ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ &
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
Τομέας Επικοινωνιών, Ηλεκτρονικής &
Συστημάτων Πληροφορικής

Διδακτορική Διατριβή

Ιωάννη Δ. Ψωρούλα
Διπλωματούχου Χημικού Μηχανικού
του Εθνικού Μετσόβιου Πολυτεχνείου

ΠΡΩΤΟΤΥΠΕΣ ΤΕΧΝΙΚΕΣ ΓΙΑ ΤΗΝ ΥΛΟΠΟΙΗΣΗ ΠΕΡΙΒΑΛΛΟΝΤΩΝ ΠΡΟΣΟΜΟΙΩΣΗΣ ΔΙΑΚΡΙΤΩΝ ΓΕΓΟΝΟΤΩΝ ΓΙΑ ΥΠΟΛΟΓΙΣΤΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΑΘΗΝΑ Σεπτέμβριος 2007

Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Τομέας Επικοινωνιών, Ηλεκτρονικής & Συστημάτων Πληροφορικής

Διδακτορική Διατριβή

Ιωάννη Δ. Ψωρούλα
Διπλωματούχου Χημικού Μηχανικού
του Εθνικού Μετσόβιου Πολυτεχνείου

ΠΡΩΤΟΤΥΠΕΣ ΤΕΧΝΙΚΕΣ ΓΙΑ ΤΗΝ ΥΛΟΠΟΙΗΣΗ ΠΕΡΙΒΑΛΛΟΝΤΩΝ ΠΡΟΣΟΜΟΙΩΣΗΣ ΔΙΑΚΡΙΤΩΝ ΓΕΓΟΝΟΤΩΝ ΓΙΑ ΥΠΟΛΟΓΙΣΤΙΚΑ ΣΥΣΤΗΜΑΤΑ

Επιβλέπων Καθηγητής
Βασίλειος Λούμος

Συμβουλευτική Επιτροπή
Β. Λούμος
Ελ. Καγιάφας
Γ. Στασινόπουλος

Εγκρίθηκε από την Εξεταστική Επιτροπή την 14η Σεπτεμβρίου 2007

Β. Λούμος	Ελ. Καγιάφας	Γ. Στασινόπουλος	Μ. Θεολόγου
Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ

Δ. Βέργαδος	Ι. Αναγνωστόπουλος	Η. Κουκούτσης
Λέκτορας	Λέκτορας	Επικ. Καθηγητής
Πανεπιστήμιο Αιγαίου	Πανεπιστήμιο Αιγαίου	ΕΜΠ

Ευχαριστίες

Στην ζωή ορισμένα γεγονότα τα οποία αρχικά φαίνονται ασήμαντα, τελικά είναι αυτά που πολλές φορές καθορίζουν την ζωή ενός ανθρώπου.

Θα ήθελα να ευχαριστήσω θερμά τους δασκάλους μου, κ.κ. Καθηγητές Βασίλη Λούμο και Ελευθέριο Καγιάφα για την εμπιστοσύνη που μου έδειξαν, την αμέριστη βοήθεια που είχα όλα αυτά τα χρόνια, και την κατανόηση τους σε δύσκολες στιγμές που αντιμετώπισα ακόμα και σε προσωπικό επίπεδο. Αδιαμφισβήτητη είναι η συνεισφορά του κ.κ. Καθηγητή Γεωργίου Στασινόπουλου ο οποίος μου συμπαραστάθηκε όποτε χρειάστηκε. Επίσης, θα ήθελα να ευχαριστήσω τους κ.κ. Καθηγητή Μ. Θεολόγου, Επίκ. Καθηγητή Η. Κουκούτση και το Λέκτορα Δ. Βέργαδο για την τιμή που μου έκαναν να συμμετάσχουν στην επταμελή επιτροπή της διατριβής μου.

Θα ήθελα να ευχαριστήσω επίσης, όλα τα παιδιά του εργαστηρίου, τον Άγγελο για την άριστη συνεργασία και τις πολύτιμες συμβουλές του, το Γιώργο Κούζα που αποτελούσε την ψυχή του εργαστηρίου, την Έλενα, το Νίκο, το Γιώργο Μπάρδη, το Δημήτρη, το Βασίλη και το Θεόδωρο οι οποίοι και αυτοί ήταν πάντα πρόθυμοι να βοηθήσουν. Ιδιαιτέρως, θα ήθελα να ευχαριστήσω το φίλο Χρήστο για την σημαντική του βοήθεια στις δύσκολες στιγμές και το φίλο Σωτήρη για την ηθική συμπαράσταση.

Σήμερα μου δίνεται η δυνατότητα να γράψω τα λόγια αυτά, γιατί επίσης είχα την τύχη να γνωρίσω τρεις ανθρώπους τους οποίους θα ήθελα να ευχαριστήσω ιδιαιτέρως. Το Γιάννη Αναγνωστόπουλο ο οποίος στάθηκε κάτι παραπάνω από φίλος και πραγματικά δεν μπορώ να εκφράσω με λόγια την εκτίμηση που του τρέφω. Το Δημήτρη Μανίκη ο οποίος ήταν ο πρώτος που πίστεψε σε μένα και με έφερε σε επαφή με τους δασκάλους μου, και τον Κώστα Περράκη ο οποίος υπήρξε πνευματικός μου πατέρας και είναι αυτός που με μύησε στην επιστήμη των υπολογιστών.

Τέλος, θα ήθελα να ευχαριστήσω τους φίλους μου, Γιώργο Κόκκορη, Γιώργο Αθανασίου και Βαγγέλη Πρεβενά, καθώς επίσης κ.κ Καθηγήτρια Γλυκερία Κακάλη για το κουράγιο που μου έδιναν όλα αυτά τα χρόνια.

Αφιερώνω την διατριβή μου στην οικογένειά μου και στη σύντροφο της ζωής μου, Βαρβάρα, που με έχουν στηρίξει σε οποιαδήποτε απόφαση έχω πάρει.

Αθήνα,
6 Σεπτεμβρίου 2007

Ψωρούλας Δ. Ιωάννης

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΕΡΙΕΧΟΜΕΝΑ **4**

ΕΙΣΑΓΩΓΗ **9**

ΑΝΑΛΥΣΗ ΚΕΦΑΛΑΙΩΝ – ΔΟΜΗ ΤΗΣ ΔΙΑΤΡΙΒΗΣ **10**

ΚΕΦΑΛΑΙΟ 1 10
ΚΕΦΑΛΑΙΟ 2 10
ΚΕΦΑΛΑΙΟ 3 10
ΚΕΦΑΛΑΙΟ 4 10
ΠΑΡΑΡΤΗΜΑ Ι 11
ΠΑΡΑΡΤΗΜΑ ΙΙ 11

1 ΑΛΓΟΡΙΘΜΟΙ ΕΞΙΣΟΡΡΟΠΗΣΗΣ ΦΟΡΤΙΟΥ **15**

1.1 ΤΕΧΝΟΛΟΓΙΑ ΠΛΕΓΜΑΤΟΣ (GRID COMPUTING) 16
1.2 ΠΕΡΙΓΡΑΦΗ ΑΛΓΟΡΙΘΜΩΝ ΕΞΙΣΟΡΡΟΠΗΣΗΣ ΦΟΡΤΙΟΥ 18
1.2.1 Βασικές κατηγορίες *18*
1.2.2 Πολιτικές συμπεριφοράς *19*
1.2.3 Πρότυπα παραδείγματα *20*
1.2.3.1 Αλγόριθμοι αρχικοποίησης αποστολέα (Sender initiated algorithms) 21
1.2.3.2 Αλγόριθμοι αρχικοποίησης παραλήπτη (Receiver initiated algorithms) 22
1.2.3.3 Αλγόριθμοι συμμετρικής αρχικοποίησης (Symmetrically initiated algorithms) 23

2 ΠΕΙΡΑΜΑΤΙΚΟ ΠΕΡΙΒΑΛΛΟΝ **27**

2.1 ΣΥΣΤΗΜΑΤΑ ΚΑΙ ΜΟΝΤΕΛΑ 28
2.2 ΠΡΟΣΟΜΟΙΩΣΗ 29
2.2.1 Τύποι μοντέλων προσομοίωσης *29*
2.2.2 Προσομοίωση διακριτών γεγονότων (Discrete event simulation) - Μηχανισμός εξέλιξης του χρόνου *30*
2.2.3 Τύποι προσομοίωσης διακριτών γεγονότων *31*
2.2.4 Συστατικά και οργάνωση ενός μοντέλου προσομοίωσης διακριτών γεγονότων *34*
2.2.5 Πλεονεκτήματα, μειονεκτήματα και δυσκολίες της προσομοίωσης *35*
2.3 ΠΕΙΡΑΜΑΤΙΚΟ ΠΕΡΙΒΑΛΛΟΝ ΠΡΟΣΟΜΟΙΩΣΗΣ 37
2.3.1 Βασικές οντότητες *37*
2.3.2 Αλγόριθμος κεντρικής μονάδας επεξεργασίας *38*
2.3.2.1 Μοντέλο 38
2.3.2.2 Αξιολόγηση μοντέλου 40
2.3.3 Υλοποίηση *42*
2.3.3.1 Βασικές κλάσεις 42
2.3.3.2 Έγγραφο διαμόρφωσης του συστήματος 44
2.3.3.3 Σενάριο λειτουργίας 46

3 ΠΕΙΡΑΜΑΤΙΚΟ ΜΕΡΟΣ **51**

3.1 ΕΞΕΤΑΖΟΜΕΝΟΙ ΑΛΓΟΡΙΘΜΟΙ 51
3.2 ΜΕΤΡΑ ΑΠΟΔΟΣΗΣ (PERFORMANCE METRICS) 53
3.2.1 Συνήθης μέτρα απόδοσης *53*
3.2.2 Μέτρα απόδοσης συστήματος *54*

3.3 ΕΠΙΔΡΑΣΗΣ ΤΗΣ ΤΑΧΥΤΗΤΑΣ ΕΠΙΚΟΙΝΩΝΙΑΣ 56
3.4 ΕΠΙΔΡΑΣΗ ΤΗΣ ΟΜΟΙΟΓΕΝΕΙΑΣ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ 61
3.5 ΠΡΟΒΛΗΜΑ ΠΑΡΩΧΗΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΑΣ (OUT OF DATE INFORMATION PROBLEM) 64
3.6 ΕΠΙΔΡΑΣΗ ΤΗΣ ΤΟΠΟΛΟΓΙΑΣ 68
3.6.1 Περιγραφή τοπολογιών 68
3.6.2 Υπολογισμός διορθωτικού παράγοντα 69
3.6.3 Διαφοροποίηση αποφάσεων 71
3.6.4 Υπολογισμός κόστους τοπολογιών 73
3.6.5 Πειραματικά αποτελέσματα 76
3.7 ΣΥΜΠΕΡΑΣΜΑΤΑ 82

4 ΠΡΟΤΕΙΝΟΜΕΝΟ ΕΡΓΑΛΕΙΟ ΠΡΟΣΟΜΟΙΩΣΗΣ 85

4.1 ΒΑΣΙΚΟ ΜΟΝΤΕΛΟ (BASIC MODEL) 86
4.1.1 Υλοποίηση βασικού μοντέλου 87
4.1.2 Ορισμός βασικού μοντέλου 89
4.2 ΜΗΧΑΝΙΣΜΟΣ ΕΝΑΛΛΑΚΤΙΚΩΝ ΣΥΜΠΕΡΙΦΟΡΩΝ (ALTERNATIVE BEHAVIORS MECHANISM) 91
4.2.1 Μοντέλο μηχανισμού 92
4.2.2 Ορισμός μεταβιβαστών-εξωτερικών εκτελεστών 94
4.2.3 Παραδείγματα 95
4.2.4 Ιδιαίτερη εφαρμογή του μηχανισμού 97
4.3 ΜΗΧΑΝΙΣΜΟΣ ΚΑΝΟΝΩΝ ΚΑΤΑΤΑΞΗΣ (SORTING RULES MECHANISM) 99
4.3.1 Ορισμός κανόνων 101
4.3.2 Παραδείγματα 101
4.4 ΜΗΧΑΝΙΣΜΟΣ ΜΗΝΥΜΑΤΩΝ (MESSAGING) 105
4.4.1 Εφαρμογή μηχανισμού κανόνων κατάταξης 107
4.4.1.1 Παραδείγματα 108
4.5 ΜΗΧΑΝΙΣΜΟΣ ΟΡΩΝ-ΕΝΕΡΓΕΙΩΝ (CONDITIONS – ACTIONS MECHANISM) .. 110
4.5.1 Υλοποίηση 111
4.5.2 Μηχανισμός ανίχνευσης διακεκριμένων γεγονότων (Event Detection) 114
4.5.3 Παραδείγματα 115
4.6 ΜΗΧΑΝΙΣΜΟΣ ΔΙΑΧΕΙΡΙΣΗΣ ΠΛΗΡΟΦΟΡΙΩΝ (INFORMATION MANAGEMENT) 121
4.6.1 Προδιαγραφές 121
4.6.1.1 Πρότυπο περιγραφής πληροφορίας 121
4.6.1.2 Μοντελοποίηση πληροφορίας 122
4.6.1.3 Μέσο αποθήκευσης 125
4.6.1.3.1 Εναλλακτικοί τρόποι αποθήκευσης 126
4.6.2 Υλοποίηση 128
4.6.2.1 Μηχανισμός αποθήκευσης αντικειμένων 128
4.6.2.2 Μηχανισμός ανάκτησης αντικειμένων 133
4.6.2.3 Γραφικό περιβάλλον 136
4.6.2.4 Υλοποίηση ως Διαδικτυακή Υπηρεσία 137
4.7 ΤΟΠΟΛΟΓΙΑ 140
4.8 ΣΥΓΚΡΙΣΗ ΜΕ ΥΠΑΡΧΟΥΣΕΣ ΠΛΑΤΦΟΡΜΕΣ 141
4.8.1 JavaSim 142
4.8.2 SimJava 142
4.8.3 SimKit 143

4.8.4 SSJ *144*
4.8.5 J-Sim *145*
4.8.6 Ptolemy II – DE domain *145*
4.8.7 Swarm, RePast *146*
4.8.8 Mason *147*
4.8.9 SimGrid *148*
4.8.10 GridSim *149*
4.9 ΣΥΜΠΕΡΑΣΜΑΤΑ 153

ΒΙΒΛΙΟΓΡΑΦΙΑ 157

ΑΡΘΡΑ *157*
ΒΙΒΛΙΑ *160*
ΔΙΚΤΥΑΚΟΙ ΤΟΠΟΙ *161*

ΠΑΡΑΡΤΗΜΑ Ι 167

ΠΑΡΑΡΤΗΜΑ ΙΙ 171

ΚΑΤΑΛΟΓΟΣ ΣΥΝΤΕΤΜΗΜΕΝΩΝ ΟΡΩΝ 182

ΒΙΟΓΡΑΦΙΚΟ ΣΗΜΕΙΩΜΑ 187

ΕΠΙΣΤΗΜΟΝΙΚΕΣ ΔΗΜΟΣΙΕΥΣΕΙΣ 189

ΑΡΘΡΑ ΣΕ ΠΕΡΙΟΔΙΚΑ 189
ΑΡΘΡΑ ΣΕ ΠΕΡΙΟΔΙΚΑ (ΥΠΟ ΚΡΙΣΗ) 189
ΑΡΘΡΑ ΣΕ ΣΕΙΡΕΣ ΒΙΒΛΙΩΝ 189
ΑΡΘΡΑ ΣΕ ΠΡΑΚΤΙΚΑ ΣΥΝΕΔΡΙΩΝ 189

Εισαγωγή

Εισαγωγή

Η προσομοίωση διακριτών γεγονότων είναι μια ευρέως χρησιμοποιούμενη μέθοδος για τη μελέτη πολύπλοκων συστημάτων (για παράδειγμα, ένα γενετικό ή οικονομικό σύστημα) καθώς τα περισσότερα από αυτά δεν μπορούν να περιγραφούν ικανοποιητικά με κάποιο μαθηματικό μοντέλο που μπορεί να λυθεί αναλυτικά. Επιπλέον, καθιστά δυνατή την μελέτη ενός συστήματος που έχει μακρόχρονη εξέλιξη σε πολύ μικρότερο χρόνο και αντίστροφα, και επιτρέπει την εκτίμηση της απόδοσης του, κάτω από προβλεπόμενο σύνολο λειτουργικών συνθηκών. Το γεγονός αυτό έχει σαν αποτέλεσμα την δημιουργία πλήθους πλατφορμών, που είτε αποτελούν πρωτογενώς πλατφόρμες προσομοίωσης διακριτών γεγονότων είτε στηρίζονται σε αυτές.

Τα υπολογιστικά συστήματα αποτελούν αναμφίβολα σημαντικό εργαλείο τόσο στο επιστημονικό χώρο όσο και στην βιομηχανία καθώς, είτε καθιστούν εφικτή την ολοκλήρωση κάποιων εργασιών, είτε επιτρέπουν την ολοκλήρωση τους σε πολύ πιο σύντομο χρόνο από ότι θα απαιτούνταν χωρίς αυτά. Η απόδοση των συστημάτων αυτών εξαρτάται κυρίως από την διαχείριση των πόρων τους. Η διαχείριση αυτή μεταφράζεται σε ένα σύνολο κανόνων οι οποίοι καθορίζουν τις σχέσεις και την αλληλεπίδραση μεταξύ των οντοτήτων που απαρτίζουν το σύστημα, και εφαρμόζονται, συνήθως, ανάλογα με τα χαρακτηριστικά και την κατάσταση του συστήματος σε μία δεδομένη χρονική στιγμή. Κάθε σύνολο τέτοιων κανόνων στην ουσία υλοποιεί ένα αλγόριθμο, καλούμενο αλγόριθμο εξισορρόπησης φορτίου.

Η παρούσα διατριβή εστιάζεται στην δημιουργία ενός εργαλείου που μπορεί να χρησιμοποιηθεί για την ανάπτυξη πλατφορμών προσομοίωσης παρόμοιων με τις προαναφερθείσες. Ο σχεδιασμός και η υλοποίηση του εργαλείου αυτού πραγματοποιείται βάσει νέων τεχνικών προγραμματισμού, συστημάτων και αρχιτεκτονικών τα οποία διασφαλίζουν με τον καλύτερο δυνατό τρόπο την ευελιξία, την επεκτασιμότητα και την διαφάνεια του συστήματος ως προς τον χρήστη, χαρακτηριστικά τα οποία δεν εμφανίζονται στις υπάρχουσες πλατφόρμες. Επιπλέον, κατά τον σχεδιασμό του, ελήφθησαν υπόψη τα μοντέλα λειτουργίας των βασικών κατηγοριών των αλγόριθμων εξισορρόπησης φορτίου με σκοπό την δημιουργία μιας «φιλικής» διεπιφάνειας προς την υλοποίηση πλατφορμών προσομοίωσης υπολογιστικών συστημάτων.

Πρόδρομος του εν λόγω εργαλείου αποτελεί η υλοποίηση πειραματικού περιβάλλοντος του οποίου η εύκαμπτη αρχιτεκτονική επιτρέπει την εφαρμογή πλήθους σεναρίων λειτουργίας απλών αλγόριθμων εξισορρόπησης φορτίου. Το περιβάλλον αυτό χρησιμοποιείται για την μελέτη των βασικών παραγόντων, όπως η ταχύτητα επικοινωνίας, η τοπολογία, το φορτίο, η πολυπλοκότητα του πρωτοκόλλου επικοινωνίας μεταξύ των οντοτήτων, οι οποίοι επηρεάζουν σημαντικά την αποτελεσματικότητα ενός υπολογιστικού συστήματος και κατά συνέπεια αποτελούν τους βασικούς παράγοντες τους οποίους θα πρέπει να λαμβάνει υπόψη ένας αλγόριθμος εξισορρόπησης φορτίου. Η μελέτη αυτή στηρίζεται στην συγκριτική εξέταση των πειραματικών αποτελεσμάτων που προκύπτουν από την εφαρμογή τριών αλγορίθμων, του κλασικού στατικού αλγόριθμου Round-Robin και δύο δυναμικών, του LUIA και UIA οι οποίοι κατασκευάστηκαν για τις ανάγκες τις παρούσας διατριβής.

Στις παραγράφους που ακολουθούν παρουσιάζεται συνοπτική περιγραφή των κεφαλαίων της παρούσας εργασίας.

Ανάλυση κεφαλαίων – δομή της διατριβής

Η διατριβή είναι δομημένη σε 4 κεφάλαια και 2 παραρτήματα τα οποία και περιγράφονται ακολούθως.

Κεφάλαιο 1

Στο **κεφάλαιο 1** αναφέρονται οι βασικές κατηγορίες και τα μοντέλα λειτουργίας των αλγόριθμων εξισορρόπησης φορτίου. Η αναφορά αυτή είναι απαραίτητη για την κατανόηση i) των υπό εξέταση αλγόριθμων του κεφαλαίου 3 και ii) των σχεδιαστικών κανόνων του προτεινόμενου εργαλείου προσομοίωσης στο κεφάλαιο 4. Επιπλέον, αναφέρονται εν συντομία τα βασικά χαρακτηριστικά της σχετικά νέας τεχνολογίας του πλέγματος υπολογισμού (grid computing), αφού αρκετές πλατφόρμες προσομοίωσης (με τις οποίες θα ασχοληθούμε στο κεφάλαιο 4) για την μελέτη τέτοιων συστημάτων έχουν ήδη υλοποιηθεί.

Κεφάλαιο 2

Στο **κεφάλαιο 2** περιγράφεται η υλοποίηση ενός ευέλικτου συστήματος (καλούμενο και ως πειραματικό περιβάλλον, experimental framework) προσομοίωσης διακριτών γεγονότων για την μελέτη απλών αλλά βασικών αλγόριθμων εξισορρόπησης φορτίου. Αναλυτικότερα, στο κεφάλαιο αυτό παρουσιάζονται, η μοντελοποίηση των βασικών οντοτήτων (υπολογιστική μονάδα, φορτίο) και οι θεμελιώδεις κλάσεις του πειραματικού περιβάλλοντος, ο αλγόριθμος για την προσομοίωση της κεντρικής μονάδας επεξεργασίας, καθώς επίσης το έγγραφο μέσω του οποίου καθορίζεται το μοντέλο του συστήματος που επιθυμείται να προσομοιωθεί. Στο τέλος του κεφαλαίου, παρατίθεται υποδειγματικό παράδειγμα κατάστρωσης και εκτέλεσης πειράματος.

Στην αρχή του κεφαλαίου, για την καλύτερη κατανόηση του εν λόγω πειραματικού περιβάλλοντος και γενικότερα πλατφορμών προσομοίωσης διακριτών γεγονότων, με τις οποίες θα ασχοληθούμε στο κεφάλαιο 4, πραγματοποιείται αναφορά για την μοντελοποίηση και την προσομοίωση συστημάτων.

Κεφάλαιο 3

Στο **κεφάλαιο 3** περιέχονται τα αποτελέσματα από την προσομοίωση τριών απλών αλγόριθμων εξισορρόπησης φορτίου. Σκοπός δεν είναι η ανάπτυξη ενός νέου αλγορίθμου αλλά η μελέτη της επίδρασης βασικών παραμέτρων, όπως η ταχύτητα επικοινωνίας και η τοπολογία, στην αποτελεσματικότητα ενός αλγορίθμου και κατά συνέπεια στην απόδοση ενός συστήματος. Η μελέτη αυτή οδηγεί σε χρήσιμα συμπεράσματα τα οποία μπορούν να γενικευτούν σε οποιαδήποτε σύστημα με παρόμοια χαρακτηριστικά και εφαρμοζόμενους μηχανισμούς.

Αναλυτικότερα, μελετάται η επίδραση i) της ταχύτητας επικοινωνίας μεταξύ των οντοτήτων του συστήματος συναρτήσει του επιβαλλόμενου φορτίου, ii) της ομοιογένειας του συστήματος, iii) του προβλήματος της παρωχημένης πληροφορίας και iv) της τοπολογίας, στην απόδοση του συστήματος.

Επιπλέον, παρουσιάζονται και σχολιάζονται τα μεγέθη (μέτρα απόδοσης) τα οποία ορίστηκαν με σκοπό την μελέτη των πειραματικών αποτελεσμάτων και την αξιολόγηση των εφαρμοζόμενων αλγόριθμων εξισορρόπησης φορτίου.

Σημειώνεται ότι, η διεξαγωγή των πειραμάτων πραγματοποιήθηκε με χρήση του πειραματικού περιβάλλοντος που περιγράφεται στο κεφάλαιο 2.

Κεφάλαιο 4

Στο **κεφάλαιο 4** περιγράφεται η υλοποίηση ενός εργαλείου προσομοίωσης, καλούμενο ως DynConPBTool (Dynamic Configurable Property-Based Simulation Toolkit), που μπορεί να χρησιμοποιηθεί για την ανάπτυξη πλατφορμών, που είτε αποτελούν πρωτογενώς πλατφόρμες προσομοίωσης διακριτών γεγονότων είτε στηρίζονται σε αυτές. Ο σχεδιασμός και η υλοποίηση του εργαλείου αυτού πραγματοποιείται με τέτοιο τρόπο ώστε να διασφαλίζεται η

ευελιξία, η επεκτασιμότητα και η διαφάνεια του συστήματος ως προς τον χρήστη, χαρακτηριστικά τα οποία δεν εμφανίζονται στις υπάρχουσες πλατφόρμες. Επιπλέον, κατά τον σχεδιασμό του, ελήφθησαν υπόψη τα μοντέλα λειτουργίας των βασικών κατηγοριών των αλγόριθμων εξισορρόπησης φορτίου με σκοπό την δημιουργία μιας «φιλικής» διεπιφάνειας προς την υλοποίηση πλατφορμών προσομοίωσης υπολογιστικών συστημάτων.

Το DynConPBTool αποτελείται από ένα σύνολο μηχανισμών, όπως οι "εναλλακτικών συμπεριφορών", "όρων-ενεργειών και ανίχνευσης διακεκριμένων γεγονότων" και "διαχείρισης πληροφοριών" οι οποίοι αλληλεπιδρούν μεταξύ τους και με το υπόλοιπο σύστημα, χωρίς όμως να χάνουν την αυτονομία τους (loosely coupled components). Το γεγονός αυτό έχει σαν αποτέλεσμα να μπορούν να εφαρμοστούν σε οποιοδήποτε σύστημα προσομοίωσης. Οι παραπάνω μηχανισμοί, το μοντέλο καθώς και τα υπόλοιπα χαρακτηριστικά του εν λόγω εργαλείου περιγράφονται λεπτομερώς στο κεφάλαιο 4, συνοδευόμενα από παραδείγματα έτσι ώστε να γίνει κατανοητή η χρήση και η λειτουργία τους.

Επιπλέον, στο τέλος του κεφαλαίου αυτού πραγματοποιείται μελέτη της επίδρασης του προτεινόμενου εργαλείου σε υπάρχουσες πλατφόρμες προσομοίωσης, από όπου και αποδεικνύεται η αξία του.

Παράρτημα Ι

Το **παράρτημα Ι** περιέχει τις περιγραφές των κλάσεων του πειραματικού περιβάλλοντος καθώς επίσης τις μεταξύ τους σχέσεις υπό την μορφή διαγραμμάτων UML.

Παράρτημα ΙΙ

Το **παράρτημα ΙΙ** περιέχει τις περιγραφές των κλάσεων του DynConPBTool καθώς επίσης τις μεταξύ τους σχέσεις υπό την μορφή διαγραμμάτων UML.

Κεφάλαιο 1

Αλγόριθμοι Εξισορρόπησης Φορτίου

1 Αλγόριθμοι εξισορρόπησης φορτίου

Η απόδοση των υπολογιστικών συστημάτων, είτε με την μορφή ενός συνόλου αυτόνομων υπολογιστών συνδεδεμένων με τοπικό δίκτυο (distributed computing system) είτε γενικότερα με την μορφή ενός υπολογιστικού πλέγματος (grid computing), εξαρτάται κυρίως από τον βαθμό και τον τρόπο αξιοποίησης των μονάδων από τις οποίες απαρτίζεται. Συστήματα υψηλών προδιαγραφών μπορεί να αποβούν αναποτελεσματικά καθώς η διαχείριση των εργασιών δεν πραγματοποιείται κατάλληλα. Για παράδειγμα, η συνύπαρξη μονάδων (ή διεκπεραιωτών) με έντονη επεξεργαστική δραστηριότητα και άλλων με υψηλό ποσοστό αδράνειας αποτελεί ένδειξη κακής διαχείρισης των υπολογιστικών πόρων ενός συστήματος. Οι αλγόριθμοι εξισορρόπησης φορτίου (Load Balancing Algorithms) καλούνται να δώσουν λύση σε τέτοιου είδους προβλήματα, αναπτύσσονται δηλαδή με σκοπό την κατανομή των εργασιών με τέτοιο τρόπο ώστε το σύστημα να παρουσιάζει την βέλτιστη απόδοση.

Πριν αναπτυχθούν οι βασικές κατηγορίες των αλγόριθμων εξισορρόπησης φορτίου αξίζει να αναφερθούν τα βασικά χαρακτηριστικά των υπολογιστικών πλεγμάτων και η διαφοροποίηση τους από τα συνήθης υπολογιστικά συστήματα.

1.1 Τεχνολογία πλέγματος (Grid computing)

Η μεγάλη ανάπτυξη του Internet, η τεχνολογική εξέλιξη των προσωπικών υπολογιστών και η ανάπτυξη του ενδιάμεσου λογισμικού (middleware) έχουν δημιουργήσει μια άλλη δυναμική στον όρο υπολογιστικό περιβάλλον. Ο συνδυασμός όλων αυτών δίνει την δυνατότητα να διαμοιραστούν πόροι, όπως είναι η υπολογιστική ισχύς, ο αποθηκευτικός χώρος και άλλα σε παγκόσμιο επίπεδο, κατανεμημένα. Για παράδειγμα ένας ερευνητής μπορεί να μοιράζεται μέσω του Internet, την υπολογιστική ισχύ του, τον αποθηκευτικό χώρο του και γενικά τα πειραματικά δεδομένα του με χιλιάδες άλλους επιστήμονες από διάφορα κράτη. Οι νέες αυτές τεχνολογίες είναι γνωστές ως τεχνολογίες Πλέγματος-Grid.

Η κύρια διαφορά ανάμεσα στα grids και τα συνήθη κατανεμημένα υπολογιστικά συστήματα (π.χ. clusters), είναι στον τρόπο διαχείρισης των υπολογιστικών πόρων, καθώς στην πρώτη περίπτωση χρησιμοποιείται μια πιο αποκεντρωμένη και πιο σύνθετη αρχιτεκτονική (προκειμένου να διασφαλίσει τα προαναφερθέντα χαρακτηριστικά) σε σύγκριση με την δεύτερη.

Τα κύρια χαρακτηριστικά ενός Grid είναι τα εξής :

- *Μεγάλη κλίμακα*. Ένα Grid πρέπει να είναι ικανό να χειρίζεται υπολογιστικούς πόρους που μπορεί να κυμαίνονται από λίγους μόνο μέχρι και αρκετά εκατομμύρια. Έτσι, ανακύπτει το σοβαρό πρόβλημα της αποφυγής πιθανής υποβάθμισης της απόδοσης όσο το μέγεθος του Grid αυξάνει.
- *Γεωγραφική κατανομή*. Οι πόροι σε ένα Grid μπορούν να βρίσκονται σε διαφορετικά γεωγραφικά μέρη.
- *Ετερογένεια*. Ένα Grid διαθέτει πόρους υλικού και λογισμικού που μπορεί να διαφέρουν πολύ, από δεδομένα, αρχεία και προγράμματα μέχρι αισθητήρες, επιστημονικά όργανα, συσκευές απεικόνισης, υπολογιστές, υπερυπολογιστές και δίκτυα υπολογιστών, οι οποίοι όμως όσο διαφορετικοί και να είναι θα πρέπει να είναι δυνατό να συνεργάζονται μεταξύ τους για την επίτευξη των λειτουργιών του.
- *Διαμοίραση πόρων*. Οι πόροι σε ένα Grid ανήκουν σε πολλούς διαφορετικούς οργανισμούς που επιτρέπουν σε άλλους οργανισμούς να τους χρησιμοποιήσουν. Επομένως, μπορούν να χρησιμοποιηθούν απομακρυσμένοι πόροι από τις διάφορες εφαρμογές και έτσι αυξάνεται η απόδοση και μειώνεται το κόστος εκτέλεσης.
- *Διαφορετικές πολιτικές διαχείρισης*. Κάθε οργανισμός μπορεί να έχει διαφορετική πολιτική διαχείρισης και ασφάλειας κάτω από τις οποίες δίνει πρόσβαση των πόρων του σε άλλους. Αυτό έχει σαν αποτέλεσμα, το ήδη δύσκολο πρόβλημα της ασφάλειας δικτύου να περιπλέκεται ακόμα περισσότερο λόγω της ανάγκης να ληφθούν υπόψη όλες οι διαφορετικές αυτές πολιτικές.
- *Συντονισμός πόρων*. Οι πόροι σε ένα Grid πρέπει να συντονίζονται έτσι ώστε να παρέχουν αθροιστικά πολύ μεγάλες υπολογιστικές δυνατότητες.
- *Διαφανής πρόσβαση*. Ένα Grid πρέπει να φαίνεται στον απλό χρήστη σαν ένας απλός πραγματικός υπολογιστής, χωρίς να τον αναγκάζει να εμπλέκεται σε τεχνικές λεπτομέρειες.
- *Αξιόπιστη πρόσβαση*. Ένα Grid πρέπει να εγγυάται την παροχή υπηρεσιών κάτω από σαφείς απαιτήσεις ποιότητας υπηρεσίας. Η ανάγκη για αξιόπιστες υπηρεσίες είναι βασική, αφού οι χρήστες απαιτούν εγγυήσεις για προβλέψιμη, παρατεταμένη και υψηλού επιπέδου απόδοση.
- *Συνεπής πρόσβαση*. Ένα Grid πρέπει να δομηθεί με προτυποποιημένες υπηρεσίες, πρωτόκολλα και διεπαφές, επομένως κρύβοντας την ετερογένεια των πόρων του αλλά επιτρέποντας την κλιμάκωση. Χωρίς τα πρότυπα αυτά, η ανάπτυξη εφαρμογών και η απρόσκοπτη πρόσβαση θα ήταν αδύνατες.
- *Απρόσκοπτη πρόσβαση*. Το Grid πρέπει να παραχωρεί πρόσβαση στους διαθέσιμους πόρους, προσαρμοζόμενο σε ένα δυναμικό περιβάλλον στο οποίο τα σφάλματα των πόρων είναι κάτι το δεδομένο. Με αυτό δεν υπονοείται πως οι πόροι είναι διαθέσιμοι από όλους, αλλά ότι το Grid πρέπει να έχει τέτοια συμπεριφορά έτσι ώστε να παρέχει τη μέγιστη απόδοση από τους διαθέσιμους κάθε φορά πόρους.

Σύμφωνα με τα παραπάνω χαρακτηριστικά, ο ακόλουθος ορισμός για το Grid μπορεί να δοθεί. Ως Grid χαρακτηρίζεται μια μεγάλης κλίμακας γεωγραφικά κατανεμημένη υποδομή υλικού και λογισμικού, που αποτελείται από ετερογενείς δικτυωμένους πόρους που ανήκουν και διαμοιράζονται από πολλούς οργανισμούς, οι οποίοι είναι συντονισμένοι να παρέχουν διαφανή, αξιόπιστη, απρόσκοπτη και συνεπή υποστήριξη σε ένα ευρύ φάσμα εφαρμογών. *[Γερονάτσιος, 2006]*

Ως προς τις λειτουργίες τους τα Grid διακρίνονται σε δύο μεγάλες κατηγορίες.

- *Υπολογιστικά Grids* (*Computational Grids*), τα οποία αποτελούν την συλλογή κατανεμημένων υπολογιστικών υποδομών (συνήθως clusters) οι οποίες λειτουργούν ως ενιαίος επεξεργαστής. Η ωφέλεια είναι η πραγματοποίηση επεξεργασιών με μεγάλες υπολογιστικές απαιτήσεις ταχύτερα, πιο αποτελεσματικά, με μικρό κόστος και χρησιμοποιώντας υπάρχουσες υποδομές. Τέτοιες επεξεργασίες χρησιμοποιούνται τόσο στον επιστημονικό χώρο όσο και στην βιομηχανία.

- *Grids Δεδομένων* (*Data Grids*) που επιτρέπουν στους χρήστες και στις εφαρμογές να διαχειρίζονται εύκολα και αποτελεσματικά πληροφορίες που είναι αποθηκευμένες σε διαφορετικές τοποθεσίες. Επιπλέον, παρέχουν μειωμένο κόστος διότι δεν υπάρχει ανάγκη για μεταφορά, αντιγραφή και συγκέντρωση δεδομένων σε ένα κεντρικό σημείο.

Τα βασικά στρώματα μίας υποδομής Πλέγματος είναι τα ακόλουθα:

- *Στρώμα δικτύου* (*Network layer*), το οποίο είναι το στρώμα όπου περιλαμβάνονται οι φυσικές διασυνδέσεις και ο αντίστοιχος δικτυακός εξοπλισμός.

- *Στρώμα πόρων* (*Resource layer*), όπου οι πόροι μπορεί να είναι υπολογιστικοί ή αποθηκευτικοί με έμφαση στους πρώτους και στις διάφορες υλοποιήσεις (π.χ. clusters).

- *Στρώμα ενδιάμεσου λογισμικού* (*Middleware layer*), όπου περιλαμβάνονται όλες οι απαραίτητες υπηρεσίες ανακοίνωσης, διαμοιρασμού, χρονοπρογραμματισμού και πρόσβασης πόρων, ασφάλειας με έμφαση στην εξακρίβωση ταυτότητας (authentication) και εξουσιοδότηση (authorization) και καθώς και μέτρηση χρήσης (accounting).

- Στρώμα εφαρμογών και υπηρεσιών (Application & Serviceware layer), όπου τελικά συνδέονται οι διάφορες εφαρμογές. *[Μπογόρδος, 2005]*

Το στρώμα του ενδιάμεσου λογισμικού, προφανώς έχει ιδιαίτερη σημασία για την ανάπτυξη της τεχνολογίας Grid καθώς είναι υπεύθυνο για την διαχείριση των υπολογιστικών πόρων. Αρχικά, δημιουργήθηκε ένας αριθμός από κατά παραγγελία middleware το οποίο όμως τελικά δεν απέδωσε τα αναμενόμενα αποτελέσματα. Η αποτυχία αυτή οδήγησε στην αναζήτηση ενός μοντέλου που θα οδηγούσε στην ενοποίηση όλων των πόρων και υπηρεσιών ανεξαρτήτου υλοποίησης. Η απάντηση ήλθε από τον Ian Foster και τους συνεργάτες του, οι οποίοι περιέγραψαν την *Ανοικτή Αρχιτεκτονική Υπηρεσιών Grid* (*Open Grid Services Architecture-OGSA*) που παρέχει έναν ομοιόμορφο τρόπο περιγραφής υπηρεσιών Grid και ορίζει έναν κοινό τρόπο λειτουργίας τους *[Foster, 02]*. Η OGSA χρησιμοποιεί έναν συνδυασμό από τα καλύτερα χαρακτηριστικά των τεχνολογιών του Grid και των *Διαδικτυακών υπηρεσιών* (*Web Services*, *WS*). Σύμφωνα με την αρχιτεκτονική αυτή, όλοι οι πόροι, είτε φυσικοί είτε λογικοί, μοντελοποιούνται και παρουσιάζονται σαν υπηρεσίες οι οποίες «κτίζονται» πάνω στο μοντέλο της *Αρχιτεκτονικής Ανοικτών Υπηρεσιών* (*Service Oriented Architecture*, SOA) που ουσιαστικά υλοποιείται με την βοήθεια των WS. *[Γερονάτσιος, 2006]*

1.2 Περιγραφή αλγόριθμων εξισορρόπησης φορτίου

Στις παραγράφους που ακολουθούν επιχειρείται μία στοιχειώδης κατάταξη των αλγορίθμων εξισορρόπησης φορτίου βάσει βιβλιογραφικών αναφορών. Οριοθετείται το βασικό πλαίσιο συμπεριφοράς τους και παρέχονται πρότυπα παραδείγματα για την κατανόηση των κανόνων και αρχών λειτουργίας τους.

1.2.1 Βασικές κατηγορίες

Μερικές από τις αξιόλογες προσπάθειες ταξινόμησης των εν λόγω αλγορίθμων αποτελούν οι εργασίες *[Casavant, 88]* και *[Shivaratri, 92]*. Σύμφωνα με αυτές, οι αλγόριθμοι ανάλογα με την πληροφορία και τους μηχανισμούς που χρησιμοποιούν για τον καθορισμό της συμπεριφοράς του συστήματος διαχωρίζονται στις ακόλουθες τρεις κύριες κατηγορίες:

➢ *Στατικοί αλγόριθμοι* (*static algorithms*), οι οποίοι αφενός χρησιμοποιούν προϋπάρχουσα πληροφορία του συστήματος, αφετέρου οι κανόνες που εφαρμόζουν παραμένουν αμετάβλητοι με τον χρόνο. Χαρακτηριστικό παράδειγμα αποτελεί ο ντετερμινιστικός κυκλικός αλγόριθμος *round-robin*, η συμπεριφορά του οποίου εξαρτάται από το πλήθος N των υπολογιστικών μονάδων, καθώς η ανάθεση της εργασίας i γίνεται στην μονάδα με δείκτη το υπόλοιπο της διαίρεσης του i με το N. Οι στατικοί αλγόριθμοι, στην πλειοψηφία των περιπτώσεων, δεν είναι αποτελεσματικοί αφού δεν λαμβάνουν υπόψη την κατάσταση του συστήματος κατά την διεξαγωγή των αποφάσεων τους.

➢ *Δυναμικοί αλγόριθμοι (dynamic algorithms)*. Στην κατηγορία αυτή ανήκουν οι αλγόριθμοι οι αποφάσεις των οποίων βασίζονται στην κατάσταση του συστήματος (για παράδειγμα, το πλήθος των εργασιών ή το φορτίο στις υπολογιστικές μονάδες) και κατά συνέπεια εμφανίζουν αυξημένη αποτελεσματικότητα έναντι των στατικών αλγόριθμων. Η αποτελεσματικότητα τους οφείλεται στην ικανότητα εκμετάλλευσης διακυμάνσεων της καταστάσεως του συστήματος, ενώ μειονέκτημα τους είναι η επιπρόσθετη επιβάρυνση του από την συλλογή, αποθήκευση και επεξεργασία των πληροφοριών για την λήψη αποφάσεων.

Οι δυναμικοί αλγόριθμοι μπορούν να διαχωριστούν περαιτέρω στους
i) *αλγόριθμους καταμερισμού του φορτίου* (*load sharing algorithms*) οι οποίοι έχουν σαν στόχο την βελτιστοποίηση της απόδοσης του συστήματος με την ελαχιστοποίηση των μονάδων που βρίσκονται σε κατάσταση αδράνειας (*unshared state/idle state – αμέτοχη / αδρανής κατάσταση*) *[Livny, 82]* και
ii) στους *αλγόριθμους εξισορρόπησης φορτίου (load balancing algorithms)*. Αυτοί δεν επιδιώκουν απλώς την ελαχιστοποίηση των αδρανών καταστάσεων αλλά επιπλέον επιχειρούν την εξισορρόπηση του φορτίου στο σύνολο των μονάδων. Η εφαρμογή τους απαιτεί επιπρόσθετη δέσμευση πόρων η οποία πολλές φορές υπερισχύει έναντι των πλεονεκτημάτων που παρουσιάζουν. *[Krueger, 87]*. Η λειτουργία των αλγόριθμων αυτών στηρίζεται στην παραδοχή ότι οι διαθέσιμες πληροφορίες για την κατάσταση του συστήματος είναι αρκετά ακριβείς ώστε να αποφευχθεί η ατέρμονη μεταφορά των εργασιών χωρίς την ολοκλήρωση τους *[Gupta, 99]*.

Στην παρούσα εργασία οι αλγόριθμοι γενικά αναφέρονται ως *εξισορρόπησης φορτίου*. Στις περιπτώσεις όπου απαιτείται η διάκριση μεταξύ των δύο κατηγοριών διευκρινίζεται και ακολουθείται η παραπάνω ονοματολογία.

➢ *Προσαρμοστικοί αλγόριθμοι (adaptive algorithms)*. Οι αλγόριθμοι αυτοί αποτελούν ιδιαίτερη κατηγορία των δυναμικών αλγόριθμων. Κύριο χαρακτηριστικό τους είναι η ικανότητα προσαρμογής των παραμέτρων τους και των μηχανισμών τους στην κατάσταση του συστήματος. Για παράδειγμα, ένας αμιγώς δυναμικός αλγόριθμος εφαρμόζει ενιαία πολιτική για όλες τις περιοχές λειτουργίας του συστήματος ακόμα και σε αυτές όπου δεν είναι εξίσου αποδοτικός. Αντιθέτως, ένας προσαρμοστικός αλγόριθμος θα μπορούσε να

αποτελούνταν από δύο δυναμικούς όπου ο καθένας θα ενεργοποιούνταν στις περιπτώσεις που παρουσιάζει αυξημένη αποτελεσματικότητα. Στην εργασία *[Krueger, 94]* προτείνεται ένα μοντέλο λειτουργίας το οποίο εκμεταλλεύεται τα πλεονεκτήματα των αλγόριθμων αρχικοποίησης παραλήπτη (*receiver initiated*) και αποστολέα (*sender initiated*), ενώ παράλληλα κατευθύνει με δυναμικό τρόπο την αναζήτηση της κατάλληλης συμπληρωματικής (αποστολέα ή δέκτη, όπως θα δούμε στην συνέχεια) μονάδας προς την μεταφορά της εργασίας. Ανάλογες προσπάθειες πραγματοποιήθηκαν στις εργασίες *[Shivaratri, 90]*, *[Svensson, 92]*.

Επιπρόσθετος διαχωρισμός των αλγόριθμων μπορεί να λάβει χώρα σύμφωνα με το εύρος συμμετοχής των υπολογιστικών μονάδων στην διαδικασία λήψεως αποφάσεων. Έτσι σε ένα

- *κεντρικό* (*centralized*) ή συγκεντρωτικό μοντέλο λειτουργίας, οι αρμοδιότητες των υπολογιστικών μονάδων έχουν αυστηρά αποκλειστικό χαρακτήρα. Χαρακτηριστικό παράδειγμα αποτελεί η ύπαρξη μόνο μίας μονάδας υπεύθυνης για την καταγραφή και διατήρηση της πληροφορίας του συστήματος. Οι αλγόριθμοι αυτοί δεν παρουσιάζουν ιδιαίτερη αξιοπιστία καθώς μία πιθανή δυσλειτουργία τους μπορεί να προκαλέσει την κατάρρευση ολόκληρου του συστήματος. Πιθανή λύση αποτελεί η ύπαρξη επιπρόσθετων μονάδων οι οποίες θα αναλαμβάνουν τον ρόλο αυτών που δεν δύνανται να επιτελέσουν το έργο τους. Επιπλέον, αδυναμία παρόμοιων συστημάτων αποτελεί η αυξημένη πιθανότητα συμφόρησης των κεντρικών μονάδων λόγο της εκτεταμένης επικοινωνίας με τις υπόλοιπες. Το πλεονέκτημα που παρουσιάζουν είναι η σχετικά εύκολη υλοποίηση τους.

- Αντιθέτως, το *αποκεντρωτικό* (*decentralized*) μοντέλο ενθαρρύνει την συμμετοχή πολλών μονάδων προς την υλοποίηση ενός ρόλου. Κύρια πλεονέκτημα τέτοιων συστημάτων είναι η επεκτασιμότητα (*scalability*) που παρουσιάζουν, καθώς και η ικανότητα ταχείας ανάκτησης σε περίπτωση αποτυχίας. Μειονέκτημα, ο αυξημένος βαθμός πολυπλοκότητας.

1.2.2 Πολιτικές συμπεριφοράς

Η συμπεριφορά ενός αλγόριθμου εξισορρόπησης φορτίου είναι αποτέλεσμα των ακόλουθων τεσσάρων πολιτικών *[Shivaratri, 92]*.

- Η *πολιτική μεταφοράς* (*Transfer policy*) η οποία αποφαίνεται για την συμμετοχή του διεκπεραιωτή στην διαδικασία μεταφοράς εργασίας, ως αποστολέας (*sender*) η παραλήπτης (*receiver*). Ο κύριος όγκος των προτεινόμενων πολιτικών βασίζονται σε ένα είδος πολιτικής κατωφλίου (*Threshold policy*) κατά την οποία η απόφαση λαμβάνεται με κριτήριο την υπέρβαση ή όχι ενός ορίου της τιμής κάποιου στοιχείου του διεκπεραιωτή. Το στοιχείο αυτό συνήθως είναι το φορτίο με δύο οριακές τιμές, μία ανώτερη T_{high} και μία κατώτερη T_{low}. Για τιμές του φορτίου μεγαλύτερες της T_{high} ο διεκπεραιωτής αποστέλλει εργασία, ενώ για μικρότερες του T_{low} λαμβάνει. Εναλλακτικά των πολιτικών μεταφοράς κατωφλίου συναντώνται οι σχετικές πολιτικές μεταφοράς (*relative transfer policies*) όπου λαμβάνονται υπόψη η κατάσταση των διεκπεραιωτών (π.χ. το φορτίο) σε σύγκριση με τους υπόλοιπους.

- Η *πολιτική επιλογής* (*Selection policy*) βάσει της οποίας πραγματοποιείται η εκλογή της εργασίας προς μεταφορά. Αυτό συμβαίνει όταν ο διεκπεραιωτής χαρακτηριστεί από την πολιτική μεταφοράς, ως αποστολέας. Μία συνήθης υλοποίηση αποτελεί η επιλογή εκείνης της εργασίας που προκάλεσε τον εν λόγω χαρακτηρισμό του διεκπεραιωτή. Βέβαια μπορεί να ληφθούν υπόψη και άλλοι παράγοντες όπως η πρόοδος εκτέλεσης της εργασίας, δηλαδή αν έχει υποστεί μερική επεξεργασία ή όχι. Στην πρώτη περίπτωση (*preemptive task transfer*) η διαδικασία μεταφοράς είναι ιδιαίτερα δαπανηρή καθώς απαιτείται λεπτομερής καταγραφή της κατάστασης της εργασίας, σε αντίθεση με την δεύτερη (*nonpreemptive task transfer*). Για τον λόγο αυτό, το πλήθος των εργασιών που διεκπεραιώνονται σε μονάδες διαφορετικές από αυτές όπου εκχωρήθηκαν αρχικά θα πρέπει να είναι περιορισμένο *[Eager, 88]*. Παρόλα αυτά, αποτελέσματα άλλων ερευνών *[Harchol, 96]*, *[Krueger, 88]* έδειξαν ότι η προώθηση τέτοιων

εργασιών μπορεί να βοηθήσει σε μεγαλύτερο βαθμό στην αύξηση της αποτελεσματικότητας του συστήματος, ακόμα και στην περίπτωση όπου το κόστος μνήμης είναι υψηλό.

➢ Η *πολιτική τοποθεσίας* (*Location policy*) είναι υπεύθυνη για την εύρεση του κατάλληλου «συνεργάτη» (*transfer partner*) για την λήψη ή αποστολή εργασίας.

➢ Η *πολιτική πληροφόρησης* (*Information policy*) η οποία αποφασίζει τον χρόνο, τον τόπο και το είδος της πληροφορίας του συστήματος που πρέπει να αποθηκευτεί με σκοπό την χρήση της για την λήψη των αποφάσεων του αλγόριθμου. Σύμφωνα με τους *[Kumar, 87]*, *[Shivaratri, 92]* οι πολιτικές πληροφόρησης διακρίνονται στις ακόλουθες κατηγορίες:
i) *πολιτικές καθοδηγούμενες βάσει απαιτήσεως* (*Demand driven policies*). Κατά τις αποκεντρωτικές αυτές πολιτικές, μια μονάδα συλλέγει πληροφορίες για τις υπόλοιπες στην περίπτωση όπου επιθυμεί να γίνει παραλήπτης ή αποστολέας μίας εργασίας.

ii) *Περιοδικές πολιτικές* (*periodic policies*), οι οποίες μπορεί να είναι κεντρικές ή αποκεντρωτικές, και καταγράφουν περιοδικά την κατάσταση του συστήματος. Συνήθως, οι πολιτικές αυτές δεν προσαρμόζουν τον ρυθμό ενημέρωσης της λίστας που διατηρούν τις πληροφορίες, με την δραστηριότητα του υπολογιστικού πλέγματος. Αυτό έχει σαν άμεσο αποτέλεσμα την επιπρόσθετη επιβάρυνση του συστήματος ή την εμφάνιση του προβλήματος της παρωχημένης πληροφορίας για έναν υψηλό ή χαμηλό ρυθμό ενημέρωσης αντίστοιχα. Το φαινόμενο αυτό μπορεί να αντιμετωπιστεί μερικώς με την δυναμική ρύθμιση του ρυθμού *[Zhou, 88]*.

iii) *Πολιτικές καθοδηγούμενες βάσει μεταβολής καταστάσεως* (*State change driven policies*), όπου οι οντότητες ενημερώνουν τους ενδιαφερόμενους για την κατάσταση τους όταν αυτή μεταβληθεί σε κάποιο βαθμό *[Shin, 89]*. Η αύξηση της αποδοτικότητας ενός συστήματος με την εφαρμογή των πολιτικών αυτών είναι περίπου της ίδιας τάξεως με αυτή των περιοδικών. Επιπλέον, παρουσιάζουν παρόμοια προβλήματα αφού ένα σύστημα με υψηλό ή χαμηλό ρυθμό μεταβολής του μεγέθους βάσει του οποίου λαμβάνει χώρα η έγερση της πολιτικής αυτής, οδηγεί στην συμφόρηση του συστήματος ή στο πρόβλημα της παρωχημένης πληροφορίας αντίστοιχα. Ενδεχόμενη λύση αποτελεί ο συνδυασμός της εν λόγω πολιτικής με περιοδική χαμηλού ρυθμού ενημέρωσης. *[Zhou, 87]*, *[Zhou, 88]*, *[Kremien, 92]*

Κατά την υλοποίηση μίας πολιτικής πληροφόρησης, εκτός από τον τρόπο έγερσης του μηχανισμού της, λαμβάνεται επιπλέον υπόψη α) το περιεχόμενο της πληροφορίας που μεταφέρεται. Συγκεκριμένα, πρέπει να καθοριστεί αν το μήνυμα ενημέρωσης περιέχει στοιχεία που αφορούν μόνο την οντότητα (*άμεση πληροφορία - direct information*) από την οποία αποστέλλεται ή περιλαμβάνει πληροφορίες και για άλλες μονάδες του συστήματος (*έμμεση πληροφορία -indirect information*). Στην περίπτωση ενθυλάκωσης έμμεσης πληροφορίας, γεννιέται η επιπρόσθετη ανάγκη προσδιορισμού των μονάδων τα στοιχεία των οποίων θα μεταφερθούν. *[Barak, 85]*, *[Stankovic, 84]* και β) οι παραλήπτες του μηνύματος ενημέρωσης, όπου μπορεί να αποτελούν όλες τις οντότητες του συστήματος (*broadcast*) *[Zhou, 87]* ή υποσύνολο αυτών (*multicast*) *[Theimer, 89]*.

1.2.3 Πρότυπα παραδείγματα

Ακολούθως περιγράφονται ορισμένοι βασικοί τύποι αλγόριθμων εξισορρόπησης φορτίου οι οποίοι και αποτελούν πρότυπα υλοποίησης άλλων. Οι περισσότεροι από αυτούς δεν στηρίζονται σε αυστηρά μαθηματικά μοντέλα αλλά στην διαίσθηση του ερευνητή ότι η εφαρμογή τους μπορεί να βελτιώσει την απόδοση του συστήματος (*Heuristics algorithms*).

1.2.3.1 Αλγόριθμοι αρχικοποίησης αποστολέα (Sender initiated algorithms)

Στην κατηγορία αυτή ανήκουν οι αλγόριθμοι όπου η ενεργοποίηση τους πραγματοποιείται όταν ένας υπέρφορτος (ο *αποστολέας*) διεκπεραιωτής προσπαθεί να αποστείλει μία εργασία σε κάποιον λιγότερο απασχολημένο (ο *παραλήπτης*). Παραδείγματα αποτελούν οι εργασίες *[Bryant, 81]*, *[Ramamritham, 84]*, *[Rotithor, 90]*.

Οι *Eager*, *Lazowska* και *Zahorjan [Eager, 86b]* μελέτησαν τρεις απλούς αλλά αποδοτικούς αλγόριθμους που ανήκουν στην κατηγορία αυτή και έχουν τα εξής χαρακτηριστικά:

➢ Ακολουθούν ενιαία πολιτική μεταφοράς με την μορφή πολιτικής κατωφλίου. Το μέγεθος που ευθύνεται για τον χαρακτηρισμό μίας μονάδας ως *αποστολέα* ή *παραλήπτη* είναι ο αριθμός των προς διεκπεραίωση εργασιών στην ουρά αναμονής του. Στην περίπτωση όπου η είσοδος μίας εργασίας προκαλέσει την αύξηση του μεγέθους αυτού πάνω από ένα όριο *T* τότε η μονάδα χαρακτηρίζεται ως αποστολέας. Αντιθέτως, μια μονάδα χαρακτηρίζεται ως *παραλήπτης* όταν υπάρχει η δυνατότητα λήψης εργασίας χωρίς την υπέρβαση του παραπάνω ορίου.

➢ Κοινή πολιτική επιλογής. Επιλέγονται μόνο οι πρόσφατες εργασίες.

➢ Διαφέρουν στις πολιτικές τοποθεσίας οι οποίες περιγράφονται ακολούθως:

i) *Πολιτική τυχαίας τοποθεσίας* (*Random location policy*)**.** Η μεταφορά της εργασίας πραγματοποιείται σε τυχαίο διεκπεραιωτή χωρίς την ανταλλαγή μηνυμάτων καταστάσεως μεταξύ των μονάδων. Ο παραλήπτης είτε αναλαμβάνει την επεξεργασία της, είτε την προωθεί ξανά σε κάποια άλλη μονάδα στην περίπτωση όπου είναι υπερφορτωμένος (το μέγεθος της ουράς των εργασιών είναι μεγαλύτερο ενός ορίου *T*) κ.ο.κ. Το μοντέλο αυτό μπορεί να προκαλέσει την αστάθεια του συστήματος καθώς οι οντότητες, στην περίπτωση αδυναμίας εύρεσης κατάλληλου διεκπεραιωτή, είναι δυνατό να αναλώνονται στην ανταλλαγή μηνυμάτων και όχι στην ολοκλήρωση εργασιών. Το φαινόμενο αυτό είναι δυνατόν να αντιμετωπιστεί με την εφαρμογή ορίου στο πλήθος μεταφορών της εργασίας.

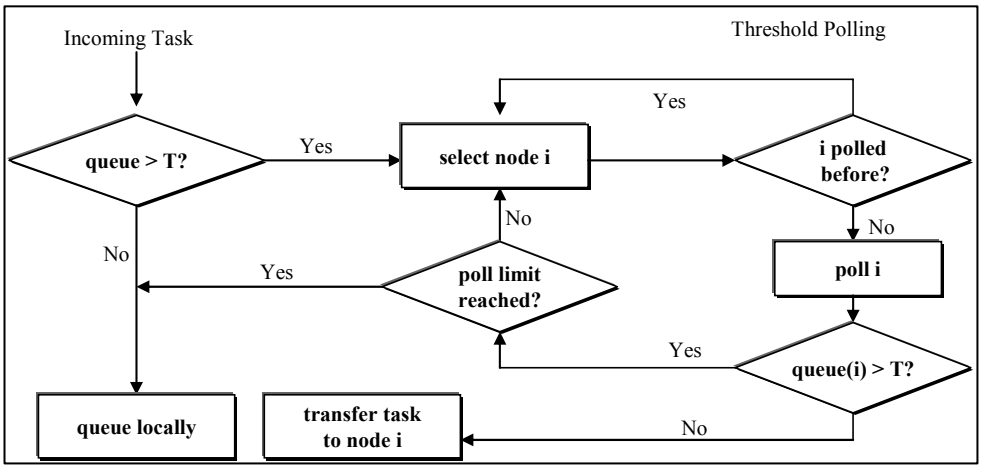
ii) *Πολιτική τοποθεσίας κατωφλίου (Threshold location policy)*. Η πολιτική αυτή έχει σαν στόχο την αποφυγή των περιττών μεταφορών που συμβαίνουν με την εφαρμογή της πολιτικής τυχαίας τοποθεσίας, και επιτυγχάνεται με κόστος την ανταλλαγή μηνυμάτων για την ανάκτηση της πληροφορίας των υποψήφιων παραληπτών. Συγκεκριμένα, επιλέγεται τυχαία μία μονάδα η οποία και πραγματοποιεί έλεγχο αν η αποδοχή της εργασίας θα προκαλέσει υπέρβαση του ορίου του μεγέθους της ουράς της. Στην περίπτωση θετικής εκτίμησης, η διαδικασία επαναλαμβάνεται με άλλον τυχαία επιλεγμένο διεκπεραιωτή. Διαφορετικά, η μονάδα δεσμεύεται για την εκτέλεση της εργασίας ανεξάρτητα της κατάστασης που θα βρίσκεται με την λήψη της. Εάν δεν εντοπιστεί κατάλληλη μονάδα μετά από ορισμένο πλήθος προσπαθειών (*poll limit*), τότε η μονάδα στην οποία αρχικά ανατέθηκε η εργασία αναλαμβάνει την ολοκλήρωση της.

iii) *Συντομότερη πολιτική τοποθεσίας (Shortest location policy)*. Πολιτική που χρησιμοποιεί περισσότερη πληροφορία σε σύγκριση με την προηγούμενη, για την λήψη αποφάσεων. Συγκεκριμένα, επιλέγονται τυχαία σταθερός αριθμός μονάδων από όπου και λαμβάνονται τα μεγέθη των ουρών τους. Η μονάδα με την μικρότερη ουρά και εφόσον το μέγεθος της δεν υπερβαίνει ένα όριο *T*, θεωρείται κατάλληλη για την μεταφορά. Η εργασία ολοκληρώνεται σε αυτόν τον διεκπεραιωτή ανεξάρτητα της κατάστασης που θα βρίσκεται με την λήψης της. Σε αντίθετη περίπτωση, δηλαδή το μέγεθος της μικρότερης ουράς υπερβαίνει το όριο *T*, η εργασία διεκπεραιώνεται από την αρχική μονάδα από όπου προήλθε.

Η αύξηση της αποδοτικότητας του συστήματος με χρήση της τελευταίας πολιτικής, σε σύγκριση με την πολιτική τοποθεσίας κατωφλίου, είναι οριακή. Το γεγονός αυτό αποδεικνύει ότι η λήψη αποφάσεων βάσει λεπτομερών πληροφοριών του συστήματος δεν οδηγεί πάντοτε σε ανάλογη αύξηση της αποτελεσματικότητας του.

Οι αλγόριθμοι αρχικοποίησης αποστολέα δεν ανταποκρίνονται σε συνθήκες υψηλού φορτίου, αφού η πιθανότητα εύρεσης μονάδας με χαμηλό φορτίο (παραλήπτη), προς την μεταφορά της εργασίας, δεν είναι μεγάλη.

Στο Σχήμα 1 απεικονίζεται η λειτουργία του αλγόριθμου αρχικοποίησης αποστολέα με εφαρμογή της πολιτικής τοποθεσίας κατωφλίου.



Σχήμα 1 - *Αλγόριθμος αρχικοποίησης αποστολέα με εφαρμογή της πολιτικής κατωφλίου.*

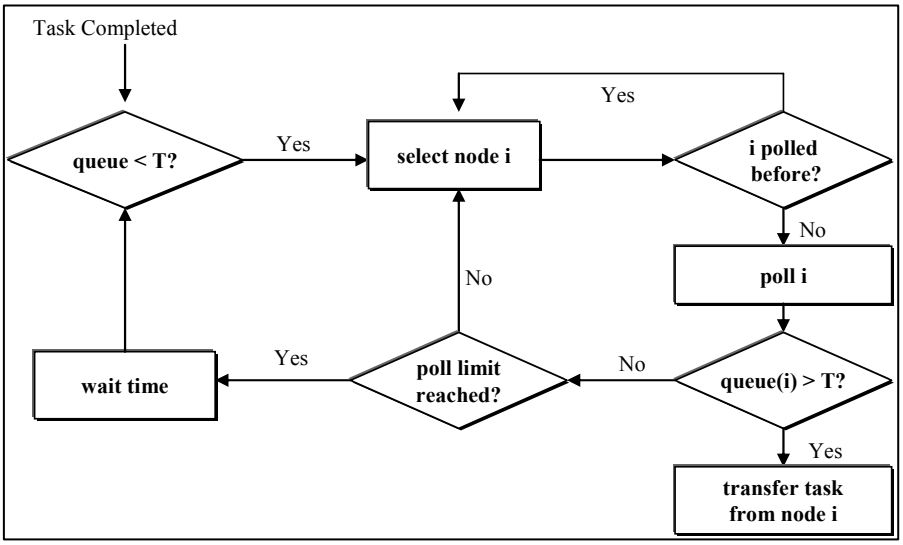
1.2.3.2 Αλγόριθμοι αρχικοποίησης παραλήπτη (Receiver initiated algorithms)

Στην κατηγορία αυτή ανήκουν οι αλγόριθμοι όπου η ενεργοποίηση τους πραγματοποιείται όταν μία μονάδα με χαμηλό φορτίο (*παραλήπτης*) προσπαθεί να λάβει εργασία από κάποια με υψηλότερο φορτίο (*αποστολέας*). Ομοίως με τους αλγόριθμους αρχικοποίησης αποστολέα, ο χαρακτηρισμός μίας μονάδας ως παραλήπτη ή αποστολέα βασίζεται στο μέγεθος της ουράς αναμονής των εργασιών (αν υπολείπεται η υπερβαίνει ενός ορίου T). Ένας τέτοιος αλγόριθμος μελετήθηκε από τους *Livny* και *Melman [Livny, 82]* καθώς και από τους *Eager*, *Lazowska* και *Zahorjan [Eager, 86a]*, όπου και ακολουθεί σύντομη περιγραφή του.

Κάθε διεκπεραιωτής με την ολοκλήρωση μίας εργασίας πραγματοποιεί έλεγχο του μεγέθους της ουράς αναμονής. Στην περίπτωση που το μέγεθος αυτό είναι μικρότερο από ένα όριο *T*, ο διεκπεραιωτής χαρακτηρίζεται ως παραλήπτης και εγείρεται η πολιτική τοποθεσίας για την εύρεση κατάλληλου αποστολέα. Κατά την πολιτική αυτή επιλέγεται τυχαία μονάδα η οποία και εξετάζει την ύπαρξη εργασίας όπου ενδεχόμενη προώθηση της στον παραλήπτη θα προκαλούσε την μείωση του μεγέθους της ουράς της κάτω από ένα όριο *T*. Εάν αυτό είναι αδύνατο, ο διεκπεραιωτής αποστέλλει την εργασία (με χρήση οποιασδήποτε πολιτικής επιλογής) στον παραλήπτη. Διαφορετικά, επαναλαμβάνεται η πολιτική τοποθεσίας. Η διαδικασία τερματίζεται είτε με την εύρεση αποστολέα, είτε με την αδυναμία εντοπισμού κατάλληλης μονάδας μετά το πέρας ορισμένου πλήθους προσπαθειών (*poll limit*).

Οι αλγόριθμοι αυτοί είναι αποδοτικοί σε συστήματα υψηλών φορτίων αφού υπάρχει μεγάλη πιθανότητα εύρεσης αποστολέα με μικρό αριθμό αναζητήσεων (*probing*).

Στο Σχήμα 2 απεικονίζεται η λειτουργία του αλγόριθμου αρχικοποίησης παραλήπτη.

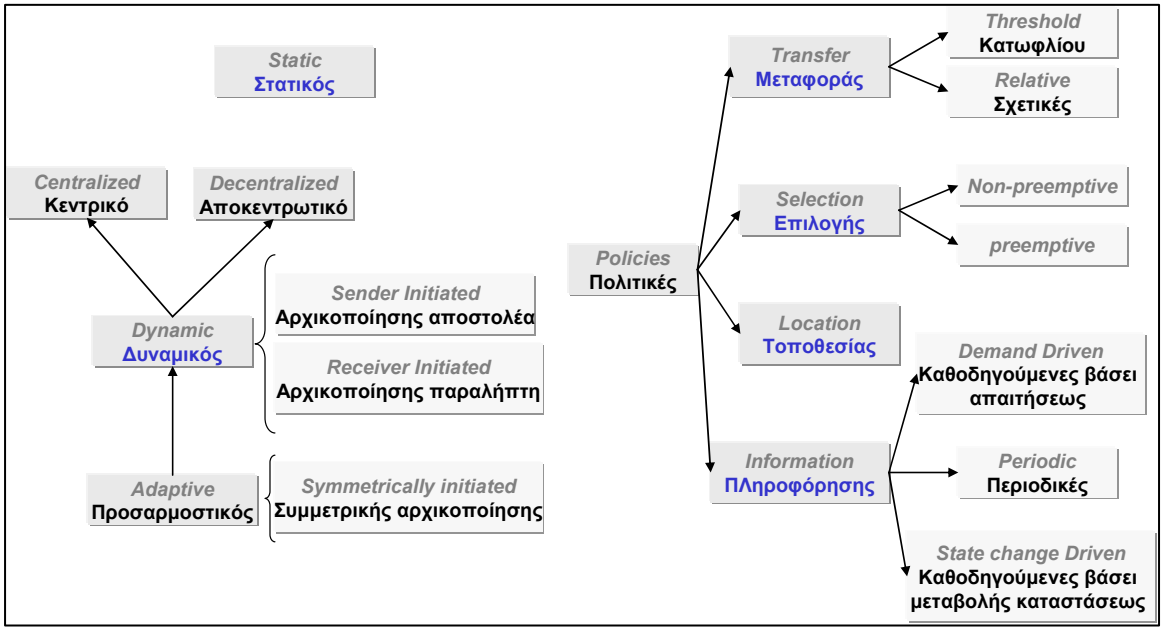


Σχήμα 2 - *Αλγόριθμος αρχικοποίησης παραλήπτη.*

1.2.3.3 Αλγόριθμοι συμμετρικής αρχικοποίησης (Symmetrically initiated algorithms)

Στην κατηγορία αυτή ανήκουν οι αλγόριθμοι όπου ο μηχανισμός εξισορρόπησης φορτίου ενεργοποιείται και από τα δύο είδη μονάδων, παραλήπτης ή αποστολέας. Προκύπτουν από τον συνδυασμό των πολιτικών μεταφοράς και τοποθεσίας των αλγόριθμων αρχικοποίησης παραλήπτη και αποστολέα, με αποτέλεσμα να παρουσιάζουν τα πλεονεκτήματα και τα μειονεκτήματα τους.

Στο Σχήμα 3 απεικονίζεται η εν λόγω κατάταξη των αλγορίθμων εξισορρόπησης φορτίου.



Σχήμα 3 – Κατάταξη αλγόριθμων εξισορρόπησης φορτίου.

Κεφάλαιο 2

Πειραματικό Περιβάλλον

2 Πειραματικό περιβάλλον

Στο παρόν κεφάλαιο περιγράφεται η υλοποίηση ενός ευέλικτου συστήματος (καλούμενο και ως πειραματικό περιβάλλον, experimental framework) προσομοίωσης διακριτών γεγονότων για την μελέτη απλών αλγόριθμων εξισορρόπησης φορτίου. Αναλυτικότερα στο κεφάλαιο αυτό παρουσιάζονται, η μοντελοποίηση των βασικών οντοτήτων (υπολογιστική μονάδα, φορτίο) και οι θεμελιώδεις κλάσεις του πειραματικού περιβάλλοντος, ο αλγόριθμος για την προσομοίωση της κεντρικής μονάδας επεξεργασίας, καθώς επίσης το έγγραφο μέσω του οποίου καθορίζεται το μοντέλο του συστήματος που επιθυμείται να προσομοιωθεί. Στο τέλος του κεφαλαίου, παρατίθεται υποδειγματικό παράδειγμα κατάστρωσης και εκτέλεσης πειράματος.

Στην αρχή του κεφαλαίου, για την καλύτερη κατανόηση του εν λόγω πειραματικού περιβάλλοντος και γενικότερα των πλατφορμών προσομοίωσης διακριτών γεγονότων, με τις οποίες θα ασχοληθούμε στο κεφάλαιο 4, πραγματοποιείται αναφορά για την μοντελοποίηση και την προσομοίωση συστημάτων.

2.1 Συστήματα και Μοντέλα

Ένα σύστημα ορίζεται ως ένα σύνολο *οντοτήτων* (π.χ. άνθρωποι ή μηχανές) που ενεργούν και αλληλεπιδρούν με στόχο κάποιο λογικό τερματισμό. Στην πράξη, η έννοια του συστήματος εξαρτάται από τους στόχους της συγκεκριμένης μελέτης. Το σύνολο των οντοτήτων που συγκροτούν ένα σύστημα για μία μελέτη, μπορεί να αποτελεί υποσύνολο του συνολικού συστήματος για μία άλλη.

Ως *κατάσταση ενός συστήματος* (*system state*) ορίζεται το σύνολο των μεταβλητών (*μεταβλητές κατάστασης*) που είναι απαραίτητες για την περιγραφή του συστήματος σε μια χρονική στιγμή.

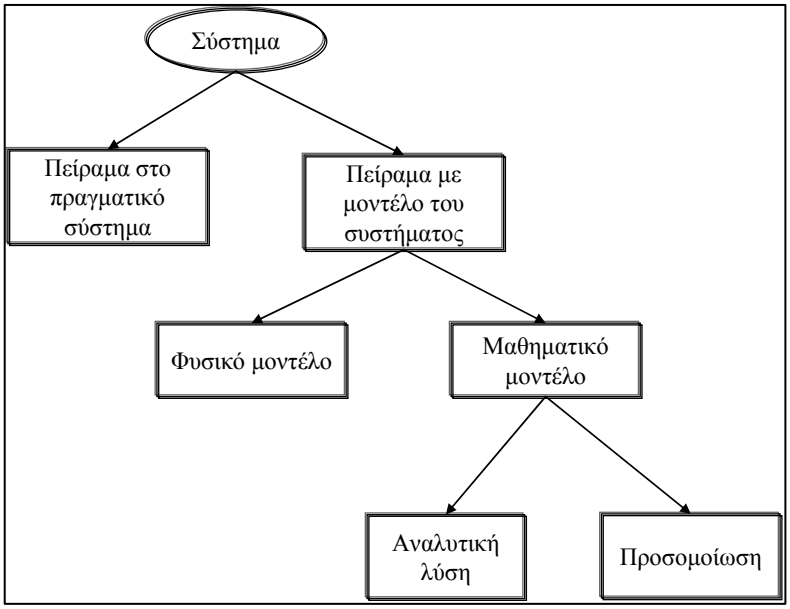
Τα συστήματα μπορούν να είναι *διακριτά* (*discrete*) ή *συνεχή* (*continuous*). Διακριτό σύστημα είναι αυτό στο οποίο οι μεταβλητές κατάστασης μπορούν να αλλάξουν σε διακεκριμένα χρονικά σημεία, σε αντίθεση με το συνεχές όπου οι μεταβλητές κατάστασης μεταβάλλονται συνεχώς.

Η μελέτη ενός συστήματος προκύπτει συνήθως από την ανάγκη α) εξέτασης των σχέσεων μεταξύ των επιμέρους τμημάτων που το απαρτίζουν ή β) πρόβλεψης της απόδοσης του κάτω από νέες, πιθανόν, συνθήκες. Οι τρόποι με τους οποίους μπορεί να μελετηθεί ένα σύστημα, όπως χαρακτηριστικά απεικονίζονται και στο Σχήμα 1, είναι οι ακόλουθοι.

➢ *Πείραμα με το πραγματικό σύστημα ή πείραμα με μοντέλο του συστήματος*. Στην περίπτωση όπου η πρόσβαση στο φυσικό σύστημα είναι εφικτή, η μελέτη μπορεί να πραγματοποιηθεί άμεσα σε αυτό, εξασφαλίζοντας ταυτόχρονα και την ακρίβεια των πειραματικών αποτελεσμάτων. Αυτός ο τρόπος μελέτης συνήθως δεν είναι εφαρμόσιμος καθώς, είτε το κόστος είναι ιδιαίτερα υψηλό είτε προκαλεί προβλήματα στην κανονική λειτουργία του συστήματος (για παράδειγμα κατά την εξέταση οριακών συνθηκών). Αντιθέτως, στην περίπτωση όπου το σύστημα δεν υφίσταται (αφού μπορεί να βρίσκεται ακόμα στην φάση του σχεδιασμού) ή δεν είναι εφικτή η πραγματοποίηση άμεσου πειράματος σε αυτό, αναγκαία είναι η κατασκευή μοντέλου.

➢ *Φυσικό ή μαθηματικό μοντέλο*. Το μοντέλο ενός συστήματος μπορεί να έχει την μορφή ενός α) *φυσικού μοντέλου*, να αποτελεί δηλαδή μικρογραφία του υπό εξέταση συστήματος. Για παράδειγμα η μινιατούρα ενός πλοίου που δοκιμάζεται σε μια δεξαμενή για τη μελέτη των υδροδυναμικών ιδιοτήτων του πραγματικού συστήματος (πλοίου) και β) μαθηματικού μοντέλου, δηλαδή ένα σύνολο λογικών και ποσοτικών σχέσεων οι οποίες χειριζόμενες και μεταβαλλόμενες κατάλληλα, επιτρέπουν την πρόβλεψη της συμπεριφοράς του συστήματος, κάτω από διάφορες συνθήκες λειτουργίας. Ένα παράδειγμα μαθηματικού μοντέλου είναι η γνωστή σχέση $d = r \times t$, όπου *r* και *t* είναι η ταχύτητα και ο χρόνος ταξιδιού αντίστοιχα, και *d* είναι η απόσταση που έχει διανυθεί.

➢ *Αναλυτική λύση ή προσομοίωση*. Από τη στιγμή που έχει δημιουργηθεί ένα μαθηματικό μοντέλο θα πρέπει να αναζητηθεί ο τρόπος επίλυσης του. Εάν το μοντέλο είναι αρκετά απλό, η αναλυτική λύση είναι αυτή που επακριβώς περιγράφει την συμπεριφορά του αντίστοιχου συστήματος. Στο παράδειγμα της προηγούμενης παραγράφου, ο χρόνος *t* που απαιτείται για να διανυθεί απόσταση *d* με ταχύτητα *r*, δίνεται από την σχέση $t = d/r$. Το μοντέλο αυτό είναι πολύ απλό, πράγμα που δεν συμβαίνει στις περισσότερες περιπτώσεις, αφού η αναλυτική επίλυση μπορεί να είναι αρκετά πολύπλοκη (αν όχι ανέφικτη) και να απαιτεί μεγάλη υπολογιστική ισχύ. Έτσι, σε πολυσύνθετα συστήματα το μοντέλο μελετάται με τη χρήση προσομοίωσης, δηλαδή με την εκτέλεση αριθμητικών πειραμάτων όπου εξετάζεται η επίδραση της μεταβολής των εισόδων (δεδομένα) στις εξόδους (μέτρα απόδοσης) του συστήματος. *[Γαροφαλάκης, 99]*



Σχήμα 1 – *Μέθοδοι μελέτης συστήματος.*

2.2 Προσομοίωση

Στις παραγράφους που ακολουθούν αναφέρονται γενικές πληροφορίες για την μέθοδο της προσομοίωσης με έμφαση σε αυτή των «Διακριτών γεγονότων».

2.2.1 Τύποι μοντέλων προσομοίωσης

Τα μοντέλα προσομοίωσης μπορούν να ταξινομηθούν σε έναν ή περισσότερους από τους ακόλουθους τύπους.

➢ *Στατικά ή δυναμικά μοντέλα προσομοίωσης*. Στατικό μοντέλο προσομοίωσης καλείται αυτό που απεικονίζει ένα σύστημα α) σε μία δεδομένη χρονική στιγμή ή β) που είναι ανεξάρτητο του χρόνου. Αντίθετα, ένα δυναμικό μοντέλο προσομοίωσης περιγράφει ένα σύστημα όπως αυτό εξελίσσεται με την πάροδο του χρόνου.

➢ *Ντετερμινιστικά ή στοχαστικά μοντέλα προσομοίωσης*. Ντετερμινιστικό μοντέλο προσομοίωσης ονομάζεται αυτό που δεν περιέχει *πιθανοτικά* (δηλαδή «τυχαία») τμήματα. Για παράδειγμα, ένα πολύπλοκο σύστημα διαφορικών εξισώσεων που περιγράφει μία χημική αντίδραση, μπορεί να αποτελεί ένα τέτοιο μοντέλο. Σε ένα ντετερμινιστικό μοντέλο, με δεδομένο το σύνολο των ποσοτήτων και των σχέσεων εισόδου του, η έξοδος είναι καθορισμένη. Αντιθέτως, στοχαστικό μοντέλο προσομοίωσης καλείται αυτό που περιέχει τουλάχιστον ορισμένα τμήματα με «τυχαία» είσοδο. Τα περισσότερα υπολογιστικά συστήματα που βασίζονται στα συστήματα αναμονής (queuing systems), χρησιμοποιούν στοχαστικά μοντέλα προσομοίωσης.

➢ *Αυτο-οδηγούμενα ή ιχνο-οδηγούμενα μοντέλα προσομοίωσης*: Σε ένα αυτο-οδηγούμενο (*self-driven*) μοντέλο υπάρχει μία εσωτερική πηγή τυχαίων αριθμών. Οι τυχαίοι αυτοί αριθμοί κατευθύνουν τα τμήματα του μοντέλου, δηλαδή χρησιμοποιούνται για τον προσδιορισμό των χρονικών σημείων εμφάνισης των γεγονότων του συστήματος. Βασικό χαρακτηριστικό ενός αυτο-οδηγούμενου μοντέλου είναι ότι αποτελεί ένα αυτάρκες μοντέλο,

καθώς η λειτουργία του δεν απαιτεί εξωτερικές εισόδους (*inputs*). Αντίθετα, ένα ιχνο-οδηγούμενο (*trace-driven*) μοντέλο καθοδηγείται από ακολουθίες εισόδου που προέρχονται από δεδομένα (*trace data*) τα οποία προέκυψαν από τη λειτουργία ενός πραγματικού συστήματος. Τέτοια δεδομένα μπορούν να παραχθούν στα περισσότερα υπολογιστικά συστήματα που διαθέτουν ενσωματωμένα *προγράμματα ιχνηλάτησης* (*tracing programs*) τα οποία παρακολουθούν και καταγράφουν τις δραστηριότητες του συστήματος. Τα ιχνο-οδηγούμενα μοντέλα παρουσιάζουν πλεονεκτήματα όπως ότι α) αποφεύγονται οι δυσκολίες της πιθανοτικής ανάλυσης που απαιτείται με τη χρήση κατανομών για την περιγραφή των εισόδων και β) είναι εύκολο να επιβεβαιωθούν. Το πρόβλημα με τα ιχνο-οδηγούμενα μοντέλα είναι το μικρό εύρος εφαρμογών που μπορούν να αντιμετωπίσουν. Οι εφαρμογές αυτές πρακτικά περιορίζονται για την μελέτη μετατροπών σε υπάρχοντα υπολογιστικά συστήματα.

➢ *Συνεχή ή διακριτά μοντέλα προσομοίωσης*: Οι ορισμοί των συνεχών και διακριτών μοντέλων προσομοίωσης, είναι ανάλογοι με τους ορισμούς των συνεχών και διακριτών συστημάτων που αναφέρθηκαν στην παράγραφο 2.1. Πρέπει να σημειωθεί ότι ένα διακριτό μοντέλο δεν χρησιμοποιείται μόνο για την αναπαράσταση ενός διακριτού συστήματος και ένα διακριτό σύστημα δεν αναπαριστάται μόνο από ένα διακριτό μοντέλο προσομοίωσης. Η απόφαση για τη χρήση ενός διακριτού ή ενός συνεχούς μοντέλου για ένα συγκεκριμένο σύστημα, εξαρτάται από τους ιδιαίτερους στόχους της μελέτης. Για παράδειγμα, ένα μοντέλο της ροής πακέτων δεδομένων σε ένα WAN, θα είναι διακριτό εάν το σημείο ενδιαφέροντος είναι τα χαρακτηριστικά και η κίνηση των επιμέρους πακέτων και κατά συνέπεια των επιμέρους χρηστών. Αντίθετα, αν μας ενδιαφέρει μόνο η συνολική κίνηση, η ροή των πακέτων θα μπορούσε ίσως να περιγραφεί με διαφορικές εξισώσεις σε ένα συνεχές μοντέλο. *[Γαροφαλάκης, 99]*

Στην παρούσα εργασία τόσο η προσομοίωση της μονάδας επεξεργασίας μίας υπολογιστικής μονάδας όσο και οι πλατφόρμες προσομοίωσης που θα μας απασχολήσουν αναφέρονται σε μοντέλα με κύρια συνιστώσα την προσομοίωση διακριτών γεγονότων (Discrete Event Simulation, DES). Για τον λόγο αυτό, στις ακόλουθες παραγράφους περιγράφονται τα ιδιαίτερα χαρακτηριστικά αυτού του τύπου προσομοίωσης.

2.2.2 Προσομοίωση διακριτών γεγονότων (Discrete event simulation) - Μηχανισμός εξέλιξης του χρόνου

Γενικά οι χρόνοι που εμπλέκονται σε ένα σύστημα προσομοίωσης είναι α) ο *φυσικός χρόνος* (*physical time*) που αντιστοιχεί στο χρόνο του φυσικού συστήματος, β) ο *χρόνος προσομοίωσης* (*simulation time*), ο οποίος χρησιμοποιείται για την μοντελοποίηση του φυσικού χρόνου, και γ) ο *χρόνος εκτέλεσης* (*wallclock time*) που αναφέρεται στην εκτέλεση της προσομοίωσης. *[Fujimoto, 00]*

Ο δυναμικός χαρακτήρας των μοντέλων προσομοίωσης διακριτών γεγονότων επιβάλλει την δυνατότητα καταγραφής της τρέχουσας τιμής του χρόνου προσομοίωσης, καθώς επίσης και την υλοποίηση μηχανισμού αύξησής του χρόνου αυτού από μία τιμή σε μία άλλη. Η μεταβλητή του μοντέλου προσομοίωσης που φιλοξενεί τον χρόνο αυτό ονομάζεται *ρολόι προσομοίωσης* (*simulation clock*). Η μονάδα χρόνου που χρησιμοποιεί το ρολόι είναι συνήθως η ίδια με αυτή που χρησιμοποιούν οι παράμετροι εισόδου, ενώ κατά κανόνα ο χρόνος που καταγράφει το ρολόι με το χρόνο που απαιτείται για την εκτέλεση της προσομοίωσης στον υπολογιστή (wallclock time), δεν ταυτίζονται.

Ιστορικά έχουν επικρατήσει δύο βασικές μέθοδοι για την εξέλιξη του ρολογιού προσομοίωσης: α) η *εξέλιξη με βάση το χρόνο του επόμενου γεγονότος* (*Next-Event Time Advance, NETA*) και β) η *εξέλιξη σταθερής αύξησης του χρόνου* (F*ixed-Increment Time*

Advance, FITA). Η πρώτη μέθοδος είναι πιο διαδεδομένη και επιπλέον η δεύτερη μπορεί να θεωρηθεί ως ειδική περίπτωση της πρώτης.

Η μέθοδος NETA λαμβάνει υπόψη ότι, κατά την DES οι αλλαγές της κατάστασης του συστήματος πραγματοποιούνται στις χρονικές στιγμές εμφάνισης (έγερσης) των γεγονότων. Έτσι κατά την μέθοδο αυτή, το ρολόι προσομοίωσης αρχικοποιείται στο μηδέν, και αφού καθοριστούν οι χρόνοι έγερσης των μελλοντικών γεγονότων, μεταβαίνει στο μικρότερο (πιο κοντινού στο μέλλον) χρόνο έγερσης μεταξύ των γεγονότων αυτών (δηλαδή οι ανενεργές ενδιάμεσες περίοδοι δεν συνυπολογίζονται). Στην συνέχεια, η κατάσταση του συστήματος ενημερώνεται λαμβάνοντας υπόψη το γεγονός που εμφανίστηκε, επανακαθορίζονται οι χρόνοι έγερσης των υπόλοιπων γεγονότων και το ρολόι μεταβαίνει ξανά στον μικρότερο χρόνο έγερσης κ.ο.κ. Η διαδικασία αυτή συνεχίζεται μέχρι την ικανοποίηση κάποιας προκαθορισμένης συνθήκης τερματισμού της προσομοίωσης.

Στην δεύτερη μέθοδο, το ρολόι προσομοίωσης εξελίσσεται με σταθερές αυξήσεις Δt μονάδων χρόνου κάθε φορά. Μετά από κάθε ενημέρωση του ρολογιού, πραγματοποιείται έλεγχος για να εξακριβωθεί εάν θα έπρεπε να έχουν εμφανισθεί κάποια γεγονότα κατά το προηγούμενο χρονικό διάστημα Δt . Αν εμφανίσθηκαν γεγονότα στο διάστημα αυτό, θεωρείται ότι αυτά λαμβάνουν χώρα στο τέλος του χρονικού διαστήματος και η κατάσταση του συστήματος ενημερώνεται κατάλληλα. *[Γαροφαλάκης, 99]*

2.2.3 Τύποι προσομοίωσης διακριτών γεγονότων

Η προσομοίωση διακριτών γεγονότων μπορεί να ταξινομηθεί σε δύο επιμέρους τύπους.

➢ Προσομοίωση με *προσέγγιση χρονοδρομολόγησης γεγονότων* (*Event scheduling/Driven Approach*, EDA). Στον τύπο αυτό, ως *γεγονός* ορίζεται η μεταβολή της κατάστασης του συστήματος. Η εξέλιξη του μοντέλου λαμβάνει χώρα με μια σειρά γεγονότων, και είναι δυνατόν να περιγραφεί από τους χρόνους που συμβαίνουν τα γεγονότα αυτά και τις μεταβολές που προκαλούν στο σύστημα. Σημειώνεται ότι, η εμφάνιση ενός γεγονότος και κατά συνέπεια η αντίστοιχη μεταβολή του συστήματος πραγματοποιείται ακαριαία, δηλαδή σε μηδενικό χρόνο προσομοίωσης.

Το κυριότερο στοιχείο αυτού του τύπου προσομοίωσης είναι η *λίστα γεγονότων* (*event list*) η οποία αποτελείται από ένα σύνολο γεγονότων (*events*) που λαμβάνουν χώρα κάποια χρονική στιγμή. Κάθε γεγονός περιγράφεται από ένα ζεύγος της μορφής (*τύπος γεγονότος, χρόνος γεγονότος*) -(*event type, event time*)-, όπου *τύπος γεγονότος* προσδιορίζει μια συγκεκριμένη μεταβολή της κατάστασης του συστήματος, και *χρόνος γεγονότος* είναι ο χρόνος προσομοίωσης που συμβαίνει η μεταβολή αυτή (ή εγείρεται το γεγονός αυτό).

Με την υπόθεση ότι η λίστα γεγονότων υλοποιείται ως ένα διατεταγμένο σύνολο βάσει των χρόνων παρουσίας των γεγονότων, οι βασικές λειτουργίες (με σειρά εκτέλεσης) του εν λόγω τύπου προσομοίωσης, όπως χαρακτηριστικά απεικονίζονται στο Σχήμα 2, μπορούν να περιγραφούν ως ακολούθως. Σημειώνεται ότι, οι συμβολισμοί E.type και E.time αναπαριστούν τον τύπο και τον χρόνο αντίστοιχα του γεγονότος E.

1. Αρχικοποίηση του ρολογιού προσομοίωσης με την μηδενική τιμή και τοποθέτηση των αρχικών γεγονότων στην λίστα με βάση τον χρόνο έγερσης τους.

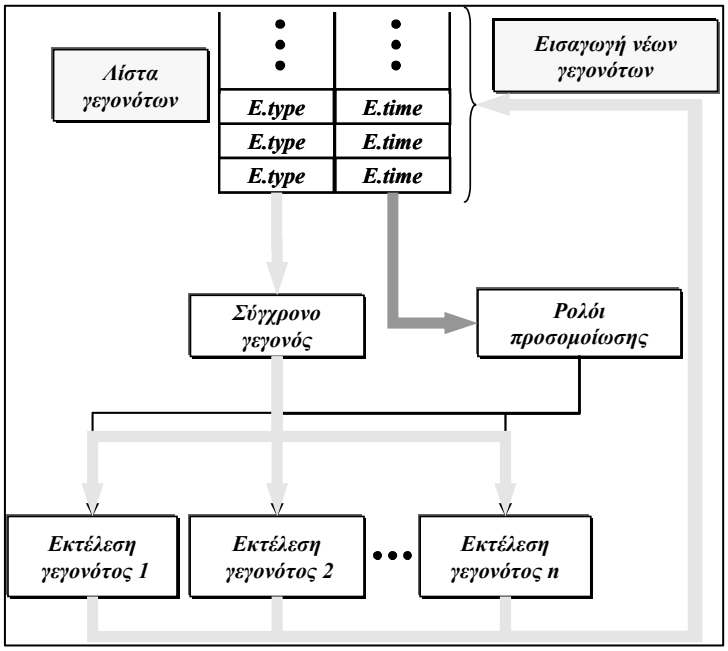
2. Ανάκτηση του πρώτου στοιχείου, E, της λίστας. Στην περίπτωση όπου η λίστα είναι κενή η προσομοίωση τερματίζεται.

3. Μετάβαση του χρόνου προσομοίωσης σε αυτόν, E.time, του γεγονότος E. Εάν ο χρόνος E.time είναι μεγαλύτερος του χρόνου προσομοίωσης, η προσομοίωση τερματίζεται.

4. Επιλογή της κατάλληλης ρουτίνας εκτέλεσης (καλούμενη και ως *ρουτίνα εξυπηρέτησης γεγονότος*, *event service routine*) βάσει του τύπου, E.type, του γεγονότος E.

5. Εκτέλεση της ρουτίνας που επιλέχτηκε. Κατά την εκτέλεση της ρουτίνας, η πληροφορία του συστήματος μπορεί να ανανεωθεί, νέα γεγονότα E' (με $E'.time \geq E.time$) μπορούν να παραχθούν, ενώ ο χρόνος προσομοίωσης παραμένει αμετάβλητος.

6. Μετά την εκτέλεση της ρουτίνας ο έλεγχος μεταβαίνει ξανά στην λειτουργία 2. *[Sinclair, 04]*



Σχήμα 2 – *Προσέγγιση χρονοδρομολόγησης γεγονότων.*

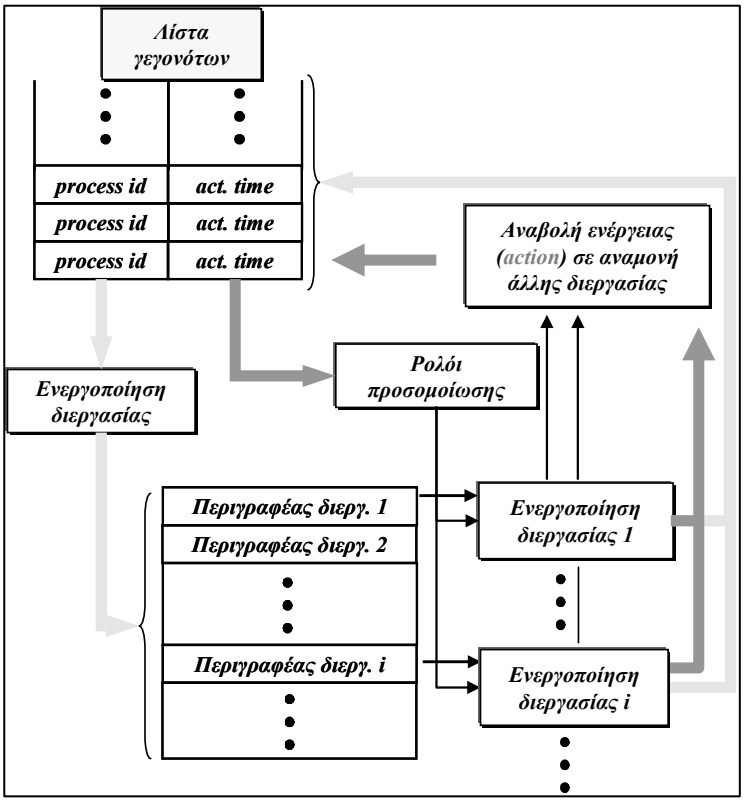
➢ Προσομοίωση με την *προσέγγιση διαδικασίας/διεργασίας* (*Process Approach/oriented*, *PA*),

Δεν είναι σφάλμα να ισχυριστούμε ότι όλες οι προσομοιώσεις διακριτών γεγονότων υλοποιούνται βάσει της προσέγγισης χρονοδρομολόγησης γεγονότων, καθώς η προσομοίωση με την προσέγγιση διεργασίας ουσιαστικά καλύπτει τον βασικό μηχανισμό του πρώτου τύπου με ένα υψηλότερο επίπεδο χαρακτηριστικών και λειτουργιών.

Στην προσομοίωση με την προσέγγιση διεργασίας, ένα σύνολο συσχετιζόμενων τύπων γεγονότων ομαδοποιούνται μέσα σε μία *διεργασία*, όπως αυτή εννοιολογικά ορίζεται σε ένα λειτουργικό σύστημα. Έτσι, μια διεργασία α) αποτελείται από το σώμα του κώδικα, τους πόρους που δεσμεύονται για την εκτέλεσή της (κυρίως μνήμη) και την κατάσταση της (το τρέχον σημείο εκτέλεσης του κώδικα και τις τιμές των μεταβλητών που χρησιμοποιούνται), β) συνήθως, έχει μη-μηδενική διάρκεια ζωής (σε αντίθεση με το γεγονός που λαμβάνει χώρα ακαριαία) καθώς μπορεί να είναι ενεργή σε όλο τον χρόνο της προσομοίωσης και γ) η εκτέλεσή της μπορεί να ανασταλεί, είτε για ορισμένο χρονικό διάστημα είτε έως ότου αλληλεπιδράσει με κάποια άλλη διεργασία.

Κατά την δημιουργία αυτού του είδους προσομοιώσεων, εκτός από την διαχείριση της λίστας γεγονότων, θα πρέπει να ληφθούν υπόψη επιπλέον λειτουργίες όπως είναι α) η δημιουργία, η ενεργοποίηση και η χρονοδρομολόγηση διεργασιών, β) η αναστολή εκτέλεσης διεργασιών, γ) ο τερματισμός διεργασιών και δ) ο συγχρονισμός και επικοινωνία μεταξύ των διεργασιών.

Η γενική δομή μίας προσομοίωσης με την προσέγγιση διαδικασίας απεικονίζεται στο Σχήμα 3. Στην περίπτωση αυτή, η λίστα γεγονότων έχει απλοποιηθεί καθώς υφίσταται μόνο ένας τύπος γεγονότος, αυτός που υποδεικνύει την έναρξη ή την συνέχιση (μετά από αναστολή) μιας διεργασίας. Το στοιχείο E.type αντικαθίσταται από το στοιχείο process.id, το οποίο αντιπροσωπεύει τον *κωδικό του περιγραφέα* (*descriptor id*) (η ακόμα και τον ίδιο τον περιγραφέα) μιας διεργασίας. Ο περιγραφέας χρησιμοποιείται για την εύρεση και την ανάκτηση του περιβάλλοντος εκτέλεσης της αντίστοιχης διεργασίας (βάσει των πληροφοριών που φιλοξενούνται σε αυτόν), προκειμένου να ξεκινήσει ή να συνεχιστεί η εκτέλεση της, όταν αυτό υποδειχθεί από την εμφάνιση του κατάλληλου γεγονότος.



Σχήμα 3 – *Προσέγγιση Διεργασίας.*

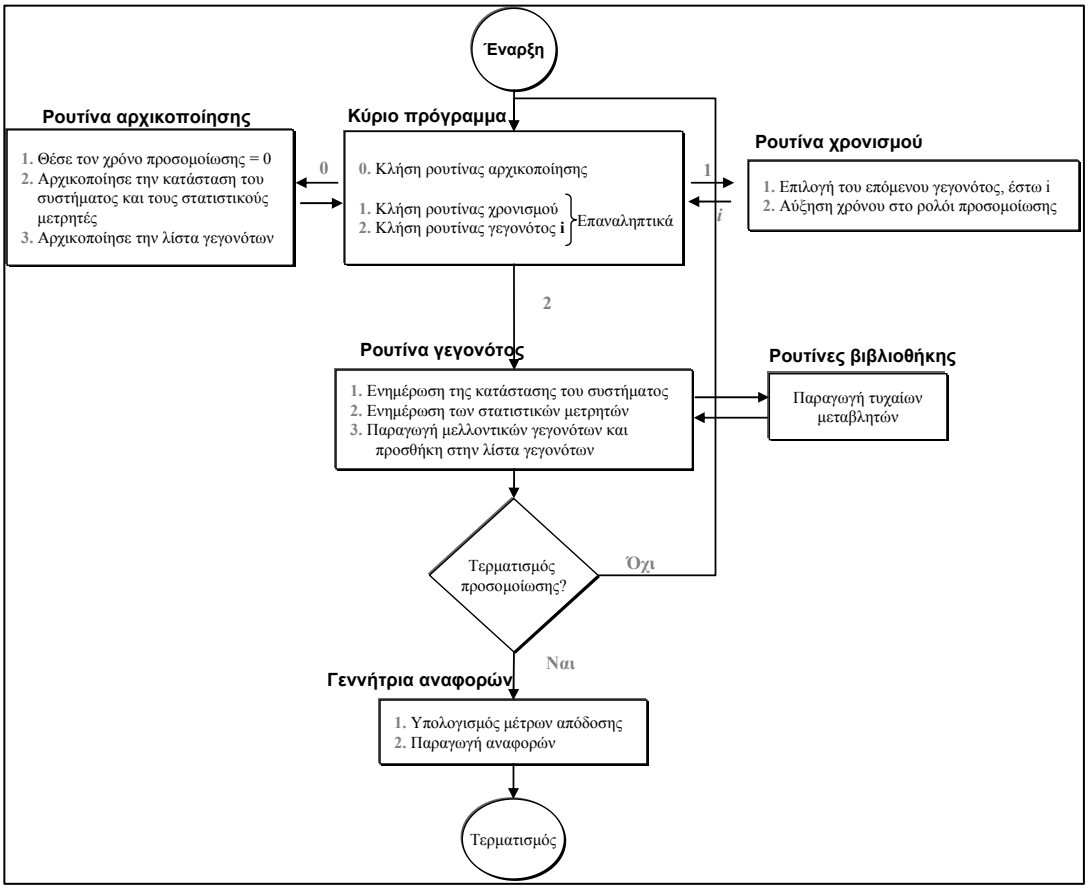
Όπως απεικονίζεται και στο Σχήμα 3, μία διεργασία μπορεί να ανασταλεί με δύο τρόπους. Στο πρώτο τρόπο, η διεργασία απλώς καθυστερεί την εκτέλεση της για συγκεκριμένο διάστημα του χρόνου προσομοίωσης, οπότε και ο αντίστοιχος κωδικός του περιγραφέα επανεισάγεται στην λίστα γεγονότων. Στον δεύτερο, η διεργασία βρίσκεται σε αναμονή μέχρι την επαλήθευση μίας συνθήκης. Στην περίπτωση αυτή ο περιγραφεάς επανεισάγεται στην λίστα μόνο μέσω της δράσης μίας άλλης διεργασίας (επαλήθευση της συνθήκης πάνω στην οποία περιμένει η διεργασία). *[Sinclair, 04]*

2.2.4 Συστατικά και οργάνωση ενός μοντέλου προσομοίωσης διακριτών γεγονότων

Τα περισσότερα μοντέλα προσομοίωσης διακριτών γεγονότων που χρησιμοποιούν τη μέθοδο εξέλιξης με βάση το χρόνο του επόμενου γεγονότος, περιλαμβάνουν τα παρακάτω τμήματα:

- *Κατάσταση συστήματος* (*system state*). Είναι το σύνολο των μεταβλητών κατάστασης που είναι απαραίτητες για την περιγραφή του συστήματος σε μία χρονική στιγμή.
- *Ρολόι προσομοίωσης* (*simulation clock*). Αποτελεί τη μεταβλητή που περιέχει την τρέχουσα τιμή του χρόνου προσομοίωσης.
- *Λίστα γεγονότων* (*event list*). Καλείται η λίστα που περιέχει την επόμενη χρονική στιγμή εμφάνισης κάθε τύπου γεγονότος.
- *Μετρητές στατιστικών* (*statistical counters*). Ορίζονται οι μεταβλητές που χρησιμοποιούνται για την αποθήκευση στατιστικών μετρήσεων της απόδοσης του συστήματος.
- *Ρουτίνα αρχικοποίησης* (*initialization routine*). Είναι το υποπρόγραμμα που αρχικοποιεί το μοντέλο προσομοίωσης τη χρονική στιγμή μηδέν.
- *Ρουτίνα χρονισμού* (*timing routine*). Είναι το υποπρόγραμμα που αναγνωρίζει το επόμενο γεγονός από τη λίστα γεγονότων και ακολούθως αυξάνει το ρολόι προσομοίωσης στη χρονική στιγμή που το γεγονός αυτό θα εμφανισθεί.
- *Ρουτίνες γεγονότων* (*event routines*). Αποτελούν τα υποπρογράμματα που ενημερώνουν την κατάσταση συστήματος όταν εμφανίζεται ένα συγκεκριμένο είδος γεγονότος (υπάρχει μία τέτοια ρουτίνα για κάθε είδος γεγονότος).
- *Ρουτίνες βιβλιοθήκης* (*library routines*). Αναφέρονται σε ένα σύνολο υποπρογραμμάτων που δημιουργούν τυχαίες εμφανίσεις τιμών από πιθανοτικές κατανομές, που έχουν ορισθεί ως μέρος του μοντέλου προσομοίωσης.
- *Γεννήτρια αναφορών* (*report generator*). Είναι το υποπρόγραμμα που υπολογίζει εκτιμήσεις των επιθυμητών μέτρων απόδοσης από τους μετρητές στατιστικών και παράγει αναφορές όταν τελειώσει η εκτέλεση του προσομοιωτή.
- *Κυρίως πρόγραμμα* (*main program*). Ονομάζεται το πρόγραμμα που καλεί τη ρουτίνα χρονισμού για να καθοριστεί το επόμενο γεγονός και μεταφέρει τον έλεγχο στην αντίστοιχη ρουτίνα γεγονότος για να ενημερωθεί κατάλληλα η κατάσταση του συστήματος. Ελέγχει επίσης αν πρέπει να τερματισθεί η προσομοίωση και καλεί τότε τη γεννήτρια αναφορών. *[Γαροφαλάκης, 99]*

Οι λογικές σχέσεις ανάμεσα στα παραπάνω τμήματα φαίνονται στο Σχήμα 4.



Σχήμα 4 – *Προσομοίωση Διακριτών Γεγονότων, οργάνωση λειτουργιών.*

2.2.5 Πλεονεκτήματα, μειονεκτήματα και δυσκολίες της προσομοίωσης

Η προσομοίωση είναι μια ευρέως χρησιμοποιούμενη και συνεχώς πιο δημοφιλής μέθοδος για τη μελέτη πολύπλοκων συστημάτων. Έχει φυσικά τα πλεονεκτήματα και τα μειονεκτήματά της, αλλά όμως και πολλές αιτίες εξ αιτίας των οποίων ορισμένες προσομοιώσεις δεν καταλήγουν στο επιθυμητό αποτέλεσμα.

Ορισμένα πιθανά πλεονεκτήματα της χρήσης της μεθόδου της προσομοίωσης είναι τα παρακάτω:

➢ Τα περισσότερα σύνθετα συστήματα του πραγματικού κόσμου με "τυχαίες" παραμέτρους, δεν μπορούν να περιγραφούν ικανοποιητικά με κάποιο μαθηματικό μοντέλο που μπορεί να λυθεί αναλυτικά. Έτσι, η προσομοίωση είναι συχνά η μόνη διαθέσιμη μέθοδος μελέτης.

➢ Η προσομοίωση επιτρέπει την εκτίμηση της απόδοσης ενός υπάρχοντος συστήματος, κάτω από κάποιο προβλεπόμενο σύνολο λειτουργικών συνθηκών.

➢ Μέσω της προσομοίωσης, μπορούν να συγκριθούν εναλλακτικές προτεινόμενες σχεδιάσεις ή εναλλακτικές πολιτικές λειτουργίας του συστήματος, ώστε να προσδιορισθεί η βέλτιστη λύση που ικανοποιεί τις προδιαγραφές που έχουν ορισθεί.

➢ Σε ένα μοντέλο προσομοίωσης υπάρχει καλύτερος έλεγχος των πειραματικών συνθηκών, σε σύγκριση με την εκτέλεση πειραμάτων σε πραγματικό σύστημα .

➢ Η προσομοίωση επιτρέπει τη μελέτη ενός συστήματος που έχει μακρόχρονη εξέλιξη (π.χ. ένα οικονομικό σύστημα), σε πολύ μικρότερο χρόνο και αντίστροφα.

➢ Είναι μια μέθοδος οικονομική, αφού είναι δυνατό να υλοποιηθεί πλέον σε μικρούς υπολογιστές με τη χρήση γλωσσών προγραμματισμού γενικού σκοπού όπως η Java, C++, C κ.λ.π.

➢ Η προσομοίωση μπορεί να υλοποιηθεί από μηχανικούς που δεν είναι απαραίτητο να έχουν εκτεταμένες μαθηματικές γνώσεις, παρά μόνο τη δυνατότητα να κατανοούν βασικές έννοιες στατιστικής και να μπορούν να εφαρμόζουν ήδη έτοιμα μαθηματικά εργαλεία.

Ορισμένα μειονεκτήματα της προσομοίωσης είναι τα παρακάτω:
➢ Κάθε εκτέλεση ενός μοντέλου προσομοίωσης διακριτών γεγονότων παράγει μόνο εκτιμήσεις των πραγματικών χαρακτηριστικών του μοντέλου, για ένα συγκεκριμένο σύνολο παραμέτρων εισόδου. Έτσι, είναι πιθανό να χρειασθούν πολλές διαφορετικές ανεξάρτητες εκτελέσεις του μοντέλου για κάθε σύνολο παραμέτρων εισόδου που θα μελετηθεί. Για το λόγο αυτό, η προσομοίωση δεν είναι γενικά τόσο καλή μέθοδος για βελτιστοποίηση, όσο είναι για τη σύγκριση εναλλακτικών σχεδιαστικών λύσεων του συστήματος.

➢ Τα μοντέλα προσομοίωσης συχνά απαιτούν πολύ χρόνο και πόρους για να αναπτυχθούν.

➢ Ο μεγάλος όγκος αριθμών που παράγονται από μία μελέτη προσομοίωσης ή η εντύπωση που δημιουργούν οι τυχόν γραφικές αναπαραστάσεις των αποτελεσμάτων της, συχνά ενισχύουν μία τάση να δίνεται μεγαλύτερη εμπιστοσύνη στα αποτελέσματα αυτά από όσο πρέπει. Αν το μοντέλο δεν είναι μία αρκετά έγκυρη αναπαράσταση του συστήματος, τα αποτελέσματα της προσομοίωσης, ανεξάρτητα του πόσο εντυπωσιακά είναι, θα προσθέσουν λίγη χρήσιμη πληροφορία για το πραγματικό σύστημα.

Από τη στιγμή που έχει ληφθεί η απόφαση να χρησιμοποιηθεί προσομοίωση για τη μελέτη ενός συστήματος, έχει παρατηρηθεί ότι μπορούν να εμφανισθούν αρκετά προβλήματα στην πορεία υλοποίησης ενός επιτυχημένου προσομοιωτή, όπως:
➢ Όχι καλά ορισμένοι στόχοι κατά την έναρξη της μελέτης.

➢ Ακατάλληλο επίπεδο λεπτομέρειας του μοντέλου.

➢ Χειρισμός της μελέτης με προσομοίωση, σαν να ήταν βασικά μία δύσκολη άσκηση προγραμματισμού.

➢ Έλλειψη στοιχειωδών γνώσεων επιχειρησιακής έρευνας και στατιστικής.

➢ Χρήση εμπορικών πακέτων προσομοιωτών που μπορεί να περιέχουν λάθη ή να μην υλοποιούν τη λογική του συστήματος.

➢ Αποτυχία στη σωστή καταγραφή των πηγών τυχαιότητας του συστήματος.

➢ Χρήση αυθαίρετων κατανομών για την περιγραφή των εισόδων του προσομοιωτή.

➢ Χρήση λανθασμένων μέτρων απόδοσης. *[Γαροφαλάκης, 99]*

2.3 Πειραματικό περιβάλλον προσομοίωσης

Για την μελέτη των υπό εξέταση αλγόριθμων εξισορρόπησης φορτίου θεωρήθηκε αναγκαία η υλοποίηση καταλλήλου πειραματικού περιβάλλοντος προσομοίωσης (experimental simulation framework). Το περιβάλλον αυτό είναι υλοποιημένο σε γλώσσα προγραμματισμού Java, στηρίζεται στον πυρήνα της πλατφόρμας J-Sim (*[J-Sim, hp]*) (ο οποίος ουσιαστικά αποτελεί ένα σύστημα προσομοίωσης διακριτών γεγονότων-Discrete Event Simulation System) και επιτρέπει την προσομοίωση απλών αλγόριθμων εξισορρόπησης φορτίου που
i) ακολουθούν κεντρικές πολιτικές (*centralized policies*). Η συλλογή των πληροφοριών για την κατάσταση του συστήματος, καθώς επίσης η επιλογή του βέλτιστων διεκπεραιωτών για την ανάθεση των εργασιών αποτελούν ευθύνη μίας κεντρικής οντότητας (που καλείται και ως *συντονιστής -coordinator-, [Hura, 02]*), η οποία κατά συνέπεια αποτελεί σημείο αναφοράς για την διάταξη των υπόλοιπων οντοτήτων σον χώρο.
ii) δεν ακολουθούν κάποια ιδιαίτερη πολιτική επιλογής (*selection policy*). Οι εργασίες παράγονται και σειριακά ανατίθενται στον κατάλληλο διεκπεραιωτή,
iii) η ανανέωση της πληροφορίας για την κατάσταση του συστήματος λαμβάνει χώρα είτε με ρητή αίτηση της μονάδας στην οποία φιλοξενείται η πληροφορία αυτή (Demand-driven policy), είτε αυτοβούλως από τις υπόλοιπες μονάδες στην περίπτωση που η κατάσταση τους μεταβληθεί σε κάποιο βαθμό (State-change-driven policy) *[Lin, 92]* και
iv) η πολιτική τοποθεσίας υλοποιείται α) μέσω μίας συνάρτησης, καλούμενη ως *συνάρτησης αποτελεσματικότητας*, eff_x, (παράγραφος 2.3.3.1, κλάση *vpcs.StatusMessage*) η οποία λαμβάνει υπόψη την κατάσταση του διεκπεραιωτή. Βέλτιστος διεκπεραιωτής θεωρείται αυτός με την μικρότερη τιμή της eff_x ή β) διαφορετικά από τον χρήστη όπως αυτός επιθυμεί (π.χ. με τυχαία επιλογή των διεκπεραιωτών).

Στο παράρτημα Ι παρουσιάζονται οι σχέσεις των βασικών κλάσεων του πειραματικού περιβάλλοντος υπό την μορφή UML διαγραμμάτων κλάσεων.

2.3.1 Βασικές οντότητες

Για την προσομοίωση ενός υπολογιστικού συστήματος απαιτείται η υλοποίηση δύο τουλάχιστον οντοτήτων, η υπολογιστή μονάδα και η εργασία (ή το φορτίο). Στον Πίνακα 1 περιγράφονται οι μεταβλητές κατάστασης των παραπάνω οντοτήτων και κατεπέκταση του συστήματος. Αναγκαία επίσης είναι η εύρεση αλγορίθμου προσομοίωσης της συμπεριφοράς της κεντρικής μονάδας επεξεργασίας του διεκπεραιωτή η οποία και περιγράφεται στην παράγραφο 2.1.2. Σημειώνεται ότι οι μεταβλητές κατάστασης αποτελούν τις πρωτογενώς μετρούμενες ποσότητες βάσει των οποίων παράγονται τα μεγέθη για την αξιολόγηση της αποτελεσματικότητας των αλγορίθμων.

Πίνακας 1 – *Μεταβλητές κατάστασης των οντοτήτων και του συστήματος.*

Διεκπεραιωτής	***Μονάδες πόρου***. Εκφράζει το μέγεθος του πόρου του διεκπεραιωτή. Ο ορισμός του μεγέθους αυτού προέκυψε από το μοντέλο προσομοίωσης της κεντρικής μονάδας επεξεργασίας.
	Ταχύτητα. Εκφράζει την ταχύτητα επεξεργασίας του διεκπεραιωτή. Το μέγεθος αυτό αντιπροσωπεύει την ταχύτητα ενός υπολογιστή και προέκυψε από το μοντέλο προσομοίωσης της κεντρικής μονάδας επεξεργασίας.
	Τοπολογικός συντελεστής. Εκφράζει την «δικτυακή απόσταση» ενός διεκπεραιωτή από ένα σημείο αναφοράς (στο συγκεκριμένο μοντέλο, το σημείο αναφοράς είναι η γεννήτρια παραγωγής εργασιών) και προέκυψε από την ανάγκη μίας γενικής έκφρασης της καθυστέρησης που προέρχεται από την ανταλλαγή μηνυμάτων μεταξύ των οντοτήτων του συστήματος.
	Καθυστέρηση ανά μονάδα τοπολογικού συντελεστή. Εκφράζει την καθυστέρηση σε μονάδες χρόνου ανά μονάδα τοπολογικού συντελεστή. Ορίστηκε για την διαστασιολόγηση του τοπολογικού συντελεστή.
	Εναπομείναντες/ υπόλοιπες μονάδες εργασίας προς διεκπεραίωση. Εκφράζει τις μονάδες εργασίας που απομένουν για την ολοκλήρωση της.
	Αριθμός ενεργών εργασιών. Εκφράζει τον αριθμό των προς διεκπεραίωση εργασιών.
	Αριθμός διεκπεραιωμένων εργασιών. Εκφράζει τον αριθμό των εργασιών που έχουν ολοκληρωθεί.
	Αριθμός απορριφθέντων εργασιών. Εκφράζει τον αριθμό των εργασιών που απορρίφθηκαν από την εφαρμοζόμενη πολιτική στον διεκπεραιωτή.
Εργασία	***Μονάδες εργασίας***. Εκφράζει το μέγεθος μίας εργασίας.
	Προτεραιότητα. Εκφράζει την κλάση προτεραιότητας της εργασίας.
	Χρόνος γένεσης. Εκφράζει τον χρόνο γένεσης της εργασίας.
	Χρόνος άφιξης. Εκφράζει τον χρόνο άφιξης της εργασίας στον διεκπεραιωτή.
	Χρόνος ολοκλήρωσης. Εκφράζει τον χρόνο διεκπεραίωσης της εργασίας.

2.3.2 Αλγόριθμος κεντρικής μονάδας επεξεργασίας

Στην βιβλιογραφία απαντώνται δύο κατηγορίες προσομοίωσης της κεντρικής μονάδας επεξεργασίας (*CPU*). Στην πρώτη κατηγορία η CPU δεν αποτελεί τίποτε άλλο από μία συνάρτηση πυκνότητας πιθανότητας (π.χ. *poisson*) η οποία «παράγει» τους «*χρόνους εξυπηρέτησης*» των εργασιών *[Karatza, 02]* (ουσιαστικά ένα σύστημα αναμονής με συναρτήσεις που καθορίζουν την κατανομή του χρόνου παραμονής των πελατών σε αυτό). Στην δεύτερη κατηγορία, η προσομοίωση πραγματοποιείται με εφαρμογή των αρχών που διέπουν το πραγματικό σύστημα και είναι αυτή που θα εφαρμοστεί στην συγκεκριμένη περίπτωση. *[Gomoluch, 03]*

2.3.2.1 Μοντέλο

Η κεντρική μονάδα επεξεργασίας στηρίζεται στην πολιτική της «*αναλογικής κατανομής πόρων*» (*Proportional Share Resource Allocation*) *[Jeffay, 98]* (ουσιαστικά μία time-shared πολιτική) η οποία βασίζεται στους ακόλουθους κανόνες:
i) ο πόρος (στην ουσία ο χρόνος της CPU) διαιρείται σε διακριτές ποσότητες οι οποίες δεσμεύονται για ορισμένο χρονικό διάστημα προς την εκτέλεση μιας διεργασίας (εργασίας).

ii) Για κάθε διεργασία, το *βάρος* (*weight*) καθορίζει το σχετικό (σε σχέση με τις υπόλοιπες εργασίες) ποσοστό του πόρου που δεσμεύεται για την διεργασία αυτή στην μονάδα του χρόνου.

Εάν w_i αναπαριστά το βάρος της διεργασίας *i* και *A(t)* το σύνολο των ενεργών διεργασιών την χρονική στιγμή *t*, τότε η ποσότητα $f_i(t)$ (βαθμός δέσμευσης πόρου) ορίζει το κλάσμα του πόρου που δεσμεύεται για την εκτέλεση της διεργασίας *i* την χρονική στιγμή *t*, όπως περιγράφεται από την Σχέση 1:

$$f_i(t) = \frac{w_i}{\sum_{j \in A(t)} w_j} \quad \text{(Σχέση 1)}$$

Στην περίπτωση που ο βαθμός δέσμευσης πόρου, $f_i(t)$, της διεργασίας *i* παραμένει σταθερός για το χρονικό διάστημα $[t_1, t_2]$, τότε η διεργασία *i* μπορεί να χρησιμοποιήσει τον πόρο, στο χρονικό διάστημα $[t_1, t_2]$, για $(t_2 - t_1) \times f_i(t)$ μονάδες χρόνου. Σημειώνεται ότι η γένεση και η καταστροφή διεργασιών έχει σαν αποτέλεσμα την μεταβολή των μεγεθών *A(t)* και $f_i(t)$, καθώς από την Σχέση 1 είναι προφανές ότι, η αύξηση του αθροίσματος των βαρών των διεργασιών προκαλεί μείωση των αντίστοιχων βαθμών δέσμευσης πόρων (για κάθε διεργασία διαθέτονται λιγότερες μονάδες του πόρου στην μονάδα του χρόνου) και αντίστροφα.

Σύμφωνα με τα παραπάνω, απαραίτητη προϋπόθεση για την υλοποίηση του μοντέλου της CPU είναι ο καθορισμός των μεγεθών που αντιπροσωπεύουν τον πόρο και την διεργασία καθώς και η ποσοτικοποίηση αυτών μέσω κάποιων μονάδων μέτρησης. Αναγκαία επιπλέον είναι η συσχέτιση μεταξύ του μεγέθους μιας εργασίας με αυτό του πόρου που απαιτείται για την διεκπεραίωση της εργασίας αυτής. Με βάση τους προηγούμενους συλλογισμούς διατυπώνονται οι ακόλουθες προτάσεις:

i) Ο πόρος του διεκπεραιωτή αποτελείται από ένα σύνολο μονάδων, που καλούνται μονάδες πόρου.
ii) Το μέγεθος μίας εργασίας εκφράζεται από ένα σύνολο μονάδων, που καλούνται μονάδες εργασίας.
iii) Για κάθε εργασία δεσμεύεται ο μέγιστος αριθμός διαθέσιμων μονάδων του πόρων.
iv) Ως μονάδα χρόνου ορίζεται το χρονικό διάστημα που απαιτείται από μία μονάδα πόρου να διεκπεραιώσει μία μονάδα εργασίας. Η σχέση αυτή ισχύει για την κεντρική μονάδα επεξεργασίας αναφοράς (baseline), CPU_b, της οποίας η ταχύτητα επεξεργασίας θεωρούμε ότι ισούται με την μονάδα.
v) Η ταχύτητα, sp_i, μίας κεντρικής μονάδας επεξεργασίας i, CPU_i, εκφράζεται με έναν θετικό ακέραιο αριθμό ο οποίος δηλώνει την αναλογία μεταξύ της CPU_i και της CPU_b , δηλαδή $sp_i = (CPU_i / CPU_b)$.
vi) Το βάρος μίας εργασίας εκφράζεται με την κλάση προτεραιότητας της εργασίας αυτής.

Σύμφωνα με την προηγούμενη ανάλυση και για τυχαίο χρονικό διάστημα Δt_m όπου ο αριθμός των ενεργών εργασιών στην CPU παραμένει σταθερός, προκύπτουν οι ακόλουθες σχέσεις. Από την Σχέση 2 υπολογίζονται οι διαθέσιμες μονάδες πόρου, cpu_i, για την εκτέλεση της εργασίας *i*. Το ποσοστό των μονάδων του πόρου που δεσμεύονται επί του συνόλου καθορίζεται από το πηλίκο της προτεραιότητας της εργασίας i με το άθροισμα των προτεραιοτήτων του συνόλου των ενεργών εργασιών. Η ποσότητα αυτή μεταβάλλεται με την μεταβολή των ενεργών εργασιών, δηλαδή με την ολοκλήρωση ή την ανάθεση νέων εργασιών στην CPU. Η Σχέση 3 υπολογίζει τις μονάδες της εργασίας *i*, tu_i^m , που ολοκληρώνονται μέσα σε χρονικό διάστημα Δt_m. Η ποσότητα tu_i^m είναι ανάλογη του χρόνου Δt_m, της ταχύτητας επεξεργασίας *sp* και των δεσμευμένων μονάδων του πόρου για την εργασία *i*, cpu_i.

$$cpu_i = \frac{p_i}{\sum_{j=1}^{n} p_j} \times cpu \quad \text{(Σχέση 2)}$$

$$tu_i^m = \Delta t_m \times cpu_i \times sp \quad \text{(Σχέση 3)}$$

Ο χρόνος που διέρχεται μεταξύ της αφίξεως μίας εργασίας *i* στην CPU (t_k) και της ολοκλήρωσης (t_u) δίνεται από την σχέση $\sum_{m=k}^{u} \Delta t_m$, όπου $\sum_{m=k}^{u} tu_i^m = tu_i$ και tu_i το μέγεθος της εργασίας *i*. Στην εργασία των Gomoluch et al. [19] η CPU μοντελοποιήθηκε με παρόμοιο τρόπο.

Έτσι για παράδειγμα α) μία εργασία εκατό μονάδων διεκπεραιώνεται στην μονάδα του χρόνου αν οι μονάδες πόρου που είναι διαθέσιμες για την εργασία αυτή είναι εκατό και η ταχύτητα επεξεργασίας έχει το μέγεθος της μονάδας, β) εργασία με ίδιες μονάδες αλλά με διπλάσια προτεραιότητα από κάποια άλλη, ολοκληρώνεται στο μισό χρόνο από τον ίδιο πόρο και γ) πόρος με διπλάσια ταχύτητα από κάποιον άλλο, διεκπεραιώνει τις ίδιες εργασίες στο μισό χρόνο.

2.3.2.2 Αξιολόγηση μοντέλου

Ο αλγόριθμος της κεντρικής μονάδας επεξεργασίας που χρησιμοποιείται στο σύστημα προσομοίωσης εξισορρόπησης φορτίου αξιολογήθηκε και συγκρίθηκε με πραγματικά αποτελέσματα. Συγκεκριμένα πραγματοποιήθηκαν τα ακόλουθα δύο πειράματα:

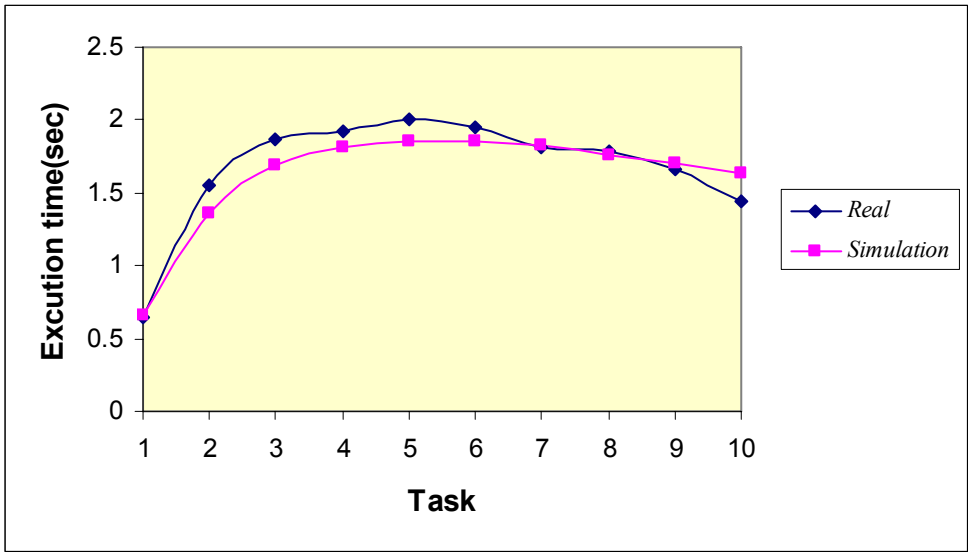
i) Για την μελέτη της συμπεριφοράς μίας πραγματικής κεντρικής μονάδας υλοποιήθηκε πρόγραμμα σε Java το οποίο ανά τακτά χρονικά διαστήματα παράγει ένα ξεχωριστό περιβάλλον εκτέλεσης (κοινώς *thread*) όπου και εκτελείται ένα σύνολο εντολών (διπλός κενός βρόγχος). Για κάθε *thread* που παράγεται καταγράφονται ο χρόνος έναρξης και ολοκλήρωσης του, από όπου και υπολογίζεται ο χρόνος εκτέλεσης του. Στο πείραμα παρήχθησαν δέκα στο πλήθος *threads* με ρυθμό δημιουργίας 0.1 sec.

ii) Στο περιβάλλον προσομοίωσης επαναλαμβάνουμε ανάλογο πείραμα. Στο XML έγγραφο διαμόρφωσης (παράγραφος 2.3.3.2) του συστήματος ορίζουμε α) ένα διεκπεραιωτή με μονάδες πόρου και ταχύτητα επεξεργασίας 100 και 1 και β) μεσοδιάστημα και συνολικός χρόνος παραγωγής εργασιών 0.1 sec και 10 sec αντίστοιχα.

Αρχικά θέτουμε τις μονάδες εργασίας αυθαίρετα και πραγματοποιούμε το πρώτο πείραμα προσομοίωσης. Από τα αποτελέσματα που λαμβάνουμε συγκρίνουμε τον χρόνο εκτέλεσης της πρώτης εργασίας (t_{1s}) με τον αντίστοιχο που προέκυψε από το πείραμα στην πραγματική μονάδα (t_{1r}). Επαναλαμβάνουμε την προσομοίωση μεταβάλλοντας τις μονάδες εργασίας έτσι ώστε τελικά οι δύο χρόνοι εκτέλεσης να είναι προσεγγιστικά ίδιοι ($t_{1s} \approx t_{1r}$). Με τον τρόπο αυτό εξασφαλίζεται κοινή βάση σύγκρισης για τις δύο σειρές των αποτελεσμάτων οι οποίες περιέχονται και απεικονίζονται στον Πίνακα 2 και στο Διάγραμμα 1 αντίστοιχα.

Πίνακας 2 – *Αποτελέσματα εκτέλεσης εργασιών σε πραγματικό περιβάλλον καθώς και σε αυτό της προσομοίωσης.*

	Χρόνοι εκτέλεσης	
Εργασία	*Πραγματικό*	*Προσομοίωση*
1	**0.641**	**0.657**
2	**1.547**	**1.359**
3	**1.875**	**1.687**
4	**1.922**	**1.812**
5	**2.000**	**1.859**
6	**1.953**	**1.859**
7	**1.812**	**1.828**
8	**1.781**	**1.764**
9	**1.656**	**1.702**
10	**1.437**	**1.641**



Διάγραμμα 1 – *Μεταβολή χρόνου εκτέλεσης των εργασιών σε πραγματικές συνθήκες και στην προσομοίωση.*

Από την σύγκριση των αποτελεσμάτων διαπιστώνουμε ότι i) Ο ρυθμός μεταβολής του χρόνου εκτέλεσης είναι παρόμοιος και για τις δύο σειρές αποτελεσμάτων. Αρχικά παρατηρείται μια απότομη αύξηση του χρόνου εκτέλεσης (η διαφορά των χρόνων εκτέλεσης μεταξύ της πρώτης εργασίας και των υπολοίπων είναι σημαντική) καθώς το σύστημα συνεχώς επιβαρύνεται με την δημιουργία νέων εργασιών. Ακολουθεί μείωση του ρυθμού αύξησης των χρόνων καθώς δεν παράγονται νέες εργασίες με αποτέλεσμα ο πόρος του συστήματος να μην υφίσταται περαιτέρω επιβάρυνση προς την διεκπεραίωση νέων εργασιών. Τέλος η διεκπεραίωση κάποιων εργασιών έχει σαν άμεσο αποτέλεσμα την αποδέσμευση τμήματος του πόρου και χρησιμοποίηση του προς την διεκπεραίωση των υπόλοιπων. Γεγονός που επιβεβαιώνεται με την μείωση του χρόνου εκτέλεσης, και ii) όλοι οι χρόνοι είναι της ίδιας τάξης μεγέθους.

Από την παραπάνω αξιολόγηση μπορούμε να θεωρήσουμε ότι ο αλγόριθμος είναι αξιόπιστος και μπορεί να χρησιμοποιηθεί για την προσομοίωση της συμπεριφοράς της κεντρικής μονάδος επεξεργασίας.

2.3.3 Υλοποίηση

Το πειραματικό περιβάλλον προσομοίωσης υλοποιήθηκε σε γλώσσα προγραμματισμού Java με χρήση του πυρήνα της πλατφόρμας J-Sim. Οι κλάσεις, ανάλογα με λειτουργία τους, μπορούν να ταξινομηθούν στις ακόλουθες κατηγορίες.
i) Κλάσεις οντοτήτων, όπου περιέχονται οι κλάσεις που ενσαρκώνουν τις κύριες οντότητες του συστήματος που προσομοιώνεται (όπως οι υπολογιστικές μονάδες και οι εργασίες).
ii) Κλάσεις μηνυμάτων, οι οποίες αναφέρονται στα μηνύματα που ανταλλάσσονται μεταξύ οντοτήτων και σηματοδοτούν την έγερση γεγονότων σε καθορισμένους χρόνους.
iii) Κλάσεις εξαιρέσεων, οι οποίες εκφράζουν αστοχίες του συστήματος.
iv) Βοηθητικές κλάσεις, στις οποίες ανήκουν αυτές που συνεισφέρουν στην δημιουργία του μοντέλου προσομοίωσης (όπως κατασκευή της τοπολογίας του υπολογιστικού συστήματος βάσει των παραμέτρων που έχουν οριστεί στο XML έγγραφο).
v) Εξωτερικά οριζόμενες κλάσεις. Οι κλάσεις αυτές ορίζονται εξωτερικά από τον χρήστη για την υλοποίηση της συμπεριφοράς του μοντέλου προσομοίωσης, όπως κατανομή παραγωγής εργασιών, κατανομή μονάδων εργασίας και κατανομή προτεραιοτήτων εργασιών)

Το μοντέλο προσομοίωσης ορίζεται μέσω ενός εγγράφου XML, καλούμενο ως έγγραφο διαμόρφωσης του συστήματος. Στο έγγραφο αυτό καθορίζονται στοιχεία όπως, η τοπολογία του υπολογιστικού συστήματος (απόσταση των μονάδων από ένα σημείο αναφοράς), τα χαρακτηριστικά των μονάδων (όπως μέγεθος πόρων, ταχύτητα επεξεργασίας), η εφαρμοζόμενη πολιτική πληροφόρησης και τοποθεσίας, οι κλάσεις και οι μέθοδοι υλοποίησης των διάφορων κατανομών που χρησιμοποιούνται κ.α.

Στις παραγράφους που ακολουθούν περιγράφονται α) μερικές από τις βασικές κλάσεις του συστήματος προσομοίωσης, β) το έγγραφο διαμορφώσεις του συστήματος και γ) ένα χαρακτηριστικό παράδειγμα.

Λεπτομερής αναφορά των κλάσεων του περιβάλλοντος προσομοίωσης παρατίθεται στο παράρτημα Ι.

2.3.3.1 Βασικές κλάσεις

Οι βασικές κλάσεις του περιβάλλοντος προσομοίωσης είναι οι ακόλουθες.

- *VPC*. Εκφράζει την υπολογιστική μονάδα και υλοποιεί το μοντέλο προσομοίωσης της κεντρικής μονάδας επεξεργασίας. Ένα αντικείμενο της κλάσης αυτής, δηλαδή μια υπολογιστική μονάδα, συσχετίζεται με δύο τύπους μηνυμάτων: α) αυτό που μεταφέρει μια εργασία (κλάση VTask) και β) αυτό που φιλοξενεί πληροφορίες για την κατάσταση της (το *μήνυμα κατάστασης*, κλάση StatusMessage). Κάθε μονάδα ενημερώνει για την κατάστασης της την οντότητα που υλοποιείται μέσω της κλάσης InfoCenter. Η ενημέρωση της οντότητας αυτής λαμβάνει χώρα α) με ρητή αίτηση της προς τις μονάδες του υπολογιστικού συστήματος ή β) αυτοβούλως από τις ίδιες τις μονάδες όταν η μεταβολή κάποιας από τις μεταβλητές κατάστασης τους βρεθεί εκτός συγκεκριμένων ορίων.

- *InfoCenter*. Αναπαριστά την υπεύθυνη οντότητα για την διατήρηση της λίστας (καλούμενη ως *λίστα καταστάσεως*, *state list*) στην οποία φιλοξενείται η κατάσταση των υπολογιστικών μονάδων και κατά επέκταση του συστήματος (ουσιαστικά η λίστα αυτή αποτελείται από το σύνολο των *μηνυμάτων καταστάσεως* των μονάδων). Ο InfoCenter διαθέτει δύο τύπους

λειτουργίας: α) τον LIVE_MODE όπου οι αποφάσεις των εφαρμοζόμενων πολιτικών στηρίζονται σε «σύγχρονο» στιγμιότυπο της κατάστασης του συστήματος. Στην περίπτωση αυτή, πριν την λήψη μιας απόφασης, ο InfoCenter αποστέλλει αιτήσεις στους διεκπεραιωτές για αποστολή των μηνυμάτων καταστάσεως τους σε αυτόν και β) τον NO_LIVE_MODE όπου οι αποφάσεις πραγματοποιούνται σύμφωνα με την υπάρχουσα πληροφορία στην λίστα καταστάσεως.

➢ *TaskGenerator*. Αντιπροσωπεύει την υπεύθυνη οντότητα για την παραγωγή και την ανάθεση των εργασιών στις κατάλληλες υπολογιστικές μονάδες, βάσει της εφαρμοζόμενης πολιτικής τοποθεσίας.

Στο σημείο αυτό πρέπει να επισημανθεί ότι οι κλάσεις InfoCenter και TaskGenerator ουσιαστικά αντιπροσωπεύουν μία κεντρική οντότητα, τον συντονιστή, καθώς η παραγωγή των εργασιών και η εφαρμογή των πολιτικών πληροφόρησης και τοποθεσίας πραγματοποιούνται από την οντότητα αυτή.

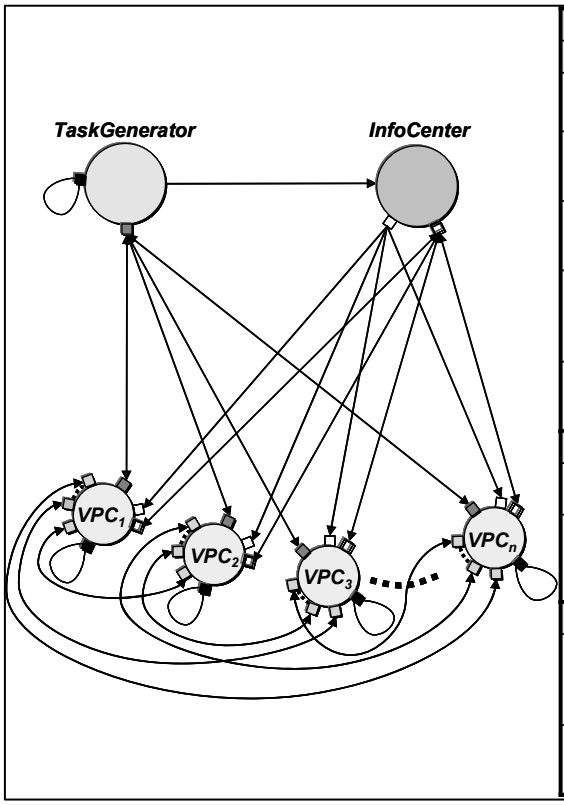
➢ *WMessage*. Αποτελεί την βασική κλάση υλοποίησης όλων των μηνυμάτων που ανταλλάσσονται μεταξύ των οντοτήτων. Η κλάση αυτή ενθυλακώνει όλες τις απαραίτητες πληροφορίες για τα σημαντικά γεγονότα που συμβαίνουν στον κύκλο ζωής ενός μηνύματος, όπως είναι α) ο χρόνος δημιουργίας και καταστροφής του και β) οι χρόνοι άφιξης και αναχώρησης σε και από μία οντότητα αντίστοιχα.

➢ *VTask*. Εκφράζει την εργασία. Μία εργασία κατά τον κύκλο ζωής της μπορεί να χαρακτηριστεί ως α) NEW_TASK, στην περίπτωση που δεν έχει ανατεθεί σε κάποια υπολογιστική μονάδα, β) NOT_COMPLETE, όταν εκτελείται σε κάποια μονάδα χωρίς ακόμα να έχει ολοκληρωθεί και γ) COMPLETE, όταν η εργασία ολοκληρωθεί.

➢ *StatusMessage*. Αναπαριστά το μήνυμα που εσωκλείει τις πληροφορίες για την κατάσταση μιας υπολογιστικής μονάδας (*μήνυμα κατάστασης*), όπως το πλήθος των ενεργών εργασιών, το μέγεθος του φορτίου κ.α. Η κλάση αυτή υλοποιεί την συνάρτηση αποτελεσματικότητας (efficiency function), eff_x, η οποία καθορίζει την βέλτιστη μονάδα για την εκτέλεση μίας εργασίας, εκφράζοντας με τον τρόπο αυτό την εφαρμοζόμενη πολιτική τοποθεσίας. Επί του παρόντος η eff_x περιγράφεται από την σχέση $eff_x = \frac{(tLCoef \times totalLoad)/(ntCoef \times nTasks)}{spCoef \times speed} + topoCoef \times topoFactor$, όπου *totalLoad*, *nTasks*, *speed* και *topoFactor* είναι το φορτίο, το πλήθος των ενεργών εργασιών, η ταχύτητα επεξεργασίας και ο τοπολογικός συντελεστής της υπολογιστικής μονάδας αντίστοιχα. Οι συντελεστές *spCoef*, *topoCoef*, *tLCoef* και *ntCoef* ορίζονται στο έγγραφο διαμόρφωσης του συστήματος.

Η εφαρμογή της eff_x αποτελεί την προκαθορισμένη πολιτική τοποθεσίας του συστήματος. Παρόλα αυτά, ο χρήστης έχει την δυνατότητα να υλοποιήσει τις δικές του πολιτικές και να τις χρησιμοποιήσει με την κατάλληλη δήλωση τους στο έγγραφο διαμόρφωσης του συστήματος (όπως έγινε και για την υλοποίηση του αλγόριθμου Round-Robin).

Στο Σχήμα 5 απεικονίζεται η συσχέτιση μεταξύ των βασικών κλάσεων, σε επίπεδο επικοινωνίας με την αποστολή και λήψη μηνυμάτων.



VPC – Ports			
Συμ.	Τύπος	Χρησιμότητα	Σύνδεση με
	Fork	Για την σηματοδότηση ολοκλήρωσης μιας εργασίας	Με τον εαυτό του (1 σύνδεση)
	Common	Για την επικοινωνία με τις ομότιμες οντότητες	(n-1) VPCs Διπλής κατεύθυνσης
	Common	Για την επικοινωνία με τις υπόλοιπες οντότητες	TaskGenerator Διπλής κατεύθυνσης
	Common	Για την λήψη μηνυμάτων που απευθύνονται σε όλες τις οντότητες (π.χ. αίτηση από τον InfoCenter για αποστολή μηνύματος καταστάσεως)	InfoCenter Μονής κατεύθυνσης (λήψη μόνο)
	Common	Για το μοντέλο εξυπηρετητή-πελάτη (πλευρά πελάτη).	InfoCenter Διπλής κατεύθυνσης
TaskGenerator – Ports			
	Fork	Για την σηματοδότηση δημιουργίας μιας νέας εργασίας	Με τον εαυτό του (1 σύνδεση)
	Common	Για την επικοινωνία με τις υπόλοιπες οντότητες (χρησιμοποιείται για την ανάθεση των εργασιών στα VPC)	(n-1) VPCs Διπλής κατεύθυνσης
InfoCenter – Ports			
	Common	Χρησιμοποιείται για την αποστολή μηνυμάτων που απευθύνονται σε όλες τις οντότητες (αίτηση για την λήψη μηνυμάτων καταστάσεως)	(n-1) VPCs Μονής κατεύθυνσης (αποστολή μόνο)
	Server	Για το μοντέλο εξυπηρετητή-πελάτη (πλευρά εξυπηρετητή).	(n-1) VPCs Διπλής κατεύθυνσης

Σχήμα 5 – *Συνδεσμολογία βασικών κλάσεων.*

2.3.3.2 Έγγραφο διαμόρφωσης του συστήματος

Το έγγραφο διαμόρφωσης του συστήματος είναι ένα XML έγγραφο το οποίο χρησιμοποιείται για τον ορισμό του μοντέλου της προσομοίωσης και αποτελείται από τις ακόλουθες τέσσερις περιοχές.

➢ *Περιοχή Διεκπεραιωτών* (στοιχείο <vpcs>), όπου περιγράφονται οι υπολογιστικές μονάδες. Στην περιοχή αυτή ο χρήστης έχει την δυνατότητα να ορίσει τα χαρακτηριστικά των μονάδων του υπολογιστικού συστήματος, θέτοντας τις επιθυμητές τιμές στα κατάλληλα σημεία του XML εγγράφου. Μια μονάδα περιγράφεται από τις ιδιότητες και το περιεχόμενο ενός στοιχείου <vpc> Για παράδειγμα, η ταχύτητα επεξεργασίας, οι μονάδες του πόρου και ο τοπολογικός συντελεστής φιλοξενούνται στις ιδιότητες speed, cpu_power και topology_factor αντίστοιχα, ενώ η τιμές των ιδιοτήτων id και replicas ορίζουν τον μοναδικό κωδικό αναγνώρισης και το αριθμό των αντιγράφων της υπολογιστικής μονάδας. Υπενθυμίζεται ότι ο τοπολογικός συντελεστής ορίζεται σε σχέση με τον συντονιστή (σημείο αναφοράς).

Επιπλέον, στην περιοχή αυτή καθορίζονται παράμετροι σχετικοί με την συμπεριφορά των εφαρμοζόμενων πολιτικών καθώς και με γενικά χαρακτηριστικά του μοντέλου. Για παράδειγμα,
α) μια θετική τιμή της ιδιότητας above_notify_thres υποδεικνύει το ποσοστό μεταβολής του φορτίου της υπολογιστικής μονάδας πάνω από το οποίο θα αποσταλεί μήνυμα ενημέρωσης στον συντονιστή (το όριο ανανέωσης, όπως ήδη έχει αναφερθεί). Αντιθέτως, μία αρνητική τιμή απενεργοποιεί την προαναφερθείσα διαδικασία,
β) η τιμή της ιδιότητας mdelay_per_tf εκφράζει τον χρόνο, σε sec ανά μονάδα τοπολογικού συντελεστή, που απαιτείται για την επικοινωνία μεταξύ των οντοτήτων του συστήματος και

γ) η τιμή της ιδιότητας cpu_thres καθορίζει το κατώτατο όριο των μονάδων του πόρου ενός διεκπεραιωτή που μπορεί να διατεθούν για την εκτέλεση μίας εργασίας.

➢ *Περιοχή κατανομής παραγωγής εργασιών* (στοιχείο < task_generation_distribution>). Στην περιοχή αυτή ορίζονται κατανομές μεσοδιαστήματος παραγωγής εργασιών (σε sec). Μια κατανομή περιγράφεται από τις ιδιότητες και το περιεχόμενο ενός στοιχείου <fun> όπου
α) ορίζονται το όνομα της κλάσης και της μεθόδου από την οποία υλοποιείται (τιμές των ιδιοτήτων method_name και class_name αντίστοιχα),
β) υπάρχει η δυνατότητα ορισμού συνόλου παραμέτρων οι οποίες μπορούν να ανακτηθούν και να χρησιμοποιηθούν από την μέθοδο υλοποίησης της κατανομής. Οι παράμετροι αυτές περιγράφονται από τα ζεύγη τιμών των ιδιοτήτων name και val οι οποίες φιλοξενούν τα ονόματα και τις τιμές των παραμέτρων αντίστοιχα. Το όνομα μιας παραμέτρου χρησιμοποιείται ως κλειδί για την ανάκτηση της μέσα στην αντίστοιχη μέθοδο,
γ) η τιμή της ιδιότητας is_enable (true ή false) λειτουργεί ως σημαία για την χρήση της αντίστοιχης μεθόδου στο μοντέλο προσομοίωσης και
δ) η τιμή της ιδιότητας id του στοιχείου <task> αναφέρεται στο αντίστοιχο στοιχείο της περιοχής των εργασιών όπου καθορίζονται τα χαρακτηριστικά των παραγόμενων εργασιών (μέγεθος και κλάση προτεραιότητας).

➢ *Περιοχή πολιτικής τοποθεσίας* (στοιχείο <location_policy_functions>). Μια πολιτική τοποθεσίας ορίζεται από τις ιδιότητες και το περιεχόμενο του στοιχείου <fun> με τρόπο παρόμοιο με αυτό της περιοχής κατανομής παραγωγής εργασιών.

➢ *Περιοχή Εργασιών* (στοιχείο <tasks_properties>). Στην περιοχή αυτή περιγράφονται τα χαρακτηριστικά (μέγεθος -σε μονάδες εργασίας- και προτεραιότητα) διαφόρων «τύπων» εργασιών. Ένας τύπος εργασιών ορίζεται από τις ιδιότητες και το περιεχόμενο ενός στοιχείου <task>. Στο στοιχείο αυτό οι τιμές των χαρακτηριστικών των εργασιών περιγράφονται α) ως σταθερές τιμές των ιδιοτήτων tunits και priority, για το μέγεθος και την προτεραιότητα αντίστοιχα ή β) με τα αντίστοιχα ονόματα των κλάσεων και των μεθόδων μέσω των οποίων επιστρέφονται. Ο χρήστης έχει την δυνατότητα ορισμού πλήθος «τύπων» εργασιών χρησιμοποιώντας την τιμή της ιδιότητας id ως αναγνωριστικό στοιχείο του «τύπου».

Σημειώνεται ότι, στην περίπτωση που το έγγραφο δεν συμπληρωθεί επαρκώς ή ο χρήστης επιθυμεί την δημιουργία αντιγράφων υπολογιστικών μονάδων, το σύστημα αρχικοποιεί και κατασκευάζει το μοντέλο προσομοίωσης χρησιμοποιώντας πληροφορίες από προκαθορισμένο σύνολο τιμών και κατάλληλη γεννήτρια κωδικών αναγνώρισης.

Στο Σχήμα 6 παρουσιάζεται ένα χαρακτηριστικό παράδειγμα εγγράφου διαμόρφωσης του συστήματος.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration SYSTEM "config.dtd">
<configuration>
 <description>The configuration file for the creation of VPCs grid</description>

 <vpcs description="VPCS template">
  <vpc id="vpc1" replicas="2" speed="1" cpu_power="100" topology_factor="1"/>
  <vpc id="vpc2" replicas="3" speed="2" cpu_power="100" topology_factor="3"/>
  <rest_info cpu_thres="0." above_notify_thres="-0.1" mdelay_per_tf="0."/>
 </vpcs>

 <task_generation_distribution description="function of tasks inter arrival">
  <fun description="const_interval" class_name="vpcs.ArrivalFuncs" method_name="constIntervalFunc" is_enable="true">
   <param description="interval" val="0.05"/>
   <task id="taskv1"/>
  </fun>
 </task_generation_distribution>

 <location_policy_functions description="function of VPC selection">
  <fun description="efficiency" class_name="vpcs.SelectionFuncs" method_name="effLiveUpdBasedSelection" is_enable="true">
   <param name="speed_cf" val="1"/>
   <param name="topology_cf" val="0"/>
   <param name="tload_cf" val="-1"/>
   <param name="ntasks" val="0"/>
  </fun>
  <fun description="Round Robin" class_name="vpcs.SelectionFuncs" method_name="roundRobinSelection" is_enable="false"/>
  <fun description="random" class_name="vpcs.SelectionFuncs" method_name="randSelection" is_enable="false"/>
 </location_policy_functions>

 <tasks_properties>
  <task id="taskv1" tunits="110" priority="1"/>
 </tasks_properties>

</configuration>
```

Σχήμα 6 – *XML έγγραφο διαμόρφωσης του συστήματος.*

2.3.3.3 Σενάριο λειτουργίας

Για την καλύτερη κατανόηση του πειραματικού περιβάλλοντος ακολουθεί υποδειγματικό παράδειγμα κατάστρωσης και εκτέλεσης πειράματος.

Έστω ότι επιθυμούμε να προσομοιώσουμε ένα σύστημα το οποίο έχει τα ακόλουθα χαρακτηριστικά:

➢Αποτελείται από α) δύο υπολογιστικές μονάδες με μέγεθος πόρου, ταχύτητα επεξεργασίας και τοπολογικό συντελεστή 100, 1και 1, και β) τρεις υπολογιστικές μονάδες με τιμές των παραπάνω χαρακτηριστικών, 100, 2 και 3, αντίστοιχα. Υπενθυμίζεται ότι ο τοπολογικός συντελεστής ορίζεται χρησιμοποιώντας ως σημείο αναφοράς τον συντονιστή.

➢ Το μέγεθος των παραγόμενων εργασιών είναι σταθερό και ίσο με 110 μονάδες, ενώ το μεσοδιάστημα παραγωγής τους ορίζεται από την επιστρεφόμενη τιμή της μεθόδου constIntervalFunc η οποία υλοποιείται στην κλάση vpcs.ArrivalFuncs. Η μέθοδος αυτή απλώς ανακτά και επιστρέφει την τιμή της παραμέτρου interval (η οποία έχει οριστεί στο στοιχείο <param>) με αποτέλεσμα το μεσοδιάστημα παραγωγής των εργασιών να είναι σταθερό και ίσο με 0.05 sec.

➢ Η εφαρμοζόμενη πολιτική τοποθεσίας υλοποιείται από την μέθοδο effLiveUpdBasedSelection της κλάσης vpcs.SelectionFuncs. Η μέθοδος αυτή ενσαρκώνει την προκαθορισμένη πολιτική τοποθεσίας του περιβάλλοντος προσομοίωσης, όπου η επιλογή της βέλτιστης υπολογιστικής μονάδας πραγματοποιείται βάσει της συνάρτησης αποτελεσματικότητας. Παράλληλα η τιμή της ιδιότητας above_notify_thres είναι αρνητική,

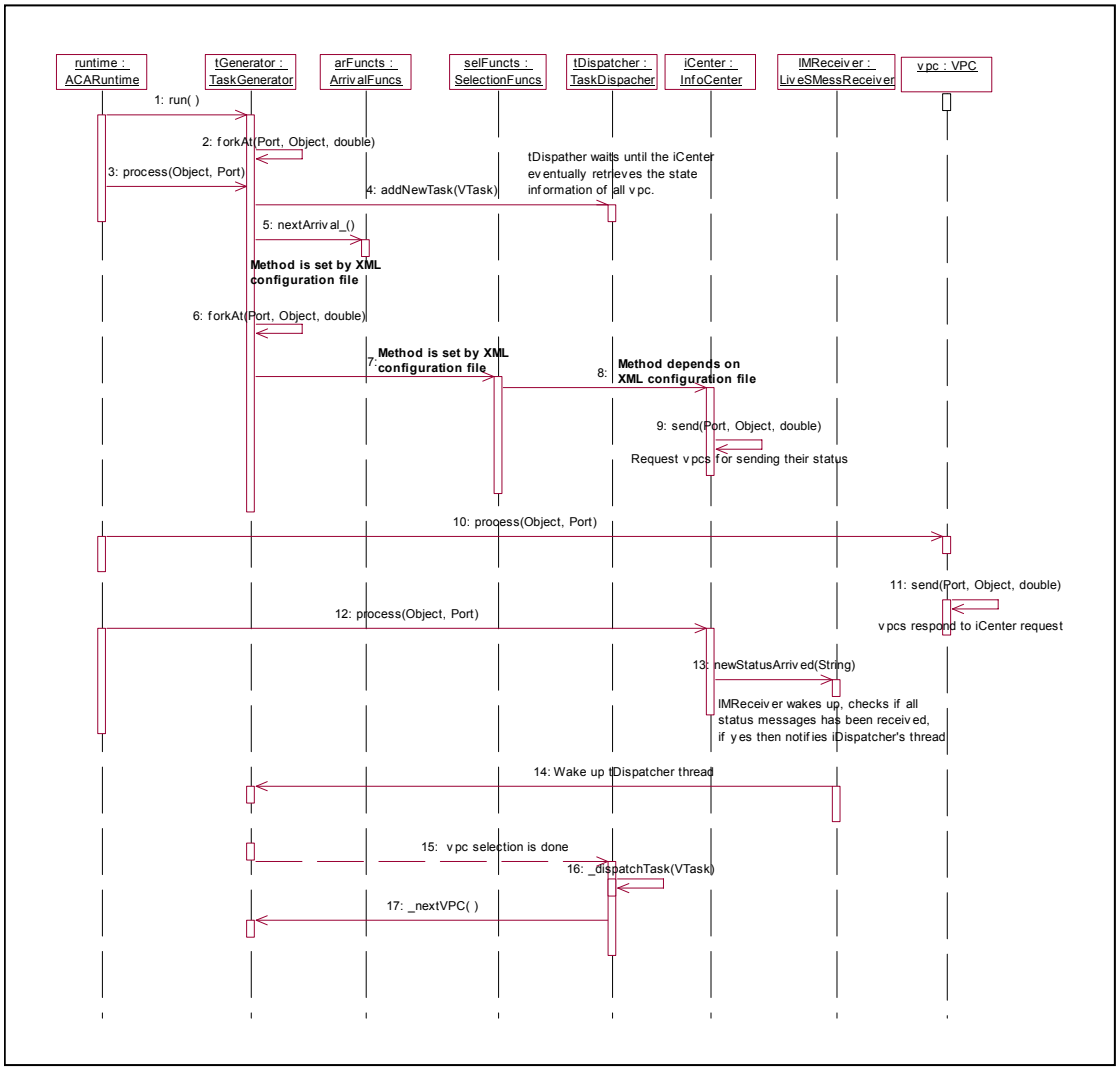
γεγονός που δηλώνει ότι οι αποφάσεις του συστήματος στηρίζονται σε «σύγχρονο» στιγμιότυπο της κατάστασης του (κατάσταση λειτουργίας του InfoCenter, LIVE_MODE)

Το έγγραφο διαμόρφωσης του συστήματος, για το προηγούμενο σενάριο προσομοίωσης έχει την μορφή που απεικονίζεται στο Σχήμα 6.

Η έναρξη της προσομοίωσης λαμβάνει χώρα με την εκτέλεση της κλάσης *vpcs.GridGenerator* και παραμέτρους α) το όνομα του εγγράφου διαμόρφωσης του συστήματος (π.χ. config.xml) και β) το χρονικό διάστημα (σε sec) παραγωγής εργασιών. Για παράδειγμα, "java vpcs.GridGenerator config.xml 500".

Η συμπεριφορά του συστήματος σε επίπεδο κλάσεων, στο συγκεκριμένο πείραμα, μπορεί να περιγραφεί ως ακολούθως. Η υπεύθυνη κλάση για την γένεση εργασιών, TaskGenerator, παράγει εργασίες (κλάση VTask) με ρυθμό 20 εργασίες/sec. Για κάθε εργασία, ζητείται από την κλάση που διατηρεί την λίστα κατάστασης των υπολογιστικών μονάδων, InfoCenter, να επιστρέψει την βέλτιστη μονάδα με βάση την σύγχρονη εικόνα του συστήματος. Έτσι, η InfoCenter αποστέλλει μηνύματα σε όλες τις υπολογιστικές μονάδες (κλάση VPC) προκειμένου να την ενημερώσουν για την κατάσταση τους (μήνυμα κατάστασης, κλάση StatusMessage). Με την λήψη και του τελευταίου μηνύματος κατάστασης (από την InfoCenter), λαμβάνει χώρα η επιλογή του βέλτιστης μονάδας βάσει της συνάρτησης αποτελεσματικότητας και ενημερώνεται η TaskGenerator. Τέλος, η TaskGenerator αποστέλλει την εργασία στην μονάδα που έχει επιλεχτεί.

Το παραπάνω σενάριο προσομοίωσης μπορεί να απεικονιστεί, σε επίπεδο μεθόδων, από το UML διάγραμμα του Σχήματος 7



Σχήμα 7 – *UML διάγραμμα ακολουθίας μεθόδων.*

Κεφάλαιο 3

Πειραματικό Μέρος

3 Πειραματικό Μέρος

Στο κεφάλαιο αυτό περιέχονται τα αποτελέσματα από την προσομοίωση τριών απλών αλγόριθμων εξισορρόπησης φορτίου. Σκοπός δεν είναι η ανάπτυξη ενός νέου αλγορίθμου αλλά η μελέτη της επίδρασης βασικών παραμέτρων, όπως η ταχύτητα επικοινωνίας και η τοπολογία, στην αποτελεσματικότητα ενός αλγορίθμου και κατά συνέπεια στην απόδοση ενός συστήματος. Η μελέτη αυτή οδηγεί σε χρήσιμα συμπεράσματα τα οποία μπορούν να γενικευτούν σε οποιαδήποτε σύστημα με παρόμοια χαρακτηριστικά και εφαρμοζόμενους μηχανισμούς.

Αναλυτικότερα, μελετάται η επίδραση i) της ταχύτητας επικοινωνίας μεταξύ των οντοτήτων του συστήματος συναρτήσει του επιβαλλόμενου φορτίου, ii) της ομοιογένειας του συστήματος, iii) του προβλήματος της παρωχημένης πληροφορίας και iv) της τοπολογίας, στην απόδοση του συστήματος.

Επιπλέον, παρουσιάζονται και σχολιάζονται τα μεγέθη (μέτρα απόδοσης) τα οποία ορίστηκαν με σκοπό την μελέτη των πειραματικών αποτελεσμάτων και την αξιολόγηση των εφαρμοζόμενων αλγόριθμων εξισορρόπησης φορτίου.

Σημειώνεται ότι, η διεξαγωγή των πειραμάτων πραγματοποιήθηκε με χρήση του πειραματικού περιβάλλοντος που περιγράφεται στο κεφάλαιο 2.

3.1 Εξεταζόμενοι αλγόριθμοι

Οι αλγόριθμοι που εξετάζονται είναι οι ακόλουθοι:

i) Ο *Live Update Information Algorithm* (*LUIA*), ο οποίος αναπτύχθηκε για τις ανάγκες των πειραμάτων. Ο αλγόριθμος αυτός α) μπορεί να χαρακτηριστεί ως κεντρικός αφού η εφαρμογή της πολιτικής πληροφόρησης, τοποθεσίας και επιλογής πραγματοποιείται από μία οντότητα η οποία επιπλέον είναι υπεύθυνη για την παραγωγή των εργασιών χωρίς να λαμβάνει η ίδια μέρος στην εκτέλεση τους. Η οντότητα αυτή, λόγω των πολλαπλών ρόλων της, στον παρόν κείμενο αναφέρεται ως *συντονιστής* (*coordinator*) η κατά περίπτωση *γεννήτρια εργασιών* (*task generator*). Ο συντονιστής διατηρεί ουρά στην οποία αναμένουν οι εργασίες οι οποίες δεν έχουν εξυπηρετηθεί ακόμα, δεν έχουν δηλαδή αποσταλεί σε μονάδα β) δεν ακολουθεί κάποια ιδιαίτερη πολιτική επιλογής. Οι εργασίες αποστέλλονται σειριακά, σύμφωνα με τον χρόνο παραγωγής τους, στην κατάλληλη μονάδα μετά την εφαρμογή της πολιτικής πληροφόρησης, γ) η επιλογή μιας μονάδας για την εκτέλεση μίας εργασίας (πολιτική τοποθεσίας) πραγματοποιείται βάσει του χρόνου που απαιτείται για την ολοκλήρωση του φορτίου με το οποίο έχει ήδη επιβαρυνθεί σε μία δεδομένη χρονική στιγμή. Το κριτήριο αυτό εκφράζεται μέσω της συνάρτησης, f_x, την οποία καλούμε συνάρτηση αποτελεσματικότητας, και περιγράφεται από την σχέση $f_x = totalLoad / speed$, όπου *totalLoad* και *speed* το φορτίο και η ταχύτητα επεξεργασίας της μονάδας. Βέλτιστη μονάδα θεωρείται αυτή με την μικρότερη τιμή της f_x, ενώ σε περίπτωση ισότητας επιλέγεται αυτή με την μεγαλύτερη ταχύτητα επεξεργασίας, δ) η απαιτούμενη πληροφορία για τον υπολογισμό των συναρτήσεων αποτελεσματικότητας των υπολογιστικών μονάδων (πολιτική πληροφόρησης) συλλέγεται με ρητή αίτηση της οντότητας όπου φιλοξενείται η πληροφορία αυτή (δηλαδή του συντονιστή), προς τις μονάδες του συστήματος για καταγραφή της κατάστασης τους. Με την απάντηση όλων των μονάδων πραγματοποιείται η επιλογή της βέλτιστης στην οποία και τελικά αποστέλλεται η εργασία. Η πολιτική πληροφόρησης εφαρμόζεται για κάθε εργασία που περιμένει στην ουρά του συντονιστή να εξυπηρετηθεί με αποτέλεσμα ο συνολικός αριθμός των ανταλλασσόμενων μηνυμάτων είναι $n \times (2 \times n + 1)$ (για κάθε εργασία, *n* ζεύγη αιτήσεων-απαντήσεων και ένα μήνυμα για την αποστολή της εργασίας), όπου *n* το πλήθος των εργασιών και ε) η εκτέλεση των εργασιών πραγματοποιείται άμεσα με την άφιξη τους στις

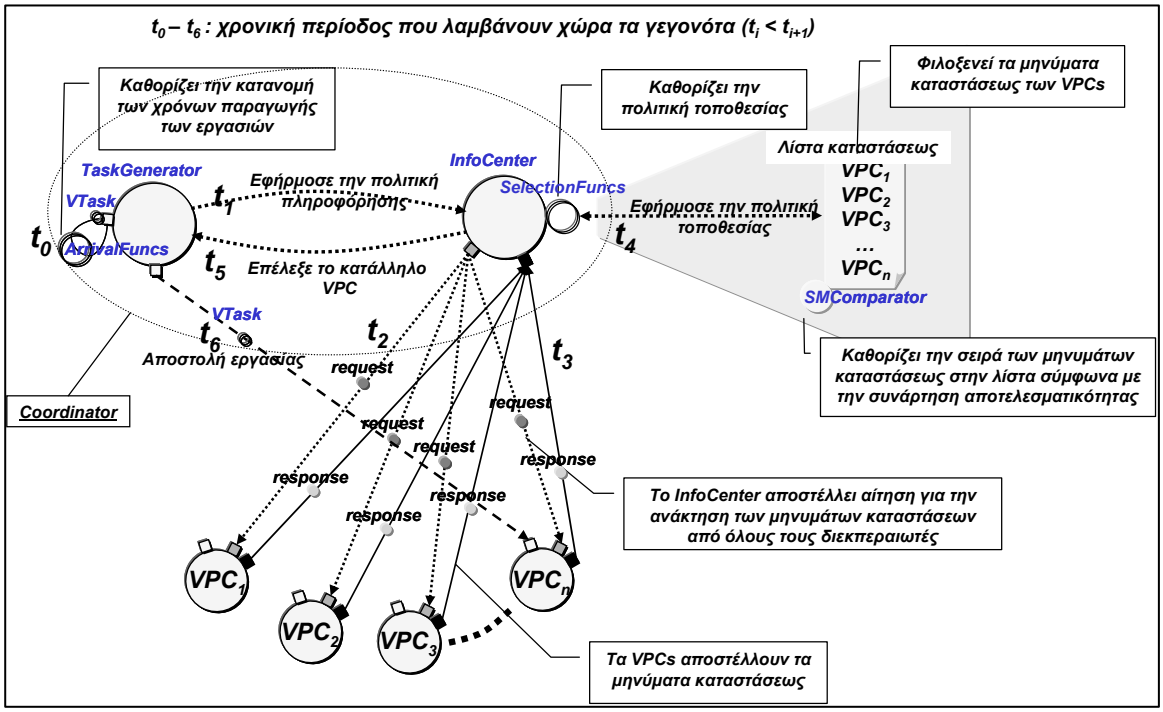
υπολογιστικές μονάδες (δεν υπάρχει χρόνος αναμονής σε ουρά στην μονάδα που θα εκτελεστεί η εργασία). Στο Σχήμα 1 περιγράφεται η λειτουργία του LUIA και απεικονίζονται, σε επίπεδο υλοποίησης, οι οντότητες που συμμετέχουν στην προσομοίωση (κοίτα παράγραφο 2.3.3.1 και παράρτημα Ι).

ii) *Update Information Algorithm* (*UIA*), ο οποίος αποτελεί παραλλαγή του LUIA. Η διαφοροποίηση μεταξύ των δύο αλγόριθμων έγκειται στην πολιτική πληροφόρησης. Στον UIA, η ενημέρωση του συντονιστή για την κατάσταση του συστήματος πραγματοποιείται αυτοβούλως από τις μονάδες όταν η μεταβολή του φορτίου τους υπερβεί ένα όριο το οποίο και καλούμε *όριο ενημέρωσης* (*update threshold*).

iii) Ο ντετερμινιστικός κλασικός αλγόριθμος *Round-Robin* (*RR*) *[Rasch, 70]*. Κατά την εφαρμογή του RR η ανάθεση των εργασιών από τον συντονιστή στις υπολογιστικές μονάδες πραγματοποιείται κυκλικά, αδιαφορώντας για την κατάσταση του συστήματος. Στην περίπτωση αυτή α) τα ανταλλασσόμενα μηνύματα μεταξύ των οντοτήτων περιορίζονται σε αυτά της αποστολής των εργασιών (ένα μήνυμα ανά εργασία) και β) δεν υπάρχει συσσώρευση εργασιών στην ουρά αναμονής του συντονιστή αφού η εργασίες αποστέλλονται κατευθείαν σε κάποια μονάδα.

Ο συντονιστής αποτελεί σημείο αναφοράς για την δημιουργία της τοπολογίας του συστήματος με αποτέλεσμα οι τοπολογικοί συντελεστές των υπόλοιπων μονάδων να αναφέρονται σε σχέση με την οντότητα αυτή.

Σημειώνεται ότι, παρόλο που ένα υπολογιστικό σύστημα μπορεί να αποτελείται από χιλιάδες μονάδες (ιδιαίτερα με την μορφή υπολογιστικού πλέγματος), στις προσομοιώσεις μας, για λόγους απλότητας, θεωρούμε ότι το σύστημα αποτελείται από ο μικρό αριθμό μονάδων και με απλή τοπολογία.



Σχήμα 1 – *Λειτουργία του LUIA και σε επίπεδο υλοποίησης.*

3.2 Μέτρα Απόδοσης (Performance Metrics)

Η αποτελεσματικότητα ενός υπολογιστικού συστήματος καθορίζεται από τα κριτήρια, καλούμενα ως μέτρα απόδοσης, που χρησιμοποιούνται κάθε φορά για την αξιολόγηση του και ποικίλουν με τις ανάγκες που καλείται να καλύψει. Για παράδειγμα, κριτήριο αξιολόγησης μπορεί να αποτελέσει ο μέσος χρόνος εκτέλεσης ή το πλήθος των ολοκληρωμένων εργασιών ανάλογα αν το σημαντικό στοιχείο του συστήματος είναι η ταχύτητα εκτέλεσης ή το πλήθος των ολοκληρωμένων εργασιών αντίστοιχα.

Στις παραγράφους που ακολουθούν περιγράφονται τα συνήθης μέτρα απόδοσης καθώς επίσης αυτά που χρησιμοποιούνται για την αξιολόγηση των πειραματικών αποτελεσμάτων.

3.2.1 Συνήθης μέτρα απόδοσης

Γενικά, τα μέτρα απόδοσης μπορούν να τοποθετηθούν σε δύο κατηγορίες: Τα *μέτρα ενδιαφέροντος χρήστη* (*user-oriented measures*) και τα *μέτρα ενδιαφέροντος συστήματος* (*system-oriented measures*).

➢Τα μέτρα ενδιαφέροντος χρήστη περιλαμβάνουν ποσότητες όπως:
i) Το *χρόνο ολοκλήρωσης* (*turnaround time*), που συνήθως αναφέρεται σε ένα batch περιβάλλον και είναι ο χρόνος από την υποβολή μιας εργασίας μέχρι τη στιγμή που γίνονται διαθέσιμα τα αποτελέσματα.

ii) Το *χρόνο απόκρισης* (*response time*), ο οποίος σε ένα πραγματικού χρόνου (real-time) ή αλληλεπιδραστικό (interactive) σύστημα, είναι ο χρόνος από την άφιξη μιας απαίτησης (request) μέχρι την ικανοποίησή της από το σύστημα.

iii) Το *χρόνο αντίδρασης του συστήματος* (*system reaction time*), ο οποίος σε ένα αλληλεπιδραστικό σύστημα είναι ο χρόνος από την άφιξη στο σύστημα μίας απαίτησης, μέχρι τη στιγμή που αυτή θα πάρει το πρώτο time slice εξυπηρέτησης (χρόνος που μπορεί να ταυτίζεται με τον χρόνο παραμονής της απαίτησης στην ουρά του εξυπηρετητή).

Οι παραπάνω ποσότητες συνήθως θεωρούνται τυχαίες μεταβλητές και κατά συνέπεια ασχολούμαστε με την κατανομή τους, τη μέση τιμή, τη διακύμανσή τους και άλλες σχετικές ποσότητες. Ακόμα, μερικές φορές ταξινομούμε τις εργασίες ή τις απαιτήσεις, σε διαφορετικές *κλάσεις προτεραιότητας*. Παράγοντες που καθορίζουν την κλάση προτεραιότητας στην οποία τοποθετείται μία εργασία, είναι το πόσο επείγουσα θεωρείται, η σημασία της, τα χαρακτηριστικά των απαιτήσεών της σε πόρους του συστήματος. Συνήθως υπολογίζουμε τα μέτρα απόδοσης για κάθε κλάση προτεραιότητας χωριστά. *[Γαροφαλάκης, 99]*

➢Τα μέτρα ενδιαφέροντος συστήματος είναι συνήθως:
i) Η *Ρυθμοαπόδοση* (*throughput*), που ορίζεται ως ο μέσος αριθμός εργασιών που εξυπηρετούνται από το σύστημα στη μονάδα του χρόνου. Είναι ένα μέτρο της παραγωγικότητας του συστήματος. Αν οι εργασίες φθάνουν στο σύστημα με τρόπο ανεξάρτητο της κατάστασης του συστήματος, τότε το throughput είναι ίσο με το μέσο ρυθμό αφίξεων, εφόσον το σύστημα μπορεί να εξυπηρετεί τις εργασίες χωρίς να δημιουργούνται συνεχώς αυξανόμενες ουρές από εργασίες που περιμένουν να εξυπηρετηθούν. Στην περίπτωση αυτή το throughput δεν είναι μέτρο απόδοσης, αλλά μέτρο του φορτίου εργασίας του συστήματος. Κατά συνέπεια, το throughput έχει νόημα ως μέτρο απόδοσης, αν, είτε υπάρχει πάντα εργασία που περιμένει εξυπηρέτηση από το σύστημα, ή ο μηχανισμός αφίξεων εργασιών εξαρτάται από την κατάσταση του συστήματος. *[Γαροφαλάκης, 99]*

ii) Η *Χρησιμοποίηση* (*utilization*) ενός πόρου. Σε κάθε χρονική στιγμή ένας πόρος μπορεί να χαρακτηριστεί ως α) *απασχολημένος* (*busy*) όταν μέρος του να διατίθεται για την

εξυπηρέτηση εργασιών ή β) *αδρανής* (*idle*) στην περίπτωση που είναι ελεύθερος εργασιών. Η μέση χρησιμοποίηση ενός πόρου μπορεί να υπολογιστεί από την σχέση $Average\ Resource\ Utilization = \frac{\int_0^T B dt}{T}$ όπου B = 0 ή 1 ανάλογα αν ο πόρος είναι αδρανής ή απασχολημένος αντίστοιχα, dt το χρονικό διάστημα κατά το οποίο παρατηρήθηκε η συγκεκριμένη κατάσταση του πόρου και T ο χρόνος προσομοίωσης. *[Chung, 04]*

3.2.2 Μέτρα απόδοσης συστήματος

Στις παραγράφους που ακολουθούν παρουσιάζονται και σχολιάζονται τα μεγέθη τα οποία ορίστηκαν με σκοπό την μελέτη των πειραματικών αποτελεσμάτων και την αξιολόγηση των εφαρμοζόμενων αλγόριθμων εξισορρόπησης φορτίου. Σημειώνεται ότι στο τέλος κάθε προσομοίωσης το σύστημα παράγει αναφορά με τις τιμές των εν λόγω μεγεθών.

➢ *Χρόνος προσομοίωσης* (*simulation time*) ορίζεται το χρονικό διάστημα που διέρχεται από την γένεση της πρώτης εργασίας μέχρι την ολοκλήρωση όλων των εργασιών.

➢ *Χρόνος επεξεργασίας* (*processing time*) μίας εργασίας ορίζεται ως ο πραγματικός χρόνος που δεσμεύεται από μια CPU για την διεκπεραίωση της. Ο χρόνος αυτός εξαρτάται αποκλειστικά από το μέγεθος της και τα χαρακτηριστικά του επεξεργαστή (ή πόρου) (όπως άλλωστε φαίνεται και από το μοντέλο προσομοίωσης της CPU όπου οι σχέσεις μεταξύ ταχύτητας επεξεργασίας, μονάδων πόρου και μονάδων εργασιών είναι προκαθορισμένες). Έτσι, ο χρόνος επεξεργασίας μίας εργασίας είναι ανεξάρτητος της εφαρμοζόμενης πολιτικής και μπορεί να υπολογιστεί, για τον συγκεκριμένο πόρο, από την ατομική (χωρίς δηλαδή να εκτελούνται άλλες εργασίες) εκτέλεση της εργασίας σε αυτόν.

➢ *Μικτός χρόνος επεξεργασίας* (*mixed processing time*) μίας εργασίας είναι ο χρόνος που διέρχεται από την άφιξη της μέχρι την ολοκλήρωση της στην υπολογιστική μονάδα (υποθέτουμε ότι η εργασία ολοκληρώνεται εξολοκλήρου στον διεκπεραιωτή που αρχικά ανατέθηκε -nonpreemptive transfer-). Το μέγεθος αυτό εξαρτάται από το χρονικό διάστημα και το ποσοστό του πόρου που διατίθενται για την εκτέλεση της εργασίας αυτής, καθώς επίσης και από την αναμονή της στην μονάδα μέχρι τη στιγμή που πάρει το πρώτο time slice εξυπηρέτησης. Για παράδειγμα, ο μικτός χρόνος επεξεργασίας μιας εργασίας με χρήση του 50% του πόρου μίας μονάδας είναι διπλάσιος από αυτόν που απαιτείται με χρήση του 100% του πόρου αυτού (θεωρώντας ότι η εξυπηρέτηση της εργασίας στην μονάδα ξεκινάει με την άφιξη της σε αυτή). Σημειώνεται ότι κατά διάρκεια του εν λόγω χρόνου ο πόρος είναι *απασχολημένος* (busy), αφού μέρος αυτού διατίθεται στην εκτέλεση εργασιών.

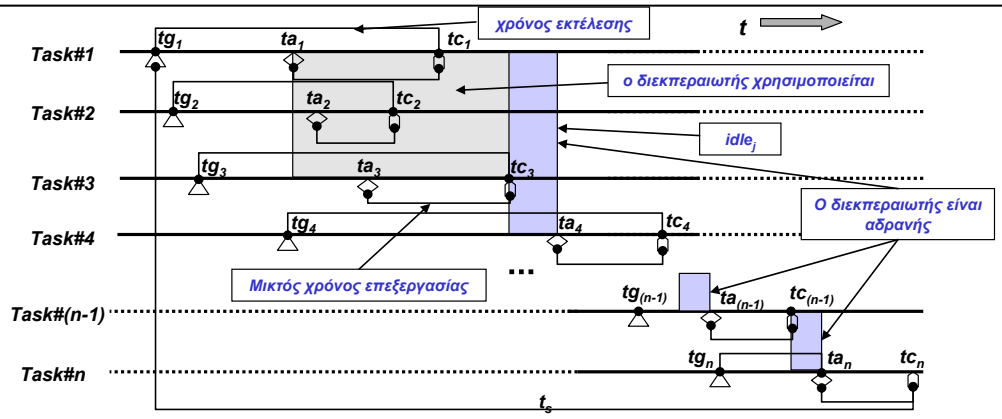
➢ *Χρόνος εκτέλεσης* (*execution time*) μιας εργασίας καλείται το χρονικό διάστημα που διέρχεται από την γένεση της μέχρι την ολοκλήρωση της (κάτι ανάλογο με τον χρόνο απόκρισης και ολοκλήρωσης).

Για τα προαναφερθέντα μέτρα απόδοσης ορίζονται και οι αντίστοιχες μέσες τιμές στο σύνολο των ολοκληρωμένων εργασιών. Έτσι, ορίζονται ο μέσος χρόνος επεξεργασίας (mean/average processing time), ο μέσος μικτός χρόνος επεξεργασίας (mean/average mixed processing time) και ο μέσος χρόνος εκτέλεσης (mean/average execution time) των εργασιών σε μία μονάδα.

➢ Το *επί τοις εκατό ποσοστό αδράνειας ενός διεκπεραιωτή* (*%idle time*) ορίζεται ως το % του χρόνου προσομοίωσης όπου ο διεκπεραιωτής είναι ελεύθερος εργασιών.

➢ *Αριθμός ολοκληρωμένων εργασιών* (*completed tasks*) εκφράζει το πλήθος των εργασιών που εξυπηρετήθηκαν από μια υπολογιστική μονάδα.

Στο Σχήμα 2 απεικονίζεται τα σημαντικότερα μέτρα απόδοσης μαζί με κάποιες διευκρινίσεις. Σημειώνεται ότι τα παραπάνω μεγέθη είναι δυνατόν να γενικευτούν και σε επίπεδο συστήματος. Για παράδειγμα το % ποσοστό αδράνειας του συστήματος ορίζεται ως το μέγεθος που εκφράζει το % ποσοστό του χρόνου επί του χρόνου προσομοίωσης, όπου το σύστημα είναι αδρανές.



Στοιχειώδης μεγέθη	
tg_i	Χρόνος παραγωγής
ta_i	Χρόνος άφιξης
tc_i	Χρόνος ολοκλήρωσης
$tm_i = tc_i - ta_i$	Μικτός χρόνος επεξεργασίας
$te_i = tc_i - tg_i$	Χρόνος εκτέλεσης
$tp_i = tu_i/(cpu \times sp)$	Ο χρόνος επεξεργασίας της εργασίας i, μεγέθους tu_i, στον διεκπεραιωτή με μονάδες πόρου cpu και ταχύτητα επεξεργασίας sp

Μέτρα απόδοσης διεκπεραιωτή	
$\frac{\sum_{j=1}^{m} idle_j}{t_s} \times 100$	% ποσοστό αδράνειας διεκπεραιωτή
$\frac{\sum_{i=1}^{n} tm_i}{n} \quad \frac{\sum_{i=1}^{n} tp_i}{n}$	Μέσος -μικτός- χρόνος επεξεργασίας
$\frac{\sum_{i=1}^{n} te_i}{n}$	Μέσος χρόνος εκτέλεσης
t_S	Χρόνος προσομοίωσης

Σχήμα 2 – *Μέτρα απόδοσης.*

Από την σύγκριση του μέσου χρόνου επεξεργασίας και του μέσου μικτού χρόνου επεξεργασίας (είτε σε επίπεδο επεξεργαστή, είτε σε επίπεδο συστήματος) μπορούν να εξαχθούν συμπεράσματα κατά πόσο ο επεξεργαστής ή το σύστημα μπορεί να ανταποκριθεί ή όχι στο επιβαλλόμενο φορτίο. Μικρή απόκλιση των μεγεθών αυτών υποδεικνύει ότι ο πόρος της μονάδας ή οι πόροι του συστήματος κατανέμονται σε μικρό πλήθος εργασιών με αποτέλεσμα η απόδοση του, για το επιβαλλόμενο φορτίο, να είναι πιθανώς ικανοποιητική.

Η συνεκτίμηση του μέσου χρόνου εκτέλεσης και του μέσου μικτού χρόνου επεξεργασίας μπορεί να αποκαλύψει κατά πόσο οι πολιτικές του εφαρμοζόμενου αλγορίθμου συγκλίνουν ταχέως ή όχι, και να αποτελέσει «μέτρο» της ταχύτητας επικοινωνίας μεταξύ των οντοτήτων του συστήματος.

Τέλος, μεγάλη τιμή του μεγέθους %idle αποτελεί ένδειξη ότι α) η μονάδα είναι σε θέση να δεχθεί επιπλέον φορτίο χωρίς να προκληθεί σημαντική επιβάρυνση στον χρόνο εκτέλεσης των εργασιών, αφού για μεγάλο χρονικό διάστημα ο πόρος της παραμένει ανεκμετάλλευτος, ή β) η μονάδα δεν συμμετέχει σημαντικά στην εξυπηρέτηση των εργασιών με αποτέλεσμα να υπάρχει δυνατότητα απομάκρυνσής της και χρήση της σε διαφορετική δραστηριότητα. Βέβαια για την διεξαγωγή ορθών συμπερασμάτων θα πρέπει να συνεκτιμηθούν και άλλοι παράγοντες όπως το ποσοστό των ολοκληρωμένων εργασιών της μονάδας επί του συνόλου του συστήματος. Έτσι στην πρώτη περίπτωση, το υψηλό ποσοστό αδράνειας μπορεί να μην οφείλεται στην ικανότητα της μονάδας να εξυπηρετεί ταχέως τις εργασίες, αλλά στον περιορισμένο αριθμό εργασιών που ανατίθενται σε αυτή, βάσει του εφαρμοζόμενου

αλγόριθμου εξισορρόπησης φορτίου. Οπότε, επιβάρυνση της μονάδας αυτής με επιπλέον φορτίο θα οδηγούσε σε μείωση της απόδοσης του συστήματος. Ενώ στην δεύτερη περίπτωση, το υψηλό ποσοστό αδράνειας μπορεί να αποδίδεται στην υψηλή ταχύτητα επεξεργασίας της μονάδας με αποτέλεσμα η απομάκρυνσή της να οδηγήσει επίσης σε μείωση της αποτελεσματικότητας του συστήματος.

3.3 Επίδρασης της ταχύτητας επικοινωνίας

Η απόδοση ενός υπολογιστικού συστήματος μπορεί να επηρεαστεί σημαντικά από την ταχύτητα επικοινωνίας μεταξύ των οντοτήτων που το απαρτίζουν, ιδιαίτερα στην περιπτώσεις που το πρωτόκολλο επικοινωνίας που χρησιμοποιούν απαιτεί την ανταλλαγή μεγάλου πλήθους μηνυμάτων. Από την εφαρμογή των αλγόριθμων RR και LUIA στο πείραμα που ακολουθεί, αποκαλύπτεται η στενή σχέση μεταξύ απόδοσης και ταχύτητας επικοινωνίας καθώς επίσης και ορισμένα βασικά χαρακτηριστικά της συμπεριφοράς των αλγόριθμων αυτών και γενικότερα των κατηγοριών στις οποίες ανήκουν.

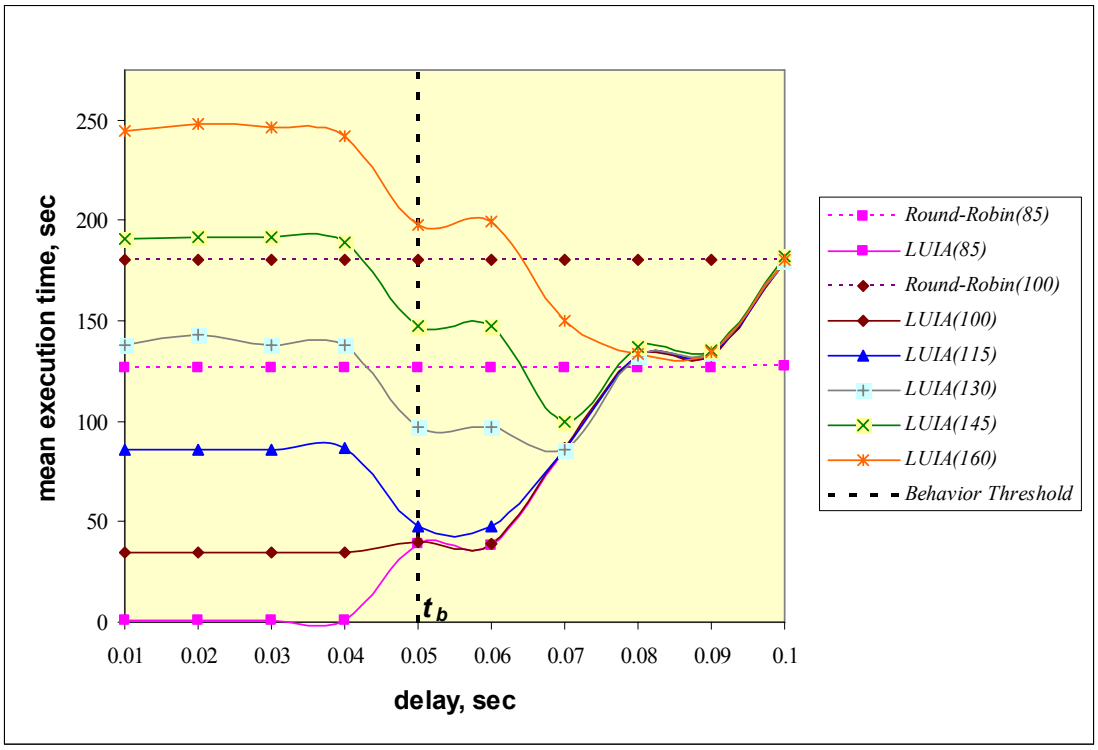
Οι παράμετροι του προς προσομοίωση υπολογιστικού συστήματος ορίζονται στον Πίνακα 1. Οι τιμές της παραμέτρου delay και του μεγέθους των εργασιών μεταβάλλονται από 0.01 έως 0.1 sec και 85 έως 160 μονάδες εργασίας (μ.ε.) με βήμα αύξησης 0.01 sec και 15 μ.ε. αντίστοιχα.

Πίνακας 1 – Παράμετροι προσομοίωσης.

Πλέγμα διεκπεραιωτών				
Αριθμός Διεκπεραιωτών	***Μονάδες πόρου (cpu)***	***Ταχύτητα επεξεργασίας (sp)***	***Τοπολογικός συντελεστής (tf)***	***Καθυστέρηση ανά μονάδα τοπ. συντ.(delay)(σε sec)***
3	100	1	1	0.01...0.1 βήμα: 0.01
2	100	3	1	
Εργασίες				
Κατανομή παραγωγής εργασιών	σταθερός ρυθμός: 10 εργασίες/sec			
Κατανομή μονάδων εργασιών	μονάδες εργασίας (μ.ε.): 85...160/βήμα: 15			
Κατανομή προτεραιοτήτων εργασιών	σταθερή προτεραιότητα: 1			
Διάστημα παραγωγής εργασιών	300 sec			
Συντελεστές Συνάρτησης Αποτελεσματικότητας	*speed_cf* = 1, *topology_cf* = 0, *tload_cf* = -1, *ntasks* = 0			

Κρίσιμες παράμετροι προσομοίωσης:
Η επιλογή του ρυθμού παραγωγής των εργασιών και της καθυστέρησης ανά μονάδα τοπολογικού συντελεστή έγινε με τέτοιο τρόπο ώστε: για τον αλγόριθμο *RR*, ο χρόνος εκτέλεσης της μικρότερης εργασίας στην υπολογιστική μονάδα με την χαμηλότερη ταχύτητα επεξεργασίας να είναι μεγαλύτερος από το μεσοδιάστημα άφιξης των εργασιών στην μονάδα αυτή.
Έτσι, θεωρώντας ότι η μικρότερη εργασία, 85 μ.ε. εκτελείται ατομικά στην μονάδα με την μικρότερη sp (μία μονάδα), ο χρόνος εκτέλεσης της εργασίας αυτής είναι $\frac{80}{100 \times 1} = 0.85\,\text{sec}$
Το μεσοδιάστημα άφιξης εργασιών είναι:
(συνολικός αριθμός υπολογιστικών μονάδων) × (ρυθμό παραγωγής εργασιών)$^{-1}$ = $5 \times 0.1 = 0.5\,\text{sec}$, με αποτέλεσμα η προαναφερθείσα συνθήκη να ικανοποιείται αφού 0.85 > 0.5. Σε αντίθετη περίπτωση, το σύστημα είναι σε θέση να πραγματοποιεί την εκτέλεση των εργασιών ατομικά και κατά συνέπεια τα πειραματικά αποτελέσματα δεν είναι αξιολογήσιμα (ο Round-Robin γίνεται πλήρως ανταγωνιστικός και δεν είναι ευδιάκριτη η σύγκριση με τον *LUIA*).

Τα αποτελέσματα της προσομοίωσης απεικονίζονται στις καμπύλες του Διαγράμματος 1, όπου περιγράφεται η μεταβολή του μέσου χρόνου εκτέλεσης των εργασιών (*mean execution time*) συναρτήσει της καθυστέρησης ανά μονάδα τοπολογικού συντελεστή, για διάφορα μεγέθη εργασιών και για τους εφαρμοζόμενους αλγόριθμους LUIA και RR. Για παράδειγμα, η καμπύλη LUIA(160) περιγράφει την μεταβολή του μέσου χρόνου εκτέλεσης με την εφαρμογή του αλγόριθμου LUIA και μέγεθος κάθε εργασίας 160 μ.ε. Σημειώνεται ότι οι καμπύλες που προκύπτουν από την εφαρμογή του RR και απεικονίζουν απόδοση συστήματος πολύ χειρότερη από τις αντίστοιχες καμπύλες (για το ίδιο δηλαδή μέγεθος εργασίας) που προκύπτουν από την εφαρμογή του LUIA, αποκλείονται από το διάγραμμα για λόγους απλότητας.



Διάγραμμα 1 – *Μεταβολή του μέσου χρόνου εκτέλεσης συναρτήσει της καθυστέρησης ανά μονάδα τοπολογικού συντελεστή, για διάφορα φορτία και τους εφαρμοζόμενους αλγορίθμους.*

Με την εφαρμογή του LUIA, η απόδοση του συστήματος επηρεάζεται σημαντικά από τον χρόνο παραμονής (*PQ*) μιας εργασίας στην ουρά του συντονιστή, δηλαδή από τον χρόνο που μεσολαβεί από την γένεση της μέχρι και την εύρεση των κατάλληλου διεκπεραιωτή για την εκτέλεση της. Ο χρόνος αυτός εν γένει, εξαρτάται από το μεσοδιάστημα γεννήσεως των εργασιών ($(\textit{ρυθμός παραγωγής})^{-1}$) (*SG*), και τον χρόνο που απαιτείται για την ολοκλήρωση της πολιτικής πληροφόρησης, έστω *TP*.

Η σχέση μεταξύ των μεγεθών *TP* και *SG* καθορίζει την συμπεριφορά του υπολογιστικού συστήματος. Για παράδειγμα, σύμφωνα με το μοντέλο προσομοίωσης και για delay=0.25sec, ο *SG* είναι διπλάσιος του *TP* αφού η λήξης της πολιτικής πληροφόρησης σηματοδοτείται με την λήψη και του τελευταίου μηνύματος καταστάσεως το οποίο αναπόφευκτα προέρχεται από την πιο απομακρυσμένη μονάδα. Επειδή *SG*>*TP* η ουρά αναμονής του συντονιστή είναι κενή, γεγονός που έχει σημαντική επίπτωση στην απόδοση του συστήματος όπως θα δούμε στην συνέχεια.

Χαρακτηριστικό μέγεθος για το υπό εξέταση σύστημα είναι η τιμή της καθυστέρησης ανά μονάδα τοπολογικού συντελεστή για την οποία η ισότητα *TP* = *SG* επαληθεύεται. Η τιμή αυτή καλείται «οριακός χρόνος συμπεριφοράς», συμβολίζεται με t_b και για τις συγκεκριμένες παραμέτρους προσομοίωσης λαμβάνει την τιμή 0.05 sec. Το t_b καθορίζει την συμπεριφορά του υπολογιστικού συστήματος αφού για:

➢ *SG* > *TP* η ουρά αναμονής του συντονιστή είναι κενή με αποτέλεσμα η εφαρμογή της πολιτικής πληροφόρησης για κάθε νέα εργασία να πραγματοποιείται άμεσα σε χρόνο *TP*. Στην περίπτωση αυτή, ο χρόνος εκτέλεσης εξαρτάται κυρίως από α) τον ρυθμό παραγωγής και το μέγεθος των εργασιών και β) τα χαρακτηριστικά της υπολογιστικής μονάδας, με την ταχύτητα επικοινωνίας να έχει δευτερεύοντα ρόλο. Πράγματι, από το Διάγραμμα 1 φαίνεται ότι, για delay < t_b οι μέσοι χρόνοι εκτέλεσης των εργασιών, για συγκεκριμένο μέγεθος εργασίας, είναι ουσιαστικά ίδιοι.

➢ *SG* < *TP*, μία εργασία αποστέλλεται στον βέλτιστο διεκπεραιωτή σε χρόνο μεγαλύτερο από τον *TP*. Αυτό οφείλεται στο γεγονός ότι, η πολιτική πληροφόρησης δεν εφαρμόζεται άμεσα με την δημιουργία μιας νέας εργασίας, καθώς στην ουρά του συντονιστή ήδη αναμένουν να εξυπηρετηθούν εργασίες με μικρότερο χρόνο παραγωγής από την εργασία αυτή. Στην περίπτωση αυτή, η αποτελεσματικότητα του συστήματος ελέγχεται από το *TP* και μέγεθος του φορτίου.

Συγκεκριμένα, μικρές τιμές *TP* και υψηλά φορτία μπορούν να οδηγήσουν στην μείωση του μέσου χρόνου εκτέλεσης καθώς η αύξηση του χρόνου ολοκλήρωσης της πολιτικής πληροφόρησης οδηγεί ουσιαστικά στην ελάττωση του μεσοδιαστήματος αφίξεως των εργασιών στους διεκπεραιωτές. Αυτό έχει σαν αποτέλεσμα την αύξηση του ποσοστού του πόρου ενός διεκπεραιωτή που διατίθεται για την ολοκλήρωση μίας εργασίας στην μονάδα του χρόνου και κατά συνέπεια την μείωση του μικτού χρόνου επεξεργασίας. Η μείωση του μικτού χρόνου επεξεργασίας προκαλεί με την σειρά της μείωση του χρόνου εκτέλεσης, εφόσον το κέρδος από την μείωση αυτή δεν υπερκεράζεται από το κόστος της αύξησης του χρόνου παραμονής των εργασιών στην ουρά του συντονιστή. Χαρακτηριστικά παραδείγματα αποτελούν οι καμπύλες LUIA(130) και LUIA(145) για 0.05<*delay*<0.07 και η LUIA(160) για 0.05<*delay*<0.08.

Στην περίπτωση μικρών εργασιών, η επίδραση του *TP* στην μείωση του μικτού χρόνου επεξεργασίας είναι ανεπαίσθητη καθώς οι υπολογιστικές μονάδες ήδη δύνανται να τις ολοκληρώσουν ταχέως. Ταυτόχρονα, ο χρόνος παραμονής των εργασιών στην ουρά του συντονιστή είναι σημαντικός προκαλώντας αύξηση των αντίστοιχων χρόνων εκτέλεσης τους, όπως χαρακτηριστικά φαίνεται στην καμπύλη LUIA(85). Βέβαια υπάρχουν περιπτώσεις όπου μεταξύ μικτών χρόνων επεξεργασίας και χρόνων παραμονής δημιουργείται ένα είδος ισορροπίας έτσι ώστε τελικά ο χρόνος εκτέλεσης να παραμένει σταθερός, όπως απεικονίζεται στην LUIA(100) για 0.05<*delay*<0.07.

Μεγάλες τιμές *TP*, ανεξαρτήτως φορτίου, οδηγούν στην αύξηση του χρόνου εκτέλεσης καθώς η εφαρμογή της πολιτικής πληροφόρησης αποτελεί το ελέγχον στάδιο της διαδικασίας. Παράδειγμα αποτελεί όλοι η σειρά καμπυλών του LUIA, για *delay*>0.07 sec. Για πολύ μεγάλες τιμές *delay* ο χρόνος *TP* και κατά συνέπεια ο *PQ* ελέγχει εξολοκλήρου την διαδικασία και όλες οι καμπύλες εκφυλίζονται σε μία.

Από την μελέτη του Διαγράμματος 1 επιπλέον προκύπτει ότι:
i) Σε συστήματα όπου οι αποφάσεις λαμβάνονται με την ανταλλαγή πολλών μηνυμάτων η επίδραση της ταχύτητας επικοινωνίας παίζει καθοριστικό ρόλο. Από την καμπύλη LUIA(145) ενδεικτικά φαίνεται ότι, αύξηση του *delay* κατά 0.01 sec μπορεί να προκαλέσει μεταβολή του μέσου χρόνου εκτέλεσης έως και 32% περίπου.

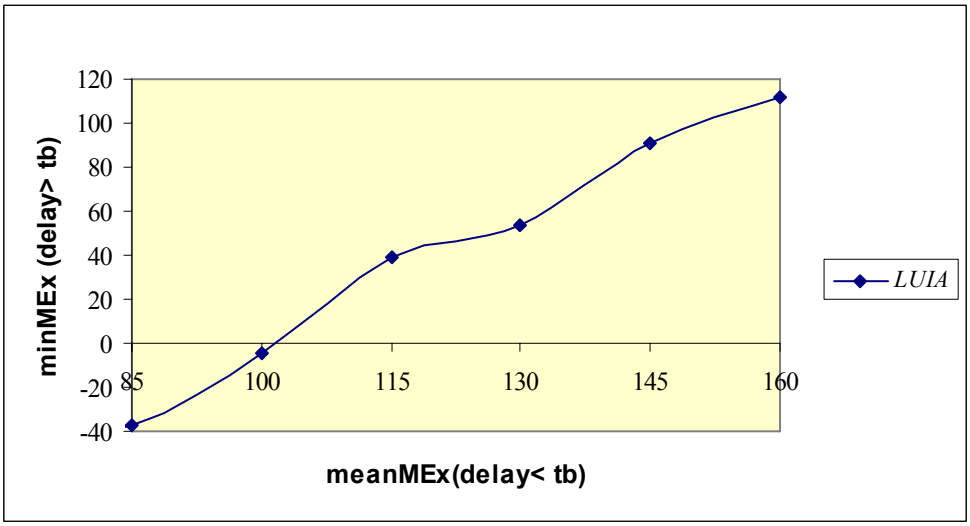
ii) Στους στατικούς αλγόριθμους, όπως ο RR, ο μέσος χρόνος εκτέλεσης των εργασιών δεν επηρεάζεται σημαντικά από την ταχύτητα επικοινωνίας μεταξύ των μονάδων καθώς δεν εμπλέκονται ανταλλαγές μηνυμάτων μεταξύ αυτών.

iii) Ένας δυναμικός αλγόριθμος μπορεί να είναι αποτελεσματικότερος από έναν στατικό με εξαίρεση τις περιπτώσεις όπου η ταχύτητα επικοινωνίας μεταξύ των οντοτήτων δεν βοηθάει στην ολοκλήρωση των εφαρμοζόμενων πολιτικών σε ικανοποιητικό χρόνο.

Για παράδειγμα, ο αλγόριθμός LUIA υπερτερεί του RR σχεδόν για όλο το εύρος τιμών της *delay*. Εξαίρεση αποτελούν κάποιες περιοχές της καμπύλης LUIA(85) όπου το υψηλό κόστος επικοινωνίας (*delay*>0.7) σε συνδυασμό με το μικρό μέγεθος του επιβαλλόμενου φορτίου φέρουν τον RR αποτελεσματικότερο. Οι Eager at al *[Eager, 86b]* κατάληξαν επίσης ότι, εξαιρετικά απλοί αλγόριθμοι μπορεί να αποβούν αποτελεσματικότεροι από σαφώς πιο πολύπλοκους που εκμεταλλεύονται περισσότερη πληροφορία του συστήματος.

iv) Αύξηση της αποτελεσματικότητας του LUIA με την αύξηση του *delay*. Χαρακτηριστικό παράδειγμα αποτελούν οι καμπύλες LUIA(115) έως LUIA(160) όπου ο ελάχιστος μέσος χρόνος εκτέλεσης των εργασιών συμβαίνει στην περιοχή όπου το *delay* έχει σημαντική επίδραση στο σύστημα (*delay* > t_b). Στον LUIA, η συμπεριφορά αυτή επιτυγχάνεται με την «ελεγχόμενη» αύξηση του χρόνου ολοκλήρωσης της πολιτικής πληροφόρησης, η οποία έχει σαν αποτέλεσμα την διάθεση περισσοτέρων μονάδων πόρου ανά εργασία στην μονάδα του χρόνου, όπως ήδη έχει αναφερθεί.

v) Αύξηση της επίδρασης του delay στην βελτίωση της αποτελεσματικότητας του LUIA με την αύξηση του φορτίου. Αυτό μπορεί να επιβεβαιωθεί με την σύγκριση των μεταβολών της απόδοσης του συστήματος που παρατηρούνται, με την εφαρμογή του αλγορίθμου LUIA, υπό συνθήκες εξάρτησης και ανεξαρτησίας από το delay, για διάφορα φορτία. Συγκεκριμένα, κατασκευάζεται το Διάγραμμα 2 όπου αποτυπώνεται η μεταβολή της διαφοράς του μέσου όρου των μέσων χρόνων εκτέλεσης για περιοχές του delay (*meanMEx(delay*< t_b*)*) όπου η επίδραση της ταχύτητας δεν είναι σημαντική, με τον ελάχιστο μέσο χρόνο εκτέλεσης στην περιοχή της καμπύλης για delay> t_b (*minMEx (delay*> t_b*)*), συναρτήσει του φορτίου.



Διάγραμμα 2 - *Μεταβολή της διαφοράς του μέσου όρου των μέσων χρόνων εκτέλεσης για delay* < t_b *με τον ελάχιστο μέσο χρόνο εκτέλεσης για delay* > t_b *, συναρτήσει του επιβαλλόμενου φορτίου.*

Από το παραπάνω διάγραμμα παρατηρείται ότι, η επίδραση του delay στην βελτίωση της απόδοσης του συστήματος αυξάνεται με το επιβαλλόμενο φορτίο. Αύξηση της αποτελεσματικότητας ενός αλγόριθμου με το μέγεθος των εργασιών έχει διαπιστωθεί και σε παλαιότερες εργασίες όπως στην *[Mirchandaney, 89]*.

3.4 Επίδραση της ομοιογένειας του συστήματος

Η ομοιογένεια ενός συστήματος, δηλαδή κατά πόσο τα χαρακτηριστικά των υπολογιστικών μονάδων που την απαρτίζουν είναι της ίδιας δυναμικότητας, μπορούν να επηρεάσουν σημαντικά την αποτελεσματικότητα ενός αλγορίθμου και κατεπέκταση την απόδοση του συστήματος.

Αλγόριθμοι που βελτιώνουν την απόδοση ενός ανομοιογενούς συστήματος, δεν παρουσιάζουν την ίδια αποτελεσματικότητα σε ομοιογενή, καθώς η σχεδίαση τους δεν προβλέπει την αξιοποίηση των ιδιαίτερων χαρακτηριστικών των υπολογιστικών μονάδων. Στις περιπτώσεις αυτές η χρήση ενός απλού αλγόριθμου είναι προτιμότερη.

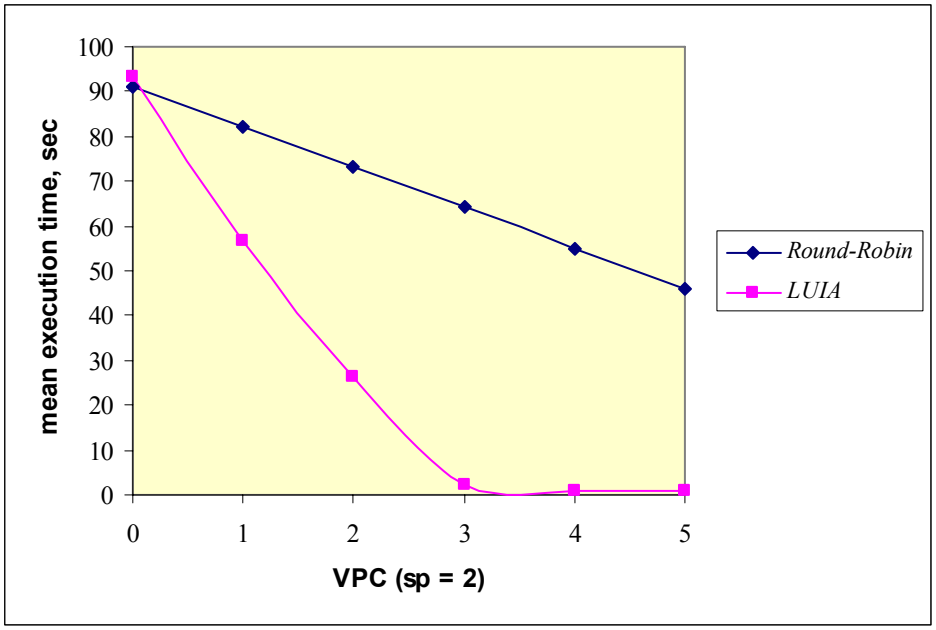
Τα παραπάνω συμπεράσματα μπορούν εύκολα να επιβεβαιωθούν από τα πειραματικά αποτελέσματα που λαμβάνουμε με την προσομοίωση των αλγορίθμων RR και LUIA σε ένα υπολογιστικό σύστημα με τις παραμέτρους που περιγράφονται στον Πίνακα 2.

Πίνακας 2 – *Παράμετροι προσομοίωσης.*

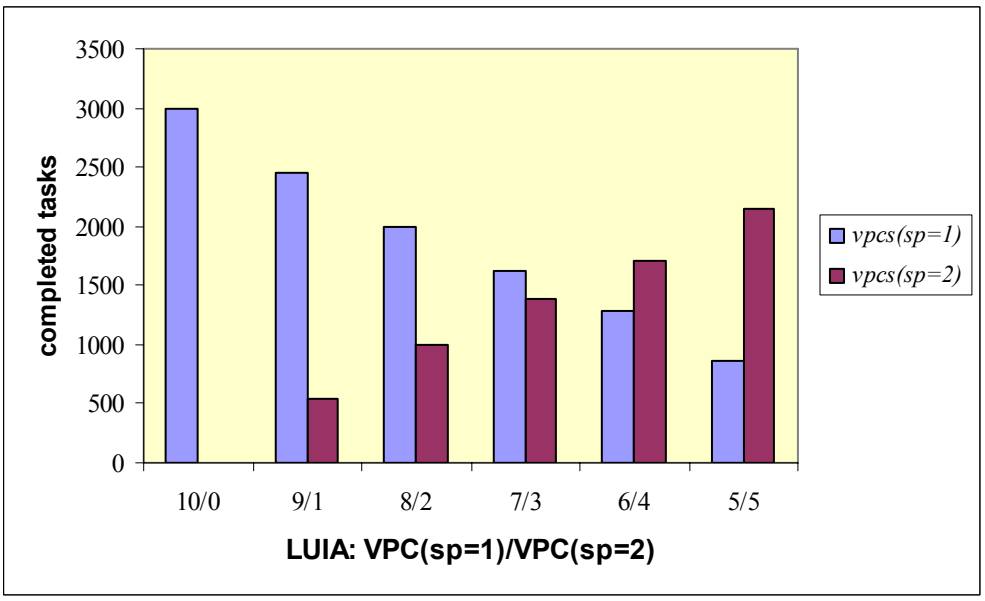
Πλέγμα διεκπεραιωτών				
Αριθμός Διεκπεραιωτών	***Μονάδες πόρου (cpu)***	***Ταχύτητα επεξεργασίας (sp)***	***Τοπολογικός συντελεστής (tf)***	***Καθυστέρηση ανά μονάδα τοπ. συντ.(delay)(σε sec)***
5	100	1	1	0
5	100	1...2	1	
Εργασίες				
Κατανομή παραγωγής εργασιών	Ρυθμός παραγωγής: 10 εργασίες/sec			
Κατανομή μονάδων εργασιών	Σταθερές μονάδες εργασίας: 130			
Κατανομή προτεραιοτήτων εργασιών	Σταθερή προτεραιότητα: 1			
Διάστημα παραγωγής εργασιών	300 sec			
Συντελεστές Συνάρτησης Αποτελεσματικότητας	*speed_cf* = 1, *topology_cf* = 0, *tload_cf* = -1, *ntasks* = 0			
Κρίσιμες παράμετροι προσομοίωσης: Για την καλύτερη μελέτη της επίδρασης της ταχύτητας επεξεργασίας ως μέτρο ανομοιογένειας του συστήματος, απομακρύνουμε την επίδραση της ταχύτητας επικοινωνίας μεταξύ των οντοτήτων, θεωρώντας ότι η καθυστέρηση ανά μονάδα τοπολογικού συντελεστή είναι μηδενική.				

Στο παρόν σύστημα, ορίζουμε ως μέτρο ανομοιογένειας την διαφοροποίηση των υπολογιστικών μονάδων ως προς την ταχύτητα επεξεργασίας τους, sp. Θεωρούμε επίσης ότι η ανομοιογένεια του συστήματος αυξάνεται με την διαφοροποίηση των ταχυτήτων επεξεργασίας των υπολογιστικών μονάδων έως και στο μισό του πληθυσμού τους. Έτσι, για καθένα ένα από τους LUIA και RR εκτελούμε έξι διαδοχικά πειράματα, όπου σε κάθε ένα από αυτά οι διεκπεραιωτές με ταχύτητα επεξεργασίας μεγέθους ένα, αυξάνονται κατά μία μονάδα.

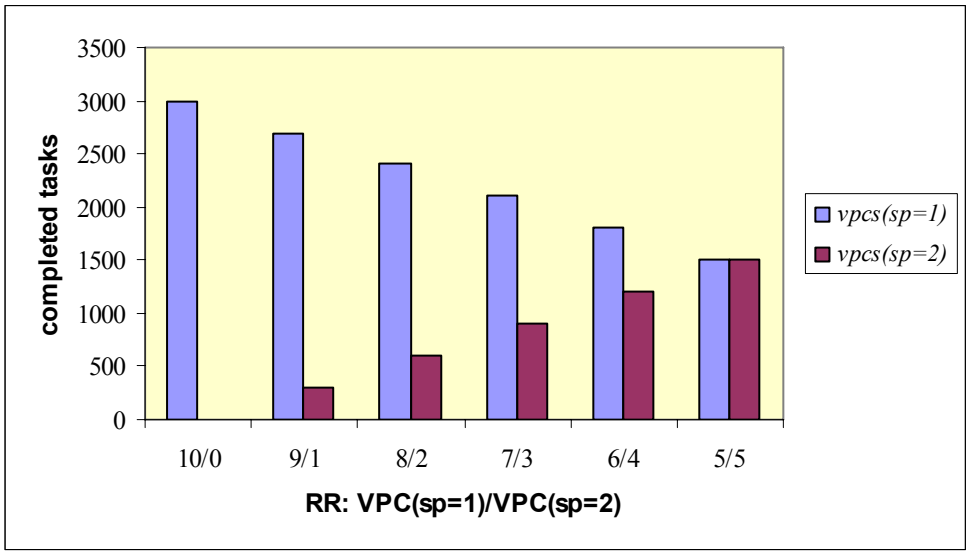
Τα πειραματικά αποτελέσματα συνοψίζονται στα διαγράμματα 3, 4 και 5 όπου απεικονίζεται η μεταβολή του μέσου χρόνου εκτέλεσης των εργασιών συναρτήσει α) του πλήθους των υπολογιστικών μονάδων με ταχύτητα επεξεργασίας μεγέθους δύο και β) του πλήθους των ολοκληρωμένων εργασιών ανά διεκπεραιωτή, συναρτήσει της ανομοιογένειας του συστήματος για τους LUIA και RR.



Διάγραμμα 3 - *Μεταβολή του μέσου χρόνου εκτέλεσης συναρτήσει του πλήθους των υπολογιστικών μονάδων με ταχύτητα επεξεργασίας μεγέθους δύο, για τους αλγόριθμους RR και LUIA.*



Διάγραμμα 4 - *Μεταβολή των ολοκληρωμένων εργασιών ανά υπολογιστική μονάδα, συναρτήσει της ανομοιογένειας του συστήματος, για τον αλγόριθμο LUIA.*



Διάγραμμα 5 - *Μεταβολή των ολοκληρωμένων εργασιών ανά υπολογιστική μονάδα, συναρτήσει της ανομοιογένειας του συστήματος, για τον αλγόριθμο RR.*

Από την μελέτη του Διαγράμματος 3 παρατηρείται:
i) Αύξηση της αποτελεσματικότητας του LUIA με την αύξηση της ανομοιογένειας του συστήματος. Ο LUIA λαμβάνει υπόψη την ταχύτητα επεξεργασίας των μονάδων με αποτέλεσμα να αξιοποιεί την παρουσία τους στο σύστημα, όπως άλλωστε φαίνεται στο διάγραμμα 4.

ii) Αύξηση της αποτελεσματικότητας του RR σε σαφώς χαμηλότερα επίπεδα από ότι στον LUIA. Ο RR δεν λαμβάνει υπόψη τα χαρακτηριστικά των υπολογιστικών μονάδων, κατά συνέπεια η επίδραση της διαφοροποίησης των ταχυτήτων επεξεργασίας στην πολιτική τοποθεσίας να είναι περιορισμένη. Σε αντίθεση με τον LUIA, οι εργασίες μοιράζονται εξίσου σε όλους τους διεκπεραιωτές, όπως φαίνεται στο διάγραμμα 5.

iii) Στην περίπτωση ομοιογενούς συστήματος οι δύο αλγόριθμοι φαίνεται να είναι εξίσου αποδοτικοί.

3.5 Πρόβλημα παρωχημένης πληροφορίας (out of date information problem)

Ένα από τα σημαντικά προβλήματα των αλγόριθμων εξισορρόπησης φορτίου είναι αυτό της *παρωχημένης πληροφορίας* (*out of date information problem*), το οποίο εμφανίζεται σε αλγόριθμους όπου η λήψη των αποφάσεων στηρίζεται σε πληροφορίες σχετικές με την κατάσταση του συστήματος και γενικότερα σε πληροφορίες που μεταβάλλονται με τον χρόνο. Το πρόβλημα αυτό προκαλείται από την χρονική διαφορά μεταξύ καταγραφής και χρήσης της πληροφορίας, με αποτέλεσμα οι αποφάσεις του αλγόριθμου να λαμβάνονται βάσει πληροφοριών που πιθανόν απεικονίζουν παλαιότερη και όχι την σύγχρονη κατάσταση του συστήματος.

Για παράδειγμα, κατά την εφαρμογή του αλγόριθμου LUIA ο συντονιστής αποστέλλει αιτήσεις στις μονάδες του συστήματος για καταγραφή και αποστολή των καταστάσεων τους. Αν η καταγραφή της κατάστασης μιας μονάδας πραγματοποιείται την χρονική στιγμή t_1 και ο συντονιστής λαμβάνει γνώση για την κατάσταση αυτή την χρονική στιγμή t_2, τότε μεταξύ του διαστήματος t_2-t_1 υπάρχει μεγάλη πιθανότητα η κατάσταση της μονάδας αυτής να έχει μεταβληθεί.

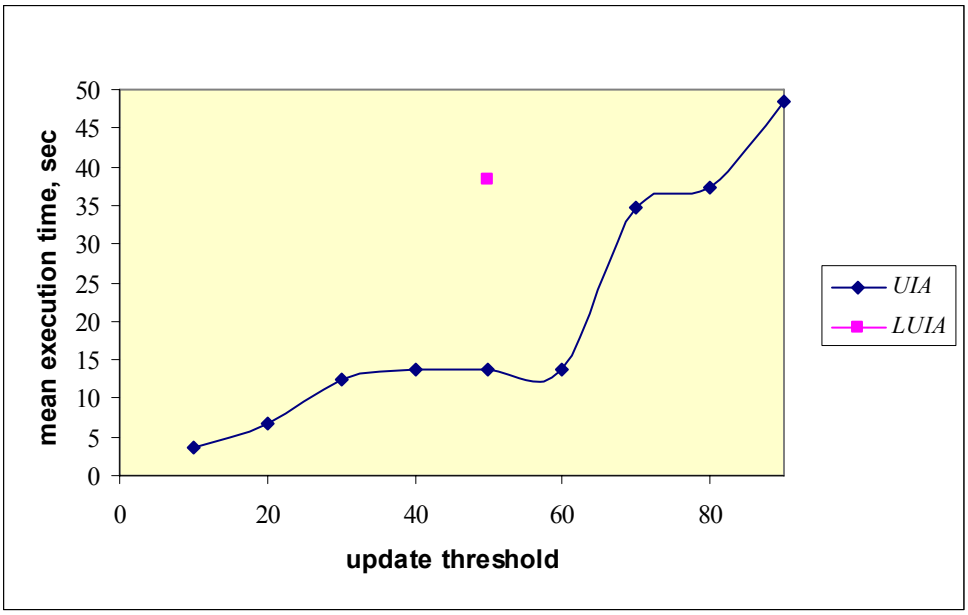
Είναι προφανές ότι το πρόβλημα της παρωχημένης πληροφορίας γίνεται εντονότερο με την μείωση της ταχύτητας επικοινωνίας και τον ρυθμό μεταβολής της κατάσταση του συστήματος ο οποίος μπορεί να εξαρτάται από πολλούς παραμέτρους (π.χ. ρυθμός παραγωγής εργασιών, ρυθμός εξυπηρέτησης εργασιών, ταχύτητα επεξεργασίας των μονάδων κ.α.).

Για την μελέτη του εν λόγω προβλήματος πραγματοποιούμε προσομοίωση συστήματος με εφαρμογή του αλγόριθμου UIA και χαρακτηριστικά που περιγράφονται στον Πίνακα 3.

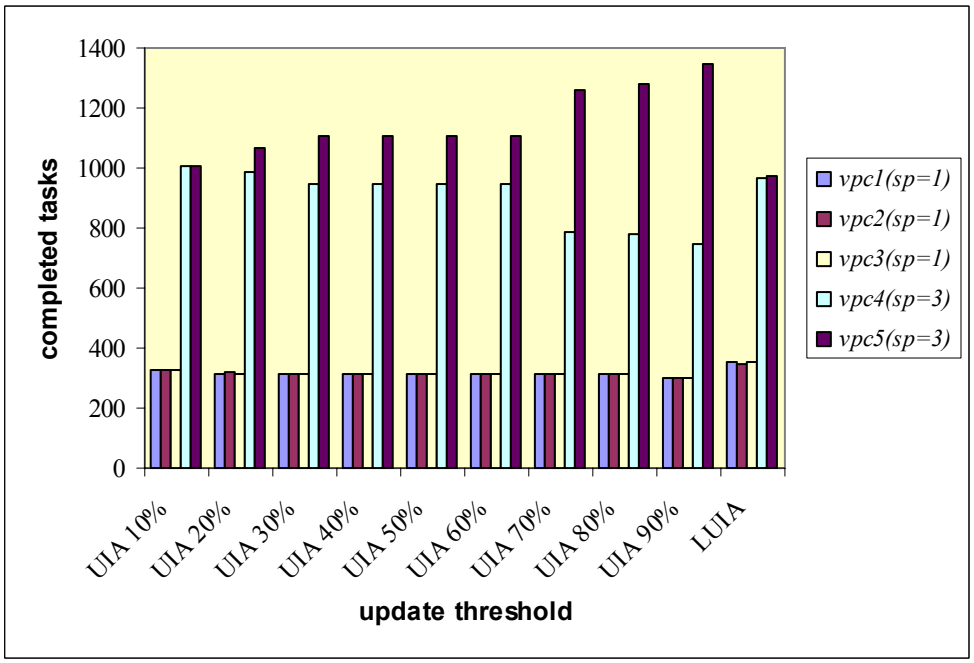
Πίνακας 3 – Παράμετροι προσομοίωσης.

Πλέγμα διεκπεραιωτών				
Αριθμός Διεκπεραιωτών	***Μονάδες πόρου (cpu)***	***Ταχύτητα επεξεργασίας (sp)***	***Τοπολογικός συντελεστής (tf)***	***Καθυστέρηση ανά μονάδα τοπ. συντ. (delay) (σε sec)***
3	100	1	1	0.05
2	100	3	1	
Εργασίες				
Γεννήτρια μεσοδιάστηματος γεννήσεων	Ρυθμός παραγωγής: 10 εργασίες/sec			
Γεννήτρια μονάδων εργασιών	Σταθερές μονάδες εργασίας: 90			
Γεννήτρια προτεραιοτήτων	Σταθερή προτεραιότητα: 1			
Χρόνος παραγωγής εργασιών	300 sec			
Συντελεστές Συνάρτησης αποτελ.	*speed_cf* = 1, *topology_cf* = 0, *tload_cf* = -1, *ntasks* = 0			
Ποσοστιαία μεταβολή μον. Εργ.	10...90/βήμα: 10			

Εκτελούμε σειρά πειραμάτων όπου στον εφαρμοζόμενο αλγόριθμο UIA μεταβάλουμε το όριο ενημέρωσης από 10%-90% με βήμα αύξησης 10%. Τα αποτελέσματα της προσομοίωσης συνοψίζονται στα Διαγράμματα 6 και 7 όπου απεικονίζεται η μεταβολή του μέσου χρόνου εκτέλεσης και του αριθμού των ολοκληρωμένων εργασιών ανά διεκπεραιωτή συναρτήσει του ορίου ενημέρωσης. Επιπλέον, καταγράφονται, για τις παραπάνω τιμές των παραμέτρων προσομοίωσης, οι αντίστοιχες τιμές που προκύπτουν με την εφαρμογή του αλγόριθμου LUIA (υπενθυμίζουμε ότι ο LUIA δεν εξαρτάται από το όριο ενημέρωσης).



Διάγραμμα 6 – *Μεταβολή του μέσου χρόνου εκτέλεσης συναρτήσει του ορίου ανανέωσης, για τους αλγόριθμους UIA και LUIA.*



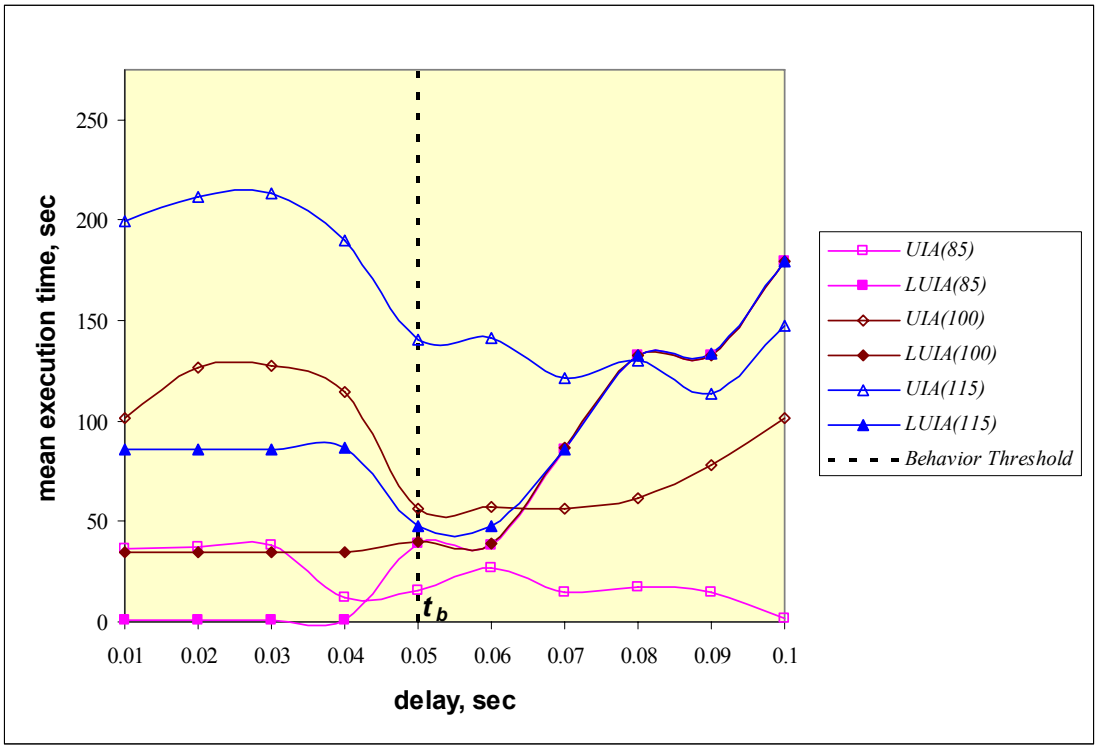
Διάγραμμα 7 – *Μεταβολή των ολοκληρωμένων εργασιών ανά υπολογιστική μονάδα συναρτήσει του ορίου ανανέωσης, για τους αλγόριθμους UIA και LUIA.*

Από την εξέταση Διαγράμματος 6 προκύπτουν οι ακόλουθες παρατηρήσεις.
i) Αύξηση του μέσου χρόνου εκτέλεσης των εργασιών με την αύξηση του ορίου ανανέωσης, καθώς η επιλογή των βέλτιστων μονάδων βασίζεται ολοένα και σε «παλαιότερη» πληροφορία σε σχέση με αυτή που περιγράφει την κατάσταση του συστήματος την στιγμή που λαμβάνει χώρα η επιλογή αυτή.

ii) Ο αλγόριθμος UIA είναι αποτελεσματικότερος από τον LUIA για τιμές του ορίου ανανέωσης μικρότερες των 60 ποσοστιαίων μονάδων. Αυτό συμβαίνει επειδή α) για αυτή την περιοχή τιμών του ορίου ανανέωσης, το στιγμιότυπο της κατάστασης του συστήματος βάσει του οποίου λαμβάνουν χώρα οι αποφάσεις της πολιτικής τοποθεσίας, δεν διαφέρει πολύ από την κατάσταση του συστήματος την χρονική στιγμή που πραγματοποιούνται οι αποφάσεις αυτές και β) στον UIA η πολιτική μεταφοράς εφαρμόζεται άμεσα σε κάθε νέα εργασία (δεν έχουμε συσσώρευση εργασιών στην ουρά του συντονιστή) καθώς χρησιμοποιείται η υπάρχουσα πληροφορία για την κατάσταση του συστήματος που διατηρείται στον συντονιστή.

Για μεγάλες τιμές του ορίου ενημέρωσης, η αστοχία του UIA είναι προφανής. Η πληροφορία που χρησιμοποιείται για την ανάθεση των εργασιών δεν ανταποκρίνεται στην πραγματικότητα. Το σύστημα εμφανίζει την τάση να επιλέγει τον διεκπεραιωτή με μεγαλύτερη ταχύτητα επεξεργασίας (Διάγραμμα 7).

Τέλος, πραγματοποιούμε προσομοίωση συστήματος με παραμέτρους αυτές του Πίνακα 1, όριο ενημέρωσης 60 και εφαρμοζόμενους αλγόριθμους τους LUIA και UIA. Τα αποτελέσματα της προσομοίωσης απεικονίζονται στο Διάγραμμα 8 όπου περιγράφεται η μεταβολή του μέσου χρόνου εκτέλεσης των εργασιών συναρτήσει του delay, για διάφορα μεγέθη εργασιών και για τους εφαρμοζόμενους αλγόριθμους. Σημειώνεται ότι οι καμπύλες (όπως UIA(130), UIA(145) και UIA(160)) που προκύπτουν από την εφαρμογή του UIA και απεικονίζουν απόδοση συστήματος πολύ χειρότερη από τις αντίστοιχες καμπύλες (για το ίδιο δηλαδή μέγεθος εργασίας) που προκύπτουν από την εφαρμογή του LUIA, αποκλείονται από το διάγραμμα για λόγους απλότητας.



Διάγραμμα 8 – *Μεταβολή του μέσου χρόνου εκτέλεσης συναρτήσει της καθυστέρησης ανά μονάδα τοπολογικού συντελεστή, για διάφορα φορτία και τους εφαρμοζόμενους αλγορίθμους.*

Από το Διάγραμμα 8 φαίνεται ότι για αυξημένη τιμή του delay ο UIA είναι αποτελεσματικότερος του LUIA, αν και οι αποφάσεις της πολιτικής τοποθεσίας του πρώτου αλγόριθμου λαμβάνονται βάσει παλαιότερων πληροφοριών σε σύγκριση με αυτές που χρησιμοποιούνται στον δεύτερο. Αυτό συμβαίνει καθώς η αρνητική επίδραση του delay (για το συγκεκριμένο εύρος τιμών του) στον LUIA είναι σαφώς ισχυρότερη από αυτή στον UIA. Η τελευταία παρατήρηση στην ουσία επισημαίνει την ανάγκη ύπαρξης ισορροπίας μεταξύ της ακρίβειας της πληροφορίας που χρησιμοποιείται από ένα αλγόριθμο και του κόστους (συνήθως κόστος επικοινωνίας) που προκύπτει από την ανάκτηση της πληροφορίας αυτής.

Για μεγάλα μεγέθη εργασιών ο UIA δεν είναι αποδοτικός αφού επιφέρει ταχεία επιβάρυνση των υπολογιστικών μονάδων με υψηλό φορτίο, προκαλώντας έτσι την υπερφόρτωση του συστήματος. Υπενθυμίζουμε ότι, στον UIA οι εργασίες αποστέλλονται απευθείας στον βέλτιστο διεκπεραιωτή χωρίς να μεσολαβεί κάποιο χρονικό διάστημα ώστε το σύστημα να ολοκληρώσει μέρος του φορτίου του χωρίς την επιβάρυνση του με επιπρόσθετο.

3.6 Επίδραση της τοπολογίας

Σκοπός της ακόλουθης πειραματικής διαδικασίας είναι η μελέτη της επίδρασης της τοπολογίας στην αποτελεσματικότητα του συστήματος. Αποδεικνύεται ότι, η συνεκτίμηση της δομής με τις υπόλοιπες πληροφορίες του πλέγματος είναι απαραίτητη για την λήψη ορθών αποφάσεων.

Στο παρόν κείμενο ο όρος *τοπολογία* ή *δομή* συνδέεται με το σύνολο των πιθανών θέσεων που μπορεί να καταλάβει ένας διεκπεραιωτής στο πλέγμα. Το πλέγμα θεωρείται ότι αποτελείται από σταθερό αριθμό θέσεων/ οπών με καθορισμένα χαρακτηριστικά επικοινωνίας μεταξύ τους (τοπολογικός συντελεστής και καθυστέρηση ανά μονάδα τοπολογικού συντελεστή).

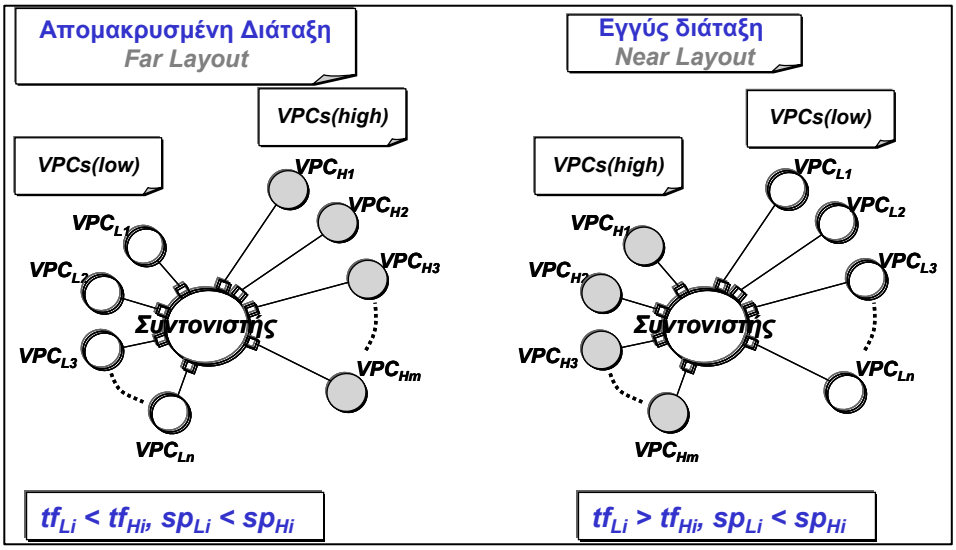
Μέτρο επιρροής της τοπολογίας μπορεί να αποτελέσει το μέγεθος της μεταβολής της απόδοσης του συστήματος που προέρχεται από την διαφοροποίηση των επιλογών της πολιτικής τοποθεσίας. Η διαφοροποίηση των αποφάσεων ενός αλγορίθμου (για συγκεκριμένη τοπολογία) μπορεί να συμβεί με την μεταβολή του βαθμού αξιοπιστίας της πληροφορίας βάσει της οποίας λαμβάνονται οι αποφάσεις αυτές. Η αξιοπιστία της πληροφορίας συνδέεται άμεσα με το πρόβλημα της παρωχημένης πληροφορίας.

Για κάθε μία από τις υπό εξέταση τοπολογίες (παράγραφος 3.6.1), διεξάγονται δύο σειρές πειραμάτων με εφαρμογή του αλγόριθμου LUIA. Η διαφορά των δύο σειρών έγκειται στο γεγονός ότι, στη μία εξ αυτών, οι αποφάσεις λαμβάνονται υπό την απουσία του προβλήματος της παρωχημένης πληροφορίας. Αυτό επιτυγχάνεται με την εισαγωγή κατάλληλου διορθωτικού παράγοντα στην αποτίμηση των μονάδων εργασίας στον διεκπεραιωτή.

3.6.1 Περιγραφή τοπολογιών

Το υπό εξέταση υπολογιστικό πλέγμα έχει σταθερό αριθμό θέσεων και αποτελείται από δύο ομάδες διεκπεραιωτών, τις ομάδες της υψηλής και χαμηλής ταχύτητας, τις οποίες για συντομία καλούμε ως *VPCs(high)* και *VPCs(low)* αντίστοιχα. Η διαφοροποίηση τους έγκειται στην ταχύτητα επεξεργασίας. Οι υπολογιστικές μονάδες της ομάδας VPCs(high) έχουν μεγαλύτερη ταχύτητα επεξεργασίας από αυτές της ομάδας VPCs(low).

Οι τοπολογίες που θα μελετηθούν είναι δύο. Η πρώτη φέρει την VPCs(high) σε απόσταση μικρότερη από την γεννήτρια εργασιών (οντότητα *vpcs.InfoCenter*) συγκριτικά με αυτή που απέχει η VPCs(low) και καλείται *εγγύς διάταξη /τοπολογία* (*Near Layout, NL*), ενώ στη δεύτερη, καλούμενη ως *απομακρυσμένη διάταξη/ τοπολογία* (*Far Layout, FL*), συμβαίνει το αντίστροφο (σχήμα 3).



Σχήμα 3 – *Τοπολογίες πλέγματος.*

3.6.2 Υπολογισμός διορθωτικού παράγοντα

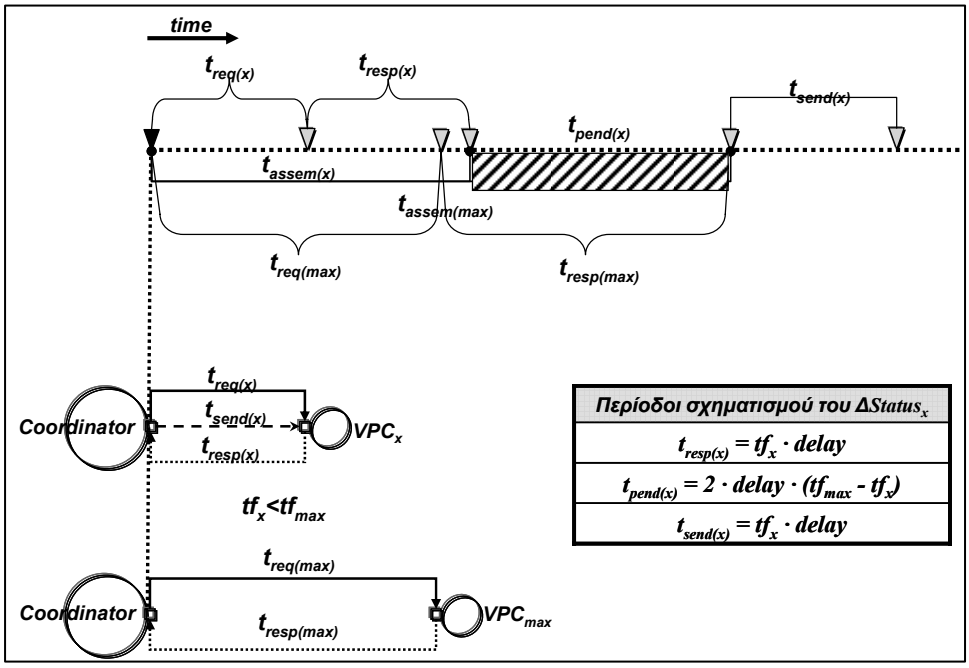
Στον LUIA, η απόκλιση της πραγματικής εικόνας του συστήματος από εκείνη βάσει της οποίας λαμβάνονται οι αποφάσεις της πολιτικής τοποθεσίας, οφείλεται στην χρονική διαφορά μεταξύ καταγραφής και χρήσης της πληροφορίας (παράγραφος 3.5).

Στο ιδανικό σενάριο όπου ο συντονιστής (οντότητα vpcs.InfoCenter) γνωρίζει την κατάσταση κάθε διεκπεραιωτή την χρονική στιγμή που θα φτάσει μία εργασία, η εξαγωγή των αποφάσεων στηρίζεται σε ακριβή στοιχεία.

Για την πραγματοποίηση του παραπάνω σεναρίου απαιτείται η εκτίμηση της διαφοράς, έστω *ΔStatus*, μεταξύ της κατάστασης της υπολογιστικής μονάδας στο χρονικό σημείο που λαμβάνει την αίτηση για αποτύπωση της εικόνας της, και αυτής που θα βρίσκεται με την άφιξη της εργασίας στην περίπτωση που εκλεγεί από την πολιτική τοποθεσίας.

Στην συγκεκριμένη περίπτωση, για την διαμόρφωση της *ΔStatus* αρκεί η εκτίμηση της διαφοράς των εναπομεινάντων προς επεξεργασία μονάδων εργασίας, αφού είναι η μοναδική μεταβλητή κατάστασης που συμμετέχει στην συνάρτηση αποτελεσματικότητας (παράγραφος 3.1).

Για κάθε διεκπεραιωτή VPC_x, η $\Delta Status_x$ σχηματίζεται αθροιστικά στις ακόλουθες τρεις χρονικές περιόδους (Σχήμα 4). Οι ποσότητες tf_x και *delay* εκφράζουν τον τοπολογικό συντελεστή και την καθυστέρηση ανά μονάδα τοπολογικού συντελεστή αντίστοιχα.



Σχήμα 4 – *Χρονικές περίοδοι διαμόρφωσης της ποσότητας ΔStatus.*

i) *Χρόνος ανταπόκρισης (response time)* $t_{resp(x)}$. Εκφράζει το χρόνο που απαιτείται από τον VPC_x για την αποστολή μηνύματος κατάστασης στον συντονιστή, και υπολογίζεται από το γινόμενο των μεγεθών tf_x και *delay* όπως φαίνεται στην Σχέση 1.

$$t_{resp(x)} = tf_x \cdot delay \quad \text{(Σχέση 1)}$$

ii) *Χρόνος αναμονής (pending time)* $t_{pend(x)}$. Ορίζει το χρονικό διάστημα παραμονής του μηνύματος καταστάσεως του VPC_x στον συντονιστή μέχρι την λήξη της διαδικασίας αποκομιδής της πληροφορίας του συστήματος. Η διαδικασία αυτή, η διάρκεια (t_{assem}) της οποίας προκύπτει από την Σχέση 2, τερματίζεται με την λήψη και του τελευταίου μηνύματος κατάστασης το οποίο αναπόφευκτα προέρχεται από τον πιο απομακρυσμένο διεκπεραιωτή, VPC_{max}.

$$t_{assem} = 2 \cdot tf_{\max} \cdot delay \quad \text{(Σχέση 2)}$$

Η Σχέση 2 προκύπτει από την 3 καθώς η τελευταία εκφράζει γενικότερα τον χρόνο ανάκτησης ($t_{assem(x)}$) των στοιχείων ενός διεκπεραιωτή VPC_x. Ο χρόνος αυτός υπολογίζεται από το άθροισμα του χρόνου μεταφοράς, $t_{req(x)}$, της αίτησης (για αποστολή μηνύματος καταστάσεως) από τον συντονιστή προς τον VPC_x, και του χρόνου αποστολής, $t_{resp(x)}$, της απάντησης του VPC_x στον συντονιστή.

$$t_{assem(x)} = 2 \cdot tf_x \cdot delay \quad \text{(Σχέση 3)},$$

Συνεπώς, ο ζητούμενος χρόνος αναμονής $t_{pend(x)}$ ισούται με την διαφορά του χρόνου συλλογής της πληροφορίας για τον VPC_x από αυτόν που απαιτείται για την διαμόρφωση της συνολικής εικόνας του συστήματος. Έτσι, από τις Σχέσεις 2 και 3 προκύπτει η Σχέση 4:

$$t_{pend(x)} = 2 \cdot delay \cdot (tf_{\max} - tf_x) \quad \text{(Σχέση 4)}$$

iii) *Χρόνος αποστολής* $t_{send(x)}$ (*sending time*). Το μέγεθος αυτό εκφράζει τον χρόνο που απαιτείται για την αποστολή της εργασίας από την γεννήτρια εργασιών στον διεκπεραιωτή VPC_x, και υπολογίζεται από την Σχέση 5.

$$t_{send(x)} = tf_x \cdot delay \quad \text{(Σχέση 5)}$$

Σύμφωνα με τα παραπάνω, οι μονάδες εργασίας που ολοκληρώνονται από τον VPC_x στην χρονική περίοδο $t_{resp(x)} + t_{pend(x)} + t_{send(x)}$ εκφράζουν την $\Delta Status_x$ (Σχέση 6). Η ποσότητα αυτή αποτελεί τον διορθωτικό παράγοντα που πρέπει να ληφθεί υπόψη κατά την αποτίμηση των προς επεξεργασία μονάδων εργασίας του VPC_x.

$$\Delta Status_x = (t_{resp(x)} + t_{pend(x)} + t_{send(x)}) \cdot cpu_x \cdot sp_x = 2 \cdot delay \cdot tf_{\max} \cdot cpu_x \cdot sp_x \text{ (Σχέση 6)}$$

όπου cpu_x και sp_x οι μονάδες πόρου και η ταχύτητα επεξεργασίας του διεκπεραιωτή VPC_x αντίστοιχα.

Αν t_1 είναι η χρονική στιγμή όπου ο VPC_x λαμβάνει αίτηση για καταγραφή και αποστολή της κατάστασης του, και $TUs(t_1)$ είναι οι προς διεκπεραίωση μονάδες εργασίας του, τότε η εγγεγραμμένη τιμή του φορτίου TUs_r στο μήνυμα καταστάσεως που αποστέλλεται, λαμβάνοντας υπόψη τον διορθωτικό παράγοντα, θα πρέπει να είναι:

$$TUs_r(t_1) = TUs(t_1) - \Delta Status_x \text{ (Σχέση 7)}$$

3.6.3 Διαφοροποίηση αποφάσεων

Η διαφοροποίηση των αποφάσεων του συστήματος με και χωρίς την εφαρμογή του διορθωτικού παράγοντα αποκαλύπτεται με την σύγκριση των αντίστοιχων συναρτήσεων αποτελεσματικότητας. Στα πειράματα που λαμβάνουν χώρα, δεχόμαστε ότι η ποσοτικοποίηση της αποδοτικότητας $fx_i(t)$ ενός διεκπεραιωτή VPC_i , την χρονική στιγμή t, εκφράζεται με την σχέση:

$$fx_i(t) = \frac{TUs_i(t)}{sp_i} \text{ (Σχέση 8)}$$

όπου $TUs_i(t)$ και sp_i, το προς διεκπεραίωση φορτίο και η ταχύτητα επεξεργασίας του VPC_i αντίστοιχα. Η σχέση αυτή ουσιαστικά αποτελεί ένα μέτρο του χρόνου που απαιτείται από τον VPC_i για την ολοκλήρωση TUs_i μονάδων εργασίας. Σημειώνεται ότι, στις προσομοιώσεις η Σχέση 8 εμφανίζεται αρνητικά προσημασμένη, αφού το μέγεθος TUs_i πολλαπλασιάζεται με τον συντελεστή (-1). Η προσήμανση της fx_i εξαρτάται αποκλειστικά από τις συμβάσεις σύγκρισης που χρησιμοποιούνται στην υλοποίηση της κλάσης vpcs.SMComparator (παράρτημα Ι).

Από τον συνδυασμό των Σχέσεων 6, 7 και 8 προκύπτει η σχέση 9 η οποία εκφράζει την συνάρτηση αποτελεσματικότητας fxc_i που λαμβάνει υπόψη τον διορθωτικό παράγοντα.

$$fxc_i(t) = \frac{TUs_i(t) - \Delta Status_i}{sp_i} = \frac{TUs_i - 2 \cdot delay \cdot tf_{\max} \cdot cpu_i \cdot sp_i}{sp_i} = fx_i(t) - C_S \text{ (Σχέση 9)}$$

Θεωρώντας ότι όλοι οι διεκπεραιωτές έχουν τις ίδιες μονάδες πόρου, τότε $C_S = 2 \cdot delay \cdot tf_{\max} \cdot cpu$ (Σχέση 10), ποσότητα ανεξάρτητη του χρόνου και των χαρακτηριστικών της υπολογιστικής μονάδας.

Η αναζήτηση των περιπτώσεων όπου το αποτέλεσμα της σύγκρισης δύο υπολογιστικών μονάδων, έστω VPC_1 και VCP_2, διαφέρει ανάλογα με την χρησιμοποιούμενη συνάρτηση αποτελεσματικότητας, μας οδηγεί στον εντοπισμό της διαφοροποίησης των αποφάσεων του συστήματος που οφείλονται στην προσθήκη του διορθωτικού παράγοντα.

Η διερεύνηση επικεντρώνεται στην σχέση μεταξύ των ποσοτήτων fx_i και C_S αφού είναι αυτή που διαμορφώνει την απόκλιση μεταξύ των fxi και $fxci$. Αν fxc_1, fx_1 και fxc_2, fx_2 είναι οι συναρτήσεις αποτελεσματικότητας με και χωρίς την διόρθωση για τα VPC_1 και VPC_2 αντίστοιχα, και λαμβάνοντας υπόψη ότι:

i) Σύμφωνα με την πολιτική τοποθεσίας, καταλληλότερος διεκπεραιωτής θεωρείται αυτός με την μικρότερη τιμή της συνάρτησης αποτελεσματικότητας fx_i, ενώ στην περίπτωση ισότητας επιλέγεται αυτός με την μεγαλύτερη ταχύτητα επεξεργασίας.

ii) Μηδενική τιμή της fx_i χαρακτηρίζει διεκπεραιωτή που βρίσκεται σε κατάσταση αδράνειας (*idle state*).

iii) Η ανισότητα $fx_i < C_S$ ($fxc_i = 0$) υποδηλώνει ότι ο VPC_i επέρχεται σε κατάσταση αδράνειας το πολύ μέχρι το χρονικό διάστημα που υπολείπεται (από την στιγμή που λαμβάνει ο VPC_i την αίτηση για αποστολή μηνύματος κατάστασης) για την ολοκλήρωση της πολιτικής ενημέρωσης και τοποθεσίας.
iv) Η ανισότητα $fx_1 < fx_2$ εκφράζει ότι ο χρόνος εκτέλεσης του υπόλοιπου φορτίου του VPC_1 είναι μικρότερος από αυτόν του VPC_2. Στην περίπτωση όπου επιπλέον ισχύει $fxc_1 = fxc_2 = 0$, ο VPC_1 μεταπίπτει σε κατάσταση αδράνειας σε χρόνο μικρότερο από ότι ο VPC_2.

Η διαφοροποίηση στην πολιτική τοποθεσίας λαμβάνει χώρα στην περίπτωση όπου $fx_1 < fx_2 < C_S$ ($fxc_1 = fxc_2 = 0$) και οι συγκρινόμενοι διεκπεραιωτές διαφέρουν ως προς την ταχύτητα επεξεργασίας. Πράγματι, θεωρώντας ο VPC_1 έχει μικρότερη ταχύτητα επεξεργασίας από τον VPC_2, η χρήση της fx_i υποδεικνύει τον VPC_1 ως βέλτιστο υποψήφιο διεκπεραιωτή (αφού $fx_1 < fx_2$). Αντιθέτως, η fxc_i προάγει τον VPC_2 καθώς οι διεκπεραιωτές βρίσκονται σε κατάσταση αδράνειας και $sp_2 > sp_1$, όπου sp_1 και sp_2 οι ταχύτητες επεξεργασίας των VPC_1 και VPC_2 αντίστοιχα. Αναλυτικότερα, στην περίπτωση όπου:

- $fx_1 \neq fx_2 > C_s$ ή $fx_1 > C_s$ *και* $fx_2 < C_s$ ή $fx_1 < C_s$ *και* $fx_2 > C_s$, οι αποφάσεις του συστήματος είναι ανεξάρτητες από την χρησιμοποιούμενη συνάρτηση αποτελεσματικότητας. Για παράδειγμα,
  - ♦ Αν $fx_1 > fx_2 > C_s$, τότε, και στις δύο περιπτώσεις, καταλληλότερος διεκπεραιωτής είναι ο VPC_2 αφού
    - χωρίς την διόρθωση $fx_1 > fx_2$, και
    - με την διόρθωση ισχύει $fx_1 - C_S > fx_2 - C_S$ $(fxc_1 > fxc_2)$.
  - ♦ Αν $fx_1 < C_s$ και $fx_2 > C_s$, τότε, και στις δύο περιπτώσεις, ο καταλληλότερος διεκπεραιωτής είναι ο VPC_1 αφού
    - χωρίς την διόρθωση $fx_1 < fx_2$, και
    - με την διόρθωση ισχύει $fxc_1=0 < fxc_2$.
- $fxc_1 = fxc_2 = 0$ και $fx_1 > fx_2$. Δεδομένου ότι, μεταξύ δύο αδρανών διεκπεραιωτών (το οποίο συμβαίνει με την fxc_i), επιλέγεται αυτός με την μεγαλύτερη ταχύτητα επεξεργασίας, η απόφαση για το ποίος είναι ο βέλτιστος είναι ανεξάρτητη της χρησιμοποιούμενης συνάρτησης αποτελεσματικότητας. Έτσι, επιλογή της πολιτικής τοποθεσίας αποτελεί ο VPC_2.
- $fxc_1 = fxc_2 = 0$ και $fx_1 < fx_2$. Αποτελεί την μοναδική περίπτωση όπου υπάρχει διαφοροποίηση στη λήψη αποφάσεων. Με χρήση της fx_i εκλέγεται ο VPC_1 (αφού $fx_1<fx_2$), ενώ με την εφαρμογή της fxc_i επιλέγεται ο VPC_2 (αφού οι διεκπεραιωτές θα βρίσκονται σε κατάσταση αδράνειας και $sp_2>sp_1$).

Σύμφωνα με τα παραπάνω παρατηρούμε ότι η παρουσία του διορθωτικού παράγοντα «ωθεί» το σύστημα στην ανάθεση μεγαλύτερου αριθμού εργασιών στους διεκπεραιωτές υψηλής ταχύτητας σε σχέση με αυτόν στην περίπτωση απουσίας του.

3.6.4 Υπολογισμός κόστους τοπολογιών

Η τοπολογία του πλέγματος αποτελεί έναν από τους σημαντικότερους παράγοντες διαμόρφωσης της απόδοσης του. Συστήματα με σαφώς καλύτερα χαρακτηριστικά από κάποια άλλα, αποδεικνύονται λιγότερο αποτελεσματικά, καθώς ο εφαρμοζόμενος αλγόριθμος δεν λαμβάνει υπόψη την διάταξη των υπολογιστικών μονάδων στον χώρο.

Στην παράγραφο 3.6.3 αποδείχθηκε ότι η διαφοροποίηση των αποφάσεων με την εφαρμογή της fxc_i αντί της fx_i, έγκειται στην ανάθεση επιπρόσθετων εργασιών στους διεκπεραιωτές υψηλής ταχύτητας. Στις επόμενες παραγράφους εξηγείται πως αυτή η διαφοροποίηση (δηλαδή η εκτέλεση μίας εργασίας σε αδρανή διεκπεραιωτή χαμηλής ταχύτητας -VPC_1- αντί σε διεκπεραιωτή υψηλής ταχύτητας-VPC_2-) επηρεάζει την αποτελεσματικότητα του συστήματος συναρτήσει της τοπολογίας του. Οι υπό εξέταση τοπολογίες, όπως ήδη έχει αναφερθεί, είναι δύο: αυτή που φέρει την ομάδα των διεκπεραιωτών υψηλής ταχύτητας «κοντά» στην γεννήτρια εργασιών, και αντίστροφα. Η πρώτη καλείται *εγγύς διάταξη* ενώ η δεύτερη *απομακρυσμένη διάταξη*.

Η επίδραση της διαφοροποίησης των αποφάσεων (συναρτήσει της τοπολογίας) υπό την μορφή κόστους ή κέρδους στην απόδοση του συστήματος, θεωρούμε ότι μπορεί να εκφραστεί από την Σχέση 11, ενώ στο Σχήμα 5 απεικονίζονται οι καταστάσεις των διεκπεραιωτών κατά την διαφοροποίηση των αποφάσεων.

$$\text{cost(type)} = \text{topology(type)} + \text{execution(type)} \quad \text{(Σχέση 11)}$$

όπου α) *type* συμβολίζει την τοπολογία του πλέγματος, NL και FL, με *near* και *far* αντίστοιχα, β) *cost(type)* αντιπροσωπεύει το συνολικό κόστος/ κέρδος, γ) *topology(type)* αναπαριστά το κόστος/ κέρδος που οφείλεται στην τοπολογία και εκφράζεται με τις μονάδες της εργασίας που θα μπορούσαν να επεξεργαστούν ή επεξεργάζονται λόγω της θέσης του διεκπεραιωτή στο πλέγμα και δ) *execution(type)* υποδεικνύει το κόστος εκτέλεσης που προέρχεται από την ολοκλήρωση της εργασίας σε διεκπεραιωτή χαμηλής (VPC_1) αντί υψηλής ταχύτητας (VPC_2). Σημειώνεται ότι στην περίπτωση όπου οι παραπάνω ποσότητες εκφράζουν κέρδος τότε είναι αρνητικά προσημασμένες. Η Σχέση 11, η οποία καλείται «*συνάρτηση κόστους διάταξης*», δεν έχει ποσοτικό χαρακτήρα αλλά χρησιμοποιείται για τον έλεγχο της συμπεριφοράς του συστήματος.

Σύμφωνα με τα παραπάνω και τα χαρακτηριστικά της τοπολογίας NL, από την Σχέση 11 προκύπτει η Σχέση 12, καλούμενη ως *συνάρτηση κόστους εγγύς διάταξης*.

$$\text{cost(near)} = cpu_2 \times sp_2 \times delay \times (tf_1 - tf_2) + execution\ (near) \quad \text{(Σχέση 12)}$$

όπου $\text{topology(near)} = cpu_2 \times sp_2 \times delay \times (tf_1 - tf_2)$ και $tf_1 > tf_2$. Ο όρος *topology(near)* εκφράζει τις μονάδες εργασίας που θα μπορούσαν να ολοκληρωθούν στο VPC_2 κατά την διάρκεια μεταφοράς της στο VPC_1, με αποτέλεσμα να παριστάνει κόστος. Η ποσότητα *execution(near)* εκφράζει επίσης κόστος καθώς η εργασία εκτελείται σε διεκπεραιωτή μικρότερης ταχύτητας.

Ομοίως, η Σχέση 13 εκφράζει την *συνάρτηση κόστους απομακρυσμένης διάταξης*.

$$\text{cost}(far) = -cpu_1 \times sp_1 \times delay \times (tf_2 - tf_1) + execution(far) \quad \text{(Σχέση 13)}$$

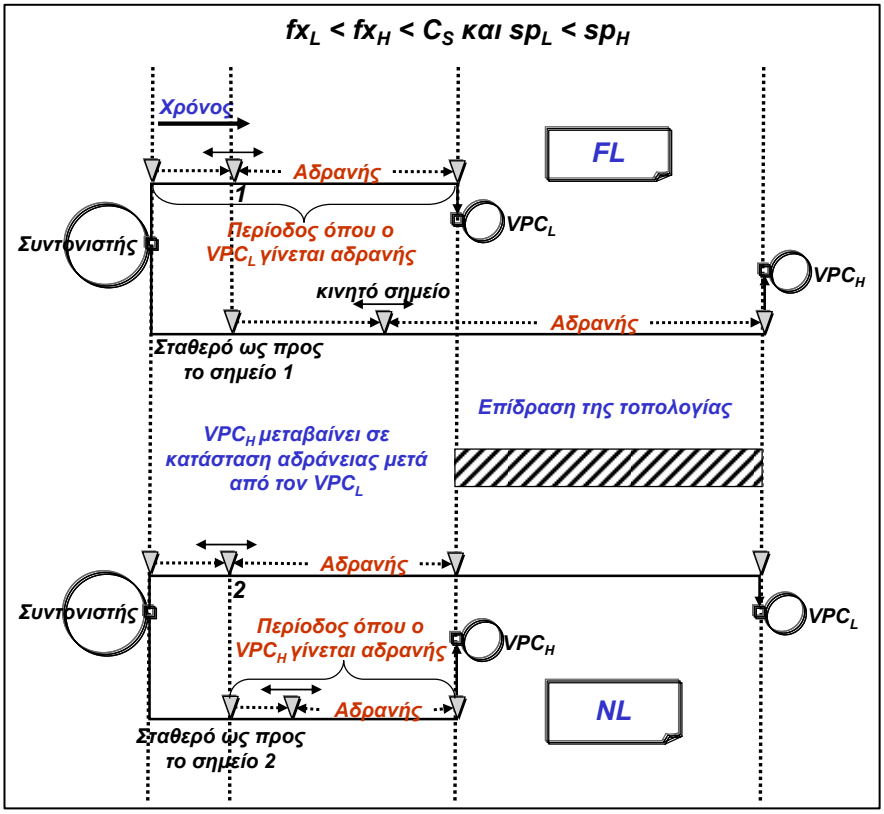
όπου $topology(far) = -cpu_1 \times sp_1 \times delay \times (tf_2 - tf_1)$ και $tf_1 < tf_2$. Σε αντίθεση με την NL, η μεταφορά της εργασίας στον VPC_1 ενισχύει την αποτελεσματικότητα του συστήματος. Η αποστολή της εργασίας στον VPC_2 θα σήμαινε απώλεια χρόνου επεξεργασίας στον διεκπεραιωτή VPC_1 ίση με $delay \times (tf_2 - tf_1)$ μονάδες χρόνου. Έτσι ο παράγοντας *topology(far)* παριστάνει κέρδος. Ο όρος *execution(far)* εκφράζει κόστος αφού η εργασία εκτελείται σε διεκπεραιωτή μικρότερης ταχύτητας.

Από τις Σχέσεις 12 και 13, συμπεραίνουμε ότι:

➢ στην εγγύς διάταξη η επιλογή ενός «απομακρυσμένου» αδρανούς διεκπεραιωτή χαμηλής ταχύτητας αντί ενός «εγγύς» αδρανούς υψηλής ταχύτητας, οδηγεί πάντα στην μείωση της απόδοσης του συστήματος. Η μείωση αυτή αυξάνεται με την αύξηση της αποστάσεως μεταξύ των δύο διεκπεραιωτών (παράγοντας (tf_2-tf_1)).

➢ Αντιθέτως, στην απομακρυσμένη διάταξη η κατάσταση δεν είναι μονοσήμαντη. Εάν $|\text{topology}(\text{far})| > \text{execution}(\text{far})$ η ποσότητα *cost(far)* είναι αρνητική, και η εργασία είναι προτιμότερο να εκτελεστεί στον κοντινό αδρανή διεκπεραιωτή χαμηλής ταχύτητας. Στην περίπτωση αυτή το κέρδος που θα απέφερε η ολοκλήρωση της εργασίας σε διεκπεραιωτή υψηλής ταχύτητας υπερκεράζεται από την απώλεια χρόνου λόγω μεταφοράς της σε αυτόν. Η πιθανότητα να ισχύει $\text{topology}(far) > \text{execution}(far)$ αυξάνεται με την απόσταση μεταξύ των διεκπεραιωτών χαμηλής και υψηλής ταχύτητας.

➢ Η επίδραση της εξεταζόμενης «επιλογής» στην απόδοση του συστήματος, είναι σαφώς ισχυρότερη στην εγγύς διάταξη αφού $\text{cost}(near) > \text{cost}(far)$.



Σχήμα 5 - *Καταστάσεις διεκπεραιωτών κατά την διαφοροποίηση των αποφάσεων.*

Τα ίδια συμπεράσματα προκύπτουν και με την σύγκριση των χρόνων εκτέλεσης μίας εργασίας μεγέθους TU_S στους VPC_1 και VPC_2 για τις υπό εξέταση τοπολογίες. Στις ακόλουθες εκφράσεις, η έναρξη μέτρησης του χρόνου εκτέλεσης λαμβάνει χώρα με την άφιξη της εργασίας στον πιο κοντινό διεκπεραιωτή, αφού η χρονική περίοδος που προηγείται (αναμονή στην ουρά της γεννήτριας εργασιών και tf_{min} *x delay*, όπου tf_{min} ο τοπολογικός συντελεστής του κοντινότερου διεκπεραιωτή στην γεννήτρια) είναι κοινή για όλους. Επίσης θεωρείται ότι δεν πραγματοποιείται άφιξη νέας εργασίας μέχρι την ολοκλήρωση της συγκεκριμένης, οπότε όλες οι μονάδες, *cpu*, του πόρου του διεκπεραιωτή διατίθενται αποκλειστικά για αυτήν.

Συγκεκριμένα για την τοπολογία NL ισχύει:

$t_{VPC1} = delay \cdot (tf_1 - tf_2) + \frac{TU_S}{cpu \cdot sp_1}$ (σχέση 14), $t_{VPC2} = \frac{TU_S}{cpu \cdot sp_2}$ (Σχέση 15)

όπου t_{VPC1} και t_{VPC2} ο χρόνος εκτέλεσης της εργασίας στον VPC_1 και VPC_2 αντίστοιχα.
Από τις Σχέσεις 14 και 15 προκύπτει ότι $t_{VPC1} > t_{VPC2}$ αφού $\frac{TU_S}{cpu \cdot sp_1} > \frac{TU_S}{cpu \cdot sp_2}$,

Άρα η εργασία είναι προτιμότερο να εκτελεστεί στον VPC_2.

Ενώ για την τοπολογία FL προκύπτει:

$t_{VPC1} = \frac{TU_S}{cpu \cdot sp_1}$ (Σχέση 16), $t_{VPC2} = delay \cdot (tf_2 - tf_1) + \frac{TU_S}{cpu \cdot sp_2}$ (Σχέση 17)

όπου t_{VPC1} και t_{VPC2} ο χρόνος εκτέλεσης της εργασίας στον VPC_1 και VPC_2 αντίστοιχα.

Επειδή $\frac{TU_S}{cpu \cdot sp_1} > \frac{TU_S}{cpu \cdot sp_2}$, η σχέση μεταξύ του t_{VPC1} και t_{VPC2} δεν μπορεί να καθοριστεί μονοσήμαντα.
Η εργασία είναι προτιμότερο να εκτελεστεί στον VPC_1 αν $t_{VPC1} < t_{VPC2}$, από όπου, με αντικατάσταση των Σχέσεων 16 και 17, προκύπτει

$$\frac{TU_S}{cpu} \cdot \left(\frac{1}{sp_1} - \frac{1}{sp_2} \right) < delay \cdot (tf_2 - tf_1) \text{ (Σχέση 18)}$$

Η ανισότητα 18 ενισχύεται με την αύξηση της απόστασης μεταξύ των διεκπεραιωτών χαμηλής και υψηλής ταχύτητας.

Από την παρούσα μελέτη γίνεται φανερό ότι η τοπολογία θα πρέπει να λαμβάνεται σοβαρά υπόψη στην διεξαγωγή αποφάσεων. Για παράδειγμα, η αποτελεσματικότητα του αλγόριθμου LUIA (ο οποίος δεν συνεκτιμά την τοπολογία) θα μπορούσε να βελτιωθεί, αν η πολιτική τοποθεσίας για την απομακρυσμένη διάταξη, στις περιπτώσεις επιλογής μεταξύ δύο αδρανών διεκπεραιωτών
i) διαφορετικών ταχυτήτων, συνεκτιμούσε και το αποτέλεσμα της Σχέσης 18 (και δεν επέλεγε απλά αυτόν με την μεγαλύτερη ταχύτητα επεξεργασίας), και
ii) ίδιων ταχυτήτων ($sp_1 = sp_2$), επέλεγε αυτόν με τον μικρότερο τοπολογικό συντελεστή.

Μία γενίκευση της Σχέσης 18 αποτελεί ή 19

$$TU_S \cdot \left(\frac{1}{cpu_{1i} \cdot sp_1} - \frac{1}{cpu_{2i} \cdot sp_2} \right) < delay \cdot (tf_2 - tf_1) \text{ (Σχέση 19)}$$

όπου cpu_{1i} και cpu_{2i} οι μονάδες των πόρων των διεκπεραιωτών VPC_1 και VPC_2 αντίστοιχα που διατίθενται για την ολοκλήρωση TU_S μονάδων εργασίας. Η σχέση αυτή καλύπτει και τις περιπτώσεις μη αδρανών υπολογιστικών μονάδων ανεξαρτήτου ταχύτητας.

3.6.5 Πειραματικά αποτελέσματα

Επιβεβαίωση της παραπάνω μελέτης αποτελεί η προσομοίωση του αλγόριθμου LUIA σε σύστημα με παραμέτρους που καταγράφονται στον Πίνακα 4.

Πίνακας 4 – Παράμετροι προσομοίωσης.

Πλέγμα διεκπεραιωτών				
Αριθμός Διεκπεραιωτών	***Μονάδες πόρου (cpu)***	***Ταχύτητα επεξεργασίας(sp)***	***Τοπολογικός συντελεστής(tf)***	***Καθυστέρηση ανά μονάδα τοπ. συντ.(delay) (σε sec)***
3	100	1	1/0	0.01...0.115 βήμα: 0.005
3	100	2	0/1	
Εργασίες				
Κατανομή παραγωγής εργασιών	σταθερός ρυθμός: 10 εργασίες/sec			
Κατανομή μονάδων εργασιών	μονάδες εργασίας (μ.ε.): 70			
Κατανομή προτεραιοτήτων εργασιών	σταθερή προτεραιότητα: 1			
Διάστημα παραγωγής εργασιών	300 sec			
Συντελεστές Συνάρτησης Αποτελεσματικότητας	*speed_cf* = 1, *topology_cf* = 0, *tload_cf* = -1, *ntasks* = 0			

Κρίσιμες παράμετροι προσομοίωσης:

i) Το εύρος τιμών της καθυστέρησης ανά μονάδα τοπολογικού συντελεστή (*delay*) ορίζεται με τέτοιο τρόπο ώστε για κάποιες τιμές, η απόδοση του συστήματος να είναι ανεξάρτητη από το χρόνο παραμονής της εργασίας στην ουρά της γεννήτριας εργασιών. Πράγματι, για $delay < 0.05$, ο χρόνος σύγκλισης $2 \times tf_{max} \times delay$ sec, της πολιτικής τοποθεσίας, είναι μικρότερος από το μεσοδιάστημα γένεσης διαδοχικών εργασιών, 0.1 sec, όπου tf_{max} ο μέγιστος τοπολογικός συντελεστής του πλέγματος. Η παραπάνω ρύθμιση είναι απαραίτητη καθώς στην περιοχή τιμών $delay>0.05$ ελέγχον στάδιο είναι ο χρόνος παραμονής της εργασίας στην ουρά της γεννήτριας, και η επίδραση της διαφοροποίησης των αποφάσεων στην απόδοση του συστήματος δεν είναι ορατή.

ii) Η διαφοροποίηση των αποφάσεων με την εισαγωγή του διορθωτικού παράγοντα, λαμβάνει χώρα όταν $fx_1<fx_2<C_S$, όπου fx_1, fx_2 οι συναρτήσεις αποτελεσματικότητας δύο διεκπεραιωτών διαφορετικής ταχύτητας, και $C_S = 2 \times delay \times tf_{max} \times cpu$. Η πιθανότητα ισχύς της παραπάνω ανισότητας, για σταθερά χαρακτηριστικά του πλέγματος ($delay$, tf_{max}, cpu, sp_i), αυξάνεται με την ικανότητα του συστήματος να ανταποκρίνεται στο επιβαλλόμενο φορτίο (μείωση των fx_i).

Η παραπάνω συνθήκη μπορεί να εξασφαλιστεί με την ρύθμιση του φορτίου ώστε η ανάθεση εργασιών στην ομάδα των διεκπεραιωτών χαμηλής ταχύτητας να πραγματοποιείται οριακά. Αυτό συμβαίνει εάν ο χρόνος εκτέλεσης (t_{ex}) μίας εργασίας σε διεκπεραιωτή υψηλής ταχύτητας είναι της ίδιας τάξεως με το άθροισμα του χρόνου που απαιτείται για την επανεκλογή του ως βέλτιστο ($t_{reassign}$), και του χρόνου μεταφοράς (t_{send}) της εργασίας σε αυτόν. Επειδή η ομάδα των διεκπεραιωτών χαμηλής ταχύτητας, στην οριακή αυτή περίπτωση, δεν αποτελεί επιλογή της πολιτικής τοποθεσίας, ο χρόνος επανεκλογής υπολογίζεται βάσει του πλήθους των διεκπεραιωτών της ομάδας υψηλής ταχύτητας.

Σύμφωνα με τα παραπάνω, για το υπό εξέταση σύστημα έχουμε:

$$t_{reassign} = (\text{ρυθμός παραγωγής εργασιών})^{-1} \times (\text{πλήθος διεκπεραιωτών υυψηλή ταχύτητας}) = 0.1 \times 3 = 0.3\text{sec}$$

και

$$t_{send} = tf \times delay = (0 \text{ or } 1) \times 0.05 = 0 \text{ ή } 0.05$$

Ο χρόνος εκτέλεσης εργασίας 70 μονάδων ισούται με $t_{ex} = \frac{70}{cpu \times sp} = \frac{70}{100 \times 2} = 0.35\text{sec}$

Σύμφωνα με τα παραπάνω $t_{reassign} + t_{send} \cong t_{ex}$. Έτσι, φορτία μεγέθους της τάξεως των 70 μονάδων είναι κατάλληλα για την διεξαγωγή των προσομοιώσεων.

Το υπολογιστικό πλέγμα αποτελείται από δύο ομάδες διεκπεραιωτών (υψηλής και χαμηλής ταχύτητας) οι θέσεις των οποίων εναλλάσσονται προς τον σχηματισμό των υπό εξέταση τοπολογιών, NL και FL. Σε κάθε τοπολογία διεξάγεται σειρά πειραμάτων όπου μεταβάλλεται η παράμετρος *delay* από 0.01 έως 0.115 sec, με βήμα αύξησης 0.005 sec. Τα αποτελέσματα της προσομοίωσης συνοψίζονται στα Διαγράμματα 9 έως 13, όπου εφαρμόζεται η εξής ονοματολογική σύμβαση. Οι όροι «fx», «fxc» αναφέρονται σε πειράματα που έλαβαν χώρα με την εφαρμογή της συνάρτησης αποτελεσματικότητας *fx* και *fxc*. Οι όροι «near», «far» υποδηλώνουν την τοπολογία του πλέγματος NL και FL, ενώ οι «low», «high» χαρακτηρίζουν την ομάδα των διεκπεραιωτών χαμηλής και υψηλής ταχύτητας αντίστοιχα.
Έτσι, η καμπύλη με όνομα «fx-near-low» προκύπτει από την προσομοίωση του αλγόριθμου LUIA σε πλέγμα τοπολογίας NL με χρήση της συνάρτησης αποτελεσματικότητας *fx*, και αφορά την ομάδα των διεκπεραιωτών χαμηλής ταχύτητας.

Στα διαγράμματα 9 και 10 απεικονίζεται η μεταβολή του μέσου χρόνου εκτέλεσης με την καθυστέρηση ανά μονάδα τοπολογικού συντελεστή, *delay*, για τις δύο τοπολογίες και τις εφαρμοζόμενες συναρτήσεις αποτελεσματικότητας. Επειδή οι χρόνοι εκτέλεσης των εργασιών που προκύπτουν για *delay*<0.05 (οριακός χρόνος συμπεριφοράς, παράγραφος 3.3) είναι σαφώς διαφορετικής τάξεως μεγέθους από αυτούς για *delay*>0.05, οι εν λόγω μεταβολές αποτυπώνονται σε δύο αντίστοιχα ξεχωριστά διαγράμματα. Από την μελέτη των διαγραμμάτων αυτών προκύπτουν τα ακόλουθα.
➢Για *delay<0.05* παρατηρείται ότι:
i) Η απόκλιση των καμπύλων λειτουργίας fx-near και fxc-near αυξάνεται με την αύξηση του *delay*. Η συμπεριφορά αυτή συμφωνεί με την ανάλυση που προηγήθηκε, αφού, για την εγγύς διάταξη η επιλογή ενός αδρανούς διεκπεραιωτή χαμηλής αντί υψηλής ταχύτητας οδηγεί πάντα σε μείωση της αποτελεσματικότητας του συστήματος, ανάλογης του *delay* (κοίτα Σχέση 12).

ii) Στην απομακρυσμένη διάταξη, η επίδραση της προαναφερόμενης επιλογής εκφράζεται με το άθροισμα δύο αντίθετα προσημασμένων μεγεθών (κοίτα Σχέση 13), η σχέση των οποίων καθορίζει την συμβολή της, θετική ή αρνητική, στην αύξηση της αποτελεσματικότητας του συστήματος. Από τις καμπύλες fx-far και fxc-far προκύπτει ότι για *delay<0.035*, οι μέσοι χρόνοι εκτέλεσης των εργασιών με εφαρμογή της *fxc* είναι μικρότεροι από αυτούς που προκύπτουν με χρήση της *fx*. Για το συγκεκριμένο εύρος τιμών του *delay*, η απόλυτη τιμή της παραμέτρου *topology(far)* είναι μικρότερη από αυτή της *execution(far)* με αποτέλεσμα η διαφοροποίηση των αποφάσεων μεταξύ *fx* και *fxc* να επιδρά αρνητικά στην απόδοση του συστήματος (κοίτα Σχέση 13). Η μείωση της απόκλισης μεταξύ των δύο καμπυλών με την αύξηση του *delay*, ερμηνεύεται με την μείωση της διαφοράς μεταξύ των *topology(far)* και *execution(far)*, καθώς το *topology(far)* είναι ανάλογο του *delay*. Για την οριακή τιμή του delay, 0.035 sec, όπου $topology(far) \cong execution(far)$, η αποτελεσματικότητα του συστήματος είναι ανεξάρτητη της εφαρμοζόμενης συνάρτησης αποτελεσματικότητας, και οι αντίστοιχοι μέσοι χρόνοι εκτέλεσης είναι περίπου ίδιοι. Για *0.035<delay<0.05,* δεν παρουσιάζεται αισθητή διαφοροποίηση μεταξύ των καμπυλών fx-far και fxc-far.

iii) Η απόκλιση μεταξύ των καμπυλών fx-near και fxc-near είναι μεγαλύτερη από αυτή των fx-far και fxc-far καθώς, στην εγγύς διάταξη η διαφοροποίηση των αποφάσεων έχει σαφώς ισχυρότερη επίδραση στην απόδοση του υπολογιστικού πλέγματος από αυτή στην απομακρυσμένη ($cost(near) > cost(far)$).

➢Για *delay>0.05* η επίδραση του διορθωτικού παράγοντα στην αποτελεσματικότητα του συστήματος δεν είναι εμφανής. Ο χρόνος εκτέλεσης καθορίζεται από τον χρόνο αναμονής των εργασιών στην ουρά της γεννήτριας, ο οποίος είναι πολύ μεγαλύτερος από τον μικτό χρόνο επεξεργασίας.

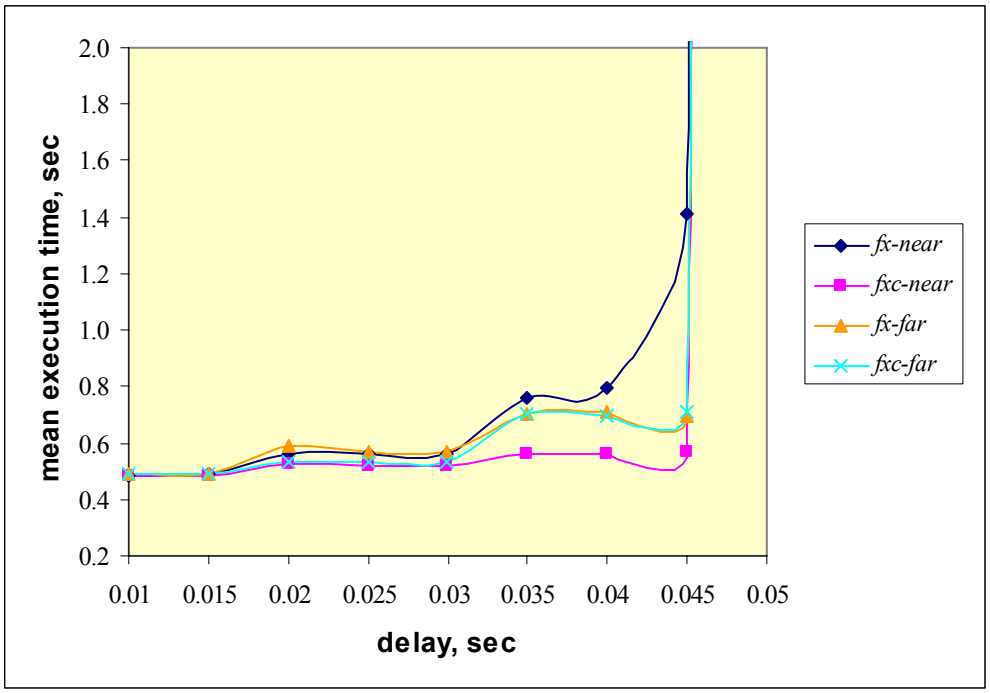
Στα Διαγράμματα 11 και 12 απεικονίζεται η μεταβολή του μέσου όρου των ολοκληρωμένων εργασιών και του μέσου % ποσοστού αδράνειας ανά ομάδα διεκπεραιωτών, συναρτήσει της καθυστέρησης ανά μονάδα τοπολογικού συντελεστή, για τις υπό εξέταση τοπολογίες και τις εφαρμοζόμενες συναρτήσεις αποτελεσματικότητας. Η εξέταση των διαγραμμάτων αυτών οδηγεί στα ακόλουθα συμπεράσματα.
i) Η εισαγωγή του διορθωτικού παράγοντα στην συνάρτηση αποτελεσματικότητας, οδηγεί στην μείωση των ολοκληρωμένων εργασιών και κατά συνέπεια στην αύξηση του % ποσοστού αδράνειας των VPCs(low), ανεξαρτήτως τοπολογίας. Η μεταβολή αυτή, και δεδομένου ότι ο συνολικός αριθμός των εργασιών παραμένει σταθερός, επιφέρει ανάλογη αύξηση των ολοκληρωμένων εργασιών και μείωση του % ποσοστού αδράνειας των διεκπεραιωτών της VPCs(high). Η παραπάνω συμπεριφορά είναι αναμενόμενη καθώς η εφαρμογή της *fxc* «*ωθεί*» το σύστημα προς ανάθεση περισσοτέρων εργασιών στους διεκπεραιωτές υψηλής ταχύτητας από ότι η *fx*.

ii) Οι αποκλίσεις των καμπυλών fx-near-low, fxc-near-low και fx-near-high, fxc-near-high, για τις μεταβολές των ολοκληρωμένων εργασιών και των % ποσοστών αδρανείας, είναι μεγαλύτερες από τις αντίστοιχες της απομακρυσμένης διάταξης, καθώς η επίδραση της διαφοροποίησης των αποφάσεων μεταξύ των *fx* και *fxc* είναι ισχυρότερη στην εγγύς διάταξη.

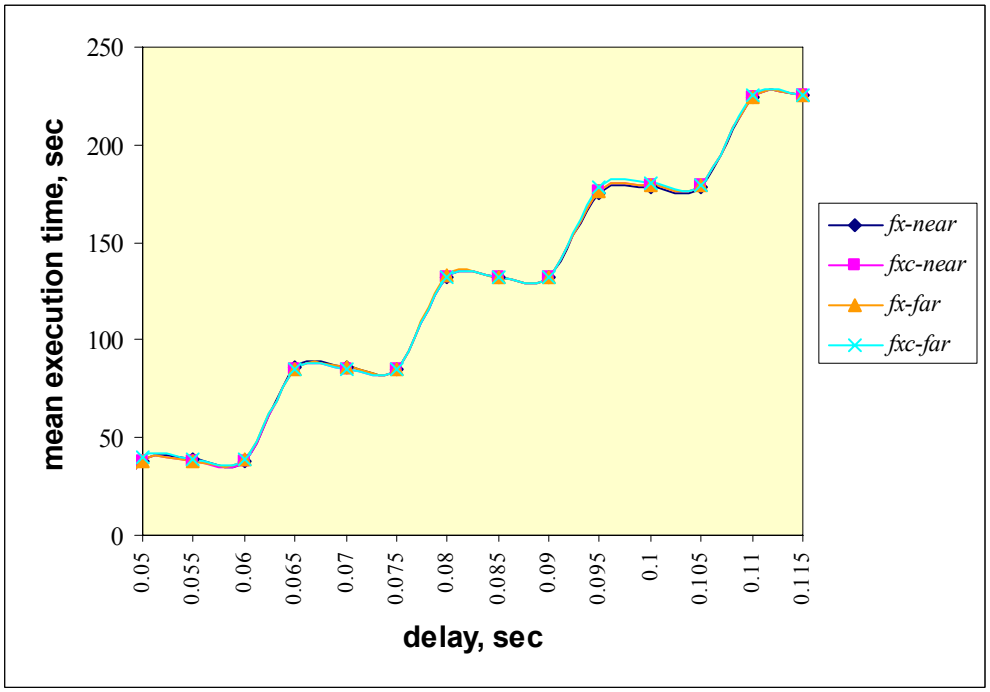
iii) Το πρόβλημα της παρωχημένης πληροφορίας διαδραματίζει σημαντικό ρόλο ακόμα και στην διαμόρφωση της δομής ενός υπολογιστικού πλέγματος. Για παράδειγμα, στο Διάγγραμμα 12 φαίνεται ότι, για *delay* = 0.075 οι τιμές του % ποσοστού αδράνειας της ομάδας των διεκπεραιωτών χαμηλής ταχύτητας στην εγγύς διάταξη, για τις *fx* και *fxc*, είναι 62% και 96% αντίστοιχα. Διεκπεραιωτές με πολύ υψηλό ποσοστό αδράνειας, είναι πιθανόν να απομακρυνθούν από το σύστημα χωρίς να μεταβληθεί σημαντικά η απόδοση του. Έτσι η εφαρμογή της *fxc* προσφέρει την δυνατότητα απομάκρυνσης ή χρήσης αυτών των διεκπεραιωτών σε διαφορετική δραστηριότητα.

iv) Η επιρροή της διαφοροποίησης των αποφάσεων στο πλήθος των διεκπεραιωμένων εργασιών, και κατά συνέπεια στο % ποσοστό αδράνειας, είναι ορατή για όλο το εύρος των τιμών της καθυστέρησης ανά μονάδα τοπολογικού συντελεστή (σε αντίθεση με το μέσο όρο εκτέλεσης).

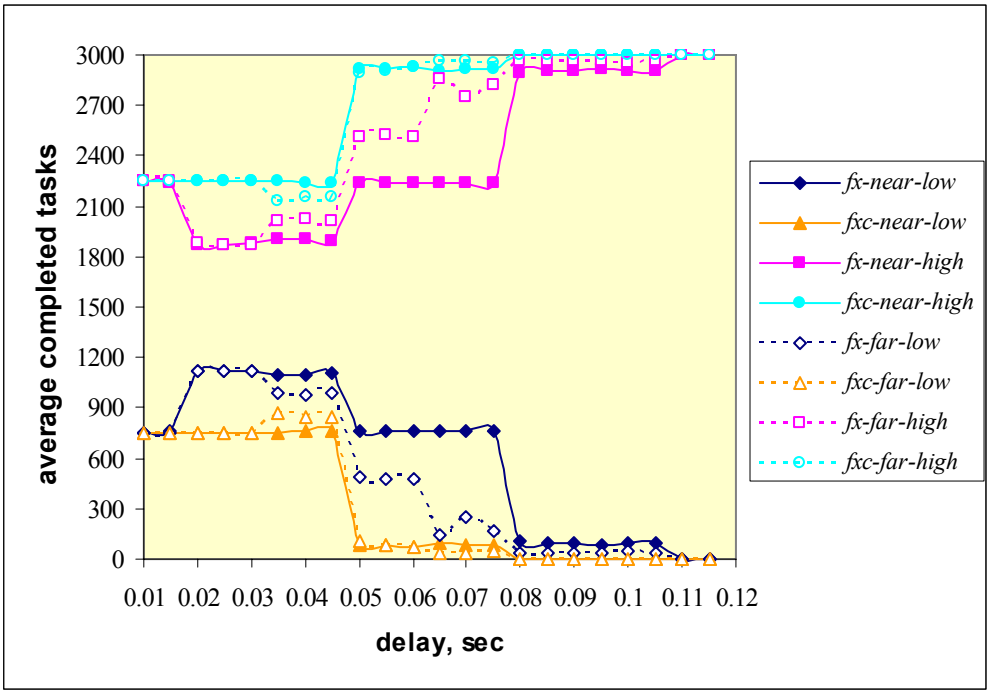
Στο Διάγραμμα 13 περιγράφονται η μεταβολή του μέσου χρόνου εκτέλεσης ανά ομάδα διεκπεραιωτών, συναρτήσει της παραμέτρου *delay*, για τις συναρτήσεις *fx* και *fxc* και τοπολογία NL (αφού η επίδραση της διαφοροποίησης στην NL είναι ξεκάθαρη). Σημειώνεται ότι η προαναφερθείσα μεταβολή περιορίζεται για *delay* < 0.05, καθώς για μεγαλύτερες τιμές η συμβολή της διαφοροποίησης των αποφάσεων στο μέσο χρόνο εκτέλεσης δεν είναι εμφανής. Από το διάγραμμα αυτό προκύπτει ότι η μείωση του μέσου χρόνου εκτέλεσης, στην εγγύς διάταξη, με την εισαγωγή του διορθωτικού παράγοντα (κοίτα επίσης Διάγραμμα 9) οφείλεται κυρίως στην μείωση του μέσου χρόνου εκτέλεσης των εργασιών της ομάδας VPCs(low), καθώς η απόκλιση μεταξύ των καμπυλών fx-near-low και fxc-near-low είναι σαφώς μεγαλύτερη από αυτή των fx-near-high και fxc-near-high. Από την προηγούμενη διαπίστωση και δεδομένου ότι η *fxc* προκαλεί μετακίνηση εργασιών από την VPCs(low) προς την VPCs(high), προκύπτει ότι η VPCs(low) παρουσιάζει μεγαλύτερη ευαισθησία στις μεταβολές του επιβαλλόμενου φορτίου. Παρόμοια συμπεριφορά διαπιστώνεται και από την μελέτη του Διαγράμματος 13, όπου η απόκλιση των καμπυλών fx-near-low και fxc-near-low είναι μεγαλύτερη από αυτή των fx-near-high και fxc-near-high. Από τα παραπάνω είναι προφανές ότι το κόστος της αύξησης του επιβαλλόμενου φορτίου στους διεκπεραιωτές της VPCs(high) είναι μικρότερο από το κέρδος που επιφέρει η απαλλαγή των VPCs(low) από αυτό.



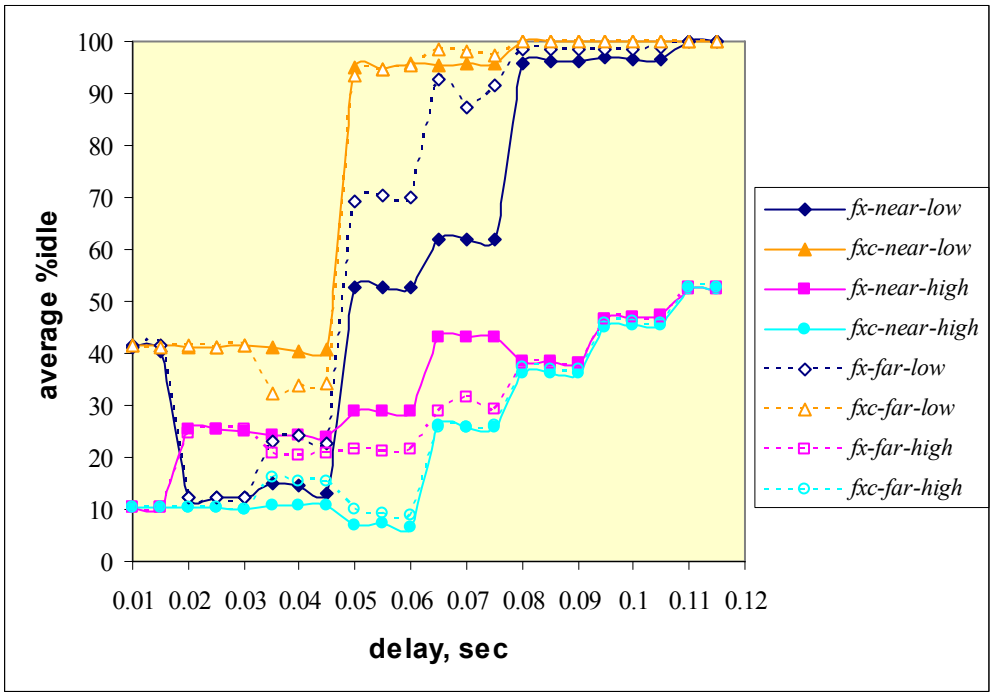
Διάγραμμα 9 – *Μεταβολή του μέσου χρόνου εκτέλεσης συναρτήσει της καθυστέρησης ανά μονάδα τοπολογικού συντελεστή < 0.05 sec για τις διατάξεις NL και FL και για τις συναρτήσεις αποτελεσματικότητας fx, fxc.*



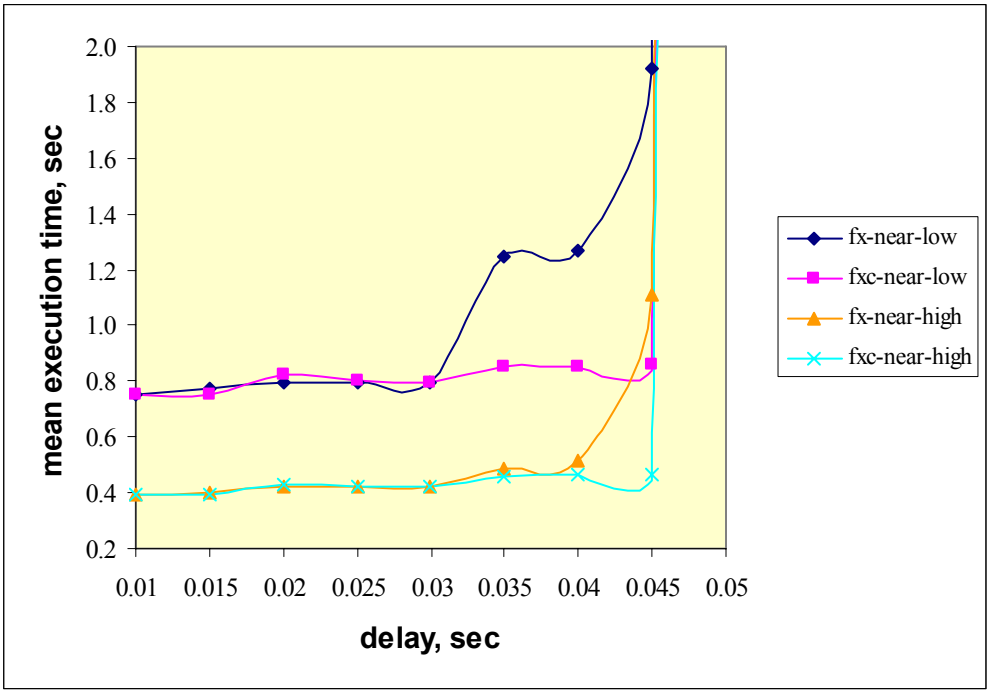
Διάγραμμα 10 – *Μεταβολή του μέσου χρόνου εκτέλεσης συναρτήσει της καθυστέρησης ανά μονάδα τοπολογικού συντελεστή ≥ 0.05 sec για τις διατάξεις NL και FL και για τις συναρτήσεις αποτελεσματικότητας fx, fxc.*



Διάγραμμα 11 – *Μεταβολή του μέσου όρου των ολοκληρωμένων εργασιών ανά ομάδα διεκπεραιωτών, συναρτήσει της καθυστέρησης ανά μονάδα τοπολογικού συντελεστή, για τις διατάξεις NL και FL και για τις συναρτήσεις αποτελεσματικότητας fx, fxc.*



Διάγραμμα 12 – *Μεταβολή του μέσου όρου του % ποσοστού αδράνειας ανά ομάδα διεκπεραιωτών, συναρτήσει της καθυστέρησης ανά μονάδα τοπολογικού συντελεστή, για τις διατάξεις NL και FL και για τις συναρτήσεις αποτελεσματικότητας fx, fxc.*



Διάγραμμα 13 – *Μεταβολή του μέσου χρόνου εκτέλεσης ανά ομάδα διεκπεραιωτών, συναρτήσει της καθυστέρησης ανά μονάδα τοπολογικού συντελεστή, delay < 0.05, για την διάταξη NL και για τις συναρτήσεις αποτελεσματικότητας fx και fxc.*

3.7 Συμπεράσματα

Τα συμπεράσματα που εξάγονται από την μελέτη των τριών αλγορίθμων, RR, LUIA και UIA μπορούν να συνοψιστούν στις ακόλουθες παραγράφους.

Η ταχύτητα επικοινωνίας μεταξύ των οντοτήτων του συστήματος και το επιβαλλόμενο φορτίο μπορούν να θεωρηθούν ως οι πιο σημαντικοί παράγοντες επίδρασης στην απόδοση ενός υπολογιστικού συστήματος, καθώς η συμπεριφορά του εξαρτάται (επηρεάζεται) άμεσα ή έμμεσα από αυτά.

Η λειτουργία ενός αλγορίθμου θα πρέπει να ρυθμίζεται με τέτοιο τρόπο ώστε να διατηρεί υψηλή αποτελεσματικότητα ανεξαρτήτως της κατάστασης του συστήματος. Για παράδειγμα, ένας αλγόριθμος που προκύπτει από τον συνδυασμό των RR, LUIA και UIA θα μπορούσε να είναι αποτελεσματικότερος από ότι ο κάθε ένας από αυτούς ξεχωριστά.

Πολύπλοκες πολιτικές πληροφόρησης που σαν μοναδικό στόχο έχουν την εξασφάλιση όσο το δυνατό περισσότερο σύγχρονων πληροφοριών για την κατάσταση του συστήματος, είναι συνήθως αναποτελεσματικές. Χαρακτηριστικά παράδειγμα αποτελούν οι αλγόριθμοι RR και UIA οι οποίοι, κάτω από ορισμένες συνθήκες, μπορεί να αποδειχθούν αποτελεσματικότεροι από τον LUIA. Για τον λόγω αυτό, επιτακτική ανάγκη αποτελεί η ισορροπία μεταξύ του κόστους της ανάκτησης μιας πληροφορίας και της ακρίβειας της πληροφορίας αυτής.

Σε συνθήκες αυξημένου φορτίου, η απόδοση του συστήματος μπορεί να αυξηθεί με την αύξηση της καθυστέρησης στην επικοινωνία μεταξύ των οντοτήτων του συστήματος (είτε αυτή προέρχεται από την πολυπλοκότητα της εφαρμοζόμενης πολιτικής, είτε από την χαμηλή ταχύτητα επικοινωνίας).

Οι Σχέσεις 12 και 13 επιβεβαιώνουν την αλληλεξάρτηση μεταξύ τοπολογίας και απόδοσης ενός συστήματος. Οι εξισώσεις αυτές δείχνουν ότι κατά τον σχεδιασμό ενός αλγορίθμου θα πρέπει να λαμβάνεται υπόψη τόσο η κατάσταση και τα χαρακτηριστικά των υπολογιστικών μονάδων, όσο και η σχετική απόσταση μεταξύ αυτών.

Το πρόβλημα της παρωχημένης πληροφορίας θα πρέπει να αντιμετωπισθεί κατάλληλα καθώς συντελεί στην μείωση της απόδοσης του συστήματος. Για παράδειγμα, υπολογιστικές μονάδες που θα μπορούσαν να βρίσκονται σε κατάσταση αδράνειας, τελικά απασχολούνται, στερώντας την δυνατότητα χρησιμοποίησης τους σε διαφορετικές δραστηριότητες.

Κάτω από τις συνθήκες της NL, αποδείχθηκε ότι η μεταφορά φορτίου από τις υπολογιστικές μονάδες με χαμηλές ταχύτητες σε αυτές των υψηλών ταχυτήτων έχει σαν αποτέλεσμα την αύξηση της απόδοσης του συστήματος.

Ένα αλγόριθμος εξισορρόπησης φορτίου θα πρέπει να λαμβάνει υπόψη την ομογένεια του συστήματος προκειμένου να έχει την δυνατότητα να εκμεταλλευτεί όσο το δυνατόν περισσότερο τα ιδιαίτερα χαρακτηριστικά των επιμέρους μονάδων του συστήματος.

Κεφάλαιο 4

Προτεινόμενο Εργαλείο Προσομοίωσης

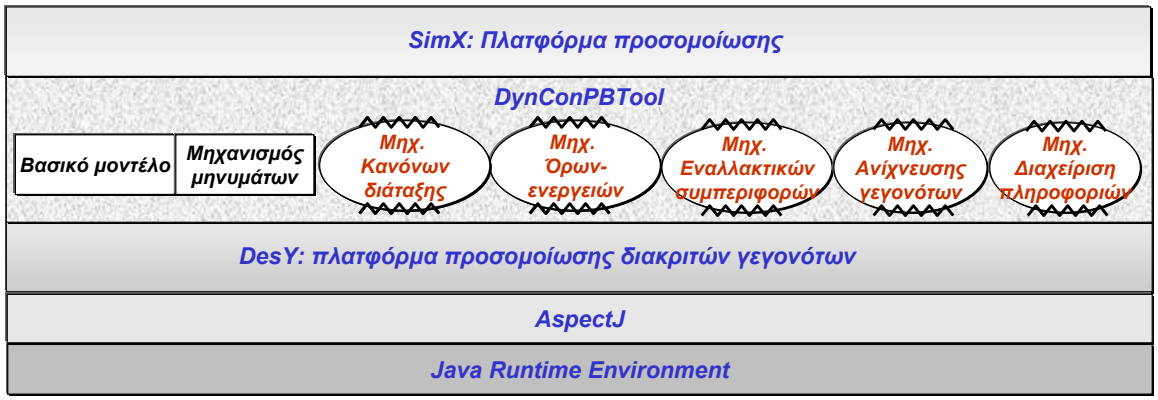
4 Προτεινόμενο εργαλείο προσομοίωσης

Στην ενότητα αυτή περιγράφεται η υλοποίηση ενός εργαλείου, καλούμενο ως DynConPBTool (dynamic configurable property-based simulation toolkit), το οποίο μπορεί να χρησιμοποιηθεί για την ανάπτυξη πλατφορμών, που αποτελούν πρωτογενώς πλατφόρμες προσομοίωσης διακριτών γεγονότων (Discrete Event Simulation Platform, DESP) ή στηρίζονται σε αυτές.

Ο σχεδιασμός και η υλοποίηση του εν λόγω εργαλείου πραγματοποιείται βάσει νέων τεχνικών προγραμματισμού (όπως η AspectJ), συστημάτων (εγγενείς βάσεις δεδομένων XML, native XML Databases) και αρχιτεκτονικών (διαδικτυακές υπηρσίες, web services) τα οποία α) ενισχύουν την ανεξαρτησία του σε επίπεδο εφαρμογών, β) βοηθούν στον αυστηρό διαχωρισμό των λειτουργιών του, σχηματίζοντας αυτόνομους επιμέρους μηχανισμούς οι οποίοι μπορούν να εφαρμοστούν κατά περίπτωση και γ) διασφαλίζουν με τον καλύτερο δυνατό τρόπο την ευελιξία, την επεκτασιμότητα και την διαφάνεια του συστήματος ως προς τον χρήστη (είτε σε επίπεδο προγραμματιστή, είτε σε επίπεδο χρήστη εφαρμογών).

Το DynConPBTool έχει ένα επιπλέον χαρακτηριστικό, ότι κατά τον σχεδιασμό του, ελήφθησαν υπόψη τα μοντέλα λειτουργίας των βασικών κατηγοριών των αλγόριθμων εξισορρόπησης φορτίου (τα οποία και αναφέρονται σε προηγούμενο κεφάλαιο) με σκοπό την δημιουργία ενός «φιλικού» εργαλείου προς την υλοποίηση πλατφορμών προσομοίωσης υπολογιστικών συστημάτων.

Το προτεινόμενο εργαλείο, σε επίπεδο αρχιτεκτονικής, δύναται να εφαρμοστεί όπως χαρακτηριστικά απεικονίζεται στο Σχήμα 1.



Σχήμα 1 – *Αρχιτεκτονική συστήματος.*

Αν θεωρηθεί ότι μια πλατφόρμα προσομοίωσης, έστω η SimX, έχει σαν πυρήνα μια πλατφόρμα διακριτών γεγονότων, DesY, τότε το DynConPBTool δρα ως διεπιφάνεια μεταξύ της DesY και τον υπόλοιπων τμημάτων της πλατφόρμας SimX.

Χαρακτηριστικό είναι ότι, μερικοί από τους μηχανισμούς του εργαλείου αυτού, όπως αυτοί που υλοποιούνται μέσω Aspects (οι μηχανισμοί "εναλλακτικών συμπεριφορών" και "όρων-ενεργειών και ανίχνευσης διακεκριμένων γεγονότων") καθώς και ο μηχανισμός καταγραφής πληροφοριών, είναι δυνατόν να χρησιμοποιηθούν αυτόνομα από οποιαδήποτε άλλο σύστημα, χωρίς δηλαδή να απαιτείται η εφαρμογή ολόκληρου του εργαλείου.

Στις παραγράφους που ακολουθούν περιγράφεται η DynConPBTool. Σημειώνεται επίσης ότι, στο παράρτημα ΙΙ πραγματοποιείται λεπτομερής περιγραφή των βασικών κλάσεων του

προτεινόμενου εργαλείου προσομοίωσης, καθώς και των σχέσεων μεταξύ αυτών, υπό την μορφή UML διαγραμμάτων κλάσεων.

4.1 Βασικό Μοντέλο (Basic Model)

Στις ακόλουθες παραγράφους περιγράφεται το μοντέλο του προτεινόμενου εργαλείου. Βασικές μονάδες του μοντέλου αποτελούν οι *οντότητες* (*components*) και οι *ιδιότητες* (*properties*). Οι οντότητες ενσαρκώνουν τα κύρια αντικείμενα του προς προσομοίωση συστήματος των οποίων την συμπεριφορά επιθυμούμε να προσομοιώσουμε. Οι ιδιότητες απεικονίζουν τα χαρακτηριστικά και κατ'επέκταση την κατάσταση των οντοτήτων, και είναι αυτές που ουσιαστικά οριοθετούν και υπαγορεύουν την συμπεριφορά του συστήματος γενικότερα. Για παράδειγμα, σε ένα υπολογιστικό πλέγμα (grid computing) το κύριο αντικείμενο είναι η υπολογιστική μονάδα, ενώ μερικά από τα χαρακτηριστικά του μπορεί να είναι η υπολογιστική και η αποθηκευτική ικανότητα.

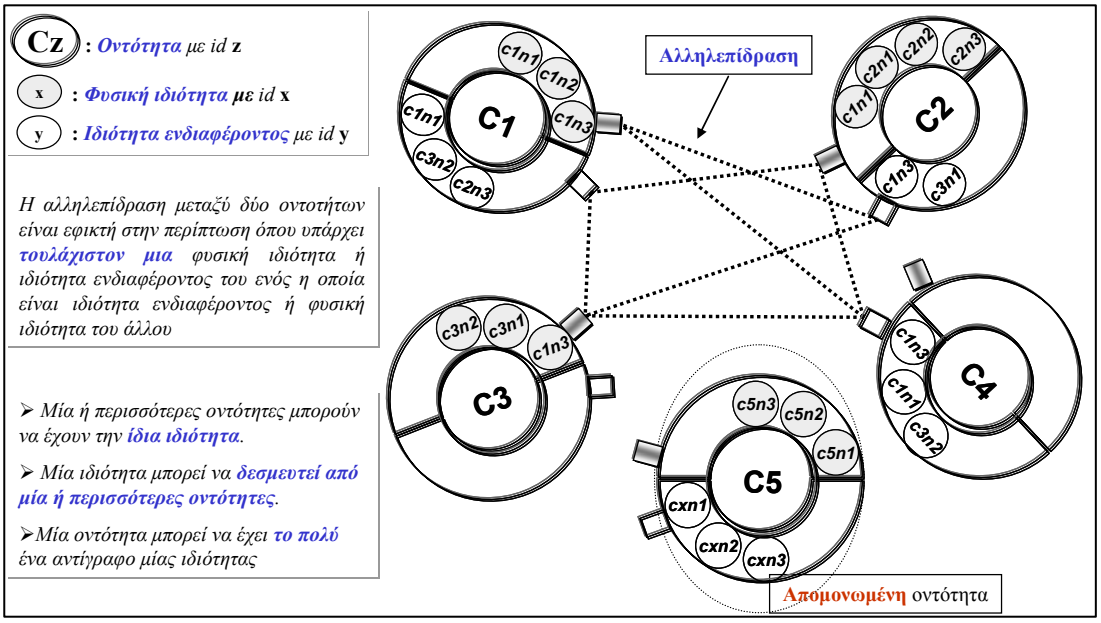
Η αλληλεπίδραση μεταξύ των οντοτήτων βασίζεται στην ακόλουθη απλή αρχή:
Σκοπός κάθε οντότητας αποτελεί είτε η *δέσμευση* ποσοστού συγκεκριμένων ιδιοτήτων (*ιδιότητες ενδιαφέροντος*, *interesting properties*) από τις υπόλοιπες οντότητες είτε και η αυτόβουλη *διάθεση/προσφορά* ποσοστού των ιδιοτήτων που κατέχει (*αυτοφυείς ιδιότητες*, *natural/belong properties*) σε άλλες οντότητες, εφόσον αυτές ενδιαφέρονται, για κάποιο χρονικό διάστημα.

Το παραπάνω γενικό μοντέλο προέκυψε κατά κύριο λόγο από την ειδικότερη συμπεριφορά που συναντάται σε ένα υπολογιστικό σύστημα όπου μία υπολογιστική μονάδα αποστέλλει μέρος του φορτίου της σε μία άλλη. Η μεταφορά φορτίου ουσιαστικά μεταφράζεται σε αποδέσμευση και δέσμευση ποσοστού του πόρου (του χρόνου επεξεργασίας) για κάποιο χρονικό διάστημα της μονάδος που απελευθερώνεται από το φορτίο και της μονάδος που επιβαρύνεται με αυτό αντίστοιχα.

Η ρητή διάκριση μεταξύ *δέσμευσης* και *διάθεσης/προσφοράς* μίας ιδιότητας όπως διατυπώνεται στην παραπάνω αρχή έχει σαν στόχο την επιπρόσθετη προσαρμογή του μοντέλου στις τρεις μεγάλες κατηγορίες των προσαρμοστικών αλγόριθμων εξισορρόπησης φορτίου, δηλαδή στους αλγόριθμους αρχικοποίησης αποστολέα, αρχικοποίησης παραλήπτη και συμμετρικής αρχικοποίησης, όπως περιγράφονται στην παράγραφο 1.2.

Θεωρώντας λοιπόν, ότι η υπολογιστική μονάδα αποτελεί οντότητα του μοντέλου προσομοίωσης έχοντας ως αυτοφυή ιδιότητα τον πόρο, η ομοιότητα της λειτουργίας ενός υπολογιστικού συστήματος και του μοντέλου του προτεινόμενου εργαλείου είναι προφανής.

Στο Σχήμα 2 απεικονίζεται το βασικό μοντέλου του εν λόγω εργαλείου.



Σχήμα 2 – *Βασικό μοντέλο.*

4.1.1 Υλοποίηση βασικού μοντέλου

Οι βασικές μονάδες του μοντέλου υλοποιήθηκαν μέσω των ακόλουθων κλάσεων τα χαρακτηριστικά των οποίων, οι βασικές λειτουργίες και οι σχέσεις μεταξύ αυτών περιγράφονται στην συνέχεια:

➢ SComponent

Η SComponent υλοποιεί τις θεμελιώδης λειτουργίες μίας *οντότητας* και αποτελεί την βασική κλάση για την υλοποίηση όλων των κλάσεων που περιγράφουν την συμπεριφορά των οντοτήτων του συστήματος.

Η αλληλεπίδραση μίας οντότητας με τις υπόλοιπες έγκειται στην επιθυμία της για δέσμευση ή προσφορά ποσοστού κάποιων ιδιοτήτων για ορισμένο χρονικό διάστημα. Η δέσμευση ή προσφορά μιας ιδιότητας (ενδιαφέροντος ή αυτοφυείς αντίστοιχα) πραγματοποιείται με αποστολή *μηνύματος/αίτησης* (*αίτηση δέσμευσης*, *bound request*) από την οντότητα που επιθυμεί να δεσμεύσει ή να προσφέρει μέρος κάποιας ιδιότητας προς τις οντότητες που θεωρεί καταλληλότερες για την δέσμευση ή την προσφορά αντίστοιχα. Η αξιολόγηση της ανάγκης για δέσμευση ή προσφορά μιας ιδιότητας καθώς και η ανίχνευση των κατάλληλων οντοτήτων πραγματοποιείται με βάση των συνόλου των αυτοφυών ιδιοτήτων τους.

Οι παραπάνω κανόνες λειτουργίας καλύπτουν το γενικό μοντέλο διαπραγμάτευσης μεταξύ των μονάδων ενός υπολογιστικού συστήματος συμπεριλαμβάνοντας φυσικά και τις περιπτώσεις εφαρμογής στατικών αλγορίθμων όπου οι εφαρμοζόμενες πολιτικές στηρίζονται σε ντετερμινιστικά, όπως είναι ο Round-Robin, η στοχαστικά λειτουργικά σχήματα. Κατά το μοντέλο αυτό η υπολογιστική μονάδα αφού αποφασίσει ότι πρέπει να συμμετάσχει στην διαδικασία μεταφοράς της εργασίας (transfer policy), αποστέλλει ή δέχεται ανάλογα φορτίο στην ή από την μονάδα αντίστοιχα που συγκλίνει περισσότερο στα κριτήρια με βάση των οποίων λαμβάνονται οι αποφάσεις της εφαρμοζόμενης πολιτικής τοποθεσίας (location policy). Για παράδειγμα, κριτήριο συμμετοχής μίας μονάδας στην διαδικασία μεταφοράς

φορτίου μπορεί να αποτελέσει η υπέρβαση του φορτίου της πάνω από ένα όριο (ένα είδος πολιτικής κατωφλίου), ενώ η επιλογή της μονάδας που θα αποσταλεί ένα φορτίο μπορεί να πραγματοποιηθεί με σύνθετο κριτήριο που λαμβάνει υπόψη το φορτίο και το πλήθος των εργασιών των μονάδων.

Κάθε οντότητα έχει δικαίωμα να μεταβάλλει το ποσό και το χρόνο δέσμευσης των αυτοφυών ιδιοτήτων της για οποιαδήποτε αίτηση δέσμευσης, εάν θεωρηθεί αναγκαίο. Η συμπεριφορά αυτή είναι θεμελιώδης για τα υπολογιστικά συστήματα αφού ο χρόνος που διαθέτει μία υπολογιστική μονάδα για την εκτέλεση μίας εργασίας μεταβάλλεται με το συνολικό φορτίο που επιβάλλεται στην μονάδα αυτή.

➢ SCompProperty

Η κλάση αυτή υλοποιεί την *ιδιότητα*. Οι ιδιότητες απεικονίζουν την κατάσταση της οντότητας στην οποία ανήκουν και αποτελούν το σημείο ενδιαφέροντος για τις υπόλοιπες οντότητες του συστήματος. Κάθε οντότητα σχετίζεται με δύο σύνολα ιδιοτήτων. Το πρώτο σύνολο, καλούμενο ως *αυτοφυείς ιδιότητες* (*natural/belong properties*), απαρτίζεται από τις ιδιότητες που χαρακτηρίζουν την οντότητα και κατ'επέκταση περιγράφουν την κατάσταση της οποιαδήποτε χρονική στιγμή. Στο δεύτερο σύνολο, καλούμενο ως *ιδιότητες ενδιαφέροντος* (*interesting properties*), ανήκουν οι ιδιότητες που η οντότητα ενδιαφέρεται να δεσμεύσει από τις υπόλοιπες οντότητες. Η εισαγωγή των ιδιοτήτων ενδιαφέροντος στο προτεινόμενο μοντέλο ενισχύει επιπλέον την δημιουργία ενός «έξυπνου» μηχανισμού ανταλλαγής μηνυμάτων και γενικότερα αλληλεπίδρασης μεταξύ των οντοτήτων του συστήματος, όπως θα αναλυθεί στην συνέχεια. Για παράδειγμα, η αποστολή μηνυμάτων καταστάσεως μεταβολής της ιδιότητας μιας οντότητας πραγματοποιείται μόνο προς τις οντότητες που ενδιαφέρονται για την ιδιότητα αυτή.

Είναι προφανές ότι η ελάχιστη αλληλεπίδραση μίας οντότητας με τις υπόλοιπες (εφόσον είναι «φυσικά» συνδεδεμένες μεταξύ τους –παράγραφος 4.7-) επιτυγχάνεται όταν οι ιδιότητες ενδιαφέροντος ή οι αυτοφυείς ιδιότητες της οντότητας είναι αυτοφυείς ιδιότητες ή ιδιότητες ενδιαφέροντος κάποιων άλλων οντοτήτων. Σε αντίθετη περίπτωση η συγκεκριμένη οντότητα είναι *απομονωμένη* (*isolated*).

Κάθε ιδιότητα είναι μοναδική με βάση το όνομα που δίνεται στο XML έγγραφο διαμόρφωσης του συστήματος, γεγονός που φέρει τον περιορισμό ότι μία οντότητα μπορεί να έχει το πολύ ένα *αντίγραφο* της ίδιας ιδιότητας, διαφορετικά δεν θα ήταν δυνατός ο διαχωρισμός μεταξύ αυτών.

Από την περιγραφή του μοντέλου είναι φανερή η ανάγκη της ποσοτικοποίησης και διαστασιολόγησης μίας ιδιότητας. Θεωρούμε ότι η ιδιότητα αποτελείται από διακριτές μονάδες, καλούμενες ως *μονάδες ιδιότητας* (*property units*). Η προκαθορισμένη τιμή των συνολικών μονάδων μίας ιδιότητας είναι 100. Η τιμή αυτή μπορεί πολύ εύκολα να μεταβληθεί ή ακόμα να αντικατασταθεί από συνάρτηση από την οποία παράγεται όπως θα δούμε στην συνέχεια (*μηχανισμός εναλλακτικών συμπεριφορών - alternative behaviors mechanism*).

Πρέπει να σημειωθεί ότι μία μονάδα ιδιότητας δεν μπορεί να δεσμευτεί από περισσότερες από μία οντότητες ταυτόχρονα. Οι μονάδες ιδιότητας που είναι διαθέσιμες (καλούμενες ως *διαθέσιμες μονάδες ιδιότητας*, *available property units*) για δέσμευση από μία οντότητα (συμπεριλαμβανομένης και της οντότητας που ανήκει η ιδιότητα) είναι αυτές που δεν είναι ήδη δεσμευμένες. Είναι προφανές ότι ο συνολικός αριθμός των δεσμευμένων μονάδων μίας ιδιότητας (καλούμενες ως *δεσμευμένες μονάδες ιδιότητας, bound property units*) δεν μπορεί να υπερβαίνει το πλήθος των συνολικών μονάδων της.

➢ BoundToken

Η κλάση αυτή φιλοξενεί πληροφορίες που σχετίζονται με την δέσμευση κάποιας ιδιότητας μιας οντότητας για συγκεκριμένη αίτηση δέσμευσης. Για παράδειγμα περιέχει το όνομα και το ποσό της ιδιότητας που δεσμεύεται, το χρονικό διάστημα της δέσμευσης, το όνομα της οντότητας για την οποία θα δεσμευτεί η ιδιότητα. Είναι προφανές ότι πολλά τέτοιου είδους αντικείμενα μπορεί να αναφέρονται στην ίδια ιδιότητα της οντότητας αφού πολλές οντότητες μπορούν ταυτόχρονα να δεσμεύουν κάποιο ποσό της ίδιας ιδιότητας.

Επιπλέον, στην κλάση αυτή περιέχεται το *ιστορικό* ή το *ίχνος* (*history/trace*) της δέσμευσης όπου καταγράφονται οι μεταβολές του ποσού και του χρόνου δέσμευσης της ιδιότητας για την συγκεκριμένη αίτηση δέσμευσης (υπενθυμίζουμε ότι η οντότητα στην οποία ανήκει η ιδιότητα έχει το δικαίωμα να μεταβάλει την ποσότητα και τον χρόνο δέσμευσης για οποιαδήποτε αίτηση δέσμευσης), καθώς και ένας χρονομέτρης (κλάση BTimer). Ο χρονομέτρης είναι το αντικείμενο όπου φυλάσσονται όλες οι «χρονικές» πληροφορίες σχετικές με την συγκεκριμένη αίτηση δέσμευσης. Αυτές οι πληροφορίες, όπως είναι ο *χρόνος λήξης δέσμευσης* (*expiration bound time*), δηλαδή ο χρόνος όπου μετά το πέρας του η αντίστοιχη δεσμευμένη ποσότητα της ιδιότητας απελευθερώνεται και είναι διαθέσιμη για δέσμευση από οποιαδήποτε οντότητα, αξιοποιούνται κυρίως από τον μηχανισμό που συντονίζει τις δεσμεύσεις των ιδιοτήτων μίας οντότητας (κλάση ProperitesBinder). Ο χρονομέτρης είναι μία κλάση γενικής χρήσεως καθώς η ύπαρξη της συνδέεται με την καταγραφή χρονικών σημείων (όπως η έναρξη και η λήξη μιας διαδικασία) και περιόδων (χρόνος από και μέχρι την έναρξη και λήξη μιας διαδικασίας αντίστοιχα) σχετικά με μία διαδικασία (για παράδειγμα η είσοδος και η έξοδος ενός αντικειμένου από μία ουρά αναμονής).

➢ ProperitesBinder

Υλοποιεί τον κεντρικό μηχανισμό για την δέσμευση ποσοστού των ιδιοτήτων της οντότητας για κάποιο χρονικό διάστημα. Τα κύρια συστατικά της κλάσης αυτής είναι α) μία ουρά (PropertiesBinder.BQueue) τα στοιχεία της οποίας είναι αντικείμενα τύπου BoundToken διατεταγμένα με βάση την χρόνο λήξης δέσμευσης της αντίστοιχης αίτησης δέσμευσης στην οποία αναφέρονται, και β) μία ανεξάρτητη διαδικασία (PropertiesBinder.Notifier) η οποία είναι υπεύθυνη για την σηματοδότηση των χρόνων λήξης δέσμευσης του συνόλου των αιτήσεων δέσμευσης που έγιναν στην οντότητα.

Αξίζει να σημειωθεί ότι ο πυρήνας υλοποίησης της BQueue (κλάση DoubleSortTreeMap) αποτελεί μια τροποποιημένη μορφή της κλάσης java.util.TreeMap έτσι ώστε να επιτρέπεται ο ορισμός των κριτηρίων βάσει των οποίων πραγματοποιείται η διάταξη των στοιχείων της (των κλειδιών της για την ακρίβεια). Στην συγκεκριμένη περίπτωση τα κριτήρια που ορίστηκαν είναι ο μοναδικός κωδικός αναγνώρισης των στοιχείων που φιλοξενούνται στην ουρά, δηλαδή των αντικειμένων BoundToken και ο αντίστοιχος χρόνος λήξης δέσμευσης με τον οποίο συσχετίζονται (κοίτα DoubleSortMapValInterface και BoundToken). Με τον τρόπο αυτό είναι δυνατή η ύπαρξη αντικειμένων BoundToken που αντιστοιχούν σε αιτήσεις δέσμευσης με τους ίδιους χρόνους λήξης.

4.1.2 Ορισμός βασικού μοντέλου

Η περιγραφή των οντοτήτων του συστήματος πραγματοποιείται στην περιοχή που οριοθετείται από το στοιχείο <components> του εγγράφου διαμόρφωσης του συστήματος όπως φαίνεται στο Σχήμα 3. Κάθε οντότητα περιγράφεται από τις ιδιότητες και το περιεχόμενο του στοιχείου <component>. Οι τιμές των ιδιοτήτων id, listid, replicas και implementation ορίζουν τον μοναδικό κωδικό ταυτοποίησης της οντότητας, τον κωδικό ταυτοποίησης της λίστας καταστάσεως (παράγραφο 4.4), τον αριθμό αντιγράφων της οντότητας και το όνομα της κλάσης υλοποίησης της. Εάν η ιδιότητα implementation

παραληφθεί, η δημιουργία των οντοτήτων πραγματοποιείται από την βασική κλάση SComponent.

Πρέπει να σημειωθεί ότι στην περίπτωση παραγωγής πολλαπλών αντιγράφων οι κωδικοί ταυτοποίησης τους προκύπτουν αυτόματα από το σύστημα με πρώτο συνθετικό την τιμή της ιδιότητας id και δεύτερο τον αύξοντα αριθμό σχηματισμού τους. Για παράδειγμα αν οι τιμές των ιδιοτήτων id και replicas είναι vpc και 5 αντίστοιχα, οι κωδικοί ταυτοποίησης των οντοτήτων που σχηματίζονται είναι vpc, vpc1, vpc2, vpc3 και vpc4.

Το στοιχείο <properties> περιέχει τα στοιχεία <belong> και <interest> όπου ορίζονται οι αυτοφυείς και οι ιδιότητες ενδιαφέροντος για την οντότητα αντίστοιχα. Η τιμή της ιδιότητας name ενός στοιχείου <prop> αναφέρεται στο κωδικό ταυτοποίησης κάποιας από τις ιδιότητες του συστήματος που ορίζονται στην αντίστοιχη περιοχή του εγγράφου διαμόρφωσης (Σχήμα 4). Υπενθυμίζεται ότι μία οντότητα δεν μπορεί να έχει παραπάνω από ένα αντίγραφο της ίδιας ιδιότητας.

```
<components>
        <component id="..." listid="..." replicas="..." implementation="...">
                <properties>
                        <belong>/*αυτοφυείς ιδιότητες*/
                                <prop name="...">
                                ...
                        </belong>
                        <interest>/*ιδιότητες ενδιαφέροντος*/
                                <prop name="...">
                                ...
                        </interest>
                </properties>
        </component>
        ...
</components>
```

Σχήμα 3 – *Τμήμα από το XML έγγραφο διαμόρφωσης του συστήματος, περιγραφή οντοτήτων του συστήματος.*

Η περιγραφή των ιδιοτήτων λαμβάνει χώρα στο στοιχείο <properties> του XML εγγράφου διαμόρφωσης του συστήματος, όπως φαίνεται στο Σχήμα 4. Ο βασικός τους ορισμός πραγματοποιείται στο στοιχείο <definition> όπου οι τιμές των ιδιοτήτων id και value κάθε στοιχείου <prop> ορίζουν τον μοναδικό κωδικό ταυτοποίησης και το πλήθος των συνολικών μονάδων μίας ιδιότητας αντίστοιχα. Ο σχολιασμός της ιδιότητας value_gen_id καθώς και του στοιχείου <value_generation> θα πραγματοποιηθεί στην παράγραφο 4.2 όπου αναπτύσσεται ο μηχανισμός εναλλακτικών συμπεριφορών.

```
<properties>
        <definition>
                <prop id="..." value="..." value_gen_id="..."/>
                ...
        </definition>
        <value_generation>
                <generator id="..." class_name="..." mth_name="...">
                        <param name="..." value="..."/>
                        ...
                </generator>
                ...
        </value_generation>
</properties>
```

Σχήμα 4 – *Τμήμα από το XML έγγραφο διαμόρφωσης του συστήματος, περιγραφή ιδιοτήτων του συστήματος.*

4.2 Μηχανισμός εναλλακτικών συμπεριφορών (alternative behaviors mechanism)

Μια από τις σημαντικότερες ανάγκες σε ένα μοντέλο προσομοίωσης είναι η δυνατότητα *δυναμικής διαμόρφωσης* (*dynamic configuration*) του. Ο χρήστης θα πρέπει να έχει την δυνατότητα να μεταβάλλει εξωτερικά την συμπεριφορά του συστήματος που προσομοιώνει και να δημιουργεί εναλλακτικά σενάρια λειτουργίας του χωρίς να είναι απαραίτητο να μεταγλωττίσει ξανά τον πηγαίο κώδικα. Αυτό είναι εφικτό με την ύπαρξη κάποιου μηχανισμού ο οποίος, κάτω από ορισμένες συνθήκες, παρακάμπτει την εκτέλεση των μεθόδων που υλοποιούν την *προκαθορισμένη συμπεριφορά* (*default behavior*) του συστήματος, και *αναθέτει* (*delegate*) την εκτέλεση σε μεθόδους οι οποίες ενσαρκώνουν την *εναλλακτική συμπεριφορά* του (*alternative behavior*). Για παράδειγμα, ας αναλογιστούμε την ευελιξία που προσφέρει η παραπάνω δυνατότητα στην περίπτωση προσομοίωσης ενός υπολογιστικού συστήματος όπου μπορούμε να μεταβάλλουμε την εφαρμοζόμενη πολιτική πληροφόρησης απλά με την αλλαγή της τιμής μίας παραμέτρου.

Η σχεδίαση ενός τέτοιου μηχανισμού πρέπει να γίνει με τέτοιο τρόπο ώστε:

- η εφαρμογή της εναλλακτικής συμπεριφοράς να πραγματοποιείται αυτόματα από το σύστημα χωρίς την ρητή η δυναμική κλήση της μέσα από τον πηγαίο κώδικα της αντίστοιχης προκαθορισμένης συμπεριφοράς. Με τον τρόπο αυτό αφενός εξασφαλίζεται η ανεξαρτησία μεταξύ προκαθορισμένων και εναλλακτικών συμπεριφορών, αφετέρου ο ορισμός εναλλακτικών σεναρίων σε οποιαδήποτε σημείο εκτέλεσης του προγράμματος χωρίς την παρέμβαση του χρήστη είναι εφικτός. Και
- να είναι εφικτός ο ορισμός οσοδήποτε εναλλακτικών σεναρίων για μία προκαθορισμένη συμπεριφορά.

Οι παραπάνω σχεδιαστικοί κανόνες οδήγησαν στην χρησιμοποίηση μιας σχετικά καινούργιας τεχνικής προγραμματισμού, καλούμενη ως *Aspect Oriented programming* (*AOP*). Η τεχνική αυτή δεν καλείται να αντικαταστήσει τον κλασικό αντικειμενοστραφή προγραμματισμό (*Object Oriented Programming - OOP*) αλλά να λύσει πρόβλημα που προκύπτουν από αυτόν, κυρίως στην περιπτώσεις όπου δεν υπάρχει σαφής διάκριση των ρόλων μεταξύ των κλάσεων μίας εφαρμογής. Κύριο χαρακτηριστικό της είναι η δυνατότητα εφαρμογής νέων συμπεριφορών (*Advices*) σε διακριτά σημεία εκτέλεσης του κώδικα (*Join Points*), όπως κλήσεις μεθόδων, κατασκευές αντικειμένων και προσβάσεις πεδίων, τα οποία καθορίζονται μέσω εκφράσεων (*Pointcuts*). Η AOP μπορεί να υλοποιηθεί σε οποιαδήποτε γλώσσα

προγραμματισμού όπως C, C++, C#, Java, Visual Basic, ακόμα και Cobol. Στην συγκεκριμένη περίπτωση χρησιμοποιείται η AspectJ η οποία αποτελεί την υλοποίηση της AOP σε Java. *[Laddad, 03]*, *[Gradecki, 03]*

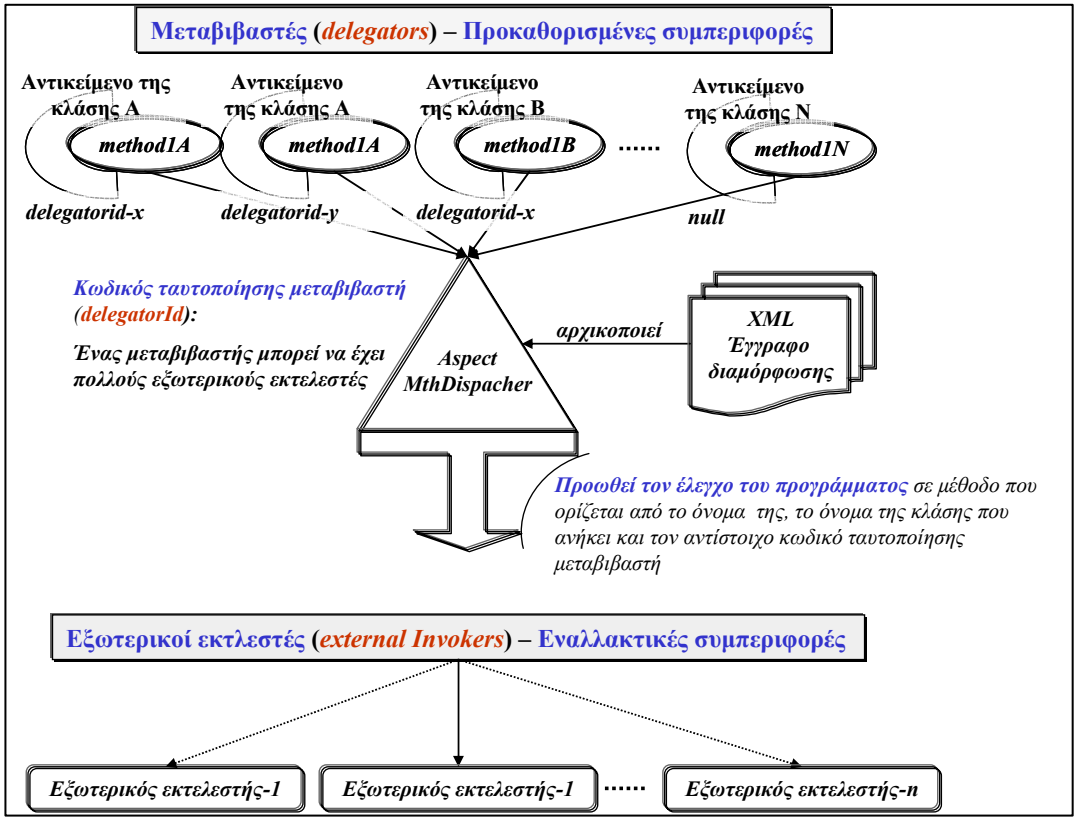
Στις παραγράφους που ακολουθούν περιγράφεται το μοντέλο και η λειτουργία του μηχανισμού εναλλακτικών συμπεριφορών.

4.2.1 Μοντέλο μηχανισμού

Σύμφωνα με το μοντέλο του μηχανισμού εναλλακτικών συμπεριφορών, ως *μεταβιβαστής* (*delegator*) χαρακτηρίζεται η μέθοδος που μπορεί να *μεταβιβάσει/προωθήσει* την εκτέλεση της σε μία άλλη μέθοδο καλούμενη ως *εξωτερικός εκτελεστής* (*external invoker*).

Πυρήνας του μηχανισμού αυτού αποτελεί το aspect MthDispacher. Το MthDispacher ανιχνεύει την κλήση των μεταβιβαστών με την βοήθεια του προκαθορισμένου σχήματος ορισμού που ακολουθούν. Στην περίπτωση που ο μεταβιβαστής συνοδεύεται από εξωτερικό εκτελεστή, ο μηχανισμός παρακάμπτει την εκτέλεση του μεταβιβαστή και προωθεί τον έλεγχο του προγράμματος στον εξωτερικό εκτελεστή, διαφορετικά ο έλεγχος παραμένει στον μεταβιβαστή. Με τον τρόπο αυτό η προκαθορισμένη συμπεριφορά μίας κλάσης αντικαθίσταται από μία εναλλακτική.

Τα ελάχιστα στοιχεία για την συσχέτιση μεταβιβαστών και εξωτερικών εκτελεστών είναι τα ονόματα των αντίστοιχων μεθόδων και κλάσεων όπου ανήκουν οι μέθοδοι αυτές στην κατάλληλη περιοχή του XML έγγραφου διαμόρφωσης του συστήματος. Τα στοιχεία αυτά είναι ικανά για την συσχέτιση μεταβιβαστών-εξωτερικών εκτελεστών σε επίπεδο κλάσεων αφού κάθε μεταβιβαστής των αντικείμενων της ίδιας κλάσεως μπορεί να έχει ταυτόχρονα το πολύ ένα εξωτερικό διεκπεραιωτή. Για την διεύρυνση του μοντέλου σε επίπεδο αντικειμένων ορίστηκε μια επιπλέον παράμετρο καλούμενη ως *κωδικός ταυτοποίησης μεταβιβαστών* (*delegatorId*). Η παράμετρος αυτή είναι χαρακτηριστική για κάθε αντικείμενο με μεταβιβαστές επιτρέποντας στον ίδιο μεταβιβαστή αντικειμένων της ίδιας κλάσεως να συσχετίζεται με περισσότερους από έναν εξωτερικούς εκτελεστές. Για παράδειγμα, έστω ότι τα αντικείμενα A και B παράγονται από την κλάση C η οποία περιέχει τον μεταβιβαστή με όνομα foo(). Στην περίπτωση όπου οι κωδικοί ταυτοποίησης μεταβιβαστών για τα αντικείμενα A και B είναι διαφορετικοί, π.χ. delIdA και delIdB αντίστοιχα, τότε η συσχέτιση δύο διαφορετικών εξωτερικών εκτελεστών, π.χ. invA() και invB(), για τον μεταβιβαστή foo() των αντικειμένων A και B αντίστοιχα είναι δυνατή. Πρέπει να σημειωθεί ότι, για κάθε μεταβιβαστή ο μηχανισμός αρχικά αναζητά την ύπαρξη εξωτερικού εκτελεστή σε επίπεδο αντικειμένου και εν συνεχεία σε επίπεδο κλάσεως, στην περίπτωση αποτυχίας. Στο Σχήμα 5 απεικονίζεται το μοντέλο του εν λόγω μηχανισμού.



Σχήμα 5 – *Μοντέλο μηχανισμού εναλλακτικών συμπεριφορών.*

Η κλάση που περιέχει μεταβιβαστές θα πρέπει να υλοποιεί τις διεπιφάνειες (interfaces) AutoGenValueMechInterface και java.lang.Cloneable. Ο ρόλος της AutoGenValueMechInterface είναι διπλός, αφενός χρησιμοποιείται ως *σηματοδότησης* (*mark interface*) για την ανίχνευση των αντικειμένων που περιέχουν μεταβιβαστές, αφετέρου η επιστρεφόμενη τιμή της μοναδικής ορισμένης μεθόδου σε αυτή (delegatorId()) χρησιμοποιείται για την συσχέτιση ζευγών μεταβιβαστή-εξωτερικού εκτελεστή σε επίπεδο αντικειμένων, όπως έχει προαναφερθεί. Η υλοποίηση του java.lang.Cloneable απαιτείται για την αντιμετώπιση των καταστάσεων όπου η ανταλλαγή πληροφοριών μεταξύ μεταβιβαστή και εξωτερικού εκτελεστή δεν μπορεί να πραγματοποιηθεί μέσω του ορίσματος του μεταβιβαστή. Στις περιπτώσεις αυτές η επικοινωνία πραγματοποιείται μέσω ενός αντίγραφου του αντικειμένου που περιέχει τον μεταβιβαστή, που ανακτάται με την εφαρμογή της μεθόδου getDelegator() στο όρισμα του εξωτερικού εκτελεστή.

Η αυτόματη ενεργοποίηση και η εφαρμογή της εναλλακτικής συμπεριφοράς από το σύστημα επιβάλλει την δημιουργία ενός πρωτοκόλλου επικοινωνίας μεταξύ μεταβιβαστών, εξωτερικών εκτελεστών και του εν λόγω μηχανισμού. Το πρωτόκολλο αυτό υλοποιείται μέσω ενός προκαθορισμένου σχήματος ορισμού των σχετικών μεθόδων. Συγκεκριμένα, ο μεταβιβαστής πρέπει να είναι μία μέθοδος που έχει σαν μοναδικό όρισμα ένα αντικείμενο τύπου Object, και επιστρεφόμενη τιμή ένα αντικείμενο τύπου AutoGeneratedValue, ενώ ο εξωτερικός εκτελεστής έχει σαν μοναδικό όρισμα ένα αντικείμενο τύπου InvokerArgument και επιστρεφόμενη τιμή ένα αντικείμενο τύπου AutoGeneratedValue. Οι παραπάνω κλάσεις είναι υλοποιημένες με τέτοιο τρόπο ώστε να προσφέρουν την μέγιστη χρηστικότητα. Για παράδειγμα η κλάση InvokerArgument παρέχει τις κατάλληλες μεθόδους ώστε να είναι δυνατή η ανάκτηση του αρχικού ορίσματος του μεταβιβαστή στην θέση του οποίου εκτελείται ο αντίστοιχος εξωτερικός εκτελεστής, η ανάκτηση του συνόλου των παραμέτρων που έχουν οριστεί για τον συγκεκριμένο εξωτερικό εκτελεστή (περισσότερα στην παράγραφο

όπου περιγράφεται η συσχέτιση μεταβιβαστών-εξωτερικών εκτελεστών στο XML έγγραφο διαμόρφωσης του συστήματος) καθώς και ένα αντίγραφο του μεταβιβαστή.

4.2.2 Ορισμός μεταβιβαστών-εξωτερικών εκτελεστών

Ο ορισμός των ζευγών μεταβιβαστών-εξωτερικών εκτελεστών πραγματοποιείται στην περιοχή που οριοθετείται από το στοιχείο <delegators-invokers> του XML εγγράφου διαμόρφωσης του συστήματος, όπως φαίνεται στο σχήμα 6. Στην περιοχή αυτή κάθε στοιχείο <delg-invk> περιγράφει μία συσχέτιση ενός μεταβιβαστή με έναν εξωτερικό εκτελεστή. Πιο συγκεκριμένα, οι τιμές των ιδιοτήτων class_name και mth_name των στοιχείων <delegator> και <invoker> αναφέρονται στα ονόματα των κλάσεων και των μεθόδων του μεταβιβαστή και του εξωτερικού εκτελεστή αντίστοιχα. Ο ορισμός της τιμή της ιδιότητας id που φιλοξενεί τον κωδικός ταυτοποίησης μεταβιβαστών στην περίπτωση όπου επιθυμούμε συσχέτιση μεταβιβαστή-εξωτερικού εκτελεστή σε επίπεδο αντικειμένων είναι προαιρετικός, σε αντίθεση με τα υπόλοιπα προαναφερθέντα στοιχεία που είναι υποχρεωτικά.

Επιπλέον, για κάθε εξωτερικό εκτελεστή μπορεί να οριστεί σύνολο αλφαριθμητικών παραμέτρων οι οποίες μπορούν να ανακτηθούν και επομένως να χρησιμοποιηθούν από αυτόν. Κάθε παράμετρος περιγράφεται από το ζεύγος των τιμών των ιδιοτήτων name και value του στοιχείου <param>, οι οποίες ορίζουν το όνομα και την τιμή της παραμέτρου αντίστοιχα. Επειδή η ανάκτηση της τιμής μίας παραμέτρου πραγματοποιείται βάσει του ονόματός της, η ύπαρξη παραμέτρων με το ίδιο όνομα (για τον ίδιο φυσικά εξωτερικό εκτελεστή) είναι απαγορευτική. (στην περίπτωση ύπαρξης παραμέτρων με το ίδιο όνομα, το σύστημα δεν τις απομακρύνει αλλά τις αποθηκεύει με διαφορετικό όνομα). Ο εξωτερικός ορισμός παραμέτρων προσφέρει επιπρόσθετη ευελιξία στο σύστημα αφού στην ουσία προσδίδει ένα επιπλέον επίπεδο διαφοροποίησης μεταξύ των εναλλακτικών συμπεριφορών (κατά κάποιο τρόπο είναι δυνατή η δημιουργία υπό-συμπεριφορών).

```
<delegators-invokers>
        <delg-invk>
                <delegator class_name="..." mth_name="..." id="..."/>
                <invoker class_name="..." mth_name="...">
                        <param name="..." value="..."/>
                        ...
                </invoker>
        </delg-invk>
        ...
</delegators-invokers>
```

Σχήμα 6 – *Τμήμα από το XML έγγραφο διαμόρφωσης του συστήματος, ορισμός ζευγών μεταβιβαστών / εξωτερικών εκτελεστών.*

4.2.3 Παραδείγματα

Για την πληρέστερη κατανόηση της λειτουργίας και της χρησιμότητας του εν λόγω μηχανισμού ας εξετάσουμε τα δύο ακόλουθα παραδείγματα. Στο πρώτο παράδειγμα η υλοποίηση δεν λαμβάνει υπόψη τον μηχανισμό εναλλακτικών συμπεριφορών σε αντίθεση με το δεύτερο.

Παράδειγμα 1
Κατά τον σχεδιασμό της προσομοίωσης ενός υπολογιστικού συστήματος αποφασίζεται ότι η εφαρμοζόμενη πολιτική πληροφόρησης πρέπει να είναι περιοδική. δηλαδή οι υπολογιστικές μονάδες στέλνουν μηνύματα καταστάσεως στις υπόλοιπες ανά τακτά χρονικά διαστήματα. Η μέθοδος που υπολογίζει και επιστρέφει το μεσοδιάστημα αποστολής των μηνυμάτων είναι η getNextTime() της κλάσης delegators.Functions Αποφασίζεται ότι ο χρόνος αυτός πρέπει να είναι σταθερός και ίσος με 3 sec, οπότε μία πιθανή υλοποίηση της μεθόδου είναι αυτή του Σχήματος 7.

```
//επιστρέφει τον επόμενο χρόνο (σε ms) όπου η οντότητα θα στείλει μήνυμα ενημέρωσης
public long  getNextTime(Object obj) {
        return 3000;
}
```

Σχήμα 7 – *Τμήμα κώδικα, πολιτική πληροφόρησης.*

Παρόλο που η υλοποίηση οριστικοποιείται και ο κώδικας μεταγλωττίζεται, το μοντέλο αναθεωρείται εκ των υστέρων. Σύμφωνα με αυτό, το μεσοδιάστημα αποστολής των μηνυμάτων καταστάσεως των υπολογιστικών μονάδων προκύπτει από διαφορετικές εξισώσεις από την αρχική. Τα νέα δεδομένα επιβάλλουν σημαντικές αλλαγές στην υλοποίηση του συστήματος καθώς μπορεί να οδηγήσουν ακόμα και σε μεταβολές στην ιεραρχία των κλάσεων. Για παράδειγμα, είναι πιθανή η δημιουργία νέας κλάσεως που περιέχει την νέα υλοποίηση της μεθόδου getNextTime() ή η προσθήκη νέας μεθόδου στην υπάρχουσα κλάση (delegators.Functions). Επιπλέον, αναπόφευκτες είναι οι τροποποιήσεις στις κλάσεις που ενσαρκώνουν τις υπολογιστικές μονάδες αφού δεν ακολουθούν ενιαία πολιτική πληροφόρησης.

Παράδειγμα 2
Σε αυτό το παράδειγμα ακολουθείται το προηγούμενο σενάριο με την μόνη διαφορά ότι στην υλοποίηση του μοντέλου λαμβάνεται υπόψη ο μηχανισμός εναλλακτικών συμπεριφορών. Στην περίπτωση αυτή η μέθοδος που υπολογίζει και επιστρέφει το μεσοδιάστημα αποστολής διαδοχικών μηνυμάτων καταστάσεως ορίζεται με βάση τις προδιαγραφές ενός μεταβιβαστή. Έτσι μια πιθανή υλοποίηση της μεθόδου είναι αυτή του Σχήματος 8.

```
//επιστρέφει τον επόμενο χρόνο (σε ms) όπου η οντότητα θα στείλει μήνυμα ενημέρωσης
public AutoGeneratedValue getNextTime(Object obj) {
   return new AutoGeneratedValue(new Long(3000));
}
```

Σχήμα 8 – *Τμήμα κώδικα, υλοποίηση πολιτικής πληροφόρησης μέσω μεταβιβαστή.*

Έχοντας υπόψη ότι σε μελλοντικά μοντέλα είναι πιθανόν να υπάρχει διαφοροποίηση στις εφαρμοζόμενες περιοδικές πολιτικές πληροφόρησης σε επίπεδο υπολογιστικών μονάδων φροντίζουμε επιπλέον να συσχετίσουμε των κωδικό ταυτοποίησης μίας υπολογιστικής μονάδας με τον κωδικό ταυτοποίησης μεταβιβαστών. Υπενθυμίζουμε ότι ο κωδικός ταυτοποίησης μεταβιβαστών επιστρέφεται από τα αντικείμενα της κλάσης που υλοποιεί μεταβιβαστές, δηλαδή αντικείμενα της κλάσης delegators.Functions.

Με τον τρόπο αυτό, η μεταβολή της πολιτικής πληροφόρησης για τις υπολογιστικές μονάδες δεν απαιτεί καμία παρέμβαση στην αρχική υλοποίηση του συστήματος. Για παράδειγμα, έστω ότι το μεσοδιάστημα αποστολής των μηνυμάτων καταστάσεως της υπολογιστικής μονάδας με κωδικό ταυτοποίησης «vpc1» περιγράφεται από την σχέση t = 10*Math.random() + 20 (Σχέση 1), όπου Math.random() γεννήτρια τυχαίων αριθμών μεταξύ 0 και 1, ενώ για τις υπόλοιπες μονάδες είναι σταθερό και ίσο με 2 sec, δηλαδή t = 2 (Σχέση 2). Η εφαρμογή των εναλλακτικών σεναρίων πραγματοποιείται με την υλοποίηση δύο εξωτερικών εκτελεστών βάσει των παραπάνω εξισώσεων και την αντιστοίχηση τους στους κατάλληλους μεταβιβαστές μέσω του XML εγγράφου διαμόρφωσης του συστήματος. Έτσι, μια πιθανή υλοποίηση των εξωτερικών εκτελεστών, λαμβάνοντας υπόψη και τις προδιαγραφές ορισμού με τις οποίες πρέπει να συμμορφώνονται, είναι αυτή που απεικονίζεται στο Σχήμα 9.

```
public AutoGeneratedValue getAlterNextTime(InvokerArgument arg) {
   java.util.Properties params = arg.getInvokerParameters();
   double a = Double.parseDouble(arg.getProperty("a"));
   double b = Double.parseDouble(arg.getProperty("b"));
   long res = a*Math.random() + b;
   return new AutoGeneratedValue(new Long(res));
}

public AutoGeneratedValue getAlterNextTimeConst(InvokerArgument arg) {
   return new AutoGeneratedValue(new Long(2000));
}
```

Σχήμα 9 – *Τμήμα κώδικα, εξωτερικοί εκτελεστές.*

Αν οι παραπάνω μέθοδοι υλοποιούνται στην κλάση externalInvokers.Invokers και a, b είναι τα ονόματα των παραμέτρων οι τιμές των οποίων ανακτούνται και χρησιμοποιούνται στον εξωτερικό εκτελεστή getAlterNextTime, τότε η περιοχή περιγραφής αντιστοιχήσεων μεταβιβαστών-εξωτερικών εκτελεστών του εγγράφου διαμόρφωσης του συστήματος θα είναι αυτή που φαίνεται στο Σχήμα 10.

```
<delegators-invokers>
        <delg-invk>
                <delegator class_name="delegators.Functions" mth_name="getNextTime" id="vpc1">
                <invoker class_name="externalInvokers.Invokers" mth_name="getAlterNextTime">
                        <param name="a" value="10"/>
                        <param name="b" value="20"/>
                </invoker>
        </delg-invk>
        <delg-invk>
                <delegator class_name=" delegators.Functions" mth_name="getNextTime">
                <invoker class_name="externalInvokers.Invokers" mth_name=" getAlterNextTimeConst "/>
        </delg-invk>
</delegators-invokers>
```

Σχήμα10 – *Τμήμα από το XML έγγραφο διαμόρφωσης του συστήματος., περιοχή μεταβιβαστών/ εξωτερικών εκτελεστών.*

Παρατηρείστε ότι οι συντελεστές της Σχέσης 1 ορίστηκαν ως παράμετροι στον εξωτερικό εκτελεστή προκειμένου να είναι δυνατή η άμεση μεταβολή τους εάν απαιτηθεί, προσδίδοντας με αυτό τον τρόπο επιπλέον ευελιξία στην προσομοίωση. Υπενθυμίζουμε επίσης ότι ο μηχανισμός αρχικά αναζητά την ύπαρξη εξωτερικού εκτελεστή σε επίπεδο αντικειμένου και εν συνεχεία σε επίπεδο κλάσεως. Έτσι, παρόλο που ο εξωτερικός εκτελεστής getAlterNextTimeConst εφαρμόζεται για όλους τους μεταβιβαστές, για τον μεταβιβαστή με κωδικό αναγνώρισης «vpc1» εφαρμόζεται ο εξωτερικός εκτελεστής getAlterNextTime.

4.2.4 Ιδιαίτερη εφαρμογή του μηχανισμού

Πολλές φορές, οι συνθήκες του εξεταζόμενου προβλήματος επιβάλλουν τον προσδιορισμό της τιμής των συνολικών μονάδων μίας ιδιότητας βάσει ενός αλγορίθμου. Για παράδειγμα, στην περίπτωση προσομοίωσης υπολογιστικών πλεγμάτων η υπολογιστικής ισχύς των διεκπεραιωτών μπορεί να ακολουθεί κατανομή Poisson. Για την αντιμετώπιση παρόμοιων σεναρίων θεωρήθηκε αναγκαία η ενσωμάτωση του μηχανισμού εναλλακτικών συμπεριφορών στον καθορισμό της τιμής των συνολικών μονάδων μίας ιδιότητας. Ο χρήστης δεν έχει παρά να ορίσει, στην κατάλληλη περιοχή του εγγράφου διαμόρφωσης του συστήματος, τα απαραίτητα στοιχεία για τον προσδιορισμό της μεθόδου που υλοποιεί τον συγκεκριμένο αλγόριθμο. Είναι προφανές ότι η μέθοδος αυτή πρέπει να ακολουθεί τις προδιαγραφές ενός εξωτερικού εκτελεστή. Επιπλέον, το αντικείμενο (το οποίο ανακτάται με την μέθοδο value() της κλάσης AutoGeneratedValue) που αναπαριστά την τιμή του επιστρεφόμενου από την μέθοδο αντικειμένου πρέπει να είναι τύπου java.lang.Long.

Ο ορισμός των εξωτερικών εκτελεστών για την παραγωγή των συνολικών μονάδων μίας ιδιότητας πραγματοποιείται στην περιοχή του στοιχείου <value_generation> όπως φαίνεται στο Σχήμα 12. Οι τιμές των ιδιοτήτων id, class_name και mth_name ενός στοιχείου <generator> προσδιορίζουν τον μοναδικό κωδικό ταυτοποίησης, το όνομα της κλάσης και της μεθόδου υλοποίησης ενός εξωτερικού εκτελεστή αντίστοιχα. Τα ζεύγη τιμών των ιδιοτήτων name και value των στοιχείων <param> ορίζουν τα ονόματα και τις τιμές αντίστοιχα των παραμέτρων που μπορούν να ανακτηθούν και χρησιμοποιηθούν από τον συγκεκριμένο εξωτερικό εκτελεστή. Η συσχέτιση ενός εξωτερικού εκτελεστή με μία ιδιότητα πραγματοποιείται μέσω της τιμής της ιδιότητας value_gen_id του αντίστοιχου στοιχείου <prop>. Η τιμή αυτή αναφέρεται σε έναν από τους κωδικούς ταυτοποίησης των εξωτερικών εκτελεστών που ορίζονται στο στοιχείο <value_generation>. Σημειώνεται ότι τιμή των συνολικών μονάδων μίας ιδιότητας που ορίζεται από την τιμή της ιδιότητας value του αντίστοιχου στοιχείου <prop> λαμβάνεται υπόψη μόνο στην περίπτωση που η ιδιότητα αυτή δεν έχει συσχετισθεί με κάποιον εξωτερικό εκτελεστή.

Για την καλύτερη κατανόηση του μηχανισμού εξετάζουμε το ακόλουθο παράδειγμα. Στα Σχήματα 11 και 12 απεικονίζονται τα τμήματα του XML εγγράφου διαμόρφωσης του συστήματος όπου περιγράφονται οι οντότητες και οι ιδιότητες του εξεταζόμενου συστήματος αντίστοιχα. Επίσης στο Σχήμα 13 φαίνεται η υλοποίηση του εξωτερικού εκτελεστή που θα χρησιμοποιηθεί για την παραγωγή των συνολικών μονάδων της ιδιότητας με κωδικό cpu.

```
<components>
        <component id="vpc" replicas="5">
                <properties>
                        <belong><prop name="cpu"></belong>
                </properties>
        </component>
</components>
```

Σχήμα11 – *Τμήμα από το XML έγγραφο διαμόρφωσης του συστήματος, περιοχή οντοτήτων.*

```
<properties>
        <definition>
                <prop id="cpu" value="100" value_gen_id="mygenerator"/>
        </definition>
        <value_generation>
        <generator id="mygenerator" class_name="PropertyValuesGenerators.Generator"
                                                                mth_name="generate"></generator>
        </value_generation>
</properties>
```

Σχήμα12 – *Τμήμα από το XML έγγραφο διαμόρφωσης του συστήματος, περιοχή ιδιοτήτων.*

```
public class Generator {
        private static long currentCpu = 1;

        public  AutoGeneratedValue generate(InvokerArgument obj) {
                return new AutoGeneratedValue (new Float(currentCpu++));
        }
}
```

Σχήμα13 – *Τμήμα κώδικα.*

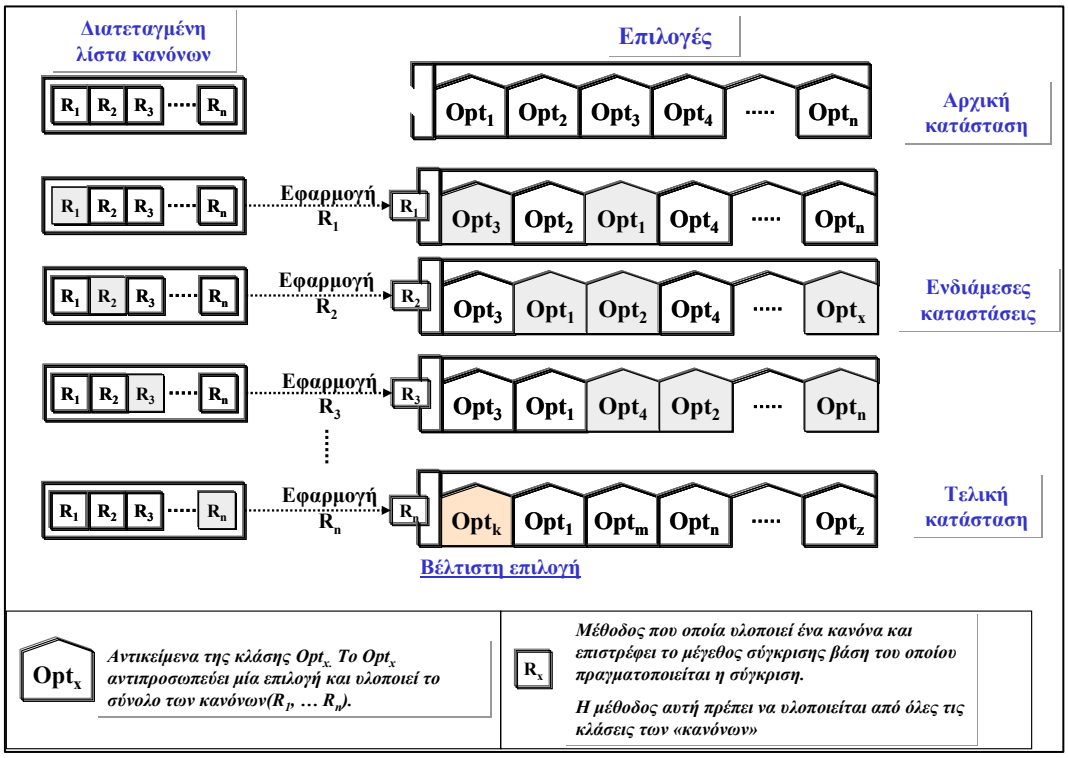
Επειδή ο εξωτερικός εκτελεστής καλείται αυτόματα από το σύστημα κάθε φορά που ένα αντίγραφο της αντίστοιχης ιδιότητας δημιουργείται σε μία οντότητα, κάθε μία από τις σχηματιζόμενες οντότητες vpc, vpc1, vpc2, vpc3 και vpc4 θα έχει την ιδιότητα cpu με τιμή 1, 2, 3, 4, 5 αντίστοιχα.

4.3 Μηχανισμός κανόνων κατάταξης (sorting rules mechanism)

Από την καθημερινής μας εμπειρία είναι γνωστό ότι η βέλτιστη *επιλογή/ απόφαση* μεταξύ συνόλου *επιλογών/ αποφάσεων* οποιασδήποτε κλίμακας και σπουδαιότητας πραγματοποιείται βάσει κάποιων κριτηρίων τα οποία βασίζονται στην κατάσταση και τις συνθήκες που επηρεάζουν την επιλογή αυτή. Η σημαντικότητα των κριτηρίων αυτών και κατ'επέκταση το βάρος συμμετοχής τους στην διεξαγωγή των αποφάσεων δεν είναι συνήθως ίδια.

Ανάλογη συμπεριφορά παρατηρείται και σε οποιοδήποτε σύστημα. Για παράδειγμα σε ένα υπολογιστικό σύστημα, η επιλογή της μονάδος όπου αποστέλλεται ένα φορτίο μπορεί να πραγματοποιείται πρωτίστως με το μέγεθος του φορτίου που ήδη επιβαρύνεται η μονάδα και κατά δεύτερο λόγο με τον χρόνο που απαιτείται για την αποστολή του φορτίου στην μονάδα αυτή.

Με βάση το παραπάνω μοντέλο, αναπτύχθηκε μηχανισμός που επιτρέπει τον ορισμό κριτηρίων και την διαδοχική τους εφαρμογή, σύμφωνα με προκαθορισμένη σειρά βαρύτητας/ προτεραιότητας, σε σύνολο επιλογών. Τα κριτήρια αυτά οδηγούν στην *βαθμωτή διάταξη* των επιλογών, υποδεικνύοντας με αυτό τον τρόπο την βέλτιστη από αυτές, όπως χαρακτηριστικά απεικονίζεται στο Σχήμα 14. Η βασική αρχή που ακολουθείται είναι ότι κάθε κριτήριο επηρεάζει την σειρά διάταξης των επιλογών που χαρακτηρίστηκαν ως ισοδύναμες (οι επιλογές δεν διαφοροποιούνται) από κριτήρια υψηλότερης βαρύτητας από αυτό. Για παράδειγμα, έστω ότι η διάταξη του συνόλου επιλογών {e5, e4, e3, e2, e1} καθορίζεται από το διατεταγμένο ζεύγος κριτηρίων {c1, c2}. Αν μετά την εφαρμογή του c1 το σύνολο επιλογών που προκύπτει είναι {e1, e2, e5, e4, e3} με τα στοιχεία e5, e4 ισοδύναμα, τότε το c2 μπορεί να επηρεάσει την σχετική διάταξη των e5 και e4.



Σχήμα 14 – *Μοντέλο λειτουργίας των κανόνων κατάταξης.*

Σε επίπεδο υλοποίησης μία επιλογή είναι ένα αντικείμενο που μπορεί να παράγεται από οποιαδήποτε κλάση που αφενός περιέχει τις παραμέτρους που λαμβάνονται υπόψη από τα εφαρμοζόμενα κριτήρια και αφετέρου υλοποιεί τις αντίστοιχες μεθόδους που ορίζουν τα κριτήρια αυτά, καλούμενες και ως *κανόνες διάταξης/ κατάταξης* ή απλώς *κανόνες*. Κοινό

χαρακτηριστικό των κλάσεων από όπου προκύπτουν οι επιλογές του ίδιου συνόλου, δηλαδή του συνόλου των επιλογών όπου μια εκ των οποίων θα χαρακτηριστεί ως βέλτιστη, είναι να έχουν τους ίδιους κανόνες διάταξης. Για την εξασφάλιση της παραπάνω συνθήκης οι επιλογές είτε πρέπει να αποτελούν αντικείμενα της ίδιας κλάσης, είτε να προέρχονται από διαφορετικές κλάσεις η οποίες όμως κληρονομούν από μία βασική κλάση όπου ορίζονται οι κανόνες διάταξης. Το σύνολο των επιλογών μπορεί να οριστεί από ένα αντικείμενο τύπου κάποιας λίστας ανάλογα με την γλώσσα προγραμματισμού που χρησιμοποιείται για την υλοποίηση της πλατφόρμας. Για παράδειγμα, στην συγκεκριμένη υλοποίηση μπορεί να χρησιμοποιηθεί η κλάση java.util.Vector.

Η εφαρμογή ενός κανόνα σε μία επιλογή έχει σαν αποτέλεσμα την επιστροφή ενός μεγέθους, καλούμενο ως *μέγεθος σύγκρισης*. Η *φυσική διάταξη* των μεγεθών σύγκρισης που προκύπτουν από την εφαρμογή ενός κανόνα στο σύνολο των επιλογών είναι αυτή που καθορίζει την διάταξη των επιλογών για το συγκεκριμένο κανόνα. Ως φυσική διάταξη των στοιχείων ενός συνόλου χαρακτηρίζεται η διάταξη που υπαγορεύεται από την φυσιολογία των στοιχείων αυτών. Για παράδειγμα στην περίπτωση ενός συνόλου ακεραίων αριθμών ή αλφαριθμητικών από το 1 έως το 5 ή το a έως e, η φυσική διάταξη είναι {1,2,3,4,5} και {a,b,c,d,e} αντίστοιχα.

Είναι προφανές ότι όλοι οι κανόνες πρέπει να διατηρούν μία κοινή φορά διάταξης των στοιχείων στα οποία εφαρμόζονται. Η υλοποίηση τους πρέπει να γίνεται με τέτοιο τρόπο ώστε η βέλτιστη επιλογή για κάθε κανόνα να αντιστοιχεί στο τελευταίο (ή στο πρώτο, εξαρτάται την συμβάσεις που έχει ορίσει ο χρήστης) στοιχείο του διατεταγμένου συνόλου επιλογών που προκύπτει από την επίδραση του κανόνα αυτού.

Όπως στην περίπτωση του μηχανισμού εναλλακτικών συμπεριφορών, έτσι και εδώ οι κανόνες διάταξης πρέπει να ακολουθούν ένα συγκεκριμένο πρότυπο ορισμού ώστε να είναι δυνατή η δυναμική τους κλήση και η σύγκριση των αντίστοιχων μεγεθών σύγκρισης που επιστρέφουν. Έτσι, μία μέθοδος μπορεί να χαρακτηριστεί ως κανόνας διάταξης όταν α) δεν έχει ορίσματα και η επιστρεφόμενη τιμή είναι αντικείμενο τύπου που υλοποιεί την διεπιφάνεια java.util.Comparable, ή β) ακολουθεί τις προδιαγραφές ενός μεταβιβαστή.

Η υλοποίηση της μεθόδου compareTo(Object) της java.util.Comparable είναι αυτή καθορίζει την φυσική διάταξη των μεγεθών σύγκρισης (περισσότερες πληροφορίες στο API). Υπενθυμίζουμε ότι η επιστρεφόμενη τιμή ενός μεταβιβαστή είναι ένα αντικείμενο τύπου AutoGeneratedValue ο οποίος υλοποιεί την προαναφερθείσα διεπιφάνεια.

Κατά την χρήση ενός μεταβιβαστή ως κανόνα διάταξης θα πρέπει να τονιστεί ότι το όρισμα του μεταβιβαστή είναι εικονικό. Ο μεταβιβαστής καλείται αυτόματα από το σύστημα χωρίς να είναι εφικτός ο προσδιορισμός του ορίσματος του. Κάτω από αυτές τις συνθήκες η ανταλλαγή πληροφοριών μεταξύ μεταβιβαστή και εξωτερικού εκτελεστή πραγματοποιείται μέσω ενός αντίγραφου του μεταβιβαστή που ανακτάται με την εφαρμογή της μεθόδου getDelegator() στο όρισμα του εξωτερικού εκτελεστή. Στην περίπτωση αυτή ο χρήστης θα πρέπει να υλοποιήσει κατάλληλα την μέθοδο clone() του μεταβιβαστή ώστε η διαχείριση των δεδομένων του αντίγραφου που θα προκύψει να είναι ασφαλής (δημιουργία deep copy). Βεβαίως, υπάρχει και η δυνατότητα ορισμού συνόλου παραμέτρων μέσω του εγγράφου διαμόρφωσης του συστήματος, οι οποίες μπορούν να ανακτηθούν και να χρησιμοποιηθούν από τον αντίστοιχο εξωτερικό εκτελεστή, όπως περιγράφτηκε στην παράγραφο 4.2.2.

Πυρήνας του παραπάνω μηχανισμού αποτελεί η κλάση GenericComparator, καλούμενος και ως *γενικός ταξινομητής*, η οποία εφαρμόζεται σε λίστες (collections) με τον ίδιο τρόπο που εφαρμόζεται η κλάση java.util.Comparator. Ένα από τα ορίσματα του κατασκευαστή της κλάσης αυτής είναι ο μονοδιάστατος πίνακας με τα ονόματα των μεθόδων (κανόνες διάταξης) για την κατάταξη του συνόλου των επιλογών που εφαρμόζεται ο γενικός ταξινομητής.

4.3.1 Ορισμός κανόνων

Ο χρήστης έχει την δυνατότητα να ορίσει τα ονόματα και την σειρά των κανόνων διάταξης μέσω του XML εγγράφου διαμόρφωσης του συστήματος, όπως φαίνεται στο Σχήμα 15. Οι κανόνες ορίζονται στην περιοχή που οριοθετείται από το στοιχείο <sorting_rules> όπου κάθε διατεταγμένο σύνολο κανόνων περιγράφεται από ένα στοιχείο <rule>. Η ταυτοποίηση των κανόνων και κατ' επέκταση η ανάκτηση τους κατά την υλοποίηση της προσομοίωσης πραγματοποιείται με χρήση της τιμής της ιδιότητας id η οποία πρέπει να είναι μοναδική. Κάθε κανόνας περιγράφεται από τις τιμές των ιδιοτήτων name και order ενός στοιχείου <mth> οι οποίες ορίζουν το όνομα και τον σχετικό βαθμό βαρύτητας του κανόνα αντίστοιχα. Ο βαθμός αυτός ορίζεται με ένα ακέραιο αριθμό ο οποίος καθορίζει την σειρά εφαρμογής του κριτηρίου στο σύνολο των επιλογών. Όσο μικρότερος είναι ο ακέραιος αριθμός τόσο μεγαλύτερη είναι η βαρύτητα του αντίστοιχου κανόνα.

```
<sorting_rules>
        <rule id="..">
                <mth name="..." order="..."/>
                ...
        </rule>
        ...
</sorting_rules>
```

Σχήμα 15 – *Τμήμα από το XML έγγραφο διαμόρφωσης του συστήματος, περιοχή κανόνων διάταξης.*

4.3.2 Παραδείγματα

Για την καλύτερη κατανόηση και χρησιμότητα του παραπάνω μηχανισμού εξετάζουμε το ακόλουθο πρόβλημα. Έστω ότι ένα υπολογιστικό σύστημα αποτελείται από πέντε διεκπεραιωτές και μία γεννήτρια φορτίου η οποία παράγει και αποστέλλει φορτία στους διεκπεραιωτές. Η επιλογή του διεκπεραιωτή πραγματοποιείται με βάση τους ακόλουθους κανόνες (οι οποίοι ουσιαστικά περιγράφουν μία πολιτική τοποθεσίας), οι οποίοι παρουσιάζονται με σειρά προτεραιότητας α) το φορτίο με το οποίο είναι ήδη επιβαρυμένος ένας διεκπεραιωτής πρέπει να είναι όσο το δυνατόν μικρότερο και β) η επεξεργαστική ισχύς του διεκπεραιωτή πρέπει να είναι όσο το δυνατόν μεγαλύτερη.

Κάθε φορά που η γεννήτρια καλείται να αποστείλει ένα φορτίο εφαρμόζει την βέλτιστη επιλογή από ένα σύνολο πέντε επιλογών. Μία επιλογή για κάθε υποψήφιο διεκπεραιωτή για την παραλαβή του φορτίου. Σύμφωνα με την ανάλυση του μηχανισμού κανόνων κατάταξης οι επιλογές θα υλοποιηθούν από μία κλάση η οποία περιέχει τις πληροφορίες που συσχετίζονται με τους εφαρμοζόμενους κανόνες, δηλαδή το φορτίο και την επεξεργαστική ισχύ του διεκπεραιωτή, καθώς και τους κανόνες αυτούς. Έτσι μια πιθανή υλοποίηση της κλάσης είναι αυτή του Σχήματος 16.

```
public class BestVPC implements AutoGenValueMechInterface, java.lang.Cloneable {
        private float _load;//φορτίο του διεκπεραιωτή
        private float _speed;//επεξεργαστική ισχύς του διεκπεραιωτή

        //πιθανώς κάποια μέθοδος αρχικοποίησης των πεδίων της κλάσης
        public setInfo(float load, float speed){…}

        //μέθοδοι για την ανάκτηση των πεδίων της κλάσης
        public float getLoad(){return _load;}
        public float getSpeed(){return _speed;}

        //ορισμός κριτηρίων
        public  AutoGeneratedValue load(Object obj) {
                return new AutoGeneratedValue (new Float(-_load));
        }

        public Float speed() {return new Float(_speed);}

        //υλοποίηση των μεθόδων των interface
        public String delegatorId(){…}
        public Object clone() {…}
}
```

Σχήμα 16 – *Τμήμα κώδικα, υλοποίηση επιλογής.*

Σημειώστε στο παραπάνω κώδικα ότι τα επιστρεφόμενα αντικείμενα των κανόνων διάταξης, τα μεγέθη σύγκρισης, είναι τύπου AutoGeneratedValue και Float τα οποία υλοποιούν το Comparable interface. Για τον κανόνα load, η επιστρεφόμενη τιμή σχηματίζεται από την αρνητική του φορτίου _load έτσι ώστε να εξασφαλιστεί κοινή φορά διάταξης των στοιχείων στα οποία οι κανόνες load και speed εφαρμόζονται (το ίδιο αποτέλεσμα επιτυγχάνεται αν ο κανόνας speed επέστρεφε την αρνητική τιμή της επεξεργαστικής ισχύς _speed). Επιπλέον ο κανόνας load έχει οριστεί με τις προδιαγραφές μεταβιβαστή ώστε να υπάρχει η δυνατότητα αντικατάστασης του από κάποιον εξωτερικό εκτελεστή.

Με βάση την παραπάνω υλοποίηση η περιοχή περιγραφής των κανόνων κατάταξης θα έχει την μορφή που φαίνεται στο Σχήμα 17.

```
<sorting_rules>
        <rule id="best_vpc">
                <mth name="load" order="1"/>
                <mth name="speed" order="2"/>
        </rule>
</sorting_rules>
```

Σχήμα17 – *Τμήμα από το XML έγγραφο διαμόρφωσης του συστήματος, περιοχή κανόνων κατάταξης.*

Κατά την υλοποίηση της προσομοίωσης δημιουργούνται πέντε αντικείμενα τύπου BestVPC, ένα για κάθε επιλογή, τα οποία φιλοξενούνται σε λίστα (π.χ. ένα αντικείμενο τύπου java.util.Vector) πάνω στην οποία θα εφαρμοστεί ο γενικός ταξινομητής. Ο γενικός ταξινομητής αρχικοποιείται με το διατεταγμένο σύνολο κανόνων που έχει οριστεί στο

έγγραφο διαμόρφωσης του συστήματος και ανακτάται μέσω του αντίστοιχου κωδικού ταυτοποίησης του, δηλαδή του “best_vpc”.

Έστω ότι οι τιμές _load και _speed για τις επιλογές vpc1, vpc2, vpc3, vpc4 και vpc5 είναι {21,2}, {23,8}, {20,4}, {20,2}, {20,3} αντίστοιχα. Η διάταξη των συνόλου των επιλογών μετά την εφαρμογή του πρώτου κριτηρίου, κατά αύξουσα σειρά καταλληλότητας, είναι {vpc2, vpc1, vpc3, vpc4, vpc5}, ενώ μετά την εφαρμογή του δεύτερου κριτηρίου η διάταξη μεταβάλλεται στην {vpc2, vpc1, vpc4, vpc5, vpc3}. Εάν επιθυμείται η μεταβολή της σειράς βαρύτητας των κανόνων, ο χρήστης απλώς ανταλλάσσει τις τιμές των αντίστοιχων ιδιοτήτων order στο έγγραφο διαμόρφωσης του συστήματος.

Στις περιπτώσεις όπου οι αλλαγές στην πολιτική τοποθεσίας της γεννήτριας εργασιών δεν καλύπτονται με την απλή αλλαγή της σειράς εφαρμογής των κανόνων στο σύνολο των επιλογών, ο χρήστης έχει την δυνατότητα να παρέμβει δυναμικά στο μοντέλο μέσω του μηχανισμού εναλλακτικών συμπεριφορών. Ο μηχανισμός αυτός επιτρέπει τον χρησιμοποίηση πλήθους εναλλακτικών παραμετροποιημένων κανόνων στην θέση εκείνων που στην κύρια υλοποίηση του μοντέλου προσομοίωσης έχουν οριστεί με τις προδιαγραφές ενός μεταβιβαστή.

Για παράδειγμα, έστω ότι η εκλογή του βέλτιστου διεκπεραιωτή πραγματοποιείται από μία συνάρτηση της μορφής a x (ταχύτητα επεξεργασίας διεκπεραιωτή) + b x (φορτίο διεκπεραιωτή), όπου a και b παράμετροι που προσδιορίζονται σταδιακά με δοκιμή και σφάλμα βάσει των πειραματικών αποτελεσμάτων της προσομοίωσης. Η διαδικασία «δοκιμή και σφάλμα» εμπλέκει την επανάληψη μεγάλου πλήθους πειραμάτων. Το γεγονός αυτό σχεδόν επιβάλει την χρήση του παραπάνω μηχανισμού αρκεί να αναλογιστούμε τις συνέπειες αν η διεξαγωγή κάθε πειράματος συνοδεύονταν και από αλλαγές τον υλοποίηση του μοντέλου προσομοίωσης.

Ο χρήστης πρέπει να αντιστοιχίσει σε κάποιο κανόνα με προδιαγραφές μεταβιβαστή ένα εξωτερικό εκτελεστή ο οποίος θα υλοποιεί την παραπάνω συνάρτηση. Επιπλέον οι παράμετροι a και b θα ορίζονται ως παράμετροι του συγκεκριμένου εξωτερικού εκτελεστή στο έγγραφο διαμόρφωσης του συστήματος.

Σύμφωνα με τα παραπάνω μία πιθανή υλοποίηση του εξωτερικού εκτελεστή καθώς επίσης η περιοχή περιγραφής των αντιστοιχήσεων μεταβιβαστών-εξωτερικών εκτελεστών και η περιοχή περιγραφής κανόνων ταξινόμησης στο έγγραφο διαμόρφωσης του συστήματος, απεικονίζονται στα Σχήματα 18, 19 και 20 αντίστοιχα.

```
public AutoGeneratedValue getSortingRule(InvokerArgument arg) {
        //ανάκτηση των παραμέτρων
        java.util.Properties params = arg.getInvokerParameters();
        float a = Long.parseLong(arg.getProperty("a"));
        float b = Long.parseLong(arg.getProperty("b"));

        //ανάκτηση στοιχείων από τον μεταβιβαστή
        BestVPC delegator = (BestVPC)arg.getDelegator();
        float load = delegator.getLoad();
        float speed = delegator.getSpeed();

        //συνάρτηση κανόνα
        float res = a*speed + b*load;

        return new AutoGeneratedValue(new Long(res));
}
```

Σχήμα 18 – *Τμήμα κώδικα, εξωτερικός εκτελεστής.*

Έστω ότι το πλήρες όνομα της κλάσης του μεταβιβαστή και του εξωτερικού εκτελεστή είναι coreModel.BestVPC και externalInvokers.Invokers αντίστοιχα.

```
<delegators-invokers>
        <delg-invk>
                <delegator class_name="coreModel.BestVPC" mth_name="load">
                <invoker class_name="externalInvokers.Invokers" mth_name=" getSortingRule ">
                        <param name="a" value="10"/>
                        <param name="b" value="20"/>
                </invoker>
        </delg-invk>
</delegators-invokers>
```

Σχήμα19 – *Τμήμα από το XML έγγραφο διαμόρφωσης του συστήματος, περιοχή ορισμού μεταβιβαστών/ εξωτερικών εκτελεστών.*

```
<sorting_rules>
        <rule id="best_vpc">
                <mth name="load" order="1"/>
        </rule>
</sorting_rules>
```

Σχήμα20 – *Τμήμα από το XML έγγραφο διαμόρφωσης του συστήματος, περιοχή ορισμού κανόνων κατάταξης.*

Ο μηχανισμός κανόνων κατάταξης σε συνδυασμό με τον μηχανισμό παραγωγής αυτομάτων τιμών προσδίδει επιπλέον ευελιξία στην προτεινόμενη πλατφόρμα προσομοίωσης. Ο χρήστης είναι σε θέση να μεταβάλει άμεσα και δυναμικά τα κριτήρια βάση των οποίων λαμβάνονται οι αποφάσεις ενός συστήματος (είτε με την αλλαγή της σειράς βαρύτητας των κριτηρίων είτε με την αντιστοίχηση εξωτερικών εκτελεστών) χωρίς να προβεί σε αλλαγές στην υλοποίηση του μοντέλου προσομοίωσης.

4.4 Μηχανισμός Μηνυμάτων (Messaging)

Στις ακόλουθες παραγράφους περιγράφεται το βασικό μοντέλο επικοινωνίας μεταξύ των οντοτήτων. Το μοντέλο αυτό χαρακτηρίζεται για την απλότητα και την λειτουργικότητα του η οποία προκύπτει από την γενικότερη αρχιτεκτονική της προτεινόμενης πλατφόρμας προσομοίωσης.

Τα μηνύματα επικοινωνίας μεταξύ των οντοτήτων διακρίνονται σε δύο κατηγορίες, στις *αιτήσεις δέσμευσης* (*bind-request*) και στα *μηνύματα καταστάσεως* ή *ενημέρωσης* (*state message*). Οι αιτήσεις δέσμευσης ή απλώς αιτήσεις είναι τα μηνύματα που αποστέλλονται από μία οντότητα για την δέσμευση ή την πρόσκληση δέσμευσης μίας ιδιότητας. Υπενθυμίζουμε ότι η αίτηση που αναφέρεται σε *δέσμευση* μίας ιδιότητας, καλούμενη και ως *αίτηση εξωτερικής δέσμευσης* (*out-bind request*), διαφοροποιείται από αυτή που αναφέρεται σε *πρόσκληση δέσμευσης* μίας ιδιότητας, καλούμενη και ως *αίτηση εσωτερικής δέσμευσης* (*in-bind request*). Η πρώτη αφορά την δέσμευση μίας αυτοφυούς ιδιότητας της οντότητας που λαμβάνει την αίτηση από την οντότητα που την αποστέλλει, ενώ η δεύτερη αφορά την δέσμευση μίας αυτοφυούς ιδιότητας της οντότητας που αποστέλλει την αίτηση από την οντότητα που την λαμβάνει (κοίτα επίσης παράγραφο 4.1).

Ένα μήνυμα αίτησης περιέχει πληροφορίες όπως ο κωδικός ταυτοποίησης του αποστολέα και του παραλήπτη, ο κωδικός ταυτοποίησης της ιδιότητας στην οποία αναφέρεται η αίτηση καθώς το ποσό και ο χρόνος δέσμευσης της ιδιότητας αυτής. Κάθε αίτηση δέσμευσης μπορεί να συνοδεύεται από ένα αντίστοιχο μήνυμα, καλούμενο ως *απάντηση αίτησης δέσμευσης*. Το μήνυμα αυτό χρησιμοποιείται από τον παραλήπτη προκειμένου να ενημερώσει τον αποστολέα της αντίστοιχης αίτησης αν το αίτημα του για δέσμευση μίας ιδιότητας έγινε αποδεκτό ή απερρίφθη.

Η αποστολή των αιτήσεων δέσμευσης λαμβάνει χώρα με δύο τρόπους, σύγχρονα ή ασύγχρονα. Στην πρώτη περίπτωση οι ενέργειες του αποστολέα αναστέλλονται έως ότου λάβει όλες τις απαντήσεις από τις αιτήσεις δέσμευσης για την συγκεκριμένη ιδιότητα (οπότε και η αποστολή απαντήσεων επιβάλλεται), ενώ στην δεύτερη περίπτωση ενημερώνεται απλώς για την λήψη των απαντήσεων.

Το μήνυμα ενημέρωσης περιέχει πληροφορίες για την κατάσταση των αυτοφυών ιδιοτήτων μίας οντότητας για μία δεδομένη χρονική στιγμή. Τα μηνύματα αυτά έχουν ιδιαίτερη σημασία αφού οι αποφάσεις μίας οντότητας λαμβάνονται βάσει τόσο της κατάστασης των αυτοφυών ιδιοτήτων της όσο και της κατάστασης των ιδιοτήτων του συνόλου των οντοτήτων που την ενδιαφέρουν, δηλαδή των ιδιοτήτων ενδιαφέροντος. Για παράδειγμα η απόφαση ενός διεκπεραιωτή για την πολιτική μεταφοράς που θα ακολουθήσει μπορεί να εξαρτάται από το φορτίο του και τις ταχύτητες επεξεργασίας των υπολοίπων διεκπεραιωτών του συστήματος.

Η κατάσταση μίας ιδιότητας περιγράφεται από ένα αντικείμενο τύπου PropertyState που περιέχει στοιχεία όπως το πλήθος των συνολικών, διαθέσιμων και δεσμευμένων μονάδων της ιδιότητας καθώς και το πλήθος των αιτήσεων δέσμευσης που εξυπηρετούνται από την οντότητα. Το τελευταίο μέγεθος αποτελεί ένα μέτρο κατακερματισμού και κατανομής της συγκεκριμένης ιδιότητας στις οντότητες του συστήματος. Το μέτρο αυτό έχει ιδιαίτερη σημασία και παίζει καθοριστικό ρόλο στην λειτουργία υπολογιστικών συστημάτων. Για παράδειγμα, η συμπεριφορά ενός διεκπεραιωτή ο οποίος επιβαρύνεται με φορτίο που προέρχεται από μία μονάδα μπορεί να είναι διαφορετική από αυτή στην περίπτωση όπου το ίδιο φορτίο προέρχεται αθροιστικά από πολλές διαφορετικές μονάδες ή αποτελείται από πολλά ανεξάρτητα φορτία.

Κάθε οντότητα έχει την δυνατότητα να διατηρεί μία λίστα, καλούμενη ως *λίστα καταστάσεως* (*state list*). Η λίστα αυτή περιέχει τα μηνύματα ενημέρωσης εκείνων των οντοτήτων που

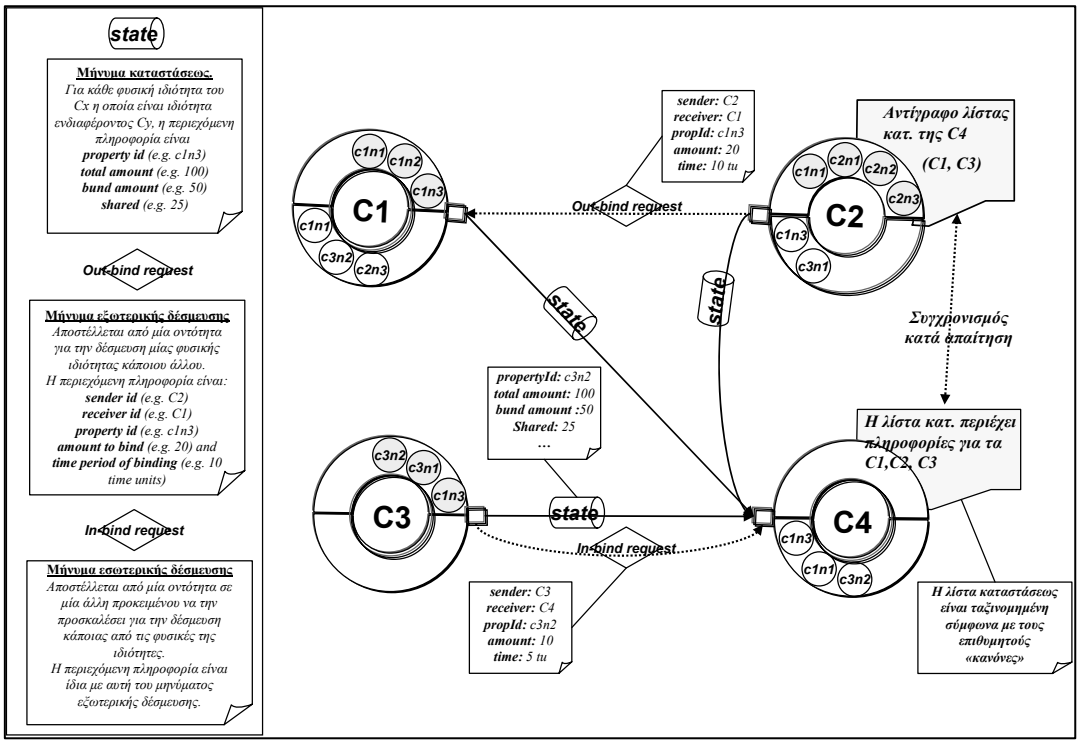
περιέχουν τουλάχιστον μία αυτοφυή ιδιότητα η οποία αποτελεί ιδιότητα ενδιαφέροντος της οντότητας στην οποία ανήκει η λίστα αυτή.

Η εισαγωγή της λίστας καταστάσεως στο μοντέλο της προτεινόμενης πλατφόρμας ουσιαστικά υλοποιεί μία από τις θεμελιώδεις αρχές λειτουργίας ενός υπολογιστικού συστήματος βάσει της οποίας οι μονάδες διατηρούν πληροφορίες για την κατάσταση του συστήματος. Πράγματι, κάθε λίστα αποτελεί το συνδετικό κρίκο της οντότητας στην οποία φιλοξενείται με το υπόλοιπο σύστημα αφού κάθε στοιχείο της περιγράφει το στιγμιότυπο της κατάστασης των ιδιοτήτων μίας εκ των υπολοίπων οντοτήτων για μία δεδομένη χρονική στιγμή.

Η λίστα ενημερώνεται ανάλογα με την πολιτική των οντοτήτων που αποστέλλουν τα μηνύματα καταστάσεως. Για παράδειγμα, μία οντότητα μπορεί να στέλνει μηνύματα ενημέρωσης με περιοδικό τρόπο ενώ κάποια άλλη με κριτήριο το μέγεθος μεταβολής των διαθέσιμων μονάδων μίας ιδιότητάς της.

Μία λίστα καταστάσεως μπορεί είτε να ανήκει πρωτογενώς στην οντότητα, καλούμενη και ως *πρωτογενής λίστα καταστάσεως*, είτε να λαμβάνεται από κάποια άλλη οντότητα και να αποτελεί ένα *αντίγραφο*. Η διαφοροποίηση μεταξύ πρωτογενούς και αντίγραφου λίστας καταστάσεως πραγματοποιείται για την προσαρμογή του προτεινόμενου μοντέλου στις *κεντρικές* (*centralize*) και *αποκεντρωτικές* (*decentralize*) πολιτικές πληροφόρησης που εφαρμόζονται στα υπολογιστικά συστήματα (παράγραφος 1.2.1).

Στο Σχήμα 21 απεικονίζεται οι αλληλεπιδράσεις μεταξύ των οντοτήτων ενός συστήματος, βάσει του μοντέλου που περιγράφτηκε.



Σχήμα 21 – *Μοντέλο επικοινωνίας μεταξύ των οντοτήτων.*

4.4.1 Εφαρμογή μηχανισμού κανόνων κατάταξης

Καθώς οι αποφάσεις μίας οντότητας λαμβάνονται βάσει του περιεχόμενου της λίστας καταστάσεως που διατηρεί, θεωρήθηκε σκόπιμο η ανάπτυξη ενός μηχανισμού διεξαγωγής αποφάσεων με εφαρμογή κριτηρίων πάνω στην λίστα αυτή. Αν κάθε στοιχείο της λίστας (μήνυμα ενημέρωσης) αποτελεί και μία επιλογή (μία εναλλακτική απόφαση για την οντότητα) είναι φανερό ότι ο εν λόγω μηχανισμός υλοποιείται πολύ εύκολα με την βοήθεια του μηχανισμού κανόνων κατάταξης που περιγράφτηκε στην παράγραφο 4.3.

Επειδή κάθε οντότητα πιθανώς εφαρμόζει διαφορετικά κριτήρια από τις υπόλοιπες τα οποία, σύμφωνα με τον μηχανισμό κανόνων διάταξης, ορίζονται στην κλάση υλοποίησης των επιλογών (δηλαδή των μηνυμάτων καταστάσεως στην συγκεκριμένη περίπτωση), ο εν λόγω μηχανισμός είναι απαραίτητο να παρέχει την δυνατότητα ορισμού της κλάσης υλοποίησης των μηνυμάτων ενημέρωσης μίας λίστας καταστάσεως. Κάθε τέτοια κλάση πρέπει να κληρονομεί από μια βασική (BaseStateMessage) η οποία φιλοξενεί τις καταστάσεις των ιδιοτήτων μίας οντότητας και παρέχει τις κατάλληλες μεθόδους για την ανάκτηση αυτών.

Οι απαραίτητες πληροφορίες για την εφαρμογή του μηχανισμού μηνυμάτων ορίζονται στην περιοχή που οριοθετείται από το στοιχείο <statelists> του εγγράφου διαμόρφωσης του συστήματος όπως φαίνεται στο Σχήμα 22. Κάθε λίστα καταστάσεως περιγράφεται από τις τιμές των ιδιοτήτων id, class_name, sortrule_id και acquired του στοιχείου <list>. Οι τιμές των ιδιοτήτων αυτών ορίζουν τον κωδικό ταυτοποίησης της λίστας, το όνομα της κλάσης υλοποίησης των μηνυμάτων καταστάσεως από τα οποία απαρτίζεται, τον κωδικό ταυτοποίησης του συνόλου των κανόνων κατάταξης που εφαρμόζονται και τον κωδικό ταυτοποίησης της οντότητας από όπου λαμβάνεται αντίγραφο της λίστας στην περίπτωση όπου αυτή δεν διατηρείται πρωτογενώς στην οντότητα, αντίστοιχα.

Ο κωδικός ταυτοποίησης της λίστας χρησιμοποιείται στην περιοχή περιγραφής των οντοτήτων έτσι ώστε να καθοριστεί ο τύπος των μηνυμάτων καταστάσεως που στελεχώνουν την λίστα μίας οντότητας και κατά συνέπεια τον τύπο μηνυμάτων καταστάσεως που αποστέλλουν οι υπόλοιπες οντότητες σε αυτή. Ο κωδικός ταυτοποίησης του συνόλου των κανόνων κατάταξης αναφέρεται σε κάποιον από αυτούς που έχουν οριστεί στην αντίστοιχη περιοχή του εγγράφου διαμόρφωσης.

Πρέπει να σημειωθεί ότι στην περίπτωση που η τιμή της ιδιότητας acquired του στοιχείου <statelists> είναι ορισμένη, οι τιμές των υπόλοιπων ιδιοτήτων δεν λαμβάνονται υπόψη αφού η αντίστοιχη οντότητα δεν διατηρεί πρωτογενή λίστα καταστάσεως και δεν απαιτείται η ενημέρωση της από τις υπόλοιπες οντότητες.

Επιπλέον, η χρήση του δεσμευμένου από το σύστημα κωδικού λίστας default υποδεικνύει τον σχηματισμό λίστας από μηνύματα καταστάσεως της βασικής κλάσης και την εφαρμογή προκαθορισμένου συνόλου εικονικών κανόνων το οποίο δεν έχει καμία επίδραση στην διάταξη των στοιχείων της. Στην περίπτωση που δεν οριστεί κάποια τιμή, η οντότητα δεν περιέχει λίστα με αποτέλεσμα οι αποφάσεις της να μην συνεκτιμούν την συνολική κατάσταση του συστήματος.

```
<statelists>
        <list id="..." class_name="..." sortrule_id="..." acquired="...">
        ...
</statelists>
```

Σχήμα 22 – *Τμήμα από το XML έγγραφο διαμόρφωσης του συστήματος, περιοχή ορισμού λιστών καταστάσεως.*

4.4.1.1 Παραδείγματα

Για την καλύτερη κατανόηση του παραπάνω μηχανισμού θα προσαρμόσουμε το παράδειγμα εφαρμογής των κανόνων κατάταξης της παραγράφου 4.3.3 στις λίστες καταστάσεως.

Έτσι, στα τμήματα του XML εγγράφου διαμόρφωσης του συστήματος που αναφέρονται τα Σχήματα 19 και 20 προσθέτουμε τα τμήματα που περιγράφουν τις οντότητες του υπολογιστικού συστήματος και τις λίστες καταστάσεως όπως φαίνεται στα Σχήματα 23 και 24 αντίστοιχα. Επιπλέον, λαμβάνουμε υπόψη την υλοποίηση του εξωτερικού εκτελεστή του Σχήματος 18.

```
<components>
        <component id="vpc" replicas="5"
                                implementation=" coreModel.MySComponent ">
                <properties>
                        <belong>
                                <prop name="load">
                                <prop name="speed">
                        </belong>
                </properties>
        </component>
        <component id="loadgenerator" listid="mystatelist">
                <properties>
                        <interest>
                                <prop name="load">
                        </interesr>
                </properties>
        </component>
</components>
```

Σχήμα 23 – *Τμήμα από το XML έγγραφο διαμόρφωσης του συστήματος, περιοχή ορισμού οντοτήτων.*

```
<statelists>
        <list id="mystatelist" class_name="coreModel.BestVPC"  sortrule_id=" best_vpc">
</statelists>
```

Σχήμα 24 – *Τμήμα από το XML έγγραφο διαμόρφωσης του συστήματος, περιοχή ορισμού λιστών καταστάσεως.*

Στην περιοχή περιγραφής των οντοτήτων ορίζουμε τους πέντε διεκπεραιωτές με κωδικούς vpc, vpc1, vpc2, vpc3 και vpc4, σύμφωνα με την ονοματολογία που ακολουθείται κατά την παραγωγή πολλαπλών αντιγράφων (παράγραφος 4.1.2). Επειδή οι διεκπεραιωτές δεν λαμβάνουν μέρος στις αποφάσεις του συστήματος, δεν υπάρχει λόγος διατήρησης λίστας καταστάσεως από αυτές. Έτσι, η τιμή της ιδιότητας listid στο αντίστοιχο στοιχείο <component> δεν είναι ορισμένη. Επιπλέον, η κλάση υλοποίησης των διεκπεραιωτών είναι η coreModel.MySComponent.

Κάθε διεκπεραιωτής έχει δύο αυτοφυείς ιδιότητες, το φορτίο και την επεξεργαστική ισχύ. Η επεξεργαστική ισχύς αποτελεί μία ιδιότητα όπου το σύνολο των δεσμευμένων μονάδων της παραμένει σταθερό και ίσο με το μηδέν στο χρόνο, σε αντίθεση με το αντίστοιχο μέγεθος της ιδιότητας «φορτίο» όπου μεταβάλλεται. Οι συνολικές μονάδες της ιδιότητας «φορτίο»

αντιστοιχούν στο μέγιστο φορτίο με το οποίο μπορεί να επιβαρυνθεί ο διεκπεραιωτής ενώ οι δεσμευμένες μονάδες της ίδιας ιδιότητας χαρακτηρίζουν το επιβαλλόμενο φορτίο για μία δεδομένη χρονική στιγμή. Η ιδιότητα «επεξεργαστική ισχύς» περιγράφεται από την τιμές των συνολικών ή των διαθέσιμων μονάδων της αφού αυτές συμπίπτουν.

Η γεννήτρια φορτίου ορίζεται με το όνομα loadgenerator και είναι αυτή που αποφασίζει σε ποίους διεκπεραιωτές θα στείλει τα παραγόμενα φορτία. Έτσι, η ύπαρξη πρωτογενούς λίστας καταστάσεως, έστω με κωδικό mystatelist, και η εφαρμογή συνόλου κανόνων σε αυτή, έστω με κωδικό best_vpc, θεωρείται απαραίτητη.

Η υλοποίηση των μηνυμάτων καταστάσεως πραγματοποιείται από την κλάση coreModel.BestVPC η υλοποίηση της οποίας φαίνεται στο σχήμα 25.

```
public class BestVPC extends BaseStateMessage implements AutoGenValueMechInterface,
                                                                    java.lang.Cloneable {

        public float getLoad(){
                PropertyState psLoad = (PropertyState)propsState.get("load");
                return (float)psLoad.getBoundAmount();
        }

        public float getSpeed(){
                PropertyState psSpeed = (PropertyState)propsState.get("speed");
                return (float)psSpeed.getTotalAmount();
        }

        //ορισμός κριτηρίων
        public  AutoGeneratedValue load(Object obj) {
                return new AutoGeneratedValue (new Float(-getLoad()));
        }

        public Float speed() {return new Float(getSpeed());}

        //υλοποίηση των μεθόδων των interface
        public String delegatorId(){…}
        public Object clone(){…}

}
```

Σχήμα 25 – *Τμήμα κώδικα, υλοποίηση ενός μηνύματος καταστάσεως.*

Η κλάση BestVPC κληρονομεί από την κλάση BaseStateMessage αφού χρησιμοποιείται για την δημιουργία μηνυμάτων καταστάσεως ενώ παράλληλα υλοποιεί τις διεπιφάνειες AutoGenValueMechInterface και java.lang.Cloneable εφόσον περιέχει μεταβιβαστές. Βασικό πεδίο της κλάσης BaseStateMessage είναι το propsState το οποίο φιλοξενεί τις καταστάσεις των αυτοφυών ιδιοτήτων μίας οντότητας που αποτελούν ιδιότητες ενδιαφέροντος για την οντότητα που διατηρεί την πρωτογενή λίστα.

Οι ιδιότητες ενδιαφέροντος για την γεννήτρια φορτίου είναι αυτές με κωδικούς load και speed. Λαμβάνοντας υπόψη ότι η κατάσταση μιας ιδιότητας ανακτάται βάσει του κωδικού της, η εφαρμογή των μεθόδων get("load") και get("speed") στο propsState επιστρέφει την κατάσταση των ιδιοτήτων load και speed αντίστοιχα με την μορφή ενός αντικειμένου PropertyState.

Σημειώνεται ότι η ανάκτηση της τιμής της ιδιότητας load πραγματοποιείται μέσω της μεθόδου getBoundAmount. Η μέθοδος αυτή επιστρέφει το πλήθος των δεσμευμένων μονάδων μίας ιδιότητας όπου στην συγκεκριμένη περίπτωση αναπαριστά το επιβαλλόμενο φορτίο. Αντιθέτως, η ανάκτηση της τιμής της ιδιότητας speed γίνεται μέσω της μεθόδου getTotalAmount ή getAvailableAmount οι οποίες επιστρέφουν τις συνολικές και τις διαθέσιμες μονάδες μιας ιδιότητας αντίστοιχα.

Ο μηχανισμός αποστολής μηνυμάτων εκμεταλλεύεται πλήρως το μοντέλο αλληλεπίδρασης των οντοτήτων παρέχοντας ένα σύνολο μεθόδων που χαρακτηρίζονται για την απλότητα και ευελιξία τους. Για παράδειγμα, η μέθοδος sendStateMessage(String propertyId) αποστέλλει μηνύματα καταστάσεως στις οντότητες που έχουν ιδιότητα ενδιαφέροντος με κωδικό ταυτοποίησης propertyId και ταυτόχρονα διατηρούν πρωτογενή λίστα καταστάσεως. Επιπλέον, κάθε μήνυμα συμφωνεί με τις προδιαγραφές που έχουν οριστεί για την λίστα καταστάσεως της οντότητας όπου προορίζεται.

4.5 Μηχανισμός Όρων-Ενεργειών (Conditions – Actions mechanism)

Μία πλατφόρμα πρέπει να παρέχει τα κατάλληλα εργαλεία σε επίπεδο υλοποίησης (API – Application Programming Interface) τα οποία καλύπτουν και διευκολύνουν την ανάπτυξη ενός μοντέλου προσομοίωσης. Τα εργαλεία αυτά είτε θα πρέπει να αντιμετωπίζουν πρωτογενώς τα πιο σημαντικά και τα πιο συνήθη σενάρια προσομοίωσης είτε θα πρέπει να είναι σχεδιασμένα με γενικό τρόπο ώστε να μπορούν να προσαρμοστούν στις ανάγκες του εκάστοτε προβλήματος.

Στις παραγράφους που ακολουθούν περιγράφονται κάποια ενδεικτικά σενάρια προσομοίωσης υπολογιστικών συστημάτων και πώς αυτά μπορούν να ενταχθούν σε ένα γενικότερο πλαίσιο συμπεριφοράς το οποίο υλοποιείται από την προτεινόμενη πλατφόρμα προσομοίωσης.

Ένα σύνηθες μοντέλο διαπραγμάτευσης μεταξύ των διεκπεραιωτών ενός υπολογιστικού συστήματος κατά την εφαρμογή αποκεντρωτικών πολιτικών τοποθεσίας είναι το ακόλουθο: ο διεκπεραιωτής εκδηλώνει την επιθυμία του να λειτουργήσει ως δότης ή δέκτης φορτίου με την αποστολή μηνύματος-αίτησης στους υπόλοιπους διεκπεραιωτές, τους υποψήφιους δέκτες και δότες αντίστοιχα. Οι υποψήφιοι διεκπεραιωτές αφού αποτιμήσουν την αίτηση απαντούν με μηνύματα αποδοχής ή απόρριψης ανάλογα με το αν συμφωνούν ή απορρίπτουν την συγκεκριμένη αίτηση αντίστοιχα. Τα μηνύματα αποδοχής ενδεχομένως συμπεριλαμβάνουν και την κατάσταση των αντίστοιχων διεκπεραιωτών που τα αποστέλλουν αφού η επιλογή του ιδανικού υποψηφίου θα πραγματοποιηθεί μεταξύ αυτών που αποδέχονται την αίτηση να λάβουν μέρος στην μεταφορά φορτίου. Η επιλογή του βέλτιστου δέκτη/δότη από τον διεκπεραιωτή που απέστειλε το μήνυμα-αίτηση για μεταφορά φορτίου, ανεξαρτήτως των εφαρμοζόμενων κριτηρίων, πραγματοποιείται με την πλήρωση κάποιων συνθηκών. Για παράδειγμα ο διεκπεραιωτής μπορεί να αποφασίζει
α) με την λήψη των απαντήσεων από όλους του υποψήφιους δέκτες/δότες στην περίπτωση που επιθυμεί να συνεκτιμήσει όλες τις πληροφορίες του συστήματος,
β) με την λήψη κάποιων μηνυμάτων αποδοχής μέσα σε ένα συγκεκριμένο χρονικό διάστημα αφού μπορεί να θεωρεί ότι οι καθυστερημένες απαντήσεις συνήθως προέρχονται από ακατάλληλους υποψήφιους διεκπεραιωτές (για παράδειγμα λόγω της μεγάλης αποστάσεως μεταξύ τους),
γ) με την λήψη συγκεκριμένου πλήθους μηνυμάτων αποδοχής για παρόμοιους λόγους με την δεύτερη περίπτωση.

Το παραπάνω παράδειγμα μπορεί να ενταχθεί σε ένα ευρύτερο μοντέλο συμπεριφοράς. Στο μοντέλο αυτό, ως *όρος* (condition set) ορίζεται ένα σύνολο από *συνθήκες* (*conditions*) κάθε μία από τις οποίες μπορεί να χαρακτηριστεί ως *αληθή* ή *ψευδή* στην περίπτωση που

ικανοποιείται ή δεν ικανοποιείται (ή διαφορετικά εκπληρώνεται ή δεν εκπληρώνεται) αντίστοιχα. Ο χαρακτηρισμός ενός όρου ως *αληθή ή ψευδή* δεν είναι αναγκαίο να προκύπτει από την ικανοποίηση ή την μη ικανοποίηση αντίστοιχα όλων των συνθηκών που τον απαρτίζουν αλλά από τον συνδυασμό αληθών και ψευδών συνθηκών. Επίσης, ως *ενέργεια* ορίζεται η εκτέλεση μίας διεργασίας, μίας μεθόδου σε επίπεδο υλοποίησης που ενσαρκώνει μία συγκεκριμένη συμπεριφορά. Το κεντρικό σημείο του μοντέλου είναι η δυνατότητα συσχέτισης μεταξύ οντοτήτων, ενεργειών και όρων. Βάσει της συσχέτισης αυτής η εκπλήρωση ή όχι ενός όρου μέσα σε χρονικό διάστημα που καθορίζεται από μία οντότητα, υπαγορεύει την εφαρμογή της κατάλληλης ενέργειας και κατ' επέκταση της συμπεριφοράς της οντότητας αυτής.

Το παραπάνω μοντέλο παρόλο που φαίνεται υπερβολικά απλό η γενικό, μπορεί να αποτελέσει τον πυρήνα για την υλοποίηση πλήθος σεναρίων προσομοίωσης. Για παράδειγμα στο προαναφερθέν τυπικό σενάριο πολιτικής τοποθεσίας ως όρος μπορεί να χαρακτηριστεί η λήψη των απαντήσεων από όλους του υποψήφιους δέκτες/ δότες φορτίου και σαν ενέργεια η αποστολή ή παραλαβή του φορτίου στον βέλτιστο δέκτη ή δότη αντίστοιχα.

4.5.1 Υλοποίηση

Στις παραγράφους που ακολουθούν δίνεται λεπτομερής περιγραφή της υλοποίησης του μοντέλου που περιγράφτηκε στην προηγούμενη παράγραφο.

Οι κλάσεις AbstractLock και LockCondition αποτελούν τις βασικές κλάσεις υλοποίησης ενός *όρου* και μίας *συνθήκης* αντίστοιχα. Από την AbstractLock προκύπτουν οι τρεις τύποι όρων οι οποίοι διαφοροποιούνται στις προϋποθέσεις βάσει των οποίων μπορούν να χαρακτηριστούν ως αληθείς. Συγκεκριμένα, ανάλογα με τον τύπο ένας όρος ικανοποιείται όταν
α) το σύνολο των συνθηκών του είναι αληθής (κλάση LockAll),
β) μία τουλάχιστον από τις συνθήκες του είναι αληθής (κλάση LockAny) και
γ) οι καταστάσεις (αληθής ή ψευδής) των συνθηκών από τις οποίες αποτελείται συμφωνούν με ένα προκαθορισμένο σύνολο καταστάσεων (κλάση LockPredifined).

Για παράδειγμα η ικανοποίηση ενός όρου μπορεί να συμβαίνει όταν οι καταστάσεις των συνθηκών του, cond1, cond2 και cond3 είναι αληθής, ψευδής και αληθής αντίστοιχα. Η εισαγωγή του τελευταίου τύπου έχει σαν στόχο την κάλυψη όλων των πιθανών εκδοχών ικανοποίησης ενός όρου συμπεριλαμβανομένου και των δύο πρώτων τύπων. Ο ορισμός των συνθηκών ενός όρου καθώς και η τροποποίηση των καταστάσεών τους πραγματοποιείται μέσω των κατάλληλων μεθόδων που παρέχονται από την κλάση AbstractLock (μέθοδοι addCondition(…), removeCondition(…), setConditionStatus(…), περισσότερες πληροφορίες στο API).

Σημειώνεται ότι ο όρος έχει δύο καταστάσεις λειτουργίας, την *αποδέσμευση με την ολοκλήρωση* (UNblock With EXpiration, UNWEX εν συντομία) και την *αποδέσμευση με την λήξη* (UNblock With CompLetion, UNWCL) (περισσότερες πληροφορίες από τον σχολιασμό των σταθερών UNBLOCK_WITH_COMPLETION και UNBLOCK_WITH_EXPIRATION στο API) οι οποίες χρησιμοποιούνται και σαν δείκτες για την διαμόρφωση της κατάλληλης συμπεριφοράς των κλάσεων που συσχετίζονται με την ύπαρξη όρων, όπως συμβαίνει στην περίπτωση της κλάσης Executor.

Επιπλέον, σε κάθε όρο είναι δυνατή η καταχώρηση αντικειμένων (listeners) που επιθυμούν να ειδοποιηθούν για σημαντικά γεγονότα που λαμβάνουν χώρα κατά την διάρκεια ζωής του, όπως είναι η τροποποίηση των καταστάσεων των συνθηκών του και η ικανοποίηση του. Τα καταχωρημένα αντικείμενα πρέπει να προέρχονται από κλάσεις που υλοποιούν

συγκεκριμένες διεπιφάνειες ανάλογα με τα γεγονότα για τα οποία επιθυμούν να λάβουν γνώση (διεπιφάνειες ConditionModificationListener, ConditionsetExpirationListener).

Η *ενέργεια* ενσαρκώνεται από την κλάση Action. Η κλάση αυτή περιέχει όλες τις απαραίτητες πληροφορίες για την δυναμική κλήση μίας μεθόδου, όπως είναι το όνομα και τα ορίσματα της μεθόδου καθώς και το αντικείμενο πάνω στο οποίο θα εφαρμοστεί η μέθοδος αυτή.

Ο συνδετικός κρίκος μεταξύ όρων-ενεργειών και οντοτήτων καθώς επίσης η λογική του προτεινόμενου μοντέλου, καλούμενου και ως *μηχανισμός όρων-ενεργειών*, υλοποιείται από τις κλάσεις Executor και TaskDescriptor. Οι κλάσεις αυτές ουσιαστικά αποτελούν πολύ πιο ευέλικτες και δυναμικές υλοποιήσεις των κλάσεων java.util.Timer και java.util.TimerTask αντίστοιχα οι οποίες περιέχονται στο πυρήνα υλοποίησης της γλώσσας προγραμματισμού java.

Συγκεκριμένα, κάθε αντικείμενο TaskDescriptor περιγράφει ένα σύνολο ενεργειών που απαρτίζεται από α) την *ενέργεια σε περίπτωση επιτυχίας* (Action IN Success, AINS εν συντομία), β) την *ενέργεια σε περίπτωση αποτυχίας* (Action IN Failure, AINF εν συντομία) και γ) την *ενέργεια παροχής του επόμενου προγραμματισμένου χρόνου εκτέλεσης* (Action IN giving the next scheduled Time, AINT εν συντομία).

Σημειώνεται ότι, για λόγους που θα γίνουν περισσότερο αντιληπτοί στην συνέχεια οι μέθοδοι που περιγράφονται από τις δύο πρώτες ενέργειες δεν είναι απαραίτητο να ακολουθούν κάποιο πρότυπο ορισμού. Αντιθέτως η συσχετιζόμενη μέθοδος με την τελευταία ενέργεια είναι απαραίτητο να επιστρέφει ένα αντικείμενο τύπου java.lang.Long ή AutoGeneratedValue του οποίου η τιμή (δηλαδή το αντικείμενο που επιστρέφεται μέσω της μεθόδου value()) πρέπει να είναι ένα αντικείμενο τύπου java.lang.Long επίσης. Αυτό είναι απαραίτητο αφού η επιστρεφόμενη τιμή της AINT εκφράζει χρόνο.

Ο TaskDescriptor διαθέτει δύο καταστάσεις λειτουργίας, την *επανάληψη σε κάθε περίπτωση* (Repeat In Any Case, RIAC εν συντομία) και την *διακοπή σε περίπτωση αποτυχίας* (Stop In FaiLure, SIFL εν συντομία). Επιπλέον, μπορεί να συσχετιστεί με έναν όρο οποιοδήποτε τύπου. Η κατάσταση λειτουργίας καθώς και ο όρος έχουν άμεση επίδραση στην διαμόρφωση της συμπεριφοράς του Executor όπως θα δούμε στην συνέχεια.

Ο Executor είναι υπεύθυνος για τον προγραμματισμό και την επαναληπτική εκτέλεση των ενεργειών που περιγράφονται από τους TaskDescriptor βάσει κάποιων κανόνων. Οι κανόνες αυτοί εξαρτώνται άμεσα από την ικανοποίηση ή μη ικανοποίηση του όρου με τον οποίο συνδέεται ο TaskDescriptor και υποδεικνύουν την ενέργεια που πρέπει να εφαρμοστεί (μία από τις AINS και AINF) καθώς και τον χρόνο εκτέλεσης της.

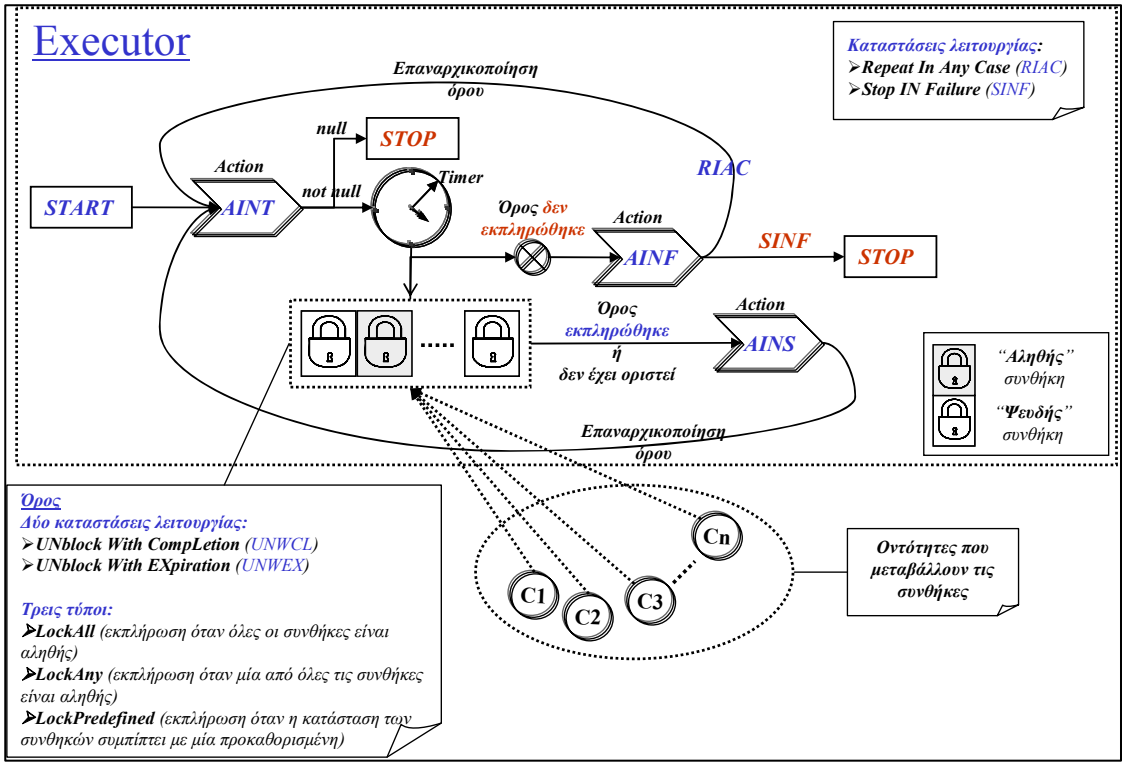
Η επιστρεφόμενη τιμή της AINT, καλούμενη ως *προγραμματισμένος χρόνος εκτέλεσης* (Scheduled execution TIME, STime εν συντομία), καθορίζει το μέγιστο χρονικό διάστημα σε ms μέσα στο οποίο πραγματοποιείται η εκτέλεση μίας εκ των ενεργειών AINS και AINF του TaskDescriptor. Ο προγραμματισμός εκτέλεσης μιας ενέργειας καθίσταται ανέφικτος και συνεπώς η δραστηριότητα του TaskDescriptor τερματίζεται όταν η επιστρεφόμενη τιμή της AINT είναι null.

Οι κανόνες που διέπουν την συμπεριφορά του Executor μπορούν να κατηγοριοποιηθούν με βάση την ύπαρξη ή την μη ύπαρξη όρου στον TaskDescriptor. Έτσι,
α) Στην περίπτωση όπου ο TaskDescriptor δεν συνδέεται με κάποιον όρο, ο Executor προγραμματίζει την επαναληπτική εκτέλεση της ενέργειας AINS ανά χρονικά διαστήματα που καθορίζονται από την επιστρεφόμενη τιμή της ενέργειας AINT. Η εκτέλεση της AINS τερματίζεται όταν η επιστρεφόμενη τιμή της AINT είναι null.
β) Στην περίπτωση όπου ο TaskDescriptor συσχετίζεται με κάποιον όρο η συμπεριφορά του Executor υπαγορεύεται από τους ακόλουθους κανόνες:

1β) Εάν ο όρος ικανοποιηθεί μέσα στον προγραμματισμένο χρόνο εκτέλεσης τότε η εκτέλεση της ενέργειας AINS λαμβάνει χώρα είτε την στιγμή που ικανοποιείται ο όρος είτε με την πάροδο του χρόνου STime αν η κατάσταση λειτουργίας του όρου είναι UNWCL ή UNWEX αντίστοιχα. Στην συνέχεια ο Executor αφού αρχικοποιήσει τις συνθήκες του όρου προγραμματίζει ξανά, εφόσον αυτό είναι εφικτό (δηλαδή η επιστρεφόμενη τιμή της AINT δεν είναι null) , την εκτέλεση του TaskDescriptor.

2β) Εάν ο όρος δεν ικανοποιηθεί με το πέρας του STime τότε πραγματοποιείται η εκτέλεση της AINF, εφόσον αυτή έχει οριστεί. Στην συνέχεια, εάν η κατάσταση λειτουργίας του TaskDescriptor είναι RIAC, ο Executor αφού αρχικοποιήσει τις συνθήκες του όρου, προγραμματίζει ξανά (εφόσον αυτό είναι εφικτό) την εκτέλεση του TaskDescriptor. Αντιθέτως, στην περίπτωση που η κατάσταση λειτουργίας είναι SINF η δραστηριότητα του TaskDescriptor τερματίζεται.

Στο Σχήμα 26 παρουσιάζεται συνοπτικά η λειτουργία του μηχανισμού όρων-ενεργειών.



Σχήμα 26 - *Λειτουργία του μηχανισμού όρων-ενεργειών.*

Η παραπάνω υλοποίηση, ανεξάρτητα από την ύπαρξη όρων, αποτελεί μια σαφώς πιο ευέλικτη και γενικότερη εκδοχή του μοντέλου λειτουργίας των αντίστοιχων κλάσεων java.util.Timer και java.util.TimerTask. Συγκεκριμένα, οι ενέργειες απαλλάσσονται από την υποχρέωση να ακολουθούν ένα περιοριστικό πρότυπο ορισμού αφού η εφαρμογή τους πραγματοποιείται με δυναμικό τρόπο (σε αντίθεση με την java.util.TimerTask). Ο ρυθμός εκτέλεσης μίας ενέργειας δεν είναι απαραίτητα σταθερός αλλά προσδιορίζεται δυναμικά από την AINT, ενώ η δυνατότητα διαμόρφωσης των κριτηρίων διακοπής της εφαρμογής μίας ενέργειας (null επιστρεφόμενη τιμή από την AINT) επιτρέπει τον καθορισμό του συνολικού αριθμού των κλήσεων της (συμπεριφορές που δεν υποστηρίζονται από την java.util.Timer).

Η εισαγωγή του όρου προσδίδει στο μοντέλο μία άλλη διάσταση και δυναμική. Ο όρος επηρεάζει άμεσα την συμπεριφορά μίας οντότητας αφού η εκπλήρωση ή μη-εκπλήρωση του υπαγορεύει την εκτέλεση των αντίστοιχων μεθόδων. Επειδή η ικανοποίηση ενός όρου μπορεί να εξαρτάται από συνθήκες η οποίες καθορίζονται και τροποποιούνται από οντότητες διαφορετικές από αυτή όπου επιδρά άμεσα, ένας όρος μπορεί να διαμορφώσει τους κανόνες

αλληλεπίδρασης μεταξύ των οντοτήτων του συστήματος. Για παράδειγμα σε ένα υπολογιστικό σύστημα, η εφαρμογή της πολιτικής τοποθεσίας μπορεί να συνδέεται με την επαλήθευση ενός όρου ο οποίος σχηματίζεται από συνθήκες που κάθε μία από αυτές υποδηλώνει αν μία οντότητα έχει αποστείλει μήνυμα ενημέρωσης ή όχι. Στην περίπτωση αυτή, η ενέργεια μίας οντότητας η οποία μεταφράζεται ως η επιλογή του βέλτιστου διεκπεραιωτή και η αποστολή του φορτίου σε αυτόν, εξαρτάται από την συμπεριφορά των υπόλοιπων οντοτήτων του συστήματος, δηλαδή την αποστολή μηνύματος ενημέρωσης.

Η ευελιξία του μηχανισμού όρων-ενεργειών πολλαπλασιάζεται με την εφαρμογή του μηχανισμού εναλλακτικών συμπεριφορών στις ενέργειες που περιγράφονται από τον TaskDescriptor. Αυτό είναι δυνατόν αρκεί οι αντίστοιχοι μέθοδοι των AINS, AINF και AINT να ακολουθούν τις προδιαγραφές ορισμού ενός μεταβιβαστή.

Σημειώνεται ότι η εκτέλεση των AINS και AINF πραγματοποιείται σε ξεχωριστές ανεξάρτητες διεργασίες (processes) ή νήματα (threads) οι οποίες αρχικοποιούνται και διαχειρίζονται από αντικείμενα τύπου ThreadPool. Λεπτομέρειες για τα χαρακτηριστικά και την λειτουργία της ThreadPool περιέχονται στο API. Αυτό που πρέπει να σημειωθεί είναι ότι σε κάθε νήμα, το οποίο υλοποιείται μέσω της κλάσης ThreadPool.Worker, είναι δυνατόν να καταχωρηθούν αντικείμενα τα οποία ενδιαφέρονται να ειδοποιηθούν για την ολοκλήρωση και το αποτέλεσμα της ενέργειας που εκτελείται στο νήμα αυτό. Τα αντικείμενα αυτά πρέπει να προέρχονται από κλάσεις που υλοποιούν την κατάλληλη διεπιφάνεια (διεπιφάνεια ActionCompletionListener).

4.5.2 Μηχανισμός ανίχνευσης διακεκριμένων γεγονότων (Event Detection)

Συμπλήρωμα του μηχανισμού όρων-ενεργειών αποτελεί ο *μηχανισμός ανίχνευσης διακεκριμένων γεγονότων*. Ο μηχανισμός αυτός έχει σαν στόχο την συσχέτιση γεγονότων που συμβαίνουν κατά την διάρκεια της προσομοίωσης ενός συστήματος με την κλήση αντίστοιχων προκαθορισμένων μεθόδων. Βάσει των κανόνων που διέπουν την λειτουργία του μοντέλου της προτεινόμενης πλατφόρμας, το γεγονός που διαδραματίζει ίσως τον σημαντικότερο ρόλο στην εξέλιξη της συμπεριφορά των οντοτήτων και κατ'επέκταση ολόκληρου του συστήματος είναι η μεταβολή της κατάστασης των ιδιοτήτων τους. Για παράδειγμα κατά την προσομοίωση υπολογιστικών πλεγμάτων η μεταβολή του φορτίου σε ένα διεκπεραιωτή μπορεί να προκαλέσει την ενεργοποίηση της εφαρμοζόμενης σε αυτόν πολιτικής μεταφοράς και πληροφόρησης.

Πυρήνας του μηχανισμού αυτού, όπως και στον μηχανισμό εναλλακτικών συμπεριφορών, είναι το aspect PropertyChangedDetector. Το aspect αυτό είναι υπεύθυνο για την ανίχνευση των γεγονότων ενδιαφέροντος και την εκτέλεση αντίστοιχων προκαθορισμένων μεθόδων σε *ανεξάρτητες διεργασίες* (*threads*). Έτσι, στην περίπτωση μεταβολής της καταστάσεως των αυτοφυών ιδιοτήτων μιας οντότητας πραγματοποιείται αυτόματη κλήση της μεθόδου naturalPropertyChanged. Η μέθοδος αυτή είναι ορισμένη στην βασική κλάση υλοποίησης των οντοτήτων SComponent χωρίς να πραγματοποιεί κάποια λειτουργία. Η λειτουργία θα πρέπει να υλοποιηθεί, εφόσον είναι επιθυμητό, με *υπερκάλυψη* (*overriding*) της μεθόδου μέσω της κλάσης υλοποίησης των οντοτήτων. Το όρισμα της μεθόδου, ένα αντικείμενο τύπου NaturalPropertyChangedEvent, περιέχει πληροφορίες για την κατάσταση του συνόλου των αυτοφυών της ιδιοτήτων της οντότητας καθώς επίσης και την κατάσταση της μεταβληθείσας ιδιότητας ξεχωριστά.

4.5.3 Παραδείγματα

Για την πληρέστερη κατανόηση των παραπάνω μηχανισμών εξετάζουμε τα ακόλουθα παραδείγματα. Τα παραδείγματα αυτά αναφέρονται στις δύο πλέον τυπικές πολιτικές πληροφόρησης που εφαρμόζονται σε ένα υπολογιστικό σύστημα, τις περιοδικές και τις πολιτικές κατωφλίου. Στις περιοδικές πολιτικές ο διεκπεραιωτής ενημερώνει τις υπόλοιπες οντότητες για την κατάσταση του ανά τακτά χρονικά διαστήματα, ενώ στις πολιτικές κατωφλίου η ενημέρωση πραγματοποιείται όταν το ποσό ή η μεταβολή ενός ή πολλών χαρακτηριστικών του διεκπεραιωτή υπερβαίνει ή υπολείπεται ενός ορίου.

Παράδειγμα 1
Με την βοήθεια του μηχανισμού όρων-ενεργειών η υλοποίηση μίας περιοδικής πολιτικής πραγματοποιείται πολύ εύκολα. Έστω ότι ο ρυθμός αποστολής μηνυμάτων ενημέρωσης ενός διεκπεραιωτή (σε sec) προσδιορίζεται από την σχέση stateRate = a*Rand(0…1), όπου a σταθερά και Rand(0…1) γεννήτρια τυχαίων αριθμών μεταξύ 0 και 1. Αν θεωρήσουμε έναν TaskDescriptor όπου οι ενέργειες AINT και AINS περιγράφουν την παραπάνω συνάρτηση και την μέθοδο αποστολής μηνυμάτων καταστάσεως αντίστοιχα, τότε μία πιθανή υλοποίηση της περιοδικής πολιτικής απεικονίζεται στα ακόλουθα σχήματα.

```
public AutoGeneratedValue getScheduleTime(Object arg) {
        return new AutoGeneratedValue(new Long(0));
}
```

Σχήμα 27 – *Τμήμα κώδικα, μεταβιβαστής.*

Ας υποθέσουμε ότι η μέθοδος getScheduleTime που ορίζεται στην κλάση coreModel.Timer είναι αυτή που προορίζεται για την υλοποίηση της σχέσης stateRate. Επειδή η stateRate μπορεί να έχει οποιαδήποτε μορφή επιλέγεται να μην υλοποιηθεί άμεσα από την getScheduleTime (Σχήμα 27) άλλα μέσω κάποιου εξωτερικού εκτελεστή, την μέθοδο getAlterScheduleTime για παράδειγμα, όπως φαίνεται στο Σχήμα 28. Έτσι, η getScheduleTime ακολουθεί τις προδιαγραφές ενός μεταβιβαστή, και αφού ουσιαστικά δεν εκτελείται ποτέ από το σύστημα επιστρέφει απλώς ένα αντικείμενο τύπου AutoGeneratedValue όπως επιβάλλεται από τον ορισμό της.

```
public AutoGeneratedValue getAlterScheduleTime(InvokerArgument arg) {
   //ανάκτηση της παραμέτρου
   java.util.Properties params = arg.getInvokerParameters();
   double a = Double.parseDouble(arg.getProperty("a"));

    long res = a*Math.random()*1000; //μετατροπή σε ms

   return new AutoGeneratedValue(new Long(res));
}
```

Σχήμα 28 – *Τμήμα κώδικα, εξωτερικός εκτελεστής.*

Αν ο εξωτερικός εκτελεστής getAlterScheduleTime ορίζεται στην κλάση externalInvokers.Invokers τότε το τμήμα του εγγράφου διαμόρφωσης του συστήματος που προσδιορίζει τις συσχετίσεις μεταβιβαστών-εξωτερικών εκτελεστών θα έχει την μορφή του Σχήματος 29.

```
<delegators-invokers>
        <delg-invk>
                <delegator class_name="coreModel.Timer" mth_name="getScheduleTime">
                <invoker class_name="externalInvokers.Invokers" mth_name=" getAlterScheduleTime ">
                        <param name="a" value="10"/>
                </invoker>
        </delg-invk>
</delegators-invokers>
```

Σχήμα 29 – *Τμήμα από το XML έγγραφο διαμόρφωσης του συστήματος, περιοχή ορισμού μεταβιβαστών/ εξωτερικών εκτελεστών.*

Στο τμήμα κώδικα του Σχήματος 30 κατασκευάζεται ο TaskDescriptor που περιγράφει το σύνολο των ενεργειών που θα εκτελεστούν από τον Executor. Στην συγκεκριμένη περίπτωση τα βασικά ορίσματα για την δημιουργία του TaskDescriptor είναι αυτά που απαιτούνται για τον σχηματισμό των AINT και AINS.

Η AINT περιγράφει την μέθοδο που υλοποιεί τον ρυθμό αποστολής των μηνυμάτων ενημέρωσης και δημιουργείται με την βοήθεια των ορισμάτων coreModel.Timer.class, getScheduleTime και new Object[]{new Object()}, τα οποία αντιστοιχούν στην κλάση, στο όνομα και στα ορίσματα της μεθόδου αυτής.

Η AINS περιγράφει την μέθοδο sendStateMessage() η οποία είναι υπεύθυνη για την αποστολή των μηνυμάτων καταστάσεως. Η μέθοδος αυτή ορίζεται στην κλάση SComponent και δεν έχει ορίσματα. Σύμφωνα με τα παραπάνω και με την παραδοχή ότι το ακόλουθο τμήμα κώδικα υφίσταται στην κλάση υλοποίησης των οντοτήτων του συστήματος (άρα κληρονομεί από την SComponent), τα ορίσματα αρχικοποίησης της AINS είναι τα this, sendStateMessage και null. Όπου this η αναφορά του αντικειμένου πάνω στο οποίο εφαρμόζεται η sendStateMessage() (περισσότερες πληροφορίες στο API).

```
…
try {
   TaskDescriptor tdesc = new TaskDescriptor(
          this, “sendStateMessage”, null
          coreModel.Timer.class, “getScheduleTime”, new Object[]{new Object()},
          null, TaskDescriptor.REPEAT_ANY_CASE,
          null, null, null );
          //Δημιουργία Executor
          Exector executor = new Executor();
          //ενεργοποίηση TaskDescriptor
          executor.executeTask(tdesc);

}catch(Exception ex) {ex.printStackTrace();}
…
```

Σχήμα 30 – *Τμήμα κώδικα, δημιουργία TaskDescriptor.*

Παράδειγμα 2
Η πολιτική κατωφλίου υλοποιείται με συνδυασμό των μηχανισμών όρων-ενεργειών και ανίχνευσης διακεκριμένων γεγονότων.

Έστω ότι η αποστολή των μηνυμάτων καταστάσεως πραγματοποιείται όταν η απόλυτη τιμή της ποσοστιαίας μεταβολής του φορτίου, %loadChange, του διεκπεραιωτή υπερβαίνει ένα όριο, loadThreshold. Ο υπολογισμός του μεγέθους %loadChange πραγματοποιείται κάθε φορά που ο διεκπεραιωτής επιβαρύνεται ή απελευθερώνεται από κάποιο φορτίο, μέσω της σχέσης %loadChange = |(currentLoad – stateDrivenLoad)|/ stateDrivenLoad * 100 όπου currentLoad και stateDrivenLoad το τρέχον φορτίο και το φορτίο που είχε ο διεκπεραιωτής κατά την τελευταία αποστολή μηνυμάτων καταστάσεως αντίστοιχα. Το μέγεθος αυτό συγκρίνεται στην συνέχεια με την τιμή του loadThreshold προκειμένου ο διεκπεραιωτής να αποφανθεί αν θα αποστείλει η όχι μηνύματα ενημέρωσης στις υπόλοιπες οντότητες του συστήματος.

Με την βοήθεια του μηχανισμού όρων-ενεργειών, η αποστολή των μηνυμάτων καταστάσεως, η οποία υλοποιείται μέσω της μεθόδου sendStateMessage(), μπορεί να πραγματοποιείται σαν μία ανεξάρτητη διεργασία (thread) που εκτελείται επαναληπτικά με την επαλήθευση ενός όρου. Ο όρος αυτός αποτελείται από μία συνθήκη, την stateDispatch, η οποία επαληθεύεται στην περίπτωση που το μέγεθος %loadChange υπερβαίνει το όριο loadThreshold, και η κατάσταση λειτουργίας του είναι UNBLOCK_WITH_COMPLETION αφού η αποστολή των μηνυμάτων καταστάσεως θα πρέπει να συμβαίνει με την εκπλήρωση του. Όπως και στο πρώτο παράδειγμα, η μέθοδος sendStateMessage() περιγράφεται από την ενέργεια AINS.

Επιπλέον, ο προγραμματισμένος χρόνος εκτέλεσης θα πρέπει να είναι ο μέγιστος δυνατός έτσι ώστε η επαναληπτική εκτέλεση της AINS και γενικότερα του αντίστοιχου TaskDescriptor να μην τερματίζεται για όσο χρονικό διάστημα παρέλθει μέχρι την ικανοποίηση του όρου. Έτσι η AINT περιγράφει μία μέθοδο, την getScheduleTime, η οποία επιστρέφει την μέγιστη τιμή που μπορεί να λάβει ένα αντικείμενο τύπου java.lang.Long.

Σημειώνεται επίσης ότι, η κατάσταση λειτουργίας του TaskDescriptor είναι αδιάφορη αφού ο προγραμματισμένος χρόνος εκτέλεσης είναι τόσο μεγάλος που είναι πρακτικά αδύνατο να μην πραγματοποιηθεί η AINS εντός του διαστήματος αυτού.

Διατηρώνοντας τις παραδοχές που έγιναν στο προηγούμενο παράδειγμα (για τις κλάσεις υλοποίησης και εκτέλεσης των μεθόδων), στα τμήματα κώδικα που απεικονίζονται στα Σχήματα 31 και 32 πραγματοποιείται η υλοποίηση της μεθόδου που περιγράφει η AINT και η δημιουργία του TaskDecsriptor αντίστοιχα.

Υπενθυμίζεται ότι οι μέθοδοι getScheduleTime και sendStateMessage ορίζονται στην κλάση coreModel.Timer και SComponent αντίστοιχα.

```
public Long getScheduleTime(Object arg) {
        return new Long(Long.MAX_VALUE);
}
```

Σχήμα 31 – *Τμήμα κώδικα, υλοποίηση της AINT.*

```
…
//ο όρος
private AbstractLock  _al = null;
/*επιβαλλόμενο φορτίο στον διεκπεραιωτή κατά την τελευταία φορά που αποστάλθηκαν
μηνύματα ενημέρωσης*/
private long _stateDrivenLoad = 0;
…

try {
  //δημιουργία κατασκευαστή όρων
  ConditionSetFactory cf = new ConditionSetFactory();
   //δημιουργία συνθήκης
   LockCondition lc = new LockCondition("stateDispatch");
   //δημιουργία όρου
   _al = cf.getLock(ConditionSetFactory.All, null, null,
                              AbstractLock.UNBLOCK_WITH_COMPLETION,
                                        AbstractLock.UNBLOCKED_BEHAVIOR);
    //προσθήκη συνθήκης στον όρο
    _al.addCondition(lc);
   //δημιουργία TaskDescriptor
   TaskDescriptor tdesc = new TaskDescriptor(
           this, "sendStateMessage", null
           coreModel.Timer.class, "getScheduleTime", new Object[]{new Object()},
           _al, TaskDescriptor.REPEAT_ANY_CASE,
           null, null, null );
   //Δημιουργία Executor
   Exector executor = new Executor();
   //ενεργοποίηση TaskDescriptor
   executor.executeTask(tdesc);

}catch(Exception ex) {ex.printStackTrace();}
…

//επιστρέφει το _stateDrivenLoad
public long getStateDrivenLoad() {return _stateDrivenLoad;}

//αποθηκεύει την νέα τιμή του _stateDrivenLoad
public void setStateDrivenLoad(long stateDrivenLoad) {_stateDrivenLoad = stateDrivenLoad;}

//επιστρέφει τον όρο που ενεργοποιεί την αποστολή των μηνυμάτων ενημέρωσης
public void getMessageDispatchCondition() {return _al;}
```

Σχήμα 32 – *Τμήμα κώδικα, δημιουργία TaskDescriptor, Executor.*

Ο έλεγχος μεταβολής του φορτίου του διεκπεραιωτή και ικανοποίησης της συνθήκης stateDispatch, που υπαγορεύει την αποστολή μηνυμάτων καταστάσεως, πραγματοποιείται με την βοήθεια του μηχανισμού ανίχνευσης διακεκριμένων γεγονότων.

Όπως προαναφέρθηκε, η μέθοδος naturalPropertyChanged εκτελείται αυτόματα από το σύστημα κάθε φορά που υφίσταται αλλαγή στην κατάσταση των αυτοφυών ιδιοτήτων της οντότητας. Έτσι, στο συγκεκριμένο παράδειγμα η κλήση της πραγματοποιείται με την μεταβολή του φορτίου, αφού το φορτίο αποτελεί αυτοφυή ιδιότητα του διεκπεραιωτή.

Για την εκμετάλλευση των δυνατοτήτων που παρέχονται από τον μηχανισμό εναλλακτικών συμπεριφορών, η υλοποίηση του ελέγχου επαλήθευσης της συνθήκης stateDispatch επιλέγεται να πραγματοποιηθεί σε εξωτερικό εκτελεστή. Η κλήση του εξωτερικού εκτελεστή επιτυγχάνεται με την δημιουργία του αντίστοιχου μεταβιβαστή ο οποίος επιπλέον τον

τροφοδοτεί με τις απαραίτητες πληροφορίες για την αποτίμηση της συνθήκης. Στα Σχήματα 33 και 34 απεικονίζονται οι υλοποιήσεις των προαναφερθέντων μεθόδων.

```
...
public synchronized void naturalPropertyChanged(NaturalPropertyChangedEvent event) {
   stateDispatchEvaluation(event); //κλήση του μεταβιβαστή
}

//μεταβιβαστής
public AutoGeneratedValue stateDispatchEvaluation(Object arg) {
          return new AutoGeneratedValue(new Long(0));
}
…
```

Σχήμα 33 – *Τμήμα κώδικα, υλοποίηση της naturalPropertyChanged και του μεταβιβαστή που καλείται μέσα από αυτή.*

Ο εξωτερικός εκτελεστής actualSTDispEvaluation αντλεί τις απαραίτητες πληροφορίες από το όρισμα του μεταβιβαστή stateDispatchEvaluation, καθώς και από το αντικείμενο του διεκπεραιωτή όπου λαμβάνει χώρα η συγκεκριμένη μεταβολή του φορτίου. Οι πληροφορίες αυτές αναφέρονται στην μεταβληθείσα ιδιότητα και στον όρο με το οποίο ενεργοποιείται η αποστολή των μηνυμάτων ενημέρωσης στους υπόλοιπους διεκπεραιωτές του συστήματος. Ο όρος αυτός ουσιαστικά αποτελεί τον συνδετικό κρίκο μεταξύ του μηχανισμού όρων-ενεργειών και ανίχνευσης διακεκριμένων γεγονότων και φιλοξενείται σε πεδίο της κλάσης υλοποίησης των διεκπεραιωτών η οποία στην συγκεκριμένη περίπτωση είναι η VPC.

Ο εξωτερικός εκτελεστής υπολογίζει την απόλυτη τιμή, change, της ποσοστιαίας μεταβολής του φορτίου του διεκπεραιωτή και την συγκρίνει με το όριο threshold. Στην περίπτωση που change>threshold η κατάσταση της συνθήκης του όρου μεταβάλλεται σε αληθής σηματοδοτώντας ταυτόχρονα την εκπλήρωση του. Η ικανοποίηση του όρου εγείρει την εκτέλεση της ενέργειας AINS, η οποία τελικώς μεταφράζεται στην αποστολή των μηνυμάτων ενημέρωσης. Ο Executor εν συνεχεία αρχικοποιεί τις συνθήκες του όρου, θέτοντας την κατάσταση της μοναδικής συνθήκης από την οποία αποτελείται, ως ψευδής και επανεργοποιεί τον αντίστοιχο TaskDescriptor για την εκτέλεση της AINS όταν ο όρος εκπληρωθεί εκ νέου.

Σημειώνεται ότι το όριο threshold καθορίζεται από την τιμή αντίστοιχης παραμέτρου του εξωτερικού εκτελεστή η οποία ορίζεται στο έγγραφο διαμόρφωσης του συστήματος, προσδίδοντας ιδιαίτερη ευελιξία στην διεξαγωγή των πειραμάτων.

```
public AutoGeneratedValue actualSTDispEvaluation(InvokerArgument arg) {
   //ανάκτηση της παραμέτρου που ορίζει το κατώφλι της μεταβολής του φορτίου
   java.util.Properties params = arg.getInvokerParameters();
   float threshold = Float.parseDouble(arg.getProperty("threshold"));

   /*Ανάκτηση του αντικειμένου που υλοποιεί τον αντίστοιχο μεταβιβαστή, ουσιαστικά το
   αντικείμενο του διεκπεραιωτή που υφίσταται η μεταβολή του φορτίου*/
   VPC vpc = (VPC)arg.getDelegator();
   //ανάκτηση πληροφοριών σχετικών με την μεταβολή των ιδιοτήτων μίας οντότητας
   NaturalPropertyChangeEvent  event = (NaturalPropertyChangeEvent)arg.getDelegatorArgument();
   //ανάκτηση πληροφοριών σχετικών με την μεταβληθείσα ιδιότητα
   PropertyState loadState = event.getChangedProperty();

   //το φορτίο του διεκπεραιωτή
   long currentLoad = loadState.getBoundAmount();
   //το φορτίο του διεκπεραιωτή κατά την τελευταία φορά που αποστάλθηκαν μηνύματα ενημέρωσης
   long stateDrivenLoad = vpc. getStateDrivenLoad()

  //σχέση που υποδεικνύει την επαλήθευση η όχι του όρου
  float change = Math.abs(currentLoad - stateDrivenLoad)*100/ stateDrivenLoad;
   if(change > threshold) {//εάν ο όρος επαληθεύεται
      AbstarctLock al = vpc.getMessageDispatchCondition();//ανάκτηση του όρου
       al.setConditionStatus("stateDispatch", true);//επαλήθευση της συνθήκης και κατ'επέκταση του όρου
      /*αποθήκευση της νέας τιμής του φορτίου του διεκπεραιωτή κατά την τελευταία
     αποστολή μηνυμάτων καταστάσεως*/
      vpc.setMessageDispatchCondition(currentLoad);
   }

   return new AutoGeneratedValue(new Long(0));
}
```

Σχήμα 34 – *Τμήμα κώδικα, υλοποίηση εξωτερικού εκτελεστή που καλείται από τον μεταβιβαστή του Σχήματος 33.*

4.6 Μηχανισμός διαχείρισης πληροφοριών (Information Management)

Μία πλατφόρμα είναι απαραίτητο να διαθέτει μηχανισμό διαχείρισης πληροφοριών κατά την διάρκεια μίας προσομοίωσης. Οι πληροφορίες αυτές σχετίζονται συνήθως με μεγέθη που απεικονίζουν την κατάσταση του συστήματος και χρησιμοποιούνται για την αποτίμηση και την αξιολόγηση του εφαρμοζόμενου στο σύστημα μοντέλου.

Στο προτεινόμενο εργαλείο αναπτύχθηκε μηχανισμός που δεν περιορίζεται απλώς στην καταγραφή κάποιων χαρακτηριστικών μεγεθών, όπως συμβαίνει συνήθως, αλλά παρέχει την δυνατότητα αποθήκευσης οποιασδήποτε πληροφορίας, η οποία σε επίπεδο υλοποίησης περιέχεται σε ένα αντικείμενο (object persistence), χωρίς σχεδόν κανένα περιορισμό.

4.6.1 Προδιαγραφές

Η ευελιξία, η επεκτασιμότητα και η δυναμικότητα του εν λόγω μηχανισμού εξασφαλίζεται από τις προδιαγραφές που παρουσιάζονται στις παραγράφους 4.6.1.1, 4.6.1.2 και 4.6.1.3, και είναι αυτές που οδήγησαν στην συγκεκριμένη υλοποίηση του εν λόγω μηχανισμού.

4.6.1.1 Πρότυπο περιγραφής πληροφορίας

Η περιγραφή της πληροφορίας θα πρέπει να πραγματοποιείται από πρότυπο όπου η απεικόνιση και ο εντοπισμός της πληροφορίας μπορεί να γίνει με ξεκάθαρο και συστηματικό τρόπο.

Η απαίτηση αυτή μας οδήγησε στην χρήση της περιγραφικής γλώσσας *XML* (*Extensible Markup Language*). Τα βασικά χαρακτηριστικά της XML, δηλαδή το ιεραρχικό μοντέλο δόμησης της, η επεκτασιμότητα, η αναγνωσιμότητα και η ικανότητα αυτό-περιγραφής της (self-describing/documenting/explanatory), την κατέστησαν ως το πιο διαδεδομένο πρότυπο περιγραφής και μεταφοράς πληροφοριών. Παράλληλα, αναπτύχθηκαν γλώσσες όπως η *XPath* (*XML Path Language*) και η *XPointer* (*XML Pointer Language*), που επιτρέπουν την πλοήγηση σε ένα XML έγγραφο και τον εντοπισμό των επιμέρους στοιχείων από τα οποία αποτελείται, ενώ η γλώσσα *XSLT* (*Extensible Stylesheet Language Transformations*) καθιστά δυνατό τον μετασχηματισμό ενός XML εγγράφου σε οποιοδήποτε άλλο έγγραφο κειμένου. Οι γλώσσες αυτές ενίσχυσαν την θέση της XML και αποτελούν πλέον αναπόσπαστο κομμάτι των τεχνολογιών που την συνοδεύουν στην ανάπτυξη εφαρμογών. *[w3s, xml]*, *[w3s, xslt]*, *[w3s, xpath]*, *[w3s, xpointer]*

Η χρήση της XML σαν περιγραφικό μέσο πληροφοριών αποτέλεσε κριτήριο για την απόρριψη, σε επίπεδο υλοποίησης, της μίας εκ των δύο βασικών μεθόδων που υποστηρίζονται εγγενώς από τον πυρήνα της γλώσσας προγραμματισμού Java, για την εγγραφή και την ανάκτηση αντικειμένων. Η μέθοδος αυτή, η *Serialization*, πέρα από ότι το πρότυπο αποθήκευσης των πληροφοριών που ακολουθεί δεν επιτρέπει καν την ανάγνωση (δυαδική μορφή) ή την τροποποίηση των πληροφοριών από χρήστες χωρίς ειδικές γνώσεις, επιπλέον εφαρμόζεται σε αντικείμενα που πληρούν συγκεκριμένες προδιαγραφές. Για παράδειγμα, η κλάση του αντικείμενου πρέπει να διαθέτει κατασκευαστή χωρίς ορίσματα (default constructor). *[Richardson, 05]*

4.6.1.2 Μοντελοποίηση πληροφορίας

Η μοντελοποίηση των πληροφοριών θα πρέπει να πραγματοποιείται με τρόπο γενικό, απλό, διάφανο (transparent) για τον χρήστη και ανεξάρτητο από την υλοποίηση του εκάστοτε συστήματος πάνω στο οποίο εφαρμόζεται.

Η απαίτηση αυτή μας οδήγησε στην απόρριψη μιας ακόμα μεθόδου, παρόλο που το χρησιμοποιούμενο πρότυπο περιγραφής της καταγραφείσας πληροφορίας είναι η XML. Η μέθοδος αυτή, η *XMLEncoder/Decoder*, η οποία υποστηρίζεται εγγενώς από τον πυρήνα της γλώσσας προγραμματισμού Java (μαζί με την Serialization), εκτός του ότι εφαρμόζεται σε αντικείμενα με συγκεκριμένες σχεδιαστικές προδιαγραφές (JavaBeans), η μοντελοποίηση της πληροφορίας που ακολουθεί δεν είναι κατανοητή από τον απλό χρήστη (με άμεσο αποτέλεσμα την αδυναμία ανάκτησης επιμέρους πληροφοριών).

Η ικανοποίηση της παραπάνω απαίτησης εξασφαλίζεται με χρήση της κατάλληλης υλοποιήσεως μίας μεθόδου, καλούμενης ως *XML Data Binding*. Η μέθοδος αυτή σχετίζεται με την αποθήκευση (διαδικασία καλούμενη ως *marshalling*) και τον επανασχηματισμό (*unmarshalling*) αντικειμένων σε και από έγγραφα XML (για λόγους διάκρισης τα καλούμε ως έγγραφα αποθήκευσης ή καταστάσεως XML (*XML state files*)) αντίστοιχα.

Στις περισσότερες υλοποιήσεις ο χρήστης καλείται να συμμορφωθεί με κάποιες προδιαγραφές οι οποίες συνήθως αναφέρονται αφενός στο πρότυπο κατασκευής των κλάσεων των αντικειμένων που επιθυμείται να αποθηκευτούν, αφετέρου στον καθορισμό της δομής των εγγράφων αποθήκευσης XML και στην συσχέτιση τους με τους τύπους των αντικείμενων που περιγράφουν, μέσω των *σχημάτων σύνδεσης* (*binding schemas*) (όπως συμβαίνει για παράδειγμα στις υλοποιήσεις JAXB, XMLBeans και Castor XML). Ο πηγαίος κώδικας των κλάσεων που συμφωνούν με τα σχήματα σύνδεσης καθώς επίσης ο πηγαίος κώδικας των συνοδευτικών βοηθητικών κλάσεων και διεπιφανειών που συνήθως απαιτείται, παράγονται μέσω εργαλείων που παρέχει η εκάστοτε υλοποίηση.

Η συμμόρφωση των κλάσεων σε αυστηρούς σχεδιαστικούς κανόνες οδηγεί σχεδόν πάντα στην ύπαρξη δύο μοντέλων αντικειμένων, στο πρωτεύον το οποίο υλοποιήθηκε σύμφωνα με τις προδιαγραφές της βασικής εφαρμογής, για παράδειγμα το μοντέλο προσομοίωσης, και στο δευτερεύον μοντέλο που σχηματίστηκε βάσει των κανόνων που απαιτούνται από την υλοποίηση της μεθόδου καταγραφής πληροφοριών. Η ύπαρξη δύο μοντέλων είναι προφανές ότι μειώνει την ευελιξία οποιαδήποτε εφαρμογής αφού απαιτείται αφενός η ανάπτυξη επιπρόσθετου κώδικα για την μεταξύ τους σύνδεση και επικοινωνία, αφετέρου η παράλληλη συντήρηση τους. Η ακαμψία μίας τέτοιας εφαρμογής φαίνεται αν αναλογιστούμε την διαδικασία που πρέπει να ακολουθηθεί στην περίπτωση που επιθυμείται η αποθήκευση ενός νέου τύπου αντικειμένων. Η διαδικασία αυτή εμπλέκει μετατροπές στα σχήματα σύνδεσης, στο δευτερεύον και στο πρωτεύον μοντέλο κλάσεων, και απαιτεί φυσικά την μεταγλώττιση του κώδικα.

Μερικές υλοποιήσεις, όπως η JAXB 2.0 και η Castor XML, παρέχουν την δυνατότητα άμεσης σύνδεσης του πρωτεύοντος μοντέλου κλάσεων με τα έγγραφα αποθήκευσης XML χωρίς την δημιουργία δευτερεύοντος μοντέλου. Ωστόσο οι υλοποιήσεις αυτές εξακολουθούν να έχουν περιορισμούς οι οποίοι δεν είναι επιθυμητοί στην συγκεκριμένη περίπτωση, δηλαδή την διατήρηση σχήματος σύνδεσης και την ύπαρξη βασικών κανόνων κατασκευής των τύπων των αντικειμένων που επιθυμείται να αποθηκευτούν (για παράδειγμα, στην περίπτωση του Castor XML η αποθήκευση των τιμών ιδιωτικών πεδίων γίνεται με χρήση του πρότυπου ονοματολογίας των JavaBeans). *[Richardson, 05]*, *[McLaughlin, 02]*, *[Java, jaxb]*, *[GlassFish, jaxb]*, *[Bourret, bind]*, *[Castor, xmlMap]*, *[XMLBeans, intro]*

Από το σύνολο των υλοποιήσεων της XML Data Binding επιλέχτηκε η XStream *[XStream, hp]*, *[XStream, Tillman]*, η οποία δεν παρουσιάζει τους παραπάνω περιορισμούς και κατά συνέπεια καλύπτει τις προδιαγραφές του συστήματος μας. Συγκεκριμένα στην XStream :

- Δεν απαιτούνται αλλαγές στον μοντέλο των αντικειμένων. Η XStream υποστηρίζει τη καταγραφή οποιοδήποτε πεδίου (δημόσιων (public), ιδιωτικών (private) και αμετάβλητων (final)) ανεξάρτητα από το πρότυπα σχηματισμού των κλάσεων. Χαρακτηριστικό παράδειγμα αποτελεί ότι οι κλάσεις δεν υποχρεούνται να έχουν κατασκευαστή χωρίς ορίσματα (default constructor) και δεν υπακούουν σε συγκεκριμένα πρότυπα ονοματολογίας. Με τον τρόπο αυτό αποκλείεται σε κάθε περίπτωση η ανάγκη διατήρησης δύο μοντέλων αντικειμένων η οποία κατά κανόνα επιφέρει δυσκολίες και προβλήματα στην ανάπτυξη και στην συντήρηση της εφαρμογής.

- Η καταγραφή και η ανάκτηση των αντικειμένων πραγματοποιείται υπό την απουσία σχημάτων σύνδεσης και οποιοδήποτε παραγόμενων βάσει του σχήματος συνοδευτικών κλάσεων ή διεπιφανειών που θα επέβαλαν την μεταγλώττιση του κώδικα. Το χαρακτηριστικό αυτό σε συνδυασμό με το γεγονός ότι δεν απαιτούνται αλλαγές στον μοντέλο των αντικειμένων εξασφαλίζει την απόλυτη διαφάνεια του συστήματος προς τον χρήστη.

- Η μοντελοποίηση της κατάστασης των αντικειμένων στο έγγραφο αποθήκευσης XML γίνεται με τρόπο κατανοητό χωρίς την άσκοπη επανάληψη πληροφοριών, γεγονός που συνηγορεί στην συστηματική και αποδοτική αναζήτηση στοιχείων μέσα σε αυτό.

- Πολύ εύκολο στην χρήση του. Όπως φαίνεται στο ακόλουθο παράδειγμα, Σχήματα 35 και 36, η καταγραφή και η ανάκτηση ενός αντικειμένου της κλάσεως Person πραγματοποιείται μέσα σε τρεις γραμμές κώδικα, ενώ το αντίστοιχο έγγραφο αποθήκευσης XML χαρακτηρίζεται για την απλότητα και την σαφήνεια του.

```
//κύρια κλάση
package mPackage
…
class Person {
   private String name;
   private mainToy = new Computer(“Comodore”);
   private List toys = new ArrayList();
   ...

   public static void main(String[] args) {
      Person person = new Person("Joe");
      person.addToy(new Computer("Apple"));
      person.addToy(new Computer("Spectrum"));
      person.addToy(new Car("blue"));

     //δημιουργία βασικού εργαλείου καταγραφής και ανάκτησης ενός αντικειμένου
     XStream xstream = new XStream(new DomDriver());
     String xml = xstream.toXML(person);//καταγραφή αντικειμένου
     Person p = (Family)xstream.fromXML(xml);//ανάκτηση αντικειμένου
     ....
   }
}

//βοηθητικές κλάσεις
package sPackage
…
public class Computer { String type; …}

package sPackage
…
public class Car { String color; …}
```

Σχήμα 35 – *Τμήμα κώδικα, υπόδειγμα κλάσης προς καταγραφή.*

```
<mPackage.Person>
        <name>Joe</name>
        <mainToy> <type>Comodore</type></mainToy>
        <toys>
                <sPackage.Computer> <type>Apple</type></sPackage.Computer>
                <sPackage.Computer> <type>Spectrum</type> </sPackage.Computer>
                <sPackage.Car> <color>blue</color> </sPackage.Car>
        </toys>
</mPackage.Person>
```

Σχήμα 36 – *Καταγραφή αντικειμένου της κλάσεως του Σχήματος 35, υπό την μορφή εγγράφου XML, με χρήση της XStream.*

4.6.1.3 Μέσο αποθήκευσης

Η καταγραφείσα πληροφορία θα πρέπει να διατηρείται σε ένα ολοκληρωμένο σύστημα διαχείρισης δεδομένων, μία βάση δεδομένων, ώστε αφενός να διασφαλίζεται η ακεραιότητα της, αφετέρου να παρέχεται η δυνατότητα ανάκτησης της μέσω ερωτημάτων.
Οι βάσεις που δύνανται να διαχειριστούν την αποθήκευση και την ανάκτηση εγγράφων XML μπορεί να διακριθούν σε δύο κατηγορίες, στις *βάσεις δεδομένων με χαρακτηριστικά XML* και στις *εγγενείς βάσεις δεδομένων XML* (*XML-enable databases* και *native XML databases*). Η θεμελιώδης διαφορά μεταξύ των δύο κατηγοριών έγκειται στον τρόπο αποθήκευσης των πληροφοριών. Συγκεκριμένα, στην πρώτη κατηγορία η αποθήκευση βασίζεται στην αντιστοίχηση μεταξύ του σχήματος της βάσης και του σχήματος των εγγράφων XML. Η αντιστοίχηση αυτή καθορίζεται στο στάδιο σχεδιασμού της εφαρμογής και συνήθως υλοποιείται με την *μέθοδο αντιστοίχησης βάσει πινάκων* ή την *μέθοδο αντιστοίχησης βάσει αντικειμένων* (*table-based mapping* και *object-relational mapping* αντίστοιχα). Στην κατηγορία αυτή ουσιαστικά ανήκουν σχεσιακές ή αντικειμενοστραφείς βάσεις δεδομένων οι οποίες διαθέτουν μηχανισμό μεταφοράς και προσαρμογής των εγγράφων XML στο μοντέλο δεδομένων τους. Αντιθέτως, οι εγγενείς βάσεις δεδομένων XML χρησιμοποιούν ένα καθορισμένο σύνολο δομών στις οποίες μπορούν να αποθηκευτούν οποιαδήποτε έγγραφα XML ανεξαρτήτου σχήματος. *[Bourret, gnative]*, *[Bourret, xmldbs]*

Στην συγκεκριμένη περίπτωση τα έγγραφα XML που επιθυμείται να αποθηκευτούν δεν υπακούουν σε κάποιο σχήμα, αφού μπορεί να περιγράφουν την κατάσταση οποιοδήποτε αντικειμένου του συστήματος. Άλλωστε ο καθορισμός σχημάτων εξαρτώμενων από την αποθηκευμένη πληροφορία θα μείωνε την ευελιξία του συστήματος. Το παραπάνω γεγονός οδήγησε στην επιλογή μίας εγγενούς βάσης δεδομένων XML ως μέσο αποθήκευσης και διαχείρισης των καταγραφέντων πληροφοριών. Αξίζει να σημειωθεί ότι ακόμα και αν υπήρχε η δυνατότητα ορισμού ενός γενικού σχήματος XML και η σύνδεση του με βάση δεδομένων με χαρακτηριστικά XML, η αναπόφευκτη πολυπλοκότητα του θα οδηγούσε στην κατασκευή είτε πολλών στηλών με ακαθόριστες τιμές (απώλεια χώρου), είτε πολλών πινάκων το οποίο δεν είναι αποδοτικό. Επιπλέον οι εγγενείς βάσεις XML επιτυγχάνουν ταχύτερη αποθήκευση και ανάκτηση των πληροφοριών αφού το στάδιο μετατροπής των εγγράφων σε κάποιο σχεσιακό μοντέλο και αντίστροφα δεν υφίσταται (όπως συμβαίνει στις XML-enable βάσεις). Παράλληλα οι βάσεις αυτές υποστηρίζουν άμεσα τεχνολογίες σχετικές με αυτή της XML, όπως είναι ερωτήματα XML μέσω της γλώσσας *XQuery* (*XML Query Language*) *[w3s, xquery]*.

Από το πλήθος των εγγενών βάσεων XML επιλέχτηκε η Berkeley DB XML (BDB XML) *[Berkeley, hp]*. Τα κυριότερα χαρακτηριστικά της BDB XML τα οποία την κατέστησαν την καλύτερη υποψήφιο βάση για την υλοποίηση του προτεινόμενου μηχανισμού είναι τα ακόλουθα:

- Δυνατότητα επιλογής της μεθόδου αποθήκευσης των εγγράφων XML. Η BDB XML διαθέτει δύο μεθόδους αποθήκευσης. Στην πρώτη μέθοδο, την *μέθοδο αποθήκευσης εγγράφου* (*document storage method*), τα έγγραφα αποθηκεύονται όπως υφίστανται σε ένα σύστημα αρχείων, δηλαδή αδιάσπαστα. Αντιθέτως στην δεύτερη, την *μέθοδο αποθήκευσης κόμβου* (*node storage method*), τα έγγραφα αποθηκεύονται σε τμήματα (nodes). Η επιλογή της καταλληλότερης μεθόδου εξαρτάται από τις προδιαγραφές και τις ανάγκες της εκάστοτε εφαρμογής. Για παράδειγμα στις περιπτώσεις όπου α) η ταχύτητα ανάκτησης της πληροφορίας είναι σημαντικότερη από την ταχύτητα εκτέλεσης ερωτημάτων, β) η ανάκτηση ολόκληρου και όχι μέρους του εγγράφου XML είναι η συνηθέστερη διαδικασία που εφαρμόζεται και γ) το μέγεθος των εγγράφων XML είναι τόσο μικρό ώστε η αύξηση της αποδοτικότητας που προσφέρεται από την χρήση της μεθόδου αποθήκευσης κόμβου είναι αμελητέα, και τότε θα πρέπει να χρησιμοποιηθεί η μέθοδος αποθήκευσης εγγράφου.

- Η ανάκτηση των εγγράφων πραγματοποιείται μέσω ερωτημάτων που διαρθρώνονται με χρήση της γλώσσας αναζήτησης XQuery. Η XQuery σχεδιάστηκε για την δημιουργία

ερωτημάτων σε έγγραφα XML και εφαρμόζεται για την ανάκτηση τμημάτων ή ολόκληρων εγγράφων, καθώς και συγκεκριμένων τιμών από ένα ή περισσότερα έγγραφα. Επίσης, μπορεί να χρησιμοποιηθεί για την μετατροπή και την διαχείριση των τιμών που επιστρέφονται από το ερώτημα. Σημειώνεται ότι η XQuery αποτελεί επέκταση των γλώσσας XPath 2.0 η οποία με την σειρά της βασίζεται στις προδιαγραφές της γλώσσας XPath 1.0.

➢ Η BDB XML παρέχει έναν ευέλικτο μηχανισμό δημιουργίας ευρετηρίου (indexing) ο οποίος βελτιώνει σημαντικά την απόδοση της εκτέλεσης των ερωτημάτων. *[Berkeley, gs]*

4.6.1.3.1 Εναλλακτικοί τρόποι αποθήκευσης

Πριν από την περιγραφή του προτεινόμενου μηχανισμού διαχείρισης πληροφοριών αξίζει να αναφερθούν τρεις επιπλέον κατηγορίες τεχνολογιών. Οι τεχνολογίες αυτές θα μπορούσαν να χρησιμοποιηθούν στην συγκεκριμένη υλοποίηση στην περίπτωση που αφενός το πρότυπο της αποθηκευμένης πληροφορίας δεν ήταν η XML, αφετέρου η επεκτασιμότητα και η διαφάνεια του συστήματος δεν αποτελούσαν θεμελιώδεις παράγοντες σχεδιασμού του.

➢ ***Αντικειμενοστραφείς βάσεις δεδομένων (Object oriented Database Management System, ODBMS).***

Στις αντικειμενοστραφείς βάσεις η αποθήκευση αντικειμένων πραγματοποιείται άμεσα χωρίς την αντιστοίχησης τους σε διαφορετικό μοντέλο δεδομένων.

Η ύπαρξη ενιαίου μοντέλου αποτελεί την κύρια διαφορά και το βασικότερο πλεονέκτημα των ODBMS έναντι των σχεσιακών βάσεων δεδομένων (*Relational Database Management System*, *RDBMS*). Ο χρήστης απαλλάσσεται από την δημιουργία, την συντήρηση και την εξασφάλιση επικοινωνίας μεταξύ των δύο διαφορετικών μοντέλων, σε επίπεδο βάσης και εφαρμογής αντίστοιχα, όπως συμβαίνει στις RDBMS. *[serviceArch, odbmsPros]*

Η προσαρμογή δεδομένων σε διαφορετικές δομές καλείται *αντίσταση αντιστοίχησης* (*impedance mismatch*) και αναπόφευκτα επιφέρει μείωση της απόδοση ενός συστήματος διαχείρισης πληροφοριών. Στις RDBMS η αντίσταση αντιστοίχησης είναι υπαρκτή αφού η προσαρμογή του μοντέλου της βάσεως με αυτό των αντικειμένων και αντίστροφα είναι απαραίτητη (αντιστοίχηση αντικειμένων-πινάκων, πεδίων αντικειμένων-στηλών), με αποτέλεσμα να παρουσιάζουν μειωμένη απόδοση έναντι των ODBMS ιδιαίτερα σε πολύπλοκες δομές αντικειμένων. *[serviceArch, imped]*

Η ODBMS θα μπορούσε να χρησιμοποιηθεί για την υλοποίηση του προτεινόμενου μηχανισμού διαχείρισης πληροφοριών στην περίπτωση που οι υπάρχουσες ODBMS υλοποιήσεις δεν παρουσίαζαν τα ακόλουθα μειονεκτήματα.
i) Τα περισσότερα συστήματα περιλαμβάνουν στάδια *προ-επεξεργασίας* (*pre-processing*) ή *μετά-επεξεργασίας* (*post-processing*). Τα στάδια αυτά συσχετίζονται με τροποποιήσεις και προσαρμογές που επιβάλλονται για την διασύνδεση και την επικοινωνία εφαρμογής και βάσης δεδομένων σύμφωνα με ένα σύνολο *μετά-πληροφοριών* (*metadata*) που ορίζονται άμεσα ή έμμεσα (με χρήση εργαλείων) από τον χρήστη. Η κύρια διαφορά μεταξύ των δύο σταδίων έγκειται στο τρόπο πραγματοποίησης των αλλαγών στο επίπεδο της εφαρμογής. Η προ-επεξεργασία επιδρά στον πηγαίο κώδικα σε αντίθεση με την μετά-επεξεργασία όπου οι απαραίτητες τροποποιήσεις συμβαίνουν στα μεταγλωττισμένα αρχεία. *[ObjectStore, Landis], [Franklin , sbs]*

Η ObjectStore PSE (Persistent Storage Engine) *[ObjectStore, Landis]* αποτελεί χαρακτηριστικό παράδειγμα υλοποίησης ODBMS με στάδιο μέτα-επεξεργασίας. Στο στάδιο αυτό λαμβάνει χώρα η δημιουργία του σχήματος της βάσης (schema generation), σύμφωνα

με την δομή των κλάσεων των αντικειμένων που επιθυμείται να αποθηκευτούν, καθώς και η μεταβολή των μεταγλωττισμένων αρχείων του πηγαίου κώδικα της εφαρμογής (code annotation όπως αναφέρεται στα βοηθητικά έγγραφα της εν λόγω βάσης).

Σε αντίθεση με την ObjectStore, η EyeDB πραγματοποιεί τις απαραίτητες αλλαγές με προ-επεξεργασία όπου παράγονται τα απαραίτητα πρότυπα κλάσεων (Java stubs) τα οποία θα πρέπει να χρησιμοποιηθούν για την επικοινωνία με την βάση. *[EyeDB, hp]*, *[EyeDB, bind]*

ii) Σχεδόν όλες οι υλοποιήσεις ODBMS θέτουν σχεδιαστικούς περιορισμούς στις κλάσεις των καταγραφέντων αντικειμένων και επιπλέον απαιτούν τον ορισμό μετά-πληροφοριών που χρησιμοποιούνται για την διαμόρφωση του συστήματος.

Το περιεχόμενο και το πρότυπο των μετά-πληροφοριών εξαρτάται από την εκάστοτε υλοποίηση. Για παράδειγμα στην objectDB η οποία συμφωνεί με τις προδιαγραφές *JDO* (*Java Data Object*) απαιτείται ο καθορισμός, μέσω ενός εγγράφου XML, στοιχείων σχετικών με τις κλάσεις που θα αποθηκευτούν. Αντιθέτως στην EyeDB απαραίτητη είναι η περιγραφή της δομής των κλάσεων με την βοήθεια της γλώσσας EyeDB ODL η οποία βασίζεται στην *ODL* (*Object Definition Language*) που καθορίζεται από τον οργανισμό Object Data Management Group (ODMG). *[EyeDB, odl], [ObjectDB, man]*

Στο σημείο αυτό αξίζει να αναφέρουμε λίγα λόγια για τον ODMG. Ο ODMG είναι ένας οργανισμός που έχει σαν σκοπό την δημιουργία προτύπων για την αποθήκευση αντικειμένων αντικειμενοστραφών γλωσσών προγραμματισμού σε βάσεις δεδομένων. Τα πρότυπα αυτά αναφέρονται στο μοντέλο ενός αντικειμένου και την διασύνδεση του με γλώσσες προγραμματισμού (programming language bindings) καθώς επίσης στις γλώσσες περιγραφής και αναζήτησης αντικειμένων. Χαρακτηριστικά, η γλώσσα ODL χρησιμοποιείται για την περιγραφή κλάσεων, ενώ η *Object Interchange Format* (OIF) καθορίζει τις προδιαγραφές για την καταγραφή και ανάκτηση αντικειμένων της βάσης μέσω αρχείων και επιπλέον μπορεί να χρησιμοποιηθεί για την ανταλλαγή δεδομένων μεταξύ βάσεων. *[Bierman, 00]*, *[ODMG, hp], [jyd, hp]*

iii) Η γλώσσα αναζήτησης αντικειμένων εξαρτάται από την εκάστοτε υλοποίηση του ODBMS. Η προσπάθεια καθιέρωσης της γλώσσας *Object Query Language* (*OQL*) από την ODMG ως γλώσσας αναζήτησης αντικειμένων σε συστήματα ODBMS δεν έχει αποδώσει μέχρι στιγμής τα αναμενόμενα αποτελέσματα. *[serviceArch, odbmsProb]*

Τα περισσότερα συστήματα χρησιμοποιούν αποκλειστικά δική τους γλώσσα αναζήτησης (όπως το db4o που παρέχει τρεις διαφορετικούς μηχανισμούς αναζήτησης, QBE, Native Queries και SODA, [*db4o, man*]) ή κάποια άλλη που ναι μεν βασίζεται στην OQL αλλά δεν συμμορφώνεται/ συμβαδίζει (not compliant) πλήρως με αυτή (όπως στο EyeDB, *[EyeDB, oql]*). Αυτό έχει σαν αποτέλεσμα την μείωση της ευελιξίας της εφαρμογής αφού η αλλαγή του υποκείμενου συστήματος ODBMS επιφέρει αλλαγές και στις εκφράσεις αναζήτησης (query expressions) των αντικειμένων.

➢ ***Αντιστοίχηση αντικειμενοστραφούς σε σχεσιακό πρότυπο (Object to Relational Mapping, ORM).***

Η ORM υλοποιεί μία προσπάθεια ανάπτυξης εφαρμογών για την διασύνδεση σχεσιακών βάσεων δεδομένων με τις αρχές αντικειμενοστραφών γλωσσών προγραμματισμού. Η προσπάθεια αυτή έχει σαν τελικό σκοπό τον σχηματισμό «εικονικών» αντικειμενοστραφών βάσεων ώστε η αποθήκευση των αντικειμένων να πραγματοποιείται χωρίς η διαδικασία μεταφοράς και μετατροπής των πληρορφοριών από το μοντέλο της βάσης στο μοντέλο των αντικειμένων και αντίστροφα, να είναι ορατή στον χρήστη. Παράδειγμα υλοποίησης της τεχνολογίας ORM αποτελεί η Hibernate. *[wiki, orm]*, *[Richardson, 05]*

Η Hibernate δρα πάνω σε μία σχεσιακή βάση επιτρέποντας την αποθήκευση και την ανάκτηση αντικειμένων σε γλώσσα προγραμματισμού Java σε και από την βάση αυτή αντίστοιχα. Ένα από τα βασικά της αρχιτεκτονικά χαρακτηριστικά είναι η χρήση της ανάκλασης (reflection) των κλάσεων (το οποίο υποστηρίζεται εγγενώς στην Java) αντί της δυναμικής παραγωγής πηγαίου κώδικά (build-time code generation), εξασφαλίζοντας με τον τρόπο αυτό την διαφάνεια του συστήματος σε αρκετά μεγάλο βαθμό. Η συσχέτιση μεταξύ των αντικειμένων και του μοντέλου της βάσεως επιτυγχάνεται μέσω ενός εγγράφου XML όπου περιγράφονται οι αντιστοιχήσεις των κλάσεων των αντικειμένων και των πεδίων τους, με τους πίνακες και τις στήλες της βάσης. *[Richardson, 05]*, *[Bauer, 05]*

Σημειώνεται ότι κανένα από τα μέχρι στιγμής υπάρχοντα συστήματα ORM δεν εξασφαλίζει την πλήρη διαφάνεια μεταξύ βάσης και χρήστη καθώς πάντα απαιτείται ο ορισμός μετά-πληροφοριών (metadata) που συσχετίζονται με την ύπαρξη της βάσης.

➢ ***Αντικείμενα δεδομένων Java (Java Data Objects, JDO).***

Η JDO αποτελεί μία προδιαγραφή για την αποθήκευση και ανάκτηση αντικειμένων (object persistence) της γλώσσας προγραμματισμού Java με κύριο χαρακτηριστικό την διαφάνεια ως προς την *υπηρεσία αποθήκευσης* (*persistence service*).

Η διαφάνεια αυτή επιτυγχάνεται γιατί η JDO αφενός δεν αναφέρεται στον ορισμό συσχετίσεων μεταξύ πεδίων κλάσεων και μοντέλου μέσου αποθήκευσης (π.χ. τους πινάκες και τις στήλες μίας σχεσιακής βάσης δεδομένων όπως συμβαίνει στην ORM), αφετέρου θέτει χαλαρούς κατασκευαστικούς περιορισμούς στις κλάσεις των αντικειμένων που μπορούν να αποθηκευτούν (δεν είναι απαραίτητο να κληρονομούν ή να υλοποιούν συγκεκριμένες κλάσεις ή διεπιφάνειες αντίστοιχα). Στην JDO, η «βάση δεδομένων» αντικαθίσταται από οποιοδήποτε αποθηκευτικό μέσο (datastore) ανεξαρτήτου μορφής και σχήματος, ακόμα και απλών αρχείων. Η αναζήτηση των αντικειμένων δεν πραγματοποιείται βάσει ερωτημάτων που εμπλέκουν στοιχεία του αποθηκευτικού μέσου (π.χ. πίνακες και στήλες) αλλά βάσει πληροφοριών που συσχετίζονται με το μοντέλο και τα χαρακτηριστικά των αντικειμένων.

Ο προγραμματιστής αυτό που πρέπει να καθορίσει, μέσω ενός εγγράφου XML, είναι οι κλάσεις αντικειμένων, τα πεδία των κλάσεων και τα αντικείμενα που επιθυμεί να αποθηκευτούν.

Οι προδιαγραφές της JDO 1.0 και JDO 2.0 αναπτύχθηκαν υπό τον έλεγχο της *Java Community Process* (*JCP*) και περιγράφονται από τα JSR 12 και JSR 243 (*Java Specification Request*) αντίστοιχα. Η JDO έχει ενσωματωθεί σε πολλαπλά επίπεδα με την αρχιτεκτονική της Java EE (Java Enterprise Edition). *[Ezzio, 03]*, *[Java, jdo]*, *[JSR, 12]*.

4.6.2 Υλοποίηση

Ο προτεινόμενος μηχανισμός διαχείρισης πληροφοριών ουσιαστικά αποτελείται από δύο επιμέρους μηχανισμούς, τον μηχανισμό αποθήκευσης και ανάκτησης αντικειμένων. Οι μηχανισμοί αυτοί περιγράφονται ακολούθως.

4.6.2.1 Μηχανισμός αποθήκευσης αντικειμένων

Όπως ήδη έχει αναφερθεί η απεικόνιση της καταστάσεως των αντικειμένων πραγματοποιείται σε έγγραφα XML, τα *έγγραφα αποθήκευσης* ή *καταστάσεως*, με την βοήθεια της εφαρμογής XStream.

Ένα έγγραφο καταστάσεως πρέπει επιπλέον να περιέχει εκείνα τα στοιχεία τα οποία εξασφαλίζουν την μοναδικότητα του ως προς το αντικείμενο που περιγράφει έτσι ώστε να είναι δυνατή η μεταξύ τους αντιστοίχηση. Η αντιστοίχηση αυτή επιτυγχάνεται με την βοήθεια ενός κωδικού, μοναδικού για το αντικείμενο ως προς τα υπόλοιπα της ίδιας κλάσης, που καλείται *κωδικός ταυτοποίησης καταστάσεως αντικειμένου* (*State Object Identifier*, *SOId*), και το όνομα της κλάσης αυτής.

Είναι προφανές ότι πολλά έγγραφα καταστάσεως μπορεί να αναφέρονται στο ίδιο αντικείμενο αφού είναι πιθανό να απεικονίζουν την κατάσταση του σε διαφορετικά σημεία του κύκλου ζωής του. Το γεγονός αυτό ώθησε στον ορισμό ενός επιπλέον μεγέθους, καλούμενο ως *κωδικός στιγμιότυπου καταστάσεως* (*Snapshot State Code*, *SSC*), το οποίο χρησιμοποιείται για την διάκριση των εγγράφων αποθήκευσης που αναφέρονται στο ίδιο αντικείμενο. Ο κωδικός στιγμιότυπου καταστάσεως καλείται και *κριτήριο ταξινόμησης στιγμιότυπων* (*Snapshot Sorting Rule*, *SSR*) αφού, όπως θα δούμε παρακάτω, ορίζει την σειρά κατάταξης των στοιχείων που προκύπτουν από την αναζήτηση πληροφοριών σε έγγραφα XML που αναφέρονται στο ίδιο αντικείμενο (δηλαδή με το ίδιο SOId).

Λόγω της σημαντικότητας των δύο παραπάνω μεγεθών (SOId και SSC) κάθε αντικείμενο που επιθυμείται να αποθηκευτεί προτείνεται (χωρίς να είναι απαραίτητο όπως θα δούμε στην συνέχεια) να υλοποιεί την διεπιφάνεια SerializedDBXML. Στην διεπιφάνεια αυτή ορίζονται οι μέθοδοι getStateObjId() και sortingValue() οι οποίες επιστρέφουν τα SOId και SSC αντίστοιχα.

Για την παραγωγή των SOId και SSC ο χρήστης μπορεί να αξιοποιήσει τις δυνατότητες της κλάσης KeyGenerator η οποία χρησιμοποιείται εκτενώς στην υλοποίηση της εν λόγω πλατφόρμας προσομοίωσης. Η KeyGenerator, μέσω της μεθόδου generateKey, ορίζει ομάδα κωδικών και εξασφαλίζει την μοναδικότητα του παραγόμενου κωδικού στην ομάδα που ανήκει. Για παράδειγμα, αν η ομάδα component περιέχει τους κωδικούς component1 και component2, τότε η κλήση της μεθόδου generateKey("component", null) επιστρέφει τον κωδικό component3 (περισσότερες πληροφορίες στο API). Σημειώνεται η δυνατότητα αποθήκευσης και ανάκτησης των κωδικών στην και από την βάση αντίστοιχα ώστε η αρχικοποίηση της KeyGenerator σύμφωνα με τους ήδη καταχωρημένους κωδικούς να είναι εφικτή.

Στο παρόν σύστημα διαχείρισης πληροφοριών πραγματοποιείται αυτόματη καταγραφή της καταστάσεως μιας αυτοφυούς ιδιότητας με την έγερση κάποιου γεγονότος (event-driven). Το γεγονός αυτό μπορεί να είναι α) η μεταβολή των διαθέσιμων μονάδων της ιδιότητας, β) η διέλευση ορισμένου χρονικού διαστήματος που προσδιορίζεται από προκαθορισμένη μέθοδο η οποία πληροί τις προδιαγραφές μεταβιβαστή και γ) η ικανοποίηση συγκεκριμένης συνθήκης. Επί του παρόντος η καταγραφή μιας ιδιότητας λαμβάνει χώρα μόνο με την μεταβολή των διαθέσιμων μονάδων της η οποία ανιχνεύεται μέσω του aspect PropertyChangedDetector.
Στις παραγράφους που ακολουθούν περιγράφεται ο τρόπος αποθήκευσης των ιδιοτήτων ο οποίος συγχρόνως αποτελεί υπόδειγμα για την αποθήκευση οποιοδήποτε άλλου αντικειμένου από τον χρήστη.

Όπως έχει ήδη αναφερθεί η κατάσταση μίας ιδιότητας, σε επίπεδο υλοποίησης, περιγράφεται από ένα αντικείμενο της κλάσης PropertyState όπως αυτή απεικονίζεται στο Σχήμα 37. Έτσι, το σύνολο των καταστάσεων μιας αυτοφυούς ιδιότητας, κατά την διάρκεια της προσομοίωσης, μπορεί να αποτυπωθεί με την αποθήκευση των αντίστοιχων καταστάσεων του συσχετιζόμενου PropertyState αντικειμένου.

```
public class PropertyState implements SerializedDBXML {

	//πεδία που περιγράφουν την κατάσταση της ιδιότητας με κωδικό _propertyId
	/*μοναδικός κωδικός ιδιότητας που προκύπτει από την εφαρμογή της μεθόδου
	KeyGenerator.generateKey(…)*/
	private String _propertyId = null;

	private long _totalAmount = 100;//συνολικές μονάδες ιδιότητας

	private long _boundAmount = 0;//δεσμευμένες μονάδες ιδιότητας

	private int _nShared = 0;//κατακερματισμός ιδιότητας

	//μέθοδοι ανάκτησης των τιμών των πεδίων της κλάσης
	....

	//υλοποίηση των μεθόδων που ορίζονται στην διεπιφάνεια SerializedDBXML
	//κωδικός ταυτοποίησης καταστάσεως αντικειμένου
	public String getStateObjId() {
		return _propertyId;
	}

	/*κωδικός στιγμιότυπου καταστάσεως. Ο χρόνος καταγραφής της καταστάσεως
	του αντικειμένου, σε ms*/
	public String sortingValue() {
		return (new Long(SimulationSystem.currentTime())).toString();
	}
}
```

Σχήμα 37 – *Τμήμα κώδικα της κλάσης PropertyState.*

Σύμφωνα με τα παραπάνω, η κλάση PropertyState πρέπει να υλοποιεί την διεπιφάνεια SerializedDBXML στην οποία ορίζονται οι μέθοδοι getStateObjId και sortingValue. Η διάκριση μεταξύ των αντικειμένων της PropertyState και κατά συνέπεια των αντικειμένων που αντιπροσωπεύουν τις ιδιότητες (κλάση SCompProperty) πρέπει να πραγματοποιηθεί σε δύο επίπεδα. Στο πρώτο επίπεδο θα πρέπει να εξασφαλίζεται ο διαχωρισμός των αντικειμένων των ιδιοτήτων οι οποίες έχουν τον ίδιο κωδικό ταυτοποίησης αλλά ανήκουν σε διαφορετικές οντότητες, ενώ στο δεύτερο θα λαμβάνει χώρα η διάκριση αυτών που ανήκουν στην ίδια οντότητα. Με δεδομένο ότι κάθε οντότητα έχει το πολύ ένα αντίγραφο μιας ιδιότητας και ότι οι κωδικοί ταυτοποίησης τόσο των οντοτήτων όσο των ιδιοτήτων είναι μοναδικοί (παράγραφος 4.1.1), η διάκριση μεταξύ των αντικειμένων της PropertyState, επιτυγχάνεται από έναν κωδικό που προκύπτει από τον συνδυασμό των κωδικών της ιδιότητας και της οντότητας στην οποία ανήκει.

Λαμβάνοντας υπόψη τα τελευταία στοιχεία η μέθοδος getStateObjId επιστρέφει (μέσω του πεδίου _propertyId) το αλφαριθμητικό που προκύπτει από την εφαρμογή της μεθόδου generateKey της κλάσης KeyGenerator, με ορίσματα τα KeyGenerator.PROPERTY_GROUP και new String[] {_owner, _name}, όπου KeyGenerator.PROPERTY_GROUP το προκαθορισμένο όνομα της ομάδας κωδικών που χρησιμοποιείται για την διάκριση των αντικειμένων των ιδιοτήτων και _owner, _name ο κωδικός ταυτοποίησης της οντότητας και της ιδιότητας αντίστοιχα (όπως προκύπτουν από το XML έγγραφο διαμόρφωσης του συστήματος). Αν για παράδειγμα οι τιμές των ορισμάτων _owner και _name είναι vpc1 και load αντίστοιχα τότε ο κωδικός του αντικειμένου της ιδιότητας είναι vpc1###load.

Για την διάκριση μεταξύ των καταστάσεων που αναφέρονται στο ίδιο αντικείμενο μιας ιδιότητας χρησιμοποιείται ο χρόνος καταγραφής των καταστάσεων αυτών. Έτσι, η μέθοδος

sortingValue() επιστρέφει τον χρόνο του συστήματος, σε ms, κατά την αποθήκευση του αντίστοιχου PropertyState αντικειμένου.

Η αποθήκευση ενός αντικειμένου πραγματοποιείται μέσω της μεθόδου serializedObject(SerializedDBXML) της κλάσης ClassSerializer. Η μέθοδος αυτή είναι υπεύθυνη για α) την απεικόνιση της κατάστασης του αντικειμένου σε XML έγγραφο με την βοήθεια της εφαρμογής XStream, β) την τροποποίηση του εγγράφου αυτού ώστε να εξασφαλίζεται η μοναδικότητα του ως προς το αντικείμενο που αντιπροσωπεύει/ αναπαριστά και τέλος γ) την αποθήκευση του στην βάση με κλήση της μεθόδου DBStoreDataInterface.store(SerializedDescriptor).

Στα Σχήματα 38 και 39 περιγράφονται τα XML έγγραφα όπως προκύπτουν από την απεικόνιση της κατάστασης ενός αντικειμένου τύπου PropertyState μέσω της XStream και όπως αποθηκεύονται στην βάση με την προσθήκη των απαραίτητων στοιχείων αντίστοιχα.

```
<pBSimPlt.coreComponents.PropertyState>
        <_propertyId>vpc1###load</_propertyId>
        <_totalAmount>100</_totalAmount>
        <_boundAmount>55</_boundAmount>
        <_nShared>2</_nShared>
</pBSimPlt.coreComponents.PropertyState >
```

Σχήμα 38 - *XML έγγραφο που προκύπτει από την XStream.*

```
<item id="vpc1###load" sortingvalue="1166804118406">
        <_propertyId>vpc1###load</_propertyId>
        <_totalAmount>100</_totalAmount>
        <_boundAmount>55</_boundAmount>
        <_nShared>2</_nShared>
</item>
```

Σχήμα 39 – *Αποθηκευμένο XML έγγραφο (μετά την τροποποίηση).*

Από την σύγκριση των δύο παραπάνω εγγράφων παρατηρείται ότι η μεταξύ τους διαφοροποίηση έγκειται στο εξωτερικό στοιχείο τους. Το όνομα του εξωτερικού στοιχείου του εγγράφου που προκύπτει από την εφαρμογή της XStream ταυτίζεται με το πλήρες όνομα της κλάσης του αντικειμένου το οποίο απεικονίζεται στο έγγραφο αυτό. Αντιθέτως, το έγγραφο που αποθηκεύεται στην βάση έχει σαν εξωτερικό στοιχείο το <item> με ιδιότητες τις id και sortingvalue. Οι ιδιότητες αυτές φιλοξενούν τις τιμές που επιστρέφονται από την κλήση των μεθόδων getStateObjId και sortingValue αντίστοιχα του αντικειμένου που περιγράφεται.

Ο μετασχηματισμός του εγγράφου αποθήκευσης στην τελική του μορφή επιτυγχάνεται με χρήση της γλώσσας XSLT βάσει ενός *προτύπου μετατροπής* (*transformation template*) το οποίο φαίνεται στο Σχήμα 40.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
        <xsl:template match="/*">
                <xsl:element name="item">
                <xsl:attribute name="id">stateObjId</xsl:attribute>
                <xsl:attribute name="sortingvalue">sortvalue</xsl:attribute>
                <xsl:copy-of select="*"/>
        </xsl:element>
        </xsl:template>
</xsl:stylesheet>
```

όπου **stateObjId** και **sortvalue** οι επιστρεφόμενες τιμές των μεθόδων getStateObjId και sortingvalue αντίστοιχα.

Σχήμα 40 – *XSLT έγγραφο*.

Τα έγγραφα καταστάσεως των αντικειμένων μιας κλάσης, π.χ. της pBSimPlt.coreComponents.PropertyState, αποθηκεύονται στην βάση ως στοιχεία πρώτου επιπέδου ενός έγγραφου XML με εξωτερικό στοιχείο το οποίο έχει το όνομα της κλάσης αυτής, όπως χαρακτηριστικά φαίνεται στο Σχήμα 41.

```
<pBSimPlt.coreComponents.PropertyState>
        <item id="vpc1###load" sortingvalue="1166804118406"> .... </item>
        <item id="vpc2###hd" sortingvalue="1166804118406"> .... </item>
        <item id="vpc1###load" sortingvalue="1166804119406"> .... </item>
        …
        <item id="vpc2###load" sortingvalue="1166804119500"> .... </item>
        …
</pBSimPlt.coreComponents.PropertyState>
```

Όπου vpc1, vpc2 και load, hd κωδικοί ταυτοποίησης οντοτήτων και ιδιοτήτων αντίστοιχα.

Σχήμα 41 – *Έγγραφα καταστάσεως των αντικείμενων της κλάσης pBSimPlt.coreComponents.PropertyState*.

Είναι προφανές ότι τα μεγέθη SOId και SSC δεν προέρχονται από την ικανοποίηση κάποιου κανόνα του σχήματος της βάσης αλλά από την ανάγκη συσχέτισης ενός εγγράφου ή εγγράφων XML με την κατάσταση ή τις καταστάσεις ενός αντικειμένου κατά την διάρκεια της προσομοίωσης. Τα μεγέθη αυτά θα μπορούσαν κάλλιστα να σχηματισθούν ώστε να εκφράζουν οποιαδήποτε άλλου είδους συσχέτιση ή πληροφορία επιθυμείται, η ακόμα και να παραληφθούν. Για τον λόγο αυτό παρέχεται μία επιπλέον μέθοδος, η serializedObject(Object), η οποία μπορεί να χρησιμοποιηθεί για την καταγραφή αντικειμένων χωρίς να θέτει κάποιο σχεδιαστικό περιορισμό σε αυτά. Στην περίπτωση αυτή, οι τιμές των ιδιοτήτων id και sortingvalue των εγγράφων καταστάσεως που προκύπτουν είναι απροσδιόριστες.

4.6.2.2 Μηχανισμός ανάκτησης αντικειμένων

Η αναζήτηση και η ανάκτηση δεδομένων από την εγγενή βάση XML πραγματοποιείται μέσω των μεθόδων query(String objId, String fpath) και query(String objIdXPathExp, String sortvalueXPathExp, String objId, String fpath) της κλάσεως StateTraceManager.

Η query(String objId, String fpath) χρησιμοποιείται για την ανάκτηση πληροφοριών από αντικείμενα τα οποία αποθηκεύτηκαν μέσω της serializedObject(SerializedDBXML) και κατά συνέπεια συμμορφώνονται με τις προδιαγραφές που ορίστηκαν στις προηγούμενες παραγράφους (υλοποίηση της διεπιφάνειας SerializedDBXML). Η μέθοδος αυτή επιστρέφει το σύνολο των αντικειμένων μιας κλάσης που προσδιορίζονται από την fpath και σχηματίζονται από τα αντίστοιχα έγγραφα αποθήκευσης με κωδικό ταυτοποίησης καταστάσεως objId (δηλαδή αποτελούν στιγμιότυπα καταστάσεως ίδιου αντικειμένου). Η fpath αποτελεί μία έκφραση (expression), καλούμενη ως *διαδρομή θέσης πεδίου* (*Field Path Location*, *FPathL*) και χρησιμοποιείται για τον εντοπισμό δεδομένων σε οποιοδήποτε σημείο της δομής των αποθηκευμένων αντικειμένων. Οι κανόνες σχηματισμού της διαδρομής είναι ίδιοι με αυτούς της γλώσσας XPath, η οποία χρησιμοποιείται για τον εντοπισμό στοιχείων των εγγράφων XML, καθώς η ομοιότητα μεταξύ εγγράφων XML και γράφων αντικειμένων είναι χαρακτηριστική (ιεραρχική δομή, στοιχείο XML – κόμβος γράφου, όνομα στοιχείου – όνομα κόμβου).

Για παράδειγμα, έστω ότι αντικείμενα τύπου packageP.Parent έχουν αποθηκευτεί στην βάση από όπου επιθυμείται η ανάκτηση α) αντικειμένων τύπου packageP.Parent και β) τιμών του πεδίου cfield των packageC.Child αντικειμένων που περιέχονται στην δομή των packageP.Parent αντικειμένων. Σύμφωνα με τις δομές των εν λόγω κλάσεων οι οποίες περιγράφονται στο Σχήμα 42, οι εκφράσεις που θα χρησιμοποιηθούν για την ανάκτηση των παραπάνω δεδομένων είναι οι «packageP.Parent» και «packageP.Parent/pfield/cfield» για την πρώτη και δεύτερη περίπτωση αντίστοιχα.

```
package packageP
public class Parent implements SerializedDBXML {
   //πεδία
   packageC.Child pfield = new packageC.Child();
   String type = null;
   …

   //μέθοδοι
   public String getStateObjId() {...}
   public String sortingValue() {...}
   ....
}
```

```
package packageC
public class Child {
        //πεδία
        int cfield = 2;
        …
}
```

Σχήμα 42 – *Τμήμα κώδικα, κλάσεις προς καταγραφή.*

Τα επιστρεφόμενα από την μέθοδο αντικείμενα εσωκλείονται σε αντικείμενα τύπου QResultItem και περιέχονται σε μία συνδεδεμένη λίστα όπου είναι ταξινομημένα κατά αύξουσα σειρά με βάση την *φυσική διάταξη* (παράγραφος 4.3) των τιμών του κριτηρίου ταξινόμησης στιγμιότυπων (SSR) των αντίστοιχων εγγράφων καταστάσεως από όπου προκύπτουν. Κάθε αντικείμενο που ανακτάται από την βάση λαμβάνεται με εφαρμογή της μεθόδου getValue() πάνω στο αντίστοιχο αντικείμενο QResultItem. Η μέθοδος αυτή επιστρέφει ένα αντικείμενο τύπου Object με αποτέλεσμα να απαιτείται η ρητή προσαρμογή του στον κατάλληλο τύπο (explicit casting) από τον χρήστη.

Η query(String objIdXPathExp, String sortvalueXPathExp, String fpath) αποτελεί γενικευμένη μορφή της πρώτης μεθόδου αφού μπορεί επιπλέον να χρησιμοποιηθεί για την ανάκτηση δεδομένων από αντικείμενα τα οποία δεν υλοποιούν την διεπιφάνεια SerializedDBXML (δηλαδή αντικείμενα που αποθηκεύτηκαν μέσω της μεθόδου serializedObject(Object)). Στην μέθοδο αυτή απαιτείται ο καθορισμός δύο επιπρόσθετων παραμέτρων, των objIdXPathExp και sortvalueXPathExp οι οποίες αποτελούν FPathL εκφράσεις και προσδιορίζουν τα δεδομένα που αντιπροσωπεύουν τα μεγέθη SOId και SSC των αντικειμένων μίας κλάσεως αντίστοιχα. Στην συγκεκριμένη περίπτωση τα SOId και SSC δεν εκφράζονται αποκλειστικά από παραστάσεις αλφαριθμητικών, όπως επιβάλλεται από τον ορισμό των μεθόδων της διεπιφάνειας SerializedDBXML, αλλά από οποιοδήποτε τύπο ο οποίος υλοποιεί τις κατάλληλες μεθόδους (equals(Object) και compareTo(Object) της διεπιφάνειας Comparable) για την σύγκριση και τον καθορισμό της φυσικής διάταξης των αντικειμένων του τύπου αυτού. Επειδή επί του παρόντος η ανάκτηση δεδομένων τα οποία εσωκλείονται στα αποθηκευμένα αντικείμενα περιορίζεται σε αυτά που εκφράζουν *πρωταρχικούς* (*primitive*) ή *αλφαριθμητικούς* τύπους, τα μεγέθη SOId και SSC πρέπει να προκύπτουν από τους τύπους αυτούς. Σημειώνεται επίσης ότι τα παραπάνω δεδομένα επιστρέφονται ως αλφαριθμητικά, δηλαδή υπό την μορφή αντικειμένων τύπου String.

Στο Σχήμα 43 περιγράφεται, σε επίπεδο υλοποίησης, η ανάκτηση των καταστάσεων μίας ιδιότητας (π.χ. η load) με χρήση των δύο προαναφερθέντων μεθόδων. Η υλοποίηση βασίζεται στα στοιχεία της παραγράφου 4.6.2.1 όπου αναλύεται ο τρόπος αποθήκευσης των ιδιοτήτων. Επιπλέον, θεωρείται ότι η κλάση PropertyState περιέχει το πεδίο _recordTime (τύπου long) όπου φιλοξενείται ο χρόνος καταγραφής του αντίστοιχου αντικειμένου στην εγγενή βάση δεδομένων XML.

```
....
//υπεύθυνο αντικείμενο για την αναζήτηση δεδομένων από την βάση
StateTraceManager stm = new StateTraceManager();

// κωδικός ταυτοποίησης καταστάσεως αντικειμένου που επιθυμείται να ανακτηθεί
String pid = "vpc1###load";

//έκφραση για τον εντοπισμό των δεδομένων που επιθυμείται να ανακτηθούν
String fpath = "pBSimPlt.coreComponents.PropertyState";

/*---Πρώτος τρόπος ανάκτησης των καταστάσεων της ιδιότητας load στην οντότητα vpc1
χρήση της μεθόδου query(String objId, String fpath)*/

LinkedList qResProps = stm.query(pid, fpath); //ανάκτηση δεδομένων

/*----Δεύτερος τρόπος ανάκτησης των καταστάσεων της ιδιότητας load στην οντότητα vpc1
χρήση της μεθόδου query(String objIdXPathExp, String sortvalueXPathExp, String fpath)*/

/*έκφραση για τον εντοπισμό του δεδομένου που εκφράζει το μέγεθος SOId για τα αντικείμενα τύπου
PropertyState*/
String objIdXPathExp = "pBSimPlt.coreComponents.PropertyState/_propertyId";
/*έκφραση για τον εντοπισμό του δεδομένου που εκφράζει το μέγεθος SSC για τα αντικείμενα τύπου
PropertyState*/
String sortvalueXPathExp = "pBSimPlt.coreComponents.PropertyState/_recordTime";
//ανάκτηση δεδομένων
LinkedList qResProps = stm.query(objIdXPathExp, sortvalueXPathExp, fpath)

ListIterator listProps = qResProps.listIterator();

while(listProps.hasNext()) {
        QResultItem wrappedProp = (QResultItem) listProps.next();
        //ανάκτηση ιδιότητας με κατάλληλο προσαρμογή των τύπου
        PropertyState propState = (PropertyState) wrappedProp.getValue();
        /*ανάκτηση επιμέρους στοιχείου της κατάστασης της ιδιότητας, π.χ. το σύνολο των
        δεσμευμένων μονάδων της*/
        long ba = propState.getBoundAmount();
        ….
}
….
```

Σχήμα 43 – *Τμήμα κώδικα, ανάκτηση καταστάσεως μίας ιδιότητας.*

Στο προηγούμενο παράδειγμα ο χρήστης θα μπορούσε να λάβει άμεσα κάποιο από τα επιμέρους στοιχεία της κατάστασης της ιδιότητας load προσδιορίζοντας το δεδομένο που αναπαριστά το στοιχείο αυτό (σε αντιδιαστολή με την ανάκτηση του στοιχείου μέσω κλήσης της κατάλληλης μεθόδου στο PropertyState αντικείμενο). Για παράδειγμα η ανάκτηση των δεσμευμένων μονάδων (πεδίο _boundAmount της κλάσης PropertyState) πραγματοποιείται αν η έκφραση που φιλοξενείται στην μεταβλητή fpath αντικατασταθεί από την "pBSimPlt.coreComponents.PropertyState/_boundAmount". Στην περίπτωση αυτή, τα αντικείμενα που επιστρέφονται από την εφαρμογή της μεθόδου getValue() στα αντίστοιχα QResultItem αντικείμενα, είναι τύπου String και εκφράζουν την χρονική μεταβολή των δεσμευμένων μονάδων της ιδιότητας load στην οντότητα vpc1.

Η ταξινόμηση των αποτελεσμάτων μιας αναζήτησης βάσει ενός δυναμικά προσδιοριζόμενου μεγέθους προσδίδει επιπρόσθετη ευελιξία στο προτεινόμενο σύστημα διαχείρισης πληροφοριών. Για το ίδιο σύνολο αποτελεσμάτων μπορούν να χρησιμοποιηθούν διάφορα

κριτήρια ταξινόμησης ανάλογα με τις ανάγκες της προσομοίωσης. Η χρήση του χρόνου καταγραφής των πληροφοριών ως κριτήριο ταξινόμησης στιγμιότυπων (SSR) αποτελεί την συνηθέστερη περίπτωση αφού η κατάταξη των επιστρεφόμενων δεδομένων από την αναζήτηση απεικονίζει την μεταβολή τους συναρτήσει του χρόνου.

4.6.2.3 Γραφικό περιβάλλον

Επί του παρόντος υλοποιείται γραφικό περιβάλλον (κοινώς GUI, Graphical User Interface) μέσω του οποίου ο χρήστης έχει την δυνατότητα να αναζητά και να ανακτά πληροφορίες από την βάση δεδομένων. Το περιβάλλον αυτό, καλούμενο και ως Database Information Center (DIC), απαρτίζεται από τις ακόλουθες τρεις περιοχές, όπως χαρακτηριστικά φαίνεται και από το Σχήμα 44:

➢ *Περιοχή γραφικής απεικόνισης αντικειμένων* (*Graphical Object Area*, *GOA*)

Στην περιοχή αυτή λαμβάνει χώρα η απεικόνιση των αποθηκευμένων αντικειμένων σε δενδρική μορφή. Ο χρήστης έχει την δυνατότητα να συλλέξει πληροφορίες (π.χ. ο τύπος και η τιμή) για οποιοδήποτε στοιχείο (από οποιοδήποτε σημείο της δομής ...) ενός αντικειμένου με απλή επιλογή του αντίστοιχου κόμβου του δέντρου από το οποίο αναπαρίσταται.

Η GOA ουσιαστικά αντικατοπτρίζει στιγμιότυπα της βάσης, επιτρέποντας στον χρήστη να έχει σαφή εικόνα της καταγραφείσας πληροφορίας σε μια χρονική στιγμή.

➢ *Περιοχή κατασκευής ερωτημάτων αναζήτησης* (*Query Composition Area*, *QCA*)

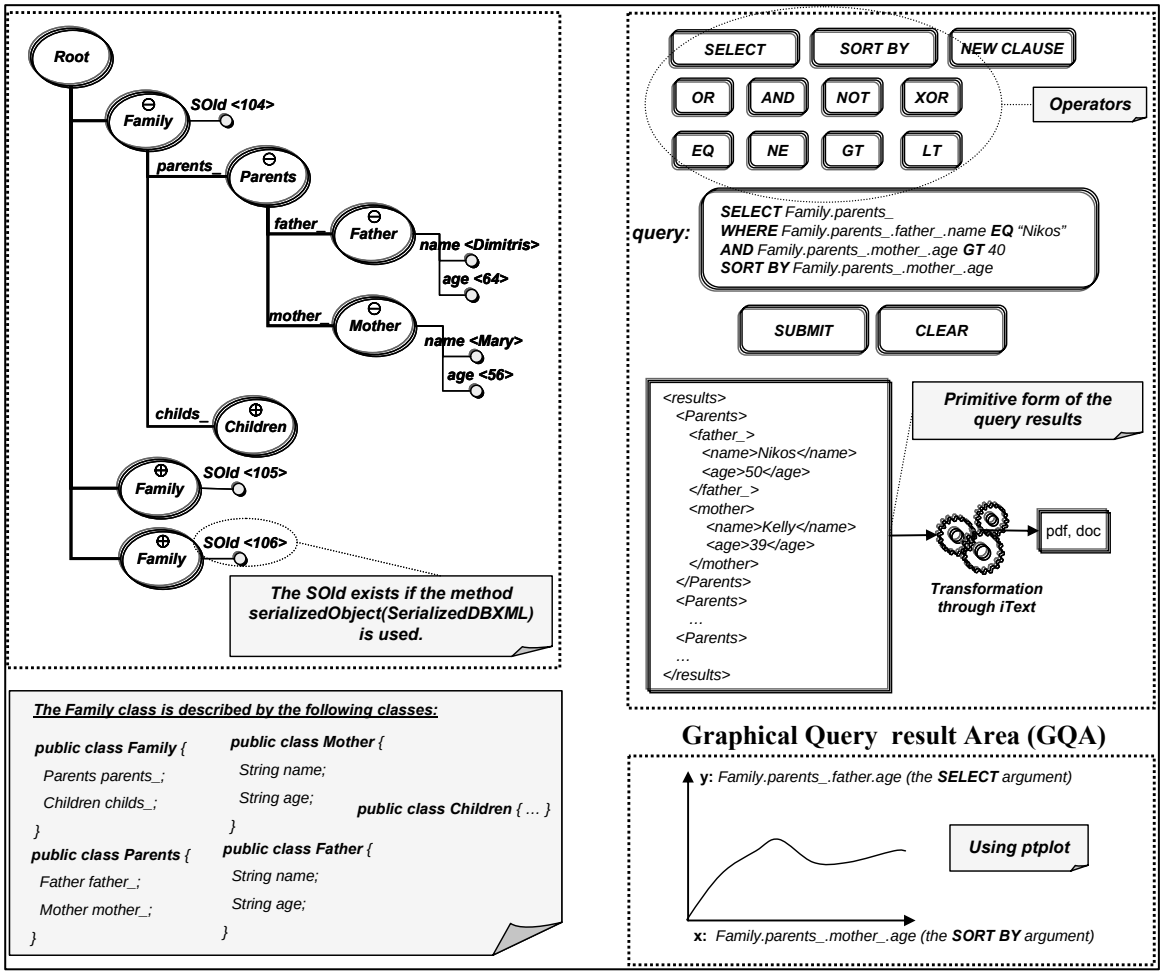
Στην QCA παρέχεται η δυνατότητα δημιουργίας ερωτημάτων αναζήτησης δεδομένων τα οποία μπορούν να αναφέρονται σε οποιοδήποτε σημείο της δομής των αποθηκευμένων αντικειμένων. Τα αποτελέσματα της αναζήτησης περιγράφονται μέσω XML (ανάλογη μορφή με αυτή που καταγράφονται τα αντικείμενα μέσω της XStream) και δύνανται να ταξινομηθούν βάσει πληροφοριών που εσωκλείονται στα αντικείμενα πάνω στα οποία εφαρμόζεται το αντίστοιχο ερώτημα (ο προσδιορισμός των διαφόρων στοιχείων γίνεται με τρόπο παρόμοιο με αυτόν που ακολουθείται για τον σχηματισμό των FPathL εκφράσεων).

Αναφορικά με τις κλάσεις που περιγράφονται στο Σχήμα 42, μερικές χαρακτηριστικές μορφές ερωτημάτων είναι οι ακόλουθες: «επίστρεφε τις τιμές του στοιχείου pfield/cfield των αντικειμένων της κλάσης packageP.Parent», «επίστρεφε τα αντικείμενα της κλάσης packageP.Parent», «επίστρεφε τα αντικείμενα της κλάσης packageP.Parent ταξινομημένα βάσει των τιμών του στοιχείου pfield/cfield της δομής τους», «επίστρεφε τα αντικείμενα της κλάσης packageP.Parent όπου το στοιχείο pfield/cfield της δομής τους έχει τιμή μεγαλύτερη του 30», «επίστρεφε τα αντικείμενα της κλάσης packageP.Parent όπου το στοιχείο type έχει τιμή ίση με “superior” ΚΑΙ το στοιχείο pfield/cfield έχει τιμή μεγαλύτερη του 200». Είναι προφανές ότι η περιοχή GOA μπορεί να χρησιμοποιηθεί για τον εντοπισμό των επιθυμητών πληροφοριών προς τον σχηματισμό των ερωτημάτων.

Αξίζει να σημειωθεί ότι τα αποτελέσματα δύναται να αποθηκευτούν σε διάφορες μορφές αρχείων (π.χ. doc, pdf ή απλώς σε txt) με την βοήθεια του εργαλείου iText.

➢ *Περιοχή γραφικής απεικόνισης αποτελεσμάτων αναζήτησης* (*Graphical Query result Area*, *GQA*)

Στην περιοχή αυτή, με την βοήθεια της εφαρμογής ptplot, πραγματοποιείται η απεικόνιση της μεταβολής των αποτελεσμάτων μίας αναζήτησης συναρτήσει του μεγέθους, βάσει του οποίου πραγματοποιείται η ταξινόμηση τους. Αυτό βέβαια είναι εφικτό στην περίπτωση που τα εμπλεκόμενα μεγέθη εκφράζουν αριθμητικούς τύπους δεδομένων.



Σχήμα 44 – *Γραφικό περιβάλλον διαχείρισης πληροφοριών.*

4.6.2.4 Υλοποίηση ως Διαδικτυακή Υπηρεσία

Οι μηχανισμοί καταγραφής και ανάκτησης πληροφοριών δεσμεύουν υψηλό ποσοστό (resource consuming) των πόρων του συστήματος στο οποίο εκτελούνται αφού εμπλέκουν συναλλαγές (transactions) με τη βάση δεδομένων. Για τον λόγο αυτό, θεωρήθηκε αναγκαία η προσαρμογή τους σε μία αρχιτεκτονική η οποία επιτρέπει την εκτέλεση των χρονοβόρων (time-consuming) λειτουργιών σε ξεχωριστό από την υπόλοιπη πλατφόρμα περιβάλλον. Εκτιμήθηκε ότι η αρχιτεκτονική αυτή, στην συγκεκριμένη περίπτωση, εντάσσεται στο μοντέλο λειτουργίας της τεχνολογίας των *δικτυακών υπηρεσιών* (*web services*).

Ως διαδικτυακή υπηρεσία χαρακτηρίζεται οποιαδήποτε υπηρεσία είναι διαθέσιμη στο δίκτυο και μπορεί να προσπελαστεί μέσω καθιερωμένων πρωτοκόλλων όπως τα HTTP και SMTP. Κύριο χαρακτηριστικό της, έναντι αντίστοιχων παραδοσιακών τεχνολογιών, όπως οι *CORBA* (*Common Object Request Broker Architecture*), *DCOM* (*Distributed Component Object Model*), Java *RMI* (*Remote Method Invocation*), *GCI* (*Common Gateway Interface*), είναι ότι η προτυποποίηση του *συστήματος μηνυμάτων* (*messaging system*) και γενικότερα η περιγραφή των πληροφοριών που απαιτούνται για την αλληλεπίδραση με την υπηρεσία αυτή (*service description*), βασίζονται στην XML. Το χαρακτηριστικό αυτό σε συνδυασμό με ουδέτερη φύση της XML οδηγεί στην ανεξαρτησία της υλοποίησης των δικτυακών υπηρεσιών τόσο σε επίπεδο γλώσσας προγραμματισμού όσο και σε επίπεδο λειτουργικού συστήματος.

Σύμφωνα με την αρχιτεκτονική της εν λόγω τεχνολογίας οι δύο βασικοί ρόλοι των εμπλεκόμενων συστημάτων, συχνά καλούμενοι και ως δράστες, είναι ο *προμηθευτής* (*service provider*) και ο *καταναλωτής υπηρεσίας* (*service requestor*), δηλαδή αυτός που διαθέτει και καταναλώνει μια υπηρεσία αντίστοιχα.
Η επικοινωνία μεταξύ των δραστών πραγματοποιείται μέσω πρωτοκόλλων βασισμένων στην XML (σε αντίθεση με τα δυαδικά πρωτόκολλα επικοινωνίας *IIOP* (*Internet Inter-ORB Protocol*) και *JRMP* (*Java Remote Method Protocol*) που εφαρμόζονται στις τεχνολογίες CORBA και RMI αντίστοιχα) με επικρατέστερο από αυτά το *SOAP* (*Simple Object Access Protocol*). Το SOAP σε γενικές γραμμές ορίζει τις προδιαγραφές για την μεταφορά δεδομένων και την κλήση απομακρυσμένων μεθόδων (όπως το πρωτόκολλο XML-*RPC*/XML, *Remote Procedures Calls*) μέσω XML μηνυμάτων (SOAP μηνύματα) πάνω από παραδοσιακά πρωτόκολλα χαμηλότερου επιπέδου όπως είναι SMTP, HTTP και FTP. Η *σύνδεση* (*binding*) του SOAP με ευρέως χρησιμοποιούμενα πρωτόκολλα είχε σαν σκοπό την αποδοχή του από την πλειοψηφία των συστημάτων χωρίς περιορισμούς. *[Chappell, 02]*, *[Nagappan, 03]*, *[Cerami, 02]*

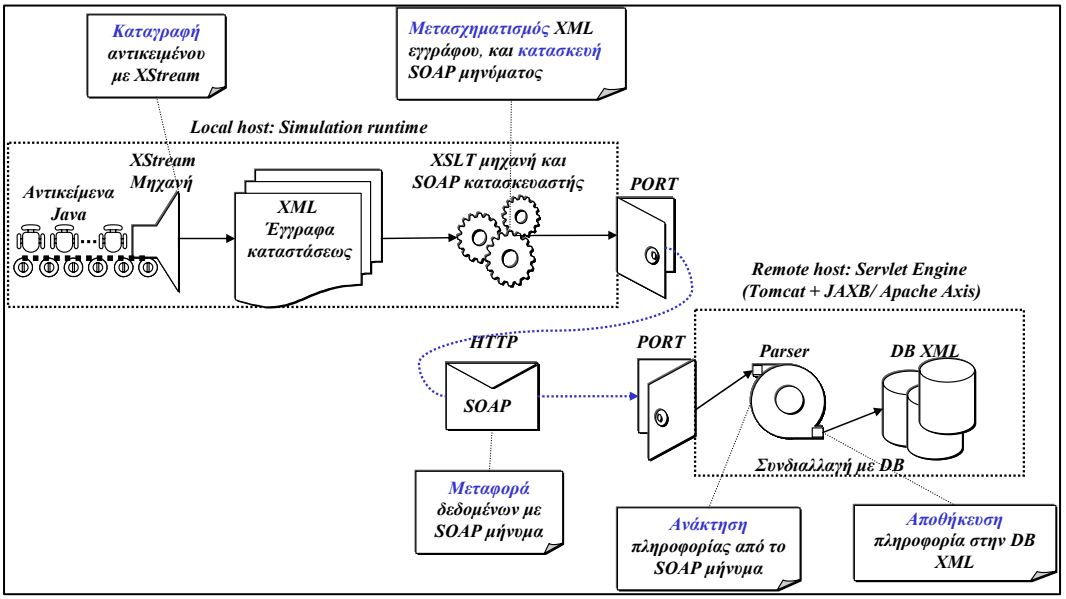
Στη συγκεκριμένη υλοποίηση, το σύστημα διαχείρισης δεδομένων αναλαμβάνει το ρόλο του προμηθευτή των υπηρεσιών της αποθήκευσης και ανάκτησης πληροφοριών στην και από την βάση δεδομένων αντίστοιχα. Τα έγγραφα καταστάσεως που πρόκειται να καταγραφούν, οι απαραίτητες FPathL εκφράσεις για τον εντοπισμό των στοιχείων που επιθυμείται να ανακτηθούν, καθώς και κάποιες άλλες πληροφορίες, εσωκλείονται και αποστέλλονται από τους καταναλωτές των εν λόγω υπηρεσιών μέσω SOAP μηνυμάτων.

Η λειτουργία του συστήματος διαχείρισης δεδομένων ως προμηθευτή υπηρεσιών υλοποιείται μέσω ενός servlet. Τα servlets ανήκουν στην τεχνολογία της Java και μπορούν να χαρακτηριστούν ως οντότητες που επεκτείνουν την λειτουργικότητα ενός εξυπηρετητή (server) παρέχοντας επιπλέον υπηρεσίες. Έτσι, ένα servlet δεν υφίσταται σαν ανεξάρτητη εφαρμογή αλλά φιλοξενείται από έναν εξυπηρετητή μέσω του οποίου επικοινωνεί και παρέχει τις υπηρεσίες του στους αντίστοιχους καταναλωτές. Η αλληλεπίδραση μεταξύ servlet και εξυπηρετητή είναι εφικτή μόνο στην περίπτωση που ο τελευταίος πληροί ένα σύνολο προδιαγραφών (JSR 154) οι οποίες τον καθιστούν *υποδοχέα* (*servlet container*) ή *μηχανή servlet* (*servlet engine*) όπως είθισται να καλείται. Παράδειγμα μηχανής servlet αποτελεί ο δικτυακός εξυπηρετητής (web server) Tomcat ο οποίος και χρησιμοποιείται στην παρούσα υλοποίηση. *[Java, servlets]*, *[JSR, 154]*, *[Tomcat, hp]*

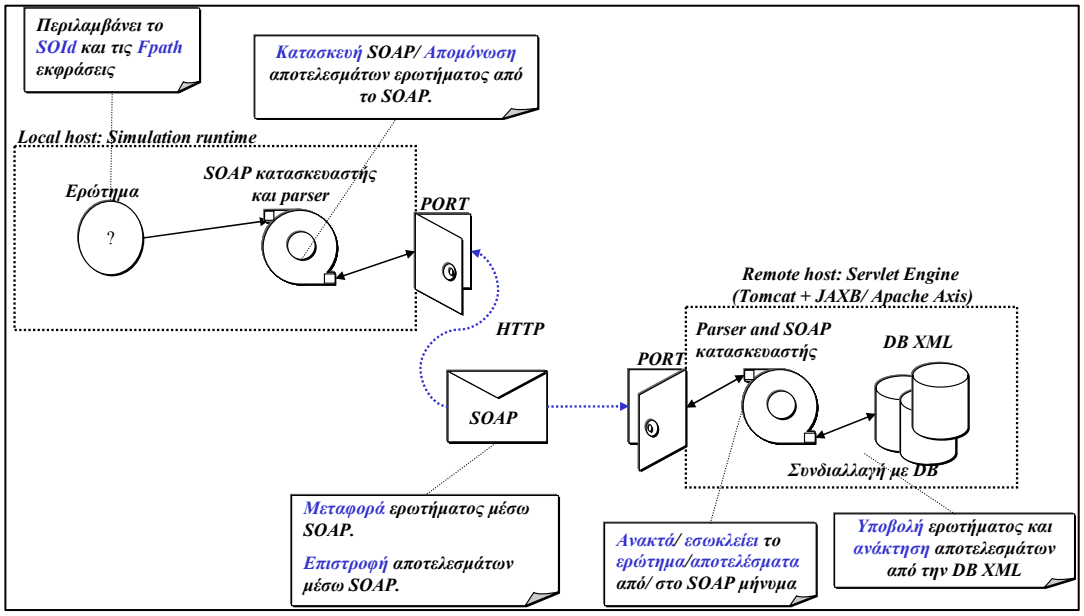
Η αρχιτεκτονική του εν λόγω συστήματος απεικονίζεται στα Σχήματα 45α και 45β. Οι απαραίτητες πληροφορίες (όπως το έγγραφο καταστάσεως, οι FPathL εκφράσεις) για την παροχή των υπηρεσιών ενσωματώνονται στο *σώμα* (*body*) ενός SOAP μηνύματος το οποίο αποστέλλεται στο servlet (κλάση DBServlet) που πραγματοποιεί τις συναλλαγές με την βάση. Εκεί, λαμβάνει χώρα η επεξεργασία του μηνύματος αυτού για την εξαγωγή των παραπάνω πληροφοριών οι οποίες χρησιμοποιούνται για την διεκπεραίωση των υπηρεσιών μέσω κλήσεως των κατάλληλων μεθόδων (μέθοδοι DBStoreDataInterface.store(SerializedDescriptor) και query(...)). Στην περίπτωση ανάκτησης δεδομένων, το servlet επιστρέφει τα αποτελέσματα της αναζήτησης ξανά μέσω ενός SOAP μηνύματος.

Σύμφωνα με το παραπάνω μοντέλο λειτουργίας, οι συναλλαγές με την βάση πραγματοποιούνται στην φυσική τοποθεσία του servlet DBServlet η οποία μπορεί να είναι διαφορετική από αυτή της υπόλοιπης πλατφόρμας, αποφορτίζοντας με τον τρόπο αυτό το περιβάλλον εκτέλεσής της.

Η δυνατότητα αποθήκευσης οποιωνδήποτε αντικειμένων στην βάση επιτρέπει την ανάπτυξη μηχανισμού που καθιστά εφικτή την τμηματική διεξαγωγή των προσομοιώσεων (δημιουργία σημείων ελέγχου, checkpoints). Έτσι, ο χρήστης είναι σε θέση να διακόπτει και να αποκαθιστά την εκτέλεση μιας προσομοίωσης κατά βούληση.



Σχήμα 45α– *Μηχανισμός καταγραφής πληροφοριών ως διαδικτυακή υπηρεσία.*



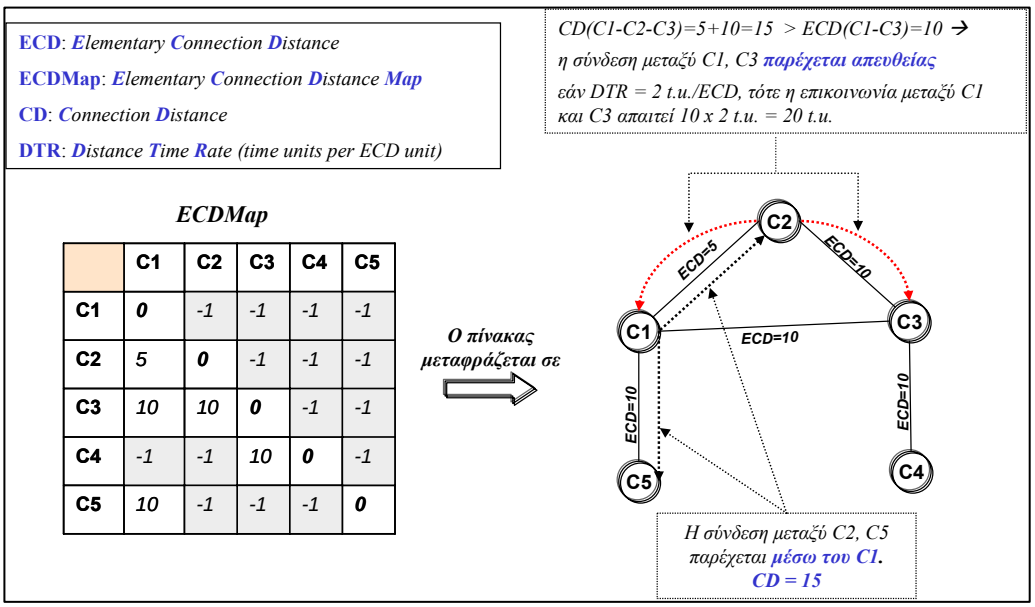
Σχήμα 45β– *Μηχανισμός ανάκτησης πληροφοριών ως διαδικτυακή υπηρεσία.*

4.7 Τοπολογία

Μια πλατφόρμα θα πρέπει να περιέχει την έννοια της *τοπολογίας* ή της *φυσικής σύνδεσης* μεταξύ των οντοτήτων η οποία είναι αυτή που πολλές φορές καθορίζει την συμπεριφορά του προς προσομοίωση συστήματος.

Στην συγκεκριμένη πλατφόρμα ο όρος *απόσταση* και *σημείο/θέση* δεν περιορίζονται στα όρια της απεικόνισης των οντοτήτων στο χώρο, αλλά έχουν σαν στόχο την έκφραση του χρόνου που απαιτείται για την μεταξύ τους επικοινωνία. Αυτό προέκυψε από το γεγονός ότι βασικό χαρακτηριστικό των υπολογιστικών πλεγμάτων δεν είναι η γεωγραφική θέση των επιμέρους μονάδων αλλά η ταχύτητα επικοινωνίας μεταξύ αυτών. Έτσι ο όρος *τοπολογία* αντικαθίσταται από τον όρο *συνδεσμολογία* καθώς ο τελευταίος ταιριάζει περισσότερο στο χαρακτήρα του μοντέλου που αναπτύσσεται ακολούθως.

Στο προτεινόμενο μοντέλο συνδεσμολογίας ορίζεται το αδιάστατο μέγεθος *στοιχειώδης απόσταση διασύνδεσης* (*Elementary Connection Distance*, *ECD*) το οποίο εκφράζει το αντίστροφο της σχέσης μεγέθους της ταχύτητας επικοινωνίας ενός ζεύγους οντοτήτων με αυτές των υπόλοιπων ζευγών του συστήματος, μέσω ενός θετικού πραγματικού αριθμού. Είναι προφανές ότι η τιμή του ECD για το πρώτο ζεύγος είναι αυθαίρετη και πρέπει να χρησιμοποιηθεί ως αναφορά για προσδιορισμό των υπολοίπων διατηρώντας τις επιθυμητές αναλογίες. Για παράδειγμα, ας υποθέσουμε ότι ένα σύστημα αποτελείται από τις οντότητες c1, c2, c3, c4 και c5 όπως χαρακτηριστικά απεικονίζεται στο Σχήμα 46. Με δεδομένο ότι η ταχύτητα επικοινωνίας μεταξύ των (c1, c2) είναι διπλάσια αυτής των (c2, c3), (c3,c4), (c1,c3) και (c1, c5) μερικές από τις δυνατές τιμές των αντίστοιχων στοιχειωδών αποστάσεων διασύνδεσης είναι {5,4,3}, {10,8,6}, {10,8,6}, {10,8,6} και {10,8,6}. Σημειώνεται ότι, κατά σύμβαση το ECD μπορεί να λάβει μηδενική ή αρνητική τιμή στην περίπτωση που η επικοινωνία των συσχετιζόμενων οντοτήτων πραγματοποιείται ακαριαία (η ταχύτητα επικοινωνίας τείνει στο άπειρο) ή δεν υφίσταται αντίστοιχα.



Σχήμα 46 – *Παράδειγμα τοπολογίας ενός συστήματος.*

Τα ECD μπορούν να απεικονιστούν με έναν τετραγωνικό δυσδιάστατο πίνακα, που καλείται *χάρτης στοιχειωδών αποστάσεων διασύνδεσης* (*ECDMap*), με γραμμές και στήλες τα ονόματα των οντοτήτων του συστήματος και στοιχεία τις τιμές των ECD των αντιστοίχων ζευγών που σχηματίζουν. Σύμφωνα με τα παραπάνω, τα στοιχεία της διαγωνίου του ECDMap ισούνται με το μηδέν καθώς η επικοινωνία μίας οντότητας με τον εαυτό της πραγματοποιείται ακαριαία. Είναι φανερό ότι η απουσία ECD για ένα ζεύγος οντοτήτων δεν οδηγεί απαραίτητα

στην μεταξύ τους απομόνωση αφού η επικοινωνία τους μπορεί να εξασφαλιστεί έμμεσα μέσω κάποιων άλλων (οντοτήτων) και μάλιστα με τρόπο όχι μοναδικό. Έτσι, ορίζεται το μέγεθος *απόσταση διασύνδεσης* (*Connection Distance*, *CD*) το οποίο εκφράζεται με το άθροισμα των ECD των ζευγών μέσω των οποίων επιτυγχάνεται η διασύνδεση δύο οντοτήτων. Είναι προφανές ότι το CD ταυτίζεται με το ECD στην περίπτωση όπου οι σχετιζόμενες οντότητες επικοινωνούν άμεσα. Στο παράδειγμα του Σχήματος 46 οι c1 και c4 συνδέονται μέσω των c2 και c3, με αποτέλεσμα μερικές από τις δυνατές τιμές του αντίστοιχου CD είναι οι {25, 20, 15}.

Σημειώνεται ότι, στην περίπτωση όπου η επικοινωνία δύο οντοτήτων μπορεί να λάβει χώρα με περισσότερους από έναν τρόπο, επιλέγεται αυτός που επιτυγχάνει το μικρότερο CD. Έτσι στο παράδειγμα του Σχήματος 46 η ανταλλαγή μηνυμάτων μεταξύ των c1 και c3 πραγματοποιείται απευθείας και όχι μέσω του c2 αφού οι τιμές των αντίστοιχων αποστάσεων διασύνδεσης είναι 10 και 15.

Για την έκφραση της ταχύτητας επικοινωνίας των οντοτήτων σε μονάδες χρόνου είναι απαραίτητος ο προσδιορισμός ενός ακόμη μεγέθους, καλούμενο ως *ρυθμός χρόνου απόστασης* (*Distance Time Rate*, *DTR*), το οποίο ουσιαστικά καθορίζει τις μονάδες χρόνου ανά μονάδα ECD. Αν για παράδειγμα το ECD του ζεύγους (c1, c2) είναι 5 και η τιμή του DTR είναι 2sec τότε ο χρόνος που απαιτείται για την αποστολή ενός μηνύματος από την c1 στην c2 και αντίστροφα, είναι 10 sec.

Για κάθε μοντέλο προσομοίωσης, ο χρήστης καλείται να ορίσει τον χάρτη στοιχειωδών αποστάσεων διασύνδεσης και τον ρυθμό χρόνου απόστασης. Σε αντίθετη περίπτωση, τα ECDMap και ECD παράγονται αυτόματα βάσει των ακόλουθων κανόνων: α) κάθε οντότητα επικοινωνεί απευθείας με εκείνες των οποίων οι αυτοφυείς ιδιότητες είναι ιδιότητες ενδιαφέροντος για την οντότητα αυτή, β) το ECD για κάθε ζεύγος οντοτήτων ισούται με την μονάδα και γ) μία μονάδα του ECD αντιστοιχεί σε μία μονάδα χρόνου προσομοίωσης.

Ανάλογος τρόπο αναπαράστασης της συνδεσμολογίας των οντοτήτων ενός συστήματος χρησιμοποιείται και στην πλατφόρμα OptorSim *[Optorsim, hp]*, *[OptorSim, userg]*.

4.8 Σύγκριση με υπάρχουσες πλατφόρμες

Στις επόμενες παραγράφους πραγματοποιείται μελέτη της επίδρασης του προτεινόμενου εργαλείου σε υπάρχουσες πλατφόρμες. Από την μελέτη αυτή αποδεικνύεται ότι οι επιμέρους μηχανισμοί του DynConPBTool αποτελούν μέσο για την βελτίωση οποιουδήποτε συστήματος προσομοίωσης ανεξάρτητα από το πεδίο εφαρμογής του.

Μεγάλης σπουδαιότητας είναι η ευελιξία που μπορεί να προσφέρει η χρήση του μηχανισμού α) εναλλακτικών συμπεριφορών σε μεθόδους που ελέγχουν την συμπεριφορά του μοντέλου προσομοίωσης, β) όρων ενεργειών στις μεθόδους οι οποίες είναι υπεύθυνες για τον χρονοπρογραμματισμό εκτέλεσης διεργασιών και έγερσης γεγονότων και γ) διαχείρισης πληροφοριών για την καταγραφή δεδομένων και την δημιουργία σημείων ελέγχου κατά την διάρκεια μίας προσομοίωσης.

Στις παραγράφους που ακολουθούν πραγματοποιείται σύντομη αναφορά διάφορων συστημάτων προσομοίωσης και περιγράφεται ο τρόπος με τον οποίο οι επιμέρους μηχανισμοί του προτεινόμενου εργαλείου μπορούν να εφαρμοστούν σε αυτά.

4.8.1 JavaSim

Η JavaSim αποτελεί μία πλατφόρμα προσομοίωσης διακριτών γεγονότων (*Discrete Event Simulation Platform*, *DESP*) βασισμένη στην *προσέγγιση διεργασίας* (*PA*) και υλοποιημένη σε γλώσσα προγραμματισμού Java. *[javasim, hp]*

Θεμελιώδης στοιχεία του μοντέλου της αποτελούν οι *οντότητες* (*entities*) και οι *ιδιότητες* (*attributes*). Οι οντότητες ενσαρκώνουν τα κύρια αντικείμενα του προς προσομοίωση συστήματος ενώ οι ιδιότητες απεικονίζουν την κατάσταση τους σε μία δεδομένη χρονική στιγμή. Η αλληλεπίδραση μεταξύ των οντοτήτων και η μεταβολή της κατάστασης τους πραγματοποιείται μέσω γεγονότων (events).

Οι οντότητες θεωρούνται ως ανεξάρτητες διεργασίες (processes) οι οποίες εκτελούνται ψευδό-παράλληλα (δηλαδή, ανά μονάδα πραγματικού χρόνου εκτελείται το πολύ μία διεργασία, ενώ στην μονάδα χρόνου προσομοίωσης μπορούν να εκτελεστούν περισσότερες από μία) και βρίσκονται σε μία από τις τέσσερις ακόλουθες καταστάσεις: ενεργή (active), αναμονής (suspended), παθητική (passive) και τερματισμού (terminated).

Στην JavaSim υλοποιείται σειρά μεθόδων που επιτρέπουν την ενεργοποίηση και την εκτέλεση διεργασιών σε μία χρονική στιγμή ή με το πέρας ενός χρονικού διαστήματος, με κυριότερες τις ActivateAt(double atTime), ActivateDelay(double delay), ReActivateAt(double atTime) και ReActivateDelay(double delay). Αυτές μπορούν να ενταχθούν στον μηχανισμό όρων-ενεργειών επιτρέποντας τον δυναμικό προσδιορισμό του χρόνου ενεργοποίησης των αντίστοιχων διεργασιών. *[javasim, man]*

Η JavaSim δεν διαθέτει σύστημα καταγραφής δεδομένων εκτός από κάποιες κλάσεις, όπως οι Mean, Variance και TimeVariance, οι οποίες χρησιμοποιούνται για την συλλογή στατιστικών στοιχείων. Έτσι, το προτεινόμενο σύστημα διαχείρισης πληροφοριών αναμφισβήτητα θα μπορούσε να χρησιμοποιηθεί για την αποθήκευση των επιθυμητών δεδομένων κατά την διάρκεια της προσομοίωσης, και την ανάκτηση τους προς την στατιστική τους επεξεργασία.

4.8.2 SimJava

Η SimJava, όπως η JavaSim, είναι άλλη μία πλατφόρμα προσομοίωσης διακριτών γεγονότων, υλοποιημένη σε γλώσσα προγραμματισμού Java σύμφωνα με την προσέγγιση της διεργασίας. Βασικές μονάδες του μοντέλου της είναι οι *οντότητες* (κλάση Sim_entity) οι οποίες συνδέονται μέσω των *θυρών* (κλάση Sim_port) και επικοινωνούν μεταξύ τους με την αποστολή και την λήψη γεγονότων (κλάση Sim_event) σε δεδομένες χρονικές στιγμές. *[simjava, hp]*

Οι οντότητες αναπαρίστανται από αντικείμενα της κλάσης Sim_entity η οποία εσωκλείει όλες τις απαραίτητες λειτουργίες τους για την διεξαγωγή της προσομοίωσης. Ο χρήστης έχει την δυνατότητα να ορίσει ένα νέο τύπο οντότητας κατασκευάζοντας μια κλάση που κληρονομεί από την Sim_entity και υπερκαλύπτει (override) την μέθοδο body() η οποία και καθορίζει την συμπεριφορά της.

Ο χρονοπρογραμματισμός (ή χρονοδρομολόγηση) ενός γεγονότος πραγματοποιείται από μία σειρά μεθόδων (sim_schedule) (μέθοδοι χρονοδρομολόγησης) οι οποίες έχουν σαν κύρια ορίσματα την θύρα μέσα από την οποία αποστέλλεται και το χρόνο μετά το πέρας του οποίου λαμβάνει χώρα η άφιξή του στην οντότητα για την οποία προορίζεται.

Είναι προφανές ότι η εφαρμογή του μηχανισμού εναλλακτικών συμπεριφορών στην μέθοδο body προσδίδει σημαντική ευελιξία στο σύστημα καθώς ο χρήστης έχει την δυνατότητα να υλοποιήσει έμμεσα, μέσω εξωτερικών εκτελεστών, νέους τύπους οντοτήτων και κατεπέκταση

να εφαρμόσει διαφορετικά σενάρια προσομοίωσης με δυναμικό τρόπο. Ταυτόχρονα η προσαρμογή των μεθόδων χρονοδρομολόγησης στον μηχανισμό όρων-ενεργειών επιτρέπει τον δυναμικό ορισμό του χρόνου άφιξης των γεγονότων.

Η SimJava διαθέτει δύο μηχανισμούς καταγραφής πληροφοριών. Ο πρώτος μηχανισμός χρησιμοποιείται για την αποτύπωση (trace) εσωτερικών πληροφοριών όπως οι ενέργειες και τα γεγονότα που έλαβαν χώρα κατά την διάρκεια της προσομοίωσης. Οι πληροφορίες αυτές αποθηκεύονται σε ένα αρχείο και μπορούν να χρησιμοποιηθούν για τον έλεγχο της ορθής συμπεριφοράς του υπό εξέταση συστήματος ώστε ο χρήστης να προβεί στις αναγκαίες διορθώσεις, εφόσον αυτό είναι απαραίτητο.

Ο δεύτερος μηχανισμός είναι υπεύθυνος για την παραγωγή στατιστικών μετρήσεων προς την αξιολόγηση του μοντέλου που μελετάται. Οι μετρήσεις αυτές διακρίνονται σε τρεις κατηγορίες: α) μετρήσεις ρυθμού (rate based measurements), οι οποίες βασίζονται στο πλήθος των εμφανίσεων ενός γεγονότος κατά την διάρκεια της προσομοίωσης, β) μετρήσεις καταστάσεως (state based measurements) οι οποίες απεικονίζουν την κατάσταση των οντοτήτων στο χρόνο, και γ) μετρήσεις μεσοδιαστήματος (interval based measurements). Οι μετρήσεις μεσοδιαστήματος δεν αναφέρονται τόσο στην ίδια την οντότητα αλλά στα γεγονότα που διέρχονται από αυτή, καθώς εκφράζουν το χρονικό διάστημα που κάποια γεγονότα συσχετίζονται με μια οντότητα (για παράδειγμα ο χρόνος αναμονής τους στην οντότητα). Αξίζει να σημειωθεί ότι ο χρήστης έχει την δυνατότητα καταγραφής μεγεθών δικής του επιλογής μέσω της μεθόδου update().

Το σύστημα αποθήκευσης πληροφοριών της SimJava είναι αξιόλογο αλλά σε καμία περίπτωση δεν μπορεί να θεωρηθεί ανταγωνιστικό με το προτεινόμενο αντίστοιχο μηχανισμό καθώς α) η αποθηκευμένη πληροφορία περιορίζεται σε αλφαριθμητικούς τύπους δεδομένων και β) το μέσο αποθήκευσης είναι απλά αρχεία. *[simjava, man]*

4.8.3 SimKit

Η SimKit αποτελεί μία DES πλατφόρμα βασισμένη στην *προσέγγιση χρονοδρομολόγησης γεγονότων* (*Event-Driven Approach, EDA*) και υλοποιημένη σε γλώσσα προγραμματισμού Java. *[simkit, hp]*

Θεμελιώδης στοιχεία του μοντέλου της αποτελούν οι οντότητες και τα γεγονότα. Τα κύρια χαρακτηριστικά και οι λειτουργίες των οντοτήτων κληρονομούνται από την βασική κλάση SimEntityBase ενώ η συμπεριφορά τους καθορίζεται από μεθόδους που λαμβάνουν χώρα με την έγερση γεγονότων (κλάση SimEvent). Η συσχέτιση μεταξύ μεθόδων-γεγονότων επιτυγχάνεται μέσω ενός πρότυπου ονοματολογίας με το οποίο οφείλουν να συμμορφώνονται. Σύμφωνα με το πρότυπο αυτό, η εκτέλεση της μεθόδου do<xxx> συμβαίνει με την έγερση του γεγονότος <xxx>.

Ο χρονοπρογραμματισμός ενός γεγονότος πραγματοποιείται μέσω της μεθόδου waitDelay η οποία έχει σαν ορίσματα το όνομα του και το χρονικό διάστημα μετά το πέρας του οποίου λαμβάνει χώρα. Η προσαρμογή της waitDelay στον μηχανισμό όρων-ενεργειών και η εφαρμογή του μηχανισμού εναλλακτικών συμπεριφορών στις μεθόδους που εκτελούνται με την έγερση των αντίστοιχων γεγονότων μπορούν να οδηγήσουν σε σημαντική βελτίωση της ευελιξίας του SimKit. Με τον τρόπο αυτό είναι εφικτά τόσο ο δυναμικός ορισμός του χρόνου που εγείρεται ένα γεγονός, όσο και η εφαρμογή διαφορετικών υλοποιήσεων, μέσω εξωτερικών εκτελεστών, της αντίστοιχης μεθόδου που εκτελείται.

Η αλληλεπίδραση μεταξύ των οντοτήτων, σε αντίθεση με τις προηγούμενες πλατφόρμες, πραγματοποιείται με την καταχώρηση τους ως ακροατές (SimEventListener και PropertyChangeListener) στην έγερση γεγονότων και στις μεταβολές καταστάσεων. Για κάθε

γεγονός ή μεταβολή της κατάστασης μιας οντότητας εκτελούνται οι αντίστοιχες προκαθορισμένες μέθοδοι (processSimEvent(SimEvent) και propertyChange(PropertyChangeEvent)) των καταχωρημένων ακροατών στο γεγονός ή στην μεταβολή αυτή.

Η SimKit διαθέτει στοιχειώδη μηχανισμό συλλογής στατιστικών στοιχείων βασισμένο στην καταχώρηση ακροατών σε μεταβολές καταστάσεως. Ο μηχανισμός αυτός, σε σύγκριση με το προτεινόμενο σύστημα διαχείρισης πληροφοριών, είναι ανεπαρκής καθώς η καταγραφείσα πληροφορία αφενός περιορίζεται σε αριθμητικούς τύπους δεδομένων (τύποι που κληρονομούν από την βασική κλάση Number), αφετέρου διατηρείται μόνο προσωρινά στην μνήμη. *[simkit, begm]*, *[simkit, dep]*, *[Buss, 02]*

4.8.4 SSJ

Η SSJ είναι μία DESP, υλοποιημένη σε γλώσσα προγραμματισμού Java, που υποστηρίζει την προσομοίωση συστημάτων με χρήση τόσο της προσέγγισης διεργασίας όσο και της προσέγγισης δρομολόγησης γεγονότων. *[ssj, hp]*

Βασικές οντότητες της πλατφόρμας αποτελούν οι *διεργασίες* (*processes*) ή τα *γεγονότα* (*events*) ανάλογα με την επιλογή της PA ή της EDA ως μέθοδο προσομοίωσης αντίστοιχα. Κάθε γεγονός πρέπει να κληρονομεί από την κλάση Event, η οποία παρέχει επίσης τις μεθόδους schedule και reschedule, που είναι υπεύθυνες για τον χρονοπρογραμματισμό του. Οι μέθοδοι αυτές έχουν όρισμα έναν πραγματικό αριθμό ο οποίος εκφράζει τις μονάδες του χρόνου προσομοίωσης μετά το πέρας των οποίων λαμβάνει χώρα το γεγονός. Η συμπεριφορά ενός γεγονότος υλοποιείται από τον χρήστη μέσω της μεθόδου actions() η οποία εκτελείται αυτόματα με την έγερση του.

Ομοίως, μια διεργασία πρέπει να προέρχεται από την βασική κλάση Process. Επειδή αυτή κληρονομεί από την κλάση Event, ο χρονοπρογραμματισμός και η συμπεριφορά μιας διεργασίας ελέγχεται με τρόπο αντίστοιχο αυτού που χρησιμοποιείται στην περίπτωση ενός γεγονότος.

Η ανάγκη συγχρονισμού και αλληλεπίδρασης των διεργασιών, καθώς κάθε μία από αυτές πραγματοποιείται σε ξεχωριστό περιβάλλον εκτέλεσης (thread/ execution context), επιτυγχάνεται μέσω μία σειράς κλάσεων οι οποίες περιγράφονται εν συντομία ακολούθως: α) Resource, τα αντικείμενα της κλάσης αυτής εκφράζουν πόρους πεπερασμένης χωρητικότητας οι οποίοι μπορούν να δεσμευτούν και να απελευθερωθούν από τις διεργασίες, β) Bin, αντιστοιχεί σε μία στοίβα από όμοια κουπόνια (tokens) και μία λίστα διεργασιών οι οποίες αναμένουν για την παραλαβή κουπονιών, στην περίπτωση όπου η στοίβα είναι άδεια και γ) Condition, αποτελεί μία συνθήκη με μία λίστα διεργασιών που περιμένει την επαλήθευση της συνθήκης αυτής.

Η SSJ διαθέτει επιπλέον κάποιες κλάσεις όπως τις StatProbe, Tally και Accumulate, οι οποίες χρησιμοποιούνται για την συλλογή στατιστικών δεδομένων και τον υπολογισμό διαστημάτων εμπιστοσύνης. *[ssj, man], [Pierre, 02]*, *[Pierre, 05]*

Όπως και στις προηγούμενες πλατφόρμες, η εφαρμογή του μηχανισμού όρων-ενεργειών και εναλλακτικών συμπεριφορών στις μεθόδους χρονοπρογραμματισμού (schedule και reschedule) και σε αυτές που ελέγχουν την συμπεριφορά των βασικών οντοτήτων του συστήματος (actions) αντίστοιχα, αυξάνουν σημαντικά την επεκτασιμότητα της εν λόγω πλατφόρμας. Παράλληλα, ο λιτός μηχανισμός συλλογής στατιστικών στοιχείων της SSJ μπορεί να αντικατασταθεί από τον σαφώς πληρέστερο προτεινόμενο μηχανισμό διαχείρισης πληροφοριών.

4.8.5 J-Sim

Η J-Sim, όπως επίσης οι NEST, MaRS, REAL και Ns-2, αποτελεί μία πλατφόρμα προσομοίωσης δικτύων υλοποιημένη σε γλώσσα προγραμματισμού Java. *[J-Sim, hp]*, *[Nest, hp]*, *[MaRS, hp]*, *[REAL, hp]*, *[ns-2, hp]*

Πυρήνας της J-Sim είναι μία DESP βασισμένη τόσο στην PA όσο και στην EDA. Θεμελιώδης μονάδα του μοντέλου της αποτελεί η *οντότητα* (*component*) η οποία μπορεί να απαρτίζεται από σύνολο άλλων οντοτήτων προς τον σχηματισμό πολύπλοκων πληθυσμών. Σε κάθε οντότητα ορίζονται ένα ή περισσότερα σημεία επαφής (καλούμενα ως *θύρες-ports*) μέσω των οποίων συνδέεται και ανταλλάσσει μηνύματα με τις υπόλοιπες οντότητες του συστήματος, σχηματίζοντας έτσι *κανάλια επικοινωνίας* (*channels*). Ένα κανάλι επικοινωνίας μπορεί να χαρακτηριστεί ως «*μονής*» (*unidirectional*) ή «*διπλής κατεύθυνσης*» (*bidirectional*), επιτρέποντας την αποστολή ή την λήψη μηνυμάτων δια μέσω του ενός ή και των δύο άκρων του αντίστοιχα.

Η J-Sim επιπλέον διαθέτει δύο είδη θυρών με ιδιαίτερα χαρακτηριστικά. Το πρώτο είδος ακολουθεί την αρχιτεκτονική πελάτη – εξυπηρετητή (client-server architecture) όπου η οντότητα-πελάτης αποστέλλει αίτηση στην οντότητα-εξυπηρετητή για χρήση των υπηρεσιών της. Στην περίπτωση αυτή, ο πελάτης βρίσκεται σε κατάσταση αναμονής μέχρι την ολοκλήρωση της υπηρεσίας, η οποία επιβεβαιώνεται με την λήψη κατάλληλου μηνύματος. Το δεύτερο είδος χρησιμοποιείται για την αποστολή μηνυμάτων μιας οντότητας στον εαυτό της.

Η αποστολή ενός μηνύματος σε διαφορετική ή στην ίδια οντότητα μπορεί να λάβει χώρα σε μία χρονική στιγμή ή με το πέρας ενός χρονικού διαστήματος (δηλαδή σε απόλυτο ή σχετικό χρόνο), με την βοήθεια των μεθόδων sendAt και send ή forkAt και fork αντίστοιχα. Οι μέθοδοι αυτές παίρνουν σαν ορίσματα τους χρόνους αποστολής του μηνύματος, την θύρα μέσω της οποίας θα αποσταλεί και τις πληροφορίες που θα ενθυλακώσει. Η συμπεριφορά μίας οντότητας καθορίζεται από την μέθοδο process() η κλήση της οποίας πραγματοποιείται αυτόματα με την λήψη μηνύματος σε ξεχωριστό (ARuntime) ή στο ίδιο (SESimulator) περιβάλλον εκτέλεσης με τις άλλες μεθόδους, ανάλογα με την εφαρμοζόμενη τεχνική προσομοίωσης (PA ή EDA αντίστοιχα). *[J-Sim, acc]*, *[J-Sim, cwg]*, *[J-Sim, wwjs]*, *[J-Sim, thesis]*

Είναι προφανές ότι η εφαρμογή του μηχανισμού όρων-ενεργειών και εναλλακτικών συμπεριφορών στις μεθόδους αποστολής μηνυμάτων (send, sendAt, fork και forkAt) και σε αυτές που ελέγχουν την συμπεριφορά των οντοτήτων (process) αντίστοιχα, αυξάνουν σημαντικά την επεκτασιμότητα της εν λόγω πλατφόρμας. Επιπλέον, η J-Sim δεν διαθέτει κάποιο σύστημα καταγραφής δεδομένων με αποτέλεσμα ο προτεινόμενος μηχανισμός διαχείρισης πληροφοριών να αποτελεί αναμφισβήτητα μία αξιόπιστη λύση.

4.8.6 Ptolemy II – DE domain

Η Ptolemy II (*[PtolemyII, hp]*) χρησιμοποιείται για την προσομοίωση υβριδικών συστημάτων. Υβριδικό χαρακτηρίζεται το σύστημα που προσομοιώνεται με συνδυασμό συνεχών και διακριτών μεθόδων μοντελοποίησης. Μερικές από τις μεθόδους αυτές, όπως αναφέρονται στα εγχειρίδια της εν λόγω πλατφόρμας, είναι οι: Discrete-Event (DE), Continuous-Time (CT), Synchronous Dataflow (SDF), Finite State Machines (FSMs), Giotto, Communicating Sequential Processes (CSP), Distributed Discrete Event (DDE), Process Networks (PN) και Parameterized Synchronous Dataflow (PSDF). Η υλοποίηση της DE από την Ptolemy II ουσιαστικά αποτελεί μία DESP η αρχιτεκτονική της οποίας περιγράφεται ακολούθως.

Βασικές μονάδες της είναι οι *δράστες* (*Actors*) οι οποίοι ενσαρκώνουν τις κύριες οντότητες του υπό προσομοίωση συστήματος. Συγκεκριμένα ορίζονται δύο τύποι δραστών, ο AtomicActor και ο CompositeActor. Ο πρώτος εκφράζει μία απλή οντότητα, ενώ ο δεύτερος αντιπροσωπεύει ένα σύνολο οντοτήτων, επιτρέποντας με τον τρόπο αυτό την δημιουργία πολύπλοκων πληθυσμών.

Σε κάθε δράστη ορίζονται ένα ή περισσότερα σημεία επαφής (καλούμενα ως *θύρες-ports*) μέσω των οποίων συνδέεται και ανταλλάσσει μηνύματα με τους υπόλοιπους δράστες του συστήματος. Η αποστολή των μηνυμάτων μπορεί να λάβει χώρα σε σχετικό ή απόλυτο χρόνο με την βοήθεια των μεθόδων fireAtRelativeTime και fireAt αντίστοιχα.

Η συμπεριφορά ενός δράστη ελέγχεται από οκτώ μεθόδους (preintialize, initialize, prefire, fire, postfire, stopFire, wrapup και terminate) οι οποίες υλοποιούνται από τον χρήστη, ενώ η κλήση τους λαμβάνει χώρα σύμφωνα με ένα προκαθορισμένο σύνολο κανόνων. Για παράδειγμα η preinitialize καλείται μόνο μία φορά, η prefire προηγείται της fire, ενώ η postfire έπεται. Η κλήση των τριών τελευταίων μεθόδων πραγματοποιείται με την λήψη μηνυμάτων από τον αντίστοιχο δράστη.

Ομοίως με τις προηγούμενες πλατφόρμες, η εφαρμογή του μηχανισμού όρων-ενεργειών και εναλλακτικών συμπεριφορών στις μεθόδους αποστολής μηνυμάτων (fireAtRelativeTime, fireAt) και σε αυτές που καθορίζουν την συμπεριφορά της οντότητας (prefire, fire, postfire κ.τ.λ.π) αντίστοιχα, προσδίδει επιπρόσθετη ευελιξία στην εν λόγω πλατφόρμα.

Η Ptolemy II παρόλο που διαθέτει αυτόνομη εφαρμογή (Ptplot) για την δυσδιάστατη απεικόνιση αριθμητικών δεδομένων δεν παρέχει κάποιο μηχανισμό για την καταγραφή τους. Έτσι, ο προτεινόμενος μηχανισμός διαχείρισης πληροφοριών αποτελεί αναμφισβήτητα μία αξιόπιστη λύση.

Αξίζει να σημειωθεί ότι ο σχηματισμός των μοντέλων προσομοίωσης δύναται να πραγματοποιηθεί γραφικά, με χρήση του εργαλείου Vergil. Οι απαραίτητες πληροφορίες για την κατασκευή του μοντέλου, όπως η συνδεσμολογία και οι κλάσεις υλοποίησης των δραστών, αποθηκεύονται σε ένα έγγραφο XML το οποίο υπακούει σε ένα προκαθορισμένο σχήμα, καλούμενο ως MoML (Modeling Markup Language). Έτσι, ο χρήστης έχει την δυνατότητα καθορισμού του μοντέλου και με απευθείας τροποποίηση του αντίστοιχου εγγράφου XML. *[PtolemyII, intro]*, *[PtolemyII, pd]*, *[PtolemyII, sa]*

4.8.7 Swarm, RePast

Οι Swarm (*[Swarm, hp]*) και RePast (*[Repast, hp]*) αποτελούν πλατφόρμες για την διεξαγωγή προσομοιώσεων που στηρίζονται στην αλληλεπίδραση πρακτόρων (*Agent-Based Simulation Platforms*, *ABSP*) και είναι υλοποιημένες σε γλώσσα προγραμματισμού Java. Στις παραγράφους που ακολουθούν περιγράφεται μόνο η πλατφόρμα RePast καθώς τα δύο αυτά συστήματα παρουσιάζουν σημαντικές ομοιότητες, τουλάχιστον όσον αναφορά στα βασικά τους χαρακτηριστικά. Αυτό οφείλεται στο γεγονός ότι ο αρχικός στόχος της RePast ήταν η απλοποίηση της πλατφόρμας Swarm, παρόλο που στη συνέχεια αναπτύχθηκε σαν ανεξάρτητη εφαρμογή ενσωματώνοντας νέα στοιχεία και λειτουργίες. Περισσότερες πληροφορίες για την Swarm μπορούν να βρεθούν στις πηγές *[Swarm, tue]*, *[Swarm, ug]*.

Στην RePast, η προσομοίωση μπορεί να χαρακτηριστεί ως ένα σύνολο πρακτόρων οποιουδήποτε τύπου οι οποίοι αρχικοποιούνται και ελέγχονται από ένα μοντέλο (model) βάσει ενός χρονοπρογράμματος (schedule).

Θεμελιώδης κλάσεις της είναι η Agent και η SimpleModel. Η πρώτη ενσαρκώνει τους *πράκτορες* του συστήματος ενώ η δεύτερη είναι υπεύθυνη για την δημιουργία και την

εφαρμογή του μοντέλου προσομοίωσης. Χαρακτηριστικές μέθοδοι της SimpleModel είναι οι setup, buildModel, preStep, step και postStep οι οποίες υλοποιούνται από τον χρήστη. Συγκεκριμένα, στην setup αρχικοποιούνται οι μεταβλητές του μοντέλου ενώ η buildModel είναι υπεύθυνη για την δημιουργία των πρακτόρων και την προσθήκη τους στην λίστα όπου πρέπει φιλοξενούνται (η λίστα είναι ένα αντικείμενο τύπου ArrayList το οποίο περιέχεται στο πεδίο agentList του SimpleModel). Οι μέθοδοι preStep, step και postStep εκτελούνται αυτόματα και διαδοχικά, με την σειρά που αναφέρονται, σε κάθε χρονικό βήμα (time step) και ορίζουν την συμπεριφορά των πρακτόρων.

Ο σχηματισμός του μοντέλου μέσω της κλάσης SimpleModel έχει σαν σκοπό την απλοποίηση της χρονοδρομολόγησης μεθόδων, δηλαδή την κλήση μεθόδων σε μία δεδομένη χρονική στιγμή. Ο χρήστης εναλλακτικά, μπορεί να κατασκευάσει αντίστοιχη κλάση η οποία οφείλει να κληρονομεί από την SimModelImpl (αυτή υλοποιεί τις περισσότερες μεθόδους της διεπιφάνειας SimModel). Μία τέτοια κλάση θα πρέπει πρωτίστως να υλοποιεί μεθόδους υπεύθυνες για α) την αρχικοποίηση του υπό προσομοίωση συστήματος, και β) την κατασκευή των πρακτόρων και τον καθορισμό της συμπεριφοράς τους μέσω του χρονοπρογραμματισμού των κατάλληλων μεθόδων τους.

Η χρονοδρομολόγηση μίας μεθόδου ενός αντικειμένου, καλούμενη και ως μέθοδος δρομολόγησης (scheduled method), πραγματοποιείται με την βοήθεια της κλάσης Schedule η οποία παρέχει σειρά μεθόδων (scheduleActionAt και scheduleActionAtInterval) για τον σκοπό αυτό. Οι μέθοδοι αυτές, βάσει των ορισμάτων που δέχονται, μπορούν να διαχωριστούν σε δύο κατηγορίες. Στην πρώτη κατηγορία η κλήση της μεθόδου δρομολόγησης λαμβάνει χώρα με τρόπο δυναμικό καθώς στα ορίσματα της αντίστοιχης μεθόδου της (κλάσης) Schedule (που εφαρμόζεται), περιέχεται το όνομα της μεθόδου δρομολόγησης και το αντικείμενο πάνω στο οποίο θα κληθεί. Στη δεύτερη κατηγορία, η κλήση της μεθόδου δρομολόγησης λαμβάνει χώρα σαν μέρος της υλοποίησης της μεθόδου execute ενός αντικειμένου τύπου BasicAction, το οποίο αποτελεί και όρισμα της αντίστοιχης μεθόδου της Schedule.

Σύμφωνα με τα παραπάνω ο μηχανισμός εναλλακτικών συμπεριφορών και όρων-ενεργειών μπορεί να εφαρμοστεί στις μεθόδους που ελέγχουν την συμπεριφορά των πρακτόρων (όπως για παράδειγμα είναι οι preStep, step, postStep και οι μέθοδοι δρομολόγησης) και στις μεθόδους της Schedule αντίστοιχα.
Η RePast διαθέτει στοιχειώδη μηχανισμό καταγραφής δεδομένων με την βοήθεια της κλάσης DataRecorder. Σύμφωνα με τον μηχανισμό αυτό, ο χρήστης ορίζει «πηγές δεδομένων» (data sources) οι οποίες δεν είναι τίποτα άλλο από κλάσεις οι οποίες, μέσω μίας μεθόδου (είτε προκαθορισμένη μέσω διεπιφάνειας, DataSource ή NumericDataSource, είτε ορισμένη δυναμικά κατά την δημιουργία της «πηγής») επιστρέφουν την πληροφορία που επιθυμείται να καταγραφεί. Ο μηχανισμός αυτός δεν μπορεί να θεωρηθεί ανταγωνιστικός με το προτεινόμενο σύστημα διαχείρισης πληροφοριών καθώς α) η αποθηκευμένη πληροφορία δεν επεκτείνεται σε επίπεδο αντικειμένων αλλά περιορίζεται σε αλφαριθμητικούς τύπους δεδομένων και β) το μέσο αποθήκευσης είναι απλά αρχεία. *[Repast, Collier]*, *[Repast, man]*

4.8.8 Mason

Η Mason (*[Mason, hp]*) είναι μια ακόμη ABSP, υλοποιημένη σε γλώσσα προγραμματισμού Java. Η βασικές λειτουργίες και η αρχιτεκτονική της εν λόγω πλατφόρμας παρουσιάζουν ομοιότητες με την RePast. Για παράδειγμα, ο χρονοπρογραμματισμός των μεθόδων ελέγχεται από την κλάση Schedule η οποία είναι υπεύθυνη για την κλήση προκαθορισμένης μεθόδου (step) που υλοποιείται από όλες τις κλάσεις των πρακτόρων (διεπιφάνεια Steppable). Για τον λόγο αυτό θα αναφερθούμε μόνο στα σημεία που οι δύο πλατφόρμες διαφέρουν, όπου οι προτεινόμενοι μηχανισμοί μπορούν να εφαρμοστούν.

Η πιο αξιοσημείωτη διαφορά μεταξύ των RePast και Mason είναι ότι η δεύτερη παρέχει την δυνατότητα δημιουργίας σημείων ελέγχου με αποτέλεσμα να καθιστά εφικτή την τμηματική διεξαγωγή των προσομοιώσεων. Η καταγραφή των απαραίτητων πληροφοριών πραγματοποιείται με χρήση της βασικής μεθόδου απεικόνισης αντικειμένων σε δυαδική μορφή (Serialization), που υποστηρίζεται από τον πυρήνα της γλώσσας προγραμματισμού Java. Η μέθοδος αυτή πέρα από ότι το πρότυπο αποθήκευσης των δεδομένων δεν επιτρέπει την ανάγνωση ή την τροποποίηση τους από κοινούς χρήστες, που στην συγκεκριμένη περίπτωση μπορεί να μην είναι τόσο σημαντικό, εφαρμόζεται επιπλέον σε αντικείμενα που πληρούν συγκεκριμένες προδιαγραφές. Το προτεινόμενο σύστημα διαχείρισης πληροφοριών θα μπορούσε να αποτελέσει μια εναλλακτική λύση καθώς δεν θέτει περιορισμούς στα καταγραφέντα αντικείμενα. *[Luke, 05]*

4.8.9 SimGrid

Η SimGrid (*[SimGrid, hp]*) αποτελεί ένα εργαλείο (simulation toolkit) το οποίο παρέχει ένα σύνολο βασικών λειτουργιών που επιτρέπουν την δημιουργία πλατφορμών προσομοίωσης (*Grid Computing Simulation Platforms*, *GCSP*) με σκοπό την μελέτη αλγόριθμων που εφαρμόζονται τόσο σε υπολογιστικά πλέγματα όσο και σε πλέγματα δεδομένων. Το εργαλείο αυτό είναι υλοποιημένο σε γλώσσα προγραμματισμού C και έχει σαν κύριο χαρακτηριστικό ότι επιτρέπει την εκτέλεση προσομοιώσεων βάσει δεδομένων που καταγράφτηκαν από την λειτουργία πραγματικών συστημάτων (trace-driven simulation).

Θεμελιώδης οντότητες του μοντέλου της είναι ο *πράκτορας* (*agent*), η *τοποθεσία* (*location*), η *εργασία* (*task*), το *μονοπάτι* (*path*) και το *κανάλι*. Ο πράκτορας είναι υπεύθυνος για την λήψη των αποφάσεων ανάθεσης και εκτέλεσης εργασιών. Η τοποθεσία χαρακτηρίζεται από τον πόρο (resource) που διαθέτει (για την διεκπεραίωση των εργασιών) και αποτελεί την θέση (βάσει της τοπολογίας του υπό προσομοίωση συστήματος) όπου ένας πράκτορας φιλοξενείται. Η εργασία αντιπροσωπεύει μία δραστηριότητα η οποία περιλαμβάνει υπολογισμούς και μεταφορά δεδομένων. Το μονοπάτι αναπαριστά ένα σύνολο συνδέσμων μεταξύ των τοποθεσιών, ενώ τα κανάλια χρησιμοποιούνται ως θύρες επικοινωνίας μεταξύ των πρακτόρων. Ο πόρος στην SimGrid παρουσιάζεται με την μορφή υπολογιστικού (CPU) η δικτυακού (link) πόρου. Χαρακτηριστικά του πρώτου αποτελούν η υπολογιστική ταχύτητα (speed) (συγκριτικά με μία τιμή αναφοράς) και η διαθεσιμότητα (από 0 έως 100%), ενώ χαρακτηριστικά του δεύτερου είναι η καθυστέρηση πρόσβασης (latency) και το εύρος ζώνης (bandwidth). Λαμβάνοντας υπόψη τα προαναφερθέντα μεγέθη, ο υπολογισμός του χρόνου εκτέλεσης μιας εργασίας πραγματοποιείται από την σχέση $\frac{computational\ cost}{speed} + latency + \frac{data\ size}{bandwidth}$, όπου «computational cost», το κόστος εκτέλεσης της εργασίας και «data size» το μέγεθος των δεδομένων που θα μεταφερθούν.

Αξίζει επίσης να σημειωθεί ότι η SimGrid χρησιμοποιεί τρία μοντέλα για την κατανομή των πόρων μεταξύ των εργασιών: α) το FIFO (First In First Out) όπου οι εργασίες εκτελούνται με την σειρά που προγραμματίστηκαν, β) το FRFO (First Ready First Out) βάσει του οποίου λαμβάνει χώρα πρώτα η εκτέλεση των «έτοιμων» (ready) εργασιών και γ) το καταμερισμένο (shared) μοντέλο όπου όλες οι «έτοιμες» εργασίες «εξυπηρετούνται» από τον πόρο ταυτόχρονα. Στην περίπτωση αυτή, ο χρήστης έχει την δυνατότητα ορισμού του τύπου καταμερισμού που θα χρησιμοποιηθεί, ο δίκαιος (fair) ή ο αναλογικός (proportional). Ο τελευταίος τύπος λαμβάνει υπόψη τις προτεραιότητες των εργασιών και υλοποιήθηκε για την προσομοίωση της κεντρικής μονάδας επεξεργασίας που αναφέρεται στην παράγραφο 2.3.2.1. *[Legrand, 03]*

Όπως και στα προηγούμενα συστήματα, η SimGrid υλοποιεί σύνολο μεθόδων όπου μπορεί να εφαρμοστεί ο μηχανισμός εναλλακτικών συμπεριφορών και όρων-ενεργειών.

Χαρακτηριστικό παράδειγμα αποτελεί η μέθοδος gras_timer_repeat(double interval, void_f_void_t action) η οποία εγείρει την εκτέλεση της μεθόδου action κάθε interval sec.

Το εν λόγω εργαλείο δεν διαθέτει κάποιο μηχανισμό αποθήκευσης πληροφοριών εκτός από την δυνατότητα ανίχνευσης χαρακτηριστικών μεγεθών (όπως είναι ο χρόνος ολοκλήρωσης μιας εργασίας και οι χρησιμοποιούμενοι πόροι) σε συνδυασμό με ένα είδος μάλλον «παραπλανητικής» (όπως αναφέρεται στα εγχειρίδια) γραφικής τους απεικόνισης με την βοήθεια της εφαρμογής Paje. Έτσι ο προτεινόμενος μηχανισμός διαχείρισης πληροφοριών αποτελεί αναμφισβήτητα μία αξιόπιστη λύση.

Αξίζει να σημειωθεί ότι η SimGrid χρησιμοποιεί την XML ως μέσο περιγραφής της τοπολογίας και των χαρακτηριστικών των οντοτήτων του υπό προσομοίωση συστήματος. Για παράδειγμα ο καθορισμός των υπολογιστικών και των δικτυακών πόρων μπορεί να πραγματοποιηθεί από ακολουθίες XML στοιχείων της μορφής <cpu name="Cpu A" power="100.00" availability_file="trace_A.txt" /> και <network_link name="LinkA" bandwidth="10.0" latency="0.2"/> αντίστοιχα. Οι τιμές των ιδιοτήτων power και availability_file προσδιορίζουν την ταχύτητα και το όνομα του αρχείου στο οποίο περιέχεται το ίχνος (trace) της διαθεσιμότητας ενός υπολογιστικού πόρου, ενώ οι τιμές των bandwidth και latency καθορίζουν την καθυστέρηση πρόσβασης και το εύρος ζώνης ενός δικτυακού πόρου αντίστοιχα. Επιπλέον, υπάρχει η δυνατότητα ορισμού αλφαριθμητικών παραμέτρων οι οποίες μπορούν να χρησιμοποιηθούν από τις συναρτήσεις στις οποίες αναφέρονται. *[SimGrid, faqs]*

Στην SimGrid, ο καθορισμός των στοιχείων του υπό προσομοίωση συστήματος μέσω της XML, όπως περιγράφτηκε στην προηγούμενη παράγραφο, εμπίπτει στον σαφώς γενικότερο τρόπο περιγραφής ενός συστήματος που εφαρμόζεται στην προτεινόμενη πλατφόρμα. Αυτό γίνεται φανερό με την αντιστοίχηση των πρακτόρων στις οντότητες (components) και των χαρακτηριστικών των πόρων (π.χ. speed, latency, bandwidth) στις ιδιότητες (properties).

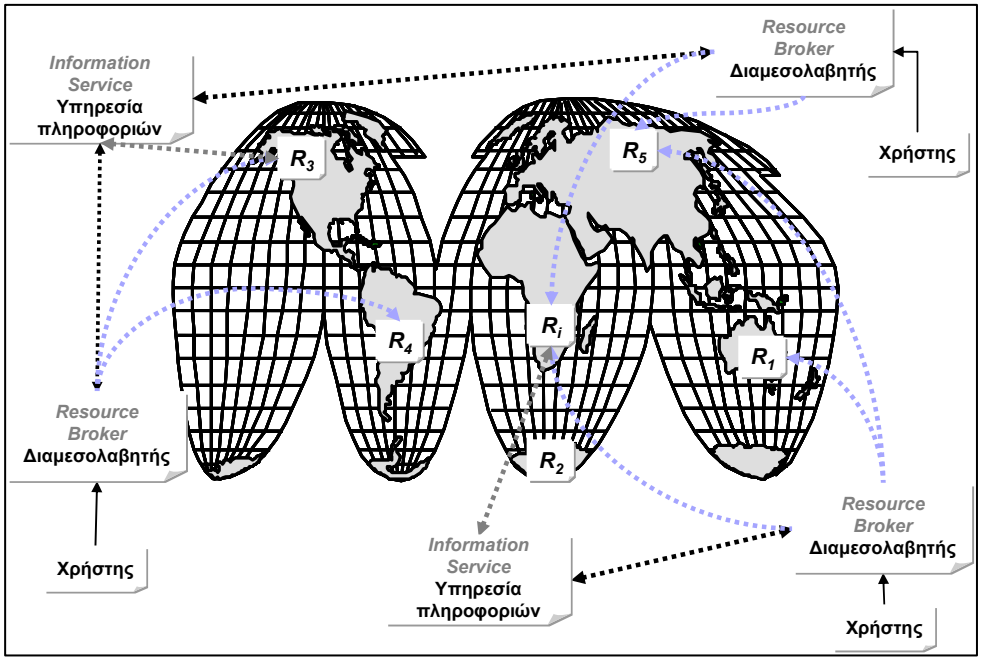
4.8.10 GridSim

Η GridSim ([GridSim, hp]) αποτελεί μία πλατφόρμα (GCSP), υλοποιημένη σε γλώσσα προγραμματισμού Java, που έχει σαν στόχο την μοντελοποίηση και την προσομοίωση πλεγμάτων.

Πυρήνας της εν λόγω πλατφόρμας είναι η SimJava με αποτέλεσμα να κληρονομεί τα βασικά της χαρακτηριστικά τόσο σε επίπεδο αρχιτεκτονικής όσο και σε επίπεδο υλοποίησης. Για παράδειγμα, θεμελιώδεις μονάδες του μοντέλου της GridSim είναι οι *οντότητες* οι οποίες συνδέονται μέσω των *θυρών*, επικοινωνούν μεταξύ τους με την αποστολή (μέσω των μεθόδων sim_schedule) και την λήψη γεγονότων σε συγκεκριμένα χρονικά σημεία, ενώ η συμπεριφορά τους καθορίζεται από την υλοποίηση της κατάλληλης μεθόδου (body). Για τον λόγο αυτό, η εφαρμογή των μηχανισμών εναλλακτικών συμπεριφορών και όρων-ενεργειών στην GridSim πραγματοποιείται με ίδιο τρόπο όπως και στην SimJava, ενώ ο προτεινόμενος μηχανισμός διαχείρισης πληροφοριών αποτελεί αναμφισβήτητα μία αξιόπιστη λύση για την καταγραφή δεδομένων κατά την διάρκεια της προσομοίωσης.

Πριν αναφερθούμε στα ιδιαίτερα χαρακτηριστικά της εν λόγω πλατφόρμας σημειώνουμε ότι οι βασικές λειτουργίες και η αρχιτεκτονική ενός υπολογιστικού πλέγματος, όπως χαρακτηριστικά απεικονίζεται στο Σχήμα 47, μπορούν να περιγραφούν συνοπτικά ως ακολούθως: κάθε χρήστης αλληλεπιδρά με μία μονάδα, την Resource Broker (RB), η οποία είναι υπεύθυνη για την αναζήτηση και την επιλογή των κατάλληλων πόρων (Resource, R) προς την εκτέλεση των εργασιών **του**. Η επιλογή των πόρων πραγματοποιείται σύμφωνα με πληροφορίες που διατίθενται για αυτούς, από μονάδες καλούμενες ως Grid Information Service (GIS), καθώς επίσης από τις προδιαγραφές που ορίζει ο χρήστης. Μια RB είναι

υπεύθυνη επιπλέον για τον έλεγχο προόδου ολοκλήρωσης των εργασιών, την προσαρμογή του προγράμματος εκτέλεσης τους σε απρόβλεπτες μεταβολές της κατάστασης του πλέγματος (για παράδειγμα στην περίπτωση αστοχίας ενός η περισσοτέρων πόρων), καθώς και για την επιστροφή των αποτελεσμάτων τους στον χρήστη. *[Buyya, 02]*



Σχήμα 47 – Αρχιτεκτονική υπολογιστικού πλέγματος.

Η ανάγκη προσαρμογής του μοντέλου της GridSim στην βασική αρχιτεκτονική ενός υπολογιστικού πλέγματος, οδήγησε στον διαχωρισμό των οντοτήτων της σε επιμέρους τύπους: α) ο *Χρήστης* (*User*, κλάση GridUser), ο οποίος αντιπροσωπεύει κάποιον χρήστη του πλέγματος. Η διαφοροποίηση μεταξύ των χρηστών έγκειται σε χαρακτηριστικά όπως ο τύπος (π.χ. χρόνος εκτέλεσης), ο ρυθμός παραγωγής και η εφαρμοζόμενη στρατηγική προγραμματισμού των εργασιών που επιθυμείται να διεκπεραιωθούν (π.χ. ελαχιστοποίηση του χρόνου ή του υπολογιστικού κόστους), β) ο *Διαμεσολαβητής* (*Broker*, κλάση Broker), ο οποίος είναι υπεύθυνος για την αναζήτηση εκείνων των πόρων που εξασφαλίζουν όσο το δυνατόν περισσότερο την ολοκλήρωση των εργασιών σύμφωνα με την πολιτική και τις προδιαγραφές που έχει θέσει ο χρήστης. Οι αποφάσεις του διαμεσολαβητή λαμβάνουν χώρα βάσει των πληροφοριών που τροφοδοτείται από την Υπηρεσία Πληροφοριών, ενώ μετά την εύρεση των κατάλληλων πόρων οι εργασίες αποστέλλονται σε αυτούς, γ) ο *Πόρος* (*resource*, κλάση GridResource), που αναπαριστά ένα πόρο του πλέγματος. Οι διαφορές μεταξύ των πόρων εστιάζονται σε χαρακτηριστικά όπως το κόστος επεξεργασίας (cost of processing), η ταχύτητα επεξεργασίας (speed of processing) και η εσωτερική πολιτική προγραμματισμού εκτέλεσης των εργασιών (π.χ. time-shared ή space-shared), και δ) η *Υπηρεσία Πληροφοριών* (*Information Service*, κλάση GridInformationService), η οποία διατηρεί πληροφορίες για τους διαθέσιμους πόρους του πλέγματος. Κάθε διαμεσολαβητής έρχεται σε επαφή με μια τέτοια οντότητα προκειμένου να αναζητήσει και να λάβει σχετικές πληροφορίες. *[Buyya, 02]*

Ιδιαίτερη σημασία στο σημείο αυτό έχει ο έλεγχος της δυνατότητας προσαρμογής του προτεινόμενου συστήματος στο μοντέλο της GridSim, και κατεπέκταση σε αυτό ενός υπολογιστικού πλέγματος. Έτσι, στον Πίνακα 1 πραγματοποιείται η συσχέτιση των βασικών μονάδων της GridSim και των λειτουργιών τους, με αυτά της προτεινόμενης πλατφόρμας, επιβεβαιώνοντας με τον τρόπο αυτό την ικανότητα της τελευταίας να αποτελεί πραγματικά βασικό εργαλείο για την ανάπτυξη συστημάτων προσομοίωσης υπολογιστικών πλεγμάτων.

Πίνακας 1 – Συσχέτιση οντοτήτων GridSim με αυτές του προτεινόμενου εργαλείου

GridSim	*Προτεινόμενο σύστημα*
Χρήστης	• Οντότητα με ιδιότητες ενδιαφέροντος τις ιδιότητες που εκφράζουν τους πόρους του πλέγματος. • Δεν διαθέτει λίστα καταστάσεως καθώς οι σχετικές πληροφορίες για την κατάσταση του συστήματος διατηρούνται στις οντότητες που αντιπροσωπεύουν τις Υπηρεσίες Πληροφοριών. • Οι παραγόμενες εργασίες καθώς και οι προδιαγραφές για την εκτέλεση τους, προωθούνται στην οντότητα που εκφράζει τον Διαμεσολαβητή.
Διαμεσολαβητής	• Δεν διαθέτει πρωτογενή αλλά αντίγραφο λίστας καταστάσεως μιας Υπηρεσίας Πληροφοριών, από όπου ενημερώνεται για την κατάσταση του συστήματος. • Επιλέγει τους κατάλληλους πόρους, για την εκτέλεση των εργασιών, σύμφωνα με την κατάσταση του συστήματος και τις προδιαγραφές του χρήστη. • Αποστέλλει τις εργασίες σε εκείνες τις οντότητες των οποίων οι αυτοφυείς ιδιότητες αποτελούν τους επιλεγμένους πόρους για την εκτέλεση των εργασιών αυτών. • Επιστρέφει τα αποτελέσματα των εργασιών στους αντίστοιχους Χρήστες.
Υπηρεσία Πληροφοριών	• Οντότητα με ιδιότητες ενδιαφέροντος τουλάχιστον αυτές των οντοτήτων οι οποίες εκφράζουν χρήστες και προωθούν τις εργασίες τους σε εκείνους τους Διαμεσολαβητές που διατηρούν αντίγραφο λίστας καταστάσεως της οντότητας αυτής. Αυτό είναι απαραίτητο καθώς μία Υπηρεσία πληροφοριών θα πρέπει να ενημερώνεται για τις ιδιότητες οι οποίες εκφράζουν τους πόρους που ενδιαφέρουν τους παραπάνω χρήστες. • Διατηρεί πρωτογενής λίστα καταστάσεως. • Εφαρμόζει την πολιτική πληροφόρησης.
Πόρος	• Οι πόροι εκφράζονται με τις αυτοφυείς ιδιότητες κάποιων οντοτήτων. Οι οντότητες αυτές καθορίζουν την εσωτερική πολιτική εκτέλεσης των εργασιών που τους αποστέλλονται από τους Διαμεσολαβητές.

Τέλος, στον Πίνακα 2 αναφέρονται συνοπτικά, σε επίπεδο υλοποίησης, οι μέθοδοι καθορισμού της συμπεριφοράς των οντοτήτων, οι μέθοδοι χρονοδρομολόγησης γεγονότων και διεργασιών, καθώς επίσης και τα συστήματα διαχείρισης πληροφοριών, για τις υπό εξέταση πλατφόρμες.

Πίνακας 2 – Χαρακτηριστικά πλατφορμών

Πλατφόρμα	*Συμπεριφορά Συστ.*	*Χρονοδρομολόγηση*	*Διαχείριση Πληροφοριών*
JavaSim	*run()*	*ActivateAt(…), ActivateDelay(…) …*	*X*
SimJava	*body()*	*sim_schedule(…)*	➢ *Αποτύπωση εσωτερικών πληροφοριών* ➢ *Στατιστικές μετρήσεις (state based και interval based measurements)* ➢ *Μεγέθη κατά βούληση (μόνο αλφαριθμητικά)* ➢ *Μέσο αποθήκευσης: αρχεία*
SimKit	*doXXX(), όπου XXX το γεγονός*	*waitDelay(…)*	➢ *Καταγραφή με την καταχώρηση ακροατών (listeners) σε μεταβολές τιμών πεδίων (μόνο αριθμητικά)* ➢ *Μέσο αποθήκευσης: μνήμη*
SSJ	*actions()*	*schedule(…)* *reschedule(…)*	➢ *Στατιστικά στοιχεία*
J-Sim	*process()*	*send(…), sendAt(…)* *fork(…), forkAt(…)*	*X*
Ptolemy II-DE	*prefire(), fire(), postfire(), …*	*fireAt(), fireAtRelativeTime()*	*X*
Swarm, RePast	*preStep(), step(), postStep()*	*scheduleActionAt(…), scheduleActionAtInterval(…) …*	➢ *Ορίζει πηγές δεδομένων (DataSource, NumericDataSource)* ➢ *Μόνο αλφαριθμητικά* ➢ *Μέσο αποθήκευσης: αρχεία*
Mason	*step()*	*scheduleOnce(…), scheduleRepeating(…) …*	➢ *Δημιουργία σημείων ελέγχου*
SimGrid	-	*gras_timer_repeat(...) ....*	➢ *Καταγραφή χαρακτηριστικών μεγεθών*
GridSim	*Βασισμένη στην SimJava*		

4.9 Συμπεράσματα

Τα πλεονεκτήματα του προτεινόμενου εργαλείου προσομοίωσης μπορούν να συνοψιστούν ως ακολούθως:

➢ Το DynConPBTool αποτελείται από ένα σύνολο «γενικών» μηχανισμών, όπως οι «εναλλακτικών συμπεριφορών», «όρων-ενεργειών και ανίχνευσης διακεκριμένων γεγονότων», «διαχείρισης πληροφοριών» και «κανόνων κατάταξης» οι οποίοι μπορούν να εφαρμοστούν σε οποιοδήποτε σύστημα προσομοίωσης.

➢ Κατά τον σχεδιασμό του DynConPBTool, ελήφθησαν υπόψη τα μοντέλα λειτουργίας των βασικών κατηγοριών των αλγόριθμων εξισορρόπησης φορτίου με αποτέλεσμα την δημιουργία μιας «φιλικής» διεπιφάνειας για την υλοποίηση πλατφορμών προσομοίωσης υπολογιστικών συστημάτων. Χαρακτηριστικό παράδειγμα αποτελεί το μοντέλο επικοινωνίας μεταξύ των οντοτήτων και ο μηχανισμός μηνυμάτων ευρύτερα.

➢ Το μοντέλο του DynConPBTool ικανοποιεί το ευρύτερο μοντέλο λειτουργίας των πλατφορμών προσομοίωσης, σύμφωνα με το οποίο οι βασικές μονάδες του συστήματος (μεταφρασμένες σαν οντότητες, πράκτορες ή οτιδήποτε άλλο) αλληλεπιδρούν μεταξύ τους με σκοπό την μεταβολή της καταστάσεως τους.

➢ Μέσω του μηχανισμού εναλλακτικών συμπεριφορών, ο χρήστης έχει την δυνατότητα να μεταβάλλει εξωτερικά την συμπεριφορά του συστήματος που προσομοιώνει και να δημιουργεί εναλλακτικά σενάρια λειτουργίας του χωρίς να είναι απαραίτητο να μεταγλωττίσει ξανά τον πηγαίο κώδικα.

➢ Ο μηχανισμός διαχείρισης πληροφοριών δεν περιορίζεται απλώς στην καταγραφή κάποιων χαρακτηριστικών μεγεθών, όπως συμβαίνει στα υπόλοιπα συστήματα, αλλά παρέχει την δυνατότητα αποθήκευσης οποιασδήποτε πληροφορίας (οποιαδήποτε αντικειμένο, σε επίπεδο υλοποίησης), χωρίς σχεδόν κανένα περιορισμό, εξασφαλίζοντας ταυτόχρονα απόλυτη διαφάνεια ως προς τον χρήστη.

➢ Οι μηχανισμοί όρων-ενεργειών και ανίχνευσης γεγονότων παρέχουν ένα πραγματικά χρήσιμο σύνολο λειτουργιών, καθώς είναι δυνατή α) η συσχέτιση της συμπεριφοράς (ενέργειες) των οντοτήτων και κατεπέκταση του συστήματος μέσα από ένα σύνολο συνθηκών (όροι) και β) η δυναμική ανίχνευση γεγονότων και ο καθορισμός της συμπεριφοράς των οντοτήτων σύμφωνα με τα γεγονότα αυτά. Η ευελιξία του μηχανισμού αυτού πολλαπλασιάζεται με την εφαρμογή του μηχανισμού εναλλακτικών συμπεριφορών.

➢ Η ανεξαρτησία των επιμέρους μηχανισμών, του εν λόγω εργαλείου, παρέχει την δυνατότητα άμεσης εφαρμογής τους σε υπάρχουσες πλατφόρμες. Χαρακτηριστικά παραδείγματα αποτελούν η χρήση του μηχανισμού α) εναλλακτικών συμπεριφορών σε μεθόδους που ελέγχουν την συμπεριφορά του μοντέλου προσομοίωσης, β) όρων ενεργειών στις μεθόδους οι οποίες είναι υπεύθυνες για τον χρονοπρογραμματισμό εκτέλεσης διεργασιών και έγερσης γεγονότων και γ) διαχείρισης πληροφοριών για την καταγραφή δεδομένων και την δημιουργία σημείων ελέγχου κατά την διάρκεια μίας προσομοίωσης.

➢ Ο σχεδιασμός και η υλοποίηση του DynConPBTool πραγματοποιείται βάσει νέων τεχνικών προγραμματισμού (όπως η AspectJ), συστημάτων (εγγενείς βάσεις δεδομένων XML, native XML Databases) και αρχιτεκτονικών (διαδικτυακές υπηρσίες, web services) τα οποία διασφαλίζουν με τον καλύτερο δυνατό τρόπο την ευελιξία και την επεκτασιμότητα του συστήματος.

➢ Η διαμόρφωση του μοντέλου καθώς επίσης και μεγάλο μέρος της συμπεριφοράς του συστήματος που προσομοιώνεται, ελέγχεται μέσω ενός εγγράφου XML. Με τον τρόπο αυτό,

ο χρήστης έχει την δυνατότητα εκτέλεσης σειράς πειραμάτων χωρίς να απαιτούνται μεταβολές στην υλοποίηση του συστήματος του.

Το DynConPBTool παρουσιάζει το ακόλουθο μειονέκτημα:

- Ο δυναμικός έλεγχος και η δυναμική διαμόρφωση της συμπεριφοράς του προς προσομοίωση συστήματος επιφέρει επιπρόσθετο κόστος στο χρόνο εκτέλεση της προσομοίωσης. Παρόλα αυτά, το κόστος αυτό δεν θεωρείται ιδιαίτερα σημαντικό αρκεί να αναλογιστεί κανείς την ραγδαία εξέλιξη των υπολογιστών.

Βιβλιογραφία

ΒΙΒΛΙΟΓΡΑΦΙΑ

ΑΡΘΡΑ

[Barak, 85] A. Barak and A. Shiloh, "A distributed load balancing policy for multicomputer", Software-Practice and Experience, vol. 15, no. 9, 1985, pp. 901-913.

[Bierman, 00] Bierman, G. M. Using XML as an Object Interchange Format, Department of Computer Science, University of Warwick, May 17, 2000, citeseer.nj.nec.com/374889.html. Current as of December 20th, 2000.

[Bryant, 81] R. M. Bryant and R. A. Finkel, "A stable distributed scheduling algorithm", in Proceedings 2nd International Conference of Distributed Computing Systems, April 1981, pp. 314-323.

[Buss, 02] Arnold Buss, "Simkit: component based simulation modeling with Simkit", Proceedings of the 34th conference on Winter simulation, San Diego, California, 2002, pp. 243-249.

[Buyya, 02] Rajkumar Buyya and Manzur Murshed, "GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing", Concurrency Computat.: Pract. Exper. 2002, vol. 14, pp. 1175–1220.

[Casavant, 88] Thomas L. Casavant and Jon G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems", IEEE Transactions on Software Engineering, vol. 14, no. 2, February 1998, 141-154.

[Eager, 86a] D. L. Eager, E. D. Lazowska and J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing", Performance Evaluation, vol. 6, no. 1, March 1986, pp. 53-68.

[Eager, 86b] D. L. Eager, E. D. Lazowska and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems", IEEE Transactions on Software Engineering, vol. SE-12, no. 5, May 1986, pp. 662-675.

[Eager, 88] D. L. Eager, E. D. Lazowska, J. Zahorjan, "The limited performance benefits of migrating active processes for load sharing", SIGMETRICS, May 1988, pp. 662-675.

[Foster, 02] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration", Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

[Gomoluch, 03] Jacek Gomoluch and Michael Schroeder, "Market-based Resource Allocation for Grid Computing: A Model and Simualtion", 1st International Workshop on Middleware for Grid Computing (MGC2003), Rio de Janeiro, Brazil, June 2003, pp. 211-218.

[Gupta, 99] Deepak Gupta and Pradip Bepari, "Load Sharing in Distributed Systems", Proceedings National Workshop on Distributed Computing", Jadavpur University, Calcutta, January 1999.

[Harchol, 96] M. Harchol-Balter and A. B. Downey, "Exploiting Process Lifetime Distributions for Dynamic Load Balancing", ACM Transactions on Computer Systems, vol. 15, Issue 3, August 1997, pp. 253 – 285.

[Hura, 02] Gurdeep S. Hura, Sheeja Mohan and T. Srikanthan, "On Load Sharing in Distributed Systems: A Novel Approach", Transactions of the Society for Design and Process Science, vol. 6, no. 1, March 2002, pp. 59-81.

[Jeffay, 98] Kevin Jeffay, F. Donelson Smith, Arun Moorthy, James Anderson, "Proportional Share Scheduling of Operating System Services for Real-Time Applications", in Proceedings of the 19th IEEE Real-Time System Symposium, Madrid, Spain, December 1998, pp. 480-491.

[Karatza, 02] Helen D. Karatza and Ralph C. Hilzer, "Load Sharing in Heterogeneous Distributed Systems", Proceedings of the 34th conference on Winter simulation, San Diego, California, 2002, pp. 489-496.

[Kremien, 92] Orly Kremien and Jeff Kramer, "Methodical Analysis of Adaptive Load Sharing Algorithms", IEEE Transactions on Parallel and Distributed Systems, Vol. 3 , No. 6, 1992, pp 747-760.

[Krueger, 87] P. Krueger and M. Livny, "The Diverse Objectives of Distributed Scheduling Policies", Proc. Seventh International Conference Distributed Computing Systems, IEEE, Los Alamitos,
Calif., 1987, pp. 242-249.

[Krueger, 88] Phillip Krueger and Miron Livny, "A Comparison of Preemptive and Non-Preemptive Load Distributing", in 8th International Conference on Distributed Computing Systems, San Jose, CA, USA, June 1988, pp. 123-130.

[Krueger, 94] Phillip Krueger and Niranjan G. Shivaratri, "Adaptive Location Policies for Global Scheduling", IEEE Transactions on Software Engineering, vol. 20, no. 6, June 1994, pp. 432-444.

[Kumar, 87] A. Kumar, M. Singhal and T. L. Ming, "A model for distributed decision making: An expert system for load balancing in distributed systems", in Proc. 11th Symposium Operating Systems, IEEE, 1987, pp. 507-513.

[Legrand, 03] Arnaud Legrand, Loris Marchal and Henri Casanova, "Scheduling Distributed Applications: the SimGrid Simulation Framework", Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'03)

[Livny, 82] M. Livny and M. Melman, "Load Balancing in Homogeneous Broadcast Distributed Systems", Proceedings of ACM Computer Network Performance Symposium, College Park, Maryland, United States, 1982, pp. 47-55.

[Lin, 92] Hwa-Chun Lin and C.S. Raghavendra, "A Dynamic Load-Balancing Policy With a Central Job Dispatcher (LBC)", IEEE Transactions on Software Engineering, vol. 18, no. 2, February 1992, pp. 148-158.

[Luke, 05] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan, "MASON: A Multi-Agent Simulation Environment", Simulation, Vol. 81, No. 7, 2005, pp. 517-527.

[Mirchandaney, 89] R. Mirchandaney, D., Towsley and J. A. Stankovic, "Analysis of the effect of delays on load sharing", IEEE Transactions on Computers, vol. C-38, no. 11, November 1989, pp. 1513-1525.

[Pierre, 02] Pierre L'Ecuyer, Lakhdar Meliani and Jean Vaucher, "SSJ: a framework for stochastic simulation in Java", Proceedings of the 34th conference on Winter simulation, San Diego, California, 2002, pp. 234-242.

[Pierre, 05] Pierre L'Ecuyer, Eric Buist, "Simulation in java with SSJ", Proceedings of the 37th conference on Winter simulation, Orlando, Florida, 2005, pp. 611-620.

[Ramamritham, 84] K. Ramamritham and J. A. Stankovic, "Dynamic task scheduling in distributed hard real-time systems", IEEE Software, vol. 1, no. 3, July 1984, pp. 65-75.

[Rasch, 70] Philip J. Rasch, "A Queuing Theory Study of Round-Robin Scheduling of Time-Shared Computer Systems", Journal of the ACM, vol. 17, no.1 , pp. 131-145, January 1970.

[Rips, 96] Sabina Rips, "Adaptive Load Distribution in Strong Dynamic Heterogeneous Systems", Proc. 2nd International Conference on Massively Parallel Systems, MPCS'96, Ischia, Italy, May 1996.

[Rotithor, 90] H. G. Rotithor and S. S. Pyo, "Decentralized decision making in adaptive task sharing", in Proceedings 2nd IEEE Symposium of Parallel and Distributed Processing, Dallas, TX, USA, December 1990, pp. 34-41

[Shin, 89] K. G. Shin and Y. C. Chang, "Load sharing in distributed real-time systems with state change broadcasts", IEEE Transactions on Computers, vol. C-38, no. 8, August 1989, pp. 1124-1142.

[Shivaratri, 90] N. G. Shivaratri and P. Kruger, "Two Adaptive Location Policies for Global Scheduling", Proceedings 10th International Conference Distributed Computing Systems, IEEE, Paris, France, May 28 - June 1, 1990, pp. 502-509.

[Shivaratri, 92] Niranjan G. Shivaratri, Phillip Krueger, and Mukesh Singhal, "Load Distributing for Locally Distributed Systems", IEEE Computer, December 1992, pp. 33-44.

[Stankovic, 84] J. A. Stankovic, "Simulations of three adaptive, decentralized algorithms", IEEE Transactions on Software Engineering, vol. SE-11, no. 10, 1985, pp. 1141-1152.

[Svensson, 92] Anders Svensson, "Dynamic Alternation Between Load Sharing Algoritms", Proc. of the 25th Hawaii International Conference on System Sciences, vol. 1, January 1992, pp. 193-201.

[Theimer, 89] M. M. Theimer and K. A. Lantz, "Finding idle machines in a workstation-based distributed system", IEEE Transactions on Software Engineering, vol. 15, no. 11, 1989, pp. 1444-1458.

[Zhou, 87] S. Zhou and D. Ferrari, "A measurement study of load balancing performance", in Proc. 7th International Conference DCS, IEEE, New York, 1987, pp. 490-497.

[Zhou, 88] Songnian Zhou, "A Trace-Driven Simulation Study of Dynamic Load Balancing", IEEE Transactions on Software Engineering, vol. 14, no.9, 1988, pp. 1327-1341.

[Γερονάτσιος, 2006] Γερονάτσιος Σπυρίδων, "Grid Computing", εργασία στα πλαίσια μεταπτυχιακού μαθήματος του τμήματος "Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων Πανεπιστημίου Αιγαίου", επιβλέπων καθηγητής: Αναγνωστόπουλος Ιωάννης.

[Μπογόρδος, 2005] Μπογόρδος Ηλίας, "Μελέτη Επίδοσης Η/Υ Για Υποδομές Συστοιχιών τύπου Grid", διπλωματική εργασία, Εθνικό Μετσόβιο Πολυτεχνείο, Σχολή Εφαρμοσμένων Μαθηματικών και Φυσικών Επιστημών, επιβλέπων καθηγητής: Αλεξόπουλος Θεόδωρος.

ΒΙΒΛΙΑ

[Bauer, 05] Christian Bauer and Gavin King, "Hibernate in Action", Manning Publications Co., 2005, ISBN 1932394-15-X.

[Cerami, 02] Ethan Cerami, "Web Services Essentials Distributed Applications with XML-RPC, SOAP, UDDI & WSDL", O'Reilly, First Edition, February 2002, ISBN: 0-596-00224-6.

[Chappell, 02] David Chappell and Tyler Jewell, "Java Web Services", O'Reilly, First Edition, March 2002, ISBN: 0-596-00269-6

[Chung, 04] Christopher A. Chung, "Simulation Modeling Handbook, A Practical Approach", 2004, CRC Press.

[Ezzio, 03] David Ezzio, "Using and Understanding Java Data Objects", Apress, 2003, ISBN:1590590430.

[Fujimoto, 00] Richard M. Fujimoto, "Parallel and Distributed Simulation Systems", Wiley-Interscience, United States of America, 2000, ISBN: 0-471-18383-0.

[Gradecki, 03] Joseph D. Gradecki and Nicholas Lesiecki, "Mastering AspectJ Aspect-Oriented Programming in Java", Wiley Publishing, Inc., 2003, ISBN: 0-471-43104-4.

[Laddad, 03] Ramnivas Laddad, "AspectJ in Action Practical Aspect-Oriented Programming", Manning Publications Co, 2003, ISBN 1-930110-93-6.

[McLaughlin, 02] Brett McLaughlin, "Java and XML Data Binding", O'Reilly, First Edition, May 2002, ISBN: 0-596-00278-5.

[Nagappan, 03], Ramesh Nagappan, Robert Skoczylas and Rima Patel Sriganesh, "Developing Java™
Web Services, Architecting and Developing Secure Web Services Using Java", Wiley Publishing Inc., 2003, ISBN: 0-471-23640-3.

[Perros, 03] Harry Perros, "Computer Simulation Techniques: The definitive introduction!", Computer Science Department NC State University, Raleigh, 2003.

[Richardson, 05] W. Clay Richardson, Donald Avondolio, Joe Vitale, Scot Schrager, Mark W. Mitchell and Jeff Scanlon, "Professional Java™, JDK™ 5 Edition", Wiley Publishing, Inc., 2005, ISBN: 0-7645-7486-8

[Sinclair, 04] J.B. Sinclair, "Simulation of Computer Systems and Computer Networks: A Process-Oriented Approach", 2004.

[Γαροφαλάκης, 99] Γιάννης Δ. Γαροφαλάκης, "Σημειώσεις για το μάθημα: Προσομοίωση Πληροφοριακών Συστημάτων ", Πανεπιστήμιο Πατρών Πολυτεχνική Σχολή, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Δεύτερη Έκδοση, 1999 .

ΔΙΚΤΥΑΚΟΙ ΤΟΠΟΙ

[db4o, man] http://www.db4o.com/downloads/db4o-6.1-java.zip, (the manual is included)

[Berkeley, hp] http://www.oracle.com/database/berkeley-db/xml/index.html, Oracle Berkeley DB XML.

[Berkeley, gs] http://www.oracle.com/technology/documentation/berkeley-db/xml/gsg_xml/java/index.html, Getting Started with Berkeley DB XML.

[Bourret, bind] http://www.rpbourret.com/xml/XMLDataBinding.htm, XML Data Binding Resources.

[Bourret, gnative] http://www.xml.com/pub/a/2005/03/30/native.html, Going Native: Making the Case for XML Databases.

[Bourret, xmldbs] http://www.rpbourret.com/xml/XMLAndDatabases.htm, XML and Databases.

[Castor, xmlMap] http://www.castor.org/xml-mapping.html, Castor XML Mapping.

[EyeDB, bind] http://doc.eyedb.org/manual/html/JavaBinding/index.html, EyeDB Java Binding.

[EyeDB, odl] http://doc.eyedb.org/manual/html/ODL/index.html, EyeDB Object Definition Language.

[EyeDB, oql] http://doc.eyedb.org/manual/html/OQL/index.html EyeDB Object Query Language.

[EyeDB, hp] http://www.eyedb.org/, Welcome to EyeDB.

[Franklin , sbs] http://archive.devx.com/dbzone/articles/sf0701/sf0701-1.asp, Using Object-Oriented Databases: A Step-By-Step Tutorial

[GlassFish, jaxb] https://jaxb.dev.java.net/, Overview.

[GridSim, hp] http://gridbus.csse.unimelb.edu.au/gridsim/, GridSim, Introduction.

[J-Sim, acc] http://www.j-sim.org/whitepapers /aca.html, “The Autonomous Component Architecture”.

[J-Sim, cwg] http://www.j-sim.org/guide/cwg.html, “Component Writer’s Guide”.

[J-Sim, hp] http://www.j-sim.org/, “J-Sim Home page”.

[J-Sim, thesis] http://www.j-sim.org/whitepapers/tyan_thesis.pdf, Hung-ying Tyan, “Design, Realization and Evaluation of a Component-Based Compositional Software Architecture for Network Simulation”, Dissertation, Ohio State University, 2002.

[J-Sim, wwjs] http://www.j-sim.org/tutorial/jsim_tutorial.html, “Tutorial: Working With J-Sim”.

[Java, jaxb] http://java.sun.com/developer/technicalArticles/WebServices/jaxb/, Java Architecture for XML Binding (JAXB).

[Java, jdo] http://java.sun.com/products/jdo/, Java Data Objects (JDO).

[Java, servlets] http://java.sun.com/products/servlet/overview.html, Java Servlet Technology Overview.

[javasim, hp] http://javasim.ncl.ac.uk/, JavaSim Home Page.

[javasim, man] http://javasim.ncl.ac.uk/progman/javasim.pdf, Programmer's Manual.

[JSR, 12] http://www.jcp.org/en/jsr/detail?id=12, JSR 12.

[JSR, 154] http://jcp.org/en/jsr/detail?id=154, JSR 154.

[jyd, hp] http://www.jyd.com/jyddown.html, JYD Object Database.

[Mason, hp] http://cs.gmu.edu/~eclab/projects/mason/, MASON

[MaRS, hp] http://www.ccs.neu.edu/home/matta/software.html, MaRS software.

[Nest, hp] ftp://ftp.cs.columbia.edu/nest/, Nest software.

[ns-2, hp] http://www.isi.edu/nsnam/ns/, The Network Simulator - ns-2.

[ObjectDB, hp] http://www.objectdb.com/, ObjectDB for Java/JDO.

[ObjectDB, man] http://www.objectdb.com/database/jdo/manual/, ObjectDB for Java/JDO - Developer's Guide.

[ObjectStore, Landis] http://research.sun.com/forest/COM.Sun.Labs.Forest.PJava.PJW2.14_pdf.pdf, ObjectStore PSE: a Persistent Storage Engine for Java.

[ODMG, hp] http://www.odmg.org/, Object Data Management Group, The Standard for Storing Objects.

[OptorSim, hp] http://edg-wp2.web.cern.ch/edg-wp2/optimization/optorsim.html, OptorSim Home page.

[OptorSim, userg] http://edg-wp2.web.cern.ch/edg-wp2/optimization/downloads/v2_0/edg-optorsim/doc/userguide-optorsim.ps, OptorSim user guide.

[PtolemyII, hp] http://prolemy.eecs.berkeley.edu/ptolemyII/, PtolemyII Home page.

[PtolemyII, intro] http://ptolemy.eece.berkeley.edu/ptolemyII/ptII4.0/ptII4.0.1/doc/design/ptIIdesign1-intro.pdf, “Introduction to Ptolemy II”.

[PtolemyII, pd] http://ptolemy.eece.berkeley.edu/ptolemyII/ptII4.0/ptII4.0.1/doc/design/ptIIdesign3-domains.pdf, “Ptolemy II Domains”.

[PtolemyII, sa] http://ptolemy.eece.berkeley.edu/ptolemyII/ptII4.0/ptII4.0.1/doc/design/ptIIdesign2-software.pdf, “Ptolemy II Software Architecture”

[REAL, hp] http://minnie.tuhs.org/REAL/, The REAL Network Simulator.

[Repast, Collier] http://www.econ.iastate.edu/tesfatsi/RepastTutorial.Collier.pdf, N. Collier, RePast: An extensible framework for agent simulation

[Repast, hp] http://repast.sourceforge.net/, Recursive Porus Agent Simulation Toolkit.

[Repast, man] http://repast.sourceforge.net/tutorials.html, Howto Documentation.

[serviceArch, odbmsPros] http://www.service-architecture.com/object-oriented-databases/articles/one_model_to_reduce_development_and_maintenance_costs.html, "One model to reduce development and maintenance costs".

[serviceArch, imped] http://www.service-architecture.com/object-oriented-databases/articles/lack_of_impedance_mismatch.html, "Lack of impedance mismatch".

[serviceArch, odbmsProb] http://www.service-architecture.com/object-oriented-databases/articles/why_use_sql_instead_of_an_oodbms.html, Why Use SQL Instead of an OODBMS?.

[SimGrid, faqs] http://simgrid.gforge.inria.fr/doc/faq.html, Frequently Asked Questions.

[SimGrid, hp] http://simgrid.gforge.inria.fr/, The SimGrid project.

[simjava, hp] http://www.icsa.inf.ed.ac.uk/research/groups/hase/simjava/, simjava.

[simjava, man] http://www.icsa.inf.ed.ac.uk/research/groups/hase/simjava/guide/tutorial.html, The SimJava Tutorial.

[simkit, hp] https://diana.nps.edu/Simkit/, Simkit.

[simkit, begm] https://diana.nps.edu/~ahbuss/papers/BasicEventGraphModeling.pdf, Basic Event Graph Modeling.

[simkit, dep] https://diana.nps.edu/~ahbuss/papers/Discrete Event Programming with Simkit.pdf, Discrete Event Programming with Simkit.

[ssj, hp] http://www.iro.umontreal.ca/~simardr/ssj/, SSJ: Stochastic Simulation in Java.

[ssj, man] http://www7.informatik.uni-erlangen.de/~ksjh/teaching/0203W/simmod_1/f/ue/pdf/guide.pdf, SSJ A Java Library for Stochastic Simulation, User's Guide.

[Swarm, hp] http://www.swarm.org/wiki/Main_Page, "Swarm Home page".

[Swarm, tue] http://www.essex.ac.uk/ccfea/swarm/SwarmTutorial/web/swarm_tutorial.html, "University of Essex, Center for Computational Finance and Economic Agents, Swarm Tutorial"

[Swarm, ug] http://www.swarm.org/swarmdocs-2.1.1/userbook/userbook.html, "Swarm User Guide"

[Tomcat, hp] http://tomcat.apache.org/, Apache Tomcat.

[wiki, orm] http://en.wikipedia.org/wiki/Object-relational_mapping, Object-relational mapping.

[w3s, xml] http://www.w3schools.com/xml/default.asp, XML Tutorial.

[w3s, xpointer] http://www.w3schools.com/xlink/default.asp, XLink and XPointer Tutorial.

[w3s, xpath] http://www.w3schools.com/xpath/default.asp, XPath Tutorial.

[w3s, xslt] http://www.w3schools.com/xsl/default.asp, XSLT Tutorial.

[w3s, xquery] http://www.w3schools.com/xquery/default.asp, XQuery Tutorial.

[XMLBeans, intro] http://xmlbeans.apache.org/, Welcome to XMLBeans.

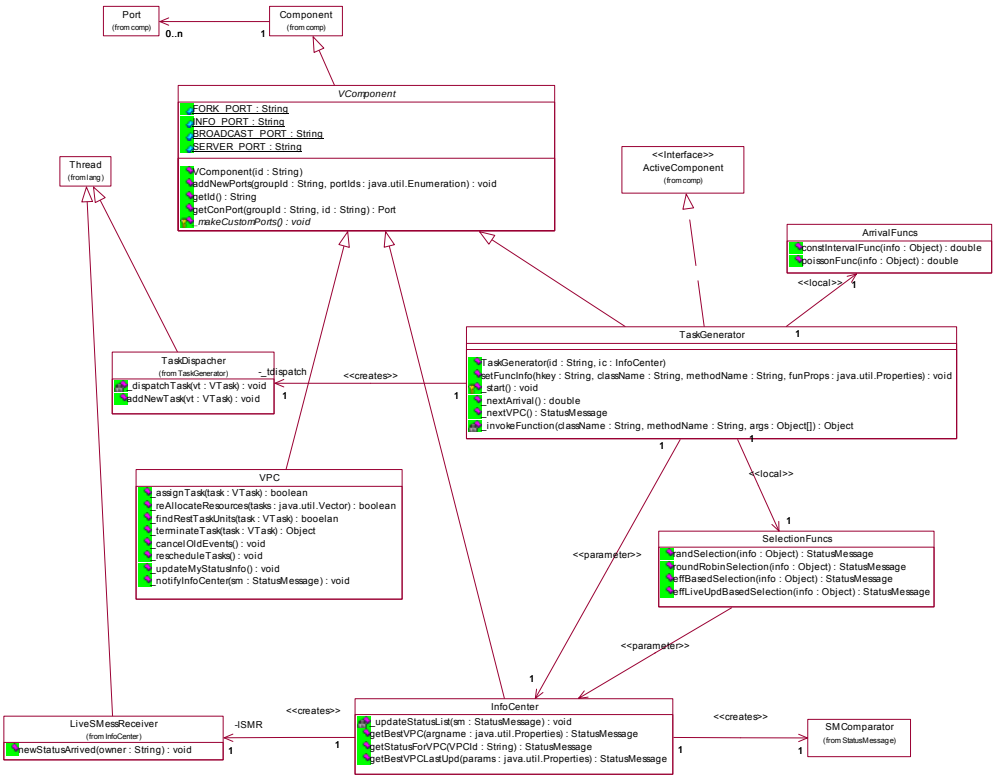
[XStream, hp] http://xstream.codehaus.org/, XStream.

[XStream, Tillman] http://www.devx.com/xml/Article/30068, Get Moving with Java-XML Serialization Using XStream.

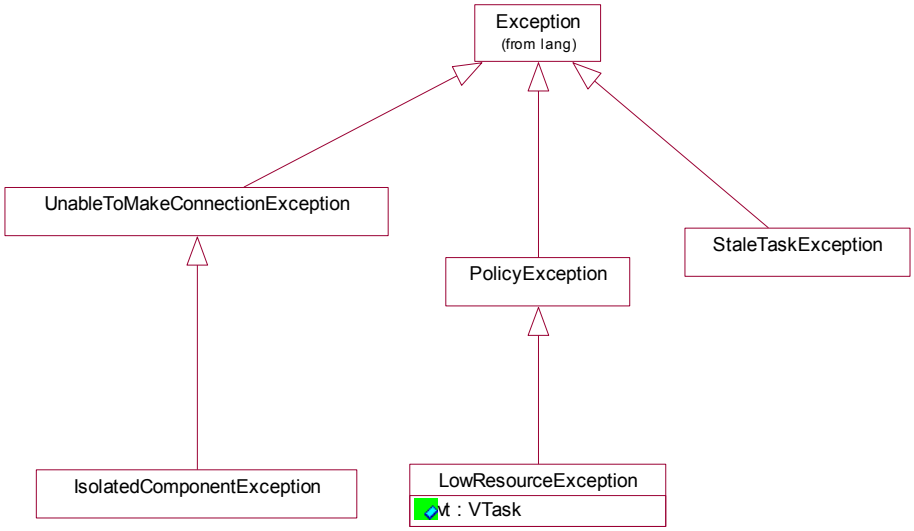
Παραρτήματα

Παράρτημα I

Περιγραφή των βασικών κλάσεων του πειραματικού περιβάλλοντος και απεικόνιση των σχέσεων μεταξύ αυτών υπό την μορφή UML διγαμμάτων.

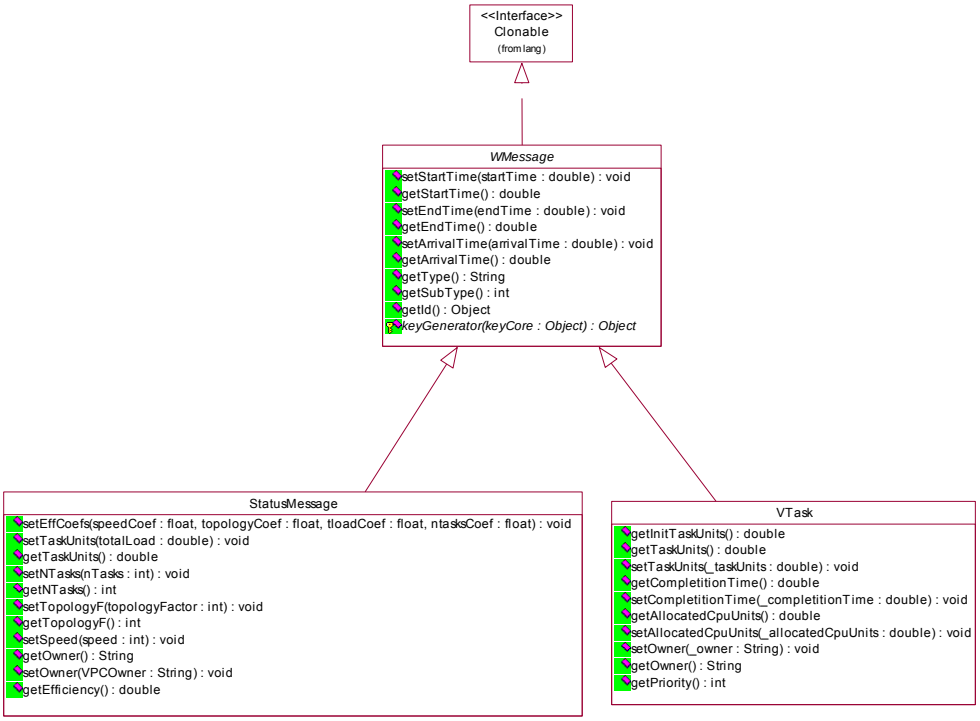


vpcs.VPC
It incarnates the most essential entity of the real system; the processor.
vpcs.VComponent
The basic class component. Every user's custom component has to inhere from this parent class. The common initialization part such as default port connections takes place in this class.
vpcs.TaskGenerator
Generates the tasks and assigns them to the proper VPC according the used methods for location policy and task generation interval.
vpcs.TaskGenerator.TaskDispacher
An up and running thread responsible for task dispatching. The newly generated tasks from TaskGenerator are hosted in an internal queue and dispatched when the information policy is completed.
vpcs.InfoCenter
This component acquires, holds and provides the information of the system's state. Its operational mode is dictated by the user through the XML configuration file.
vpcs.InfoCenter.LiveSMessReceiver
An up and running thread. It checks whether or not the InfoCenter has received the last updated status messages from all VPCs during the LIVE_MODE operational mode. When all information is acquired, TaskGenerator is notified.
vpcs.ArrivalFuncs
Contains the user defined methods for the estimation of task generation interval. The selection of the method takes place through XML configuration file.
vpcs.SelectionFuncs
Implements the user-defined location policies. The user should take care of using the suitable "keys" (based on the XML configuration file) in order to retrieve the respective function's parameters.



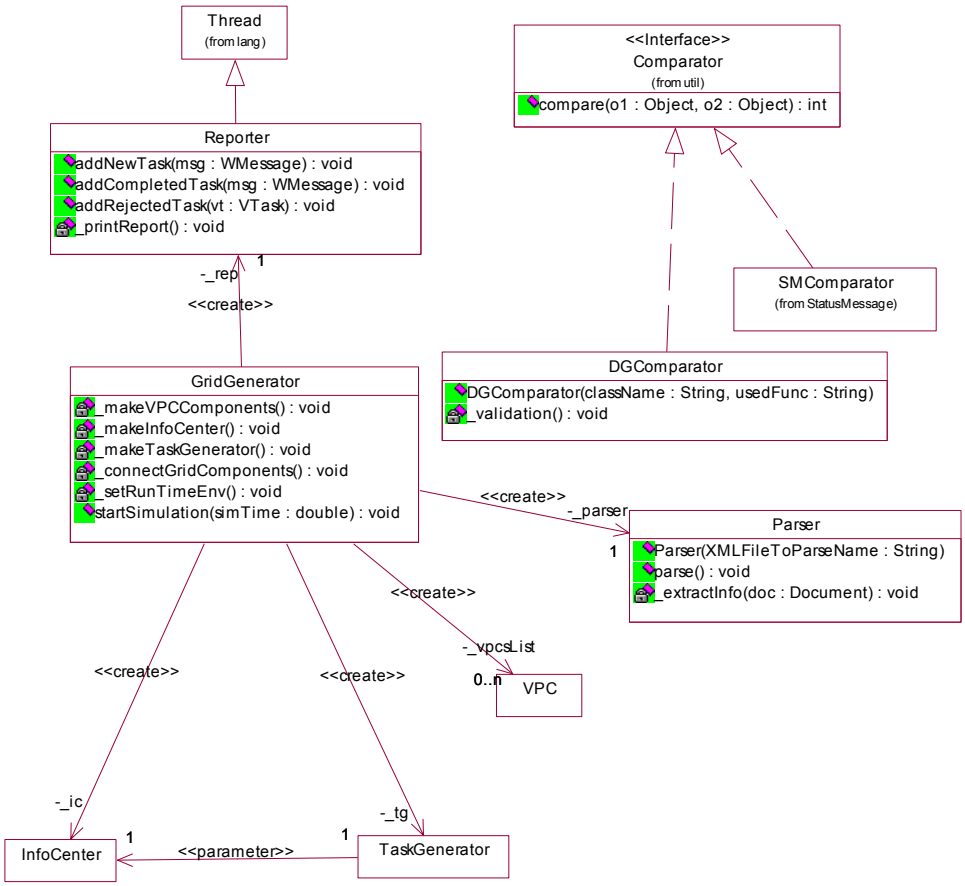
vpcs.UnableToMakeConnectionException
It is triggered when the framework is unable to create either a port or a connection.
vpcs.IsolatedComponentException
It is triggered when a component is isolated as result no message could receive or send.
vpcs.PolicyException
It is triggered when the task does not comply with the component policy.
vpcs.LowResourceException
It is triggered when a VTask does not abide by the VPC's resource allocation policy. More specific, it is occured when the estimated allocated cpu resources for a task, is lower than the predefined in configuration file.
vpcs.StaleTaskException
It is triggered when the VTask has already been completed by previous execution context.

ΠΑΡΑΡΤΗΜΑΤΑ



vpcs.WMessage
The basic class of the messages. It holds important events during it life cycle (such as its creation, destruction and the arrival time to a destination). The descendant classes should implement the method that returns the unique key for the message.
vpcs.StatusMessage
Holds information about the state of the VPC.
vpcs.VTask
Represents the task.

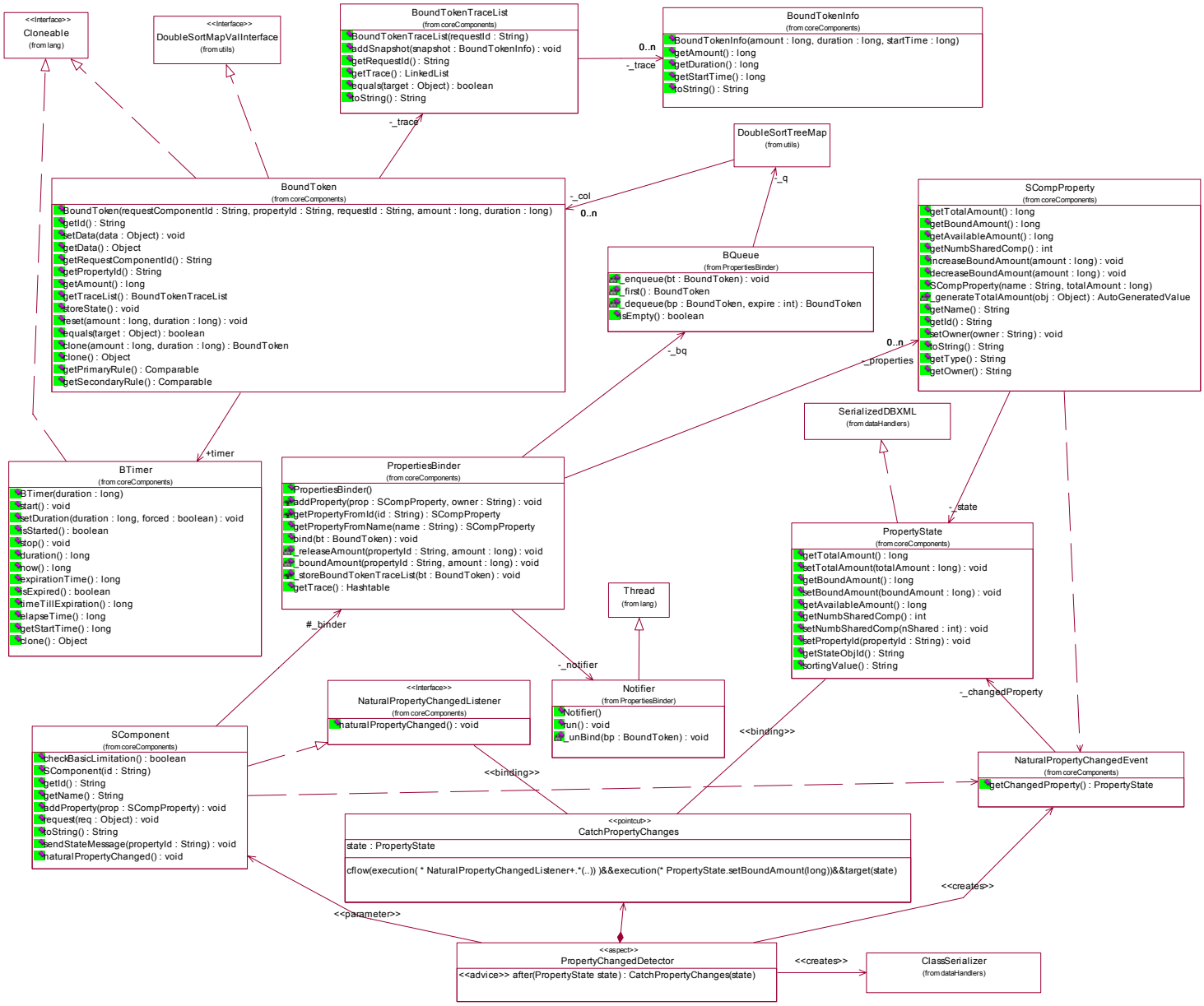
ΠΑΡΑΡΤΗΜΑΤΑ



vpcs.GridGenerator
Constructs the grid. Creates grid's components according to configuration file, connects them and starts the simulation.
vpcs.Reporter
Generates the report of the simulation. Its data is dynamically updated during the simulation.
vpcs.Parser
Parses the XML configuration file and retrieves the required information for grid construction.
vpcs.SMComparator
A custom comparator for StatusMessage objects based on "getEfficiency" method.
vpcs.DGComparator
A generic comparator based on a method. The comparison method should return a java.lang.Double object.
vpcs.XMLCustomTagsInterface
Holds various constants used to retrieve the values of the functions' parameters.
vpcs.XMLKeysInterface
Holds constants that should be used from any class that handles or uses the parsed information of the XML configuration file.

Παράρτημα II

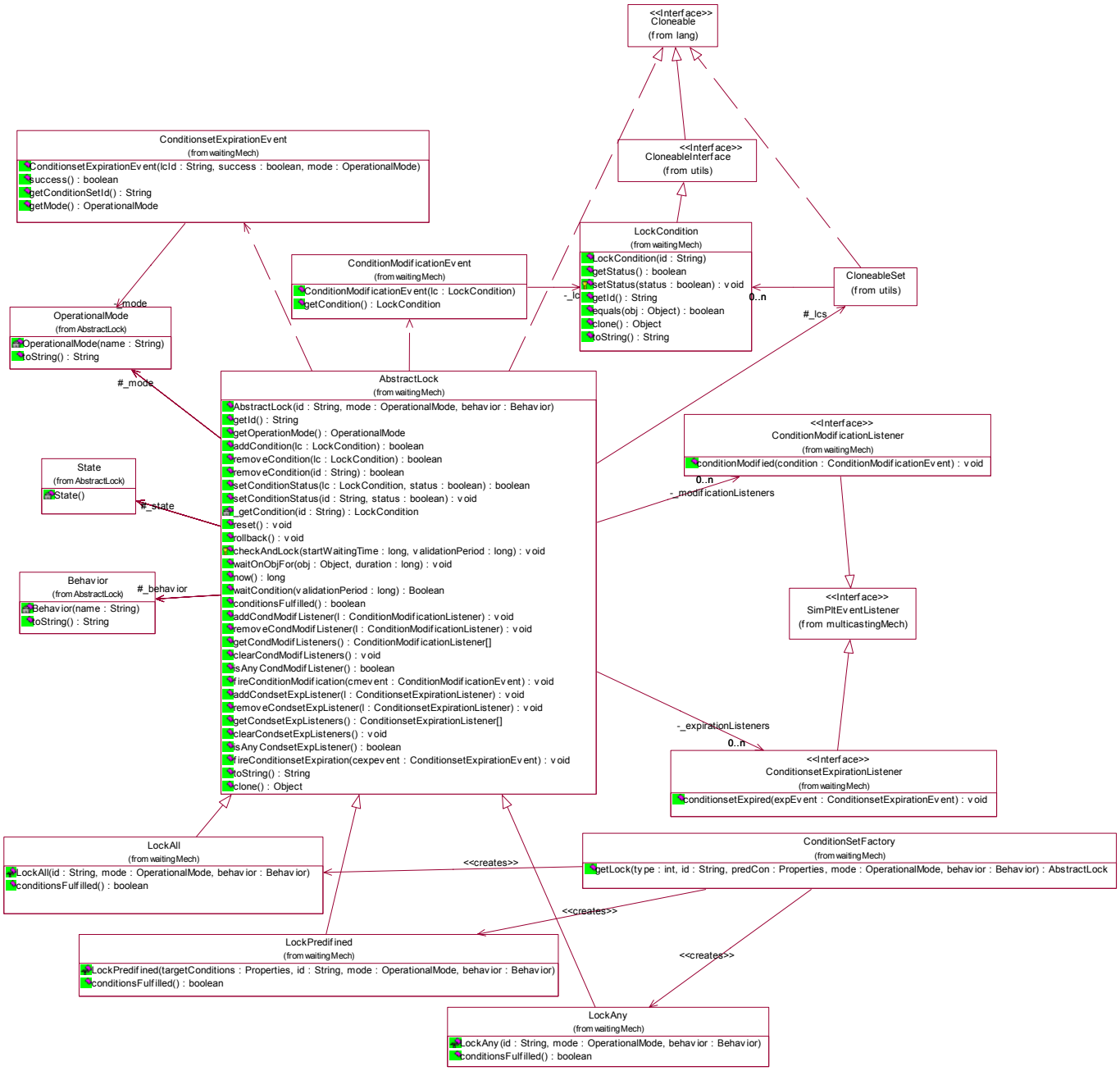
Περιγραφή των βασικών κλάσεων του DynConPBTool και απεικόνιση των σχέσεων μεταξύ αυτών υπό την μορφή UML διγαμμάτων.



pBSimPlt.coreComponents.SComponent
The basic simulation component. The owner of a set of properties (SCompProperty), by which the negotiation among the other components takes place. Its solid purpose is to accordingly evaluate (accept or reject) and realise either the requests for binding a portion of its properties or the invitations to bind properties of other components.
pBSimPlt.coreComponents.SCompProperty
Incarnates a foundanmental part of the components, the property. Components can have more than one properties. Components' objective is to either make requests for binding some portion of the properties which belong to other components or make invitations to other components for binding some of their properties. The quantification of the property is realized by the units (as long number). The default value of the units of a property is 100. The user could specify the units in the configuration file, either by an explicit value or a value generated by a specified method (using the auto generated value mechanism).

pBSimPlt.coreComponents.SCompProperty.PropertyState
Holds the state of the propery.
pBSimPlt.coreComponents.PropertiesBinder
Coordinates the properties' bindings.
pBSimPlt.coreComponents.PropertiesBinder.BQueue
Holds BoundToken objects sorted by the ascending order of expiration time of the related binding.
pBSimPlt.coreComponents.PropertiesBinder.Notifier
An up and running thread which checks if the completetion of the bindings.
pBSimPlt.coreComponents.BTimer
It holds time-related information about a procedure. More specific, it marks the time when the procedure starts (e.g. when an object enters in the queue, see at BoundToken), and the time when the procedure should be ended (e.g. when an object should leave the queue). It gives the time period elapsed after the commencement of the procedure, and the remaining time until the procedure fulfillment.
pBSimPlt.coreComponents.BoundTokenTraceList
Holds the trace for a specific binding request. The trace consists of a set of snapshots where a specific amount of the units of a property is bound for a specific time period.
pBSimPlt.coreComponents.BoundTokenInfo
It is the elementary element of the trace of a property bound request since it holds the basic information of a snapshot of the respective property binding; where specific amount of units of the property are bound for specific time period.
pBSimPlt.coreComponents.BoundToken
Holds the information about a bound request. When the component accepts a property's bound request, an object of this type is formed holding information such as: * the component id that made the request, * the property id, * the request id, * the amount and * the duration of the property binding. It maintains a timer which is responsible for coordinating the times related to the property binding. The timer should start counting when the PropertiesBinder insert this object into its queue (BQueue) which means that either the bound request is accepted by the component or the component recalculates the amount and the binding period for some reasons.
pBSimPlt.coreComponents.PropertyChangedDetector
Detects the property changes using the interface NaturalPropertyChangedListener and the method PropertyState.setBoundAmount(long). Informs the respective listener invoking the method naturalPropertyChanged(NaturalPropertyChangedEvent), and stores the property state in the DBXML.
pBSimPlt.coreComponents.NaturalPropertyChangedListener
The listener interface for receiving natural property change events. The class that is interested in processing an natural property change event for a natural property implements this interface. When the natural property change event occurs, that object's naturalPropertyChanged method is invoked.
pBSimPlt.coreComponents.NaturalPropertyChangedEvent
An event which indicates the change of the state of a natural property.

ΠΑΡΑΡΤΗΜΑΤΑ



pBSimPlt.waitingMech.AbstractLock

Provides the basic condition set mechanism. Compose the set of conditions LockCondition. The concept/model is that one or more threads, the blocked threads, wait for the fulfillment of a set of conditions, the condition set, for a specific period of time, the validation or expiration period, at most, by other threads, the modifiers.

* It has two operational modes: in the first (UNBLOCK_WITH_COMPLETION), the blocked threads wait for the conditions completion till the validation period expires.
* In the second (UNBLOCK_WITH_EXPIRATION), the threads stop waiting when the condition set is completed. In this case the time period for which a blocked thread waits for condition set accomplishment can be equal to expiration period at most.

The above behavior could be degraded in simpler one where there are only modifiers. The blocked threads and the related quantities (the expiration period) are absent from the model since there are not any threads waiting on the condition set completion (the method waitCondition(long) has not any affect). In this case, despite the operational mode which a condition is regarded to have, this mode has no any influence on the condition set mechanism and could be used as a flag in conjunction with other operations (see at Executor). The former behavior (where blocked thread participate in the model) is called blocked, whereas the latter is called unblocked.

A condition set could have registered listeners at some of its important events, such as when a modification of a condition occurs and whether the satisfaction of the condition set is done or not during the expiration period.

In the unblocked behavior the expiration period has no use, and the condition set will be satisfied eventually (even in infinite time). Consequently, in unblocked behavior the listeners are notified about the condition set's modifications and condition set successful completion.

Note that the methods which implement the actions triggered by the notifications should include only light-weight operations since their execution is not performed to a seperate thread.

pBSimPlt.waitingMech.AbstractLock.Behavior

Implements the possible values of the AbstractLock._behavior.

pBSimPlt.waitingMech.AbstractLock.OperationalMode

Implements the possible values of the AbstractLock._mode.

pBSimPlt.waitingMech.AbstractLock.State

Implement the possible values of the AbstractLock._state.

pBSimPlt.waitingMech.LockAll

Provides the mechanism where the condition set is accomplished as long as all the included conditions become true.

pBSimPlt.waitingMech.LockAny

Provides the mechanism where the condition set is accomplished as long as any of the lock condition becomes true.

pBSimPlt.waitingMech.LockCondition

The condition of a condition set. The status of this condition combined with the status' of the rest conditions which belong to the same set, determine the result of the set (fulfillment or not).

pBSimPlt.waitingMech.LockPredifined

Provides the mechanism where the condition set is accomplished as long as the status' of its conditions are the same with the predefined condition set status. The conditions which are not included in the condition set status have no influence on the accomplishment of the condition set.

pBSimPlt.waitingMech.ConditionModificationEvent

An event which indicates the condition modification of a condition set.

pBSimPlt.waitingMech.ConditionsetExpirationEvent

An event which indicates whether the condition set is successfully accomplished or not.

pBSimPlt.waitingMech.ConditionSetFactory

Construct a type of a condition set.

pBSimPlt.waitingMech.ConditionModificationListener

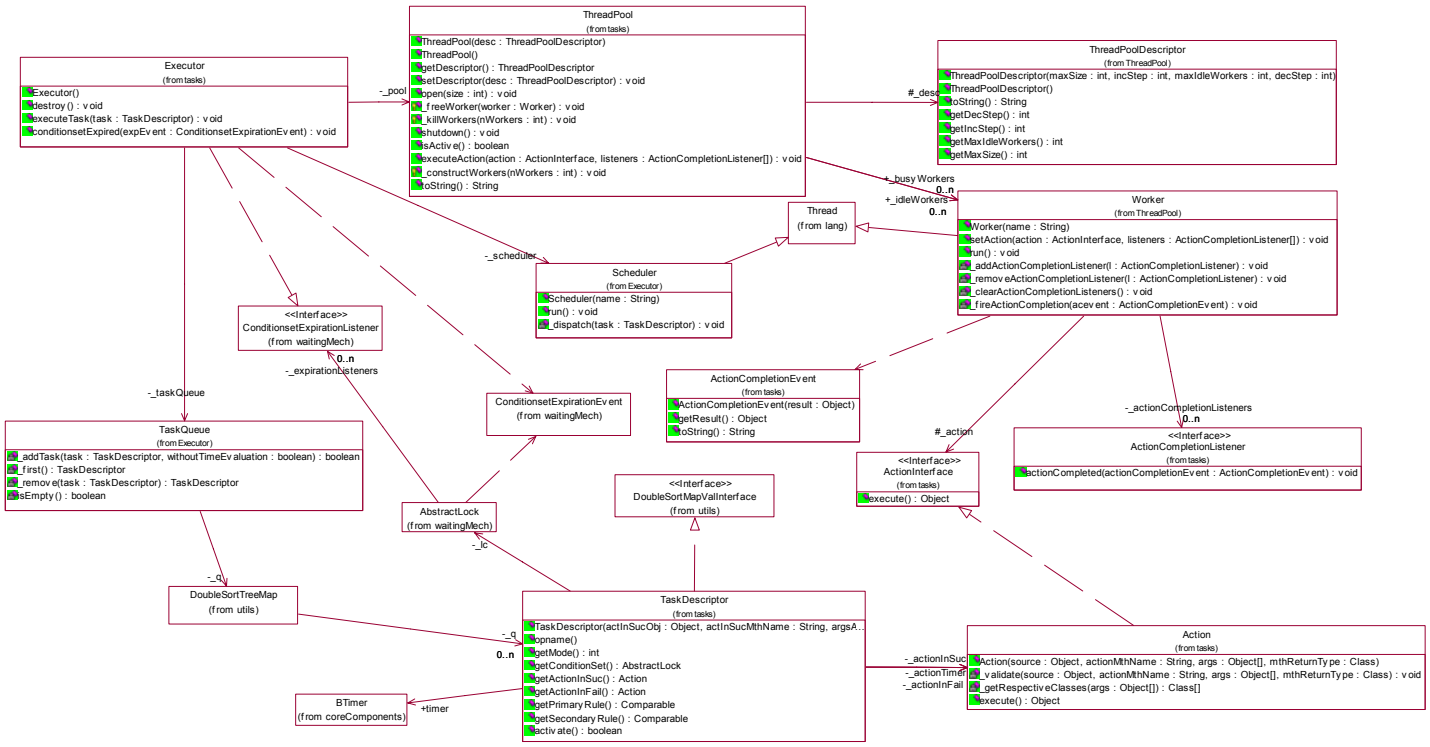
The listener interface for receiving condition modification events. The class that is interested in processing an condition modification event implements this interface, and the object created with that class is registered with a condition set, using the addCondModifListener method. When the modification event occurs, that object's conditionModified method is invoked.

pBSimPlt.waitingMech.ConditionsetExpirationListener

The listener interface for receiving condition accomplishment events. The class that is interested in processing an condition accomplishment event implements this interface, and the object created with that class is registered with a condition set, using the addCondExpListener method. When the condition set accomplishment event occurs, that object's conditionExpired method is invoked.

The methods which implement this listener should include only light-weight operations since their execution is not performed to a seperate thread.

ΠΑΡΑΡΤΗΜΑΤΑ



pBSimPlt.tasks.Action

Describes and holds all the necessary information about a method so as this method could be dynamically invoked. The required information for a dynamically method invocation are:

* the method name,
* the method's arguments,
* an object of the class which declares this method or simply the declaration class in case this class has default constructor (in order to use the default constructor to create the object).

The execution of the method is performed by invoking the execute() method of the action object.

pBSimPlt.tasks.ActionCompletionEvent

An event which indicates the completion of an action/task by a ThreadPool.Worker.

pBSimPlt.tasks.Executor

This class, the executor, along with the TaskDescriptor, the task descriptor, implements a much more flexible approach of the java.util.Timer and the java.util.TimerTask classes.

It schedules at a specific time, the scheduled execution time or the scheduled time in short (STime), the execution of the actions specified in a TaskDescriptor object. That is, the task decriptor is an object consists of a set of actions Action which are accordingly used and executed by the executor.

The task descriptor has three actions, the action in success (AINS), the action in failure (AINF), and the action in giving the next scheduled time (AINT). The actions AINS and AINF are executed in separate threads provided by a thread pool, since they describe the actual tasks that user desires to execute at specific time. Their execution is determined according the executor's behavior/model which is described later in this section.

The invoked methods which actually incarnate the tasks and they are described in action objects do not follow a predifined name pattern since the classes to which belong have not to implement specific interfaces as usual. The methods and the respective declaration classes are dynamically constructed and stored in Action objects. There is also no need to define a fixed scheduled time at all. The user can define its own method which is responsible for returning the scheduled time every time is required according the executor behavior. The executor behavior is described as follows, please also refer to {TaskDescriptor} and Action} in order to perceive the model throurouly.

The executor's behavior could be classified in two categories considering whether a condition set is defined in the task descriptor or is not defined. Lets take at first the simpler situation where no condition set is specified.

* In this situation, the executor repeatively schedules the execution of the method described and stored in AINS, according to the value returned by the method described in the AINT. We say repeatively since the executor, after dispatching the action to the proper thread for execution, does not

stop rescheduling the AINS until the AINT returns null.
* In case a condition set is defined, the behavior of the executor is more complicated since it depends on the condition set (Recall that a condition set consists of some conditions which should partly or thoroughly be accomplished in order that condition set to be completed. For full description of a condition set functionality refer to AbstractLock). The rules that determine the executor's behavior are:
1. If the condition set is completed until the STime passes, the AINS is executed either at the STime, in case the condition set's mode is UNBLOCK_WITH_EXPIRATION, or at the time of condition set's completion, in case the condition set's mode is UNBLOCK_WITH_COMPLETION. After that, the executor reinitialize the condition set (restore its initial values) and reshedule (if it is possible, the AINT does not return null) the execution of the AINS.
2. If the condition set is not completed until the STime passes, the AINF (if it is defined) is executed at the STime. If the mode of the task descriptor is STOP_IN_FAILURE the executor rejects the descriptor permantly. If the mode of the task descriptor is REPEAT_ANY_CASE the executor reinitialize the condition set (if it is possible; in case the condition set has the blocked behavior the reinitialization is possible only when the condition check is terminated) and reshedule (if it is possible, the AINT does not return null) the execution of the AINS

In current implementation, it is recommended to use condition sets with unblocked behavior

pBSimPlt.tasks.Executor.Scheduler

Implements the executor's model. Determines when and which action of the task descriptor should be executed. Dispatches the action to the thread pool when the proper time arrives. Suggests the rescheduling of an action when it is dictate by the model. We say suggest since the final decision is taken by the task queue (the resheduling is not permited if the AINT returns null and the executor is under destruction).

pBSimPlt.tasks.Executor.TaskQueue

The queue which holds the task descriptors sorted according to their sceduled time.

pBSimPlt.tasks.TaskDescriptor

Holds the actions which are properly used by the executor.

It contains the following three actions:

* The action in success (AINS). Holds the method which is dynamically invoked by the executor either within the scheduled time in case the specified condition set is accomplished, or at scheduled time in the absence of a condition set. This action is mandatory and should be specified consequently.
* The action in failure (AINF). Holds the method which is dynamically invoked by the executor in case the specified condition set is not accomplished within the scheduled time. This action is optional and affects the executor's behavior if a condition set is specified.
* The action in giving the next scheduled time (AINT). Holds the method which is dynamically invoked by the executor in order to determine the scheduled time. The return type of the method must be java.lang.Long and the return value represents the delay, in ms, in which the specified action might be executed.

It has two modes, the stop in failure and the repeat any case which accordingly used by the executor's model (for more information please refer to Executor).

It is this object which actually participate in the scheduling mechanism of the exector and implements the DoubleSortMapValInterface interface consequenly.

pBSimPlt.tasks.ThreadPool

Implements a thread pool. The characteristic quantities (operational parameters) of this implementation of the thread pool are:

* the maximun number of the available threads/workers. The number of the workers that could offer the thread pool can not exceed the specified limit.
* the maximun number of the workers which are in suspending state, waiting to overtake an imminent task/action assignment (the so-called idle workers).
* The construction step. The number of the workers which are additionally constructed in case there are not any idle workers to serve a task.
* The destruction step. The number of the workers which are destructed by the thread pool in case the number of the idle workers ecxeed a predified number. This happens in order to avoid the existence of increased number of idle workers caused by major fluctuations of the ratio of the task assignments.
* The maximun number of the idle workers above which the thread pool destructs workers as much the quantity decrease step indicates.

Note that the class represents a task should implement the ActionInterface interface. In addition to that, listeners could be register at the ActionCompletionEvent event for a specified action, triggered when the completion of this action occurs.

pBSimPlt.tasks.ThreadPool.ThreadPoolDescriptor

Contains the operational parameters of the thread pool described in ThreadPool. Note that the operational partameters could be defined either during the thread pool construction or when the thread pool is inactive, via the setDescriptor method.

pBSimPlt.tasks.ThreadPool.Worker

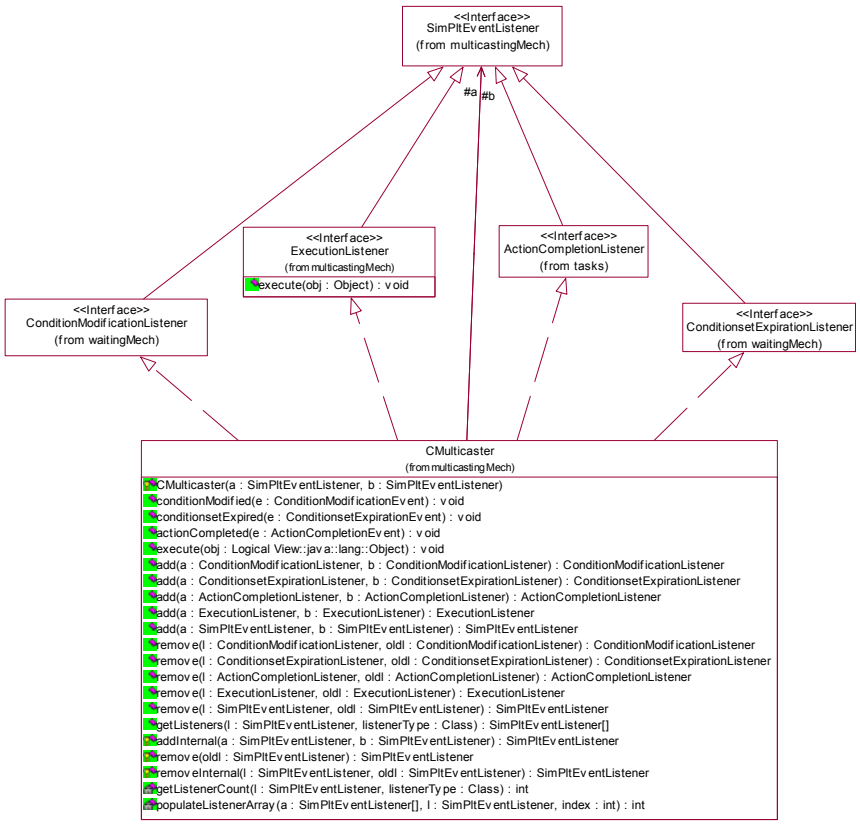
Implements the thread, the worker, that undertakes the execution of an action.

pBSimPlt.tasks.ActionCompletionListener

The listener interface for receiving action completion events. The class that is interested in processing an action completion event for an action implements this interface. The object created with that class is automatically registered by the ThreadPool.Worker which undertakes the execution of the this action, when this object is included in the listeners passed as argument in the method executeAction() When the action completion event occurs, that object's actionCompleted method is invoked.

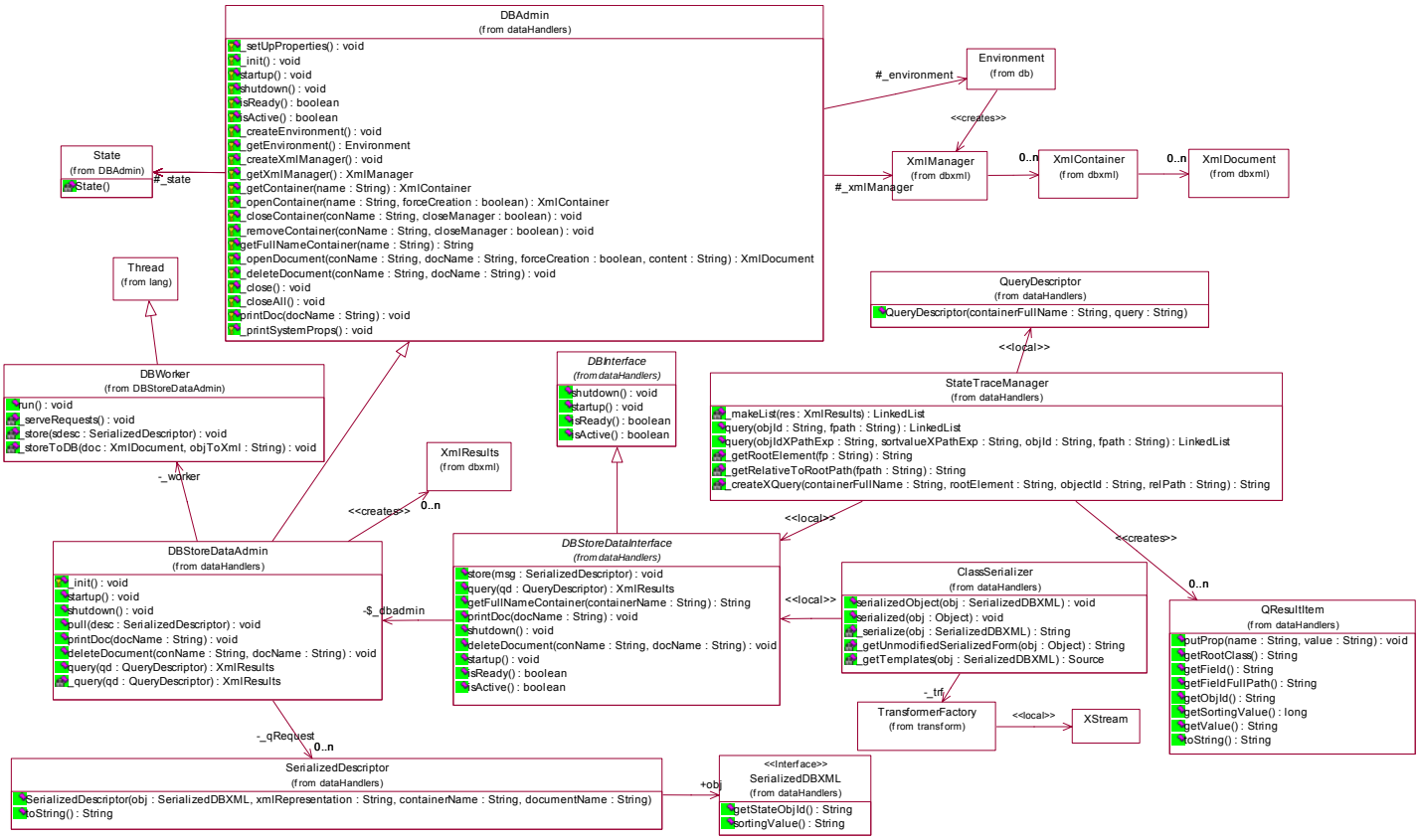
pBSimPlt.tasks.ActionInterface

This interface should be implememnted by all classes represent tasks which are executed by a worker constructed by a thread pool ThreadPool.



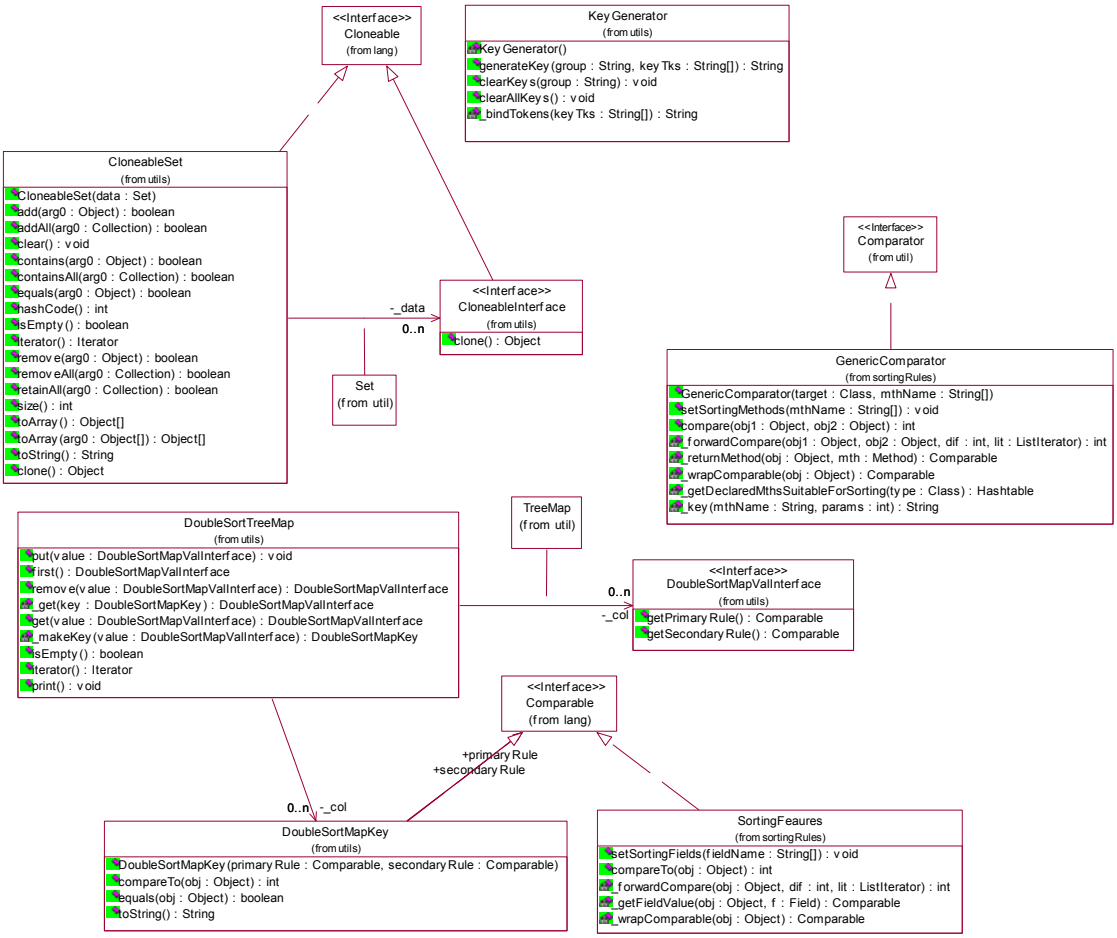
pBSimPlt.multicastingMech.CMulticaster
A class which implements efficient and thread-safe multi-cast event dispatching for the AWT events defined in the java.awt.event package. This class will manage an immutable structure consisting of a chain of event listeners and will dispatch events to those listeners. Because the structure is immutable, it is safe to use this API to add/remove listeners during the process of an event dispatch operation (there is no need of creating a copy of the listeners before the commence of the notification procedure). However, event listeners added during the process of an event dispatch operation will not be notified of the event currently being dispatched. This class is based on java.awt.AWTEventMulticaster.
pBSimPlt.multicastingMech.SimPltEventListener
A tag interface that all event listener interfaces must extend.
pBSimPlt.multicastingMech.ExecutionListener
The listener interface for receiving execution events. The class that is interested in processing an execution event (a worker thread probably) implements this interface, and the object created with that class is registered with the execution, using the addExecutionListener method. When the execution event occurs, that object's execute method is invoked.

ΠΑΡΑΡΤΗΜΑΤΑ



pBSimPlt.dataHandlers.DBAdmin
Controls the interactions between an environment handler, a xmlnanager handler and the opened containers. The configuration parameters reside in the DBXML.properties file located at the folder of the source/bin code of this class. One environment handler is releated to one xmlmanager, and the same xmlmanager is used for opening many containers. The containers are internally registered in this class so illegal actions could be avoided (for exmaple it is forbiden to close the manager and the environment if all the containers are not closed before). The class is NOT thread safe.
pBSimPlt.dataHandlers.DBAdmin.State
Express the states of the DB.
pBSimPlt.dataHandlers.DBStoreDataAdmin
Holds additional mechnisms for storing XML documents.
pBSimPlt.dataHandlers.DBStoreDataAdmin.DBWorker
An up and running thread responsible for serving the requests for data storage in DB. When the DB's shutdown process is in progress the thread serves the remaining requests and closes the DB.
pBSimPlt.dataHandlers.DBInterface
The basic operations in DB.
pBSimPlt.dataHandlers.DBStoreDataInterface
The interface of DBStoreDataAdmin. This class delegates the control to the real class that handles DB's interactions DBStoreDataAdmin. it is used a combination of delegation and singleton pattern (regarding the DBStoreDataAdmin object) since the objective is to use in a static way (through DBStoreDataInterface) the functionality of an object which is not naturally static (DBStoreDataAdmin).
pBSimPlt.dataHandlers.ClassSerializer
It is responsible for object serialization in Berkeley DB XML.
pBSimPlt.dataHandlers.QResultItem
Holds the information extracted from the query.
pBSimPlt.dataHandlers.QueryDescriptor

Describes the query.
pBSimPlt.dataHandlers.SerializedDescriptor
Describes the serialized object, including the object and its serialized form.
pBSimPlt.dataHandlers.StateTraceManager
It is responsible for making query at Berkeley DB.
pBSimPlt.dataHandlers.SerializedDBXML
It should be implemented by every serialized class in BDBXML.



pBSimPlt.utils.CloneableSet
Essentially, a Set that makes a deep copy of itself via the clone method. Its methods delegates the requests to the respective methods of the Set collection. In order to have a kind of abstraction in the clone method, the elements of this collection should implement the CloneableInterface.
pBSimPlt.utils.DoubleSortMapKey
The automatically generated key that maps a specified stored value in a DoubleSortTreeMap map. Determines the ascending order of the map's entries.
pBSimPlt.utils.DoubleSortTreeMap
The implementation of this sorted map uses the java.util.TreeMap class inheriting its behavior and its performance advantages. The stored values are provided alone. The respective keys are automatically constructed according to the information extracted by the values via a predefined interface. The keys are objects of DoubleSortMapKey type. Their order is determined by two comparable (implement the Comparable interface) objects, the primary and the secondary sorting rule which are retrieved by the methods defined in the DoubleSortMapValInterface. This interface should be implemented by the class from which the values of this collection derive. The key ascending order is firstly determined by the primary rule, whereas the relative order between keys with the same primary sorting rule is specified by the secondary sorting rule. Two keys are equal if they have the same primary and secondary sorting rule.
pBSimPlt.utils.KeyGenerator

Generates a unique id, via the method generateKey, for the specified group of objects, based on the specified string tokens. The derived key comprises the string tokens separated by the DELIM.

The method checks the validity of the tokens (if it is not null) firstly, and secondly, if the derived key has already been existed.

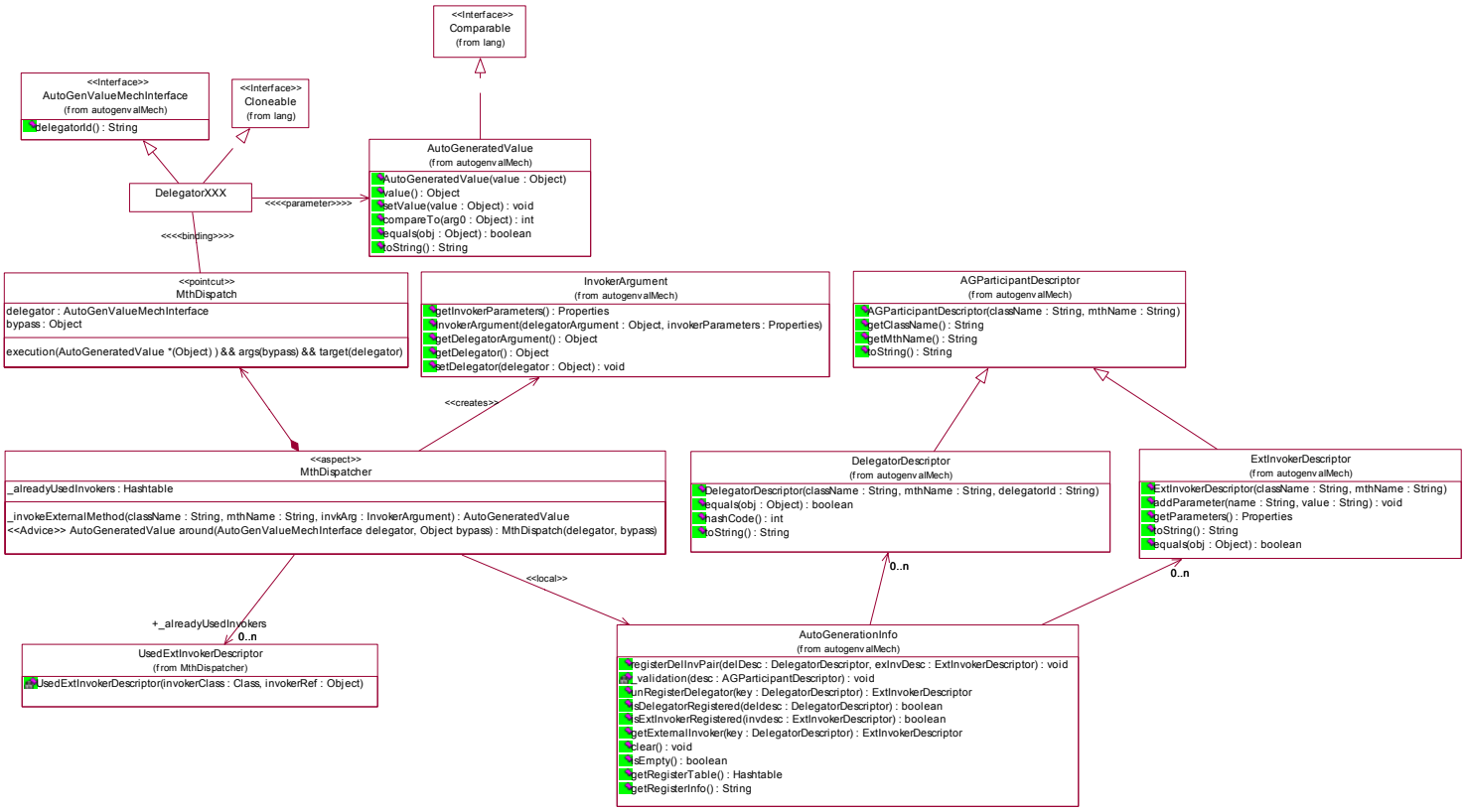
* In the latter case a token is automatically generated according to an auto generated number which is appended to the last string token.
For example:
if the specified string tokens are {CompX, propY}, the proposed key is compX###propY. If this key has already been used by another object of the specified group, the system appends to the last token an auto generated number ensuring the uniqueness of the key (for example compX###propY145).
* In the former case a generated key is based on the name of the group that the id belongs to and an auto generated number. For example if the array of strings which holds the tokens is either null or holds a null token value, or is empty, and the id belongs to component group, then an imminent id could be the component473.

pBSimPlt.utils.CloneableInterface

Instead of Cloneable interface. Achieve a kind of abstraction in the clone methods of collections which contains objects that implements this interface

pBSimPlt.utils.DoubleSortMapValInterface

This inteface must be implemented by the class from which the values of the DoubleSortTreeMap derives. The methods return the primary and the secondary sorting rule.



pBSimPlt.autogenvalMech.MthDispatcher

The mechanism of the auto generated values/or alternative behaviors is described as follows:
A delegator is a method that could delegate its functionality to another method defined in other class. A delegator is primarily defined by the pair of the name of method and the name of the class that this method belongs to. The pair of the method that the delegator's method forward its control and the class that this method belongs to is called external invoker (or invoker).

The aforementioned design does not support a distinction on object basis, since a delegator method of all the objects which derived from the same class could have only one respective external invoker (Every delegator could have only one invoker). In order to be an object basis invoker differentiation the following concept is introduced:

A class which has delegators should implement a method of a predifined interface (AutoGenValueMechInterface). This method returns a string, the so-called delegatorId, which is used to distinguish an invoker among many for the same delegator. For example, suppose that objects A and B derives from the same class C which has the delegator method foo(). If the delegatorId for object A is delId1 and the delegator id for object B is delId2 then there is the potential to register the invoker invA() for object with delId1 and the invoker invB() for object with delId2.

Note that, the delegatorId for SComponent and SCompProperty could be represented by their names defined in the XML configuration file. So the method delegatorId() returns SComponent.getName() and SCompProperty.getName() respectively.

In the XML configuration file the pairs (delegator className, delegator methodName) and the respective pairs (invoker className, invoker methodName) are madantory. In case the invoker is registered for a specific delegator, the delegatorId must follow the above pairs. If the delegationId is defined the invoker is called specific delegatorId invoker, otherwise general invoker. The general invoker is applied to all objects which have not a respective specific delegatorId invoker defined. The delegator descriptor that derives from a general invoker, sets the delegatedId equal to null. The characterization of an invoker takes palce the time when it is registered with the respective delegator since the delegatorId is that which determines whether the invoker is general or not.

Note that for each delegator, the MthDispatcher tries to find the respective specific delegatorId invoker at first. Afterwards, in case of failure, it tries to locate the respective general invoker.

A class could have any number of delegators. The delegator's and the invoker's methods should follow a certain singnature pattern:

* the delegator method must have one argument of Object object type and AutoGeneratedValue return type.
* The invoker method must have one argument of InvokerArgument object type and AutoGeneratedValue return type.

These restrictions from the delegators's part, are checked via the singnature of the pointcut MitDispatch (since it catches only the delegators' methods which comply with the aforementioned restrictions); while from invokers' part the restriction accompanied with some additional checkings are performed by the AutoGenerationInfo.

The delegator's method pass its argument to the invoker's method encapsulated in a InvokerArgument object along with the parameters which are defined via the XML configuration file (see at InvokerArgument).

The delegator does not forward its execution to the invoker if the delegator's method argument is null or no respective invoker is defined.

The current aspect catches the execution of the delegators, and forwards their execution to the respective invokers (if there is any, and it is the desirable by delegators)

pBSimPlt.autogenvalMech.MthDispatcher.UsedExtInvokerDescriptor

Holds information about the already used external invoker.

pBSimPlt.autogenvalMech.AutoGenValueMechInterface

Every class that has delegators should implement this class.

pBSimPlt.autogenvalMech.AutoGeneratedValue

Represents the auto generated value. The object that is either returned by the delegator's method or by the invoker's method.

pBSimPlt.autogenvalMech.InvokerArgument

Implements the argument of the external invoker. Holds the argument passed through the delegator and the parameters defined via the XML configuration file.

pBSimPlt.autogenvalMech.AGParticipantDescriptor

The root class for the objects that hold information about the delegator and the invoker (the participants of the alternative behaviors mechanism).

pBSimPlt.autogenvalMech.DelegatorDescriptor

Holds the information about a delegator.

pBSimPlt.autogenvalMech.ExtInvokerDescriptor

Holds the information about an external invoker.

pBSimPlt.autogenvalMech.AutoGenerationInfo

Registers the external invokers and the respective delegators. The registration is perfomed by a hashtable (registration table) having keys DelegatorDescriptor objects which hold the information about the delegators and values ExtInvokerDescriptor objects which house the information of the respective external invokers.

A DelegatorDescriptor is unique since each delegator's method with specified delegatorid could have only one external invoker.

Κατάλογος συντετμημένων όρων

ABSP…… *Agent-Based Simulation Platform*
AINF……. *Action In Failure*
AINS……. *Action In Success*
AINT……. *Action In Giving the Next Scheduled Time*
AOP…….. *Aspect Oriented Programming*
CD……. *Connection Distance*
CORBA……. *Object Request Broker Architecture*
DCOM…….. *Distributed Component Object Model*
DES…….. *Discrete Event Simulation*
DESP……. *Discrete Event Simulation Platform*
DIC……... *Database Information Center*
DTR…….. *Distance Time Rate*
DynConPBTool… *Dynamic Configurable Property-Based Simulation Toolkit*
ECD…….. *Elementary Connection Distance*
ECDMap…….. *Elementary Connection Distance Map*
EDA…….. *Event Driven Approach*
FITA……. *Fixed-Increment Time Advance*
FL…….. *Far Layout*
FPathL……... *Field Path Location*
GCI…… *Common Gateway Interface*
GCSP……. *Grid Computing Simulation Platform*
GOA…….. *Graphical Object Area*
GQA……. *Graphical Query Result Area*
IIOP…….. *Internet Inter-ORB Protocol*
JAXB…… *Java Architecture for XML Binding*
JCP…… *Java Community Process*
JDO……... *Java Data Objects*
JRMP……. *Java Remote Method Protocol*
JSR…… *Java Specification Request*
LUIA……. *Live Update Information Algorithm*
NETA…… *Next-Event Time Advance*
NL……. *Near Layout*
ODBMS…… *Object Oriented Management System*
ODL…….. *Object Definition Language*
ODMG…….. *Object Data Management Group*
OGSA…… *Open Grid Services Architecture*
OIF…… *Object Interchange Format*
OOP…….. *Object Oriented Programming*
OQL…….. *Object Query Language*
ORM……. *Object to Relational Mapping*
PA……. *Process Aproach*
QCA…….. *Query Composition Area*
RDBMS…… *Relational Database Management System*
RIAC…… *Repeat In Any Case*
RMI…….. *Remote Method Invocation*
RPC……... *Remote Procedures Calls*
RR……. *Round-Robin*
SINF…….. *Stop In Failure*
SOA……... *Service Oriented Architecture*
SOAP……. *Simple Object Access Protocol*
SOId…….. *State Object Identifier*

ΠΑΡΑΡΤΗΜΑΤΑ

SSC…… *Snapshot State Code*
SSR…… *Snapshot Sorting Rule*
STime…… *Scheduled Time*
UIA…… *Update Information Algorithm*
UML……... *Unified Modeling Language*
UNWCL……. *Unblock With Completion*
UNWEX……. *Unblock With Expiration*
WS……. *Web Services*
XML…….. *Extensible Markup Language*
XPath……. *XML Path Language*
XPointer……. *XML Pointer Language*
XSLT……. *Extensible Stylesheet Language Transformations*
XQuery…….. *XML Query Language*

ΠΑΡΑΡΤΗΜΑΤΑ

Βιογραφικό Σημείωμα

ΒΙΟΓΡΑΦΙΚΟ ΣΗΜΕΙΩΜΑ

Ο κ. Ψωρούλας Ιωάννης γεννήθηκε στην 1 Δεκεμβρίου 1972 στην Αθήνα. Αποφοίτησε από το 5ο Λύκειο Ζωγράφου Αττικής το 1990 και το ίδιο έτος εισήχθη μέσω των πανελληνίων εξετάσεων στο Γεωλογικό Τμήμα του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών. Το έτος 1992, κατόπιν και πάλι μέσω των πανελληνίων εξετάσεων εισήχθη στην Σχολή Χημικών Μηχανικών του Εθνικού Μετσόβιου Πολυτεχνείου από όπου έλαβε το 1998 το δίπλωμα του με βαθμό "Λίαν Καλώς (8.16)" και σειρά επιτυχίας 22^{ος} στους 182.

Τον Νοέμβριο του 1998 κατατάχθηκε στον Στρατό Ξηράς, από όπου απολύθηκε το Μάιο του 2000.

Από τον Αύγουστο του 2000 έως τον Φεβρουάριο του 2001 εργάστηκε ως υπεύθυνος παραγωγής και ποιοτικού ελέγχου στην βιομηχανική μονάδα παραγωγής εγγράψιμων ψηφιακών δίσκων (CD-R), Intersonic ABEE.

Στην συνέχεια και για περίπου έξη μήνες (έως τον Αύγουστο του 2001) εργάστηκε στην εταιρεία πληροφορικής Intrasoft International. Εκεί απασχολήθηκε α) ως HDO (Help Desk Operator) β) ως Tester εφαρμογών και γ) ως προγραμματιστής για την ανάπτυξη εφαρμογών με χρήση της γλώσσας προγραμματισμού Visual Basic.

Τον Οκτώβριο του 2002 έγινε δεκτός από τον Τομέα Πληροφορικής (νυν Τμήμα Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής) της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Ε.Μ.Π., για την απόκτηση διδακτορικού διπλώματος.

Κατά τη διάρκεια των μεταπτυχιακών του σπουδών, παρακολούθησε και ολοκλήρωσε τα ακόλουθα έξη μαθήματα: 1) μη γραμμικά φαινόμενα σε ηλεκτρονικές κατασκευές βιομηχανικών εγκαταστάσεων, 2) υλικά και περιβάλλον, 3) τεχνητά νευρωνικά δίκτυα και μηχανική μάθηση, 4) μηχανική του παγκόσμιου ιστού (web engineering), 5) αλγόριθμοι ψηφιακής επεξεργασίας σημάτων και εφαρμογές και 6) τεχνολογία της επικοινωνίας.

Η ερευνητική δραστηριότητα του κ. Ψωρούλα Ιωάννη εστιάζεται στις διαδυκτιακές υπηρεσίες, στα συστήματα προσομοίωσης, στα υπολογιστικά πλέγματα καθώς επίσης στην τεχνολογία πολυμέσων και λογισμικού.

Από τον Σεπτέμβριο του 2001 είναι επιστημονικός συνεργάτης στο εργαστήριο τεχνολογίας πολυμέσων του Ε.Μ.Π., όπου κατά τη διάρκεια των μεταπτυχιακών του σπουδών συμμετείχε σε αρκετά ερευνητικά προγράμματα, χρηματοδοτούμενα από την ευρωπαϊκή κοινότητα καθώς επίσης και στην συγγραφή προτάσεων.

Τα έργα στα οποία έχει συμμετάσχει είναι τα ακόλουθα:

• MANTRIP (MANagement Testing & Reconfiguration of IP based networks using mobile software agents), IST-1999-10921.

Ο κύριος στόχος του έργου αυτού ήταν η υλοποίηση ενός συνόλου πρωτοποριακών εργαλείων διαχείρισης δικτυακών πόρων, με την βοήθεια της τεχνολογίας των κινητών πρακτόρων (Mobile Agent Technology, MAT). Ο κ. Ψωρούλας Ιωάννης στο έργο αυτό, σχεδίασε και υλοποίησε το γραφικό περιβάλλον (Graphical User Interface) της κεντρικής εφαρμογής διαχείρισης του συστήματος καθώς επίσης σύνολο «κινητών πρακτόρων», σε γλώσσα προγραμματισμού Java.

• ERMIS (Electronic commeRce Measurements through Intelligent agentS) , IST-1999-21051.

Το έργο αυτό αφορούσε την μελέτη και την στατιστική μέτρηση νέων μεθόδων και δεικτών που σχετίζονται με το εξελισσόμενο πεδίο του ηλεκτρονικού εμπορίου. Μέσω του προγράμματος αυτού μελετήθηκαν καινούργιες μέθοδοι ενισχυμένης αναζήτησης ευφυούς αναγνώρισης της πληροφορίας στο διαδίκτυο με έμφαση στις υπηρεσίες και τα μοντέλα της

ψηφιακής οικονομίας. Παράλληλα, υλοποιήθηκαν στατιστικές εφαρμογές που αποσκοπούσαν στον υπολογισμό του πληθυσμού των ιστοσελίδων που προσφέρουν υπηρεσίες ηλεκτρονικού εμπορίου καθώς επίσης και στην συνεχή και real-time μέτρηση του ρυθμού ανανέωσης, δημιουργίας και ακύρωσης των ιστοσελίδων αυτών στον κυβερνοχώρο. Η συμμετοχή του κ. Ψωρούλα Ιωάννη σχετίζονταν με την ανάπτυξη εργαλείων τόσο σε γλώσσα προγραμματισμού Java όσο και σε C.

• HERMES (An Accompanying measure for European Commision's IT Nodes actions in the areas of Technologies for Business Processes, Electronic Commerce, Multimedia Content and Tools and Web Technologies), IST-2000-26429.

Ο βασικός στόχος του έργου αυτού ήταν η πραγματοποίηση «σεμιναρίων» σε επιλεγμένες περιοχές εφαρμογής της τεχνολογίας της πληροφορίας, όπως είναι το ηλεκτρονικό εμπόριο, πολυμεσικό περιεχόμενο και εργαλεία, διαδικτυακές τεχνολογίες. Ο ρόλος του κ. Ψωρούλα Ιωάννη στο έργο αυτό ήταν ο σχεδιασμός και η ανάπτυξη «μαθημάτων» σε γλώσσα προγραμματισμού VRML, η οποία χρησιμοποιείται για την τρισδιάστατη μοντελοποίηση και απεικόνιση αντικειμένων στο χώρο.

Ο κ. Ψωρούλας Ιωάννης, κατά το διάστημα 2003-2004, εργάστηκε στην εταιρία «Διακοινοτική» όπου ήταν υπεύθυνος για τον σχεδιασμό και την ανάπτυξη εφαρμογών σε πλατφόρμα GIS (Geographic Information System) για μεγάλους δημόσιους οργανισμούς (ΕΥΔΑΠ). Ενώ, από τον Φεβρουάριο του 2007 έως σήμερα εργάζεται ως αναλυτής, προγραμματιστής και σύμβουλος πληροφορικής στην εταιρία Datamation, για τον σχεδιασμό και την ανάπτυξη αυτόνομων ή δικτυακών διατραπεζικών εφαρμογών και εργαλείων.

Μερικές από τις γλώσσες προγραμματισμού και τα πακέτα προγραμμάτων που κατέχει αναφέρονται ενδεικτικά ακολούθως:

α) Γλώσσες προγραμματισμού: Java (Servlets, JSP, JavaBeans, JAXP, JAXB, JAXM/SOAP), AspectJ, C/C++, FORTRAN, BASIC, Excel Visual Basic for Applications, HTML, XML, XSLT, XPath/XQuery, JavaScript, VRML, JUnit.

β) Βάσεις δεδομένων/ εξυπηρετητές:Tomcat, MySQL, Berkeley DB XML.

γ) Λειτουργικά Συστήματα: MS-DOS, Unix, Windows.

δ) Εργαλεία & IDEs: Rational Rose (Software Engineering-Database Design), Eclipse, Borland JBuilder, Ant.

Τέλος, ο κ. Ψωρούλας Ιωάννης παραδίδει σειρές διαλέξεων στο πλαίσιο των μαθημάτων "Γραφική με Υπολογιστές" και "Τεχνολογίες Πολυμέσων" με έμφαση στο γραφικό και στο δικτυακό προγραμματισμό με την γλώσσα προγραμματισμού Java. Τα μαθήματα αυτά προσφέρονται από το πρόγραμμα σπουδών της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών του Ε.Μ.Π.

ΕΠΙΣΤΗΜΟΝΙΚΕΣ ΔΗΜΟΣΙΕΥΣΕΙΣ

ΑΡΘΡΑ ΣΕ ΠΕΡΙΟΔΙΚΑ

[Π1] "A study of the parameters concerning load balancing algorithms", **I. Psoroulas**, I. Anagnostopoulos, V. Loumos and E. Kayafas, International Journal of Computer Science and Network Security – IJCSNS, Vol. 7, no.4, pp. 202-214, 2007.

[Π2] "Quality of Service Management in IP Networks Using Mobile Agent Technology", Mota T., Gouveris S., Pavlou G., Michalas A., **Psoroulas J**., Lecture Notes In Computer Science, Vol. 2521, pp. 193-205, 2002.

ΑΡΘΡΑ ΣΕ ΠΕΡΙΟΔΙΚΑ (υπό κρίση)

[π1] "License plate recognition from still images and video sequences: A survey", C. Anagnostopoulos, I. Anagnostopoulos, **I. Psoroulas,** V. Loumos and E. Kayafas, IEEE Transactions on Intelligent Transportation Systems (under review process).

ΑΡΘΡΑ ΣΕ ΣΕΙΡΕΣ ΒΙΒΛΙΩΝ

[A1] "Implementing an Internet image search service based on the AJAX web programming techniques", I. Anagnostopoulos, C. Anagnostopoulos, **I. Psoroulas** and I. Maglogiannis, Studies in Computational Intelligence, Springer Book Series (in press).

[A2] "Cost-Effective Service Task Assignment In Multi-Domain Distributed Computing Environments", Malati Louta, Angelos Michalas, **Ioannis Psoroulas** and Evangelos Loutas, 2nd International Conference on E-Business and Telecommunication Networks, ICETE 2005, pp. 164-168, INSTICC Press 2005, ISBN 972-8865-32-5, October 3-7, 2005, Reading, UK.

[A3] "A man-machine meta-search interface, for handling multiple query submissions in web search services", Anagnostopoulos I., **Psoroulas I**., Anagnostopoulos C., Loumos V., Kayafas E., Recent Advances in Computers, pp. 192-197, WSEAS Press, 2002.

ΑΡΘΡΑ ΣΕ ΠΡΑΚΤΙΚΑ ΣΥΝΕΔΡΙΩΝ

[Σ1] "Increazing interactivity in IPTV using MPEG-21 descriptors", Christos-Nikolaos Anagnostopoulos, George Tsekouras , Damianos Gavalas, Daphne Economou and **Ioannis Psoroulas**, 4th IFIP Conference on Artificial Intelligence Applications & Innovations (AIAI) September 19-21, 2007, Special Session:Intelligent Internet Systems: Emerging Technologies and Applications (IIS-ETA 07).

[Σ2] "An Artificial Neural Network Approach for Classifying E-Commerce Web Pages", Anagnostopoulos I**.**, Kouzas G., Anagnostopoulos C., **Psoroulas I.**, Vergados D., Loumos V. and Kayafas E., pp. 237-242, IASTED AI 2003, February 10-13, 2003, Innsbruck, Austria.

[Σ3] "Information fusion meta-search interface for precise photo acquisition on the web", I. Anagnostopoulos, C. Anagnostopoulos, **I. Psoroulas**,V. Loumos and E. Kayafas", 25th International Conference on Information Technology Interfaces, ITI 2003, pp. 375-381, June 16-19, 2003, Cavtat/Dubrovnik, CROATIA.

[Σ4] "UMSE: A User defined Meta-Search Engine, for effective Information revealing", Anagnostopoulos I., **Psoroulas I.**, Loumos V. and Kayafas E., 8th International Conference on OPTIMIZATION OF ELECTRICAL AND ELECTRONIC EQUIPMENT, OPTIM2002 May 16-17, pp. 797-802, 2002, Brasov, ROMANIA.

[Σ5] "Software Measurements for Protection Capacity Optimisation in SONET", Anagnostopoulos I., Anagnostopoulos C., **Psoroulas I.**, Kayafas E., and Loumos V., paper accepted for presentation at the 12th IMEKO TC 4 International Symposium "Electrical Measurement and Instrumentation", pp. 213-218, September 25-27, 2002, Zagreb, Croatia.

[Σ6] "Implementing a customised meta-search interface for user query personalisation", Anagnostopoulos I., **Psoroulas I.**, Loumos V. and Kayafas E., 24th International Conference on Information Technology Interfaces, ITI 2002, pp. 79-84, June 24-27, 2002, Cavtat/Dubrovnik, CROATIA.