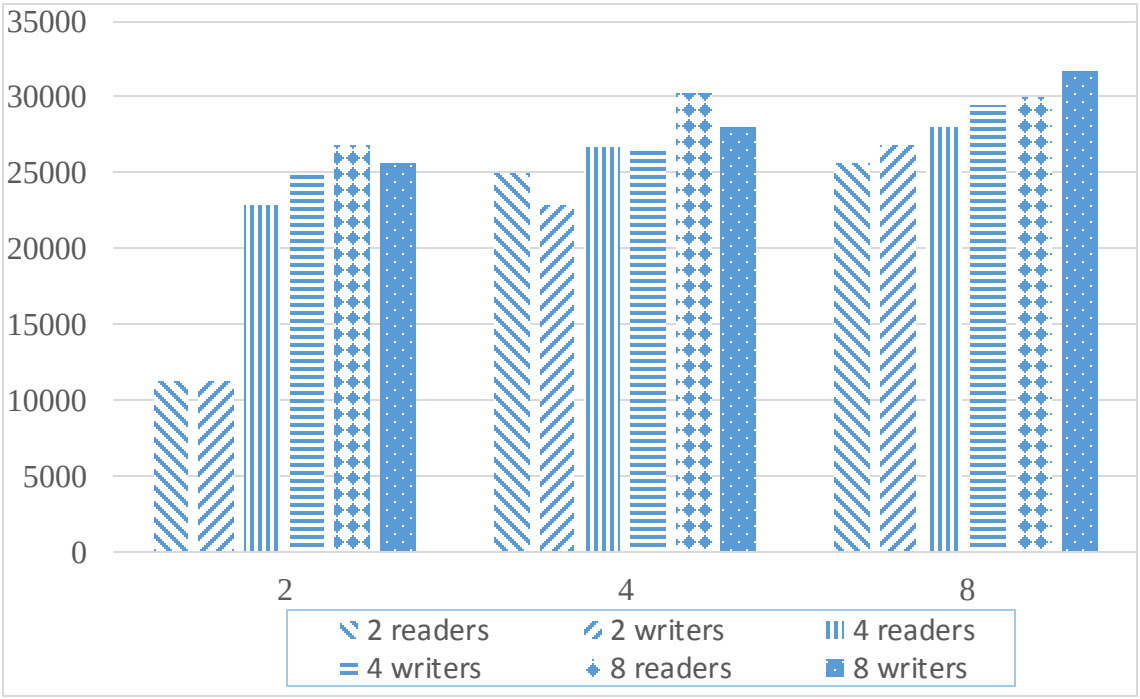
1. Το πρώτο σενάριο αποτελείται από 2,4,8 σταθερούς αναγνώστες και αλλαγή των γραφέων σε 2, 4, 8 σε κάθε περίπτωση.
2. Το επόμενο σενάριο επαναλαμβάνεται η διαδικασία με 2,4,8 σταθερούς γραφείς και αλλαγή των αναγνωστών σε 2, 4, 8 σε κάθε περίπτωση.

Με την σταθεροποίηση του αριθμού των αναγνωστών ή γραφέων, μπορεί να παρατηρηθεί η απόδοση του αλγορίθμου με την αύξηση των γραφέων ή αναγνωστών αντιστοίχως. Πιο απλά, γίνεται ο έλεγχος του κατά πόσο επηρεάζεται ο χρόνος εκτέλεσης από την κάθε λειτουργία ξεχωριστά. Δηλαδή εάν, για παράδειγμα, στο σενάριο με 8 γραφείς και 2 αναγνώστες παρατηρηθεί διαφορετική καθυστέρηση από το σενάριο με 2 γραφείς και 8 αναγνώστες. Επιπλέον τα συγκεκριμένα σενάρια βοηθούν στον υπολογισμό της επεκτασιμότητας του αλγορίθμου αφού αυξάνονται οι διεργασίες σε κάθε σενάριο. Κατά την εκτέλεση γίνονται μετρήσεις του χρόνου εκτέλεσης μίας λειτουργίας. Η χρονική περίοδος που λαμβάνεται υπόψη είναι από την έναρξη της λειτουργίας μέχρι και τον επιτυχή τερματισμό της, αλλά όχι τον τερματισμό ολόκληρης της εκτέλεσης. Αφού ολοκληρωθεί μία εργασία παραμένει στο σύστημα χωρίς όμως να εκτελεί κάποια ενέργεια αφού βρίσκεται σε κατάσταση αδράνειας (idle).

5.3 Αποτελέσματα και Συμπεράσματα

Να σημειωθεί ότι, σε κάθε εκτέλεση των πειραμάτων η αποφυγή καταρρεύσεων τέθηκε αδύνατη. Συνεπώς μελετούμε και την ευρωστία της υλοποίησης σε δυσμενείς συνθήκες που ήταν και το ζητούμενο. Επιπλέον, κάθε πείραμα εκτελέστηκε αρκετές φορές ούτως ώστε να γίνει ορθή δειγματοληψία του χρόνου εκτέλεσης.

Στο σημείο αυτό, ας επισημάνουμε ακόμη μία φορά, ότι κάθε λειτουργία γραφής/ανάγνωσης χρειάζεται δύο φάσεις ολοκλήρωσης. Συνολικά, η εκτέλεση του αλγορίθμου έχει καθυστέρηση λειτουργίας ίση με 3 γύρους επικοινωνίας, δύο κατά την φάση query και propagate αλλά και ένα γύρο κατά την ενέργεια notifyq ο οποίος όμως εκτελείται μόνο μία φορά. Επιπλέον, οι γύροι επικοινωνίας αυξάνονται κάθε φορά που γίνεται επανεκκίνηση της διαδικασίας traverse αλλά δεν θα μελετηθούν σε αυτό το μέρος του αλγορίθμου Dynastore.



Σχήμα 5.1 Μέσος Χρόνος εκτέλεσης με σταθερό αριθμό διεργασιών (2,4,8) και εναλλαγή των γραφέων/αναγνωστών σε 2,4,8 σε κάθε περίπτωση όπως εξηγήθηκε στα σενάρια

Στην γραφική παράσταση παρουσιάζεται ο μέσος χρόνος ολοκλήρωσης μίας λειτουργίας έχοντας σε κάθε περίπτωση σταθερό αριθμό γραφέων ή αναγνωστών (2, 4, 8) και αυξάνοντας τον αριθμό των αναγνωστών ή γραφέων αντίστοιχα (2 , 4, 8). Για παράδειγμα, η τρίτη και τέταρτη μπάρα αντιστοιχούν στην εκτέλεση με 4 αναγνώστες και 2 γραφείς και 4 γραφείς με 2 αναγνώστες αντίστοιχα (δηλαδή ο αριθμός 2 στον οριζόντιο άξονα αντιπροσωπεύει τον αριθμό των σταθερών διεργασιών στο σενάριο). Παρατηρούμε ότι η διαφορά της καθυστέρησης που προσθέτει μια λειτουργία γραφής με μία λειτουργία ανάγνωσης είναι ελάχιστη. Συνεπώς ο χρόνος εκτέλεσης δεν επηρεάζεται από το είδος της λειτουργίας. Επιπλέον , παρατηρούμε ότι όσο αυξάνεται ο αριθμός των διεργασιών (από 4 σε 16) ο χρόνος εκτέλεσης του αλγορίθμου αυξάνεται αλλά ελάχιστα. Αυτό, οφείλεται στις καθυστερήσεις του δικτύου εφόσον μεγαλύτερος αριθμός διεργασιών μελών ισούται με μεγαλύτερο αριθμό μηνυμάτων στο δίκτυο καθώς και αναμονή περισσότερων απαντήσεων.

Κεφάλαιο 6

Συμπεράσματα

6.1 Γενικά Συμπεράσματα

Στην παρούσα Διπλωματική Εργασία υλοποιήθηκε και αξιολογήθηκε η κατανεμημένη υπηρεσία γραφής και ανάγνωσης του συστήματος Dynastore. Η υλοποίηση έγινε μέσω του εργαλείου TEMPO και η αξιολόγηση στο ρεαλιστικό και ασύγχρονο δίκτυο PlanetLab. Κατά την αξιολόγηση διαπιστώθηκε η ευρωστία του αλγόριθμου σε δυσμενείς συνθήκες. Επιπλέον, παρατηρήθηκε ότι οι λειτουργίες γραφής και ανάγνωσης δεν έχουν κάποια διαφορά στον χρόνο εκτέλεσης και ότι ο συνολικός χρόνος εκτέλεσης επηρεάζεται ελάχιστα από τον αριθμό των κόμβων, κυρίως λόγω καθυστερήσεων του δικτύου. Τέλος, διαπιστώθηκε και η ορθότητα της υλοποίησης μας.

6.2 Προβλήματα και Παραδοχές

Κατά την διάρκεια της υλοποίησης του αλγορίθμου εντοπίστηκαν κάποιες αδυναμίες σχετικά με το εργαλείο TEMPO οι οποίες είναι οι εξής :

- Παρουσιάστηκε αδυναμία στην μετάφραση του τύπου δεδομένων Seq[Seq[E]] ο οποίος έπρεπε να τροποποιηθεί στον κώδικα που παράχθηκε σε γλώσσα Java ούτως ώστε να αναγνωρίζεται. Πιο απλά, ο κώδικας που δημιουργήθηκε παρουσίαζε συντακτικά σφάλματα σε κάθε αναφορά του συγκεκριμένου τύπου λόγω του ότι δεν δημιουργήθηκε ορθός constructor για αυτό το αντικείμενο.
- Στην περίπτωση της πλειάδας updade[View, View] που χρησιμοποιήθηκε παρουσιάστηκε πρόβλημα κατά την ανάθεση τιμών στο δεύτερο στοιχείο της

πλειάδας. Συγκεκριμένα, κατά την μετάφραση η ανάθεση όλων των τιμών που αφορούσαν το δεύτερο στοιχείο γινόταν στο πρώτο στοιχείο.

- Επιπλέον, κατά την ένωση (∪) δύο ακολουθιών μέσω της εντολής || η νέα ακολουθία που δημιουργείται περιείχε διπλότυπα αντικείμενα. Απαιτήθηκε η σύνταξη ενός βρόχου for για την ορθή ένωση των στοιχείων.
- Λόγω της προδιαγραφής του καναλιού επικοινωνίας TCP δεν ήταν δυνατή η εκτέλεση περισσότερων από μία διεργασιών σε κάθε μηχανή. Επομένως, για την της αποσφαλμάτωση, χρειάστηκε η εκτέλεση του κώδικα σε συστοιχία υπολογιστών.

Όσο αφορά το κατανεμημένο σύστημα PlanetLab, η εύρεση μηχανής η οποία λειτουργούσε ορθά ήταν σχετικά δύσκολη.

6.3 Μελλοντική Εργασία

Σε αυτή τη Διπλωματική Εργασία έχει μελετηθεί, υλοποιηθεί και αξιολογηθεί η κατανεμημένη υπηρεσία γραφής/ανάγνωσης του συστήματος Dynastore σε τοπικό δίκτυο αλλά και στο δίκτυο PlanetLab. Ο αλγόριθμος, όπως έχουμε ήδη αναφέρει, ανέχεται μόνο σφάλματα κατάρρευσης και όχι κάποιου άλλου είδους σφάλματος. Ωστόσο, εξίσου σημαντική είναι και η παροχή ασφάλειας ενός κατανεμημένου συστήματος σε άλλου είδους σφάλματα (κακόβουλα) . Ως μελλοντική εργασία λοιπόν, θα μπορούσε να δημιουργηθεί μία νέα υπηρεσία η οποία θα παρέχει ανοχή του συστήματος Dynastore σε άλλου είδους σφαλμάτων εκτός της κατάρρευσης.

6.4 Οφέλη από τη Διπλωματική Εργασία

Με τη ολοκλήρωση της Ατομικής Διπλωματικής Εργασίας έχω αποκομίσει γνώσεις γύρο από διαφόρους τομείς της πληροφορικής με τους οποίους δεν είχα την ευκαιρία κατά την διάρκεια των προπτυχιακών μου σπουδών. Συγκεκριμένα, αποκτήθηκαν αρκετές γνώσεις γύρω από τον τομέα Κατανεμημένων Συστημάτων αλλά και Κατανεμημένων Αλγόριθμων. Συγκεκριμένα, δόθηκε η ευκαιρία μελέτης του αλγόριθμου Dynastore. Επιπλέον, θεωρώ σημαντική αλλά και ενδιαφέρουσα την ευκαιρία που παρουσιάστηκε για την εκμάθηση και ενασχόληση με τα Αυτόματα Εισόδου/ Εξόδου (IOA), των Χρονισμένων Αυτομάτων Εισόδου/Εξόδου και του

εργαλείου TEMPO. Εξίσου σημαντικές είναι και οι γνώσεις που αποκτήθηκαν όσο αφορά τα κανάλια MPI και TCP, τα οποία χρησιμοποιήθηκαν καθ' όλη την διάρκεια εκπόνησης της Ατομικής Διπλωματική Εργασίας. Τέλος, παρόλα τα προβλήματα και δυσκολίες που παρουσιάστηκαν, δόθηκε η δυνατότητα εκμάθησης και χρήσης της πλατφόρμας PlanetLab.

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

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

[4] 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.

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

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

[7] Distributed Atomic Storage Alexander A. Shvartsman, Vincent Gramoli, and Nicolas Nicolaou Synthesis Lectures on Distributed Computing Theory Morgan & Claypool publishers, in press.

[8] Dynamic Atomic Storage without Consensus, Marcos K Aguilera, Idit Keidar, Dahlia Makhli, Alexandre Shrader. Journal of the ACM, Vol 58, No. 2, Article 7, Aprill 2011

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

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

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

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

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

[14] Veromondo, Inc. http://www.veromondo.com .

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

[16] Χ. Γεωργίου (2014), Σημειώσεις Μαθήματος ΕΠΛ 601 – Κατανεμημένα Συστήματα, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου.

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

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

[19] Gilbert, Seth, Nancy Lynch, and Alexander Shvartsman."Rambo: a robust, reconfigurable atomic memory service for dynamic networks." Distributed Computing 23.4.2010

[20] Σωτηρούλα Ιωάννου (2014). Προδιαγραφή και Πειραματική Αξιολόγηση Χρονισμένου Κατανεμημένου Αλγόριθμου Διαχείρισης Αρχείων με την Χρήση του εργαλείου TEMPO, Διπλωματική Εργασία, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου.

[21] Χριστίνα Γεωργίου (2013). Προδιαγραφή και Αξιολόγηση Χρονισμένων Κατανεμημένων Αλγόριθμων Εκλογής Αρχηγού με την Χρήση του εργαλείου TEMPO, Διπλωματική Εργασία, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου.

Παράρτημα Α

Σε αυτό το παράρτημα η προδιαγραφές και τα λεξιλόγια των καναλιών MPI και TCP για το εργαλείο TEMPO και τη γλώσσα TIOA

Α.1 Κανάλι Επικοινωνίας MPI

Α.1.1 Προδιαγραφή Λεξιλογίου MPI (Vocabulary.tioa)

```
%% 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[data: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
```

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

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

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

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

```
automaton ReceiveMediator
signature
      output RECEIVE(m:Null[mpi_message])
      input probe(s:Nat)
states
      toRecv:Seq[mpi_message] := {};
transitions
      output RECEIVE(m) where len(toRecv) = 0
      pre
            m = nil();
      output RECEIVE(m) where len(toRecv) ~= 0
      pre
            m = embed(head(toRecv));
      eff
            toRecv := tail(toRecv);
      input probe(s)
      locals
            status:Null[mpi_status] := nil();
      eff
            status := MPI_Iprobe(s);
            if (status ~= nil()) then
                  if (MPI_Test(val(status))) then
                  toRecv := toRecv |- MPI_Irecv(val(status),s);
                  fi
            fi
```

A.2 Προδιαγραφή Καναλιού TCP

A.2.1 Προδιαγραφή Λεξιλογίου TCP (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 alg_specific_voc
types
    message : String
end
vocabulary tcp_specific_voc
    imports  TCPObjectsVoc, TCPNodeVoc ,alg_specific_voc
    types
        Chan_message : Tuple[x : message, 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
       output TCP_bind(local : Node)
       output TCP_accept
       output TCP_stopAccepting
       output TCP_stopListening
       output TCP_rCloseStream(remote : Node)
       output TCP_rClose(remote : Node)
       input TCP_getError(e : Null[JVMError], remote : Node)
       input TCP_respAccept(error : Null[JVMError])
       input TCP_respBind(error : Null[JVMError], local : Node)
       input TCP_respRead(m : Null[Chan_message])

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;
```

ΠΑΡΑΡΤΗΜΑ Β

Στο παράρτημα δίνεται η προδιαγραφή του αλγόριθμου Reader-Writer-Recon

Β.1 Προδιαγραφή Λεξιλόγιου Reader-Writer-Recon (Vocabulary.tioa)

```
%%% TCPObjects Voc
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 dynastore
    imports TCPNodeVoc
    types

        Tag          : Tuple[timestamp:Nat,IP:Node],
        Reply        : Tuple[IP:Node,num:Nat],
        Change       : Tuple[act: Nat,IP:Node], % <<+,->,IP>
        View         : Seq[Change],
        Notify       : Tuple[sender:Node,view : View],
        D_Status     : Enumeration [idle,active],
        Op_Type      : Enumeration [read,write,recon,none],
        Phase        : Enumeration [query,propagate,none],
        Acc          : Tuple[IP:Node,m: Tuple
[mtype:String,ph:Phase,pn:Nat,v:Nat,t:Tag]],
        Stage        : Enumeration
[idle,traverse,traverse_rec1,traverse_rec2,traverse_fix,pick_new_ts,update,up
date_done,scan,scan_done,contactq_start,contactq,notifyq,notifyq_done],
        D_message    : Tuple[mtype:String, ph:Phase,view : View, pn:Nat,
v:Nat, t:Tag, type:String, obj:Node, VT:val_tag_obj, C:Nat, rank : Nat],
        Update       : Tuple[x :View,y : change]


end
vocabulary tcp_specific_voc
    imports  TCPObjectsVoc, TCPNodeVoc ,dynastore%%alg_specific_voc,
    types
        Chan_message : Tuple[x : D_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
    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.2 Προδιαγραφή Αυτόματου Reader-Writer-Recon (ReaderWriterRecon.tioa)

```
automaton ReaderWriterRecon(IP: Node)
signature
    input RWRread
    input RWRwrite(v : Nat)
    input recon(cnf : View)
    input RECEIVE(m : Null[Chan_message])
    input scan_ack(changes : View)
    input update_ack
    input fail
    input init_view(view : View)

    output RWRread_ack(v : Nat)
    output RWRwrite_ack
    output recon_ack
    output SEND(m : Null[Chan_message])
    output scan
    output update(cng : change)
```

```
    internal init
    internal traverse
    internal breadth_first_search
    internal set_desired_view
    internal propagate
    internal traverse_fix

    internal contactq
    internal contactq_fix
    internal notifyq_fix
    internal pick_new_ts

    internal prep_contactq
    internal prep_replies
    internal prep_notify

states
    %% Value of the automaton
    value       : Nat := 0;
    %% Tag of the automaton
    tag         : Tag := [0,[0,0,0,0]];
    %% Max value received from curview.members
    maxV        : Nat := 0;
    %% Max tag received from curview.members
    maxTag      : Tag := [0,IP];
    %% Curview.members who are waiting for reply
    ReplySet    : Seq[Reply] := {};
    %% Curview.members who replied
    acc         : Seq[Acc] := {};
    %% Set of changes collected by WeakSnapshot automata
    changesets  : Seq[View] := {};
    %% New configuration
    newconfig   : View := {};
    %% Phase number
    pnum        : Nat := 0;
    %% Operation type
    op_type     : Op_Type:=none;
    %% Status of the automato
    status      : D_Status := idle;
    %% Stage of the automato
    stage       : Stage := idle;
    %% Phase of the automato
    phase       : Phase := query;
    %% True = failed, False = notfailed
    failed      : Bool  := false;
    %% Initial View
    InitView    : View := {};
    %% Current View
    curview     : View := {};
    %% Desired View
    desiredview : View := {};
    %% Received/Sent Notify on Views
    NotifySet   : Seq[View] := {};
    %% Set who replied on a NOTIFY message
    rcvd_notify : Seq[Notify] := {};
    %% Set of nodes/views in the DAG
    front       : Seq[View] := {};
```

```
    %% tuple [view to be updated,update]
    to_be_updated: Update := [{},[0,[0,0,0,0]]];
    %% Proccess clock
    clock : AugmentedReal := 0;
    %% process messages
    msgs : Seq[Null[Chan_message]] := {};
    %% succesful operations
    success : Nat := 0;
    %% Forward a Notify?
    forwardnotify :Nat := 0;


transitions
    input init_view(view)
    eff
        InitView := view;
        curview  := view;


    internal prep_replies
    locals
        tview : View := {};
    pre
       ~failed /\ status = active;
    eff
        %Reply to requests
        for i : Nat where i< len(ReplySet) do
            msgs := msgs |-
embed([["REPLY",phase,{},ReplySet[i].num,value,tag,"",
[0,0,0,0],[[[0,[0,0,0,0]],0],[0,0,0,0]], 0,0],IP,ReplySet[i].IP]);
        od
        ReplySet    := {};
        %forward any notify
        if(forwardnotify=1) then
            forwardnotify:=0;
            for i : Nat where i < len(NotifySet) do
                tview := NotifySet[i];
                for j : Nat where j< len(tview)do
                    if (tview[j].act = 1 /\ tview[j].IP ~=IP ) then
                        msgs := msgs |-
embed([["NOTIFY",phase,tview,0,0,[0,[0,0,0,0]],"",
[0,0,0,0],[[[0,[0,0,0,0]],0],[0,0,0,0]], 0,0],IP,tview[j].IP]);
                    fi
                od
            od
        fi

   internal prep_notify
   locals
        tview : View := {};
    pre
       ~failed /\ status = active;
       stage = notifyq;
    eff
        %Send notify to view members. View members are denoted as [1,IP] in a
view
        if(curview \notin NotifySet) then
              for i : Nat where i< len(curview) do
```

```
                    if (curview[i].act = 1 /\ curview[i].IP ~=IP) then
                        msgs := msgs |-
embed([["NOTIFY",phase,curview,0,0,[0,[0,0,0,0]],"",
[0,0,0,0],[[[0,[0,0,0,0]],0],[0,0,0,0]], 0,0],IP,curview[i].IP]);
                    fi
                od
        else
            stage := notifyq_done;
        fi



    internal prep_contactq
    pre
       ~failed /\ status = active;
    eff
        %Send read/write requests
        if (stage = contactq) then
            for i : Nat where i< len(curview) do
                if (curview[i].act = 1 /\ curview[i].IP ~=IP) then
                    msgs := msgs |-
embed([["REQUEST",phase,{},pnum,maxV,maxTag,"",
[0,0,0,0],[[[0,[0,0,0,0]],0],[0,0,0,0]], 0,0],IP,curview[i].IP]);
                fi
            od
        fi


    internal init
    pre
        ~failed /\ status = idle; %% switched from "status = active" to
"status = idle" for testing purposes
        [1,IP] \in InitView /\ [0,IP] \notin InitView;
    eff
        status := active;

    input fail
    eff
        failed := true ;

    input RWRread
    eff
        if (~failed /\ status = active) then
            op_type := read;
            newconfig := {};
            stage := traverse;
        fi
    output RWRread_ack(v)
    pre
        ~failed /\ status = active;
        op_type = read;
        stage = notifyq_done ;
        v=maxV ;
   eff
        stage := idle;
        success := success + 1;
```

```
   input RWRwrite(v)
   eff
        if  (~failed /\ status = active) then
            value := v;
            op_type := write;
            newconfig := {};
            stage := traverse;
        fi
    output RWRwrite_ack
    pre
        ~failed /\ status=active ;
        op_type=write;
        stage=notifyq_done;
    eff
        stage:=idle;
        success := success + 1;

    input recon(cnf)
    eff
        if  (~failed /\ status= active) then
            op_type:=recon;
            newconfig:=cnf;
            stage:=traverse;
        fi

    output recon_ack
    pre
        ~failed /\ status= active;
        op_type= recon;
        stage = notifyq_done;
    eff
        stage := idle;
        success := success + 1;

    output SEND(m)
    pre
        msgs ~= {};
        m = head(msgs);
        ~failed /\ status = active;


    eff
        msgs := tail(msgs);

    input RECEIVE(m)
    locals
        reply: String := "REPLY";
        request: String := "REQUEST";
        notify: String := "NOTIFY";
        tview : View := {};
        total : Nat :=0;
    eff
        if (~failed /\ status=active) then
            if (val(m).x.mtype =reply) then
                acc := acc |-
[val(m).sender,[val(m).x.mtype,val(m).x.ph,val(m).x.pn,val(m).x.v,val(m).x.t]
];
            fi
```

```
            if (val(m).x.mtype =request) then
                if (val(m).x.ph=propagate /\
val(m).x.t.timestamp>tag.timestamp) then
                    value := val(m).x.v;
                    tag   := val(m).x.t;
                fi
                if([val(m).sender,val(m).x.pn] \notin ReplySet) then
                    ReplySet := ReplySet |- [val(m).sender,val(m).x.pn];
                fi
            fi
            if (val(m).x.mtype =notify) then
                if (stage=notifyq) then %% collect replies
                    rcvd_notify := rcvd_notify |-
[val(m).sender,val(m).x.view];
                else
                    if (val(m).x.view \notin NotifySet) then %% add the view
the first time we receive it
                        NotifySet := NotifySet |- val(m).x.view;
                        forwardnotify:=1;
                        tview := val(m).x.view;
                        for i:Nat where i<len(curview) do
                            if(curview[i] \in tview) then
                                total :=total +1;
                            fi
                        od
                        if(len(curview) < len(tview)) then
                                 if (total = len(curview)) then % then
curview C view

                                     if ([1,IP] \in tview) then
                                             status := active;
                                     fi
                                     curview := tview ;
                                     stage := traverse; %%restart traverse
                                 fi
                        fi
                    fi
                fi
            fi
        fi


    internal notifyq_fix
    locals
        curview_members : Nat :=0;
        rcvdnot_members : Nat :=0;
    pre
        ~failed /\ status = active;
        stage = notifyq;
    eff
        for i:Nat where i< len(rcvd_notify) do
            if (rcvd_notify[i].view = curview) then
                rcvdnot_members := rcvdnot_members + 1;
            fi
        od
        for i:Nat where i< len(curview) do

            if (curview[i].act = 1 /\ [0,curview[i].IP] \notin curview) then
                curview_members := curview_members + 1 ;
```

```
            fi
        od
        if(rcvdnot_members >= (div(curview_members,2)+1)) then
            stage:=notifyq_done;
            rcvd_notify:={};
        fi

    internal traverse

    pre
        ~failed /\ status = active ;
        stage = traverse ;
    eff
        front := {};
        front := front |- curview;
        desiredview := curview;
        for i:Nat where i < len(newconfig) do% desiredview = curview U
newconfig
            if (newconfig[i] \notin desiredview) then
                desiredview := desiredview |- newconfig[i];
            fi
        od
        stage := traverse_rec1;

    internal breadth_first_search
    locals
      %  w       : Seq[Change] := {};
        minlen  : Nat := 0;
        minpos  : Nat := 0;
        updated : change := [0,[0,0,0,0]];
    pre
        ~failed /\ status = active;
        stage=traverse_rec1;
      %  w:=l s.t |l| = min l in front {|l|}
    eff
        minlen :=len(front[0]);

        for i:Nat where i < len(front) do
            if (minlen > len(front[i])) then
                minlen := len(front[i]);
                minpos := i;
            fi
        od
        curview := front[minpos];

        if ([1,IP] \notin curview) then
            status := idle; % halt if not a members of the view
        else
            if curview ~= desiredview then
               % tobeupdated:= <w,desiredview\w>

                for i:Nat where i < len(desiredview) do
                    if (desiredview[i] \notin curview) then
                        updated := desiredview[i];
                    fi
                od
                to_be_updated.x := curview;
                to_be_updated.y := updated;
```

```
                stage:= update;
            else
                phase:=query;
                stage:=scan;
            fi
        fi

    internal set_desired_view
    locals
        tfront : Seq[View] := {}; %front := front \ w;
        tview  : View :={};
        c      : View :={};
    pre
        ~failed /\ status = active;
        stage = traverse_rec2;
        changesets ~= {};
    eff
        %front := front \ w;
        for i:Nat where i < len(front) do
           if (front[i] ~= curview) then
                tfront := tfront |- front[i];
            fi
        od
        front:=tfront;
        desiredview := curview ;
        for  i:Nat where i< len(changesets) do
            tview:=curview;
            c := changesets[i];
            for k:Nat where k<len(c) do
                %desiredview = desiredview U c
                %front = front U {w U c}
                if(c[k] \notin desiredview) then
                    desiredview := desiredview |- c[k];
                fi
                if(c[k] \notin tview) then
                    tview := tview |- c[k];
                fi
            od
            if (tview \notin front) then
                front := front |- tview;
            fi
        od
        stage := traverse_rec1;

    internal propagate
    pre
        ~failed /\ status = active;
        stage = traverse_rec2;
        changesets = {};
    eff
        phase :=propagate;
        if (op_type = write) then
            stage := pick_new_ts;
        else
            stage := contactq_start;
        fi
     internal traverse_fix
     pre
```

```
        ~failed /\ status = active;
        stage = traverse_fix;
    eff
        curview := desiredview;
        stage:= notifyq;

    output scan
    pre
        ~failed /\ status = active;
        stage = scan;



    input scan_ack(changes)
    eff
        if (~failed /\ status = active) then
            if(changes ~= {}) then
                changesets := changesets |- changes;
            fi
            if (phase = query) then
                stage := contactq_start;
            else % phase = propagate
                if (changesets ~= {}) then
                    stage := traverse_rec1;
                else
                    stage := traverse_fix;
                fi
            fi
        fi

    internal pick_new_ts
    pre
        ~failed /\ status = active;
        stage = pick_new_ts;
        changesets = {};
    eff
        maxTag := [maxTag.timestamp+1,IP];
        maxV:=value;
        stage:= contactq_start;

    internal contactq
    pre
        ~failed /\ status = active;
        stage = contactq_start;
    eff
        acc := {};
        pnum := pnum+1;
        stage:= contactq;

    internal contactq_fix
    locals
        max_tag : Tag := [0,[0,0,0,0]]; % max tag in acc
        tag_pos : Nat := 0;             % max tag position in acc
        t       : Tag := [0,[0,0,0,0]];
        v       : Nat := 0;
        replied : Nat := 0; % number of replies
        members : Nat := 0; % total curview.members
    pre
```

```
        ~failed /\ status = active;
        stage = contactq;
        %repliedP ={j:<j,m>} in acc /\ m.pn = pnum
        %majority(w.members) C= repliedP

    eff
        for i:Nat where i< len(acc) do
            if (acc[i].m.pn = pnum) then
                replied := replied + 1;
            fi
        od
        for i:Nat where i< len(curview) do
            %if an IP exist and is not to be 0d
            if (curview[i].act = 1 /\ [0,curview[i].IP] \notin curview) then
                members := members + 1 ;
            fi
        od
        if(replied >= (div(members,2)+1)) then
            if (op_type = read) then
                % t := max(j,m) where  <j,m> in acc{m.tag}
                % v := v' : <t,v'> in acc
                max_tag :=  acc[0].m.t;
                tag_pos :=  0;
                for i:Nat where i<len(acc) do
                    if (acc[i].m.t.timestamp>max_tag.timestamp) then
                      max_tag :=  acc[i].m.t;
                      tag_pos :=  i;
                    fi
                od
                t := max_tag;
                v := acc[tag_pos].m.v;
                if (t.timestamp > maxTag.timestamp) then
                    %(maxV,maxtag) := (t,v)
                    maxV := v;
                    maxTag :=t;
                fi
            fi
            if (phase = query) then
                stage := traverse_rec2;
            else
                stage := scan; %phase = propagate
            fi
        fi
    output update(cng)
    pre
        ~failed /\ status = active;
        [curview,cng] = to_be_updated;
        stage = update;
    eff
        to_be_updated := [{},[0,[0,0,0,0]]];


    input update_ack
    eff
        if (~failed /\ status = active) then
            phase := query;
            stage := scan;
        fi
```

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

B.3 Προδιαγραφή Αυτόματου Σύνθεσης (Composition.tioa)

```
%%% .: Vocabulary :.
include "Vocabulary.tioa"
imports dynastore, TCPObjectsVoc, TCPNodeVoc, tcp_specific_voc, JVMSocket,
JVMServerSocket, ChannelVoc
%%% .: TCP Mediator Automata :.
include "TCPRecvMed.tioa"
include "TCPSendMed.tioa"
include "TCPChanMed.tioa"
%%% .: Algorithm Automata :.
include "ReaderWriterRecon.tioa"

automaton Composition(n1:Nat, n2:Nat, n3:Nat, n4:Nat, port:Nat, timeout:Nat,
OP : Nat)
components
    RWR : ReaderWriterRecon([n1,n2,n3,n4]);
    SM  : SendMed(port, timeout);
    RM  : RecvMed(port, timeout);
    CM  : ChanMed(port, timeout);

schedule
    states
        %IP of this Node
        IP      : Node              := [n1,n2,n3,n4];
        %IP of nodes in view
        DYNodes : Seq[Node]         := {};
        %Message to send
        ms      : Null[Chan_message]:= nil();
        %Message to receive
        mr      : Null[Chan_message]:= nil();

        isConn  : Bool              := false;

        error   : Null[JVMError]    := nil();
        %Number of runs
        runs    : Nat               := 0;
        %Initial View of the system.
        InitView: View       :={};
        % Update
        %update_d: View       :={};
        % Reconfiguration
        recon_r : View       :={};
        % Scan Results
        scan_cnf: View  :={};
        % Value
        val     :Nat                :=5;
        % wait for messages
        new_msg  :Bool               := true;
do
```

```
        DYNodes := DYNodes |- [150,244,58,161];
        DYNodes := DYNodes |- [194,29,178,14];
        DYNodes := DYNodes |- [129,242,19,197];
        DYNodes := DYNodes |- [130,206,158,138];
        runs    := 1 ;
        InitView:= InitView |- [1,[150,244,58,161]];
        InitView:= InitView |- [1,[194,29,178,14]];
        InitView:= InitView |- [1,[129,242,19,197]];
        InitView:= InitView |- [1,[130,206,158,138]];
        %Initialize
        fire input RWR.init_view(InitView);
        %% 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(DYNodes) do
            if(IP ~= DYNodes[n]) then
                fire output SM.TCP_senderOpen(DYNodes[n], port);
                follow SM.DELAY duration len(DYNodes)*200;
                %% accept only on new connection
                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(DYNodes[n], port, isConn);
            fi
        od %end of for
        if (error =nil()) then
            follow RWR.Time duration timeout;
        fi
        fire internal RWR.init;
        %% Read
        if (OP = 0) then
            fire input RWR.RWRread;
        fi
        %% Write
        if (OP = 1) then
            fire input RWR.RWRwrite(val);
        fi

        for i: Nat where i < runs do
            follow RWR.Time duration 1;
            %% Recon
            if (OP = 2) then
                fire input RWR.recon(recon_r);
            fi
            fire internal RWR.traverse;
            while (RWR.changesets~={} ) do   % Restart traverse if changesets
=/= {}
                fire internal RWR.breadth_first_search;
                if( RWR.stage = update ) then
                    fire output RWR.update(RWR.to_be_updated.y);
```

```
                    fire input  RWR.update_ack;
                fi
                fire output RWR.scan;
                fire input RWR.scan_ack(scan_cnf);
                fire internal RWR.contactq;
                % prepare messages to contact quorum
                fire internal RWR.prep_contactq;
                %SEND
                new_msg:=true;
                while (new_msg = true) do
                    ms:=nil();
                    fire output RWR.SEND(ms);
                    if(ms~=nil()) then
                        fire output SM.TCP_write(ms,
val(ms).sender,val(ms).destination);
                    else
                        new_msg := false;
                    fi
                od
                follow SM.DELAY duration 200;
                % extract messages from channel if there are any
                fire output RM.TCP_read;
                new_msg := true;
                while (new_msg = true) do
                    fire output CM.TCP_respRead(mr);
                    if( mr ~= nil()) then
                        fire output RM.RECEIVE (mr);
                        print val(mr);
                    else
                        new_msg := false;
                    fi
                od
                % prepare replies if any
                fire internal RWR.prep_replies;
                %SEND
                new_msg:=true;
                while (new_msg = true) do
                    ms:=nil();
                    fire output RWR.SEND(ms);
                    if(ms~=nil()) then
                        fire output SM.TCP_write(ms,
val(ms).sender,val(ms).destination);
                    else
                        new_msg := false;
                    fi
                od
                follow SM.DELAY duration 200;
                % extract messages from channel if there are any
                fire output RM.TCP_read;
                new_msg := true;
                while (new_msg = true) do
                    fire output CM.TCP_respRead(mr);
                    if( mr ~= nil()) then
                        fire output RM.RECEIVE (mr);
                        print val(mr);
                    else
                        new_msg := false;
                    fi
```

```
                od
                fire internal RWR.contactq_fix;
                fire internal RWR.set_desired_view;
            od

            fire internal RWR.propagate;
            if (OP = 1) then
                fire internal RWR.pick_new_ts;
            fi
            fire internal RWR.contactq;
            % prepare messages to contact quorum
            fire internal RWR.prep_contactq;
            %SEND
            new_msg:=true;
            while (new_msg = true) do
                ms:=nil();
                fire output RWR.SEND(ms);
                if(ms~=nil()) then
                    fire output SM.TCP_write(ms,
val(ms).sender,val(ms).destination);
                else
                    new_msg := false;
                fi
            od
            follow SM.DELAY duration 200;
            % extract messages from channel if there are any
            fire output RM.TCP_read;
            new_msg := true;
            while (new_msg = true) do
                fire output CM.TCP_respRead(mr);
                if( mr ~= nil()) then
                    fire output RM.RECEIVE (mr);
                    print val(mr);
                else
                    new_msg := false;
                fi
            od
            fire internal RWR.prep_replies;
            %SEND
                new_msg:=true;
                while (new_msg = true) do
                    ms:=nil();
                    fire output RWR.SEND(ms);
                    if(ms~=nil()) then
                        fire output SM.TCP_write(ms,
val(ms).sender,val(ms).destination);
                    else
                        new_msg := false;
                    fi
                od
            follow SM.DELAY duration 200;
            % extract messages from channel if there are any
            fire output RM.TCP_read;

                new_msg := true;
                while (new_msg = true) do
                    fire output CM.TCP_respRead(mr);
                    if( mr ~= nil()) then
```

```
                        fire output RM.RECEIVE (mr);
                        print val(mr);
                    else
                        new_msg := false;
                    fi
                od
            fire internal RWR.contactq_fix;
            fire output RWR.scan;
            fire input RWR.scan_ack(scan_cnf);
            fire internal RWR.traverse_fix;
             % prepare messages to notify quorum
            fire internal RWR.prep_notify;
            %SEND
            new_msg:=true;
            while (new_msg = true) do
                ms:=nil();
                fire output RWR.SEND(ms);
                if(ms~=nil()) then
                    fire output SM.TCP_write(ms,
val(ms).sender,val(ms).destination);
                else
                    new_msg := false;
                fi
            od

            follow SM.DELAY duration 200;
            % extract messages from channel if there are any
            fire output RM.TCP_read;
            new_msg := true;
            while (new_msg = true) do
                fire output CM.TCP_respRead(mr);
                if( mr ~= nil()) then
                    fire output RM.RECEIVE (mr);
                    print val(mr);
                else
                    new_msg := false;
                fi
            od
            fire internal RWR.prep_replies;
            %SEND
            new_msg:=true;
            while (new_msg = true) do
                ms:=nil();
                fire output RWR.SEND(ms);
                if(ms~=nil()) then
                    fire output SM.TCP_write(ms,
val(ms).sender,val(ms).destination);
                else
                    new_msg := false;
                fi
            od
            follow SM.DELAY duration 200;
            % extract messages from channel if there are any
            fire output RM.TCP_read;
            new_msg := true;
            while (new_msg = true) do
                fire output CM.TCP_respRead(mr);
                if( mr ~= nil()) then
```

```
                    fire output RM.RECEIVE (mr);
                    print val(mr);
                else
                    new_msg := false;
                fi
            od
            fire internal RWR.notifyq_fix;
            if (OP = 0 ) then
                fire output RWR.RWRread_ack(val);
                print val;
            fi
            if (OP = 1 ) then
                fire output RWR.RWRwrite_ack;
            fi
            if (OP = 2 ) then
               fire output RWR.recon_ack;
            fi
        od
    od
```

ΠΑΡΑΡΤΗΜΑ Γ

Οι μηχανές στο κατανεμημένο δίκτυο PlanetLab δεν υποστηρίζουν την εκτέλεση προγραμμάτων Java αφού δεν είναι εγκατεστημένο το περιβάλλον Java κατά την αρχικοποίηση του κόμβου. Ακολουθούν οδηγίες για εγκατάσταση της Sun Java:

1. Επισκεπτόμαστε την ιστοσελίδα www.oracle.com και από την επιλογή Java→Java SE→Downloads επιλέγουμε την πιο πρόσφατη έκδοση JRE για Linux Συστήματα 32 ή 64 bit ανάλογα με τη μηχανή που βρισκόμαστε. Αντιγράφουμε τον σύνδεσμο που υπάρχει στο αρχείο *.rpm π.χ. \http://download.oracle.com/otn-pub/java/jdk/8u45-b14/jre-8u45-linux-i586.rpm
2. Έπειτα, με την βοήθεια ενός επεξεργαστή κειμένου, προσθέστε μπροστά από τον σύνδεσμο την εντολή «sudo wget --no-cookies --no-check-certificate --header "Cookie: gpw_e24=http%3A%2F%2Fwww.oracle.com%2F; oraclelicense=accept-securebackup-cookie"». Εκτελέστε την εντολή σε μία μηχανή στο δίκτυο PlanetLab για να κατεβεί το πακέτο εγκατάστασης.
3. Αφού κατεβεί το αρχείο εκτελούμε την εντολή sudo rpm –iv filename όπου filename είναι το όνομα του κατεβασμένου αρχείου για την εγκατάσταση.
4. Με το τέλος της εντολής εκτελούμε java –version για να βεβαιωθούμε ότι εγκαταστάθηκε με επιτυχία το πακέτο