ΠΡΟΣΟΜΟΙΩΣΗ ΚΑΤΑΝΕΜΗΜΕΝΟΥ ΣΥΣΤΗΜΑΤΟΣ ΑΝΕΥΡΕΣΗΣ ΑΡΧΕΙΩΝ

2η εργασία για το μάθημα Κατανεμημένα Συστήματα & Διαδίκτυο


```
/**
 * Αυγουστάκης Χρυσοβαλάντης (880)
 * Κρητικός Απόστολος (914)
 * Φιλίππου Γεώργιος (1236)
 */
```

Περιεχόμενα

Περιγραφή του προβλήματος

Η παρούσα ακαδημαϊκή εργασία στοχεύει στην υλοποίηση ενός κατανεμημένου συστήματος ανεύρεσης αρχείων. Αυτό αποτελείται από 10 συνδεδεμένα κατανεμημένα μέρη (peers) που δέχονται αιτήματα για αρχεία και τα αναζητούν στα γειτονικά τους ομότιμα μέρη[1]. Το πλήθος των γειτόνων για κάθε ένα απ' αυτά τα μέρη είναι 1 γείτονας τουλάχιστον και μέγιστο 3 γείτονες.

Τα κατανεμημένα μέρη εξυπηρετούν αιτήματα αρχείων[2]. Αρχικά αναζητούν το αρχείο στην εσωτερική βάση αρχείων τους και αν το βρουν ικανοποιούν το αίτημα. Διαφορετικά διαδίδουν το αίτημα στους γείτονες τους. Αυτοί με τη σειρά τους επαναλαμβάνουν την παραπάνω διαδικασία μέχρι να βρεθεί ένας ομότιμος που κατέχει το αρχείο ή να γίνει υπέρβαση ενός ορίου αναμετάδοσης / διάδοσης του αιτήματος. Στην πρώτη περίπτωση εκκινείται μια διαδικασία μεταφόρτωσης[3] (downloading), ενώ στη δεύτερη ο χρήστης πληροφορείται την αποτυχία εύρεσης του αρχείου.

[1] *Τα κατανεμημένα μέρη θα ονομάζονται κατά σύμβαση «ομότιμα μέρη» ή «ομότιμοι».*
[2] *Ως «αρχεία» θα αναφέρονται τα εικονικά αρχεία που χειρίζεται η εφαρμογή.*
[3] *Ο όρος αυτός μπορεί να αναφέρεται στο κείμενο και ως «κατέβασμα».*

Η δική μας σκοπιά

Η φύση ενός κατανεμημένου συστήματος επιβάλλει την ταυτόχρονη λειτουργία των ομοτίμων μερών απ' τα οποία αποτελείται. Στην εφαρμογή μας λοιπόν επιλέξαμε να αντιστοιχήσουμε κάθε ομότιμο μέρος σε ένα νήμα. Έτσι λοιπόν κάθε ένα απ' αυτά τα νήματα προσομοιώνει τη λειτουργία ενός ομότιμου μέρους.

Από την περιγραφή του προβλήματος προκύπτει το θέμα της γειτνίασης μεταξύ των ομοτίμων. Δηλαδή θα πρέπει για κάθε μια απ' αυτές τις οντότητες να καθοριστεί ένα πλήθος από άλλες οντότητες με τις οποίες θα συνδέεται με σχέσεις γειτνίασης. Αυτομάτως τίθενται κάποιοι περιορισμοί:

- Η σχέση γειτνίασης πρέπει να είναι αμφίδρομη. Δηλαδή όταν ένα μέρος θεωρεί ένα άλλο μέρος γείτονα του, τότε και το δεύτερο μέρος θα πρέπει να θεωρεί το πρώτο γείτονα του.
- Το πλήθος των γειτόνων για κάθε ομότιμο μέρος πρέπει να είναι σαφώς καθορισμένο. Στην συγκεκριμένη περίπτωση έχει καθοριστεί ότι το πλήθος αυτό θα βρίσκεται στο (κλειστό) διάστημα [1,3].

Θα πρέπει να οριστεί ένας αλγόριθμος ο οποίος να λαμβάνει υπόψη τα παραπάνω και να δημιουργεί τις σχέσεις γειτνίασης. Η περιγραφή του αλγορίθμου που αναπτύχθηκε για την εφαρμογή μας γίνεται παρακάτω.

Θέλοντας να επιτύχουμε ένα μεγαλύτερο επίπεδο ρεαλισμού για την προσομοίωση μας, αποφασίσαμε να βασιστούμε στο πρότυπο ανταλλαγής μηνυμάτων για την επικοινωνία μεταξύ των ομοτίμων (μοντέλο Request – Reply / Client – Server). Ως εκ τούτου ορίσαμε το σύνολο των μηνυμάτων που ανταλλάσσονται μεταξύ των ομοτίμων καθώς και μια δομή μοναδική για κάθε ομότιμο που χρησιμεύει ως μέσο εναποθήκευσης των μηνυμάτων. Περισσότερα για τα μηνύματα και τη δομή εναποθήκευσης αναφέρονται σε επόμενες ενότητες.

Είναι προφανές ότι κάθε ομότιμο μέρος οφείλει να κατέχει τη δική του βάση αρχείων. Η βάση αυτή περιέχει, ανά πάσα στιγμή, τα αρχεία του ομότιμου και επεκτείνεται κάθε φορά που γίνεται λήψη ενός νέου αρχείου. Το πρόβλημα καθορίζει σαφέστατα ότι η βάση μόνο θα επεκτείνεται, οπότε δεν προσφέρονται περαιτέρω λειτουργίες όπως λόγου χάρη η διαγραφή αρχείων (βλέπε παρακάτω).

Δεδομένων των όσων αναφέραμε παραπάνω για τις βάσεις των αρχείων προκύπτει το θέμα της αρχικοποίησης των βάσεων των ομοτίμων. Έτσι λοιπόν υπάρχει μια οντότητα – γεννήτρια που παράγει αρχεία και τα κατανέμει τυχαία στους ομότιμους.

Τελειώνοντας τη σύντομη περιγραφή των βασικών «δομικών λίθων» της εφαρμογής θα πρέπει να αναφερθούμε στο γραφικό περιβάλλον. Σκοπός του είναι αφενός να συλλέξει πληροφορίες απ’ τον χρήστη που θα καθορίσουν τη λειτουργία της εφαρμογής και αφετέρου να εμφανίσει στον χρήστη αρκετές πληροφορίες, ώστε να κατανοήσει ο τελευταίος τις εσωτερικές λειτουργίες της εφαρμογής, τις οποίες δεν μπορεί να αντιληφθεί με διαφορετικό τρόπο.

Έτσι λοιπόν δημιουργήσαμε τρεις διεπαφές χρήστη. Η πρώτη χρησιμοποιείται για την αρχικοποίηση του συστήματος, καλώντας τον χρήστη να καθορίσει τον αριθμό των αρχείων που θα περιέχει αρχικά το κάθε ομότιμο μέρος. Η δεύτερη αποτελεί ένα παράθυρο στο οποίο εμφανίζεται ένας γράφος που αναπαριστά τις σχέσεις γειτνίασης των ομοτίμων. Η τρίτη είναι η βασική διεπαφή αλληλεπίδρασης με τον χρήστη. Σε αυτή μπορεί να αναζητήσει συγκεκριμένα αρχεία σε κάποιο συγκεκριμένο ομότιμο μέρος και να παρακολουθήσει την εξέλιξη του αιτήματος του. Είναι σημαντικό να τονίσουμε ότι μπορεί να παρακολουθήσει περισσότερα του ενός αιτήματα να εξελίσσονται ταυτόχρονα.

Σχήμα ονοματολογίας του συστήματος

Όταν αναφερόμαστε στο σχήμα ονοματολογίας της εφαρμογής μας, ουσιαστικά μιλάμε για ένα σύνολο κανόνων σχετικών με τη σύνταξη μιας εντολής του χρήστη προς το σύστημα. Η μοναδική εντολή που μπορεί να δώσει ο χρήστης στο σύστημα είναι ένα αίτημα εύρεσης ενός αρχείου από ένα συγκεκριμένο ομότιμο μέρος.

Σε ένα ρεαλιστικό σύστημα θα υπήρχε χρήση πρωτοκόλλων καθώς επίσης και σαφής προσδιορισμός μιας σειράς από καταλόγους για την εύρεση ενός αρχείου, γεγονός που καθιστά απαραίτητη τη χρήση ενός σχήματος ονοματολογίας. Στο σύστημα μας όμως δεν χρησιμοποιούνται πρωτόκολλα επικοινωνίας, ούτε υποστηρίζεται η έννοια του καταλόγου και για το λόγο αυτό το σχήμα ονοματολογίας είναι πολύ απλό.

Καταρχάς ο χρήστης πρέπει να προσδιορίσει τον ομότιμο στον οποίο θέλει να απευθύνει το αίτημα. Αυτό γίνεται ξεκινώντας την εντολή του με τη χρήση του - peer<αριθμός> -. Αφού έχει καθορίσει τον ομότιμο πρέπει να καθορίσει και το όνομα του αρχείου που τον ενδιαφέρει. Χρησιμοποιεί την πλάγια γραμμή '/' ως διαχωριστικό και στη συνέχεια γράφει το όνομα του αρχείου. Ο χρήστης δηλαδή γράφει εντολές τις μορφής - *peer<αριθμός>/<όνομα αρχείου>* -.

Το σύστημα εσωτερικά θα αντιστοιχίσει το *<αριθμός>* σε ένα αναγνωριστικό ενός ομότιμου. Θα δημιουργήσει ένα νέο αίτημα για το αρχείο που προσδιορίζεται απ' το *<όνομα αρχείου>* και θα το αποστείλει στον ομότιμο που αντιστοιχεί στο αναγνωριστικό που παρήγαγε.

Εάν η σύνταξη της εντολής δεν ακολουθεί την παραπάνω μορφή, τότε το σύστημα θα ενημερώσει τον χρήστη ότι έκανε λάθος. Εάν ο χρήστης προσδιορίσει έναν ομότιμο που δεν υπάρχει το σύστημα πάλι θα τον ενημερώσει για το λάθος του. Για όνομα αρχείου ο χρήστης μπορεί να δώσει οποιοδήποτε όνομα επιθυμεί αρκεί να μην περιέχει λευκούς χαρακτήρες.

Θέματα υλοποίησης και αρχιτεκτονική του έργου

Εν γένει, οι κλάσεις που συμμετέχουν στην υλοποίηση του συστήματος μπορούν να χωριστούν σε 3 κατηγορίες.

- Πυρήνας συστήματος
- Διεπαφή χρήστη (GUI)
- Απεικόνιση των ομότιμων σε δίκτυο (γράφος).

Παρακάτω θα περιγράψουμε τις κλάσεις του συστήματος ανά κατηγορία, ακολουθώντας αυστηρά την παραπάνω σειρά.

Η κλάση File

Δημιουργεί εικονικά αρχεία, τα οποία περιγράφονται από το όνομά τους (fileName) και το μέγεθός τους (fileSize – σε KB).

Η κλάση FileGenerator

Αποτελεί μία γεννήτρια αρχείων (τύπου File). Σαν ορίσματα δέχεται τον αριθμό των αρχείων που επιθυμούμε να παραχθούν και τον πίνακα που περιέχει τα ομότιμα μέρη.

Η βασική μέθοδος της συγκεκριμένης κλάσης είναι η `distributeFilesToPeers()` που αναλαμβάνει να δημιουργήσει (χρησιμοποιώντας τη μέθοδο `createFile(final int fileNumber)`) και να διανείμει τα εικονικά αρχεία στα ομότιμα μέρη (ανάλογα με το πόσα αρχεία πρέπει να έχει το κάθε ένα όπως ορίστηκε από τον χρήστη κατά την αρχικοποίηση του συστήματος).

Επίσης υποστηρίζεται η μέθοδος `generateRandomFileName()` η οποία επιστρέφει ένα τυχαίο όνομα αρχείου ώστε να μπορεί να χρησιμοποιηθεί από τη γεννήτρια παραγωγής αιτημάτων. Θεωρήσαμε πιο ρεαλιστικό να καθορίσουμε και μια πιθανότητα μη εγκυρότητας του αρχείου που παράγεται, η οποία είναι της τάξεως του 2%.

Η κλάση MsgType (enumeration)

Προσδιορίζει τον τύπο του μηνύματος σε:

REQUEST_FOR_FILE	Αίτηση που γίνεται προς ένα ομότιμο μέρος για λήψη ενός αρχείου.
FILE_FOUND	Το ομότιμο μέρος που ζήτησε το αρχείο πληροφορείται από το ομότιμο μέρος που το κατέχει ότι βρέθηκε.
UNABLE_TO_FIND_FILE	Το ομότιμο μέρος που ζήτησε το αρχείο πληροφορείται ότι δεν βρέθηκε. Προηγήθηκε διάδοση του μηνύματος βάθους 10 επιπέδων.

START_UPLOADING	Ζητείται από τον ομότιμο που λαμβάνει το μήνυμα να αρχίσει το ανέβασμα του αρχείου*.

**Όταν γίνεται η διάδοση ενός αιτήματος μπορεί περισσότεροι του ενός ομότιμοι να κατέχουν το αρχείο αυτό. Ο αρχικός αποστολέας του αιτήματος αποφασίζει από ποιον θα κατεβάσει το αρχείο και του αποστέλλει μήνυμα του συγκεκριμένου τύπου. Έτσι εξασφαλίζεται ότι κάθε ομότιμος θα κατεβάσει το αρχείο μόνο μια φορά.*

Η κλάση Message

Υλοποιεί ένα μήνυμα που απευθύνεται από ένα ομότιμο μέρος σε ένα άλλο.

Εν γένει, ο δημιουργός, δέχεται σαν ορίσματα, το αρχικό ομότιμο μέρος που μετέδωσε το μήνυμα, το ομότιμο μέρος στο οποίο μεταδόθηκε τελευταία το μήνυμα, το είδος του μηνύματος (που είναι τύπου MsgType) και το όνομα του αρχείου το οποίο αφορά το μήνυμα. Εναλλακτικά προσφέρεται ένας δεύτερος δημιουργός ο οποίος λαμβάνει υπόψη του (εκτός των άλλων) και το πλήθος των ως τώρα μεταδόσεων σε γείτονες. Αυτό βοηθάει στο να μπορεί το περιβάλλον να γνωρίζει πότε έχουν συμβεί οι μέγιστες επιτρεπτές μεταδόσεις και να μην επιδιώξει άλλες.

Η κλάση περιλαμβάνει επίσης μεθόδους για την αυξομείωση του πλήθους μεταδόσεων του μηνύματος, καθώς και την επιστροφή ή τον καθορισμό των τιμών των χαρακτηριστικών της.

Η κλάση Peer (νήμα)

Προσομοιώνει ένα ομότιμο μέρος (peer), ενώ κάθε στιγμιότυπό της αποτελεί ένα νήμα για το σύστημα.

Κύρια λειτουργία του κάθε ομότιμου είναι να εξάγει αιτήματα απ' την ουρά μηνυμάτων του και να τα εξυπηρετεί. Αναλυτικότερα στην βασική μέθοδο της κλάσης (run – πυροδότηση του νήματος –) το ομότιμο μέρος ανασύρει το επόμενο κατά σειρά μήνυμα, ελέγχει τον τύπο του και στη συνέχεια προχωράει στη διεκπεραίωσή του με την αντίστοιχη μέθοδο.

Παρακάτω παρατίθεται ο πίνακας με τους τύπους μηνυμάτων όπως αυτοί αναφέρθηκαν παραπάνω και τις αντίστοιχες ενέργειες – μεθόδους με τις οποίες ανταποκρίνεται ο ομότιμος.

ΤΥΠΟΣ ΜΗΝΥΜΑΤΟΣ	ΑΝΤΙΣΤΟΙΧΗ ΜΕΘΟΔΟΣ PEER

REQUEST_FOR_FILE	**searchFile**(Message request)
FILE_FOUND	**startDownloading**(Message request)
UNABLE_TO_FIND_FILE	**searchFailed**(String fileName)
START_UPLOADING	**startUploading**(Peer destination, String fileName)

Ας δούμε όμως και λίγο αναλυτικότερα τη λειτουργία των προαναφερθέντων μεθόδων:

- **`searchFile`**`(Message request)`: Αναλαμβάνει να αναζητήσει το αρχείο που αναφέρεται στο μήνυμα request. Αν το αρχείο βρεθεί, στέλνεται μήνυμα στο ομότιμο μέρος που ζήτησε το αρχείο (στον αρχικό δηλαδή μεταδότη) και ξεκινά η διαδικασία διάθεσης (upload). Αν το αρχείο δεν βρεθεί το σύστημα ελέγχει αν μπορεί να γίνει αναμετάδοση του μηνύματος σε γείτονες. Αν αυτό δεν είναι δυνατό, τότε στέλνεται ένα μήνυμα στον αρχικό μεταδότη και τον ενημερώνει ότι το αρχείο δεν μπόρεσε να εντοπιστεί. Σε αντίθετη περίπτωση (αν δηλαδή μπορεί να γίνει αναμετάδοση), τότε εντοπίζονται οι γείτονες του τρέχοντος ομότιμου μέρους και εφόσον αυτοί δεν είναι ο αρχικός μεταδότης ή ο μεταδότης από τον οποίο το αίτημα έφθασε στον τρέχοντα, γίνεται μετάδοση του μηνύματος.

- **`startDownloading`**`(Message request)`: Με την μέθοδο αυτή ο ομότιμος που ζήτησε το αρχείο, αρχίζει να το μεταφορτώνει (downloading) αφού πρώτα έχει στείλει μήνυμα στον ομότιμο που το κατέχει να το διαθέσει (uploading). Η μέθοδος αυτή εκτός των άλλων βοηθά στο να αποφευχθεί ένα conflict το οποίο οφείλεται στη λειτουργία της γεννήτριας παραγωγής αυτόματων αιτημάτων. Έτσι όταν αρχίζει η μεταφόρτωση ενός αρχείου σε ένα ομότιμο μέρος, τοποθετείται στην βάση των αρχείων του, στη θέση που πρόκειται να μπει το αρχείο που μεταφορτώνεται, ένα κενό αρχείο ώστε να αγνοηθεί μια νέα αυτόματη παραγωγή αιτήματος για αυτό το αρχείο, στο συγκεκριμένο ομότιμο μέρος, αφού σε λίγο πρόκειται να το διαθέτει.

- **`startUploading`**`(Peer destination, String fileName)`: Ξεκινά τη διαδικασία διάθεσης ενός αντίγραφου του αρχείου που ζητήθηκε προς το ομότιμο μέρος που το ζήτησε.

- **`searchFailed`**`(String fileName)`: Ενημερώνει το ομότιμο μέρος στο οποίο απευθύνεται ότι η αναζήτηση για το αρχείο με όνομα fileName απέτυχε.

Εκτός των προαναφερθέντων η κλάση Peer υποστηρίζει και κάποιες επιπλέον μεθόδους για την εισαγωγή νέων μηνυμάτων στην ουρά μηνυμάτων και νέων αρχείων στην βάση αρχείων. Οι υλοποιήσεις είναι απλές και κρίνεται περιττή η περαιτέρω ανάλυσή τους.

Πίνακες κατακερματισμού

Κάθε ομότιμος διαθέτει μια δομή στην οποία να αποθηκεύει τα αρχεία του. Αυτή η βάση αρχείων υλοποιείται με τη χρήση ενός πίνακα κατακερματισμού (hash table). Η επιλογή της συγκεκριμένης δομής έγκειται στην ταχύτερη επιστροφή αποτελεσμάτων, είτε πρόκειται για μια επιτυχία εύρεσης ενός αρχείου είτε για μια αποτυχία.

Ουρές

Η χρήση μηνυμάτων για την επικοινωνία των ομοτίμων μεταξύ τους απαιτεί ένα μέσο εναπόθεσης των μηνυμάτων αυτών. Έτσι κάθε οντότητα μπορεί να εξάγει ένα μήνυμα – αίτημα κάθε φορά από το μέσο αυτό για να το εξυπηρετήσει.

Επειδή χρησιμοποιούνται νήματα για να προσομοιώσουν τη λειτουργία των μερών και του εξυπηρέτη, προκύπτει ένα πρόβλημα συγχρονισμού κατά την πρόσβαση των ουρών τους για εγγραφή ή ανάγνωση ενός μηνύματος. Το πρόβλημα αυτό λύεται με τη χρήση συγχρονισμένων ουρών.

Η κλάση AdjacencyInfo

Καταγράφει τις συνδέσεις μεταξύ των ομότιμων μερών. Περιλαμβάνει 2 χαρακτηριστικά. Το peerIndex (ακέραιος) είναι το αναγνωριστικό του ομότιμου μέρους (κατά σύμβαση αύξοντας αριθμός) και το neighbours (συνδεδεμένη λίστα ακεραίων) το οποίο κράτα τα αναγνωριστικά των ομότιμων μερών με τα οποία συνδέεται το εκάστοτε ομότιμο μέρος.

Η κλάση AdjacencyGenerator

Αναλαμβάνει να κατασκευάσει το τυχαίο δίκτυο των ομότιμων μερών (για τις ανάγκες της προσομοίωσης). Επειδή αν χρησιμοποιήσουμε τυχαία ανάθεση γειτόνων σε όλους τους κόμβους (ομότιμα μέρη) υπάρχει περίπτωση να προκύψει μη συνδεδεμένος γράφος, σχεδιάσαμε έναν αλγόριθμο ώστε να αποφύγουμε αυτήν την ανεπιθύμητη εξέλιξη.

Ο συγκεκριμένος αλγόριθμός γνωρίζει ανά πάσα στιγμή ποιος είναι ο τρέχον και ο επόμενος διαθέσιμος κόμβος. Τρέχον κόμβος χαρακτηρίζεται

ο εκάστοτε επιλεγμένος (ξεκινώντας από τον πρώτο που έχει αναγνωριστικό 0). Επόμενος διαθέσιμος χαρακτηρίζεται ο επόμενος κόμβος που βρίσκεται μετά τον επιλεγμένο και δεν έχει κανένα γείτονα.

Με αυτά τα δεδομένα παράγεται για εκάστοτε τρέχοντα κόμβο ένας τυχαίος ακέραιος ο οποίος αποτελεί τον επιθυμητό αριθμό γειτόνων. Στη συνέχεια πραγματοποιούνται συνδέσεις του τρέχοντα κόμβου με τον επόμενο διαθέσιμο και τους επόμενούς αυτού, ίσες με τον επιθυμητό αριθμό γειτόνων που παρήχθη παραπάνω.

Στη συνέχεια παρουσιάζουμε τον αλγόριθμο σε μορφή βημάτων με σκοπό την καλύτερη εποπτεία από τον αναγνώστη.

Βήμα 1^{ο}: Θέτουμε *τρέχοντα κόμβο* το ομότιμο μέρος με αναγνωριστικό 0 και *επόμενο διαθέσιμο* το ομότιμο μέρος με αναγνωριστικό 1.

Βήμα 2^{ο}: Παράγουμε έναν τυχαίο ακέραιο μεταξύ 1[1] και του 3 που αντιπροσωπεύει τον *αριθμό γειτόνων* που θα έχει ο τρέχον κόμβος (στη συγκεκριμένη περίπτωση ο πρώτος με αναγνωριστικό 0).

Βήμα 3^{ο}: Για i από 1 ως *αριθμό γειτόνων* συνδέουμε τον *τρέχοντα κόμβο* με τον *επόμενο διαθέσιμο* ενώ σε κάθε βήμα αυξάνουμε τον *επόμενο διαθέσιμο* κατά 1.

Βήμα 4^{ο}: Αυξάνουμε τον *τρέχοντα κόμβο* κατά 1.

Βήμα 5^{ο}: Παράγουμε έναν τυχαίο ακέραιο μεταξύ 1 και 2[2] που αντιπροσωπεύει τον *αριθμό γειτόνων* που θα έχει ο *τρέχον κόμβος*. Κατόπιν επαναλαμβάνουμε τα βήματα 3 και 4.

Βήμα 6^{ο}: Επαναλαμβάνουμε τα βήματα 4-5 έως ότου ο *επόμενος διαθέσιμος κόμβος* ξεπεράσει το όριο (αναγνωριστικό 10).

Βήμα 7^{ο}: Απ' τις παραπάνω διαδικασίες δεν θα καθορίσουν όλοι των αριθμό των γειτόνων, οπότε για αυτούς που απομένουν επαναλαμβάνονται τα βήματα 3 και 4 για παραγωγές τυχαίων ακεραίων μεταξύ 0 και 1.

[1] Σημειώνουμε ότι σε αυτό το βήμα παράγεται αριθμός επιθυμητών γειτόνων μεταξύ 1 και 3 για να αποφευχθεί η περίπτωση όπου ο πρώτος κόμβος μπορεί να μείνει ασύνδετος.

[2] Καθορίζεται ως όριο το 2 και όχι το 3, επειδή γνωρίζουμε ότι ο κάθε ομότιμος έχει έναν απ' τους προηγουμένους γείτονα.

Η κλάση UploadingProcess (νήμα)

Η κλάση αυτή χρησιμοποιείται για να προσομοιώσει μια καθυστέρηση που παρατηρείται κατά τη μεταφόρτωση των αρχείων απ' το ένα ομότιμο μέρος στο άλλο. Λαμβάνει υπόψη του τους ρυθμούς ανεβάσματος και κατεβάσματος των δυο συμβαλλομένων ομοτίμων. Βάσει αυτών υπολογίζεται ο χρόνος που θα διαρκέσει η διαδικασία. Το νήμα δεν κάνει απολύτως τίποτα για αυτό το χρονικό διάστημα, αλλά μετά προσπαθεί να καταγράψει το αρχείο στον ομότιμο – προορισμό.

Η κλάση RandomRequestGenerator (νήμα)

Παράγει αυτόματα τυχαία αιτήματα για να έχουμε μια πιο ρεαλιστική προσομοίωση του κατανεμημένου συστήματος, στο οποίο συνήθως συμβαίνουν πολλά ερωτήματα ταυτόχρονα. Η βασική μέθοδός της κλάσης (run – πυροδότηση του νήματος –) δημιουργεί ένα τυχαίο όνομα αρχείου χρησιμοποιώντας ένα στιγμιότυπο της κλάσης FileGenerator και παράγοντας ένα τυχαίο ακέραιο επιλέγει ένα ομότιμο μέρος. Κατόπιν έχοντας τα δύο αυτά στοιχεία δημιουργεί ένα μήνυμα τύπου *REQUEST_FOR_FILE* και το αποστέλλει στο κατάλληλο ομότιμο μέρος. Κατόπιν το νήμα «κοιμάται» για ένα χρονικό διάστημα ώστε να μπορέσει το αίτημα να εξυπηρετηθεί ομαλά από το σύστημα.

Η κλάση Simulation

Η λειτουργία αυτής της κλάσης επικεντρώνεται στο συντονισμό και την οργάνωση του συστήματος. Αναλαμβάνει να δημιουργήσει όλα τα απαραίτητα στοιχεία (εικονικά αρχεία, ομότιμα μέρη, τις μεταξύ τους συνδέσεις) και να αρχικοποιήσει τις διεπαφές χρήστη (GUI). Επίσης διαχειρίζεται τα νήματα που έχουν δημιουργηθεί και τα συντονίζει με τις μεθόδους παύσης και διακοπής.

Η κλάση InitialWindow

Αποτελεί μια διεπαφή χρήστη (GUI), στην οποία καλούμαστε να δηλώσουμε πόσα αρχεία θέλουμε να κατέχει το κάθε ένα από τα 10 ομότιμα μέρη (κατά σύμβαση ο αριθμός των αρχείων δεν μπορεί να είναι μικρότερος από 10). Κατόπιν η κλάση καλώντας τις κατάλληλες μεθόδους παράγει και διανέμει τυχαία αρχεία στα ομότιμα μέρη και τέλος τα παρουσιάζει στον χρήστη με ένα πλαίσιο κειμένου.

Η κλάση BasicWindow

Αποτελεί την βασική διεπαφή χρήστη (GUI) του συστήματος. Περιλαμβάνει και αυτή 10 πλαίσια κειμένου στα οποία καταγράφεται η κίνηση στα αντίστοιχα ομότιμα μέρη. Επιτρέπει στο χρήστη να δημιουργήσει χειροκίνητα ένα αίτημα γράφοντος το στο μικρό πλαίσιο κειμένου (βάσει των κανόνων ονοματολογίας) και πατώντας το κουμπί *αναζήτηση* ή να εκκινήσει την παραγωγή αυτόματων αιτημάτων κάνοντας κλικ στο κουμπί *αυτόματα αιτήματα*.

Τα αυτόματα αιτήματα δεν ξεκινούν μέχρι ο χρήστης να πατήσει το ανάλογο κουμπί. Έτσι μπορεί να εισάγει τα δικά του αιτήματα και να παρακολουθήσει ευκολότερα την εξέλιξη τους, αφού δεν θα εξυπηρετούνται άλλα ταυτόχρονα με αυτά.

Η κλάση Graph

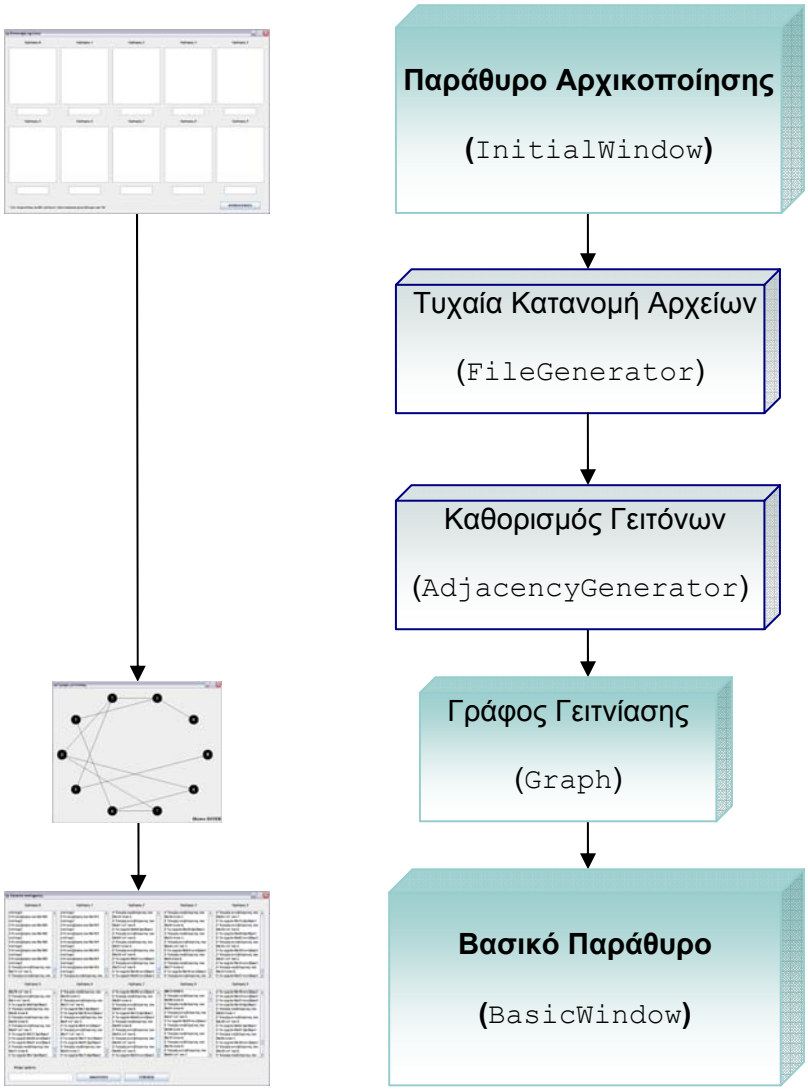
Παράθυρο μη διαδραστικό με το χρήστη. Η κλάση αυτή αποτελεί απλώς ένα εποπτικό εργαλείο για να μπορεί ο χρήστης της προσομοίωσης να δει το δίκτυο που έχει δημιουργηθεί μεταξύ των ομότιμων μερών. Για την απεικόνιση του δικτύου χρησιμοποιούνται αντικείμενα Graphics2D από την αντίστοιχη βιβλιοθήκη της Java.

Σχόλια

Στις απαιτήσεις του συστήματος προτείνονταν η αναμετάδοση του αιτήματος για ένα αρχείο να σταματά μετά την δεύτερη μετάδοση. Θέλοντας να προσεγγίσουμε περισσότερο την πραγματικότητα αυξήσαμε το πλήθος των μεταδόσεων σε 10. Έτσι λοιπόν υπάρχει περίπτωση ένας μεγάλος αριθμός αντιγράφων του αρχικού μηνύματος να κυκλοφορεί στο δίκτυο. Σε αυτήν την περίπτωση μια πιθανή αποτυχία προκαλεί την αποστολή πολλαπλών μηνυμάτων αποτυχίας στον αρχικό αποστολέα. Ο χρήστης μπορεί να αντιληφθεί το αντιληφθεί αυτό ως συνεχόμενα μηνύματα αποτυχίας στο αντίστοιχο πλαίσιο κειμένου.

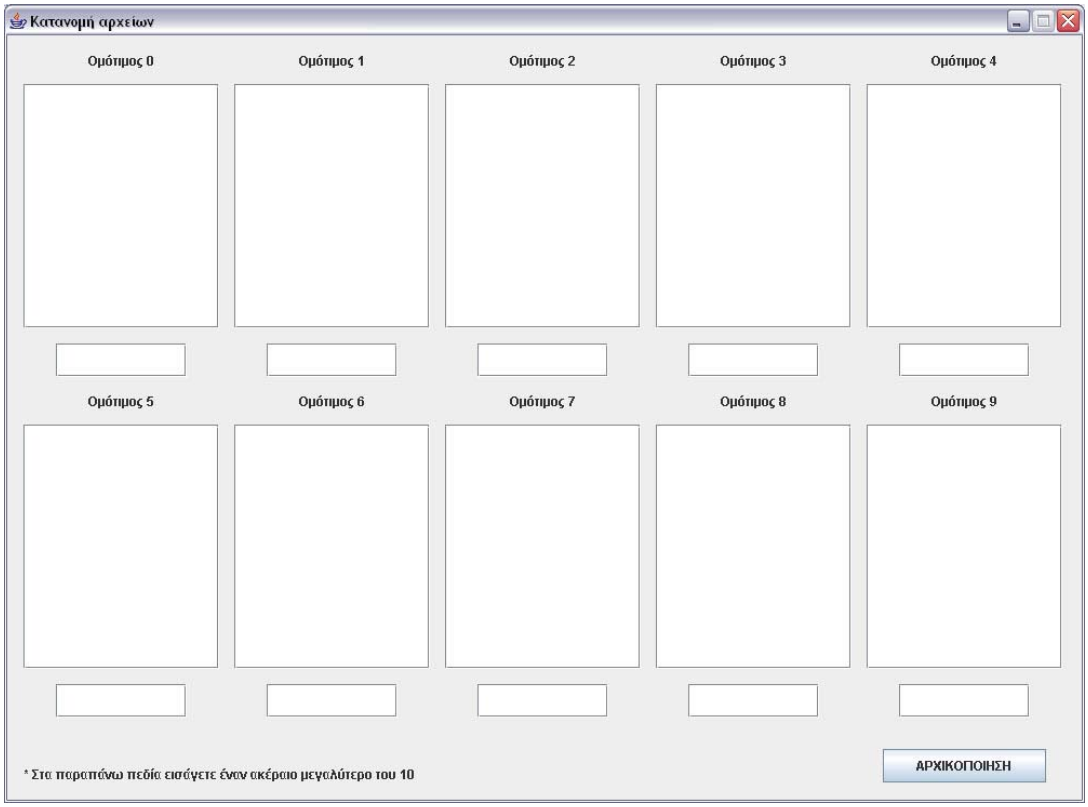
Περίπτωση χρήσης του συστήματος

Για να ξεκινήσει η εφαρμογή εκτελούμε το αρχείο Simulate.bat.



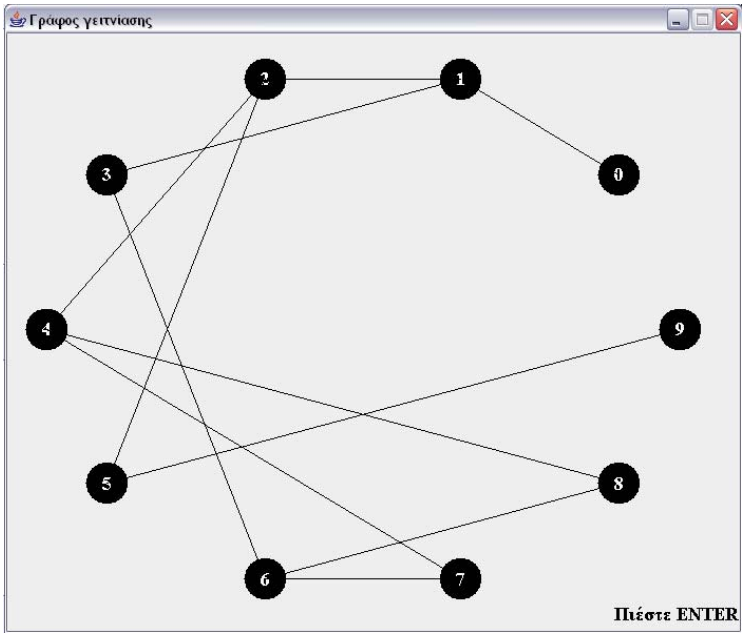
Σχήμα 1. *Η ροή ελέγχου της εφαρμογής.*

Εμφανίζεται το παράθυρο αρχικοποίησης (*Κατανομή αρχείων*) στο οποίο καλούμαστε να εισάγουμε το πλήθος των αρχείων που θέλουμε να κατέχει αρχικά το κάθε ομότιμο μέρος (κατά σύμβαση μεγαλύτερο του 10). Μετά από αυτό, γίνεται η τυχαία κατανομή των αρχείων στα ομότιμα μέρη. Η λειτουργία αυτή αποκρύπτεται από τον χρήστη. Τελικά εμφανίζονται στο ίδιο παράθυρο οι λίστες με τα αρχεία όπως αυτά έχουν κατανεμηθεί.



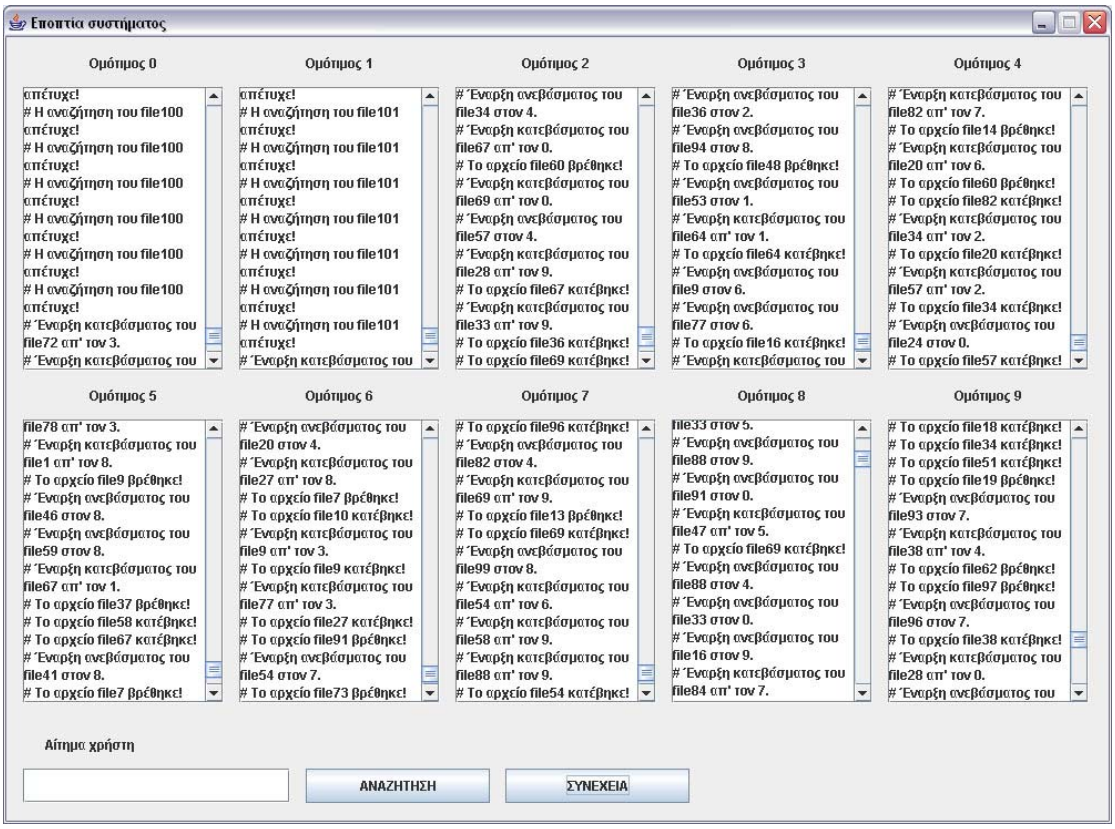
Σχήμα 2. *Το παράθυρο αρχικοποίησης.*

Ακολουθεί η δημιουργία του δικτύου των ομότιμων μερών. Η διαδικασία ανάθεσης γειτόνων γίνεται με βάση τον αλγόριθμο που αναλύθηκε παραπάνω και αποκρύπτεται από τον χρήστη. Αφότου το δίκτυο δημιουργηθεί, εμφανίζεται στον χρήστη ένα παράθυρο (*Γράφος γειτνίασης*) που περιέχει τον γράφο που το αναπαριστά.



Σχήμα 3. *Ένας γράφος γειτνίασης.*

Στο σημείο αυτό τελειώνει η αρχικοποίηση και ξεκινά η ουσιαστική λειτουργία του συστήματος.

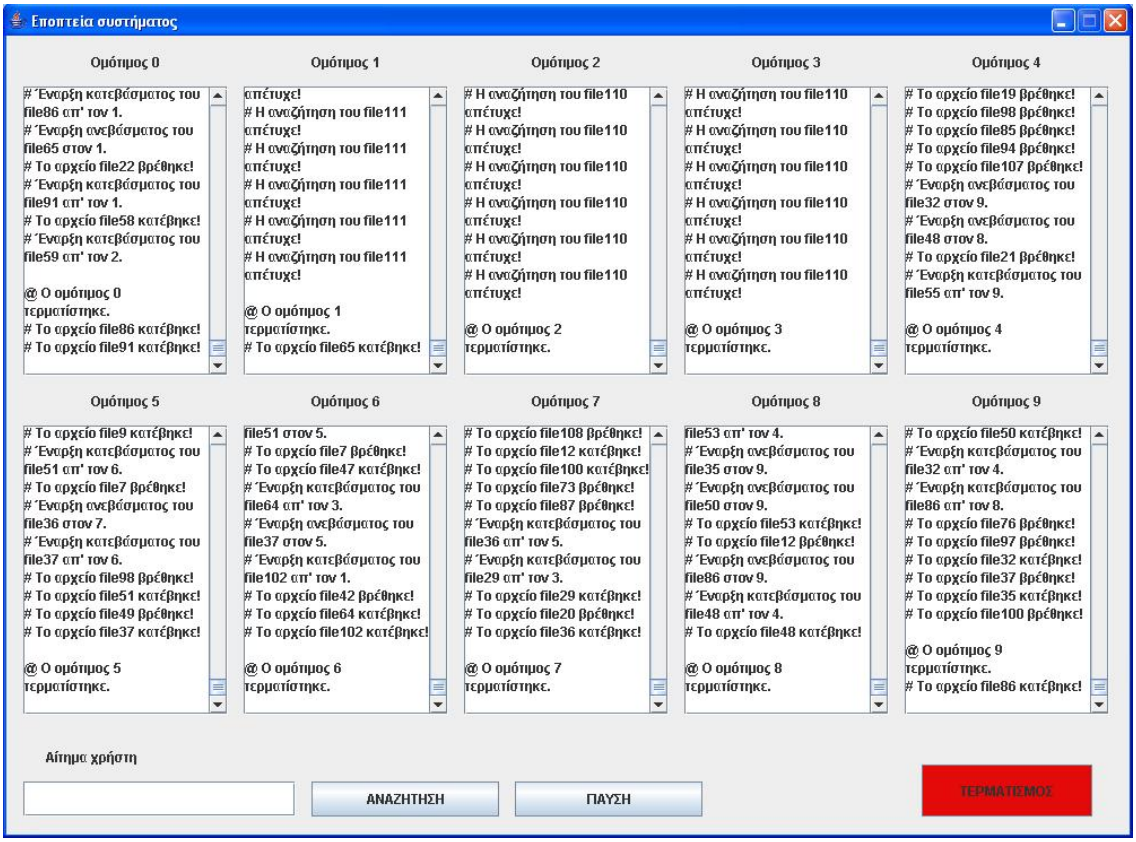


Σχήμα 4. *Το βασικό παράθυρο αλληλεπίδρασης με τον χρήστη.*

Υπάρχουν 2 τρόποι με τους οποίους μπορούμε να απευθύνουμε αιτήματα προς το σύστημα. Ο αυτόματος, κατά τον οποίο το σύστημα παράγει αιτήματα «κρυφά» από τον χρήστη και ο χειροκίνητος, κατά τον οποίο ο χρήστης παράγει ένα δικό του αίτημα που ακολουθεί το πρότυπο που ορίζεται στο σχήμα ονοματολογίας. Και στις δύο περιπτώσεις δημιουργείται το αντίστοιχο με το αίτημα μήνυμα και μπαίνει στην ουρά του κατάλληλου ομότιμου μέρους, ενώ κατάλληλο μήνυμα παρουσιάζεται στον χρήστη.

Από τη στιγμή που το πρώτο αίτημα έχει δημιουργηθεί, ξεκινά η διαδικασία διεκπεραίωσης των αιτημάτων από τα ομότιμα μέρη. Κάθε φορά που τελειώνει η επεξεργασία ενός αιτήματος (είτε αυτή ήταν επιτυχής είτε όχι) επιστρέφεται κατάλληλο μήνυμα στον χρήστη.

Λόγω του μεγάλου αριθμού των ομοτίμων μερών τα οποία δημιουργούν το κατανεμημένο σύστημα, θα ήταν εξαιρετικά χρονοβόρο να περιμένουμε την κατανομή όλων των αρχείων σε όλα τα ομότιμα μέρη. Έτσι αποφασίσαμε ότι το σύστημά μας θα τερματίζει όταν όλα τα ομότιμα μέρη κατέχουν τουλάχιστον το 50% του συνόλου των αρχείων*(σελ. 16).



Σχήμα 5. *Το βασικό παράθυρο μετά τον τερματισμό της προσομοίωσης.*

Μετά τον τερματισμό της προσομοίωσης παράγεται ένα αρχείο (*Output.txt*) της παρακάτω μορφής:

```
Ομότιμος 0:
 με ρυθμό ανεβάσματος <UPLOAD_RATE> KB/s
 και ρυθμό κατεβάσματος <DOWNLOAD_RATE> KB/s
 έλαβε <A> αιτήματα
 εξυπηρέτησε <C> αιτήματα
 κατέβασε <D> αρχεία
 αρχικά είχε <I> αρχεία
 και τερμάτισε με <F> αρχεία.
-----------------------------------------
Ομότιμος 1:
 με ρυθμό ανεβάσματος <UPLOAD_RATE1> KB/s
 και ρυθμό κατεβάσματος <DOWNLOAD_RATE1> KB/s
 έλαβε <A1> αιτήματα
 εξυπηρέτησε <C1> αιτήματα
 κατέβασε <D1> αρχεία
 αρχικά είχε <I1> αρχεία
 και τερμάτισε με <F1> αρχεία.
-----------------------------------------
```

```
.
.
.

Ομότιμος 9:
 με ρυθμό ανεβάσματος <UPLOAD_RATE9> KB/s
 και ρυθμό κατεβάσματος <DOWNLOAD_RATE9> KB/s
 έλαβε <A9> αιτήματα
 εξυπηρέτησε <C9> αιτήματα
 κατέβασε <D9> αρχεία
 αρχικά είχε <I9> αρχεία
 και τερμάτισε με <F9> αρχεία.
-----------------------------------------
```

Σχήμα 6. *Η μορφή των αποτελεσμάτων που εμφανίζονται στο αρχείο <Output.txt>.*

Γίνεται λοιπόν εμφανές ότι το αρχείο αυτό περιέχει μια αποτίμηση της προσομοίωσης. Είναι απαλλαγμένο από τα μηνύματα που περιγράφουν σχολαστικά την προσομοίωση και περιορίζεται απλά στο να αναφέρει τα φυσικά χαρακτηριστικά του κάθε ομότιμου (UPLOAD_RATE, DOWNLOAD_RATE), τα στατιστικά που αφορούν τα αιτήματα που ελήφθησαν από τον κάθε ομότιμο και πόσα τελικά εξυπηρετήθηκαν. Επίσης αναφέρεται ο αριθμός αρχείων που είχε αρχικά και τελικά ο κάθε ομότιμος.

Μετά την παραγωγή αυτού του αρχείου η εφαρμογή τερματίζεται.

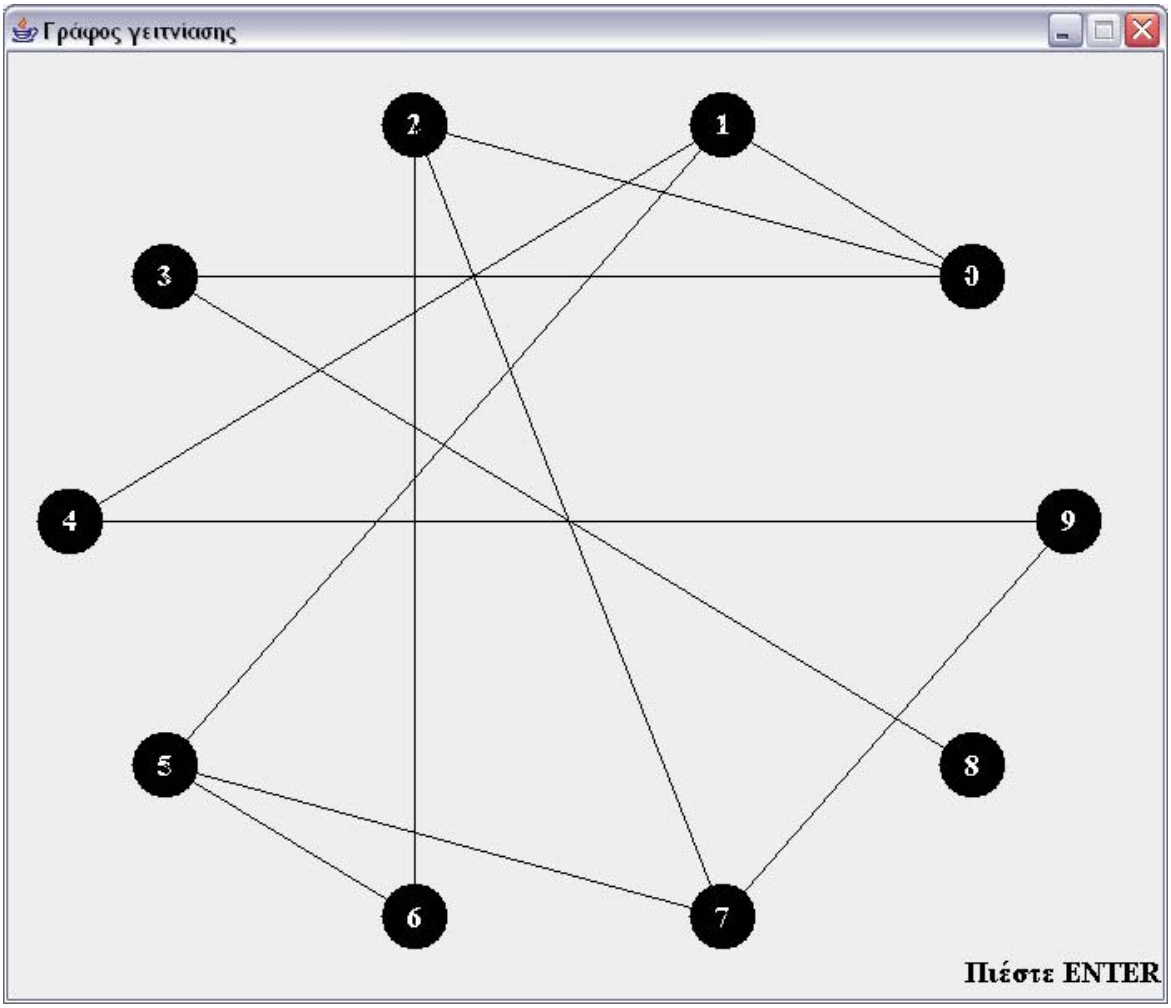
* *Όταν οι βάσεις αρχείων όλων των ομοτίμων συμπληρωθούν κατά αυτό το ποσοστό, ακόμη υπάρχουν αρχεία που κατεβαίνουν. Έτσι ο χρήστης μπορεί να δει στο βασικό παράθυρο να εμφανίζονται μηνύματα που τον πληροφορούν για το κατέβασμα ενός αρχείου και μετά τον τερματισμό της προσομοίωσης. Μια άλλη παρενέργεια παρατηρείται στο αρχείο εξόδου, όπου το άθροισμα του αρχικού αριθμού αρχείων και του αριθμού των αρχείων που κατέβασε ίσως να είναι μικρότερο από τον τελικό αριθμό των αρχείων (!). Η εξήγηση είναι η εξής: Όταν εκκινηθεί η διαδικασία κατεβάσματος θα ενημερωθεί ο μετρητής των αρχείων που κατέβηκαν αλλά ο μετρητής των τελικών αρχείων θα ενημερωθεί μόνο όταν τερματίσει η διαδικασία αυτή. Το γεγονός αυτό, σε συνδυασμό με το γεγονός ότι όταν τερματίζει η προσομοίωση εκκρεμούν διαδικασίες κατεβάσματος, που δεν προλαβαίνουν να τερματιστούν προτού γίνει η την εγγραφή των αποτελεσμάτων στο αρχείο εξόδου, δημιουργούν το παράδοξο που αναφέρθηκε.*

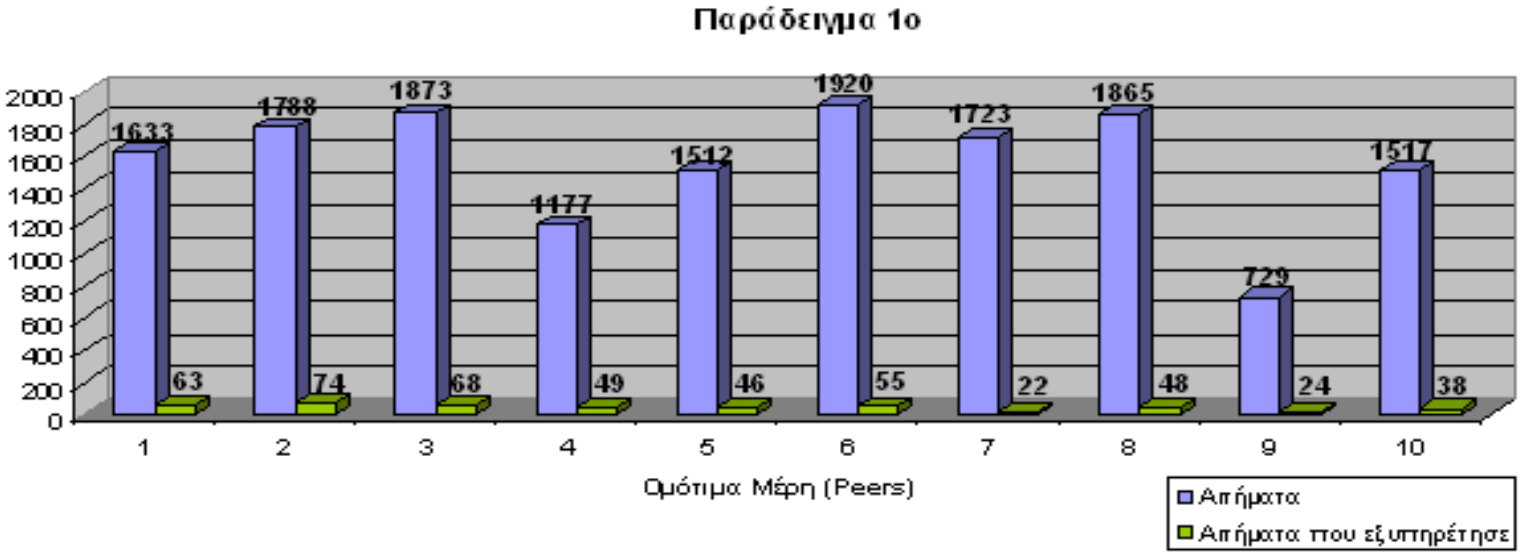
Παραδείγματα

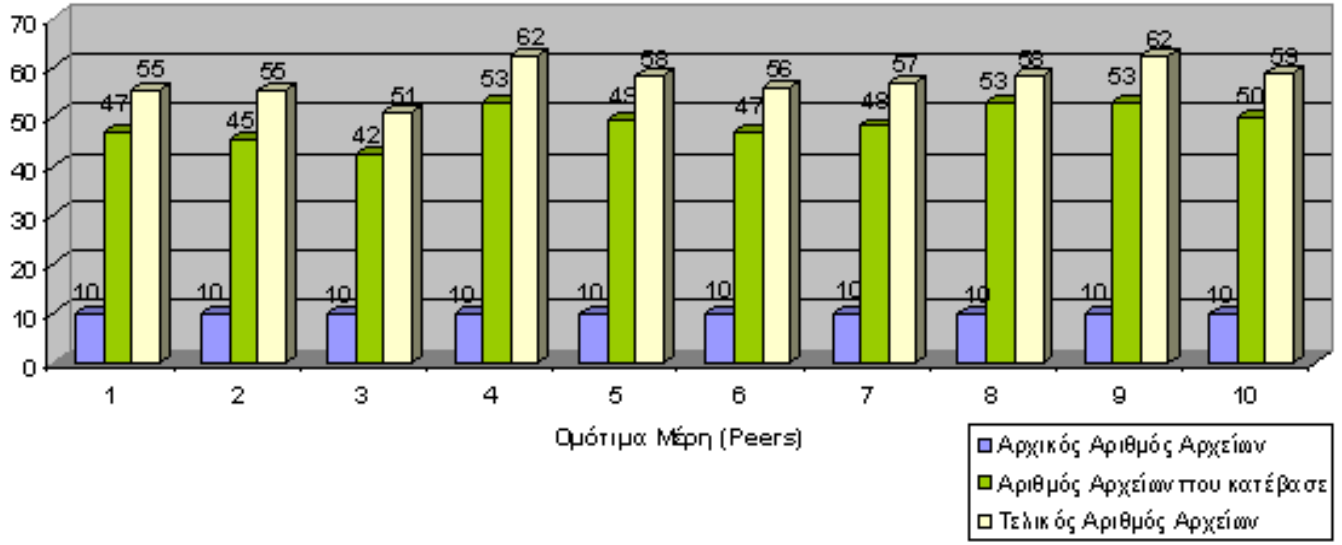
Ακολουθούν τρία παραδείγματα εκτέλεσης για τα οποία εμφανίζουμε το γράφο γειτνίασης και παρουσιάζουμε τα στατιστικά αποτελέσματα της κάθε εκτέλεσης.

Με βάση τα παραδείγματα αυτά συμπεράναμε τα εξής:
Κατά βάση οι ομότιμοι που είχαν τους περισσότερους γείτονες ήταν και αυτοί που δεχτήκαν και εξυπηρέτησαν τα περισσότερα αιτήματα. Οι όποιες ασυμφωνίες με το κανόνα αυτόν οφείλονται κυρίως στον αρχικό αριθμό αρχείων του εκάστοτε ομότιμου.
Επίσης αξίζει να αναφέρουμε πως στο τελευταίο παράδειγμα που οι ομότιμοι αρχικοποιήθηκαν με περισσότερα αρχεία παρατηρήθηκε πως η συνολική «κίνηση» στο δίκτυο ήταν μικρότερη. Αυτό ήταν αναμενόμενο μιας και ο κάθε ομότιμος έθετε λιγότερα αιτήματα αναζήτησης αρχείου στους γείτονες του.

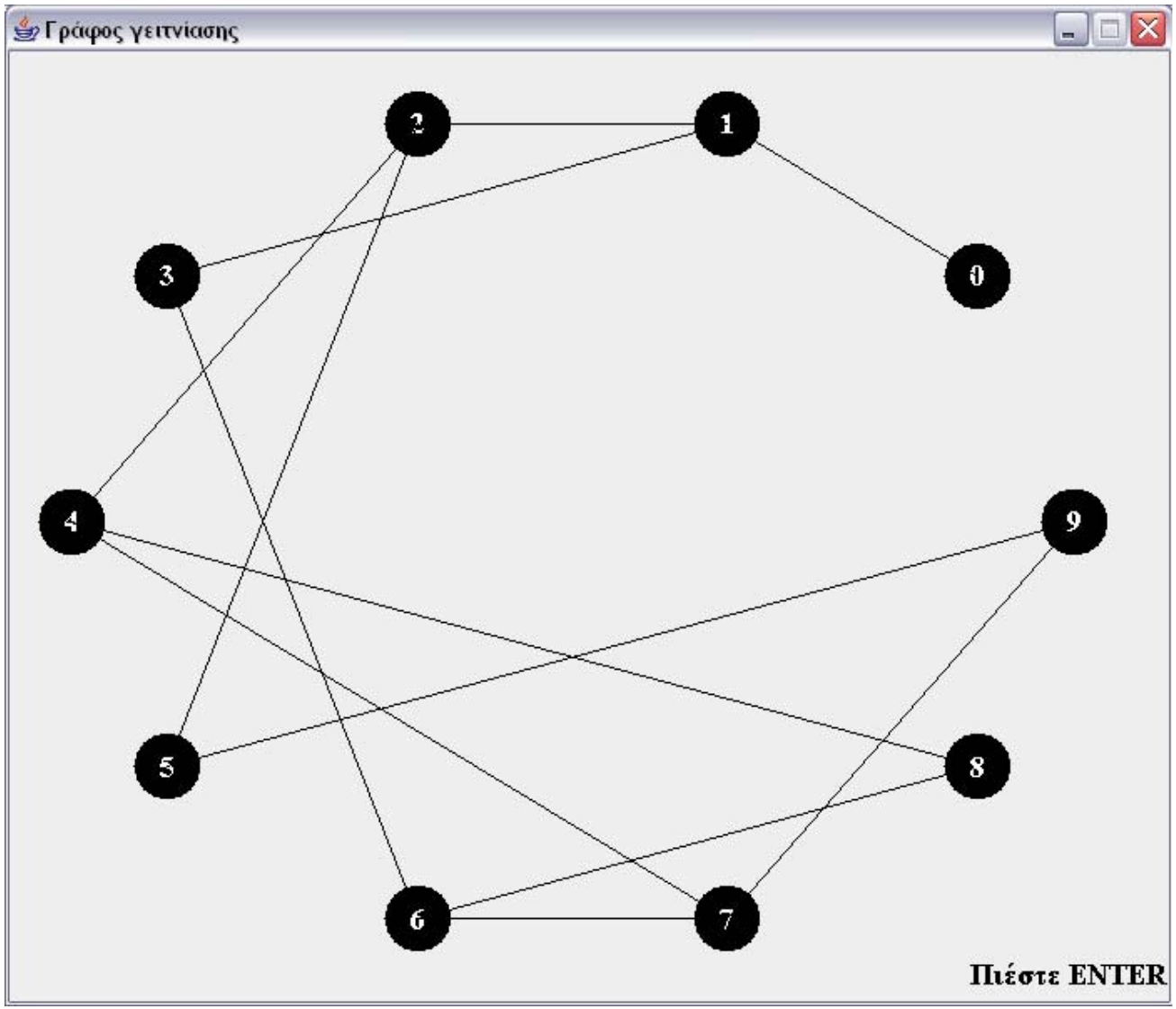
<u>Παράδειγμα 1ο</u>



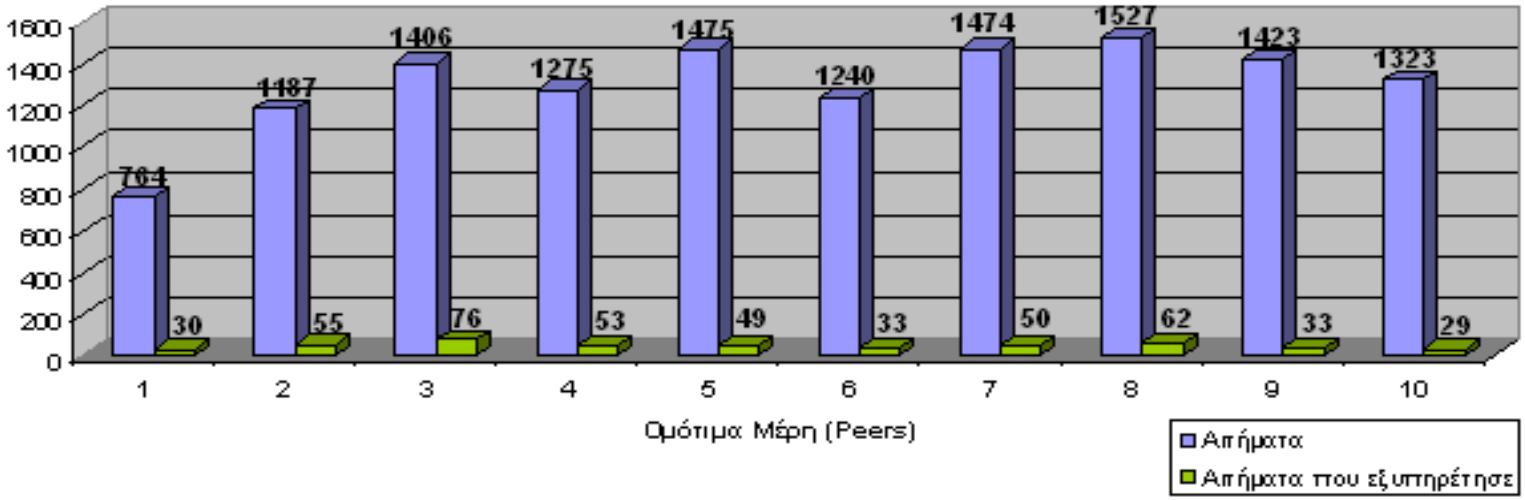

Παράδειγμα 1ο
2000
1800
1600
1400
1200
1000
800
600
400
200
0
1633
63
1788
74
1873
68
1477
49
1542
46
1920
55
1723
22
1865
48
729
24
1517
38
1
2
3
4
5
6
7
8
9
10
Ομότιμα Μέρη (Peers)
Απήματα
Απήματα που εξυπηρέτησε

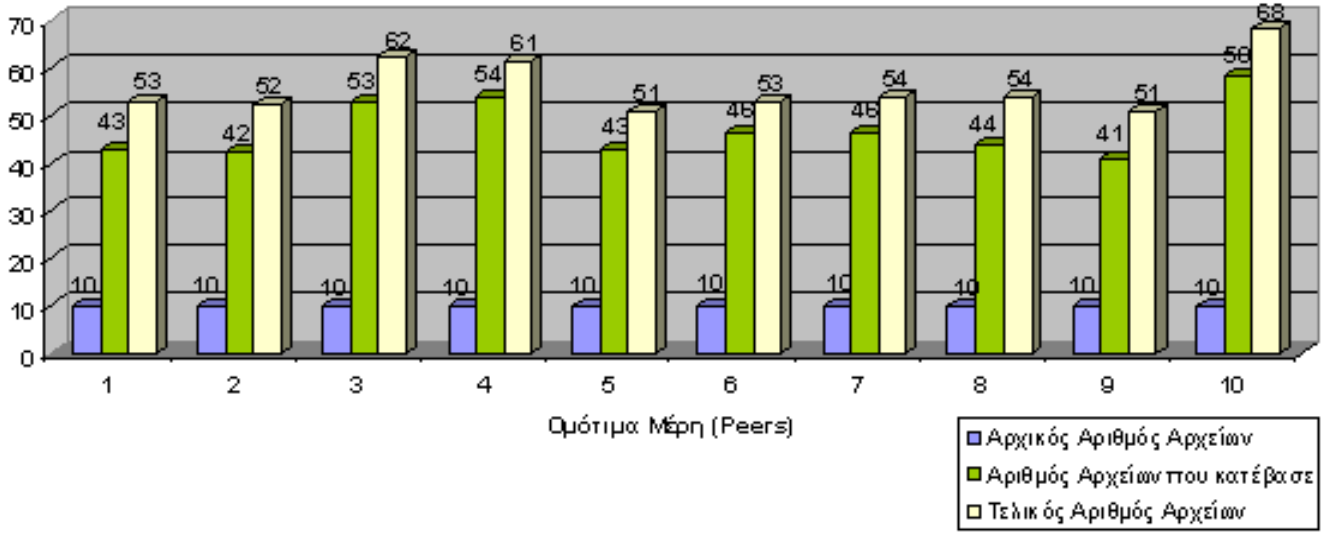

70
60
50
40
30
20
10
0
10
47
55
10
45
55
10
42
51
10
53
62
10
49
58
10
47
56
10
48
57
10
53
58
10
53
62
10
50
59
1
2
3
4
5
6
7
8
9
10
Ομότιμα Μέρη (Peers)
Αρχικός Αριθμός Αρχείων
Αριθμός Αρχείων που κατέβασε
Τελικός Αριθμός Αρχείων

Παράδειγμα 2ο

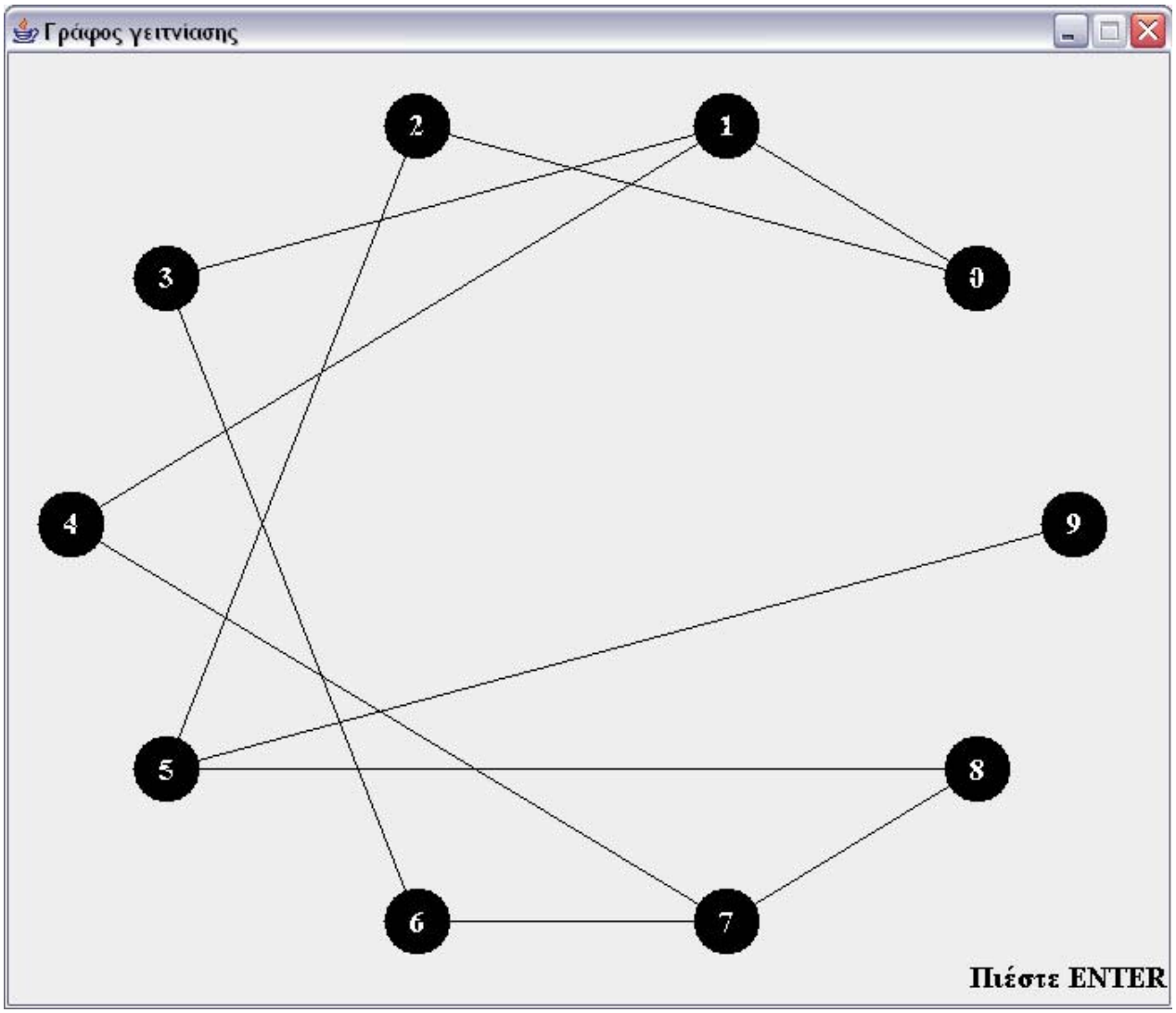


Παράδειγμα 2ο

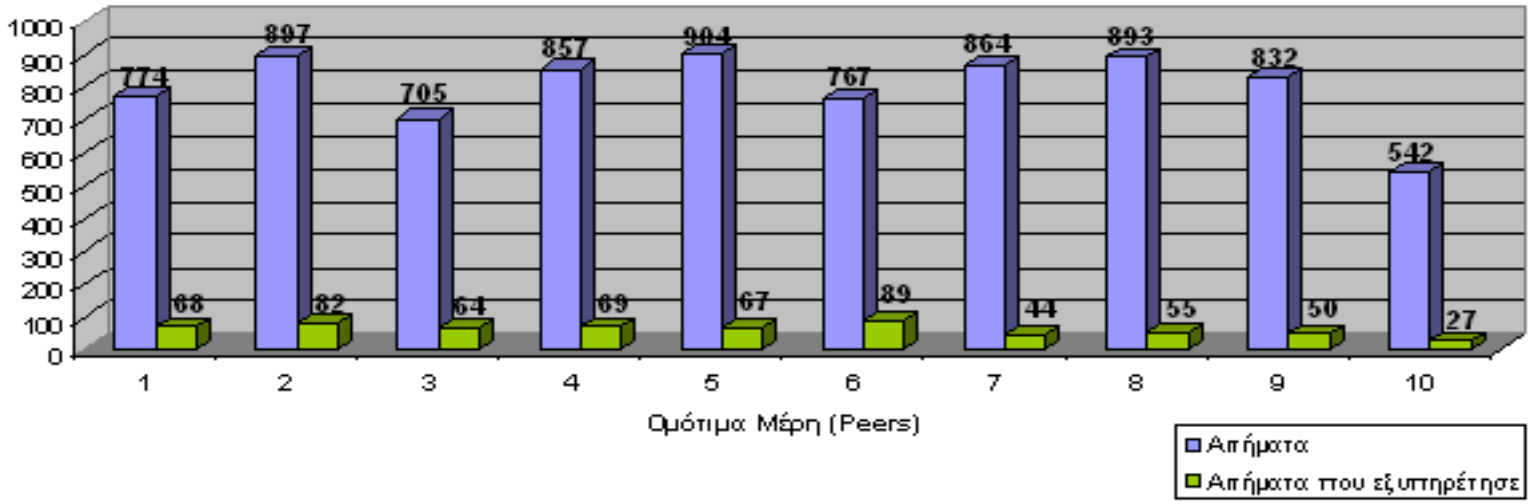


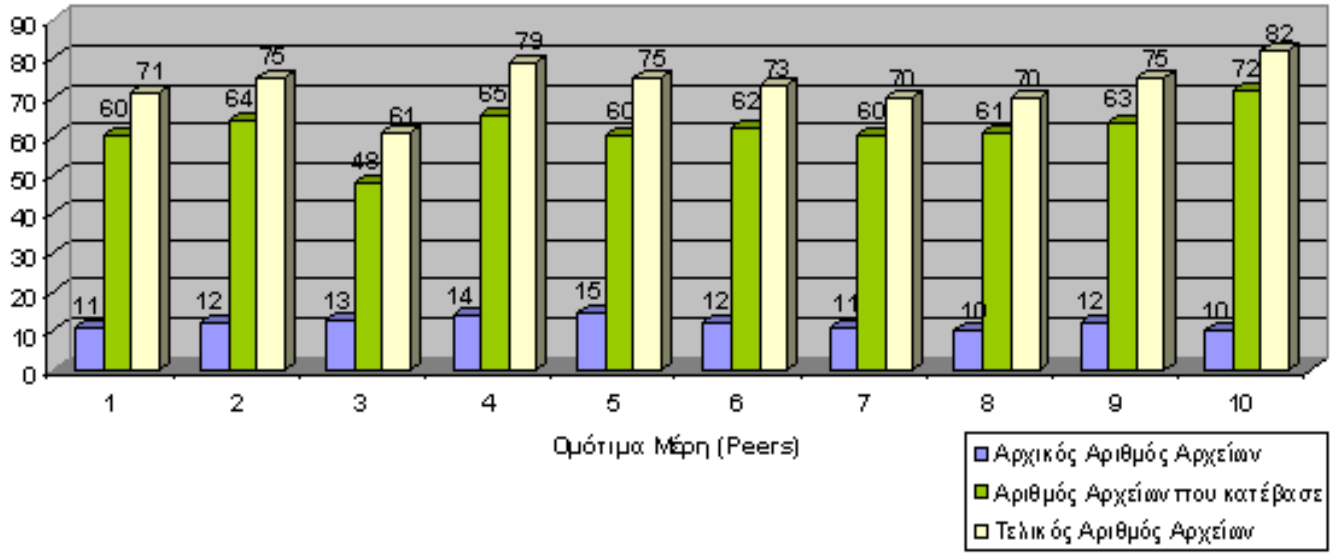


Παράδειγμα 3ο



Παράδειγμα 3ο





Πηγές

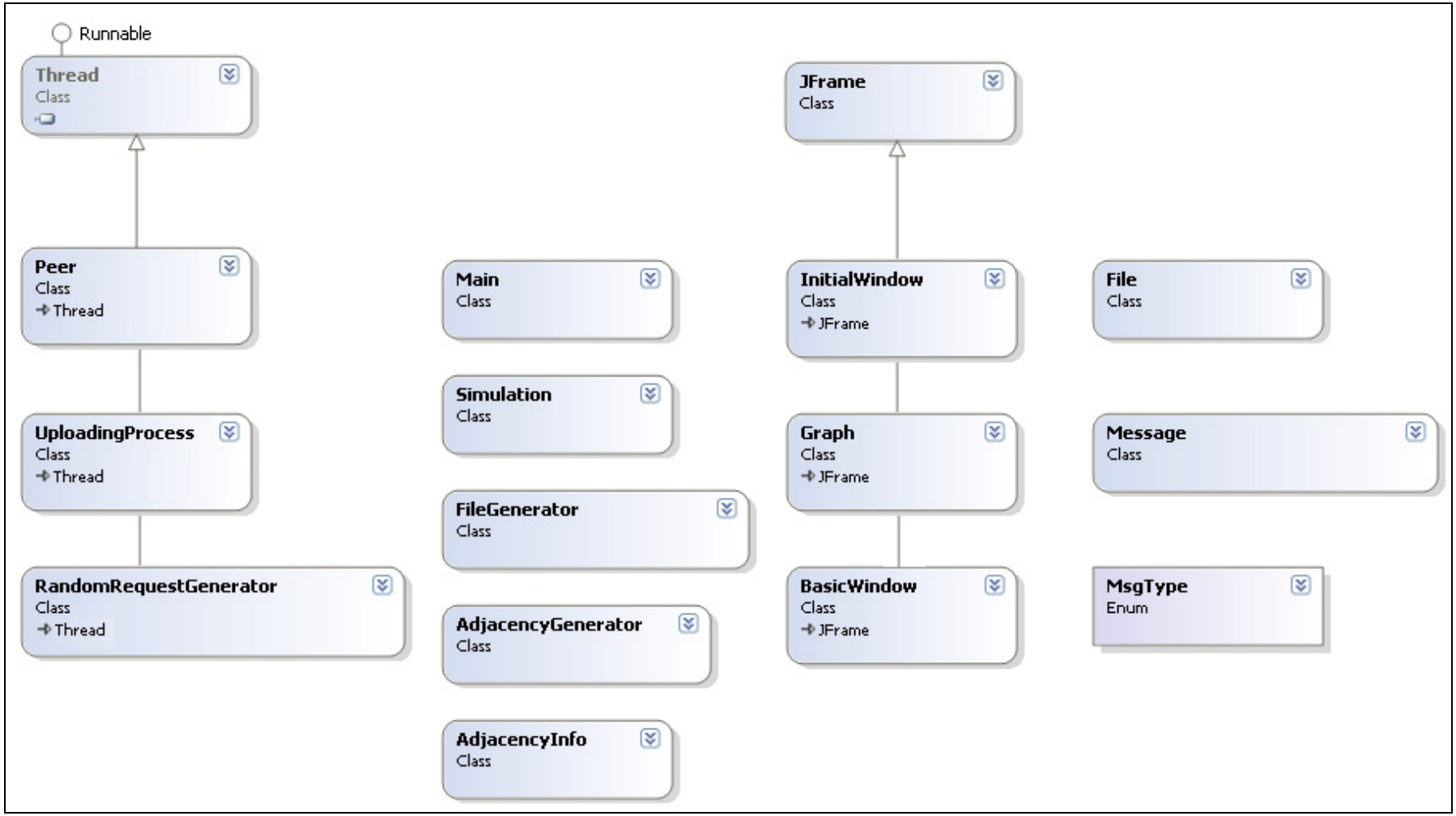
Βιβλιογραφία

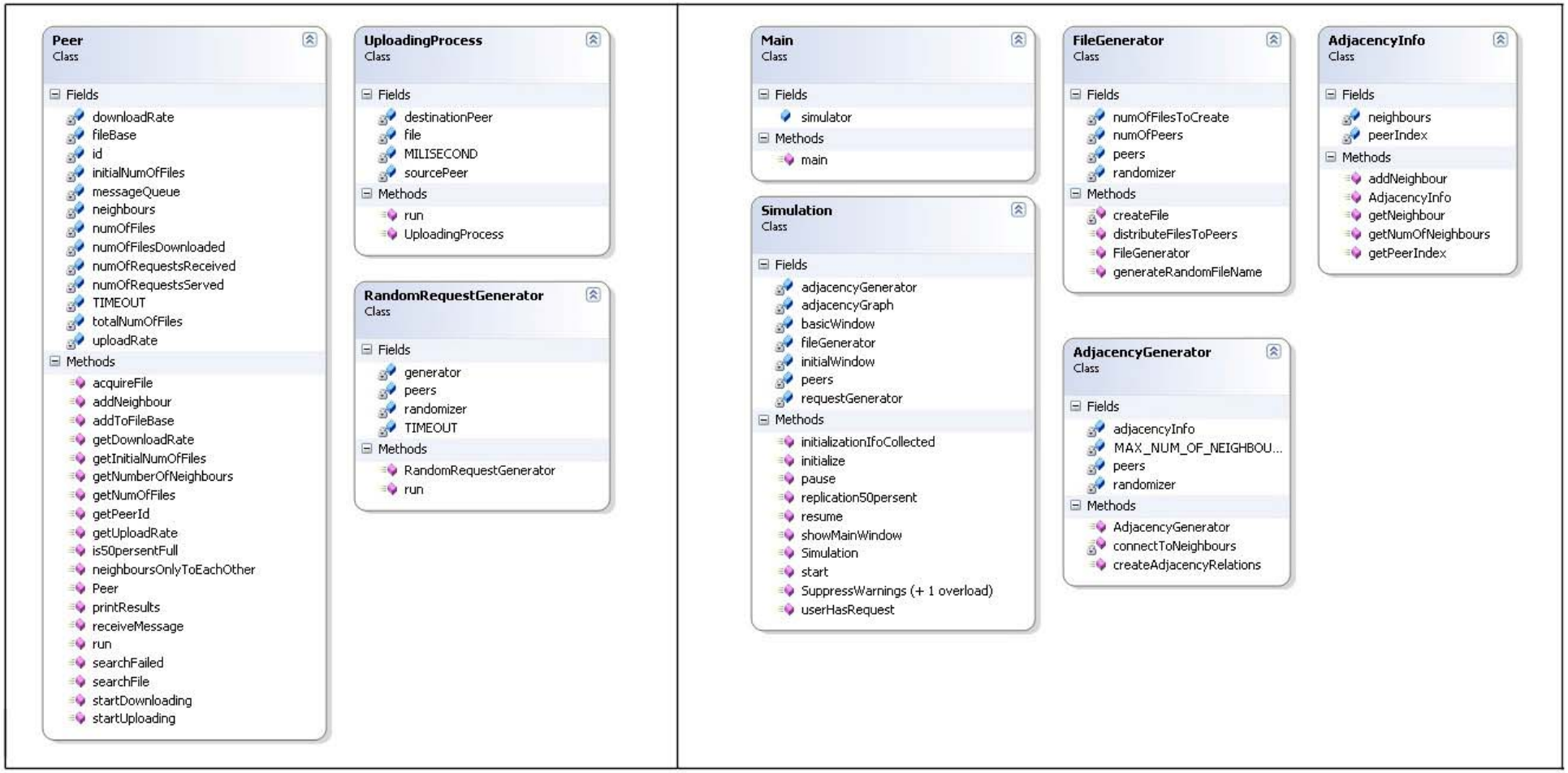
- Κατανεμημένα συστήματα με Java, Ι. Κ. Κάβουρας, Ι. Ζ. Μήλης, Γ. Β. Ξυλωμένος, Α. Α. Ρουκουνάκη, έκδοση 2η (2005), Κλειδάριθμος.
- Αντικειμενοστρεφής ανάπτυξη λογισμικού με τη UML, Β. Γερογιάννης, Γ. Κακαρότζας, Α. Καμέας, Γ. Σταμέλος, Π. Φιτσιλής, Κλειδάριθμος.
- Java 2 how to program, Deitel & Deitel, 4η εκδοση (2002), Prentice Hall.
- Multithreaded Programming with Java technology, Bil Lewis, Daniel J. Berg, 1999, Prentice Hall.

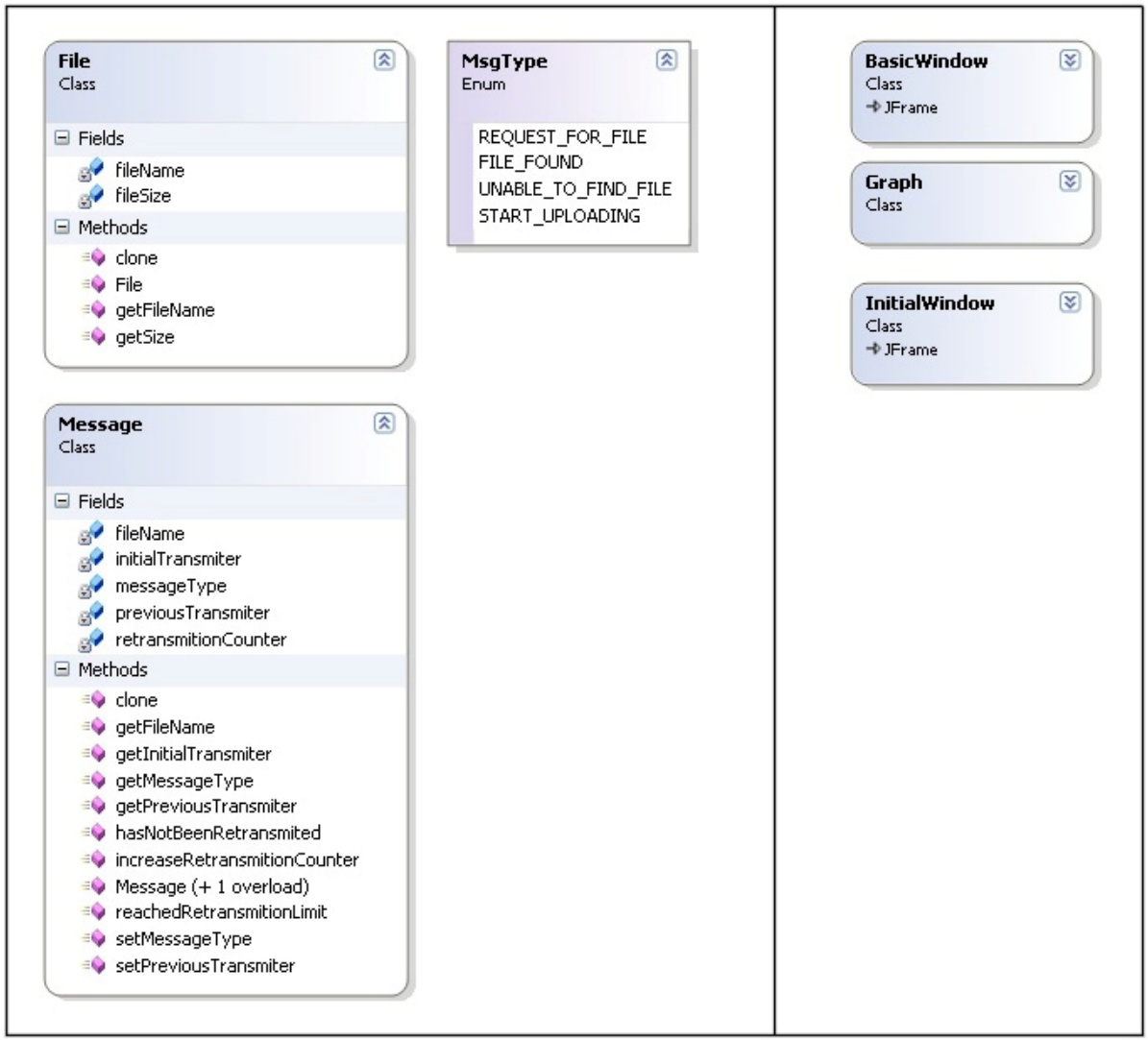
Ιστοσελίδες

- (Hashing) http://www.dcc.uchile.cl/~rbaeza/handbook/hbook.html
- (Perfect Hashing) http://burtleburtle.net/bob/hash/perfect.html
- (Double Hashing) http://www.nist.gov/dads/HTML/doublehashng.html
- http://www.brpreiss.com/books/opus5/html/page205.html
- http://planetmath.org/encyclopedia/Hashing.html

ΔΙΑΓΡΑΜΜΑ ΚΛΑΣΕΩΝ




Peer
Class
Fields
downloadRate
fileBase
id
initialNumOfFiles
messageQueue
neighbours
numOfFiles
numOfFilesDownloaded
numOfRequestsReceived
numOfRequestsServed
TIMEOUT
totalNumOfFiles
uploadRate
Methods
acquireFile
addNeighbour
addToFileBase
getDownloadRate
getInitialNumOfFiles
getNumberOfNeighbours
getNumOfFiles
getPeerId
getUploadRate
is50persentFull
neighboursOnlyToEachOther
Peer
printResults
receiveMessage
run
searchFailed
searchFile
startDownloading
startUploading
UploadingProcess
Class
Fields
destinationPeer
file
MILISECOND
sourcePeer
Methods
run
UploadingProcess
RandomRequestGenerator
Class
Fields
generator
peers
randomizer
TIMEOUT
Methods
RandomRequestGenerator
run
Main
Class
Fields
simulator
Methods
main
Simulation
Class
Fields
adjacencyGenerator
adjacencyGraph
basicWindow
fileGenerator
initialWindow
peers
requestGenerator
Methods
initializationIfoCollected
initialize
pause
replication50persent
resume
showMainWindow
Simulation
start
SuppressWarnings (+ 1 overload)
userHasRequest
FileGenerator
Class
Fields
numOfFilesToCreate
numOfPeers
peers
randomizer
Methods
createFile
distributeFilesToPeers
FileGenerator
generateRandomFileName
AdjacencyGenerator
Class
Fields
adjacencyInfo
MAX_NUM_OF_NEIGHBOU...
peers
randomizer
Methods
AdjacencyGenerator
connectToNeighbours
createAdjacencyRelations
AdjacencyInfo
Class
Fields
neighbours
peerIndex
Methods
addNeighbour
AdjacencyInfo
getNeighbour
getNumOfNeighbours
getPeerIndex


File
Class
Fields
fileName
fileSize
Methods
clone
File
getFileName
getSize
MsgType
Enum
REQUEST_FOR_FILE
FILE_FOUND
UNABLE_TO_FIND_FILE
START_UPLOADING
BasicWindow
Class
JFrame
Graph
Class
InitialWindow
Class
JFrame
Message
Class
Fields
fileName
initialTransmiter
messageType
previousTransmiter
retransmitionCounter
Methods
clone
getFileName
getInitialTransmiter
getMessageType
getPreviousTransmiter
hasNotBeenRetransmited
increaseRetransmitionCounter
Message (+ 1 overload)
reachedRetransmitionLimit
setMessageType
setPreviousTransmiter