ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΡΗΤΗΣ

Τίτλος Πτυχιακής Εργασίας

"Distributed Logic over Peer to Peer Networks"

Τμήμα Εφαρμοσμένης Πληροφορικής και Πολυμέσων

Τζεφεράκος Renato
Ημερομηνία: Μάιος 2009

Εισηγητής: **Παπαδάκης Νικόλαος**

Περίληψη

Αντικείμενο της παρούσας πτυχιακής εργασίας είναι η μελέτη των λογικών συνθηκών που διέπουν τα ομότιμα δίκτυα. Τα ομότιμα δίκτυα κυριαρχούν στη σύγχρονη κοινωνία της πληροφορίας καθώς προσφέρουν αυξημένη υπολογιστική ισχύ καθώς και αποθηκευτικό χώρο συνδυάζοντας τους πόρους των ομότιμων κόμβων που συγκροτούν το δίκτυο .

Η πτυχιακή εργασία παρουσιάζει μια λογική προσέγγιση της αλγοριθμικής μεθοδολογίας που χρησιμοποιείται για την επίλυση λογικών προβλημάτων στο θεωρητικό της σκέλος καθώς και μια πρακτική προσέγγιση ελέγχου συνθηκών μεταξύ των ομότιμων πόρων.

Το πρακτικό τμήμα της πτυχιακής αναπτύχθηκε σε Java αντικειμενοστραφή γλώσσα προγραμματισμού για την επίτευξη της μεγαλύτερης δυνατής συμβατότητας μεταξύ πλατφορμών.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω εκ βάθους καρδίας όλους όσους συμμετείχαν με τη βοήθεια τους στη περάτωση της παρούσας πτυχιακής εργασίας. Ειδικότερα, τον κύριο Παπαδάκη Νικόλαο για την πολύτιμη καθοδήγηση και βοήθεια του καθ'όλη τη διάρκεια μελέτης της.

Τέλος θα ήθελα να ευχαριστήσω όλους όσος έδειξαν την απαραίτητη ψυχολογική κατανόηση και προσέφεραν την υποστήριξη τους κατά τη διάρκεια της εργασίας.

Με εκτίμηση
Renato

Περιεχόμενα

Κεφάλαιο 1ο: Εισαγωγή

Κεφάλαιο 2ο: Peer to Peer Networks

Κεφάλαιο 3ο: Logic Resolver

Κεφάλαιο 4ο

Παράρτημα

Κεφάλαιο 1ο

Παράθεμα

Η λογική στην επιστήμη των υπολογιστών περιγράφει ένα θέμα όπου η μια σειρά λογικών εκφράσεων εφαρμόζονται στην αναπαράσταση της γνώσης και της τεχνητής νοημοσύνης συμπεριλαμβάνοντας τα ψηφιακά κυκλώματα και τις βάσεις δεδομένων.

Η χρήση των κατανεμημένων συστημάτων, στην επίλυση λογικών προβλημάτων αποτελεί ένα ενδιαφέρον και απαιτητικό θέμα στην επιστήμη των υπολογιστών, όπου αλγόριθμοι στα ερωτήματα (queries) δικτύων επιταχύνουν την διαδικασία επεξεργασίας.

Η παρούσα εργασία παρουσιάζει ένα μοντέλο Peer to Peer συλλογισμού με τη χρήση ερωτημάτων δια μέσου των ομότιμων κόμβων.

Στο πρώτο σκέλος της εργασίας γίνεται μια γενική αναφορά στη λογική των ομότιμων δικτύων. Εν συνεχεία, επιχειρείται μια ανάλυση των δικτύων και η παρουσίαση των πρακτικών εφαρμογών τους σε εμπορικό επίπεδο. Στο τρίτο κεφάλαιο παρουσιάζεται ο Logic Resolver και η λογική της αλγοριθμικής διαδικασίας επίλυσης λογικών προβλημάτων. Τέλος, ακολουθεί το παράρτημα με τον κώδικα που χρησιμοποιήθηκε για την εξομοίωση τόσο του δικτύου όσο και του ελέγχου συνθηκών σε περιβάλλον ομότιμου δικτύου.

Στο παράρτημα γίνεται περιγραφή ενός προγράμματος instant messaging το οποίο στηρίζεται στο Java Application Interface και στη λογική του socket programming. Με τον τρόπο αυτό μπορούμε να εξομοιώσουμε και τοπικά σε ένα κόμβο τις συνθήκες που επικρατούν σε ένα ομότιμο δίκτυο πραγματοποιώντας λογικές συγκρίσεις συμβολοσειρών μεταξύ των μηνυμάτων που ανταλλάσσονται μεταξύ των κόμβων στο πρόγραμμα.

Εισαγωγή

Το peer to peer δίκτυο (P2P) υπολογιστών χρησιμοποιεί διαφορετική συνδεσμολογία μεταξύ των συμμετεχόντων και το συσσωρευτικό εύρος ζώνης των συμμετεχόντων δικτύων παρά τους συμβατικούς συγκεντρωμένους πόρους όπου ένας σχετικά χαμηλός αριθμός κεντρικών υπολογιστών παρέχουν την κεντρική αξία σε μια υπηρεσία ή μια εφαρμογή.

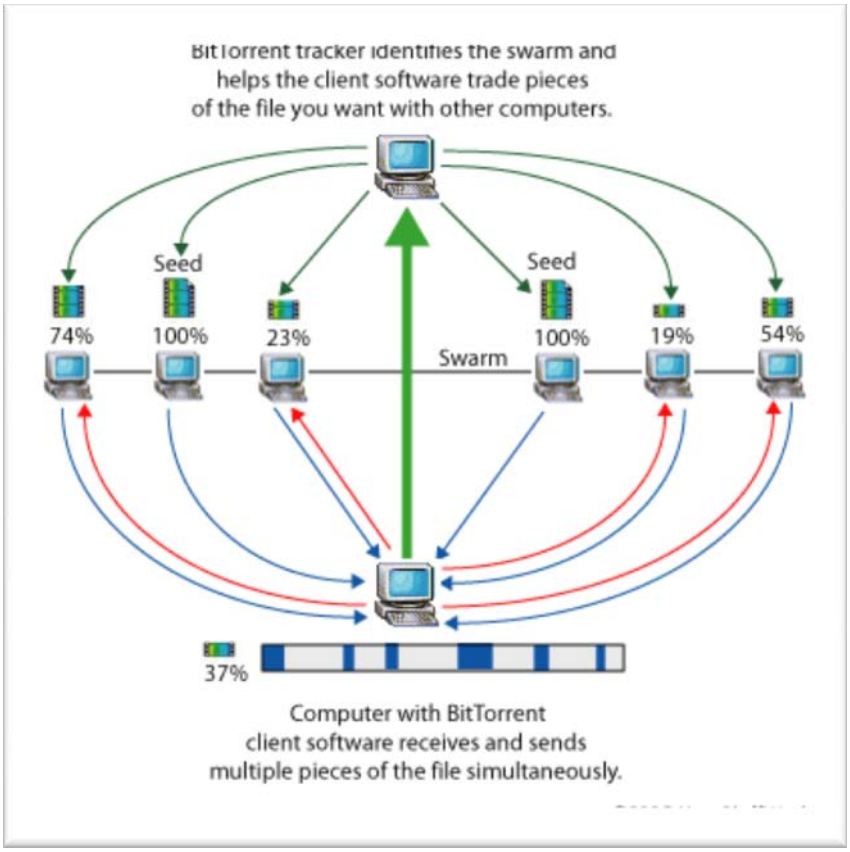
Τα P2P δίκτυα χρησιμοποιούνται χαρακτηριστικά για τη σύνδεση κόμβων μέσω μεγάλων ad hoc συνδέσεων. Τέτοια δίκτυα είναι χρήσιμα για πολλούς σκοπούς.

Η Διανομή των αρχείων που εμπεριέχουν ήχο, εικόνα, δεδομένα ή οτιδήποτε άλλο σε ψηφιακή μορφή είναι πολύ διαδεδομένη και τα real-time δεδομένα όπως η τηλεφωνία πραγματοποιείται με τη χρήση της τεχνολογίας των Peer to Peer δικτύων.

A **peer-to-peer** based network

A **server** based network

Ένα αυθεντικό peer to peer δίκτυο δεν έχει την έννοια πελάτες ή κεντρικοί υπολογιστές αλλά μόνο ομότιμους κόμβους που ταυτόχρονα λειτουργούν και όπως " πελάτες" και όπως "κεντρικοί υπολογιστές" για τους άλλους κόμβους στο δίκτυο. Αυτό το μοντέλο δικτύων διαφέρει από το μοντέλο “ πελάτης - κεντρικός υπολογιστής” όπου η επικοινωνία γίνεται συνήθως από και προς ένα κεντρικό υπολογιστή.

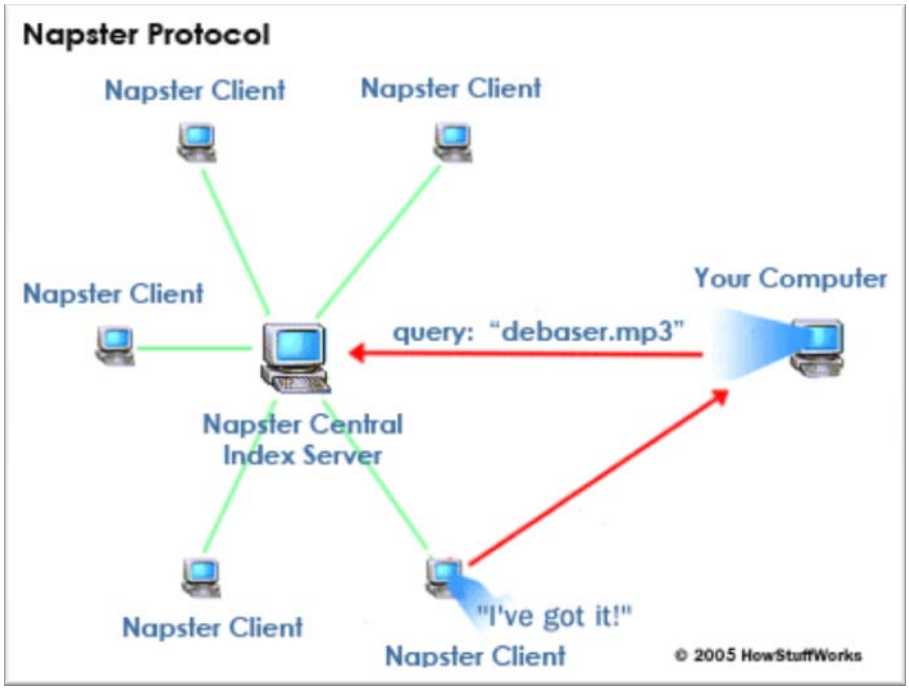


Ένα χαρακτηριστικό παράδειγμα μεταφοράς αρχείων που δεν υπόκειται στη κατηγορία των P2P δικτύων είναι οι “FTP Server” όπου οι λειτουργίες του πελάτη διαφέρουν αρκετά από τις λειτουργίες του κεντρικού υπολογιστή: ο πελάτης πραγματοποιεί αιτήματα μεταφοράς αρχείων και ο κεντρικός υπολογιστής αντιδρά

για να ικανοποιήσει αυτά τα αιτήματα.

Ιστορική Αναδρομή των Peer to Peer δικτύων

Στις αρχές του 1999 ο Shawn Fanning ξεκίνησε την υλοποίηση μιας ιδέας, η οποία θα του έδινε τη δυνατότητα αυτός και οι φίλοι του να αναζητήσουν στο Internet μουσικά κομμάτια MP3 της προτίμησής τους.



Μερικούς μήνες αργότερα, η Napster Inc. μετρούσε πάνω από 21 εκατομμύρια χρήστες. Σε καμία περίπτωση όμως ο 18χρονος τότε μαθητής δεν μπορούσε να φανταστεί ότι το δημιούργημά του θα άλλαζε τον τρόπο με τον οποίο απολαμβάνουμε πολυμεσικές εφαρμογές και γενικά να επικοινωνούμε [1]

Κατηγοριοποίηση των P2P networks

Τα Ομότιμα δίκτυα μπορούν να κατηγοριοποιηθούν για την χρήση που προορίζονται:

- Μεταφορά αρχείων
- Τηλεφωνία
- Μετάδοση Multimedia περιεχομένου
- Φόρουμ συζητήσεων

Κεφάλαιο 2ο

Μορφές Peer-to-Peer δικτύων

Τα Peer 2 Peer δίκτυα χωρίζονται σε τρεις κατηγορίες:

Συγκεντρωτικά P2P δίκτυα

Πολλοί, όταν αναφέρονται σε αυτά, χρησιμοποιούν τη φράση «πρώτης γενιάς P2P δίκτυα». Εδώ, υπάρχει ένας κεντρικός Index Server στον οποίο αποθηκεύονται οι πληροφορίες για τα περιεχόμενα των καταλόγων που οι συμμετέχοντες επιθυμούν να μοιράζονται. Οι χρήστες μπορούν να αναζητήσουν στους Index Servers αυτούς τα αρχεία που ψάχνουν, χρησιμοποιώντας ένα κατάλληλο πρόγραμμα-πελάτη. Όταν το αρχείο βρεθεί, ανοίγει μια σύνδεση μεταξύ των δύο χρηστών για τη μεταφορά του. Σε αυτή τη κατηγορία ανήκουν το Napster, το DC++ και το WinMX.

Αποκεντρωτικά P2P δίκτυα

Η φιλοσοφία εδώ είναι εντελώς διαφορετική. Κάθε σύστημα που συμμετέχει αποτελεί ταυτόχρονα client και server (ή αλλιώς servent). Μόλις κάποιος συνδεθεί μέσω ενός ανάλογου προγράμματος-πελάτη P2P, κάνει γνωστή την παρουσία του σε ένα μικρό αριθμό υπολογιστών ήδη συνδεδεμένων οι οποίοι με τη σειρά τους προωθούν τη δήλωση παρουσίας του σε ένα μεγαλύτερο δίκτυο υπολογιστών κ.λ.π .
Πλέον ο χρήστης έχει τη δυνατότητα να αναζητήσει οποιαδήποτε πληροφορία μεταξύ των διαμοιραζόμενων αρχείων. Τα δίκτυα αυτά λέγονται και δεύτερης γενιάς. Η μεταφορά των αρχείων είναι όμοια με αυτή των αποκεντρωτικών P2P δικτύων. Σε αυτή τη κατηγορία ανήκουν το Kazaa το Gnutella και το Bearshare.

P2P δίκτυα τρίτης γενιάς

Είναι αυτά τα οποία διαθέτουν χαρακτηριστικά ανωνυμίας όπως το Freenet το I2P και το Entropy είναι αποκεντρωτικού τύπου και η φιλοσοφία του βασίζεται εκτός από την ανωνυμία, στην υψηλή βιωσιμότητα του, στο συνεχή διαμοιρασμό των αρχείων και στην κωδικοποίησή τους έτσι ώστε κανείς να μην μπορέσει ποτέ να αποκτήσει κανένα είδος ελέγχου πάνω σε αυτό.
Τα δίκτυα αυτού του τύπου είναι υπό ανάπτυξη και έχουν χαρακτηριστεί ως μικρά παγκόσμια δίκτυα.

Προτερήματα Peer to Peer Networks

Ένας σημαντικός στόχος στα P2P δίκτυα είναι η προσπάθεια όλοι οι ομότιμοι κόμβοι να παρέχουν πόρους, συμπεριλαμβανομένων του εύρους ζώνης και αποθηκευτικού χώρου καθώς και υπολογιστικής δύναμης.

Κατά συνέπεια, καθώς οι κόμβοι προστίθενται και αυξάνεται η απαίτηση συστημάτων, η συνολική δυνατότητα του δικτύου αυξάνεται. Το γεγονός αυτό έρχεται σε αντίθεση με την αρχιτεκτονική “client –server” όπου όσο προστίθενται κόμβοι στο δίκτυο τόσο μικρότερη είναι η υπολογιστική δύναμη και επομένως επιβραδύνεται η μεταφορά δεδομένων για όλους τους χρήστες.

Η διανεμημένη φύση των ομότιμων δικτύων αυξάνει επίσης και την ανοχή του δικτύου σε σφάλματα αντιγράφοντας δεδομένα σε περισσότερους από ένα κόμβους και επιτρέποντας στους ομότιμους κόμβους να αναζητήσουν και να βρουν τα δεδομένα που αιτούνται χωρίς να εξαρτώνται από ένα κεντρικό υπολογιστή περιεχομένων. Σε αυτή τη περίπτωση αποφεύγονται όλα τα σφάλματα λειτουργίας του συστήματος.

Η θεώρηση των Ομότιμων δικτύων στην επιστήμη της πληροφορικής

Τεχνικά, μια απολύτως καθαρή P2P εφαρμογή πρέπει να εφαρμόζει μόνο τα πρωτόκολλα που δεν αναγνωρίζουν τις έννοιες "του κεντρικού υπολογιστή" και "του πελάτη". Τέτοιες ομότιμες εφαρμογές δικτύων είναι σπάνιες.

Τα περισσότερα δίκτυα και εφαρμογές που χαρακτηρίζονται ως ομότιμα στηρίζονται σε κάποια χαρακτηριστικά αντίθετα στην ομότιμη λογική , όπως το DNS (Domain Name System).

Επίσης, οι εφαρμογές που χρησιμοποιούνται πλέον συχνά κάνουν χρήση διαφόρων πρωτόκολλων και λειτουργούν τόσο ως “client”, ”server” και “peer” παράλληλα ή για κάποιο χρονικό διάστημα.

Πλήρως αποκεντρωμένα δίκτυα ομότιμων κόμβων χρησιμοποιούνται για πολλά χρόνια: παραδείγματα τέτοια είναι τα Usenet (1979) & WWIVnet(1987).

Πολλά ομότιμα συστήματα χρησιμοποιούν ισχυρότερους κόμβους (super peers, super nodes) δεδομένου ότι οι κεντρικοί υπολογιστές και οι πελάτης - ομότιμοι κόμβοι συνδέονται σε μια αστεροειδή μορφή με έναν κόμβο.

Προηγούμενες Υλοποιήσεις Peer to Peer συστημάτων

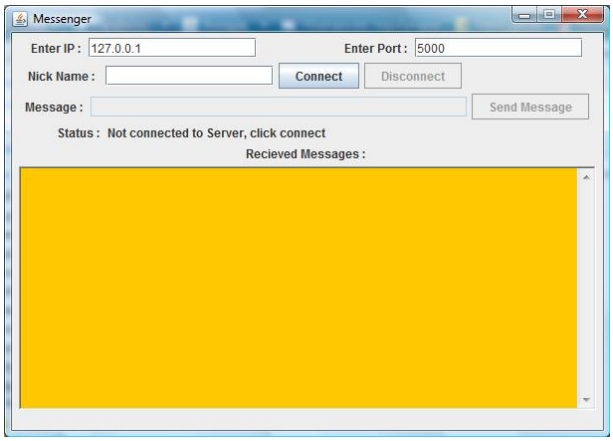
Τα συστήματα multi-agent multi-context [1,2]είναι παραδείγματα προηγούμενης εξέλιξης αυτών των προτύπων. Τα πρότυπα, όπως [3], είναι βασισμένα στη non-monotonic epistemic logic, και επιτρέπουν στους διαχειριζόμενους κόμβους να παρέχουν τα αμοιβαία ασυμβίβαστα στοιχεία .

Άλλα πρότυπα [4] προτείνουν ένα (P2P) σύστημα συμπεράσματος που εξετάζει τις συγκρούσεις που προκαλούνται από τις αμοιβαία ασυμβίβαστες πηγές πληροφοριών, με την ανίχνευση τους και το συλλογισμό χωρίς αυτές.
Μια κοινή ανεπάρκεια εκείνων των δύο προτύπων είναι ότι οι συγκρούσεις δεν επιλύονται πραγματικά χρησιμοποιώντας κάποιες εξωτερικές πληροφορίες εμπιστοσύνης ή προτεραιότητας, αλλά είναι μάλλον απομονωμένες.

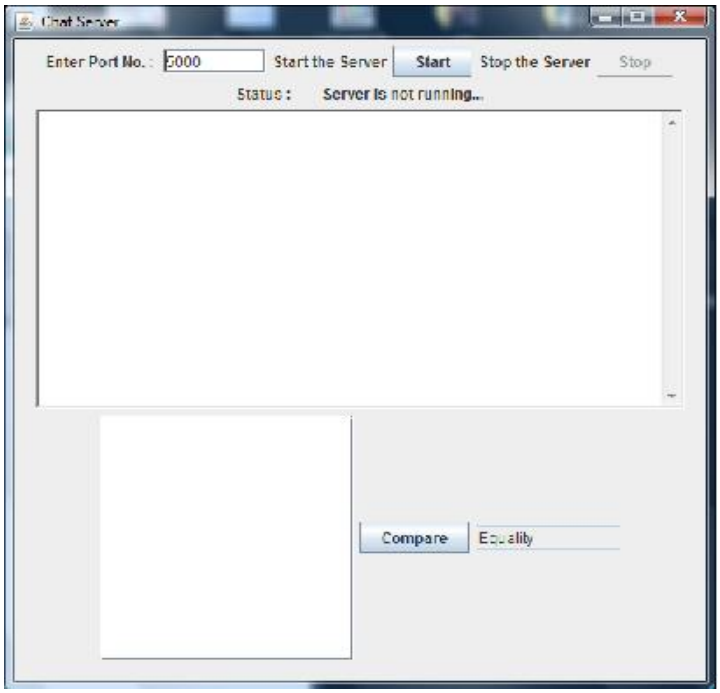
Άλλα πρότυπα[5, 6] προτείνουν εστίαση σε εκείνες τις πιθανές συγκρούσεις που μπορούν να προκύψουν από την αλληλεπίδραση των διαφορετικών πλαισίων (Network Knowledge) μέσω ενός συνόλου κανόνων συμπεράσματος καθώς και την επίλυση των συγκρούσεων χρησιμοποιώντας μια προτεραιότητα εμπιστοσύνης.

Εντούτοις, αυτά τα πρότυπα δεν εστιάζουν στην ανακούφιση του δικτύου από τις περιττές ερωτήσεις και τις ερωτήσεις που καθυστερούν και τον κύκλο γύρω από ποιες αιτίες μια υπερφόρτωση στο δίκτυο.

Η εφαρμογή που παρουσιάζεται βασίζεται σε ένα καθαρά P2P δίκτυο όπου κάθε κόμβος του δικτύου ενεργεί ως ομότιμος κάνοντας χρήση τόσο του client role όσο και του server role λειτουργώντας ως servant.
Κάθε κόμβος έχει τη διακριτά χαρακτηριστικά όπως User ID & User IP καθώς επίσης και τη δυνατότητα διάκρισης της σύνδεσης του με άλλο κόμβο βάσει IP διεύθυνσης και θύρας σύνδεσης.



Client



Server

Δίκτυα Peer to Peer και πρωτόκολλα

Κάποια χαρακτηριστικά παραδείγματα εφαρμογών της αρχιτεκτονικής των Peer to Peer δικτύων.

Network or Protocol	**Use**	**Applications**
BitTorrent	File sharing/Software distribution/Media distribution	ABC, AllPeers, Vuze (formerly Azureus), BitComet, BitLord, BitTornado, BitTorrent, Burst!, Deluge, FlashGet, G3 Torrent, Halite, KTorrent, LimeWire, MLDonkey, Opera, QTorrent, rTorrent, Shareaza, TorrentFlux, Transmission, Tribler, µTorrent, Xunlei
Domain Name System	Internet information retrieval	
Freenet	Distributed data store	Entropy, Freenet
Gnutella	File sharing	Acquisition, BearShare, Cabos, FilesWire, FrostWire,
P2PTV	Video stream or file sharing	TVUPlayer, Joost, CoolStreaming, Cybersky-TV, TVants, PPLive, LiveStation
WWIVnet	Distributed network	
Windows Peer-to-Peer	Distributed peer application development, collaboration	

Κεφάλαιο 3^{ο}

Μια λογική προσέγγιση των δικτύων Peer to Peer

Ένα peer to peer σύστημα αποτελείται από δυο κύρια τμήματα, το Logic Resolver, όπου γίνεται η επεξεργασία των λογικών συνθηκών και το Network, όπου παρουσιάζεται το ομότιμο δίκτυο.

Logic Resolver

Στο πρώτο τμήμα περιγράφεται ένας λογικός συντάκτης συνθηκών (Logic Editor) όπου ο χρήστης παραμετροποιεί τις τοπικές συνθήκες και τους κανόνες που θα εφαρμόζονται στη λειτουργία του Peer to Peer Δικτύου.

Κάθε κόμβος του ομότιμου δικτύου φορτώνει τις συνθήκες και τις παραμέτρους του στον αλγόριθμο που χρησιμοποιείται για την επίλυση του λογικού προβλήματος. Ως πρώτο βήμα ορίζεται η προσπάθεια επίλυσης του προβλήματος με τις τοπικές συνθήκες του κόμβου κάνοντας χρήση επαναλαμβανόμενων μεθόδων σε τοπικό επίπεδο.
Αυτή η μέθοδος μπορεί να αποκλείσει τα ελλείποντα γεγονότα που δεν είναι απαραίτητα για να επιλύσουν το πρόβλημα ή ακόμα και να την επιλύσουν, παρά ότι υπάρχουν ελλείποντα γεγονότα.

Αφού ολοκληρωθεί η αρχική διαδικασία επίλυσης του λογικού προβλήματος με τη χρήση του τοπικού συλλογισμού ακολουθείται η διαδικασία του διανεμημένου με τη χρήση των παραμέτρων και των συνθηκών των ομότιμων κόμβων. Ο κόμβος που πραγματοποιεί την αλγοριθμική διαδικασία αποθηκεύει και διαβιβάζει οποιαδήποτε ελλείποντα γεγονότα στο ομότιμο δίκτυο. Σε αυτό το σημείο υπεισέρχεται το τμήμα δικτύου (Network Component).

Network Component

Στο τμήμα δικτύου ο κόμβος αποστέλλει στους γνωστούς ομότιμους κόμβους τα άγνωστα γεγονότα που του είναι απαραίτητα για την επίλυση του λογικού προβλήματος με τους υπόλοιπους κανόνες με τη μορφή ερωτήματος (query) αναμένοντας απάντηση σε ένα συγκεκριμένο χρονικό διάστημα που καθορίζεται από τις τοπικές του παραμέτρους.

Εάν ένας εκ των γνωστών κόμβων δεν έχει τις απαιτούμενες συνθήκες τότε μπορεί να διαβιβάσει την ερώτηση σε έναν άγνωστο, για τον αρχικό, κόμβο. Η πλημμύρα ερώτησης (query flood) αποφεύγεται χάρις στους αλγορίθμους περικοπής.

Όταν ο χρόνος αναμονής για το ερώτημα που αποστέλλεται στους ομότιμους κόμβους λήγει, τότε επαναξιολογούνται οι μη επιλυμένοι κανόνες δεδομένων των πρόσφατα προσεγγισμένων γεγονότων. Η ασυνέπεια αντιμετωπίζεται χρησιμοποιώντας τις τιμές των παραμέτρων των ομότιμων κόμβων.

Η λογική του Logic Resolver

Το τμήμα Resolver παρέχει μια διεπαφή όπου οι χρήστες δημοσιοποιούν τα προβλήματα λογικής καθώς επίσης και τα τοπικά δεδομένα που έχουν για την επίλυση των προβλημάτων.
Ένας έλεγχος σύνταξης εκτελείται πριν από την αξιολόγηση του προβλήματος. Εάν έχουμε έναν επιτυχή έλεγχο σύνταξης η διαδικασία προχωρά στην επίλυση του προβλήματος.

Ο Resolver μπορεί να αξιολογήσει τους σύνθετους κανόνες λογικής χρησιμοποιώντας τα βασικά σύμβολα λογικής. Τα βασικά σύμβολα που χρησιμοποιούνται είναι τα εξής:

Λογικά Σύμβολα

‘$\rightarrow$’ Αν μια συνθήκη είναι αληθής τότε ισχύει και η συνθήκη που ακολουθεί.

‘$\neg$’ Η συνθήκη που ακολουθεί είναι αληθής αν και μόνο αν η συνθήκη που προηγείται είναι ψευδής.

‘$\vee$’ Η συνθήκη είναι αληθής αν ένα από τα δυο μέρη της συνθήκης είναι αληθή ή και τα δυο ψευδή.

’$\wedge$’ Η συνθήκη είναι αληθής αν και μόνο αν και τα δυο μέρη της συνθήκης είναι αληθή.

‘$\leftrightarrow$’ Η συνθήκη που ακολουθεί είναι αληθής αν και μόνο αν είναι η πρώτη είναι αληθής. Αν η πρώτη συνθήκη είναι ψευδής τότε και αυτή που έπεται είναι ψευδής.

Η λογική του αλγορίθμου επίλυσης λογικών προβλημάτων

Κατά τη διαδικασία της λογικής επίλυσης του προβλήματος κάθε κόμβος θα πρέπει να κάνει χρήση καταρχήν των τοπικών συνθηκών και κανόνων προτού απευθυνθεί στους ομότιμους συνδέσμους.

Εν συνεχεία ενημερώνεται το τμήμα δικτύου (Network Component) για να μπορεί ο κόμβος να αιτηθεί τα επιπλέον γεγονότα και κανόνες που είναι απαραίτητα όσο επεξεργάζεται το λογικό πρόβλημα και θα προσπαθήσει να επιλύσει το πρόβλημα λογικής χρησιμοποιώντας τα τοπικά γεγονότα και τα προσωρινά γεγονότα δικτύων.

Εάν χρειαστεί ένας κανόνας τα γεγονότα που δεν είναι τοπικά διαθέσιμα, θα προσπαθήσει να αξιολογήσει τον κανόνα με τον καθορισμό των ελλειπόντων γεγονότων ως ψευδή. Το αποτέλεσμα, αν είναι αληθές, αποθηκεύεται ως γεγονός και τα τοπικά γεγονότα ανανεώνονται ενώ ενημερώνεται και το Network Component με απώτερο σκοπό τη βελτιστοποίηση της απόδοσης του δικτύου καθώς κάθε Peer μπορεί να ζητήσει τα νέα γεγονότα κατά τη διάρκεια της επίλυσης ενός τοπικού λογικού προβλήματος. Κατόπιν, ο κόμβος θα προσπαθήσει με τη χρήση όλων των κανόνων που δημιουργήθηκαν ή προϋπήρχαν να διεκπεραιώσει τη διαδικασία επαναλαμβάνοντας τη διαδικασία έως ότου δε προκύπτει λύση βάσει των τοπικών συνθηκών και κανόνων ή δεν είναι απαραίτητη περεταίρω ανάλυση.

Χρησιμοποιώντας τον προαναφερθέντα αλγόριθμο, ένας Peer του δικτύου μπορεί να αποκλείσει τα ελλείποντα γεγονότα που δεν είναι απαραίτητα για την επίλυση του προβλήματος.
Τα υπόλοιπα ελλείποντα γεγονότα θα συγχωνευθούν σε μια ερώτηση με έναν συγκεκριμένο χρόνο αποστολής (Time to leave, TTL) και θα αποσταλούν στους ομότιμους κόμβους που είναι συνδεδεμένοι.

Όταν ο χρόνος εκπνεύσει, γίνεται έλεγχος για παρειλημμένες απαντήσεις:

- ✓ Εάν δεν υπάρχει καμία απάντηση, δεν μπορεί να υπάρξει καμία περαιτέρω αξιολόγηση του προβλήματος
- ✓ Στην αντίθετη περίπτωση, επαναξιολογεί τους εκκρεμείς κανόνες με τα πρόσφατα προσεγγισμένα γεγονότα

Παρόλα αυτά, δεν είναι σίγουρο πως όλα τα ελλείποντα γεγονότα θα ανακτηθούν από το δίκτυο καθιστώντας αυτή την επαναξιολόγηση απαραίτητη δεδομένου πως δεν υπάρχει καμία εφαρμογή της απλοποίησης κανόνων.
Εάν ο κανόνας είναι αληθινός, αποθηκεύεται ως γεγονός και τα τοπικά γεγονότα όσο και τα γεγονότα δικτύων ανανεώνονται.

Ο Αλγόριθμος του Logic Resolver

Ο ρόλος του Logic Resolver έγκειται στον επηρεασμό της κυκλοφορίας στο ομότιμο δίκτυο (Peer to Peer Network) δεδομένου πως απλοποιεί τα ερωτήματα που αποστέλλονται (queries).

Παράρτημα επεξήγησης αλγορίθμου

- *Local_Facts_l*: λίστα τοπικών γεγονότων
- *Local_Rules_l*: λίστα τοπικών κανόνων
- *connected_peers*: λίστα συνδεδεμένων ομότιμων κόμβων.
- *tmp_net_facts*: λίστα προσωρινών γεγονότων δικτύου
- *unresolved_rules_l*, *present_unresolved_rules _l*: *τοπική λίστα κανόνων που χρησιμοποιούνται για την εφαρμογή του αλγορίθμου*
- *missing_facts_l*, *new_facts*, *network_response_l*: *τοπική λίστα γεγονότων που χρησιμοποιούνται για την εφαρμογή του αλγορίθμου*
- *empty_list*: μια άδεια λίστα

Οι παράμετροι του *Logic_Resolver*:

- *Facts_l:* list of facts
- *Rules_l*: list of rules

Αλγόριθμος

```
Facts_l = Local_Facts_l
Rules_l = Local_Rules_l

Logic_Resolver( Facts_l, Rules_l )
present unresolved_rules _l = empty_list
present_unresolved_rules_l =
        try_evaluate( Facts_l, Rules_l + tmp_net_facts )
while( unresolved_rules _l !=  present_unresolved_rules_l )
        unresolved_rules = present_unresolved_rules_l
        present_unresolved_rules_l =
                try_evalueate( Facts_l, Rules_l + tmp_net_facts )
        Notify_Network_Component( Facts_l )
end while
missing_facts_l =
        retrieve_missing_facts( Facts_l, present_unresolved_rules )
Ask_Peers( missing_facts_l, connected_peers  )
network_response_l = read_from_peers()
if(  network_response_l != empty_list )
        Facts_l = network_response + Facts_l
        unresolved_rules_l = try_evalueate( Facts_l,Rules_l )
end if
```

return *unresolved_rules_l*

- try_evaluate(Facts_l, Rules_l) χρησιμοποιείται για να αξιολογήσει τους κανόνες στον κατάλογο των κανόνων (Rules_l). Εάν ένας κανόνας ισχύει, το αποτέλεσμά του επισυνάπτεται στον κατάλογο των γεγονότων (Facts_l). Οι κανόνες που δεν μπορούν να επιλυθούν επιστρέφονται σε έναν κατάλογο.

- *rule*: αντιπροσωπεύει έναν κανόνα

- *new_facts*: τοπικός κατάλογος γεγονότων που χρησιμοποιούνται για να εκτελεστεί ο αλγόριθμος

- *unresolved_rules_l*: τοπικός κατάλογος κανόνων που χρησιμοποιούνται για να εκτελέσουν τον αλγόριθμο

Οι παράμετροι του try_evaluate

- *Facts_l:* κατάλογος γεγονότων

- *Rules_l*: κατάλογος κανόνων

try_evaluate(*Facts_l*, *Rules_l*)

foreach(*rule* **in** *Rules_l*)

if(*check_rule*(*rule*, *Facts_l*) == **TRUE**)

new_facts = *rule_result*(*rule*, *Facts_l*)

Facts_l = *new_facts* + *Facts_l*

end if

else

unresolved_rules_l = *rule* + *unresolved_rules_l*

end else

end foreach

return *unresolved_rules_l*

Οι συνθήκες

- retrieve_missing_facts(Rules_l, Facts_l),
- check_rule(rule, Rules_l),
- Notify_Network_Component(Facts_l)
- rule_result(rule)

θεωρούνται ασήμαντες και ο ψευδοκώδικας τους παρακάμπτεται.

Η συνθήκη

- *retrieve_missing_facts*(*Rules_l*, *Facts_l*)

έχει ως εισαγωγή έναν κατάλογο κανόνων, και ως παραγωγή ένας κατάλογος γεγονότων που είναι απαραίτητα για το ψήφισμα των κανόνων.

Η συνθήκη

- *check_rule*(*rule*, *Facts_l*)

έχει ως εισαγωγή έναν κανόνα και έναν κατάλογο γεγονότων (Facts_l), και ως παραγωγή μια λογική έκφραση. Τα γεγονότα που δεν διατίθενται αλλά απαιτούνται για να αξιολογηθεί ο κανόνας, τίθενται ως Ψευδή.

Η συνθήκη

- *Notify_Network_Component*(*Facts_l*)

έχει ως εισαγωγή τον πρόσφατα ανανεωμένο κατάλογο γεγονότων. Με αυτήν την συνάρτηση, δηλώνουμε στο τμήμα δικτύων με τα νέα γεγονότα.

Η συνθήκη

- *rule_result*(*rule*, *Facts_l*)

έχει ως εισαγωγή έναν κανόνα και έναν κατάλογο γεγονότων (Facts_l), και ως παραγωγή το αποτέλεσμα που προέρχεται από τον κανόνα.

Η συνθήκη

- *Ask_Peers*(*missing_facts_l*, *connected_peers*)

είναι μια συνάρτηση του τμήματος δικτύων και θα επεξηγηθεί αναλυτικότερα στη συνέχεια

Η συνθήκη

- *read_from_peers*()

είναι μια λειτουργία του τμήματος δικτύων. Το μόνο ξεχωριστό σε αυτήν την λειτουργία είναι το γεγονός πως η σύγκριση για τις τιμές εμπιστοσύνης στην αμφισβητούμενη απάντηση των ομότιμων κόμβων εκτελείται εδώ.

Η λογική του Network Component

Αυτή η μελέτη προτείνει ένα P2P Logic Resolver, κατά συνέπεια ένα καθαρό P2P δίκτυο εφαρμόστηκε. Θα εστιάσουμε τόσο στο σχέδιο, όπου οι βασικοί κανόνες εφαρμόστηκαν όσο και στο δίκτυο FSM (finite state machine).

Κάθε ομότιμος κόμβος είναι μοναδικός (μοναδικό όνομα, μοναδική διεύθυνση δικτύων). Αρχικά ο χρήστης καλείται για να συμπληρώσει το όνομά και τον επιθυμητό αριθμό θύρας όπου θα γίνεται η σύνδεση. Δίνουμε αυτήν την ελευθερία στο χρήστη, ώστε να μπορούν να δρομολογηθούν διαφορετικές περιπτώσεις του προγράμματος για την ίδια μηχανή.

Ένα νήμα Listener δημιουργείται που περιμένει τα εισερχόμενα αιτήματα για τη σύνδεση. Κάθε φορά που λαμβάνει ένας ομότιμος κόμβος ένα αίτημα από έναν άλλο κόμβο που δεν συνδέεται, ακόμα κι αν δεν είναι γνωστός σε αυτόν, επαναπροσανατολίζεται το κόμβο σε μια νέα θύρα. Κατόπιν δημιουργεί ένα νέο νήμα που διαχειρίζεται τα μηνύματα μεταξύ τους. Εάν λαμβάνει ένα αίτημα για τη σύνδεση, από έναν κόμβο που συνδέθηκε ήδη, το αίτημα απορρίπτεται. Χρησιμοποιώντας τον προαναφερθέντα έλεγχο, αποφεύγουμε

- (α) να συνδέονται χρήστες με το ίδιο όνομα
- (β) τη δημιουργία μη απαραίτητων νημάτων που διαχειρίζονται την ίδια σύνδεση.

Ο χρήστης αποθηκεύει τις γνωστές διευθύνσεις των κόμβων σε ένα αρχείο. Για κάθε κόμβο που επιθυμεί να συνδεθεί, αρχίζει ένα νέο νήμα Connector που χειρίζεται τα μεταξύ τους μηνύματα. Μπορεί επίσης να συνδεθεί με έναν νέο κόμβο και να τον προσθέσει στο αρχείο. Έχουμε τον ίδιο έλεγχο όπως στον ακροατή νημάτων για τις εισερχόμενες συνδέσεις, όταν προσπαθούμε να συνδεθούμε με ένα νέο κόμβο.

Ο συγχρονισμός νημάτων είναι ζωτικής σημασίας για την εφαρμογή για να εργαστεί κανονικά.
Κάθε νήμα τρέχει έναν βρόχο του δικτύου FSM. Οι βασικές καταστάσεις του βρόχου είναι

- Ping
- Pong
- Query
- Answer
- Exit

Επεξήγηση Καταστάσεων Βρόχου

- PING, PONG: αυτές οι καταστάσεις χρησιμοποιούνται για την ανίχνευση του δικτύου και ελέγχεται ποιος κόμβος συνδέεται

- QUERY: αυτή η κατάσταση περιγράφει τις ενέργειες που πραγματοποιεί ένας κόμβος που λαμβάνει ερωτήματα. Εάν τα γεγονότα που ζητούνται είναι γνωστά σε αυτόν τότε επιστρέφεται μια έκθεση στον κόμβο που αιτήθηκε την αναφορά. Εάν κάποια ή κανένα γεγονότα δεν είναι γνωστά (είτε τοπικά, είτε προσωρινά γεγονότα δικτύων) τότε διαβιβάζεται το ερώτημα σε όλους τους γνωστούς κόμβους που δεν περιλαμβάνονται στο ιστορικό της ερώτησης. Κατόπιν αναμένεται η απάντησή τους έως ότου λήξει το TTL της ερώτησης. Τέλος, εάν η απάντηση από τους ομότιμους κόμβους της δεν περιέχει όλα τα απαιτημένα γεγονότα, ελέγχεται εάν μπορεί να εξαγάγει αυτά τα γεγονότα από τους τοπικούς κανόνες που δεν μπορεί να αξιολογήσει. Εάν αυτό είναι δυνατό τότε ζητά τα συγκεκριμένα ελλείποντα γεγονότα Αφότου ολοκληρώνεται η προαναφερθείσα διαδικασία, τα γεγονότα που λαμβάνονται από τους ομότιμους κόμβους συγχωνεύονται με την απάντησή και αποστέλλονται στο κόμβο που πραγματοποίησε την ερώτηση. Επιπλέον εκείνα τα γεγονότα αποθηκεύονται στους προσωρινούς απομονωτές, οι προσωρινοί απομονωτές δικτύων (Temporary Network Buffers). Αυτοί οι απομονωτές λειτουργούν χρησιμοποιώντας έναν αλγόριθμο βασισμένο στον «πιο ελάχιστα πρόσφατα χρησιμοποιημένο» αλγόριθμο (Least Recently Used) [11]

- Answer: Αυτή η κατάσταση περιγράφει τις ενέργειες που ο κόμβος πραγματοποιεί όταν λαμβάνει μια απάντηση από έναν άλλο ομότιμο κόμβο

- Exit: αυτή η κατάσταση περιγράφει τις ενέργειες που πραγματοποιεί ένας κόμβος όταν ένας άλλος αποσυνδέεται από αυτόν.

Το Network Component εμπεριέχει τη συνάρτηση

- Ask Peers (missing facts, connected peers).

Το σχέδιό του, αν και απλό, μειώνει τις ερωτήσεις δικτύων, ως εκ τούτου επιταχύνει την απάντηση των κόμβων σε μια όμοια ερώτηση. Προτού διαβιβαστεί ένα ερώτημα δημιουργείται ένα ιστορικό ερωτημάτων σε αυτό. Γεγονός που σημαίνει πως ένας κατάλογος που τους ερωτηθέντες κόμβους για τα ελλείποντα γεγονότα δημιουργείται και αποστέλλεται πριν το ερώτημα.
Ο δέκτης ελέγχει αυτόν τον κατάλογο, και εάν πρέπει να διαβιβάσει την ερώτηση στους συνδεδεμένους κόμβους, αποκλείει τους κόμβους που βρίσκονται σε εκείνο τον κατάλογο.

- Peer: ένας συγκεκριμένος κόμβος
- connected_peers: οι συνδεδεμένοι κόμβοι
- query: flag δικτύου που δείχνει πως ένα ερώτημα ακολουθεί
- send(data): συνθήκη που στέλνει δεδομένα στο κόμβο

Οι παράμετροι της συνάρτησης Ask_Peers:

- missing_facts*:* κατάλογος με τα άγνωστα δεδομένα που θα αιτηθούν από τους άλλους κόμβους
- connected_peers: κατάλογος με τους συνδεδεμένους κόμβους

Ο αλγόριθμος της συνάρτησης Ask Peers και οι παραλλαγές της

Ask_Peers(*missing _facts*, *connected_peers*)

foreach(*peer* **in** *connected_peers*)
send(**query**)
send(*connected_peers*)
send(*missing_facts*)
end foreach

Η παρούσα συνάρτηση χρησιμοποιείται από τον κόμβο που πραγματοποιεί το αρχικό ερώτημα ενώ οι κόμβοι που θέλουν να προωθήσουν το ερώτημα χρησιμοποιούν μια παραλλαγμένη συνάρτηση.

Ορίζοντας ως:

- query_history: ένα κατάλογο με τους κόμβους που έχει αποσταλεί ήδη το ερώτημα
- is_forwarded: ένα flag δικτύου που υποδηλώνει πως ένα προωθημένο ερώτημα ακολουθεί
- Mapping(list): Μια συνθήκη που επιστρέφει ένα κατάλογο όπου τα επαναλαμβανόμενα στοιχεία του καταλόγου αφαιρούνται από το ερώτημα

Ακολουθεί η παραλλαγή του αλγορίθμου για τους κόμβους που προωθούν το ερώτημα:

```
Ask_Peers( missing _facts, connected_peers, query_history )
foreach( peer in connected_peers)
    if( ( connected_peers contains peer ) == FALSE )
        send( query )
        send( is_forwarded )
        send( Mapping ( connected_peers + query_history ) )
        send( missing_facts )
    end if
end foreach
```

Η συνάρτηση Query

Η συνάρτηση Query αποτελεί τη καρδιά των δικτύων Finite-state machine

Επεξηγήσεις αλγορίθμου Query

- query_history : κατάλογος με τους ήδη ερωτηθέντες, από τον κόμβο που δημιουργήθηκε το ερώτημα, κόμβους
- read_from_peer(): συνάρτηση που χρησιμοποιείται για την ανάγνωση δεδομένων από ένα κόμβο. Αξιοσημείωτο σε αυτήν την συνάρτηση είναι ότι η σύγκριση για τις τιμές εμπιστοσύνης στην αμφισβητούμενη απάντηση των κόμβων πραγματοποιείται σε αυτό το τμήμα του αλγορίθμου.
- raw_query: τοπική μεταβλητή που χρησιμοποιείται για να ελέγξει αν ο κόμβος προωθεί το ερώτημα ή αιτείται ελλείποντα γεγονότα.
- facts_asked: τοπική μεταβλητή που χρησιμοποιείται για την αποθήκευση των παρελληφθέντων ελλειπόντων γεγονότων.
- Fact: το γεγονός που επεξεργάζεται λογικά.
- local_facts_l: τα τοπικά γεγονότα του κόμβου.
- my_answer: μεταβλητή όπου οι απαντήσεις των κόμβων που πρόκειται να αποσταλούν υποβάλλονται σε επεξεργασία.
- unknown_facts: κατάλογος γεγονότων που δεν μπορούν να απαντηθούν με τις τοπικές συνθήκες του κόμβου.
- tmp_net_facts: προσωρινή μνήμη των γεγονότων δικτύου.
- peers_answer: απάντηση στα ερωτήματα που προωθούνται στο κόμβο.
- result: μεταβλητή που χρησιμοποιείται για να ανακτήσει τα γεγονότα από έναν τοπικό κανόνα που έχει υποτεθεί ως αληθής.
- assume_valid(rule): συνθήκη που χρησιμοποιείται για να επιστρέψει τα γεγονότα από έναν κανόνα που έχει υποτεθεί ως αληθής.
- my_unknown_facts: κατάλογος γεγονότων, του τρέχοντος κόμβου, απαραίτητων ώστε να ισχύσουν οι τοπικοί κανόνες.
- peers_answer_me: κατάλογος απαντημένων γεγονότων του τρέχοντος κόμβου απαραίτητων ώστε να ισχύσουν οι τοπικοί κανόνες.

- Answer: σημαία δικτύου που υποδηλώνει πως ακολουθεί απάντηση ερωτήματος.

Ο Αλγόριθμος της συνάρτησης Query

Query()

query_history = *read_from_peer*()
raw_query = *read_from_peer*()

if (raw_data **contains** "**is_forwarded**")

    raw_query = *read_from_peer*()

end if

facts_asked = *raw_query*

foreach (*fact* **in** *facts_asked*)
        if (*local_facts_l* **contains** *fact*)
            my_answer += *fact*
        end if
end foreach

unknown_facts = *facts_asked* - **Mapping**(*local_facts* + *tmp_net_facts*)

if (*unknown_facts* != *empty_list*)
    Ask_Peers(*unknown_facts, connected_peers, query_history*)
    *peers_a*nswer = *read_from_peers()*
end if

if(*peers_answer* **!=** *unknown_facts*)
    tmp_net_facts += *peers_answer*
    my_answer += *peers_answer*
    foreach(*rule* **in** *local_rules_l*)
        result = *assume_valid(rule)*
        if (*unknown_facts* **contains** *result*)
            my_unknown_facts += *result*
        end if
    end foreach

```
end if


if ( my_unknown_facts != empty_list )
        Ask_Peers( my_unknown_facts, connected_peers )
        peers_answer_me = read_from_peers()
end if

if ( peers_answer_me != empy_list )
        tmp_net_facts += peers_answer_me
        my_answer += peers_answer_me
end if

send( answer )
send( my_answer )
```

P2P Resolver σε λειτουργία

Οι αλγόριθμοι και οι τεχνικές που χρησιμοποιούνται στη παρούσα μελέτη παρουσιάζονται στο παρακάτω παράδειγμα.

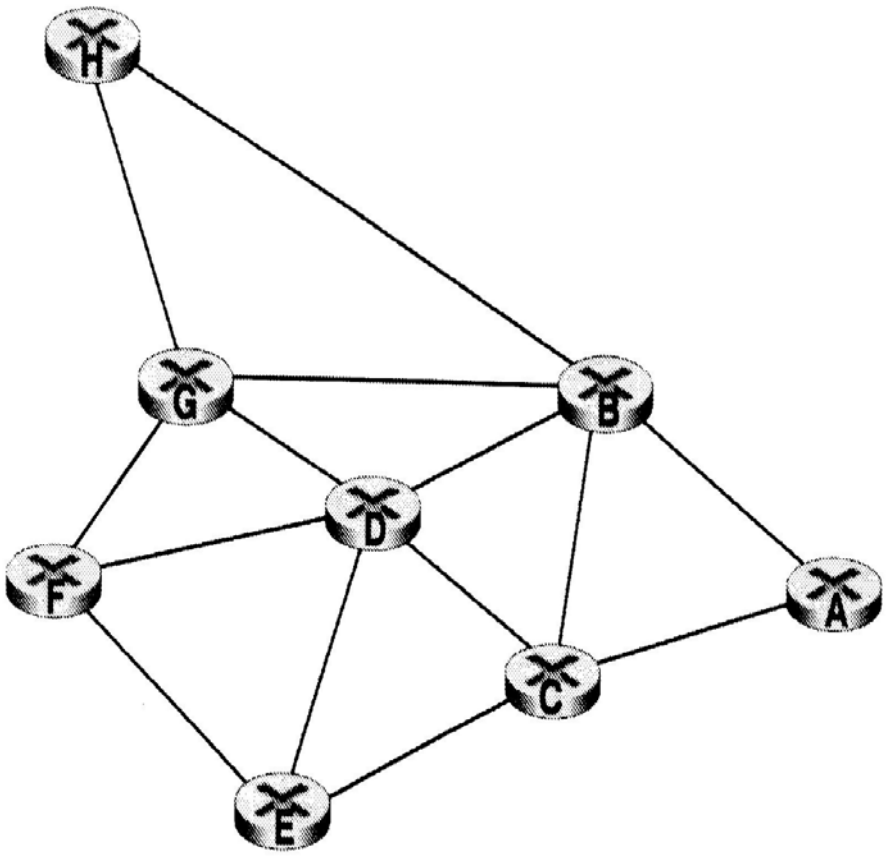


Figure 1. A P2P System of Multiple Logic Problems

Οι κόμβοι στο παρών δίκτυο περιέχουν δεδομένα και συνθήκες που περιγράφονται στον πίνακα 1. Υποθέτουμε πως ο κόμβος D πρέπει να λύσει το λογικό πρόβλημα $a \wedge b \wedge c \wedge d \rightarrow e$

Nodes	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**
Facts	g…	…	c…	f	$\neg$c…	d…	d…	b…
Rules	…	$g \rightarrow b$	…	$f \rightarrow a$, $a \wedge b \wedge c \wedge d \rightarrow e$	…	…	…	…

Table 1. Facts and Rules of network's nodes

Και δε μπορεί να το επιλύσει με τις τοπικές συνθήκες και τους τοπικούς κανόνες που διαθέτει.

Βήματα επίλυσης λογικού προβλήματος

- Στο πρώτο βήμα, ο κόμβος D θα προσπαθησει να επιλύσει το πρόβλημα κάνοντας χρήση των τοπικών γνώσεων του. Από αυτές θα εξαγάγει το στοιχεία “a” αλλά χρειάζεται ακόμη 3 στοιχεία.

- Στο δεύτερο βήμα, δημιουργεί ένα ερώτημα που αποστέλλει στους γνωστούς του ομότιμους κόμβους. Ένας κατάλογος των ερωτηθέντων κόμβων αποστέλλεται.

- Στο τρίτο βήμα, κάθε κόμβος συνδεδεμένος με το κόμβο D αναζητά τα γεγονότα που αιτήθηκαν. Εάν έχουν κάποιο ή κανένα στοιχείο από αυτά που ερωτήθηκαν, θα προωθήσουν το ερώτημα στους κόμβους που δεν έχουν ερωτηθεί ακόμη βάση του ιστορικού της διαδρομής του ερωτήματος που έχουν παραλάβει. Οι κόμβοι B, C, E, F, G δεν θα αλληλοερωτηθούν για αυτές τις συνθήκες.

- Ο κόμβος B δεν έχει απευθείας απάντηση για το στοιχείο “b” αλλά μπορεί να το εξαγάγει από τις τοπικές του γνώσεις, επομένως ζητά το στοιχείο g από τους γνωστούς του κόμβους και ο κόμβος A παρέχει το στοιχείο g.

- Ο κόμβος H βρίσκεται σε μεγάλη απόσταση από το κόμβο D. Υποθέτουμε πως η απάντηση δε θα έχει αποσταλεί εντός του TTL (time to leave) του ερωτήματος.

- Σημειώστε ότι οι κόμβοι C και E έχουν αντικρουόμενα γεγονότα.

- Υποθέτοντας πως η τιμή εμπιστοσύνης του C είναι μεγαλύτερη από την τιμή εμπιστοσύνης του E τότε η απάντηση του C θα εκληφθεί ως δεδομένη.

- Όταν το TTL περάσει, ο κόμβος D παραλαμβάνει όλα τα γεγονότα που ζήτησε και μπορεί να επιλύσει το λογικό πρόβλημα.

Κεφάλαιο 4ο

Συμπεράσματα

Στη παρούσα πτυχιακή εργασία παρουσιάζεται ένα μοντέλο επίλυσης λογικών προβλημάτων σε ένα «καθαρό» περιβάλλον ομότιμου δικτύου .

Τα γεγονότα και οι συνθήκες είναι κατανεμημένα στους ομότιμους κόμβους και δεν βρίσκονται αποθηκευμένα σε ένα κεντρικό διακομιστή, επιτρέποντας έτσι την βελτιστοποίηση της απόδοσης της διαδικασίας επίλυσης.

Οι αλγόριθμοι που προτείνονται απλοποιούν τα τοπικά λογικά προβλήματα καθώς επίσης μειώνουν τον αριθμό των ερωτημάτων που ανταλλάσσονται στο δίκτυο. Με αυτή την αλγοριθμική διαδικασία δεν απαιτούνται ερωτήματα για τα ελλείποντα δεδομένα στους κόμβους καθώς επίσης και η επανάληψη των ίδιων ερωτήσεων.

Γενικότερα, ο τρόπος αντιμετώπισης των λογικών προβλημάτων στο παρών peer to peer δίκτυο μπορεί να βρει εφαρμογή και σε άλλες θεωρήσεις της επιστήμης της πληροφορικής.

Ειδικότερα, είναι γενικά αποδεκτό ότι η χρήση τέτοιων δικτύων ενώνει χρήστες από όλο τον κόσμο λειτουργώντας χωρίς λογοκρισία, ελέγχους ή φραγμούς προάγοντας τη βασική ιδέα της δημιουργίας του παγκοσμίου ιστού που δεν είναι άλλη από την ελεύθερη διακίνηση ιδεών και τη δωρεάν παροχή υπηρεσιών και πληροφοριών.

Η απλή δομή, το μηδαμινό κόστος και η άναρχη ροή πληροφορίας είναι τα στοιχεία που καθιστούν τη λειτουργία των P2P δικτύων μοναδική. Η φιλοσοφία τους δίνει τη δυνατότητα στους συμμετέχοντες της δημιουργίας δυναμικά αναπτυσσόμενων χώρων, το περιεχόμενο των οποίων καθορίζεται από τους ίδιους τους χρήστες.

Παράρτημα

Περίληψη Εφαρμογής

Η πρακτική εφαρμογή της λογικής προσέγγισης των ομότιμων δικτύων βασίζεται στη δημιουργία ενός προγράμματος βασισμένο σε αντικειμενοστραφή προγραμματισμό java για την επίτευξη της μέγιστης δυνατής συμβατότητας μεταξύ πλατφόρμων.

Η εφαρμογή στηρίζεται στη λογική της ανταλλαγής άμεσων μηνυμάτων μεταξύ των κόμβων του δικτύου μέσω των οποίων μπορεί να γίνεται ο έλεγχος των λογικών συνθηκών και γεγονότων που διέπουν το δίκτυο.

Ειδικότερα, η εφαρμογή βασίζεται στη δημιουργία δυο εκτελέσιμων αρχείων με την δυνατότητα να εκτελούνται τοπικά σε κάθε κόμβο διατηρώντας τη δυνατότητα αλληλεπίδρασης με τους συνδεδεμένους κόμβους. Για τον λόγο αυτό δημιουργήθηκαν 3 κλάσεις στις οποίες στηρίζεται η εφαρμογή

- Client. Java
- Server. Java
- Message. Java

Η κλάση Message

Η κλάση Message αποτελεί το σύνδεσμο μεταξύ των δύο κύριων κλάσεων καθώς πρόκειται για το μήνυμα που ανταλλάσσεται μεταξύ των κόμβων.

Κώδικας Κλάσης

```
import java.io.Serializable;


public class Message implements Serializable
 {
    public volatile String senderID;
    public volatile String msgText;

@Override
    public String toString() {
        return ""+senderID + " >> " + msgText;
    }
```

Η κλάση Client. Java

Πρόκειται για την κλάση που πραγματοποιεί τη σύνδεση και την αποστολή των μηνυμάτων μεταξύ των κόμβων.

Κώδικας Κλάσης

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import java.util.*;



class client extends JFrame implements ActionListener,Runnable
{
	Socket socket = null;
	JLabel l1,l2,l3,l4,l5,l6,jtf3;
	JTextField jtf1,jtf2,jtf4,jtf5;
	TextArea ta;
	Message msg = new Message();
	InetAddress host;
	int port = 5000;
	Thread t = null;
	JButton jb,jb2,jb3;

	client(String s)
	{
		super(s);

		myadapter a = new myadapter(this);
		addWindowListener(a);

		l5 = new JLabel("Enter IP : ");
		add(l5);
		jtf4 = new JTextField(15);
		add(jtf4);
		jtf4.setText("127.0.0.1");
```

```
add(new JLabel("                          "));

l6 = new JLabel("Enter Port : ");
add(l6);
jtf5 = new JTextField(15);
add(jtf5);
jtf5.setText("5000");

l1 = new JLabel("Nick Name :  ");
add(l1);
jtf1 = new JTextField(15);
add(jtf1);

jb2 = new JButton("Connect");
add(jb2);
jb2.addActionListener(this);

jb3 = new JButton("Disconnect");
add(jb3);
jb3.addActionListener(this);
jb3.setEnabled(false);

add(new JLabel("                                    "));

l2 = new JLabel("Message : ");
add(l2);
jtf2 = new JTextField(34);
add(jtf2);
jtf2.setEditable(false);

jb = new JButton("Send Message");
add(jb);
jb.addActionListener(this);
jb.setEnabled(false);

l3 = new JLabel("Status : ");
add(l3);
jtf3 = new JLabel("Not connected to the server...");
add(jtf3);
```

```
        add(new                                                JLabel("
"));


        l4 = new JLabel("Recieved Messages : ");
        add(l4);
        ta = new TextArea("",15,80);
        add(ta);
        ta.setFont(Font.getFont("verdana"));
        ta.setBackground(Color.ORANGE);
        ta.setEditable(false);

        jtf3.setText("Not connected to Server, click connect");

    }

    public void actionPerformed(ActionEvent ae)
    {
        try{
            String str = ae.getActionCommand();

            if(str.equals("Disconnect"))
            {
                try
                {
                    jb.setEnabled(false);
                    jtf2.setEditable(false);
                    jb2.setEnabled(true);
                    jb3.setEnabled(false);
                    jtf4.setEditable(true);
                    jtf5.setEditable(true);
                    jtf1.setEditable(true);
                    socket.close();
                    socket = null;

                }
                catch(Exception e)
                {}
            }
```

```
                if(str.equals("Send Message"))
                {
                        msg.senderID = jtf1.getText();

                        msg.msgText = jtf2.getText();
                        jtf2.setText("");

                        if(!msg.senderID.equals("")                &&
!msg.msgText.equals(""))
                        {
                                sendData();
                        }
                        else
                        jtf3.setText("Message was not sent, type a
message");
                }

                if(str.equals("Connect"))
                {
                        try{
                        host                                        =
InetAddress.getByName(jtf4.getText());
                        String p = jtf5.getText();

                        try{
                                if(socket!=null)
                                {
                                        socket.close();
                                        socket = null;
                                }
                                }
                                catch(Exception e)
                                {}

                        if(!jtf1.getText().equals(""))
                        {
```

```
                    socket = new
Socket(host,Integer.parseInt(p));



                    ObjectOutputStream obj = new
ObjectOutputStream(socket.getOutputStream());
                    msg.senderID = jtf1.getText();
                    msg.msgText = " is now online at "+new
Date().toString();
                    obj.writeObject(msg);

                    jtf2.setEditable(true);
                    jb.setEnabled(true);
                    jb2.setEnabled(false);
                    jb3.setEnabled(true);
                    jtf4.setEditable(false);
                    jtf5.setEditable(false);
                    jtf1.setEditable(false);

                    jtf3.setText("Connection established with
Server, start chatting");

                    t = new Thread(this,"Reading");
                    t.start();
                    }
                    }
                catch(Exception e)
                {
                    jtf3.setText("Could not connect to Server,
connect again");
                }
                }
                }
                catch(Exception e)
                {
                    jtf3.setText("Action Error");
                }
        }
```

```
public void sendData()
	{
		try
		{
			ObjectOutputStream obj = new
ObjectOutputStream(socket.getOutputStream());
			if(!msg.senderID.equals("") &&
!msg.msgText.equals(""))
			{
				obj.writeObject(msg);
				jtf3.setText("Message was sent
successfully");
			}
			msg.senderID = "";
			msg.msgText = "";
		}
		catch(Exception e)
		{
				jtf3.setText("Error occured while sending
message");
		}
	}


public void run()
	{
		try
		{

			while(true)
			{
				ObjectInputStream obj = new
ObjectInputStream(socket.getInputStream());
				Message msg = new Message();
				msg = (Message) obj.readObject();
```

```
                        if(msg.senderID!=null                    &&
msg.msgText!=null)
                        ta.append(msg.senderID+"                 >>
"+msg.msgText+"\n");

                    }

                }



catch(Exception e)
                {
                    jtf2.setEditable(false);
                    jb2.setEnabled(true);
                    jb.setEnabled(false);
                    jb3.setEnabled(false);
                    jtf4.setEditable(true);
                    jtf5.setEditable(true);
                    jtf1.setEditable(true);
                    jtf3.setText("Connection Lost");
                }
        }

}

class demo
{
        public static void main(String a[])
        {
                client f = new client("Messenger");
                f.setLayout(new FlowLayout());
                f.setSize(600,425);
                f.setResizable(false);
                f.setVisible(true);
        }
}
```

```
class myadapter extends WindowAdapter
{
	client f;
	public myadapter(client j)
	{
		f = j;
	}
	public void windowClosing(WindowEvent we)
	{
		f.setVisible(false);
		try{
			f.socket.close();
			f.dispose();
			}
			catch(Exception e)
			{
			}
			System.exit(0);
	}
}
```

Η κλάση Server. Java

Πρόκειται για τη κλάση που δέχεται τις συνδέσεις στον κόμβο καθώς και επεξεργάζεται τα δεδομένα για τις λογικές συνθήκες δημιουργώντας ένα ιστορικό με τα εισερχόμενα μηνύματα και ελέγχοντας τα.

Κώδικας Κλάσης

```
import java.net.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

class data {

    public volatile int signal;
    public volatile int count;
}

class ser extends JFrame implements ActionListener, Runnable {

    Thread t;
    JButton a, b;
    JTextField jtf2;
    JLabel jtf;
    TextArea ta;
    JList list;
    JButton compareButton;
    JTextField equalityTextField;
    ServerSocket server;
    Message msg = new Message();
    data flag = new data();
    Socket count[] = new Socket[5];
    int cj = 5;
    int cl = 0;
```

```
ser(String s) {
    super(s);

    flag.signal = 0;
    flag.count = 0;


    JLabel l3 = new JLabel("Enter Port No. : ");
    add(l3);

    jtf2 = new JTextField(7);
    jtf2.setText("5000");
    add(jtf2);

    JLabel l1 = new JLabel("Start the Server");
    add(l1);

    a = new JButton("Start");
    a.addActionListener(this);
    add(a);

    JLabel l2 = new JLabel("Stop the Server");
    add(l2);

    b = new JButton("Stop");
    b.addActionListener(this);
    add(b);
    b.setEnabled(false);

    JLabel l4 = new JLabel("Status : ");
    add(l4);

    add(new JLabel("    "));

    jtf = new JLabel("Server is not running...");
    add(jtf);

    ta = new TextArea("", 15, 70);
    ta.setEditable(false);
    ta.setBackground(Color.WHITE);
```

```
        ta.setFont(Font.getFont("verdana"));
        add(ta);

list = new JList(new DefaultListModel());
JScrollPane listScrollPane = new JScrollPane(list);
        listScrollPane.setPreferredSize(new Dimension(200, 200));
        add(listScrollPane);
compareButton = new JButton("Compare");
        compareButton.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {
                boolean flagSel=false;
                if (!list.isSelectionEmpty()) {
                    Object[] messages = list.getSelectedValues();
                    Message fMessage = (Message) messages[0];
                    for (int i = 1; i < messages.length; i++) {
                        Message message = (Message) messages[i];
                        System.out.println("Mess: " + message);
if (!(fMessage.msgText.equals(message.msgText))) {
                            flagSel=false;
                            break;
                        } else {
                            flagSel=true;
                        }
                    }
                    if(flagSel){
                        equalityTextField.setText("EQUALS!");
                    }else{
                        equalityTextField.setText("NOT EQUALS!");
                    }
                }
            }
        });
        add(compareButton);


        equalityTextField = new JTextField(10);
        equalityTextField.setEditable(false);
        equalityTextField.setText("Equality");
        add(equalityTextField);

        mywindowadapter a = new mywindowadapter(this);
```

```
        addWindowListener(a);


    }

    public void actionPerformed(ActionEvent ae) {
        try {
            String str = ae.getActionCommand();

            if (str.equals("Start")) {
                String str2 = jtf2.getText();
                if (!str2.equals("")) {
                    try {
                        server = new ServerSocket(Integer.parseInt(str2));
                        jtf.setText("Server is running....");
                        jtf2.setEnabled(false);

                        a.setEnabled(false);
                        b.setEnabled(true);

                        flag.count = 0;

                        count = new Socket[5];
                        cj = 5;
                        cl = 0;

                        t = new Thread(this, "Running");
                        t.start();

                    } catch (Exception e) {
                        jtf.setText("Either the port no. is invalid or is in
use");
                    }
                } else {
                    jtf.setText("Enter port no.");
                }
            }
            if (str.equals("Stop")) {
                try {
                    server.close();
                } catch (Exception ee) {
                    jtf.setText("Error closing server");
```

```
            }
            jtf.setText("Server is closed");
            jtf2.setEnabled(true);
            a.setEnabled(true);
            b.setEnabled(false);
            server = null;
            t = null;

            for (int i = 0; i < flag.count; i++) {
                try {
                    count[i].close();
                } catch (Exception e) {
                }
            }

        }
    } catch (Exception ex) {
    }
}

public void run() {

    while (true) {
        if (server.isClosed()) {
            return;
        }
        try {

            Socket client = server.accept();

            ObjectInputStream obj = new
ObjectInputStream(client.getInputStream());
            msg = (Message) obj.readObject();

            ta.append(msg.senderID + " >> " + msg.msgText + "\n");

            if (cl < cj) {
                count[cl] = client;
                cl++;
            } else {
                Socket temp[] = new Socket[cj];
                for (int i = 0; i < cj; i++) {
```

```
                temp[i] = count[i];
            }

            count = new Socket[cj + 5];
            for (int i = 0; i < cj; i++) {
                count[i] = temp[i];
            }
            count[cj] = client;
            cj = cj + 5;
            cl++;
        }

        flag.count = cl;

        for (int i = 0; i < flag.count; i++) {
            try {
                ObjectOutputStream objw = new
ObjectOutputStream(count[i].getOutputStream());
                objw.writeObject(msg);
            } catch (Exception e) {
            }
        }

        new newthread(client, msg, flag, count, this, server);

    } catch (Exception e) {
        jtf.setText("Server is stopped");
        jtf2.setEnabled(true);
        try {
            server.close();
        } catch (Exception ey) {
            jtf.setText("Error closing server");
        }

    }


  }


}
```

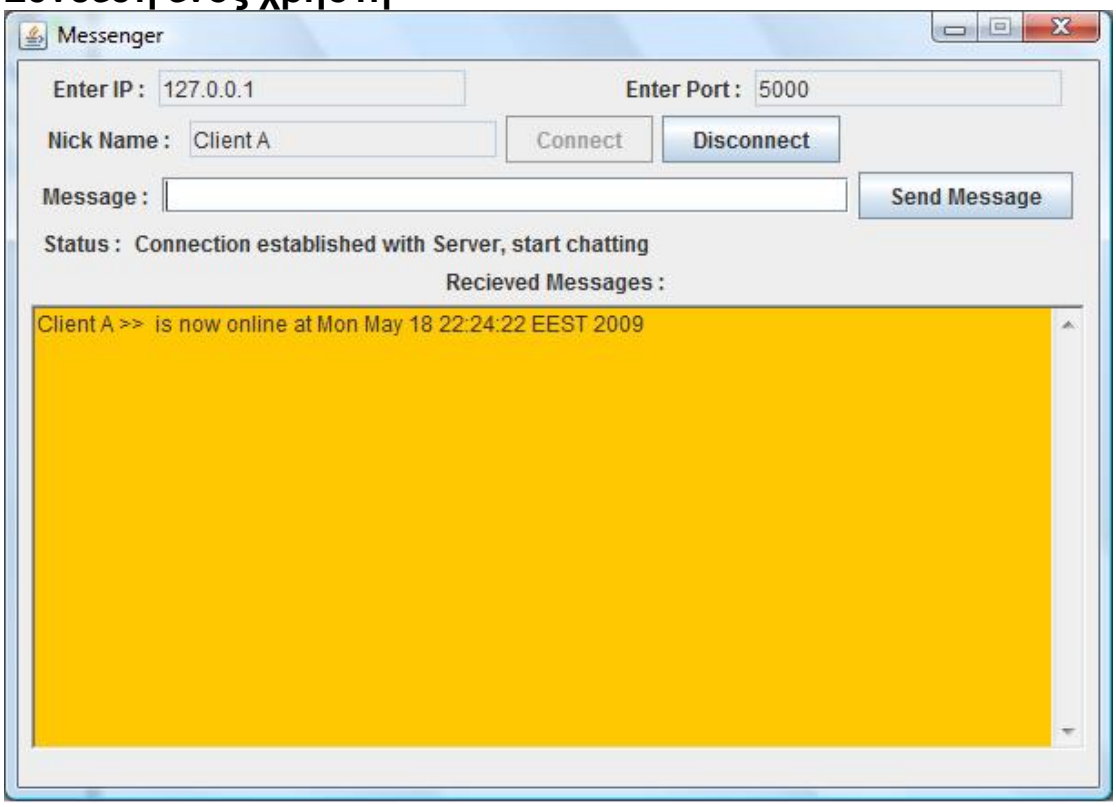
```
}


class server {

    public static void main(String a[]) throws IOException {
        ser f = new ser("Chat Server");
        f.setLayout(new FlowLayout());
        f.setSize(550, 550);
        f.setResizable(false);
        f.setVisible(true);
    }
}

class newthread implements Runnable {

    Thread t;
    Socket client;
    Message msg;
    data flag;
    Socket count[];
    ser f;
    ServerSocket server;

    newthread(Socket client, Message msg, data flag, Socket
count[], ser f, ServerSocket server) {
        t = new Thread(this, "Client");
        this.server = server;
        this.client = client;
        this.msg = msg;
        this.f = f;
        this.flag = flag;
        this.count = count;
        t.start();
    }
```

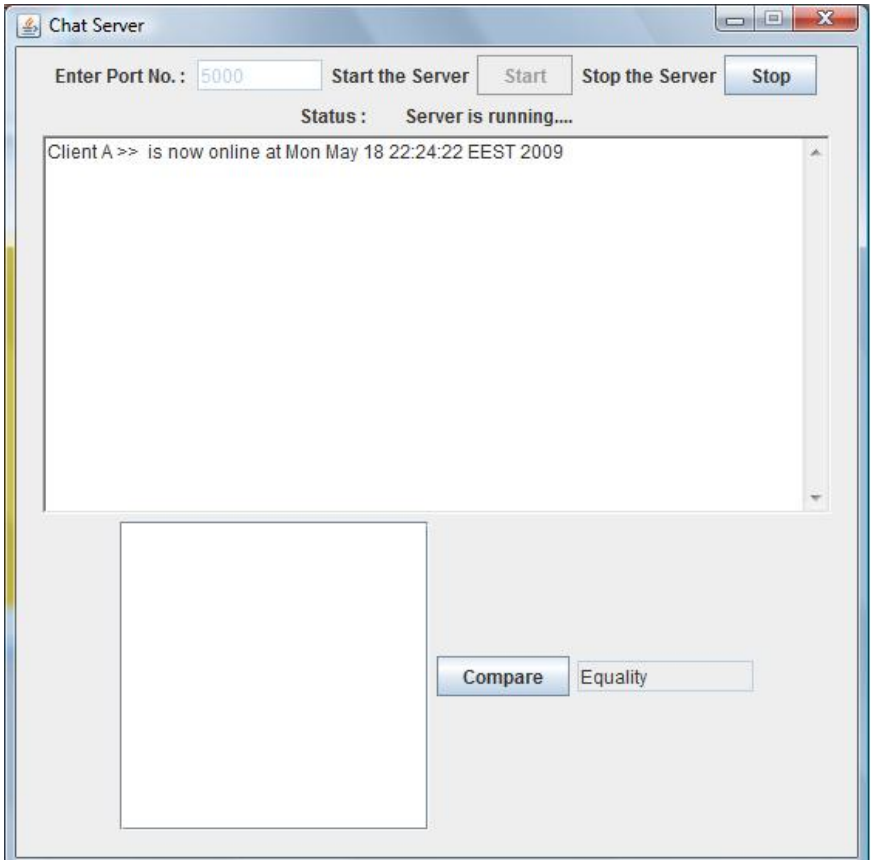
```
    public void run() {
        String name = msg.senderID;
        try {
            while (server.isClosed() != true) {
                ObjectInputStream obj = new
ObjectInputStream(client.getInputStream());
                msg = (Message) obj.readObject();
                if (msg.senderID != null && msg.msgText != null) {
                    f.ta.append(msg.senderID + " >> " + msg.msgText +
"\n");
                    ((DefaultListModel)
f.list.getModel()).addElement(msg);
                }
                name = msg.senderID;

                for (int i = 0; i < flag.count; i++) {
                    try {
                        ObjectOutputStream objw = new
ObjectOutputStream(count[i].getOutputStream());
                        objw.writeObject(msg);
                    } catch (Exception e) {
                    }
                }
            }

            if (server.isClosed()) {
                for (int i = 0; i < flag.count; i++) {
                    try {
                        count[i].close();
                    } catch (Exception e) {
                    }
                }
            }

        }
```

```
 catch (Exception e) {
          f.ta.append(name + " is offline\n");
          try {
             msg.msgText = " is offline\n";
             for (int i = 0; i < flag.count; i++) {
                try {
                   ObjectOutputStream objw = new
ObjectOutputStream(count[i].getOutputStream());
                   objw.writeObject(msg);
                } catch (Exception ex) {
                }
             }
             client.close();
          } catch (Exception ex) {
          }
       }
    }
}

class mywindowadapter extends WindowAdapter {

    ser f;

    public mywindowadapter(ser j) {
       f = j;
    }

    public void windowClosing(WindowEvent we) {
       f.setVisible(false);
       try {
          f.server.close();
       } catch (Exception e) {
       }
       f.dispose();
       System.exit(0);
    }
}
```

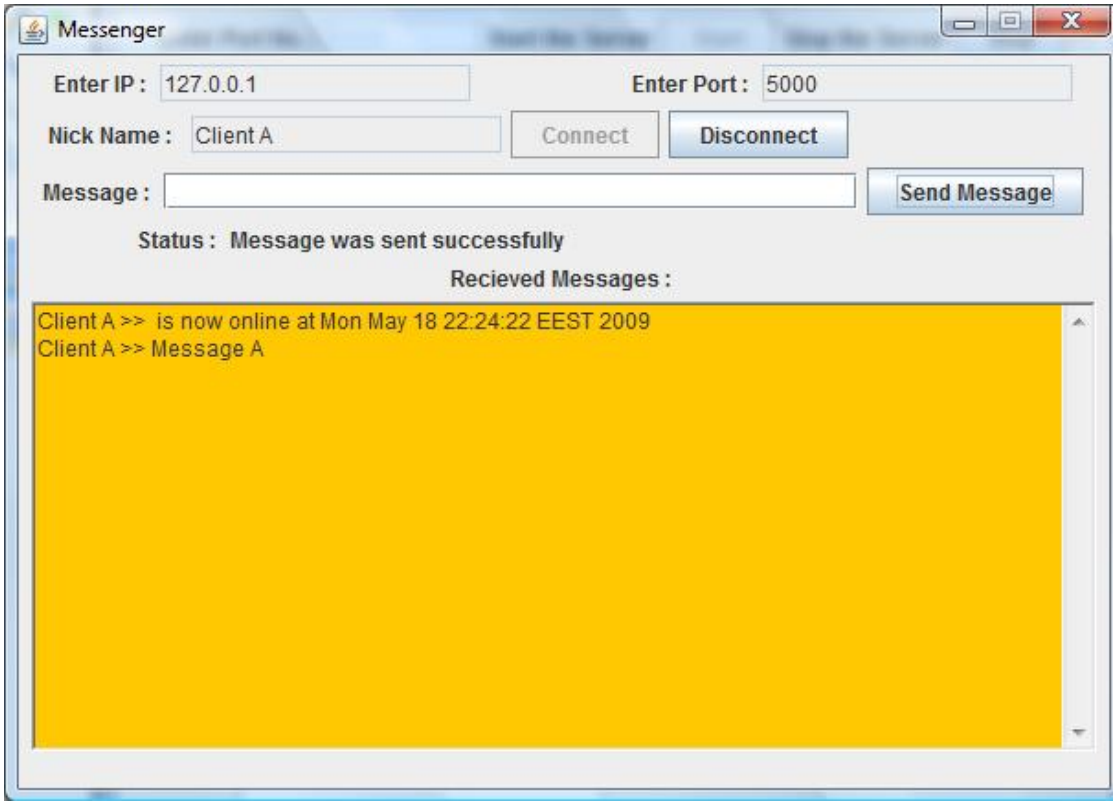
Παραδείγματα Χρήσης της Εφαρμογής

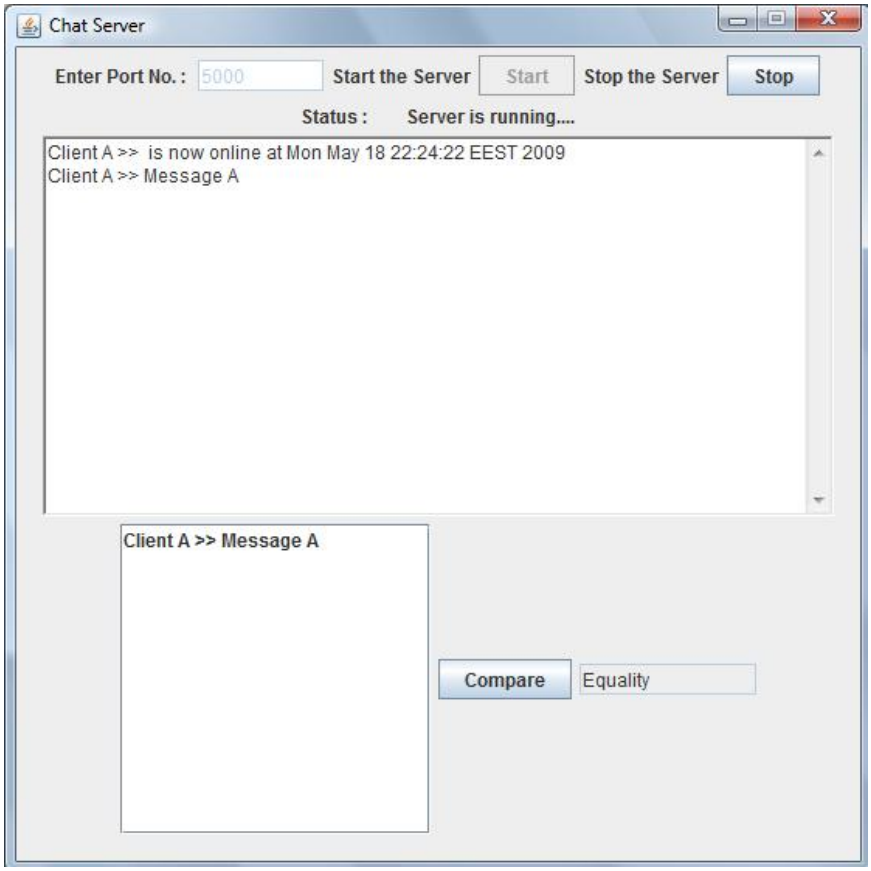
Σύνδεση ενός χρήστη





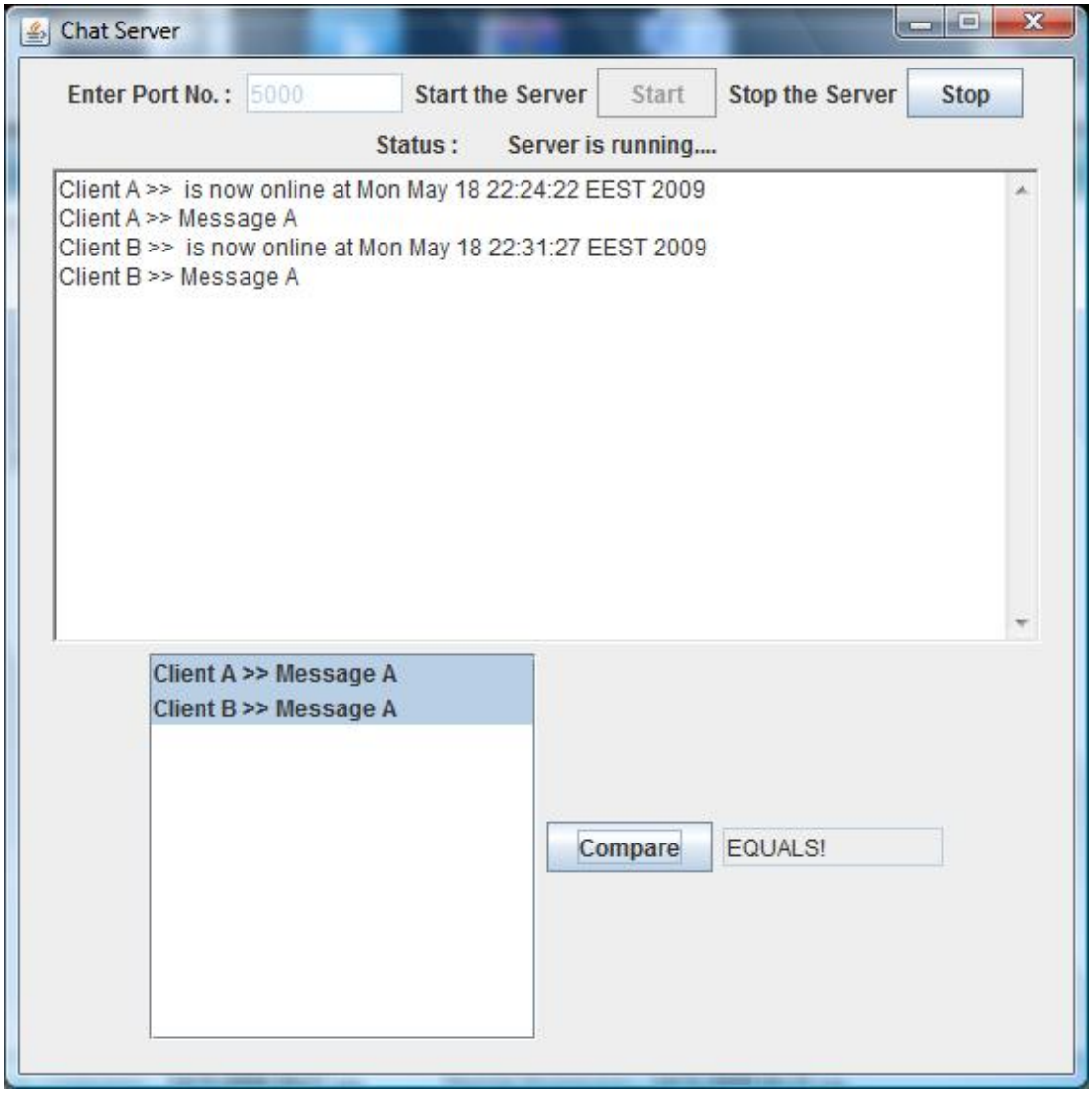
Αποστολή Μηνύματος



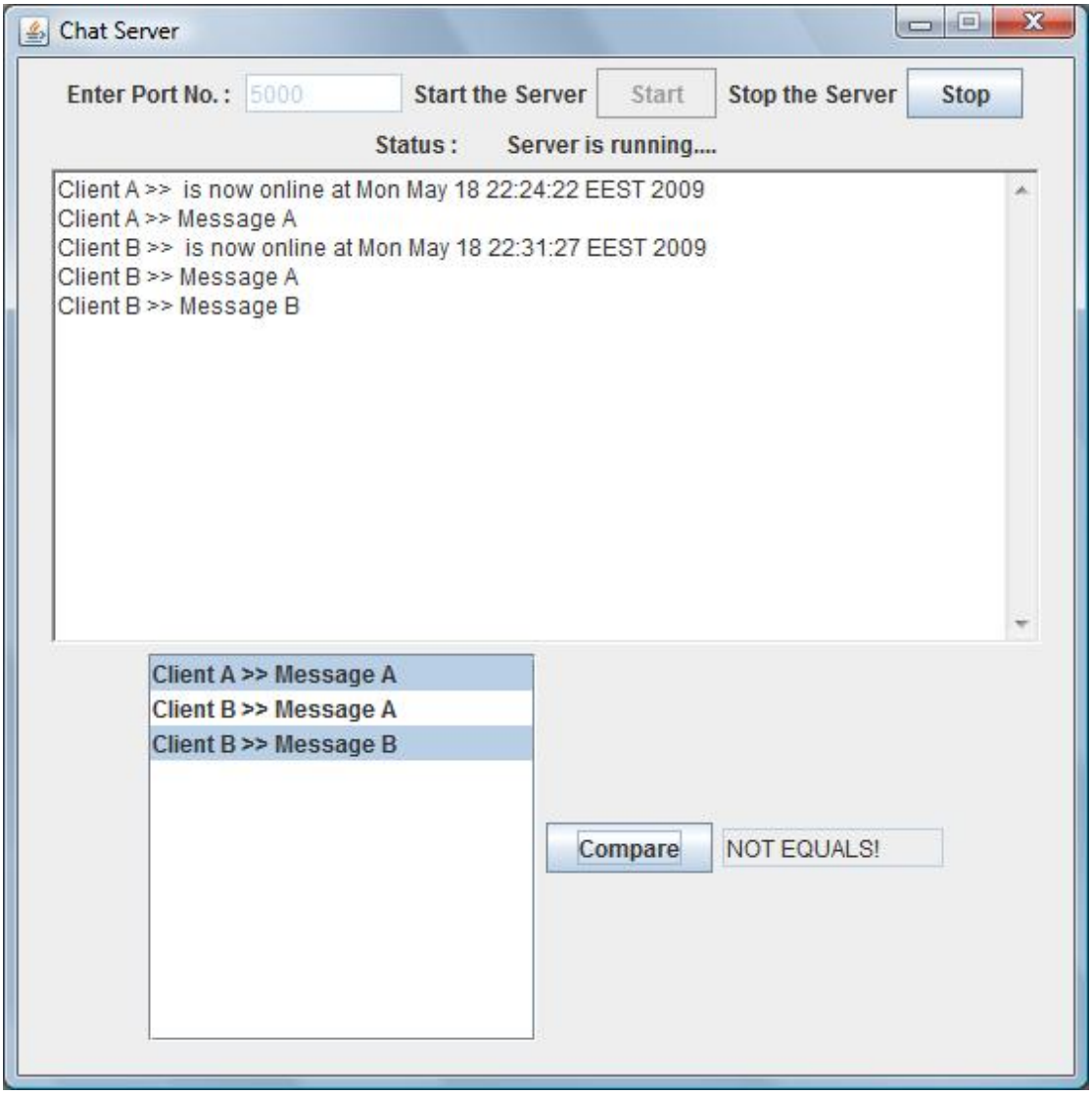


Έλεγχος Ισότητας Μηνυμάτων

Περίπτωση Αληθούς Σύγκρισης


Chat Server
Enter Port No. :
5000
Start the Server
Start
Stop the Server
Stop
Status :
Server is running....
Client A >> is now online at Mon May 18 22:24:22 EEST 2009
Client A >> Message A
Client B >> is now online at Mon May 18 22:31:27 EEST 2009
Client B >> Message A
Client A >> Message A
Client B >> Message A
Compare
EQUALS!

Περίπτωση Μη αληθούς Σύγκρισης



Βιβλιογραφία

[1] Yoav Shoham, Kevin Leyton-Brown, Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations Cambridge University Press; 2008.

[2] Michael Wooldridge. An Introduction to MultiAgent Systems. John Wiley & Sons: 2002.

[3] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati. Inconsistency Tolerance in P2P Data Integration: an Epistemic Logic Approach. Journal of Database Programming Languages; 2005. Volume 3774 of Lecture Notes in Computer Science; 90–105.

[4] Philippe Chatalic, Gia Hien Nguyen, Marie-Christine Rousset. Reasoning with Inconsistencies in Propositional Peer-to-Peer Inference Systems. European Conference on Artificial Intelligence; 2006; 352–356.

[5] G. Antoniou, A. Bikakis. Distributed Defeasible Reasoning in Multi-Context Systems. Journal on Non-Monotonic Reasoning; 2008.

[6] G. Antoniou and A. Bikakis. Distributed Reasoning with Conflicts in a Multi-Context Framework. First International Conference on Advanced Intelligence; 2008.

[7] Giunchiglia, F., Serafini, L. Multilanguage hierarchical logics, or: how we can do without modal logics. Journal in Artificial Intelligence; 1994; 65.

[8] Ghidini, C., Giunchiglia, F. Local Models Semantics, or contextual reasoning=locality+compatibility. Journal in Artificial Intelligence; 2001; 127; 221-259.

[9] Roelofsen, F., Serafini, L. Minimal and Absent Information in Contexts. International Joint Conderences on Artificial Intelligence; 2005; 558-563.

[10] Brewka, G., Roelofsen, F., Serafini, L. Contextual Default Reasoning. International Joint Conderences on Artificial Intelligence; 2007; 268-273.

[11] *Least Recently Used* (LRU) algorithm as described in: http://en.wikipedia.org/wiki/Cache_algorithms.

[12] G. Antoniou and A. Bikakis. Local and Distributed Defeasible Reasoning in Multi-Context Systems. European Conference of Ambient Inteligence; 2008; Under Review.