ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ
ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΣΤΑ
ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη περιβάλλοντος προσομοίωσης μέσω διαδικτύου (web based simulation environment): για εκπαιδευτικούς σκοπούς

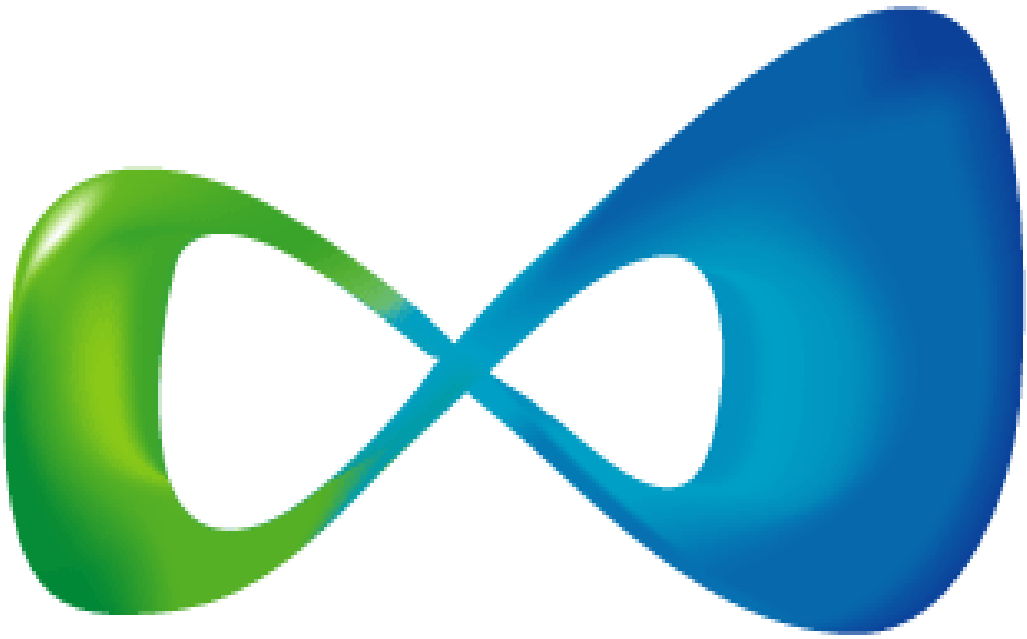
Κοντογιάννης Αντώνιος
(Α.Ε.Μ. 8/03)

Επιβλέπων Καθηγητής: Γεωργίου Ανδρέας
Εξεταστής Καθηγητής: Οικονομίδης Αναστάσιος

Θεσσαλονίκη, Οκτώβριος 2007

simmate

A WEB BASED SIMULATION ENVIRONMENT



Contact Information:

Developer:
Kontogiannis Antonis
email: akontog@gmail.com

Supervisor:
Georgiou Andreas
email: acg@uom.gr

ΔΗΛΩΣΗ

Δηλώνω υπεύθυνα ότι το παρών κείμενο αποτελεί προϊόν προσωπικής μελέτης και εργασίας και πώς όλες οι πηγές που χρησιμοποιήθηκαν για τη συγγραφή της δηλώνονται σαφώς είτε στις παραπομπές είτε στο βιβλιογραφικό κατάλογο.

ΠΕΡΙΛΗΨΗ

Η μίμηση της λειτουργίας ενός συστήματος, ή μέρους αυτού, με σκοπό τη διερεύνηση πιθανών βελτιώσεων και την ανεύρεση των επιδράσεων διαφορετικών πολιτικών στο πραγματικό σύστημα, καλείται προσομοίωση του συστήματος. Η προσομοίωση με τη βοήθεια ηλεκτρονικού υπολογιστή περικλείει τον πειραματισμό σε ένα, βασισμένο στον υπολογιστή, μοντέλο του αρχικού συστήματος. Το μοντέλο χρησιμοποιείται σαν ένα όχημα για πειραματισμό. Για να γίνει όμως αυτό, είναι απαραίτητη μία σειρά ελιγμών. Πρώτα από όλα, πρέπει να χτιστεί ένα μοντέλο του αρχικού συστήματος. Το μοντέλο αυτό θα είναι μία αφαίρεση του αρχικού – πραγματικού συστήματος που θα περιέχει όλους τους απαραίτητους παράγοντες. Στη συνέχεια απαραίτητο είναι ένα κατάλληλο λογισμικό, σχεδιασμένο να αναπαριστά το μοντέλο και να παράγει όλα τα απαραίτητα δεδομένα. Στο πεδίο αυτό προσπαθεί το simmate να φανεί χρήσιμο. Ένα τρίτο βήμα είναι η παραγωγή συμπερασμάτων από τα δεδομένα του προηγούμενου βήματος. Εν κατακλείδει, κρίνεται απαραίτητη η αξιολόγηση της όλης μεθόδου.

Το simmate είναι ένα εύκολο στη χρήση πρόγραμμα που επιχειρεί να δώσει στους χρήστες μία μικρή ιδέα γύρω από την προσομοίωση αλλά και τη θεωρία ουρών. Δεν είναι ένα πρόγραμμα φτιαγμένο για την αγορά, αλλά για σπουδαστές που θέλουν να πειραματιστούν με τα θέματα αυτά χωρίς μεγάλη αρχική προσπάθεια. Δεν προσπαθεί να γίνει ανταγωνιστικό, αλλά να παρέχει λίγες, αλλά ευδιάκριτες λειτουργίες με δυνατότητα όμως μελλοντικής ανάπτυξης

Η παρούσα πρώτη έκδοση, στοχεύει στη γραφική αναπαράσταση συστημάτων ουρών, δείχνοντας οπτικά όλες τις διαδικασίες. Περιέχει όλες τις απολύτως αναγκαίες οντότητες, όπως εξυπηρετητές, πελάτες, γεννήτριες και δρομολογητές, και προσπαθεί να είναι κατά το δυνατόν πλήρης όσον αφορά τα αποτελέσματα που παράγει. Εφόσον προορίζεται κυρίως για σπουδαστές και όχι για επαγγελματίες, ο χρήστης δεν είναι υποχρεωμένος να προγραμματίσει το μοντέλο, αλλά να το σχεδιάσει με τη βοήθεια του ποντικιού του στην οθόνη, χρησιμοποιώντας όλες τις συμβάσεις που είναι συνηθισμένος να χρησιμοποιεί σε άλλα προγράμματα που έχει χειριστεί.

ABSTRACT

When someone simulates a system, he/she imitates the operation of a real system, aiming at investigating possible improvements and discovering the effects of different policies on the real system. Computer simulation involves experimentation on a computer - based model of the system. The model is used as a vehicle for experimentation. To do so a series of maneuver is needed. First of all a model of the real system needs to be built. This model will be an abstraction of the real model containing all the important factors needed. Secondly, software designed for representing the model and producing the necessary data, the field where simmate tries to be useful. Afterwards conclusions of the data from the previous step have to be made. A validation of the method is in conclusion essential.

Simmate is, an easy to use, little program, which attempts to give its users a clue about simulation and queuing theory. It's not a program made for the market rather for students who want to experiment themselves for a while. It doesn't try to be competitive rather than an easy to use program with little but distinct capabilities and potentiality for further development.

Its current first version is aiming at graphical representation of queue systems showing visually all processes. It contains all indispensable entities like servers, customers, generators, queues and routers and trying to be complete concerning the data produced. Since it is appropriated mainly for students, not for professionals, the user isn’t bounded to program the model, rather than draw it with the computer mouse in the screen, using all the conventions he is used to by other programs.

ΕΥΧΑΡΙΣΤΙΕΣ

Η εκπόνηση της παρούσας διπλωματικής εργασίας θα ήταν αδύνατη χωρίς τη συνδρομή ορισμένων ανθρώπων, στους οποίους οφείλω ειλικρινείς ευχαριστίες.

Κατ' αρχήν θα ήθελα να ευχαριστήσω τον καθηγητή μου κ. Ανδρέα Γεωργίου, γιατί, εκτός από την πολύτιμη βοήθεια και καθοδήγησή του, ο τρόπος που δίδαξε το μάθημά του, με έκανε να αγαπήσω αυτόν τον τομέα της επιστήμης.

Ευχαριστώ θερμά την αδερφή μου, Κλημεντίνη Κοντογιάννη, αλλά και το συνάδελφό μου, Στυλιανό Ζαπρούδη, για τη φιλολογική επιμέλεια στο ελληνικό κομμάτι, καθώς και τη συνάδελφο μου Μαρία Ζιγρικά για το αντίστοιχο αγγλικό.

Ευχαριστώ επίσης τον φίλο μου Γεώργιο Τσίτρα για το χρόνο που ανιδιοτελώς αφιέρωσε για το λογότυπο της εφαρμογής και τις ιδέες του σε θέματα παρουσίασης. Αν και πολλές από αυτές δεν υλοποιήθηκαν, υπήρξαν στο σύνολό τους άκρως εποικοδομητικές.

Ουσιαστικό κίνητρο για την ενασχόλησή μου με τον προγραμματισμό υπήρξε ο Bruce Eckel, στα βιβλία του οποίου ανέτρεχα καθ' όλη τη διάρκεια των σπουδών μου, αλλά και κατά την εκπόνηση αυτής της εργασίας.

Ευχαριστώ πολύ τους γονείς μου, Κωνσταντία και Αθανάσιο Κοντογιάννη για όλα τα υπόλοιπα.

ΠΕΡΙΕΧΟΜΕΝΑ

ΔΗΛΩΣΗ **4**

ΠΕΡΙΛΗΨΗ **5**

ABSTRACT **6**

ΕΥΧΑΡΙΣΤΙΕΣ **7**

ΠΕΡΙΕΧΟΜΕΝΑ **8**

1. ΣΥΣΤΗΜΑΤΑ ΟΥΡΩΝ ΚΑΙ ΠΡΟΣΟΜΟΙΩΣΗ **12**

1.1 Συστηματα ουρων 12
1.1.1 Πληθυσμός πελατών (customer population) *13*
1.1.2 Πρότυπο αφίξεων (arrival pattern) *14*
1.1.3 Πειθαρχία ουράς (queueing discipline) *14*
1.1.4 Χρόνοι εξυπηρέτησης (service times) *16*
1.1.5 Έξοδος *17*
1.1.6 Σύστημα συμβολισμού ουρών (Kendall notation) *17*
1.2 Προσομοίωση (Simulation) 18
1.2.1 Πρόοδος χρόνου (Time – advance) *21*
1.2.1.1 Μηχανισμός σταθερού διαστήματος (Fixed time increment) 21
1.2.1.2 Μηχανισμός επόμενου γεγονότος (Variable time increment) 22
1.2.2 Διακριτά και συνεχή συστήματα *23*
1.2.2.1 Συνεχής αλλαγή (Continuous change) 23
1.2.2.2 Διακριτή αλλαγή (Discrete Change) 24
1.2.3 Λίστα επόμενων γεγονότων (Future Event List) *24*
1.2.4 Στοχαστικοί ή προσδιοριστικοί χρόνοι *31*
1.2.4.1 Προσδιοριστικοί χρόνοι (deterministic durations) 31
1.2.4.2 Στοχαστικοί χρόνοι (Stochastic durations) 31
1.2.5 Ορισμοί στη προσομοίωση διακριτού γεγονότος. *32*
1.2.5.1 Σύστημα (System) 32
1.2.5.2 Μοντέλο (Model) 32
1.2.5.3 Κατάσταση συστήματος (System state) 32
1.2.5.4 Οντότητα (Entity) 33
1.2.5.5 Ιδιότητες (Attributes) 33
1.2.5.6 Ομάδα (Set) 33
1.2.5.7 Γεγονός (Event) 33
1.2.5.8 Δραστηριότητα (Activity) 34
1.2.5.9 Καθυστέρηση (Delay) 34
1.3 Τυχαίοι αριθμοί 34
1.3.1 Παραγωγή τυχαίων αριθμών *34*
1.3.1.1 Μέθοδοι παραγωγής τυχαίων αριθμών 34
1.3.1.2 Γεννήτριες τυχαίων αριθμών (random number generators) 35
1.3.2 Στατιστικές Κατανομές *36*
1.3.2.1 Εκθετική κατανομή (exponential distribution) 38
1.3.2.2 Ομοιόμορφη κατανομή (uniform distribution) 39
1.3.2.3 Τριγωνική κατανομή (Triangular distribution) 40
1.3.2.4 Κατανομή Weibull (Weibull distribution) 41
1.3.2.5 Εμπειρικές κατανομές (Empirical distributions) 41
1.3.2.6 Διωνυμική κατανομή (Binomial distribution) 42
1.3.2.7 Κατανομή Poisson (Poisson distribution) 46
1.3.2.8 Κανονική κατανομή (Normal distribution) 46

2. Η ΕΦΑΡΜΟΓΗ SIMMATE ΚΑΙ Η ΥΛΟΠΟΙΗΣΗ ΤΗΣ **48**

2.1 Γενικά 48
2.2 Μηχανισμός επομενου γεγονοτος στο simmate 48
2.3 Παραγωγη τυχαιων αριθμων 51
2.3.1 Ο σπόρος (seed) *51*
2.3.2 Κατανομές *51*
2.3.2.1 Εκθετική κατανομή 53

2.3.2.2 Ομοιόμορφη κατανομή 53
2.3.2.3 Τριγωνική κατανομή 54
2.3.2.4 Κατανομή weibull 54
2.3.2.5 Εμπειρικές κατανομές 55
2.3.2.6 Διωνυμική κατανομή 56
2.3.2.7 Κατανομή Poisson 56
2.3.2.8 Κανονική κατανομή 57
2.3.2.9 Σταθερή κατανομή 58
2.4 ΑΝΤΙΚΕΊΜΕΝΑ ΣΤΟ SIMMATE 58
2.4.1 Μετρητής (Count) *65*
2.4.2 Έξοδος (Exit) *65*
2.4.3 Γεννήτρια (Generator) *65*
2.4.4 Εξυπηρετητής (Server) *66*
2.4.5 Ουρά (Queue) *68*
2.4.6 Δρομολογητής (Routing) *69*
2.4.7 Παρτίδα (Batch) *71*
2.4.8 Πηγή (Resource) *71*
2.4.9 Σκανδάλη (Trigger) *71*
2.4.10 Προσθήκη νέων αντικειμένων *72*
2.4.10.1 Δημιουργία αντικειμένου χωρίς ιδιότητες 72
2.4.10.2 Προσθήκη ιδιοτήτων στο αντικείμενο. 75
2.4.10.3 Δημιουργία βασικού μενού ιδιοτήτων. 76
2.4.10.4 Προσθήκη στατιστικών 77
2.4.10.5 Δημιουργία μενού στατιστικών. 77
2.4.10.6 Υλοποιήση λειτουργιών προσομοίωσης. 79
2.5 ΥΛΟΠΟΊΗΣΗ ΛΕΙΤΟΥΡΓΙΩΝ ΔΙΕΠΑΦΉΣ 81
2.5.1 Πλαίσια ιδιοτήτων *81*
2.5.1.1 Γενικό πλαίσιο ιδιοτήτων 81
2.5.1.2 Πλαίσιο ιδιοτήτων πελάτη 84
2.5.1.3 Πλαίσιο ιδιοτήτων επόμενου γεγονότος 85
2.5.1.4 Πλαίσιο βοήθειας εφαρμογής 85
2.5.2 Χειρισμός μοντέλου *87*
2.5.2.1 Δημιουργία μηχανισμού Undo / Redo 87
2.5.2.2 Λειτουργία Cut – Copy - Paste 88
2.5.2.3 Μετακίνηση αντικειμένων 89
2.5.3 Ανάδραση *89*
2.5.4 Κίνηση *90*
2.6 ΤΟ ΣΥΣΤΗΜΑ ΑΡΧΕΙΩΝ 91
2.6.1 Μορφή Αποθήκευσης Αρχείων *94*

3. ΕΠΙΛΟΓΟΣ 96

3.1 ΧΡΉΣΗ ΤΗΣ ΕΦΑΡΜΟΓΉΣ 96
3.2 ΠΑΡΑΔΕΊΓΜΑΤΑ 96
3.2.1 Οδοντιατρική κλινική Καρδίτσας *96*
3.2.1.1 Το σύστημα 96
3.2.1.2 Το μοντέλο 97
3.2.2 Fast Food Nation *98*
3.2.2.1 Το σύστημα 99
3.2.2.2 Το μοντέλο 99
3.3 ΣΥΜΠΕΡΆΣΜΑΤΑ - ΜΕΛΛΟΝΤΙΚΉ ΕΡΓΑΣΙΑ 100

ΠΑΡΑΡΤΗΜΑ 103

1. SIMMATE ENVIRONMENT (ΤΟ ΠΕΡΙΒΑΛΛΟΝ ΤΟΥ SIMMATE) 103
1.1 Help Introduction (Εισαγωγή στη βοήθεια) *103*
1.1.1 Help Contents 103
1.1.2 Help Index 104
1.1.3 Help Toolbar 105
1.2 Getting started - Building a simple model (Ξεκινώντας - Δημιουργώντας ένα απλό μοντέλο) *106*
1.2.1 The system (Το σύστημα) 106
1.2.2 The model (Το μοντέλο) 107
1.2.3 Running the model (Εκτελώντας το μοντέλο) 108
1.3 Building Models (Δημιουργία μοντέλων) *109*
1.3.1 Building Objects (Δημιουργία αντικειμένων) 109

1.3.2 Building Connections (Δημιουργία συνδέσεων) 110
1.4 Simulation Time (Χρόνος προσομοίωσης) *110*
1.5 Managing simmate objects (Χειρισμός αντικειμένων εφαρμογής) *111*
1.5.1 Counter (Μετρητής) 112
1.5.2 Exit (Εξοδος) 113
1.5.3 Generator (Γεννήτρια) 113
1.5.4 Server (Εξυπηρετητής) 113
1.5.5 Queue (Ουρά) 114
1.5.6 Routing (Δρομολογητής) 114
1.5.7 Batch (Επεξεργαστής παρτίδας) 115
1.5.8 Resource (Πηγή) 116
1.5.9 Trigger (Σκανδάλη) 116
1.6 Control features (Χαρακτηριστικά ελέγχου) *116*
1.6.1 Selecting, moving and coping objects (Επιλογή, μετακίνηση και αντιγραφή αντικειμένων) 116
1.6.2 Undo – Redo mechanism (Μηχανισμός αναίρεσης) 116
1.7 Using Dialogues (Χρησιμοποίηση διαλόγων) *117*
1.7.1 Zoom (Εστίαση) 117
1.7.2 Find (Εύρεση) 117
1.7.3 Background (Φόντο) 118
1.7.4 File open (Άνοιγμα αρχείου) 118
1.7.5 File save (Αποθήκευση αρχείου) 119
1.7.6 Distribution menu (Επιλογές κατανομών) 120
1.7.7 Simulation Menu (Επιλογές προσομοίωσης) 122
1.7.8 Future Event List Panel (Πλαίσιο επόμενου γεγονότος) 123
1.7.9 Customer Panel (Πλαίσιο πελάτη) 124
1.7.10 Standard Toolbar (Τυπική Εργαλειοθήκη) 124
1.8 Keyboard Shortcuts (Συντομεύσεις πληκτρολογίου) *125*
1.9 Menu reference *126*
1.9.1 File Menus 126
3.9.2 Edit Menus 126
1.9.3 Library Menu 127
1.9.4 Run Menu 127
1.9.5 View Menu 127
1.9.6 Help Menu 128
1.10 Examples (Παραδείγματα) *128*
1.10.1 Fast Food Nation (Το έθνος της ταχυφαγείας) 128
1.10.1.1 The system (Το σύστημα) 128
1.10.1.2 The model (Το μοντέλο) 129
ΒΙΒΛΙΟΓΡΑΦΙΑ 131
ΔΙΕΥΘΥΝΣΕΙΣ ΣΤΟ ΔΙΑΔΙΚΤΥΟ 133

1. ΣΥΣΤΗΜΑΤΑ ΟΥΡΩΝ ΚΑΙ ΠΡΟΣΟΜΟΙΩΣΗ

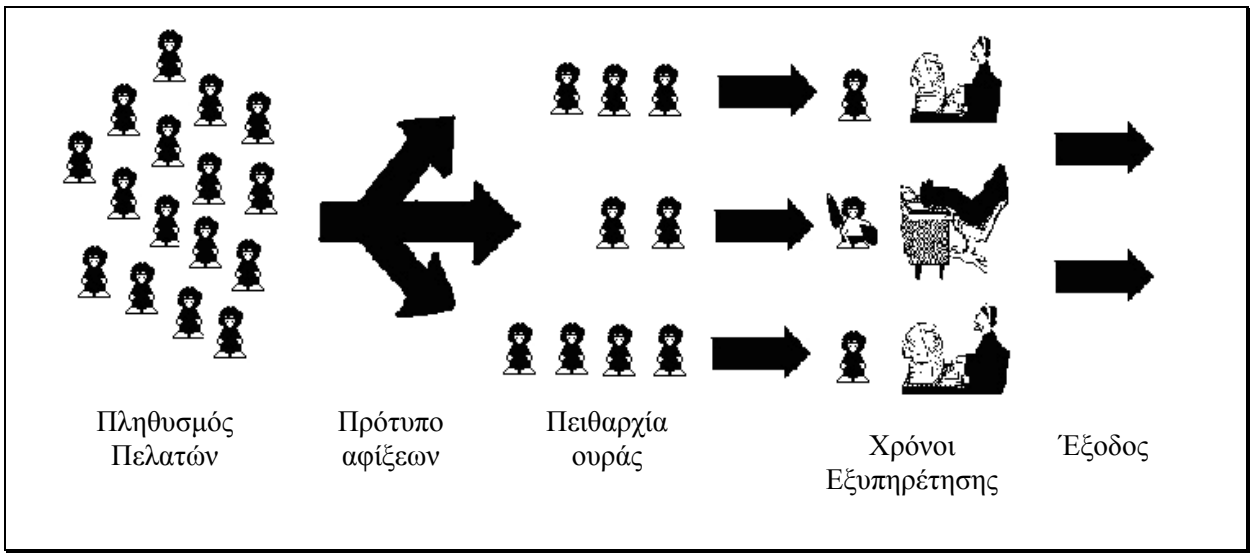
1.1 Συστήματα ουρών

Ο διαμοιρασμός των πόρων και η αναμονή σε ουρές είναι ένα φαινόμενο που συναντάται σε κάθε τομέα της σύγχρονης ζωής. Από την ουρά στο μπακάλικο της γειτονιάς μέχρι τις ουρές των πακέτων στους δρομολογητές του διαδικτύου, η ανάλυση και πρόβλεψη της συμπεριφοράς τους έχει αποκτήσει τεράστια σημασία. Η καλύτερη κατανομή των πόρων, η οποία, για παράδειγμα, θα μπορούσε να οδηγήσει σε μείωση του μέσου όρου αναμονής σε ουρά σε μία μεγάλη τράπεζα, αποτελεί ζητούμενο και μπορεί να αποβεί χρυσοφόρα. Σημαντικά παραδείγματα χαρακτηριστικών λειτουργίας ενός συστήματος ουρών όπως ο μέσος όρος αναμονής δίνονται στη συνέχεια[Anderson, Sweeney and Williams, 2003].

- Η πιθανότητα καμία μονάδα - πελάτης να μην υπάρχει στο σύστημα.
- Ο μέσος όρος μονάδων στην ουρά αναμονής.
- Ο μέσος όρος μονάδων στο σύστημα (ο αριθμός των μονάδων που αναμένουν και ο αριθμός των μονάδων που εξυπηρετούνται.
- Ο μέσος χρόνος που μια μονάδα ξοδεύει στην αναμονή.
- Ο μέσος χρόνος που μια μονάδα ξοδεύει στο σύστημα (στην αναμονή και στην εξυπηρέτηση).
- Η πιθανότητα ότι μια αφιχθείσα μονάδα πρέπει να περιμένει για εξυπηρέτηση.
- Η εκμετάλλευση των εξυπηρετητών του συστήματος.

Ο μελετητής κάποιου συστήματος που εμπεριέχει ουρές θα πρέπει να μελετήσει αυτά τα χαρακτηριστικά για να πάρει αποφάσεις που αφορούν το σύστημά του. Αυτές μπορεί για παράδειγμα να αφορούν την αύξηση ή μείωση εξυπηρετητών στο σύστημα αν για παράδειγμα ο μέσος όρος μονάδων στο σύστημα κρίνεται μεγάλος ή η εκμετάλλευση των ακριβών εξυπηρετητών κρίνεται μικρή.

Η μελέτη αυτού του κλάδου της επιχειρησιακής έρευνας, είχε ως αποτέλεσμα την ανάπτυξη της θεωρίας των ουρών, η οποία ξεκίνησε από τον A.K. Erlang [Brockmeyer, 1948], στον οποίο βασίζει και πολλές από τις αρχές της. Στο επόμενο σχήμα παρουσιάζεται η γενική δομή ενός συστήματος ουράς, με βάση την οποία θα γίνει και η ανάλυση των βασικών συστατικών που αποτελούν ένα τέτοιο σύστημα [Tanner, 1995].



Υπάρχει ένας πληθυσμός πελατών οι οποίοι είναι πιθανοί πελάτες στο σύστημα. Φτάνουν σε αυτό σύμφωνα με κάποιο πρότυπο αφίξεων και, εάν βρουν όλους τους εξυπηρετητές απασχολημένους, τότε περιμένουν σε ουρά αναμονής έως ότου επιλεγούν σύμφωνα με κάποια πειθαρχία ουράς. Από τη στιγμή που θα επιλεγεί, εξυπηρετείται για κάποιο χρονικό διάστημα που καθορίζεται και αυτό από πρότυπο, και εξέρχεται του συστήματος, είτε ξαναμπαίνοντας στον πληθυσμό πελατών, είτε βγαίνοντας γενικώς από το περιβάλλον του συστήματος.

1.1.1 Πληθυσμός πελατών (customer population)

Ο πληθυσμός των πελατών είναι ο αριθμός των πιθανών πελατών στο προς μελέτη σύστημα ουράς. Χρησιμοποιώντας το παράδειγμα ενός κινηματογράφου σε ένα μικρό χωριό, θα μπορούσε κάποιος να υποθέσει ότι ο πληθυσμός αυτός είναι το σύνολο των κατοίκων του χωριού. Σημαντικό παράγοντα αποτελεί η υπόθεση ότι ο πληθυσμός έξω από το σύστημα είναι τόσο μεγάλος, σε σύγκριση με τον πληθυσμό μέσα σε αυτό, ώστε να θεωρηθεί άπειρος ή όχι. Στην περίπτωση που δε μπορεί να θεωρηθεί άπειρος, το σύνολο των πελατών που είναι ήδη στο σύστημα επηρεάζει το ρυθμό των αφίξεων νέων πελατών.

Για παράδειγμα, ας υποθέσουμε ότι σε ένα εργοστάσιο υπάρχουν τρία μηχανήματα συγκεκριμένου τύπου, τα οποία χρειάζονται συντήρηση από κάποιον τεχνικό ανά τακτά και συγκεκριμένα χρονικά διαστήματα. Η πιθανότητα άφιξης νέου πελάτη (μηχάνημα) στο σύστημα (ουρά - τεχνικός), εξαρτάται κατά πολύ από το εάν υπάρχει ήδη κάποιο μηχάνημα για συντήρηση. Το σύστημα αυτό αναφέρεται και ως κλειστό. Από την άλλη πλευρά, ο αριθμός των πιθανών επιβατών σε μία αεροπορική εταιρεία, ή των συνδρομητών μίας εταιρείας κινητής τηλεφωνίας που θα κάνουν κάποια κλήση, είναι τόσο μεγάλος που μπορεί να θεωρηθεί άπειρος σε σχέση με αυτούς που ήδη πετάνε ή όσους μιλάνε ήδη στο τηλέφωνο. Το σύστημα αυτό αναφέρεται και ως ανοιχτό. Γενικά, στην περίπτωση που ο πληθυσμός μπορεί να θεωρηθεί άπειρος (infinite calling population, οι υπολογισμοί είναι ευκολότεροι, ενώ, στην αντίθετη περίπτωση πεπερασμένου πληθυσμού (finite calling population), κάθε νέα άφιξη ή αναχώρηση από το προς μελέτη σύστημα αλλάζει και τον τρόπο υπολογισμού της νέας άφιξης. Άπειρος μπορεί να θεωρηθεί ο πληθυσμός των πελατών, όταν ο αριθμός τους μέσα στο σύστημα, στο οποίο μπορεί να εξυπηρετούνται ή να περιμένουν, είναι ένα πολύ μικρό ποσοστό του συνόλου των πιθανών πελατών. Συνήθως

ενώ στην πρώτη περίπτωση άπειρου πληθυσμού η υπόθεση για το πρότυπο αφίξεων προκύπτει όπως θα συζητηθεί στην παράγραφο που ακολουθεί από διάφορα πρότυπα που προσεγγίζουν την άφιξη των πελατών γενικά, στην περίπτωση πεπερασμένου πληθυσμού το πρότυπο αφίξεων ορίζεται με βάση κάθε πελάτη και την πιθανότητά του να εισέλθει στο σύστημα..

1.1.2 Πρότυπο αφίξεων (arrival pattern)

Στο μεγαλύτερο ποσοστό των συστημάτων ουρών (που αποτελεί και το ποσοστό του οποίου η μελέτη έχει νόημα) οι πελάτες έρχονται τυχαία. Βέβαια, αυτό το «τυχαία» δεν είναι τόσο τυχαίο στην θεωρία των ουρών. Υπάρχουν πολύ συγκεκριμένα πρότυπα που μπορούν να προσομοιώσουν μαθηματικά αυτό το «τυχαία». Η κατά Poisson διαδικασία άφιξης είναι για παράδειγμα ένα τέτοιο πρότυπο που μπορεί να προσομοιώσει με μεγάλη ακρίβεια αφίξεις σε ένα εστιατόριο ή μία τράπεζα ή ακόμα κλήσεις σε ένα τηλεφωνικό δίκτυο. Όλα αυτά τα πρότυπα προσομοιώνουν συγκεκριμένα το χρόνο που παρεμβάλλεται μεταξύ δυο διαδοχικών αφίξεων. Παραδείγματα τέτοιων πρότυπων, καθώς και ο τρόπος που αυτά μας βοηθούν να μελετήσουμε το σύστημα ουρών, αναπτύσσονται σε επόμενη παράγραφο.

Ακόμη ένα σημαντικό μέρος της μελέτης των αφίξεων που την κάνει πιο δύσκολη είναι οι διάφορες συνήθειες των πελατών [Οικονόμου και Γεωργίου, 2005]. Οι πελάτες υπάρχει περίπτωση να αποθαρρύνονται από κάποιες συγκεκριμένες καταστάσεις του συστήματος (μεγάλες ουρές) και να μην εισέρχονται καν στο σύστημα (balking) αλλά και να εισέρχονται, αλλά, λόγω της μεγάλης αναμονής, να αποχωρούν (reneging). Επιπλέον, σε περιπτώσεις πολλών ουρών υπάρχει η περίπτωση εναλλαγής από την μία ουρά στην άλλη(switching – jockeying). Πολλοί πελάτες μπορούν, επίσης, να έρχονται κατά ομάδες, όπως στο παράδειγμα του κινηματογράφου του χωριού, όπου έρχονται σε παρέες, ή μεμονωμένα. Πολύ διαφορετικός είναι και ο τρόπος αντιμετώπισης του τρόπου αφίξεων όταν ο πληθυσμός των πελατών δε μπορεί να θεωρηθεί άπειρος.

Σε κάθε περίπτωση, οι παραπάνω συμπεριφορές είναι πολύ δύσκολο να προβλεφθούν και να μελετηθούν. Όταν ο πελάτης του συστήματος είναι άνθρωπος και όχι κάποιο αντικείμενο υπεισέρχονται παράγοντες ψυχολογίας του πελάτη και άρα μη μετρήσιμοι με μαθηματικούς όρους. Για παράδειγμα, φαινόμενα αποχώρησης μπορεί να συμβούν πιο συχνά σε μία αναμονή στοχαστικής διάρκειας όπου ο πελάτης δε γνωρίζει πόσο χρόνο θα κρατήσει ή σε μία αναμονή που δεν είναι αυστηρή η πειθαρχία ουράς, αλλά πελάτες που έπονται, εξυπηρετούνται πρώτοι. Μία διαδικασία που θα απαιτήσει διαδοχικές εξυπηρετήσεις και ουρές μπορεί να κρατήσει τον πελάτη πιο πολύ εάν βρίσκεται στη μέση της διαδικασίας, αλλά και γενικά η διαδικασία της εξυπηρέτησης, έστω και αν κρατάει περισσότερο από την αναμονή, κάνει τους πελάτες πιο υπομονετικούς. Πολλοί παράγοντες όπως οι προαναφερθέντες δυσχεραίνουν το μελετητή ενός συστήματος ουρών και καθιστούν την αυστηρή μαθηματική μελέτη του πρακτικά αδύνατη και την προσομοίωση που θα αναφερθεί στη συνέχεια επιβεβλημένη.

1.1.3 Πειθαρχία ουράς (queueing discipline)

Το στοιχείο της πειθαρχίας στην περιγραφή ενός συστήματος ουρών αναφέρεται στη μέθοδο, με την οποία επιλέγεται ο επόμενος πελάτης από ένα πλήθος πελατών που περιμένουν στην ουρά. Έτσι, η πιο απλή και «δίκαιη» μέθοδος είναι η μέθοδος *First in First out* (FIFO), ή αλλιώς *Fist Come First Served* (FCFS), κατά την οποία ο πελάτης που έχει

εισέλθει πρώτος στην ουρά, από το σύνολο των πελατών που περιμένουν, είναι και πρώτος που θα βγει από αυτή. Παραδείγματα τέτοιων ουρών συναντώνται σε κάθε υπηρεσία, ταμείο, κ.λπ. Η αντίστροφη μέθοδος είναι η *Last in First Out* (LIFO), σύμφωνα με την οποία ο τελευταίος που θα μπει στην ουρά είναι ο πρώτος που θα βγει. Παράδειγμα τέτοιας ουράς είναι μία στοίβα με πιάτα. Το τελευταίο από τα πιάτα που έχει τοποθετηθεί στη στοίβα είναι και το πρώτο που θα πάρουμε από αυτή. Άλλη μέθοδος είναι η επιλογή του επόμενου πελάτη στην τύχη. Τέλος, οι πελάτες μπορούν να επιλεγούν κατά προτεραιότητα, όπως, για παράδειγμα, η επιλογή των αποσκευών των επιβατών business class κατά προτεραιότητα από αυτές των επιβατών της economy class. Στην περίπτωση αυτή υπάρχουν δυο διαφορετικές εκδοχές. Πρώτη εκδοχή, η απλή προτεραιότητα ή προτεραιότητα χωρίς διακοπή (non preemptive), στην οποία, αφού τελειώσει η εξυπηρέτηση του πελάτη επιλέγεται ο πελάτης με την υψηλότερη προτεραιότητα από αυτούς που βρίσκονται στην ουρά αναμονής. Δεύτερη εκδοχή, η απόλυτη προτεραιότητα ή προτεραιότητα με διακοπή (preemptive), κατά την οποία, όταν ένας πελάτης εισέρχεται στην ουρά αναμονής, εάν εξυπηρετείται κάποιος πελάτης με χαμηλότερη προτεραιότητα, διακόπτεται και αρχίζει η δική του εξυπηρέτηση. Συνοπτικά, λοιπόν, η επιλογή των πελατών από την ουρά αναμονής γίνεται κυρίως με έναν από τους παρακάτω τρόπους

- first-come-first-served (FCFS) ή first-in-first-out (FIFO)
- last-come-first-served (LCFS) ή last-in-first-out (LIFO)
- priority
    - προτεραιότητα χωρίς διακοπή (non preemptive)
    - προτεραιότητα με διακοπή (preemptive)
- service in random order (SIRO) ή απλά random

Υπάρχουν, ακόμη, ουρές, στις οποίες σαν επόμενος προς εξυπηρέτηση πελάτης επιλέγεται αυτός που απαιτεί τον μικρότερο (ή και τον μεγαλύτερο) χρόνο εξυπηρέτησης, ή αυτός με τη μικρότερη διορία αποπεράτωσης, γεγονός που καθιστά τη μελέτη τους πιο πολύπλοκη διαδικασία. Παράμετροι που μπορούν να περιπλέξουν ακόμη περισσότερο τη διαδικασία, και συμβαίνουν συνήθως όταν οι πελάτες είναι άνθρωποι, είναι η περίπτωση που ο πελάτης, αφού περιμένει για λίγο στην ουρά, αποφασίσει να φύγει ή η αλλαγή της κατά τη διάρκεια της αναμονής σε αναζήτηση ταχύτερης εξυπηρέτησης που αναφέρθηκαν και στην προηγούμενη παράγραφο.

Επιπλέον χαρακτηριστικό μέγεθος στην ουρά είναι το μέγεθός της που μπορεί να θεωρηθεί άπειρο ή πεπερασμένο με συγκεκριμένη χωρητικότητα. Όταν η χωρητικότητα αυτή καλυφθεί, το σύστημα δε θα μπορεί να δεχθεί επιπλέον πελάτες (blocked). Υπάρχουν πεδία εφαρμογής, στα οποία ένα πεπερασμένο σύστημα και η απώλεια ή όχι πελατών λόγω μίας γεμάτης ουράς είναι άμεσου ενδιαφέροντος. Έτσι, στο παράδειγμα του τηλεφωνικού δικτύου, όταν το σύστημα είναι απασχολημένο και δε μπορεί να εξυπηρετήσει κάποια κλήση, με συνέπεια αυτή είτε να χάνεται, είτε να καθυστερεί, η πιθανότητα να συμβεί αυτό και ο χρόνος αναμονής αντίστοιχα είναι σημαντικότητα μεγέθη για την εταιρεία κινητής τηλεφωνίας [Hock, 1996].

Ακραία περίπτωση πεπερασμένης ουράς σε ένα σύστημα αποτελεί το σύστημα με μηδενική χωρητικότητα ουράς. Στο σύστημα αυτό ένας εισερχόμενος πελάτης που θα βρει όλους τους εξυπηρετητές απασχολημένους, δε θα εισέλθει στο σύστημα από το οποίο θεωρείται χαμένος. Το συγκεκριμένο σύστημα είναι από τα πιο εκτενώς μελετημένα στη

βιβλιογραφία και παρουσιάστηκε πρώτη φορά το 1917 από τον Erlang [Erlang 1917] σε μελέτη του για την τηλεφωνία και προς τιμή του ονομάζεται Erlang loss system.

Παράγοντας που μπορεί να επηρεάσει ιδιαίτερα το πλήθος των πελατών στην ουρά αναμονής αλλά και γενικότερα όλο το σύστημα, είναι η κατάσταση στην οποία βρίσκεται. Έτσι, είναι πολύ πιθανό κατά την έναρξη της λειτουργίας του το σύστημα να επηρεάζεται από κάποιους παράγοντες που δεν θα το επηρεάζουν στη συνέχεια και θα κάνουν και τις μετρήσεις δυσκολότερες αλλά και μη χρήσιμες για την κανονική λειτουργία του συστήματος. Η κατάσταση αυτή ονομάζεται και μεταβατική κατάσταση (transient state) και παράδειγμά της είναι μία τράπεζα που μόλις ανοίγει σε μέρα πληρωμής συντάξεων, με δεκάδες συνταξιούχους να περιμένουν έξω. Η ουρά αναμονής απευθείας γεμίζει με δεκάδες πελάτες και θα περάσει αρκετή ώρα μέχρι αυτοί να εξυπηρετηθούν και να έλθει η τράπεζα στην ομαλή της λειτουργία. Η λειτουργία αυτή ονομάζεται κατάσταση ισορροπίας (steady state) και συμπεριφορά του συστήματος δεν εξαρτάται από τις αρχικές συνθήκες που επικρατούν κατά την έναρξη της λειτουργίας του. Συνήθως, για να φτάσει το σύστημα σε κατάσταση ισορροπίας, πρέπει να παρέλθει κάποιο χρονικό διάστημα από την εκκίνησή του, στη διάρκεια του οποίου θα εξαλειφθούν οι συνθήκες που το επηρέαζαν κατά την εκκίνησή του. Το χρονικό αυτό διάστημα ποικίλει από πρακτικά μηδενικό, στη περίπτωση για παράδειγμα ενός τηλεπικοινωνιακού δικτύου με πρακτικά άπειρα πακέτα έως πρακτικά άπειρο για συστήματα που μπορεί ποτέ να μην περιέλθουν σε κατάσταση ισορροπίας. Πολλές είναι οι περιπτώσεις οπού η μεταβατική κατάσταση δεν συμβαίνει στην αρχή της λειτουργίας αλλά και σε κάποιο σημείο κατά τη διάρκειά της. Χαρακτηριστικό τέτοιο παράδειγμα είναι η ώρα κυκλοφοριακής αιχμής σε κάποιο οδικό δίκτυο που συμβαίνει όταν τελειώνει το οκτάωρο ή κλείνουν τα μαγαζιά. Η κατάσταση αυτή δεν μπορεί να χαρακτηριστεί σαν σταθερή αλλά και συμβαίνει κάποιες στιγμές στη διάρκεια λειτουργίας του συστήματος.

1.1.4 Χρόνοι εξυπηρέτησης (service times)

Ένα σημαντικότατο χαρακτηριστικό ενός συστήματος ουράς είναι ο χρόνος που θα κάνει ένας εξυπηρετητής να αντιμετωπίσει έναν πελάτη. Τα άτομα δηλαδή που θα μπορέσει να εξυπηρετήσει στη μονάδα του χρόνου, ή το ρυθμό εξυπηρέτησης. Όπως και στην περίπτωση των αφίξεων, έτσι και εδώ θα μπορούσε ο ρυθμός αυτός να είναι αυστηρά καθορισμένος, και άρα εύκολα μετρήσιμος. Στην πλειονότητα, όμως, των περιπτώσεων θα πρέπει να βρεθεί και πάλι κάποιο μαθηματικό μοντέλο που θα μοντελοποιεί τη συμπεριφορά του εξυπηρετητή. Οι περιπτώσεις αυτές μπορούν να προσομοιωθούν με τον ίδιο τρόπο, όπως και οι χρόνοι αφίξεων, με πιο συχνή περίπτωση την εκθετική κατανομή των χρόνων εξυπηρέτησης. Πρέπει να σημειωθεί στο σημείο αυτό ότι καθώς και στις δυο περιπτώσεις γίνονται υποθέσεις και απλουστεύσεις που απαιτούν κάποια μορφή αξιολόγησης αυτών που θα μπορούσε για παράδειγμα να επιτευχθεί αν υπάρχει το πραγματικό σύστημα με συλλογή δεδομένων από αυτό και σύγκρισή του με το μοντέλο του.

Παράγοντες που ενδιαφέρουν τον μελετητή αναφορικά με το μηχανισμό εξυπηρέτησης, εκτός από το ρυθμό εξυπηρέτησης, είναι οι θέσεις και οι φάσεις εξυπηρέτησης ενός συστήματος. Έτσι, ένα σύστημα ουρών μπορεί να έχει είτε έναν μόνο εξυπηρετητή (single channel), είτε πολλούς εξυπηρετητές (multiple channels) που να δουλεύουν παράλληλα ακόμα και εξυπηρετώντας πελάτες από μία μόνο ουρά. Παράδειγμα τέτοιου συστήματος έχουν συναντήσει οι περισσότεροι σε κάποια τράπεζα ή ταχυδρομείο όπου υπάρχει μία ουρά, της οποίας η πειθαρχία καθορίζεται από κάποιο μηχανισμό που εκδίδει χαρτάκια προτεραιότητας, αλλά πολλούς υπαλλήλους που εργάζονται παράλληλα. Πολλές φορές πάντως, ένας

μοναδικός γρήγορος εξυπηρετητής που έχει την ίδια συνολικά «χωρητικότητα» εξυπηρέτησης με πολλούς εξυπηρετητές μπορεί να έχει εντελώς διαφορετική συμπεριφορά σε ένα σύστημα. Επίσης κάποιος πελάτης μπορεί να έχει ανάγκη είτε από μία και μόνο εξυπηρέτηση (single phase), είτε από πολλές φάσεις εξυπηρέτησης (multiple phases), για κάθε μία από τις οποίες είναι πιθανό να υπάρχει διαφορετική ουρά. Μία επίσκεψη σε κάποια δημόσια υπηρεσία μάλλον θα σας κάνει μάρτυρες μίας τέτοιας περίπτωσης.

1.1.5 Έξοδος

Η έξοδος από το σύστημα ουρών σπάνια παρουσιάζει ενδιαφέρον για τους μελετητές. Στη συντριπτική πλειοψηφία των περιπτώσεων θεωρείται απλά ότι κάθε πελάτης που εξέρχεται από το σύστημα επανέρχεται απευθείας στο περιβάλλον και στον πληθυσμό πελατών. Υπάρχει περίπτωση όμως ο πελάτης να μην επανέρχεται στον πληθυσμό ή να επανέρχεται με κάποια καθυστέρηση. Βέβαια, όταν ο πληθυσμός θεωρείται άπειρος, όποια και αν είναι η συμπεριφορά του εξερχόμενου πελάτη, δε θα επηρεαστεί καθόλου το πρότυπο αφίξεων. Το πρότυπο θα επηρεαστεί όμως πολύ από τέτοιου είδους συμπεριφορές όταν ο πληθυσμός δε θεωρείται άπειρος.

Για παράδειγμα, αυτές οι συμπεριφορές επηρεάζουν το σύστημα σε περιπτώσεις όπως αυτή της συντήρησης των μηχανών ενός εργοστασίου που αναφέρθηκε προηγουμένως. Εάν, για παράδειγμα, αφού τελείωνε η συντήρηση κάποιου μηχανήματος και η μηχανή εξερχόταν από το σύστημα προς μελέτη, αλλά περνούσε και από συνεργείο καθαρισμού, αν και δεν ενδιαφέρει τον μελετητή της συντήρησης εφόσον το συνεργείο καθαρισμού δεν ανήκει στο προς μελέτη σύστημα, θα το εμπόδιζε να ξαναμπεί σε λειτουργία και άρα στον πληθυσμό των πιθανών πελατών του συστήματος. Στην περίπτωση των εκλογών όπου κάθε ψηφοφόρος έχει δικαίωμα να ψηφίσει μία φορά και με πληθυσμό πελατών τους εγγεγραμμένους σε κάποιον εκλογικό κατάλογο, κάθε ψηφοφόρος, αφού φύγει από την κάλπη και το προς μελέτη σύστημα, φεύγει και από τον πληθυσμό πελατών, κάτι που φυσικά θα επηρεάσει τις μετέπειτα αφίξεις.

1.1.6 Σύστημα συμβολισμού ουρών (Kendall notation)

Από τις προηγούμενες παραγράφους μπορεί κανείς να παρατηρήσει μία πληθώρα στοχαστικών διαδικασιών και διαφορετικών παραμέτρων που ορίζουν ένα σύστημα ουρών. Ο David G. Kendall, ένας Βρετανός στατιστικός, από το 1953 ανέπτυξε έναν τρόπο σύντομου συμβολισμού, γνωστό σαν *Kendall notation* που από τότε καθιερώθηκε σαν τρόπος περιγραφής των συστημάτων ουρών. Αρχικά, ο συμβολισμός του περιείχε τρεις παράγοντες, το A/B/C σύστημα συμβολισμού [Kendall, 1953] που στη συνέχεια επεκτάθηκε, για να συμπεριλάβει και άλλα χαρακτηριστικά των συστημάτων ουρών. Ακολουθεί περιγραφή αυτού του συμβολισμού:

A / B / X / Y / Z

- A : Πρότυπο αφίξεων πελατών
- B : Πρότυπο εξυπηρέτησης
- X : Αριθμός εξυπηρετητών
- Y : Χωρητικότητα συστήματος (Μέγιστος αριθμός πελατών που μπορούν να φιλοξενηθούν στο σύστημα)
- Z : Πληθυσμός πελατών

Έτσι για παράδειγμα τα A και B μπορούν να πάρουν κάποιες από τις παρακάτω τιμές ανάλογα με τις κατανομές που προσδιορίζουν τα πρότυπα αφίξεων και εξυπηρέτησης αντίστοιχα.

- M : Εκθετική κατανομή
- D : Εκφυλισμένη ή προσδιοριστική κατανομή
- E_k : Erlang κατανομή (k = παράμετρος shape)
- G : Γενική κατανομή

Έτσι, αν υποθέσουμε τον επόμενο συμβολισμό συστήματος ουρών

M/M/m/K/N

έχουμε ένα σύστημα με άφιξη πελατών που ακολουθούν την εκθετική κατανομή, εκθετική κατανομή για εξυπηρέτηση πελατών, m εξυπηρετητές, ένα μέγιστο K πελατών στο σύστημα και N πιθανούς πελάτες. Πολλές φορές, στη συχνή περίπτωση που ο πληθυσμός πελατών θεωρείται άπειρος, το τελευταίο σύμβολο Z παραλείπεται. Όταν η χωρητικότητα του συστήματος θεωρείται άπειρη, παραλείπεται και το σύμβολο Y.

1.2 Προσομοίωση (Simulation)

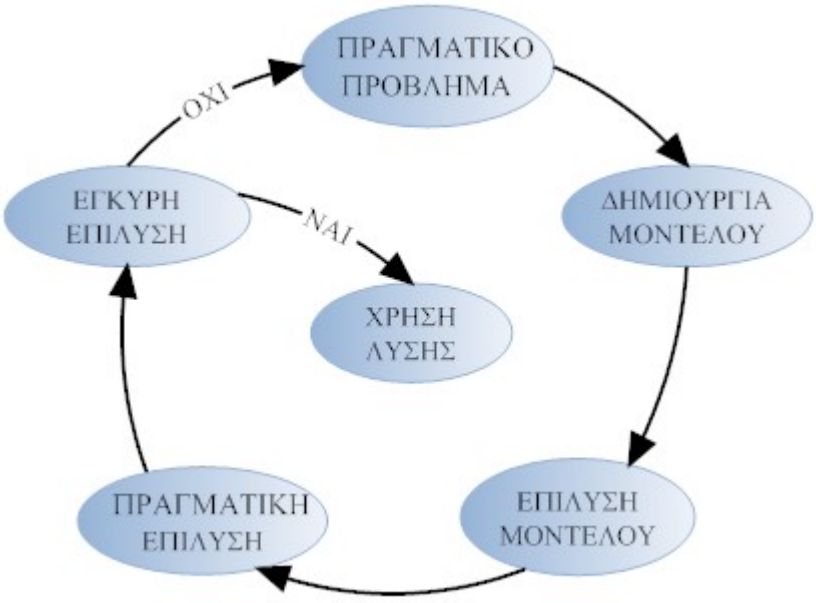
Ένα σύστημα μπορεί να οριστεί σαν μία συλλογή από αλληλεπιδρώντα συστατικά, το οποίο δέχεται δεδομένα και παράγει αποτελέσματα για κάποιο συγκεκριμένο σκοπό [Chung, 2004]. Πολλές φορές μας ενδιαφέρει η μελέτη ενός συστήματος για την κατανόηση των σχέσεων μεταξύ των συστατικών του ή για την πρόβλεψη της συμπεριφοράς του συστήματος κάτω από διαφορετικές συνθήκες. Σε κάποιες από αυτές, το σύστημα επιτρέπει την άμεση μελέτη του, κάτι που δεν είναι όμως πάντα εφικτό για διάφορους λόγους. Ο πειραματισμός με ένα σύστημα μπορεί να είναι αδύνατος ή καθόλου πρακτικός, όταν αυτό είναι υπό κατασκευή, ή ακόμα και αν αυτό υπάρχει. Εύκολα γίνεται αντιληπτό ότι ο πειραματισμός με το σύστημα ουρών σε ένα πολυκατάστημα, αυξομειώνοντας τον αριθμό των ενεργών ταμείων, μπορεί να στοιχίσει στην εταιρεία πολλούς εκνευρισμένους, χαμένους πελάτες και αρκετά χρήματα. Μία αναπαράσταση του συστήματος με σκοπό την μελέτη του θα ήταν στο σημείο αυτό πολύ χρήσιμη. Η αναπαράσταση αυτή καλείται μοντέλο του συστήματος.

Το μοντέλο αυτό μπορεί να είναι φυσικό μοντέλο, όπως, για παράδειγμα, η σμίκρυνση μίας μεγάλης κατασκευής με σκοπό τον πειραματισμό της συμπεριφοράς της κάτω από διαφορετικές συνθήκες, ή μαθηματικό μοντέλο, όπως, για παράδειγμα, η αντικατάσταση των ταμείων και των πελατών της τράπεζας με μία πληθώρα μαθηματικών εξισώσεων. Η μελέτη τέτοιων μοντέλων, υπαρκτών ή φανταστικών, κατά τη διάρκεια του χρόνου, με σκοπό την κατανόηση τους και την αποτίμηση στρατηγικών για τη λειτουργία τους, καλείται προσομοίωση. Αποτελεί φυσικά πειραματική μέθοδο που έχει σαν σκοπό είτε τη βελτιστοποίηση είτε απλώς τη μελέτη της λειτουργίας ενός ήδη υπάρχοντος συστήματος, ή ακόμα και τη βέλτιστη δημιουργία ενός νέου.

Τα πλεονεκτήματα της χρήσης της προσομοίωσης στη μελέτη συστημάτων είναι πολλά και έχουν γίνει εμφανέστερα τα τελευταία χρόνια με την ανάπτυξη που έχει επιτευχθεί στους ηλεκτρονικούς υπολογιστές που έχουν γίνει το κυρίαρχο εργαλείο στη μέθοδο αυτή. Κυρίαρχο πλεονέκτημα της μεθόδου, που μάλιστα λόγω της ανάπτυξης των υπολογιστών έχει ενταθεί, είναι η οικονομία χρόνου που επιτυγχάνεται. Ένα πείραμα που διεξάγεται στον υπολογιστή, δεν είναι απαραίτητο να λειτουργήσει σε πραγματικό χρόνο, αλλά μπορεί να γίνει προσομοίωση αυτού. Προσομοίωση που μπορεί να γίνει όλο και πιο σύντομη όσο γρηγορότερη σε επίπεδα υλικού και λογισμικού γίνεται η υπολογιστική δύναμη.

Όπως ειπώθηκε στο παράδειγμα του πολυκαταστήματος, προσόν της μεθόδου αποτελεί το γεγονός ότι το σύστημα μπορεί να μελετηθεί χωρίς να γίνει καμία, ή ελάχιστη, παρέμβαση σε αυτό. Πολλές φορές μάλιστα, όταν το σύστημα δεν έχει δημιουργηθεί ακόμη αλλά είναι προς μελέτη, είναι και η μόνη μέθοδος μελέτης του. Με τη βοήθεια του υπολογιστικού συστήματος, και από τη στιγμή της δημιουργίας του μοντέλου με σχεδόν μηδαμινό κόστος, είναι δυνατή η επανάληψη του πειράματος πολλές φορές, είτε για ασφαλέστερα πειραματικά αποτελέσματα, είτε για τη δοκιμή διαφορετικών συμπεριφορών, όταν αλλάζουν συγκεκριμένα μέρη του μοντέλου. Σαφές προβάδισμα στις περισσότερες των πολύπλοκων περιπτώσεων παρουσιάζει η προσομοίωση έναντι των αναλυτικών μεθόδων. Στις μεθόδους αυτές ο μελετητής θα πρέπει να αναπαραστήσει κάθε πτυχή του συστήματος με κάποια μαθηματική μέθοδο κάτι που τον αναγκάζει να προβεί σε πολλές παραδοχές, οι οποίες ενδέχεται να απέχουν από την πραγματικότητα. Στην περίπτωση της προσομοίωσης δεν είναι απαραίτητο να γίνουν όλες αυτές οι παραδοχές, καθώς μία αφηρημένη παράσταση των γενικών λειτουργιών του μοντέλου και η παρακολούθηση των αποτελεσμάτων είναι αρκετή για τη μελέτη του συστήματος.

Γενικότερα η μεθοδολογία και τα βήματα της λύσης ενός προβλήματος σε ένα σύστημα με τη βοήθεια κάποιου μοντέλου φαίνονται παραστατικά στο επόμενο σχήμα [Tanner, 1995] που φυσικά εφαρμόζεται και στην περίπτωση της προσομοίωσης. Πρώτο βήμα αποτελεί η κατανόηση του προβλήματος, δεύτερο η δημιουργία μια αφαίρεσης του και τρίτο η επίλυση της αφαίρεσης αυτής. Στη συνέχεια πρέπει να διαπιστωθεί εάν η επίλυση αυτή του μοντέλου έχει λογική για το πραγματικό σύστημα και εάν όχι τότε πρέπει να επαναληφθεί η διαδικασία. Εάν έχει λογική τότε η λύση που βρέθηκε μπορεί πλέον να χρησιμοποιηθεί στο πραγματικό σύστημα.

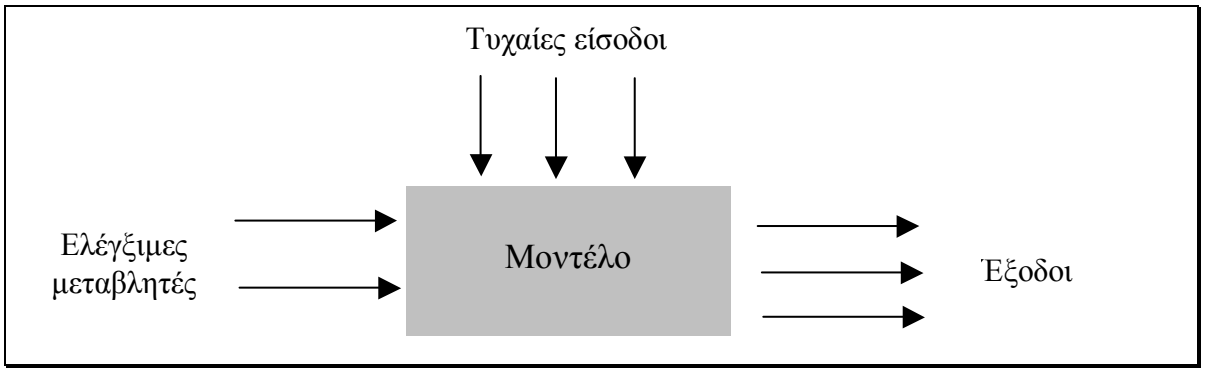


Η αρχικές προσπάθειες του μελετητή πρέπει να επικεντρωθούν στην κατανόηση του πραγματικού προβλήματος. Από τη στιγμή που αυτό έχει κατανοηθεί αρκετά, ακολουθεί η δημιουργία του μοντέλου. Αποφασίζεται ποια στοιχεία του πραγματικού συστήματος θα συμπεριληφθούν σε αυτό και ποια θα παραληφθούν αποφασίζοντας γενικότερα το επιθυμητό επίπεδο λεπτομέρειας. Σε μεγάλο βαθμό, η επιτυχία της όλης μεθόδου εξαρτάται από την πιστότητα του μοντέλου του προς μελέτη συστήματος. Για την πιστότητα αυτή, θα πρέπει να καθοριστούν με προσοχή τα όρια του προς μελέτη συστήματος.

Ένα σύνολο αντικειμένων που αλληλεπιδρά για την επίτευξη ενός σκοπού μπορεί να αποτελέσει το όριο ενός συστήματος που πιθανότατα όμως θα δέχεται κάποιες επιδράσεις από αλλαγές που λαμβάνουν χώρα εκτός των ορίων αυτών. Οι επιδράσεις αυτές λέμε ότι προέρχονται από το περιβάλλον του συστήματος [Gordon, 1978]. Στη δημιουργία μοντέλων θα πρέπει λοιπόν να δοθεί μεγάλη προσοχή στον καθορισμό των ορίων αυτών μεταξύ του προς μελέτη συστήματος και του περιβάλλοντός του. Αυτό το πρώτο βήμα της απόσπασης από το πραγματικό πρόβλημα των σημαντικών και μόνο στοιχείων απαιτεί την περισσότερη ικανότητα και ίσως τέχνη στη προσομοίωση.

Με δημιουργημένο το μοντέλο, θα πρέπει αυτό να επιλυθεί. Η επίλυση στην προσομοίωση πρακτικά σημαίνει την διεξαγωγή πολλών πειραμάτων. Για τη διεξαγωγή ενός πειράματος προσομοίωσης θα πρέπει ο αναλυτής να θέσει στο μοντέλο κάποιες ελέγξιμες εισόδους. Κάποιες άλλες είσοδοι θα παραχθούν τυχαία. Το μοντέλο με τη χρήση αυτών των μεταβλητών, ελέγξιμων ή όχι, θα παράγει κάποια αποτελέσματα. Με την εισαγωγή διαφόρων τιμών στις ελέγξιμες μεταβλητές – εισόδους και την διεξαγωγή, αντίστοιχα, διαφόρων πειραμάτων, ο αναλυτής μπορεί να δει πως οι μεταβλητές αυτές επηρεάζουν το μοντέλο και κατά συνέπεια το σύστημα. Τρόπο μελέτης του συστήματος όμως, κυρίως για ακραίες περιπτώσεις, αποτελεί και ο μερικός έλεγχος των τυχαίων εισόδων Για την επιλογή τους υπάρχουν διάφορες τεχνικές που συνοψίζονται στη συνέχεια[Anderson, Sweeney and Williams, 2003].

- Βασικό σενάριο (base–case scenario): Επιλογή των εισόδων από τις πιο πιθανές τιμές ώστε να μελετηθεί το σύστημα στην συνηθισμένη του συμπεριφορά.
- Σενάριο χειρότερης περίπτωσης (worst-case scenario): Επιλογή των χειρότερων πιθανών εισόδων και μελέτη του συστήματος στο χειρότερο ενδεχόμενο.
- Σενάριο καλύτερης περίπτωσης (best-case scenario): Επιλογή των καλύτερων πιθανών εισόδων και μελέτη του συστήματος στην καλύτερη περίπτωση.



Το βήμα αυτό είναι η φάση όπου ο μελετητής καλείται να κάνει χρήση όλων των γνώσεων του για την προσομοίωση αλλά και για τον τομέα στον οποίο αυτή εφαρμόζεται. Συνήθως είναι το πιο δύσκολο βήμα τεχνικά και το πιο ενδιαφέρον ακαδημαϊκά αλλά φυσικά είναι ζωτικό να ληφθεί υπόψη ότι είναι απλά ένα βήμα και αν τα άλλα βήματα δεν έχουν προσεχθεί και δεν έχουν γίνει σωστά όλη η επίλυση του προβλήματος δεν θα ανταποκρίνεται στη πραγματικότητα.

Μετά την επίλυση του μοντέλου, θα πρέπει η λύση που έχει ο μελετητής πλέον στα χέρια του να μεταφραστεί σε λύση του πραγματικού προβλήματος και να ελεγχθεί για την λογική της βάση σε αυτό. Πολύ πιθανό είναι να ανακαλυφθεί ότι ένα σημαντικό συστατικό του συστήματος που έχει σημαντική επίδραση στην απόδοση του, έχει παραληφθεί. Αντιστρόφως, μπορεί να γίνει αντιληπτό κάποιο συστατικό που μπορεί να έχει περιληφθεί στο μοντέλο αλλά θα μπορούσε

να παραληφθεί. Πολλές φορές μπορεί να παρατηρηθούν ακόμα και συμπεριφορές του μοντέλου που δεν θα μπορούσαν να συμβούν στο πραγματικό σύστημα, όπως για παράδειγμα σε ένα σύστημα ουρών, περισσότεροι εξυπηρετητές από ότι θα ήταν εφικτό, η οργάνωσή τους σε ανέφικτα πρότυπα. Στο βήμα που έπεται της μετάφρασης της λύσης από το μοντέλο στο πραγματικό σύστημα θα πρέπει να αποφασιστεί αν το πρώτο συμπεριφέρεται όπως το δεύτερο σε σχέση με τα στοιχεία που μας ενδιαφέρουν, να γίνει δηλαδή επικύρωση (validation) του μοντέλου. Εάν αυτό ισχύει τότε έχουμε βρει τη λύση του προβλήματος και μπορούμε να την εφαρμόσουμε στο πραγματικό σύστημα. Διαφορετικά όλη η διαδικασία θα πρέπει να επαναληφθεί λαμβάνοντας υπόψη ότι νέο έχει προκύψει από όλα τα προηγούμενα βήματα.

Οι πιθανές εφαρμογές της προσομοίωσης μπορούν να είναι πρακτικά άπειρες. Φυσικά, η δημιουργία του μοντέλου θα πρέπει να είναι μία κοντινή προσαρμογή του πραγματικού συστήματος, χωρίς όμως να αποτελεί αντιγραφή του ίδιου του συστήματος. Ανάλογα με το είδος της μελέτης που επιθυμείται, ο σχεδιαστής θα αποφασίσει το βαθμό της ακρίβειας και της λεπτομέρειας που κρίνεται απαραίτητη. Οι πρακτικές αποφάσεις που θα πρέπει να ληφθούν γύρω από τα παραπάνω θέματα, οριοθετούν και τα βασικά είδη προσομοίωσης [Pidd, 1998].

- Πρόοδος χρόνου
- Διακριτή ή συνεχής αλλαγή
- Στοχαστικοί ή προσδιοριστικοί χρόνοι

1.2.1 Πρόοδος χρόνου (Time – advance)

Βασικό πλεονέκτημα της προσομοίωσης αποτελεί η ευκολία χειρισμού του χρόνου σε ένα μοντέλο, σε αντίθεση με το πραγματικό σύστημα. Λίγα δευτερόλεπτα επεξεργασίας ενός μοντέλου σε έναν υπολογιστή δίνουν ως αποτέλεσμα τη συμπεριφορά ενός συστήματος σε βάθος εβδομάδων. Όταν, όμως, μελετούμε το χρόνο, δεν ενδιαφερόμαστε για το χρόνο εκτέλεσης (run time) (που είναι προτιμότερο να είναι κατά το δυνατόν μικρότερος), αλλά για το χρόνο προσομοίωσης (simulation time).

Όταν στο προς μελέτη σύστημα δεν υπεισέρχεται ο χρόνος σαν στοιχείο της λύσης, είτε γιατί μελετούμε το σύστημα σε μία χρονική στιγμή, είτε γιατί ο χρόνος δεν επηρεάζει το σύστημα, τότε έχουμε ένα παράδειγμα στατικής προσομοίωσης (static simulation). Η προσομοίωση, αντίθετα, μοντέλων στα οποία πρέπει να ληφθεί υπόψη η αλλαγή και εξέλιξη του συστήματος κατά τη διάρκεια του χρόνου αναφέρονται σαν μοντέλα δυναμικής προσομοίωσης (dynamic simulation)

Το πρόγραμμα το οποίο υλοποιεί την προσομοίωση θα πρέπει να περιέχει τον τρόπο χειρισμού της αλλαγής του χρόνου στο σύστημα και να κρατά στατιστικά την πάροδο του χρόνου προσομοίωσης. Αυτό επιτυγχάνεται αυξάνοντας το ρολόι της προσομοίωσης (simulation clock). Μπορούν να χρησιμοποιηθούν δύο βασικές τεχνικές, οι οποίες εξαρτώνται από τον τρόπο με τον οποίο γίνεται η πορεία στον χρόνο[Garrido J.M., 2001]. Οι τεχνικές αυτές είναι:

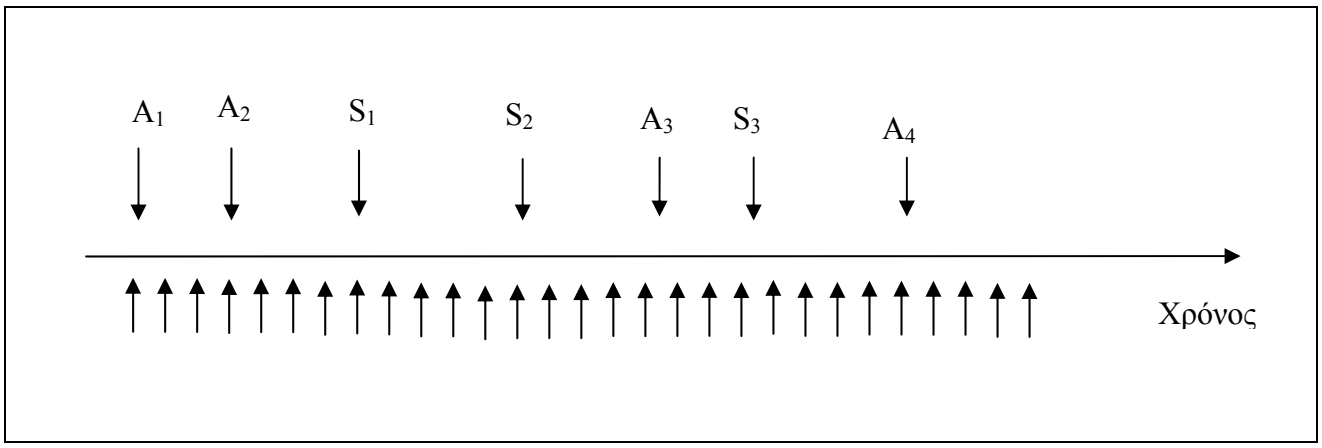
- Μηχανισμός σταθερού διαστήματος - Fixed time increment (time – slicing)
- Μηχανισμός επόμενου γεγονότος - Variable time increment (next – event technique)

1.2.1.1 Μηχανισμός σταθερού διαστήματος (Fixed time increment)

Σε αυτού του είδους τις προσομοιώσεις, ο χρόνος προχωράει σε τακτικά διαστήματα, συνήθως κάθε ένα δευτερόλεπτο ή ένα λεπτό, γενικώς κάθε Δt. Σε κάθε πρόοδο του χρόνου, τα γεγονότα (events) που καλούνται για εκτέλεση, πρέπει να εκτελεστούν.

Ας θεωρήσουμε το παράδειγμα ενός απλού συστήματος με έναν εξυπηρετητή και μία ουρά. Τα γεγονότα που θα συμβούν είναι η άφιξη του πρώτου πελάτη στο χρόνο 0 και η άφιξη του δεύτερου πελάτη στο χρόνο 3,8.

Στο παρακάτω σχήμα για ένα μοντέλο συστήματος ουρών, και συγκεκριμένα στο κάτω μέρος του, μπορεί να φανεί πότε το σύστημα μελετάται για αλλαγή καταστάσεως. Στο σύστημα αυτό συμβαίνουν κάποια συγκεκριμένα γεγονότα, αφίξεων (A) και εξυπηρετήσεων (S) πελατών, τα οποία φαίνονται στο πάνω σκέλος του σχήματος,

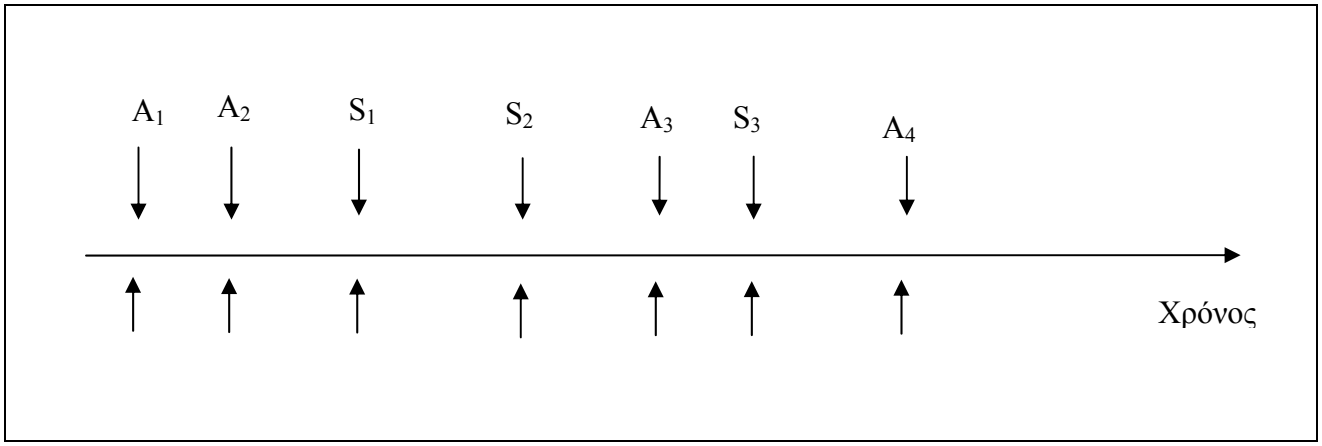


Παρατηρείται ότι, ενώ γεγονότα συμβαίνουν κάθε αρκετά δευτερόλεπτα, το σύστημα μελετάται για αλλαγές κάθε δευτερόλεπτο. Όπως γίνεται αντιληπτό, αυτού του είδους η προσομοίωση είναι γενικά μη αποτελεσματική σε όρους υπολογιστικής ταχύτητας, αφού πολλά διαστήματα μπορούν να περάσουν χωρίς καμία αλλαγή στην κατάσταση του συστήματος. Επίσης, έχει το μειονέκτημα της ανακρίβειας των αποτελεσμάτων.

1.2.1.2 Μηχανισμός επόμενου γεγονότος (Variable time increment)

Η τεχνική αυτή είναι βασισμένη σε ορισμένες απλές ιδέες. Κάθε γεγονός προκαλεί άλλα γεγονότα να συμβούν, τα οποία με τη σειρά τους προκαλούν εκ νέου κάποια γεγονότα, κ.ο.κ.. Μόνο γεγονότα μπορούν να προκαλέσουν άλλα γεγονότα και, αφού ισχύει αυτό, η κατευθυντήρια γραμμή που ακολουθείται είναι η εξής:

I. Το ρολόι της προσομοίωσης αρχικοποιείται στο 0 και όλα τα αρχικά γεγονότα τοποθετούνται σε μία λίστα, τη λεγόμενη λίστα των μελλοντικών γεγονότων (future event list).
II. Το ρολόι στη συνέχεια αυξάνεται στο χρόνο του πρώτου χρονικά γεγονότος στη λίστα. Το γεγονός αυτό θα πρέπει να αφαιρεθεί από τη λίστα.
III. Η κατάσταση του συστήματος θα ενημερωθεί ανάλογα με το γεγονός.
IV. Πιθανά νέα γεγονότα θα τοποθετηθούν στη λίστα.
V. Αν η λίστα που θα προκύψει δεν είναι άδεια, πάμε στο βήμα II.



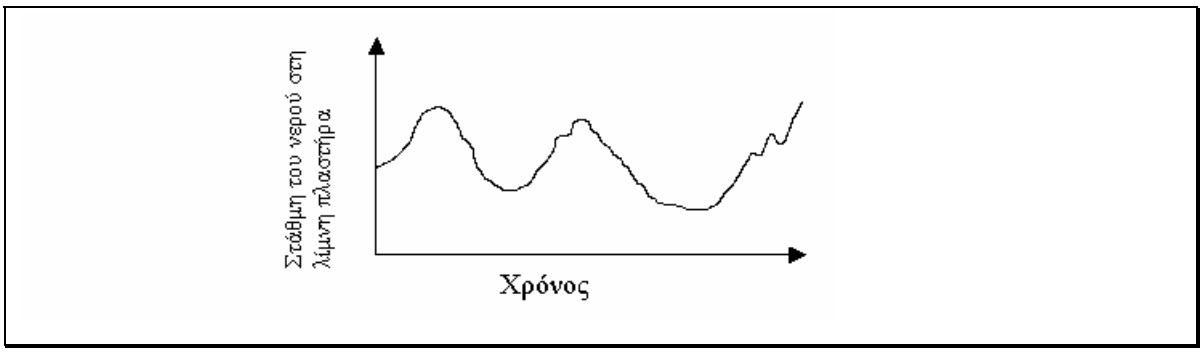
Θεωρώντας το ίδιο μοντέλο και τα ίδια γεγονότα με αυτά του προηγούμενου παραδείγματος, βλέπουμε στο παραπάνω σχήμα ότι το αντίστοιχο μοντέλο θα μελετηθεί πολύ λιγότερες φορές σε σύγκριση με το προηγούμενο. Ένα φανερό πλεονέκτημα λοιπόν της τεχνικής αυτής, αφού το ρολόι κάνει άλματα από τη μία χρονική στιγμή στην άλλη, είναι ότι η κεντρική μονάδα απασχολείται με υπολογισμούς μόνο όταν κάτι, ένα γεγονός, συμβεί. Επιπρόσθετα, με την τεχνική αυτή επιτυγχάνουμε πιο ακριβείς μετρήσεις. Οι μετρήσεις αυτές γίνονται ακριβώς όταν συμβαίνουν οι αλλαγές, και όχι κάθε χρονική στιγμή της επόμενης, προκαθορισμένης μελέτης του συστήματος, όπως στη προηγούμενη τεχνική.

1.2.2 Διακριτά και συνεχή συστήματα

Στην πράξη, πολύ λίγα συστήματα είναι εξ ολοκλήρου διακριτά ή συνεχή, αλλά από τη στιγμή που η αλλαγή του ενός είδους κυριαρχεί της άλλης, μπορούμε να ταξινομήσουμε ένα σύστημα σαν διακριτό ή συνεχές [Law and Kelton 2000]. Η κατηγορία στην οποία θα ταξινομήσουμε το σύστημα, πάντως, είναι συνάρτηση του πως λειτουργεί το ρολόι της προσομοίωσης.

1.2.2.1 Συνεχής αλλαγή (Continuous change)

Τα συστήματα συνεχούς αλλαγής είναι συστήματα στα οποία οι μεταβλητές κατάστασης αλλάζουν συνεχώς κατά τη διάρκεια του χρόνου. Η στάθμη του νερού στο φράγμα της λίμνης Πλαστήρα στην πόλη καταγωγής μου είναι ένα τυπικό, και δυστυχώς επίκαιρο, παράδειγμα τέτοιου συστήματος. Για τέτοιου είδους προσομοιώσεις συμπεριφοράς υγρών υπάρχει πληθώρα παραδειγμάτων προσομοιώσεων και στο διαδίκτυο [http://user.it.uu.se/~psa/], [http://www.scitrav.com/wwater/sasspro/sasspro.asp].

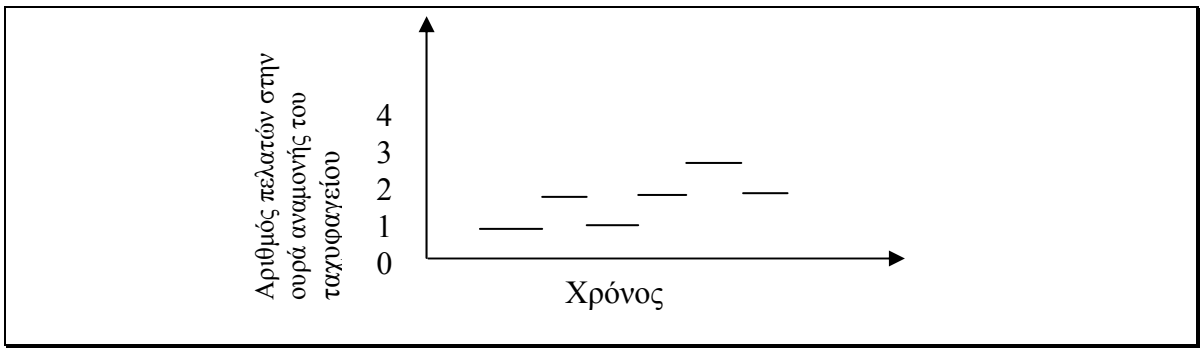


Στην υπόθεση του παραπάνω σχήματος βλέπουμε κάποιες περιόδους αύξησης της στάθμης, ίσως κατά τη διάρκεια του φθινοπώρου και του χειμώνα, όπου έχουμε βροχές και χιονοπτώσεις. Ίσως οι μειώσεις να οφείλονται σε άνυδρα καλοκαίρια με μεγάλη κατανάλωση. Οι αυξομειώσεις αυτές δε γίνονται σε συγκεκριμένες χρονικές στιγμές, αλλά καθ' όλη τη διάρκεια του χρόνου.

Ένα περιστατικό στην περίπτωση αυτή μπορεί να συμβεί σε οποιαδήποτε χρονική στιγμή ή να αλλάζει συνεχώς, προκαλώντας συνεχή αλλαγή στην κατάσταση του συστήματος.

1.2.2.2 Διακριτή αλλαγή (Discrete Change)

Τα συστήματα διακριτής αλλαγής είναι συστήματα, των οποίων η κατάσταση αλλάζει μόνο σε διακριτές χρονικές στιγμές. Ένα τέτοιο παράδειγμα είναι ένα ταχυφαγείο, το οποίο θα αποτελέσει παράδειγμα σε επόμενο κεφάλαιο της εργασίας αυτής.



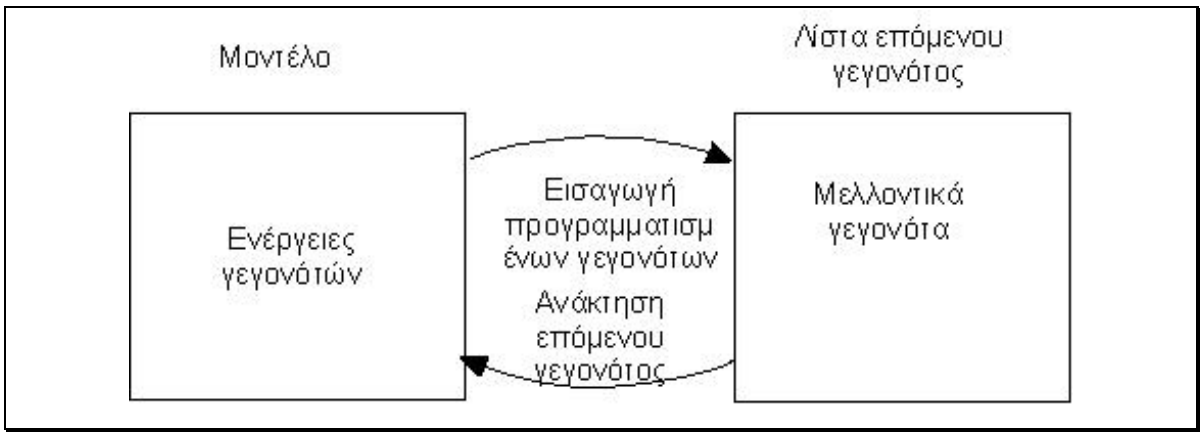
Το σύνολο των πελατών που περιμένουν για παραγγελία αλλάζει μόνο όταν κάποιος πελάτης φτάσει στο ταχυφαγείο, κάτι που συνεπάγεται την αύξηση της ουράς αναμονής κατά έναν πελάτη, ή όταν τελειώνει μία παραγγελία, κάτι που συνεπάγεται ότι ένας πελάτης από την ουρά αναμονής θα πάει για παραγγελία μειώνοντάς τους κατά έναν. Ποτέ η κατάσταση ενός τέτοιου συστήματος δεν αλλάζει ανάμεσα σε δύο γεγονότα.

Ένα στιγμιαίο περιστατικό, σαν τα παραπάνω, που συμβαίνει σε κάποια διακριτή χρονική στιγμή και επηρεάζει την κατάσταση του συστήματος, όπως θα αναφερθεί και στη συνέχεια, ορίζεται σαν διακριτό γεγονός ή απλούστερα σαν γεγονός (event). Τα γεγονότα αυτά είναι και τα μόνα που μπορούν να προκαλέσουν αλλαγή στο μοντέλο.

1.2.3 Λίστα επόμενων γεγονότων (Future Event List)

Η λίστα επόμενων γεγονότων (future event list) αποτελεί έναν τρόπο διαφύλαξης κάθε διαφορετικής ενέργειας που συμβαίνει σε κάθε εκτέλεση της προσομοίωσης [Law and Kelton, 2000]. Οτιδήποτε συμβαίνει κατά τη διάρκεια μίας εκτέλεσης και μπορεί να επηρεάσει την κατάσταση του συστήματος, καλείται γεγονός (event). Παράδειγμα τυπικών γεγονότων στην προσομοίωση ενός συστήματος ουρών είναι η άφιξη ενός πελάτη στο σύστημα, καθώς και η έναρξη ή ο τερματισμός της εξυπηρέτησης ενός πελάτη. Τέτοιου είδους γεγονότα επηρεάζουν το σύστημα, καθώς μπορούν να αλλάξουν τον αριθμό των πελατών που περιμένουν στην ουρά και στο σύστημα γενικότερα, ενώ αλλάζουν επιπλέον και την

κατάσταση κάποιων πόρων, όπως οι εξυπηρετητές, οι οποίοι από απασχολημένοι (busy) μπορούν να μετατραπούν σε αδρανείς (idle) ή αντίστροφα..



Στην περίπτωση που έχουμε σύστημα διακριτής αλλαγής, ο χρόνος της προσομοίωσης προωθείται από γεγονός σε γεγονός και κάθε φορά ενημερώνεται τόσο η κατάσταση του συστήματος, όσο και η νέα κατάσταση της λίστας των επόμενων γεγονότων. Στην αντίθετη περίπτωση, ο χρόνος προωθείται συνεχώς, γεγονός που κάνει τους υπολογισμούς πολύ πιο δύσκολους.

Σε σύστημα διακριτού γεγονότος, που είναι και αυτό που μας ενδιαφέρει, λοιπόν και σε κάποια συγκεκριμένη στιγμή t, η λίστα περιέχει όλα τα γεγονότα τα οποία ήδη έχουν προγραμματιστεί για κάθε χρονική στιγμή t_i και τις στιγμές αυτές. Η λίστα είναι ταξινομημένη σύμφωνα με τις χρονικές αυτές στιγμές όπως φαίνεται στη συνέχεια.

$$t < t_1 \leq t_2 \leq t_3 \leq t_4 \leq t_5 \leq \ldots \leq t_n$$

Η χρονική στιγμή t είναι η τιμή του ρολογιού της προσομοίωσης εκείνη τη στιγμή και το γεγονός με το χρόνο t_1 είναι το επικείμενο γεγονός, το γεγονός που θα συμβεί στη συνέχεια. Ο χρόνος θα μεταφερθεί στο χρόνο t_1 το επικείμενο γεγονός θα αφαιρεθεί από τη λίστα και θα εκτελεστεί. Η εκτέλεσή του θα δημιουργήσει μία νέα εικόνα του συστήματος και της λίστας, εικόνα που εξαρτάται από το ίδιο το γεγονός. Το γεγονός αυτό μπορεί να προκαλέσει τη δημιουργία νέων γεγονότων, τα οποία ανάλογα με το χρόνο εκτέλεσής τους θα πρέπει να μπουν στη σωστή θέση στην ανανεωμένη λίστα. Η διαδικασία αυτή εκτελείται μέχρι να τελειώσει η προσομοίωση.

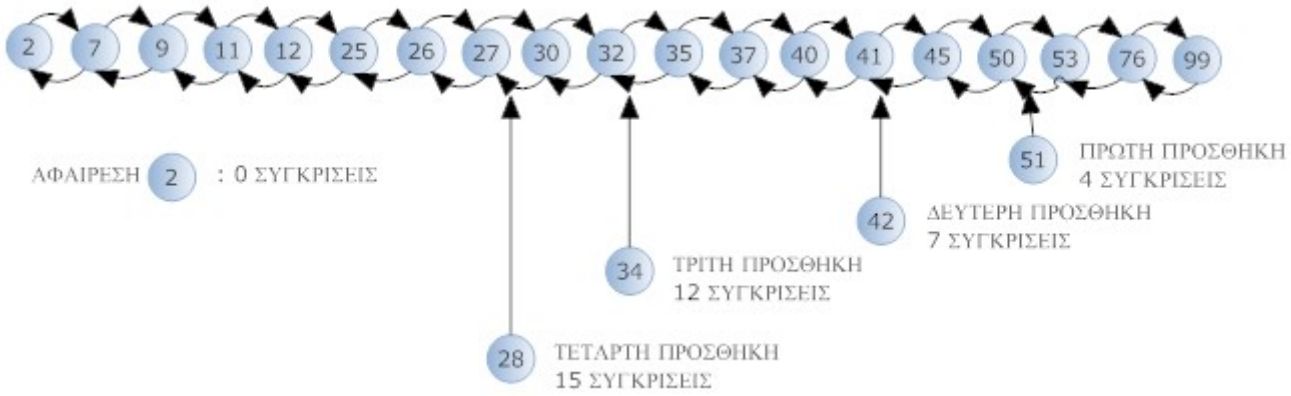
Η υλοποίηση της διαδικασίας επόμενου γεγονότος είναι ίσως η πιο δαπανηρή στα πλαίσια υπολογιστικών πόρων καθώς έχει να κάνει με μία λίστα η οποία αλλάζει συνεχώς κατά τη διάρκεια της προσομοίωσης. Ο αποδοτικός τρόπος διαχείρισής του προγραμματιστικά θα οδηγήσει σε αποδοτικό πρόγραμμα προσομοίωσης. Οι πιο σημαντικές ενέργειες αφορούν ενέργειες σε λίστα, τη λίστα επόμενου γεγονότος,, και είναι οι ακόλουθες [Fishman G.S.,2001]

a. η προσθήκη γεγονότος στη σωστή θέση στη λίστα
b. η αφαίρεση του επικείμενου γεγονότος από τη λίστα
c. μετά την αφαίρεση, εύρεση του επόμενου γεγονότος από τα εναπομείναντα στη λίστα
d. η εύρεση κάποιου γεγονότος με συγκεκριμένες ιδιότητες και αφαίρεση – ακύρωσή του από τη λίστα

Η ενέργεια a αντιστοιχεί σε χρονικό προγραμματισμό ενός γεγονότος προς εκτέλεση, η b σε επιλογή του επόμενου γεγονότος για εκτέλεση, η c σε εξακρίβωση του επόμενου γεγονότος για εκτέλεση και η ενέργεια d στην ακύρωση ενός προγραμματισμένου γεγονότος. Η τελευταία αυτή ενέργεια δεν συμβαίνει πολύ συχνά και συνακόλουθα δεν επηρεάζει την

συνολική απόδοση του αλγορίθμου. Επίσης η ενέργεια b συμβαίνει πάντα μαζί με την ενέργεια c και μπορεί να θεωρηθεί σαν μια ενέργεια. Ο τρόπος με τον οποίο θα γίνουν αυτές και γενικότερα ο τρόπος με τον οποίο είναι οργανωμένη η λίστα, επηρεάζει πολύ την αποδοτικότητα του όλου μοντέλου. Στην ιδανική περίπτωση και δεδομένου της συνολικής επιρροής του αλγορίθμου αυτού στη συνολική απόδοση της προσομοίωσης θα πρέπει φυσικά να χρησιμοποιηθεί ο βέλτιστος αλγόριθμος σε υπολογιστικούς πόρους. Είναι όμως αδύνατο να βρεθεί ένας τέτοιος αλγόριθμος δεδομένου ότι όταν δοκιμάζονται σε διαφορετικά μοντέλα κανείς δεν παρουσιάζει ομοιόμορφα κάποια καλύτερη απόδοση[McCormack and Sargent,1981].

Μια μέθοδος υλοποίησης της λίστας επόμενου γεγονότος είναι να υλοποιηθεί σαν μια διπλά συνδεδεμένη λίστα (single linear event list). Ας υποτεθεί ότι οι αφαιρέσεις γεγονότων γίνονται από την αριστερή πλευρά και οι προσθήκες από δεξιά και οι αριθμοί μέσα σε κάθε κύκλο δηλώνουν τον χρόνο εκτέλεσης κάθε γεγονότος στο επόμενο σχήμα[Fishman G.S.,2001]



Ο αλγόριθμος προγραμματίζει τέσσερα γεγονότα με χρόνο εκτέλεσης 51, 42, 34 και 28 και στη συνέχεια αφαιρεί το επικείμενο γεγονός της λίστας. Όπως παρατηρείται οι προσθήκες των τεσσάρων γεγονότων θα απαιτήσουν 4 + 7 + 12 + 15 = 38 συγκρίσεις. Γενικότερα η προσθήκη γεγονότος a σε μια διπλά συνδεδεμένη λίστα που ήδη περιέχει N γεγονότα θα απαιτήσει στη χειρότερη περίπτωση (το νεοεισελθόν γεγονός να είναι και το επικείμενο γεγονός) O(N) υπολογιστικό χρόνο, αλλά η αφαίρεση του επικείμενου γεγονότος b απαιτεί σταθερό χρόνο ανεξάρτητο του N. Γίνεται φανερό ότι ενώ η λειτουργία b είναι ανεξάρτητη του μοντέλου, η λειτουργία της προσθήκης εξαρτάται από τη μορφή της κατανομής των χρόνων και δεν μπορεί να εκφραστεί εύκολα σαν συνάρτηση του μεγέθους της λίστας. Συνακόλουθα όταν μελετούνται τέτοιοι αλγόριθμοι μελετάται η χειρότερη περίπτωση και η επιρροή του μεγέθους της προσομοίωσης στον υπολογιστικό χρόνο.

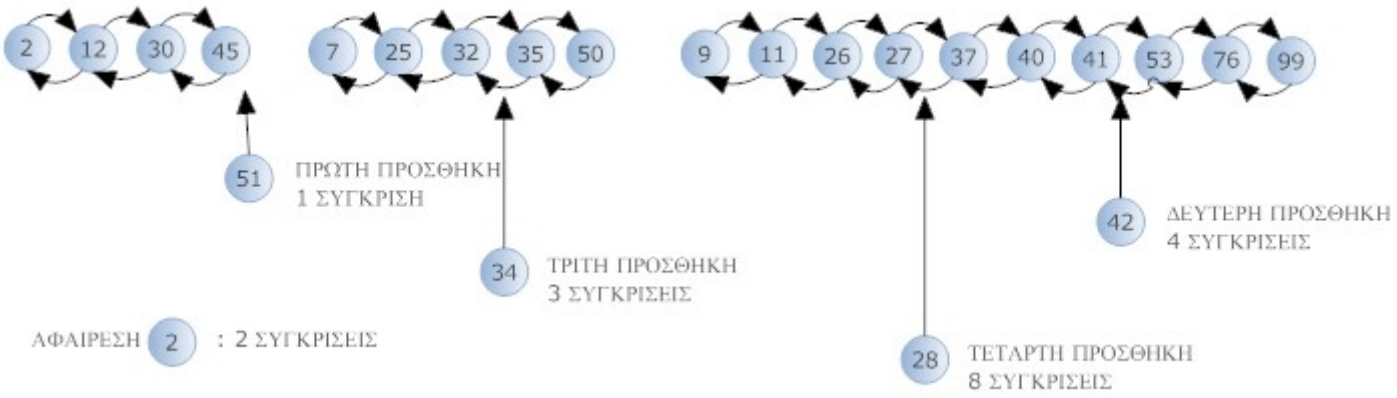
Μία άλλη προσέγγιση για την μείωση του χρόνου αυτού βασίζεται στις πολλαπλές λίστες, μια για κάθε τύπο γεγονότος (multiple linear event list). Στον προγραμματισμό για παράδειγμα ενός γεγονός τύπου i οι ενέργειες του αλγορίθμου είναι οι ακόλουθες.

1. Δημιουργία του γεγονότος
2. Καθορισμός και εκχώρηση του χρόνου εκτέλεσης στην ιδιότητα χρόνου του γεγονότος
3. Γραμμική αναζήτηση για προσθήκη του γεγονότος στη λίστα i
4. Αν είναι πρώτο στη λίστα, ανανέωση της διεύθυνσης επικεφαλίδας (header)
5. Αν είναι τελευταίο στη λίστα, ανανέωση της διεύθυνσης ουράς (trailer)

Όταν τερματίζεται η εκτέλεση ενός γεγονότος η συνάρτηση προγραμματισμού θα πρέπει να επιλέξει το επόμενο προς εκτέλεση γεγονός.

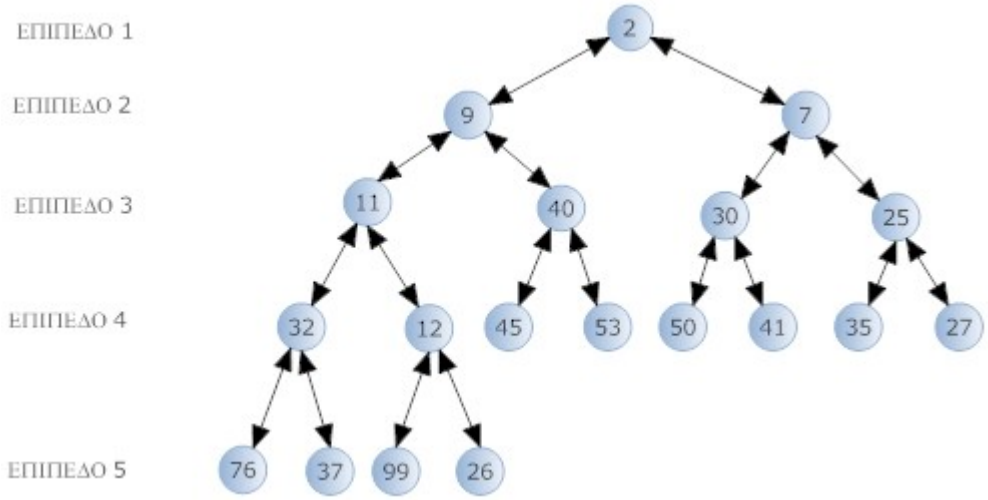
1. Αναζήτηση της λίστας των επικεφαλίδων για τον προσδιορισμό του γεγονότος με την μικρότερη τιμή χρόνου εκτέλεσης.
2. Αφαίρεση του γεγονότος από τη λίστα.
3. Ανανέωση της διεύθυνσης επικεφαλίδας.
4. Πρόοδος του χρόνου προσομοίωσης στην τιμή του χρόνου εκτέλεσης του γεγονότος
5. Εκτέλεση του γεγονότος

Στη συνέχεια παρουσιάζονται οι ίδιες ενέργειες με την μέθοδο της διπλά συνδεδεμένης λίστας.



Η προσομοίωση και πάλι προγραμματίζει τέσσερα γεγονότα με χρόνους εκτέλεσης 51, 42, 34 και 28 πριν εκτελεστεί το επικείμενο γεγονός με χρόνο εκτέλεσης 2. Οι τέσσερεις εισαγωγές θα απαιτήσουν 1 + 4 + 3 + 8 = 16 συγκρίσεις σε αντίθεση με τις 38 που απαιτήθηκε στην προηγούμενη περίπτωση. Εν τούτοις, η διαγραφή απαιτεί δύο συγκρίσεις λόγω της ανάγκης να ελεγχθεί ποια από τις πρώτες καταχωρήσεις στις τρεις λίστες έχει τον μικρότερο χρόνο εκτέλεσης. Στην περίπτωση αυτή ο υπολογιστικός χρόνος για την αφαίρεση του επικείμενου γεγονότος εξαρτάται από τις διαφορετικές λίστες λόγω των συγκρίσεων που θα πρέπει να γίνει μεταξύ των πρώτων γεγονότων κάθε μίας. Η εισαγωγή εξαρτάται από το μέγεθος της λίστας i που θα επιλεχτεί αλλά και από το νούμερο των διαφορετικών λιστών λόγω της γραμμικής αναζήτησης για την κατάλληλη λίστα i.

Μία τρίτη προσέγγιση στην διαχείριση της λίστας επόμενων γεγονότων αποτελεί ο δυαδικός σωρός (binary heap)[Williams 1964]. Χρησιμοποιώντας την ίδια λίστα επόμενων γεγονότων με ακριβώς τους ίδιους χρόνους εκτέλεσης έπεται το δέντρο του οποίου η δομή και οι λειτουργίες θα εξηγηθούν στη συνέχεια.



Κάθε γεγονός - κόμβος (node) έχει έναν συγκεκριμένο χρόνο εκτέλεσης που αναφέρεται ρητά στο σχήμα. Παιδιά (children) ενός κόμβου u που βρίσκεται σε κάποιο επίπεδο i ονομάζονται οι κόμβοι που βρίσκονται στο επίπεδο i+1 και είναι συνδεδεμένοι στον κόμβο u. Ο κόμβος u ονομάζεται γονέας (parent) των κόμβων αυτών. Ένας κόμβος που δεν έχει παιδιά ονομάζεται φύλλο (leaf). Ο κόμβος που δεν έχει γονιό και βρίσκεται στο επίπεδο 1 είναι η ρίζα (root) του δέντρου. Το βάθος (depth) κάποιου κόμβου είναι ο αριθμός των κόμβων στη διαδρομή που τον συνδέει με την ρίζα. Στοίβα (heap) καλείται το δέντρο στο οποίο σε κάθε υποδέντρο η ρίζα έχει μικρότερη τιμή από όλους τους κόμβους του. Δυαδική στοίβα (binary heap) την οποία και χρησιμοποιεί ο υπό μελέτη αλγόριθμος είναι η στοίβα στην οποία κάθε κόμβος έχει το πολύ δύο παιδιά.

Η δομή δεδομένων για να υποστηριχτεί η δυαδική στοίβα είναι ένας πίνακας στον οποίο οι κόμβοι αποθηκεύονται στη σειρά από την κορυφή (ρίζα) στη βάση του δέντρου. Συνακόλουθα η ρίζα καταλαμβάνει τη θέση 1 του πίνακα με τα παιδιά του να καταλαμβάνουν τις θέσεις 2 και 3. Γενικότερα οι κόμβοι σε βάθος i καταλαμβάνουν τις θέσεις 2^i, 2^i+1,…, $2^{i+1}-1$. Στην περίπτωση που η στοίβα διατηρείται έτσι ώστε τα στοιχεία της, N, να καταλαμβάνουν πάντα τις l αριστερά θέσεις σε κάθε επίπεδο, ποτέ το βάθος του δέντρου δεν μπορεί να ξεπεράσει το $\log_2 N$.

Η διαδικασίες της προσθήκης η αφαίρεσης γεγονότος από τη λίστα επόμενου γεγονότος χρησιμοποιούν τους αλγορίθμους siftup(x) και siftdown(x) όπου x το γεγονός και key(x) ο χρόνος επικείμενης εκτέλεσής του. Ο πρώτος αλγόριθμος παίρνει ένα στοιχείο x του δέντρου, του οποίου ο χρόνος key(x) έχει ελαττωθεί και το μετακινεί προς τα πάνω στο δέντρο στην κατάλληλη θέση. Ο δεύτερος παίρνει ένα γεγονός του οποίου το key(x) έχει αυξηθεί και το μετακινεί προς τα κάτω στην κατάλληλη θέση.

Αλγόριθμος siftup(x)

```
until SUCCESS:
    if x is the root, SUCCESS
    otherwise:
        let y be the parent of x
        if key(y) <= key(x), SUCCESS
        otherwise:
            swap entries in x and y.
```

Αλγόριθμος siftdown(x)

until SUCCESS:

if x is a leaf, SUCCESS

otherwise:

let y be the child of x with minimal key

if key(y) >= key(x), SUCCESS

otherwise:

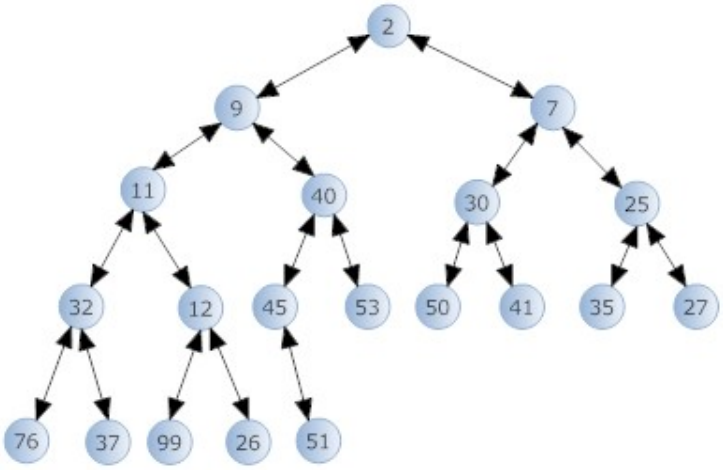
swap entries in x and y.

Για την προσθήκη ενός γεγονότος x στη λίστα θα πρέπει να ακολουθηθούν τα επόμενα βήματα.

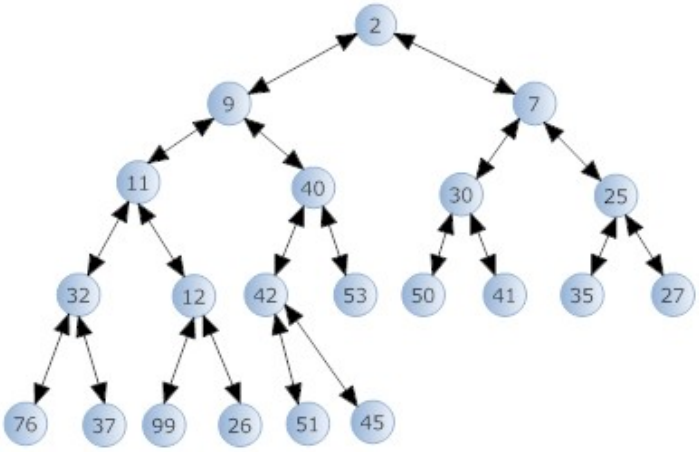
- Εισαγωγή γεγονότος x σαν φύλλο
- Εκτέλεση αλγορίθμου siftup(x)

Για την αφαίρεση ενός γεγονότος από τη λίστα θα πρέπει να γίνουν τα ακόλουθα.

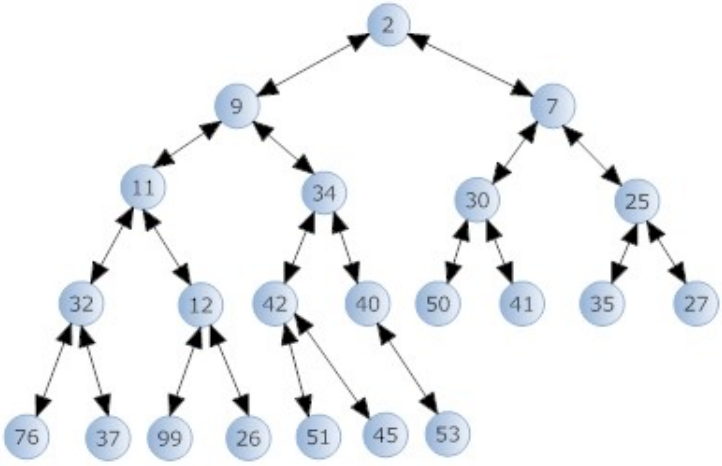
- Διαγραφή του γεγονότος που βρίσκεται στη ρίζα της στοίβας
- Μετακίνηση του τελευταίου φύλλου στην ρίζα
- Εκτέλεση siftdown(y) εναλλαγές



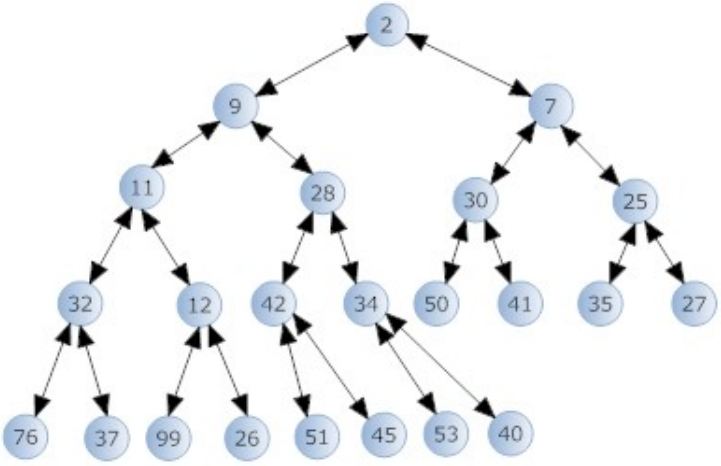
ΠΡΩΤΗ ΠΡΟΣΘΗΚΗ 51 :1 ΣΥΓΚΡΙΣΗ, 0 ΕΝΑΛΛΑΓΕΣ



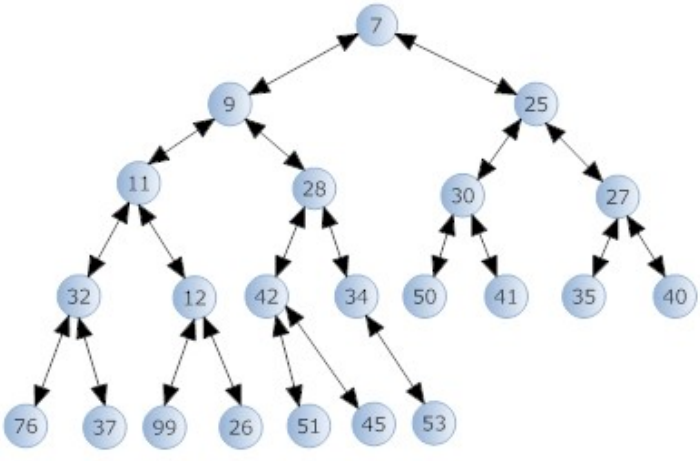
ΔΕΥΤΕΡΗ ΠΡΟΣΘΗΚΗ 42 :2 ΣΥΓΚΡΙΣΕΙΣ, 1 ΕΝΑΛΛΑΓΗ



ΤΡΙΤΗ ΠΡΟΣΘΗΚΗ 34 :3 ΣΥΓΚΡΙΣΕΙΣ, 2 ΕΝΑΛΛΑΓΕΣ



ΤΕΤΑΡΤΗ ΠΡΟΣΘΗΚΗ 28 :3 ΣΥΓΚΡΙΣΕΙΣ, 2 ΕΝΑΛΛΑΓΕΣ



ΑΦΑΙΡΕΣΗ 2 :1 ΜΕΤΑΦΟΡΑ ΑΠΟ LEAF ΣΕ ROOT,
3 ΣΥΓΚΡΙΣΕΙΣ, 3 ΕΝΑΛΛΑΓΕΣ

Επαναλαμβάνοντας την ίδια διαδικασία με τις ίδιες τιμές των προηγούμενων μεθόδων παρατηρείται ότι για την διαδοχική προσθήκη γεγονότων με χρόνους 51, 42, 34, 28 απαιτούνται 1 + 2 + 3 + 3 = 9 συγκρίσεις και 0 + 1 + 2 + 2 = 5 εναλλαγές σε αντιδιαστολή με τις 38 συγκρίσεις της πρώτης προσέγγισης και τις 16 της δεύτερης προσέγγισης των

πολλαπλών λιστών. Σημειώνεται ότι οι εναλλαγές και οι συγκρίσεις είναι συγκρίσιμες όσο αφορά τους υπολογιστικούς χρόνους. Η αφαίρεση του επικείμενου γεγονότος απαιτεί μια μεταφορά από το φύλλο στη ρίζα, τρεις συγκρίσεις και τρεις εναλλαγές, γεγονός που την κάνει πιο χρονοβόρα από τις προηγούμενες περιπτώσεις. Ο συνδυασμός όμως όλων των πράξεων στη λίστα επόμενου γεγονότος θέτει την τελευταία αυτή μέθοδο χειρισμού της προτιμότερη σε υπολογιστικούς χρόνους. Υπάρχουν διάφοροι άλλοι αλγόριθμοι [McCormack and Sargent,1981], που έχουν επινοηθεί για τον χειρισμό της λίστας που ξεφεύγουν όμως από τους στόχους της εργασίας αυτής.

1.2.4 Στοχαστικοί ή προσδιοριστικοί χρόνοι

1.2.4.1 Προσδιοριστικοί χρόνοι (deterministic durations)

Προσδιοριστικού χρόνου, είναι τα μοντέλα στα οποία όλες οι μεταβλητές είναι γνωστές με βεβαιότητα. Ελάχιστες καταστάσεις του πραγματικού κόσμου μπορούν να μελετηθούν χρησιμοποιώντας τέτοιου είδους μοντέλα. Είναι σχεδόν σίγουρο ότι σε τέτοιες καταστάσεις τα μαθηματικά θα κάνουν καλύτερη δουλειά από την προσομοίωση, δεν παύουν όμως τέτοιου είδους μοντέλα να είναι πολύ χρήσιμα για τα πρώτα βήματα της διαδικασίας κατασκευής ενός μοντέλου. Είναι δεδομένο ότι κάθε αλλαγή αποτελεσμάτων που θα παρατηρηθεί δεν έχει γίνει λόγω της τυχαιότητας των μεταβλητών, αλλά λόγω της αλλαγής στο μοντέλο. Άλλωστε, όσες φορές και να πειραματιστεί κάποιος με ένα τέτοιο μοντέλο, τα αποτελέσματα που θα προκύψουν θα είναι πανομοιότυπα. Ένα σύστημα διαφορικών εξισώσεων που μπορούν να περιγράψουν μία χημική αντίδραση μπορούν να αποτελέσουν ένα τέτοιου είδους μοντέλο. Η προσθήκη της τυχαιότητας σε ένα τέτοιο μοντέλο θα το μετατρέψει σε στοχαστικό ή σε μοντέλο Monte Carlo.

1.2.4.2 Στοχαστικοί χρόνοι (Stochastic durations)

Κατά την μοντελοποίηση πραγματικών φαινομένων, δεν υπάρχουν πολλές περιπτώσεις στις οποίες τα γεγονότα και οι διαδικασίες μπορούν να προβλεφθούν. Ένα σύστημα μπορεί να αποτελείται από προβλέψιμες αλλά και απρόβλεπτες διαδικασίες που σε κάθε διαφορετική περίπτωση θα έκαναν όλο το σύστημα να συμπεριφέρεται διαφορετικά. Αναπόφευκτα, ενώ η είσοδος στο σύστημα μπορεί να είναι της ίδιας μορφής, η έξοδος μπορεί κάθε φορά να είναι διαφορετική. Χρήσιμα συμπεράσματα θα μπορούσαν να εξαχθούν εάν η προσομοίωση επαναλαμβανόταν πολλές φορές και ακολουθούσε στατιστική μελέτη στα αποτελέσματα.

Το γεγονός ότι παραδοσιακοί τρόποι δημιουργίας ακολουθιών τυχαίων αριθμών, απαραίτητων για την μοντελοποίηση τέτοιων συστημάτων, είναι το ρίξιμο ζαριών, το ανακάτεμα ή το γύρισμα της ρουλέτας οδήγησαν στην ονομασία της προσομοίωσης με ενσωματωμένη κάποια τυχαιότητα, προσομοίωση monte carlo. Όρος που περιγράφει γενικώς οποιαδήποτε αλγοριθμική διαδικασία περιλαμβάνει τυχαίους αριθμούς. Για να θεωρηθεί, γενικώς, μία προσομοίωση Monte Carlo πρέπει να χρησιμοποιεί κάποια μέθοδο παραγωγής τυχαίων αριθμών [Kleijnen 1974].

Η πλειοψηφία των συστημάτων που υπάρχουν στον κόσμο είναι τέτοιας μορφής καθώς ο αληθινός κόσμος είναι στοχαστικός και όχι καθορισμένος. Κανένας, για παράδειγμα, δεν μπορεί να είναι σίγουρος για το πότε ακριβώς θα έρθει ο επόμενος πελάτης σε ένα ταχυδρομείο ή για τον σκοπό του ερχομού του και πόση ώρα θα χρειαστεί ο ταμίας για να τον εξυπηρετήσει, ή πότε ο ταμίας αυτός θα κάνει διάλειμμα και πόση ώρα θα κρατήσει αυτό κ.ο.κ. Ακόμα και σε ένα πολύ απλό σύστημα, υπάρχουν πάρα πολλοί παράγοντες οι οποίοι δεν μπορούν να προβλεφθούν. Παρόλα αυτά, μία κοντινή

περιγραφή μπορεί να γίνει χρησιμοποιώντας στατιστικά μοντέλα. Στη θεωρία ουρών, όπου οι στοχαστικοί χρόνοι άφιξης και εξυπηρέτησης είναι συνήθως η περίπτωσή χρόνων που δεν μπορούν να προβλεφθούν, μία πολύ κοντινή περιγραφή μπορεί να υποτεθεί, για να τους περιγράψει. Όπως θα ειπωθεί και στη συνέχεια οι συμπεριφορές αυτές μπορούν να αναπαρασταθούν υπό τη μορφή κατανομών. Οι κατανομές αυτές όμως θα πρέπει να στηριχτούν σε κάποιες τυχαίες μεταβλητές. Κάθε υπολογιστής έχει έναν μηχανισμό για την παραγωγή τυχαίων αριθμών συνήθως απλής μορφής από τον οποίο θα πρέπει να εξαχθούν όλες αυτές οι κατανομές που προσεγγίζουν συμπεριφορές.

1.2.5 Ορισμοί στη προσομοίωση διακριτού γεγονότος.

Πριν όμως προχωρήσουμε στη μελέτη των αριθμών αυτών, χρήσιμο είναι να δούμε κάποιους κοινώς αποδεκτούς ορισμούς για σημαντικούς παράγοντες της προσομοίωσης διακριτού γεγονότος που είναι και το είδος που μελετά αυτή η εργασία [Banks and Carson, 1984]. Οι όροι αυτοί όταν θα χρησιμοποιούνται στη συνέχεια της εργασίας θα αναφέρονται στους επόμενους ορισμούς.

1.2.5.1 Σύστημα (System)

Σύστημα καλούμε μία συλλογή οντοτήτων που αλληλεπιδρά στη διάρκεια του χρόνου για την επίτευξη ενός ή περισσότερων στόχων.

Το σύστημα ήταν αυτό που εξ αρχής έπρεπε να μελετηθεί και το ίσως το πιο δύσκολο κομμάτι μίας προσομοίωσης είναι η απεικόνιση ενός συστήματος ως μοντέλου. Παράδειγμα συστήματος είναι το κατάστημα fast food της γειτονιάς μας που σκέφτεται αν θα πρέπει να προσλάβει επιπλέον ταμεία και/ή ψήστη για να εξυπηρετεί πιο γρήγορα τους πελάτες του.

Μπορούμε να θεωρήσουμε εδώ για τις ανάγκες του παραδείγματός μας ότι η διαδικασία παραγγελίας για κάθε πελάτη είναι να στηθεί στην ουρά του ταμία να δώσει μία παραγγελία και μετά αφού πληρώσει να στηθεί σε μία νέα ουρά, αυτή του ψήστη, όπου εκ νέου θα περιμένει τη σειρά του να δώσει το χαρτί με την παραγγελία που πήρε από το ταμείο.

1.2.5.2 Μοντέλο (Model)

Μία αφηρημένη αναπαράσταση ενός συστήματος,, συνήθως περιέχει λογικές και/η μαθηματικές σχέσεις που περιγράφουν το σύστημα με όρους καταστάσεως, οντοτήτων και των ιδιοτήτων τους, ομάδων, γεγονότων, δραστηριοτήτων και καθυστερήσεων.

Έτσι, στο παραπάνω σύστημα του fast food, το μοντέλο δεν θα αναπαραστήσει ούτε τη ζαρντινιέρα στην είσοδο του, ούτε το αν ο ταμίας φοράει πράσινα παπούτσια. Αυτό που θα αναπαριστά είναι, για τον ταμία και τον ψήστη για παράδειγμα δυο κατανομές που θα προσεγγίζουν τον τρόπο με τον οποίο εξυπηρετούν τους πελάτες και για τους πελάτες ίσως μία κατανομή που θα προσεγγίζει τον τρόπο με τον οποίο εισέρχονται στο κατάστημα και μία άλλη που θα προσπαθεί να παραστήσει τις παραγγελίες τους στα πλαίσια ίσως του χρόνου που θα απαιτείται από τον ψήστη για την διεκπεραίωσή τους..

1.2.5.3 Κατάσταση συστήματος (System state)

Μία συλλογή μεταβλητών που περιέχουν κάθε απαραίτητη πληροφορία για να περιγραφεί το σύστημα σε κάθε χρονική στιγμή.

Το τι θα θεωρηθεί βέβαια απαραίτητο προς περιγραφή αφήνεται στη κρίση του μελετητή και είναι ίσως ο σημαντικότερος παράγοντας που οδηγεί στην επιτυχία ή αποτυχία της προσομοίωσης.

Τη χρονική στιγμή δώδεκα το μεσημέρι η κατάσταση του συστήματος έχει ως εξής: Στην ουρά για τον ψήστη στο fast food μας περιμένουν 4 άτομα, ο ψήστης είναι απασχολημένος. Στην ουρά για το ταμείο η ουρά είναι κενή και ο ταμίας είναι αδρανής. Πρέπει να τονιστεί ότι η κατάσταση αυτή δεν περιγράφει πλήρως τα σύστημα σε κάποια χρονική στιγμή. Για να γίνει αυτό πρέπει απαραίτητα να συμπεριληφθεί η λίστα επόμενου γεγονότος (future event list) καθώς και κάθε τιμή στατιστικού ή μετρητή, απαραίτητα για τον υπολογισμό στατιστικών στο τέλος της προσομοίωσης.

1.2.5.4 Οντότητα (Entity)

Κάθε αντικείμενο και συστατικό στο σύστημα που απαιτεί σαφή αναπαράσταση στο μοντέλο.

Σαφή αναπαράσταση στο παράδειγμά μας απαιτούν ο ταμίας, ο ψήστης και ο πελάτης. Ένας διαχωρισμός που επιβάλλεται μεταξύ τους είναι αυτός των μόνιμων και προσωρινών οντοτήτων σε ένα σύστημα. Έτσι, ενώ στο παράδειγμα ο ταμίας και ο ψήστης είναι σταθεροί στο σύστημα, ο πελάτης είναι προσωρινός, θα πάρει την παραγγελία, θα τη δώσει στον ψήστη, θα εξυπηρετηθεί και από αυτόν και θα αναχωρήσει από το σύστημα. Είναι προφανές ότι στη διαδικασία υλοποίησης της προσομοίωσης θα χρησιμοποιηθεί εντελώς διαφορετική λογική για τις μόνιμες από ότι για τις προσωρινές οντότητες.

1.2.5.5 Ιδιότητες (Attributes)

Τα χαρακτηριστικά μίας δεδομένης οντότητας.

Ο ταμίας έχει σαν χαρακτηριστικό τον τρόπο με τον οποίο εξυπηρετεί πελάτες, για παράδειγμα μία συγκεκριμένη κατανομή με συγκεκριμένες παραμέτρους.

1.2.5.6 Ομάδα (Set)

Μία συλλογή συσχετιζόμενων, είτε μόνιμα είτε προσωρινά, οντοτήτων κατηγοριοποιημένα με κάποιον λογικό τρόπο.

Όλοι οι πελάτες που περιμένουν κάποια χρονική στιγμή στην ουρά και εξυπηρετούνται με first come – first served πολιτική αποτελούν μία τέτοια ομάδα.

1.2.5.7 Γεγονός (Event)

Ένα στιγμιαίο περιστατικό που αλλάζει την κατάσταση του συστήματος.

Τέτοιο περιστατικό στο παράδειγμά μας είναι η άφιξη ενός πελάτη με πιθανότερη αλλαγή, στην περίπτωση που υπάρχει ουρά στο ταμείο, στην αύξηση της ουράς κατά ένα ή, στην περίπτωση που δεν υπάρχει, στο να ξεκινήσει να δουλεύει ο ταμίας. Γεγονός αποτελεί και ο τερματισμός ενός ψησίματος από τον ψήστη που σημαίνει ότι μπορεί να αρχίσει να εξυπηρετείται ο επόμενος πελάτης και άρα η μείωση της ουράς του ψήστη κατά ένα ή, σε περίπτωση μη ύπαρξης επόμενου, το πέρασμα του ψήστη σε αδρανή κατάσταση.

1.2.5.8 Δραστηριότητα (Activity)

Μία καθορισμένη χρονική διάρκεια, της οποίας η διάρκεια είναι γνωστή στο ξεκίνημά της.

Η διάρκεια αυτή μπορεί να υπολογίζεται είτε στοχαστικά είτε προσδιοριστικά. Αυτό που μας ενδιαφέρει όμως είναι ότι, με το που ξεκινάει η δραστηριότητα, η χρονική στιγμή που θα περατωθεί είναι γνωστή. Στο μοντέλο έχει καθοριστεί ο χρόνος που απαιτείται για έναν ψήστη, έστω και αν αυτό έχει γίνει με όρους κάποιας κατανομής. Με το που ξεκινάει το ψήσιμο υπολογίζεται εξαρχής ο χρόνος που θα απαιτηθεί για αυτό, έστω και με τη βοήθεια της κατανομής αυτής.

1.2.5.9 Καθυστέρηση (Delay)

Μία ακαθόριστη χρονική διάρκεια, της οποίας η διάρκεια δεν είναι γνωστή μέχρι αυτή να τερματίσει.

Έστω ότι στο fast food μας ή επιλογή του επόμενου προς εξυπηρέτηση πελάτη γίνεται στην τύχη (random). Η διάρκεια που θα περιμένει ένας πελάτης στην ουρά είναι ακαθόριστη, εφόσον δε γνωρίζουμε ποια χρονική στιγμή θα εξυπηρετηθεί αυτός, κάτι που εξαρτάται από μελλοντικές αφίξεις πελατών, καθώς και με τη στατιστική κατανομή, η οποία θα καθορίσει πότε θα επιλεγεί ο συγκεκριμένος πελάτης.

1.3 Τυχαίοι αριθμοί

Είναι φανερό ότι, με τον τρόπο που δουλεύουν οι υπολογιστές, η παραγωγή τυχαίων αριθμών είναι αδύνατη. Οι αριθμοί αυτοί, που δημιουργούνται σύμφωνα με κάποια μέθοδο, καλούνται ψευδοτυχαίοι αριθμοί. Η πιο συχνή μέθοδος είναι η παραγωγή κάθε αριθμού, εφαρμόζοντας κάποια μαθηματική συνάρτηση στον προηγούμενο αριθμό που παράχθηκε. Κάτι τέτοιο βέβαια σημαίνει ότι πρέπει κάπως να παραχθεί ο πρώτος αριθμός x_0 που καλείται και σπόρος (seed). Είναι προφανές ότι ο αριθμός αυτός είναι πολύ σημαντικός για την ακολουθία των αριθμών που θα παραχθούν. Αν ο αριθμός αυτός είναι πάντα ο ίδιος, η ακολουθία αριθμών θα είναι και αυτή πάντα η ίδια, κάτι που, αν και δεν είναι και η πιο τυχαία συμπεριφορά, μπορεί να αποτελέσει και πολύτιμο εργαλείο για τον μελετητή.

Ο γενικότερος στόχος της παραγωγής τυχαίων αριθμών είναι η προσομοίωση διαφόρων στοχαστικών συμπεριφορών στη φύση. Για να επιτευχθεί αυτό, διάφορα μαθηματικά μοντέλα έχουν κατασκευαστεί, οι κατανομές. Από την uniform κατανομή, που είναι αυτή που παράγει κάθε πρόγραμμα υπολογιστή, θα πρέπει να δημιουργηθούν διάφορες άλλες κατανομές που να μπορούν να περιγράψουν τις συμπεριφορές αυτές.

1.3.1 Παραγωγή τυχαίων αριθμών

1.3.1.1 Μέθοδοι παραγωγής τυχαίων αριθμών

Στη φύση υπάρχουν πολλοί τρόποι παραγωγής τυχαίων αριθμών, από τη ρίψη ενός ζαριού μέχρι τη μέτρηση της χρονικής στιγμής που μία θερμή κάθοδος εκπέμπει ένα ηλεκτρόνιο. Μία τέτοιας μορφής, όμως, φυσική διαδικασία είναι πολύ δύσκολο να αναπαραχθεί στο περιβάλλον ενός προσωπικού υπολογιστή. Γενικώς, υπάρχουν τρεις [Forsythe, Malcolm and Moler, 1977] κατηγορίες μεθόδων για την παραγωγή τυχαίων αριθμών.

Η πρώτη μέθοδος περιλαμβάνει τη χρήση κάποιας τυχαίας φυσικής διαδικασίας, όπως αυτές που αναφέρθηκαν. Η ευκολία παραγωγής τέτοιων φαινομένων αποτελεί πλεονέκτημα, αλλά υπεισέρχονται πρακτικές δυσκολίες, όπως, για

παράδειγμα, η χρονομέτρηση των φαινομένων και οι περιπτώσεις σφάλματος. Όμως, η φυσική διαδικασία δε μπορεί να αναπαραχθεί. Είναι πολύ δύσκολο να γραφεί ένα πρόγραμμα στον υπολογιστή που να χρησιμοποιεί μία τέτοια διαδικασία.

Μία δεύτερη μέθοδος παραγωγής τυχαίων αριθμών αποτελεί η παραγωγή τυχαίων αριθμών εκτός του προγράμματος και στη συνέχεια η αποθήκευση τους σε κάποιο αποθηκευτικό μέσο για μετέπειτα χρήση τους από κάποιο πρόγραμμα. Πολλά χρόνια πριν, επιστήμονες που ήθελαν να χρησιμοποιήσουν ακολουθίες τυχαίων αριθμών, είτε έριχναν ζάρια, είτε τραβούσαν μπαλάκια από κάποιο ανακατεμένο δοχείο ,είτε γενικώς χρησιμοποιούσαν κάποιον από τους τρόπους της πρώτης μεθόδου, με τη διαφορά ότι κατέγραφαν τα αποτελέσματα για περαιτέρω χρήση. Η πρώτη τέτοιου είδους καταγραφή παρουσιάστηκε το 1927 από τον Tippett [Tippett and Pearson, 1927] και περιλάμβανε έναν πίνακα 41600 τυχαίων αριθμών. Πολλά τέτοια παραδείγματα από τη χρόνια χρήση τέτοιων μεθόδων χωρίς τη χρήση κάποιου αλγορίθμου έχουν προκύψει, με τελευταία προσπάθεια από την RAND Corporation to 1955 [Rand Corp., 1955]. Ένα πρόβλημα που αντιμετώπισαν οι στατιστικολόγοι ήταν ότι σε κάθε χρήση του αρχείου, αν το διάβασμα ξεκινούσε από την αρχή, η ακολουθία των τυχαίων αριθμών θα ήταν συνεχώς η ίδια, συμπεριφορά όχι και τόσο στοχαστική. Πολύπλοκες μέθοδοι τυχαίας εισόδου κατασκευάστηκαν για τη λύση του προβλήματος αυτού. Σημαντικό, επίσης μειονέκτημα αποτελεί η μεγάλη απαίτηση της μεθόδου για μνήμη, καθώς ένας μεγάλος πίνακας πρέπει να αποθηκευτεί στον υπολογιστή, αλλά και η αργή ανάκτηση κάθε φορά από τη μνήμη του επόμενου τυχαίου αριθμού.

Η τρίτη και πιο συνηθισμένη μέθοδος είναι η χρήση κάποιου αλγορίθμου, ως μέρος του προγράμματος, που να παράγει τυχαίους αριθμούς. Τέτοιοι αριθμοί παράγονται, όποτε χρειαστούν, και η αναπαραγωγή τους για έλεγχο είναι πολύ εύκολη. Βέβαια, αριθμοί που παράγονται από αλγόριθμο δε μπορεί να είναι τυχαίοι και συνηθίζεται να ονομάζονται ψευδοτυχαίοι. Η μέθοδος αυτή αποτελεί την πιο ενδιαφέρουσα και αυτή που χρησιμοποιείται, τουλάχιστον τις τελευταίες δεκαετίες, στην προσομοίωση, καθώς, εκτός του ότι οι παραπάνω είναι δαπανηρές, η τελευταία μέθοδος παρουσιάζει ένα πολύ σημαντικό πλεονέκτημα. Όταν κριθεί απαραίτητο, μπορεί να επαναληφθεί παράγοντας ακριβώς την ίδια ακολουθία αριθμών, γεγονός πολύ χρήσιμο όταν πρέπει για παράδειγμα να συγκριθούν δυο συστήματα με τα ίδια ακριβώς δεδομένα. Οι αλγόριθμοι που παράγουν τους τυχαίους αριθμούς ονομάζονται γεννήτριες τυχαίων αριθμών.

1.3.1.2 Γεννήτριες τυχαίων αριθμών (random number generators)

Οι μέθοδοι παραγωγής ψευδοτυχαίων αριθμών επικεντρώνονται στην παραγωγή αριθμών ομοιόμορφα κατανεμημένων στο (0,1), αφού, όπως θα δειχθεί στη συνέχεια, από τους αριθμούς αυτούς μπορεί να προκύψει οποιαδήποτε ακολουθία αριθμών. Η πιο συνηθισμένη μέθοδος παραγωγής τους είναι η αναδρομική, κατά την οποία, αφού γίνει η επιλογή του πρώτου αριθμού ή σπόρου (seed), χρησιμοποιείται μία συνάρτηση $x_{\kappa+1} = f(x_\kappa)$ η οποία μας δίνει τον επόμενο αριθμό. Μετά από την παραγωγή κάποιων αριθμών η συνάρτηση θα αρχίσει να επαναλαμβάνει την ίδια ακολουθία αριθμών. Το πλήθος των αριθμών αυτών μέχρι να αρχίσει η επανάληψη ονομάζεται περίοδος. Η επιλογή της συνάρτησης θα πρέπει γίνει έτσι, ώστε η περίοδος αυτή να είναι όσο το δυνατόν μεγαλύτερη, καθώς είναι αδύνατο με τη χρήση υπολογιστικών και μόνο πόρων να μην υπάρξει αυτή η επανάληψη.

Η συνάρτηση[Lehmer, 1949] που χρησιμοποιείται συχνότερα επιτυγχάνει την παραγωγή της ακολουθίας των τυχαίων αριθμών X_κ ως εξής:

$$X_{\kappa+1}= (\alpha X_\kappa +c) \bmod m \quad \kappa\geq 0$$

Η επιλογή τεσσάρων «μαγικών» [Knuth,1969] αριθμών που καθορίζουν την παραπάνω συνάρτηση γίνεται πολύ σημαντική. Οι αριθμοί αυτοί είναι:

- m ακέραιο υπόλοιπο (modulus) $m > 0$
- α ο πολλαπλασιαστής (multiplier) $0 \le \alpha < m$
- c η προσαύξηση (increment) $0 \le c < m$
- X_0 η αρχική τιμή ή σπόρος (starting value – seed) $0 \le X_0 < m$

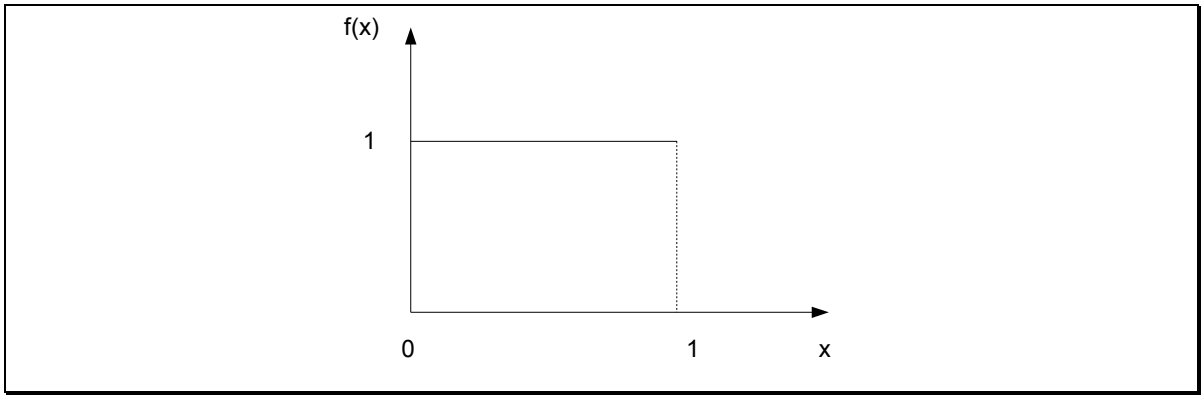
Αν έτσι, οι αριθμοί που θα επιλεχθούν είναι οι m=10, X_0=7, a=7, c=7 η ακολουθία που θα προκύψει είναι

7, 6, 9, 0, 7, 6, 9, 0, 7,…

Γίνεται φανερό ότι η ακολουθία των αριθμών τείνει να επαναλαμβάνεται, στη συγκεκριμένη περίπτωση κάθε τέσσερις μόλις αριθμούς. Είναι πολύ σημαντική η επιλογή των μαγικών αριθμών, ώστε η περίοδος να γίνει όσο το δυνατόν πιο μεγάλη, με το τέσσερα να αποτελεί έναν πολύ μικρό αριθμό για αποδεκτή περίοδο.

1.3.2 Στατιστικές Κατανομές

Στην πιο διαδεδομένη μορφή αριθμών που παράγει κάποιο πρόγραμμα υπολογιστή, κάθε νέος αριθμός είναι ένα ανεξάρτητο δείγμα που έχει παρθεί από μία συνεχή ομοιόμορφη κατανομή ανάμεσα στο 0 και το 1.



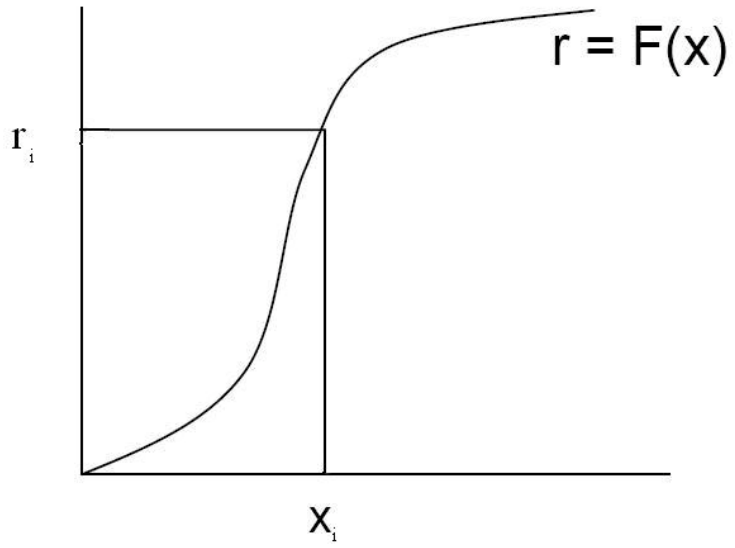
Έτσι, η συνάρτηση πιθανότητας των τυχαίων αριθμών που ακολουθούν την παραπάνω κατανομή είναι:

$$f(x) = \begin{matrix} 1, 0 \le x \le 1 \\ 0 \; διαφορετικά \end{matrix}$$

Ο στόχος είναι, με δεδομένη αυτή την κατανομή που μπορεί να παραχθεί εύκολα από οποιαδήποτε διαδεδομένη γλώσσα προγραμματισμού, να δημιουργηθεί μία μεγάλη γκάμα κατανομών, ικανή να προσομοιώσει διάφορες συμπεριφορές. Πολλές από τις κατανομές αυτές είναι διακριτές που περιγράφουν φαινόμενα, στα οποία μόνο γεγονότα με ακέραιες τιμές μπορούν να συμβούν και άλλες συνεχείς για φαινόμενα, στα οποία μπορεί να προκύψει οποιαδήποτε τιμή. Όταν προσομοιώνουμε ένα μοντέλο, όμως, αυτό που μας ενδιαφέρει είναι η παραγωγή δειγμάτων που να ακολουθούν μία από αυτές τις κατανομές. Για το σκοπό αυτό, υπάρχουν διάφορες τεχνικές, με πιο διαδεδομένη και απλή την τεχνική της αντίστροφης μεταμόρφωσης (inverse transform technique). Χρησιμοποιείται, επίσης, η τεχνική της αποδοχής – απόρριψης (acceptance – rejection), όπως, επίσης, η μέθοδος polar (polar method).

Η τεχνική της αντίστροφης μεταμόρφωσης χρησιμοποιείται, όταν η συνάρτηση συσσωρεμένης πιθανότητας (CDF) F(x) είναι τόσο απλή, ώστε η αντίστροφή της F^{-1} να μπορεί να υπολογιστεί. Περιπτώσεις κατανομών όπου χρησιμοποιείται αυτή η τεχνική είναι η εκθετική (exponential), η ομοιόμορφη, η τριγωνική (triangular), η weibull, αλλά και οι εμπειρικές κατανομές (empirical) που κάθε μία ξεχωριστά θα συζητηθεί στη συνέχεια. Τα βήματα της τεχνικής αυτής φαίνονται στη συνέχεια

- Βήμα 1: Υπολογίζεται η CDF της επιθυμητής τυχαίας μεταβλητής X, F(x).



- Βήμα 2: Τίθεται F(x) = R στο πεδίο τιμών της συνάρτησης.
- Βήμα 3: Λύνεται η εξίσωση F(x) = R ως προς x.
- Βήμα 4: Παράγονται ομοιόμορφα κατανεμημένοι αριθμοί $R_1,R_2,R_3,\ldots$ στο (0,1) και υπολογίζονται οι επιθυμητές κατανεμημένες τιμές χρησιμοποιώντας τον τύπο $X_i=F^{-1}(R_i)$

Σε περιπτώσεις που δε μπορεί να εφαρμοστεί η παραπάνω τεχνική, εφαρμόζεται μία λιγότερο αποδοτική μέθοδος, η τεχνική της αποδοχής – απόρριψης (acceptance – rejection). Στη μέθοδο αυτή παράγονται ομοιόμορφα κατανεμημένοι αριθμοί στο (0,1) μέχρι να ισχύσει κάποια συνθήκη. Η τεχνική αυτή βασίζει την αποδοτικότητά της στην ελαχιστοποίηση του αριθμού των απορρίψεων. Τα βήματα της τεχνικής αυτής φαίνονται στη συνέχεια:

- Βήμα 1: Παραγωγή τυχαίου αριθμού R από τη γνωστή συνάρτηση u(0,1)
- Βήμα 2α: Αν το R είναι μεγαλύτερο ή ίσο με κάποιο αριθμό Y, τότε δέξου X=R και πήγαινε στο βήμα 3
- Βήμα 2β: Αν είναι μικρότερο, απέρριψε και επέστρεψε στο βήμα 1
- Βήμα 3: Αν είναι απαραίτητος και άλλος αριθμός στο [Y,1), επανέλαβε τη διαδικασία, αλλιώς σταμάτα.

Κάθε φορά που εκτελείται το βήμα 1, ένας νέος αριθμός R πρέπει να παραχθεί και η συνάρτηση u(0,1) να κληθεί. Το βήμα 2α είναι το βήμα αποδοχής του R και το βήμα 2β το βήμα απόρριψης. Γενικώς, συνέχεια παράγονται τυχαίοι αριθμοί R, μέχρι να ικανοποιήσουν κάποια συνθήκη. Η τεχνική αυτή βρίσκει εφαρμογή στην κατανομή Poisson που αναφέρεται στη συνέχεια, αλλά και στην κατανομή gamma.

Για τη παραγωγή αριθμών που να ακολουθούν την κανονική κατανομή χρησιμοποιείται πολύ συχνά η μέθοδος polar [Box, 1958] και [Ross 1997]. Κατά τη μέθοδο αυτή, ακολουθούνται τα επόμενα βήματα, τα οποία υπολογίζουν δύο μεταβλητές X_1 και X_2, κατανεμημένες κανονικά με την ταυτόχρονη χρήση δυο ομοιόμορφα κατανεμημένων αριθμών.

- Βήμα 1: Ο αλγόριθμος παράγει δύο ανεξάρτητες ομοιόμορφα κατανεμημένες μεταβλητές στο (0,1) u_1 και u_2 με το γνωστό πλέον τρόπο.
- Βήμα 2: Τίθεται $v_1 = 2*u_1 - 1$ και $v_2 = 2*u_2 - 1$, κάτι που σημαίνει ότι v_1 και v_2 είναι ομοιόμορφα κατανεμημένες στο (-1,1) και υπολογίζεται το $S = v_1^2 + v_2^2$
- Βήμα 3: Αν S >=1, πηγαίνουμε στο πρώτο βήμα και η διαδικασία επαναλαμβάνεται Εάν το S είναι μικρότερο της μονάδος, παράγονται δύο επιθυμητές, κανονικά κατανεμημένες, μεταβλητές σύμφωνα με τους τύπους: $X_1 = v_1\sqrt{-2\ln S / S}$ και $X_2 = v_2\sqrt{-2\ln S / S}$

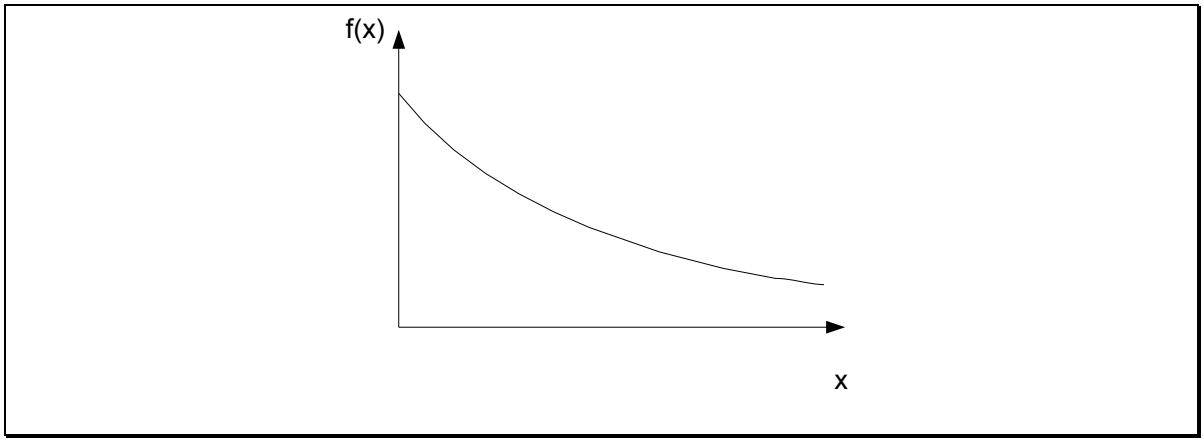
Στη συνέχεια αναλύονται κάποιες κατανομές, αυτές που υλοποιούνται από την εφαρμογή, και εξηγείται η χρήση της κατάλληλης κάθε φορά μεθόδου.

1.3.2.1 Εκθετική κατανομή (exponential distribution)

Η συνεχής αυτή κατανομή είναι κατάλληλη, για να δώσει τους χρόνους μεταξύ ανεξάρτητων γεγονότων που συμβαίνουν με σταθερό ρυθμό. Είναι πολύ χρήσιμη στην προσομοίωση, καθώς ο χρόνος μεταξύ αφίξεων των οντοτήτων σε πολλά συστήματα ακολουθεί, ή μπορεί να υποτεθεί ότι ακολουθεί, εκθετική κατανομή. Είναι, επίσης, η μόνη από τις κατανομές χωρίς μνήμη (memoryless), κάτι που σημαίνει ότι, για παράδειγμα, αν υποθέσουμε ότι η κατανομή αναπαριστά τη διάρκεια ζωής μίας μπαταρίας, αν η μπαταρία έχει ήδη δουλέψει κάποιο χρόνο s, η πιθανότητα να αντέξει ακόμα χρόνο t είναι όσο ήταν αρχικά η πιθανότητα να δουλέψει χρόνο t. Μία τυχαία μεταβλητή X είναι κατανεμημένη εκθετικά, με μέσο όρο εμφανίσεων στη μονάδα του χρόνου την παράμετρο λ, αν η συνάρτηση πιθανότητας δίνεται από τον παρακάτω τύπο:

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & \textit{διαφορετικά} \end{cases}$$

Η γραφική της παράσταση έχει την παρακάτω μορφή.



Στόχος μας είναι να καταφέρουμε να πάρουμε δείγματα που να ακολουθούν την παραπάνω κατανομή. Για να γίνει αυτό, αρχικά, θα χρειαστούμε την αθροιστική συνάρτηση κατανομής, η οποία δίνεται από τον τύπο:

$$F(x)=\int_{-\infty}^{x} F(t)dt=\begin{matrix}1-e^{-\lambda x}, & x\geq 0\\ 0, & x<0\end{matrix}$$

Ο στόχος της διαδικασίας [Schmeiser, 1980] είναι να παραχθεί μία ακολουθία από ενδιάμεσους χρόνους X_1, X_2, X_3,… Η τεχνική της αντίστροφης μεταμόρφωσης μπορεί να εφαρμοστεί στην περίπτωση που η παραπάνω συνάρτηση είναι τόσο απλή που να μπορεί εύκολα να υπολογιστεί η αντίστροφή της, F^{-1}.

Το πρώτο βήμα, λοιπόν, είναι ο υπολογισμός της αθροιστικής συνάρτησης κατανομής που για την εκθετική συνάρτηση είναι:

$$F(x)=1-e^{-\lambda x}, x\geq 0$$

Στη συνέχεια, θα πρέπει να θέσουμε στο πεδίο ορισμού την F(x) ίση με R, κάτι που στην κατανομή που μελετάμε μεταφράζεται σε $1 - e^{-\lambda X} = R$ για $\chi \geq 0$. Από τη στιγμή που η X είναι τυχαία μεταβλητή, η $1 - e^{-\lambda X}$ είναι εξίσου τυχαία μεταβλητή που καλείται R. Πολύ χρήσιμο είναι το γεγονός ότι το $1 - e^{-\lambda X}$,όπως φυσικά και το R, ακολουθούν την uniform κατανομή στο διάστημα (0,1).

Λύνοντας την εξίσωση για X βρίσκουμε:

$$X=-1/\lambda \ln(1-R)$$

Η συνάρτηση αυτή καλείται και γεννήτρια τυχαίων αριθμών και με τη βοήθεια της θα παραχθεί μία ακολουθία τιμών που είναι και ο αρχικός στόχος της διαδικασίας. Εφόσον, όπως ειπώθηκε, το R ακολουθεί την uniform κατανομή, μπορεί με ευκολία να παραχθεί από τον υπολογιστή μία ακολουθία τυχαίων αριθμών R_1, R_2, R_3,… και να υπολογιστούν οι τυχαίες μεταβλητές με τον παρακάτω τύπο:

$$X_i=-1/\lambda \ln(1-R_i)$$

Μία απλοποίηση, η οποία γίνεται συνήθως, είναι η αντικατάσταση του $1 - R_i$ με το R_i αφού και τα δύο είναι κατανεμημένα ομοιόμορφα στο (0,1). Έτσι, τελικά, η συνάρτηση καταλήγει στη μορφή:

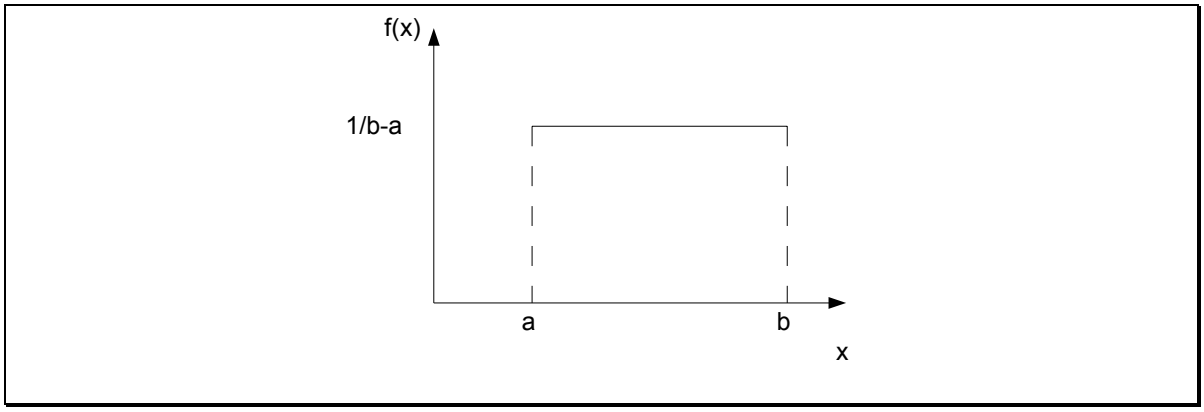
$$X_i=-1/\lambda \ln R_i$$

Με δεδομένη, λοιπόν, μία ακολουθία αριθμών R_1, R_2, R_3,… που ακολουθούν την uniform κατανομή στο διάστημα (0,1), κάτι που προκύπτει εύκολα από τις πιο διαδεδομένες γλώσσες προγραμματισμού, μπορεί να παραχθεί μία ακολουθία X_1, X_2, X_3,… αριθμών που ακολουθεί την εκθετική κατανομή.

1.3.2.2 Ομοιόμορφη κατανομή (uniform distribution)

Η κατανομή αυτή που μπορεί να είναι συνεχής, αλλά και να παίρνει διακριτές τιμές, δε συναντάται συχνά στη φύση, είναι όμως πολύ χρήσιμη για τη δειγματοληψία άλλων κατανομών. Καλείται, επίσης, και ορθογώνια (rectangular) κατανομή και παραμετροποιείται από τη μεγαλύτερη και μικρότερη τιμή. Η συνάρτηση πιθανότητας με a και b, μικρότερη και μεγαλύτερη τιμή αντίστοιχα, αλλά και η γραφική της παράσταση δίνεται στη συνέχεια.

$$f(x)=\begin{matrix}1/b-a, & a\leq x\leq b\\ 0, & \textit{διαφορετικά}\end{matrix}$$



Η διαδικασία παραγωγής των ενδιάμεσων χρόνων θα είναι η αντίστοιχη με αυτήν της εκθετικής κατανομής, με χρήση της τεχνικής της αντίστροφης μεταμόρφωσης.

Καταρχάς, η αθροιστική συνάρτηση πιθανότητας είναι η παρακάτω:

$$F(x)=\begin{cases}0, & x<a\\ x-a/b-a, & a\le x\le b\\ 1, & x>b\end{cases}$$

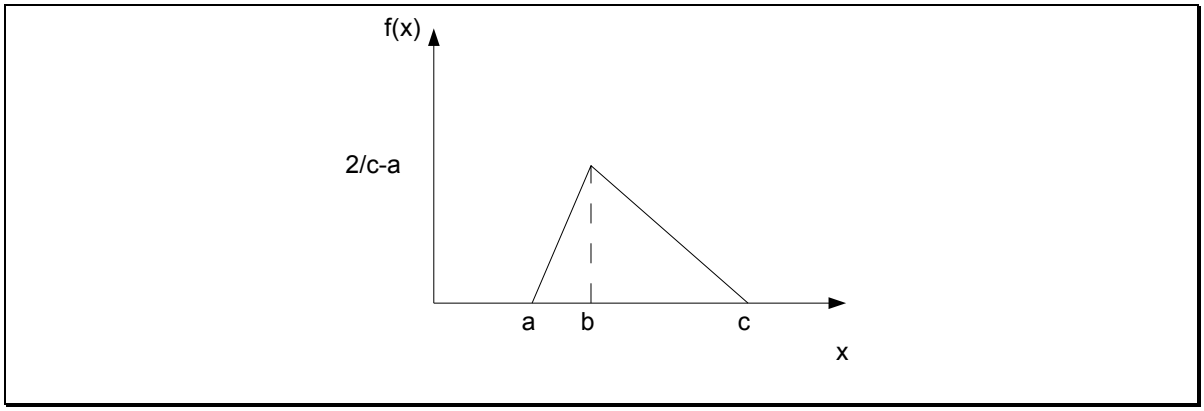
Όμως, F(X) – (X-a)/(b-a) = R και λύνοντας ως προς X έχουμε X = a + (b-a)R

Παράγοντας την ακολουθία αριθμών R_1, R_2, R_3,… και εφαρμόζοντας τον παραπάνω τύπο, δημιουργούμε την ζητούμενη ακολουθία X_1, X_2, X_3,…

1.3.2.3 Τριγωνική κατανομή (Triangular distribution)

Η συνάρτηση πιθανότητάς αυτής της συνεχής κατανομής και μία γραφική της παράσταση είναι:

$$f(x)=\begin{cases}2(x-a)/(b-a)(c-a), & a\le x\le b\\ 2(c-x)/(c-b)(c-a), & b<x\le c\\ 0, & \textit{διαφορετικά}\end{cases}$$



Η αθροιστική συνάρτηση πιθανότητας που προκύπτει είναι:

$$F(x)=\begin{cases}0, & x\le a\\ (x-a)^2/(b-a)(c-a), & a<x\le b\\ 1-(c-x)^2/(c-b)(c-a), & b<x\le c\\ 1, & x>c\end{cases}$$

και θέτοντας την ίση με R και λύοντας ως προς X βρίσκουμε:

$$X=\begin{cases}\sqrt{(b-a)(c-a)*R}+a, & a\le R\le (b-a)/(c-a)\\ c-\sqrt{(1-R)(c-b)(c-a)}, & (b-a)/(c-a)<R\le c\end{cases}$$

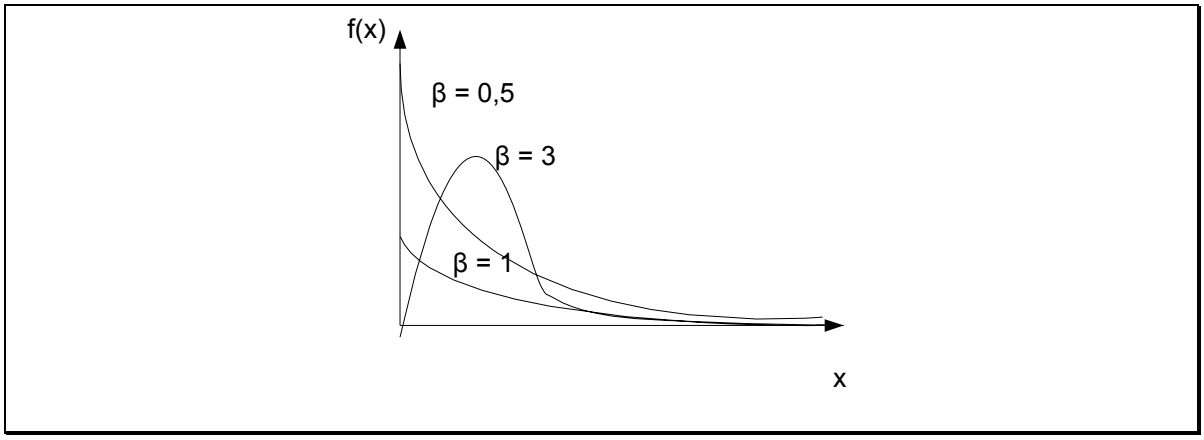
Δημιουργούμε έτσι την ζητούμενη ακολουθία X_1, X_2, X_3,… με τη βοήθεια της R_1, R_2, R_3,…

1.3.2.4 Κατανομή Weibull (Weibull distribution)

Χρησιμοποιείται συχνά, για να προσομοιώσει τον χρόνο μέχρι την αποτυχία και είναι συνεχής. Βρίσκει εφαρμογή σε μηχανές και ηλεκτρονικά εξαρτήματα. Μπορεί, ανάλογα με την τιμή της παραμέτρου shape β της, να αποκτήσει τις ιδιότητες άλλων κατανομών. Η εκθετική κατανομή είναι μία ειδική κατηγορία Weibull κατανομής. Η συνάρτηση πιθανότητας για την ειδική κατηγορία της, με παράμετρο θέσης (location parameter) ν ίση με το μηδέν και για scale α > 0 και shape β > 0, δίνεται από την παρακάτω εξίσωση:

$$f(x)=\begin{cases}\beta/\alpha^{\beta}\,x^{\beta-1}\,\mathbf{e}^{-(x/\alpha)^{\beta}}, & x\ge 0\\ 0, & x<0\end{cases}$$

Στο επόμενο σχήμα φαίνονται η μορφή των γραφικών της παραστάσεων για τρεις διαφορετικές τιμές της παραμέτρου β.



Δουλεύοντας με τον ίδιο τρόπο, όπως στις προηγούμενες περιπτώσεις, βρίσκουμε:

$$F(x)=1-\mathbf{e}^{-(x/\alpha)^{\beta}}=R$$

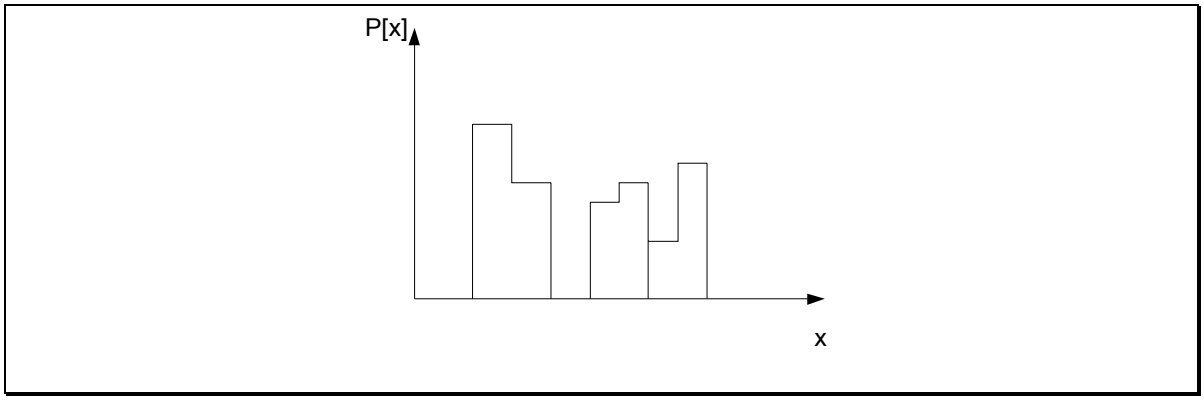
και ως προς X

$$X=\alpha[-\ln(1-R)]^{1/\beta}$$

δημιουργώντας τη ζητούμενη ακολουθία αριθμών που ακολουθούν την weibull κατανομή από μία ακολουθία αριθμών που ακολουθούν την uniform κατανομή στο (0,1).

1.3.2.5 Εμπειρικές κατανομές (Empirical distributions)

Οι εμπειρικές κατανομές μπορεί να είναι είτε συνεχείς είτε διακριτές και χρησιμοποιούνται, όταν δεν μπορεί να βρεθεί καμία ικανή από τις γνωστές κατανομές να περιγράψει το μοντέλο.



Για την απλή, αλλά και πιο συχνά χρησιμοποιούμενη περίπτωση, στην οποία κάθε διάστημα της εμπειρικής κατανομής μπορεί να περιγραφεί από μία ξεχωριστή, ομοιόμορφη κατανομή, η τεχνική της αντίστροφης μεταμόρφωσης μπορεί να εφαρμοστεί με τον ίδιο τρόπο με αυτή, με τη μόνη διαφορά ότι πρέπει, πριν να εφαρμοστεί, να βρεθεί σε ποιο από τα διαφορετικά διαστήματα της εμπειρικής θα πρέπει να εφαρμοστεί. Με τη χρήση ενός μόνο αριθμού R, ομοιόμορφα κατανεμημένου στο (0,1) και ανάλογα την πιθανότητα του κάθε διαστήματος, γίνεται η επιλογή του και στη συνέχεια για το διάστημα αυτό υπολογίζεται ομοίως με την ομοιόμορφη κατανομή το X = a + (b-a)R, μόνο που τα a και b εδώ είναι οι μέγιστες και ελάχιστες τιμές του τρέχοντος διαστήματος και όχι όλης της κατανομής.

1.3.2.6 Διωνυμική κατανομή (Binomial distribution)

Η διακριτή αυτή κατανομή χρησιμοποιείται σε πεπερασμένα προβλήματα δειγματοληψίας, όπου σε κάθε παρατήρηση μπορούν να υπάρχουν δύο πιθανά αποτελέσματα (επιτυχία και αποτυχία), και δίνει τη πιθανότητα να συμβούν ακριβώς s επιτυχίες σε n προσπάθειες. Οι προσπάθειες αυτές ονομάζονται και προσπάθειες Bernoulli (Bernoulli trials). Δοθέντος ενός πειράματος με πιθανότητα επιτυχίας p και αποτυχίας 1-p και n προσπάθειες η πιθανότητα s επιτυχιών (και άρα n - s αποτυχιών) είναι:

$$P=[s\,επιτυχίες\,σε\,n\,προσπάθειες]=(1-p)^s\,p^{n-s}$$

Ο τύπος αυτός μας δίνει μόνο την πιθανότητα να συμβούν αρχικά s επιτυχίες και στη συνέχεια n-s αποτυχίες, όπως φαίνεται και στη συνέχεια:

$$P[SSS...SSSFFF...FFF]=(1-p)^s\,p^{n-s}$$

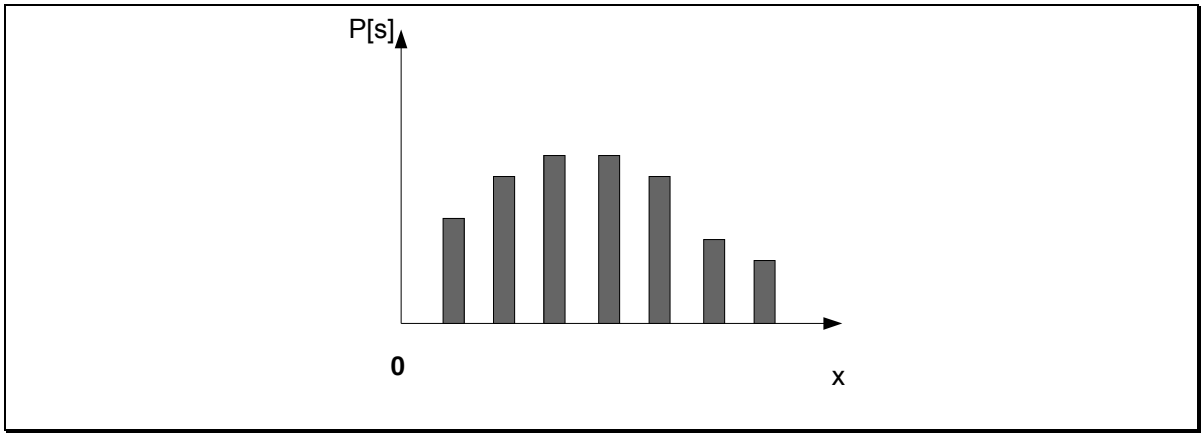
Η πιθανότητα να συμβούν s επιτυχίες από όλους τους συνδυασμούς n προσπαθειών και s επιτυχιών είναι:

$$P[s]=\binom{n}{s}(1-p)^s\,p^{n-s}$$

όπου $\binom{n}{s}$ ο διωνυμικός συντελεστής (binomial coefficient) που δίνει τον αριθμό των τρόπων επιλογής s μη διατεταγμένων αποτελεσμάτων από n διαφορετικές επιλογές. Η τιμή της δίνεται από τον τύπο:

$$\binom{n}{s}=n!/s!(n-s)!$$

Η μορφή μίας τέτοιας κατανομής φαίνεται στο επόμενο παράδειγμα γραφικής παράστασής της



Η τεχνική της αντίστροφης μεταμόρφωσης δε μπορεί να εφαρμοστεί στη περίπτωση της διωνυμικής κατανομής, για την οποία ένα πλήθος τεχνικών [Kachitvichyanukul and Schmeiser, 1988] έχει δοκιμαστεί. Με μία πρώτη σκέψη μπορεί εύκολα να σκεφτεί κανείς ότι, αφού η κατανομή αυτή δεν είναι τίποτα άλλο από n δοκιμές Bernoulli με πιθανότητα επιτυχίας p για κάθε προσπάθεια, και εφόσον ο υπολογιστής μπορεί εύκολα να μας «προσφέρει» n διαφορετικούς αριθμούς στο διάστημα (0,1), μπορεί να θεωρηθεί ότι οι αριθμοί από 0 έως p θα σημαίνουν επιτυχημένη προσπάθεια και από p έως 1 αποτυχημένη. Ο αλγόριθμός αυτός αναπτύσσεται στη συνέχεια.

$\kappa \leftarrow 0$

$\chi \leftarrow 0$

Επανέλαβε

$u \leftarrow U(0,1)$

$\kappa \leftarrow \kappa + 1$

Αν $u \leq p$ τότε

$\chi \leftarrow \chi + 1$

Μέχρις ότου ($\kappa = n$)

Το χ είναι ο αριθμός των επιτυχημένων προσπαθειών και άρα ο ζητούμενος αριθμός. Για να εκτελέσουμε τον αλγόριθμο αυτό θα πρέπει να εκτελέσουμε n φορές τον αλγόριθμο παραγωγής των ομοιόμορφα κατανεμημένων αριθμών στο (0,1), U(0,1). Έτσι, εξαρτώμενος κατά πολύ από τον αλγόριθμο αυτόν είναι πολύ αργός και δεν χρησιμοποιείται παρά μόνο για πολύ μικρά n. Διάφορες βελτιώσεις έχουν προκύψει στην πορεία, με τελευταία αυτή που προτείνουν οι Schmeister και Kachitvichyanukul [Kachitvichyanukul and Schmeiser, 1988] και υλοποιείται και στη παρούσα.

- Βήμα 0: Θέτονται οι σταθερές σαν συναρτήσεις των n και p

$r \leftarrow \min(p, 1-p)$, $q \leftarrow 1-r$, $f_M \leftarrow nr+r$,

$M \leftarrow \lfloor f_M \rfloor$, $p_1 \leftarrow \lfloor 2.195\sqrt{nrq} - 4.6q \rfloor + 0.5$

$X_M \leftarrow M + 0.5$, $X_L \leftarrow X_M - P_I$, $X_R \leftarrow X_M + P_I$,

$c \leftarrow 0.134 + 20.5/(15.3 + M)$,

$a \leftarrow (f_M - x_L)/(f_M - X_L r)$, $\lambda_L - a(1 + a/2)$,

$a \leftarrow (\chi_R - f_M)/(x_R q)$, $\lambda_R \leftarrow a(1 + a/2)$,

$p_2 \leftarrow p1(1 + 2c)$,

$p_3 \leftarrow p_2 + c/\lambda_L$,

$p_4 \leftarrow p3 + c/\lambda_R$.

- Βήμα 1: Δημιουργία ομοιόμορφα κατανεμημένου στο $(0, p_4)$ για την επιλογή της περιοχής. Εάν έχει επιλεγεί η περιοχή 1 τότε δημιουργία μίας τριγωνικά κατανεμημένης μεταβλητής

Generate $u \sim U(0, p_4)$, $v \sim U(0,1)$

if $(u>p_1)$

go to 2

else

$y \leftarrow \lfloor x_M - p_1 v + u \rfloor$, go to 6

- Βήμα 2: Περιοχή 2 τότε

$y \sim U(x_L-0.5, x_R-0.5)$

if $(u>p_2)$

go to 3

else

$x \leftarrow x_L + (u - p_1)/c$

$v \leftarrow vc + 1 - |M - x + .5|/p_1$

if $(v>1)$

go to 1

else

$y \leftarrow \lfloor x \rfloor$, go to 5

- Βήμα 3: Περιοχή 3, η αριστερή εκθετική «ουρά»

if $(u>p_3)$

go to 4

else

$y \leftarrow \lfloor x_L + \ln(v)/\lambda_L \rfloor$

if $(y<0)$

go to 1

else

$v \leftarrow v(u-p_2)\lambda_L$, go to 5

- Βήμα 4: Περιοχή 4, η δεξιά εκθετική «ουρά»

$y \leftarrow \lfloor x_R + \ln(v)/\lambda_R \rfloor$

if y>n

go to 1

else

$v \leftarrow v(u - p_3)\lambda_R$

- Βήμα 5: Σύγκριση αποδοχής/απόρριψης

5.0 Έλεγχος για την κατάλληλη μέθοδο αποτίμησης της f(y)

$k \leftarrow |y-M|$

if $(k<20)$ and $(k <(nrq)/2 – 1)$

go to 5.2

5.1 Αποτίμηση της f(y) μέσω της αναδρομικής σχέσης $f(y) = f(y-1)(a/x-s)$. Ξεκίνημα της αναζήτησης από την επικρατούσα τιμή (κορυφή - mode)

$S \leftarrow r/q$, $a \leftarrow s(n+1)$, $F \leftarrow 1.0$

if $(M<y)$

$i \leftarrow M$

repeat

$i \leftarrow i+1$

$F \leftarrow F (a/i – s)$

until $(i=y)$

else if $(M>y)$

$i \leftarrow y$

repeat

$i \leftarrow i+1$

$F \leftarrow F (a/i – s)$

until $(i=M)$

end if

if $v>F$

go to 1

else

go to 6

5.2 Συμπίεση, έλεγχος της ln(v) έναντι του άνω και κάτω ορίου της ln(f(y))

$\rho \leftarrow (k/(nrq))((k(k/3+0.625)+1/6)(nrq)+0.5)$

$t \leftarrow -k^2/(2(nrq))$

$A \leftarrow \ln(v)$

if $A < t- \rho$, go to 6

if $A > t+\rho$, go to 1

5.3. Τελικός έλεγχος αποδοχής απόρριψης

$x_1 \leftarrow y+1, f_1 \leftarrow M+1$, $z \leftarrow n+1 – M$, $w \leftarrow n-y+1$

$x_2 \leftarrow x_1^2$, $f_2 \leftarrow f_1^2$, $z_2 \leftarrow z^2$, $w_2 \leftarrow w^2$

if $A > x_M\ln(f_1/x_1) + (n- M + 0.5)\ln(z/w) + (y – M)\ln(wr/x_1q) + (13860. - (462 - (132 - (99 - 14O/f_1)/f_2)/f_2)/f_2)/f_1/166320.$

$+ (13860 - (462 - (132 - (99 - 14O./z_2)/z_2)/z_2)/z_2)/z/16632O + (13860 - (462 - (132 - (99 - 14O./x_2)/x_2)/x_2)/x_2)/x_1/l6632O$

$+ (13860 - (462 - (132 - (99 - 140./w_2)/w_2)/w_2)/w_2)/w/166320$, go to1

- Βήμα 6: Έλεγχος για το αν η πιθανότητα p είναι μεγαλύτερη από ½, και εάν είναι επιστροφή n-y

if (p>1/2)

y ← n – y

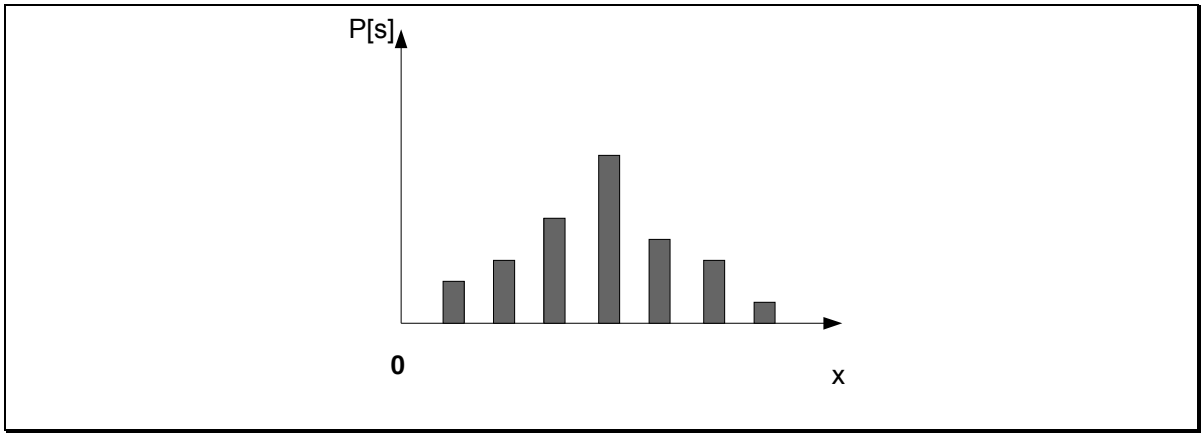
return

1.3.2.7 Κατανομή Poisson (Poisson distribution)

Η κατανομή αυτή είναι διακριτή και χρησιμοποιείται συχνά, για να προσομοιώσει αριθμό γεγονότων, όπως για παράδειγμα τον αριθμό των τηλεφωνημάτων σε μία επιχείρηση ή τον αριθμό των ατυχημάτων σε μία διασταύρωση, σε μία συγκεκριμένη χρονική περίοδο. Δίνει την πιθανότητα ακριβώς n επιτυχιών σε N προσπάθειες, αν τα γεγονότα συμβαίνουν ανεξάρτητα και σε σταθερό ρυθμό. Η συνάρτηση πιθανότητας βρίσκεται, αν στην διωνυμική κατανομή θέσουμε το N ότι τείνει στο άπειρο και αν δούμε την κατανομή σαν συνάρτηση του αριθμού των επιτυχιών που αναμένονται ν = N p:

$$P(v)=v^n e^v / n!$$

Η γραφική της παράσταση έχει την παρακάτω μορφή



Η τεχνική που εφαρμόζεται για την δειγματοληψία στη Poisson είναι αυτή της αποδοχής – απόρριψης (acceptance – rejection). Στη περίπτωσή κατανομής με μέσο όρο α, η διαδικασία έχει ως εξής:

I. Τίθεται n=0, P=1
II. Παράγεται ένας αριθμός R_{n+1} και τίθεται $P=P*R_{n+1}$
III. Αν $P<e^{-a}$ τότε γίνεται δεκτό το N=n αλλιώς, απορρίπτεται το n γίνεται το n=n+1 και ο αλγόριθμος πηγαίνει στο βήμα II.

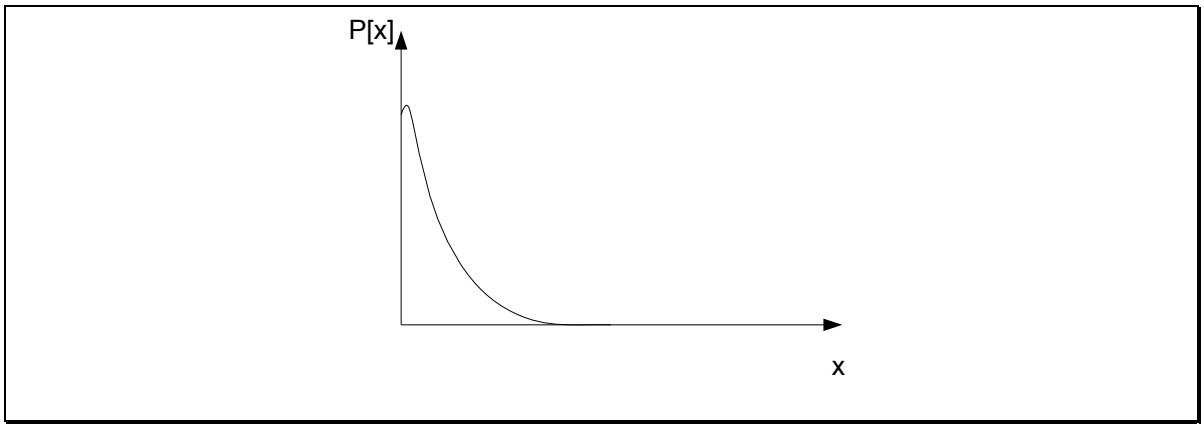
Το n που θα προκύψει τελικά είναι ο ζητούμενος αριθμός N. Για τη διαδικασία αυτή, από τη στιγμή που θα προκύψει ο ζητούμενος αριθμός N, θα χρειαστεί άλλη μία επανάληψη και άρα n + 1 τυχαίοι αριθμοί. Ο μέσος όρος, λοιπόν, φορών που θα καλεστεί η U(0,1) είναι α+1.

1.3.2.8 Κανονική κατανομή (Normal distribution)

Επίσης, γνωστή σαν Gaussian κατανομή. Κάτω από γενικές συνθήκες, ο μέσος όρος των τιμών n παρατηρήσεων, ανεξάρτητα της μορφής της αρχικής κατανομής, πλησιάζει την κανονική κατανομή. Συνεπώς, χρησιμοποιείται πολύ συχνά για μία τεράστια ποικιλία φαινομένων, από αποτελέσματα ψυχολογικών τεστ μέχρι μετρήσεις φωτονίων. Διαφόρων ειδών

λάθη μπορούν να προσεγγιστούν από την κατανομή αυτή. Μία κανονική με μέσο όρο μ και τυπική απόκλιση (standard deviation) σ, έχει την παρακάτω συνάρτηση πιθανότητας και γραφική παράσταση.

$$P(x)=e^{-(x-\mu)^2/(2\sigma^2)}/\sigma\sqrt{2\pi}$$



Για τη παραγωγή αριθμών που να ακολουθούν την κανονική κατανομή χρησιμοποιείται πολύ συχνά όπως αναφέρθηκε η μέθοδος polar. Επιλέγονται δυο αριθμοί ομοιόμορφα κατανεμημένοι x και y ώστε -1 < x < 1 και -1 < y < 1 αλλά και $S = x^2+y^2 <1$ και επιστρέφονται οι απαιτούμενες τιμές $X_1=v_1\sqrt{-2\ln S/S}$ και $X_2=v_2\sqrt{-2\ln S/S}$.

Κατά μέσο όρο η διαδικασία παραγωγής των X_1 and X_2 επαναλαμβάνεται 1,27 φορές.

2. Η ΕΦΑΡΜΟΓΗ Simmate ΚΑΙ Η ΥΛΟΠΟΙΗΣΗ ΤΗΣ

2.1 Γενικά

Σκοπός της παρούσας διπλωματικής εργασίας ήταν η υλοποίηση μίας εφαρμογής προσομοίωσης, η οποία να λειτουργεί μέσα από το διαδίκτυο με την απλή επίσκεψη στην αντίστοιχη σελίδα. Η προσομοίωση που υλοποιήθηκε ήταν η προσομοίωση διακριτού γεγονότος. Χρησιμοποιήθηκε ο μηχανισμός επόμενου γεγονότος (variable time increment) και η τεχνική της λίστας μελλοντικών γεγονότων (future event list).

Βασικός στόχος της εργασίας ήταν η δημιουργία μίας ολοκληρωμένης εφαρμογής. Καθώς η γκάμα των εφαρμογών της προσομοίωσης είναι τεράστια και ήταν αδύνατο να γίνει, στα πλαίσια μίας διπλωματικής εργασίας, η ολοκληρωμένη εφαρμογή σε πολλούς από τους τομείς αυτής, αποφασίστηκε η εφαρμογή της στη θεωρία ουρών με προσπάθεια, όμως, εύκολης επεκτασιμότητας σε πιθανή μελλοντική εργασία. Για τη θεωρία των ουρών κρίθηκε απαραίτητη η υλοποίηση συγκεκριμένων οντοτήτων, όπως της ουράς, της γεννήτριας πελατών, του εξυπηρετητή. Δημιουργήθηκαν, επίσης, βοηθητικές οντότητες, όπως αυτή του μετρητή και της εξόδου, αλλά και μία οντότητα για τη δρομολόγηση πελατών.

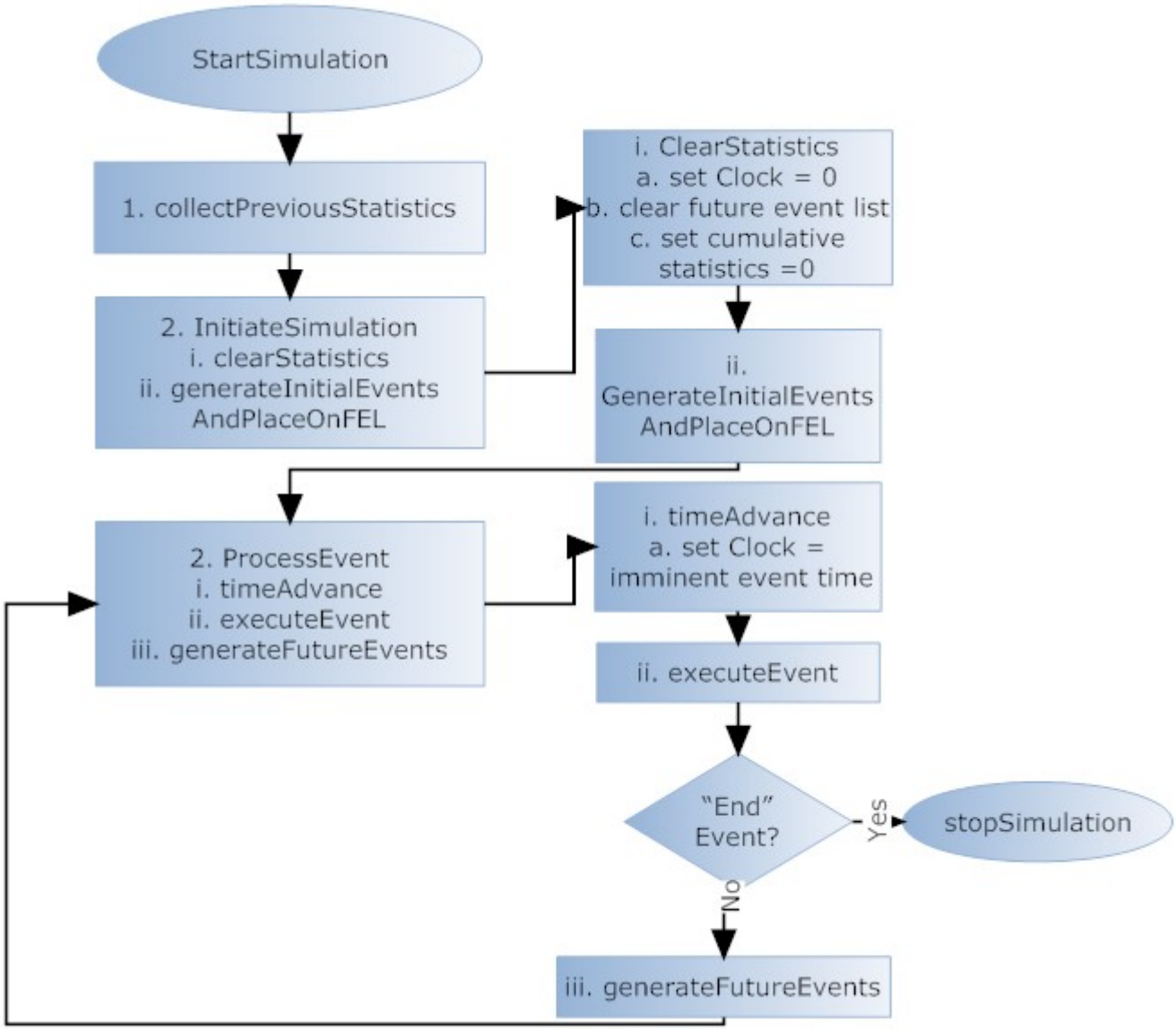
Με δεδομένη τη χρήση του για εκπαιδευτικούς σκοπούς, το simmate περιέχει πληθώρα γραφικών παραστάσεων, είτε για κάθε μεμονωμένη οντότητα είτε για το σύνολο του μοντέλου. Οι γραφικές αυτές παραστάσεις παρουσιάζονται σε πραγματικό χρόνο κατά τη διάρκεια της προσομοίωσης, η οποία μπορεί να γίνει στους ρυθμούς που επιθυμεί ο χρήστης και να παυτεί και να ξεκινήσει ξανά κατά το δοκούν. Ένας χρήστης που δε θέλει να δει τη διαδικασία, αλλά τον ενδιαφέρουν μόνο τα αποτελέσματα, μπορεί, φυσικά, να δει κατευθείαν αυτά, με την πρόσθετη επιλογή να συγκρίνει αποτελέσματα πολλών προσομοιώσεων στη σειρά.

Για την υλοποίησή της εφαρμογής χρησιμοποιήθηκε η γλώσσα προγραμματισμού java. Πιο συγκεκριμένα, η τελευταία έκδοση τη στιγμή που ξεκίνησε η εργασία αυτή, ήταν η Java (TM) 2 SDK, Standard Edition, Version 1.4.2 [http://java.sun.com/j2se/1.4.2] και το περιβάλλον που προτιμήθηκε ήταν η δωρεάν έκδοση του JCreator™ 3.10 LE [http://www.jcreator.com]. Χρησιμοποιήθηκαν επιπλέον κλάσεις της ανοιχτού κώδικα βιβλιοθήκης JDOM για τη δημιουργία του συστήματος αρχείων που έγινε σύμφωνα με τα πρότυπα της XML [http://www.w3.org/xml]. Το σύστημα βοήθειας, τέλος, της εφαρμογής χρησιμοποιεί HTML για τη μορφοποίηση των κειμένων.

2.2 Μηχανισμός επόμενου γεγονότος στο simmate

Είναι φανερό ότι για ένα σύστημα ουρών, στο οποίο οι αφίξεις και οι εξυπηρετήσεις πελατών είναι τα μόνα γεγονότα που μπορούν να αλλάξουν την κατάσταση του συστήματος, η προσομοίωση διακριτού γεγονότος και ο μηχανισμός του επόμενου γεγονότος είναι η κατάλληλη επιλογή και είναι αυτή που χρησιμοποιήθηκε.

Η δομή που υλοποιείται από την κλάση *SimulationRun* φαίνεται στο παρακάτω σχέδιο.



Με την έναρξη της προσομοίωσης (*startSimulation*), θα πρέπει να κρατηθούν όλα τα στατιστικά της προηγούμενης προσομοίωσης σε λίστες (*collectPreviousStatistics*). Έτσι, ο χρήστης θα μπορεί να κάνει συγκρίσεις μεταξύ πολλών προσομοιώσεων, αλλά και θα έχει στη διάθεση του σημαντικά στατιστικά, όπως ο μέσος όρος και η τυπική απόκλιση κάθε στατιστικού που τον ενδιαφέρει. Αμέσως μετά, η προσομοίωση αρχικοποιείται (*initiateSimulation*). Η αρχικοποίηση περιλαμβάνει την εκκαθάριση κάθε στατιστικού (*clearStatistics*), και τη δημιουργία όλων των αρχικών γεγονότων, τα οποία και θα τοποθετηθούν στην κατάλληλη θέση στη λίστα (*generateInitialEventsAndPlaceOnFEL*). Στην εκκαθάριση των στατιστικών, το ρολόι του συστήματος (clock) θα πρέπει να μηδενιστεί και όλα γενικά τα στατιστικά του μοντέλου, αλλά και κάθε ξεχωριστού αντικειμένου, πρέπει να αρχικοποιηθούν, ώστε να είναι έτοιμα για μέτρηση των νέων δεδομένων. Για τη δημιουργία των αρχικών γεγονότων θα πρέπει να προσπελαστεί όλη η λίστα των αντικειμένων του μοντέλου και για κάθε γεννήτρια, που είναι και το μόνο αντικείμενο που αρχικά μπορεί να προκαλέσει γεγονός, να υπολογιστεί ένας χρόνος άφιξης, ανάλογα με την κατανομή που ακολουθεί. Το κάθε γεγονός που θα δημιουργηθεί θα μπει στη λίστα μελλοντικών γεγονότων (*FutureEventList*) στη σωστή θέση, ανάλογα με το χρόνο που αναμένεται να συμβεί.

Στη συνέχεια επεξεργάζεται το πρώτο χρονικά γεγονός στη λίστα, το επικείμενο γεγονός (processEvent), διαδικασία που περιλαμβάνει τρία βήματα. Αρχικά, ο χρόνος πρέπει να προωθηθεί στο χρόνο του γεγονότος (timeAdvance) και στη συνέχεια αυτό να εκτελεστεί. Κατά την εκτέλεση (*executeEvent*) εξετάζεται αν το γεγονός αυτό είναι γεγονός τερματισμού, που σημαίνει ότι η προσομοίωση θα πρέπει να τελειώσει στο σημείο αυτό (*stopSimulation*), ανεξάρτητα αν υπάρχουν και άλλα γεγονότα στη λίστα. Εάν δεν είναι, καλείται η εκτέλεση για το συγκεκριμένο αντικείμενο και στη συνέχεια δημιουργούνται τα νέα γεγονότα (*generateFutureEvent*) που προκλήθηκαν από το γεγονός που μόλις εκτελέστηκε. Στη συνέχεια, επεξεργάζεται το επόμενο γεγονός στη λίστα (*processEvent*) και η διαδικασία συνεχίζεται, μέχρι να βρεθεί γεγονός τερματισμού της προσομοίωσης.

Στην συνέχεια εξηγείται ο τρόπος αποθήκευσης και χειρισμού της λίστας επόμενου γεγονότος για τον οποίο επιλέχθηκε ο πιο απλός, αλλά και λιγότερο αποδοτικός τρόπος από αυτούς που εξηγήθηκαν στην παράγραφο 1.2.3. Η μέθοδος της διπλά συνδεδεμένης λίστα (single linear event list). Για την υλοποίηση της, το βασικό συστατικό είναι μία λίστα τύπου LinkedList, η *FEList*, της κλάσης *FutureEventList*, η οποία κρατάει κάθε μελλοντικό γεγονός, το οποίο είναι της μορφής *FutureEvent*. Στην τελευταία αυτή κλάση κρατείται ο χρόνος, *time*, στον οποίο το γεγονός πρόκειται να συμβεί, ο τύπος του γεγονότος, *event*, και το αντικείμενο το οποίο προκάλεσε το γεγονός, *objectsource*. Κάθε φορά που ένα γεγονός πρόκειται να προστεθεί στη λίστα, αυτό θα πρέπει να μπει στο κατάλληλο χρονικά σημείο. Η προσθήκη γίνεται με τη βοήθεια της συνάρτησης *addToList* που καλεί με τη σειρά της την *returnIndex* της *FutureEventList*. Η *returnIndex* επιστρέφει την κατάλληλη θέση του νέου γεγονότος στη λίστα, αφού τη διατρέχει από τα πιο σύντομα γεγονός προς τα μεταγενέστερα. Στη περίπτωση που δε βρεθεί

```
public int returnIndex(double time){
        FutureEvent currentfe;
        for(int i=0;i<FEList.size();i++){
                currentfe = (FutureEvent)FEList.get(i);
                if (currentfe.getTime()>time)
                        return i;
        }
        return FEList.size();
}
```

Η αφαίρεση του επικείμενου γεγονότος είναι πολύ απλή και γίνεται με την απλή αφαίρεση από τη λίστα των επόμενων γεγονότων του πρώτου στοιχείου στη συνάρτηση *removeFromList* της ίδιας κλάσης που διαχειρίζεται κάθε πράξη επί της λίστας. Χωρίς να είναι απαραίτητο για την υλοποίηση του αλγορίθμου, αλλά χρήσιμο για επεξηγηματικούς λόγους, έχει προστεθεί μία λίστα παρελθόντων γεγονότων, με τη βοήθεια της οποίας μπορούν να φανούν όλα τα γεγονότα της προσομοίωσης, είτε έχουν συμβεί, είτε πρόκειται να συμβούν. Έτσι, σε κάθε αφαίρεση του επικείμενου γεγονότος γίνεται μία πράξη πρόσθεσης παρελθόντος γεγονότος. Από τη στιγμή που το simmate δημιουργήθηκε σαν εκπαιδευτικό και όχι ανταγωνιστικό πρόγραμμα, κρίθηκε χρήσιμη μία επεξηγηματική λειτουργία που επιβραδύνει λίγο τον αλγόριθμο σε παράγοντες υπολογιστικών πόρων, αλλά διευκολύνει πολύ έναν μελετητή της προσομοίωσης. Η συγκεκριμένη συνάρτηση παίρνει τη παρακάτω μορφή.

```
public FutureEvent removeFromList(){
```

```
        FutureEvent fe;
        fe = (FutureEvent)FEList.remove(0);
        PEList.add(fe);
        return fe;
}
```

2.3 Παραγωγή τυχαίων αριθμών

Η εφαρμογή χρησιμοποιεί την κλάση Random [http://java.sun.com/j2se/1.4.2/docs/api/] και τη συνάρτησή της *nextDouble()* για την παραγωγή των τυχαίων, ομοιόμορφα κατανεμημένων, αριθμών στο (0,1), από τους οποίους προκύπτουν και όλες οι υπόλοιπες κατανομές τυχαίων αριθμών. Η περίοδος της συνάρτησης αυτής, μέχρι να αρχίσει να παράγει την ίδια ακολουθία αριθμών είναι 2^{48}. Η συνάρτηση αυτή χρησιμοποιεί τη συνάρτηση ($X_{\kappa+1}= (\alpha X_\kappa +c) \bmod m\ \kappa\geq 0$) Lehmer, που αναφέρθηκε σε προηγούμενο κεφάλαιο.

Το πρόγραμμα java που χρησιμοποιείται από την nextDouble() είναι το εξής:

```
public double nextDouble() {
    return (((long)next(26) << 27) + next(27))
      / (double)(1L << 53);
 }
```

Όπως έχει ειπωθεί το $X_{\kappa+1}$ είναι το τρέχον προς παραγωγή νούμερο, το X_κ είναι το προηγούμενο και X_0 το πρώτο νούμερο ή σπόρος. α είναι ο πολλαπλασιαστής, c η προσαύξηση και m το ακέραιο υπόλοιπο. Η επιλογή όλων αυτών των μεταβλητών είναι το μυστικό για την παραγωγή τυχαίων αριθμών. Η εφαρμογή αφήνει όλα τα υπόλοιπα στην sun, εκτός του σπόρου (seed), για τον οποίο έχει αφεθεί κάποια επιλογή στην αντιμετώπιση από το χρήστη.

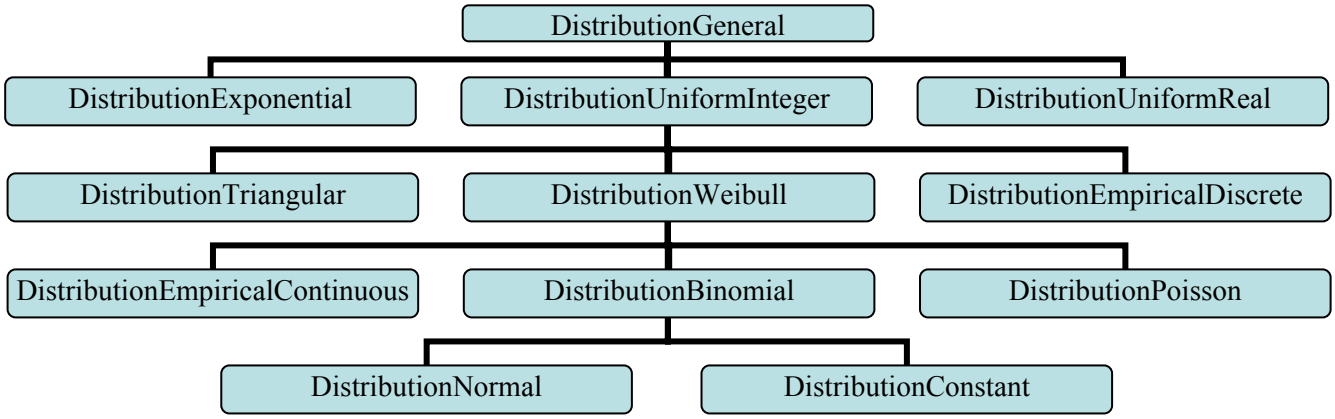
2.3.1 Ο σπόρος (seed)

Εξ ορισμού, η κλάση Random χρησιμοποιεί έναν σπόρο που προκύπτει από το ρολόι του υπολογιστή. Υπάρχει, όμως, η μέθοδος *setSeed (long seed)*, με την οποία μπορεί να τεθεί άλλος σπόρος. Η εφαρμογή χρησιμοποιεί αυτή τη μέθοδο, με την οποία ο χρήστης μπορεί να θέσει κάποιον δικό του σπόρο. Είναι προφανές ότι, αν δύο στιγμιότυπα της κλάσης Random έχουν δημιουργηθεί με τον ίδιο σπόρο και έχει γίνει η ίδια ακολουθία κλήσεων της *nextDouble()* μεθόδου, θα παράγουν και θα επιστρέψουν πανομοιότυπες ακολουθίες αριθμών. Αυτή η συμπεριφορά μπορεί να είναι πολύ χρήσιμη, όταν, για παράδειγμα, θα θέλουμε να συγκρίνουμε δυο διαφορετικά συστήματα. Μπορεί να γίνει η παραγωγή των ίδιων ακριβώς αριθμών για κάθε αντικείμενο που δεν είναι προς σύγκριση και να μελετηθεί η συμπεριφορά του υπόλοιπου συστήματος.

2.3.2 Κατανομές

Για την υλοποίηση όλων των κατανομών, σημείο αφετηρίας είναι η υλοποιημένη στη γλώσσα προγραμματισμού ομοιόμορφη κατανομή στο (0,1) και στόχος μία καλή προσέγγιση της συμπεριφοράς κάθε κατανομής. Οι πιο διαδεδομένες

μέθοδοι αναφέρθηκαν στο πρώτο κεφάλαιο της παρούσης και υλοποιήθηκαν στις κλάσεις του προγράμματος που αφορούν τις κατανομές, οι οποίες όλες κληρονομούν ιδιότητες τους από την κλάση *DistributionGeneral*.



Συναρτήσεις που περιέχονται στην *DistributionGeneral* και υπερφορτώνωνται από τις παραπάνω κλάσεις αφορούν στην αρχικοποίηση των αξόνων όλων των υποστηριζόμενων διαγραμμάτων κατανομών. Έτσι, για τη συνάρτηση κατανομής πιθανότητας (PDF) και την ποσοστιαία παρουσίαση της, υλοποιείται η *initiatePDFPercentageAxisXNumbers()* για τον οριζόντιο άξονα, η *initiateCDFPercentageAxisYNumbers()* για τον κάθετο άξονα και η *initiatePDFPercentageAxisNumbers()* και για τους δυο άξονες που ουσιαστικά καλεί τις άλλες δυο. Για τη μη ποσοστιαία παρουσίαση της συνάρτησης, οι αρχικοποιήσεις αντίστοιχα γίνονται με τις *initiatePDFAxisXNumbers()*, *initiatePDFAxisYNumbers() και initiatePDFAxisNumbers()*. Όσον αφορά στη συσωρευτική συνάρτηση (CDF), οι συναρτήσεις για την ποσοστιαία παρουσίαση στους αντίστοιχους άξονες είναι οι *initiateCDFPercentageAxisXNumbers(),initiatePDFPercentageAxisYNumbers()* και *initiateCDFPercentageAxisNumbers()*, ενώ η αρχικοποίηση της μη ποσοστιαίας παρουσίασης γίνεται με τις *initiateCDFAxisXNumbers()*, *initiateCDFAxisYNumbers()* και *initiateCDFAxisNumbers()*. Ακόμη, στην κλάση αυτή υλοποιείται ο σχεδιασμός όλων αυτών των διαγραμμάτων με τις συναρτήσεις *paintPDFPercentageDiagram(Graphics2D g2,int []intervalsarray)* για την ποσοστιαία παρουσίαση της συνάρτησης κατανομής πιθανότητας και *paintPDFDiagram(Graphics2D g2,int []intervalsarray)* για την παρουσίασή της σε απόλυτους αριθμούς, *paintCDFPercentageDiagram(Graphics2D g2,int []intervalsarray)* και *paintCDFDiagram(Graphics2D g2,int []intervalsarray)* για τις αντίστοιχες συσωρευτικές κατανομές πιθανότητας. Όλες οι προαναφερθείσες συναρτήσεις αφορούν λοιπόν στο σχεδιασμό διαγραμμάτων. Σημαντική δουλειά μεν για την παρουσίαση των στατιστικών, αλλά όχι η κύρια δουλειά των κατανομών.

Η συνάρτηση στην οποία θα πρέπει να υλοποιηθεί η παραγωγή των χρόνων των αφίξεων και εξυπηρετήσεων που θα ακολουθούν κάποια από τις υλοποιημένες κατανομές είναι η *generateInterarrivalTime()*. Η συνάρτηση αυτή δεν υλοποιείται στην κλάση *DistributionGeneral*, αλλά σε όλες τις κλάσεις που κληρονομούν από αυτή, σε κάθε μία με διαφορετικό τρόπο και μέθοδο που εξηγείται στη συνέχεια. Ορίζεται δε στην κλάση *DistributionGeneral* για λόγους πολυμορφισμού. Το διάγραμμα κλάσης (class diagram) [Dennis, Wixom and Tegarden, 2002] της οποίας φαίνεται στη συνέχεια.

2.3.2.1 Εκθετική κατανομή

Το simmate χρησιμοποιεί την τεχνική της αντίστροφης μεταμόρφωσης, για να παράγει αριθμούς που ακολουθούν την κατανομή αυτή στην κλάση *DistributionExponential*. Χρησιμοποιώντας τη γνωστή ομοιόμορφη κατανομή στο (0,1) και τους αριθμούς R_1, R_2, R_3.... που αυτή παράγει, οι επιθυμητές μεταβλητές X_1,X_2,X_3.... δίνονται από τον τύπο:

$X_1 = -1/\lambda \ln(1-u)$

Και με τη χρήση της απλοποίησης, αντικαθίσταται το 1 – u με το u και ο τύπος που χρησιμοποιείται από το simmate για την παραγωγή εκθετικά κατανεμημένων μεταβλητών γίνεται:

$X_1 = -1/\lambda \ln(u)$

Έτσι, στη συνάρτηση *generateInterarrivalTime()* που είναι υπεύθυνη να επιστρέψει το χρόνο που αντιστοιχεί στην κατανομή αυτή, η εξίσωση που δίνει τον χρόνο γίνεται

*time = -(mean)*Math.log(activity.random.nextDouble());*

Στην εξίσωση, mean είναι ο μέσος (expected value, mean ή mathematical expectation) που είναι και mean = λ^{-1}, το Math.log είναι συνάρτηση της java που δίνει το νεπέριο λογάριθμο του *activity.random.nextDouble()* που επιστρέφει το u, τον επόμενο αριθμό της ακολουθίας της ομοιόμορφης κατανομής στο (0,1).

2.3.2.2 Ομοιόμορφη κατανομή

Για την παραγωγή ομοιόμορφα κατανεμημένων αριθμών η εφαρμογή χρησιμοποιεί δυο κλάσεις, μία για παραγωγή ακεραίων DistributionUniformInteger, και μία για παραγωγή πραγματικών τιμών DistributionUniformReal. Η τεχνική που χρησιμοποιείται είναι η ίδια με αυτή της εκθετικής κατανομής, η τεχνική της αντίστροφης μεταμόρφωσης. . Χρησιμοποιώντας και πάλι τη γνωστή ομοιόμορφη κατανομή στο (0,1) και τους αριθμούς R_1, R_2, R_3.... που αυτή παράγει, οι επιθυμητές μεταβλητές X_1,X_2,X_3.... δίνονται από τον τύπο:

$X_i = a + (b-a)R$

Στη συνάρτηση λοιπόν, *generateInterarrivalTime* ο τύπος που θα χρησιμοποιηθεί στην *DistributionUniformReal* γίνεται:

```
time = min + activity.random.nextDouble()*(max - min);
```

Με την ελάχιστη διαφοροποίηση της στρογγυλοποίησης για τις ακέραιες τιμές της *DistributionUniformInteger* ο τύπος γίνεται:

```
time = Math.round(min + activity.random.nextDouble()*(max - min));
```

Στις εξισώσεις αυτές, min είναι η ελάχιστη τιμή a της κατανομής, max είναι η μέγιστη τιμή b και *activity.random.nextDouble()* η γνωστή συνάρτηση που επιστρέφει την επόμενη τιμή της ομοιόμορφης κατανομής στο (0,1).

2.3.2.3 Τριγωνική κατανομή

Στην κατανομή αυτή που υλοποιείται στην *DistributionTriangular* κλάση και με τη χρήση και πάλι της αντίστροφης μεταμόρφωσης, θέτοντας τη συνάρτηση πιθανότητας ίση με R και λύνοντας ως προς X όπως εξηγείται στην παράγραφο 1.3.2.3 προκύπτει ο επόμενος τύπος

$$X=\begin{cases}\sqrt{(b-a)(c-a)*R}+a, & a\le R\le(b-a)/(c-a)\\ c-\sqrt{(1-R)(c-b)(c-a)}, & (b-a)/(c-a)<R\le c\end{cases}$$

Μπορεί έτσι εύκολα να δημιουργηθεί η ζητούμενη ακολουθία X_1, X_2, X_3,… με τη βοήθεια της R_1, R_2, R_3,… που προκύπτει από την *activity.random.nextDouble()* με το ακόλουθο τμήμα κώδικα

```
r = activity.random.nextDouble();
if (r<=((mostlikely - min)/(max - min))){
        time = Math.sqrt(((mostlikely - min)*(max - min))*r)+min;
}
else{
        time = max - Math.sqrt((1-r)*(max-mostlikely)*(max - min));
}
```

Η μεταβλητή *mostlikely* στον παραπάνω κώδικα είναι η πιο πιθανή περίπτωση της τριγωνικής κατανομής *b*, η *min*, η ελάχιστη πιθανή τιμή *a* και *max* η μέγιστη πιθανή τιμή *c*.

2.3.2.4 Κατανομή weibull

Δουλεύοντας με τον ίδιο τρόπο, όπως και στις προηγούμενες περιπτώσεις, στην κλάση DistributionWeibull βρίσκουμε τον τύπο που δίνει την επόμενη τιμή της κατανομής σαν συνάρτηση της επόμενης τιμής της ομοιόμορφης στο (0,1) κατανομής

$$X = \alpha[-\ln(1-R)]^{1/\beta}$$

Το κομμάτι κώδικα που υλοποιεί την παραπάνω παράσταση στη συνάρτηση generateInterarrivalTime

```
time = a*Math.pow(-Math.log(activity.random.nextDouble()),1/b);
```

Στο οποίο η μεταβλητή a είναι η παράμετρος scale και b είναι η παράμετρος shape. Οι συναρτήσεις Math.pow και Math.log της java επιστρέφουν τη δύναμη και τον νεπέριο λογάριθμο αντίστοιχα.

2.3.2.5 Εμπειρικές κατανομές

Η παραγωγή εμπειρικά κατανεμημένων ακολουθιών αριθμών υλοποιείται από την *DistributionEmpiricalDiscrete* στη διακριτή της έκδοση και από την *DistributionEmpiricalContinuous* στην συνεχή της έκδοση. Καθώς υλοποιείται η πιο απλή μορφή τους, όπου κάθε διάστημα μπορεί να περιγραφεί από μία ξεχωριστή ομοιόμορφη κατανομή, εφαρμόζεται η τεχνική της αντίστροφης μεταμόρφωσης, όπως ακριβώς και στην ομοιόμορφη κατανομή με τη διαφοροποίηση ότι πρέπει να προστεθεί ένα πρώτο στάδιο, στο οποίο επιλέγεται το διάστημα στο οποίο θα γίνει η διαδικασία.

Με την παραγωγή ενός αριθμού R_i, ομοιόμορφα κατανεμημένου στο (0,1), επιλέγεται το διάστημα και με νέα παραγωγή αριθμού R ο ίδιος αριθμός στο επιλεγμένο διάστημα. Ο τύπος που δίνει τον εμπειρικά κατανεμημένο αριθμό από κάθε αριθμό R_i είναι ο $X_i = a + (b-a)R_i$. Στο πρόγραμμα ο τύπος αυτός υλοποιείται στη γραμμή

```
time = element.getMinTime() + r*(element.getMaxTime() - element.getMinTime());
```

Όπου element είναι ένα από τα κομμάτια που απαρτίζουν την εμπειρική κατανομή, υλοποιείται από την κλάση EmpiricalTableElementContinuous (EmpiricalTableElementDiscrete στην περίπτωση διακριτής κατανομής) και από μόνο του είναι μία ομοιόμορφη κατανομή, της οποίας το ελάχιστο και το μέγιστο δίνεται από τις συναρτήσεις getMinTime() και getMaxTime(). Για να βρεθεί ποιο από τα κομμάτια θα επιλεχθεί, όπως φαίνεται στον κώδικα, διατρέχεται όλη η λίστα των επιμέρους κατανομών και ανάλογα και με την πιθανότητα της κάθε μίας (getProbability()) βρίσκει πού πρέπει να ανήκει ο αριθμός R.

```
double r = activity.random.nextDouble();
for(int i=0;i<activity.empiricalcontinuouslist.size();i++){
        element = (EmpiricalTableElementContinuous)activity.empiricalcontinuouslist.get(i);
        currentprobability = element.getProbability();
        if ((r<=(prevprobability+currentprobability)/totalprobability) &&((r>=prevprobability/totalprobability))){
                choice = i;
                break;
        }
        prevprobability+=currentprobability;
}
element = (EmpiricalTableElementContinuous)activity.empiricalcontinuouslist.get(choice);
```

r = activity.random.nextDouble();
time = element.getMinTime() + r*(element.getMaxTime() - element.getMinTime());

2.3.2.6 Διωνυμική κατανομή

Για την παραγωγή τυχαίων μεταβλητών που ακολουθούν τη διωνυμική κατανομή, χρησιμοποιήθηκαν στο simmate δύο διαφορετικοί αλγόριθμοι που υλοποιούνται στην κλάση *DistributionBinomial*. Αν το n (ο αριθμός των προσπαθειών Bernoulli – Bernoulli trials) πολλαπλασιασμένο με το ελάχιστο των p και 1 – p (η πιθανότητα επιτυχίας ή αποτυχίας) είναι μικρότερο από 10, τότε εφαρμόζεται ένας απλός αλγόριθμος βασισμένος στο γεγονός ότι η διωνυμική κατανομή είναι ένα σύνολο από προσπάθειες Bernoulli. Αρχικά, ένας μετρητής και ο ζητούμενος χρόνος τίθενται μηδέν. Σε μία επανάληψη, κατά την οποία ο μετρητής αυξάνεται κατά 1, παράγεται με το γνωστό τρόπο ένας ομοιόμορφα κατανεμημένος τυχαίος αριθμός στο (0,1) u και, αν ο u είναι μικρότερος ή ίσος με τη πιθανότητα p, τότε και ο χρόνος αυξάνει κατά ένα. Η επανάληψη αυτή εκτελείται n φορές και στην ουσία εκτελεί τόσες φορές τις προσπάθειες Bernoulli.

```
if (n*r<10){
        do{
                u = activity.random.nextDouble();
                k++;
                if (u<=p){
                        time++;
                }
        }while (k<n);
}
```

Ο τρόπος αυτός, αν και ακριβής, είναι πολύ αργός στην περίπτωση που έχουμε να κάνουμε με πολύ μεγαλύτερα n. Έτσι, στη περίπτωση που το n, πολλαπλασιασμένο με το ελάχιστο των p και 1 – p, είναι μεγαλύτερο από 10, υλοποιείται ο αλγόριθμος των Schmeister και Kachitvichyanukul [Kachitvichyanukul, 1988], του οποίου η υλοποίηση αναφέρεται στην παράγραφο 1.3.2.6

2.3.2.7 Κατανομή Poisson

Στην κατανομή αυτή που είναι διακριτή η τεχνική που χρησιμοποιήθηκε για να παραχθεί κάθε φορά ο επόμενος αριθμός, είναι η τεχνική της αποδοχής – απόρριψης (acceptance – rejection), κατά την οποία γίνεται η παραγωγή κάθε φορά του επόμενου ομοιόμορφα κατανεμημένου στο (0,1) αριθμού R και αν ισχύει κάποια συνθήκη τότε γίνεται ο αριθμός δεκτός. Σύμφωνα με τη διαδικασία της παραγράφου 1.3.2.7, ο αλγόριθμος που προκύπτει δίνεται στη συνέχεια

```
n ← 0
P ← 1
Όσο P>e^-a
        u ← U(0,1)
        P ← P *u
```

n ← n + 1

Τέλος επανάληψης

Στην κλάση *DistributionPoisson* που υλοποιεί την κατανομή αυτή, η συνάρτηση generateInterarrivalTime που επιστρέφει την ακολουθία των αριθμών, φαίνεται στη συνέχεια.

```
public double generateInterarrivalTime(){
        double elambda = Math.exp(-1*mean);
        double product = 1;
        int count =  0;
        int time=0;
        while (product >= elambda) {
                product *= activity.random.nextDouble();
                time = count;
                count++;
        }
        double result =(new Integer(time)).doubleValue();
        last_interval = (int)(da.simulationinfo.getTimeIntervals()*((result-intervalmin)/(intervalmax-intervalmin)));
        return result;
}
```

2.3.2.8 Κανονική κατανομή

Η κατανομή αυτή υλοποιήθηκε με τη μέθοδο polar, της οποίας η υλοποίηση στη γλώσσα java φαίνεται στη συνέχεια. Οι v1 και v2 μεταβλητές είναι ομοιόμορφα κατανεμημένες στο (-1,1) αφού *v1 = 2*u1 – 1* και *v2 = 2*u2 – 1* όπου u1 και u2 προκύπτουν από την *activity.random.nextDouble()*. Το S το οποίο είναι ίσο με το άθροισμα των τετραγώνων των v1 και v2, *S = Math.pow(v1,2)+Math.pow(v2,2)* γίνεται δεκτό μόνον αν είναι μεγαλύτερο της μονάδας. Αν ισχύει *S>=1* ο αλγόριθμος βγαίνει από τον βρόχο και υπολογίζονται οι δυο τιμές X1 και X2 σύμφωνα με τους τύπους $X_1=v_1\sqrt{-2\ln S/S}$ και $X_2=v_2\sqrt{-2\ln S/S}$ που υλοποιούνται στον κώδικα *X1 = v1 * Math.sqrt(-2*Math.log(S)/S)* ή *X2 = v2 * Math.sqrt(-2*Math.log(S)/S)*. Με τη χρήση μίας flag μεταβλητής, της use_last, και οι δύο τιμές χρησιμοποιούνται σε δύο στη σειρά εκτελέσεις του αλγορίθμου.

```
if (use_last==1){
        X1 = X2;
        use_last = 0;
}
else{
        do{
                u1 = activity.random.nextDouble();
                u2 = activity.random.nextDouble();
```

```
            v1 = 2*u1 - 1;
            v2 = 2*u2 - 1;
            S = Math.pow(v1,2)+Math.pow(v2,2);
        }while (S>=1);
        use_last = 1;
}
time = mean  + X1 * stddev;
```

2.3.2.9 Σταθερή κατανομή

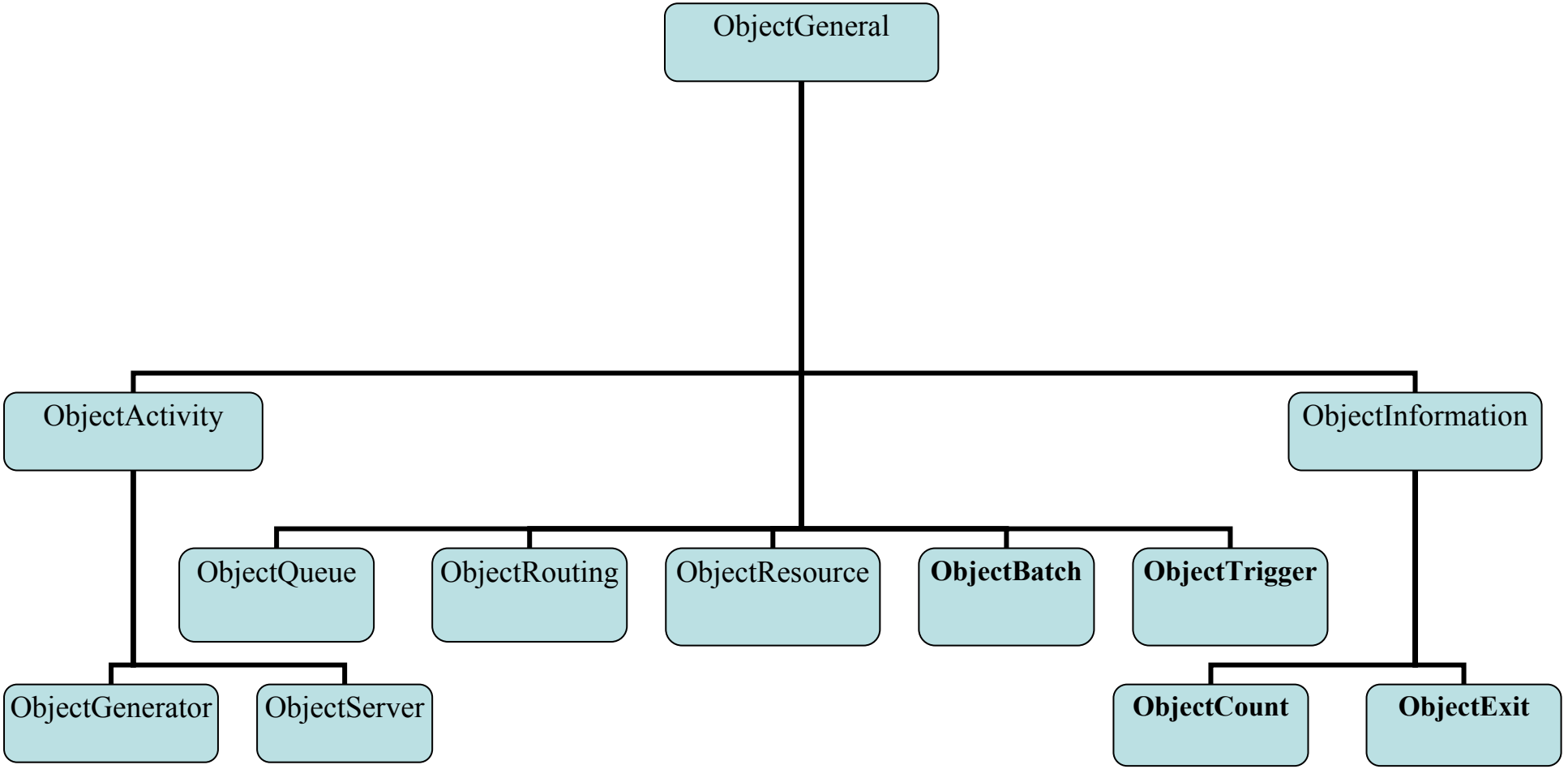
Η περίπτωση αυτή είναι η μόνη που αφορά προσδιοριστικούς χρόνους και στην ουσία ο όρος στον τίτλο του υποκεφαλαίου είναι αδόκιμος. Στην ουσία, στη συνάρτηση *generateInterarrivalTime* της κλάσης *DistributionConstant* το μόνο που πρέπει να γίνει είναι να επιστραφεί η τιμή της σταθεράς που έχει δοθεί από τον χρήστη.

2.4 Αντικείμενα στο Simmate

Στην παρούσα πρώτη έκδοση του simmate έχουν υλοποιηθεί εννέα διαφορετικά αντικείμενα, τα οποία βρίσκουν εφαρμογή σε μοντέλα συστημάτων ουρών. Τα αντικείμενα αυτά είναι τα εξής:

- Μετρητές (Count)
- Έξοδοι (Exit)
- Γεννήτριες (Generator)
- Εξυπηρετητές (Server)
- Ουρές (Queue)
- Δρομολογητές (Router)
- Παρτίδες (Batch)
- Πηγές (Resource)
- Σκανδάλες (Trigger)

Για όλα αυτά τα αντικείμενα, τα οποία θα εξηγηθούν στη συνέχεια, υπάρχουν κάποιες κοινές λειτουργίες, για τις οποίες κρίθηκε απαραίτητη η δημιουργία μίας αρχικής κλάσης, από την οποία θα κληρονομούσαν τις κοινές αυτές λειτουργίες. Ειδικά οι γεννήτριες και οι εξυπηρετητές μοιράζονται πολλές λειτουργίες που αφορούν την παραγωγή τυχαίων αριθμών που θα ακολουθούν κάποια συγκεκριμένη κατανομή και για αυτό κληρονομούν τις λειτουργίες αυτές από μία ξεχωριστή κλάση. Επιπρόσθετα, οι μετρητές και οι έξοδοι έχουν πολλές κοινές ιδιότητες που αφορούν μετρήσεις, ιδιότητες που κληρονομούνται με τη σειρά τους από μία τρίτη κλάση. Ολόκληρη η ιεραρχία που αφορά τα αντικείμενα φαίνεται στο διάγραμμα της επόμενης σελίδας.


ObjectGeneral
ObjectActivity
ObjectInformation
ObjectQueue
ObjectRouting
ObjectResource
ObjectBatch
ObjectTrigger
ObjectGenerator
ObjectServer
ObjectCount
ObjectExit

Η αρχική κλάση *ObjectGeneral* κληρονομεί από την κλάση *JComponent*, η οποία προσφέρει σε όλα τα αντικείμενα διάφορες ιδιότητες απαραίτητες για τη διεπαφή. Για παράδειγμα, με μία απλή χρήση της συνάρτησης *setCursor*, ο κέρσορας, όταν βρίσκεται πάνω από ένα οποιαδήποτε αντικείμενο, μετατρέπεται σε χέρι για να καταλάβει ο χρήστης ότι το αντικείμενο μπορεί να μετακινηθεί. Επιπρόσθετα, κανένας υπολογισμός δε χρειάζεται να γίνει προγραμματιστικά για την εύρεση της θέσης των αντικειμένων αυτών, κάτι που γίνεται αυτόματα από την java για κάθε αντικείμενο που κληρονομεί από τη συγκεκριμένη κλάση. Αυτό προϋποθέτει όμως και την προσθήκη, η αφαίρεση στην περιοχή του μοντέλου κάθε φορά που δημιουργείται ή διαγράφεται αντίστοιχα του αντικειμένου με τις συναρτήσεις *addToArea()* και *removeFromArea()*. Κάθε φορά, επίσης, που ο χρήστης επιθυμεί να μετακινήσει ή να αλλάξει το μέγεθος του αντικειμένου μια σειρά από συναρτήσεις πρέπει να καλεστούν, οι changeShapes(), moveObject(int newpointx,int newpointy), createEmptyEllipses() και calculateConnectors().

Η κλάση *ObjectGeneral* περιέχει όλες τις λειτουργίες που έχουν να κάνουν με τη διεπαφή, όπως τις λειτουργίες του ποντικιού (*mouseClicked(MouseEvent e)*, *mouseReleased(MouseEvent e)*, *mouseExited(MouseEvent e)*, *mouseEntered(MouseEvent e)*, *mouseDragged(MouseEvent e)*, *mouseMoved(MouseEvent e)*), που είναι κοινές για όλα τα αντικείμενα αλλά και τις λειτουργίες σχεδίασης των αντικειμένων (*paintComponent(Graphics g)*, *paintExterior(Graphics2D g2)*, *paintInterior(Graphics2D g2)*). Ακόμα, περιέχει συναρτήσεις που θέτουν και επιστρέφουν κάποιες κοινές μεταβλητές, όπως το αν το όνομα θα είναι ορατό ή όχι, *getNameVisible()* και *setNameVisible(boolean visible)*, και αν το αντικείμενο είναι επιλεγμένο, *isSelected()* και *setSelected(boolean set)*. Παρέχει συναρτήσεις που καθορίζουν ή επιστρέφουν τους συνδέσμους κάθε αντικειμένου *setHasInput(boolean input)*, *getHasInput()*, *setHasOutput(boolean output)* και *getHasOutput()* και ανάλογα με το τι περιέχουν δημιουργούν και τους συνδέσμους τους, *createEmptyEllipses()*. Επίσης παρέχει τις συναρτήσεις για τη διαχείριση των συνδέσμων και των συνδέσεων του αντικειμένου, *getinputConnector()*, *getoutputConnector()*, *gethelpinConnector()*, *addOutputConnection(Connection connection)*, *addInputConnection(Connection connection)*, *addHelpInConnection(Connection connection)*, *removeOutputConnection(Connection connection)*, *removeInputConnection(Connection connection)*, *removeHelpInConnection(Connection connection)*. Οι συναρτήσεις αυτές στην εξορισμού λειτουργία τους είναι απλές και απλά αφαιρούν ή προσθέτουν την σύνδεση στη κατάλληλη λίστα του αντικειμένου. Με' αυτόν τον τρόπο ορίζονται στην αρχική κλάση αλλά κάποιες κλάσεις όπως αυτή των δρομολογητών απαιτούν μια πιο εξεζητημένη λειτουργία που θα εξηγηθεί στη συνέχεια και ορίζεται διαφορετικά στην αντίστοιχη κλάση.

Κάποιες κοινές ιδιότητες όλων των αντικειμένων όπως ο τύπος του κάθε αντικειμένου, αν είναι δηλαδή για παράδειγμα γεννήτρια ή εξυπηρετητής, *getType()* και *setType(String type)* περιέχονται επίσης στην αρχική κλάση αντικειμένου *ObjectGeneral*. Άλλες τέτοιες ιδιότητες που υλοποιούνται από συναρτήσεις στη κλάση αυτή είναι οι δυο συναρτήσεις που θέτουν και επιστρέφουν το όνομα του αντικειμένου, *setObjectName(String name)* και *getObjectName()* αλλά και οι δυο κλάσεις που θέτουν και επιστρέφουν τα πιθανά σχόλια που συνοδεύουν κάθε αντικείμενο, *getObjectComments()* και *setObjectComments(String comments)*. Επίσης, περιέχει συναρτήσεις που θέτουν το μέγεθος του αντικειμένου στο μοντέλο, *setObjectSize(double newsize,double oldsize)* και *setObjectSize(double size)* αλλά και το επιστρέφουν, *getObjectSize()* και επιστρέφουν το σημείο που βρίσκεται σε αυτό, *getPoint()*, καθώς επίσης και το τετράγωνο που αυτό καταλαμβάνει, *getRectangle()*. Μια επιπλέον ιδιότητα του αντικειμένου είναι η εμφάνισή του που στις περισσότερες περιπτώσεις αναπαρίσταται με τη βοήθεια κάποιας εικόνας από κάποιον πίνακα του οποίου ο δείκτης,

imageindex τίθεται η επιστρέφεται με τις συναρτήσεις *getImageIndex()* και *setImageIndex(int i)*. Η ίδια η εικόνα, για λόγους, πάλι, παρουσίασης επιστρέφεται με τη συνάρτηση *returnObjectIcon()*. Στην προσπάθεια καλύτερης δρομολόγησης χρησιμοποιήθηκε μια κλάση που βαθμολογεί κάθε αντικείμενο ανάλογα με την κατάστασή του, η getGrade(double grade), της οποίας η χρήση εξηγείται και στο αντικείμενο που υλοποιεί τους αλγορίθμους αυτούς, αυτό του δρομολογητή.

Υλοποιεί ακόμη συναρτήσεις που διαχειρίζονται μεταβλητές για τη σχεδίαση των διαγραμμάτων κάθε αντικειμένου, *getDiagramSelected()*, *getDiagramXaxisName()*, *getDiagramYaxisName()*, *getDiagramName()*, *getDiagramString(int index)*, *setDiagramSelected(int selected)*, αλλά και την εμφάνιση των στατιστικών, *getResultLabel(int index)* και *getResultLabelLength()*. Παρέχει, επίσης, όλες τις συναρτήσεις που χρησιμοποιούνται από τις κλάσεις που κληρονομούν από αυτή για ευκολότερο προγραμματισμό. Τέτοιες συναρτήσεις είναι και αυτές που αρχικοποιούν τους άξονες για κάθε διάγραμμα ανάλογα με το αντικείμενο που έχει επιλεχθεί, *initiateAxisNumbers()*, *initiateAxisXNumbers()* και *initiateAxisYNumbers()* και η συνάρτηση που σχεδιάζει το κατάλληλο διάγραμμα, *paintDiagram(Graphics2D g2)*.

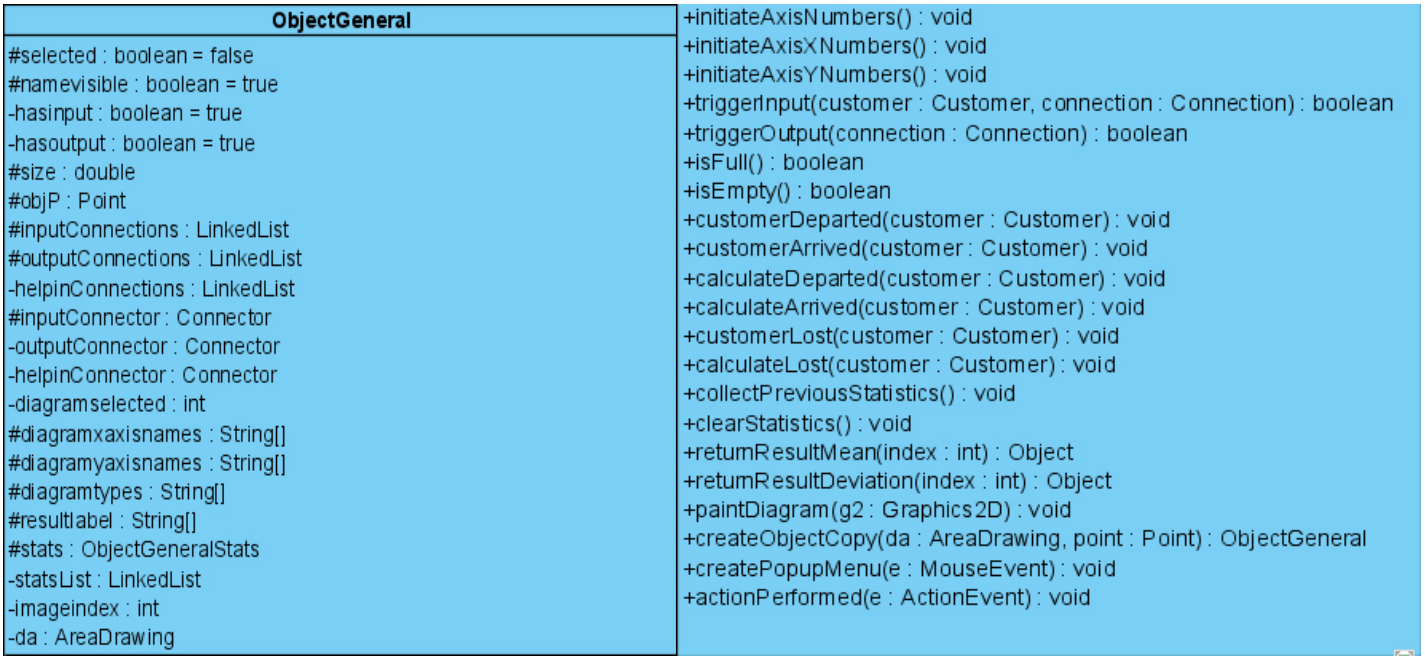
Υπάρχουν επίσης σημαντικές συναρτήσεις για την υλοποίηση της προσομοίωσης που πρέπει να υλοποιηθούν σε κάθε αντικείμενο και περιέχονται και στο γενικό αντικείμενο *ObjectGeneral*. Αυτές είναι οι *triggerInput(Customer customer,Connection connection)* και *triggerOutput(Connection connection)*, οι οποίες καθορίζουν τη συμπεριφορά του κάθε αντικειμένου, όταν δέχονται κάποιο ερέθισμα, είτε από αντικείμενο που είναι συνδεδεμένο με την είσοδο, είτε από αντικείμενο που είναι συνδεδεμένο με την έξοδο. Όταν μπορούν να δεχτούν το αντικείμενο ή αντίστοιχα να παραδώσουν κάποιο αντικείμενο, τότε οι συναρτήσεις αυτές επιστρέφουν true. Στην αντίστροφη περίπτωση επιστρέφουν false. Πολύ σημαντικό είναι το αποτέλεσμα των συναρτήσεων αυτών να είναι ανεξάρτητο από ποιο αντικείμενο τις κάλεσε, ώστε να μπορεί το πρόγραμμα να είναι εύκολα επεκτάσιμο. Να μη χρειάζονται αλλαγές σε όλα τα αντικείμενα κάθε φορά που ένα νέο αντικείμενο προστίθεται στο πρόγραμμα. Συναρτήσεις βοηθητικές στην υλοποίηση των παραπάνω είναι οι συναρτήσεις που επιστρέφουν αν ένα αντικείμενο είναι «γεμάτο» και δε μπορεί να δεχτεί άλλον πελάτη ή «άδειο» και δε μπορεί να στείλει κάποιον πελάτη σε πιθανό ερέθισμα. Οι συναρτήσεις αυτές που όλα τα αντικείμενα κληρονομούν από την *ObjectGeneral*, είναι οι *isFull()* και *isEmpty()*.

Οι παραπάνω συναρτήσεις, όταν γίνεται δεκτός ή παραδίδεται ένας πελάτης, θα πρέπει να καλέσουν και μία συνάρτηση, η οποία θα κάνει τους απαραίτητους υπολογισμούς, καθώς η άφιξη ή η αναχώρηση ενός πελάτη αλλάζει πιθανότατα και την κατάσταση του συστήματος. Οι συναρτήσεις αυτές που χωρίς κάποια υλοποίηση υπάρχουν και στη γενική κλάση αντικειμένου, είναι οι *customerDeparted(Customer customer)*, όταν πρόκειται για αναχώρηση πελάτη και η *customerArrived(Customer customer)*, όταν πρόκειται για άφιξη. Για καθαρότερο προγραμματισμό, οι ενέργειες που αφορούν στατιστικά και την άφιξη ή αναχώρηση ενός πελάτη έχουν διαχωριστεί από τις παραπάνω συναρτήσεις, αλλά καλούνται από αυτές και είναι οι *calculateDeparted(Customer customer){}* και *calculateArrived(Customer customer)*. Διαφορετική προσέγγιση πρέπει να ακολουθηθεί όταν ο πελάτης δεν αναχωρεί απλά από το αντικείμενο, αλλά χάνεται από το σύστημα. Η προσέγγιση αυτή υλοποιείται από τις συναρτήσεις *customerLost(Customer customer)* και *calculateLost(Customer customer)*. Στην πιο απλή περίπτωση, αυτή δηλαδή που υλοποιείται και στην *ObjectGeneral*, οι συναρτήσεις για άφιξη, αναχώρηση ή χάσιμο ενός πελάτη απλά καλούν αυτές του αντίστοιχου υπολογισμού.

Στόχο στην υλοποίηση κάθε αντικειμένου αποτελεί και η επεξεργασία των δεδομένων εξόδου κάθε προσομοίωσης. Κάθε κλάση αντικειμένου της μορφής *Object<όνομα αντικειμένου>* συνοδεύεται από μια βοηθητική κλάση της μορφής

Object<όνομα αντικειμένου>Stats. Η ιεραρχία των κλάσεων αυτών είναι πανομοιότυπη με την ιεραρχία που φαίνεται στο προηγούμενο σχήμα, των κλάσεων αντικειμένων. Έτσι, η βοηθητική κλάση της αρχικής κλάσης αντικειμένων, *ObjectGeneral*, είναι η *ObjectGeneralStats* η οποία περιέχει μεταβλητές και συναρτήσεις μόνο για τις αφίξεις και αναχωρήσεις δεδομένων που είναι κοινές για όλα τα αντικείμενα των οποίων οι βοηθητικές κλάσεις, τις κληρονομούν. Επιπλέον, λόγω της επιλογής διατήρησης στατιστικών από πολλές διαδοχικές εκτελέσεις προσομοίωσης πρέπει τα στατιστικά κάθε μιας από αυτές να διατηρούνται σε κάποια λίστα, την *statsList*, η διαχείριση της οποίας υλοποιείται από την συνάρτηση *collectPreviousStatistics()*. Λόγω της διαφορετικής φύσης των στατιστικών, κάθε αντικείμενο διαφοροποιεί την υλοποίηση της συνάρτησης και στην αρχική περιέχεται μόνο η διεπαφή της, Το ίδιο συμβαίνει και με την *clearStatistics()* που αρχικοποιεί τα στατιστικά πριν το ξεκίνημα κάποιας νέας προσομοίωσης. Για την απλοποίηση των πλαισίων που παρουσιάζουν τα στατιστικά κάθε αντικειμένου κάθε ένα από αυτά επιστρέφει τις τιμές που του αναλογούν με τη βοήθεια των συναρτήσεων, *returnResultMean(int index)* και *returnResultDeviation(int index)*, διεπαφές των οποίων βρίσκονται στην *ObjectGeneral* αλλά και της συνάρτησης *returnResult(int index)*, διεπαφή της οποίας υπάρχει στην *ObjectGeneralStats*.

Για την επίτευξη της αντιγραφής ή αποκοπής και επικόλλησης, κάθε αντικείμενο πρέπει να είναι σε θέση να αναπαράγει τον εαυτό του. Καλεί γι' αυτό τον λόγο την συνάρτηση *createObjectCopy(AreaDrawing da,Point point)* ή οποία όπως γίνεται αντιληπτό είναι διαφορετική για κάθε αντικείμενο. Μια διεπαφή της όμως για λόγους πολυμορφισμού βρίσκεται στη κλάση *ObjectGeneral*. Τέλος, μια ακόμα λειτουργία, κοινή για κάθε αντικείμενο είναι ένα βοηθητικό πλαίσιο επιλογών που παρέχεται και υλοποιείται με τις συναρτήσεις *createPopupMenu(MouseEvent e)* και *actionPerformed(ActionEvent e)*. Όλος ο κώδικας της μάλλον άκομψα μεγάλης αυτής συνάρτησης, αλλά σημαντική για την λειτουργία κάθε αντικειμένου που εξηγείται στη συνέχεια, όπως και ολόκληρος ο κώδικας της εφαρμογής παρέχεται στον συνοδευτικό οπτικό δίσκο. Στη συνέχεια όμως παρουσιάζεται το διάγραμμα της κλάσης [Dennis, Wixom and Tegarden, 2002] με την προσθήκη βέβαια και της συνάρτησης κατασκευαστή *ObjectGeneral(Point x,String type,double size,AreaDrawing da)*.



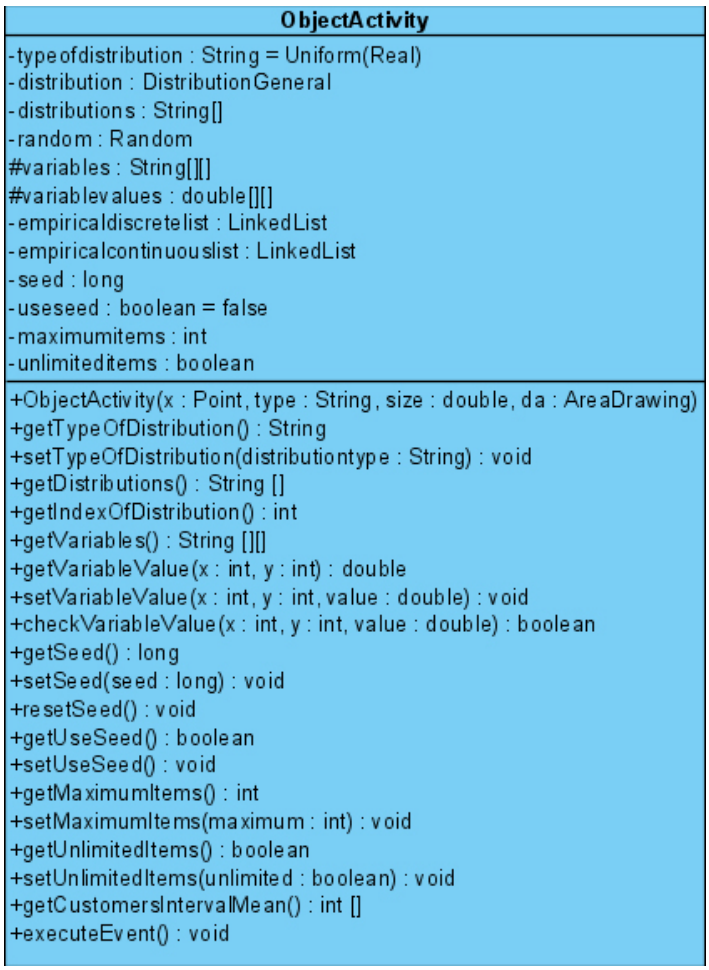
+ObjectGeneral(x : Point, type : String, size : double, da : AreaDrawing) +addToArea() : void +removeFromArea() : void +changeShapes() : void +moveObject(newpointx : int, newpointy : int) : void +createEmptyEllipses() : void +calculateConnectors() : void +mouseClicked(e : MouseEvent) : void +mousePressed(e : MouseEvent) : void +mouseReleased(e : MouseEvent) : void +mouseExited(e : MouseEvent) : void +mouseEntered(e : MouseEvent) : void +mouseDragged(e : MouseEvent) : void +mouseMoved(e : MouseEvent) : void +paintComponent(g : Graphics) : void +paintExterior(g2 : Graphics2D) : void +paintInterior(g2 : Graphics2D) : void +setNameVisible(visible : boolean) : void +getNameVisible() : boolean +isSelected() : boolean +setSelected(set : boolean) : void +setHasInput(input : boolean) : void +getHasInput() : boolean +setHasOutput(output : boolean) : void +getHasOutput() : boolean +getinputConnector() : Connector +getoutputConnector() : Connector +gethelpinConnector() : Connector	+addOutputConnection(connection : Connection) : void +addInputConnection(connection = Connection) : void +addHelpInConnection(connection : Connection) : void +removeOutputConnection(connection : Connection) : void +removeInputConnection(connection : Connection) : void +removeHelpInConnection(connection : Connection) : void +getType() : String +setType(type : String) : void +setObjectName(name : String) : void +getObjectName() : String +getObjectComments() : String +setObjectComments(comments : String) : void +setObjectSize(newsize : double, oldsize : double) : void +setObjectSize(size : double) : void +getObjectSize() : double +getPoint() : Point +getRectangle() : Rectangle +getImageIndex() : int +setImageIndex(i : int) : void +returnObjectIcon() : ImageIcon +getGrade(grade : double) : double +getDiagramSelected() : int +setDiagramSelected(selected : int) : void +getDiagramXaxisName() : String +getDiagramYaxisName()() : String +getDiagramName() : String +getDiagramString(index : int) : String +getResultLabel(index : int) : String +getResultLabelLength()() : int

Η κλάση *ObjectActivity* περιέχει όλες τις λειτουργίες που είναι κοινές στις γεννήτριες και τους εξυπηρετητές, λειτουργίες που έχουν να κάνουν με την παραγωγή τυχαίων αριθμών. Στην κλάση αυτή κρατούνται ο τύπος της κατανομής, *typeofdistribution* από μια γκάμα επιλογών στον πίνακα *distributions*, αλλά και η ίδια η κατανομή *DistributionGeneral*. Για να επιστραφεί η να τεθεί ο τύπος της χρησιμοποιούνται οι συναρτήσεις, *getTypeOfDistribution()* και *setTypeOfDistribution(String distributiontype)* και ολόκληρος ο πίνακας η *getDistributions()*. Μια ακόμα συνάρτηση επιστρέφει τον δείκτη της ισχύουσας επιλογής κατανομής από τον πίνακα *distributions*, η *getIndexOfDistribution()*. Η ομοιόμορφα κατανεμημένη μεταβλητή στο διάστημα (0,1) που παράγεται από τη java δίνεται από την κλάση Random της οποίας στιγμιότυπο, *random*, βρίσκεται σε κάθε αντικείμενο εξυπηρετητή ή γεννήτριας.

Κάθε μια από τις υποστηριζόμενες κατανομές απαιτεί κάποιες συγκεκριμένες μεταβλητές των οποίων το όνομα κρατείται σε έναν πίνακα *variables* για λόγους παρουσίασης και οι τιμές σε έναν άλλον πίνακα *variablevalues*. Οι πίνακες αυτοί είναι δυσδιάστατοι με κάθε γραμμή να καταλαμβάνεται από τις μεταβλητές κάποιας συγκεκριμένης κατανομής. Η κλάση αυτή κρατάει, επιπλέον, δυο λίστες για τις εμπειρικές κατανομές, τις *empiricaldiscretelist* και *empiricalcontinuouslist*, που δεν μπορούν λόγω της δυναμικής τους φύσης να διατηρηθούν σε μια στατική δομή δεδομένων όπως αυτή του πίνακα. Υπάρχει συνάρτηση που επιστρέφει όλο τον πίνακα των ονομάτων, *getVariables()*, συνάρτηση που επιστρέφει την τιμή της γραμμής *χ* και στήλης *y*, *getVariableValue(int x,int y)* αλλά και της θέτει την τιμή *value*, *setVariableValue(int x,int y,double value)*. Υπάρχει και μια βοηθητική συνάρτηση που ελέγχει την τιμή που θα τεθεί στην τιμή για εγκυρότητα και επιστρέφει αντίστοιχο μήνυμα, η *checkVariableValue(int x,int y,double value)*.

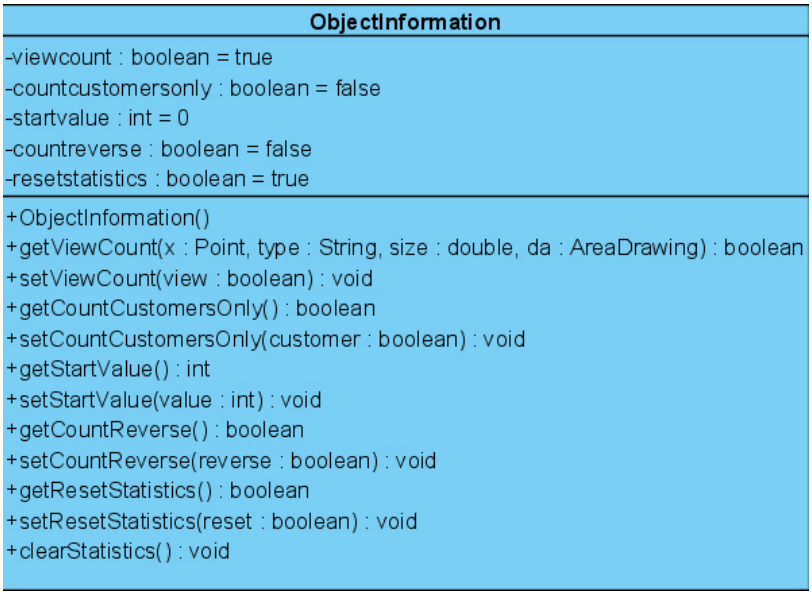
Επιπρόσθετα, παρέχονται συναρτήσεις για τον χειρισμό του σπόρου, *seed* που μπορεί να παράγονται είτε αυτόματα είτε να δίνονται από τον χρήστη, *useseed*. Ο χειρισμός των μεταβλητών αυτών γίνεται από τις *getSeed()*, *setSeed(long seed)*, *resetSeed()*, *getUseSeed()* και *setUseSeed(boolean useseed)*. Μια κοινή λειτουργία και στα δυο αντικείμενα που

κληρονομούν από την κλάση *ObjectActivity* είναι η επιλογή για παραγωγή ή εξυπηρέτηση άπειρων η συγκεκριμένου αριθμού πελατών σε κάθε προσομοίωση, η οποία υποστηρίζεται από τις μεταβλητές, *maximumitems* και *unlimiteditems* και από τις συναρτήσεις που τις χειρίζονται, *getMaximumItems()*, *setMaximumItems(int maximum)*, *getUnlimitedItems()* και *setUnlimitedItems(boolean unlimited)*. Υπάρχει τέλος μια βοηθητική συνάρτηση για την παραγωγή διαγραμμάτων μέσων όρων χρόνων για όλους τους πελάτες που έχουν περάσει από το συγκεκριμένο αντικείμενο, getCustomersIntervalMean(), και μια διεπαφή της συνάρτησης executeEvent() που υλοποιείται μόνο από τα δυο αυτά αντικείμενα που διαχειρίζονται γεγονότα. Εκτός από τον ολοκληρωμένο κώδικα της κλάσης που μπορεί να βρεθεί στον συνοδευτικό οπτικό δίσκο στη συνέχεια παρουσιάζεται ένα διάγραμμα της ολοκληρωμένης κλάσης [Dennis, Wixom and Tegarden, 2002].



Στην κλάση *ObjectInformation* περιέχονται όλες οι κοινές μεταβλητές των μετρητών και εξόδων. Οι μεταβλητές αυτές έχουν να κάνουν με στατιστικά στοιχεία της προσομοίωσης, καθώς τα αντικείμενα αυτά δε μπορούν να αλλάξουν κάτι στην προσομοίωση, παρά μόνον προσφέρουν στοιχεία γι' αυτήν. Περιέχονται επίσης συναρτήσεις για το χειρισμό των παραπάνω μεταβλητών. Τα μεγέθη που περιλαμβάνονται αναλυτικά στην κλάση αυτή είναι το αν οι πελάτες που πέρασαν από το αντικείμενο θα είναι ορατοί στο μοντέλο ή όχι, *viewcount* και *getViewCount()*, *setViewCount(boolean view)*, αν τα στατιστικά που θα κρατούνται θα αφορούν μόνο πελάτες ή κάθε μονάδα που θα περνά από το αντικείμενο, *countcustomersonly* και *getCountCustomersOnly()*, *setCountCustomersOnly(boolean customer)*. Επιπλέον, περιλαμβάνει μια μεταβλητή ορισμένη από τον χρήστη για την τιμή εκκίνησης του μετρήματος πελατών στην αρχή κάθε προσομοίωσης, *startvalue* και *getStartValue()*, *setStartValue(int value)* αλλά και για το αν θα μετρούνται αυτοί αντίστροφα ή όχι,

countreverse και *getCountReverse()*, *setCountReverse(boolean reverse)*. Τέλος μια μεταβλητή, *resetstatistics* και *getResetStatistics()*, *setResetStatistics(boolean reset)* επιτρέπει τον χρήστη να επιλέγει αν μετά από κάθε προσομοίωση τα στατιστικά θα αρχικοποιούνται ή όχι. Τα συγκεκριμένα κοινά στατιστικά που περιέχονται και στα δυο είδη αντικειμένων που κληρονομούν από την κλάση αυτή, αρχικοποιούνται με τη βοήθεια της συνάρτησης *clearStatistics()*.



2.4.1 Μετρητής (Count)

Είναι το πιο απλό αντικείμενο του Simmate, με μόνη δουλειά την καταμέτρηση των πελατών που περνούν από αυτόν. Μία επιπλέον ιδιότητα που ίσως φανεί χρήσιμη, είναι η δυνατότητα καταμέτρησης πελατών από πολλαπλές προσομοιώσεις και ο απαραίτητος μηδενισμός του μετρητή, όταν ο χρήστης το επιθυμεί. Ακόμη, υπάρχει η επιλογή αρχικοποίησης σε διαφορετική τιμή από τη μηδενική και η δυνατότητα αντίστροφης καταμέτρησης.

Όπως φαίνεται και στο σχήμα, το αντικείμενο αυτό κληρονομεί από την κλάση *ObjectInformation*, η οποία με τη σειρά της κληρονομεί από τη γενικότερη κλάση αντικειμένων *ObjectGeneral*.

2.4.2 Έξοδος (Exit)

Ένα ακόμα απλό αντικείμενο, απαραίτητο όμως για την παραγωγή χρήσιμων στατιστικών στοιχείων. Για την παραγωγή οποιουδήποτε στατιστικού αφορά στη φυσιολογική έξοδο ενός πελάτη, ο πελάτης πρέπει οπωσδήποτε να περάσει από μία τέτοια έξοδο. Υπάρχου δυο τρόποι για κάποιον πελάτη να εξέλθει του μοντέλου. Είτε να περάσει από ένα αντικείμενο εξόδου είτε να χαθεί “lost”. Στην περίπτωση που τίποτα από αυτά δεν έχει συμβεί, ο πελάτης βρίσκεται ακόμα στο σύστημα. Υπάρχουν και κάποιες επιπρόσθετες λειτουργίες όμοιες με αυτές του μετρητή που αναφέρθηκαν παραπάνω και προκύπτουν από την υπερκλάση της *ObjectInformation*.

Ομοίως με τους μετρητές, το αντικείμενο αυτό, λοιπόν, κληρονομεί από την κλάση *ObjectInformation*, η οποία με τη σειρά της κληρονομεί από την *ObjectGeneral*.

2.4.3 Γεννήτρια (Generator)

Είναι τα αντικείμενα από τα οποία ξεκινά κάθε προσομοίωση. Εάν δεν υπάρχουν στο μοντέλο, δεν πρόκειται να γεννηθεί κανένας πελάτης. Οι γεννήτριες απαιτούν τη χρήση κατανομών για να προσομοιώσουν την άφιξη πελατών, λειτουργίες, οι οποίες κληρονομούνται από την κλάση *ObjectActivity*.

Το αντικείμενο αυτό, όπως και ο εξυπηρετητής, μπορεί να προκαλέσει ένα γεγονός, την άφιξη πελάτη στο σύστημα. Η υλοποίηση της συμπεριφοράς του μοντέλου, όταν γίνεται αυτό, υλοποιείται στη συνάρτηση *executeEvent()*. Η συνάρτηση αυτή αρχικά καλεί την *getCustomer()*, η οποία και είναι υπεύθυνη να δημιουργήσει ένα νέο στιγμιότυπο πελάτη, τον οποίο και προσθέτει στη λίστα με τους πελάτες του μοντέλου, *da.customerList.add(customer)*. Στη συνέχεια στέλνεται το ερέθισμα για το νέο πελάτη στο συνδεδεμένο με την έξοδο αντικείμενο. Εάν δεν υπάρχει τέτοιο ή δεν μπορεί να δεχτεί πελάτη, τότε χάνεται. Διαφορετικά, ο πελάτης μπαίνει στη λίστα με τους εν κινήσει πελάτες *da.simulationrun.customersanimatedList.add(customer)*. Το αντικείμενο αυτό δεν έχει έξοδο και άρα δε χρειάζεται να υλοποιήσει τη συνάρτηση *triggerInput* και, εάν της σταλεί κάποιο ερέθισμα στην έξοδο, δε μπορεί να στείλει πελάτη και, έτσι, η *triggerOutput* επιστρέφει πάντα false, όποτε καλείται.

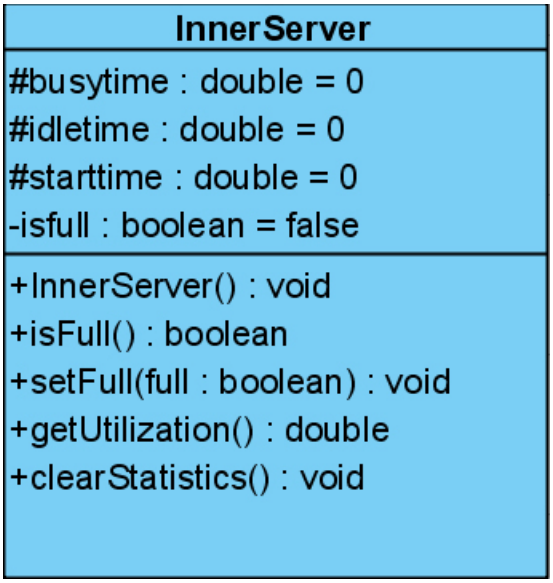
2.4.4 Εξυπηρετητής (Server)

Αντικείμενα, τα οποία σύμφωνα με κάποια κατανομή εξυπηρετούν τους πελάτες. Υποστηρίζουν ακριβώς τις ίδιες κατανομές με τις γεννήτριες, τις ιδιότητες των οποίων κληρονομούν από την κλάση *ObjectActivity*. Με τον τρόπο αυτό γίνεται η προσομοίωση του χρόνου εξυπηρέτησης κάθε πελάτη. Κάθε πελάτης που παραμένει για κάποιο χρονικό διάστημα στο αντικείμενο, θεωρείται ότι για αυτό το διάστημα εξυπηρετείται, εκτός και αν το αντικείμενο τεθεί σαν καθυστέρηση (delay). Τότε, η παραμονή του πελάτη στο αντικείμενο θεωρείται σαν αναμονή και προσμετράται ανάλογα στα στατιστικά.

Στο αντικείμενο αυτό, όπως και στην άφιξη πελάτη στη γεννήτρια, το να τελειώσει ένας πελάτης την εξυπηρέτηση θεωρείται γεγονός. Η όλη προσομοίωση, όπως αναφέρθηκε, λειτουργεί με το μηχανισμό επόμενου γεγονότος και ο τερματισμός εξυπηρέτησης είναι ένα από αυτά τα γεγονότα που υλοποιείται με τη συνάρτηση *executeEvent()*. Η συνάρτηση αυτή είναι υπεύθυνη για να εξάγει τον κατάλληλο πελάτη στο μοντέλο καλώντας τη συνάρτηση *getCustomer()* και να στείλει το ερέθισμα στο αντικείμενο εξόδου εφόσον υπάρχει. Εάν δεν υπάρχει ή είναι γεμάτο, ο πελάτης χάνεται. Εάν γίνει δεκτό από το αντικείμενο, τότε θα πρέπει να ξεκινήσει και η κίνησή του και γι' αυτό μπαίνει στη λίστα με τους πελάτες εν κινήσει *da.simulationrun.customersanimatedList.add(customer)*.

Μια λειτουργία του αντικειμένου αυτού είναι η δυνατότητα πολλαπλών ταυτόχρονων εξυπηρετήσεων. Ο αριθμός των ταυτόχρονων αυτών εξυπηρετήσεων μπορεί να τεθεί από τον χρήστη και αποθηκεύεται στη μεταβλητή *simultaneouslyservice* η οποία τίθεται και επιστρέφεται από τις συναρτήσεις *setSimultaneouslyService* και *getSimultaneouslyService*. Κάθε μια από αυτές θεωρείται για τα στατιστικά της προσομοίωσης σαν διαφορετικός εσωτερικός εξυπηρετητής για τον καθένα από τους οποίους θα πρέπει να κρατηθούν διαφορετικά στατιστικά. Αυτό γίνεται με την βοήθεια της εσωτερικής κλάσης *InnerServer* η οποία κρατάει διαφορετικά στατιστικά για την ώρα που είναι απασχολημένος *busytime*, αδρανής *idletime* αλλά και επιστρέφει σε ποια κατάσταση από τις δύο βρίσκεται, *isfull* με τις συναρτήσεις *isFull* και *setFull*. Μπορεί έτσι να επιστρέψει τον τρόπο χρήσης της *getUtilization*. Επιπρόσθετα όταν η προσομοίωση τελειώνει υλοποιείται και μια συνάρτηση η οποία επαναφέρει τα στατιστικά του *InnerServer* στην αρχική

τους τιμή. Στη συνέχεια φαίνεται το διάγραμμα κλάσης (class diagram) της κλάσης αυτής[Dennis, Wixom and Tegarden, 2002].



Το σύνολο των εσωτερικών εξυπηρετητών κρατείται στη λίστα *innerserverList* στην κλάση στατιστικών της *ObjectServer* την *ObjectServerStats*. Έτσι όταν για παράδειγμα σταματάει η προσομοίωση και πρέπει να ξεκινήσει η επόμενη θα πρέπει να αρχικοποιηθούνε όλοι οι εσωτερικοί εξυπηρετητές στη συνάρτηση *clearStatistics* της *ObjectServer*.

```
for (int i=0;i<((ObjectServerStats)stats).innerserverList.size();i++){
        innerserver = (InnerServer)((ObjectServerStats)stats).innerserverList.get(i);
        innerserver.clearStatistics();
}
```

Το γεγονός όμως ότι υπάρχουν πολλοί εσωτερικοί εξυπηρετητές σημαίνει ότι μέσα στον εξυπηρετητή μπορεί να υπάρχουν περισσότεροι από ένας πελάτης. Έτσι η συνάρτηση *getCustomer*, που επιστρέφει τον πελάτη στο αντικείμενο που είναι συνδεδεμένο με την έξοδο, γίνεται πιο πολύπλοκη. Για να κρατηθούν όλοι οι πελάτες, χρησιμοποιείται μια λίστα, η *customersservicedList* και όταν προστίθεται κάποιος πελάτης στο σύστημα μέσω της *triggerInput* που ελέγχει πρώτα αν είναι «γεμάτος» ο εξυπηρετητής καλείται η *addCustomer*. Η συνάρτηση αυτή τοποθετεί τον πελάτη στη λίστα, όμως λόγω της σημαντικότητας του χρόνου εξόδου από τον εξυπηρετητή όμως αποφασίστηκε η χρήση μιας ακόμη εσωτερικής κλάσης, της *CustomerServiced*, η οποία εκτός του πελάτη *customer* κρατά και τον χρόνο αυτό, *endtime*. Τελικά στη λίστα θα αποθηκεύεται όχι ο πελάτης αλλά αυτή η βοηθητική κλάση. Όταν τελικά τελειώσει η εξυπηρέτηση κάποιου πελάτη ο οποίος πρέπει να εξέλθει από το αντικείμενο αλλά και από τη λίστα *customersservicedList* πρέπει να αποφασιστεί ποιος πελάτης θα είναι αυτός, αφού μπορεί να υπάρχουν πολλοί εκείνη τη στιγμή που εξυπηρετούνται. Ένα κομμάτι κώδικα στην *getCustomer* ψάχνει τον πελάτη με τον πλησιέστερο χρόνο εξόδου στη λίστα

```
CustomerServiced customerserviced = (CustomerServiced)customersservicedList.get(0);
double mintime = customerserviced.endtime;
int minindex = 0;
for (int i=1;i<customersservicedList.size();i++){
        customerserviced = (CustomerServiced)customersservicedList.get(i);
```

```
        if (customerserviced.endtime<mintime){
                minindex = i;
                mintime = customerserviced.endtime;
        }
}
```

2.4.5 Ουρά (Queue)

Το αντικείμενο, στο οποίο κάποιος πελάτης μπορεί να περιμένει στο σύστημα. Ο χρόνος αναμονής θεωρείται σαν καθυστέρηση, δηλαδή, όπως αναφέρθηκε, μία ακαθόριστη χρονική διάρκεια που δεν είναι γνωστή από την αρχή. Εξαρτάται από αντικείμενα μετά την ουρά και πότε αυτά είναι έτοιμα να δεχτούν κάποιον πελάτη από την ουρά, αλλά και από αντικείμενα πριν την ουρά και με ποιον τρόπο στέλνουν πελάτες στην ουρά και με ποια πολιτική η ουρά τους προωθεί. Αν κάποια αναμονή είναι γνωστή από την αρχή, τότε θα πρέπει να χρησιμοποιηθεί ο εξυπηρετητής που θα έχει τεθεί όμως σαν καθυστέρηση.

Υποστηρίζονται τρεις πειθαρχίες ουρών από το αντικείμενο αυτό και διαφορετικές προσεγγίσεις στην υλοποίησή τους. Υπάρχει μία λίστα, η *customersqueuedList* η οποία περιέχει κάθε πελάτη που εισέρχεται στην ουρά. Όταν το αντικείμενο που έπεται της ουράς στείλει έναν πελάτη προς αυτήν, ελέγχεται για το αν είναι γεμάτη. Αν ναι, τότε απλά δεν δέχεται τον πελάτη, στέλνοντας αρνητική απάντηση πίσω στο αντικείμενο. Αν έχει χώρο, το δέχεται και το βάζει στην λίστα στο τέλος της. Όταν το αντικείμενο που ακολουθεί την ουρά κάνει αίτηση για πελάτη και η λίστα δεν είναι άδεια, υπάρχει δηλαδή διαθέσιμος πελάτης, τότε καλείται η συνάρτηση *getCustomer()*, η οποία, ανάλογα με τη πειθαρχία της ουράς, επιλέγει ποιος πελάτης θα εξέλθει πρώτος. Αν η πειθαρχία της ουράς είναι FIFO (First in First out) τότε επιλέγεται το πρώτο στοιχείο της λίστας, δηλαδή ο πρώτος πελάτης που εισήλθε σε αυτήν. Αν η πειθαρχία είναι LIFO (Last in First out) τότε επιλέγεται το τελευταίο στοιχείο της λίστας, δηλαδή ο τελευταίος πελάτης που εισήλθε σε αυτήν. Τέλος, αν η πειθαρχία είναι SIRO (Service in Random order), τότε με τη χρήση της τυχαίας συνάρτησης *Math.random()* της java επιλέγεται στην τύχη κάποιος πελάτης από τη λίστα.

```
public Customer getCustomer(){
        Customer customer;
        if (typeofqueue.equals("FIFO"))
                customer = (Customer)((ObjectQueueStats)stats).customersqueuedList.getFirst();
        else if (typeofqueue.equals("LIFO"))
                customer = (Customer)((ObjectQueueStats)stats).customersqueuedList.getLast();
        else{
                int randomchoice =
                        Math.round((float)(Math.random()*((((ObjectQueueStats)stats).customersqueuedList.size()-1))));
                customer = (Customer)((ObjectQueueStats)stats).customersqueuedList.get(randomchoice);
        }
        return customer;
```

}

2.4.6 Δρομολογητής (Routing)

Εκτελεί κάθε πιθανή δρομολόγηση στο πρόγραμμα. Μπορεί να δεχτεί πολλαπλές εισόδους και εξόδους και να υλοποιήσει διαφορετικούς αλγορίθμους δρομολόγησης, για να κατευθύνει τα αντικείμενα από αντίστοιχη είσοδο ή σε αντίστοιχη έξοδο.

Η φύση της λειτουργίας του αντικειμένου αυτού καθιστά αναγκαίο να δέχεται πολλά αντικείμενα συνδεδεμένα στην είσοδο και έξοδο αντίθετα με όλα τα αντικείμενα που προαναφέρθηκαν, που μπορούν να δεχτούν το πολύ ένα αντικείμενο. Το γεγονός ότι κάθε νέο συνδεδεμένο αντικείμενο επηρεάζει τα menu επιλογών του αντικειμένου, αλλά και τις διάφορες λίστες που θα συζητηθούν για την υλοποίηση των αλγορίθμων δρομολόγησης, καθιστά την υπερφόρτωση των συναρτήσεων *addOutputConnection*, *addInputConnection* και *removeOutputConnection* και *removeInputConnection* της *ObjectGeneral* αναγκαία. Οι συναρτήσεις αυτές σε όλα τα υπόλοιπα αντικείμενα απλά προσέθεταν ή αφαιρούσαν έναν σύνδεσμο και η χρήση τους ήταν απλή και όμοια, όπως στο παράδειγμα της *addOutputConnection* στη συνέχεια, και μπορούσε να κληρονομηθεί από την υπερκλάση τους.

```
public void addInputConnection(Connection connection){
    this.inputConnections.add(connection);
}
```

Στο συγκεκριμένο αντικείμενο θα πρέπει, με κάθε αλλαγή στους συνδέσμους τους, να ενημερώνονται τα πεδία στατιστικών *da.propertieswindow.statisticspanel*, *όπου θα προστεθεί και το νέο πεδίο στη λίστα resultlabellist.*, το αντίστοιχο menu, είτε το μεσαίο, είτε το δεξιό *da.propertieswindow.propertiespanel.panelright* που περιέχει τις επιλογές για τον αλγόριθμο εισόδου και εξόδου αντίστοιχα. Επίσης, θα ενημερωθεί και η λίστα *outputroundrobinlist* για τον αλγόριθμο round robin και η *outputweightslist* για τον weighted αλγόριθμο. Έτσι, για το παράδειγμα της *addOutputConnection* ο αλγόριθμος υλοποιήθηκε, όπως φαίνεται παρακάτω, με αντίστοιχους κώδικες για τις υπόλοιπες τρεις συναρτήσεις.

```
public void addOutputConnection(Connection connection){
    this.outputConnections.add(connection);
    int index = this.outputConnections.indexOf(connection);
    outputroundrobinlist.add(connection);
    ((ObjectRoutingStats)stats).outputitemslist.add(new Integer(0));
    outputweightslist.add(new Double(0));
    this.resultlabellist.add(index+this.inputConnections.size(),new
                String(connection.getInputObject().getObjectName()));
    if (da.propertieswindow.propertiespanel.panelright.output!=null){
        da.propertieswindow.propertiespanel.panelright.output.updateEverything();
        da.propertieswindow.propertiespanel.panelright.output.contentpanel.validate();
        da.propertieswindow.statisticspanel.createResultsMenu(this);
```

```
            da.propertieswindow.statisticspanel.validate();
        }
}
```

Η λειτουργία όμως που εκτελεί το αντικείμενο αυτό είναι οι αλγόριθμοι δρομολόγησης. Για να υλοποιηθεί η προσομοίωση θα πρέπει να υπερφορτωθούν οι *triggerInput(Customer customer,Connection connection)* και *triggerOutput(Connection connection)*, ώστε, ανάλογα με τη συνάρτηση δρομολόγησης, να προκαλεί το κατάλληλο συνδεδεμένο αντικείμενο.

```
public boolean triggerInput(Customer customer,Connection connection){
        if (this.outputConnections.size()!=0){
            if (getOutputRouting()==0){
                return triggerInputBestAlternative(customer,connection);
            }
            if (getOutputRouting()==1){
                return triggerInputRoundRobin(customer,connection);
            }
            if (getOutputRouting()==2){
                return triggerInputRandom(customer,connection);
            }
            else /*if (getOutputRouting()==3)*/{
                return triggerInputWeighted(customer,connection);
            }
        }
        return false;
}
```

Έτσι, ανάλογα με τον επιλεγμένο αλγόριθμο, καλείται διαφορετική συνάρτηση.

Η *triggerInputRoundRobin(Customer customer,Connection connection)* (ή στη περίπτωση εισόδου η *triggerOutputRoundRobin(Connection connection))* υλοποιεί τον αλγόριθμο εκ περιτροπής (*round robin)*. Ο αλγόριθμος αυτός κατανέμει τους πελάτες ομοιόμορφα από όλες τις εισόδους ή όλες τις εξόδους, με μία συγκεκριμένη σειρά, η οποία και έχει δοθεί από το χρήστη.

Η *triggerInputRandom(Customer customer,Connection connection)* (ή στη περίπτωση εισόδου η *triggerOutputRandom(Connection connection)*) υλοποιούν τον τυχαίο (*random)* αλγόριθμο. Σύμφωνα με αυτόν, η επόμενη είσοδος ή έξοδος, στην οποία θα δρομολογηθεί ο πελάτης, επιλέγεται στην τύχη και με τη χρήση της ομοιόμορφης κατανομής *nextDouble()* της κλάσης *Random* της java.

Η *triggerInputWeighted(Customer customer,Connection connection)* (ή στη περίπτωση εισόδου η *triggerOutputWeighted (Connection connection))* υλοποιεί τον *weighted* αλγόριθμο. Είναι μία διαφοροποίηση του random αλγόριθμου, στον οποίο επιλέγεται τυχαία η είσοδος ή η έξοδος του πελάτη, αλλά με διαφορετική πιθανότητα ανάλογα με το βάρος που θα θέσει ο χρήστης σε κάθε είσοδο ή έξοδο.

Η *triggerInputBestAlternative(Customer customer,Connection connection)* υλοποιεί τον *best alternative* αλγόριθμο, ο οποίος υλοποιείται μόνο για τις εξόδους, μελετά τις πιθανές διαδρομές του πελάτη και, σύμφωνα με ένα σύστημα βαθμολόγησης, τους κατανέμει στην καλύτερα βαθμολογημένη έξοδο. Κάθε αντικείμενο, ανάλογα με την κατάστασή του, προσθέτει σε έναν βαθμό, με τον οποίο καλέστηκε, τον δικό του βαθμό και καλεί το επόμενο αντικείμενο, μέχρι να φτάσει στο αντικείμενο εξόδου ή σε πιθανή απώλεια του πελάτη. Η απώλεια πελάτη δίνει το μέγιστο βαθμό και το πέρασμα του πελάτη από ένα αντικείμενο χωρίς καθυστέρηση δίνει μηδενικό βαθμό. Η διαδρομή που θα επιλεχθεί είναι αυτή με τον ελάχιστο βαθμό.

2.4.7 Παρτίδα (Batch)

Επιτρέπει σε αντικείμενα από πολλές πηγές να ενώνονται σε ένα αντικείμενο. Όπως και οι δρομολογητές επιτρέπουν παραπάνω από μία είσοδο αλλά, αντίθετα με αυτούς, μόνο μία έξοδο. Συνακόλουθα και για το αντικείμενο αυτό οι συναρτήσεις *addInputConnection(Connection connection)* και *removeInputConnection(Connection connection)* που αφορούν την προσθήκη ή αφαίρεση κάποιου συνδέσμου εισόδου, απαιτούν μια διαφορετική προσέγγιση και υπερφορτώνονται. Εφόσον ο χρήστης μπορεί να προσθέσει όσες εισόδους επιθυμεί πρέπει με κάποια δυναμική μέθοδο να διατηρηθεί η απαίτηση από κάθε σύνδεσμο και γι' αυτό το λόγο χρησιμοποιείται μια λίστα, *inputdemandlist*. Το αν το αντικείμενο μπορεί να δεχτεί πελάτη εξαρτάται από ποια είσοδο εισέρχεται αυτός και αν από την είσοδο αυτό έχει ήδη καλυφθεί η ζήτηση. Υπάρχει γι' αυτό το λόγο η συνάρτηση *isFull(int index)* της οποίας η μεταβλητή index καθορίζει τον σύνδεσμο για τον οποίο γίνεται το αίτημα αλλά και η *isFull()* που αναζητά έστω έναν σύνδεσμο για τον οποίο δεν έχει καλυφθεί. Όταν εισέλθει κάποιος πελάτης καλείται η *isFull()* και αν αυτή επιστρέψει true, αν δηλαδή το αντικείμενο έχει καλύψει τη ζήτηση από κάθε σύνδεσμο, τότε μόνο θα παράγει κάποιο πελάτη και θα κάνει trigger το αντικείμενο που είναι συνδεδεμένο με την έξοδό του.

2.4.8 Πηγή (Resource)

Μαζί με την γεννήτρια είναι τα μόνα αντικείμενα που μπορούν να προωθήσουν πελάτες στο μοντέλο. Η διαφορά της από την γεννήτρια όμως είναι ότι δεν παρέχει κάποια στοχαστική διαδικασία γι' αυτό. Ο μόνος τρόπος για να τους παράγει είναι να δεχτεί ένα ερέθισμα (trigger), από το αντικείμενο που είναι συνδεδεμένο με αυτό. Παρέχει συγκεκριμένο αριθμό πελατών που μπορεί όμως να γίνει και άπειρος αν το επιθυμεί ο χρήστης. Οι πελάτες μπορούν όπως και στη γεννήτρια να θεωρηθούν είτε μονάδες που δεν λαμβάνονται υπόψη στον υπολογισμό στατιστικών είτε πελάτες που θα πρέπει να υπολογιστούν. Κληρονομεί από την γενική κλάση αντικειμένων *ObjectGeneral* όπως κάθε αντικείμενο. Συχνότερα χρησιμοποιείται σε συνδυασμό με το αντικείμενο που δημιουργεί ερέθισμα, το αντικείμενο σκανδάλης που εξηγείτε στη συνέχεια.

2.4.9 Σκανδάλη (Trigger)

Το αντικείμενο αυτό, είναι το μοναδικό που διαθέτει και ένα πρόσθετο σύνδεσμο, τον βοηθητικό σύνδεσμο. Όταν μια μονάδα εισέλθει στην σκανδάλη από την είσοδο αυτή, το αντικείμενο που είναι συνδεδεμένο με την κανονική είσοδο της δέχεται ένα ερέθισμα. Δεν κρατά κανένα στατιστικό στοιχείο για την προσομοίωση και η μόνη συνάρτηση που

παρουσιάζει ενδιαφέρον είναι η *triggerInput(Customer customer,Connection connection)* που κληρονομείται από την *ObjectGeneral* και στην οποία πρέπει να γίνει έλεγχος για τον σύνδεσμο από τον οποίο κάνει ερώτηση για να εισέλθει ο *customer*. Στην περίπτωση που ο σύνδεσμος είναι ο βοηθητικός, *if (connection.getType()==1),* απευθείας κι αφού εκτελεστεί ο αλγόριθμος για την κίνηση του πελάτη δέχεται το ερέθισμα το αντικείμενο στην κανονική είσοδο. Στην περίπτωση που ο σύνδεσμος είναι ο κανονικός, δεν θα επιτραπεί το πέρασμα από την σκανδάλη. Λειτουργεί, λοιπόν, σαν μια πόρτα που ανοίγει μόνον αν εισέλθει πελάτης από τον βοηθητικό σύνδεσμο. Την στιγμή αυτήν, και μόνο εάν υπάρχει διαθέσιμος πελάτης στο αντικείμενο που είναι συνδεδεμένο με την είσοδο της σκανδάλης θα περάσει από αυτήν ο πελάτης. Λόγω της φύσης της λειτουργίας του, χρησιμοποιείται συχνά σε συνδυασμό με το αντικείμενο της πηγής.

2.4.10 Προσθήκη νέων αντικειμένων

2.4.10.1 Δημιουργία αντικειμένου χωρίς ιδιότητες

Απαραίτητη κρίνεται η παράθεση στο σημείο αυτό ενός υποκεφαλαίου που να εξηγεί με ποιο τρόπο γίνεται η προσθήκη νέων αντικειμένων στα ήδη υπάρχοντα στο πρόγραμμα. Για να γίνει αυτό, θα χρησιμοποιηθεί σαν παράδειγμα η προσθήκη του αντικειμένου ObjectCount με ίσως λιγότερες λειτουργίες από ότι το πραγματικό αντικείμενο του προγράμματος. Η προσθήκη επιπλέον ιδιοτήτων δεν είναι δύσκολη και γίνεται με τον ίδιο τρόπο που θα προστεθούν στη συνέχεια οι ιδιότητες του παραδείγματος. Θα εξηγηθεί στη συνέχεια όλη η διαδικασία δημιουργίας του. Η λειτουργία του αντικειμένου αυτού είναι να κρατά και να παρέχει αντικείμενα στο μοντέλο

Όπως θα αναφερθεί και στη συνέχεια, για την εκτέλεση οποιασδήποτε εντολής, για την οποία θα πρέπει να υπάρχει λειτουργία undo / redo, πρέπει να δημιουργηθεί μία κλάση, η οποία θα κληρονομεί από την Command. Έτσι, για την δημιουργία αντικειμένων υπάρχει η κλάση CommandCreateObject. Στην κλάση αυτή και στη συνάρτησή της, execute(), υπάρχει μία δομή επιλογής, στην οποία περιέχονται όλες οι πιθανές εντολές για τη δημιουργία αντικειμένων. Στην αλληλουχία αυτή των επιλογών, θα πρέπει απλά να προστεθεί και η δικιά μας:

```
else if (selected.equals("Count")){
        count = new ObjectCount(pclicked,da.objectSelected,da.simulation.objectsize,da);
        count.addToArea();
        simobject = count;
}
```

Για να υπάρχει κάποια ανάδραση στον χρήστη, ανάλογα με την κίνηση του κέρσορα και τις εντολές που δίνει μέσω των menu του προγράμματος, αλλάζει και η μορφή του δείκτη. Κάθε αντικείμενο έχει σαν ένδειξη μία μικρή επισήμανση κάτω από το δείκτη όπως για παράδειγμα ή για αντικείμενο εξόδου ή μετρητή, αντίστοιχα. Θα πρέπει, λοιπόν, να δημιουργηθεί ένας κέρσορας με την ένδειξη του νέου αντικειμένου, στην περίπτωσή μας ο , και να μπει στον φάκελο images/Cursors μαζί με όλους τους υπόλοιπους. Στην κλάση Simulation και στη συνάρτηση createAllImages(), στην οποία δημιουργούνται όλες οι εικόνες που χρησιμοποιούνται, θα πρέπει το path του νέου κέρσορα να προστεθεί στον αντίστοιχο πίνακα cursorimagepath. Στην DragAndDropAnimation, που είναι υπεύθυνη για την εναλλαγή του δείκτη, και στη συνάρτηση unsetCursorType(), της οποίας η λειτουργία είναι να επαναθέτει το σωστό δείκτη, όταν αυτός έχει αλλάξει,

λόγω του ότι τον τοποθετήσαμε πάνω από ένα υπάρχον αντικείμενο, κάτι που μετατρέπει τον δείκτη σε παλάμη, ή σε έναν συνδετήρα, κάτι που τον μετατρέπει σε στυλό, θα πρέπει να προσθέσουμε άλλη μία επιλογή:

```
else if(da.objectSelected.equals("Count")){
        da.setCursor(Images.cursors[14]);
}
```

Στην κλάση MenuBar δημιουργούνται τα menu και βρίσκονται όλες οι λειτουργίες, ανάμεσά τους και αυτές που αφορούν στη δημιουργία των αντικειμένων. Οι συγκεκριμένες λειτουργίες βρίσκονται και στη MenuLibrary. Όσον αφορά στους δείκτες, και στις δύο κλάσεις θα πρέπει να προστεθεί στην συνάρτηση actionPerformed() άλλη μία επιλογή if:

```
else if(action.equals("Count")){
        simulation.da.setCursor(Images.cursors[14]);
 }
```

Στην κλάση MenuLibrary υπάρχει ένας πίνακας, όπου υπάρχουν τα Strings για κάθε αντικείμενο, και στον οποίο θα πρέπει απλά να προσθέσουμε ένα ακόμα με το όνομα του νέου αντικειμένου στη σωστή σειρά. Θα πρέπει στο σημείο αυτό να δημιουργηθεί το icon που θα το αντιπροσωπεύει, και στα menu, αλλά και στο μοντέλο. Στη συνάρτηση createAllImages() της Simulation κλάσης και στον πίνακα librarytoolpath θα πρέπει να μπει το path για το icon του νέου αντικείμενου.

Στην κλάση MenuBar και στη συνάρτηση createLibraryMenu() θα πρέπει να τεθεί το νέο menuitem για το αντικείμενο. Επίσης, στη συνάρτηση setAllMenus() να τεθούν για το νέο menuitem τα enabled και disabled, όπως και με τα menuitems των υπόλοιπων αντικειμένων. Θα πρέπει, ακόμη, να δηλωθεί στη κλάση Buttons το νέο menuitem.

Κάθε αντικείμενο έχει και ένα συγκεκριμένο menu ανάλογα με τις λειτουργίες του. Το menu αυτό δημιουργείται από τις PanelPropertiesLeft , PanelPropertiesMiddle και PanelPropertiesRight οι οποίες περιέχονται στην κλάση PanelProperties. Η κλάση PanelPropertiesLeft περιέχει menu που είναι κοινά για όλα τα αντικείμενα και δημιουργούνται αυτόματα. Στις άλλες δυο κλάσεις και στις συναρτήσεις τους createPropertiesMenu(ObjectGeneral simobject) θα πρέπει να προσθέσουμε άλλη μία επιλογή για το νέο μας αντικείμενο:

```
else if (simobject.getType().equals("Count")){
        ObjectCount simobj = (ObjectCount)simobject;
        createCountMenu(simobj);
}
```

Η συνάρτηση που καλείται στον παραπάνω κώδικα, φυσικά θα πρέπει να δημιουργηθεί, αρχικά χωρίς καμία εντολή, μέχρι να αρχίσουμε να προσθέτουμε λειτουργίες στο αντικείμενο.

Το επόμενο βήμα είναι να δημιουργηθεί το αντικείμενο, το οποίο θα κληρονομεί από το αντικείμενο ObjectGeneral και η δημιουργία της συνάρτησης κατασκευαστή. Στο σημείο αυτό, θα πρέπει να δημιουργηθεί και η κλάση, η οποία θα κρατά όλα τα στατιστικά του αντικειμένου, για να μπορεί ο χρήστης να ανατρέξει σε αυτά οποιαδήποτε στιγμή. Η κλάση αυτή θα πρέπει να κληρονομεί από την ObjectGeneralStats και αρχικά, πριν καταχωρηθεί οποιοδήποτε στατιστικό, θα έχει την παρακάτω μορφή:

```
import java.util.LinkedList;
/** Contains every statistic for the resource object */
```

```
public class ObjectCountStats extends ObjectGeneralStats{
        ObjectCount count;

        LinkedList infoList;
        public ObjectResourceStats(ObjectCount count){
                this.count = count;
                infoList = new LinkedList();
        }
}
```

Όσον αφορά στην κλάση του αντικειμένου υπάρχουν κάποιες συναρτήσεις που είναι απαραίτητες. Για να λειτουργήσει ο μηχανισμός copy – paste, απαραίτητη είναι η συνάρτηση createObjectCopy, στην οποία θα αντιγράφουμε κάθε χρήσιμο χαρακτηριστικό του νέου αντικειμένου. Για να ζωγραφιστεί το αντικείμενο, πρέπει να υλοποιήσουμε την paintInterior και, αν το νέο αντικείμενο έχει τη μορφή κάποιου icon, το path του θα πρέπει να μπει στην κλάση Simulation στη συνάρτηση createAllImages() και στον πίνακα objectspath, αφού φυσικά θα έχει δημιουργηθεί και μπει στον φάκελο objects το αντίστοιχο icon. Άλλη μία απαραίτητη συνάρτηση είναι η returnObjectIcon, η οποία χρησιμοποιείται για να επιστρέφει το icon που το αναπαριστά, κάτι που χρησιμοποιείται στα menu της PanelPropertiesLeft. Στον κατασκευαστή θέτουμε true ή false στις μεταβλητές hasinput και hasoutput ανάλογα με το αν το αντικείμενο έχει είσοδο και έξοδο. Αμέσως μετά, δημιουργούμε τις εισόδους και εξόδους, καλώντας την createEmptyEllipses() της κλάσης ObjectGeneral, και θέτουμε το μέγεθος του αντικειμένου setObjectSize(size).

Το επόμενο βήμα είναι να θέσουμε τους πίνακες με τα αποτελέσματα και τα διαγράμματα, τα οποία ας υποθέσουμε αρχικά σαν μηδενικά. Η κλάση μας στο σημείο αυτό θα έχει την παρακάτω μορφή:

```
public class ObjectCount extends ObjectGeneral{
        public ObjectCount (Point x,String type,double size,AreaDrawing da){
                super(x,type,size,da);
                setToolTipText("<HTML><BODY><b>Count: </b>Count the number <br>"+
                                                        "of customers through this object</BODY></HTML>");
                hasinput = true;
                hasoutput = true;
                stats = new ObjectCountStats(this);
                createEmptyEllipses();
                setObjectSize(size);
                resultlabel = new String[]{""};
                diagramtypes = new String[]{"sorry no diagrams available"};
                diagramxaxisnames = new String[]{""};
                diagramyaxisnames = new String[]{""};
                setDiagramSelected(0);
        }
```

```
public ObjectGeneral createObjectCopy(AreaDrawing drawingarea){
        ObjectCount count;
        Point point= new Point();
        point.x = drawingarea.pclicked.x - (drawingarea.simulation.copysa.getRectangle().x - objP.x);
        point.y = drawingarea.pclicked.y - (drawingarea.simulation.copysa.getRectangle().y - objP.y);
        count = new ObjectCount(point,this.getType(),size,drawingarea);
        count.setImageIndex(this.getImageIndex());
        return count;
}
public void paintInterior(Graphics2D g2){
        g2.drawImage((Images.objectsicon[0]).getImage(),
                        (int)(objP.x-((double)da.zoomfactor/100)*size/2),
                        (int)(objP.y-((double)da.zoomfactor/100)*size/2),
                        (int)(((double)da.zoomfactor/100)*size),
                        (int)(((double)da.zoomfactor/100)*size), null);
 }
public ImageIcon returnObjectIcon(){
        return Images.objectsicon[0];
}
```

Για να λειτουργήσει το σύστημα αρχείων του προγράμματος, θα πρέπει να προστεθεί και η δημιουργία του νέου αντικειμένου από το αρχείο xml. Η προσθήκη που χρειάζεται στο σημείο αυτό είναι στην κλάση FileReader, η οποία διαβάζει το αρχείο και παράγει το μοντέλο, μίας επιλογής ακόμα για το νέο τύπο αντικειμένου στη συνάρτηση readFile():

```
else if (objecttype.getText().equals("Count")){
        ObjectCount count = new ObjectCount( new
        Point((int)Double.parseDouble(objectpointx.getText()),(int)Double.parseDouble(objectpointy.getText())),
        objecttype.getText(),Double.parseDouble(objectsize.getText()),this.da);
        count.addToArea();
        simobject = count;
}
```

Το αντικείμενο μας είναι έτοιμο χωρίς όμως καμία ιδιότητα.

2.4.10.2 Προσθήκη ιδιοτήτων στο αντικείμενο.

Η επόμενη κίνηση είναι να προστεθούν οι ιδιότητες του αντικειμένου αυτού και τα menu που θα τις ορίζουν. Οι μεταβλητές και οι συναρτήσεις που θα τις θέτουν αλλά και θα τις επιστρέφουν, θα μπουν στην κλάση του αντικειμένου και τα πεδία, από τα οποία θα μπορεί ο χρήστης να τις αλλάζει στις κλάσεις PanelPropertiesMiddle και PanelPropertiesRight. Η κλάση PanelPropertiesLeft έχει δημιουργήσει, όπως αναφέρθηκε, τα menu αυτόματα.

Η ιδιότητα που θα προστεθεί στο σημείο αυτό, είναι η επιλογή της εμφάνισης ή όχι στο αντικείμενο του αριθμού των πελατών. Την ιδιότητα αυτή την ορίζουμε σαν μεταβλητή τύπου Boolean στη βασική κλάση, ObjectCount, και δημιουργούμε δύο συναρτήσεις, οι οποίες θα θέτουν και θα επιστρέφουν τη μεταβλητή αυτή.

```
private boolean viewcount;
public boolean getViewCount(){
        return this.viewcount;
}
public void setViewCount(boolean view){
        this.viewcount = view;
}
```

Στον κατασκευαστή πρέπει απλά να θέσουμε την αρχική τιμή που θα έχει η μεταβλητή αυτή, για παράδειγμα:

```
viewcount = true;
```

2.4.10.3 Δημιουργία βασικού μενού ιδιοτήτων.

Θα πρέπει, όμως, μέσα από τα menu να δίνεται η δυνατότητα στο χρήστη να αλλάζει τη μεταβλητή αυτή, κάτι που θα γίνει μέσα από τη συνάρτηση createCountMenu(ObjectCount count) της PanelPropertiesMiddle:

```
        public void createCountMenu(ObjectCount count){
        border.setTitle("");
        this.setLayout(new BoxLayout(this, BoxLayout.PAGE_AXIS));
        viewcountbox = new JCheckBox("View Counts");
        viewcountbox.setOpaque(false);
        viewcountbox.setSelected(count.getViewCount());
        viewcountbox.addItemListener(new ViewCountAction(count));
        this.add(viewcountbox);
}
```

Πρέπει, ακόμα, να δημιουργηθεί η ενέργεια, η οποία είναι συνδεδεμένη με το πεδίο viewcountbox και θέτει τη νέα τιμή στη μεταβλητή:

```
public class ViewCountAction implements ItemListener{
        ObjectCount count;
        ViewCountAction(ObjectCount count){
        this.count = count;
        }
        public void itemStateChanged(ItemEvent e) {
                if (e.getStateChange() == ItemEvent.DESELECTED){
                        count.setViewCount(false);
                }
```

```
                if (e.getStateChange() == ItemEvent.SELECTED){
                        count.setViewCount(true);
                }
                da.repaint();
        }
}
```

2.4.10.4 Προσθήκη στατιστικών.

Για κάθε στατιστικό που πρέπει να διατηρείται κατά τη διάρκεια της προσομοίωσης, χρησιμοποιείται μία κλάση της μορφής Object-όνομα αντικειμένου-Stats, εδώ ObjectCountStats, η οποία στην αρχική της μορφή παρουσιάστηκε παραπάνω. Στην κλάση αυτή θα πρέπει να διατηρήσουμε την απαραίτητη μεταβλητή, που θα κρατά τον τρέχοντα αριθμό διερχόμενων πελατών, και να προσθέσουμε τις συναρτήσεις που θα τον χειρίζονται. Έτσι, θα έπρεπε π.χ.:

```
private int currenpassingtresources;
public int getCurrenpassingtPassingResources(){
        return this.currentpassingresources;
}
public void setCurrentPassingResources(int resources){
        this.currentpassingresources = resources;
}
```

Στη συγκεκριμένη, όμως, περίπτωση, και αφού η κλάση ObjectCountStats κληρονομεί από την ObjectGeneralStats, η οποία διαθέτει ήδη μηχανισμό εύρεσης των διερχομένων, δεν είναι απαραίτητη η προσθήκη των παραπάνω συναρτήσεων. Η μεταβλητή που κρατά το δεδομένο αυτό και οι συναρτήσεις είναι:

```
private int departures;
public int getDepartures(){
        return this.departures;
}
public void addDeparture(){
        this.departures++;
}
public void setDepartures(int departures){
        this.departures = departures;
}
```

2.4.10.5 Δημιουργία μενού στατιστικών.

Στο πλαίσιο των στατιστικών μπορούμε τώρα να προσθέσουμε την παραπάνω τιμή. Η κλάση που δημιουργεί όλα τα πεδία στατιστικών είναι η PanelStatistics. Αρχικά, η συνάρτηση placeResultMenus(ObjectGeneral simobject) τοποθετεί

τα κατάλληλα πεδία κάθε αντικειμένου, παίρνει από ένα πίνακα results του αντικειμένου, που έχει κληρονομηθεί από το γενικό αντικείμενο ObjectGeneral, τα ονόματα των πεδίων και τα τοποθετεί. Αυτό που πρέπει να γίνει, λοιπόν, είναι να μπει το πεδίο για την παραπάνω μεταβλητή στον κατασκευαστή της ObjectCount;

```
resultlabel = new String[]{"Count"};
```

Στη συνέχεια, η συνάρτηση updateResultsMenu(ObjectGeneral simobject) θέτει τις τιμές στα πεδία που έχουν δημιουργηθεί, καλώντας την returnResult(int index) της ObjectCountStats:

```
public Object returnResult(int index){
        if (index ==0){
                return new Double(getDepartures());
        }
        return new Object();
}
```

Στο πλαίσιο στατιστικών υπάρχει, επίσης, η δυνατότητα προβολής του μέσου όρου και της τυπικής απόκλισης κάθε παρουσιαζόμενου στατιστικού μετά από σειρά διαδοχικών προσομοιώσεων. Για να γίνει αυτό, θα πρέπει να προστεθούν στο αντικείμενο δύο συναρτήσεις που να επιστρέφουν την αντίστοιχη τιμή, οι returnResultMean(int index) και returnResultDeviation(int index).

```
public Object returnResultMean(int index){
        ObjectCountStats currentstats;
        double sum=0;
        for(int i=0;i<statsList.size();i++){
                currentstats = (ObjectCountStats)statsList.get(i);
                sum +=((Double)currentstats.returnResult(index)).doubleValue();
        }
        if (statsList.size()!=0)
                return new Double((double)(sum/statsList.size()));
        else
                return new Double(0);
}
public Object returnResultDeviation(int index){
        ObjectCountStats currentstats;
        double sum=0;
        double mean = ((Double)this.returnResultMean(index)).doubleValue();
        for(int i=0;i<statsList.size();i++){
                currentstats = (ObjectCountStats)statsList.get(i);
                sum +=Math.pow(((Double)currentstats.returnResult(index)).doubleValue()-mean,2);
        }
```

```
        if (statsList.size()!=0)
            return new Double((double)Math.sqrt((sum/statsList.size())));
        else
            return new Double(0);
    }
```

Για να υπολογιστεί, όμως, η τυπική απόκλιση θα πρέπει, μετά από κάθε προσομοίωση, να μετράται από την αρχή κάθε στατιστικό, αλλά και να κρατείται σε μία λίστα αντίστοιχου μεγέθους με τον αριθμό των συνεχών προσομοιώσεων που θέλουμε να υπολογίσουμε. Τη λειτουργία αυτή υλοποιούν οι συναρτήσεις clearStatistics() και collectPreviousStatistics() οι οποίες υλοποιούνται ως εξής:

```
    public void clearStatistics(){
        if (this.getResetStatistics()){
            ((ObjectCountStats)stats).setArrivals(0);
            ((ObjectCountStats)stats).setDepartures(0);
        }
    }
  public void collectPreviousStatistics(){
    int arrivals=0;
    int departures=0;
    if (!this.getResetStatistics()){
        arrivals = ((ObjectCountStats)stats).getArrivals();
        departures = ((ObjectCountStats)stats).getDepartures();
    }
    stats = new ObjectCountStats(this);
    ((ObjectCountStats)stats).setArrivals(arrivals);
    ((ObjectCountStats)stats).setDepartures(departures);

    statsList.add(stats);
    while(statsList.size()>da.simulationinfo.getRunsKept()){
        statsList.removeFirst();
    }
  }
```

2.4.10.6 Υλοποιήση λειτουργιών προσομοίωσης.

Μία απαραίτητη προσθήκη στο αντικείμενο είναι η υλοποίηση της συμπεριφοράς του κατά την προσομοίωση. Για την υλοποίηση αυτή θα πρέπει να υπερφορτωθούν οι συναρτήσεις, executeEvent(), triggerInput(Customer

customer,Connection connection), triggerOutput(Connection connection), customerArrived(Customer customer) και customerDeparted(Customer customer).

Η executeEvent() είναι απαραίτητη για αντικείμενα, τα οποία μπορούν να προκαλέσουν γεγονότα, όπως οι εξυπηρετητές και οι γεννήτριες, αλλά όχι ο μετρητής που εξετάζουμε. Η triggerInput(Customer customer,Connection connection) υλοποιεί τη συμπεριφορά του μοντέλου, όταν ένας πελάτης, ο customer, είναι έτοιμος να αφιχθεί στο αντικείμενο από τη σύνδεση connection. Το υπό μελέτη αντικείμενο θα πρέπει να τσεκάρει τη λίστα εξόδων του (στην ουσία όλα τα αντικείμενα έχουν μία έξοδο εκτός από τα αντικείμενα δρομολόγησης), και, αν βρει εκεί ένα αντικείμενο έτοιμο να δεχτεί τον πελάτη, θα επιστρέψει true, προσθέτοντας στη λίστα διαδρομής του πελάτη τη νέα είσοδο. Έτσι:

```
public boolean triggerInput(Customer customer,Connection connection){
        Connection currentconnection;
        boolean noneapproved = true;
        boolean approved;
        for(int index=0;index<this.outputConnections.size();index++){
                currentconnection = (Connection)this.outputConnections.get(index);
                approved = currentconnection.getInputObject().triggerInput(customer,currentconnection);
                if (approved){
                        noneapproved = false;
                        customer.addConnection(currentconnection);
                }
        }
        return !noneapproved;
}
```

Η triggerOutput(Connection connection) υλοποιεί αντίστοιχα τη συμπεριφορά, όταν το αντικείμενο που είναι συνδεδεμένο με την έξοδο έχει υποστεί μία αλλαγή και είναι έτοιμο να δεχτεί τον πιθανό πελάτη. Ο μετρητής, που δεν κάνει τίποτα άλλο από το να μετράει τους πελάτες που έχουν διέλθει από αυτόν, τσεκάρει το προηγούμενο αντικείμενο, αν μπορεί να δώσει πελάτη, καλώντας την triggerOutput για αυτό:

```
public boolean triggerOutput(Connection connection){
        Connection currentconnection;
        boolean noneapproved = true;
        boolean approved;
        for(int index=0;index<this.inputConnections.size();index++){
                currentconnection = (Connection)this.inputConnections.get(index);
                approved = currentconnection.getOutputObject().triggerOutput(currentconnection);
                if (approved){
                        noneapproved = false;
                }
        }
```

```
        return !noneapproved;
    }
```

Όταν ο πελάτης διέρχεται από κάθε αντικείμενο, στην κλάση Customer που διαχειρίζεται όλες τις λειτουργίες του πελάτη, καλούνται οι customerArrived(Customer customer) και customerDeparted(Customer customer), οι οποίες θα υλοποιηθούν στην κλάση ObjectCount με τη συμβολή των calculateArrived() και calculateDeparted(), στις οποίες γίνονται οι υπολογισμοί των νέων στατιστικών που προκύπτουν με την έλευση του πελάτη. Άλλη μία απαραίτητη λειτουργία είναι η αλλαγή κατά την αναχώρηση της νέας εικόνας που θα αναπαριστά τον πελάτη. Έτσι, οι παρακάτω συναρτήσεις θα προστεθούν στη κλάση:

```
public void calculateArrived(){
        ((ObjectCountStats)stats).addArrival();
}
public void customerArrived(Customer customer){
        calculateArrived();
}
public void calculateDeparted(){
        ((ObjectCountStats)stats).addDeparture();
}
public void customerDeparted(Customer customer){
        customer.setImageIndex(this.getImageIndex());
        calculateDeparted();
}
```

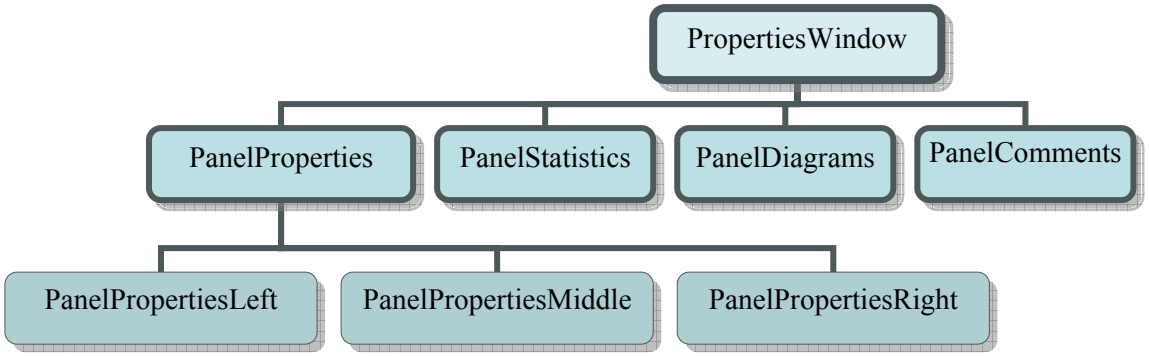
Τα μόνα στατιστικά που θα αλλάξουν είναι η προσθήκη μίας άφιξης και μίας αναχώρησης αντικειμένου.

2.5 Υλοποίηση λειτουργιών διεπαφής

2.5.1 Πλαίσια ιδιοτήτων

2.5.1.1 Γενικό πλαίσιο ιδιοτήτων

Για την δημιουργία του πλαισίου ιδιοτήτων που χρησιμοποιείται για κάθε αντικείμενο αλλά και για τις ιδιότητες της όλης προσομοίωσης χρησιμοποιεί την ιεραρχία που φαίνεται στη συνέχεια.



Η κλάση *PropertiesWindow*, κληρονομεί από την κλάση *JTabbedPane* της βιβλιοθήκης import javax.swing. Αποτελείται από τέσσερεις ταμπλέτες που υλοποιούνται από ισάριθμες κλάσεις. Μια καρτέλα με τις επιλογές του χρήστη για το αντικείμενο ή την προσομοίωση, *PanelProperties*, μια που παρουσιάζει τα αντίστοιχα στατιστικά, *PanelStatistics*, μια που παρουσιάζει τα διαγράμματα, *PanelDiagrams* και τέλος μια καρτέλα που μπορεί να δεχτεί τυχόν σχόλια από τον χρήστη, *PanelComments*.

Η *PropertiesWindow*, αφού στον κατασκευαστή της δημιουργήσει τις παραπάνω κλάσεις διαθέτει συναρτήσεις για την εμφάνιση κάθε μιας καρτέλας εάν και εφόσον ο χρήστης έχει επιλέξει να είναι αυτή ορατή, είτε με παράμετρο το αντικείμενο όταν πρόκειται για πλαίσιο αντικειμένου, είτε χωρίς όταν πρόκειται για το γενικό πλαίσιο της προσομοίωσης. Οι συναρτήσεις αυτές είναι οι *createPropertiesMenu()*, *createStatisticsMenu()*, *createDiagramsMenu()* και *createCommentsMenu()* για το γενικό πλαίσιο και οι *createPropertiesMenu(ObjectGeneral simobject)*, *createStatisticsMenu(ObjectGeneral simobject)*, *createDiagramsMenu(ObjectGeneral simobject)* και *createCommentsMenu(ObjectGeneral simobject)*.

Κάθε στιγμή που γίνεται κάποια αλλαγή στο σύστημα που μελετάται, αυτή η αλλαγή θα πρέπει να γίνεται ορατή και από τον χρήστη σε πραγματικό χρόνο. Για να γίνει αυτό όμως πρέπει το πλαίσιο που είναι ορατό την στιγμή της αλλαγής να ανανεώνεται κάτι που γίνεται με τη συνάρτηση *updateStatisticsPanel()* της *PropertiesWindow*. Η συνάρτηση αυτή καλείται από τις συναρτήσεις *calculateArrived* και *calculateDeparted* κάθε αντικειμένου που όπως αναφέρθηκε καλούνται κάθε φορά που πελάτης εισέρχεται σε ένα αντικείμενο ή εξέρχεται από αυτό.

Επιπλέον υποστηρίζεται ένα πλαίσιο επιλογών που εμφανίζεται με την συνάρτηση *createMenuPopup()* και διαθέτει επιλογές για απόκρυψη όλου του πλαισίου, κίνηση μεταξύ αντικειμένων, είτε προς το προηγούμενο είτε προς το επόμενο όταν πρόκειται για πλαίσιο αντικειμένων και επιλογή εμφάνισης των τίτλων των ταμπλετών είτε πάνω είτε κάτω από το πλαίσιο. Οι ενέργειες αυτές υλοποιούνται στις εσωτερικές κλάσεις, *ActionHideAll*, *ActionPrev*, *ActionNext*, *ActionTabsAtTop* και *ActionTabsAtBottom*.

Η πρώτη κλάση, *PanelProperties* που υλοποιεί την πρώτη καρτέλα, περιέχει τις γενικές επιλογές του χρήστη για το αντικείμενο ή για την προσομοίωση και έχει χωριστεί σε τρία μέρη που υλοποιούνται από αντίστοιχες κλάσεις, *PanelPropertiesLeft*, *PanelPropertiesMiddle* και *PanelPropertiesRight*. Η αρχική κλάση, *PanelProperties*, απλά περιέχει και δημιουργεί τις άλλες τρεις.

Η πρώτη κλάση υλοποιεί το αριστερό κομμάτι της καρτέλας που είναι κοινό για κάθε αντικείμενο αλλά και για το πλαίσιο της προσομοίωσης. Περιέχει μια εργαλειοθήκη, σχεδιασμένη με την *createNavigateBar()*, για λειτουργίες πλοήγησης μεταξύ των αντικειμένων και του γενικού πλαισίου ιδιοτήτων, υλοποιημένα από τις εσωτερικές κλάσεις,

ActionSimulationInfo, *ActionHideAll*, *ActionPrev* και *ActionNext*. Επιπλέον περιέχει επιλογές για την ονομασία κάθε αντικειμένου (ή του μοντέλου) και για την μορφή του πελάτη που εξέρχεται από το αντικείμενο (ή την εξορισμού μορφή των αντικειμένων σε ολόκληρο το μοντέλο). Οι επιλογές αυτές για πλαίσιο αντικειμένου και γενικό πλαίσιο σχεδιάζονται με τις συναρτήσεις *createPropertiesMenu(ObjectGeneral simobject)* και *createPropertiesMenu()* αντίστοιχα.

Η δεύτερη κλάση, *PanelPropertiesMiddle*, υλοποιεί το μεσαίο κομμάτι της καρτέλας αυτής που όπως και η τρίτη κλάση που υλοποιεί το δεξί κομμάτι της είναι διαφορετική για κάθε αντικείμενο αλλά και για το γενικό πλαίσιο. Η συνάρτηση createPropertiesMenu(ObjectGeneral simobject) και των δυο κλάσεων καλούν μια από τις συναρτήσεις, *createActivityMenu(ObjectActivity activity)*, *createBatchMenu(ObjectBatch batch)*, *createInformationMenu(ObjectInformation info)*, *createQueueMenu(ObjectQueue queue)*, *createResourceMenu(ObjectResource resource)*, *createRoutingMenu(ObjectRouting routing)* και *createTriggerMenu(ObjectTrigger trigger)* ανάλογα με το είδους του αντικειμένου. Η προτελευταία συνάρτηση για το πλαίσιο επιλογών του δρομολογητή καλεί και στις δυο κλάσεις μια άλλη κλάση, την *PanelInnerRouting* που δημιουργεί το κατάλληλο πλαίσιο επιλογών ανάλογα με τον αλγόριθμο δρομολόγησης που έχει επιλεχθεί. Όταν πρόκειται για γενικό πλαίσιο της προσομοίωσης, η συνάρτηση *createPropertiesMenu()* είναι υπεύθυνη για τον σχεδιασμό των επιλογών και στις δυο κλάσεις.

Στην κλάση *PanelStatistics*, υλοποιείται η δεύτερη καρτέλα που υλοποιεί την παρουσίαση των στατιστικών. Στην καρτέλα αυτή υπάρχουν δυο λίστες αντικειμένων, μια με τίτλους, *resultlabelslist*, και μια με πεδία με τις τιμές των στατιστικών, *resultfieldslist*. Οι λίστες αυτές γεμίζουν ανάλογα με το αντικείμενο στην *placeResultMenus(ObjectGeneral simobject)* ή όταν πρόκειται για γενικά στατιστικά της προσομοίωσης στην *placeResultMenus()*. Για να γίνει αυτό καλείται η συνάρτηση returnResult(int index) κάθε στατιστικού του αντικειμένου όταν παρουσιάζονται τιμές, η συνάρτηση *returnResultMean(int index)* ή η *returnResultDeviation(int index)* του αντικειμένου όταν παρουσιάζονται από την καρτέλα μέσες τιμές ή τυπική απόκλιση αντίστοιχα. Όταν πρόκειτε για τους τίτλους των στατιστικών, καλείται η συνάρτηση της γενικής κλάσης αντικειμένων, *ObjectGeneral*, *getResultLabel(int index)*.

Επιπρόσθετα, στην καρτέλα αυτήν, υπάρχει και μία εργαλειοθήκη η οποία επιτρέπει την κίνηση μεταξύ των στατιστικών, διαδοχικών προσομοιώσεων ή την παρουσίαση μέσου όρου ή τυπικής απόκλισης των στατιστικών αυτών. Η υλοποίηση των λειτουργιών αυτών γίνεται με τις εσωτερικές κλάσεις της *PanelStatistics*, *ActionPrev*, *ActionNext*, *ActionMean* και *ActionDeviation* που κληρονομούνε από την κλάση *ActionListener* και υπερφορτώνουν την συνάρτησή της *actionPerformed(ActionEvent e)*.

Η τρίτη καρτέλα των διαγραμμάτων υλοποιείται στην κλάση *PanelDiagrams* που περιέχει και την κλάση *PanelInnerDiagrams* που υλοποιεί τα ίδια τα διαγράμματα. Υπάρχει και πάλι, αντίστοιχη εργαλειοθήκη με αυτήν που μόλις αναφέρθηκε και κινείται μεταξύ διαδοχικών προσομοιώσεων χωρίς την τυπική απόκλιση που δεν βρίσκει εφαρμογή στα διαγράμματα. Μια λίστα, η *diagramlist*, παρουσιάζει τις επιλογές κάθε αντικειμένου αλλά και των γενικών διαγραμμάτων προσομοίωσης, ανάλογα με την περίπτωση. Στην συνάρτηση *updateList(ObjectGeneral simobject)* (ή *updateList()* για τα γενικά διαγράμματα) γεμίζει η λίστα αυτή παίρνοντας από τον πίνακα, *diagramtypes* της *ObjectGeneral* (ή της *SimulationInfo*) τα διαθέσιμα διαγράμματα.

Το κυρίως κομμάτι αυτής της καρτέλας, τα ίδια τα διαγράμματα, υλοποιείται από ξεχωριστή κλάση, την *PanelInnerDiagrams*. Πολύ σημαντική συνάρτηση αυτής της κλάσης που κληρονομεί ιδιότητές της από την *JPanel* είναι η

paintComponent(Graphics g) που καλώντας μια σειρά συναρτήσεων σχεδιάζει τα διαγράμματα. Η πρώτη συνάρτηση, *paintDataArea(Graphics2D g2)* σχεδιάζει το ορθογώνιο πάνω στο οποίο θα σχεδιαστεί οτιδήποτε περιλαμβάνεται στο διάγραμμα. Στη συνέχεια καλείται η *paintGrid(Graphics2D g2)* η οποία σχεδιάζει ένα πλέγμα για καλύτερη εμφάνιση. Έπεται η *paintTitle(Graphics2D g2)* η οποία θέτει και ζωγραφίζει σαν τίτλο του διαγράμματος, το όνομα που επιστρέφεται από το αντικείμενο ή την κλάση με τις λεπτομέρειες της προσομοίωσης, *SimulationInfo*. Στο επόμενο βήμα, καλείται η συνάρτηση paintAxisNames(Graphics2D g2) που σχεδιάζει τα ονόματα των αξόνων όπως αυτά επιστρέφονται ομοίως από το αντικείμενο ή την *SimulationInfo*. Σειρά έχουν οι συναρτήσεις που θέτουν στους άξονες τις σωστές τιμές ανάλογα με το τι έχει θέσει και πως έχει κινηθεί ο χρήστης, Αυτό γίνεται με τις συναρτήσεις *paintAxisXNumbers(Graphics2D g2)* και *paintAxisYNumbers(Graphics2D g2)* για κάθε έναν από τους δυο άξονες. Τέλος καλείται η *paintDiagram(Graphics2D g2)* η οποία με τη σειρά της καλεί την συνάρτηση *paintDiagram(Graphics2D g2)* του αντικειμένου ή της *SimulationInfo*.

Εκτός από το να σχεδιάζει τα διαγράμματα, η κλάση αυτή προσφέρει στον χρήστη μια σειρά από λειτουργίες του ποντικιού. Για το λόγο αυτό η *PanelInnerDiagrams* υλοποιεί (*implements*) την κλάση MouseListener και υπερφορτώνει τις συναρτήσεις, mouseClicked(MouseEvent e), *mousePressed(MouseEvent e*), *mouseReleased(MouseEvent e)*, *mouseEntered(MouseEvent e)* και *mouseExited(MouseEvent e)*. Οι δυο τελευταίες δεν υλοποιούν τίποτα, η πρώτη δημιουργεί ένα πλαίσιο επιλογών καλώντας την *createPopupMenu()*. Οι *mousePressed* και *mouseReleased* χρησιμοποιούνται σε συνδυασμό για να επιτρέψουν στον χρήστη να κινηθεί μέσα στο διάγραμμα. Στο πλαίσιο επιλογών που δημιουργείται από την *createPopupMenu* υπάρχουν επιλογές για εστίαση και στους δυο άξονες ή σε έναν από αυτούς και επαναφορά στις αρχικές τους ρυθμίσεις ανάλογα με το διάγραμμα που έχει επιλεγεί.

Η τελευταία, πιο απλή καρτέλα απλά επιτρέπει στον χρήστη να σημειώσει κάποιο σχόλιο για το συγκεκριμένο αντικείμενο ή το παράδειγμα προσομοίωσης. Υλοποιείται στην κλάση *PanelComments* η οποία περιλαμβάνει δυο κύρια αντικείμενα, το πεδίο, *namefield*, για την ονοματοδοσία του αντικειμένου ή παραδείγματος και το πεδίο, *commentarea*, για την σημείωση σχολίων.

2.5.1.2 Πλαίσιο ιδιοτήτων πελάτη

Στα αριστερά του μοντέλου, ο χρήστης μπορεί να έχει πρόσβαση σε κάποια στατιστικά στοιχεία αλλά και σε ένα διάγραμμα κάθε πελάτη μέσω ενός πλαισίου που υλοποιείται από την κλάση *PanelCustomer* με την *PanelCustomerDiagram* για τον σχεδιασμό του διαγράμματος. Το πλαίσιο περιλαμβάνει μία εργαλειοθήκη που επιτρέπει είτε την κίνηση μεταξύ των πελατών είτε την απόκρυψη του συγκεκριμένου πλαισίου. Οι λειτουργίες αυτές υλοποιούνται από τις εσωτερικές κλάσεις *ActionHide*, *ActionPrev* και *ActionNext*. Περιλαμβάνει επίσης μια σειρά από πεδία με διάφορα στατιστικά στοιχεία όπως τον χρόνο που έχει εισέλθει ο πελάτης μέσα στο σύστημα, ο χρόνος εξυπηρέτησης αλλά και αναμονής, τα οποία δημιουργούνται στην συνάρτηση createCustomersMenu() αλλά ανανεώνονται στη συνάρτηση updateCustomersMenu(Customer customer) η οποία καλείται από κάθε αντικείμενο και κάθε φορά που ο πελάτης εισέρχεται ή εξέρχεται από κάθε αντικείμενο. Για το λόγο αυτό καλείται σε κάθε συνάρτηση calculateArrived(Customer customer) και calculateDeparted(Customer customer) κάθε αντικειμένου. Υπεύθυνη για τον σχεδιασμό του διαγράμματος πελάτη είναι η paintComponent(Graphics g) της PanelCustomerDiagram που σχεδιάζει το ορθογώνιο πάνω στο οποίο θα

σχεδιαστεί το διάγραμμα αλλά και καλεί την paintDiagram(Graphics2D g2) της *PanelCustomer* που θα σχεδιάσει το ίδιο το διάγραμμα.

2.5.1.3 Πλαίσιο ιδιοτήτων επόμενου γεγονότος

Το πλαίσιο αυτό παρέχει πληροφορίες για τα γεγονότα της προσομοίωσης και υλοποιείται στην κλάση *PanelEvents*. Περιλαμβάνει μια εργαλειοθήκη με ένα πεδίο, *viewpasteventscheckbox*, με το οποίο ο χρήστης επιλέγει εάν θέλει να είναι ορατά και τα προηγούμενα γεγονότα ή μόνο τα επόμενα. Αυτό μπορεί να γίνει γιατί η εφαρμογή αντί να κρατά μια μόνο λίστα, την *FEList* των μελλοντικών γεγονότων, στη κλάση *FutureEventList* κρατά άλλη μια, την *PEList* που κρατά όλα τα γεγονότα που έχουν ήδη συμβεί. Επιλέγοντας στο πεδίο την εμφάνιση ή όχι των γεγονότων αυτών καλείται η εσωτερική κλάση *ViewPastEventAction* που προσθέτει ή όχι στο μοντέλο τα παρελθόντα γεγονότα. Επιπρόσθετα στην εργαλειοθήκη υπάρχει και ένα κουμπί, *hidebutton*, που πατώντας το ο χρήστης εξαφανίζει όλο το πλαίσιο από την περιοχή του μοντέλου.

Για την εμφάνιση όλων των γεγονότων ή όπως αναφέρθηκε μόνο των μελλοντικών, χρησιμοποιείται ένας πίνακας της κλάσης *JTable*, ο *FELtable* και ένα μοντέλο αυτού, της κλάσης *DefaultTableModel*, το *model*. Στο μοντέλο, υπάρχουν δύο στήλες, αυτές του πίνακα *columnnames[]* μια για τον χρόνο του γεγονότος και μια με το όνομα αυτού. Υπάρχει και μια συνάρτηση η οποία αδειάζει το μοντέλο από όλα τα γεγονότα εάν κάποια προσομοίωση έχει τελειώσει, η *clearFutureEventTable()*. Κάθε φορά που προστίθεται ένα γεγονός στη λίστα με τα μελλοντικά γεγονότα της *FutureEventList* προστίθεται και ένα γεγονός στο μοντέλο του πίνακα.

2.5.1.4 Πλαίσιο βοήθειας εφαρμογής

Για να δημιουργηθεί το σύστημα βοήθειας της εφαρμογής απαιτήθηκαν μια σειρά από κλάσεις η οποία δημιουργήθηκε σε ένα ξεχωριστό πακέτο, το *help*. Η αρχική κλάση, η *FrameDocumentation* κληρονομεί τις ιδιότητές της από την *JFrame* που δημιουργεί ένα ξεχωριστό παράθυρο με τα θέματα βοήθειας. Το παράθυρο χωρίζεται σε τρία μέρη, μία εργαλειοθήκη που υλοποιείται από την κλάση *FDToolBar*, το αριστερά πλαίσιο που υλοποιείται από την κλάση *FDLeftPanel* και το δεξί πλαίσιο από την *FDRightPanel*.

Στο παράθυρο περιλαμβάνεται αρχικά μία εργαλειοθήκη, η *FDToolBar*, η οποία επιτρέπει την κίνηση στην προηγούμενη, με το κουμπί *backbutton* και την κλάση *ActionBack*, ή την επόμενη σελίδα βοήθειας, *forwardbutton* και *ActionForward* καθώς επίσης και την επιστροφή στην αρχική, *homebutton* και *ActionHome* αντίστοιχα. Σημειώνεται ότι οι λειτουργίες αυτές αφορούν επιστροφή στην προηγούμενη η επόμενη σελίδα που έχει επισκεφτεί και λειτουργεί εν ήδη undo/redo. Ο τρόπος υλοποίησης θα αναλυθεί στη συνέχεια. Επιπρόσθετα περιλαμβάνεται και μια λειτουργία *showorhidebutton* και *ActionShowOrHide* που αποκρύπτει ή εμφανίζει το αριστερό πλαίσιο εφαρμογών που υλοποιείται από την *FDLeftPanel* και αναλύεται στη συνέχεια.

Το αριστερό πλαίσιο, λοιπόν, στην *FDLeftPanel* περιλαμβάνει δύο καρτέλες, την *"Contents"* που περιέχει τα περιεχόμενα όλων των σελίδων βοήθειας και την *"Index"* που περιέχει μια σειρά από λέξεις κλειδιά προς αναζήτηση από τον χρήστη. Η πρώτη καρτέλα υλοποιείται στην κλάση *FDContentsPanel* που ουσιαστικά περιέχει απλά ένα δέντρο τύπου *JTree* το *tree* που περιλαμβάνει τους τίτλους κάθε html αρχείου βοήθειας και έναν σύνδεσμο προς αυτό. Η δεύτερη καρτέλα

υλοποιείται από την κλάση *FDIndexPanel*, η οποία περιλαμβάνει ένα πεδίο, το *keywordfield*, στο οποίο ο χρήστης μπορεί να θέσει μια λέξη προς αναζήτηση και έναν πίνακα, τον *indexitems*, με υποψήφιες λέξεις ή φράσεις προς αναζήτηση. Σε συνδυασμό με τον πίνακα αυτό, υπάρχει ένας πίνακας ίσου μεγέθους, ο *urllist*, που περιλαμβάνει τόσες λίστες *LinkedList* όσες το μέγεθός του. Κάθε μία από αυτές τις λίστες περιλαμβάνει μια σειρά από συνδέσμους στα αρχεία βοήθειας μορφής html που περιλαμβάνονται στον πίνακα *htmlfiles* της κλάσης *FrameDocumentation*. Οι λίστες αυτές δημιουργούνται στην συνάρτηση *createURLList()*. Στην συνάρτηση *displayPage(int index)* υλοποιείται η συμπεριφορά του πλαισίου όταν επιλεχτεί κάποιο από τα στοιχεία του πίνακα *indexitems* με τις λέξεις ή φράσεις κλειδιά. Έτσι εάν η λίστα που αντιστοιχεί στη λέξη ή φράση είναι μεγέθους ένα, απευθείας εμφανίζεται στο δεξί μέρος του *FrameDocumentation* με μια διαδικασία που εξηγείται στη συνέχεια η σελίδα της οποίας ο σύνδεσμος υπάρχει στη λίστα. Εάν όμως η λίστα περιλαμβάνει περισσότερα από δύο στοιχεία, ο χρήστης πρέπει να επιλέξει κάποιο από αυτά και αναγκαστικά απαιτείται μια νέα διεπαφή. Η διεπαφή αυτή υλοποιείται στην *FDSelectFrame*. Η υλοποίηση της συνάρτησης *displayPage* φαίνεται στη συνέχεια.

```
public void displayPage(int index){
        if (urllist[index].size()==1)
                (new FDCommand(df,(FDHelpHTML)urllist[index].get(0))).execute();
        else if (urllist[index].size()>1)
                df.fdselect.updateList(index);
}
```

Η κλάση *FDSelectFrame* υλοποιεί ένα παράθυρο στο οποίο γίνεται η παραπάνω επιλογή. Το παράθυρο περιέχει μία λίστα, την list τύπου JList και το μοντέλο της, model, μέσω του οποίου την διαχειριζόμαστε. Επίσης δύο κουμπιά, τα displaybutton και cancelbutton, τα οποία είτε εμφανίζουν την επιλεγμένη σελίδα από τη λίστα είτε ακυρώνουν την επιλογή και αποκρύπτουν το παράθυρο. Οι λειτουργίες αυτές εκτελούνται από τις *ActionDisplay* και *ActionCancel*. Η τελευταία απλά αποκρύβει το παράθυρο ενώ η πρώτη καλεί την συνάρτηση *displayPage(int index)* η οποία είναι παρόμοια με την ομόνυμη συνάρτηση της *FDIndexPanel* με την προσθήκη της εντολής απόκρυψης παραθύρου. Φυσικά στην περίπτωση της δεν υπάρχει περίπτωση να υπάρχει παραπάνω από μια σελίδα και δεν είναι απαραίτητο το δεύτερο σκέλος της if.

```
public void displayPage(int index){
        (new FDCommand(df,(FDHelpHTML)df.leftpanel.indexpanel.urllist[listindex].get(index))).execute();
        this.setVisible(false);
 }
```

Το δεξί μέρος του πλαισίου στο οποίο εμφανίζονται οι σελίδες βοήθειας υλοποιείται από την κλάση *FDRightPanel*. Το πάνω μέρος του αποτελείται από τον τίτλο της κάθε σελίδας, *titlelabel* και δύο κουμπιά, *prevbutton* και *nextbutton* για κίνηση μεταξύ των σελίδων. Η κίνηση επιτυγχάνεται με δύο εσωτερικές κλάσεις της *FDRightPanel*, την *ActionPrev* και *ActionNext* στις οποίες ελέγχεται εάν η σελίδα είναι η πρώτη ή η τελευταία του πίνακα *htmlfiles* της *FrameDocumentation*. Στις περιπτώσεις αυτές δεν υπάρχει επόμενη ή προηγούμενη σελίδα και η πιθανή ενέργεια δεν εκτελεί τίποτα. Στη συνέχεια φαίνεται ενδεικτικά ο κώδικας για την *ActionPrev*.

```
class ActionPrev implements ActionListener{
        public void actionPerformed(ActionEvent e){
                if (editpanel.currentHTML.index!=0){
```

```
                (new FDCommand(df,FrameDocumentation.htmlfiles[editpanel.currentHTML.index-1])).execute();
            }
        }
}
```

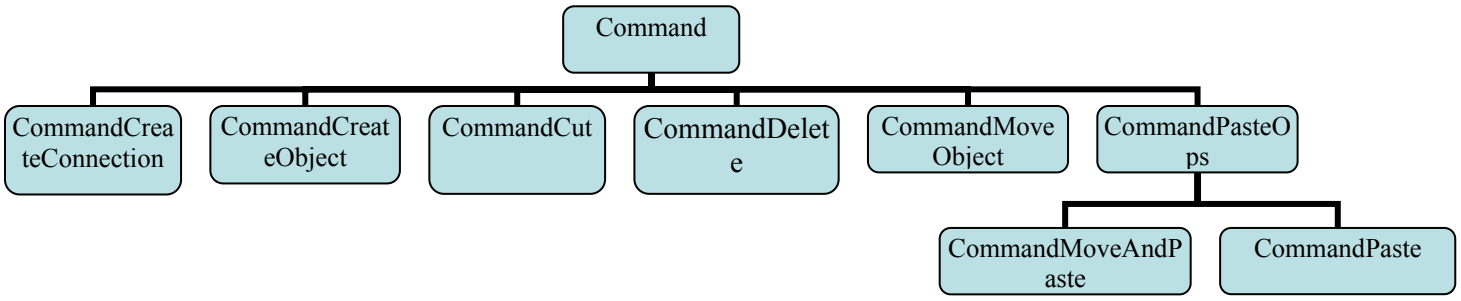
Έχει γίνει αντιληπτό από όλες τις προηγούμενες αναφορές ότι οποιαδήποτε αλλαγή στην σελίδα που εμφανίζεται δεξιά στο πλαίσιο βοήθειας υλοποιείται με την δημιουργία μιας νέας κλάσης τύπου *FDCommand*. Αυτό γίνεται για την λειτουργία undo / redo που υποστηρίζεται για τις σελίδες που έχουν επισκεφτεί. Η κλάση αυτή περιέχει τις συναρτήσεις *execute()*, *unexecute()* και *reexecute()* και κάθε αλλαγή σελίδας, δημιουργεί μια τέτοια κλάση που αποθηκεύεται στη λίστα *backlist*. Κάθε επιστροφή στην προηγούμενη σελίδα οδηγεί στην διαγραφή του τελευταίου αντικειμένου τύπου *FDCommand* από την λίστα και την εκτέλεση της *unexecute()* που οδηγεί στην προηγούμενη σελίδα και προσθέτει το αντικείμενο στην *forwardlist*. Αντίστροφα κάθε επίσκεψη στην επόμενη σελίδα, οδηγεί στην διαγραφή του τελευταίου αντικειμένου της *forwardlist* και την εκτέλεση της *reexecute()* που οδηγεί στην επόμενη σελίδα και προσθέτει το αντικείμενο στην *backlist*. Η προσέγγιση αυτή χρησιμοποιείται στον μηχανισμό undo/redo της διεπαφής και εξηγείται πιο εμπεριστατωμένα στη συνέχεια.

2.5.2 Χειρισμός μοντέλου

2.5.2.1 Δημιουργία μηχανισμού Undo / Redo

Για την υλοποίηση του απαραίτητου αυτού μηχανισμού, χρειάστηκαν δύο λίστες που κρατούν κάθε πιθανή εντολή, για την οποία κρίνεται επιθυμητή η δυνατότητα επιστροφής στην προηγούμενη κατάσταση, κάτι που γίνεται για όλες τις λειτουργίες διεπαφής. Οι λίστες αυτές, μία για να κρατάει τις εντολές που εκτελέστηκαν και μία για να κρατάει τις εντολές, για τις οποίες ήδη έγινε αναίρεση – undo, κρατούνται στην κλάση *AreaDrawing*, η οποία είναι ξεχωριστή για κάθε ανοιχτό μοντέλο, για να υπάρξει διαφορετικός μηχανισμός undo, redo για κάθε ένα από αυτά. Οι λίστες αυτές, που ονομάζονται *undoHistoryList* και *redoHistoryList*, αρχικοποιούνται με τη δημιουργία της *AreaDrawing* στον κατασκευαστή της, που γίνεται είτε κατά τη δημιουργία νέου μοντέλου, είτε κατά το άνοιγμα υπάρχοντος.

Οι εντολές, για τις οποίες υποστηρίζεται η παραπάνω λειτουργία, είναι μόνο εντολές διεπαφής και κάθε μία από αυτές θα πρέπει να κληρονομεί από την κλάση *Command*. Οι εντολές αυτές είναι η δημιουργία αντικειμένου και η δημιουργία συνδέσμου, η αποκοπή και η διαγραφή, η μετακίνηση, η επικόλληση και η μετακίνηση με επικόλληση. Κάθε μία από αυτές θα πρέπει να υλοποιεί τις συναρτήσεις *execute()*, *unexecuted()* και *reexecute()* της κλάσης *Command*. Συγκεκριμένα, η επικόλληση και η μετακίνηση με επικόλληση λόγω κάποιων ομοιοτήτων στην υλοποίηση επιλέχτηκε να κληρονομούν από μία κοινή κλάση κάποιες ιδιότητες, η οποία με τη σειρά της κληρονομεί από την κλάση *Command*.



Για να εκτελεστεί οποιαδήποτε εντολή, θα πρέπει πρώτα να δημιουργηθεί το αντίστοιχο στιγμιότυπο που έχει κληρονομήσει από την command, αυτό να εκτελεστεί, και κατά την εκτέλεση να μπει και στη λίστα *undoHistoryList* για πιθανή αναίρεση. Στην αναίρεση αυτή, αφού εκτελεστεί η αντίστοιχη συνάρτηση *unexecuted()*, θα πρέπει η εντολή να αφαιρεθεί από τη λίστα αυτή και να προστεθεί στη λίστα *redoHistoryList*. Η redo θα απαιτήσει την εκτέλεση της *reexecute()* και την αλλαγή και πάλι της λίστας.

Υπάρχει, επίσης, η επιλογή της εκτέλεσης undo ή redo περισσοτέρων από μία εντολών. Στην περίπτωση αυτή εκτελείται μία επανάληψη, κατά την οποία εκτελούνται στη σειρά όλες οι εντολές στη αντίστοιχη λίστα, μέχρι το σημείο *start* που έχει επιλεγεί.

```
for(int i=da.undoHistoryList.size()-1;i>=start;i--){
        command = (Command)da.undoHistoryList.removeLast();
        command.unexecute();
}
```

Επειδή η διατήρηση κάθε πιθανής κίνησης του χρήστη στη μνήμη μπορεί να γίνει πολυδάπανη σε υπολογιστικούς πόρους, δίνεται στο χρήστη η επιλογή του βάθους της λειτουργίας undo/redo. Για να υλοποιηθεί, χρειάζεται απλά, κάθε φορά που προσθέτουμε ένα οποιοδήποτε αντικείμενο στη λίστα, να γίνεται η σύγκριση του μεγέθους της με το βάθος που έχει επιλεχθεί. Όσο το βάθος γίνεται μεγαλύτερο, βγαίνουν αντικείμενα από την αρχή της λίστας που είναι στοιχεία που έχουν κρατηθεί πέρα από το ζητούμενο βάθος.

```
da.undoHistoryList.add(this);
while(da.undoHistoryList.size()>da.simulationinfo.getUndoDepth())
        da.undoHistoryList.removeFirst();
```

2.5.2.2 Λειτουργία Cut – Copy - Paste

Για τη λειτουργία αυτή κρατούνται δύο λίστες, *copyobjectList* και *copyconnectionList*, για κάθε αντικείμενο αλλά και για κάθε σύνδεση που έχει αντιγραφεί (copy), ή αποκοπεί (cut). Οι λίστες αυτές κρατούνται στην κλάση Simulation, γιατί, αντίθετα με το μηχανισμό undo / redo, οι λειτουργίες αυτές θα πρέπει να μπορούν να συνδυάζουν πολλά ανοιχτά μοντέλα. Θα πρέπει, δηλαδή, να μπορεί ο χρήστης να αντιγράφει ή να αποκόπτει ένα αντικείμενο ή μία ομάδα αντικειμένων και να τα επικολλάει σε ένα άλλο μοντέλο που έχει ανοίξει σε διπλανή καρτέλα.

Αφού κατά τη διάρκεια της διαδικασίας αντιγραφής (ή αποκοπής) έχει κρατηθεί η λίστα όλων των αντικειμένων, στην αντιγραφή τους θα δημιουργηθούν νέα αντικείμενα, τα οποία, όμως, θα έχουν τις ίδιες ιδιότητες με τα αρχικά αντικείμενα της λίστας. Ωστόσο, κάθε είδος αντικειμένου έχει δικά του χαρακτηριστικά που θα πρέπει να αντιγραφούν. Έτσι, σε κάθε διαφορετική κλάση αντικειμένων υπάρχει μία συνάρτηση, η *createObjectCopy(AreaDrawing da,Point point)*, η οποία, αφού δημιουργηθεί το καινούργιο αντικείμενο, αντιγράφει όλα τα χαρακτηριστικά από το αντικείμενο, το οποίο προκύπτει. Η συνάρτηση αυτή δημιουργεί και επιστρέφει το νέο αντικείμενο, το οποίο έχει όμως σαν θέση, τη θέση point στο μοντέλο της περιοχής *da*. Στην περίπτωση αντιγραφής ολόκληρης περιοχής, όλη η περιοχή θα έχει σαν θέση το σημείο αυτό και κάθε αντικείμενο τη σχετική θέση του αντικειμένου στην αρχική επιλεγμένη περιοχή, αλλά υπολογισμένο στη νέα.

Επιπρόσθετα, έχει υλοποιηθεί και η λειτουργία μετακίνησης και επικόλλησης, στην οποία, με το ταυτόχρονο πάτημα του ctrl, ο χρήστης μπορεί να σύρει το ή τα προς αντιγραφή αντικείμενα και να τοποθετήσει ένα αντίγραφό τους σε όποιο σημείο του μοντέλου επιθυμεί. Η υλοποίηση της λειτουργίας γίνεται στην κλάση *CommandMoveAndPaste*, η οποία δε χρησιμοποιεί τις λίστες που αναφέρθηκαν προηγουμένως. Αυτό επιτρέπει στο πρόγραμμα να κρατά τις λίστες με τα αντιγραμμένα στοιχεία μέχρι την επόμενη αντιγραφή και να διαχωρίζει εντελώς τη λειτουργία μετακίνησης και επικόλλησης από αυτή της αντιγραφής ή αποκοπής και επικόλλησης. Η αντιγραφή στην περίπτωση της *CommandMoveAndPaste* μπορεί να γίνει ταυτόχρονα με την επικόλληση και έτσι να μην είναι απαραίτητη η χρήση κάποιας λίστας σε κάποια άλλη κλάση, παρά μόνο σε αυτή που υλοποιεί και τη λειτουργία. Για την ακρίβεια, χρησιμοποιούνται δυο λίστες, μία για τα προς αντιγραφή αντικείμενα, η *simobjectoriginalList*, και μία για τα νέα αντικείμενα, η *simobjectList*. Οι λίστες αυτές είναι απαραίτητες για το μηχανισμό undo / redo.

Εφόσον λοιπόν και για όλες τις παραπάνω διαδικασίες είναι απαραίτητη η επιλογή undo / redo, οι λειτουργίες αυτές θα πρέπει να εκτελούνται μέσα από τις κλάσεις *CommandCopy*, *CommandPaste*, *CommandCut και CommandMoveAndPaste* που κληρονομούν από την *Command* και να κρατούνται στις ανάλογες λίστες, όπως αναφέρθηκε σε προηγούμενη παράγραφο.

2.5.2.3 Μετακίνηση αντικειμένων

Για τη μετακίνηση ενός η περισσότερων αντικειμένων στο simmate δεν είναι απαραίτητη η χρήση κάποιας λίστας, γιατί η μετακίνηση και μόνο ενός αντικειμένου δεν απαιτεί τη δημιουργία νέου, αλλά απλά την αλλαγή κάποιων ιδιοτήτων του. Οι ιδιότητες αυτές έχουν να κάνουν με τη θέση και μόνο του αντικειμένου στην περιοχή του μοντέλου και με τη μεταβλητή που την καθορίζει *objP*. Έτσι, με τη βοήθεια της συνάρτησης *moveObject* της *ObjectGeneral* που είναι ακριβώς η ίδια για όλα τα αντικείμενα που κληρονομούν από αυτή, τίθεται το σημείο του αντικειμένου σαν το νέο σημείο, στο οποίο αυτό μετακινείται. Σε περίπτωση που έχουμε παραπάνω από ένα αντικείμενα προς μετακίνηση, η συνάρτηση *moveObjects* της *AreaSelected* είναι υπεύθυνη για να καλέσει τη συνάρτηση *moveObject* στα περικλείοντα αντικείμενα

Εφόσον και αυτή, όπως και κάθε λειτουργία διεπαφής, υποστηρίζει λειτουργίες undo / redo, η παραπάνω διαδικασία θα πρέπει να υλοποιηθεί μέσα από την κλάση *CommandMoveObject* που κληρονομεί από την *Command*.

2.5.3 Ανάδραση

Σημαντικό στα πλαίσια της ευχρηστίας του προγράμματος, είναι να παρέχεται στο χρήστη ανάδραση για την κάθε αλληλεπίδρασή του με τη διεπαφή της εφαρμογής. Αρχικά, κάθε διαφορετικό συστατικό έχει και διαφορετικό κέρσορα, ο οποίος παίρνει τη μορφή χεριού, όταν το ποντίκι βρεθεί πάνω από αντικείμενο ή γραμμοσκιασμένη περιοχή, τη μορφή μολυβιού, όταν βρεθεί πάνω από σύνδεσμο, και ανάλογη μορφή με τα προς δημιουργία αντικείμενα οπουδήποτε στην περιοχή του μοντέλου. Για τις λειτουργίες ανάδρασης, δημιουργήθηκε η κλάση *DragAndDropAnimation*, της οποίας συναρτήσεις είναι υπεύθυνες για την αλλαγή του κέρσορα, καθώς και για την παρουσίαση κάποιων γραφικών ανάλογα με τη λειτουργία που εκτελεί ο χρήστης.

Έτσι, ενώ υπάρχουν οι εξ ορισμού κέρσορες σε περίπτωση που ο χρήστης πατήσει πάνω σε ένα αντικείμενο και σύρει το ποντίκι, πράγμα που σημαίνει ότι θέλει να το μετακινήσει, ο κέρσορας θα αλλάξει μορφή σε ένα άλλου τύπου χέρι

και ένα ορθογώνιο με μία απόχρωση του γκρι θα ακολουθεί τον κέρσορα, για να δείξει την πιθανή νέα θέση του αντικειμένου. Κάθε φορά που έχουμε μία ενέργεια πατήματος ποντικιού *mousePressed* στο αντικείμενο, καλείται η συνάρτηση *setShapeType* της *DragAndDropAnimation*, η οποία και ζωγραφίζει το ορθογώνιο μέσω της paintMyself και αλλάζει και τον κέρσορα με την *setCursorType*. Αντίστοιχα, μία ενέργεια ελευθέρωσης του ποντικιού mouseReleased θα επαναφέρει τον κέρσορα στην αρχική μορφή, για να δείξει στο χρήστη ότι η ενέργειά του επιτελέστηκε. Αν, κατά την ίδια κίνηση, ο χρήστης κρατά πατημένο το Control, πράγμα που σημαίνει ότι θέλει να αντιγράψει και να επικολλήσει το αντικείμενο, ο κέρσορας θα πάρει μία άλλη γνώριμη μορφή από άλλα τέτοια προγράμματα που το υποστηρίζουν. Αντίστοιχες είναι και οι συναρτήσεις της *AreaSelected* που υλοποιεί τη γραμμοσκιασμένη περιοχή, αλλά και στην περίπτωση του συνδέσμου που υλοποιείται με τη συνάρτηση *Connector*.

Όλες αυτές οι κλάσεις που υλοποιούν συστατικά του μοντέλου και βρίσκονται στην περιοχή προς σχεδίαση, αλλά και η ίδια η περιοχή, κληρονομούν τις ιδιότητες της κλάσης *JComponent*, ώστε να μπορούν με ευκολία να υλοποιήσουν τις παραπάνω λειτουργίες.

2.5.4 Κίνηση

Επιπλέον, εκτός από την ανάδραση κατά τη διαδικασία σχεδίασης του μοντέλου στην εφαρμογή, υπάρχει και ανάδραση κατά την εκτέλεση του μοντέλου με τη βοήθεια της κίνησης που υποστηρίζεται. Η συνάρτηση που ξεκινά την κίνηση αυτή είναι η *initializeAnimation()* της *SimulationRun* που καλείται από την initiateSimulation(), η οποία και ξεκινά την προσομοίωση δημιουργώντας όλα τα αρχικά γεγονότα, *generateInitialEventsAndPlaceOnFEL()* και καλώντας το πρώτο προς εκτέλεση *processEvent()*. Η *initializeAnimation()* είναι υπεύθυνη να ξεκινήσει το χρονόμετρο *Timer*, θέτοντας και τις κατάλληλες παραμέτρους, εάν αυτό δεν τρέχει ήδη. Η *processEvent()* κάθε φορά θα εκτελεί το επόμενο προς εκτέλεση γεγονός, το οποίο με τη σειρά του πιθανώς θα δημιουργεί τα επόμενα. Κατά τη διάρκεια της εκτέλεσης κάθε γεγονότος προστίθεται κάθε προς κίνηση πελάτης στη λίστα *customersanimatedList*, αλλά και κάθε σύνδεση, από την οποία θα περάσει ο πελάτης σε άλλη λίστα *routelist* στην κλάση του πελάτη *Customer*. Στη συνέχεια, η *processEvent()*θα καλεί την συνάρτηση *animateCustomers()*, η οποία παίρνει τη λίστα και, είτε καλεί τη συνάρτηση startAnimation(), είτε τη συνάρτηση startLostAnimation() της κλάσης *Customer*, ανάλογα με το αν πρόκειται για κανονική κίνηση ή κίνηση χαμένου πελάτη. Οι συναρτήσεις αυτές κάνουν τον πελάτη ορατό, setVisible(true), ξεκινούν την ενέργεια, da.*simulationrun.timer.addActionListener(actionmovement)* ή *da.simulationrun.timer.addActionListener(actionlost)*, η οποία θέτει σε κίνηση τον πελάτη με τη βοήθεια του χρονομέτρου.

Σε κάθε πελάτη που ξεκινά την κίνηση, έχει δημιουργηθεί, όπως ειπώθηκε, μία λίστα με τους συνδέσμους που θα πρέπει να κινηθεί, η *routelist*. Από σύνδεση σε σύνδεση ο πελάτης κινείται με τη βοήθεια της συνάρτησης *nextConnection()*, μέχρι να μην υπάρχει άλλη σύνδεση και η κίνησή του να σταματήσει.

```
if (!nextConnection()){
    stopAnimation();
}
```

Μέσα στην ίδια την σύνδεση ο πελάτης κινείται με τη βοήθεια της συνάρτησης *move()*, η οποία, όταν θα τελειώσει την απόσταση που πρέπει να διανύσει ο πελάτης, δηλαδή όλη τη σύνδεση, θα καλέσει την *nextConnection()*.

Όταν υπάρχει κίνηση στο μοντέλο, υπάρχουν και κάποιες εικόνες που αντιπροσωπεύουν πελάτες. Κάθε πελάτης που κινείται στο μοντέλο βρίσκεται σε μία από τέσσερις διαφορετικές δυνατές καταστάσεις. Είτε κινείται, είτε περιμένει, καθώς έχει πατηθεί το pause, είτε χάνεται από το σύστημα, είτε βρίσκεται σε κάποιο εξυπηρετητή και εξυπηρετείται. Για κάθε μία από αυτές τις καταστάσεις, ο πελάτης έχει και διαφορετικό είδωλο που το αντιπροσωπεύει. Όλα αυτά τα είδωλα υπάρχουν στο φάκελο images/customers μέσα στο jar αρχείο και σε διαφορετικό φάκελο για το κάθε διαφορετικό είδωλο. Αρχικοποιούνται δε στη κλάση *Simulation* όπως και κάθε εικόνα που χρησιμοποιείται στο πρόγραμμα. Οι πίνακες με τις εικόνες αυτές όμως βρίσκονται στην αρχική κλάση που καλεί και το applet, τη *Simulation*. Είναι τέσσερις πίνακες, *animationwalkingimage*, *animationwaitingimage*, *animationservedimage*, *animationlostimage* που περιέχουν τις εικόνες ανάλογα με την κατάσταση του πελάτη.

Για να μπει στο πρόγραμμα κάποια καινούργια εικόνα που θα αντιπροσωπεύει πελάτες θα πρέπει αρχικά να δημιουργηθεί νέος φάκελος στο images/customers με κάποιο όνομα, έστω birds. Εκεί θα βάλουμε τέσσερεις εικόνες, έστω τις lost.gif, served.gif, waiting.gif και walking.gif που αντιπροσωπεύουν τον πελάτη στις τέσσερεις πιθανές του καταστάσεις. Στην Simulation και στους πίνακες *animationwalkingpath*, *animationwaitingpath*, *animationservedpath* και *animationlostpath* απλά προσθέτουμε τις διαδρομές για την νέα εικόνα. Για παράδειγμα, στον πρώτο πίνακα θα προστεθεί το στοιχείο *"images/customers/bird/served.gif"*. Πλέον οι νέες εικόνες έχουν εμπεδωθεί στο πρόγραμμα και το επόμενο βήμα είναι να μπορεί ο χρήστης να τις επιλέξει. Απλά θα πρέπει να προστεθεί στον πίνακα imagemenu της κλάσης PanelPropertiesLeft που υλοποιεί το αντίστοιχο menu το νέο του στοιχείο "Bird".

2.6 Το σύστημα αρχείων

Για λόγους ασφαλείας, για να γίνει οποιαδήποτε λειτουργία αποθήκευσης ή ανάκτησης αρχείων από το υπολογιστικό σύστημα του χρήστη, η ίδια η γλώσσα προγραμματισμού java που χρησιμοποιήθηκε, απαιτεί μία διαδικασία πιστοποίησης. Υπάρχουν διάφοροι προμηθευτές τέτοιων πιστοποιήσεων [http://www.verisign.ch], αλλά στα πλαίσια ανάπτυξης της εργασίας αυτής προτιμήθηκε η δημιουργία μίας νέας πιστοποίησης. Γενικώς άλλωστε κάθε στάδιο της ανάπτυξης του παρόντος βασίστηκε σε ελεύθερο λογισμικό, καθώς και λογισμικό ανοιχτού κώδικα, όπως και το ίδιο το simmate. Προτού όμως δοθεί η περιγραφή της διαδικασίας στην παρούσα εργασία, καλό θα ήταν να γίνει αναφορά σε κάποιους από τους όρους που θα χρησιμοποιηθούν στη συνέχεια.

Applet: Ένα πρόγραμμα java που τρέχει απευθείας μέσω του προγράμματος φυλλομετρητή (π.χ. Mozilla Firefox) που χρησιμοποιούμε. Η html σελίδα που φορτώνει στον φυλλομετρητή περιέχει μία ετικέτα <applet> που τον ενημερώνει για το που θα βρει την κλάση java. Το simmate αποτελεί ένα τέτοιο applet και η ετικέτα στην html σελίδα που θα το καλέσει έχει την παρακάτω μορφή.

```
<applet code="Simulation.class"
        archive="SSimmate.jar,jdom.jar"
        width="800" height="600">
</applet>
```

Στην προκειμένη περίπτωση η Simulation.class είναι η αρχική κλάση του προγράμματος που περιέχει το applet, χρησιμοποιεί όμως πληθώρα άλλων βοηθητικών αρχείων, όπως επιπλέον κλάσεις, αρχεία εικόνων, αρχεία html για το

σύστημα βοήθειας που υποστηρίζει. Όπως θα φανεί στη συνέχεια όλα αυτά τα αρχεία βρίσκονται στο SSimmate.jar αρχείο. Το jdom.jar είναι μία επιπλέον βιβλιοθήκη για την χρήση xml που θα αναλυθεί στην συνέχεια.

Εξυπηρετητής(Server): Ο υπολογιστής, ο οποίος φιλοξενεί (host) τη σελίδα html, η οποία περιέχει την παραπάνω ετικέτα, αλλά και όλα τα αρχεία που συναποτελούν το applet, όπως το jar αρχείο που αναφέρθηκε. Όταν κάποιος στο διαδίκτυο συνδέεται στη σελίδα που περιέχει το applet, ο εξυπηρετητής διανέμει τα αρχεία αυτά.

Πελάτης(Client): Ο υπολογιστής, ο οποίος εμφανίζει τη σελίδα που περιέχει το applet.

Εξ ορισμού, τα applet που φορτώνουν μέσω διαδικτύου, έχουν όσο το δυνατόν λιγότερες δυνατότητες πρόσβασης στο σύστημα του πελάτη που τα κάλεσε. Έτσι, όπως προαναφέρθηκε, δε μπορούν να γράψουν ή να διαβάσουν αρχεία του πελάτη, δε μπορούν να κάνουν συνδέσεις δικτύου, καθώς επίσης να φορτώσουν βιβλιοθήκες ή να ξεκινήσουν άλλα προγράμματα στον υπολογιστή του πελάτη. Τα applets αυτά θεωρούνται αναξιόπιστα, όπως και κάθε applet που θα φορτωθεί στον υπολογιστή μας, εκτός και αν γίνει κάτι από τα παρακάτω για να θεωρηθεί αξιόπιστο.

- Το applet θα αποθηκευτεί τοπικά στον υπολογιστή και από εκεί θα φορτωθεί μέσω κάποιου φυλλομετρητή.
- Θα υπογραφεί σαν εμπιστοσύνης στην τοπική βάση του πελάτη.

Το πρόγραμμα που αναπτύχθηκε στα πλαίσια της εργασίας αυτής είχε σαν στόχο την απλότητα και την ευκολία χρήσης και γι' αυτό το λόγο, για την πρόσβαση στα αρχεία κάθε πιθανού χρήστη προτιμήθηκε ο δεύτερος τρόπος που δεν τον υποχρεώνει να αποθηκεύει το πρόγραμμα τοπικά, αλλά τον υποχρεώνει να εμπιστευτεί το δημιουργό του προγράμματος με τη διαδικασία που θα αναλυθεί στη συνέχεια και περιέχει τα επόμενα βήματα.

I. Τοποθέτηση όλων των αρχείων που αποτελούν την εφαρμογή σε ένα jar αρχείο
II. Υπογραφή του jar αρχείου
III. Εξαγωγή του πιστοποιητικού δημοσίου κλειδιού που αντιστοιχεί στο ιδιωτικό κλειδί που χρησιμοποιήθηκε, για να υπογραφεί το jar αρχείο.

Το πρώτο βήμα της διαδικασίας αυτής, λοιπόν, είναι η δημιουργία από όλα τα αρχεία του προγράμματος ενός και μόνο jar αρχείου, στο οποίο και θα γίνει η διαδικασία πιστοποίησης. Τα αρχεία αυτά βέβαια δεν περιορίζονται στις κλάσεις του προγράμματος (*.class), αλλά περιέχουν κάθε εικόνα που πιθανώς περιέχει το applet (images), στην περίπτωσή μας αρχεία για τη δημιουργία του συστήματος βοήθειας (help/helpfiles) καθώς επίσης και πακέτα που δημιουργήθηκαν για τις ανάγκες του προγράμματος (help/*.class και gui/*.class). Αυτό μπορεί εύκολα να γίνει με μία εντολή του java compiler όπως η παρακάτω:

*c:\j2sdk1.4.2_05\bin\jar cvf simmate.jar *.class help/*.class gui/*.class images help/helpfiles*

Στη συνέχεια θα πρέπει να υπογραφεί το αρχείο simmate.jar που μόλις δημιουργήθηκε. Για να γίνει όμως αυτό, πρώτα θα πρέπει να δημιουργηθεί ένα ιδιωτικό κλειδί μαζί με το αντίστοιχο δημόσιό του. Με την εντολή που θα ακολουθήσει, θα δημιουργηθεί μία βάση (keystore), στην οποία θα γίνει μία καταχώρηση με ένα νέο ζευγάρι δημόσιου/ιδιωτικού κλειδιού.

c:\j2sdk1.4.2_05\bin\keytool -genkey -alias signFiles -keystore simmate -keypass karditsa -storepass leros1

Πρέπει να σημειωθεί ότι η εντολή που θα εκτελεστεί στο παράθυρο εντολών, όπως και κάθε εντολή γραμμής εντολών, πρέπει να καταλαμβάνει μία και μόνο γραμμή. Η εντολή –genkey δημιουργεί κλειδιά, η -alias signFiles δηλώνει το ψευδώνυμο, με το οποίο πλέον θα αναφέρεται η καταχώρηση με το ζεύγος κλειδιών. Το μέρος της εντολής -keystore simmate καθορίζει το όνομα της βάσης που χρησιμοποιείται. Το μέρος -keypass karditsa καθορίζει ένα κωδικό (karditsa)

για το ιδιωτικό κλειδί που θα δημιουργηθεί, κωδικός που θα είναι απαραίτητος για να υπάρχει πρόσβαση στη καταχώρηση με το κλειδί αυτό. Τέλος το -storepass leros καθορίζει ένα κωδικό (leros1) για τη βάση. Οι κωδικοί αυτοί θα πρέπει να είναι τουλάχιστον έξι χαρακτήρων.

Με την εκτέλεση της εντολής αυτής, θα ακολουθήσει μία σειρά ερωτήσεων για προσωπικά στοιχεία του χρήστη, όπως όνομα, οργανισμός, πόλη κ.λπ. Σε περίπτωση που δε θέλουμε να ερωτηθούμε, αλλά να δώσουμε σαν προσωπικό στοιχείο το όνομα antonis, για παράδειγμα, στην παραπάνω εντολή, θα πρέπει να προσθέσουμε το -dname "cn=antonis". Φυσικά, οι λεπτομέρειες που θα είναι ορατές στο χρήστη, θα είναι ανάλογες με αυτές που δόθηκαν στο βήμα αυτό.

Στο σημείο αυτό μπορεί πλέον το αρχείο να υπογραφεί με τη χρήση της εντολής

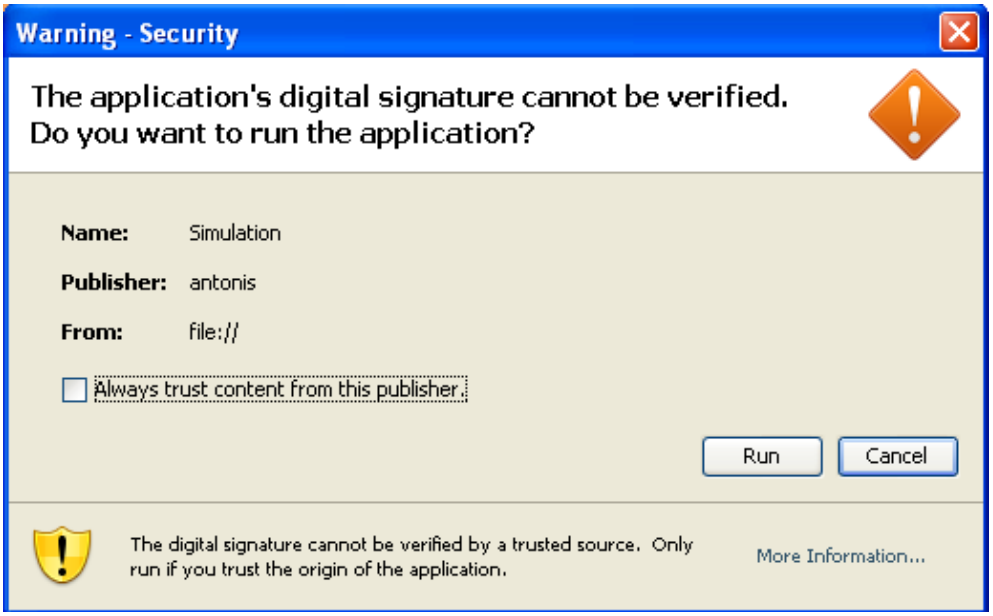
c:\j2sdk1.4.2_05\bin\jarsigner -keystore simmate -storepass leros1 -keypass karditsa -signedjar ssimmate.jar simmate.jar signFiles

Το υπογεγραμμένο αρχείο που προκύπτει από το αρχικό simmate.jar, θα λέγεται ssimmate.jar, όπως προκύπτει από την εντολή.

Το τρίτο και τελευταίο βήμα της διαδικασίας θα είναι να εξαχθεί το δημόσιο κλειδί. Οποιοσδήποτε χρήστης θέλει να έχει πρόσβαση στον κώδικα της εφαρμογής, θα πρέπει να έχει πρόσβαση στο δημόσιο κλειδί, το οποίο πρέπει να εξαχθεί και να σταλεί ένα αντίγραφο στο κάθε χρήστη. Το αρχείο που περιέχει πλέον το κλειδί και θα πρέπει να είναι προσβάσιμο στην ιστοσελίδα, είναι το simmate.cer.

c:\j2sdk1.4.2_05\bin\keytool -export -keystore simmate -storepass leros1 -alias signFiles -file simmate.cer

Ο εξυπηρετητής πλέον θα πρέπει να περιέχει τα αρχεία *ssimmate.jar, SSimmate.html, jdom.jar.* Σε περίπτωση που επιθυμείται η παράθεση έτοιμων παραδειγμάτων στον ίδιο εξυπηρετητή, θα πρέπει να περιέχονται τα xml αρχεία των παραδειγμάτων καθώς και κάποιος σύνδεσμος σε αυτά για εύκολη πρόσβαση. Ο χρήστης θα πρέπει να τα κατεβάσει και στη συνέχεια να τα ανοίξει τοπικά από την εφαρμογή του. Υπάρχει και η εναλλακτική να δημιουργηθούν νέα html αρχεία στα οποία όπως συζητήθηκε θα περνάν σαν παράμετροι τα ονόματα των παραπάνω αρχείων. Έτσι δίνοντας σαν διεύθυνση τα ονόματα αυτών των html αρχείων ο χρήστης θα μπορεί απευθείας να φορτώσει την εφαρμογή ταυτόχρονα με το παράδειγμα. Αυτό που απομένει, είναι να εμπιστευτεί ο χρήστης που, όταν προσπαθήσει να επισκεφτεί την ιστοσελίδα, θα αντιμετωπίσει ένα παράθυρο σαν αυτό που ακολουθεί.



Από τη στιγμή που θα γίνει αυτό, ο χρήστης πλέον μπορεί να αποθηκεύει μοντέλα που έχει φτιάξει ο ίδιος τοπικά στον υπολογιστή του και στη συνέχεια να τα ανακτά. Μπορεί, επίσης, να αποθηκεύει αρχεία έτοιμων μοντέλων που βρίσκονται στην ιστοσελίδα και να πειραματιστεί με αυτά. Υπάρχει επιπλέον η δυνατότητα το αρχείο να ανοίγει απευθείας από ιστοσελίδες, στις οποίες έχει περαστεί η διαδρομή του αρχείου σαν παράμετρος στο applet. Ένα από τα μοντέλα αυτά, το οποίο μπορεί να ανοίξει με αυτό τον τρόπο, αλλά υπάρχει και σε μορφή xml, εξηγείται και στο τρίτο κεφάλαιο του παρόντος. Ο τρόπος, με τον οποίο αποθηκεύονται τα μοντέλα αυτά στο αρχείο, εξηγείται στην επόμενη παράγραφο.

2.6.1 Μορφή Αποθήκευσης Αρχείων

Στα τέλη του 1990 εκδόθηκε μία γλώσσα, η XML (eXtensible Markup Language) υπό την κηδεμονία του οργανισμού World Wide Web Consortium [http://www.w3.org], με σκοπό να παρέχει έναν εύκολο στη χρήση και τυποποιημένο τρόπο αποθήκευσης αυτοπεριγραφόμενων δεδομένων. Παρόλο που επί της ουσίας η γλώσσα αυτή δεν προσφέρει τίποτα, μπορεί να αποτελέσει ένα τυποποιημένο κοινό σημείο. Το κύριο πλεονέκτημα αυτής της γλώσσας, δηλαδή, είναι ότι μπορεί να αποτελέσει όχημα ανταλλαγής πληροφοριών μεταξύ διαφορετικών προγραμμάτων. Τα δεδομένα από μία εφαρμογή, μία διαδικτυακή για παράδειγμα βάση MySQL, μπορούν να μετατραπούν σε XML και στη συνέχεια να μεταφερθούν σε ένα άλλο οποιοδήποτε πρόγραμμα, το οποίο θα μετατρέψει το XML αρχείο σε κάτι που με τη σειρά του θα γίνεται κατανοητό από το ίδιο. Η μεταφερσιμότητα αυτή, σε συνδυασμό με τον απλό και ευανάγνωστο τρόπο παρουσίασης, οδήγησε στην επιλογή της χρήσης αυτής της μορφής αρχείων στο πρόγραμμα simmate.

Στη συνέχεια έπρεπε να βρεθεί ένας τρόπος εύκολης διαχείρισης, μέσω της γλώσσας java που χρησιμοποιήθηκε, τέτοιου είδους αρχείων. Ο τρόπος που επιλέχθηκε είναι το JDOM API [http://www.jdom.org/docs/apidocs/index.html] και οι κλάσεις που χρησιμοποιούν τις μεθόδους του είναι οι FileBuilder για δημιουργία του αρχείου από το μοντέλο και FileReader για διάβασμα του αρχείου και δημιουργία του μοντέλου. Η συνάρτηση writeFile() της πρώτης δημιουργεί ένα αρχείο που έχει την μορφή που φαίνεται στη συνέχεια.

<example>

<info></info>

<objects></objects>

<connections></connections>

</example>

Οι ετικέτα *<example>* περικλείει όλο το μοντέλο που στη συνέχεια χωρίζεται σε τρία κύρια μέρη. Οι γενικές πληροφορίες της προσομοίωσης περιέχονται στο πρώτο μέρος υπό την ετικέτα *<info>*.

<info>

<title>Super Size Me</title>

<comments>

A fast food example with a teller and a roaster. It is a multiple phases system since the customer has to wait to take the order from the teller and afterwards the roaster for the order to be proceeded.

</comments>

<appearance></appearance>

<simulation></simulation>

</info>

Οι πληροφορίες αυτές περιέχουν τον τίτλο του παραδείγματος καθώς και τυχόν σχόλια του δημιουργού του μοντέλου. Περιέχονται επίσης στοιχεία που αφορούν στην εμφάνιση, *<appearance>*, όπως την εστίαση και το φόντο. Τέλος περιλαμβάνονται στοιχεία που αφορούν την προσομοίωση, *<simulation>*, και την κίνηση, τις συνεχόμενες προσομοιώσεις και αυτές που θα κρατηθούν και τον χρόνο ή τον πελάτη τερματισμού. Γενικώς περιλαμβάνονται επιλογές του χρήστη από το γενικό πλαίσιο προσομοίωσης.

Το δεύτερο κύριο μέρος αναφέρεται στα αντικείμενα του μοντέλου και περικλείεται από την ετικέτα *<objects></objects>*. Κάθε νέο αντικείμενο περικλείεται με τη σειρά του από την *<object></object>* και περιέχονται όλες οι πληροφορίες που αφορούν το συγκεκριμένο αντικείμενο όπως ο τύπος του, το όνομά του και η θέση του. Το τελευταίο μέρος του αρχείου αφορά τους συνδέσμους, ορίζεται από την ετικέτα *<connections></connections>* και την ετικέτα *<connection></connection>* για καθένα από αυτούς. Περιέχει οποιαδήποτε πληροφορία τους αφορά, όπως τον τύπο τους και τα αντικείμενα τα οποία συνδέει. Για την ακρίβεια περιέχει την θέση των αντικειμένων στην λίστα *objectList* της *AreaDrawing*. Η *readFile()* συνάρτηση της *FileReader* κλάσης διαβάζει τον παραπάνω τύπο αρχείων και εξάγει από τα αρχεία αυτά το αρχικό μοντέλο.

3. Επίλογος

3.1 Χρήση της Εφαρμογής

Στη συνέχεια θα παρουσιαστούν κάποια παραδείγματα χρήσης της εφαρμογής. Πριν γίνει όμως αυτό θα πρέπει να παρουσιαστεί ο τρόπος εκκίνησής της αλλά και ο τρόπος ανάκτησης έτοιμων αρχείων και αποθήκευσης αυτών. Είτε τοπικά στον υπολογιστή είτε σε κάποιον εξυπηρετητή διαδικτύου θα πρέπει στον ίδιο φάκελο να υπάρχουν τα αρχεία *ssimmate.jar*, *jdom.jar* και *SSimmate.html*. Τυχόν παραδείγματα μπορούν να βρίσκονται στον εξυπηρετητή αλλά για να ανοιχτούν από την εφαρμογή θα πρέπει να αποθηκευτούν τοπικά. Όλα τα σωσμένα αρχεία της εφαρμογής είναι της μορφής <όνομα>.xml και καλό θα ήταν να υπάρχει ένας σύνδεσμος για αυτά στην html σελίδα που καλεί το applet. Στην περίπτωσή μας η σελίδα είναι η SSimmate.html η οποία περιέχει συνδέσμους για το fast_food_nation.xml και karditsa_dental_clinic.xml. Η μόνη ενέργεια αφού αποθηκευτούν τα αρχεία αυτά πρέπει να είναι το άνοιγμα της html σελίδας με οποιονδήποτε φυλλομετρητή.

3.2 Παραδείγματα

Στο σημείο αυτό παρουσιάζονται παραδείγματα χρήσης της εφαρμογής. Αρχικά περιγράφεται το υποτιθέμενο σύστημα συνέχεια επιλύεται το μοντέλο με όρους της συγκεκριμένης εφαρμογής. Τα μοντέλα αυτά υπάρχουν σωσμένα στο συνοδευτι δίσκο αλλά και στον εξυπηρετητή διαδικτύου που υπάρχει η εφαρμογή. Το πρώτο σύστημα είναι μία οδοντιατρική κλιν Καρδίτσα και η λύση της υπάρχει πιο αναλυτικά στην τεκμηρίωση στο παράρτημα του παρόντος αλλά και στην ίδια την ε Σκοπός του συγκεκριμένου κεφαλαίου όμως δεν είναι η εκμάθηση της διεπαφής της εφαρμογής και για το λόγο αυτό η παρουσιάζεται πιο αφαιρετικά. Το δεύτερο σύστημα είναι το νεοϋδριθέν ταχυφαγείο Fast Food Nation στην Αγία Μαρίνα της Λ το τρίτο μία πιο σύνθετη παρουσίασή του.

3.2.1 Οδοντιατρική κλινική Καρδίτσας

3.2.1.1 Το σύστημα

Το Υπουργείο Υγείας μελετά την περίπτωση δημιουργίας μίας οδοντιατρικής κλινικής στην Καρδίτσα. Επιθυμείται κυρίως η προσέγγιση ενός αριθμού προσλήψεων προσωπικού που περιλαμβάνει αναισθησιολόγους, νοσοκόμες και οδοντιάτρους. Προς μελέτη τίθεται και το κτιριακό, ο αριθμός χώρων υποδοχής καθώς και η χωρητικότητά τους. Γίνεται έτσι μία αρχική σχεδίαση, της οποίας η περιγραφή παρουσιάζεται στη συνέχεια. Το υπό περιγραφή σύστημα θα πρέπει να μελετηθεί και να παρθούν αποφάσεις για τυχόν επιπλέον ή λιγότερες προσλήψεις και πιο ευρύχωρους ή όχι χώρους.

Αρχικά, η πελατεία της οδοντιατρικής κλινικής μπορεί να θεωρηθεί άπειρη σε σχέση με τους πελάτες που βρίσκονται ήδη στην κλινική. Δε θα διατίθεται κάποιο σύστημα κράτησης ραντεβού και ο στοχαστικός τρόπος έλευσης πελατών μπορεί να περιγραφεί σαν εκθετική προσομοίωση με μέσο όρο 8 λεπτά μεταξύ δύο πελατών. Η κλινική θα διαθέτει ένα χώρο αναμονής, στον οποίο οι πελάτες περιμένουν μία νοσοκόμα, η οποία μέσα σε ένα έως τρία λεπτά, ομοιόμορφα κατανεμημένα, χωρίζει τους πελάτες – ασθενείς σε αυτούς που χρειάζονται αναισθησία πριν δουν τον γιατρό και σε αυτούς που δε χρειάζονται. Ο χώρος αυτός θα έχει χωρητικότητα έως δέκα πελατών και, όπως σε κάθε χώρο

αναμονής, στην κλινική οι πελάτες θα εξυπηρετούνται στο πρότυπο *first come – first served*. Έχει υπολογιστεί από μελέτες ότι εξήντα από τις εκατό περιπτώσεις των πελατών μίας τέτοιας κλινικής απαιτούν αναισθησία, ενώ σαράντα όχι. Τα δύο λεπτά που χρειάζεται η νοσοκόμα για να κατηγοριοποιήσει τους πελάτες, εκλαμβάνονται από αυτόν σαν χρόνος αναμονής και όχι σαν χρόνος εξυπηρέτησης.

Η πρώτη ομάδα πελατών θα πρέπει να οδηγηθεί σε ένα δεύτερο χώρο αναμονής, για τον αναισθησιολόγο. Ο συγκεκριμένος χώρος αναμονής έχει χωρητικότητα δέκα ατόμων. Το χώρο αυτό ακολουθεί το δωμάτιο των αναισθησιολόγων, για τους οποίους υπάρχει πρόβλεψη δύο προσλήψεων. Η προσέγγιση που έχει γίνει γι' αυτούς είναι ότι για κάθε αναισθησία απαιτείται από τον καθένα χρόνος που μπορεί να περιγραφεί σαν μία τριγωνική κατανομή με ελάχιστο χρόνο τα επτά λεπτά, μέγιστο τα δέκα και πιο πιθανό χρόνο τα οκτώ λεπτά. Στην περίπτωση που και οι δύο αναισθησιολόγοι είναι ανενεργοί, μπορεί να θεωρηθεί ότι πάντα ο πρώτος είναι αυτός που θα αναλαμβάνει την εργασία. Όταν ο πελάτης τελειώνει αυτή τη διαδικασία, κατευθύνεται στον επόμενο χώρο, όπου οδηγείται και το υπόλοιπο σαράντα τοις εκατό των περιπτώσεων που δεν απαιτούν αναισθησία. Αυτός ο χώρος, που είναι και ο χώρος αναμονής για τον οδοντίατρο, μπορεί να δεχτεί μέχρι είκοσι άτομα. Οι υπό πρόσληψη οδοντίατροι είναι πεπειραμένοι οδοντίατροι που εξυπηρετούν τους πελάτες τους με έναν τρόπο που μπορεί να προσεγγιστεί με μία εκθετική κατανομή με μέσο όρο επτά λεπτά. Μετά και από τον οδοντίατρο, κάθε πελάτης οδηγείται προς την έξοδο.

Συνολικά μία εργάσιμη μέρα μπορεί να κρατήσει δεκαέξι ώρες, για τις οποίες υπάρχουν δύο ομάδες όμοιες των παραπάνω υπαλλήλων.

Στοιχεία που θα πρέπει να εξεταστούν στο σύστημα είναι τυχόν χαμένοι πελάτες, χρόνος αναμονής αλλά και εξυπηρέτησης πελατών, καθώς και πώς διαμοιράζεται ο χρόνος αυτός στους χώρους αναμονής και στους εξυπηρετητές. Αποφάσεις που μπορούν να ληφθούν είναι τυχόν επιπλέον ή λιγότερες προσλήψεις νοσοκόμας, οδοντιάτρου ή αναισθησιολόγου. Για να γίνει αυτό, θα πρέπει να δημιουργηθεί ένα μοντέλο, το οποίο θα προσομοιώνει το υπό μελέτη σύστημα, και το οποίο θα μπορεί να δοκιμαστεί πολλές φορές με διάφορες τιμές. Στην συνέχεια παρουσιάζεται ο τρόπος δημιουργίας του μοντέλου που όμως υπάρχει και έτοιμο στο αρχείο *karditsa_dental_clinic.xml*.

3.2.1.2 Το μοντέλο

Ο τρόπος δημιουργίας του μοντέλου παρουσιάζεται χωρίς λεπτομέρειες για τον τρόπο χειρισμού της διεπαφής. Για περισσότερες λεπτομέρειες γύρω από αυτό, μπορείτε να ανατρέξετε είτε στο παράρτημα του παρόντος, είτε στην τεκμηρίωση του προγράμματος.

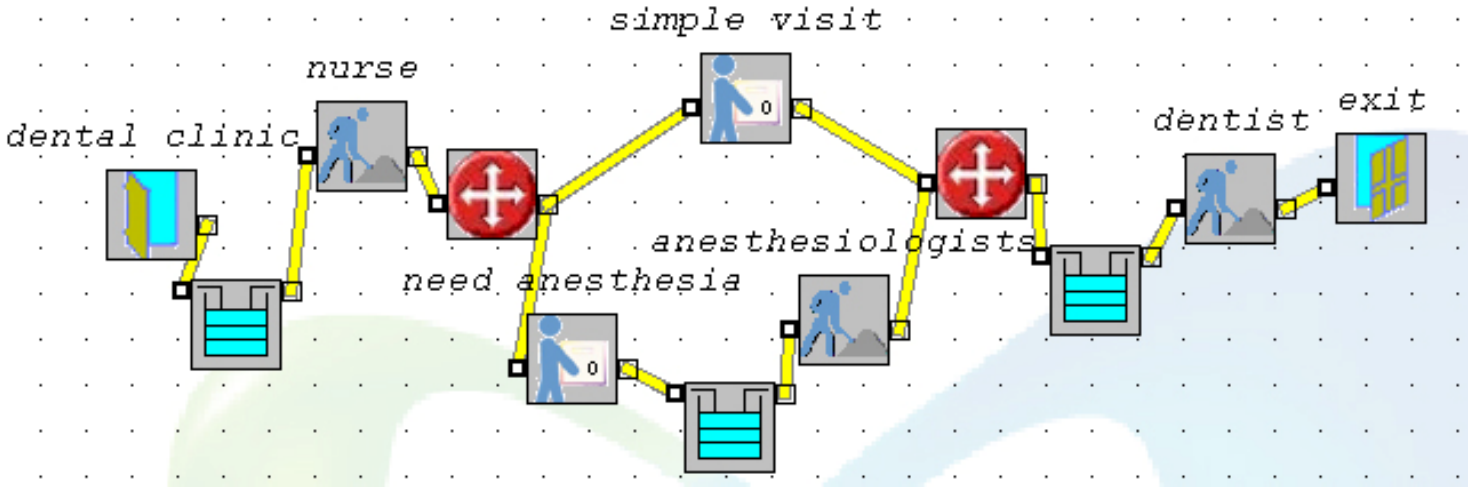
Όπως έχει περιγραφεί, ο τρόπος εισόδου των ασθενών στο σύστημα μπορεί να προσεγγιστεί με μία γεννήτρια, η οποία θα ακολουθεί την εκθετική κατανομή με μέση τιμή τα οκτώ λεπτά. Μπορούμε επιπλέον να θεωρήσουμε ότι οι πελάτες που μπορούν να εισέλθουν στο σύστημα είναι άπειροι, κάτι που θα τεθεί και στη γεννήτρια. Για τη δημιουργία του πρώτου χώρου, αλλά και κάθε χώρου αναμονής, θα δημιουργηθεί μία ουρά η οποία θα έχει μέγιστο μήκος 10. Η είσοδός της θα είναι συνδεδεμένη με την έξοδο της γεννήτριας και η έξοδός της με τη νοσοκόμα που θα μοντελοποιείται από έναν εξυπηρετητή. Ο εξυπηρετητής αυτός θα ακολουθεί ομοιόμορφη κατανομή στο διάστημα (1,3), θα μπορεί να εξυπηρετήσει άπειρους πελάτες (Unlimited Items) και θα έχει την λειτουργία καθυστέρησης (delay).

Απαραίτητη κρίνεται στη συνέχεια η προσθήκη ενός δρομολογητή, ο οποίος, εφόσον η νοσοκόμα πρέπει να κατηγοριοποιεί τους ασθενείς σε αυτούς που χρειάζονται αναισθησιολόγο, που είναι εξήντα τοις εκατό, και σε αυτούς που δεν χρειάζονται, το υπόλοιπο σαράντα θα πρέπει να ακολουθεί την weighted δρομολόγηση. Στην επίλυση του μοντέλου χρησιμοποιήθηκαν και δύο μετρητές, συνδεδεμένοι με τον δρομολογητή, ο simple visit και ο need anesthesia, καθώς κρίθηκε σημαντική η άμεση ένδειξη των μεγεθών αυτών. Η weighted δρομολόγηση θα στέλνει κατά 40% τον ασθενή στον πρώτο μετρητή και κατά 40% στον δεύτερο.

Το δεύτερο μετρητή ακολουθεί μία ουρά που αντιπροσωπεύει το χώρο αναμονής για τους αναισθησιολόγους, που έχει μέγιστο μήκος 10, και φυσικά, όπως και κάθε ουρά στο συγκεκριμένο μοντέλο, έχει πειθαρχεία FIFO. Ακολουθεί το δωμάτιο των αναισθησιολόγων, συνδεδεμένο με την ουρά, που έχει 2 αναισθησιολόγους που δουλεύουν ταυτόχρονα (Simultaneously serviced =2), οι οποίοι με τη σειρά τους μπορούν να εξυπηρετήσουν άπειρους ασθενείς (Unlimited Items) και ακολουθούν τριγωνική κατανομή με ελάχιστη τιμή τα 7 λεπτά, μέγιστη τιμή τα 10 και πιο πιθανή τιμή τα οκτώ λεπτά.

Η έξοδος του εξυπηρετητή, καθώς και η έξοδος του μετρητή των ασθενών που δεν περνούν από αναισθησιολόγο, θα πρέπει να καταλήξουν στην ίδια ουρά, που θα αντιπροσωπεύει τον χώρο αναμονής για τον οδοντίατρο. Ενδιάμεσα, λοιπόν, θα απαιτηθεί ένας δρομολογητής. Η ουρά πριν τον οδοντίατρο θα πρέπει να έχει μέγιστο μήκος 20, όση η χωρητικότητα του συγκεκριμένου χώρου. Η ουρά αυτή συνδέεται με τον εξυπηρετητή που αντιπροσωπεύει τον οδοντίατρο και ακολουθεί εκθετική κατανομή με μέσο χρόνο τα επτά λεπτά. Μπορεί, επιπλέον, ελλείψει περιορισμού, να εξυπηρετήσει άπειρους πελάτες και ακολουθείται από την έξοδο του οδοντιατρείου και του μοντέλου γενικότερα.

Η διάρκεια κατά την οποία μένει ανοιχτό το οδοντιατρείο είναι 16 ώρες που μεταφράζεται σε 960 λεπτά, κάτι που θα πρέπει να οριστεί στο κατάλληλο πεδίο *Stop Simulation at*, στο πλαίσιο γενικών επιλογών της προσομοίωσης. Επιπρόσθετα, στο πλαίσιο αυτό μπορούν να γίνουν και επιπλέον ρυθμίσεις, όπως για παράδειγμα η επιτάχυνση της κίνησης ή ακόμα και η απενεργοποίησή της και ο αριθμός των συνεχόμενων προσομοιώσεων και αυτών που θα κρατούνται και στο λυμένο παράδειγμα έχουν τεθεί στο 1 και 5 ταυτόχρονα για να αντιπροσωπεύεται μία εργάσιμη εβδομάδα. Ακολουθεί η εμφάνιση του μοντέλου στην εφαρμογή. Το αρχείο με το μοντέλο αυτό βρίσκεται στον ίδιο φάκελο με τα υπόλοιπα αρχεία της εφαρμογής με το όνομα *karditsa_dental_clinic.xml*



3.2.2 Fast Food Nation

3.2.2.1 Το σύστημα

Ένας νεαρός επιχειρηματίας στη Λέρο σκέφτεται να ανοίξει μία μικρή επιχείρηση ταχυφαγείου στην Αγία Μαρίνα. Η σκέψη του είναι, τουλάχιστον αρχικά, να προσλάβει έναν περιορισμένο αριθμό υπαλλήλων. Περιορίζεται στην σκέψη πρόσληψης ενός ταμία, δύο ψηστών και ενός καφετζή. Στο ξεκίνημα της μικρής του επιχείρησης επιθυμεί να διαθέτει μόνο συγκεκριμένα προϊόντα όπως φραπέδες, αναψυκτικά και ένα είδος burger. Μελέτες του από ανταγωνιστικό κατάστημα στο Λακκί της Λέρου που είναι οικισμός παρόμοιος σε πληθυσμό και συμπεριφορές του κόσμου, δείχνουν ότι πελάτες αναμένονται να έρχονται με έναν τρόπο που μπορεί να αναπαρασταθεί με μία εκθετική κατανομή με μέσο χρόνο πέντε λεπτών ανά πελάτη. Ο πληθυσμός της Αγίας Μαρίνας αλλά και των γύρω περιοχών επιτρέπει τον μελετητή του συστήματος να θεωρήσει τους πιθανούς πελάτες άπειρους σε σχέση με τους πελάτες που είναι ήδη στο σύστημα.

Η γενικότερη πολιτική εξυπηρέτησης που έχει αποφασιστεί είναι ο ταμίας να παίρνει τις παραγγελίες και να στέλνει τον πελάτη στον κατάλληλο υπάλληλο ανάλογα με αυτήν. Παίρνει τις παραγγελίες με στοχαστικό τρόπο που μπορεί να περιγραφεί σαν μία ομοιόμορφη κατανομή από 0.5 έως 1 λεπτό ανά πελάτη. Αναλύσεις που έχουν γίνει στη συμπεριφορά των πελατών στο νησί, δείχνουν ότι συνηθισμένοι σε άδειες από κόσμο υπηρεσίες, εάν η ουρά μπροστά από το ταμείο είναι πάνω από τρία άτομα, οι πελάτες δεν εισέρχονται στο κατάστημα και θεωρούνται χαμένοι. Οι παραγγελίες αρχικά μπορούν να χωριστούν σε δύο ειδών, φαγητού και καφέ ή αναψυκτικών Όταν αφορούν καφέ ή αναψυκτικό οι παραγγελίες πρέπει να εξυπηρετηθούν από τον καφετζή, ο οποίος, όταν η παραγγελία είναι για καφέ, χρειάζεται, σύμφωνα με τους υπολογισμούς του επιχειρηματία, δύο με τέσσερα λεπτά ομοιόμορφα κατανεμημένα, και, όταν είναι για αναψυκτικό, ένα με δύο λεπτά, επίσης ομοιόμορφα κατανεμημένα. Μπροστά από τον εξυπηρετητή οι ανυπόμονοι Λεριοί έχει υπολογιστεί ότι, παρόλο που έχουν ήδη δώσει την παραγγελία τους, εάν δουν ουρά μεγαλύτερη των πέντε πελατών, θα φύγουν από το κατάστημα. Το σαράντα τοις εκατό των συνολικών πελατών στο μαγαζί υπολογίζεται ότι θα έρχονται για τέτοιου είδους παραγγελίες και το τριάντα τοις εκατό από αυτούς θέλουν απλά αναψυκτικό. Το υπόλοιπο εξήντα τοις εκατό αφορά φαγητό και εξυπηρετείται από τους ψήστες.

Υπάρχουν δύο διαθέσιμοι ψήστες, σύμφωνα με τους υπολογισμούς του επιχειρηματία, που εξυπηρετούν τέτοιες παραγγελίες, ο ένας εκ των οποίων είναι πιο πεπειραμένος και γρήγορος από τον άλλο. Και των δύο οι χρόνοι εξυπηρέτησης μπορούν να θεωρηθούν σαν εκθετικοί με τον ένα να έχει μέσο χρόνο εξυπηρέτησης τα πέντε λεπτά και το δεύτερο τα επτά λεπτά. Ο τρόπος που δέχονται τους πελάτες μπορεί να χαρακτηριστεί σαν round robin, καθώς δέχεται τον πρώτο πελάτη ο πρώτος ψήστης, ο λιγότερο πεπειραμένος, τον δεύτερο ο πεπειραμένος ψήστης κ.ο.κ. Εάν όμως κάποιος από τους δύο έχει σειρά, αλλά είναι απασχολημένος, τότε την παραγγελία θα πρέπει να την πάρει ο άλλος. Η ουρά που μπορεί να γίνει ανεκτή από τον Λεριό μπορεί να υποτεθεί στους δεκαπέντε πελάτες, καθώς η συγκεκριμένη ουρά αφορά φαγητό και τα πράγματα σοβαρεύουν. Τέλος, γίνονται σκέψεις το κατάστημα να διατηρείται ανοιχτό από τις δώδεκα το μεσημέρι μέχρι τις έξι το απόγευμα.

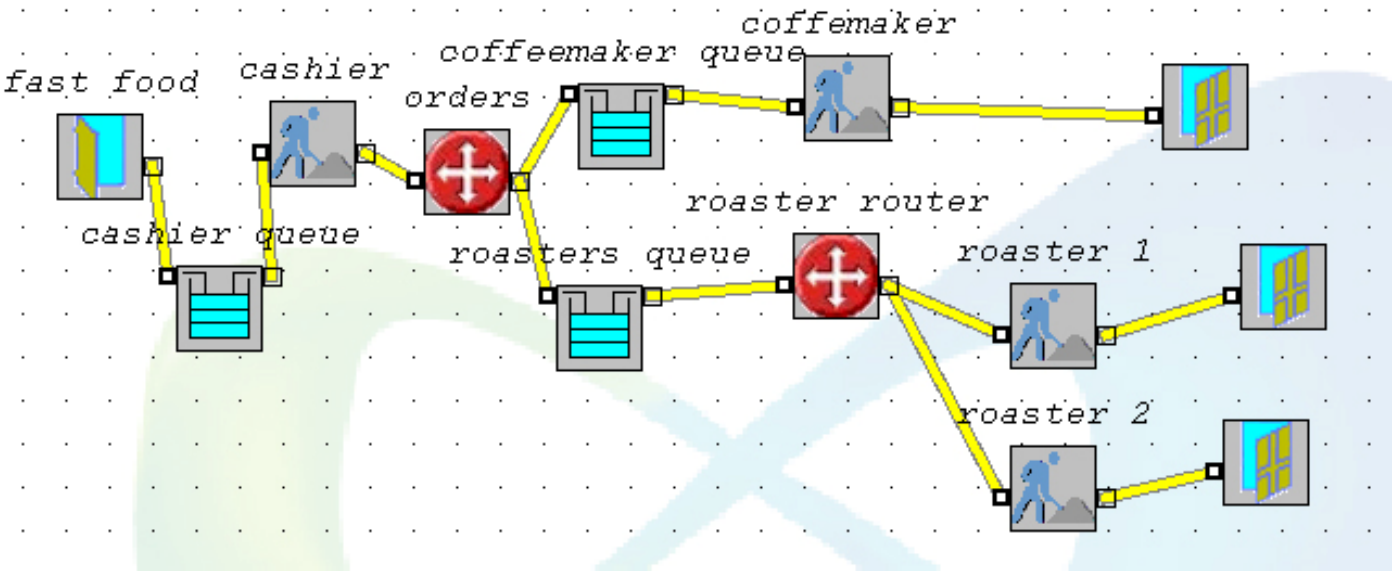
3.2.2.2 Το μοντέλο

Η άφιξη πελατών μπορεί να αναπαρασταθεί με μια γεννήτρια που ακολουθεί την εκθετική κατανομή με μέσο χρόνο τα 5 λεπτά. Η γεννήτρια αυτή θα συνδέεται με μια ουρά χωρητικότητας 3 πελατών με πολιτική first come – first served που θα ακολουθείται από τον ταμία. Ο ταμίας θα αναπαρασταθεί από έναν εξυπηρετητή ο οποίος θα μπορεί να εξυπηρετήσει

άπειρους πελάτες και θα ακολουθεί την ομοιόμορφη κατανομή από 0,5 έως 1 λεπτό. Οι πελάτες θα πρέπει στη συνέχεια να χωρίζονται σε αυτούς που επιθυμούν φαγητό και αυτούς που επιθυμούν καφέ ή αναψυκτικό. Ένας δρομολογητής που χρησιμοποιεί τον weighted αλγόριθμο δρομολόγησης συνδεδεμένος με τον ταμία μπορεί να εκτελέσει αυτή την εργασία. Με πιθανότητα 0.4 θα δρομολογεί τους πελάτες σε μία ουρά για τον καφετζή και με πιθανότητα 0.6 σε μία ουρά αναμονής πριν τους ψήστες. Οι ουρές αυτές θα πρέπει να έχουν χωρητικότητα 5 και 15 πελατών αντίστοιχα.

Για το πρώτο 40 τις 100 και σύμφωνα με την υπόθεση, κατά 30 τις εκατό οι πελάτες της κατηγορίας που περιμένουν στην ουρά για τον καφετζή επιθυμούν αναψυκτικό που απαιτεί από 1 έως 2 λεπτά χρόνου ομοιόμορφα κατανεμημένου. Το υπόλοιπο 70 τις εκατό επιθυμεί καφέ που απαιτεί 2 έως 4 λεπτά χρόνου από τον καφετζή. Η συμπεριφορά αυτή μπορεί να προσομοιωθεί από μία εμπειρική κατανομή στην οποία θα υπάρχουν δύο σκέλη, το πρώτο από 1 έως 2 λεπτά με πιθανότητα 0.3 και το δεύτερο από 2 έως 4 λεπτά με πιθανότητα 70 τις εκατό. Μετά το σερβίρισμα του καφέ ο πελάτης θεωρούμε ότι εξέρχεται του προς μελέτη συστήματος. Συνδέουμε έτσι τον εξυπηρετητή του καφετζή με ένα αντικείμενο εξόδου.

Το υπόλοιπο 60 τις εκατό θα πρέπει να δρομολογηθεί προς τους ψήστες οι οποίοι όμως λειτουργούν με τη σειρά και μάλιστα με πρώτον τον πιο αργό ψήστη εκτός και εάν ο ένας από τους δύο είναι απασχολημένος πράγμα που σημαίνει ότι θα αναλάβει ο άλλος. Για να υλοποιηθεί η συμπεριφορά αυτή απαιτείται ανάμεσα στην ουρά αναμονής και τους ψήστες ένας δρομολογητής που χρησιμοποιεί τον αλγόριθμο round robin ο οποίος θα ξεκινά από τον αργό ψήστη και θα αγνοεί τον τυχόν απασχολημένο (Ignore Full). Και οι δύο ψήστες ακολουθούν εκθετική κατανομή, ο γρήγορος με μέσο χρόνο 5 και ο αργός με μέσο χρόνο 7 λεπτά. Τους ψήστες ακολουθούν οι έξοδοι καθώς φεύγουν από το προς μελέτη σύστημα. Οι ώρες που παραμένει ανοιχτό το κατάστημα είναι 6 που μεταφράζονται σε 360 λεπτά πραγματικού χρόνου που τίθενται στο πεδίο *Stop Simulation at*.



3.3 Συμπεράσματα - Μελλοντική εργασία

Η εργασία αυτή είναι μία προσπάθεια βελτίωσης της διδασκαλίας σε περιβάλλον διαδικτύου. Για την προσπάθεια αυτή, δημιουργήθηκε το εργαλείο simmate, ένα εργαλείο προσομοίωσης για συστήματα ουρών, με εύκολη μελλοντική επέκταση και σε άλλους τομείς. Η προσομοίωση αποτελεί ένα σημαντικότατο τομέα της επιχειρησιακής έρευνας με πάρα

πολλές πρακτικές εφαρμογές σε μία ευρεία γκάμα πεδίων. Πεδία όπως η πληροφορική [Estrin, 1999], οι κατασκευές [Jägstam and Klingstam, 2002], ακόμα και οι κοινωνικές επιστήμες [Gilbert, 1995] ή η Ιατρική [http://vam.anest.ufl.edu/simulations/simulationportfolio.php]. Έχει δημιουργηθεί ακόμα και προσομοίωση της συμπεριφοράς μίας πόλης κατά τη διάρκεια κάποιου γεγονότος, όπως αυτή της Βενετίας κατά τη διάρκεια του Καρναβαλιού [Brambilla, 2005].

Έχουν αναπτυχθεί πολλές γλώσσες και πακέτα γλωσσών με αντικείμενο την ανάπτυξη και μελέτη μοντέλων συστημάτων προσομοίωσης διακριτών συστημάτων [Gariddo J.M., 2001], όπως το simpack, γραμμένο σε γλώσσα c και η προσθήκη σε c++ του Sim++. Επίσης, έχει αναπτυχθεί το Csim στην ίδια γλώσσα, αλλά και το modsim που μεταφράζει τη γλώσσα του modula-2 επίσης σε c. Η IBM ανέπτυξε τη δική της γλώσσα GPSS, η οποία, ούσα μη διαδικαστική, ήταν περιορισμένων δυνατοτήτων αν και επεκτάθηκε στις μετέπειτα εκδόσεις GPSS/V και GPSS/H. Μία επέκταση της algol για επίλυση προβλημάτων προσομοίωσης ήταν η simula, επέκταση της οποίας, ανεπτυγμένη από την Birtwistle, ήταν η Demos. Άλλα παρόμοια πακέτα είναι τα GASP, SLAM, την βιβλιοθήκη task library της c++ του Stroupstrup αλλά και της Psim και PsimL. Ακόμα και πακέτα για τη γλώσσα java έχουν αναπτυχθεί τελευταία όπως τα Javasim, JSIM, DESMO-J, Simjava και Silk, αλλά και η έκδοση της Psim σε java, η Psim-J.

Υπάρχουν στην αγορά πολλά εργαλεία όπως το extend [http://www.imaginethatinc.com] ή το simul8 [http://www.simul8.com] που, όπως είναι φυσικό, το εργαλείο αυτό δεν προσπαθεί να ανταγωνιστεί. Τα προγράμματα αυτά είναι από μόνα τους δύσκολα στη χρήση και πολύπλοκα εργαλεία και ίσως η εκμάθηση τους αποτελεί πολυτέλεια στα πλαίσια ενός μαθήματος προσομοίωσης. Οι σπουδαστές πρέπει μάλλον να ξοδεύουν περισσότερο χρόνο στη μελέτη της προσομοίωσης και όχι στο πως δουλεύει ένα συγκεκριμένο εργαλείο μίας συγκεκριμένης εταιρείας που, δεν πρέπει να ξεχνάμε, στόχος της είναι το κέρδος και άρα θα υπάρχει και η επιπλοκή των αδειών και του χρήματος.

Πρωταρχικό σκοπό στη δημιουργία του παρόντος εργαλείου αποτελεί η ευκολία στη χρήση, κάτι που ελπίζουμε να επιτεύχθηκε. Κάθε πιθανή απορία στη χρήση μπορεί να λυθεί με το σύστημα βοήθειας που παρατίθεται στο παράρτημα αλλά λειτουργεί και online στο ίδιο το πρόγραμμα. Ο χρήστης του προγράμματος δε χρειάζεται να προγραμματίσει, για να κατασκευάσει ένα μοντέλο, κάτι που βέβαια περιορίζει τις δυνατότητες του προγράμματος. Η προσοχή εστιάστηκε στην οπτικοποίηση κάποιων βασικών αρχών της προσομοίωσης.

Δυστυχώς, οι περιορισμοί χώρου και χρόνου που τίθενται, όταν ένα εργαλείο, όπως το παρόν, δημιουργείται στα πλαίσια μίας διπλωματικής εργασίας, δεν επιτρέπουν πιο ενδελεχή ανάπτυξη και ανάλυσή του. Συνακόλουθα, ενώ δόθηκε πολύ μεγάλη σημασία σε θέματα διεπαφής, αφιερώθηκε μικρότερο μέρος αυτής της εργασίας σε θέματα που αφορούν στην προσομοίωση, το κύριο δηλαδή κομμάτι του εργαλείου. Αυτό κρίθηκε όμως απαραίτητο, γιατί, χωρίς την ευκολία χειρισμού, θα χανόταν κατά τη γνώμη μας ο αρχικός στόχος της εργασίας. Με ένα πλήρες πακέτο, όσον αφορά στο χειρισμό, οποιαδήποτε μελλοντική εργασία θα πρέπει πλέον να επικεντρωθεί στο κομμάτι τις προσομοίωσης. Μία βελτίωση στα ήδη υπάρχοντα αντικείμενα θα ήταν ευπρόσδεκτη, αλλά αυτό που θα παρουσίαζε μεγάλο ενδιαφέρον είναι η προσθήκη και άλλων αντικειμένων, ώστε να επεκταθεί και η χρήση του σε άλλους τομείς της προσομοίωσης. Αυτό βέβαια σε καμία περίπτωση δε θα πρέπει να γίνει σε βάρος του πιο σημαντικού χαρακτηριστικού του παρόντος, της απλότητας.

Λόγω και του υποβάθρου, όμως του γράφοντος, ένα από τα αντικείμενα που υλοποιήθηκαν, θα ήταν θεμιτό να επεκταθεί. Το αντικείμενο του δρομολογητή, υποστηρίζει τον αλγόριθμο round robin, έναν τυχαίο και έναν αλγόριθμο με βάρη. Έγινε και μία προσπάθεια βέλτιστης δρομολόγησης, σύμφωνα με κάποια μέθοδο βαθμολόγησης κάθε πιθανού

αντικειμένου της διαδρομής. Η μέθοδος αυτή είναι πολύ απλοϊκή και σε σύνθετες περιπτώσεις είναι πολύ πιθανό να μην επιλεγεί η καλύτερη διαδρομή. Μελλοντικός στόχος της υλοποίησης της εφαρμογής, είναι, εκτός από την βελτίωση του συγκεκριμένου αλγορίθμου, η επέκταση του αντικειμένου, ώστε να υποστηρίζει διάφορους αλγορίθμους δρομολόγησης και η μελέτη αυτών, καθώς η εφαρμογή παρέχει ήδη οπτικοποίηση της διαδρομής κάθε πελάτη, κάτι που θα την διευκόλυνε.

Στην προσπάθεια να ολοκληρωθεί η εφαρμογή δόθηκε προτεραιότητα στην απλή επίλυση πολλών ζητημάτων που συναντήθηκαν στην πορεία, χωρίς να λαμβάνεται πάντα υπόψη η ταχύτητα του κάθε αλγορίθμου. Έτσι, ενώ στη περίπτωση της διωνυμικής κατανομής για παράδειγμα, κρίθηκε απαραίτητη η υλοποίηση της με τη χρήση ενός πιο αποδοτικού αλγορίθμου, όπως αυτός των Schmeister και Kachitvichyanukul, σε άλλες περιπτώσεις μελετήθηκαν αποδοτικές εναλλακτικές λύσεις, αλλά χρησιμοποιήθηκαν οι πρώτες προσεγγίσεις στο θέμα. Χαρακτηριστικό τέτοιο παράδειγμα, σημαντικότατο για την γενικότερη απόδοση της όλης εφαρμογής, η υλοποίηση του χειρισμού της λίστας επόμενου γεγονότος με τη διπλά συνδεδεμένη λίστα αντί για τις κατά πολύ προτιμότερες πολλαπλές λίστες, τον δυαδικό σωρό ή κάποια από τις πολλές υπάρχουσες προσεγγίσεις. Μία λοιπόν πιθανή μελλοντική έκδοση της παρούσης θα περιελάμβανε βελτιωμένες εκδόσεις πολλών από τους αλγορίθμους που ήδη χρησιμοποιεί.

Τα πλεονεκτήματα της χρήσης μίας γλώσσας όπως η XML, για το σύστημα αρχείων του προγράμματος, είναι δεδομένα. Με τα χρόνια, έχει καθιερωθεί σαν πρότυπο για κάθε πρόγραμμα και μοναδικός τρόπος «επικοινωνίας» μεταξύ διαφόρων προγραμμάτων. Επιπλέον, ο εύκολα αναγνώσιμος και κατανοητός τρόπος παρουσίασης των αρχείων αυτών αλλά και η δημιουργία, τελευταία, εύκολων στο χειρισμό βιβλιοθηκών για το χειρισμό τους, βοήθησε στην επιλογή της για το πρόγραμμά μας. Πιο σημαντικό στοίχημα όμως που δεν κερδήθηκε, ήταν η υλοποίηση μίας πολύ καλής ιδέας των Torsten Klie και Christoph Schalong [Klie and Schalong, 2002]. Δεδομένου ότι ο στόχος του παρόντος δεν ήταν η δημιουργία ανταγωνιστικού προϊόντος, αλλά η χρήση του σαν εκπαιδευτικό λογισμικό για την κατανόηση θεμάτων προσομοίωσης, αλλά και συστημάτων ουρών, μεγάλη, χωρίς επιτυχία, προσπάθεια αφιερώθηκε στην υλοποίηση παραδειγμάτων, τα οποία θα προσέφεραν κάποιο είδος αλληλεπίδρασης με τον χρήστη. Η XML μορφή αποθήκευσης των αρχείων, αρχικά προέκυψε από τη σκέψη να επεκταθεί η ιδέα των δυο Γερμανών φοιτητών και να χρησιμοποιηθεί σαν όχημα για την αποθήκευση όχι απλών παραδειγμάτων, όπως τελικά έγινε, αλλά σαν ολοκληρωμένων προβλημάτων με αλληλουχία ερωτήσεων και αποκρίσεων στο χρήστη, επίδοξο λύτη τους. Σημαντικότατος λοιπόν μελλοντικός στόχος μίας πιθανής νεότερης έκδοσης της παρούσας, θα ήταν να συμπεριλάβει μία πιο άμεση διεπαφή με το χρήστη με τη βοήθεια και αυτού του πολύ εύχρηστου τρόπου αποθήκευσης των αρχείων που ήδη συζητήθηκε.

Έχει γίνει κατανοητό ότι, για να είναι αποτελεσματική η προσομοίωση κάποιου μοντέλου και να παρέχει κάποια αποδεκτή λύση για το πραγματικό πρόβλημα που προσπαθεί να μοντελοποιήσει και να περιέχει στοχαστικά δεδομένα, θα πρέπει να συγκριθούν και να αναλυθούν έξοδοι πολλών διαδοχικών προσομοιώσεων. Κάποιες από αυτές θα πρέπει να αναλύουν σενάρια καλύτερης (best–case), χειρότερης (worst–case) ή βασικής περίπτωσης (base–case scenario). Πολλές θα πρέπει να διατηρηθούν σε κάποια εύκολα προσβάσιμη μορφή για μετέπειτα επεξεργασία. Η συγκεκριμένη υλοποίηση προσφέρει διατήρηση πολλών διαδοχικών στατιστικών, αλλά μόνο μέσα από τα πλαίσια της εφαρμογής και απλή επιστροφή συγκεκριμένων μεγεθών, όπως η μέση τιμή και η τυπική απόκλιση. Σημαντική λοιπόν προσθήκη στην παρούσα θα ήταν η διατήρηση των εξόδων σε κάποια κοινώς αποδεκτή μορφή, όπως θα ήταν ίσως κάποια μορφή XML όπως και τα αρχεία παραδειγμάτων ή προτιμότερα κάποια βάση δεδομένων.

ΠΑΡΑΡΤΗΜΑ

Στο παράρτημα που ακολουθεί, παρατίθεται η τεκμηρίωση του προγράμματος, όπως αυτή εμφανίζεται στην ίδια την εφαρμογή. Το κείμενο έχει τη μορφή υπερκειμένου με διάφορες συνδέσεις από θέμα σε θέμα αλλά και μία λειτουργία περιεχομένων και καταλόγου. Υπάρχει και η επιλογή της υπόδειξης με τον κέρσορα ενός συγκεκριμένου θέματος. Η χρηστικότητα λοιπόν της τεκμηρίωσης είναι πολύ καλύτερη στην ίδια την εφαρμογή και προτείνεται θερμά η χρήση αυτής και όχι του παρακάτω κειμένου που παρατίθεται ωστόσο για λόγους πληρότητας.

1. Simmate Environment (ΤΟ ΠΕΡΙΒΑΛΛΟΝ ΤΟΥ Simmate)

This program uses the same interface with most popular programs. Therefore as expected, by pressing Ctrl + Z an undo operation will occur on your program and by dragging and dropping an object it will be moved. If Ctrl is pressed when dragging it, a copy – paste operation will be executed etc. It has all the standard features of every similar package such as an undo – redo mechanism of every command in the main window, a way to find your component, a cut – copy – paste mechanism etc. In the following chapters instructions of this application are given. This chapter is also the help system that can be found as hypertext in Simmate by pressing F1 or choosing *Help Contents* from *Help* menu.

1.1 Help Introduction (Εισαγωγή στη βοήθεια)

To access the help system you can simply press F1 or use the “Help Contents” Command in the Help menu. If you need information for any component of the menu or button of the simmate environment you can alternatively press Shift + F1, press in the toolbar or choose the “What’s this” command in the Help menu. This operation will change the cursor into and every click on screen will reveal the same help system. If the button, menu item or component is clicked then help info is selected. To change the mode from “help” to normal without the help window popping, what the user can do is to just right click so that the cursor and mode be reset to normal again.

To navigate through the help files you can use the following tabs:

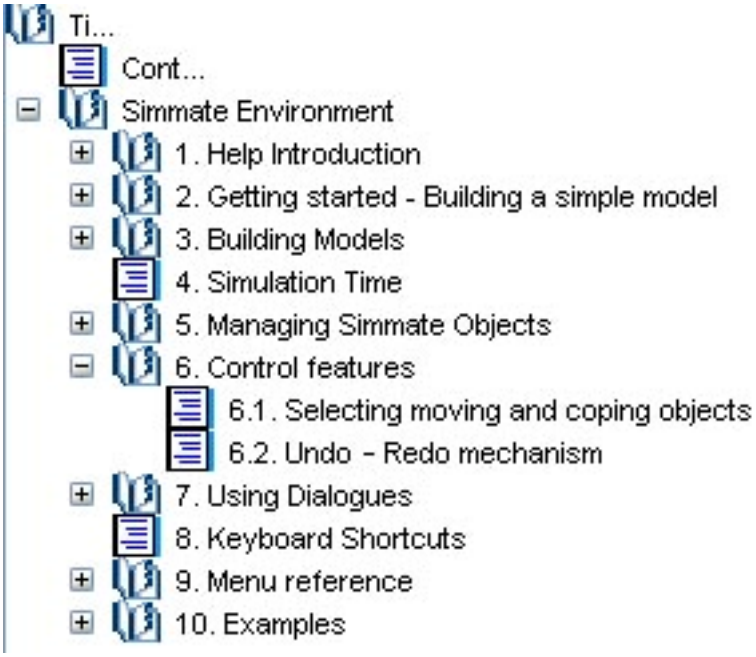
- Contents
- Index

and

- Toolbar

1.1.1 Help Contents

The user can select the contents to browse through the help contents by category. A tree appears at the left of the Help window with every help topic categorized in chapters. One of the chapters for example is that of Help Introduction with subchapters of “Help Contents”, “Help Index” and “Help Toolbar”. A more detailed sub tree for the whole documentation without subchapters can be seen below.



By clicking on the minus (-) symbol or double clicking on *Help Introduction*, the minus symbol becomes plus (+) and the only visible thing is the name of the chapter, that is the root of the tree Help Introduction. By reversing the previous procedure with the (-) and (+) symbols every leaf of the Help Introduction tree appears on the screen analytically as in the figure above. To view the *Help Information* at the left of the screen you just have to click on the leaf name. Clicking for example on *Help Index* the corresponding information will appear.

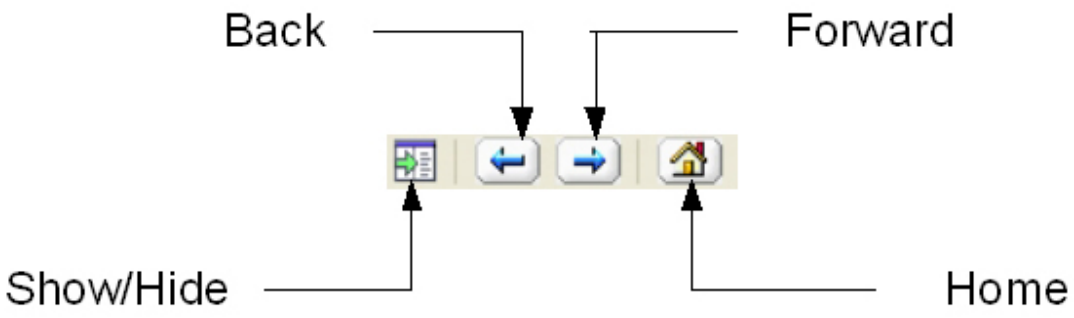
1.1.2 Help Index

Help Index should be selected if the user wishes to find information for a specific topic. In this tab the user will deal with a keyword text field, a list with several topics and a *Display* button. By typing a word in the text field, topics beginning with these written letters will be highlighted. When the desired topic is highlighted user can just press enter or the display button. This can be done without the use of the text field by just clicking on the topic. Afterwards, what is needed for the help page to appear on the right of the screen is simply the topic to be relative with one page of it. It is possible for the topic to be referred in many pages. If this is the case, a dialogue window like the one below will appear on screen for the user to choose among the relative pages.

The user just clicks on the desired page and presses *Display* or double clicks on the page. In case of a non desired page the user can cancel the action to make the dialogue window invisible.

1.1.3 Help Toolbar

In addition to *Help Index* and *Contents*, simmate help system also offers a toolbar with four operations for navigation between the help pages.



- *Show / Hide*: By pressing the button, the left side of the help window with index and contents operations disappears from the screen leaving all the space of the window to help information. When pressing the button, the left side of the window will be visible again.
- *Back*: By pressing the button, the help window information on the right size goes one page back in the history of already seen pages.
- *Forward*: With the button the help window information goes one page forward.
- *Home*: With the button it goes on the initial page.

On top of those, there are two additional buttons used for getting the user into the next or previous page of the documentation in every html file appeared in the main right side of the help window and by the page's title.

1.2 Getting started - Building a simple model (Ξεκινώντας - Δημιουργώντας ένα απλό μοντέλο)

To begin with, let's explain a model and see the procedures of laying it out, running and checking the output. The model is an example of a hypothetical dental clinic in my hometown. The analysis of the model is thorough since it is assumed that this model is the first contact of the user with the program. There are however models with a briefer description at the end of this documentation.

1.2.1 The system (Το σύστημα)

The Greek Ministry of Health and Welfare studies the case of the creation of a dental clinic in Karditsa. First of all the employment of anesthesiologists, nurses and dentists is of main interest. In addition, a study is expected regarding the number and size of reception halls in the clinic. An initial description outlined in the following paragraph must be taken into consideration and decisions should be made for more or less employments and bigger or smaller spaces.

To start with, the expected patients of the dental clinic can be assumed as infinite in regard with the patients already being in the clinic. There is no appointment system and the patients arrive in a stochastic manner that can be described as an exponential distribution with a mean of 8 minutes between two patients. In the clinic there is a waiting – room with a capacity of up to ten patients as well as a nurse dividing the customers into those needing anesthesia and those not needing one. Every waiting – room in this example is a queue with a first come – first served discipline. The nurse needs one to three minutes uniformly distributed to separate the patients. Those minutes stand for waiting time for the patient and not for service time. Sixty percent of the patient needs anesthesia and forty percent do not. Researchers came to the conclusion that in this type of clinics about sixty percent of the cases need anesthesia and forty percent do not.

The first crew of patients is led to a second waiting room for the anesthesiologist. In this room up to ten customers can wait together. What is more, there are two anesthesiologists in the same room, each one needing more or less 7 to 10 minutes with each patient and most of them needing 8 minutes. A triangular distribution could nicely produce this behaviour. In the case where both **are** idle, the first one always takes care of the patient. When those patients are ready, they move to the same queue as the patients' not needing anesthesia. The output of this server and the simple visit counter, both end at a router object which is connected with the queue for the dentist. This queue can fit up to 20 customers and leads to the dentist server. The dentist follows an exponential distribution with a mean of 7 minutes and after this the customer is guided to the exit of the whole model.

The manager of the clinic wants to examine the behaviour of the system for probable modifications. He wants to figure out whether customers are lost from the system, how much time they spent in the clinic waiting and how this waiting time is apportioned among the queues. He needs to decide if he needs to employ a new nurse, dentist or anesthesiologist or even dismiss one of them. To do so he should build a model and experiment with it by simulating its behaviour at many different times and in many different situations. Cases like the best or worst case scenario could be used here. The next subchapter will demonstrate the procedure of designing and building the model using simmate.

Finally, it should be noted that the clinic must remain open for sixteen hours during which two groups of the above employees should be needed working eight hours each.

1.2.2 The model (Το μοντέλο)

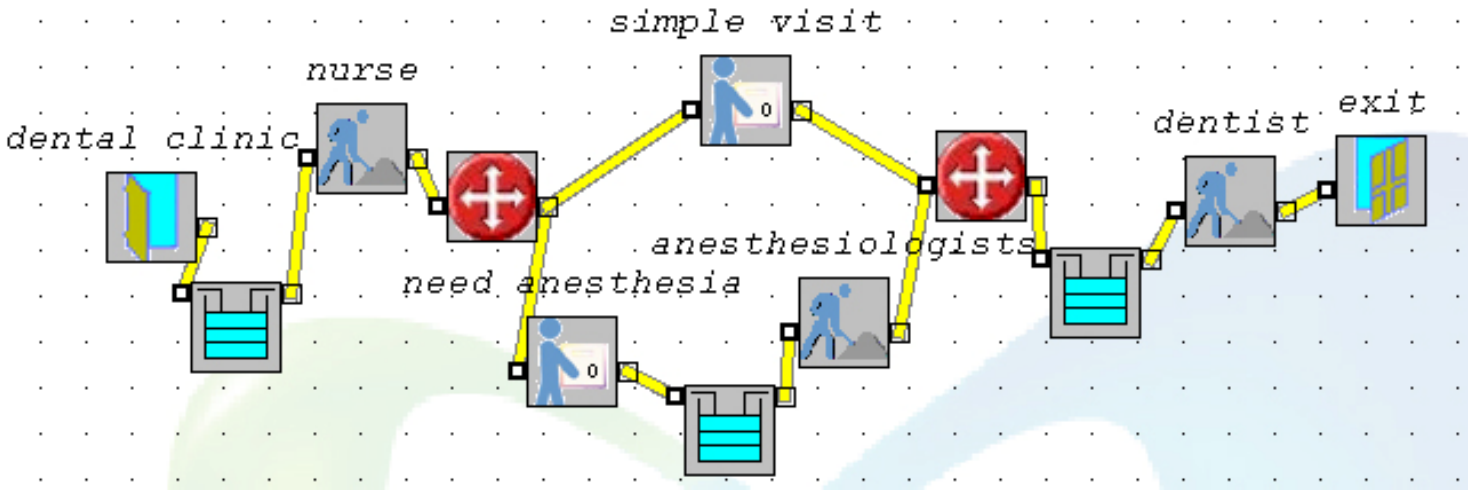
To start with, if you press the Create a New Model button in the initial screen an empty area will appear. Then click on the generator menu item on the library menu or check the library menu item on view menu and press the button on the *library tool bar*. You will notice that the cursor will change from to . That means that you have entered the "create generators mode". Every time you press the corresponding object a similar word will be written at the bottom of the cursor. To return to the default state, where no objects appear by a simple click, just right click on the model area. From now on, every click on screen will create a generator object. Just click somewhere in the area and the generator object will appear. If you are not satisfied with the position of the object just drag and drop it somewhere else. By double clicking or right clicking and choosing properties, the properties menu of the generator will appear. The name of the generator can be defined in the view: field. A default value like *Generator1* will be written in the field. That can be changed to *Karditsa dental* for example. Since the arriving pattern can be described as exponential with a mean of 8 minutes that must be defined in the Distribution menu in the middle panel of the properties menu. Choose *Exponential* from the combo and then write 8 at the *Mean* field. Unlimited items can be generated and therefore *Unlimited Items* should be checked.

The next item will be the initial queue before its separation by the nurse. Mode should be changed to queue by using one of the procedures used with the generator. Since this room can fit up to 5 persons the Maximum length will be set to 5. The name can be changed to something representative like queue separation for example and the type should be chosen as FIFO for the first come first served discipline. Actually this is the default. The separation nurse has to be created next. A server object will do the job. Constant distribution with 3 minutes time should be defined. This server counts as waiting time and therefore *Delay* checkbox should be checked. Each customer after the separation can be guided to the anesthesiologist or straight to the queue for the dentist. A router object will be needed at this point. Create it but since no connections are drawn yet, don't change the properties. Two counters counting the customers needing and not needing anesthesia would be valuable at this point. You could name them *need anesthesia* and *simple visit*.

A step that should be done at this point is drawing the appropriate connections. This is happening by dragging the mouse from the input or output of one object to the input or output of the other. You will notice that you are on a "create connection" mode by the change of cursor to a icon. Note that if a router ( )object is not the case, only one connection can be created on each object's input or output. Do so by connecting *Karditsa dental* generator with the queue, the queue with the *separation* nurse and the nurse with the router. The router should be connected with the inputs of both the counters. By doing so, the middle and right properties panel of the router with the default round robin algorithm chosen will contain: *separation* the first one and *simple visit*, *need anesthesia* the second. The second algorithm must be *weighted* and the weights must be set to 0.4 and 0.6 since sixty percent of the customers need anesthesia. A queue and the anesthesiologist server should be created and connected with the appropriate counter. The queue must be of 10 maximum length and the anesthesiologist should follow a triangular distribution with a min of 5, max of 10 and most likely service time of 8. Since there are two anesthesiologists, simultaneously serviced field should be set to 2. An additional router will be needed for associating the two types of customers into a single queue, waiting for the dentist. So a new queue is needed, with maximum

length of 20 and the dentist server which must follow an exponential distribution with a mean of 7 minutes. Finally an exit object should be connected with the dentist for customers to normally exit the model and not count as "lost".

Since the system must remain open for sixteen hours, the end time must be set to 960 since time is being counted so far in minutes. That can be changed from Simulation Options, either by pressing the button or by selecting Simulation Options from the Simulation menu. There is a keyboard shortcut as well. Ctrl + I will open the simulation properties window. The first panel has a few generic options for the simulation. One of those is *Stop Simulation At* which must be set to time 960. A few improvements can be made in this panel like making animation a bit faster by moving the slider to the right. The whole model will look like the following one:



Now we are going to move on to the next step of running the model. If you didn't manage to build the example model you can download the example *karditsa_dental_clinic.xml* from the web page and then just open it from the File Menu or by clicking the button on the main toolbar (Ctrl + O will as well do the job). Alternatively you can just use the alternate web page. Either way you are ready to move on to the next step of running the model.

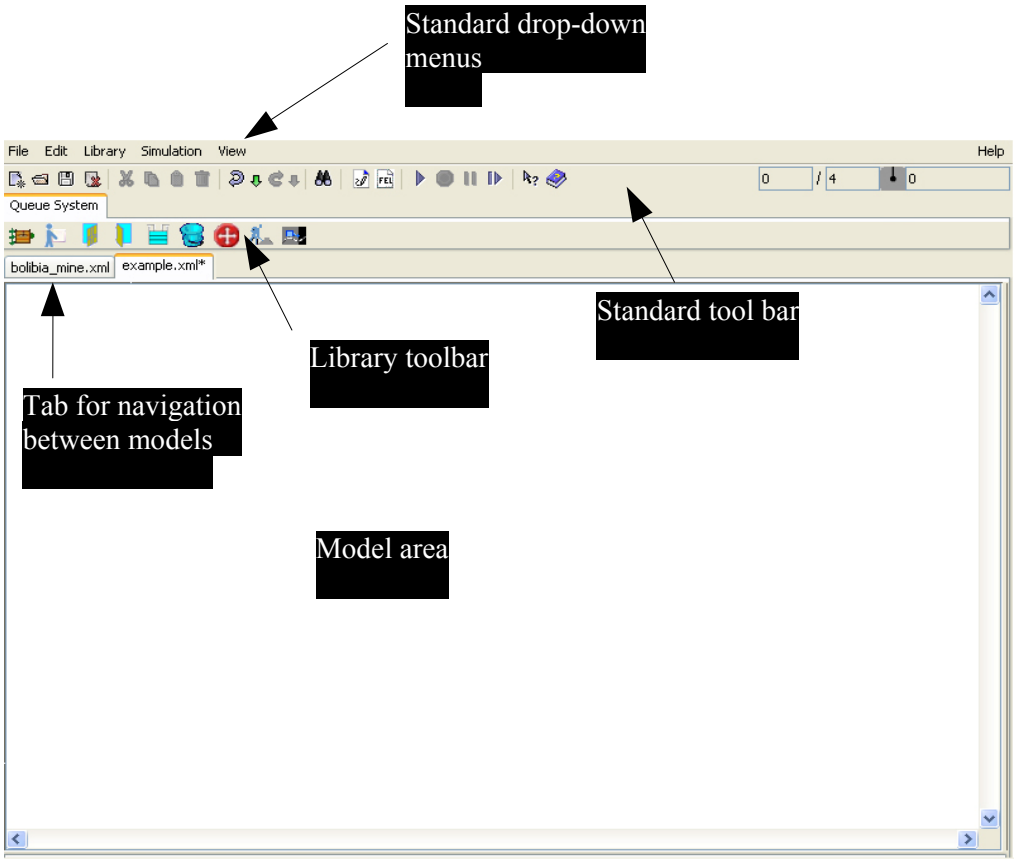
1.2.3 Running the model (Εκτελώντας το μοντέλο)

Before coming down to running the model some simulation aspects should be checked. Pressing on the standard toolbar or selecting Simulation ->Options will show the simulation properties menu. Animation could come faster by moving the slider to the right or slower by moving it to the left. No animation can be selected by unchecking the *Show animation* checkbox. For the considered model a broader simulation run must be considered. Therefore, the *Stop simulation at*: field is changed at the saved file at customer 1000 instead of time 100. Customers of the particular objects are people, so people's images are more appropriate. That can be selected from the *Animation icon*, combo box.

The simulation run buttons appear either at the standard toolbar or at the Simulation menu. There is also a selection of keyboard shortcuts. To run the model, the button, *Simulation-> Run* or ctrl+R will do the job. For a more detailed view, there is also a selection for step by step simulation. Each step represents the simulation from the execution of one event to the other. To run the model stepwise, the button, *Simulation-> Run* or ctrl+T can be used. There is also a pause choice used by pressing the button, select *Simulation->pause* from the menu or use the ctrl+P keyboard shortcut. To stop the simulation press the button, choose Simulation->stop or press escape on your keyboard.

1.3 Building Models (Δημιουργία μοντέλων)

Building models in simmate follows the window philosophy. On opening the simmate applet and after the initial page the user can see the following parts which will be explained in later chapters.



1.3.1 Building Objects (Δημιουργία αντικειμένων)

In the current first version of simmate there is one library containing nine kinds of objects, batches, counts, exits, generators, queues, resources, routers, servers and triggers. To build them one click of the mouse in the model area is enough. The mode though has to be turned to the object that must be created. That can be done by a keyboard shortcut (Alt + usually the initial of the object, e.g. Alt + R for routers – check chapter Keyboard Shortcuts), by buttons pressed in the library toolbar or by selecting the respective menu item of the library drop-down menu.

By doing so, the mode will change and the feedback given to the user will be a modification in the cursor shape. It will be transformed to one of the following: or . From now on, each click over the model area will create a new object. To shift that, a right click somewhere in the area is needed and the mode will turn back to the default.

To delete an object first of all it has to be selected. To select an object the user has to click on it and to select multiple objects a drag and drop of the mouse will create an area and all the containing objects will be selected. Afterwards a Delete, a press of the button of the toolbar or the delete menu item will do the job. For unselecting the selected objects a click somewhere else in the models area is enough.

1.3.2 Building Connections (Δημιουργία συνδέσεων)

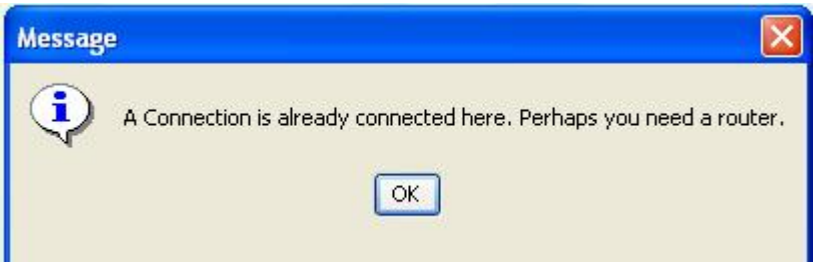
Most objects in simmate have input and output connectors. Some of them have only input (exit) or only output (generator). The way to hook the objects together is to draw a line (connection) between the input connector of one object and output connector of the other. These connectors are represented by open squares (output) and partially filled (input) squares. In addition and only for the trigger object there is a help connector.

input

output

help

It is not possible for the user to connect connectors that are inappropriate. Therefore inputs and outputs cannot be connected together and all objects except routers can have only up to one input and output. In case of wrong operation by the user something like the warning bellow will appear on screen.

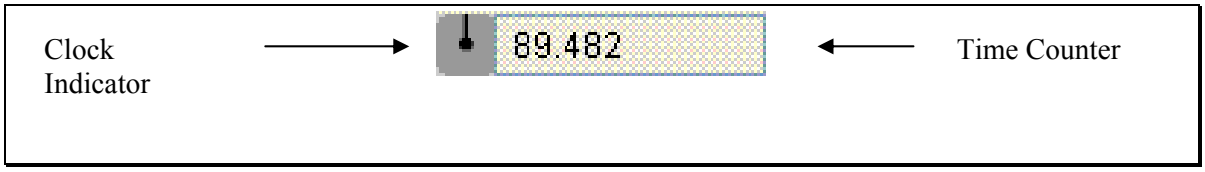


Making a connection is very simple. When the cursor is placed over an output or input connector it changes to a pen . The pen can be then dragged over an input or output connector respectively. While the pen is dragged the user can see a line shaping the connection. When the pen is over another connector the line gets thicker. By releasing the mouse a connection will be created or in case of an invalid operation a window like the ones above will appear.

To delete a connection the user has to either left click on the connection or press Delete on keyboard or menu, or press in the tool bar. She/he can also right click on the connection and chose delete from the pop up menu.

1.4 Simulation Time (Χρόνος προσομοίωσης)

The simulation clock appears on the left of the standard toolbar and counts time in standard units that can be considered seconds, minutes etc. by the user. An indication that the clock is ticking is a movement in the icon besides the change in time**.** Changes in time occur only after an event since simmate concerns a discrete event simulation.



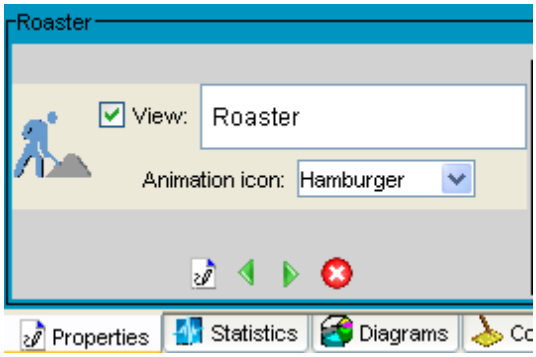
1.5 Managing simmate objects (Χειρισμός αντικειμένων εφαρμογής)

Every object of simmate has a specific operation and therefore specific menus and output information. Nevertheless there are some common operations for every implemented object.

First of all there are two ways of every object in the object properties panel to appear. Either by double or right clicking the object and selecting *Properties* in the appearing popup menu, or by selecting *Object Panel* in the *View* menu. The tabs in the panel are common for every object:

- Properties:

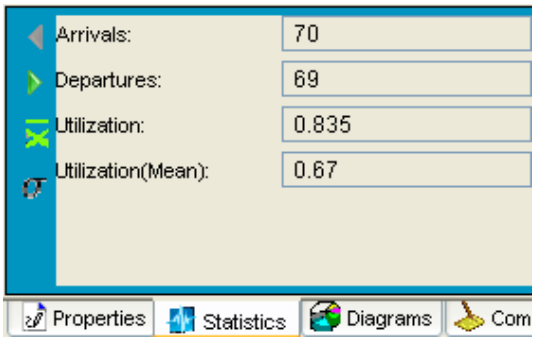
It contains menus for the user to parameterize each object. The middle and right part are different for each object but the left one is the same. It contains menus for naming the object and making that name visible at the model area. It has also a combo for selecting the icon representing the outgoing customer when animation is enabled in the model. There is a variety of choices and among them there is the *default icon* which will represent the outgoing customer as defined in the simulation menu and the *input's icon* that will leave the customer as she/he was when incoming the object. Lastly, there is a toolbar for navigation between objects.



The button is used for *simulation options* panel with information and menus for simulation runs in general. The button is used for moving to the previous object in the list and the button for moving to the next. The objects are inserted in the list sequentially. The first object user created is first on the list, the second object, second etc. If the current object is the last the button will be inactive and alternatively if it is the first then button will be inactive. The button will make this entire panel invisible.

- Statistics:

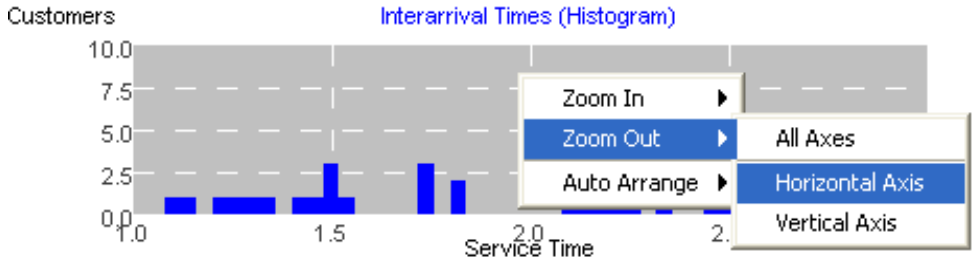
Fields representing statistical values for each object lie in this panel. Every type of object has different statistical interest and therefore different fields. Nonetheless there is a common operation for every object which is shifting between simulation runs.



The and buttons shift to the next and previous simulation run correspondingly. stands for mean of the runs and for standard deviation.

- Diagrams:

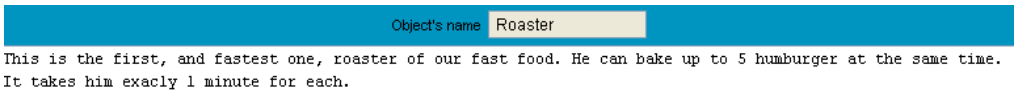
There is a bunch of different diagrams for each object of simmate but for each of those there is a standard menu for handling it. First of all user can wiggle around the diagram by just dragging and dropping the mouse towards any direction. For more sophisticated functions there is a popup menu activated by right clicking.



As it can be seen there is zoom functionality. User can zoom in or out in horizontal axis, vertical axis or both. There is also an option for arranging automatically the diagrams to the initial state.

- Comments:

This panel allows user to add some comments to every object of their model.



There is also a popup menu appearing when user clicks somewhere in all the above panels with the same navigation items (, ,) implementing the same functions (go to next object, go to previous object, hide this menu). In addition, there is an appearance choice of selecting place for the tabs, *Tabs at bottom* or *Tabs at top*.

In some cases, the user might want to concentrate to one or two of the above explained tabs. She/he can control that from the *View* menu and *Properties*, *Statistics*, *Diagrams*, *Comments* checkboxes. Only for the checked boxes the corresponding tab will appear.

1.5.1 Counter (Μετρητής)

Just passes items through and reports their number.

Dialogue Choices: There is a set of choices regarding the numeration though. The default behavior starts counting from zero in each simulation and increments the counter by one each time when an item passes through. There is a *Reset after each run* check box in the counter properties panel. When this box is unchecked the counter will not reset after each simulation and it will go on counting until the button is pressed which manually resets the counter. Since there is a different behavior in system regarding customers and items, there is a choice for counting only customers who pass through by the relative check box in the menu. There is also a choice for counter to count reversely simply by checking the corresponding checkbox.

1.5.2 Exit (Εξοδος)

The exit block passes items out of the model. There are two ways for an item to exit the model. Passing through this object is the normal one and counts as a "success", as an item has normally exited the system. Not passing through this object but exiting the system from a different object counts as a "failure", as an item has been lost from the system. Only when an item has passed through those objects, or is lost from the system, general statistics include this item. For general statistics of the model like those above, Simulation Menu should be checked. Therefore it is mandatory to have this kind of objects for items that have normally passed through a series of function and their behavior must be calculated.

Dialogue Choices: The set of choices for the exit object is exactly the same as the set for the count object mentioned previously. There is also the default behavior for an exit object starting counting from zero and incrementing the counter by one when an item is entering the object. This behavior can be changed according to the user's will. The exit counter can not be reset after each run or can be reset manually by the user**.** It can also count only customer items, count backwards and start counting from a different value and not only by zero. Items exited from this object and from the system can be viewed in the object's icon or not by checking the *View Count* Checkbox.

1.5.3 Generator (Γεννήτρια)

The generator object provides items to the simulation model according to a distribution. This distribution can be chosen among the following: Binomial, Constant, Empirical (Discrete), Poisson, Uniform (Integer), Empirical (Continuous), Exponential, Normal, Triangular, Uniform (Real) and Weibull. Changing characteristics of every distribution and generally the distribution dialogue are explained in the later chapter of Distribution Dialogue.

Dialogue Choices: Next to the distribution menu there is an additional menu for further changes in generator's characteristics. There is a combo for choosing among customer or item generation. As explained there is a whole different philosophy in holding statistics in customer and item entities. There is a checkbox for choosing unlimited generation of items, and when unchecked there is an additional field for setting the exact number of items. This generator can actually generate during one simulation run.

1.5.4 Server (Εξυπηρετητής)

The server object serves items according to the exact same distribution as the generator object previously discussed. Therefore it uses the same distribution menu. Server can sometimes act like a delay, that means each item staying in the server, if it is a delay, is an item waiting and not an item serviced.

Dialogue Choices: Server menu uses a field for setting the maximum items it can service simultaneously. There is also a checkbox for unlimited service and if unchecked the maximum items this object can service can be set in another field. Finally there is a checkbox for setting the server as a delay object.

1.5.5 Queue (Ουρά)

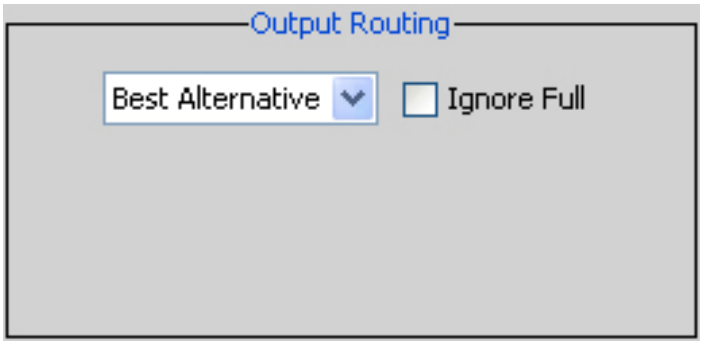
The queue object holds items and passes them to the next item when triggered in a pattern chosen by the user. Queues supported by the program are First in First out, Last in, First out and Service in Random order.

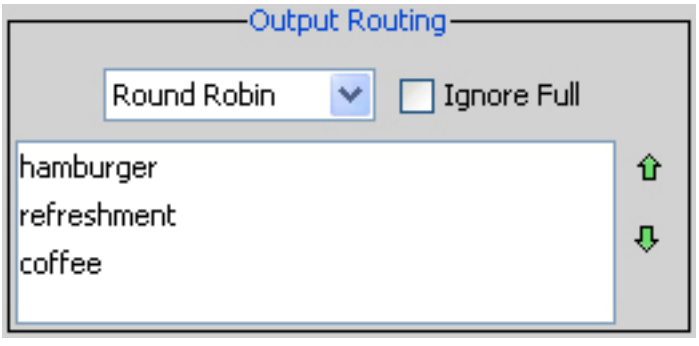
Dialogue Choices: There is a combo box for choosing which of these queue policies will be used, a field for setting the maximum length, namely the maximum number of item of the queue and a checkbox for selecting if the current length of the queue will be visible in the model area or not.

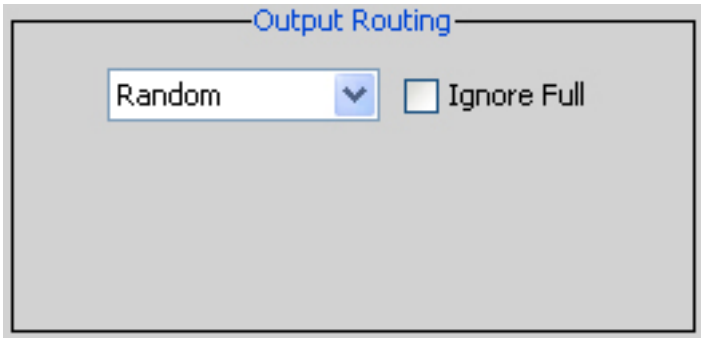
1.5.6 Routing (Δρομολογητής)

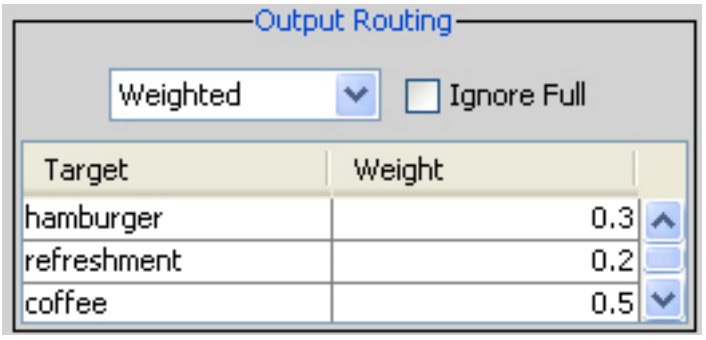
The routing object is responsible for every routing algorithm of the application. It supports the first in – first out, the last in – first out, the service in random order algorithm and even a simple best alternative algorithm. This particular object can have more than one input and output connected objects.

Dialogue Choices: Instead of the usual left menu there are two additional menus, one for the input routing algorithm and one for the output routing algorithm. They have the exact options except for an additional best alternative choice for the output routing. For the input routing this algorithm would be meaningless. The main steady panel of those both menus consists of a combo box for selecting the algorithm and a checkbox for ignoring a possible full connected object. Every choice of the combo affects the rest of the panel. The next images show the state of the panel depending on the algorithm choice.









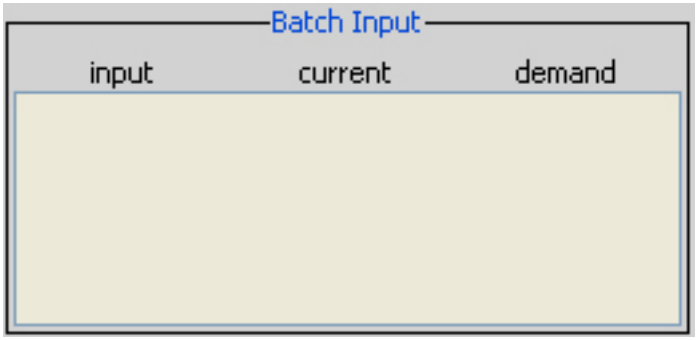
Best alternative and random algorithms don't need additional menus since there is no additional information needed. The first algorithm will search every outgoing path and search for the best one according to a grading system used by simmate. The random algorithm will make an incidental choice between every connected object. Round robin has a list with the name of every object, connected with the output or input of the router. Choosing an object and pressing the ⇧ or ⇩ button will move the object up or down the list respectively. The router will lead the first item to the first object in the list, the second item to the second object etc. When every object is used the procedure will repeat.

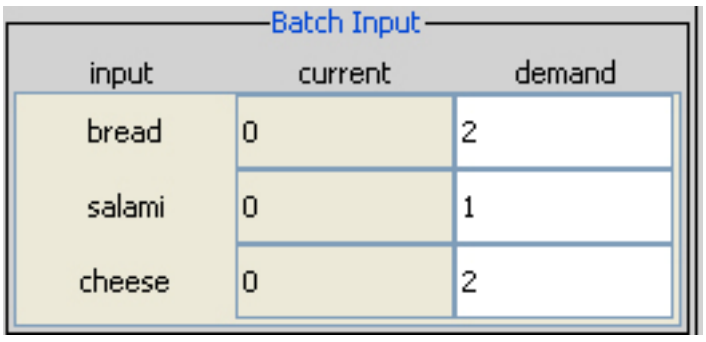
Weighted algorithm choice will create a table containing every connected object. The table consists of two columns, one with the name of the connected object and one with its weight which the user can change. The weights don't have to be among 0 and 1 since the algorithm will decide by dividing the weight of the object by the sum of every weight. Selecting the Ignore Full checkbox will make every algorithm ignore a possible full connected object and move to the next object. If for example refreshment object is full and for the time it is full, the round robin algorithm will first lead the first item to hamburger object, the next to the coffee object, the third to the hamburger object again and so on.

1.5.7 Batch (Επεξεργαστής παρτίδας)

This object can gather from its input connected object a variety of items and when its demand is fulfilled an item can be delivered to the output connected object. Like the routing object it can have more than one input but, like every other object, can have only one output.

Dialogue Choices: User can interact with this object only by setting the demand for every input connection. When batch is created, no connections are attached to it and the panel is empty. In the following figure, you can see the dialogue before and after connecting it with a few objects.





The first column represents the name of the input object, the second the current amount of the items from this object and the third column represents the demand from this input. The first batch isn't connected to any objects while the second batch collects the amount needed for a double cheese toast and passes it to the next object, possible a toaster server.

1.5.8 Resource (Πηγή)

The resource object generates items but in a different manner than the generator object. Resource can't do it stochastically but needs a trigger to do so. That is why it is frequently combined with the trigger object. It can contain a standard group of items or unlimited amount. Like the generator it can generate generic items or customers that need statistic coverage.

Dialogue Choices: Besides the standard left menu of the properties panel there is a field for setting the initial resources, a checkbox for setting unlimited or limited resources, a checkbox for viewing the resources so far and finally a combo for setting whether this object generates customers or generic items.

1.5.9 Trigger (Σκανδάλη)

Trigger objects works like a door that opens and closes for items to pass through. It is the only object that has an extra connector, the help connector. The object that enters the trigger object through this connector just opens the door and lets one item pass through its normal connectors. When it passes through, it closes the door behind. A new item has to enter from the help connector to open the door again. The most usual use of this object is combined with a resource object.

Dialogue Choices: Due to its operation it doesn't need any interaction with the user besides the left panel, common for every object, with the image and name menus.

1.6 Control features (Χαρακτηριστικά ελέγχου)

1.6.1 Selecting, moving and coping objects (Επιλογή, μετακίνηση και αντιγραφή αντικειμένων)

The way to move an object is by simply dragging it with the mouse. Dragging it with the control button pressed it will copy and paste it instead of moving it. To move or copy multiple objects the area should be defined first. Dragging and dropping the mouse will define the area and a drag of this area will move all the objects contained. A Control key pressed will result to a copy and paste of the same objects.

The movement of the object will just change the position of the object in the model area. The possible connection with other objects will remain the same. In reverse by copying and pasting the object, no connections will be copied. The copied object will have the same parameters with the initial objects (e.g. same queue length, same interarrival time distribution). Only if multiple objects are copied they will be connected when needed.

1.6.2 Undo – Redo mechanism (Μηχανισμός αναίρεσης)

By clicking the button, or alternatively pressing Ctrl + Z or selecting Undo from Edit Menu it will remove the last changes made to the model. Multiple sets of changes are stored in simmate. The number of those is set by the user. To go back to one of those, user has to select the button next to and a popup menu will appear with a short definition of every change. Just choose the name of the change and the model will return at the form before the change. Redo lets the user put back the changes which were removed by the most recent undo operations. Redo can be accessed either by clicking

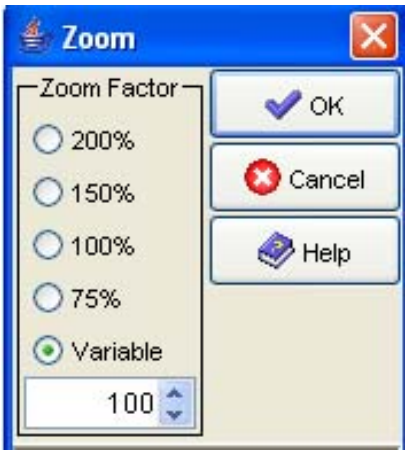
button in the standard toolbar, pressing Ctrl + Y or selecting Redo from the Edit Menu. There is also an option for multiple redo moves from the button next to . A similar popup menu like the one in undo operation will appear.

The depth of undo – redo operations can be set by the user. This can be done in the Simulation Options menu. There is a field there with the label "Depth of undo operation". Just key the number of the desirable depth. Keep in mind though, that very large depth will demand memory for keeping everything. There is already a default depth of five that in our opinion is enough for every messy user.

1.7 Using Dialogues (Χρησιμοποίηση διαλόγων)

1.7.1 Zoom (Εστίαση)

The zoom operation reduces or enlarges the screen display of simmate. To access this command user has to choose Zoom from the View Menu. A Zoom Dialog will appear on screen.

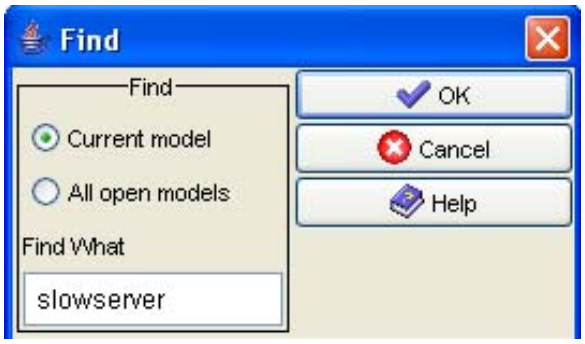


The radio buttons on the left are used to set the zoom factor. This can be done either by setting a custom choice, like 100% or 200%, or by selecting variable and set it from the spinner menu. The zoom factor can be between 10% and 400%. Pressing OK will apply the zoom factor and pressing Cancel will retain the previous value. There is also a help button, which will provide help for the zoom operation.

There is a default zoom status and size of every object in simmate. When 100% is selected every object will have those default dimensions, when 200 % is selected every object will have double size than the normal, e.t.c.

1.7.2 Find (Εύρεση)

This dialogue is used for finding an object either on the current open model or on every open model of simmate. To access it the user must choose Find from the Edit Menu. Alternatively She/he can press Ctrl + F. There is also a button in the standard menu. A Find dialogue like the one bellow will appear on screen.



The radio buttons on the left are used for searches in the current model or in every open model. The text field is used for setting the name of the object the user wants to find. Pressing OK will highlight the object if there is one. When "All open models" option is active and an object with the given name is found in a different model to the one which is currently active, the model with that object will become active and the object will be highlighted. This operation stops when an object fitting the description is found regardless of the existence of more objects with the same name. Pressing cancel will just hide the dialogue menu with no actions and pressing help will provide help for this operation.

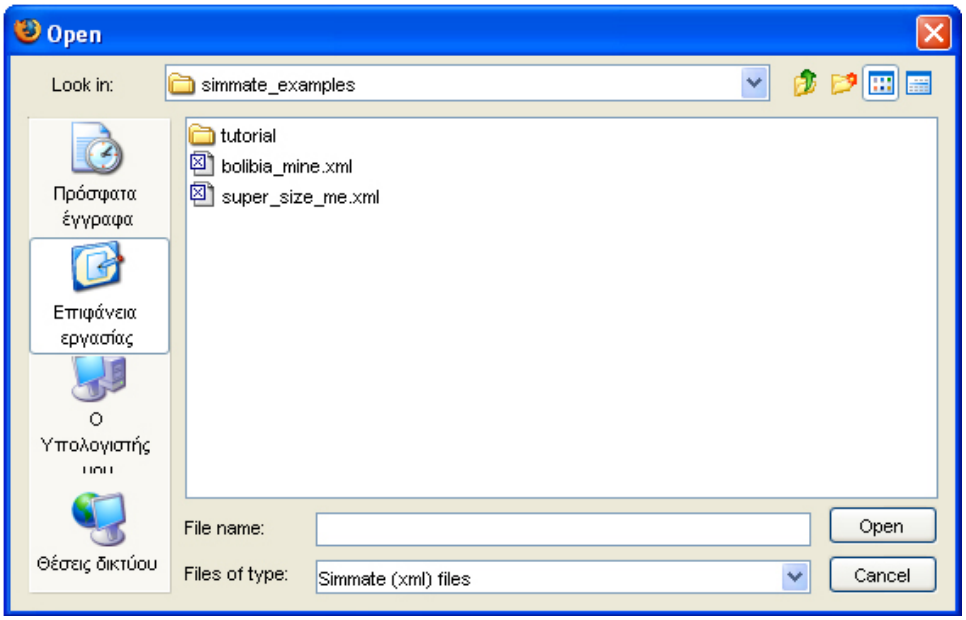
1.7.3 Background (Φόντο)

Simmate has a menu for setting the background image and appearance of the grid. This dialogue is accessed by the View Menu and the Background choice.

The background checkbox is used for selecting whether the logo of simmate will be visible as background in the screen or not. The display grid sets the visibility of a grid in the model area. When selected the spinner is enabled. With the spinner user can select the density of the grid which can be between 10 and 100 pixels. OK button will apply the changes, cancel will keep the previous adjustments and Help will provide the relative help instructions.

1.7.4 File open (Άνοιγμα αρχείου)

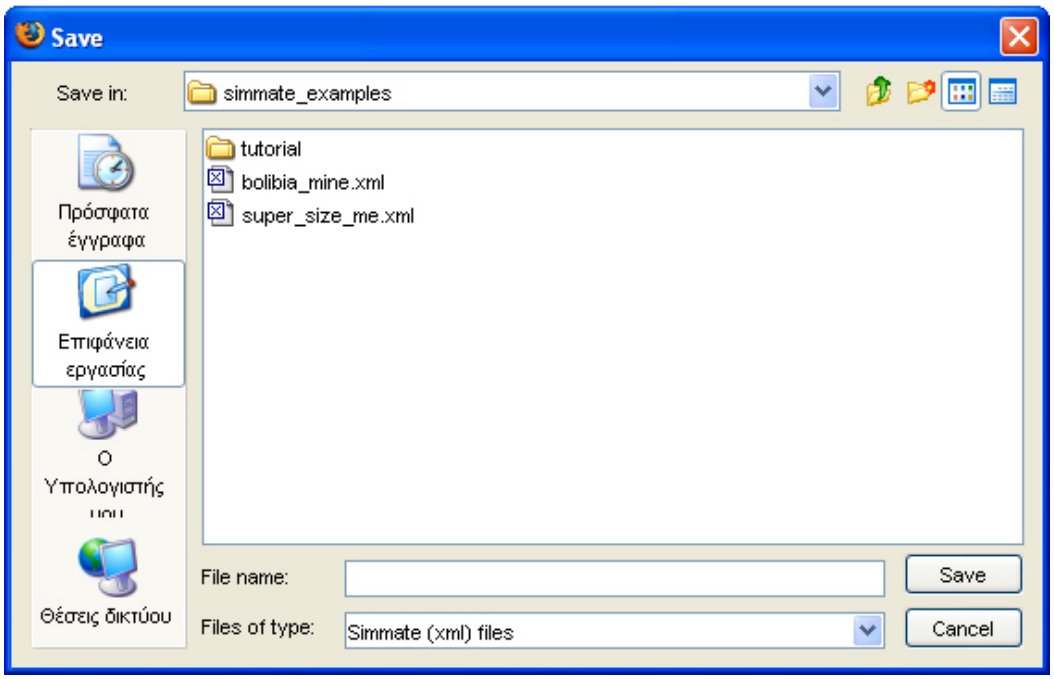
This is the menu for opening an existing model from the file system. The file open instruction is accessed via the *File menu* and the *open* choice or by pressing the button. There is also a keyboard shortcut – Ctrl + O.



The specific look and feel is windows style and works like every other file open operation the user has encountered. The left part of the menu contains a few links for the most common folders of the user's computer; the right part contains the files and folders of the current folder and the top part contains specific folder navigation and appearance operations. The only files visible by default as appeared in the bottom part of the menu are by default only those with the suffix xml. Every example of the simmate is saved at this form, as an xml document, and has of course an explicit pattern.

1.7.5 File save (Αποθήκευση αρχείου)

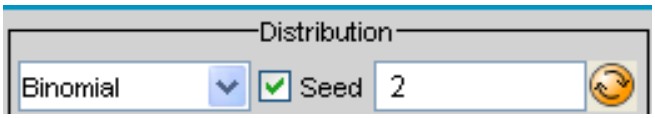
This is the dialogue for saving the current open model into a file. Save is accessed either from File menu or by the shortcut (Ctrl – s) or by pressing the button in the standard toolbar.



This is also a windows look and feel and works exactly as every file save operation and like the file open operation. The form simmate that saves the files has a specific pattern and must have an xml suffix.

1.7.6 Distribution menu (Επιλογές κατανομών)

This menu is used in the properties panel of every object that needs a stochastic factor in its behavior in the model. The objects with this behavior in this version of the program are Generation and Server objects. The first part of the menu is the same regardless of the chosen distribution.



It consists of a drop down menu where the user can choose among a variety of distributions, a checkbox where She/he can choose in case where the random function of the programming language will use a given seed or the seed will come up automatically from computer time. This given seed can be given in the following field and lastly the button can reset the random function and start again the procedure from the given (or automatically produced) seed. The second part of the menu depends on the chosen distribution.

- Binomial

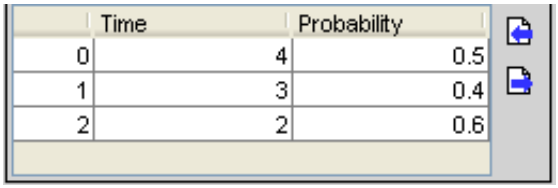
There is a field for setting the probability of a Bernoulli trial that must be less than one (and above zero) and a field for setting the number of Bernoulli trials that must be a positive integer.

- Constant

There is only a filed necessary for this distribution that must be positive and sets the interarrival time that is no deterministic.

- Empirical(Dis.)

This distribution menu consists of a table and two buttons as appears below.



	Time	Probability
0	4	0.5
1	3	0.4
2	2	0.6

The and buttons are for adding or removing lines from the adjacent table. The table has three columns, the first of which is just a numeration of the lines, the second is used for setting the interarrival or service time and the third one is for setting the probability of each of those times. The sum of the probabilities doesn't have to be 1, the calculations will occur by dividing each probability by their total.

- Poisson

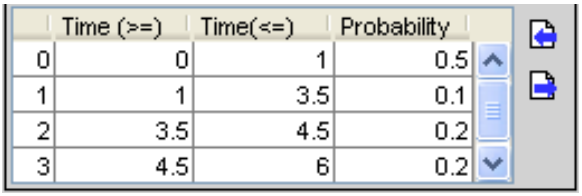
The only variable needed for this distribution is its mean and therefore there is only one field which must be positive.

- Uniform(Int.)

The Integer kind of this distribution needs two factors to be set by the user. The minimum and maximum that must be positive Integer and min must be less than max.

- Empirical(Con.)

This menu is almost the same with the discrete kind of this distribution as appears below.



	Time (>=)	Time(<=)	Probability
0	0	1	0.5
1	1	3.5	0.1
2	3.5	4.5	0.2
3	4.5	6	0.2

The and buttons are also for adding and removing lines from the table. This table has four columns. The first is just a numeration of the lines, the second and third are used for setting the limits of one of the uniform distributions comprising the distribution and the last column is used for setting the probability for this choice to be selected.

- Exponential

The only variable necessary to define this kind of distributions is its mean and therefore there is only one field that has to be a positive real number.

- Normal

The two variables, and therefore two fields, to define those distributions are mean and standard deviation. Variables that must be positive real numbers.

- Triangular

This distribution necessitates three variables to be defined. The minimum and maximum values must be greater or equal to zero. In addition maximum must be greater than min. It is the third variable and field that most likely defines the mode variable.

- Uniform(Real)

The real kind of the uniform distribution needs like the integer two variables set by the user. The minimum and maximum must be positive and in addition must stand minimum <= maximum.

- Weibull

This last kind of distributions demands the scale and shape factors to be defined. Both factors must be positive.

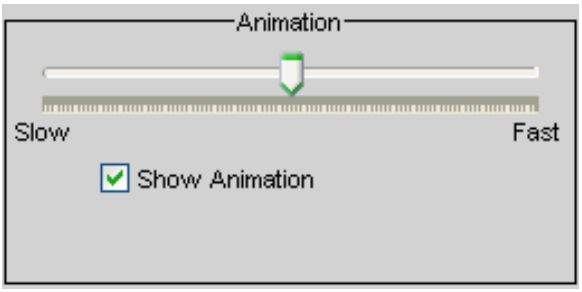
1.7.7 Simulation Menu (Επιλογές προσομοίωσης)

To access this menu user has to either press the button, or from the Simulation menu (alternatively Ctrl + I) select Options. This menu is divided into four parts, the properties, statistics, diagrams and comments panel. These panels, represented as tabs are the same to the corresponding panels for the object explained at the Managing simmate objects section.

- Properties

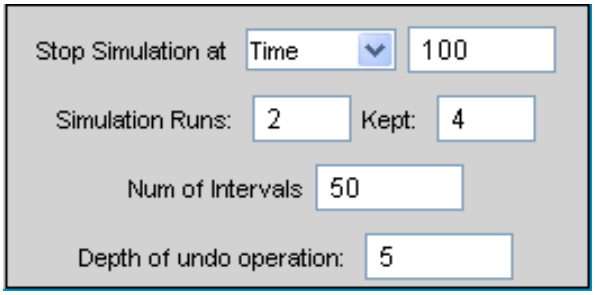
The first part of this panel contains exactly the same menus as the object properties panel. There is a view checkbox in which the names of every object become visible when it is checked. This happens when the object has the corresponding checkbox checked as well. When unchecked, no object name appears in the model area independently of every object's choice. There is a field for changing the name of the model. Take into account that there is a difference between the name of the model and the name of the file the model is saved. The drop down menu is used for setting the default icon used for animation all over the model except when it is differently set in any corresponding object menu as mentioned at the managing simmate objects section. There is also a navigation menu mentioned in the same section.

The second part of the panel concerns the animation of the model and contains a slider menu and a checkbox.



The slider is used for setting the speed of the model's animation and the checkbox for setting model's animation or not.

The third part of the panel contains some generic options about the simulation run.



The drop down menu sets the stop condition of the whole simulation. This is either a time limit, like one hundred in the image above, or a limit in customers. For example the meaning of the above first set of menus is "Stop Simulation at Time one hundred" and could easily become "Stop Simulation at Customer one hundred fifty".

The following *Simulation Runs* field stands for sequential runs of the simulation without the user's order. The *Simulation Kept* field stands for how many runs should be held in the program's lists for statistical reasons.

The *Num of Intervals* field has to do with general diagram operations

- Statistics

The statistics presented in this tab have to do with generic statistics of the whole simulation. It represents statistics for customers entering, exiting, lost and still in system, the average total time in system, the average time waiting and servicing. In addition the total utilization is represented in this statistics panel. Navigating through sequential simulation run presentations and present mean and standard deviation works identically like in the objects statistic panel.

- Diagrams

The diagram appearing in this tab concerns the whole system's utilization, the time in system, as well as waiting and servicing for each customer exiting the system. In addition and likewise the statistics panel contains diagrams for customers' average time waiting, servicing as well as both of them.

- Comments

This tab of the simulation menu is exactly the same with the relative tab of every object. It contains a field for the name of the current model and a field for possible comments of the user for this model.

1.7.8 Future Event List Panel (Πλαίσιο επόμενου γεγονότος)

To make this panel visible the user can either press the FEL button at the standard toolbar or select the *Event List* menu item at the *View* menu. This is the panel that provides feedback to the user about the status of the future event list at any time of the simulation run. Every future event of the model appears in the represented table. There is an additional choice of a *View Past Events* checkbox. If selected the table not only shows the list of every future event but also the list of every past event. Deselecting it will lead to an erasure of every line with past events and selecting it again will redraw every such line. Every object, past or future, will be represented in a chronological order from time zero till present time of the simulation.

To make this panel invisible, the button must be pressed or alternatively unselect the *Event List* menu item at the *View* menu.

1.7.9 Customer Panel (Πλαίσιο πελάτη)

There are two different ways to access this menu, either by selecting *Customer panel* in the *View* menu or by right clicking on a customer and selecting *Properties*. In this panel user can view some statistics of each customer and a diagram of each customer's utilization. She/he can see customers that have already left the system, or they are still on it. There is a navigation toolbar for moving among them or hiding the whole panel. The statistics shown in this panel are, the time that customer generated and exited from the system, the time serviced and waiting and the total time in the system. Finally, there is a pie form diagram making a visual representation of each customer's utilization.

1.7.10 Standard Toolbar (Τυπική Εργαλειοθήκη)

Every frequently used operation besides the shortcut discussed in the following subchapter and the drop down menus discussed later have an alternative way to be called via the standard toolbar. By default the standard menu is visible but user can choose it to be invisible. That can be done by selecting or unselecting *View -> Toolbars -> Standard*. The operations contained in this standard toolbar are

- , New File, creates a new blank model.
- , Open File, opens an existing model. To select the file the file open dialogue box is displayed.
- , Save File, saves the active model with its current name.
- , Close File, closes the current model.
- , Cut, cuts every selected object and keeps everything into a clipboard.
- , Copy, copies every selected object and keeps everything into a clipboard.
- , Paste, pastes everything from the clipboard.
- , Delete, deletes every selected object.
- , Undo, Undo the last operation.
- , Undo in Depth, Pop ups a menu with a list of the most recently operations. User can select one of them to go the previous state of the model.
- , Redo, Redo the last operation undone.
- , Redo in Depth, Pop ups a menu with a list of the most recently undone operations. User can select one of them to go to the state of the model before this operation.

- , Find, pop ups find dialogue. It is used for finding objects in every open model or in the current model.
- , Simulation Options, makes simulation properties panel visible. Contains lots of generic options, statistics, diagrams and presumable comments for the simulation.
- , Future Event List Panel, makes the panel for the future or past events visible or invisible.
- , Run Simulation, starts the simulation.
- , Stop Simulation, stops the simulation.
- , Pause Simulation, pauses the simulation.
- , Step Simulation, runs the simulation step by step.
- , What's this, changes the mode of the cursor and when pressed to a button, item or object opens the documentation frame.
- , Help Contents, open the documentation frame.

1.8 Keyboard Shortcuts (Συντομεύσεις πληκτρολογίου)

Description	Keyboard Shortcuts
Undo	Ctrl + Z
Redo	Ctrl + Y
Cut	Ctrl + X
Copy	Ctrl + C
Paste	Ctrl + V
Delete	Delete
Select All	Ctrl + A
Find	Ctrl + F
New File	Ctrl + N
Open File	Ctrl + O
Save	Ctrl + S
Select Batch	Alt + B
Select Count	Alt + C
Select Exit	Alt + E
Select Generator	Alt + G
Select Queue	Alt + Q
Select Resource	Alt + O
Select Routing	Alt + R
Select Server	Alt + S
Select Trigger	Alt + T

Simulation Info	Ctrl + I
Run	Ctrl + R
Stop	Escape
Step	Ctrl + T
Pause	Ctrl + P
Help	F1
What's This	Shift + F1

1.9 Menu reference

Simmate has the following menus

- File
- Edit
- Library
- Run
- View
- Help

In each of these menus every operation can be found. Some of those operations can be also called from the main toolbar, library toolbar or with the use of a keyboard shortcut.

1.9.1 File Menus

Menu Command	Alternative call	Description
New	, (Ctrl + N)	Creates a new blank model.
Open	, (Ctrl + O)	Opens an existing model. To select the file the file open dialogue box is displayed.
Close		Closes the active model. If the model contains unsaved changes you will be prompted to save the model before closing with the appearance of the File Unsaved dialogue box.
Close All		Closes all opened models. For all the unsaved models, the user will be prompted with the File Unsaved dialogue box.
Save	, (Ctrl + S)	Saves the active model with its current name.
Save As		Saves the active model with a different name.
Save All		Saves all the active models with their current names.

3.9.2 Edit Menus

Menu Command	Alternative call	Description
Undo	, (Ctrl + Z)	Reverses the last command. To reverse more than one command, user has to click the arrow, next to and a pop-up window with the last commands will appear. Just click to the appropriate command. To set the depth of the command use the simulation info properties panel.

Redo	, (Ctrl + Y)	Reverses the action of the undo command. To reverse more than one action, the user has to click the arrow, next to and a pop-up window with the last undo commands will appear. Just click to the appropriate command. To set the depth of the command use the simulation info properties panel.
Cut	, (Ctrl + X)	Deletes the chosen objects and connections and holds it for possible paste command. Attention, only one cut or copy command can be held for future paste command.
Copy	, (Ctrl + C)	Holds the chosen objects and connections for possible paste command. Only one copy or cut command can be held for future paste command.
Paste	, (Ctrl + V)	Creates and places new objects and connections on the model area. The new objects have the same characteristics as the objects users cut or copied. The location for the new objects to be drawn is calculated by the relative place of the original objects and the last point clicked. Only the connections among objects held are created with the paste command
Delete	, (Delete)	Deletes the selected objects and connections.
Select All	(Ctrl + A)	Selects every object and connection on current open model.
Find	, (Ctrl + F)	Searches for an object in model area. When pressed find dialog box appears on screen

1.9.3 Library Menu

Menu Command	Alternative call	Description
Batch	, (Alt + B)	Sets batch as the objects to be created.
Count	, (Alt + C)	Sets count as the objects to be created.
Exit	, (Alt + E)	Sets exit as the objects to be created.
Generator	, (Alt + G)	Sets generator as the objects to be created.
Queue	, (Alt + Q)	Sets queue as the objects to be created.
Resource	, (Alt + O)	Sets resource as the objects to be created.
Routing	, (Alt + R)	Sets routing as the objects to be created.
Server	, (Alt + S)	Sets server as the objects to be created.
Trigger	, (Alt + T)	Sets trigger as the objects to be created.

1.9.4 Run Menu

Menu Command	Alternative call	Description
Simulation Info	, (Ctrl + I)	Opens the simulation info panel, where user can see overall statistics and diagrams of the model and set a few program properties.
Run	,(Ctrl + R)	Starts the simulation of the current model.
Stop	, (Escape)	Stops the current simulation run.
Pause	(Ctrl + P)	Pauses the current simulation run or restarts the paused simulation run.
Step	(Ctrl + T)	Starts a new simulation or the next step of the current simulation. The simulation is executed event by event.

1.9.5 View Menu

Menu Command	Alternative call	Description
Toolbars		Contains Check boxes with options for toolbars to appear.
Standard		The check box menu item where we set whether we want the standard toolbar

		visible or not.
Library		The check box menu item where we set whether we want the library toolbar visible or not.
Properties		The check box menu item where we set whether we want the properties tab of the object's or simulation info visible or not.
Statistics		The check box menu item where we set whether we want the statistics tab of the object's or simulation info visible or not.
Diagrams		The check box menu item where we set whether we want the diagrams tab of the object's or simulation info visible or not.
Comments		The check box menu item where we set whether we want the comments tab of the object's or simulation info visible or not.
Customer Panel		The check box menu item where we set whether we want the customer panel visible or not.
Event List		The check box menu item where we set whether we want the future event window visible or not.
Show object names		The check box menu item where we set whether we want the name of each object visible or not.
Connections		The check box menu item where we set whether we want the connections between objects visible or not.
Background		Sets the background and grid of the model When pressed background dialog box appears on screen.
Zoom		Sets the zoom factor of the model. When pressed zoom dialog box appears on screen.

1.9.6 Help Menu

Menu Command	Alternative call	Description
Help Contents	, (F1)	Opens the Help frame.
What's This	, (shift + F1)	Changes the mode to help. The next menu or button pressed will pop up the help frame with help for the selected command. Right click or the first question will set the mode back to normal.

1.10 Examples (Παραδείγματα)

In this section of the help system application a few more examples of models will be presented. The first one concerns a fast food, and the title is inspired from a book (and movie) by Eric Schlosser [Schlosser, 2001].

Every example presented in this tutorial is located in the exact web site and folder with the whole application. To load them you can either save them locally and open them from the application or use an alternate url. For example instead of Simmate.html use Simmate_fast_food_nation.html and the file will be loaded together with the application.

1.10.1 Fast Food Nation (Το έθνος της ταχυφαγείας)

1.10.1.1 The system (Το σύστημα)

A young businessman in Leros considers opening a small fast food business in Agia Marina. He is thinking of hiring just a few employers to start the business. He is thinking of using a cashier, two roasters and a coffee maker. He hired a roaster from a competitor and wants to study the behaviour of his cook shop and make up his decisions. In addition when starting his business he only desired selling just a few products like coffees refreshments and one type of burger. Studies

from a similar area indicate that a close draft of incoming customers would be an exponential distribution with a mean of five minutes. Possible customers can be assumed unlimited relatively with the customers already in system.

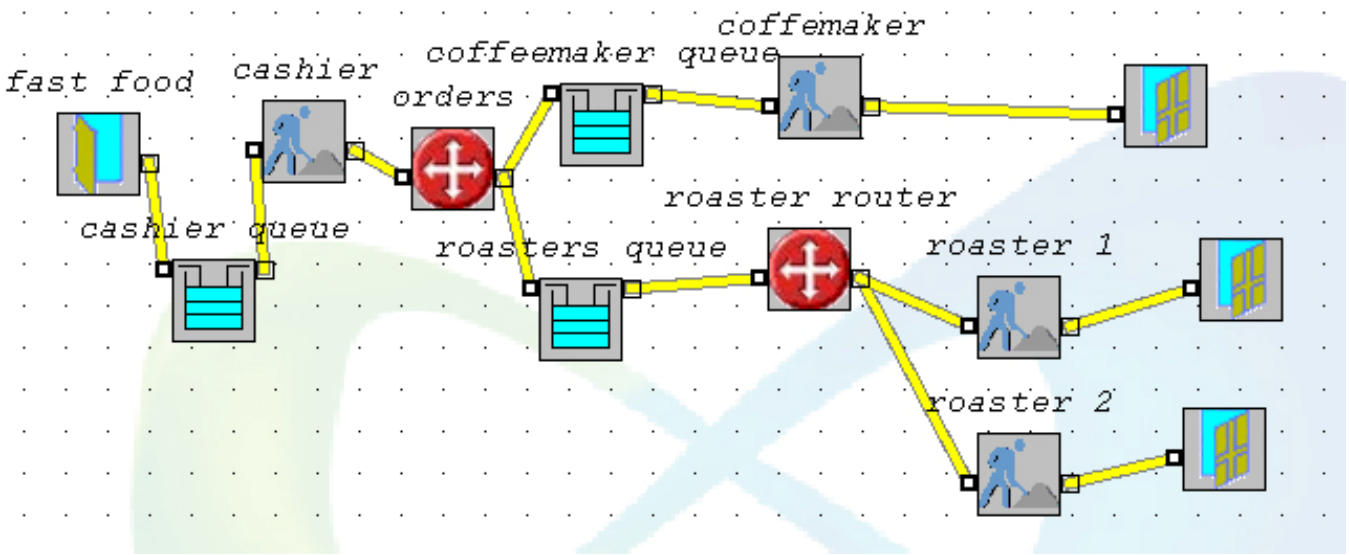
The cashier takes orders in a stochastic fashion that can be nicely represented as uniformly distributed among 0.5 and 1 minutes. There can be a queue of maximum three people in front of the cashier. There are two kinds of orders, food orders and coffee or refreshment orders. The last type of orders is serviced by the coffeemaker which when the order concerns a coffee needs uniformly distributed time among 2 and 4 minutes and when the order concerns refreshment, needs uniformly distributed time among 1 and 2 minutes. There is a queue in front of this server that its total length can be calculated as five customers. That is because the businessman has estimated that customers have the tendency to leave when the queue is bigger and she/he just wants a coffee or refreshment. Forty percent of the orders concern this type of orders, thirty of each, concern refreshment. The rest of the orders concern food orders and it is a job of the roasters.

There are two roasters that can service orders represented as exponential distributions. One of them is slower and has a mean of 7 minutes for each order and the other has a mean of only 5 minutes. They take orders in a round robin fashion but when one of them is busy the other takes the order regardless whose turn it is. There is a common queue for both roasters. The businessman has come to the conclusion that customers tend to be more patient when their order concerns food and therefore its length can be determined to fifteen customers. Finally the businessman is thinking of keeping his shop open from twelve to six o'clock in the afternoon.

1.10.1.2 The model (Το μοντέλο)

The arriving of customers can be represented with a generator with exponential distribution and a mean of 5 minutes. The generator must be connected with a first come – first served queue with a maximum capacity of 3. After the queue customers are served from the cashier which needs uniformly distributed time among 0.5 and 1. Customers have to be separated to those ordering food and refreshments or coffees. A router would be suitable at this point with a weighted routing algorithm leading customers to the coffeemaker queue or the roasters queue. Forty percent of the customers must be guided to the coffeemaker queue therefore a weight of 0.4 would be suitable. The rest sixty percent must be guided to the roasters queue therefore a weight of 0.6 would be suitable. The queues connected with the router object must hold up to 5 and 15 customers respectively.

To represent the behaviour of the coffeemaker an empirical distribution should be used. The first uniform distribution must be for customers ordering refreshment among time 0 and 1, with a probability of 0.3 and the second one for customers ordering coffee must be among 2 and 4 minutes with a probability of 0.7. Both the roasters follow an exponential distribution with a mean of 5 and 7 minutes respectively. There is only one queue though and therefore a router using the round robin algorithm and ignoring a possible busy roaster is needed.


fast food
cashier
coffeemaker queue
coffemaker
orders
cashier queue
roaster router
roasters queue
roaster 1
roaster 2

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Anderson D.R., Sweeney D.J. and Williams T.A., 2003, An introduction to management science quantitative approaches to decision making, 10th ed., Thomson, Ohio pp 526 - 623
2. Brockmeyer E., H.L. Halstrom and A. Jensen, 1948. *The Life and Works of A.K. Erlang* Transactions of the Danish Academy of Technical Sciences, No. 2 WEB-based edition
3. Tanner, M., 1995. *Practical Queueing Analysis*, McGraw-Hill, pp 36 – 38, 89 - 94
4. Οικονόμου Γ.Σ. και Γεωργίου Α.Κ., 2005, *Ποσοτική ανάλυση για τη λήψη Διοικητικών Αποφάσεων*, τόμος Β, Εκδόσεις Μπένου, Αθήνα
5. Hock N. C., 1996, *Queueing Modeling Fundamentals*, John Wiley & Sons Ltd, West Sussex
6. Erlang, A. K., 1917, *Solution of some problems in the theory of probabilities of significance in automatic*
7. *telephone exchanges.* Electroteknikeren
8. Kendall, D.G., 1953, Stochastic processes occurring in the theory of queues and their analysis by the method of imbedded Markov Chains, Annals of Mathematical Statistics, Vol 24 pp 338 – 354
9. Chung C.A., 2004. *Simulation modeling handbook. A practical Approach*, CRC Press, London
10. Gordon G., 1978, *System simulation*, 2nd ed, Prentice Hall, New Jersey
11. Pidd M., 1998. *Computer Simulation in Management Science*, John Wiley & Sons Ltd, West Sussex
12. Garrido J.M. 2001, *Object Oriented Discrete Event Simulation with java A Practical Introduction*, Kluwer Academic/Plenum Publishers, New York. pp 57-58, 64-67
13. Law, A.M. and D.W. Kelton, 2000. *Simulation Modeling and Analysis,* 3rd ed., McGraw-Hill, New York.
14. Fishman G.S., 2001, *Discrete-Event Simulation. Modeling, Programming and Analysis*, Springer-Verlag, New York.
15. McCormack W.M. and Sargent R.G., 1981. Analysis of future event set algorithms for discrete event simulation, Comm. ACM, 24, pp 801-812.
16. Williams J.W.J., 1964, *Algorithm 232*, Comm. ACM,7, pp 347-348
17. Kleijnen, J. P. C., 1974. *Statistical Techniques in Simulation*. New York: Marcel Dekker Inc.
18. Banks E. and J.S. Carson II, 1984. *Discrete Event System Simulation,* Prentice Hall, New Jersey.
19. Forsythe G.E., Malcolm M.A. and Moler C.B.,1977. *Computer Methods for mathematical Computations*, Prentice Hall, New Jersey
20. Tippett L H C. and Pearson K., 1927. *Random Sampling Numbers: Tracts for Computers, No. XV*, Cambridge University Press, London
21. The Rand Corporation, 1955, *A Million Random Digits with 100,000 Normal Deviates*, the Free Press, Glencoe, Illinois
22. Lehmer D. H., 1949. *Mathematical methods in large-scale computing units*. In Proceedings 2nd Symposium on Large-Scale Digital Calculating Machinery. Cambridge MA 141-146.
23. Knuth, D. 1969.*The Art of Computer programming, Volume 2 / Seminumerical Algorithms*, Addison – Wesley, pp 9 –20, pp 117 – 118

24. Box G.E.P. and M.E. Muller, 1958. *A note on the generation of random normal deviates*, Annals Math. Stat. 29, pp 610 – 611
25. Ross S.M., 1997. *Introduction to probability models 6th Edition*. Academic Press, San Diego
26. Schmeiser, B.W., 1980. *Random Variate Generation: A survey in Simulation with discrete models: A State of the Art View*
27. Kachitvichyanukul, V. and B. Schmeiser, February 1988. *Binomial Random Variate Generation*, Communications of the ACM 31, 2 , pp 216 – 222.
28. Dennis A., Wixom B.H. and Tegarden D., 2002, *Systems Analysis and design, an object oriented approach with UML*, John Wiley & Sons inc., New Jersey pp 193 – 198
29. Estrin D., 1999. Network Visualization with the VINT Network Animator Na*m*, *tech. report*, Computer Science Dept., Univ. Southern California, Los Angeles.
30. Jägstam M. and Klingstam P., 2002. *A handbook for integrating discrete event simulation as an aid in conceptual design of manufacturing systems* Proceedings of the 2002 Winter Simulation Conference
31. Gilbert N., 1995, *Simulation: an emergent perspective, conference,* New Technologies in the Social Sciences Bournemouth, UK
32. Brambilla M., 2005. Distrimobs, simulatore di mobilita personale: studio e progettazione dell'architettura adattiva e distribuita su cluster di blades, Facolta di Scienze Matematiche, Fisiche e Naturali, Universita di Bologna
33. Klie T. and Schalong C., 2002. A script-driven Simulation Tool as an Improvement to the Education in Web-based Virtual Learning Environments, Computer Science, and Information Management, Braunschweig.
34. Eckel, B. 2003. *Thinking in Java, Third Edition*, Prentice Hall, New Jersey, pp 264 - 270
35. Harold E. R., 2002 *Processing XML with Java, A Guide to SAX, DOM, JDOM, JAXP and TRax*, Addison-Wesley
36. Schlosser E, 2001, *Fast Food Nation: The Dark Side of the All-American Meal*, Houghton Mifflin, Boston, Massachusetts

ΔΙΕΥΘΥΝΣΕΙΣ ΣΤΟ ΔΙΑΔΙΚΤΥΟ

1. http://java.sun.com/j2se/1.4.2/docs/api/
2. http://java.sun.com/j2se/1.4.2
3. http://www.jcreator.com
4. www.jdom.org
5. http://www.w3.org
6. http://www.w3.org/xml
7. http://www.jdom.org/docs/apidocs/index.html Sun Microsystems, Inc. XML Processing, 2001.
8. http://java.sun.com/j2se/1.4/docs/guide/xml/jaxp/index.html
9. http://www.verisign.ch
10. http://vam.anest.ufl.edu/simulations/simulationportfolio.php
11. http://user.it.uu.se/~psa/
12. http://www.scitrav.com/wwater/sasspro/sasspro.asp
13. http://www.imaginethatinc.com
14. http://www.simul8.com