ΟΙΚΟΝΟΜΙΚΩΝ ΚΑΙ ΚΟΙΝΩΝΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΔΙΟΙΚΗΣΗΣ ΤΕΧΝΟΛΟΓΙΑΣ

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΠΑΡΑΛΛΗΛΩΝ ΚΑΤΑΝΕΜΗΜΕΝΩΝ ΣΥΣΤΗΜΑΤΩΝ ΥΠΟΛΟΓΙΣΤΩΝ ΜΕ JAVA

Πανεπιστημιακές Παραδόσεις

Μιχαηλίδης Παναγιώτης και Μαργαρίτης Κωνσταντίνος

ΘΕΣΣΑΛΟΝΙΚΗ 2008

Περιεχόμενα

Κατάλογος Σχημάτων

Πρόλογος

Τα τελευταία χρόνια υπάρχει τάση για επιτάχυνση υπολογισμών σε διάφορες σύγχρονες υπολογιστικές εφαρμογές όπως αριθμητική άλγεβρα, μετεωρολογία, προσομοιώσεις φυσικών φαινομένων και παγκόσμιας οικονομίας, βιοπληροφορική. Ταυτόχρονα υπάρχει και μια ραγδαία ανάπτυξη στο τομέα τηλεπικοινωνιών για διεκόλυνση της επικοινωνίας ή ανταλλαγή δεδομένων σε επιχειρηματικές εφαρμογές όπως ηλεκτρονικό εμπόριο - επιχειρείν και παροχή υπηρεσιών Διαδικτύου. Στην περίπτωση που οι υπολογιστικά απαιτητικές εφαρμογές απαιτούν τεράστια υπερυπολογιστική ισχύς είναι θέμα αντικειμένου της παράλληλης επεξεργασίας. Στην περίπτωση που επιχειρηματικές εφαρμογές απαιτούν επικοινωνία και συνεργασία μεταξύ υπολογιστικών συστημάτων για την ανταλλαγή δεδομένων είναι θέμα αντικειμένου της κατανεμημένης επεξεργασίας.

Με αφορμή τις παραπάνω εφαρμογές να απαιτούν υπολογιστική ισχύς ή και διεκόλυνση επικοινωνίας έχουν αναπτυχθεί κατά πολύ τα πεδία της παράλληλης και κατανεμημένης από πλευράς υλικού και λογισμικού. Συγκεκριμένα, με την ανάπτυξη της τεχνολογίας υπολογιστών έχουν εμφανιστεί νέα συστήματα υπολογιστών με διπλούς ή τετραπλούς επεξεργαστές Pentium ή ακόμα συμμετρικοί πολυεπεξεργαστές και με κόστος τόσο μικρό ώστε να μπορούν να χρησιμοποιηθούν από μια όχι ιδιαίτερα μεγάλη επιχείρηση ή ακόμα κι από έναν υπολογιστή γραφείου. Επίσης, με την ραγδαία ανάπτυξη τηλεπικοινωνιών και δικτύων τοπικής εμβέλειας ή ακόμα και ευρείας περιοχής έδωσε τη δυνατότητα ανάπτυξης ενός νέου υπολογιστικού συστήματος όπου πολλοί, απλοί υπολογιστές γραφείου συνδεόμενοι μέσα από το ταχύ δίκτυο μπορούν να λειτουργούν ως μια μεγάλη, εικονική παράλληλη μηχανή. Αυτή η τεχνολογία είναι γνωστή ως υπολογιστικό πλέγμα ενώ οι ομάδες τέτοιων υπολογιστών καλούνται υπολογιστικές συστοιχίες ή ακόμα κατανεμημένα συστήματα. Τέλος, εμφανίστηκαν νέα δίκτυα όπως τα ομότιμα συστήματα υπολογιστών. Ταυτόχρονα με την εξέλιξη του υλικού έχουν εμφανιστεί νέα μον-

τέλα προγραμματισμού και εργαλεία λογισμικού για την υλοποίηση εφαρμογών στα παραπάνω υπολογιστικά συστήματα.

Συνεπώς, με τις τελευταίες εξελίξεις από το πεδίο της παράλληλης και κατανεμημένης επεξεργασίας καθώς και την έλλειψη ελληνικής βιβλιογραφίας καθιστά αναγκαίο την συγγραφή ενός βιβλίου που να πραγματεύεται σε σύγχρονα θέματα όπως τα νέα παράλληλα και κατανεμημένα συστήματα υπολογιστών καθώς και νέα εργαλεία προγραμματισμού σε αυτά.

Αυτό το βιβλίο ασχολείται με τα νέα συστήματα υπολογιστών όπως συμμετρικοί πολυπεξεργαστές, υπολογιστικές συστοιχίες, κατανεμημένα συστήματα, υπολογιστικό πλέγμα και ομότιμα συστήματα ως μια ιεραρχική οργάνωση επιπέδων υλικού και λογισμικού. Έτσι, τα νέα παράλληλα και κατανεμημένα υπολογιστικά συστήματα οργανώνονται ως μια σειρά επιπέδων παρόμοια με εκείνα ενός συμβατικού και ακολουθιακού συστήματος υπολογιστή. Συνεπώς, τα συστήματα αυτά οργανώνονται ως επίπεδα αρχιτεκτονικής, ενδιάμεσου λογισμικού συστήματος, λογισμικού εφαρμογών και εφαρμογών. Το επίπεδο αρχιτεκτονικής ασχολείται με τα συσταστικά στοιχεία ενός συστήματος και την οργάνωση των στοιχείων. Το επίπεδο ενδιάμεσου λογισμικού συστήματος παρέχει την αναγκαία υποδομή για τη σωστή και αποδοτική λειτουργία ενός συστήματος καθώς και τη υποδομή υποστήριξης στους προγραμματιστές για την ανάπτυξη εφαρμογών. Το επίπεδο λογισμικού εφαρμογών παρέχει εργαλεία ή διεπαφές προγραμματιστή εφαρμογών στους προγραμματιστές ώστε να αναπτύξουν εφαρμογή γρήγορα απαλλαγμένοι από πολύπλοκες λεπτομέρειες. Το επίπεδο εφαρμογές περιλαμβάνουν εφαρμογές επιστημονικού και επιχειρηματικού ενδιαφέροντος. Σε αυτό το βιβλίο καλύπτουμε μια συνοπτική εισαγωγή στα επίπεδα αρχιτεκτονικής και ενδιάμεσου λογισμικού συστήματος των νέων συστημάτων υπολογιστών από την σκοπιά του προγραμματιστή και όχι από την σκοπιά του διαχειριστή συστήματος. Επίσης, δίνουμε ιδιαίτερη έμφαση στην ανάπτυξη εφαρμογών Java με την βοήθεια τεχνολογιών λογισμικού εφαρμογών όχι σε βάθος αλλά παρουσιάζονται ως αντιπροσωπευτικά εργαλεία που υποστηρίζουν συγκεκριμένα μοντέλα προγραμματισμού στα παράλληλα και κατανεμημένα συστήματα.

Η δομή του βιβλίου χωρίζεται σε δύο μέρη: το πρώτο μέρος περιλαμβάνει τα κεφάλαια 2 - 6 και ασχολείται με τις αρχιτεκτονικές και οργάνωση των νέων συστημάτων υπολογιστών καθώς και το ενδιάμεσο λογισμικό συστήματος. Στο δεύτερο μέρος περιλαμβάνει τα κεφάλαια 7 - 14 και ασχολείται με τις τεχνικές προγραμματισμού σε συνδυασμό με τα νέα εργαλεία προγραμματισμού για την διεκόλυνση ανάπτυξης παράλληλων και κατανεμημένων εφαρμογών στα συστήματα αυτά.

Σε αυτό το βιβλίο χρησιμοποιεί τη γλώσσα προγραμματισμού Java για τα προγραμματιστικά

παραδείγματα. Υπάρχουν δύο λόγοι για αυτό. Ο πρώτος είναι ότι οι περισσότεροι από τους φοιτητές επιστημών πληροφορικής διδάσκονται τη γλώσσα Java είτε ως πρώτη είτε ως δεύτερη γλώσσα προγραμματισμού. Ο δεύτερος λόγος είναι ότι η Java παρέχει σημαντική υποστήριξη στο παράλληλο και κατανεμημένο προγραμματισμό είτε μέσω τυποποιημένων πακέτων είτε μέσω βοηθητικών ή ενσωματωμένων εργαλείων.

Το παρόν βιβλίο απευθύνεται στους προπτυχιακούς φοιτητές επιστημών πληροφορικής στο τελευταίο έτος των σπουδών τους και σε μεταπτυχιακούς φοιτητές στο πρώτος έτος των σπουδών τους. Για την καλύτερη κατανόηση του βιβλίου απαιτείται από τους φοιτητές να είναι εξοικοιωμένοι με το προγραμματισμό Java και να έχουν βασικές γνώσεις από τις περιοχές Αρχιτεκτονικής Υπολογιστών, Λειτουργικών Συστημάτων και Δικτύων Υπολογιστών.

Αυτό το βιβλίο συνοδεύεται από CD-ROM που περιέχει όλο το εκπαιδευτικό υλικό υποστήριξης. Συγκεκριμένα, περιέχει βασικά προγράμματα του λογισμικού συστήματος και βασικές εργαλειοθήκες Java του λογισμικού εφαρμογών για διανομή Linux, οδηγίες εγκατάστασης του λογισμικού συστήματος και εφαρμογών, οδηγίες για την ανάπτυξη και μεταγλώττιση προγραμμάτων Java για διάφορες τεχνολογίες προγραμματισμού, προγράμματα όλων των παραδειγμάτων επίδειξης σε Java και αρχεία PostScript για όλα τα σχήματα που χρησιμοποιούνται σε όλο το βιβλίο και μερικούς συνδέσμους. Τέλος, το εκπαιδευτικό υλικό υποστήριξης του CD-ROM περιέχεται επίσης στην Ιστοσελίδα: `http://www.it.uom.gr/teaching/ParallelDistributedJava/`.

Ελπίζουμε ότι το βιβλίο αυτό θα αποτελέσει ένα πολύτιμο βοήθημα τόσο για τους φοιτητές όσο και για τους διδάσκοντες της επιστήμης των υπολογιστών.

Παναγιώτης Μιχαηλίδης
Κωνσταντίνος Μαργαρίτης
Θεσσαλονίκη, Ιούνιος 2007

Κεφάλαιο 1

Εισαγωγή στα Παράλληλα και Κατανεμημένα Συστήματα

1.1 Εισαγωγή

Οι υπολογιστικές εφαρμογές συχνά χρειάζονται περισσότερη υπολογιστική ισχύ από αυτή που προσφέρει ένας ακολουθιακός υπολογιστής. Οι εφαρμογές αυτές περιλαμβάνουν αριθμητική προσομοίωση των επιστημονικών και μηχανικών προβλημάτων, προσομοίωση της παγκόσμιας οικονομίας, επεξεργασία εικόνας ή σχεδίαση νέων φαρμάκων. Τα προβλήματα αυτά συχνά απαιτούν τεράστιους επαναληπτικούς υπολογισμούς πάνω σε πολλά δεδομένα ώστε να δώσουν έγκυρα αποτελέσματα. Οι υπολογισμοί αυτοί πρέπει να ολοκληρωθούν μέσα σε μια λογική χρονική περίοδο. Ένας τρόπος για να ξεπεράσουμε αυτό τον περιορισμό είναι να βελτιώσουμε την ταχύτητα των επεξεργαστών έτσι ώστε να προσφέρουν την υπολογιστική ισχύ που χρειάζονται οι υπολογιστικά απαιτητικές εφαρμογές. Για την επίτευξη της αύξησης της ταχύτητας των επεξεργαστών έχουν ενσωματωθεί βασικές αρχιτεκτονικές σχεδιασμοί όπως η κρυφή μνήμη, η διαφυλλωμένη μνήμη, η bit-παράλληλη μνήμη, η bit-παράλληλη αριθμητική, η σωλήνωση εντολών, οι πολλαπλές λειτουργικές μονάδες και η σωλήνωση δεδομένων. Παρόλο που αυτά είναι λογικά μέχρι ένα σημείο, οι μελλοντικές βελτιώσεις περιορίζονται από ορισμένους παράγοντες όπως: η ταχύτητα του φωτός, οι νόμοι της θερμοδυναμικής και το υψηλό κόστος κατασκευής επεξεργαστών. Μια βιώσιμη και αποτελεσματική λύση είναι να συνδέσουμε πολλούς επεξεργαστές μαζί και να συναρμονίσουμε τις υπολογιστικές τους προσπάθειες. Αυτό έχει σαν

αποτέλεσμα την δημιουργία συστημάτων που είναι γνωστά ως **παράλληλοι υπολογιστές** (parallel computers) και επιτρέπουν τον διαμοιρασμό μιας υπολογιστικής εργασίας μεταξύ των πολλαπλών επεξεργαστών. Ο Pfister [6] δείχνει ότι υπάρχουν τρεις τρόποι για να βελτιώσουμε την απόδοση:

- η σκληρή εργασία,
- η έξυπνη εργασία, και
- η λήψη βοήθειας.

Σε όρους τεχνολογίας υπολογιστών, η σκληρή εργασία αντιστοιχεί στο να χρησιμοποιείται ταχύτερο υλικό (όπως επεξεργαστές, μνήμη, ή περιφερειακές συσκευές υψηλής απόδοσης). Η έξυπνη εργασία αντιστοιχεί στο να εκτελούνται οι διάφορες εργασίες πιο αποτελεσματικά και σχετίζεται με τους αλγόριθμους και τεχνικές που χρησιμοποιούνται για να εκτελέσουν τις υπολογιστικές εργασίες. Τέλος, η λήψη βοήθειας αναφέρεται στο να συνεργάζονται οι υπολογιστές μεταξύ τους ώστε να λύσουν ένα υπολογιστικό πρόβλημα. Η συνεργασία μπορεί να είναι περισσότερη ή λιγότερη στενή, υποβοηθούμενη από ειδικό υλικό ή μόνο με τη χρήση κάποιου λογισμικού.

Συνεπώς, ένα **παράλληλο σύστημα** (parallel system) ή παράλληλος υπολογιστής είναι ένα ειδικό σχεδιασμένο σύστημα που περιέχει πολλαπλούς εσωτερικούς επεξεργαστές ή περιέχει πολλαπλούς υπολογιστές που είναι στενά διασυνδεδεμένα μεταξύ τους για την λύση ενός μεγάλου προβλήματος. Η προσέγγιση αυτή αυξάνει την απόδοση από το ακολουθιακό υπολογιστή και η ιδέα είναι ότι η ταυτόχρονη λειτουργία των n υπολογιστών έχει σαν αποτέλεσμα να είναι n φορές ταχύτερος. Αυτό σπάνια συμβαίνει στην πράξη για διάφορους λόγους.

Σήμερα με την ευρεία ανάπτυξη του **Διαδικτύου** (Internet) και του **Παγκόσμιου Ιστού** (World Wide Web), η διαθεσιμότητα των συστημάτων υπολογιστών είναι κρίσιμη (ή ζωτικής σημασίας) για ορισμένες εφαρμογές όπως το **ηλεκτρονικό εμπόριο** (e-commerce). Έτσι, για παράδειγμα οι **διακομιστές Ιστού** (Web servers) που χρησιμοποιούνται για να φιλοξενήσουν πολλές υπηρεσίες Διαδικτύου όχι μόνο πρέπει να είναι σε θέση να αντιμετωπίζουν τις εκατοντάδες αιτήσεις χρηστών αποτελεσματικά αλλά και πρέπει να λειτουργούν αδιάλειπτα σε 24ωρη βάση. Για αυτό το λόγο εμφανίστηκε μια νέα κατηγορία υπολογιστών που είναι γνωστοί ως **συστοιχίες υπολογιστών** (cluster of computers) οι οποίες αποτελούνται από ανεξάρτη-

τους υπολογιστές που είναι συνδεδεμένοι μεταξύ τους μέσω τοπικού δικτύου. Οι συστοιχίες υπολογιστών μπορούν να χρησιμοποιηθούν για να φιλοξενήσουν διάφορες υπηρεσίες Ιστού και οι διάφορες αιτήσεις χρηστών να εκτελούνται σε διαφορετικούς υπολογιστές της συστοιχίας. Επίσης, οι συστοιχίες υπολογιστών χρησιμοποιούνται και στην περίπτωση που ένας διακομιστής Ιστού μιας εφαρμογής ηλεκτρονικού εμπορίου παρουσιάζει πρόβλημα λειτουργίας να αντικατασταθεί από έναν άλλο υπολογιστή της συστοιχίας. Συνοπτικά λοιπόν οι συστοιχίες υπολογιστών μπορούν να χρησιμοποιηθούν σε διάφορες εφαρμογές Διαδικτύου που απαιτούν υψηλή διαθεσιμότητα (όπως διακομιστές Ιστού, μηχανές αναζήτησης, ηλεκτρονική τράπεζα κλπ) και επίσης μπορούν να χρησιμοποιηθούν σε εφαρμογές υψηλών υπολογιστικών απαιτήσεων όπως συζητήσαμε προηγουμένως.

Από την άλλη μεριά υπάρχουν εφαρμογές που δεν απαιτούν μόνο επιτάχυνση των υπολογισμών και απρόσκοπτη λειτουργία των συστημάτων υπολογιστών (ή διακομιστών) αλλά πρόσβαση σε απομακρυσμένα συστήματα υπολογιστών με σκοπό την ανταλλαγή δεδομένων και υπηρεσιών. Τέτοιες εφαρμογές είναι η πρόσβαση ιστοσελίδων σε διακομιστές Ιστού που υπάρχουν στο Διαδίκτυο, η παροχή υπηρεσιών ηλεκτρονικού εμπορίου στους πελάτες για διάφορα προϊόντα ή πρόσβαση σε συστήματα βάσεων δεδομένων για τραπεζικές συναλλαγές και κρατήσεις αεροπορικών εισιτηρίων. Οι εφαρμογές αυτές δεν απαιτούν απαιτητικούς υπολογισμούς αλλά επικεντρώνονται στην διεκόλυνση της επικοινωνίας μεταξύ απομακρυσμένων υπολογιστών για την διαφανή ανταλλαγή και ακεραιότητα των δεδομένων. Ένας τρόπος για να επιταχύνουμε την επικοινωνία μεταξύ των απομακρυσμένων υπολογιστών είναι η αύξηση της ταχύτητας του δικτύου επικοινωνίας που συνδέονται οι υπολογιστές. Γι αυτό μια καλή λύση είναι να συνδέσουμε τους απομακρυσμένους και ανεξάρτητους υπολογιστές με ένα υπερ-ταχύ δίκτυο επικοινωνίας. Αυτό έχει σαν αποτέλεσμα την δημιουργία συστημάτων που είναι γνωστά ως **κατανεμημένα συστήματα** (distributed systems). Συνεπώς, ένα κατανεμημένο σύστημα είναι είναι ένα σύστημα που αποτελείται από ανεξάρτητους πόρους (όπως οι προσωπικοί υπολογιστές, οι παράλληλοι υπολογιστές και οι συστοιχίες υπολογιστών) που είναι χαλαρά διασυνδεδεμένα μεταξύ τους μέσω Διαδικτύου (ή δικτύου επικοινωνίας υψηλής ταχύτητας) για την ταχεία ανταλλαγή υπηρεσιών στις διάφορες εφαρμογές. Σε περίπτωση που το παραπάνω σύστημα χρησιμοποιηθεί για εφαρμογές που απαιτούν μεταφορά δεδομένων μεταξύ των πόρων και συγχρόνως χρησιμοποιοιεί τους πόρους αυτούς για υπολογισμούς τότε το σύστημα ονομάζεται **πλέγμα υπολογιστών** (grid computing).

Τα τελευταία χρόνια με την ραγδαία εξάπλωση του Διαδικτύου μια ακόμη ανάγκη αναφύεται: Η ανάγκη για κοινή χρήση υπολογιστικών πόρων (αποθήκευση, υπολογιστική ισχύς, κλπ). Έτσι με αυτό το τρόπο συγκροτούνται δίκτυα από πολλούς υπολογιστές με σκοπό την ανταλλαγή αρχείων (συνήθως μουσικά κομμάτια MP3) ή για ανταλλαγή εγγράφων ή για κατανεμημένο υπολογισμό ή για παροχή κατανεμημένων υπηρεσιών. Τέτοια δίκτυα είναι γνωστά ως **ομότιμα συστήματα** (peer-to-peer systems).

Από την παραπάνω εξέταση διαφορετικών εφαρμογών προκύπτουν πέντε κατηγορίες συστημάτων που είναι οι παράλληλοι υπολογιστές, οι συστοιχίες υπολογιστών, τα κατανεμημένα συστήματα, το πλέγμα υπολογιστών και τα ομότιμα συστήματα. Αυτές τις κατηγορίες συστημάτων θα εξετάσουμε διεξοδικά στα επόμενα κεφάλαια από την πλευρά του υλικού (hardware). Πριν εξετάσουμε αναλυτικά τις αρχιτεκτονικές των συστημάτων, θα παρουσιάζουμε πρώτα μερικές αντιπροσωπευτικές εφαρμογές για καθένα ένα από τα παραπάνω συστήματα στην επόμενη ενότητα.

1.2 Εφαρμογές

Σε αυτή την ενότητα παρουσιάσουμε πέντε αντιπροσωπευτικές εφαρμογές που μπορούν να χρησιμοποιηθούν στα υπολογιστικά συστήματα που αναφέραμε στην προηγούμενη ενότητα.

1.2.1 Προσομοίωση Ν Σωμάτων

Ένα πρόβλημα που απαιτεί τεράστιους υπολογισμούς είναι η προσομοίωση της κίνησης των αστρονομικών σωμάτων στο διάστημα. Αυτό το πρόβλημα των Ν σωμάτων χρησιμοποιείται στην αστροφυσική για τον υπολογισμό των δυναμικών του ηλιακού συστήματος και των γαλαξιών. Κάθε σώμα ελκύεται από κάθε άλλο σώμα με τις δυνάμεις της βαρύτητας. Οι δυνάμεις της βαρύτητας μπορούν να υπολογιστούν από έναν απλό τύπο και η κίνηση του κάθε σώματος μπορεί να προσομοιωθεί υπολογίζοντας την συνολική δύναμη από το σώμα. Έτσι αν υπάρχουν N σώματα, τότε πρέπει να υπολογιστούν οι δυνάμεις $N-1$ για κάθε σώμα ή απαιτούν συνολικά N^2 υπολογισμοί. Μετά τον προσδιορισμό νέων θέσεων των σωμάτων, οι υπολογισμοί πρέπει να επαναληφθούν. Ας θεωρήσουμε ένα μεγάλο αριθμό σωμάτων. Ένας γαλαξίας ίσως να έχει 10^{11} άστρα. Αυτό απαιτεί 10^{22} υπολογισμούς που πρέπει να επαναληφθούν. Ακόμη και

αν χρησιμοποιήσουμε έναν αποτελεσματικό προσεγγιστικό αλγόριθμο ο οποίος απαιτεί $Nlog_2N$ υπολογισμούς, πάλι ο αριθμός των υπολογισμών παραμένει μεγάλος ($10^{11}log_210^{11}$). Έτσι λοιπόν απαιτεί σημαντικό χρόνο σε ένα σύστημα ενός συμβατικού υπολογιστή. Ακόμα και αν ο κάθε υπολογισμός θα μπορούσε να γίνει σε 1 μs (10^{-6} δευτερόλεπτα) τότε απαιτεί 10^9 χρόνια για μια επανάληψη χρησιμοποιώντας τον αλγόριθμο N^2 και περίπου ένα χρόνο για μια επανάληψη χρησιμοποιώντας τον αλγόριθμο $Nlog_2N$. Συνεπώς, το παραπάνω πρόβλημα απαιτεί υψηλή υπολογιστική ισχύς.

Το πρόβλημα των N σωμάτων μπορεί επίσης να εμφανιστεί σε προσομοιώσεις μορίων όπως είναι οι υπολογισμοί μοριακής δυναμικής που γίνονται στη φυσική και στη χημεία, όπου κάθε μόριο ενεργεί σύμφωνα με μια δύναμη υψηλής κλίμακας από κάθε άλλο μόριο. Με την άφιξη των υπερυπολογιστών υψηλής απόδοσης αυτά τα είδη των προσομοιώσεων μοριακών ή βιολογικών συστημάτων έγιναν ιδιαίτερα δημοφιλή. Αυτές οι προσομοιώσεις είναι τόσο υπολογιστικά απαιτητικές που προηγούμενες γενιές υπολογιστών μπορούσαν πρακτικά να επιλύσουν συστήματα με μόνο πολύ μικρούς αριθμούς μορίων. Οι επιστήμονες έπρεπε να καταφύγουν σε άλλες πιο απλές μαθηματικές προσεγγιστικές τεχνικές για να κατανοήσουν τη συμπεριφορά μεγάλων συστημάτων μορίων. Τώρα όμως με τη αυξανόμενη διαθεσιμότητα των παράλληλων υπολογιστών είναι εφικτό να αποκτήσει κανείς αντίληψη σχετικά με τη συμπεριφορά αυτών των βιολογικών συστημάτων μέσα από την απευθείας προσομοίωση της αλληλεπίδρασης των μορίων.

1.2.2 Μηχανή Αναζήτησης Google

Οι μηχανές αναζήτησης Διαδικτύου δίνουν την δυνατότητα στους χρήστες να αναζητήσουν πληροφορίες στο Διαδίκτυο εισάγοντας συγκεκριμένες λέξεις - κλειδιά. Μια δημοφιλή μηχανή αναζήτησης όπως η Google χρησιμοποιεί συστοιχία υπολογιστών για να ικανοποιήσει την τεράστια ποσότητα των αιτήσεων αναζήτησης παγκοσμίως που περιλαμβάνει από μια αιχμή χιλιάδων ερωτήσεων ανά δευτερόλεπτο. Μια ερώτηση Google χρειάζεται να χρησιμοποιήσει τουλάχιστον 10 εκατομμύριους κύκλους επεξεργασίας και να προσπελάσει λίγες εκαντοτάδες MBytes δεδομένων προκειμένου να επιστρέψει ικανοποιητικά αποτελέσματα αναζήτησης.

Το Google χρησιμοποιεί συστοιχία υπολογιστών σαν λύση στην υψηλή ζήτηση των πόρων συστήματος εφόσον η συστοχία υπολογιστών έχει καλύτερες αναλογίες κόστους - απόδοσης από τις εναλλακτικές παραδοσιακές υπολογιστικές πλατφόρμες υψηλής απόδοσης και επίσης

χρησιμοποιεί λιγότερη ηλεκτρική ενέργεια. Το Google εστιάζει σε δύο σημαντικούς παράγοντες σχεδιασμού: αξιοπιστία και ρυθμοαπόδοση αίτησης.

Το Google είναι σε θέση να επιτύχει την αξιοπιστία σε επίπεδο λογισμικού έτσι ώστε μια αξιόπιστη υπολογιστική υποδομή μπορεί να κατασκευαστεί σε συστοιχίες 15.000 εμπορικών PCs κατανεμημένων παγκοσμίως. Οι υπηρεσίες για Google είναι επίσης πανομοιότυπες στους διάφορους υπολογιστές της συστοιχίας ώστε να παρέχει την απαραίτητη διαθεσιμότητα. Το Google μεγιστοποιεί την συνολική ρυθμοαπόδοση αίτησης από την παράλληλη εκτέλεση των μεμονωμένων αιτήσεων αναζήτησης. Αυτό σημαίνει ότι περισσότερα αιτήματα αναζήτησης μπορούν να ολοκληρωθούν μέσα σε ένα συγκεκριμένο χρονικό διάστημα.

Μια τυπική αναζήτηση Google αποτελείται από τις παρακάτω διαδικασίες:

1. Ένας χρήστης Διαδικτύου εισάγει μια ερώτηση στην ιστοσελίδα Google.

2. Ο περιηγητής Ιστού ψάχνει την διεύθυνση πρωτοκόλλου Διαδικτύου (Internet Protocol - IP) μέσω του Διακομιστή Ονόματος Περιοχής (Domain Name Server - DNS) του www.google.com.

3. Το Google χρησιμοποιεί ένα DNS βασισμένος σε ένα σύστημα εξισορόπησης φορτίου που απεικονίζει την ερώτηση σε μια συστοιχία που είναι γεωγραφικά πλησιέστερη στο χρήστη ώστε να ελαχιστοποιηθεί ο χρόνος καθυστέρησης επικοινωνίας δικτύου. Η διεύθυνση IP της επιλεγμένης συστοιχίας επιστρέφεται.

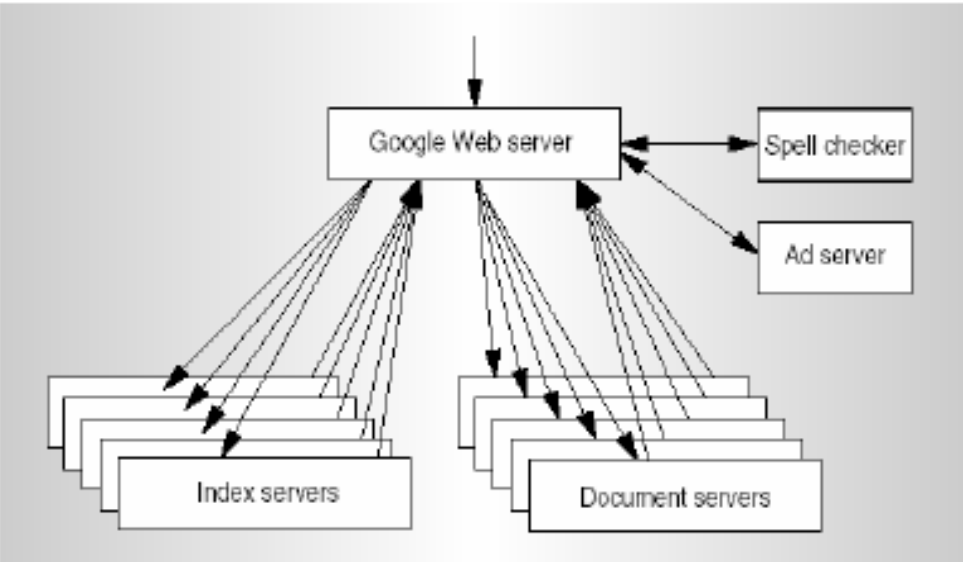
4. Έπειτα ο περιηγητής Ιστού στέλνει την αίτηση αναζήτησης μέσω του πρωτοκόλλου HTTP (HyperText Transport Protocol) στην επιλεγμένη συστοιχία με την συγκεκριμένη διεύθυνση IP.

5. Η επιλεγμένη συστοιχία επεξεργάζεται την ερώτηση τοπικά.

6. Ένας εξισορροπητής φορτίου βασισμένος στο υλικό της συστοιχίας παρακολουθεί τους διαθέσιμους διακομιστές Ιστού Google (Google Web Servers - GWSs) της συστοιχίας και διανέμει τις αιτήσεις ομοιόμορφα μέσα στη συστοιχία.

7. Μια μηχανή GWS λαμβάνει την αίτηση, συντονίζει την εκτέλεση της ερώτησης και στέλνει το αποτέλεσμα της αναζήτησης πίσω στο περιηγητή Ιστού του χρήστη.

Στο Σχήμα 1.1 δείχνει πώς ένας GWS λειτουργεί μέσα σε μια τοπική συστοιχία υπολογιστών. Η πρώτη φάση εκτέλεσης ερώτησης περιλαμβάνει διακομιστές ευρετηρίων που συμβουλεύονται έναν ανεστραμένο ευρετήριο ώστε να ταιριάζει κάθε λέξη κλειδί της ερώτησης με έναν κατάλογο ταυτισμένων εγγράφων. Επίσης, υπολογίζονται πρώτα τα αποτελέσματα σχετικότητας για το ταίριασμα των εγγράφων έτσι ώστε το αποτέλεσμα αναζήτησης που επιστρέφεται στο χρήστη να είναι ταξινομημένος κατά σχετικότητα. Στη δεύτερη φάση, οι διακομιστές εγγράφων προσκομίζουν κάθε έγγραφο από το δίσκο για να εξάγουν τον τίτλο και λέξεις κλειδιά συναφές με το περιεχόμενο του εγγράφου. Εκτός από τις δύο παραπάνω φάσεις, το GWS ενεργοποιεί επίσης τον ορθογραφικό ελεγκτή και διακομιστής διαφημίσεων. Ο ορθογραφικός ελεγκτής ελέγχει την ορθογραφία των λέξεων κλειδιών της ερώτησης, ενώ ο διακομιστής διαφημίσεων παράγει διαφημίσεις που είναι σχετικές με την ερώτηση και επομένως μπορούν να ενδιαφέρουν στους χρήστες. Συνεπώς, η μηχανή αναζήτησης Διαδικτύου Google χρησιμοποιεί συστοιχία υπολογιστών ώστε να παρέχει αξιόπιστες και αποτελεσματικές υπηρεσίες αναζήτησης Διαδικτύου.



Σχήμα 1.1: Αρχιτεκτονική Google

1.2.3 Αυτόματες Συναλλαγές μέσω Τράπεζας

Ένα παράδειγμα ενός κατανεμημένου συστήματος είναι το δίκτυο αυτόματων ταμειακών συναλλαγών τραπεζικού συστήματος που διαχειρίζεται τους λογαριασμούς των πελατών μιας τράπεζας. Το δίκτυο αυτό περιλαμβάνει μεγάλους μινι-υπολογιστές ή μεγάλες κεντρικές μονάδες υπολογιστών που είναι συνήθως τοποθετημένοι σ΄ ένα κεντρικό γραφείο, όπου αποθηκεύονται οι λεπτομέρειες των λογαριασμών όλων των πελατών. Επίσης, στο ίδιο δίκτυο περιλαμβάνει τοπικούς υπολογιστές που είναι κατανεμημένοι σε όλη την επικράτεια της Ελλάδας, όπου αποθηκεύουν πληροφορίες για τους τοπικούς πελάτες κάθε υποκαταστήματος και μικρότερους υπολογιστές οι οποίοι ελέγχουν τις μηχανές αυτόματων ταμειακών συναλλαγών γνωστές ως ATM και τις τοπικές ταμειακές μηχανές κάθε υποκαταστήματος. Όλοι αυτοί οι υπολογιστές συνδέονται μεταξύ τους με κάποιο σχετικά υπερ-ταχύς δίκτυο επικοινωνίας και καθένας από αυτούς τους υπολογιστές είναι προγραμματισμένος να εκτελεί ορισμένες λειτουργίες μέσα στο σύστημα. Για παράδειγμα, οι μεγάλες μονάδες επεξεργασίας είναι υπεύθυνες για την αποθήκευση και αρχειοθέτηση των πελατειακών καταλόγων και οι υπολογιστές που σχετίζονται με τις μηχανές ATM είναι υπεύθυνες για λειτουργίες όπως η χρέωση ενός λογαριασμού μετά από μια ανάληψη ή η ενημέρωση μιας μονάδας επεξεργασίας όταν ένας πελάτης ζητά ένα αντίγραφο κινήσεως λογαριασμού.

Όταν ένας πελάτης θέλει να κάνει ανάληψη ή κατάθεση χρημάτων σε οποιαδήποτε ταμειακή μηχανή της Ελλάδας δίνει σαν είσοδο στον τοπικό υπολογιστή τον προσωπικό του κωδικό αριθμό και το ποσό που θέλει να αναλάβει ή να καταθέσει. Σε αυτή την περίπτωση είναι πιθανό κάθε υπολογιστής να συμβουλεύεται για λόγους ταχύτητας πρώτα σε μια τοπική βάση δεδομένων και αν δεν βρεθεί ο πελάτης εκεί τότε γίνεται αναζήτηση σε κάποια κεντρική βάση ή σε κάποιο άλλο σταθμό εργασίας.

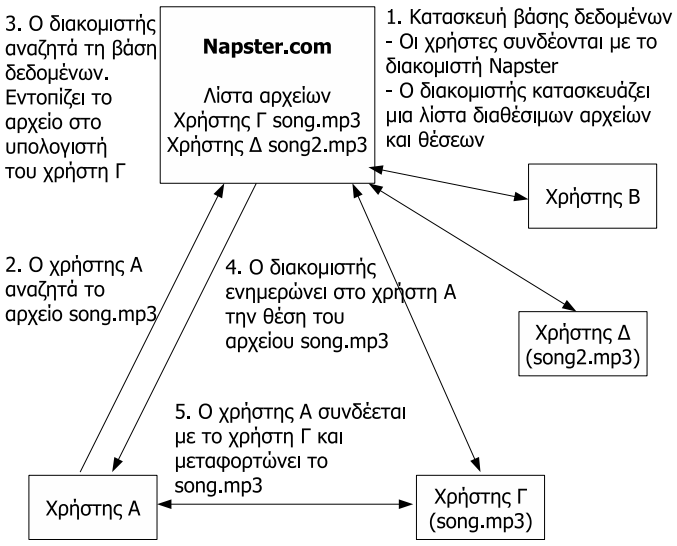
Στόχος του συστήματος αυτόματων ταμειακών συναλλαγών δεν είναι να μεγιστοποιηθεί η ταχύτητα συναλλαγής (αν και αυτό είναι επίσης επιθυμητό) αλλά να υπάρχει ασφάλεια και ακεραιότητα των δεδομένων τους. Για παράδειγμα αν ζητηθούν ταυτόχρονα από τον ίδιο λογαριασμό να αφαιρεθούν χρήματα από δύο διαφορετικούς τοπικούς υπολογιστές (γιατί π.χ. ο λογαριασμός είναι κοινός) πρέπει οι δύο συναλλαγές να μπουν σε μια σειρά ώστε να μη γίνει λάθος στην αφαίρεση.

1.2.4 Τηλειατρική

1.2.5 Ανταλλαγή Μουσικών Αρχείων και Κατανεμημένος Υπολογισμός

Σε αυτή την ενότητα θα παρουσιάζουμε δύο δημοφιλή παραδείγματα που χρησιμοποιούνται στα ομότιμα συστήματα υπολογιστών. Ένα παράδειγμα που σχετίζεται με την ανταλλαγή μουσικών αρχείων με το Napster και το δεύτερο παράδειγμα που σχετίζεται με το κατανεμημένο υπολογισμό με το SETI@home.

Το Napster το δημοφιλές πρόγραμμα ανταλαγής αρχείων MP3 ξεκίνησε μέσα στο 1999 και είχε εξελικτική επίδραση στο Διαδίκτυο εξαιτίας τους γνωστούς νομικούς και οικονομικούς περιορισμούς. Η λειτουργία του Napster είναι ως εξής: μετά την κεντρική αναζήτηση στο κατάλογο Napster, οι υπολογιστές συνδέονται μεταξύ τους και ανταλλάσουν απευθείας δεδομένα από το ένα δίσκο του υπολογιστή στον άλλο δίσκο. Αυτή η διαδικασία απεικονίζεται καλύτερα στο Σχήμα 6.2.



Σχήμα 1.2: Το Napster για ανταλλαγή μουσικών αρχείων

Πρώτα, οι χρήστες των υπολογιστών συνδέονται με το κεντρικό διακομιστή Napster και εγγράφονται στο δίκτυο. Ο κεντρικός διακομιστής αποκτά μια λίστα αρχείων MP3 η οποία προκύπτει από τα αρχεία που έχει ο χρήστης και τα προσθέτει με τα υπόλοιπα αρχεία σε μια κεντρική βάση δεδομένων. Όταν ένας χρήστης (π.χ. χρήστης Α) εκτελεί μια αναζήτηση στο

κεντρικό διακομιστή, ο Napster αναζητά την τοπική της βάση δεδομένων και επιστρέφει την διεύθυνση του ομότιμου υπολογιστή που έχει ένα αντίγραφο του αρχείου. Ο χρήστης Α τότε συνδέεται με το υπολογιστή που έχει το αρχείο (δηλαδή με το χρήστη Γ) και τότε μεταφορτώνει απευθείας το αρχείο από το δίσκο του χρήστη Γ χωρίς καμία πρόσθετη παρέμβαση.

Το Napster είναι ομότιμο σύστημα επειδή οι κόμβοι ή υπολογιστές του Napster παρακάμπτουν το σύστημα DNS και μόλις ο κεντρικός διακομιστής Napster επιλύσει τη διεύθυνση IP των υπολογιστών που φιλοξενούν ένα συγκεκριμένο μουσικό αρχείο τότε μεταφέρεται ο έλεγχος της μεταφόρτωσης αρχείων στους υπολογιστές.

Ένα από τα πιο δημοφιλή φαινόμενα που κατέστησε δυνατό ο Παγκόσμιος Ιστός είναι του κατανεμημένου υπολογισμού ή όπως είναι κοινώς γνωστό η μεταυπολογιστική εφαρμογή. Εδώ, οι χρήστες του Ιστού χρησιμοποιούν τις περιόδους αργίας των υπολογιστών τους για να φέρουν εις πέρας κάποιες βαριές υπολογιστικές εργασίες ως μέρος ενός μεγαλύτερου προγράμματος. Το πιο γνωστό πρόγραμμα κατανεμημένου υπολογισμού είναι το SETI@home που ξεκίνησε το 1996 και το οποίο είναι ένα επιστημονικό πείραμα που χρησιμοποιεί τις περιόδους αργίας των συνδεδεμένων υπολογιστών του Διαδικτύου για Αναζήτηση Εξωγήινης Νοημοσύνης (Search for ExtraTerrestrial Intelligence). Οι χρήστες Ιστού εγγράφονται στον ιστοχώρο του SETI και μεταφορτώνουν μια εφαρμογή screen saver η οποία χρησιμοποιεί διάφορους αλγορίθμους ανάλυσης σημάτων για να επεξεργαστεί ραδιο-τηλεοσκοπικά δεδομένα με σκοπό την αναζήτηση κανονικοτήτων, που ίσως υποδεικνύουν την ύπαρξη ζωής έξω από το ηλιακό μας σύστημα. Η εφαρμογή αυτή εκτελείται σε περιόδους αργίας, (π.χ όταν ένας χρήστης κάθεται σε έναν υπολογιστή και διαβάζει μια ιστοσελίδα) και συνδέεται με το διακομιστή SETI για να μεταφορτώσει τα δεδομένα για επεξεργασία και τότε επεξεργάζεται τα δεδομένα μέχρι να λύσει το πρόβλημα. Μόλις ολοκληρωθεί η επεξεργασία επιστρέφει τα αποτελέσματα πίσω στο διακομιστή για περαιτέρω ανάλυση. Στην περίπτωση που η επεξεργασία δεν ολοκληρωθεί επιτυχώς τότε τα δεδομένα που έλαβε η εφαρμογή screen saver χάνονται και αγνοούνται. Το SETI δεν αποτελεί το μόνο πρόγραμμα κατανεμημένου υπολογισμού και υπάρχουν άλλα παρόμοια προγράμματα. Το πρόγραμμα Distributed.net χρησιμοποιεί την υπολογιστική ισχύς και προσπαθεί να επιλύσει μερικά ιδιαίτερα δύσκολα προβλήματα όπως να σπάσει κρυπτογραφικούς κώδικες με τη χρήση απλών τεχνικών, δηλαδή δοκιμάζοντας όλα τα δυνατά κλειδιά ή και η επίλυση κάποιων αρκετά δύσκολων μαθηματικών προβλημάτων όπως η εύρεση πρώτων αριθμών και ο υπολογισμός του π. Τέλος, υπάρχουν εταιρείες που αξιοποιούν τον ελεύθερο χρόνο των συνδεδεμένων υπολογιστών

στο Διαδίκτυο για να επιλύσουν προβλήματα για εμπορικούς και φιλανθρωπικούς σκοπούς. Μια απο αυτές είναι η Entropia που διαθέτει τον ελεύθερο χρόνο των χρηστών υπολογιστών που εκτελούν το δικό της πρόγραμμα σε εμπορικούς σκοπούς και κατά το μεγαλύτερο μέρος σε φιλανθρωπικούς σκοπούς, όπως η έρευνα κατά του AIDS.

Κεφάλαιο 2

Συμμετρικοί Πολυεπεξεργαστές

Αυτό το κεφάλαιο ξεκινά περιγράφοντας τα πλεονεκτήματα του συμμετρικού πολυεπεξεργαστή. Έπειτα παρουσιάζουμε μια σύντομη περιγραφή της αρχιτεκτονικής συμμετρικού πολυεπεξεργαστή. Στην συνέχεια, περιγράφουμε τα συστατικά υλικού και λογισμικού που τον αποτελούν όπως επεξεργαστές, κρυφής μνήμης, πρωτόκολλα συνοχής κρυφής μνήμης και λειτουργικό σύστημα. Τέλος, παρουσιάζουμε το λογισμικό εφαρμογών και τα αντίστοιχα εργαλεία τα οποία διεκολύνουν στους προγραμματιστές την ανάπτυξη εφαρμογών για συμμετρικό πολυεπεξεργαστή.

2.1 Ωφέλη Συμμετρικών Πολυεπεξεργαστών

Μέχρι πρόσφατα, σχεδόν όλοι οι προσωπικοί υπολογιστές ενός χρήστη και οι περισσότεροι σταθμοί εργασίας περιείχαν ένα μικροεπεξεργαστή γενικού σκοπού. Καθώς αυξάνουν οι απαιτήσεις για απόδοση και καθώς εξακολουθεί να πέφτει το κόστος των μικροεπεξεργαστών, έχουν εμφανιστεί **συστήματα πολυεπεξεργαστές** (multiprocessors). Πολυεπεξεργαστής είναι ένα υπολογιστικό σύστημα στο οποίο δύο ή περισσότεροι επεξεργαστές έχουν πρόσβαση σε μια κοινή μνήμη (ή ένα κοινό χώρο διευθύνσεων) η οποία είναι ορατή από όλους τους επεξεργαστές. Το λειτουργικό σύστημα των πολυεπεξεργαστών είναι παρόμοιο με ένα κλασικό λειτουργικό σύστημα ενός επεξεργαστή. Έτσι, στα περισσότερα συστήματα πολυεπεξεργαστών υπάρχει ένα αντίγραφο του λειτουργικού συστήματος στην κοινή μνήμη, αλλά μπορεί να το προσπελάσει οποιοδήποτε επεξεργαστή. Όμως, επειδή οι δομές δεδομένων του λειτουργικού συστήματος

προσπελάζονται από διαφορετικό επεξεργαστή πρέπει να εξασφαλιστεί η ταύτοχρονη πρόσβαση ώστε να υπάρχει συνέπεια. Αυτό μπορεί να εξασφαλιστεί με την χρήση ειδικών μηχανισμών του υλισμικού που συγχρονίζουν την πρόσβαση του λειτουργικού συστήματος στην κοινή μνήμη.

Ένας πολυεπεξεργαστής, όπως όλοι οι υπολογιστές πρέπει να έχει συσκευές Εισόδου/Εξόδου (Ε/Ε), όπως δίσκους, κάρτες δικτύου και άλλες. Σε μερικά πολυεπεξεργαστικά συστήματα μόνο ορισμένοι επεξεργαστές έχουν πρόσβαση στις συσκευές Ε/Ε και επόμενως έχουν ειδική λειτουργία Ε/Ε. Σε άλλα συστήματα, όλοι οι επεξεργαστές έχουν ισότιμη πρόσβαση σε όλες τις συσκευές Ε/Ε. Όταν όλοι οι επεξεργαστές έχουν ισότιμη πρόσβαση στη μνήμη και σε όλες τις συσκευές Ε/Ε και αντιμετωπίζονται ως εναλλάξιμες μεταξύ τους από το λειτουργικό σύστημα, το σύστημα ονομάζεται **συμμετρικός πολυεπεξεργαστής** (Symmetric Multiprocessor - SMP). Ο όρος SMP αναφέρεται σε αρχιτεκτονική υπολογιστών καθώς και στην συμπεριφορά του λειτουργικού συστήματος που περιέχει αυτή την αρχιτεκτονική.

Μια οργάνωση SMP έχει πολλά πλεονεκτήματα σε σχέση με την οργάνωση ενός επεξεργαστή, που είναι τα εξής:

- Υψηλή απόδοση. Αν ο υπολογισμός μιας εφαρμογής που πρέπει να γίνει από υπολογιστή μπορεί να οργανωθεί έτσι ώστε κάποια κομμάτια του υπολογισμού να μπορούν να υλοποιηθούν παράλληλα, τότε ένα σύστημα με πολλούς επεξεργαστές θα εμφανίζει καλύτερη απόδοση από ένα σύστημα ενός επεξεργαστή του ίδιου τύπου.

- Υψηλή διαθεσιμότητα. Σε ένα συμμετρικό πολυεπεξεργαστή επειδή όλοι οι επεξεργαστές μπορούν να εκτελέσουν τις ίδιες λειτουργίες, η βλάβη ενός επεξεργαστή δεν αχρηστεύει το σύστημα. Αντίθετα, το σύστημα μπορεί να συνεχίσει να λειτουργεί με μειωμένη απόδοση. Αυτό το πλεονέκτημα είναι ιδιαίτερα χρήσιμο για εφαρμογές διακομιστές Ιστού που φιλοξενούν διάφορες υπηρεσίες Διαδικτύου.

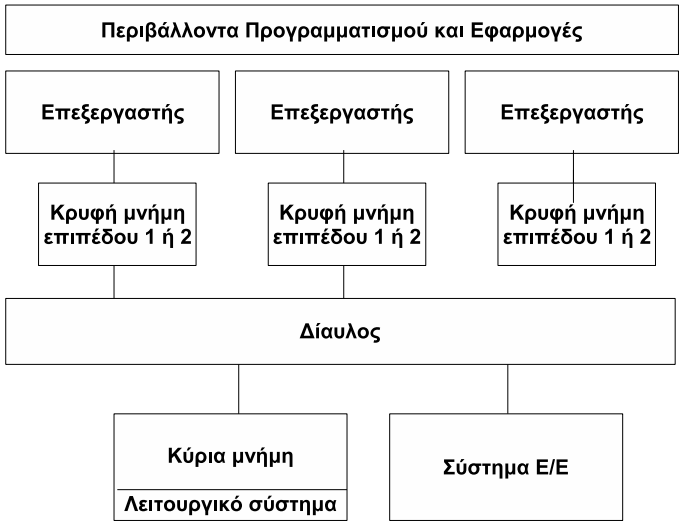
- Αυξητική κλιμάκωση. Ο χρήστης μπορεί να ενισχύσει την απόδοση συστήματος με εύκολη προσθήκη επιπλέον επεξεργαστών.

- Απόλυτη κλιμάκωση. Οι σχεδιαστές του εμπορίου μπορούν να προσφέρουν μια περιοχή προϊόντων με διαφορετικές τιμές και διαφορετικά χαρακτηριστικά απόδοσης με βάση το πλήθος των επεξεργαστών που θα υπάρχουν στο σύστημα.

Πρέπει να σημειωθεί ότι τα παραπάνω ωφέλη είναι δυναμικά και όχι εξασφαλισμένα.

2.2 Αρχιτεκτονική ενός Συμμετρικού Πολυεπεξεργαστή

Ένας συμμετρικός πολυεπεξεργαστής SMP είναι ένα υπολογιστικό σύστημα το οποίο αποτελείται από δύο ή περισσότερους αυτόνομους και όμοιους επεξεργαστές και έχουν την εικόνα ενός συστήματος στους χρήστες και στις εφαρμογές. Σαφώς, ο όρος SMP αναφέρεται σε αρχιτεκτονική υπολογιστών καθώς και στην συμπεριφορά του λειτουργικού συστήματος που περιέχει αυτή την αρχιτεκτονική. Στο Σχήμα 2.1 παρουσιάζεται η αρχιτεκτονική ενός τέτοιου SMP.



Σχήμα 2.1: Αρχιτεκτονική Συμμετρικού Πολυεπεξεργαστή

Τα βασικά συσταστικά ενός συμμετρικού πολυεπεξεργαστή είναι τα εξής:

1. Δύο ή περισσότεροι όμοιοι επεξεργαστές με συγκρίσιμες ικανότητες και είναι επίσης εξοπλισμένοι με μια κρυφή μνήμη επιπέδου 1 ή και επιπέδου 2.

2. Οι επεξεργαστές αυτοί διαμοιράζονται την ίδια κύρια μνήμη και συσκευές Ε/Ε και διασυνδέονται με κάποια μορφή εσωτερικής διασύνδεσης, έτσι ώστε ο χρόνος προσπέλασης στην μνήμη να είναι περίπου ο ίδιος για κάθε επεξεργαστή. Στην προκειμένη περίπτωση ο πιο δημοφιλής τρόπος σύνδεσης είναι ένα απλός δίαυλος.

3. Όλοι οι επεξεργαστές μπορούν να εκτελέσουν τις ίδιες λειτουργίες (και αυτό λέγεται συμμετρικός πολυεπεξεργαστής).

4. Ένα ολοκληκρωμένο λειτουργικό σύστημα το οποίο είναι φορτωμένο στη κύρια μνήμη και που προσφέρει αλληλεπίδραση μεταξύ επεξεργαστών και των εφαρμογών τους σε επίπεδα έργου, διεργασίας, αρχείου και δεδομένων.

5. Διάφορα περιβάλλοντα προγραμματισμού για την ανάπτυξη διαφόρων εφαρμογών όπως Threads και OpenMP.

Οι συμβατικοί επεξεργαστές, η κύρια μνήμη και οι συσκευές Ε/Ε, χρησιμοποιούν όλα τον ίδιο δίαυλο για επικοινωνία.

Το λειτουργικό σύστημα συμμετρικού πολυεπεξεργαστή είναι απλώς ένα κανονικό λειτουργικό σύστημα. Χειρίζεται κλήσεις συστήματος, διαχειρίζεται τη μνήμη, παρέχει ένα σύστημα αρχείων και διαχειρίζει τις συσκευές Ε/Ε. Παρόλα αυτά, υπάρχουν κάποια σημεία στα οποία έχουν ξεχωριστά χαρακτηριστικά. Σε αυτά περιλαμβάνονται το χρονοπρογραμματισμό διεργασιών σε κάθε επεξεργαστή, τον συγχρονισμό μεταξύ των επεξεργαστών και η διαχείριση πόρων.

Τα περιβάλλοντα προγραμματισμού παρέχουν εργαλεία και βιβλιοθήκες με σκοπό να διεκολύνει στους προγραμματιστές την ανάπτυξη εύκολων και μεταφέρσιμων εφαρμογών. Τέτοια εργαλεία είναι συνήθως η **πολυνημάτωση** (multithreading).

Για κάθε ένα από τα βασικά συστατικά υλικού και λογισμικού που αναφέραμε προηγουμένως για το συμμετρικό πολυεπεξεργαστή θα αναφερθούμε εκτενώς στις επόμενες ενότητες του υπολοίπου κεφαλαίου.

2.3 Επεξεργαστές και Κρυφή Μνήμη

Σε αυτή την ενότητα παρουσιάζουμε συνοπτικά την ανατομία ενός εσωτερικού επεξεργαστή από ένα σύστημα πολυεπεξεργαστή. Έτσι, στο εσωτερικό του αυτόνομου επεξεργαστή περιέχει τα ακόλουθα εξαρτήματα:

- Κάποιους **καταχωρητές** (registers). Ένας καταχωρητής είναι μια θέση αποθήκευσης όπου μπορούμε να τοποθετήσουμε ένα τμήμα δεδομένων.

- Μια **αριθμητική λογική μονάδα** (arithmetic logic unit). Είναι έναν πολύ βασικό αριθμητικό υπολογιστή για τον επεξεργαστή μας που εκτελεί τις τέσσερις γνωστές αριθμητικές πράξεις.

- Μια **μονάδα ελέγχου** (control unit). Αυτή η μονάδα διευθύνει και συντονίζει τον επεξεργαστή.

- Ένα ή περισσότερα επίπεδα **κρυφής μνήμης** (cache memory). Συνήθως, σε κάθε επεξεργαστή είναι εφοδιασμένη με μια γρήγορη κρυφή μνήμη. Στην πραγματικότητα, υπάρχουν δυο αποδεκτά επίπεδα κρυφής μνήμης, γνωστά ως κρυφή μνήμη επιπέδου 1 και κρυφή μνήμη επιπέδου 2. Η κρυφή μνήμη επιπέδου 1 είναι τοποθετημένη εσωτερικά μέσα στο τσιπ του επεξεργαστή και η κρυφή μνήμη επιπέδου 2 είναι τοποθετημένη είτε εσωτερικά είτε εξωτερικά. Γενικά, η κρυφή μνήμη αποθηκεύει τα πιο χρησιμοποιούμενα δεδομένα και η οποία παίζει διπλό ρόλο: α) μειώνει το χρόνο προσπέλασης ακόμα περισσότερο και β) μειώνει τον αριθμό των προσπελάσεων στην κύρια μνήμη.

Παρακάτω εξηγούμε σύντομα τη χρήση της κρυφής μνήμης σε ένα συμβατικό υπολογιστή με ένα επεξεργαστή για καλύτερη κατανόηση των επομένων ενοτήτων. Όπως είναι γνωστό η κρυφή μνήμη μεσολαβεί μεταξύ επεξεργαστή και κύριας μνήμης. Αντί ο επεξεργαστής να συνδέεται απ' ευθείας στη κύρια μνήμη και να ζητάει ή να δίνει δεδομένα σ' αυτήν, συνδέεται πρώτα με την μικρότερη και ταχύτερη κρυφή μνήμη και αν δεν βρει εκεί τα δεδομένα που ζητάει τότε καταφεύγει στην κύρια μνήμη. Η κυρφή μνήμη φυλάσσει τμήματα δεδομένων της κύριας μνήμης που χρησιμοποιήθηκαν πιο πρόσφατα με την ελπίδα ότι αυτά θα χρησιμοποιηθούν και στο μέλλον επιταχύνοντας έτσι την τροφοδοσία του επεξεργαστή με δεδομένα λόγω της μεγάλης ταχύτητάς της.

Τέσσερις είναι οι βασικές λειτουργίες που μπορούν να συμβούν μεταξύ του επεξεργαστή και της κρυφής μνήμης:

1. Ανάγνωση ενός δεδομένου που βρίσκεται στην κρυφή μνήμη. Αυτό καλείται **επιτυχία ανάγνωσης** (read-hit) και είναι η ιδανική περίπτωση κατά την οποία ο επεξεργαστής απλά διαβάζει το περιεχόμενο της κρυφής μνήμης χωρίς να αποταθεί στην κύρια μνήμη.

2. Ανάγνωση ενός δεδομένου που δε βρίσκεται στην κρυφή μνήμη. Αυτό καλείται **αποτυχία ανάγνωσης** (read-miss) και έχει ως αποτέλεσμα να φέρουμε από την κύρια μνήμη στην κρυφή μνήμη το τμήμα που ζητήθηκε έτσι ώστε αν στο μέλλον ξαναζητηθεί να βρίσκεται διαθέσιμο. Αν η κρυφή μνήμη είναι γεμάτη θα πρέπει να αντικατασταθεί ένα παλιό τμήμα με το καινούργιο που θα έρθει. Μπορούμε για παράδειγμα να διώξουμε το τμήμα που

χρησιμοποιήθηκε πιο παλιά (μέθοδος Least Recently Used) ή να διώξουμε κάποιο τυχαίο τμήμα.

3. Εγγραφή ενός δεδομένου που βρίσκεται στην κρυφή μνήμη. Αυτό καλείται **επιτυχία εγγραφής** (write-hit) και μπορεί να δημιουργήσει ασυνέπεια περιεχομένων μεταξύ της κρυφής μνήμης και της κύριας μνήμης, δηλαδή, να έχει άλλα δεδομένα η κρυφή μνήμη ςαςηε και άλλα η κύρια μνήμη. Υπάρχουν δύο βασικοί τρόποι για να αντιμετωπιστεί το πρόβλημα της ασυνέπειας:

   - **Μέθοδος διεγγραφή** (write-through). Κατά τη μέθοδο αυτή κάθε τμήμα που γράφεται στην κρυφή μνήμη γράφεται ταυτόχρονα και στην κύρια μνήμη οπότε οι δύο μνήμες βρίσκονται πάντα σε μόνιμη συμφωνία και δεν υπάρχει πρόβλημα.
   - **Μέθοδος επανεγγραφή** (write-back). Κατά τη μέθοδο αυτή το τμήμα γράφεται στην κρυφή μνήμη αλλά όχι στην κύρια μνήμη και τότε αυτό το τμήμα καλείται 'βρώμικο'. Έτσι βέβαια υπάρχει ασυμφωνία μεταξύ των δύο επιπέδων της μνήμης αλλά αυτό δε δημιουργεί πρόβλημα αν το τμήμα σημαδευτεί (κάνοντας το λεγόμενο dirty-bit=1). Αφού ο επεξεργαστής συμβουλεύεται πρώτα την κρυφή μνήμη θα πάρει τα σωστά δεδομένα που έγραψε εκεί. Μόνο αν το τμήμα αντικατασταθεί (πχ. εξ αιτίας μιας αποτυχίας εγγραφής) τότε πρέπει να γραφτεί στην κύρια μνήμη πριν αντικατασταθεί.

4. Εγγραφή ενός δεδομένου που δε βρίσκεται στην κρυφή μνήμη. Αυτό καλείται **αποτυχία εγγραφής** (write-miss). Πάλι δύο βασικές τεχνικές μπορούν να χρησιμοποιηθούν:

   - **Μέθοδος τοποθέτησης κατά την εγγραφή** (write-allocate). Κατά τη μέθοδο αυτή το τμήμα μεταφέρεται από την κύρια μνήμη στην κρυφή μνήμη όπου και γίνεται η εγγραφή. Έχει νόημα μόνο εφόσον κατά την επιτυχία εγγραφής χρησιμοποιηθεί το πρωτόκολλο επανεγγραφή.
   - **Μέθοδος μη τοποθέτησης κατά την εγγραφή** (no-write-allocate). Κατά τη μέθοδο αυτή το τμήμα δε μεταφέρεται στην κρυφή μνήμη και η εγγραφή γίνεται απευθείας στην κύρια μνήμη. Έχει νόημα μόνο εφόσον κατά την επιτυχία εγγραφής χρησιμοποιηθεί το πρωτόκολλο διεγγραφή.

2.4 Δίαυλος

Ο δίαυλος είναι ο απλούστερος μηχανισμός για την συγκρότηση ενός συστήματος πολλών επεξεργαστών. Η δομή και οι διεπαφές είναι βασικά ίδια όπως σε ένα σύστημα ενός επεξεργαστή που χρησιμοποιεί διασύνδεση διαύλου. Ο δίαυλος αποτελείται από γραμμές ελέγχου, διευθύνσεων και δεδομένων. Για τη διεκόλυνση μεταβιβάσεων DMA από επεξεργαστές E/E, υπάρχουν τα παρακάτω χαρακτηριστικά:

- Διευθυνσιοδότηση. Πρέπει να είναι δυνατή η διάκριση μονάδων στο δίαυλο για να προσδιορίζεται η πηγή και ο προορισμός των δεδομένων.

- Διαιτησία. Κάθε μονάδα E/E μπορεί να λειτουργήσει προσωρινά ως κύρια. Υπάρχει ένας μηχανισμός διαιτησίας μεταξύ ανταγωνιστικών αιτήσεων για έλεγχο του διαύλου που χρησιμοποιεί κάποια μορφή σχεδίου προτεραιότητας.

- Χρονικός διαμοιρασμός. Όταν μια μονάδα ελέγχει το δίαυλο, οι υπόλοιποι μονάδες είναι αποκλεισμένες και πρέπει αν χρειαστεί, να αναστείλλουν την λειτουργία τους μέχρι να πετύχουν προσπέλαση στο δίαυλο.

Αυτά τα χαρακτηριστικά μονοεπεξεργαστή μπορούν να χρησιμοποιηθούν άμεσα σε αρχιτεκτονική SMP. Στην περίπτωση αυτή, υπάρχουν πολλοί επεξεργαστές καθώς και πολλοί επεξεργαστές E/E που όλοι προσπαθούν να πετύχουν προσπέλαση στη μνήμη μέσω του διαύλου. Για καλύτερη κατανόηση, ας δούμε ένα μικρό παράδειγμα όπου όλοι επεξεργαστές χωρίς κρυφή μνήμη προσπελάζουν το δίαυλο. Όταν ένας επεξεργαστής θέλει να διαβάσει μια λέξη μνήμης, ελέγχει πρώτα αν ο δίαυλος είναι απασχολημένος. Αν ο δίαυλος είναι διαθέσιμος, ο επεξεργαστής τοποθετεί τη διεύθυνση της λέξης που θέλει στο δίαυλο, κάνει θετικά μερικά σήματα ελέγχου και περιμένει μέχρι η μνήμη να τοποθετήσει τη λέξη στο δίαυλο. Αν ο δίαυλος είναι απασχολημένος, όταν ο επεξεργαστής θέλει να διαβάζει ή να γράψει στη μνήμη, ο επεξεργαστής απλώς περιμένει μέχρι να γίνει διαθέσιμος ο δίαυλος. Εδώ είναι το πρόβλημα αυτού του σχεδιασμού. Με δύο ή τρεις επεξεργαστές, ο ανταγωνισμός για το δίαυλο θα είναι υποφερτός, αλλά με 32 ή 64 επεξεργαστές θα είναι ανυπόφορος. Το σύστημα θα περιορίζεται απόλυτα από το εύρος ζώνης του διαύλου και οι περισσότεροι επεξεργαστές θα είναι αδρανείς τον περισσότερο χρόνο.

Η λύση αυτού του προβλήματος για βελτίωση της απόδοσης είναι απαραίτητο να προστεθεί μια κρυφή μνήμη σε κάθε επεξεργαστή. Το γεγονός αυτό πολλές πράξεις ανάγνωσης μπορούν να ικανοποιούνται από την τοπική κρυφή μνήμη, θα υπάρχει πολύ μικρότερη κυκλοφορία στο δίαυλο και το σύστημα θα μπορεί να υποστηρίζει περισσότερους επεξεργαστές.

Όμως, η χρήση κρυφής μνήμης εισάγει ένα βασικό πρόβλημα. Επειδή κάθε τοπική κρυφή μνήμη περιέχει μια εικόνα τμήματος μνήμης, αν υποθέσουμε ότι ο επεξεργαστής 1 τροποποιεί μια λέξη στη κρυφή της μνήμη και αμέσως μετά ο επεξεργαστής 2 διαβάζει το δικό της αντίγραφο της λέξης από την κρυφή μνήμη της. Θα διαβάσει παρωχημένα δεδομένα, παραβιάζοντας το συμφωνητικό ανάμεσα στο λογισμικό και τη μνήμη. Το πρόβλημα αυτό είναι γνωστό στη βιβλιογραφία ως πρόβλημα **συνοχής κρυφής μνήμης** (cache coherence) και είναι εξαιρετικά σοβαρό. Αν δε λυθεί, δεν μπορεί να χρησιμοποιηθεί κρυφή μνήμη και οι πολυεπεξεργαστές διαύλου θα περιορίζονταν σε δύο ή τρεις επεξεργαστές. Για αυτό το λόγο στην επόμενη ενότητα εξετάζουμε διεξοδικά την λύση του προβλήματος αυτού από την πλευρά του υλικού.

Συνεπώς, η οργάνωση ενός διαύλου έχει αρκετά πλεονεκτήματα όπως:

- Απλότητα. Είναι η απλούστερη προσέγγιση σε οργάνωση πολλών επεξεργαστών. Η φυσική διεπαφή και η λογική διευθυνσιοδότησης, διαιτησίας και χρονικού διαμερισμού κάθε επεξεργαστή παραμένουν ίδιες όπως σε σύστημα ενός επεξεργαστή.

- Ευελιξία. Η επέκταση του συστήματος είναι γενικά εύκολη με την προσθήκη περισσότερων επεξεργαστών στο δίαυλο.

- Αξιοπιστία. Ουσιαστικά, ο δίαυλος είναι παθητικό μέσο και η βλάβη κάποιας συνδεδεμένης συσκευής δεν θα πρέπει να προκαλεί βλάβη όλου του συστήματος.

2.5 Πρωτόκολλα Διατήρησης της Συνοχής μεταξύ Κρυφής και Κύριας Μνήμης

Σε σύγχρονα συστήματα πολυεπεξεργαστών συνηθίζεται να υπάρχουν ένα ή δύο επίπεδα κρυφής μνήμης που συνοδεύουν κάθε επεξεργαστή. Η οργάνωση αυτή είναι ουσιώδης για να πετύχουμε καλή λειτουργία. Δημιουργεί, ωστόσο, ένα πρόβλημα γνωστό ως πρόβλημα συνοχής κρυφής μνήμης. Η ουσία του προβλήματος είναι η εξής: αν δύο επεξεργαστές περιέχουν στις κρυφές

τους μνήμες την ίδια λέξη κι ένας από αυτούς τροποποιήσει τα περιεχόμενα της και μετά ο δεύτερος την αναγνώσει, τότε ο τελευταίος θα ανακαλέσει τα παρωχημένα της περιεχόμενα. Στην περίπτωση αυτή οι μνήμες είναι μη συνεκτικές και τα προγράμματα θα δώσουν λανθασμένα αποτελέσματα. Έχουν προταθεί διάφορες λύσεις ή αλγόριθμοι στο πρόβλημα αυτό και αυτοί οι αλγόριθμοι κρυφής μνήμης λέγονται **πρωτόκολλα συνοχής κρυφής μνήμης** (cache coherence protocols).

Ο αντικειμενικός σκοπός του κάθε πρωτοκόλλου συνοχής κρυφής μνήμης είναι να εμποδίζει να εμφανίζονται ταυτόχρονα διαφορετικές εκδόσεις της ίδιας λέξης κρυφής μνήμης σε δύο ή περισσότερες κρυφές μνήμες. Υπάρχουν δυο βασικές προσεγγίσεις για την αντιμετώπιση του προβλήματος: τα **πρωτόκολλα κατασκόπευσης** (snoopy protocols) και **τα πρωτόκολλα καταλόγου** (directory protocols). Στην ενότητα αυτή θα συζητήσουμε τα πρωτόκολλα κατασκόπευσης που είναι ιδιαίτερα δημοφιλείς για συμμετρικούς πολυεπεξεργαστές με δίαυλο. Έπειτα θα επικεντρωθούμε στην προσέγγιση που χρησιμοποιείται περισσότερο το πρωτόκολλο MESI. Το πρωτόκολλο MESI χρησιμοποιείται από τον Pentium 4 για την κατασκόπευση του διαύλου.

Η βασική ιδέα του πρωτοκόλλου κατασκόπευσης είναι ότι ο κάθε επεξεργαστής έχει ένα δικό του ελεγκτή κρυφής μνήμης ο οποίος κατασκοπεύει αδιάκοπα το κοινό δίαυλο, παρακολουθώντας όλες τις αιτήσεις διαύλου από τους άλλους επεξεργαστές και τις κρυφές μνήμες και κάνοντας κάποιες ενέργειες σε ορισμένες περιπτώσεις. Οι συσκευές αυτές λέγονται **κρυφές μνήμες κατασκόπευσης** (snooping caches).

Για καλύτερη κατανόηση εξετάζουμε το πρωτόκολλο κατασκόπευσης για τέσσερις περιπτώσεις. Ας ονομάσουμε κρυφή μνήμη 1 την κρυφή μνήμη που κάνει τις ενέργειες και κρυφή μνήμη 2 την κρυφή μνήμη κατασκόπευσης.

- Αποτυχία ανάγνωσης (δηλαδή, ανάγνωση ενός δεδομένου που δε βρίσκεται στην κρυφή μνήμη). Στην περίπτωση που η κρυφή μνήμη 1 αποτύχει σε μια ανάγνωση, κάνει μια αίτηση στο δίαυλο για να προσκομίσει μια λέξη από τη μνήμη. Η κρυφή μνήμη 2 το παρακολουθεί αυτό, αλλά δεν κάνει τίποτα.

- Επιτυχία ανάγνωσης (δηλαδή, ανάγνωση ενός δεδομένου που βρίσκεται στην κρυφή μνήμη). Στην περίπτωση που η κρυφή μνήμη 1 παρουσιάζει μια επιτυχία σε ανάγνωση, η αίτηση ικανοποιείται τοπικά και δε γίνεται καμία αίτηση στο δίαυλο. Έτσι, η κρυφή μνήμη

2 δεν είναι ενήμερη για τις επιτυχίες στις αναγνώσεις της κρυφής μνήμης 1.

- Αποτυχία εγγραφής (δηλαδή, εγγραφή ενός δεδομένου που δε βρίσκεται στην κρυφή μνήμη). Όταν ο επεξεργαστής 1 κάνει μια πράξη εγγραφής και η κρυφή μνήμη 1 αποτύχει σε μια εγγραφή, κάνει μια αίτηση στο δίαυλο. Η κρυφή μνήμη 2 το παρακολουθεί αυτό και ελέγχει αν έχει τη λέξη που γράφεται. Είτε έχει είτε δεν έχει τη λέξη, η κρυφή μνήμη 2 δεν κάνει τίποτα.

- Επιτυχία εγγραφής (δηλαδή, εγγραφή ενός δεδομένου που βρίσκεται στην κρυφή μνήμη). Όταν ο επεξεργαστής 1 κάνει μια πράξη εγγραφής και η κρυφή μνήμη 1 παρουσιάζει μια επιτυχία σε εγγραφή, κάνει μια αίτηση στο δίαυλο. Η κρυφή μνήμη 2 το παρακολουθεί αυτό και ελέγχει αν έχει τη λέξη που γράφεται. Σε περίπτωση που έχει τη λέξη, η κρυφή μνήμη 2 σημειώνει ως άκυρη την καταχώριση της κρυφής μνήμης που περιέχει τη λέξη που μόλις τροποποιήθηκε. Ουσιαστικά, διαγράφει το στοιχείο από την κρυφή μνήμη.

Είναι δυνατή μια παραλλαγή αυτού του βασικού πρωτοκόλλου. Για παράδειγμα, σε περίπτωση επιτυχίας σε εγγραφή, η κρυφή μνήμη κατασκόπευσης κανονικά ακυρώνει την καταχώριση της που περιέχει τη λέξη που γράφεται. Εναλλακτικά, θα μπορούσε να αποδέχεται τη νέα τιμή και να ενημερώνει την καταχώριση της αντί να τη σημειώνει ως άκυρη. Θεωρητικά, η ενημέρωση της κρυφής μνήμης ισοδυναμεί με την ακύρωση της και τη νέα ανάγνωση της λέξης από την κύρια μνήμη. Σε όλα τα πρωτόκολλα κρυφής μνήμης, χρειάζεται να γίνει επιλογή μεταξύ μιας στρατηγικής ενημέρωσης και μιας στρατηγικής ακύρωσης. Τα πρωτόκολλα αυτά έχουν διαφορετική απόδοση κάτω από διαφορετικές συνθήκες φόρτου. Τα μηνύματα ενημέρωσης μεταφέρουν ωφέλιμο φορτίο, γι' αυτό και είναι μεγαλύτερα από τις ακυρώσεις, όμως μπορεί να αποτρέπουν μελλοντικές αποτυχίες κρυφής μνήμης.

2.5.1 Πρωτόκολλο Συνοχής Κρυφής Μνήμης MESI

Ένα δημοφιλές πρωτόκολλο συνοχής κρυφής μνήμης με ακύρωση εγγραφής λέγεται το πρωτόκολλο MESI από τα αρχικά των ονομάτων των τεσσάρων καταστάσεων (Modified, Exclusive, Shared, Invalid) που χρησιμοποιεί. Το πρωτόκολλο αυτό χρησιμοποιείται περισσότερο στα συστήματα πολυεπεξεργαστών του εμπορίου όπως είναι οι Pentium 4 και PowerPC για την

κατασκόπευση του διαύλου. Κάθε καταχώριση κρυφής μνήμης μπορεί να είναι σε μια από τις τέσσερις παρακάτω καταστάσεις:

1. **Άκυρη** (Invalid). Η καταχώριση της κρυφής μνήμης δεν περιέχει έγκυρα δεδομένα.

2. **Μεριζόμενη** (Shared). Η λέξη μπορεί να υπάρχει σε πολλές κρυφές μνήμες και η κύρια μνήμη είναι ενημερωμένη.

3. **Αποκλειστική** (Exclusive). Η λέξη δεν υπάρχει σε καμμία άλλη κρυφή μνήμη και η κύρια μνήμη είναι ενημερωμένη.

4. **Τροποποιημένη** (Modified). Η καταχώριση είναι έγκυρη και η κύρια μνήμη είναι άκυρη ή δεν υπάρχουν αντίγραφα.

Όταν ξεκινά ο επεξεργαστής, όλες οι καταχωρίσεις της κρυφής μνήμης είναι σημειωμένες ως άκυρες. Την πρώτη φορά που γίνεται ανάγνωση από τη μνήμη, η λέξη στην οποία γίνεται αναφορά προσκομίζεται στην κρυφή μνήμη του επεξεργαστή 1 που κάνει την ανάγνωση μνήμης, σημειώνεται ότι είναι στην κατάσταση E (αποκλειστική), αφού είναι το μόνο αντίγραφο που βρίσκεται σε κρυφή μνήμη. Οι επόμενες αναγνώσις από το επεξεργαστή χρησιμοποιούν την καταχώριση της κρυφής μνήμης και δεν πηγαίνουν στο δίαυλο. Ένας άλλος επεξεργαστής 2 μπορεί επίσης να προσκομίσει την ίδια λέξη και να την αντιγράψει στην κρυφή μνήμη της, αλλά με την κατασκόπευση του διαύλου, ο αρχικός κάτοχος (ο επεξεργαστής 1) βλέπει ότι δεν είναι πια μόνος και ανακοινώνει στο δίαυλο ότι έχει και αυτός ένα αντίγραφο. Η κατάσταση και των δύο αντιγράφων σημειώνεται ως S (μεριζόμενη). Η κατάσταση S σημαίνει λοιπόν ότι η λέξη υπάρχει σε μία ή περισσότερες κρυφές μνήμες για ανάγνωση και ότι η μνήμη είναι ενημερωμένη. Οι επόμενες αναγνώσεις από ένα επεξεργαστή σε μια λέξη την οποία έχει ήδη στην κρυφή της μνήμη σε κατάσταση S δε χρησιμοποιούν το δίαυλο και δεν προκαλούν αλλαγή της κατάστασης.

Ας δούμε τώρα τι γίνεται αν ο επεξεργαστής 2 γράψει στη λέξη κρυφής μνήμης που έχει, η οποία είναι στην κατάσταση S. Στέλνει ένα σήμα ακύρωσης στο δίαυλο, ενημερώνοντας όλους τους επεξεργαστές για να διαγράψουν τα αντίγραφα τους. Το αντίγραφο που υπάρχει τώρα στην κρυφή μνήμη περνά στην κατάσταση M (τροποποιημένη). Η λέξη δε γράφεται στην κύρια μνήμη. Αξίζει να σημειώσουμε ότι, αν μια λέξη είναι στην κατάσταση E όταν γράφεται, δε χρειάζεται σήμα στο δίαυλο για να ακυρωθούν οι άλλες κρυφές μνήμες, επειδή είναι γνωστό ότι δεν υπάρχουν άλλα αντίγραφα.

Ας δούμε τώρα τι θα συμβεί αν διαβάσει τη λέξη ο επεξεργαστής 3. Ο επεξεργαστής 2 ο οποίος έχει τώρα τη λέξη, γνωρίζει ότι το αντίγραφο στη κύρια μνήμη δεν είναι έγκυρο, γι αυτό κάνει θετικό ένα σήμα στο δίαυλο για να πει στο επεξεργαστή 3 'παρακαλώ περιμένετε', ενώ γραφέι τη λέξη της πίσω στη μνήμη. Όταν τελειώσει η ενημέρωση, ο επεξεργαστής 3 προσκομίζει ένα αντίγραφο και η λέξη σημειώνεται ως μεριζόμενη και στις δύο κρυφές μνήμες. Μετά από αυτό, ο επεξεργαστής 2 ξαναγράφει στη λέξη, με αποτέλεσμα να ακυρωθεί το αντίγραφο της κρυφής μνήμης του επεξεργαστή 3.

Τέλος, ο επεξεργαστής 1 γράφει σε μια λέξη. Ο επεξεργαστής 2 βλέπει ότι γίνεται απόπειρα εγγραφής και κάνει θετικό ένα σήμα στο δίαυλο για να πει στο επεξεργαστή 1 'παρακαλώ περιμένετε', ενώ γράφει τη λέξη της πίσω στη μνήμη. Όταν τελειώσει σημειώνει το δικό της αντίγραφο ως άκυρο, επειδή γνωρίζει ότι άλλος επεξεργαστής πρόκειται να τροποποιήσει τη λέξη. Στο σημείο αυτό, έχουμε την κατάσταση όπου ένας επεξεργαστής γράφει σε μια λέξη που δεν υπάρχει σε καμία κρυφή μνήμη. Αν χρησιμοποιείται η πολιτική της τοποθέτησης κατά την εγγραφή, η λέξη θα φορτωθεί στην κρυφή μνήμη και θα σημειωθεί ότι είναι στην κατάσταση M (τροποποιημένη). Αν η πολιτική της τοποθέτησης κατά την εγγραφή δε χρησιμοποιείται, η εγγραφή θα γίνει απευθείας στη μνήμη, και η λέξη δεν θα τοποθετηθεί σε καμία κρυφή μνήμη.

2.5.2 Συνοχή Κρυφής Μνήμης L1 - L2

Μέχρι το σημείο αυτό περιγράψαμε πρωτόκολλα συνοχής κρυφής μνήμης με την βοήθεια της δραστηριότητας συνεργασίας μεταξύ των κρυφών μνημών που είναι συνδεδεμένες στο ίδιο δίαυλο. Συνήθως, αυτές οι κρυφές μνήμες είναι κρυφές μνήμες επιπέδου 2 και κάθε επεξεργαστής έχει επίσης μια κρυφή μνήμη επιπέδου 1 που δεν συνδέεται απευθείας με τον δίαυλο και που κατά συνέπεια δεν μπορεί να εμπλακεί σε πρωτόκολλο κατασκόπευσης. Έτσι, χρειάζεται κάποιος τρόπος διατήρησης της ακεραιότητας των δεδομένων και στα δύο επίπεδα και σε όλες τις κρυφές μνήμες του συμμετρικού πολυεπεξεργαστή SMP.

Η στρατηγική είναι να επεκτείνουμε το πρωτόκολλο MESI ή οποιοδήποτε άλλο πρωτόκολλο στις κρυφές μνήμες επιπέδου 1. Έτσι, κάθε λέξη στην κρυφή μνήμη επιπέδου 1 θα περιέχει μια ένδειξη της κατάστασης. Στην ουσία, αντικειμενικός σκοπός είναι ο παρακάτω: Σε κάθε λέξη που βρίσκεται και στην κρυφή μνήμη επιπέδου 2 και στην αντίστοιχη κρυφή μνήμη επιπέδου 1, η κατάσταση της λέξης L1 θα πρέπει να παρακολουθεί την κατάσταση της λέξης L2. Ένας

απλός τρόπος για να γίνει αυτό είναι να υιοθετηθεί η μέθοδος διεγγραφής μέσα στην κρυφή μνήμη L1. Στην περίπτωση αυτή, η διεγγραφή θα γίνεται στην κρυφή μνήμη L1 και όχι στην κύρια μνήμη. Η μέθοδος διεγγραφής στην κρυφή μνήμη L1 προωθεί οποιαδήποτε τροποποίηση σε λέξη L1 προς τα έξω στην κρυφή μνήμη L2 και κατά συνέπεια την κάνει ορατή στις άλλες κρυφές μνήμες L2. Η χρήση της μεθόδου διεγγραφής στην L1 ζητά το περιεχόμενο της L1 να είναι υποσύνολο του περιεχομένου της L2. Το γεγονός αυτό με την σειρά του υποδηλώνει ότι η προσεταιριστικότητα της κρυφής μνήμης L2 θα πρέπει να είναι ίση με ή μεγαλύτερη από την προσεταιριστικότητα της L1.

Αν η κρυφή μνήμη L1 έχει την μέθοδο επανεγγραφή, η σχέση μεταξύ των δύο κρυφών μνημών θα είναι πολύπλοκη. Υπάρχουν πολλές προσεγγίσεις για διατήρηση της συνοχής αλλά δεν θα αναφέρθουμε σε αυτές διότι ξεφεύγουμε από τους στόχους του βιβλίου.

2.6 Λειτουργικό Σύστημα

Ένα λειτουργικό σύστημα SMP διαχειρίζεται τον επεξεργαστή και τους άλλους πόρους του υπολογιστή έτσι ώστε ο χρήστης να αντιλαμβάνεται ένα μόνο λειτουργικό σύστημα που ελέγχει τους πόρους του συστήματος. Μάλιστα μια τέτοια διαμόρφωση θα έπρεπε να εμφανίζεται σαν σύστημα πολυπρογραμματισμού με ένα επεξεργαστή. Και στις δύο περιπτώσεις SMP και μονοεπεξεργαστή, ανά πάσα στιγμή μπορεί να έχει ενεργές πολλές διεργασίες και ευθύνη του λειτουργικού συστήματος είναι να κάνει χρονική ανάθεση της εκτέλεσης τους και να αναθέτει πόρους. Ένας χρήστης μπορεί να αναπτύξει εφαρμογές που χρησιμοποιούν πολλές διεργασίες ή πολλά νήματα εντός διεργασιών χωρίς να δίνει σημασία αν υπάρχουν ένας ή πολλοί επεξεργαστές. Έτσι ένα λειτουργικό σύστημα πολλών επεξεργαστών θα πρέπει να προσφέρει όλη την λειτουργικότητα ενός συστήματος πολυπρογραμματισμού συν επιπλέον χαρακτηριστικά για εξυπηρέτηση πολλών επεξεργαστών. Επίσης, πρέπει να σημειώσουμε εδώ ότι υπάρχει ένα αντίγραφο του λειτουργικού συστήματος στη κύρια μνήμη αλλά μπορεί να το προσπελάσει οποιοδήποτε επεξεργαστή ο οποίος εκτελεί τις διεργασίες του κοινόχρηστου λειτουργικού συστήματος. Όταν συμβαίνει μια κλήση συστήματος, ο αντίστοιχος επεξεργαστής περνάει σε κατάσταση λειτουργίας πυρήνα και επεξεργάζεται την κλήση συστήματος. Στο υπόλοιπο της ενότητας περιγράφουμε συνοπτικά τις υπηρεσίες που μπορεί να προσφέρει ένα λειτουργικό σύστημα SMP.

2.6.1 Διαχείριση Συγχρονικών Διεργασιών

Οι κλήσεις του λειτουργικού συστήματος θα πρέπει να είναι επαναεισαγόμενες έτσι ώστε να επιτρέπουν την συγχρονική εκτέλεση του ίδιου κώδικα από πολλούς επεξεργαστές. Ενώ δύο ή περισσότεροι επεξεργαστές εκτελούν το ίδιο ή διαφορετικά τμήματα του λειτουργικού συστήματος την ίδια χρονική στιγμή, θα πρέπει οι πίνακες του λειτουργικού συστήματος και οι δομές διαχείρισης να υφίστανται κατάλληλη διαχείριση έτσι ώστε να αποφεύγονται αδιέξοδα και μη έγκυρες πράξεις. Ο απλούστερος τρόπος να λυθούν αυτά τα προβλήματα είναι να αντιστοιχεί ένα κλείδωμα (ή ένα mutex) στο λειτουργικό σύστημα ή ορισμένα τμήματα του, ώστε ολόκληρο το σύστημα ή τα τμήματα του να αποτελούν μια κρίσιμη περιοχή. Όταν κάποιος επεξεργαστής θέλει να εκτελέσει κώδικα λειτουργικού συστήματος ή ορισμένων τμημάτων του, πρέπει πρώτα να αποκτήσει το mutex. Αν το mutex είναι κλειδωμένο, ο επεξεργαστής απλώς περιμένει. Με αυτό τον τρόπο, κάθε επεξεργαστής μπορεί να εκτελέσει το λειτουργικό σύστημα ή τμήμα του αλλά μόνο ένας επεξεργαστής εκτελεί κώδικα λειτουργικού συστήματος κάθε χρονική στιγμή.

2.6.2 Συγχρονισμός Πολυεπεξεργαστών

Επειδή πολλές ενεργές διεργασίες έχουν προσπέλαση σε κοινή μνήμη ή ακόμα σε κοινούς πόρους Ε/Ε, θα πρέπει να υπάρχει φροντίδα παροχής αποτελεσματικού συγχρονισμού. Ο συγχρονισμός είναι μια δυνατότητα που επιβάλει αμοιβαίους αποκλεισμούς και διάταξη γεγονότων. Είδαμε προηγουμένως την περίπτωση κατά την οποία οι κρίσιμες περιοχές του πυρήνα και οι πίνακες πρέπει να προστατεύονται με κλειδώματα. Αυτή η περίπτωση συγχρονισμού υλοποιείται στο πολυεπεξεργαστή με τη χρήση ειδικών εντολών του υλισμικού που συγχρονίζουν την πρόσβαση σε κοινές δομές δεδομένων στην κοινή μνήμη.

2.6.3 Χρονοπρογραμματισμός

Οποιοδήποτε επεξεργαστής μπορεί να εκτελεί χρονική ανάθεση και έτσι πρέπει να αποφεύγονται συγκρούσεις. Βασικό χαρακτηριστικό του χρονοπρογραμματισμού είναι μια κεντρική λίστα ετοίμων διεργασιών ή ακόμα καλύτερα ένα σύνολο από λίστες για τις διεργασίες με διαφορετικές προτεραιότητες που είναι αποθηκευμένες στην κοινή μνήμη. Το τμήμα χρονοπρογραμματισμού θα πρέπει να αναθέτει έτοιμες διεργασίες σε διαθέσιμους επεξεργαστές. Συγκεκριμένα, όταν

ένας επεξεργαστής ολοκληρώσει την εκτέλεση της τρέχουσας διεργασίας, κλειδώνει τις λίστες χρονοπρογραμματισμού (εφόσον βρίσκονται στην κύρια μνήμη) και επιλέγει τη διεργασία με την υψηλότερη προτεραιότητα.

2.6.4 Διαχείριση Μνήμης

Η διαχείριση μνήμης σε πολυεπεξεργαστή θα πρέπει να ασχολείται με όλα τα θέματα που βρίσκουμε στους υπολογιστές ενός επεξεργαστή. Επίσης, θα πρέπουν να συντονιστούν οι μηχανισμοί σελιδοποίησης σε διαφορετικούς επεξεργαστές έτσι ώστε να επιβάλλουν συνοχή όταν πολλοί επεξεργαστές διαμοιράζονται μια σελίδα και να αποφασίζουν την αντικατάσταση της σελίδας.

2.6.5 Αξιοπιστία και Ανοχή σε Σφάλματα

Το λειτουργικό σύστημα θα πρέπει να προσφέρει ευπρεπή υποβάθμιση σε περίπτωση βλάβης επεξεργαστή. Ο χρονοπρογραμματισμός και άλλα τμήματα του λειτουργικού συστήματος θα πρέπει να αναγνωρίζουν την απώλεια ενός επεξεργαστή και να αναδομούν κατάλληλα τους πίνακες διαχείρισης.

2.7 Περιβάλλοντα Προγραμματισμού και Εργαλεία

Ο συμμετρικός πολυεπεξεργαστής διαθέτει ορισμένα εργαλεία για την εύκολη υλοποίηση παράλληλων και κατανεμημένων εφαρμογών. Το πιο δημοφιλές εργαλείο που χρησιμοποιείται είναι τα **νήματα** (threads).

2.7.1 Νήματα

Τα πολυνηματικά προγράμματα που χρησιμοποιούν πολλά νήματα συγχρονικά ταιριάζουν καλύτερα στην αρχιτεκτονική συμμετρικού πολυεπεξεργαστή επειδή τα νήματα που ανήκουν σε μια διεργασία διαμοιράζουν τους διαθέσιμους πόρους του συστήματος. Ένα εργαλείο λογισμικού που υποστηρίζει πολυνηματικό προγραμματισμό περιλαμβάνει συνήθως την δημιουργία και το συντονισμό των νημάτων. Συνεπώς, για την συγγραφή πολυνηματικών προγραμμάτων χρησιμοποιείται η βιβλιοθήκη νημάτων POSIX η οποία λέγεται pthreads. Η βιβλιοθήκη βασίζεται σε

μια συλλογή συναρτήσεων οι οποίες καλούνται μέσα από ένα παράλληλο πρόγραμμα. Ενδεικτικές συναρτήσεις που περιλαμβάνονται στην παραπάνω βιβλιοθήκη είναι δημιουργία νήματος, συγχρονισμός νημάτων και τερματισμός νήματος. Επίσης, υπάρχει μια επιπλέον διαδεδομένη βιβλιοθήκη OpenMP που χρησιμοποιεί οδηγίες προς το μεταγλωττιστή οι οποίες ενσωματώνονται στο παράλληλο πρόγραμμα. Οι παραπάνω βιβλιοθήκες λογισμικού μπορούν να χρησιμοποιηθούν σε συνδυασμό με την γλώσσα προγραμματισμού C. Τέλος, για την ανάπτυξη πολυνηματικών εφαρμογών με την γλώσσα Java υποστηρίζονται από τα πρότυπα Java Threads και Java OpenMP - JOMP.

Κεφάλαιο 3

Συστοιχία Υπολογιστών

Αυτό το κεφάλαιο εξετάζουμε τα κίνητρα για την μετατόπιση σε συστήματα υπολογιστών χαμηλού κόστους όπως η συστοιχία υπολογιστών. Γίνεται μια περιγραφή της αρχιτεκτονικής συστοιχίας υπολογιστών και επίσης περιγράφουμε τα συστατικά υλικού που την αποτελούν. Στην συνέχεια, παρουσιάζουμε τα συσταστικά λογισμικού όπως το λογισμικό συστήματος της συστοιχίας για τη σωστή λειτουργία της και το λογισμικό εφαρμογών που διεκολύνει στους προγραμματιστές και χρήστες την ανάπτυξη παράλληλων εφαρμογών. Το κεφάλαιο τελειώνει με μια περιγραφή της επέκτασης συστοιχίας υπολογιστών σε συστοιχία από συμμετρικούς πολυεπεξεργαστές (SMPs).

3.1 Υπολογισμοί Χαμηλού Κόστους και Κίνητρα

Η χρήση των παράλληλων συστημάτων σαν μέσο για την παροχή υπολογισμών υψηλής απόδοσης για μεγάλα μεγέθη εφαρμογών έχουν ερευνηθεί εκτενώς στα τελευταία χρόνια. Όμως μέχρι πρόσφατα, τα οφέλη από αυτήν την έρευνα περιορίστηκαν στα άτομα τα οποία είχαν πρόσβαση σε μεγάλες και ακριβές υπολογιστικές πλατφόρμες. Σήμερα η τάση στον παράλληλο υπολογισμό μετακινείται από τις ειδικές υπερυπολογιστικές πλατφόρμες σε φθηνά συστήματα γενικού σκοπού όπως η συστοιχία υπολογιστών που αποτελείται από επεξεργαστές ή σταθμούς εργασίας ή συμμετρικοί πολυεπεξεργαστές. Επιπλέον, ακόμη και στη περίπτωση των πολυεπεξεργαστών ή άλλων αρχιτεκτονικών βελτιώσεων της απόδοσης των υπολογιστών, η τάση βρίσκεται στο σχεδιασμό συνδεδεμένων συν-επεξεργαστών (attached co-processor) που προστίθενται στα

παραδοσιακά συστήματα με κάρτες επέκτασης. Η συστοιχία υπολογιστών περιλαμβάνει πολλά πλεονεκτήματα όπως η κατασκευή μιας τέτοιας πλατφόρμας με χαμηλό προυπολογισμό ώστε να είναι κατάλληλη για μια μεγάλη ποικιλία εφαρμογών. Αυτή η μετακίνηση οφείλεται κυρίως στις πρόσφατες εξελίξεις στα δίκτυα υψηλής ταχύτητας και την βελτιωμένη απόδοση των μικροεπεξεργαστών και των σταθμών εργασίας. Αυτές οι εξελίξεις σημαίνουν ότι η συστοιχία υπολογιστών γίνεται ένα αποτελεσματικό μέσο για παράλληλο και κατανεμημένο υπολογισμό. Ένας ακόμη σημαντικός παράγοντας είναι η προτυποποίηση πολλών εργαλείων που χρησιμοποιούνται από τις παράλληλες εφαρμογές. Παραδείγματα τέτοιων προτύπων εργαλείων είναι η βιβλιοθήκη μεταβίβασης μηνυμάτων MPI (Message Passing Interface) [8, 7] και η γλώσσα παραλληλισμού δεδομένων (data-parallel) HPF (High Performance Fotran) [4]. Η προτυποποίηση καθιστά δυνατή την ανάπτυξη, δοκιμή και εκτέλεση των εφαρμογών σε μια συστοιχία υπολογιστών και στο τελευταίο στάδιο τη μεταφορά τους με λίγη τροποποίηση πάνω στις αποκλειστικές παράλληλες πλατφόρμες. Η ακόλουθη λίστα υπογραμμίζει μερικούς από τους λόγους που η συστοιχία υπολογιστών χρησιμοποιείται περισσότερο σε σχέση με τους ειδικούς παράλληλους υπολογιστές [1]:

- Οι σταθμοί εργασίας γίνονται ολοένα και ταχύτεροι. Η απόδοση του σταθμού εργασίας έχει αυξηθεί δραματικά στα τελευταία χρόνια και διπλασιάζεται κάθε 18 έως 24 μήνες. Αυτό φαίνεται από το γεγονός ότι στην αγορά κυκλοφορούν ταχύτεροι επεξεργαστές και αποτελεσματικοί συμμετρικοί πολυεπεξεργαστές (SMP).

- Το **εύρος ζώνης των επικοινωνιών** (communication bandwidth) ανάμεσα στους σταθμούς εργασίας αυξάνεται και η **καθυστέρηση** (latency) μειώνεται καθώς καινούργιες τεχνολογίες δικτύων και πρωτοκόλλων υλοποιούνται στα τοπικά δίκτυα.

- Οι συστοιχίες σταθμών εργασίας είναι πιο εύκολο να ολοκληρωθούν ή να ενσωματωθούν στα υπάρχοντα δίκτυα από ότι οι ειδικοί παράλληλοι υπολογιστές.

- Οι προσωπικοί σταθμοί εργασίας δεν χρησιμοποιούνται τόσο αποδοτικά από τους χρήστες, άρα υπάρχει διαθέσιμη λανθάνουσα ισχύς.

- Η ανάπτυξη εργαλείων για τους σταθμούς εργασίας είναι πιο ώριμη σε σχέση με τους ειδικούς παράλληλους υπολογιστές και αυτό οφείλεται κυρίως στην έλλειψη προτύπων που υπάρχουν στα περισσότερα παράλληλα συστήματα.

- Οι συστοιχίες σταθμών εργασίας είναι φθηνές και εύκολα διαθέσιμες σε σχέση με τις ειδικές παράλληλες υπολογιστικές πλατφόρμες.

- Οι συστοιχίες μπορούν εύκολα να επεκταθούν και η χωρητικότητα του κάθε κόμβου (node) μπορεί εύκολα να αυξηθεί, εγκαθιστώντας επιπλέον μνήμη ή επεξεργαστές.

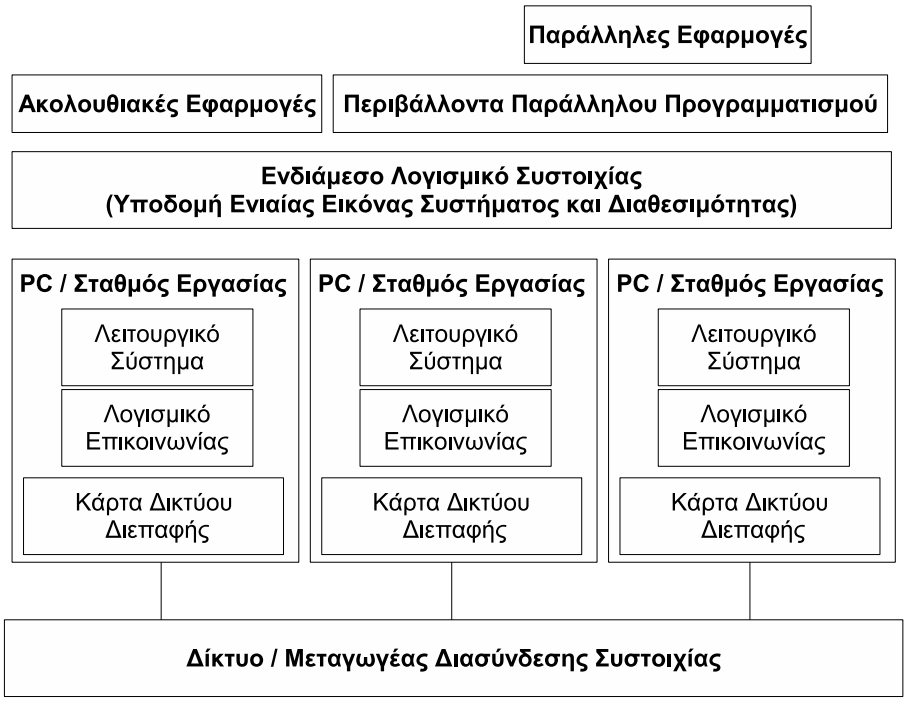
Σε βασικό επίπεδο, μια συστοιχία είναι μια συλλογή από σταθμούς εργασίας ή υπολογιστές που είναι διασυνδεδεμένοι μέσω τοπικού δικτύου. Για τους σκοπούς της παράλληλης επεξεργασίας χρησιμοποιούμε τους υπολογιστές της συστοιχίας μαζί για την επίλυση μιας μεγάλης εφαρμογής. Η συστοιχία σταθμών εργασίας είναι κατάλληλη για εφαρμογές που δεν έχουν υψηλές απαιτήσεις επικοινωνίας αφού ένα τοπικό δίκτυο έχει υψηλές καθυστερήσεις και χαμηλό εύρος ζώνης επικοινωνίας. Αν όμως οι εφαρμογές έχουν υψηλές απαιτήσεις επικοινωνίας, τότε οι συστοιχίες πρέπει να αποτελούνται από σταθμούς εργασίας υψηλής απόδοσης και το δίκτυο να έχει υψηλό εύρος ζώνης και χαμηλή καθυστέρηση. Μια τέτοια συστοιχία μπορεί να παρέχει γρήγορες και αξιόπιστες υπηρεσίες στις υπολογιστικά απαιτητικές εφαρμογές ακόμα και σε εφαρμογές με υψηλές απαιτήσεις επικοινωνίας.

3.2 Αρχιτεκτονική μιας Συστοιχίας Σταθμών Εργασίας

Μια συστοιχία ή ένα δίκτυο σταθμών εργασίας είναι ένας είδος παράλληλου ή κατανεμημένου συστήματος η οποία αποτελείται από μια συλλογή διασυνδεδεμένων ανεξάρτητων κόμβων υπολογιστών που δουλεύουν μαζί ως ένας ολοκληρωμένος υπολογιστικός πόρος.

Ένας **κόμβος υπολογιστής** (computer node) μπορεί να είναι μια μηχανή (δηλαδή, προσωπικός υπολογιστής ή σταθμός εργασίας) ή ένα σύστημα πολυεπεξεργασίας (δηλαδή, SMP) με μνήμη, με λειτουργίες Εισόδου/Εξόδου (Ε/Ε) και με ένα λειτουργικό σύστημα. Γενικά, ο όρος συστοιχία σημαίνει ότι συνδέονται δύο ή περισσότεροι υπολογιστές (ή κόμβοι) μαζί. Οι κόμβοι μπορεί να βρίσκονται σε ένα χώρο ή να είναι ξεχωριστά και να συνδέονται μέσω τοπικού δικτύου. Μια διασυνδεδεμένη συστοιχία υπολογιστών εμφανίζεται σαν ένα μοναδικό και ενιαίο σύστημα στους χρήστες και στις εφαρμογές. Η αρχιτεκτονική μιας τέτοιας συστοιχίας παρουσιάζεται στο Σχήμα 3.1.

Τα βασικά συστατικά μιας συστοιχίας υπολογιστών είναι τα εξής:



Σχήμα 3.1: Αρχιτεκτονική μιας συστοιχίας σταθμών εργασίας

1. Πολλαπλοί ανεξάρτητοι υπολογιστές υψηλής απόδοσης (όπως PCs, σταθμούς εργασίας ή SMPs)

2. Σύγχρονα λειτουργικά συστήματα (όπως πυρήνας (micro-kernel))

3. Δίκτυα ή μεταγωγείς υψηλής απόδοσης (όπως Fast Ethernet, Gigabit Ethernet και Myrinet)

4. Κάρτες δικτύων διεπαφής (Network Interface Cards - NICs)

5. Πρωτόκολλα επικοινωνίας και υπηρεσιών (όπως TCP/IP, Active και Fast Messages)

6. Υποδομή λογισμικού συστήματος της συστοιχίας (Cluster Middleware)

7. Περιβάλλοντα και εργαλεία παράλληλου προγραμματισμού (όπως μεταγλωτιστές, PVM (Parallel Virtual Machine) και MPI (Message Passing Interface))

8. Ακολουθιακές και παράλληλες κατανεμημένες εφαρμογές.

Η κάρτα δικτύου διεπαφής λειτουργεί σαν επεξεργαστής επικοινωνίας και είναι υπεύθυνη για τη μετάδοση και την λήψη πακέτων δεδομένων ανάμεσα στους κόμβους της συστοιχίας μέσω δικτύου.

Το λογισμικό επικοινωνίας προσφέρει μια γρήγορη και αξιόπιστη επικοινωνία μεταξύ των κόμβων της συστοιχίας και με τον εξωτερικό κόσμο. Συνήθως, η συστοιχία με ένα ειδικό δίκτυο όπως το δίκτυο Myrinet χρησιμοποιεί πρωτόκολλα επικοινωνίας όπως τα active messages για ταχύτερη επικοινωνία μεταξύ των κόμβων της συστοιχίας.

Η υποδομή λογισμικού συστήματος της συστοιχίας προσφέρει την δυνατότητα η ίδια συστοιχία να φαίνεται σαν ένα μοναδικό και ενιαίο παράλληλο σύστημα. Πρόκειται για ειδικό λογισμικό που επιτρέπει την ενιαία εγκατάσταση λογισμικού, διαχείριση υλικού και αποθηκευτικού χώρου, διαχείριση των διαφόρων πόρων κατά την εκτέλεση, όπως για παράδειγμα NPACI Rocks, Globus, κλπ.

Τέλος, τα περιβάλλοντα παράλληλου προγραμματισμού προσφέρουν μεταφέρσιμα και αποτελεσματικά εργαλεία για την εύκολη ανάπτυξη των εφαρμογών. Τέτοια εργαλεία είναι συνήθως βιβλιοθήκες μεταβίβασης μηνυμάτων (message passing libraries), αποσφαλματωτές (debuggers) και προφίλ απόδοσης παράλληλων προγραμμάτων (profilers).

Για κάθε ένα από τα συστατικά υλικού και λογισμικού που αναφέραμε προηγουμένως για την κατασκευή συστοιχίας θα αναφερθούμε εκτενώς στις επόμενες ενότητες του υπολοίπου κεφαλαίου.

3.3 Ταξινομήσεις Συστοιχιών

Οι συστοιχίες ταξινομούνται σε πολλές κατηγορίες σύμφωνα με τα παρακάτω κριτήρια [1]:

1. **Σκοπός εφαρμογής** (application target): Με βάση το κριτήριο αυτό οι συστοχίες ταξινομούνται σε δύο κατηγορίες: **συστοιχίες υψηλής απόδοσης** (high performance clusters) και **συστοιχίες υψηλής διαθεσιμότητας** (high availability clusters). Οι συστοιχίες υψηλής απόδοσης προσφέρονται για εφαρμογές με μεγάλες υπολογιστικές απαιτήσεις ενώ οι συστοιχίες υψηλής διαθεσιμότητας παρέχονται για εφαρμογές με ζωτικής σημασίας (όπως στην περίπτωση των διακομιστών Ιστού).

2. **Ιδιοκτησία κόμβου** (node ownership). Με βάση το κριτήριο αυτό οι συστοιχίες διακρίνονται σε δύο κατηγορίες: **αποκλειστική συστοιχία** (dedicated cluster) και μη **αποκλειστική συστοιχία** (nondedicated cluster). Η αποκλειστική συστοιχία αναφέρεται στο γεγονός ότι κάθε σταθμός εργασίας εκτελεί αποκλειστικά εργασίες ενός παράλληλου υπολογισμού, ενώ η μη αποκλειστική συστοιχία αναφέρεται στο ότι κάθε σταθμός εργασίας εκτελεί τις κανονικές του ρουτίνες και χρησιμοποιεί μόνους τους αδρανείς κύκλους της KME για να εκτελέσει παράλληλες εργασίες.

3. **Υλικό κόμβου** (node hardware). Οι συστοιχίες μπορεί να αποτελούνται από προσωπικούς υπολογιστές ή από σταθμούς εργασίας ή από μηχανές πολυπεπεξεργασίας όπως για παράδειγμα SMPs ή ακόμη και από επιμέρους τοπικά δίκτυα (grid). Στο τέλος του κεφαλαίου θα εξετάσουμε την συστοιχία από συμμετρικούς πολυεπεξεργαστές.

4. **Λειτουργικό σύστημα κόμβου** (node operating system). Οι κόμβοι μιας συστοιχίας μπορεί να εκτελούν ένα συγκεκριμένο λειτουργικό σύστημα όπως το Linux, Solaris κλπ. Έτσι βάση του λειτουργικού συστήματος μπορούμε να έχουμε διάφορες κατηγορίες συστοιχιών όπως συστοιχίες Linux (π.χ. το έργο Beowulf), συστοιχίες Solaris (π.χ. το έργο Berkeley NOW) κλπ

5. **Διαμόρφωση κόμβου** (node configuration). Το κριτήριο αυτό αφορά στην αρχιτεκτονική του κάθε κόμβου της συστοιχίας. Έτσι έχουμε τις **ομοιογενείς συστοιχίες** (homogeneous clusters) και τις **ετερογενείς συστοιχίες** (heterogeneous clusters). Οι ομοιογενείς συστοιχίες έχουν κόμβους με παρόμοιες αρχιτεκτονικές (δηλαδή, όλοι οι κόμβοι έχουν τις ίδιες ταχύτητες επεξεργασίας) και εκτελούν το ίδιο λειτουργικό σύστημα, ενώ οι ετερογενείς συστοιχίες έχουν κόμβους που έχουν διαφορετικές αρχιτεκτονικές (δηλαδή, όλοι οι κόμβοι έχουν διαφορετικές ταχύτητες επεξεργασίας) και εκτελούν διαφορετικό λειτουργικό σύστημα.

Ξεχωριστές συστοιχίες υπολογιστών μπορούν να διασυνδεθούν μεταξύ τους για να σχηματίζουν ένα μεγαλύτερο σύστημα (ή συστοιχία από συστοιχίες). Η χρήση των δικτύων ευρείας περιοχής για την διασύνδεση των επιμέρους συστοιχιών για υπολογισμούς υψηλής απόδοσης δημιουργεί ένα νέο πεδίο που λέγεται **πλέγμα υπολογιστών** (grid computing). Στο κεφάλαιο θα αναφερθούμε περισσότερες λεπτομέρειες για το πλέγμα υπολογιστών.

3.4 Υπολογιστές

Σε αυτή την ενότητα θα παρουσιάζουμε συνοπτικά την ανατομία ενός σύγχρονου ψηφιακού υπολογιστή που χρησιμοποιείται στην συστοιχία. Στο εσωτερικό ενός ψηφιακού υπολογιστή περιλαμβάνει τα εξής:

- Ο **επεξεργαστής** (processor). Αυτό το συσταστικό εκτελεί μια σειρά από εντολές (που ονομάζονται πρόγραμμα) και ελέγχει τη δραστηριότητα όλων των άλλων συστατικών εντός του υπολογιστή.

- Τα **τσιπ μνήμης** (chips memory) και η **κρυφή μνήμη** (cache memory). Τα τσιπ μνήμης χρησιμοποιούνται για την αποθήκευση των δεδομένων και των εντολών του προγράμματος. Ο χρόνος προσπέλασης της είναι μικρός και ανεξάρτητος από τη διεύθυνση που προσπελάζεται. Η κρυφή μνήμη αποθηκεύει τα πιο χρησιμοποιούμενα δεδομένα και μειώνει το χρόνο προσπέλασης ακόμα περισσότερο.

- Τα υπόλοιπα συσταστικά περιλαμβάνουν τον **σκληρό δίσκο** (hard disk), την **κάρτα δικτύου** (network card) και το τροφοδοτικό. Στους υπολογιστές της συστοιχίας δεν χρησιμοποιούν συσκευές εισόδου/εξόδου όπως πληκτρολόγια ή οθόνες παρά μόνο σε ένα υπολογιστή που λειτουργεί ως κόμβος συντονιστής.

Ο επεξεργαστής είναι η καρδιά ενός υπολογιστή και αξίζει να ρίξουμε μια ματιά στο εσωτερικό του. Έτσι, στο εσωτερικό του επεξεργαστή περιέχει τα ακόλουθα εξαρτήματα:

- Κάποιους **καταχωρητές** (registers). Ένας καταχωρητής είναι μια θέση αποθήκευσης όπου μπορούμε να τοποθετήσουμε ένα τμήμα δεδομένων.

- Μια **αριθμητική λογική μονάδα** (arithmetic logic unit). Είναι έναν πολύ βασικό αριθμητικό υπολογιστή για τον επεξεργαστή μας που εκτελεί τις τέσσερις γνωστές αριθμητικές πράξεις.

- Μια **μονάδα ελέγχου** (control unit). Αυτή η μονάδα διευθύνει και συντονίζει τον επεξεργαστή.

- Κάποιους **διαύλους** (buses). Αυτοί οι δίαυλοι επιτρέπουν να μετακινούν δεδομένα, διευθύνσεις και εντολές από ένα εξάρτημα σε άλλο.

Με τα τόσα λίγα εξαρτήματα ο επεξεργαστής ενός υπολογιστή μπορεί πραγματικά να εκτελέσει δύσκολες και απαιτητικές εργασίες.

3.5 Λειτουργικά Συστήματα

Το υλικό ενός υπολογιστή της συστοιχίας υπάρχει ανάγκη για λογισμικό συστήματος που είναι απαραίτητο να χρησιμοποιηθεί ώστε ο χρήστης να μην ασχολείται με τη ΄γυμνή΄ μηχανή. Η πιο κοινή μορφή λογισμικού συστήματος είναι το **λειτουργικό σύστημα** (operating system). Το λειτουργικό σύστημα μπορεί να είναι τόσο απλό όσο ένα μεμονωμένο πρόγραμμα monitor (το οποίο είναι ένα πρόγραμμα το οποίο χρησιμοποιείται σε μια μικρή συσκευή για να βοηθά τους προγραμματιστές να φορτώνουν και να εκτελούν τα προγράμματα τους) ή τόσο σύνθετο όσο ένα σύνολο προγραμμάτων όπως τα Microsoft Windows NT ή το λειτουργικό σύστημα ανοικτού κώδικα Linux.

Ο στόχος ενός λειτουργικού συστήματος είναι να αποκρύψει την λειτουργία του υποκείμενου υλικού υπολογιστή από το χρήστη και παρέχει ένα εύχρηστο περιβάλλον εργασίας στο χρήστη για να ολοκληρώσει συγκεκριμένες εργασίες εύκολα όπως η συγγραφή ενός εγγράφου κλπ. Με άλλα λόγια, το λειτουργικό σύστημα λειτουργεί ως μια ενδιάμεση διεπαφή ανάμεσα στο υλικό του υπολογιστή και στους χρήστες και καθορίζει τον τρόπο λειτουργίας του συστήματος υπολογιστή, ελέγχοντας και συντονίζοντας τη χρήση των πόρων του.

Στον πυρήνα του, το λειτουργικό σύστημα παρέχει ένα πλήθος υπηρεσιών στο χρήστη. Οι πιο κοινές από αυτές περιλαμβάνουν:

- Διαχείριση διεργασιών - η εκτέλεση πολλαπλών προγραμμάτων συγχρόνως ή ταυτοχρόνως. Η δυνατότητα εκτέλεσης πολλαπλών προγραμμάτων είναι γνωστή ως **πολυδιεργασία** (multitasking).

- Διαχείριση μνήμης - η διαχείριση του περιοριζόμενου πόρου της μνήμης για όλα τα εκτελούμενα προγράμματα.

- Διαχείριση συστήματος αρχείων - η παροχή μιας τυποποιημένης διεπαφής σε όλα τα αρχεία που είναι διαθέσιμα στο λειτουργικό σύστημα.

- Επικοινωνία με τις συσκευές υλισμικού - έτσι ώστε όλα τα προγράμματα να μπορούν να χρησιμοποιούν τις προσαρτημένες συσκευές μέσω μιας τυποποιημένης διεπαφής.

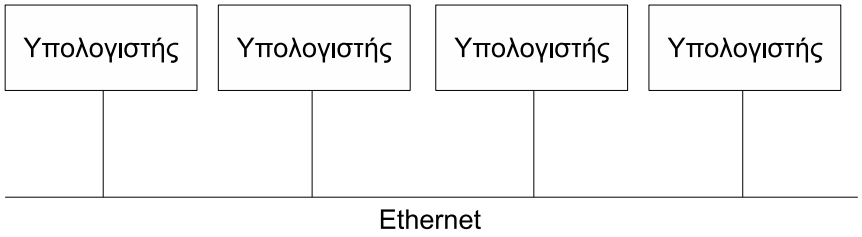
Τα τελευταία χρόνια υπάρχει η τάση να προστίθεται μια νέα έννοια στις υπηρεσίες λειτουργικού συστήματος που είναι η υποστήριξη πολλαπλών νημάτων ελέγχου μέσα σε μια διεργασία ή πρόγραμμα. Αυτή η έννοια εισάγει μια νέα διάσταση στην παράλληλη επεξεργασία, η συγχρονική εκτέλεση πολλών νημάτων μέσα στην ίδια διεργασία. Έτσι, λειτουργικά συστήματα με υποστήριξη νημάτων σημαίνει ότι υπάρχει ένα μοναδικό χώρο διευθύνσεων ανάμεσα σε πολλαπλά νήματα ελέγχου. Ο προγραμματισμός μιας διεργασίας που να εισάγει παραλληλισμός ή να δημιουργήσει πολλαπλά νήματα ελέγχου είναι γνωστός ως **πολυνημάτωση** (multithreading).

Τα λειτουργικά συστήματα κάθε υπολογιστή της συστοιχίας είναι ευσταθή και πέρα από την παροχή βασικών υπηρεσιών υποστηρίζουν πρόσθετες υπηρεσίες όπως πολυδιεργασιακή, πολυνημάτωση και δικτύωση. Οι προσθήκες αυτές δεν αλλάζουν τη βασική δομή του λειτουργικού συστήματος.

3.6 Τεχνολογίες Τοπικών Δικτύων Διασύνδεσης

Οι υπολογιστές της συστοιχίας επικοινωνούν μέσω τεχνολογιών δικτύων υψηλής ταχύτητας χρησιμοποιώντας ένα πρωτόκολλο δικτύωσης όπως είναι το TCP/IP ή ένα πρωτόκολλο χαμηλού επιπέδου όπως τα Active Messages. Επίσης, υπάρχουν δυο μετρικές για να μετρήσουμε την απόδοση των δικτύων: το **εύρος ζώνης** (bandwidth) και η **καθυστέρηση** (latency). Το εύρος ζώνης είναι η ποσότητα δεδομένων που μπορεί να μεταδοθεί στο υλικό δικτύων μέσα σε μια σταθερή χρονική περίοδο ενώ η καθυστέρηση είναι ο χρόνος για να προετοιμάσει και να μεταδώσει τα δεδομένα από έναν υπολογιστή αφετηρίας σε έναν υπολογιστή προορισμού.

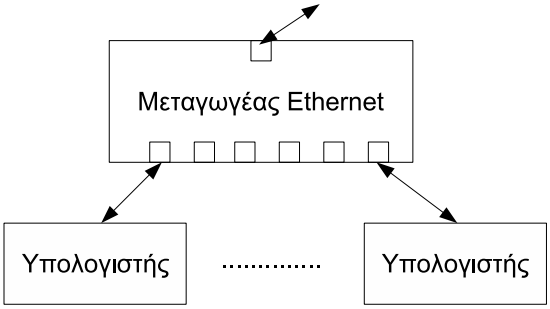
Το αρχικό πρότυπο για την σύνδεση των υπολογιστών της συστοιχίας ήταν το κλασικό Ethernet. Το Ethernet το οποίο περιγράφεται στο Πρότυπο IEEE 802.3 αποτελείται από ένα μοναδικό ομοαξονικό καλώδιο στο οποίο συνδέονται αρκετοί υπολογιστές και ονομάζεται καλώδιο Ethernet. Στην αρχική έκδοση του Ethernet κάθε υπολογιστής είχε ένα τσιπ Ethernet συνήθως σε μια προσαρτωμένη κάρτα δικτύου διεπαφής και ένα καλώδιο από την κάρτα συνδέοντας στη μέση ενός καλωδίου Ethernet, μια διάταξη γνωστή ως συνδετήρας σχήματος T όπως φαίνεται στο Σχήμα 3.2. Στην περίπτωση σύνδεσης Ethernet όλες οι μεταφορές δεδομέν-



Σχήμα 3.2: Καλώδιο Ethernet

ων ανάμεσα στον υπολογιστή αποστολέα και στον υπολογιστή παραλήπτη γίνονται σε μορφή πακέτων. Το πακέτο περιέχει συνήθως την διεύθυνση αποστολέα, την διεύθυνση παραλήπτη και τα δεδομένα. Πρέπει να σημειώσουμε ότι οι διεθύνσεις αυτές που βρίσκονται μέσα στο πακέτο είναι φυσικές διευθύνσεις των καρτών δικτύου των υπολογιστών και είναι γνωστές ως διευθύνσεις MAC (Media Access Controller). Κάθε κάρτα δικτύου διεπαφής έχει μια προκαθορισμένη και μοναδική διεύθυνση 48-δυφίων MAC που εγκαθιστάτε κατά τη διάρκεια κατασκευή της. Στο πρωτόκολλο Ethernet, όταν ένας υπολογιστής θέλει να στείλει ένα πακέτο πρέπει πρώτα να ελέγξει το καλώδιο Ethernet για να εξακριβώσει αν κάποιος άλλος υπολογιστής μεταδίδει. Αν όχι, ξεκινάει τη μετάδοση του πακέτου διαφορετικά ο υπολογιστής περιμένει μέχρι την ολοκλήρωση της τρέχουσας μετάδοσης και μετά ξεκινάει τη δική του. Αν δύο υπολογιστές αρχίζουν να μεταδίδουν τότε προκύπτει μια **σύγκρουση** (collision). Κάθε υπολογιστής ανιχνεύει τη σύγκρουση, σταματά και μετά περιμένει για ένα τυχαίο χρονικό διάστημα και ξαναπροσπαθεί τη μετάδοση του πακέτου. Αν συμβεί και δεύτερη σύγκρουση, κάνουν πίσω και ξαναπροσπαθούν να μεταδίδουν με αποτέλεσμα να διπλασιάζεται ο μέσος χρόνος αναμονής σε κάθε διαδοχική σύγκρουση.

Σήμερα το καλώδιο Ethernet έχει πλέον αντικατασταθεί με **διανομείς** (hubs) και **μεταγωγείς** (switches) ενώ συγχρόνως διατηρούν το πρωτόκολλο Ethernet. Ο διανομέας είναι απλά ένα σημείο που συνδέονται όλοι οι υπολογιστές ενώ ένας μεταγωγέας διαθέτει αρκετές θύρες στις οποίες μπορούν να συνδεθούν υπολογιστές, άλλα δίκτυα Ethernet ή άλλοι μεταγωγείς όπως φαίνεται στο Σχήμα 3.3. Οι μεταγωγείς δρομολογούν αυτόματα τα πακέτα στους υπολογιστές προορισμού και επιτρέπουν πολλαπλές ταυτόχρονες συνδέσεις ανάμεσα σε ξεχωριστά ζεύγη υπολογιστών ώστε να αποφευχθεί το πρόβλημα των συγκρούσεων.



Σχήμα 3.3: Μεταγωγέας Ethernet

Η αρχική τεχνολογία Ethernet ήταν φθηνή και εύκολη λύση για την σύνδεση των υπολογιστών αλλά δεν μπορούσε να χρησιμοποιηθεί σαν υποδομή για συστοιχία υπολογιστών διότι η αρχική του ταχύτητα μεταφοράς του Ethernet περιοριζόνταν σε 10 Mbits/sec (που σημαίνει σημαντική καθυστέρηση και χαμηλό εύρος ζώνης). Έτσι, σημαντικές βελτιώσεις του Ethernet έχουν εμφανιστεί όπως το FastEthernet που παρέχει ταχύτητα 100 Mbits/sec και το Gigabit Ethernet που παρέχει 1000 Mbits/sec. Όμως, υπάρχουν στην αγορά μια σειρά από άλλες τεχνολογίες δικτύων υψηλής ταχύτητας όπως είναι cLAN, Infiniband, SCI (Scalable Coherent Interface), ATM (Asynchronous Transfer Mode) και Myrinet με ταχύτητα 2.4 Gbits/sec. Στον Πίνακα 3.1 παρουσιάζονται ορισμένες λεπτομέρειες των παραπάνω τεχνολογιών διασύνδεσης. Η επιλογή της τεχνολογίας δικτύου διασύνδεσης συστοιχίας εξαρτάται από διάφορους παράγοντες όπως η συμβατότητα με το υλικό συστοιχίας και λειτουργικό σύστημα, τιμή και απόδοση. Στον Πίνακα 3.2 φαίνεται η σύγκριση των τεχνολογιών διασύνδεσης. Εδώ πρέπει να σημειώσουμε ότι το VIA (Virtual Interface Architecture) είναι ένα πρότυπο διεπαφής για λογισμικό επικοινωνίας χαμηλής καθυστέρησης που έχει αναπτυχθεί από κατασκευαστές υλικού και ακαδημαικών ιδρυμάτων. Το πρότυπο αυτό έχει υιοθετηθεί για χρήση από τους περισσότερους κατασκευαστές συστοιχίας. Το MPI (Message Passing Interface) παρέχει μεταβίβαση μηνυμάτων μέσω ενός συνόλου βιβλιοθήκης το οποίο μπορούν να το χρησιμοποιούν οι χρήστες για την ανάπτυξη παράλληλων και κατανεμημένων εφαρμογών. Αυτό σημαίνει ότι το MPI παρέχει ένα επίπεδο επικοινωνίας για τους χρήστες εφαρμογών και εξασφαλίζει την μεταφερσιμότητα του κώδικα εφαρμογής σε όλες τις κατανεμημένες και παράλληλες υπολογιστικές πλατφόρμες.

Τέλος, τα δίκτα διασύνδεσης εμπορίου λόγω του χαμηλού κόστους όπως είναι το Fast Ethernet και το Gigabit Ethernet χρησιμοποιούνται σε συστοιχίες υπολογιστών που εκτελούν

Πίνακας 3.1: Παραδείγματα μερικών τεχνολογιών διασύνδεσης δικτύων

Τεχνολογία Διασύνδεσης	Περιγραφή
Gigabit Ethernet	Παρέχει ένα λογικό υψηλό εύρος ζώνης με το χαμηλό κόστος αλλά πάσχει από σχετικά υψηλή καθυστέρηση. Όμως, το χαμηλό κόστος του Gigabit Ethernet είναι ιδανικό για την κατασκευή συστοιχιών υπολογιστών.
Giganet cLAN	Το Giganet cLAN αναπτύχθηκε με στόχο την υποστήριξη αρχιτεκτονικής εικονικής διεπαφής σε υλικό και επίσης υποστηρίζει χαμηλή καθυστέρηση. Όμως παρέχει χαμηλό εύρος ζώνης λιγότερο από 125 Mbytes/s και δεν είναι η κατάλληλη επιλογή για υλοποίηση συστοιχιών υψηλής απόδοσης.
Infiniband	Είναι το τελευταίο βιομηχανικό πρότυπο βασισμένο στις έννοιες VIA (Virtual Interface Architecture) και κυκλοφόρησε μέσα στο 2002. Το Infiniband υποστηρίζει σύνδεση διαφόρων συστατικών συστήματος μέσα σε ένα σύστημα όπως δίκτυα διαεπεξεργαστικά, υποσυστήματα I/O ή μεταγωγείς δικτύου πολύ-πρωτοκόλλου. Το Infiniband είναι ανεξάρτητο από τεχνολογία.
Myrinet	Η πιο διαδεδομένη τεχνολογία που χρησιμοποιείται στις συστοιχίες υπολογιστών υψηλής απόδοσης. Το βασικό πλεονέκτημα είναι ότι λειτουργεί μέσα στο χώρο του χρήστη και παρακάμπτει τις παρεμβάσεις και καθυστερήσεις του λειτουργικού συστήματος.
Scalable Coherent Interface (SCI)	Η πρώτη πρότυπη τεχνολογία διασύνδεσης που εξειδικεύεται για συστοιχίες υπολογιστών. Το SCI ορίζει ένα σχήμα καταλόγου βασισμένου κρυφής μνήμης που αποθηκεύει τις κρυφές μνήμες των συνδεδεμένων υπολογιστών συνεκτικά και υλοποιεί μια εικονική κοινή μνήμη.

Πίνακας 3.2: Σύγκριση μερικών τεχνολογιών διασύνδεσης δικτύων

Κριτήριο	Gigabit Ethernet	Giganet cLAN	Infiniband	Myrinet	SCI
Εύρος ζώνης (MBytes/s)	< 100	< 125	850	230	< 320
Καθυστέρηση (μs)	< 100	7-10	< 7	10	1-2
Διαθεσιμότητα Υλικού	Ναι	Ναι	Ναι	Ναι	Ναι
Υποστήριξη Linux	Ναι	Ναι	Ναι	Ναι	Ναι
Μεγ. αριθμ. κόμβων	1000	1000	> 1000	1000	1000
Υλοπ. πρωτοκόλλου	Υλικό	Firmware	Υλικό	Firmware	Firmware
Υποστήριξη VIA	NT/Linux	NT/Linux	Λογισμικό	Linux	Λογισμικό
Υποστήριξη MPI	MVICH, TCP	Τρ. κατασκ.	MPI/Pro	Τρ. κατασκ.	Τρ. κατασκ.

εφαρμογές οι οποίες δεν επικοινωνούν συχνά και ανταλλάσουν μεγαλύτερα μηνύματα. Αντίθετα, τα δίκτυα υψηλής ταχύτητας χρησιμοποιούνται σε συστοιχίες που εκτελούν εφαρμογές οι οποίες επικοινωνούν πολύ συχνά και ανταλλάσουν πολλά μικρά μηνύματα.

3.7 Λογισμικό Επικοινωνίας / Δικτυακές Υπηρεσίες

Οι ανάγκες επικοινωνίας των κατανεμημένων ή παράλληλων εφαρμογών είναι διάσπαρτες και ποικίλες και κυμαίνονται από τις αξιόπιστες **σημειακές επικοινωνίες** (point-to-point communications) έως τις αναξιόπιστες **επικοινωνίες πολυεκπομπής** (multicast communications). Η υποδομή επικοινωνιών πρέπει να υποστηρίξει πρωτόκολλα που χρησιμοποιούνται για μαζική μεταφορά δεδομένων, δεδομένων ρεύματος, ομαδικές επικοινωνίες και εκείνα που χρησιμοποιούνται για κατανεμημένα αντικείμενα.

Οι υπηρεσίες επικοινωνίας που απασχολούνται παρέχουν τους βασικούς μηχανισμούς που απαιτούνται από την συστοιχία για να μεταφέρει τα δεδομένα διαχείρισης και χρηστών. Επίσης, αυτές οι υπηρεσίες παρέχουν τη συστοιχία με σημαντικές παραμέτρους ποιότητας υπηρεσιών,

όπως η καθυστέρηση, το εύρος ζώνης, η αξιοπιστία και η ανοχή σφαλμάτων. Χαρακτηριστικά, οι δικτυακές υπηρεσίες σχεδιάζονται ως μια ιεραρχική στοίβα πρωτοκόλλων. Σε ένα τέτοιο στρωματοποιημένο σύστημα, κάθε στρώμα πρωτοκόλλου της στοίβας χρησιμοποιεί τις υπηρεσίες που παρέχονται από τα κατώτερα πρωτόκολλα της στοίβας. Ένα κλασικό παράδειγμα μιας τέτοιας δικτυακής αρχιτεκτονικής είναι το σύστημα ISO OSI με τα επτά επίπεδα.

Παραδοσιακά, οι υπηρεσίες λειτουργικών συστημάτων (όπως οι σωληνώσεις (pipes)/υποδοχές (sockets)) έχουν χρησιμοποιηθεί για επικοινωνία μεταξύ των διεργασιών σε συστήματα μεταβίβασης μηνυμάτων. Κατά συνέπεια, η επικοινωνία ανάμεσα στο αποστολέα και του προορισμού περιλαμβάνει ακριβές λειτουργίες, όπως η μεταβίβαση των μηνυμάτων μεταξύ πολλών στρωμάτων, αντιγραφή δεδομένων, έλεγχος προστασίας και αξιόπιστες μετρήσεις επικοινωνίας. Συχνά, οι συστοιχίες με ένα ειδικό δίκτυο/μεταγωγέα όπως το δίκτυο Myrinet που χρησιμοποιεί ελαφριά πρωτόκολλα επικοινωνίας όπως τα Active Messages για γρήγορη επικοινωνία μεταξύ των κόμβων του. Τα πρωτόκολλα αυτά παρακάμπτουν ενδεχομένως το λειτουργικό σύστημα και απομακρύνουν τις κρίσιμες επιβαρύνσεις επικοινωνίας και παρέχουν άμεση πρόσβαση επιπέδου χρήστη στη διεπαφή δικτύου.

Συχνά στις συστοιχίες, οι υπηρεσίες δικτύων θα κατασκευαστούν από μία σχετικά χαμηλού επιπέδου διεπαφή επικοινωνία API (Application Programming Interface) που μπορεί να χρησιμοποιηθεί για να υποστηρίξει μια ποικιλία βιβλιοθηκών επικοινωνίας υψηλού επιπέδου και πρωτόκολλα. Οι μηχανισμοί αυτοί παρέχουν τα μέσα για να υλοποιήσουν ένα ευρύ φάσμα μεθοδολογιών επικοινωνίας συμπεριλαμβανομένου της απομακρυσμένης κλήσης διαδικασία και διεπαφές μεταβίβασης μηνυμάτων όπως MPI και PVM.

3.8 Ενδιάμεσο Λογισμικό Συστοιχίας και Ενιαίας Εικόνας Συστήματος

Η **ενιαία εικόνα συστήματος** (Single System Image - SSI) δίνει την εντύπωση ότι μια συλλογή από διασυνδεδεμένους υπολογιστές παρουσιάζονται σαν μια ενιαία υπολογιστική μηχανή. Με άλλα λόγια, η ενιαία εικόνα συστήματος είναι μια ιδιότητα του συστήματος που κρύβει την ετερογένεια και την κατανεμημένη φύση των διαθέσιμων πόρων και φαίνεται στους χρήστες, στις εφαρμογές και στο δίκτυο σαν ένα μοναδικό υπολογιστικό σύστημα. Η ενιαία εικόνα

συστήματος υποστηρίζεται από ένα επίπεδο ενδιάμεσου λογισμικού που βρίσκεται ανάμεσα στο λειτουργικό σύστημα και στο περιβάλλον του χρήστη. Αυτό το επίπεδο ενδιάμεσου λογισμικού αποτελείται από δύο υποεπίπεδα της υποδομής λογισμικού:

- Υποδομή ενιαίου συστήματος εικόνας (Single System Image infrastructure)
- Υποδομή διαθεσιμότητας συστήματος (System Availability infrastructure)

Η υποδομή ενιαίου συστήματος εικόνας μαζί με τα λειτουργικά συστήματα σε όλους τους κόμβους της συστοιχίας από κοινού παρέχουν μια ενιαία πρόσβαση στους πόρους του συστήματος. Η υποδομή διαθεσιμότητας συστήματος παρέχει στην συστοιχία υπολογιστών υπηρεσίες καθορισμού σημείων ελέγχου, αυτόματη μεταφορά σε εφεδρεία, επανάκτηση από αποτυχία και υποστήριξη ανοχή σφάλματος μεταξύ των κόμβων της συστοιχίας.

Οι στόχοι σχεδιασμού για την ενιαία εικόνα συστήματος της συστοιχίας βασίζονται στην πλήρη διαφάνεια διαχείρισης πόρων, κλιμακωτή απόδοση και διαθεσιμότητας συστήματος για την υποστήριξη εφαρμογών του χρήστη. Το ενδιάμεσο λογισμικό της συστοιχίας παρέχει μια σειρά υπηρεσίες ενιαίας εικόνας συστήματος και διαθεσιμότητας. Οι βασικές υπηρεσίες ενιαίας εικόνας συστήματος της συστοιχίας είναι οι εξής:

- Ενιαίο σημείο εισόδου: Ένας χρήστης μπορεί να συνδεθεί με τη συστοιχία ως ένα ενιαίο σύστημα (όπως telnet beowulf.myinstitute.edu), αντί να συνδεθούμε με τους μεμονωμένους κόμβους της συστοιχίας όπως στην περίπτωση των κατανεμημένων συστημάτων (όπως telnet node1.beowulf.myinstitute.edu).
- Ενιαία ιεραρχία αρχείων: Στην είσοδο στο σύστημα, ο χρήστης βλέπει ένα σύστημα αρχείων ως μια ενιαία ιεραρχία αρχείων και καταλόγων κάτω από το ίδιο κατάλογο ρίζας. Παράδειγμα: xFS.
- Ενιαίο σημείο διαχείρισης και ελέγχου: Η ολόκληρη συστοιχία μπορεί να παρακολουθείται και να ελέγχεται από ένα μοναδικό παράθυρο χρησιμοποιώντας ένα γραφικό εργαλείο, όπως ένας σταθμός εργασίας NT διαχειρίζεται από ένα εργαλείο Task Manager.
- Ενιαία εικονική δικτύωση: Αυτό σημαίνει ότι οποιοσδήποτε κόμβος μπορεί να έχει πρόσβαση σε οποιοδήποτε δίκτυο σύνδεσης σε όλη την περιοχή συστοιχίας ακόμα κι αν το δίκτυο δεν είναι φυσικά συνδεμένος με όλους τους κόμβους της συστοιχίας.

- Ενιαίος χώρος μνήμης: Σχηματίζει μια κοινή μνήμη από όλες τις μνήμες των κόμβων της συστοιχίας.

- Ενιαίο σύστημα διαχείρισης εργασιών: Ένας χρήστης μπορεί να υποβάλει μια εργασία από οποιοδήποτε κόμβο χρησιμοποιώντας ένα διαφανές μηχανισμός υποβολής εργασίας. Οι εργασίες μπορούν να δρομολογηθούν για να εκτελεστούν είτε ομαδικά ή παράλληλα

- Ενιαία διεπαφή χρήστη: Ο χρήστης πρέπει να είναι σε θέση να χρησιμοποιήσει τη συστοιχία μέσω μιας ενιαίας γραφικής διεπαφής. Η διεπαφή πρέπει να έχει τη ίδια αίσθηση και εμφάνιση μιας διεπαφής που είναι διαθέσιμη για τους σταθμούς εργασίας (π.χ., Solaris OpenWin ή Windows NT GUI).

Οι υπηρεσίες διαθεσιμότητας της συστοιχίας υπολογιστών είναι οι εξής:

- Ενιαίος χώρος Ε/Ε: Αυτό επιτρέπει σε οποιοδήποτε κόμβο να εκτελέσει λειτουργία Ε/Ε σε τοπικές ή απομακρυσμένες περιφερειακές ή συσκευές δίσκων. Σε αυτό το σχέδιο, οι δίσκοι είναι συνδεδεμένοι με τους κόμβους συστοιχίας, RAIDs και οι περιφερειακές συσκευές σχηματίζουν ένα ενιαίο χώρο διευθύνσεων.

- Ενιαίος χώρος διεργασιών: Οι διεργασίες έχουν μια μοναδική ταυτότητα id συστοιχίας. Μια διεργασία σε οποιοδήποτε κόμβο μπορεί να δημιουργήσει θυγατρικές διεργασίες σε ίδιο ή διαφορετικό κόμβο (μέσω της κλήσης UNIX fork) ή να επικοινωνήσει με μια οποιαδήποτε άλλη διεργασία (μέσω των σημάτων και σωληνώσεων) σε έναν απομακρυσμένο κόμβο. Η συστοιχία αυτή πρέπει να υποστηρίζει τη καθολική διαχείριση διεργασιών και να επιτρέπει τον έλεγχο των διεργασιών σαν να εκτελούνται σε τοπικούς υπολογιστές.

- Σημείων ελέγχου και μετανάστευσης διεργασιών: Τα σημεία ελέγχου είναι ένας μηχανισμός λογισμικού για να αποθηκεύει περιοδικά την κατάσταση διεργασιών και ενδιάμεσα αποτελέσματα υπολογισμού στην μνήμη ή σε δίσκους. Όταν έναν κόμβο αστοχήσει, οι διεργασίες στον αποτυχημένο κόμβο μπορεί να επαναξεκινήσουν σε άλλο λειτουργικό κόμβο χωρίς απώλεια υπολογισμών. Η μετανάστευση διεργασιών επιτρέπει την δυναμική εξισσορόπηση φορτίου μεταξύ των κόμβων της συστοιχίας.

Τα πλεονεκτήματα του ενδιάμεσου λογισμικού συστοιχίας και της ενιαίας εικόνας συστήματος είναι τα ακόλουθα:

- Παρέχει στο χρήστη να μην γνωρίζει το που θα εκτελείται η εφαρμογή και την τοποθεσία των πόρων του συστήματος.

- Μειώνει τον κίνδυνο των σφαλμάτων του διαχειριστή με αποτέλεσμα να παρέχει στους χρήστες βελτιωμένη αξιοπιστία και μεγαλύτερη διαθεσιμότητα του συστήματος.

- Το σύστημα διαχείρισης και ελέγχου δεν απαιτεί έμπειρους διαχειριστές για την διαχείριση του συστήματος.

- Παρέχει απλοποιημένη διαχείριση συστήματος. Δηλαδή, προσφέρει ενέργειες που επηρεάζουν πολλούς πόρους ομαδικά και αυτές μπορούν να υλοποιηθούν με μια μόνο εντολή ακόμα και όταν οι πόροι είναι διασκορπισμένοι σε διαφορετικά συστήματα.

- Δεν απαιτεί από το χρήστη να γνωρίζει την αρχιτεκτονική του συστήματος για να χρησιμοποιήσει τους υπολογιστές αποδοτικά.

3.9 Επίπεδα Ενιαίας Εικόνας Συστήματος

Υπάρχουν δύο σημαντικά χαρακτηριστικά της ενιαίας εικόνας συστήματος που είναι τα εξής:

- Κάθε ενιαία εικόνα συστήματος έχει ένα όριο ή σύνορα.

- Η υποστήριξη ενιαίας εικόνας συστήματος μπορεί να υπάρχει σε διαφορετικά επίπεδα μέσα σε ένα σύστημα, δηλαδή ένα επίπεδο είναι ικανό να υποστηρίξει στο επόμενο επίπεδο.

Οι υπηρεσίες της ενιαίας εικόνας συστήματος και διαθεσιμότητας μπορούν να υλοποιηθούν σε ένα ή περισσότερα επίπεδα αφαίρεσης της αρχιτεκτονικής συστοιχίας υπολογιστών:

- υλικό

- λειτουργικό σύστημα (ή underware)

- ενδιάμεσο λογισμικό

- εφαρμογές

Μια καλή ενιαία εικόνα συστήματος προκύπτει συνήθως από την συνεργασία των παραπάνω επιπέδων καθώς το χαμηλότερο επίπεδο μπορεί να παρέχει μια απλοποιημένη υλοποίηση του υψηλότερου επιπέδου. Στις παρακάτω υποενότητες περιγράφουμε τα επίπεδα που υποστηρίζουν την ενιαία εικόνα συστήματος.

3.9.1 Επίπεδο Υλικού

Συστήματα όπως το κανάλι μνήμης Digital/Compaq και το υλικό κατανεμημένης κοινής μνήμης (Distributed Shared Memory - DSM) προσφέρουν ενιαία εικόνα συστήματος σε επίπεδο υλικού και επιτρέπουν στο χρήστη να φαίνεται η συστοιχία σαν ένα σύστημα κοινής μνήμης. Το κανάλι μνήμης της Digital σχεδιάστηκε ώστε να προσφέρει μια αξιόπιστη και αποτελεσματική διασύνδεση της συστοιχίας. Ετσι, το κανάλι παρέχει ένα τμήμα της καθολικής εικονικής κοινής μνήμης που απεικονίζει τμήματα της απομακρυσμένης μνήμης σαν τοπική εικονική μνήμη.

3.9.2 Επίπεδο Λειτουργικού Συστήματος

Τα λειτουργικά συστήματα συστοιχίας υποστηρίζουν την αποτελεσματική εκτέλεση των παράλληλων εφαρμογών μέσα σε ένα περιβάλλον μαζί με τις ακολουθιακές εφαρμογές. Ο στόχος είναι μια δεξαμενή πόρων σε μια συστοιχία να παρέχει καλύτερη απόδοση στις ακολουθιακές και παράλληλες εφαρμογές. Για να υλοποιηθεί ο στόχος αυτός, θα πρέπει το λειτουργικό σύστημα να υποστηρίζει την χρονοδρομολόγηση gang των παράλληλων προγραμμάτων, να αναγνωρίζει αδρανείς πόρους στο σύστημα (όπως επεξεργαστές, μνήμη και δίκτυα) και να προσφέρει μια καθολική πρόσβαση στους πόρους αυτούς. Επίσης, πρέπει το λειτουργικό σύστημα να υποστηρίξει την μετανάστευση διεργασιών ώστε να παρέχει δυναμική εξισορρόπηση φορτίου και να παρέχει ταχύτερη διαδιεργασιακή επικοινωνία για το σύστημα και για τις εφαρμογές του χρήστη. Το λειτουργικό σύστημα πρέπει να εξασφαλίζει ότι τα παραπάνω χαρακτηριστικά θα είναι διαθέσιμα στους χρήστες χωρίς την ανάγκη για επιπλέον κλήσεις συστήματος ή εντολές. Λειτουργικά συστήματα που υποστηρίζουν την ενιαία εικόνα συστήματος είναι μεταξύ άλλων SCO UnixWare, Sun Solaris-MC και MOSIX.

Άλλος τρόπος για το λειτουργικό σύστημα να υποστηρίξει την ενιαία εικόνα συστήματος είναι η κατασκευή ενός καινούργιου επιπέδου λειτουργικού συστήματος πάνω από τα υπάρχοντα λειτουργικά συστήματα και το οποίο να εκτελεί καθολική διανομή πόρων. Η λύση αυτή κάνει

το σύστημα να είναι μεταφέρσιμο και συγχρόνως μειώνει τον χρόνο ανάπτυξης. Την παραπάνω προσέγγιση ακολούθησε η Berkeley GLUnix.

3.9.3 Επίπεδο Ενδιάμεσου Λογισμικού

Ενδιάμεσο λογισμικό είναι ένα επίπεδο που βρίσκεται ανάμεσα στο λειτουργικό σύστημα και τις εφαρμογές και είναι επίσης ένας από τους κοινούς μηχανισμούς που χρησιμοποιείται για να υλοποιήσει την ενιαία εικόνα συστήματος στις συστοιχίες υπολογιστών. Σε αυτό το επίπεδο περιλαμβάνουν τα συστήματα αρχείων συστοιχίας, τα συστήματα διαχείρισης και χρονοδρομολόγησης πόρων (όπως Gondor, Loadleveler, Load Share Facility, Open Portable Batch System και Sun Grid Engine) και εικονική μηχανή Java (JVM) συστοιχίας. Τα συστήματα αρχείων συστοιχίας παρουσιάζουν τους δίσκους που είναι τοποθετημένοι στους κόμβους συστοιχίας σαν ένα ενιαίο μεγάλο χώρο αποθήκευσης και εξασφαλίζουν ότι κάθε κόμβος της συστοιχίας έχει την ίδια πλευρά δεδομένων. Επίσης, τα συστήματα χρονοδρομολόγησης εργασιών/πόρων διαχειρίζονται τους πόρους και δρομολογεί τις δραστηριότητες του συστήματος και την εκτέλεση των εφαρμογών ενώ συγχρόνως προσφέρει διαφανείς υπηρεσίες υψηλής διαθεσιμότητας. Τέλος, η εικονική μηχανή Java (JVM) για συστοιχία επιτρέπει την εκτέλεση εφαρμογών Java (τα οποία βασίζονται στο μοντέλο νημάτων) σε συστοιχίες χωρίς τροποποιήσεις.

3.9.4 Επίπεδο Εφαρμογών

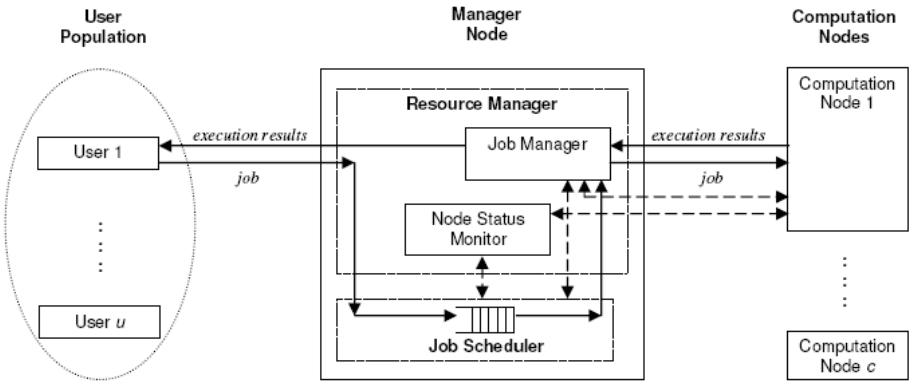
Οι εφαρμογές μπορούν να υποστηρίζουν την ενιαία εικόνα συστήματος. Το επίπεδο εφαρμογών είναι το υψηλότερο και σημαντικό επειδή το επίπεδο αυτό παρουσιάζει το τι βλέπουν οι χρήστες. Σε αυτό το επίπεδο, τα πολλαπλά τμήματα μιας εφαρμογής παρουσιάζονται στο χρήστη/διαχειριστή σαν μια ενιαία εφαρμογή. Για παράδειγμα, ένα εργαλείο διαχείρισης συστοιχίας προσφέρει ένα σημείο διαχείρισης και ελέγχου υπηρεσιών ενιαίας εικόνας συστήματος. Οι εφαρμογές αυτές μπορούν να κατασκευαστούν σαν γραφικά εργαλεία που προσφέρουν ένα παράθυρο για την παρακολούθηση και ελέγχου της συστοιχίας συνολικά ή ξεχωριστούς κόμβους (υπολογιστές) ή συγκεκριμένα τμήματα του συστήματος. Ένα παράδειγμα εργαλείου διαχείρισης συστοιχίας είναι το PARMON.

3.10 Ενδιάμεσο Λογισμικό Συστήματος Διαχείρισης Πόρων

Το σύστημα διαχείρισης πόρων συστοιχίας δρα σαν ενδιάμεσο λογισμικό συστοιχίας που υλοποιεί την ενιαία εικόνα συστήματος για μια συστοιχία υπολογιστών. Το σύστημα αυτό δίνει την δυνατότητα στους χρήστες να εκτελούν εργασίες στην συστοιχία χωρίς την ανάγκη να γνωρίζουν την πολυπλοκότητα της αρχιτεκτονικής συστοιχίας. Συγκεκριμένα, το σύστημα διαχείρισης πόρων είναι υπεύθυνο για την διανομή εφαρμογών ανάμεσα στους υπολογιστές της συστοιχίας για την μεγιστοποίηση της ρυθμοαπόδοσης και την αποτελεσματική χρήση των διαθέσιμων πόρων. Το λογισμικό που εκτελεί το σύστημα διαχείρισης πόρων αποτελείται από δύο συστατικά: ένας **διαχειριστής πόρων** (resource manager) και ένας **χρονοδρομολογητής πόρων** (resource scheduler). Ο διαχειριστής πόρων αφορά για τα προβλήματα, όπως η εύρεση και διανομή των υπολογιστικών πόρων, της αυθεντικότητας, καθώς επίσης την δημιουργία και μετανάστευση διεργασιών. Ο χρονοδρομολογητής πόρων αφορά εργασίες όπως εφαρμογές ουρών καθώς επίσης η εύρεση και ανάθεση πόρων.

Οι λόγοι που χρησιμοποιούμε το σύστημα διαχείρισης πόρων περιλαμβάνουν τα εξής: εξισορρόπηση φορτίου, χρήση αδρανών κύκλων ΚΜΕ, παροχή συστημάτων ανοχή βλαβών και διαχειρίσιμη πρόσβαση σε ισχυρά συστήματα. Αλλά ο κύριος λόγος για την ύπαρξη τους είναι η δυνατότητα να παρέχει μια αυξανόμενη και αξιόπιστη ρυθμοαπόδοση των εφαρμογών χρηστών στα συστήματα που διαχειρίζονται.

Στο Σχήμα 3.4 παρουσιάζεται μια γενική αρχιτεκτονική του συστήματος διαχείρισης πόρων συστοιχίας. Ένα σύστημα διαχείρισης πόρων διαχειρίζεται μια συλλογή πόρων όπως επεξεργαστές και δίσκους της συστοιχίας. Το σύστημα αποθηκεύει πληροφορίες κατάστασης των πόρων ώστε να είναι γνωστές ποιοι πόροι είναι διαθέσιμοι και το σύστημα να αναθέσει εργασίες στους διαθέσιμους υπολογιστές. Το σύστημα διαχείρισης πόρων χρησιμοποιεί ουρές εργασιών που αποθηκεύουν τις υποβληθείσες εργασίες μέχρι ότου υπάρχουν διαθέσιμοι πόροι για να εκτελέσουν τις εργασίες. Όταν οι πόροι είναι διαθέσιμοι, το σύστημα διαχείρισης πόρων καλεί έναν χρονοδρομολογητή εργασιών για να επιλέξει από τις ουρές ποιες εργασίες να εκτελέσουν. Έπειτα το σύστημα διαχείρισης πόρων διαχειρίζεται την εκτέλεση διεργασιών και επιστρέφει τα αποτελέσματα στους χρήστες με την ολοκλήρωση εργασίας.



Σχήμα 3.4: Αρχιτεκτονική συστήματος διαχείρισης πόρων συστοιχίας

Τα συστήματα διαχείρισης πόρων παρέχουν υπηρεσίες ενδιάμεσου λογισμικού στους χρήστες. Οι υπηρεσίες που παρέχονται από ένα σύστημα διαχείρισης πόρων είναι οι εξής:

- Μετανάστευση διεργασιών - Αυτό είναι όπου μια διεργασία μπορεί να ανασταλεί, μετακινηθεί και επαναξεκινήσει σε έναν άλλο υπολογιστή. Γενικά, η μετανάστευση διεργασιών εμφανίζεται λόγω του ενός από τους δύο λόγους: ένας υπολογιστικός πόρος έχει φορτωθεί βαριά και υπάρχουν άλλοι ελεύθεροι πόροι οι οποίοι μπορούν να χρησιμοποιηθούν ή συνδυασμό με την διαδικασία ελαχιστοποίηση επίδρασης των χρηστών όπως θα αναφερθούμε παρακάτω.

- Σημεία ελέγχου - Αυτό είναι όπου ένα στιγμιότυπο της κατάστασης εκτέλεσης του προγράμματος να αποθηκεύεται και να μπορεί να χρησιμοποιηθεί για επανεκκίνηση του προγράμματος από το ίδιο σημείο σε έναν πιο τελευταίο σταάδιο εάν είναι απαραίτητο. Τα σημεία ελέγχου θεωρείται γενικά ως μέσο παροχής αξιοπιστίας. Όταν κάποιο μέρος του συστήματος διαχείρισης πόρων αστοχήσει, τα προγράμματα που εκτελούσαν σε αυτό μπορούν να επανεκκινηθούν από κάποιο ενδιάμεσο σημείο στην εκτέλεση τους παρά επανεκκίνση από την αρχή.

- Σάρωση αδρανών κύκλων - Γενικά αναγνωρίζεται ότι μεταξύ 70% - 90% του χρόνου οι περισσότεροι σταθμοί εργασίας είναι αδρανείς. Τα συστήματα διαχείρισης πόρων μπορούν να οργανωθούν για να χρησιμοποιήσουν τους αδρανείς κύκλους CPU. Για παράδειγμα, οι

εργασίες μπορούν να υποβληθούν στους σταθμούς εργασίας κατά τη διάρκεια της νύχτας ή στα Σαββατοκύριακα. Με αυτό τον τρόπο, οι διαλογικοί χρήστες δεν επηρεάζονται από τις εξωτερικές εργασίες και εκμεταλεύουν τους αδρανείς κύκλους CPU.

- Ανοχή βλαβών - Με την παρακολούθηση των εργασιών και των πόρων, ένα σύστημα διαχείρισης πόρων μπορεί να παρέχει διάφορα επίπεδα ανοχής βλαβών. Στην απλούστερη μορφή του, η υποστήριξη ανοχή βλαβών μπορεί να σημαίνει ότι μια αποτυχημένη εργασία μπορεί να επανεκκινηθεί ή να ξαναεκτελέσει, εγγυώντας ότι η εργασία θα ολοκληρωθεί.

- Ελαχιστοποίηση επίδρασης χρηστών - Η εκτέλεση μιας εργασίας στους δημόσιους σταθμούς εργασίας μπορεί να έχει μεγάλη επίδραση στη χρησιμότητα των σταθμών εργασίας από τους διαλογικούς χρήστες. Μερικά συστήματα διαχείρισης πόρων προσπαθούν να ελαχιστοποιήσουν την επίδραση μιας εκτελούσας εργασίας σε διαλογικούς χρήστες είτε μειώνοντας την προτεραιότητα της τοπικής χρονοδρομολόγησης μιας εργασίας είτε αναστέλοντας την εργασία. Οι υπο αναστολή εργασίες μπορούν να επανεκκινηθούν αργότερα ή να μεταναστεύσουν σε άλλους πόρους συστημάτων.

- Εξισορρόπηση φορτίου - Οι εργασίες μπορούν να κατανεμηθούν μεταξύ όλων των διαθέσιμων υπολογιστικών πλατφορμών σε μια συγκεκριμένη οργάνωση. Αυτό θα επιτρέψει την αποτελεσματική χρήση όλων των πόρων, παρά μερικοί οι οποίοι να είναι οι μόνοι πόροι που οι χρήστες γνωρίζουν. Η μετανάστευση διεργασιών μπορεί επίσης να είναι της στρατηγικής εξισορρόπησης φορτίου όπου η στρατηγική αυτή ίσως ωφεληθεί με το να μετακινηθούν διεργασίες από ένα υπερφορτωμένο σύστημα σε ελαφριά φορτωμένο σύστημα.

- Πολλαπλές ουρές εφαρμογής - Οι ουρές εργασιών μπορούν να εγκατασταθούν για να βοηθήσουν στην διαχείριση των πόρων σε μια ιδιαίτερη οργάνωση. Κάθε ουρά μπορεί να παραμετροποιηθεί με συγκεκριμένες ιδιότητες. Για παράδειγμα, ορισμένοι χρήστες μικρών εργασιών έχουν προτεραιότητα να εκτελέσουν πριν τις μεγάλες εργασίες. Οι ουρές εργασιών μπορούν επίσης να εγκατασταθούν για να διαχειριστούν την χρήση ειδικών πόρων, όπως μια παράλληλη υπολογιστική πλατφόρμα ή ένα σταθμό εργασίας γραφικών υψηλής απόδοσης. Οι ουρές σε ένα σύστημα διαχείρισης πόρων μπορούν να είναι διαφανείς στους χρήστες, δηλαδή οι εργασίες διανέμονται στις ουρές μέσω λέξων κλειδιών όταν η

Πίνακας 3.3: Παραδείγματα συστημάτων διαχείρισης πόρων συστοιχίας

Σύστημα	Περιγραφή
Condor	Είναι σε θέση να ανιχνεύσει και να εκτελέσει εργασίες σε αδρανείς μη-αποκλειστικούς υπολογιστές
Loadleveler	Διαχειρίζεται πόρους και εργασίες για συστοιχίες IBM
Load Share Facility (LSF)	Υιοθετεί μια στρωματοποιημένη αρχιτεκτονική που υποστηρίζει πολλές ευκολίες επέκτασης
Open Portable Batch System (OpenPBS)	Υποστηρίζει πολλές πολιτικές χρονοδρομολόγησης βασισμένες σε μια εκτεταμένη αρχιτεκτονική χρονοδρολογητή
Sun Grid Engine (SGE)	Η επιχειρησιακή έκδοση υποστηρίζει χρονοδρομολόγηση εργασιών πάνω σε πολλαπλές συστοιχίες υπολογιστών ενός οργανισμού
Libra	Υποστηρίζει διανομή πόρων βασισμένο στις αρχές υπολογιστικής οικονομικής και τις απαιτήσεις ποιότητας των χρηστών

εργασία υποβάλλεται.

Για την αποτελεσματική διαχείριση πόρων στις συστοιχίες έχουν αναπτυχθεί παρά πολλά συστήματα διαχείρισης συστοιχίας και χρονοδρολογητές. Στον Πίνακα 3.3 παρουσιάζονται μερικά δημοφιλή συστήματα διαχείρισης πόρων συστοιχίας.

3.11 Περιβάλλοντα Προγραμματισμού και Εργαλεία

Η συστοιχία υπολογιστών διαθέτει περιβάλλοντα προγραμματισμού και εργαλεία τα οποία διεκολύνουν στους προγραμματιστές στην εύκολη ανάπτυξη παράλληλων εφαρμογών. Σε αυτή την ενότητα συζητάμε ορισμένα δημοφιλή εργαλεία προγραμματισμού που χρησιμοποιούνται σε συστοιχίες υπολογιστών.

3.11.1 Περιβάλλοντα Μεταβίβασης Μηνυμάτων

Ο προγραμματισμός σε συστοιχίες υπολογιστών απαιτεί ειδικό λογισμικό συνήθως **βιβλιοθήκες μεταβίβασης μηνυμάτων** (message passing libraries) που μας επιτρέπουν να γράφουμε παράλληλα προγράμματα για το χειρισμό της επικοινωνίας και του συγχρονισμού των διεργασιών. Τα συστήματα μεταβίβασης μηνυμάτων έχουν δύο ή περισσότερες διεργασίες οι οποίες εκτελούνται ανεξάρτητα η μια από την άλλη και η επικοινωνία τους γίνεται μέσω ρητής μεταβίβασης μηνυμάτων. Οι βιβλιοθήκες μεταβίβασης μηνυμάτων χρησιμοποιούν πρωτόκολλα επικοινωνίας για ανταλλαγή μηνυμάτων ανάμεσα στις διεργασίες. Τα πρωτόκολλα επικοινωνίας κρύβουν πολλές από τις λειτουργίες μεταβίβασης μηνυμάτων χαμηλού επιπέδου από το προγραμματιστή. Στην παρούσα φάση υπάρχουν δύο δημοφιλείς βιβλιοθήκες μεταβίβασης μηνυμάτων για επιστημονικές και μηχανικές εφαρμογές που είναι το PVM (Parallel Virtual Machine) [3] από το Oak Ridge National Laboratory και το MPI (Message Passing Interface) [8, 7] που ορίστηκε από το MPI Forum.

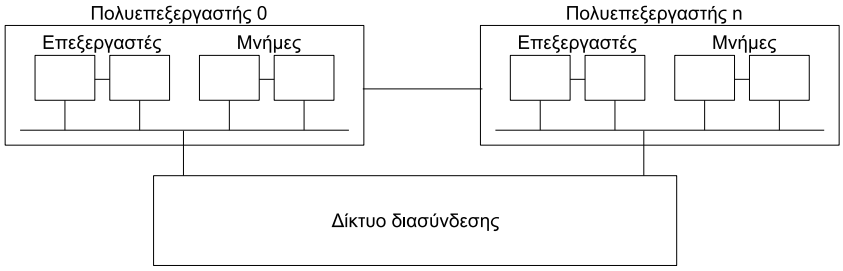
Το πιο διαδεδομένο λογισμικό επικοινωνίας για προγραμματισμό σε συστοιχίες υπολογιστών είναι το MPI. Το MPI είναι η πρότυπη βιβλιοθήκη για μεταβίβαση μηνυμάτων που μπορεί να χρησιμοποιηθεί για την ανάπτυξη μεταφέρσιμων προγραμμάτων μεταβίβασης μηνυμάτων χρησιμοποιώντας τις ακολουθιακές γλώσσες προγραμματισμού C ή Fortran. Για την γλώσσα προγραμματισμού Java υποστηρίζεται από άλλο πρότυπο που λέγεται MPJ. Το πρότυπο MPI ορίζει συντακτικά και σημασιολογικά ένα σύνολο από συναρτήσεις βιβλιοθήκης που είναι χρήσιμες για την σύνταξη προγραμμάτων μεταβίβασης μηνυμάτων. Το MPI αναπτύχθηκε από μια κοινότητα ερευνητών οι οποίοι προέρχονται από πανεπιστήμια και βιομηχανίες και υποστηρίζεται από όλους τους προμηθευτές υλικού. Επίσης, το MPI είναι διαθέσιμο σε περισσότερα παράλληλα υπολογιστικά συστήματα (από συμμετρικούς πολυεπεξεργαστές μέχρι και συστοιχίες υπολογιστών).

Η αρχική έκδοση του MPI που σήμερα λέγεται MPI-1, επεκτάθηκε με την έκδοση MPI-2. Το MPI-1 προσφέρει συναρτήσεις υψηλού επιπέδου για την διαχείριση του περιβάλλοντος μηνυμάτων, συναρτήσεις υψηλού επιπέδου για **σημειακή επικοινωνία** (point-to-point communication) ανάμεσα σε δύο διεργασίες (όπως αποστολή και λήψη μηνυμάτων), συναρτήσεις για **συλλογική επικοινωνία** (collective communication) μεταξύ των διεργασιών μιας ομάδας, συναρτήσεις για παράγωγοι τύποι δεδομένων και τέλος συναρτήσεις για εικονικές τοπολογίες. Το MPI-2 προσθέτει ορισμένα χαρακτηριστικά όπως δυναμικές διεργασίες, προσπέλαση απο-

μακρυσμένης μνήμης, συλλογική επικοινωνία χωρίς ανασταλτική, κλιμακωτή υποστήριξη E/E και πολλές νέες δυνατότητες οι οποίες ξεπερνούν τα πλαίσια αυτού του βιβλίου. Στο κεφάλαιο 9 θα παρουσιάσουμε αναλυτικά το προγραμματισμό μεταβίβασης μηνυμάτων σε συστοιχίες υπολογιστών χρησιμοποιώντας την βιβλιοθήκη MPJ.

3.12 Συστοιχία από SMPs

Με τις τελευταίες εξελίξεις στις τεχνολογίες υλικού στην περιοχή των επεξεργαστών, μνήμης και κάρτες δικτύων έχει επιτρέψει την δημιουργία μιας μικρής συστοιχίας από συμμετρικούς πολυεπεξεργαστές SMPs (8-10 πολυεπεξεργαστές) σε χαμηλό κόστος όπως φαίνεται στο Σχήμα 3.5. Οι πολλαπλοί πολυεπεξεργαστές που έχουν κάρτες δικτύου συνδέονται μεταξύ τους με



Σχήμα 3.5: Συστοιχία από συμμετρικούς πολυεπεξεργαστές

δίκτυα υψηλής ταχύτητας.

Η συστοιχία αυτή παρουσιάζει πολλά πλεονεκτήματα. Πρώτα παρέχει αρκετή υπολογιστική ισχύ στις εφαρμογές με μικρό αριθμό επεξεργαστών. Επίσης, εύκολα χρησιμοποιούνται και παρουσιάζουν καλό προγραμματιστικό ενδιαφέρον. Έτσι, ο προγραμματισμός σε αυτές τις συστοιχίες αποτελείται από δύο επίπεδα, προγραμματισμό σε κάθε πολυεπεξεργαστή μέσω του μοντέλου κοινής μνήμης και προγραμματισμό ανάμεσα στους πολυεπεξεργαστές μέσω του μοντέλου πέρασμα μηνυμάτων. Επιπλέον, οι συστοιχίες κατασκευάζονται εύκολα με μικρή προσπάθεια και αυτό έχει σαν αποτέλεσμα την ευκολότερη διαχείριση του συστήματος και καλύτερη υποστήριξη για την τοπικότητα δεδομένων σε κάθε πολυεπεξεργαστή. Για παράδειγμα, αν θέλουμε να έχουμε μια συστοιχία 32 υπολογιστών μπορεί να κατασκευαστεί είτε με 4 ή με 8 πολυεπεξεργαστές αντί 32 ξεχωριστούς υπολογιστές.

Όμως, αυτή η τάση για συστοιχίες από πολυεπεξεργαστές ανοίγει νέες ανάγκες σε ότι αφορά στις τεχνολογίες δικτύωσης. Για παράδειγμα, μια κάρτα δικτύου ίσως να μην είναι επαρκή για ένα σύστημα πολυεπεξεργαστή με 8 επεξεργαστές και υπάρχει ανάγκη για πολλαπλές συσκευές δικτύου.

Κεφάλαιο 4

Κατανεμημένα Συστήματα

Αυτό το κεφάλαιο ξεκινάμε με μια περιγραφή της αρχιτεκτονικής κατανεμημένων συστημάτων και τους στόχους της. Στην συνέχεια παρουσιάζουμε τα συστατικά υλικού που αποτελούν το κατανεμημένο σύστημα όπως το υλικό διαδικτύωσης. Επίσης, παρουσιάζουμε συνοπτικά τις υπηρεσίες ενδιάμεσου λογισμικού υποδομής του κατανεμημένου συστήματος και πρωτόκολλα επικοινωνίας υψηλότερου επιπέδου. Τέλος, αναφέρουμε υπηρεσίες του ενδιάμεσου λογισμικού χρήστη για την ανάπτυξη κατανεμημένων εφαρμογών υψηλού επιπέδου.

4.1 Αρχιτεκτονική Κατανεμημένων Συστημάτων

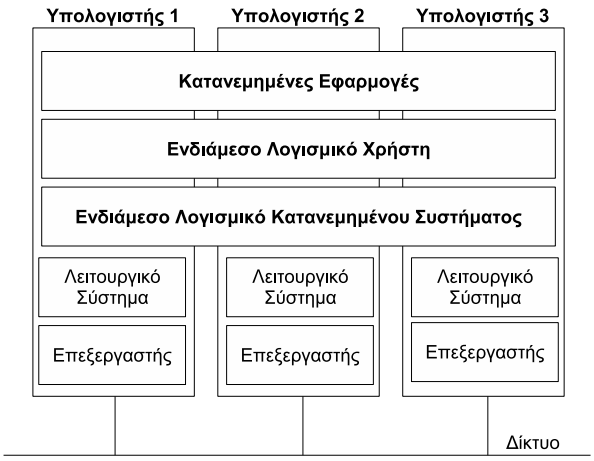
Με την ραγδαία εξέλιξη της τεχνολογίας των μικροεπεξεργαστών και των δικτύων υπάρχει σήμερα η δυνατότητα εύκολης και οικονομικής διασύνδεσης των προσωπικών υπολογιστών μέσω δικτύων ευρείας περιοχής. Αυτό έχει οδηγήσει στην σχεδίαση υπολογιστικών συστημάτων πολλών επεξεργαστών, τα οποία ονομάζονται **κατανεμημένα συστήματα** (distributed systems). Στην πράξη ο όρος κατανεμημένο σύστημα είναι πιο εξειδικευμένος: ονομάζουμε κατανεμημένο σύστημα μια συλλογή από αυτόνομους υπολογιστές που είναι διασυνδεδεμένοι μέσω ενός δικτύου ευρείας περιοχής (όπως το Διαδίκτυο) και επίσης το σύστημα είναι εξοπλισμένο με ένα κατανεμημένο λογισμικό συστήματος ώστε να παρουσιάζεται στους χρήστες σαν ένα ολοκληρωμένο υπολογιστή.

Σύμφωνα με τον παραπάνω ορισμό οι υπολογιστές του συστήματος πρέπει να είναι αυτόνομοι και το λογισμικό πρέπει να είναι οργανωμένο σε τέτοιο τρόπο ώστε να δώσει την εντύπωση στους

χρήστες ότι έχουν να αντιμετωπίζουν ένα ενιαίο σύστημα. Πέρα από αυτά τα δύο βασικά σημεία, τα κατανεμημένα συστήματα έχουν τα παρακάτω σημαντικά χαρακτηριστικά ότι:

- οι χρήστες και οι εφαρμογές σε ένα κατανεμημένο σύστημα πρέπει να συνεργάζονται και να επικοινωνούν με ομοιόμορφο τρόπο ανεξάρτητα που βρίσκονται και πως επικοινωνούν
- επεκτείνονται εύκολα
- πρέπει να είναι μόνιμα διαθέσιμα
- πρέπει να κρύβουν τις διαφορές στο υλικό των υπολογιστών και την παρασκηνιακή επικοινωνία τους από τους χρήστες

Έτσι, προκειμένου τα κατανεμημένα συστήματα να υποστηρίξουν τους ετερογενείς υπολογιστές και τα δίκτυα πρέπει το λογισμικό να διαιρεθεί σε δύο επίπεδα ώστε να παρέχει μια ομοιόμορφη αναπαράσταση του συνολικού συστήματος. Στο υψηλότερο επίπεδο είναι οι εφαρμογές και στο χαμηλότερο επίπεδο είναι το ενδιάμεσο λογισμικό το οποίο αλληλεπιδρά με το υποκείμενο δίκτυο και τα συστήματα υπολογιστών όπως φαίνεται στο Σχήμα 4.1.



Σχήμα 4.1: Αρχιτεκτονική κατανεμημένου συστήματος

Τα βασικά συστατικά ενός κατανεμημένου συστήματος είναι τα εξής:

1. Πολλαπλοί ανεξάρτητοι υπολογιστές (όπως PCs, σταθμοί εργασίας κλπ)
2. Σύγχρονα λειτουργικά συστήματα σε κάθε υπολογιστή

3. Δίκτυο για την διασύνδεση των υπολογιστών (όπως το Διαδίκτυο (Internet))

4. Ενδιάμεσο λογισμικό υποδομής του κατανεμημένου συστήματος

5. Ενδιάμεσο λογισμικό χρήστη

6. Κατανεμημένες εφαρμογές.

Το ενδιάμεσο λογισμικό παρέχει υπηρεσίες υποδομής που στόχο να κρύψουν την ετερογένεια του κατανεμημένου συστήματος και να παρέχουν μια ενιαία εικόνα συστήματος στους χρήστες και στις εφαρμογές.

Το ενδιάμεσο λογισμικό χρήστη παρέχει υπηρεσίες και εργαλεία προγραμματισμού στους προγραμματιστές για την εύκολη ανάπτυξη κατανεμημένων εφαρμογών. Τέτοια εργαλεία είναι οι **υποδοχές** (sockets), η **απομακρυσμένη κλήση διαδικασίας** (remote procedure call), η **απομακρυσμένη κλήση μεθόδων** (remote method invocation) και οι **υπηρεσίες Ιστού** (web services).

Για κάθε ένα από τα παραπάνω συστατικά υλικού και λογισμικού θα αναφερθούμε στις επόμενες ενότητες του κεφαλαίου αυτού. Για τα συστατικά υπολογιστές και λειτουργικά συστήματα δεν θα αναφερθούμε ξανά αφού είναι παρόμοια με εκείνα τα συσταστικά της συστοιχίας υπολογιστών.

Τέλος, ένα προφανές παράδειγμα κατανεμημένου συστήματος είναι ο Παγκόσμιος Ιστός που χρησιμοποιείται από τους περισσότερους χρήστες για διάφορους σκοπούς όπως ανάγνωση ιστοσελίδων, ηλεκτρονικό ταχυδρομείο ή μεταφόρτωση αρχείων. Με την χρήση ενός **περιηγητή Ιστού** (web browser), ο χρήστης μπορεί να έχει πρόσβαση στις πληροφορίες που είναι αποθηκευμένες στους διάφορους **διακομιστές Ιστού** (web servers) παγκοσμίως. Με αυτό το τρόπο ο περιηγητής κρύβει την παρασκηνιακή επικοινωνία (όπως η αποστολή αιτήσεων και επιστροφή ιστοσελίδων από κάποιο απομακρυσμένο διακομιστή) και δίνει την εντύπωση ότι οι πληροφορίες είναι αποθηκευμένες στον υπολογιστή του χρήστη. Συνεπώς, ο Ιστός είναι ένα τεράστιο κατανεμημένο σύστημα που εμφανίζεται στον χρήστη σαν ένα ενιαίος πόρος και ο οποίος είναι διαθέσιμος με ένα μόνο κλικ του ποντικιού.

4.2 Στόχοι Κατανεμημένων Συστημάτων

Σε αυτή την ενότητα παρουσιάζουμε τους βασικούς στόχους που πρέπει να υπάρχουν στην περίπτωση που σχεδιάσουμε ένα κατανεμημένο σύστημα.

- **Διαφάνεια** (transparency). Ολόκληρο το σύστημα πρέπει να εμφανίζεται σαν μια ενιαία εικόνα και οι αλληλεπιδράσεις ανάμεσα στους πόρους (π.χ. υπολογιστές, προγράμματα, αρχεία κλπ) του συστήματος πρέπει να είναι κρυμμένες από τον χρήστη. Σε αυτό το στόχο υπάρχουν διάφορα είδη διαφάνειας τα οποία αναφέρονται παρακάτω:
  - **Διαφάνεια πρόσβασης** (access transparency). Οι πόροι για πρόσβαση δεν πρέπει να διαφέρουν ανάλογα με το αν βρίσκονται σε τοπικό ή απομακρυσμένο υπολογιστή.
  - **Διαφάνεια τοποθεσίας** (location transparency). Οι χρήστες μπορούν να έχουν πρόσβαση τους πόρους τους συστήματος χωρίς να γνωρίζουν σε ποια τοποθεσία στο δίκτυο βρίσκεται. Αυτό σημαίνει ότι τα ονόματα των πόρων θα πρέπει να είναι λογικά και να μην κωδικοποιούν για την θέση των πόρων.
  - **Διαφάνεια συγχρονικότητας** (concurrency transparency). Αυτό σημαίνει οτι όταν ένα πλήθος προγραμμάτων που εκτελούνται συγχρονικά προσπελαύνουν ένα πόρο δεν πρέπει να αλληλοεμποδίζονται και δεν πρέπει να δημιουργούνται προβλήματα συνέπειας των δεδομένων. Αυτό είναι ένα κύριο πρόβλημα που ταλαιπωρεί τους προγραμματιστές για πολλά χρόνια και δεν περιορίζεται στα κατανεμημένα συστήματα.
  - **Διαφάνεια αστοχίας** (failure transparency). Αυτό σημαίνει οτι όταν συμβαίνει μια αστοχία, είτε υλισμικού ή λογισμικού, αυτή δεν πρέπει να επηρρεάζει ολόκληρο το σύστημα και πρέπει να απομονώνονται.
  - **Διαφάνεια απόδοσης** (performance transparency). Αυτό σημαίνει οτι τα προγράμματα που προσπελαύνουν ένα πόρο πρέπει να έχουν ένα σταθερό προαποφασισμένο κάτω όριο απόδοσης, όταν το σύστημα λειτουργεί σε αυτό που ονομάζουμε τυπικό φόρτο του συστήματος.
  - **Διαφάνεια πανομοιότυπης αντιγραφής** (replication transparency). Μια κοινή τεχνική για την διασφάλιση αξιοπιστίας και αύξηση της απόδοσης του συστή-

ματος είναι αυτή της πανομοιότυπης αντιγραφής δεδομένων. Εδώ διατηρούνται πιστά ή μερικώς πιστά αντίγραφα πόρων σε διάφορα σημεία του κατανεμημένου συστήματος χωρίς το πλήθος των αντιγράφων να είναι ορατά στους χρήστες. Έτσι ώστε, για παράδειγμα, ένας χρήστης έχει στη διάθεσή του ένα αντίγραφο του πόρου αποθηκευμένο είτε στο δικό του υπολογιστή ή σε τοπικό δίκτυο συνδεμένο με τον υπολογιστή του, αντί να πρέπει να έχει πρόσβαση στο πόρο μέσω μιας αργής δικτυακής σύνδεσης. Η διαφάνεια πανομοιότυπης αντιγραφής διασφαλίζει οτι δεν απαιτείται επί πλέον κώδικας στο επίπεδο της εφαρμογής για τη διαχείριση των πανομοιότυπων αντιγράφων.

  - **Διαφάνεια μετανάστευσης** (migration transparency). Αυτό σημαίνει οτι όταν ένας πόρος μετακινείται από έναν υπολογιστή σε άλλο, το πρόγραμμα που το χρησιμοποιεί δεν χρειάζεται να τροποποιηθεί.

- **Ετερογένεια** (heterogeneity). Σε ένα κατανεμημένο σύστημα μπορεί να περιέχει πόρους που είναι διαφορετικοί μεταξύ τους και πρέπει να είναι σε θέση να συνεργάζονται. Οι πόροι μπορεί είναι διαφορετικές αρχιτεκτονικές υλικού, διαφορετικά λειτουργικά συστήματα, διαφορετικά πρωτόκολλα επικοινωνίας, διαφορετικές γλώσσες προγραμματισμού και διαφορετικές μορφές αναπαραστάσεις.

- **Ανοιχτή υλοποίηση** (openness). Το σύστημα πρέπει να βασίζεται σε ανοιχτά και δημόσια πρότυπα ώστε να ώστε να μπορούν να συνεργάζονται υλοποιήσεις από διαφορετικούς κατασκευαστές οι οποίοι εκτελούνται σε διαφορετικές πλατφόρμες. η προσθήκη επιπλέον συστατικών.

- **Κλιμάκωση** (scalability). Το σύστημα πρέπει να λειτουργεί αποτελεσματικά με την προσθήκη επιπλέον χρηστών και πόρων που να μην επηρεάζει την απόδοση του συστήματος.

- **Ασφάλεια** (security). Η πρόσβαση των πόρων του συστήματος πρέπει να είναι ασφαλή ώστε να εξασφαλίσουμε ότι οι πιστοποιημένοι χρήστες έχουν την δυνατότητα να εκτελούν λειτουργίες στους πόρους αυτούς.

4.3 Υλικό Διαδικτύωσης

Το δίκτυο ευρείας περιοχής που χρησιμοποιούν τα κατανεμημένα συστήματα είναι το Διαδίκτυο. Το Διαδίκτυο αποτελείται από δυο κατηγορίες υπολογιστών τους υπολογιστές υπηρεσίας και τους **δρομολογητές** (routers) ή **πύλες** (gateways). Οι υπολογιστές υπηρεσίας είναι προσωπικοί υπολογιστές, φορητοί υπολογιστές, υπολογιστές χειρός, διακομιστές και άλλοι υπολογιστές που ανήκουν σε διαφορετικά ιδρύματα ή άτομα τα οποία θέλουν να συνδεθούν στο Διαδίκτυο. Οι δρομολογητές είναι ειδικοί υπολογιστές μεταγωγής που παραλαμβάνουν εισερχόμενα πακέτα από μια ή πολλές εισερχόμενες γραμμές και τα στέλνουν στον προορισμό τους χρησιμοποιώντας μια ή περισσότερες εξερχόμενες γραμμές. Ο δρομολογητής είναι παρόμοιος με εκείνο του μεταγωγέα που χρησιμοποιείται στο τοπικό δίκτυο αλλά διαφέρει από αυτόν σε κάποια ζητήματα που δεν θα τα εξετάσουμε εδώ. Οι δρομολογητές συνδέονται μεταξύ τους σε μεγάλα δίκτυα καθώς κάθε δρομολογητής συνδέεται με καλώδια ή οπτικές ίνες με πολλούς άλλους δρομολογητές και υπολογιστές. Οι τηλεφωνικές εταιρείες και οι **Παροχείς Υπηρεσιών Διαδικτύου** (Internet Service Providers - ISP) διαχειρίζονται μεγάλα δίκτυα δρομολογητών εθνικού επιπέδου για την εξυπηρέτηση των πελατών τους.

Τα δίκτυα περιοχών και οι δρομολογητές στους ISP συνδέονται στους δρομολογητές του δικτύου στήριξης (backbone) με συνδέσεις οπτικών ινών μέσης ταχύτητας. Με τη σειρά τους, τα δίκτυα Ethernet των ιδρυμάτων διαθέτουν ένα δρομολογητή και συνδέονται στους δρομολογητές των δικτύων περιοχών. Οι δρομολογητές των ISP συνδέονται στις στοίβες των μόντεμ που χρησιμοποιούνται από τους πελάτες των ISP. Με αυτό τον τρόπο, κάθε υπολογιστής υπηρεσίας στο Διαδίκτυο διαθέτει μια τουλάχιστον μια διαδρομή προς οποιοδήποτε άλλο υπολογιστή.

Όλα τα δεδομένα του Διαδικτύου στέλνονται με τη μορφή πακέτων. Κάθε πακέτο μεταφέρει μαζί του τη διεύθυνση παραλήπτη, η οποία χρησιμοποιείται για τη δρομολόγηση. Όταν ένα πακέτο φτάσει σε κάποιο διακομιστή, εξάγει τη διεύθυνση παραλήπτη και στην συνέχεια προσπαθεί να την εντοπίσει μέσα σε έναν **πίνακα δρομολόγησης** (routing table) για να βρει σε ποια εξερχόμενη γραμμή πρέπει να σταλεί το πακέτο. Σε περίπτωση που η διεύθυνση παραλήπτη βρίσκεται στο ίδιο δίκτυο με τον διακομιστή τότε το πακέτο το προωθεί απευθείας στον υπολογιστή, διαφορετικά προωθεί το πακέτο στο επόμενο δρομολογητή που συνδέει σε άλλο δίκτυο. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να φτάσει το πακέτο στον υπολογιστή παραλήπτη. Ο πίνακας δρομολόγησης είναι δυναμικός και ενημερώνεται συνεχώς καθώς συνδέονται και α-

ποσυνδέονται δρομολογητές και σύνδεσμοι αλλάζοντας τις συνθήκες κυκλοφορίας. Τέλος, η επικοινωνία των υπολογιστών στο Διαδίκτυο βασίζεται στα πρωτόκολλα TCP/IP που θα τα εξετάζουμε διεξοδικά παρακάτω.

4.4 Ενδιάμεσο Λογισμικό Υποδομής Κατανεμημένου Συστήματος

Ο σκοπός αυτής της ενότητας είναι να περιγράψει το πανόραμα υπηρεσιών που προσφέρονται ως τμήμα της υποδομής ενός κατανεμημένου συστήματος. Συνήθως, οι υπηρεσίες αυτές είναι τμήμα του λογισμικού συστημάτων που υποστηρίζει τις κατανεμημένες εφαρμογές. Το υπόλοιπο μέρος της ενότητας εξετάζουμε γενικά τις υπηρεσίες που υποστηρίζει το ενδιάμεσο λογισμικό υποδομής.

4.4.1 Υπηρεσίες Αρχείων

Μια από τις υπηρεσίες των κατανεμημένων συστημάτων είναι η υπηρεσία αρχείων η οποία συνήθως υποστηρίζεται από ένα σύνολο διακομιστών αρχείων και συνοδεύεται από μια υπηρεσία καταλόγου. Η υπηρεσία καταλόγου επιτρέπει την εκτέλεση λειτουργιών στους καταλόγους του συστήματος αρχείως όπως δημιουργία, διαγραφή και αναζήτηση ενός αρχείου. Η υπηρεσία αρχείων επιτρέπει την εκτέλεση λειτουργιών στα περιεχόμενα των αρχείων όπως ανάγνωση και εγγραφή. Αν και στα κατανεμημένα συστήματα οι υπηρεσίες αυτές συχνά παρέχονται ξεχωριστά, συνήθως τις αντιμετωπίζουμε ως μια ενιαία υπηρεσία αρχείων.

Η υπηρεσία αρχείων παρέχει τη δυνατότητα στους υπολογιστές του κατανεμημένου συστήματος να προσπελάζουν τα αρχεία που είναι αποθηκευμένα σε απομακρυσμένους υπολογιστές ως τοπικά αρχεία. Η υπηρεσία αρχείων μπορεί είτε να παρέχει μια ενιαία εικόνα του χώρου ονομάτων του συστήματος αρχείων είτε να επιτρέπει σε κάθε υπολογιστή να έχει τη δική της εικόνα. Τέλος, η υπηρεσία αρχείων επιτρέπει στους χρήστες να συνεργάζονται μέσω κοινών αρχείων, πράγμα που συνήθως απαιτεί και την υποστήριξη μηχανισμών για κατανεμημένο κλείδωμα των αρχείων.

4.4.2 Υπηρεσίες Ονομασίας

Σε ένα κατανεμημένο σύστημα υπάρχει ένα μεγάλο πλήθος οντοτήτων όπως υπολογιστές, αρχεία, εκτυπωτές και δίκτυα που απαιτείται να προαπελαστούν από προγράμματα, Τέτοιες οντότητες συχνά έχουν φυσικές διευθύνσεις. Για παράδειγμα, στο Διαδίκτυο, οι υπολογιστές αναγνωρίζονται μοναδικά με τη χρήση της σημειογραφίας εστιγμένης τετράδας, ας πούμε 104.55.66.90.

Μια υπηρεσία ονομασίας είναι ένας τρόπος συσχέτισης των οντοτήτων ενός κατανεμημένου συστήματος με κάποια συμβολικά ονόματα. Όπως θα δούμε αργότερα ένα παράδειγμα τέτοιας υπηρεσίας: η Υπηρεσία Ονομασίας (ή Ονομάτων) Πεδίων του Διαδικτύου συσχετίζει το όνομα ενός ξένιου υπολογιστή του Διαδικτύου με κάποιο ιεραρχικά εξουσιοδοτημένο όνομα, όπως www.open.ac.uk.

Η συσχέτιση ανάμεσα σε ένα όνομα και στο φυσικό πόρο που αναφέρεται είναι γνωστή ως **δέσμευση** (binding). Η διαδικασία ανακάλυψης της φυσικής ταυτότητας ενός πόρου με δεδομένο το όνομά του είναι γνωστή ως **ανάλυση** (resolving).

Οι υπηρεσίες ονομάτων ενός κατανεμημένου συστήματος συνήθως υλοποιούνται με ιδιαίτερα αξιόπιστο τρόπο. Η υπηρεσία ονομασίας προσπελαύνεται συνεχώς από προγράμματα του συστήματος, και σε περίπτωση δυσλειτουργίας το σύστημα υφίσταται σοβαρό πλήγμα. Μια τυπική μέθοδος υλοποίησης υπηρεσίας ονομασίας είναι η ταυτόχρονη λειτουργία του σε πλήθος υπολογιστών ή διακομιστών, όπου όλοι εκτελούν την ίδια υπηρεσία.

4.4.3 Υπηρεσίες Καταλόγου

Μια υπηρεσία καταλόγου είναι κατά αρκετούς τρόπους όμοια με μια υπηρεσία ονομασίας: όντως πολλοί προγραμματιστές τις θεωρούν συνώνυμα. Όμως υπάρχει μια σημαντική διαφορά. Η υπηρεσία ονομασίας συνδέει ένα όνομα με κάποιο φυσικό αντικείμενο όπως εκτυπωτής ή υπολογιστής. Η υπηρεσία καταλόγου συσχετίζει ένα όνομα με ένα σύνολο ιδιοτήτων και πόρων. Όταν χρησιμοποιούμε μια υπηρεσία ονομάτων αναζητούμε ένα αντικείμενο συσχετισμένο με ένα μοναδικό όνομα. Όταν χρησιμοποιούμε μια υπηρεσία καταλόγου μπορούμε να ορίσουμε ένα γενικό σύνολο κριτηρίων αναζήτησης όπως ''βρες μου όλους τους εκτυπωτές στο σύστημα που έχουν ταχύτητα μεγαλύτερη των 10 σελίδων το λεπτό και είναι συνδεμένοι στα τοπικά δίκτυα LAN1 και LAN2''. Οι υπηρεσίες καταλόγου ακόμη χρησιμοποιούνται και για την αποθήκευση γενικότερων δεδομένων, όπως στοιχεία προσωπικού, τμημάτων και εταιρειών.

4.4.4 Υπηρεσίες Χρονισμού

Οι υπολογιστές ενός κατανεμημένου συστήματος έχουν τα δικά τους ρολόγια. Αυτά τα ρολόγια είναι ακριβή για εργασίες όπως η παρουσίαση της ώρας στους χρήστες. Όμως, όταν εμπλέκονται συναλλαγές είναι ευαίσθητες στο χρόνο, αυτά τα ρολόγια μπορεί να αποδειχτούν αναξιόπιστα. Τα ρολόγια των υπολογιστών υφίστανται ελαφρούς αποσυντονισμούς, για παράδειγμα οι κρυσταλλικοί ταλαντωτές είναι ευαίσθητοι στην υπερθέρμανση και η συχνότητά τους μεταβάλλεται με τη θερμοκρασία. Έτσι, προκειμένου να παρασχεθούν ακριβή χρονικά δεδομένα, συχνά στα κατανεμημένα συστήματα υλοποιείται μια υπηρεσία χρονισμού. Τέτοιες υπηρεσίες χρονισμού βασίζονται σε ρολόγια μεγάλης ακρίβειας που χρησιμοποιούν ατομικούς ταλαντωτές με ακρίβεια της τάξης 1 προς 10^{13}. Οι υπηρεσίες χρονισμού βασίζονται σε περίπλοκους αλγορίθμους και περιλαμβάνουν έναν αριθμό από πανομοιότυπους διακομιστές, έτσι ώστε άν κάποιος καταρρεύσει να μπορεί άμεσα να αναλάβει άλλος.

4.4.5 Υπηρεσίες Ομοιοτυπίας

Ένα κατανεμημένο σύστημα περιλαμβάνει ομοιότυπα δεδομένα. Υπάρχουν αρκετοί λόγοι γι' αυτό. Αν υπάρχουν ομοιότυπα δεδομένα σε διαφορετικούς διακομιστές, τότε η κατάρρευση ενός διακομιστή δεν προκαλεί διακοπή της παρεχόμενης υπηρεσίας. Επίσης υπάρχει και αύξηση της απόδοσης αφού τα ομοιότυπα μπορεί να βρίσκονται πλησιέστερα προς το χρήστη και έτσι να μειωθεί ο χρόνος επικοινωνίας που θα ήταν απαράιτητος για τη μεταφορά δεδομένων μέσω δικτύου ευρείας περιοχής.

Η υπηρεσία ομοιοτυπίας αντιγραφής εκτελεί μια σειρά από λειτουργίες:

- Καταγράφει τη φυσική θέση των ομοιότυπων.
- Παρακολουθεί τις αλλαγές σε μια βάση δεδομένων και στη συνέχεια εφαρμόζει τις ίδιες αλλαγές στα ομοιότυπα.
- Καθυστερεί τις συναλλαγές σε ένα ομοιότυπο μέχρις ότου αυτό να συγχρονιστεί με το πλέον πρόσφατο αντίγραφο.

4.4.6 Υπηρεσίες Συναλλαγών

Μια συναλλαγή είναι μια σειρά λειτουργιών σε αποθηκευμένα δεδομένα όπου όλες οι λειτουργίες ή επιτυγχάνουν ή αποτυγχάνουν, σαν ένα ενιαίο σύνολο. Ένα παράδειγμα συναλλαγής είναι το σύνολο των λειτουργιών που απαιτούνται για την εγγραφή ενός νέου πελάτη, δηλαδή η καταγραφή των προσωπικών στοιχείων, το άνοιγμα του λογαριασμού και η ενημέρωση του αρχείου ημερολογίου. Επειδή οι τρεις λειτουργίες σχετίζονται θα πρέπει είτε να εκτελεστούν πλήρως όλες ή να μην εκτελεστούν καθόλου, αν για παράδειγμα παρουσιαστεί κάποιο σφάλμα.

Κεντρικό τμήμα πολλών κατανεμημένων συστημάτων είναι η υπηρεσία συναλλαγών που παρακολουθεί τις συναλλαγές που εκτελούνται και διασφαλίζει οτι η παραπάνω ιδιότητα, μαζί με μερικές άλλες, διατηρείται. Αυτή η υπηρεσία υλοποιείται μέσω διακομιστών που μερικές φορές ονομάζονται επόπτες συναλλαγών. Τέτοιοι διακομιστές απαιτούν ιδιαίτερα περίπλοκο λογισμικό και αποτελούν βασικό πόλο δραστηριότητας των μεγάλων προμηθευτών επιχειρηματικού λογισμικού.

4.4.7 Υπηρεσίες Ελέγχου Συγχρονικότητας

Τα κατανεμημένα συστήματα λειτουργούν συγχρονικά, για παράδειγμα ένας διακομιστής βάσεων δεδομένων θα λειτουργεί ταυτόχρονα με ένα διακομιστή Ιστού και αυτός ταυτόχρονα με ένα λογισμικό εφαρμογών. Η συγχρονικότητα επιτυγχάνει καλύτερη απόδοση: δεν έχει νόημα κάποιο πρόγραμμα που εκτελείται σε έναν υπολογιστή να περιμένει ένα άλλο πρόγραμμα που εκτελείται σε άλλο υπολογιστή να τερματιστεί, εκτός αν τα δύο προγράμματα πρέπει να επικοινωνήσουν. Μερικές φορές, η επικοινωνία είναι επιβεβλημένη, αν για παράδειγμα δύο πριγράμματα πρέπει να γράψουν στην ίδια περιοχή ενός αρχείου.

Όταν υπάρχει τέτοιου τύπου επικοινωνία, τα εμπλεκόμενα δεδομένα μπορεί να καταστραφούν. Για να μη συμβεί κάτι τέτοιο, ένα κατανεμημένο σύστημα πρέπει να παρέχει μια υπηρεσία ελέγχου συγχρονικότητας. Στην ουσία αυτή η υπηρεσία διασφαλίζει οτι οι ευαίσθητες λειτουργίες ενός προγράμματος θα ολοκληρωθούν πριν κάποιο άλλο πρόγραμμα προσπελάσει τα ίδια δεδομένα.

4.4.8 Υπηρεσίες Ασφαλείας

Όπως είναι γνωστό το Διαδίκτυο είναι μάλλον ανασφαλές δίκτυο. Οι κύριοι λόγοι γι' αυτό είναι οτι βρίσκεται παντού και οτι τα πρωτόκολλά του και οι προδιαγραφές του λογισμικού του είναι δημόσια. Αυτό σημαίνει οτι οι εισβολείς μπορούν να εισέλθουν από οποιοδήποτε υπολογιστή είναι συνδεδεμένος στο Διαδίκτυο και να εκτελέσουν κακόβουλες ενέργειες, όπως καταστροφές αρχείων, ανάγνωση ευαίσθητων πληροφοριών και χρήση τους για εγκληματικούς σκοπούς, όπως η υποκατάσταση της ταυτότητας του ιδιοκτήτη μιας πιστωτικής κάρτας.

Οι υπηρεσίες ασφαλείας είναι διάχυτες μέσα σε ένα δίκτυο. Παρέχουν κρυπτογραφικές υπηρεσίες έτσι ώστε η κυκλοφορία δεδομένων σε ένα δίκτυο είναι κωδικοποιημένη και δεν μπορεί να διαβαστεί. Παρέχουν πιστοποίηση χρηστών έτσι ώστε αυτοί να ταυτοποιούνται έγκυρα στο δίκτυο και να εκτελούν ευαίσθητες εργασίες όπως πρόσβαση σε δεδομένα ή χρήση πιστωτικής κάρτας σε συναλλαγές.

4.4.9 Υπηρεσίες Επικοινωνίας

Στα περισσότερα λειτουργικά συστήματα παρέχουν υποστήριξη πολλών πρωτοκόλλων επικοινωνίας, ένα κατανεμημένο σύστημα συνήθως παρέχει πρόσθετες υπηρεσίες επικοινωνίας υψηλού επιπέδου όπως θα δούμε στην επομένη ενότητα αναλυτικά. Οι υπηρεσίες αυτές διεκολύνουν την υλοποίηση των εφαρμογών για τις οποίες έχει σχεδιαστεί το κατανεμημένο σύστημα. Ορισμένα συστήματα παρέχουν μια υπηρεσία ομαδικής επικοινωνίας μέσω της οποίας όλα τα μηνύματα που στέλνονται στην ομάδα φτάνουν σε όλους τους υπολογιστές της ομάδας με την ίδια σειρά. Άλλα συστήματα παρέχουν επικοινωνία μέσω ουρών μηνυμάτων μεσω των οποίων ο αποστολέας και ο παραλήπτης επικοινωνούν ασύγχρονα αλλά αξιόπιστα. Η υπηρεσία ηλεκτρονικού ταχυδρομείου είναι μια άλλη μορφή επικοινωνίας υψηλού επιπέδου η οποία όμως είναι προσανατολισμένη στους χρήστες και όχι στις διεργασίες σε σχέση με την επικοινωνία μέσω ουρών μηνυμάτων. Στην ενότητα ενδιάμεσο λογισμικό χρήστη περιγράφουμε πρόσθετες υπηρεσίες επικοινωνίας υψηλού επιπέδου προσανατολισμένες στη διαφανή απομακρυσμένη κλήση διαδικασιών και μεθόδων.

4.5 Πρωτόκολλα Επικοινωνίας

4.5.1 Πρωτόκολλα

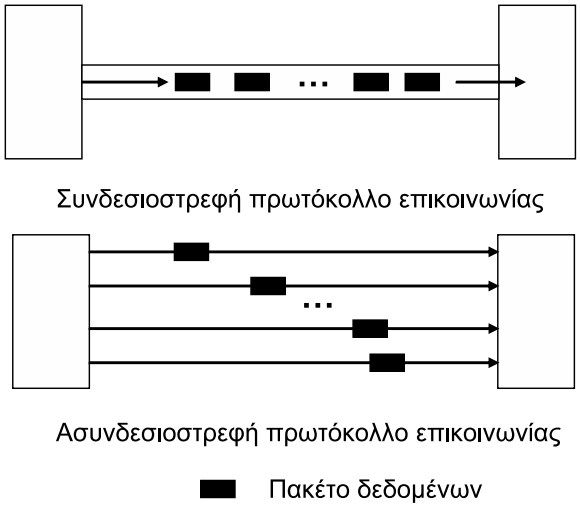
Αφού οι υπολογιστές ενός κατανεμημένου συστήματος δεν έχουν κοινή μνήμη, οποιαδήποτε μορφή επικοινωνίας μεταξύ αυτών και των διεργασιών τους πρέπει να βασίζεται σε μεταβίβαση μηνυμάτων. Όταν μια διεργασία - αποστολέας επικοινωνεί με μια διεργασία - παραλήπτη τότε η πρώτη κατασκευάζει ένα μήνυμα στο χώρο των διευθύνσεων της και καλεί μια κλήση συστήματος αποστολής. Η κλήση αυτή έχει σαν αποτέλεσμα το λειτουργικό σύστημα της πρώτης διεργασίας να στέλνει το μήνυμα μέσω του δικτύου προς την διεργασία- παραλήπτη. Στην συνέχεια ο παραλήπτης λαμβάνει το μήνυμα καλώντας μια κλήση συστήματος λήψης. Επομένως, για να είναι εφικτή η παραπάνω επικοινωνία θα πρέπει οι διεργασίες αποστολέας και παραλήπτης να συμφωνήσουν ως προς τη σημασία των μηνυμάτων που ανταλλάσονται μεταξύ τους. Το σύνολο των κανόνων που ελέγχουν την ανταλλαγή μηνυμάτων μεταξύ των διεργασιών ενός δικτύου, δηλαδή οι κανόνες που διέπουν το μορφότυπο, τη σειρά και τη σημασία των μηνυμάτων κατά την αποστολή ή παραλαβή ονομάζονται ως **πρωτόκολλα** (protocols). Το πρωτόκολλο είναι ανάλογο με την καθημερινή ζωή, όταν οι άνθρωποι μιλούν, διαβάζουν, γράφουν ή ακούν, τότε επικοινωνούν μεταξύ τους υιοθετώντας κανόνες μιας συγκεκριμένης γλώσσας (π.χ. γραμματική και συντακτικό), ώστε να μπορούν όλοι οι συμμετέχοντες να παρακολουθούν και να κατανοούν. Τα μηνύματα ή δεδομένα που ανταλλάσονται στο Διαδίκτυο είναι συνήθως σε μορφή **πακέτων** (packets).

Υπάρχουν δυο τύποι πρωτοκόλλων, τα **συνδεσιοστρεφή πρωτόκολλα** (connection-oriented protocols) και τα **ασυνδεσιοστρεφή πρωτόκολλα** (connectionless protocols). Στα συνδεσιοστρεφή πρωτόκολλα πριν γίνει η μεταφορά πακέτων δεδομένων, θα πρέπει ο αποστολέας και ο παραλήπτης να εγκαταστήσουν μια σύνδεση μεταξύ τους και πιθανόν να διαπραγματευθούν ως προς το πρωτόκολλο που θα χρησιμοποιήσουν για τη μεταξύ τους επικοινωνία. Όταν ολοκληρωθεί η μεταφορά των πακέτων δεδομένων, τότε απελευθερώνουν τη μεταξύ τους σύνδεση. Σε αυτή την περίπτωση ο αποστολέας πρέπει να καθορίζει την διεύθυνση του παραλήπτη στο πρώτο πακέτο δεδομένων κατά την στιγμή σύνδεσης και περιμένει να τερματίσει την σύνδεση. Τα υπόλοιπα πακέτα δεδομένων δεν χρειάζεται να ορίζεται η διεύθυνση του παραλήπτη εφόσον μεταδίδονται μέσα σε μια συγκεκριμένη σύνδεση. Η τηλεφωνική επικοινωνία είναι ένα

παράδειγμα συνδεσιοστρεφή πρωτοκόλλου. Αντίθετα, στα ασυνδεσιοστρεφή πρωτόκολλα δεν χρειάζεται εγκατάσταση σύνδεσης μεταξύ αποστολέα και παραλήπτη. Στις περιπτώσεις αυτές ο αποστολέας στέλνει το μήνυμα του όταν είναι έτοιμος. Σε αυτή την περίπτωση ο αποστολέας θα πρέπει να ορίζει την διεύθυνση του παραλήπτη για κάθε πακέτο δεδομένων που θέλει να στείλει. Η ταχυδρομική επικοινωνία είναι ένα παράδειγμα ασυνδεσιοστρεφή πρωτοκόλλου.

Σε ένα δίκτυο, το ασυνδεσιοστρεφή πρωτόκολλο επικοινωνίας είναι σαφώς απλό εφόσον δεν υπάρχει ανάγκη να διατηρούμε ξεχωριστές συνδέσεις. Έτσι, με την απουσία της σύνδεσης έχει σαν αποτέλεσμα τα πακέτα δεδομένων ίσως να χαθούν κατά την διάρκεια μετάδοσης ή να μεταδοθούν τα πακέτα εκτός σειράς δηλαδή όχι με την ίδια σειρά που απεστάλησαν. Από την άλλη πλευρά, το συνδεσιοστρεφές πρωτόκολλο επικοινωνίας εξασφαλίζει ότι τα πακέτα δεδομένων θα μεταδοθούν με ασφάλεια και με την ίδια σειρά κατά την διάρκεια της σύνδεσης με πρόσθετη όμως επιβάρυνση επεξεργασίας.

Στο Σχήμα 4.2 παρουσιάζεται γραφικά τις διαφορές ανάμεσα στις δύο αυτές μορφές επικοινωνίας και στο Πίνακα 4.1 παρουσιάζει συγκρίσεις στους δύο αυτούς τρόπους επικοινωνίας.



Σχήμα 4.2: Συνδεσιοστρεφή πρωτόκολλο επικοινωνίας σε σχέση με το ασυνδεσιοστρεφή πρωτόκολλο επικοινωνίας

Πίνακας 4.1: Συγκρίσεις ανάμεσα στην συνδεσιοστρεφή και ασυνδεσιοστρεφή επικοινωνία

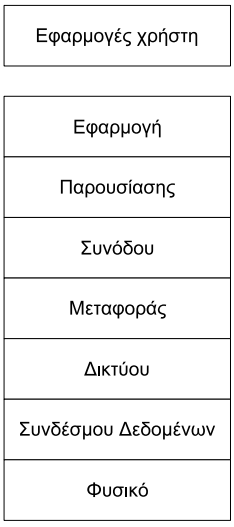
	Συνδεσιοστρεφή	Ασυνδεσιοστρεφή
Διευθυνσιοδότηση	Προσδιορίζεται κατά την στιγμή σύνδεσης	Προσδιορίζεται σε κάθε μετάδοση
Επιβάρυνση σύνδεσης	Υπάρχει για την εγκατάσταση σύνδεσης	Δεν υπάρχει
Επιβάρυνση διευθυς.	Δεν υπάρχει	Υπάρχει για κάθε μετάδοση
Σειρά παράδοσης δεδομένων	Εφικτή	Δεν είναι εφικτή
Πρωτόκολλα	Εφικτή για πρωτόκολλα που απαιτούν ανταλλαγή ενός μεγάλου όγκου δεδομένων	Εφικτή για πρωτόκολλα που απαιτούν ανταλλαγή μιας μικρής ποσότητας δεδομένων

4.5.2 Μοντέλο Αναφοράς OSI

Τα πρωτόκολλα δικτύου είναι διαιρεμένα σε επίπεδα βάσει των λειτουργιών για τις οποίες είναι υπεύθυνο κάθε επίπεδο. Το κλασικό στρωματοποιημένο σύνολο δικτυακών πρωτοκόλλων είναι γνωστό ως **μοντέλο αναφοράς για την Διασύνδεση Ανοιχτών Συστημάτων** (Open Systems Interconnection reference model, OSI). Ένα **ανοιχτό σύστημα** (open system) είναι ένα σύστημα του οποίου η αρχιτεκτονική δεν είναι μυστική και μπορεί να επικοινωνήσει με άλλα διαφορετικά ανοιχτά συστήματα σε αντίθεση με τα κλειστά συστήματα. Το μοντέλο OSI αποτελείται από επτά επίπεδα που αντιστοιχούν σε μία προκαθορισμένη ομάδα λειτουργιών όπως παρουσιάζεται στο Σχήμα 4.3. Κάθε επίπεδο περιγράφεται παρακάτω και έχει ένα όνομα: φυσικό, συνδέσμου δεδομένων, δικτύου, μεταφοράς, συνόδου, παρουσίασης και εφαρμογής.

Το **φυσικό επίπεδο** (physical layer) είναι υπεύθυνο για τη μετάδοση ηλεκτρικών σημάτων μέσω του φυσικού μέσου που αντιστοιχούν σε μέλη των κατώτερων επιπέδων όπως τα **δυφία** (bits).

Το **επίπεδο συνδέσμου δεδομένων** (data link layer) διαχειρίζεται την μεταβίβαση των στοιχειώδων δεδομένων όπως τα δυφία. Για τον σκοπό αυτό συνεργάζεται με το τελευταίο επίπεδο: το φυσικό επίπεδο. Το επίπεδο αυτό είναι υπεύθυνο για την παρακολούθηση



Σχήμα 4.3: Μοντέλο αναφοράς OSI

και ανάκτηση των λανθασμένων μεταβιβάσεων από τα κατώτερα επίπεδα, π.χ. λάθη εξαιτίας δυσλειτουργιών υλικού.

Το **επίπεδο δικτύου** (network layer) είναι υπεύθυνο για την μετάδοση των δεδομένων μεταξύ του αποστολέα και του παραλήπτη. Για να πραγματοποιήσει αυτή την εργασία, το επίπεδο δικτύου ασχολείται με την **διευθυνσιοδότηση** (addressing) και την **δρομολόγηση** (routing) των δεδομένων που θα ακολουθήσουν μέσα στο δίκτυο.

Το **επίπεδο μεταφοράς** (transport layer) είναι υπεύθυνο για την αξιόπιστη μετάδοση των δεδομένων μέσα στο δίκτυο. Σκοπός του είναι να ελέγχει για την έγκυρη ανταλλαγή δεδομένων ανάμεσα στα διάφορα μέρη του δικτύου. Αν συμβαίνει το αντίθετο, τότε οφείλει να ενημερώσει τον αποστολέα για το πρόβλημα που προέκυψε. Κανονικά, στην περίπτωση αυτή, τα δεδομένα θα αποσταλούν ξανά.

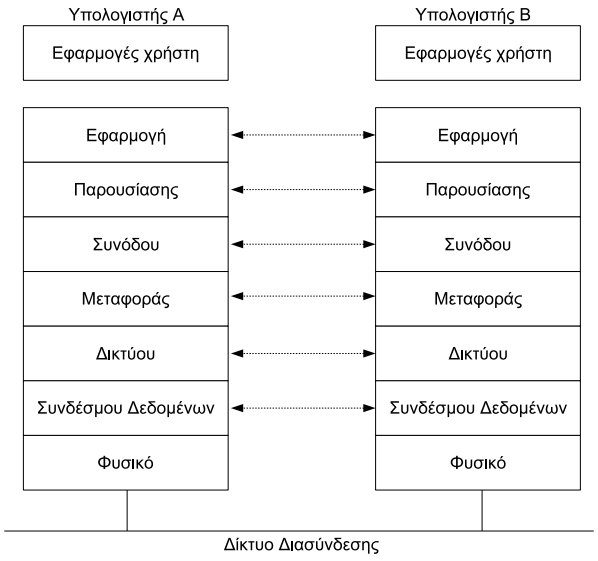
Το **επίπεδο συνόδου** (session layer) είναι υπεύθυνο για το συγχρονισμό της ανταλλαγής δεδομένων μεταξύ των εφαρμογών. Επιτρέπει σε κάθε εφαρμογή να γνωρίζει την κατάσταση των άλλων εφαρμογών. Έτσι, αν, π.χ., κάποιο πρόγραμμα περιμένει την αποστολή δεδομένων από ένα άλλο πρόγραμμα και το δεύτερο πάθει κάποια βλάβη, το επίπεδο συνόδου του δεύτερου θα ενημερώσει το πρώτο πρόγραμμα για το τι συμβαίνει.

Το **επίπεδο παρουσίασης** (presentation layer) λειτουργεί ως πύλη μεταξύ του επιπέδου εφαρμογής και των υπόλοιπων επιπέδων του μοντέλου. Το επίπεδο αυτό είναι υπεύθυνο για

την μετατροπή των πληροφοριών που βρίσκονται στο επίπεδο εφαρμογής, οι οποίες μπορεί να δίνονται σε διαφορετικές μορφές, όπως σε ASCII, σε μια μορφή που να είναι κατανοητή από το δίκτυο και ανεξάρτητη από συγκεκριμένες εφαρμογές. Διεξάγει επίσης και την αντίστροφη διαδικασία μετατροπής των δεδομένων, δηλαδή από την μορφή που χρησιμοποιεί εσωτερικά το δίκτυο σε κάποια μορφή που είναι κατανοητή από κάποια εφαρμογή του επιπέδου εφαρμογής.

Τέλος, στο **επίπεδο εφαρμογής** (application layer) εντάσσονται εφαρμογές όπως το ηλεκτρονικό ταχυδρομείο. Αυτό το επίπεδο είναι υπεύθυνο για την εμφάνιση στον χρήστη δεδομένων που μεταδίδονται μέσω του δικτύου. Επίσης, επικοινωνεί με τα άλλα επίπεδα του μοντέλου μέσω του επιπέδου παρουσίασης.

Κάθε επίπεδο στο μοντέλο OSI ορίζει μια λειτουργία επικοινωνίας δεδομένων, η οποία υλοποιείται στην πράξη από διάφορα πρωτόκολλα. Κάθε πρωτόκολλο επικοινωνεί με το αμέσως υψηλότερο με σκοπό να παρέχει υπηρεσίες και το αμέσως χαμηλότερο στον ίδιο υπολογιστή προκειμένου να χρησιμοποιήσει τις υπηρεσίες που του παρέχει. Επίσης, κάθε πρωτόκολλο επικοινωνεί με το ομόλογο του το οποίο αποτελεί την υλοποίηση του ίδιου πρωτόκολλου στο ισοδύναμο επίπεδο σε ένα απομακρυσμένο υπολογιστή όπως φαίνεται στο Σχήμα 4.4. Για να

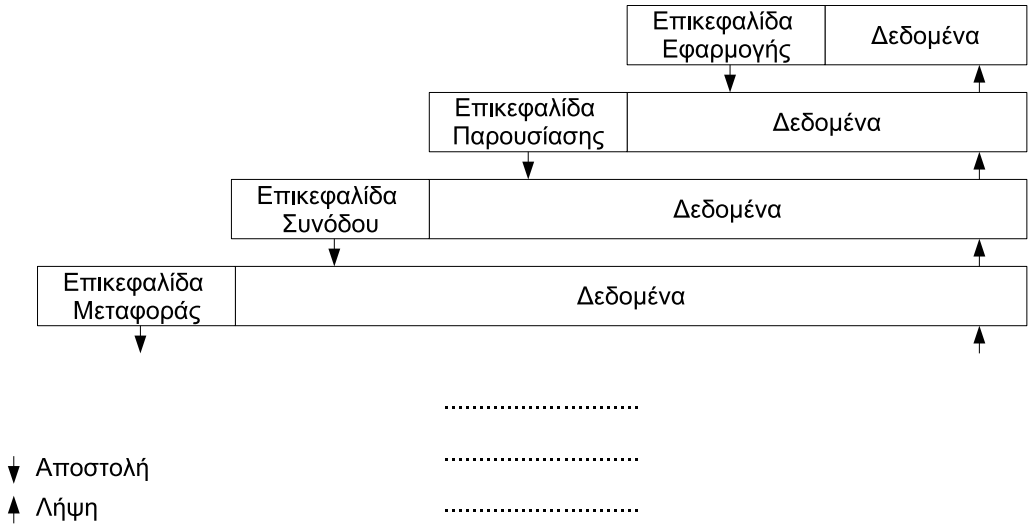


Σχήμα 4.4: Ομόλογα επίπεδα

επιτευχθεί η επικοινωνία, οι επικοινωνίες σε επίπεδο ομόλογου πρωτοκόλλου πρέπει να είναι τυποποιημένες.

Στην πραγματικότητα μόνο τα φυσικά επίπεδα επικοινωνούν άμεσα μέσω του δικτύου. Τα

επτά επίπεδα του μοντέλου OSI χρησιμοποιούν διάφορους τύπους πληροφοριών ελέγχουν για να επικοινωνήσουν (νοητά) με το ομόλογο επίπεδο σε άλλους υπολογιστές. Οι πληροφορίες ελέγχου τυπικά λαμβάνουν τη μορφή **επικεφαλίδων ελέγχου** (headers). Έτσι, όταν μια διεργασία ενός υπολογιστή θέλει να επικοινωνήσει με μια διεργασία που εκτελείται σε άλλο υπολογιστή, τότε η πρώτη κατασκευάζει ένα μήνυμα και το μεταβιβάζει στο επίπεδο εφαρμογής του υπολογιστή του. Το επίπεδο αυτό προσθέτει κάποιες πληροφορίες ελέγχου στην αρχή του μηνύματος και το μεταβιβάζει στο αμέσως χαμηλότερο επίπεδο. Έτσι, οι πληροφορίες ελέγχου και το μήνυμα για το επόμενο επίπεδο θεωρείται συνολικά ως μήνυμα και στο επίπεδο αυτό προσθέτει τις επικεφαλίδες ελέγχου στην αρχή του νέου μηνύματος που προέκυψε από το προηγούμενο υψηλότερο επίπεδο. Η διαδικασία αυτή συνεχίζεται από επίπεδο σε επίπεδο μέχρι να φτάσουμε στο φυσικό επίπεδο που στέλνει το τελικό μήνυμα μέσω του δικτύου επικοινωνίας. Η πρακτική αυτή είναι γνωστή ως **ενθυλάκωση** (encapsulation) και φαίνεται στο Σχήμα 4.5.

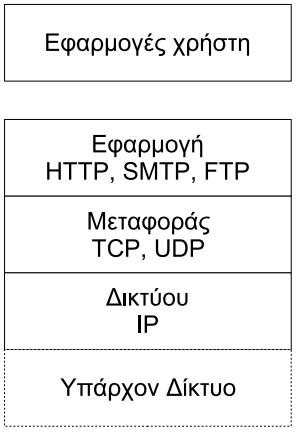


Σχήμα 4.5: Ενθυλάκωση

Όταν το μήνυμα φτάσει στην διεργασία παραλήπτη, το φυσικό επίπεδο του υπολογιστή αυτού το μεταβιβάζει στο αμέσως υψηλότερο επίπεδο. Κάθε επίπεδο αναλύει και απομακρύνει τις αντίστοιχες πληροφορίες ελέγχου που είχε προσθέσει το ομόλογο του επίπεδο στο μήνυμα πριν το μεταβιβάσει προς το αμέσως υψηλότερο επίπεδο μέχρι το μήνυμα να φτάσει στην διεργασία παραλήπτη.

4.5.3 Μοντέλο Αναφοράς TCP/IP

Το Διαδίκτυο είναι ένα ανοιχτό σύστημα το οποίο έχει μια στρωματοποιημένη αρχιτεκτονική. Αυτό σημαίνει, όπως και στην περίπτωση του μοντέλου OSI, ότι οι λειτουργίες του συστήματος ομαδοποιούνται σε επίπεδα που επικοινωνούν με κατώτερα επίπεδα ώστε να επιτευχθεί η επιθυμητή λειτουργικότητα. Έτσι, έχει κατασκευαστεί ένα λογικό μοντέλο μιας ομάδας πρωτοκόλλων που χρησιμοποιούνται στο Διαδίκτυο, το οποίο ονομάζουμε **μοντέλο αναφοράς TCP/IP** (TCP/IP reference model). Το όνομα αυτό προέρχεται από τα πιο γνωστά πρωτόκολλα του Διαδικτύου το TCP (Transmission Control Protocol) και το IP (Internet Protocol). Το TCP/IP είχε αναπτυχθεί πριν από το μοντέλο OSI και για το λόγο αυτόν τα επίπεδα του δεν ταιριάζουν ακριβώς με αυτά του μοντέλου OSI αφού άλλωστε περιέχει τέσσερα αντί για επτά επίπεδα. Το μοντέλο αυτό παρουσιάζεται στο Σχήμα 4.6.



Σχήμα 4.6: Μοντέλο TCP/IP

Στο μοντέλο αναφοράς TCP/IP δεν υπάρχει ούτε φυσικό επίπεδο ούτε επίπεδο συνδέσμου δεδομένων. Ο λόγος είναι ότι το Διαδίκτυο σχεδιάστηκε για να διασυνδέει διαφορετικά υπάρχοντα δίκτυα, τα οποία χρησιμοποιούν τα δικά τους πρωτόκολλα στα κατώτερα επίπεδα. Έτσι, το υψηλότερο επίπεδο του TCP/IP είναι το επίπεδο εφαρμογής το οποίο είναι ένας συνδυασμός των επιπέδων συνόδου, παρουσίασης και εφαρμογής του μοντέλου OSI. Αυτό σημαίνει ότι ένα μόνο επίπεδο, το επίπεδο εφαρμογής του μοντέλου TCP/IP χειρίζεται όλες τις λειτουργίες που σχετίζονται με αυτά τα τρία επίπεδα. Τα πιο γνωστά πρωτόκολλα του Διαδικτύου σε επίπεδο εφαρμογής είναι το πρωτόκολλο μεταφοράς αρχείων (File Transfer Protocol - FT-

P), το πρωτόκολλο μεταφοράς αλληλογραφίας (Simple Mail Transfer Protocol - SMTP), το πρωτόκολλο μεταφοράς υπερκειμένου (HyperText Transfer Protocol - HTTP), το πρωτόκολλο απομακρυσμένης σύνδεσης (TELNET) και το πρωτόκολλο ονομασίας (Domain Name System - DNS). Μερικά πρωτόκολλα χρησιμοποιούνται άμεσα από τους χρήστες με την μορφή εφαρμογών όπως για παράδειγμα το TELNET. Μερικά άλλα πρωτόκολλα βρίσκονται πίσω από εφαρμογές όπως το SMTP και HTTP. Τέλος, κάποια άλλα όπως το DNS χρησιμοποιούνται έμμεσα από το χρήστη ή από συναρτήσεις του λειτουργικού συστήματος.

Στο επίπεδο μεταφοράς το TCP/IP χρησιμοποιεί δύο πρωτόκολλα, το **πρωτόκολλο ελέγχου μετάδοσης** (Transmission Control Protocol - TCP) και το **πρωτόκολλο δεδομενογραμμάτων πακέτων χρήστη** (User Datagram Protocol - UDP), τα οποία παρέχουν διαφορετικά είδη υπηρεσιών. Το πρωτόκολλο TCP είναι συνδεσιοστρεφές αφού επιτρέπει την αξιόπιστη μετάδοση των δεδομένων μέσα στο δίκτυο. Η βασική λειτουργία του πρωτοκόλλου αυτού είναι η διαίρεση ενός μηνύματος το οποίο προέρχεται από το επίπεδο εφαρμογής σε μια σειρά από μικρότερα πακέτα TCP τα οποία είναι αριθμημένα ακολουθιακά και στην συνέχεια αποστέλει τα πακέτα στον προορισμό τους. Επίσης, επαναμεταδίδει τα πακέτα αν λόγω κάποιου σφάλματος τα πακέτα δεν φτάσουν στον προορισμό τους. Επιπλέον, αν κάποιο πακέτο παραληφθεί εκτός σειράς, ταξινομείται με τη βοήθεια του μηχανισμού αρίθμησης της ακολουθίας. Το πρωτόκολλο TCP χρησιμοποιεί ένα σύστημα επιβεβαιώσεων, αθροισμάτων ελέγχου και χρονισμού. Επίσης, το TCP χρησιμοποιεί μηχανισμούς ελέγχου ροής ώστε να εξασφαλίζεται η ομαλή ροή των δεδομένων μεταξύ αποστολέα και παραλήπτη. Αντίθετα, το πρωτόκολλο UDP πραγματοποιεί τις ίδιες λειτουργίες με το TCP. Ωστόσο, αντίθετα με το TCP, δεν μπορεί να επαναλάβει την διαδικασία αποστολής των πακέτων, δεν διασφαλίζει δηλαδή την αξιόπιστη μετάδοση των πακέτων. Το πλεονέκτημα αυτού του πρωτοκόλλου, είναι ότι διευκολύνει την ταχεία αποστολή δεδομένων όπως η μετάδοση ήχου και βίντεο. Ωστόσο, δεν γίνεται έλεγχος για λάθη κι έτσι, κάτω από ιδιαίτερες συνθήκες μέσα στο δίκτυο, όπως για παράδειγμα μεγάλη κυκλοφορία, μπορεί να υπάρξει απώλεια πακέτων ή μεταφορά παραποιημένων πακέτων.

Το χαμηλότερο επίπεδο του TCP/IP είναι το επίπεδο δικτύου. Τα πακέτα που προέρχονται είτε από το TCP είτε από το UDP, αποστέλλονται μαζί με τη διεύθυνση του παραλήπτη στο λογισμικό που υλοποιεί το **πρωτόκολλο Διαδικτύου** (Internet Protocol - IP). Το πρωτόκολλο IP μπορεί να θεωρηθεί ως το βασικό στοιχείο του Διαδικτύου, επειδή στο επίπεδο αυτό λαμβάνουν χώρα οι δρομολογήσεις πακέτων προς στο σωστό δίκτυο και υπολογιστή. Το

IP περιλαμβάνει ο καθορισμός του πακέτου IP που ονομάζεται **δεδομενόγραμμα IP** (IP datagram) που αποτελεί τη βασική μονάδα μετάδοσης στο Διαδίκτυο, ο καθορισμός ενός συστήματος διευθυνσιοδότησης για την αναγνώριση υπολογιστών στο δίκτυο και η δρομολόγηση των πακέτων δεδομενογραμμάτων σε απομακρυσμένα συστήματα υπολογιστών. Έτσι, τα δεδομενογράμματα πακέτα περιέχουν την πλήρη διεύθυνση IP του αποστολέα και του παραλήπτη πέρα από τα δεδομένα του μηνύματος. Τα δεδομενογράμματα πακέτα που ανήκουν στο ίδιο ή σε διαφορετικά μηνύματα μπορούν να μεταδίδονται αυτόνομα μέσω του δικτύου, πράγμα που σημαίνει ότι δύο διαδοχικά δεδομενογράμματα πακέτα μπορούν να ακολουθήσουν διαφορετικές διαδρομές και να φτάνουν στον παραλήπτη με διαφορετική σειρά από αυτήν της αποστολής τους. Επίσης, πρέπει να σημειωθεί ότι το IP είναι ασυνδεσιοστρεφές πρωτόκολλο όπως στο πρωτοκόλλο UDP και δεν παρέχει εγγυήσεις παράδοσης των μηνυμάτων στον παραλήπτη. Τέλος, το IP δεν παρακολουθεί τις διαδρομές και δεν έχει δυνατότητα αναδιοργάνωσης των δεδομενογραμμάτων πακέτων μετά την άφιξη τους.

4.5.4 Διευθυνσιοδότηση

Ένα σημαντικό χαρακτηριστικό της δικτύωσης είναι η διευθυνσιοδότηση που είναι ένας μηχανισμός για την αναγνώριση αποστολείς και παραλήπτες. Το μοντέλο TCP/IP χρησιμοποιεί τρία διαφορετικά επίπεδα διευθυνσιοδότησης: **Φυσική διεύθυνση** (physical address), **διεύθυνση IP** (IP address) και **διεύθυνση θύρας** (port address). Η φυσική διεύθυνση είναι μια διεύθυνση χαμηλού επιπέδου και αφορά την κάρτα δικτύου ενός υπολογιστή. Για παράδειγμα, είναι μια διεύθυνση των 48 δυφίων και είναι γραμμένη στο υλικό της κάρτας. Πολλές φορές, η φυσική διεύθυνση ονομάζεται και ως διεύθυνση MAC (Media Access Control). Σε αυτή την εντότητα θα εξετάζουμε αναλυτικά για την διεύθυνση IP και στην επόμενη ενότητα για τους αριθμούς θύρας.

Το μοντέλο TCP/IP απαιτεί από κάθε υπολογιστή που συνδέεται στο Διαδίκτυο να προσδιορίζεται από μια μοναδική διεύθυνση γνωστή ως διεύθυνση IP. Αυτή η μοναδική διεύθυνση εκφράζεται σε μια σημειογραφία γνωστή ως **σημειογραφία εστιγμένης τετράδας** (dotted quad notation).

Μια διεύθυνση IP αποτελείται από 32 δυφία τα οποία συνήθως ομαδοποιούμε σε τέσσερις ακέραιους αριθμούς που παίρνουν τιμές από 0 εως 255 διαχωρισμένα με τελείες (κάθε αριθμός

παίρνει ένα byte). Για παράδειγμα, μια διεύθυνση IP έχει τη μορφή 195.251.10.98. Στην πραγματικότητα, μια διεύθυνση IP χωρίζεται σε τρία μέρη:

1. Το είδος της διεύθυνσης το οποίο αναφέρεται στο μέγεθος του δικτύου και είναι γνωστό ως **κλάση** (class).

2. Την **ταυτότητα δικτύου** (network identifier) η οποία αναφέρεται σε ένα ιδιαίτερο κομμάτι ενός δικτύου που συνδέεται στο Διαδίκτυο. Η ταυτότητα δικτύου αναγνωρίζει με ένα μοναδικό τρόπο ένα τμήμα δικτύου (υποδίκτυο).

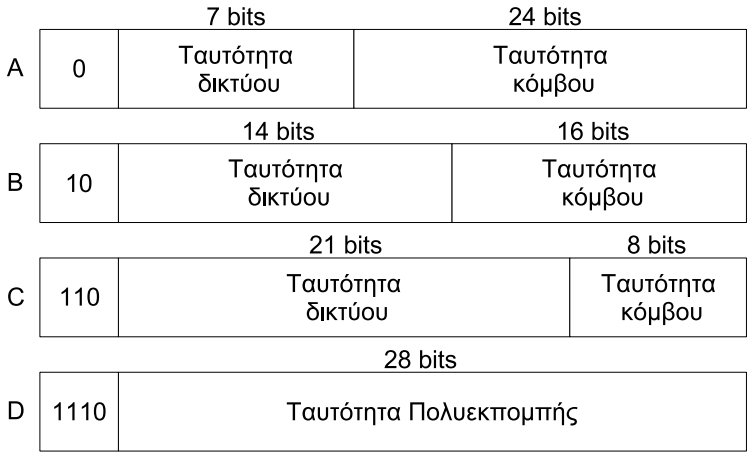
3. Την **ταυτότητα κόμβου** (host identifier) η οποία αναφέρεται και αναγνωρίζει μια μοναδική συσκευή (π.χ. υπολογιστή) που βρίσκεται συνδεδεμένη στο συγκεκριμένο τμήμα του δικτύου.

Παράδειγμα, μια πολυκατοικία είναι ένα υποδίκτυο διαμερισμάτων και βρίσκεται σε μια συγκεκριμένη διεύθυνση (οδός, αριθμός). Αυτή είναι η ταυτότητα δικτύου της. Για να πάμε σε ένα συγκεκριμένο διαμέρισμα της πολυκατοικίας (κόμβος) πρέπει ακόμα να γνωρίζουμε τον όροφο και την θέση στον όροφο. Αυτό είναι η ταυτότητα κόμβου.

Με δεδομένο ότι χρησιμοποιούνται 32 δυφία για την διεύθυνση ενός υπολογιστή και αυτά πρέπει να περιέχουν τόσο την ταυτότητα δικτύου όσο και την ταυτότητα κόμβου, τίθεται το ερώτημα πόσα από τα ψηφία αυτά χρησιμοποιούνται για το ένα και πόσα για το άλλο. Έτσι, το πλήθος των ψηφίων για τα δύο τμήματα της διεύθυνσης εξαρτάται από το μέγεθος του δικτύου και για αυτό το λόγο έχουν ταξινομηθεί τέσσερις κλάσεις δικτύων. Οι κλάσεις αυτές είναι γνωστές ως κλάση A, κλάση B, κλάση C και κλάση D. Το Σχήμα 4.7 δείχνει τον χωρισμό των 32 δυφίων για κάθε μια από αυτές τις κλάσεις.

Στην κλάση A, για ταυτότητα δικτύου χρησιμοποιούνται 7 δυφία ενώ για ταυτότητα κόμβου μέσα στο δίκτυο χρησιμοποιούνται 24 δυφία. Έτσι, στην κλάση αυτή μπορούμε να έχουμε 128 διαφορετικά δίκτυα συνδεδεμένα μεταξύ τους αν είναι κλάσης A. Το κάθε δίκτυο μπορεί να έχει μέχρι $2^{24} = 16.777.216$ υπολογιστές. Συνεπώς, οι διευθύνσεις Κλάσης A χρησιμοποιούνται για μεγάλα δίκτυα που συμπεριλαμβάνουν πολλούς υπολογιστές.

Στην κλάση B, για ταυτότητα δικτύου χρησιμοποιούνται 14 δυφία ενώ για ταυτότητα κόμβου μέσα στο δίκτυο χρησιμοποιούνται 16 δυφία. Μπορούμε να έχουμε έτσι $2^{14} = 16384$ διαφορετικά



Σχήμα 4.7: Διευθύνσεις IP

δίκτυα σε κλάση B. Το κάθε δίκτυο μπορεί να έχει μέχρι $2^{16} = 65536$ υπολογιστές. Έτσι, οι διευθύνσεις κλάσης B χρησιμοποιούνται για δίκτυα μεσαίου μεγέθους.

Στην κλάση C για ταυτότητα δικτύου χρησιμοποιούνται 21 δυφία ενώ για ταυτότητα κόμβου μέσα στο δίκτυο χρησιμοποιούνται 8 δυφία. Μπορούμε να έχουμε έτσι $2^{21} = 2097152$ διαφορετικά δίκτυα συνδεμένα μεταξύ τους σε κλάση C. Το κάθε δίκτυο μπορεί να έχει μέχρι $2^8 = 256$ υπολογιστές. Οι διευθύνσεις κλάσης C χρησιμοποιούνται για μικρότερα δίκτυα όπως το εργαστήριο ενός μικρού πανεπιστημιακού τμήματος.

Οι διευθύνσεις κλάσης D χρησιμοποιούνται για την **πολυεκπομπή** (multicasting) κατά την οποία ένα μήνυμα αποστέλλεται σε πολλούς υπολογιστές. Στην κλάση D, τα 28 δυφία που απομένουν αφού αφαιρέσουμε τα τέσσερα που δηλώνουν ότι η διεύθυνση είναι κλάσης D, σχηματίζουν αυτό που ονομάζεται διεύθυνση πολυεκπομπής και που δηλώνουν την ομάδα υπολογιστών που θα λάβουν τα δεδομένα.

Όπως γνωρίζουμε κάθε διεύθυνση IP είναι μοναδική και μπορεί να χρησιμοποιηθεί για ν' αναγνωρίσει έναν υπολογιστή σ'ενα δίκτυο TCP/IP. Ωστόσο, η σημειογραφία εστιγμένης τετράδας είναι κατάλληλη για τους υπολογιστές να θυμηθούν αριθμούς αλλά για τον άνθρωπο είναι κάπως δύσκολη στη χρήση, δηλαδή, να θυμηθεί τέσσερις αριθμούς για να αποκτήσει πρόσβαση σ' έναν υπολογιστή. Για το λόγο αυτό θα ήταν πολύ καλύτερο οι διευθύνσεις να αναπαρασταθούν με μία συμβολική ονομασία ώστε να μπορούμε να αναφερόμαστε στους υπολογιστές με ευκολομνημόνευτα ονόματα που διαχωρίζονται επίσης με τελείες. Ευτυχώς υπάρχει

μία υπηρεσία που προσφέρει αυτήν την διευκόλυνση και είναι γνωστή ως **σύστημα ονομασίας περιοχής** (Domain Name System - DNS). Κάθε φορά που περιηγούμε μια Ιστοσελίδα αναφερόμαστε σε ένα υπολογιστή του Διαδικτύου μέσω του ονόματος περιοχής βασισμένο στο πρωτόκολλο DNS.

Το όνομα περιοχής ακολουθεί ένα ιεραρχικό σχήμα απόδοσης ονομάτων σε υπολογιστές σ΄ ένα δίκτυο TCP/IP. Με άλλα λόγια, κάθε όνομα περιοχής αποτελείται από μια σειρά λέξεις που διαχωρίζονται με τελείες. Κάθε λέξη του ονόματος ονομάζεται **πεδίο** (domain). Ένα παράδειγμα ονόματος περιοχής είναι της μορφής `www.uom.gr`. Το τελευταίο πεδίο του ονόματος `gr` σε αυτή την περίπτωση, λέγεται πεδίο ανωτέρου επιπέδου και αντιστοιχεί σε μια μεγαλύτερη ομάδα υπολογιστών που σχετίζονται με την Ελλάδα. Το επόμενο πεδίο του ονόματος `uom` σε αυτή την περίπτωση, λέγεται πεδίο δευτέρου επιπέδου και αντιστοιχεί σε μια ακόμη μικρότερη ομάδα υπολογιστών του Πανεπιστημίου Μακεδονίας. Στην συνέχεια το όνομα περιοχής ίσως να ακολουθεί και άλλα υποπεδία που να αντιστοιχούν σε διαφορετικά επίπεδα του ιεραρχικού δένδρου μέχρι να φτάσουμε το πρώτο πεδίο `eos` που αντιστοιχεί σε ένα συγκεκριμένο υπολογιστή του Πανεπιστημίου Μακεδονίας, δηλαδή ο διακομιστής. Με βάση τα προηγούμενα, το τελευταίο πεδίο ενός ονόματος περιοχής αντιστοιχεί σε ένα μεγαλύτερο τμήμα δικτύου, τα ενδιάμεσα πεδία αντιστοιχούν σαφώς μικρότερα τμήματα υποδικτύων και το πρώτο πεδίο αντιστοιχεί σε ένα όνομα υπολογιστή.

Στο Διαδίκτυο αυτήν την στιγμή υπάρχουν αρκετά πεδία ανωτέρου επιπέδου όπως φαίνονται στον Πίνακα 4.2.

Πίνακας 4.2: Ανώτερα πεδία

Ανώτερο όνομα πεδίου	Οργανισμός
com	Εμπορική εταιρεία
edu	Εκπαιδευτικό ίδρυμα
gov	Κυβερνητικός οργανισμός
mil	Στρατιωτικός οργανισμός
org	Διάφοροι οργανισμοί
Πεδία χωρών	gr για Ελλάδα, fr για Γαλλία, ca για Καναδά κλπ

Είναι γνωστό ότι κάθε όνομα περιοχής αντιστοιχίζεται σε μια αντίστοιχη διεύθυνση IP.

Έτσι, όταν απευθυνόμαστε σε ένα υπολογιστή με το συμβολικό όνομα, θα πρέπει το όνομα να μεταφραστεί στην αντίστοιχη διεύθυνση IP προκειμένουν να εντοπίσει το υπολογιστή. Η διαδικασία μετάφρασης ενός ονόματος περιοχής στην αντίστοιχη διεύθυνση IP και αντίστροφα ονομάζεται **επίλυση ονόματος DNS** (DNS naming resolution). Η διαδικασία επίλυσης ονόματος λαμβάνει χώρα ως εξής: Όταν ένας υπολογιστής επιθυμεί να επικοινωνήσει με έναν άλλο υπολογιστή μέσω του συμβολικού ονόματος, το όνομα υποβάλλεται σε ένα πλησιέστερο διακομιστή ονομάτων (DNS server) για να μετατρέψει το συμβολικό όνομα σε διεύθυνση IP από την βάση της δεδομένων. Αν ο διακομιστής δεν εντοπίζει την ζητούμενη αντιστοίχιση ονόματος - διεύθυνση IP, τότε ο διακομιστής προωθεί το ζητούμενο όνομα σε ένα άλλο διακομιστή ονόματος. Η διάδοση του ζητούμενου ονόματος συνεχίζεται μέχρι ότου βρεθεί μια αντιστοίχιση και έτσι μπορεί να γίνει η επικοινωνία με το ζητούμενο υπολογιστή μέσω της διεύθυνσης IP. Συνεπώς, η όλη διαδικασία μπορεί να απαιτήσει την επικοινωνία με αρκετούς διακομιστές ονόματος αφού δεν υπάρχει ένας διακομιστής ονόματος που να έχει όλες τις καταχωρήσεις συμβολικών ονομάτων για όλους τους υπολογιστές του κόσμου. Το σύνολο των διακομιστών ονόματος ονομάζεται **Σύστημα Ονομασίας Πεδίων Διαδικτύου** (Internet Domain Name System - DNS).

Τέλος, το όνομα πεδίου localhost χρησιμοποιείται για να αναφέρεται σε ένα τοπικό υπολογιστή στο οποίο εκτελείται μια διεργασία. Αυτό το όνομα αντιστοιχίζεται πάντα στην διεύθυνση IP 127.0.0.1 η οποία είναι γνωστή ως **διεύθυνση τοπικού βρόγχου** (loopback address). Όποια δεδομένα αποστέλλονται σ΄ αυτήν την διεύθυνση από έναν υπολογιστή θα επιστρέψουν απευθείας στον ίδιο υπολογιστή.

4.5.5 Θύρες και Υποδοχές

Για να μεταφερθούν δεδομένα μεταξύ των διεργασιών αποστολέα - παραλήπτη μιας δικτυακής εφαρμογής (π.χ. ηλεκτρονικού ταχυδρομείου) που εκτελούνται σε δύο δικτυακά συνδεδεμένους υπολογιστές, πρέπει να εξασφαλιστεί κατ΄ αρχήν ότι τα δεδομένα θα παραδοθούν σε πρώτη φάση στο σωστό δίκτυο, στην συνέχεια στο σωστό υπολογιστή μέσα στο δίκτυο και τέλος ότι τα δεδομένα θα φτάσουν στην σωστή εφαρμογή ή διεργασία μέσα στον υπολογιστή.

Για να υλοποιηθούν τα παραπάνω, χρησιμοποιείται πρώτα ένα σύνολο διευθύνσεων IP ή ονομάτων πεδίων μέσω των οποίων προσδιορίζεται μονοσήμαντα κάθε υπολογιστή στο δίκτυο

TCP/IP όπως είδαμε στην προηγούμενη ενότητα. Όμως, χρειαζόμαστε ένα σχήμα διευθυνσιοδότησης για να προσδιορίζουμε μοναδικά μια διεργασία που εκτελείται σε έναν υπολογιστή. Στο Διαδίκτυο το πρωτοκόλλο που χρησιμοποιείται για την αναγνώριση διεργασιών περιλαμβάνει την χρήση μιας λογικής οντότητας γνωστή ως **διεύθυνση θύρας** ή **θύρα** (port) για συντομία. Συνεπώς χρησιμοποιούνται τεχνικές πολύπλεξης με βάση αριθμούς πρωτοκόλλων και αριθμούς θυρών ώστε να παραδοθούν τα δεδομένα στη σωστή εφαρμογή μέσα στο σωστό υπολογιστή. Παρακάτω θα περιγράψουμε σύντομα την τεχνική την οποία παραδίδονται τα δεδομένα στην σωστή εφαρμογή, μετά την άφιξη τους στο σωστό υπολογιστή.

Αφού τα δεδομένα δρομολογηθούν στο σωστό δίκτυο, φτάνουν στον σωστό υπολογιστή παραλήπτη από το δίκτυο πακέτα ή δεδομένα IP επειδή απλά έχουν σταλεί στην συγκεκριμένη διεύθυνση IP. Επομένως, τα πακέτα IP που φτάνουν στον υπολογιστή παραλήπτη, πρέπει να πολυπλεχθούν, δηλαδή να διαχωριστούν για παράδοση από το πρωτόκολλο IP σε ένα από τα πρωτόκολλα επιπέδου μεταφοράς (TCP, UDP) που βρίσκεται από πάνω. Για να προσδιοριστεί το πρωτόκολλο μεταφοράς που θα παραλάβει τα δεδομένα που φέρει το πακέτο IP, η υλοποίηση του πρωτοκόλλου IP εξετάζει τον αριθμό πρωτοκόλλου του πακέτου. Μόλις τα δεδομένα παραδοθούν στο αντίστοιχο πρωτόκολλο μεταφοράς (TCP, UDP), πρέπει στην συνέχεια να παραδοθούν από το πρωτόκολλο μεταφοράς στην σωστή εφαρμογή ή διεργασία του υπολογιστή παραλήπτη. Για να προσδιοριστεί η σωστή εφαρμογή, το αντίστοιχο πρωτόκολλο μεταφοράς ελέγχει τους αριθμούς θυρών και με βάση αυτούς προωθεί τα δεδομένα σε μια συγκεκριμένη εφαρμογή.

Για να γίνει η μεταφορά δεδομένων στην σωστή εφαρμογή καθιερώθηκαν οι λεγόμενες θύρες πρωτοκόλλου. Κάθε θύρα έχει ένα συγκεκριμένο θετικό ακέραιο αριθμό και διαφορετικές δικτυακές εφαρμογές χρησιμοποιούν διαφορετικές θύρες για την επικοινωνία τους. Συνεπώς, οι θύρες είναι στην ουσία αριθμητικά ονόματα για τις αντίστοιχες δικτυακές υπηρεσίες. Έτσι ένα πακέτο TCP δεν κατευθύνεται απλώς προς μια συγκεκριμένη διεύθυνση IP (που δείχνει τον υπολογιστή παραλήπτη) αλλά και σε μια συγκεκριμένη θύρα η οποία δείχνει την εφαρμογή ή διεργασία που θα παραλάβει τα δεδομένα.

Εδώ όμως προκύπτει το εξής πρόβλημα: Αν κάθε υπολογιστής χρησιμοποιεί διαφορετική θύρα για την εκτέλεση της ίδιας υπηρεσίας, τότε πως θα είναι εφικτή η επικοινωνία; Φανταστείτε για παράδειγμα ο υπολογιστής Α να χρησιμοποιεί την θύρα 35 για μεταφορά αρχείων και ο Β την θύρα 40. Αν ο Β στείλει δεδομένα στον Α θα χρησιμοποιεί την θύρα 40, αλλά ο Α θα τα

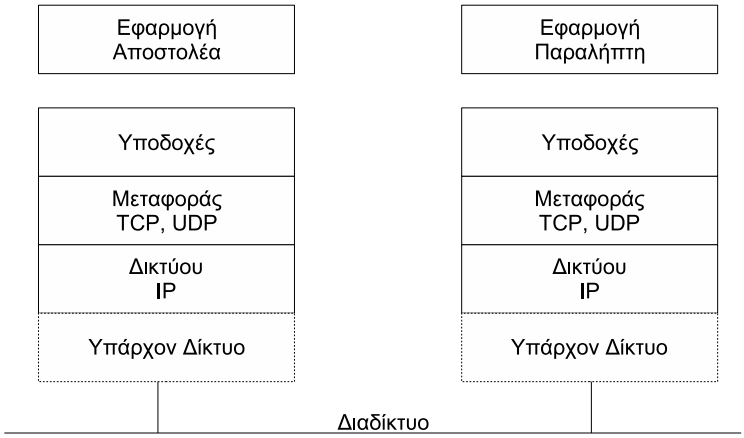
περιμένει στην 35 και η επικοινωνία δεν θα είναι δυνατή. Για να γίνει δυνατή, θα πρέπει πριν την έναρξη της μετάδοσης να υπάρξει κάποια διαπραγμάτευση μεταξύ των υπολογιστών για το ποια θύρα θα χρησιμοποιηθεί.

Για να αποφύγουμε την επιβάρυνση της πρόσθετης αυτής επικοινωνίας - διαπραγμάτευση, καθιερώθηκαν διεθνώς και έγιναν αποδεκτές κάποιες γνωστές θύρες για την εκτέλεση των πλέον κοινών δικτυακών υπηρεσιών. Οι θύρες αυτές αποτελούν τυποποιημένες που ακολουθούνται από όλα τα λειτουργικά συστήματα. Για παράδειγμα, κάθε δημοφιλή δικτυακή υπηρεσία όπως telnet, FTP, HTTP ή SMTP πραγματοποιείται μέσω των θυρών 23, 21, 80 και 25 αντίστοιχα. Οι θύρες αυτές έχουν δεσμευτεί για συγκεκριμένες χρήσεις και δεν επιτρέπεται οι προγραμματιστές εφαρμογών να τις χρησιμοποιήσουν για κάτι άλλο. Ωστόσο οι αριθμοί θυρών από το 0 έως το 1023 είναι αποκλειστικά για συγκεκριμένες υπηρεσίες, ενώ οι θύρες πάνω από το 1023 μπορούν να χρησιμοποιηθούν για οποιοδήποτε λόγο.

Ο συνδυασμός της διεύθυνσης IP του υπολογιστή στο οποίο εκτελείται το πρόγραμμα αποστολέα και του αριθμού θύρας προσδιορίζει ένα κανάλι επικοινωνίας που είναι γνωστό ως **υποδοχή** (socket) του αποστολέα και αποτελεί μια λογική έννοια. Επίσης, στην πλευρά του υπολογιστή όπου εκτελείται το πρόγραμμα παραλήπτη, η διεύθυνση IP του υπολογιστή και ο αριθμός μιας γνωστής (ή αποκλειστικής) θύρας προσδιορίζει την υποδοχή του παραλήπτη και αποτελεί πάλι μια λογική έννοια. Η επικοινωνία μεταξύ προγραμμάτων αποστολέα - παραλήπτη γίνεται από το ζεύγος αυτών των υποδοχών σε συνδυασμό με την στοίβα πρωτοκόλλων TCP/IP όπως φαίνεται στο Σχήμα 4.8.

4.6 Ενδιάμεσο Λογισμικό Χρήστη

Το ενδιάμεσο λογισμικό χρήστη παρέχει υπηρεσίες υψηλού επιπέδου στους προγραμματιστές ή χρήστες για την εύκολη ανάπτυξη εφαρμογών. Είναι όμως ξεκάθαρο, ότι τον περισσότερο καιρό ο προγραμματιστής εφαρμογών δεν πρέπει να αντιλαμβάνεται τους μηχανισμούς υλοποίησης των υπηρεσιών υποδομής του κατανεμημένου συστήματος. Ο προγραμματιστής εφαρμογών απλά πρέπει να θεωρεί ότι η ύπαρξη ενός εργαλείου ή μιας διεπαφής προγραμματισμού είναι δεδομένη και να χρησιμοποιεί τις αντίστοιχες κλήσεις συναρτήσεων μέσα στις εφαρμογές του. Οι διεπαφές προγραμματισμού παρέχουν υπηρεσίες επικοινωνίας υψηλού επιπέδου στο επίπεδο των κατανεμημένων εφαρμογών. Το υπόλοιπο μέρος της ενότητας εξετάζουμε γενικά τις υπηρεσίες



Σχήμα 4.8: Επικοινωνία αποστολέα - παραλήπτη με την χρήση υποδοχών και πρωτοκόλλων του Διαδικτύου

επικοινωνίας υψηλού επιπέδου τις οποίες μπορεί να χρησιμοποιήσει ο προγραμματιστής στις εφαρμογές τους.

4.6.1 Μεταβίβαση Μηνυμάτων

Είναι γνωστό ότι όταν μια διεργασία αποστολέας θέλει να στείλει δεδομένα σε μια άλλη διεργασία παραλήπτη, στο φυσικό επίπεδο περιλαμβάνει μετάδοση ενός ρεύματος δυφίων πάνω σε μια φυσική σύνδεση μέσω σειριακή ή παράλληλη μεταφορά δεδομένων χαμηλού επιπέδου. Αυτό το χαμηλό επίπεδο επικοινωνίας ίσως είναι κατάλληλο για συγγραφή λογισμικού ενός οδηγού δικτυακής συσκευής.

Σε ένα επόμενο επίπεδο είναι η μεταβίβαση μηνυμάτων το οποίο είναι μια αφαίρεση αυτού που πραγματικά συμβαίνει στο επίπεδο του υλισμικού και των γραμμών επικοινωνίας. Για να επικοινωνήσουν δύο διεργασίες μέσω Διαδικτύου χρησιμοποιούν ένα πρωτόκολλο επιπέδου μεταφοράς, το οποίο παρέχει ένα λογικό κανάλι επικοινωνίας (δηλαδή την υποδοχή) μεταξύ των διεργασιών διαφορετικών υπολογιστών. Συνεπώς, στην μεταβίβαση μηνυμάτων περιλαμβάνει την διεπαφή προγραμματισμού υποδοχών η οποία χρησιμοποιεί τα πρωτόκολλα μεταφοράς του TCP/IP. Χρησιμοποιώντας την διεπαφή υποδοχών, δύο διεργασίες μεταβιβάζουν δεδομένα με την χρήση μιας λογική αφαίρεση υποδοχής η οποία υπάρχει και στις δύο πλευρές. Στην περίπτωση που η μια διεργασία αποστολέας στέλνει δεδομένα στην άλλη διεργασία παραλήπτη,

πρέπει να γράψει τα δεδομένα στην υποδοχή του παραλήπτη. Στην άλλη πλευρά, όταν η διεργασία παραλήπτη παραλάβει τα δεδομένα από την άλλη, θα πρέπει να διαβάζει τα δεδομένα από την υποδοχή της. Η διεπαφή υποδοχών που υλοποιεί τις κλήσεις των υποδοχών μεταφράζει τις κλήσεις αυτές σε κατάλληλα μηνύματα του χρησιμοποιούμενου πρωτοκόλλου μεταφοράς διαφανώς προς τον προγραμματιστή. Η μόνη διάκριση που γίνεται ανάμεσα στα πρωτόκολλα είναι ανάμεσα σε πρωτόκολλα ρεύματος και πρωτοκόλλα δεδομενογραμμάτων τα οποία αντιστοιχούν σε διαφορετικούς τύπους υποδοχών. Οι υποδοχές ρεύματος υλοποιούν μέσω του TCP ενώ οι υποδοχές δεδομενογραμμάτων υλοποιούν μέσω του UDP.

Σε αυτό το βιβλίο θα ασχοληθούμε με την διεπαφή προγραμματισμού υποδοχών σε γλώσσα Java. Η διεπαφή προγραμματισμού υποδοχών Java διαθέτει κλάσεις και μεθόδους για τις υποδοχές TCP και UDP.

4.6.2 Απομακρυσμένη Κλήση Διαδικασιών και Μεθόδων

Η απομακρυσμένη κλήση διαδικασιών παρέχει μια ακόμα επιπλέον λογική αφαίρεση υψηλότερου επιπέδου η οποία επιτρέπει στις διεργασίες που εκτελούνται σε ένα υπολογιστή να καλούν διαδικασίες που εκτελούνται σε άλλες απομακρυσμένους υπολογιστές. Το σύστημα αναλαμβάνει να μεταβιβάζει παρασκηνιακά τις παραμέτρους της κλήσης ανάμεσα στις δύο διεργασίες και να επιστρέψει τα αποτελέσματα πίσω στην καλούσα διεργασία. Οι απομακρυσμένες κλήσεις είναι παρόμοιες με τις τοπικές κλήσεις διαδικασιών, αποκρύπτοντας από τον προγραμματιστή τις λεπτομέρειες της επικοινωνίας μεταξύ διαφορετικών υπολογιστών και διεργασιών.

Μια επέκταση του είναι η κλήση απομακρυσμένων μεθόδων η οποία επιτρέπει την κλήση μεθόδων αντικειμένων που βρίσκονται σε απομακρυσμένους υπολογιστές. Η διαφοροποίηση από την κλήση απομακρυσμένων διαδικασιών οφείλεται στο ότι στον προγραμματισμό με αντικείμενα οι παράμετροι μιας κλήσης μπορεί να είναι και αυτές με τη σειρά τους αντικείμενα. Αυτό το πρότυπο υποστηρίζεται από τις διεπαφές προγραμματισμού Java RMI, DCOM και CORBA.

4.6.3 Υπηρεσίες Ιστού

Οι υπηρεσίες Ιστού είναι μια επέκταση της απομακρυσμένης κλήσης μεθόδων αντικειμένων και λειτουργεί σε ακόμα διαλειτουργικό δίκτυο όπως το Διαδίκτυο. Με άλλα λόγια, οι υπηρεσίες Ιστού είναι μια τεχνολογία που επιτρέπει στις εφαρμογές να επικοινωνούν μεταξύ τους ανεξαρτήτως

πλατφόρμας και γλώσσας προγραμματισμού. Μια υπηρεσία Ιστού είναι μια διεπαφή λογισμικού που περιγράφει μια συλλογή από λειτουργίες οι οποίες μπορούν να προσεγγιστούν από το δίκτυο μέσω πρότυπων μηνυμάτων XML. Χρησιμοποιεί πρότυπα βασισμένα στη γλώσσα XML για να περιγράψει μία λειτουργία προς εκτέλεση και τα δεδομένα προς ανταλλαγή με κάποια άλλη εφαρμογή. Μια ομάδα από υπηρεσίες Ιστού οι οποίες αλληλεπιδρούν μεταξύ τους καθορίζει μια εφαρμογή υπηρεσιών Ιστού.

Οι τεχνολογίες που υποστηρίζονται για τις υπηρεσίες Ιστού είναι οι SOAP (Simple Object Access Protocol), WSDL (Web Services Defintion Language) και UDDI (Universal Discovery Description Integration) μαζί με την XML. Άλλα χαρακτηριστικά τα οποία είναι αναγκαία για μία ολοκληρωμένη κατανεμημένη τεχνολογία είναι η αξιοπιστία (reliability), η ασφάλεια (security) και οι συναλλαγές (transactions). Η παροχή αυτών των υπηρεσιών στο περιβάλλον των υπηρεσιών Ιστού είναι ευθύνη κάποιων άλλων προδιαγραφών που βρίσκονται αυτή τη στιγμή ακόμη υπό ανάπτυξη.

Κεφάλαιο 5

Πλέγμα Υπολογιστών

5.1 Πλέγματα Υπολογιστών

Σήμερα υπάρχουν ακόμη ορισμένες εφαρμογές υψηλών υπολογιστικών απαιτήσεων που απαιτούν όλο και περισσότερη υπολογιστική ισχύς πέρα από ένα συμμετρικό πολυεπεξεργαστή ή συστοιχίες υπολογιστών ή ακόμα ένας υπερυπολογιστής ειδικού σκοπού τα οποία βρίσκονται σε ένα χώρο. Αυτή η ανάγκη για μεγαλύτερη υπολογιστική ισχύς έχει συμβάλει στην δημιουργία μιας υπολογιστικής υποδομής που ενώνει κατανεμημένους πόρους ευρείας περιοχής όπως βάσεις δεδομένων, διακομιστές αποθήκευσης, παράλληλοι υπολογιστές και συστοιχίες υπολογιστών σε ένα ισχυρό και εικονικό υπολογιστή προκειμένου να λύσει προβλήματα μεγάλης κλίμακας. Αυτή η υπολογιστική υποδομή είναι γνωστή ως **πλέγμα υπολογιστών** (Grid computing). Αυτό το πλέγμα είναι ανάλογο με το πλέγμα ηλεκτρικού ρεύματος το οποίο παρέχει σταθερή, διάχυτη, αξιόπιστη και διαφανή πρόσβαση στο ηλεκτρικό ρεύμα ανεξάρτητα από την προέλευση. Στον Πίνακα 5.1 παρουσιάζεται μια σύγκριση μεταξύ του πλέγματος παροχής ρεύματος και του πλέγματος υπολογιστών. Η καινούργια αυτή προσέγγιση είναι επίσης γνωστή με διάφορα ονόματα όπως **μεταυπολογισμοί** (metacomputing), **καθολικοί υπολογισμοί** (global computing) και **υπολογισμοί Διαδικτύου** (Internet computing).

Ένα πλέγμα υπολογιστών είναι μια υποδομή υλικού και λογισμικού βασισμένη στο μοντέλο του Διαδικτύου που επιτρέπει τον διαμοιρασμό, την επιλογή και την συνάθροιση μιας ποικιλίας πόρων υψηλής απόδοσης (όπως υπερυπολογιστές, συστήματα αποθήκευσης και ειδικές συσκευές) που είναι γεωγραφικά κατανεμημένα και διαχειρίζονται από διαφορετικούς οργαν-

Πίνακας 5.1: Σύγκριση μεταξύ πλέγμα ηλεκτρικού ρεύματος και πλέγμα υπολογιστών

Πλέγμα ηλεκτρικού ρεύματος	Πλέγμα υπολογιστών
Ένας χρήστης μπορεί να συνδέσει μια συσκευή σε μια ηλεκτρική υποδοχή οπουδήποτε για να λάβει αξιόπιστη, συνεπή, φθηνή ηλεκτρική ενέργεια. Ο χρήστης δεν γνωρίζει τις λεπτομέρειες για το που και πως παράγεται η ηλεκτρική ενέργεια και πως κατανέμεται σε μια συγκεκριμένη θέση.	Ο χρήστης μπορεί να συνδέσει τον υπολογιστή του στο Διαδίκτυο και να αντλήσει υπολογιστική ή αποθηκευτική ισχύ, αξιόπιστα, φθηνά και χωρίς να απασχοληθεί με λεπτομέρειες όπως η προέλευση της ισχύος και ο τρόπος απόκτησής της.
Η υποδομή που παρέχει το ρεύμα λέγεται πλέγμα ηλεκτρικού ρεύματος. Η υποδομή αυτή συνδέει εργοστάσια ενέργειας με τα σπίτια μέσω σταθμών μετάδοσης, σταθμών ενέργειας και μετασχηματιστές.	Η υποδομή που παρέχει υπολογιστική ισχύς λέγεται πλέγμα υπολογιστών. Η υποδομή αυτή συνδέει υπολογιστικούς πόρους (όπως προσωπικούς υπολογιστές, σταθμούς εργασίας, διακομιστές και βάσεων δεδομένων) και παρέχει μηχανισμούς που είναι αναγκαίοι για την πρόσβαση των πόρων.
Το πλέγμα ρεύματος είναι διαφανές: ο χρήστης δεν χρειάζεται να γνωρίζει για το που και πως παράγεται η ηλεκτρική ενέργεια.	Το πλέγμα υπολογιστών είναι διαφανές: ο χρήστης δεν χρειάζεται να γνωρίζει ποιος υπολογιστής επεξεργάζεται την αίτηση του και που βρίσκονται τα δεδομένα που χρειάζεται. Έτσι, το λεγόμενο ενδιάμεσο λογισμικό θα αναθέσει την αίτηση του χρήστη σε κάποιο πόρο για να εκτελέσει υπολογισμούς καθώς επίσης την εύρευση και ανάκτηση των δεδομένων που χρειάζεται.
Το πλέγμα ρεύματος είναι διάχυτο: η ηλεκτρική ενέργεια είναι διαθέσιμη παντού και μπορεί να έχει πρόσβαση μέσω μιας ηλεκτρικής υποδοχής.	Το πλέγμα υπολογιστών είναι διάχυτο: οι απομακρυσμένοι υπολογιστικοί πόροι είναι προσβάσιμοι από διαφορετικές πλατφόρμες (όπως υπολογιστής γραφείου, φορητός υπολογιστής, PDAs, κινητά τηλέφωνα κλπ) και μπορεί να έχει πρόσβαση στο πλέγμα μέσω ενός περιηγητή Ιστού.
Το πλέγμα ρεύματος είναι δημόσιο: παρέχει ηλεκτρικό ρεύμα όταν ζητηθεί αλλά και χρέωση για όσο χρονικό διάστημα απαιτείται από το	Το πλέγμα υπολογιστών είναι δημόσιο: παρέχει υπολογιστική και αποθηκευτική ισχύς όταν ζητηθεί αλλά και την χρήση και χρέωση για όσο

ισμούς με σκοπό να επιλύσουν μεγάλα υπολογιστικά προβλήματα που υπάρχουν στις φυσικές επιστήμες και στο εμπόριο. Επίσης, σύμφωνα με τα τρία κριτήρια του Foster (ένας από τους ιδρυτές του πλέγματος υπολογιστών) το πλέγμα υπολογιστών είναι ένα σύστημα που πρέπει:

1. να μην υπόκειται σε κεντρικό διαχειριστικό έλεγχο των πόρων όπου οι χρήστες βρίσκονται σε διαφορετικές διαχειριστικές περιοχές.

2. να χρησιμοποιεί πρότυπα, ανοικτά και γενικού σκοπού πρωτόκολλα.

3. να παρέχει ένα μεγαλύτερο σύνολο υπηρεσιών υψηλής ποιότητας οι οποίες σχετίζονται με τον χρόνο απόκρισης, ρυθμοαπόδοση, διαθεσιμότητα και ασφάλεια.

Η χρήση του πλέγματος υπολογιστών είναι σημαντική διότι προσφέρει πολλά πλεονεκτήματα. Για παράδειγμα, τα πλέγματα

- Προσφέρουν διαμοιρασμός πόρων.
- Παρέχουν διαφανή πρόσβαση στους απομακρυσμένους πόρους.
- Επιτρέπουν την συνάθροιση πόρων από διαφορετικές τοποθεσίες κατά ζήτηση.
- Μειώνουν το χρόνο εκτέλεσης για εφαρμογές υψηλών υπολογιστικών απαιτήσεων.
- Παρέχουν πρόσβαση σε απομακρυσμένο λογισμικό και βάσεις δεδομένων.
- Λαμβάνουν τις διαφορετικές ζώνες ώρας.

5.1.1 Κατηγορίες Υπηρεσιών Πλέγματος

Ένα πλέγμα υπολογιστών μπορούμε να το δούμε σαν ένα διαφανές, ολοκληρωμένο υπολογιστικό και συνεργατικό περιβάλλον και μέσα στο πλέγμα υπάρχουν μια σειρά από δραστηριότητες υψηλού επιπέδου όπως φαίνεται στο Σχήμα 5.1. Οι πόροι του πλέγματος είναι καταχωρημένοι μέσα σε ένα ή περισσότερους **καταλόγους υπηρεσιών πλέγματος** (Grid information services). Οι χρήστες υποβάλλουν τις απαιτήσεις της εφαρμογής (όπως π.χ. τα χρονικά όρια για την χρήση των πόρων, το πόσο υπολογιστική ισχύς θα πρέπει να χρησιμοποιηθεί, χρέωση για την χρήση του πόρου κλπ) στους **μεσίτες πόρων πλέγματος** (Grid resource broker) οι οποίοι θα αναζητήσουν τους κατάλληλους πόρους κάνοντας ερωτήματα στους καταλόγους

Σχήμα 5.1: Πλέγμα υπολογιστών από άποψη υψηλού επιπέδου

υπηρεσιών και στην συνέχεια οι μεσίτες χρονοδρομολογούν τις εργασίες της εφαρμογής για εκτέλεση πάνω στους εντοπισμένους πόρους. Έπειτα, οι μεσίτες παρακολουθούν την διαδικασία επεξεργασίας ή εκτέλεσης των εργασιών μέχρι να ολοκληρωθούν. Από την σκοπιά του χρήστη, τα πλέγματα υπολογιστών μπορούν να χρησιμοποιηθούν για να προσφέρουν τους παρακάτω τύπους υπηρεσιών:

- **Υπολογιστικές υπηρεσίες** (computational services). Οι υπηρεσίες αυτές αφορά στην παροχή ασφαλών υπηρεσιών για εκτέλεση εφαρμογών σε κατανεμημένους υπολογιστικούς πόρους είτε ξεχωριστά ή συλλογικά. Οι μεσίτες πόρων παρέχουν υπηρεσίες για συλλογική χρήση των κατανεμημένων πόρων. Έτσι, ένα πλέγμα που προσφέρει υπολογιστικές υπηρεσίες ονομάζεται **υπολογιστικό πλέγμα** (computational grid).

- **Υπηρεσίες δεδομένων** (data services). Οι υπηρεσίες αυτές αφορά στην ασφαλή πρόσβαση σε κατανεμημένες βάσεις δεδομένων και την διαχείριση τους. Προκειμένου να παρέχει μια κλιμακωτή αποθήκευση και πρόσβαση σε σύνολα δεδομένων, πρέπει τα δεδομένα να αντιγραφούν ομοιότυπα, να τα καταλογογραφούν και ακόμα διαφορετικά σύνολα δεδομένων να είναι αποθηκευμένα σε διαφορετικές τοποθεσίες ώστε να προκύπτει η εικόνα της μαζικής αποθήκευσης. Η επεξεργασία των δεδομένων αναλαμβάνεται από τις υπηρεσίες υπολογιστικού πλέγματος και ένας τέτοιος συνδυασμός ονομάζεται **πλέγμα δεδομένων** (data grid). Εφαρμογές που χρειάζονται υπηρεσίες διαχείρισης, διαμοιρασμού και επεξεργασία ενός μεγάλου όγκου δεδομένων είναι ο σχεδιασμός νέων φαρμάκων

με την προσπέλαση σε κατανεμημένες χημικές βάσεις δεδομένων.

- **Υπηρεσίες εφαρμογής** (application services). Οι υπηρεσίες αυτές αφορά στην διαχείριση εφαρμογών και την διάφανη πρόσβαση σε απομακρυσμένες εφαρμογές. Οι τεχνολογίες Υπηρεσιών Ιστού (web services) αναμένονται να παίξουν ένα σημαντικό ρόλο στον ορισμό υπηρεσιών εφαρμογής. Οι υπηρεσίες εφαρμογής κατασκευάζονται πάνω στις υπολογιστικές υπηρεσίες και υπηρεσίες δεδομένων. Ένα παράδειγμα συστήματος που μπορεί να χρησιμοποιηθεί για την ανάπτυξη τέτοιων υπηρεσιών είναι το NetSolve.

- **Υπηρεσίες πληροφοριών** (information services). Οι υπηρεσίες αυτές αφορά στην εξαγωγή και παρουσίαση των δεδομένων τα οποία προκύπτουν από τις υπηρεσίες υπολογισμού, δεδομένων και εφαρμογής.

- **Υπηρεσίες γνώσης** (knowledge services). Οι υπηρεσίες αυτές αφορά στον τρόπο που η γνώση ζητείται, χρησιμοποιείται, εξάγεται, δημοσιεύεται και συντηρείται στους χρήστες προκειμένουν να υλοποιήσουν στόχους. Η γνώση είναι σε κατανοητή μορφή καθώς η πληροφορία εφαρμόζεται για να πραγματοποιήσουν ένα στόχο ή να επιλύσουν έναν πρόβλημα ή να εκτελέσουν μια απόφαση. Ένα παράδειγμα της υπηρεσίας αυτής είναι η **εξόρυξη δεδομένων** (data mining) προκειμένου να κατασκευάσουν αυτόματα μια νέα γνώση.

Επίσης υπάρχουν και άλλα είδη πλέγματος που είναι άμεσα συνδεδεμένα με τις εφαρμογές που εξυπηρετούν (π.χ. Bio-Grid, Science-Grid), το γεωγραφικό χώρο που καλύπτουν (π.χ. Hellas Grid, UK-Grid) και την τεχνολογία που χρησιμοποιούν (π.χ. Semantic Grid, P2P Grid). Συχνά ο διαχωρισμός των επιμέρους κατηγοριών είναι δύσκολος αφού οι ονομασίες και οι χαρακτηρισμοί είναι, πολλές φορές αυθαίρετοι.

5.1.2 Αρχιτεκτονική ενός Πλέγματος Υπολογιστών

Για να υλοποιηθεί μια υποδομή πλέγματος απαιτούνται έναν αριθμό υπηρεσιών όπως ασφάλεια, κατάλογος πληροφοριών, διανομή πόρων και μηχανισμούς χρέωσης σε ένα ανοικτό περιβάλλον και επίσης υπηρεσίες υψηλού επιπέδου όπως περιβάλλον ανάπτυξης εφαρμογών, διαχείριση εκτέλεσης, συνάθροιση και χρονοδρολόγηση πόρων. Τα εργαλεία λογισμικού και υπηρεσίες προκειμένου να παρέχουν στους χρήστες ένα διαφανές υπολογιστικό περιβάλλον και ενιαία

πρόσβαση πόρων σε ένα ετερογενές περιβάλλον πλέγματος είναι γνωστά ως **ενδιάμεσο λογισμικό πλέγματος** (Grid middleware).

Προκειμένου το ενδιάμεσο λογισμικό πλέγματος να παρέχει στους χρήστες ένα διαφανές υπολογιστικό περιβάλλον πρέπει να λάβουμε υπόψη ορισμένα ιδιαίτερα χαρακτηριστικά. Τα χαρακτηριστικά αυτά είναι τα εξής:

- Πολλαπλές διαχειριστικές περιοχές και αυτονομία. Οι πόροι του πλέγματος είναι γεωγραφικά κατανεμημένοι σε διάφορες διαχειριστικές περιοχές και ανήκουν από διαφορετικούς οργανισμούς. Η αυτονομία των πόρων εξαρτάται από τις τοπικές διαχειριστικές πολιτικές των οργανισμών.

- Ετερογένεια. Ένα πλέγμα είναι από τη φύση του ετερογενές. Σε ένα πλέγμα περιλαμβάνουν υπολογιστές διαφορετικής ισχύς και διαμόρφωσης οι οποίοι συνδέονται με δίκτυα διαφορετικής ταχύτητας. Επίσης, οι πόροι πλέγματος μπορεί να διαχειρίζονται από διαφορετικούς οργανισμούς οι οποίοι έχουν διαφορετικές πολιτικές διαχείρισης και εκτέλεσης εργασιών σε υπολογιστές. Επίσης, κάποιοι οργανισμοί μπορούν να αποφασίσουν την προσωρινή απομάκρυνση κάποιων πόρων πλέγματος για συντήρηση ή αναβάθμιση. Επομένως, το πλέγμα είναι ετερογενές σε όρους αρχιτεκτονικής, διαχείρισης και πρόσβασης συστημάτων.

- Κλιμάκωση. Το πλέγμα έχει την δυνατότητα να επεκταθεί και να συμπεριλάβει εκατοντάδες πόρους. Αυτό προκύπτει το πρόβλημα της ενδεχόμενης μείωση της απόδοσης καθώς αυξάνεται το μέγεθος του πλέγματος υπολογιστών. Συνεπώς, οι εφαρμογές που απαιτούν ένα μεγάλο αριθμό γεωγραφικών πόρων πρέπει να αναπτυχθούν με τέτοιο τρόπο ώστε να ανεκτικές στην καθυστέρηση και το εύρος ζώνης επικοινωνίας.

- Δυναμικό ή προσαρμοστικό. Ο αριθμός των πόρων και η συμμετοχή τους στη συνολική ισχύ του πλέγματος δεν είναι σταθερή. Επίσης, σε ένα πλέγμα η αστοχία ενός πόρου είναι κανόνας παρά εξαίρεση. Έτσι, με τόσους πολλούς πόρους που υπάρχουν σε ένα πλέγμα, η πιθανότητα κάποιος πόρος να αστοχήσει είναι υψηλή. Για αυτό το λόγο οι διαχειριστές πόρων (resource managers) ή εφαρμογές πρέπει να λάβουν υπόψη την δυναμική συμπεριφορά και να χρησιμοποιούν αποτελεσματικά τους διαθέσιμους πόρους και υπηρεσίες.

Επίσης, μέσα σε ένα περιβάλλον πλέγματος, η υποδομή του πρέπει να υποστηρίζει μια σειρά από δυνατότητες:

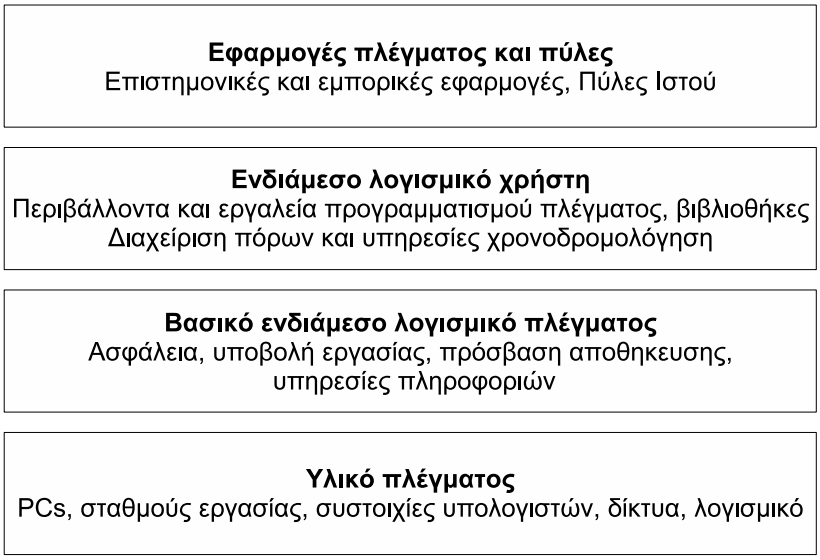
- Απομακρυσμένη αποθήκευση και/ή ομοιότυπη αντιγραφή δεδομένων.
- Δημοσίευση βάσεων δεδομένων χρησιμοποιώντας με ένα καθολικό λογικό όνομα.
- Ασφάλεια - εξουσιοδότηση πρόσβασης και ενιαία αυνθεκτικότητα.
- Ενιαία πρόσβαση σε απομακρυσμένους πόρους (δεδομένα και υπολογιστές).
- Δημοσίευση υπηρεσιών και χρέωση πρόσβασης.
- Σύνθεση κατανεμημένων εφαρμογών χρησιμοποιώντας διάσπαρτα συστατικά λογισμικού περιλαμβάνοντας κληροδοτούμενα προγράμματα.
- Εντοπισμός κατάλληλων βάσεων δεδομένων με βάση τα καθολικά τους λογικά ονόματα.
- Εντοπισμός κατάλληλων υπολογιστικών πόρων.
- Απεκόνιση και χρονοδρομολόγηση εργασιών.
- Υποβολή και παρακολούθηση της εκτέλεσης εργασιών.
- Μετανάστευση κώδικα/δεδομένων ανάμεσα στους υπολογιστές γραφείου χρηστών και κατανεμημένους πόρους.
- Μέτρηση και χρέωση για τη χρήση πόρων.

Συνπώς, τα βήματα που είναι απαραίτητα για την υλοποίηση μιας υποδομής πλέγματος που να παρέχει τις παραπάνω δυνατότητες είναι τα εξής:

- Ολοκλήρωση ξεχωριστών τμημάτων λογισμικού και υλικού σε ένα συνδυασμένο δικτυακό πόρο (όπως π.χ. η εικόνα ενός συστήματος στην περίπτωση της συστοιχίας).
- Εγκατάσταση ενδιάμεσου λογισμικού χαμηλού επιπέδου ώστε να παρέχει μια ασφαλή και διαφανή πρόσβαση στους πόρους.
- Εγκατάσταση ενδιάμεσου λογισμικού υψηλού επιπέδου ώστε να υποστηρίξει την ανάπτυξη εφαρμογών και την συνάθροιση κατανεμημένων πόρων.

- Ανάπτυξη και βελτιστοποίηση κατανεμημένων εφαρμογών που να λάβει υπόψη τους διαθέσιμους πόρους και της υποδομής του πλέγματος.

Ένα πλέγμα υπολογιστών αποτελείται από διάφορα συστατικά όπως από πόρους μέχρι τις εφαρμογές του χρήστη και τα συστατικά αυτά είναι οργανωμένα σε επίπεδα. Κάθε επίπεδο βασίζεται στις υπηρεσίες που προσφέρονται από το χαμηλότερο επίπεδο και πολλές φορές συνεργάζεται με συσταστικά του ίδιου επιπέδου. Στο Σχήμα 5.2 παρουσιάζεται η αρχιτεκτονική του πλέγματος υπολογιστών. Η αρχιτεκτονική αυτή αποτελείται από τέσσερα επίπεδα και είναι τα εξής:



Σχήμα 5.2: Αρχιτεκτονική πλέγματος υπολογιστών

- **Υλικό πλέγματος** (Grid fabric). Το επίπεδο αυτό αποτελείται όλους τους κατανεμημένους πόρους παγκοσμίως που είναι προσβάσιμοι οπουδήποτε στο Διαδίκτυο. Οι πόροι αυτοί μπορεί να είναι υπολογιστές (όπως προσωπικοί υπολογιστές ή συμμετρικοί πολυεπεξεργαστές) που εκτελούν διαφορετικά λειτουργικά συστήματα (όπως Linux ή Windows)), συστοιχίες υπολογιστών (οι οποίες φαίνονται σαν ένας πόρος στο πλέγμα), συσκευές αποθήκευσης, βάσεις δεδομένων και διακομιστές. Επίσης, στους πόρους περιλαμβάνονται ειδικά επιστημονικά όργανα όπως ραδιοτηλεσκόπιο ή αισθητήρες δικτύων οι οποίες παρέχουν δεδομένα πραγματικού χρόνου που μπορούν να μεταφερθούν άμεσα σε υπολογιστικές τοποθεσίες ή να αποθηκευτούν σε μια βάση δεδομένων. Όλοι οι παραπάνω

πόροι συνδέονται με τεχνολογίες δικτύων ευρείας περιοχής όπως στην περίπτωση των κατανεμημένων συστημάτων.

- **Βασικό ενδιάμεσο λογισμικό πλέγματος** (Core Grid middleware). Το επίπεδο αυτό προσφέρει βασικές υπηρεσίες όπως διαχείριση απομακρυσμένων διεργασιών, διανομή πόρων, πρόσβαση δεδομένων, καταχώριση (ή εγγραφή) και ανακάλυψη πληροφοριών, ασφάλεια και ποιότητα υπηρεσιών όπως οι κρατήσεις και χρεώσεις πόρων. Οι υπηρεσίες αυτές κρύβουν την πολυπλοκότητα και την ετερογένεια του υλικού πλέγματος παρέχοντας μια σταθερή και ενιαία μέθοδο πρόσβασης κατανεμημένων πόρων.

- **Ενδιάμεσο λογισμικό πλέγματος επιπέδου χρήστη** (User-level Grid middleware). Το επίπεδο αυτό χρησιμοποιεί τις διεπαφές του βασικού ενδιάμεσου λογισμικού ώστε να παρέχει υψηλότερες επιπέδου αφαιρέσεις και υπηρεσίες. Επομένως, το επίπεδο αυτό περιλαμβάνει περιβάλλοντα ανάπτυξης εφαρμογών, εργαλεία προγραμματισμού και μεσίτες πόρων για την διαχείριση πόρων και χρονοδρομολόγηση εργασιών μιας εφαρμογής για εκτέλεση πάνω στους παγκόσμιους πόρους.

- **Εφαρμογές πλέγματος και πύλες** (Grid applications and portals). Τυπικά οι εφαρμογές πλέγματος αναπτύσσονται χρησιμοποιώντας τις υπηρεσίες που παρέχονται από το ενδιάμεσο λογισμικό πλέγματος χρήστη, δηλαδή περιβάλλοντα προγραμματισμού, υπηρεσίες μεσίτες πόρων κλπ. Ένα παράδειγμα εφαρμογής είναι ένα πρόβλημα υψηλών υπολογιστικών απαιτήσεων που απαιτεί υπολογιστική ισχύς, πρόσβαση σε απομακρυσμένες βάσεις δεδομένων και ίσως χρειαστεί να επικοινωνήσει με επιστημονικά όργανα. Οι πύλες πλέγματος προσφέρουν υπηρεσίες εφαρμογών βασισμένες στον Ιστό, όπου οι χρήστες μπορούν να υποβάλλουν εργασίες σε απομακρυσμένους πόρους και να συλλέγουν τα αποτελέσματα μέσω του Διαδικτύου.

Στις επόμενες ενότητες θα παρουσιάζουμε περισσότερες λεπτομέρειες για τα επίπεδα ενδιάμεσου λογισμικού πλέγματος και ενδιάμεσου λογισμικού πλέγματος χρήστη. Για το επίπεδο υλικό πλέγματος δεν θα αναφερθούμε διότι αναφέρεται σε συστατικά υλικού (όπως υπολογιστές, συμμετρικοί πολυεπεξεργαστές, συστοιχίες υπολογιστών, δίκτυα διασύνδεσης) και σε συσταστικά λογισμικού (όπως λειτουργικά συστήματα, συστήματα διαχείρισης πόρων, πρωτόκολλα επικοινωνίας) τα οποία έχουμε παρουσιάζει εκτενώς στα προηγούμενα κεφάλαια.

5.2 Βασικό Ενδιάμεσο Λογισμικό: Globus

Για την ανάπτυξη του βασικού ενδιάμεσου λογισμικού συστήματος έχουν αναπτυχθεί πολλές ερευνητικές προσπάθειες όπως Globus, UNICORE, Legion και Gridbus. Σε αυτή την ενότητα θα περιγράψουμε το πιο δημοφιλές ενδιάμεσο λογισμικό που είναι το Globus. Το Globus είναι ένα ανοιχτό λογισμικό που μπορεί να χρησιμοποιηθεί για την κατασκευή υπολογιστικών υποδομών πλέγματος και εφαρμογών πλέγματος. Το λογισμικό αυτό επιτρέπει το ασφαλή διαμοιρασμό της υπολογιστικής δύναμης, των βάσεων δεδομένων και άλλων πόρων από εταιρικά και γεωγραφικά όρια χωρίς να καταστρέψει τη τοπική αυτονομία τους. Οι βασικές υπηρεσίες, διεπαφές και πρωτόκολλα του Globus επιτρέπουν στους χρήστες να έχουν διαφανή πρόσβαση στους απομακρυσμένους πόρους ενώ συγχρόνως διατηρείται ο τοπικός έλεγχος για το ποιος μπορεί να χρησιμοποιήσει τους πόρους και πότε. Στο Σχήμα 5.3 παρουσιάζεται η αρχιτεκτονική του λογισμικού Globus που παρέχει τέσσερις υπηρεσίες: ασφάλεια, διαχείριση πόρων, καταλόγου υπηρεσιών και διαχείριση δεδομένων.

Σχήμα 5.3: Αρχιτεκτονική Globus

Το επίπεδο τοπικών υπηρεσιών περιέχει υπηρεσίες λειτουργικών συστημάτων, υπηρεσίες δικτύων όπως TCP/IP και υπηρεσίες χρονοδρομολόγησης συστοιχιών. Το επίπεδο βασικών υπηρεσιών περιέχει τις υπηρεσίες του Globus που αναφέραμε προηγουμένως. Το επίπεδο υπηρεσιών υψηλού επιπέδου και εργαλείων περιέχει εργαλεία που ενσωματώνουν τις υπηρεσίες χαμηλού επιπέδου για υλοποίηση εφαρμογών. Οι παρακάτω ενότητες που ακολουθούν περιγράφουμε τις βασικές υπηρεσίες που παρέχει το Globus.

5.2.1 Ασφάλεια GSI

Η **υποδομή ασφάλειας πλέγματος** (Grid Security Infrastructure - GSI) παρέχει μεθόδους για αυθεντικότητα των χρηστών πλέγματος και ασφαλή επικοινωνία στο ανοικτό δίκτυο. Η υποδομή ασφάλειας είναι βασισμένη στο στρώμα ασφαλών υποδοχών (Secure Sockets Layer - SSL), στην υποδομή δημόσιου κλειδιού (Public Key Infrastructure - PKI) και στην αρχιτεκτονική πιστοποιητικού X.509. Το GSI παρέχει υπηρεσίες, πρωτόκολλα και βιβλιοθήκες για να πετύχει τους παρακάτω στόχους για την ασφάλεια πλέγματος:

- Ενιαία σιγν-ον για την χρήση υπηρεσιών πλέγματος μέσω των πιστοποιητικών χρηστών.
- Αυθεντικότητα πόρων μέσω των πιστοποιητικών ξένιων υπολογιστών.
- Κρυπτογράφηση δεδομένων.
- Έγκριση -authorization.
- Μεταβίβαση της αρχής και της εμπιστοσύνης μέσω των πληρεξούσιων και πιστοποιητικών αλυσίδας εμπιστοσύνης για τις αρχές πιστοποίησης.

Οι χρήστες αποκτούν πρόσβαση στους πόρους από τα θέματα πιστοποιητικών πλέγματος που είναι απεικονισμένα σε έναν λογαριασμό απομακρυσμένου υπολογιστή. Αυτό επίσης απαιτεί ότι η αρχή πιστοποίησης που υπέγραψε το πιστοποιητικό χρήστη να είναι έμπιστο από το απομακρυσμένο σύστημα. Οι άδειες πρόσβασης πρέπει να επιβληθούν με τον παραδοσιακό τρόπο Unix μέσω των περιορισμών του απομακρυσμένου λογαριασμού χρήστη.

Οι αρχές πιστοποίησης είναι επίσης μέρος της υλοποίησης της έννοιας των εικονικών οργανισμών. Ο χρήστης ο οποίος έχει ένα πιστοποιητικό υπογεγραμμένο από την αρχή πιστοποίησης του εικονικού οργανισμού αποκτά πρόσβαση σε πόρους που είναι αυθεντικές από την ίδια αρχή πιστοποίησης. Οι εικονικοί οργανισμοί μπορούν να συνεργαστούν μεταξύ τους αναγνωρίζοντας τις αρχές πιστοποίησης τους έτσι ώστε οι χρήστες να αποκτήσουν πρόσβαση σε πόρους ανάμεσα σε συνεργασίες. Οι μηχανισμοί αυτοί χρησιμοποιούνται σε πολλές δοκιμές πλέγματος. Ανάλογα με τη δομή της δοκιμής και τα εργαλεία που χρησιμοποιούνται, οι χρήστες μπορούν να αποκτήσουν αυτόματα πρόσβαση στους πόρους ή θα πρέπει να έρθουν σε επαφή με τους διαχειριστές συστημάτων για να εξασφαλίσουν πρόσβαση.

Οι περισσότερες υπηρεσίες απαιτούν την αμοιβαία αυθεντικότητα πριν πραγματοποιήσουν τις λειτουργίες τους. Αυτό απαιτείται συμφωνία και ασφάλεια δεδομένων και στις δύο πλευρές. Όμως, η τρέχουσα κατάσταση των εργαλείων GSI καθιστά πιθανότερο ότι μερικοί χρήστες ίσως μοιραστούν τη χρήση ενός πιστοποιητικού για να αποκτήσουν πρόσβαση στους πόρους ή μπορούν να απεικονισθούν σε ίδιο λογαριασμό του απομακρυσμένου υπολογιστή. Αυτό προκύπτει σοβαρά ερωτήματα στην αυθεντικότητα χρηστών και την εμπιστευτικότητα των δεδομένων χρηστών στο απομακρυσμένο υπολογιστή. Οι δοκιμές έχουν πολιτικές για να περιορίσουν τη συμπεριφορά αυτή.

Επίσης, η μεταβίβαση εμπιστοσύνης υλοποιείται στο Globus με την χρήση πληρεξούσιων πιστοποιητικών. Τα πληρεξούσια πιστοποιητικά είναι προσωρινά πιστοποιητικά που παράγονται δυναμικά από την ταυτότητα του χρήστη αλλά παρακάμπτει την ανάγκη για ιδιωτικά κλειδιά. Συγκεκριμένα, ένα πληρεξούσιο αποτελείται από ένα καινούργιο πιστοποιητικό το οποίο περιέχει ένα καινούργιο δημόσιο και ιδιωτικό κλειδί. Το καινούργιο πιστοποιητικό είναι υπογεγραμμένο από το ιδιοκτήτη παρά από την αρχή πιστοποίησης το οποίο δημιουργεί μια αλυσίδα εμπιστοσύνης, δηλαδή το πληρεξούσιο επιβεβαιώνεται επειδή αυτό είναι υπογεγραμμένο από το ιδιοκτήτη πιστοποιητικού και το πιστοποιητικό είναι εμπιστευμένο επειδή είναι υπογεγγραμένο από την αρχή πιστοποίησης.

5.2.2 Διαχείριση Πόρων

Το πακέτο διαχείριση πόρων επιτρέπει την διανομή πόρων μέσω της υποβολής εργασίας, μεταφοράς εκτελέσιμων αρχείων, παρακολούθηση εργασίας και συλλογή αποτελεσμάτων. Τα συστατικά Globus του πακέτου αυτού είναι:

- **Διαχειριστής Διανομής Πόρων Globus** (Globus Resource Allocation Manager - GRAM): Το GRAM παρέχει τη δυνατότητα απομακρυσμένης εκτέλεσης και παρακολούθηση για την πορεία εκτέλεσης. Το βασικό στοιχείο του GRAM είναι μια διεργασία διακομιστή ή δαίμονας που ονομάζεται gatekeeper η οποία εκτελείται στον υπολογιστή του οποίου ο χρήστης θέλει να εκτελέσει μια εργασία. Ένας πελάτης ζητά μια υποβολή εργασίας στο δαίμονα gatekeeper του απομακρυσμένου υπολογιστή. Ο δαίμονας gatekeeper ελέγχει εάν ο πελάτης είναι αυθεντικός (δηλαδή, αν το πιστοποιητικό πελάτη είναι ενεργό και υπάρχει απεικόνιση του πιστοποιητικού σε έναν οποιοδήποτε λογαριασμό σ-

το σύστημα). Μόλις τελειώσει ο έλεγχος της αυθεντικότητας, ο gatekeeper ξεκινάει το διαχειριστή εργασιών ο οποίος αρχίζει και παρακολουθεί την εκτέλεση της εργασίας. Οι διαχειριστές εργασιών δημιουργούνται ανάλογα με τον τοπικό χρονοδρομολογητή του συστήματος. Το GRAM διασυνδέει με διάφορους τοπικούς χρονοδρομολογητές όπως Portable Batch System (PBS), Load Sharing Facility (LSF) και LoadLeveler.

Οι λεπτομέρειες εργασίας προσδιορίζονται μέσω της γλώσσας προσδιορισμού πόρων Globus (Resource Specification Language - RSL), η οποία είναι μέρος του GRAM. Η RSL παρέχει σύνταξη που αποτελείται από ζευγάρια χαρακτηριστικό-τιμή για την περιγραφή πόρων που απαιτούνται για μια εργασία περιλαμβάνοντας την ελάχιστη μνήμη και αριθμός CPUs.

- **Πρόσβαση σε δευτερεύουσα αποθήκευση Globus** (Globus Access to Secondary Storage - GASS): Το GASS είναι ένας μηχανισμός αρχείου-πρόσβασης που επιτρέπει τις εφαρμογές να ανακτούνται και να ανοίγουν ή να γράφουν σε απαμακρυσμένα αρχεία. Το GASS χρησιμοποιείται για μεταφορά αρχείων εισόδου και εκτελέσιμων για μια εργασία και για την ανάκτηση εξόδου. Επίσης, χρησιμοποιείται για να έχει πρόσβαση στα πρότυπα ρεύματα εξόδου και σφαλμάτων της εργασίας. Το GASS χρησιμοποιεί ένα ασφαλές πρωτόκολλο HTTP για να διοχετεύσει δεδομένα και έχει λειτουργίες GSI για να επιβάλει άδειες πρόσβασης και για τα δεδομένα και για την αποθήκευση.

5.2.3 Υπηρεσίες Πληροφοριών

Το πακέτο υπηρεσιών πληροφοριών παρέχει στατικές και δυναμικές ιδιότητες των κόμβων που είναι συνδεδεμένοι με το πλέγμα. Το συστατικό Globus του πακέτου αυτού ονομάζεται **Υπηρεσία Παρακολούθησης και Ανακάλυψης** (Monitoring and Discovery Service - MDS).

Η MDS παρέχει υποστήριξη για δημοσίευση και αναζήτηση πληροφοριών των πόρων. Η MDS έχει μια δομή τριών επιπέδων που στο κάτω επίπεδο είναι οι **Παροχείς Πληροφοριών** (Information Providers - IPs) που συλλέγουν δεδομένα για τις ιδιότητες και την κατάσταση. Στο δεύτερο επίπεδο είναι η **Υπηρεσία Πληροφοριών Πόρων Πλέγματος** (Grid Resource Information Service - GRIS) η οποία είναι ένας δαίμονας ή διεργασία που εκτελείται σε έναν πόρο. Η GRIS αποκρίνεται στις ερωτήσεις σχετικά με τις ιδιότητες των πόρων και ενημερώνει την κρυφή μνήμη της ανά τακτικά χρονικά διαστήματα εκτελώντας αναζητήσεις σ-

τους παροχείς πληροφοριών IPs. Στο τελευταίο και υψηλότερο επίπεδο που είναι η **Υπηρεσία Ευρετηρίων Πληροφοριών Πλέγματος** (Grid Information Index Service - GIIS) η οποία ευρετηριάσει τις πληροφορίες των πόρων που παρέχονται από τα υπόλοιπα GRISs και GIISs.

Οι GRIS και GIIS εκτελούν παρασκηνιακά το πρωτόκολλο LDAP (Lightweight Directory Access Protocol) στο οποίο οι πληροφορίες αναπαριστάνονται ως μια ιεραρχία εγγραφών. Η κάθε εγγραφή αποτελείται από 0 ή περισσότερα ζευγάρια χαρακτηριστικό-τιμή. Το σύνολο IPs παρέχουν δεδομένα που αφορά τον τύπο επεξεργαστή, την αρχιτεκτονική συστημάτων, τον τύπο και έκδοση λειτουργικού συστήματος, τον αριθμό επεξεργαστών, τη διαθέσιμη μνήμη, τα επίπεδα εικονικής μνήμης και πληροφορίες συστήματος αρχείων.

5.2.4 Διαχείριση Δεδομένων

Το πακέτο διαχείριση δεδομένων παρέχει εργαλεία και βιβλιοθήκες για τη μεταφορά, την αποθήκευση και τη διαχείριση των μαζικών δεδομένων που είναι αναπόσπαστο μέρος πολλών επιστημονικών υπολογιστικών εφαρμογών. Τα στοιχεία του πακέτου αυτού είναι τα εξής:

- GridFTP: Είναι μια επέκταση του τυποποιημένου πρωτοκόλλου FTP που παρέχει ασφαλή, αποτελεσματική και αξιόπιστη μεταφορά δεδομένα μέσα στα περιβάλλοντα πλέγματος. Εκτός από τις τυποποιημένες λειτουργίες FTP, το GridFTP υποστηρίζει αυθεντική μεταφορά δεδομένων μέσω ασφάλεια GSI, μεταφορά δεδομένων τρίτων (δηλαδή, μεταφορά δεδομένων από διακομιστή σε διακομιστή), παράλληλη και μερική μεταφορά δεδομένων.

- Ομοιότυπη αντιγραφή και διαχείριση: Το συστατικό αυτό υποστηρίζει τις πολλαπλές θέσεις για το ίδιο αρχείο σε όλο το πλέγμα. Το συστατικό αυτό αποτελείται από δυο εργαλεία: ο κατάλογος ομοιότυπης αντιγραφής και το εργαλείο διαχείριση ομοιότυπης αντιγραφής. Ο κατάλογος ομοιότυπης αντιγραφής είναι ένας κατάλογος αναζήτησης που περιέχει απεικονίσεις ανάμεσα τα λογικά ονόματα αρχείων και τις φυσικές θέσεις αποθήκευσης των αντιγράφων αρχείων μέσα στο πλέγμα. Το εργαλείο διαχείρισης ομοιότυπης αντιγραφής ενσωματώνει το παραπάνω κατάλογο αναζήτησης και το GridFTP να αναπαράγει αρχεία δεδομένων σύμφωνα με τις πληροφορίες του καταλόγου. Τα παραπάνω

εργαλεία είναι σημαντικά για επιστημονικά πειράματα όπου είτε τα δεδομένα δεν μπορούν να αποθηκευτούν ολόκληρα σε μια θέση είτε όταν ένας αριθμός ομάδων που συνεργάζονται μαζί για την ανάλυση δεδομένων χρειάζονται να έχουν αποτελεσματική πρόσβαση στα αρχεία δεδομένων. Τέτοια δεδομένα θα μπορούσαν να αντιγραφούν ομοιότυπα και να μετακινηθούν πληρεσιέστερα στις διάφορες ομάδες που εκτελούν την ανάλυση.

Πρέπει να σημειώσουμε εδώ τους λόγους για την ευρεία χρήση του εργαλείου Globus. Πρώτα, γνωρίζουμε ότι το υπολογιστικό πλέγμα υποστηρίζει εφαρμογές που είναι κατασκευασμένα από διαφορετικές τεχνικές (ή μοντέλα) προγραμματισμού. Συνεπώς, αντί να υπάρχει ένα ενιαίο μοντέλο προγραμματισμού, το Globus έχει μια αντικειμενοστρεφή προσέγγιση η οποία παρέχει ένα σύνολο υπηρεσιών από τις οποίες οι προγραμματιστές επιλέγουν κάποιες από αυτές σύμφωνα με τις ανάγκες τους για την ανάπτυξη των εφαρμογών τους. Ένας τελευταίος λόγος είναι ότι το Globus είναι ένα ανοιχτό σύστημα και ανοικτού κώδικα το οποίο είναι συμβατό με την υποδομή του πλέγματος η οποία στηρίζεται σε ανοιχτά πρωτόκολλα.

5.3 Ενδιάμεσο Λογισμικό Χρήστη

Το ενδιάμεσο λογισμικό χρήστη περιλαμβάνει περιβάλλοντα προγραμματισμού και εργαλεία για την ανάπτυξη εφαρμογών σε υπολογιστικό πλέγμα. Πριν αναφερθούμε συνοπτικά τα εργαλεία προγραμματισμού που χρησιμοποιούνται για το πλέγμα, πρέπει πρώτα να αναφέρουμε ορισμένες ιδιότητες που πρέπει να διατηρεί μια εφαρμογή προκειμένουν να εκτελεστεί επιτυχώς στο υπολογιστικό πλέγμα. Συνεπώς, μια εφαρμογή πρέπει να κατέχει τις ακόλουθες ιδιότητες:

- Μεταφέρσιμη. Αυτό σημαίνει ότι η εφαρμογή πρέπει να είναι σε θέση να εκτελείται σε πολλούς διαφορετικούς τύπους συστημάτων χωρίς την ανάγκη ξαναμεταγλώττιση. Ένας όρος που μπορούμε να χρησιμοποιήσουμε για να περιγράψουμε αυτή την ιδιότητα είναι η ανεξαρτησία αρχιτεκτονικής.

- Απόδοση. Συχνά, ο κύριος λόγος για εκτέλεση εργασίας στο πλέγμα είναι για να εκτελεστεί σε λιγότερο χρόνο. Όμως, το να πετύχουμε μια υψηλή απόδοση ενώ να εκτελείται όλη η παρασκηνιακή εργασία που είναι απαραίτητη ώστε να εξασφαλιστεί η σωστή εκτέλεση του προγράμματος δεν είναι μια απλή εργασία. Έτσι, μπορούμε να πούμε ότι μια

εφαρμογή πρέπει να είναι αποδοτικά αξιόπιστη που σημαίνει ότι πρέπει να εκτελεί μια εργασία αποτελεσματικά και αξιόπιστα.

- Ανοχή σφαλμάτων και ασφάλεια. Οι εφαρμογές πλέγματος πρέπει να είναι σε θέση να ανιχνεύσουν και επανακτούνται από σφάλματα τα οποία προέρχονται από σφάλματα υπολογισμού και επικοινωνίας. Επίσης, είναι απαραίτητο να υπάρχει προστασία στα δεδομένα και στο κώδικα που εκτελείται σε διάφορους κόμβους του πλέγματος. Πρέπει να υπάρχουν μηχανισμοί ασφάλειας ώστε να παρέχουν στις εφαρμογές πελάτη και στους κόμβους του πλέγματος με ασφάλεια και ιδιωτικότητα.

Η ανάπτυξη μιας εφαρμογής πλέγματος που να κατέχει όλες τις προηγούμενες ιδιότητες φαίνεται να είναι πολύπολοκο και δύσκολο. Τα τελευταία χρόνια, η έρευνα έχει δείξει ενδιαφέροντα μοντέλα και περιβάλλοντα που απλοποιούν τον προγραμματισμό πλέγματος τα οποία είναι εύκολα προσβάσιμα στους προγραμματιστές. Το υπόλοιπο μέρος της ενότητας θα παρουσιάζουμε ορισμένα δημοφιλή περιβάλλοντα προγραμματισμού και εργαλεία που χρησιμοποιούνται για το πλέγμα υπολογιστών.

5.3.1 Μεταβίβαση Μηνυμάτων

Είναι γνωστό ότι η μεταβίβαση μηνυμάτων υποστηρίζεται από το εργαλείο MPI το οποίο χρησιμοποιείται ευρέως για προγραμματισμό σε συστοιχίες υπολογιστών. Οι εφαρμογές MPI μεταβιβάζουν δεδομένα μέσω μηνυμάτων ανάμεσα στις διεργασίες πάνω σε ένα δίκτυο Ethernet. Η διεπαφή προγραμματισμού MPI ορίζει συναρτήσεις υψηλού επιπέδου για διαχείριση διεργασιών, συναρτήσεις για σημειακή επικοινωνία ανάμεσα σε δύο διεργασίες και συναρτήσεις για συλλογική επικοινωνία ανάμεσα σε μια ομάδα διεργασιών. Όμως, η διεπαφή αυτή δεν υποστηρίζει για πλήρη προγραμματισμό σε περιβάλλον πλέγμα. Συγκεκριμένα, η διεπαφή MPI δεν υποστηρίζει για ανοχή σφαλμάτων, διαμοιρασμό και διανομή αρχείων και μηχανισμούς ασφάλειας. Όπως είπαμε αυτές οι υπηρεσίες είναι απαραίτητες για τη σωστή και αποτελεσματική εκτέλεση προγράμματος στο πλέγμα. Για αυτό το λόγο καθιερώθηκε η διεπαφή MPICH-G2 το οποίο είναι μια επέκταση του MPI κατάλληλο για πλέγμα και επιτρέπει στους χρήστες να εκτελούν προγράμματα MPI στο πλέγμα χωρίς να αλλάξουν οι εντολές. Επίσης, η MPICH-G2 παρέχει επεκτάσεις της MPICH χρησιμοποιώντας το λογισμικό του Globus, παρέχοντας στους

χρήστες που είναι εξοικοιωμένοι με το MPI ένα εύκολο τρόπο για ανάπτυξη εφαρμογών MPI για πλέγμα.

5.3.2 Κλήσεις Απομακρυσμένων Διαδικασιών

Η μέθοδος απομακρυσμένη κλήση διαδικασιών δεν είναι πολύ διαφορετική από την έννοια μεταβίβασης μηνυμάτων που είδαμε στην προηγούμενη παράγραφο. Όμως, αντί οι διεργασίες να στέλνουν μηνύματα μέσω κλήσεων συναρτήσεων βασίζονται στην κλήση διαδικασιών σε απομακρυσμένους κόμβους του πλέγματος. Αυτό το πρότυπο υποστηρίζεται από το εργαλείο GridRPC. Το GridRPC συνδυάζει το πρότυπο μοντέλο προγραμματισμού RPC με ασύγχρονη παράλληλη εργασία μεγάλους μεγέθους. Επίσης, το GridRPC παρέχει μια αφαίρεση υψηλού επιπέδου που εκτενώς αποκρύπτει την δυναμικότητα, την ανασφάλεια και την αστάθεια του πλέγματος από τους προγραμματιστές.

Στο εργαλείο GridRPC υπάρχουν τρεις τύποι οντοτήτων: πελάτης, υπηρεσία και κατάλογος υπηρεσιών. Οι πελάτες χρησιμοποιούν τις υπηρεσίες που είναι καταχωρημένες σε ένα κατάλληλο κατάλογο υπηρεσιών. Όταν ένας πελάτης RPC εκτελεί μια αναζήτηση για μια συγκεκριμένη υπηρεσία, ο κατάλογος επιστρέφει ένα χειριστής συνάρτησης ή διαδικασίας (fuction handle). Αυτός ο χειριστής αναπαριστά μια απεικόνιση από το όνομα της συνάρτησης σε ένα στιγμιότυπο της συνάρτησης αυτής σε ένα κόμβο του πλέγματος. Στην συνέχεια, ο πελάτης πραγματοποιεί μια κλήση RPC χρησιμοποιώντας το χειριστή συνάρτηση για να εκτελέσει την διαδικασία η οποία μετά την ολοκλήρωση της εκτέλεσης θα επιστρέψει το αποτέλεσμα. Ένας άλλος όρος που πρέπει να γνωρίζουμε είναι το session ID ο οποίος είναι ένας προσδιοριστής που αναπαριστά μια συγκεκριμένη μη ανασταλτική κλήση GridRPC. Τέλος, η διεπαφή προγραμματισμού GridRPC παρέχει πολλαπλούς τύπους δεδομένων και μεθόδους για αρχικοποίηση διαδικασίας, δημιουργία και τερματισμό του χειριστή διαδικασίας, κλήσεις διαδικασιών, ασύγχρονες αναμονές και αναφορές σφαλμάτων.

5.3.3 Δεξαμενή Εργασιών

Το πρότυπο δεξαμενή εργασιών περιλαμβάνει εφαρμογές οι οποίες αποτελούνται από ένα μεγάλο αριθμό ανεξάρτητων εργασιών που μπορούν να εκτελεστούν παράλληλα στο υπολογιστικό πλέγμα. Ένα παράδειγμα του προτύπου αυτού είναι οι παραμέτροι sweep, οι οποίες διανέμουν

το ίδιο πρόγραμμα σε πολλαπλούς πόρους πλέγματος για να εκτελεστούν με διαφορετικούς παραμέτρους. Τέτοιες διεργασίες ίσως απαιτούν υπηρεσίες διαμοιραζόμενων πόρων όπως βάσεις δεδομένων ή διακομιστές εξουσιοδότησης. Συνεπώς, για την υλοποίηση τέτοιων εφαρμογών με δεξαμενή εργασιών στο πλέγμα απαιτούν ένα λογισμικό όπως οι μεσίτες πόρων (resource broker). Παραδείγματα μεσίτων πόρων είναι οι Condor-G, Nimrod-G, GridBus Broker κλπ.

Οι μεσίτες πόρων είναι ένα λογισμικό που επιτρέπει στους χρήστες διαφανή πρόσβαση στους ετερογενείς πόρους πλέγματος. Επίσης, οι μεσίτες παρέχουν μια σειρά υπηρεσίες όπως ανακάλυψη πόρων, διαφανή πρόσβαση σε υπολογιστικούς πόρους, χρονοδρομολόγηση εργασιών και παρακολούθηση εργασιών. Γενικά, ένα μεσίτης πόρων ακολουθεί μια διαδικασία δυο βημάτων: Πρώτον, ο μεσίτης εκτελεί μια ανακάλυψη πόρων, δηλαδή αναγνωρίζει πόρους πλέγματος που είναι διαθέσιμοι για χρήση. Οι πόροι αυτοί ίσως αναφέρονται σε ένα προσβάσιμο κατάλογο υπηρεσιών πλέγματος ή από μια λίστα που παρέχεται από το χρήστη. Δεύτερον, επιλέγονται οι κατάλληλοι πόροι για εκτέλεση εργασιών με βάση τις απαιτήσεις της εφαρμογής και πρόσθετους εξωτερικούς περιορισμούς όπως η προσθεσμία εκτέλεσης ή το κόστος εκτέλεσης.

Ο βασικός σκοπός του δημοφιλή εργαλείου Condor ήταν η δημιουργία ενός συστήματος εκτέλεσης εργασιών σε πόρους τα οποία ήταν γεωγραφικά διάσπαρτα και τα οποία εντάσσονταν και αποσυνδέονταν από το κατανεμημένο σύστημα σε απρόβλεπτους χρόνους. Ουσιαστικά, το Condor είναι ένα σύστημα μαζικής εκτέλεσης εργασιών (batch system) το οποίο λαμβάνει κάποιες εργασίες για εκτέλεση και βάσει κάποιων αλγόριθμων δρομολόγησης και με βάσει κάποιες πολιτικές, το Condor επιλέγει σε ποιους πόρους και πότε θα εκτελεστούν οι εργασίες. Το σύστημα επίσης παρακολουθεί την πρόοδο των εργασιών και αναφέρει το πέρας των εργασιών στους χρήστες. Το Condor είναι να μπορούν να συμμετέχουν σε ένα κατανεμημένο σύστημα οποιασδήποτε μορφής πόροι, είτε πρόκειται για συστοιχίες υπολογιστών είτε για απλούς προσωπικούς υπολογιστές, οι οποίοι θα συμμετέχουν για όσο χρόνο επιθυμούν και φυσικά είναι πιθανό ότι θα είναι ετερογενείς τόσο στο υλικό όσο και στο λογισμικό που χρησιμοποιούν. Μια επέκταση του Condor για το υπολογιστικό πλέγμα είναι το Condor-G το οποίο υποστηρίζει την χρήση του λογισμικού Globus που παρέχει την υποδομή για εξουσιοδότηση, πιστοποίηση και απομακρυσμένη υποβολή εργασιών στους πόρους του πλέγματος.

5.3.4 Υπηρεσίες Πλέγματος

Στα τελευταία χρόνια, οι υπηρεσίες Ιστού έχουν μια ευρεία αποδοχή και δημοτικότητα από τους επιστήμονες ως ένα κατανεμημένο υπολογιστικό μοντέλο. Οι υπηρεσίες Ιστού βασίζονται σε ανοικτά πρότυπα Διαδικτύου όπως η XML (eXtensible Markup Language) για να περιγράψει απομακρυσμένα συστατικά λογισμικού, μεθόδους με τους οποίους προσπελάζουν εκείνα τα συστατικά και διαδικασίες με τις οποίες αναζητούνται οι μέθοδοι. Τα προσβάσιμα εκείνα συστατικά λογισμικού ονομάζονται υπηρεσίες και είναι διαθέσιμες από τους παροχείς υπηρεσιών. Γενικά, μια υπηρεσία μπορεί να οριστεί ως μια δικτυακή οντότητα που παρέχει την δυνατότητα μέσω της ανταλλαγής μηνυμάτων. Μια υπηρεσία πλέγματος είναι μια υπηρεσία Ιστού που παρέχει ένα σύνολο ορισμένων διεπαφών και οι οποίες ακολουθούν συγκεκριμένες συμβάσεις. Το μοντέλο υπηρεσιών πλέγματος βλέπει το πλέγμα ως ένα εκτεκταμένο σύνολο από υπηρεσιών πλέγματος που μπορούν να συναθροιστούν με τις ανάγκες των χρηστών δηλαδή εικονικές επιχειρήσεις και εικονικούς οργανισμούς.

Οι υπηρεσίες πλέγματος υποστηρίζονται από την προδιαγραφή OGSA (Open Grid Services Architecture) η οποία είναι ένα έργο που ο στόχος της είναι να καταστήσει διαλειτουργικότητα ανάμεσα στους ετερογενείς πόρους ευθυγραμμίζοντας ή συγκλίνοντας τις τεχνολογίες πλέγματος με την καθιερωμένη τεχνολογία υπηρεσιών Ιστού. Συγκεκριμένα, το πλαίσιο OGSA είναι μια προδιαγραφή ανοιχτής αρχιτεκτονικής που ορίζει σημασιολογικά και μηχανισμούς για την δημιουργία, αναζήτηση, προσπέλαση, χρήση, συντήρηση και καταστροφή των υπηρεσιών πλέγματος. Έτσι, ο στόχος της OGSA είναι να προτυποποιήσει όλες τις υπηρεσίες εκείνες που συναντά σε μια εφαρμογή πλέγματος (όπως οι υπηρεσίες διαχείρισης εργασιών, υπηρεσίες διαχείρισης πόρων, υπηρεσίες ασφάλειας κλπ) ορίζοντας ως ένα σύνολο διεπαφών για τις υπηρεσίες αυτές. Κάθε διεπαφή αποτελείται από ένα σύνολο ορισμών λειτουργιών οι οποίες καλούνται από μια ορισμένη ακολουθία ανταλλασόμενων μηνυμάτων.

5.3.5 Άλλα Περιβάλλοντα Προγραμματισμού

Επίσης, το πλέγμα υποστηρίζει επιπλέον περιβάλλοντα προγραμματισμού όπως κατανεμημένα αντικείμενα και κατανεμημένα νήματα. Τα κατανεμημένα αντικείμενα είναι αντικείμενα τα οποία είναι κατανεμημένα σε διάφορα πολλαπλά υπολογιστικά συστήματα που επικοινωνούν μέσω μηνυμάτων σε ένα δίκτυο επικοινωνίας. Τα κατανεμημένα αντικείμενα υποστηρίζεται από το ερ-

γαλείο ProActive το οποίο είναι μια βιβλιοθήκη Java που παρέχει μια διεπαφή προγραμματισμού για δημιουργία, εκτέλεση και διαχείριση κατανεμημένων ενεργών αντικειμένων. Επίσης, το ProActive αποτελείται από πρότυπες μόνο κλάσεις Java και δεν απαιτεί αλλαγές στην εικονική μηχανή Java (JVM) επιτρέποντας έτσι στις εφαρμογές πλέγματος να αναπτύσσονται με την χρήση κώδικα Java.

Τα κατανεμημένα νήματα είναι νήματα τα οποία εκτελούνται σε πολλαπλούς χώρους διευθύνσεων. Συνήθως, υπάρχουν δυο δημοφιλή περιβάλλοντα προγραμματισμού που υποστηρίζουν τα κατανεμημένα αντικείμενα, το Alchemi και το GTPE (Grid Thread Programming Environment). Το Alchemi είναι ένα πλαίσιο υπολογισμού πλέγματος Microsoft .NET το οποίο αποτελείται από ένα υπηρεσιοστρεφές ενδιάμεσο λογισμικό και μια διεπαφή προγραμματιστή εφαρμογών. Τα χαρακτηριστικά του Alchemi είναι ένα απλό και εύχρηστο πολυνηματικό μοντέλο προγραμματισμού. Από την άλλη μεριά, το GTPE είναι ένα περιβάλλον προγραμματισμού σε Java που αλληλεπιδρά με το μεσίτη Gridbus Broker. Το GTPE προσφέρει επιπλέον αφαίρεση της εργασίας ανάπτυξης εφαρμογής πλέγματος, την αυτοματοποίηση της διαχείρισης πλέγματος ενώ παρέχει ένα επίπεδο ψιλού κόκκου μιας λογικής ροής προγράμματος μέσω της χρήσης κατανεμημένων νημάτων.

Κεφάλαιο 6

Ομότιμα Συστήματα Υπολογιστών

Αυτό το κεφάλαιο ξεκινάει με τα κίνητρα που οδήγησαν στην δημιουργία ομότιμων συστημάτων. Στην συνέχεια δίνεται ένας πλήρης ορισμός για το ομότιμο σύστημα, τους στόχους που πρέπει να ικανοποιεί και τις διάφορες ταξινομήσεις ομότιμων συστημάτων. Επίσης, στην μέση του κεφαλαίου περιγράφει την αρχιτεκτονική υλικού και λογισμικού που συνθέτουν ένα ομότιμο σύστημα. Έπειτα περιγράφει συνοπτικά τις υπηρεσίες που προσφέρει ένα βασικό ενδιάμεσο λογισμικό συστήματος και τα εργαλεία που παρέχει το ενδιάμεσο λογισμικό εφαρμογών. Προς το τέλος του κεφαλαίου παρουσιάζει συνοπτικές πληροφορίες λειτουργία ορισμένων δημοφιλών ομότιμων συστημάτων.

6.1 Εισαγωγή

Τα τελευταία χρόνια με την ραγδαία εξάπλωση του Διαδικτύου αναφύεται μια ανάγκη για κοινή χρήση υπολογιστικών πόρων (αποθήκευση, υπολογιστική ισχύ κλπ) ή ακόμα διαμοιρασμός δεδομένων. Όπως είναι γνωστό, στα μεγάλα κατανεμημένα συστήματα επιπέδου Διαδικτύου αρχικά χρησιμοποιήθηκε το μοντέλο αρχιτεκτονικής του πελάτη - διακομιστή. Αυτό το μοντέλο υπάρχει ένα κεντρικό σύστημα το οποίο ονομάζεται διακομιστής και πολλούς πελάτες. Σε αυτό, τα δεδομένα αποθηκεύονται στους διακομιστές. Για την εύρεση κάποιας πληροφορίας ο πελάτης εντοπίζει το διακομιστή που κατέχει την επιθυμητή πληροφορία και κάνει μια αίτηση για να την αποκτήσει. Κατόπιν, ο διακομιστής ανταποκρίνεται παρέχοντας, συνήθως, την ίδια την πληροφορία. Τις περισσότερες φορές υπάρχει μικρός αριθμός διακομιστών που παρέχουν υπηρεσίες

στους υπόλοιπους πελάτες. Οι διακομιστές αυτοί είναι συνήθως διαθέσιμοι για μεγάλο χρονικό διάστημα, ενώ οι πελάτες παραμένουν συνδεδεμένοι για λίγο. Το παραπάνω κεντρικοποιημένο σύστημα αντιμετωπίζει ορισμένα βασικά προβλήματα όπως υπάρχει ένα μοναδικό σημείο αστοχίας, κατάρρευση του οποίου μπορεί να οδηγήσει σε κατάρρευση ολόκληρου του συστήματος. Επίσης, η απόδοση τους είναι μικρή όταν το πλήθος των κόμβων (ή πελατών) αυξηθεί πάρα πολύ. Τέλος, το σύστημα μπορεί να αντιμετωπίζει μονοπωλιακές καταστάσεις και άλλους νομικούς περιορισμούς.

Το παραπάνω σύστημα έχει αρχίζει να εγκαταλείπεται και συγκροτούνται συστήματα από πολλούς κατανεμημένους πόρους (δεδομένα, υπολογιστικής ισχύς, εύρος ζώνης δικτύου κλπ) οι οποίοι λειτουργούν με αποκεντρωμένο τρόπο με σκοπό την ανταλλαγή αρχείων (π.χ. μουσικά, κείμενα, κλπ), κατανεμημένο υπολογισμό ή παροχή υπηρεσιών επικοινωνίας - συνεργασίας. Τέτοια συστήματα είναι γνωστά ως **ομότιμα συστήματα** (peer-to-peer systems).

Σε ένα ομότιμο σύστημα οι κόμβοι που μετέχουν είναι ισότιμοι μεταξύ τους και δρουν και ως πελάτες και ως διακομιστές δηλαδή αποθηκεύουν, ανακτούν και συνεισφέρουν στην εύρεση πληροφοριών που αναζητούνται σε ένα δυναμικό σύστημα. Επίσης, η αποθήκευση και η διαχείριση των δεδομένων γίνεται με διαφορετικό τρόπο. Κάθε φορά που ένας νέος κόμβος εισέρχεται στο σύστημα, τα δεδομένα (ή τα αρχεία) που επιθυμεί να διαμοιραστεί είναι αμέσως διαθέσιμα στους υπόλοιπους πελάτες. Κινητήρια δύναμη για ανάπτυξη ομοτίμων εφαρμογών αποτελεί η αποκεντρωμένη και κατανεμημένη δομή τέτοιων συστημάτων που δεν απαιτούν διαχείριση και συντήρηση, οικονομικές αξιώσεις ή άλλους νομικούς περιορισμούς. Επίσης, τα ομότιμα συστήματα επιτρέπουν ένα μεγαλύτερο και διαλειτουργικό διαμοιρασμός πόρων σε χαμηλό κόστος. Οι κόμβοι προσαρμόζονται, αυτοδιοργανώνονται καθώς εισέρχονται ή αποχωρούν από το σύστημα, ικανοποιώντας την ιδιότητα της κλιμάκωσης και της ανοχής στις αστοχίες. Οι λειτουργίες του είναι κατανεμημένες στους κόμβους που μετέχουν σε ένα τέτοιο σύστημα, όπου εκατομμύρια διαφορετικοί χρήστες μπορούν να είναι παρόντες ταυτόχρονα. Όμως, στα ομότιμα συστήματα προκύπτουν ορισμένα ζητήματα ασφάλειας για τους χρήστες Γενικά, υπάρχει μια τεχνολογία και έρευνα σε εξέλιξη που αντιμετωπίζει τα ζητήματα αυτά όπως θα δούμε στην συνέχεια του κεφαλαίου αυτού.

Οι πρώτες κατανεμημένες εφαρμογές (SMTP, FTP) εμφανίζονται στα τέλη της δεκαετίας του ′80 και αρχές του ′90 αποτελούν τον πρόδρομο των ομότιμων συστημάτων. Τέλος, τα ομότιμα συστήματα κέρδιζαν έδαφος στα τέλη της δεκαετίας του 1990 με την πρώτη χαρακτηριζόμενη

ομότιμη εφαρμογή Napster για ανταλλαγή μουσικών αρχείων MP3. Με αναφορά την παραπάνω εφαρμογή εμφανίζονται την ίδια περίοδο πολλές άλλες παρόμοιες ομότιμες εφαρμογές είτε για την ανταλλαγή αρχείων (πέραν του μουσικών) είτε ακόμη και για την αξιοποίηση των χαμένων κύκλων της ΚΜΕ όπως του περίφημου έργου SETI@home. Τόσο η επιστημονική κοινότητα όσο και ο εμπορικός κόσμος δείχνουν έντονο ενδιαφέρον για τα ομότιμα συστήματα τα οποία όχι μόνο γίνονται αποδεκτά αλλά υιοθετούνται ευρέως διαμοιράζοντας αρχεία ή παρέχοντας κατανεμημένο υπολογισμό.

6.2 Ορισμός και Στόχοι

Ένας διαισθητικός ορισμός ενός ομότιμου συστήματος δίνεται από τον Clay Shirkey. ''Τα ομότιμα συστήματα είναι μια κατηγορία εφαρμογών που εκμεταλεύονται την αποθήκη πηγών, το περιεχόμενο αλλά και την ανθρώπινη παρουσία που υπάρχει στα άκρα του Διαδικτύου. Επειδή η πρόσβαση σε αυτούς τους αποκεντρωμένους πόρους συνεπάγεται λειτουργία των κόμβων σε έναν περιβάλλον ασταθούς συνδεσιμότητας και απρόβλεπτων διευθύνσεων IP, οι ομότιμοι κόμβοι πρέπει να λειτουργούν έξω από το DNS και να διαθέτουν σημαντική ή ολική αυτονομία από τους κεντρικούς διακομιστές.'' Κατά συνέπεια ένα ομότιμο σύστημα μπορεί να θεωρηθεί ως δίκτυο επιπέδου εφαρμογής πάνω από το Διαδίκτυο.

Σύμφωνα με τους ένα ομότιμο σύστημα είναι ένα σύστημα που αποτελείται από διασυνδεδεμένους κόμβους, ικανούς να αυτοδιοργανώνονται σε τοπολογίες δικτύου με σκοπό την κοινή χρήση πόρων όπως δεδομένα, κύκλους μηχανής, χώρο αποθήκευσης και εύρος ικανό να προσαρμόζεται στις αστοχίες και στις παροδικές μετακινήσεις κόμβων ενώ διατηρούν προσβάσιμη συνδετικότητα και εκτελούνται χωρίς την απαίτηση για μεσολάβηση ή υποστήριξη ενός καθολικού κεντρικού διακομιστή ή αρχής.

Όπως και σε ένα υπολογιστικό σύστημα, ο στόχος των ομότιμων συστημάτων είναι να υποστηρίζει εφαρμοργές που να ικανοποιεί τις ανάγκες και απαιτήσεις των χρηστών. Ένα ομότιμο σύστημα πρέπει να παρέχει ένα ή περισσότερους από τους παρακάτω στόχους.

- Κόστος διαμοιρασμού. Πρέπει το κόστος διαμοιρασμού να ισοκατανέμεται σε όλους τους κόμβους ενός ομότιμου συστήματος αντί να συγκεντρώνονται σε ένα κόμβο.

- Συνάθροιση πόρων και διαλειτουργικότητα. Η αποκεντρωμένη προσέγγιση του ομότι-

μου συστήματος καταλήγει στην συνάθροιση πόρων. Τα ομότιμα συστήματα πρέπει να συναθροίζουν μεγάλες ποσότητες πόρων ώστε να επιλύσουν μεγάλα απαιτητικά υπολογιστικά προβλήματα (όπως το σύστημα SETI@home) ή ακόμα να διαμοιράζουν πόρους (π.χ. δίσκους για αποθήκευση δεδομένων, εύρος ζώνης δικτύου για μεταφορά δεδομένων, κλπ) για την διεκόλυνση ανταλλαγή αρχείων (όπως το Napter). Επίσης, το ομότιμο σύστημα πρέπει να διαμοιράζει διασπαρσμένους πόρους σε διαλειτουργικό επίπεδο.

- Διοικητική αποκέντρωση. Η διοικητική αποκέντρωση είναι μια άμεση συνέπεια της συνάθροισης πόρων. Έτσι, τα μέρη του συστήματος ή ακόμα και ολόκληρου του συστήματος δεν πρέπει πλέον να διαχειρίζονται κεντρικά. Η διοικητική αποκέντρωση είναι ιδιαίτερα σημαντική προκειμένουν να αποφευχθούν αστοχίες ή συμφορήσεις απόδοσης στο σύστημα.

- Κλιμάκωση και αξιοπιστία. Το ομότιμο σύστημα πρέπει να υποστηρίζει την ιδιότητα της κλιμάκωσης και της αξιοπιστίας. Με άλλα λόγια τα ομότιμα συστήματα να είναι σε θέση να υποστηρίζουν όσο περισσότερους κόμβους, χρήστες ή ακόμα αποθηκευτικό χώρο. Η αξιοπιστία του ομότιμου συστήματος σχετίζεται με τις αστοχίες κόμβων και δικτύου, την παροδική αποσύνδεση κόμβων, την διαθεσιμότητα πόρων κλπ.

- Αυτονομία. Κάθε κόμβος ενός ομότιμου συστήματος εκτελεί εργασίες τοπικά και ανεξάρτητα εκ μέρους του χρήστη χωρίς την υποστήριξη του κεντρικού διακομιστή. Επομένως οι κόμβοι πρέπει να οργανωθούν τοπικά, μόνοι τους, βασισμένοι κυρίως σε όποιες τοπικές πληροφορίες είναι διαθέσιμες, αλληλεπιδρώντας ταυτόχρονα με τους γειτονικούς τους κόμβους.

- Ανωνυμία και ιδιωτικότητα. Μια σχετική έννοια με την αυτονομία είναι η ανωνυμία και η ιδιωτικότητα. Σε ένα ομότιμο σύστημα που οι δραστηριότητες του εκτελούνται τοπικά πρέπει οι χρήστες (ή κόμβοι) να μην παρέχουν δικές τους πληροφορίες σε άλλους χρήστες (ή κόμβους). Σε αντίθεση με τον κεντρικό διακομιστή που είναι δύσκολο να εξασφαλιστεί η ανωνυμία επειδή ο διακομιστής είναι σε θέση να αναγνωρίζει το πελάτη από την διεύθυνση του IP.

- Δυναμικό. Είναι γνωστό ότι το υπολογιστικό περιβάλλον των ομότιμων συστημάτων είναι δυναμικό. Αυτό σημαίνει ότι οι πόροι όπως οι υπολογιστικοί κόμβοι θα εισέρχονται και

θα αποχωρούν από το σύστημα συνεχώς. Συνεπώς, όταν μια εφαρμογή απαιτεί ένα υψηλό δυναμικό περιβάλλον τότε ταιριάζει η προσέγγιση των ομότιμων συστημάτων.

- Επικοινωνία ad hoc. Σχετική με την ιδιότητα δυναμικό είναι η έννοια της υποστήριξης περιβάλλοντος ad hoc. Η επικοινωνία ad hoc σημαίνει ότι τα ομότιμα συστήματα καθιστά την επικοινωνία χωρίς την υπάρχουσα υποδομή Διαδικτύου αλλά κατασκευάζει μια δική του λογική υποδομή (όπως το δίκτυο επικάλυψης). Μέσα σε αυτή την λογική υποδομή υπάρχουν αλλαγές όπου κάποιοι κόμβοι εισέρχονται και αποχωρούν.

6.3 Ταξινομήσεις

Τα ομότιμα συστήματα ταξινομούνται σε κατηγορίες σύμφωνα με τα παρακάτω κριτήρια:

- Τύπος μοντέλου: Σύμφωνα με το κριτήριο αυτό τα ομότιμα συστήματα ταξινομούνται σε δύο κατηγορίες: πραγματικά ομότιμα συστήματα και υβριδικά ομότιμα συστήματα. Στο μοντέλο πραγματικού ομότιμου συστήματος δεν υπάρχει κεντρικός διακομιστής όπως τα συστήματα Gnutella και Freenet. Στο μοντέλο υβριδικών ομότιμων συστημάτων υπάρχει ένας κεντρικός διακομιστής για να αποκτήσει και να αποθηκεύσει μεταδεδομένα όπως η ταυτότητα του κόμβου. Επίσης, στα υβριδικά συστήματα πάντα οι κόμβοι επικοινωνούν πρώτα με το κεντρικό διακομιστή και έπειτα επικοινωνούν άμεσα με άλλους ισότιμους κόμβους. Παραδείγματα του υβριδικού μοντέλου είναι τα συστήματα Napster, Groove και Magi.

- Σκοπός εφαρμογής: Υπάρχουν τρεις κατηγορίες ομότιμων εφαρμογών: επικοινωνία - συνεργασία, κατανεμημένο υπολογισμό και διαμοιρασμό αρχείων και περιεχομένου.

  – Η κατηγορία εφαρμογών επικοινωνίας - συνεργασίας περιλαμβάνει συστήματα τα οποία παρέχουν την υποδομή για την διεκόλυνση της άμεσης και πραγματικού χρόνου επικοινωνίας - συνεργασίας ανάμεσα στους ομότιμους κόμβους. Παραδείγματα της κατηγορίας αυτής είναι οι εφαρμογές chat και instant messaging όπως Chat (IRC) και Instant Messaging (ICQ, Yahoo, MSN, Jabber).

  – Η κατηγορία κατανεμημένου υπολογισμού περιλαμβάνει συστήματα τα οποία ο στόχος τους είναι να εκμεταλευτούν την διαθέσιμη υπολογιστική ισχύς ενός ομότιμου

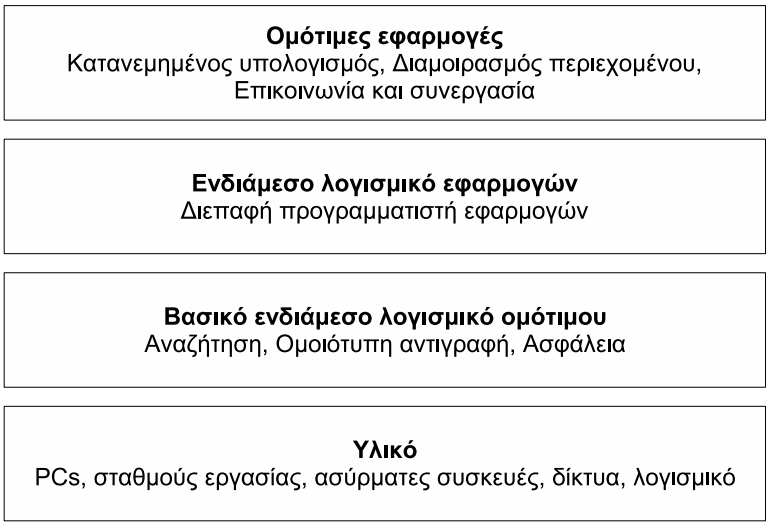
κόμβου (όπως κύκλους ΚΜΕ). Αυτό πραγματοποιείται με την διάσπαση μιας απαιτητικής υπολογιστικής εφαρμογής σε μικρότερες εργασίες και την διανομή των εργασιών αυτών σε διαφορετικούς ομότιμους κόμβους. Οι ομότιμοι κόμβοι εκτελούν την αντίστοιχη εργασία τους και επιστρέφουν τα αποτελέσματα τους. Όμως, μπορεί η ίδια εργασία να διανέμεται και να εκτελείται σε κάθε κόμβο χρησιμοποιώντας διαφορετικούς παραμέτρους. Σε αυτή την κατηγορία εφαρμογών απαιτείται να υπάρχει ένας κεντρικός συντονισμός που κυρίως αφορά στην διάσπαση και διανομή εργασιών και κατόπιν την συλλογή αποτελεσμάτων. Παραδείγματα τέτοιων συστημάτων περιλαμβάνουν τα έργα όπως το SETI@home, genome@home, Entropia, XtremWeb κλπ

– Η κατηγορία διαμοιρασμού αρχείων και περιεχομένου περιλαμβάνει συστήματα και υποδομές ειδικά για τον διαμοιρασμό ψηφιακών μέσων και άλλων δεδομένων ανάμεσα στους χρήστες. Υπάρχουν απλά και έως σύγχρονα ομότιμα συστήματα διαμοιρασμού περιεχομένου τα οποία δημιουργούν μια κατανεμημένη αποθήκευση για ασφαλή και αποτελεσματική δημοσίευση, οργάνωση, αναζήτηση, ενημέρωση και ανάκτηση δεδομένων. Υπάρχουν παρά πολλά τέτοια συστήματα και υποδομές όπως Napster, Gnutella, Limwire, Kazaa, Freenet, BitTorrent, PAST, Chord, Groove κλπ τα οποία γενικά επιτρέπουν τους ομότιμους κόμβους να αναζητήσουν και να μεταφορτώσουν αρχεία που τα διαθέτουν άλλοι κόμβοι.

6.4 Αρχιτεκτονική Ομότιμων Συστημάτων

Μέχρι στιγμής υπάρχουν αρκετές προσπάθειες για την δημιουργία ομότιμων συστημάτων. Παρόλο που δεν υπάρχει κάποιο πρότυπο που να ακολουθείται από όλους παρουσιάζουμε μια συνολική αφαιρετική αρχιτεκτονική η οποία αποτελείται από διάφορα συσταστικά για την δημιουργία ομότιμων συστημάτων. Στο Σχήμα 6.1 παρουσιάζεται η αφαιρετική αρχιτεκτονική του ομότιμου συστήματος η οποία αποτελείται από τέσσερα επίπεδα. Παρακάτω περιγράφουμε για κάθε ένα ξεχωριστά επίπεδα της αρχιτεκτονικής ομότιμου.

- Υλικό. Στο επίπεδο αυτό περιλαμβάνει μια ευρεία ποικιλία πόρων με διαφορετικές τεχνολογίες δικτύων επικοινωνίας. Από την μια μεριά περιλαμβάνει πόρους όπως οι προσωπικοί υπ-



Σχήμα 6.1: Αρχιτεκτονική ομότιμου συστήματος

ολογιστές ή σταθμοί εργασίας οι οποίοι εκτελούν διαφορετικά λειτουργικά συστήματα. Οι πόροι αυτοί συνδέονται με τις γνωστές τεχνολογίες Διαδικτύου. Από την άλλη μεριά όμως περιλαμβάνει πόρους όπως μικρές ασύρματες συσκευές (π.χ. υπολογιστές παλάμης) ή συσκευές με αισθητήρες τα οποία συνδέονται με έναν τρόπο ad hoc μέσω ασύρματων τεχνολογιών δικτύωσης. Βασική πρόκληση της επικοινωνίας στα ομότιμα συστήματα είναι να αντιμετωπίζει τα προβλήματα που σχετίζονται με την δυναμική φύση των πόρων.

- Βασικό ενδιάμεσο λογισμικό ομότιμου. Το επίπεδο αυτό παρέχει ορισμένες βασικές υπηρεσίες όπως η αναζήτηση ομότιμων κόμβων, η ομοιότυπη αντιγραφή και η ασφάλεια. Η υπηρεσία αναζήτησης ομότιμων κόμβων ασχολείται κυρίως στον εντοπισμό κόμβων και προσπαθεί να βελτιστοποιήσει την δρομολόγηση μηνυμάτων από ένα ομότιμο κόμβο σε άλλο. Η υπηρεσία ομοιότυπης αντιγραφής είναι απαραίτητη στα ομότιμα συστήματα διότι προσφέρει αξιοπιστία, διαθεσιμότητα των περιεχομένων, βελτίωση της απόδοσης και ανοχή στις αστοχίες στα διάφορα μέρη του συστήματος. Τέλος, η υπηρεσία ασφάλεια ασχολείται με θέματα όπως η ανωνυμία των χρηστών, η εμπιστοσύνη και έλεγχο πρόσβασης στα ομότιμα συστήματα. Στις επόμενες ενότητες θα παρουσιάζουμε συνοπτικά αλγορίθμους και τεχνικές που έχουν αναπτυχθεί για την παροχή των παραπάνω τριών υπηρεσιών.

- Ενδιάμεσο λογισμικό εφαρμογών. Το επίπεδο αυτό παρέχει υψηλού επιπέδου αφαιρέσεις και υπηρεσίες για την ανάπτυξη διαφόρων ομότιμων εφαρμογών. Συγκεκριμένα, το επίπεδο αυτό περιλαμβάνει εργαλεία και διεπαφές προγραμματιστή εφαρμογών.

- Ομότιμες εφαρμογές. Αναπτύσσονται ομότιμες εφαρμογές που υλοποιούν μια συγκεκριμένη λειτουργικότητα εφαρμογής οι οποίες εκτελούνται πάνω στην υποδομή ομότιμων συστημάτων. Οι εφαρμογές αυτές μπορούν να αντιστοιχούν σε κατανεμημένο υπολογισμό (π.χ. βιοτεχνολογία, επιστημονικά προβλήματα κλπ), διαμοιρασμό περιεχομένου και αρχείων (π.χ. ανταλλαγή μουσικών αρχείων κλπ) και συνεργασία - επικοινωνία (π.χ. chat, messaging κλπ).

6.5 Βασικές Υπηρεσίες του Ενδιάμεσου Λογισμικού Ομοτίμου

Στην ενότητα αυτή περιγράφουμε συνοπτικά τις τεχνικές που χρησιμοποιούνται για κάθε μια από τις υπηρεσίες του ενδιάμεσου λογισμικού όπως οι υπηρεσίες αναζήτησης, ομοιότυπης αντιγραφής και ασφάλειας.

6.5.1 Αναζήτηση

Είναι γνωστό ότι τα ομότιμα συστήματα είναι από την φύση τους κατανεμημένα και η γενική τους χρήση είναι ο διαμοιρασμός αρχείων. Το κλειδί για την χρησιμότητα τους, αλλά και μια κύρια πρόκληση από σχεδιαστική άποψη είναι η τεχνική που χρησιμοποιείται για την αναζήτηση και ανάκτηση ή μεταφορά των επιθυμητών δεδομένων. Το πρόβλημα εστιάζεται:

- Στον εντοπισμό των δεδομένων: Ο μηχανισμός που χρησιμοποιείται για το εντοπισμό του επιθυμητού δεδομένου (ή αντικειμένου).

- Στην δρομολόγηση της ερώτησης: Η στρατηγική που καθορίζει σε πόσους και ποιους γείτονες θα σταλεί μια ερώτηση που φθάνει σε έναν κόμβο και δεν μπορεί να απαντηθεί από τον ίδιο.

Η αποτελεσματικότητα της τεχνικής αναζήτησης για ένα συγκεκριμένο σύστημα εξαρτάται από τις ανάγκες της εφαρμογής. Παραδοσιακά η αναζήτηση γίνεται με βάση κάποιο κλειδί που

αντιστοιχεί στο προς αναζήτηση δεδομένο ή σε μια κανονική έκφραση. Πρόσφατες έρευνες προτάσσουν τεχνικές για υποστήριξη πολύπλοκων αναζητήσεων που αφορούν λέξεις κλειδιά για μεμονωμένα δεδομένα, περιοχή δεδομένων (range queries) ή περισσότερα από ένα χαρακτηριστικά (multi-attribute queries).

Πριν αναφερθούμε τις τεχνικές αναζήτησης και δρομολόγησης αξίζουμε να σημειώσουμε ότι οι κόμβοι (πρόκειται για προσωπικούς υπολογιστές, σταθμούς εργασίας, κλπ.) που μετέχουν σε ένα ομότιμο σύστημα σχηματίζουν ένα **δίκτυο επικάλυψης** (overlay network) πάνω από την υπάρχουσα υποδομή του Διαδικτύου. Διασυνδέονται, επικοινωνούν και ανταλλάσσουν πληροφορίες μεταξύ τους σε τοπολογίες ανεξάρτητα από το δίκτυο υποδομής Διαδικτύου διατηρώντας την αυτονομία τους. Συνεπώς, οι τεχνικές που χρησιμοποιούνται για την αναζήτηση και την ανάκτηση της πληροφορίας είναι κρίσιμος παράγοντας στα ομότιμα συστήματα αφού επηρεάζει την αποτελεσματικότητα, την απόδοση, την ικανότητα κλιμάκωσης, την ανοχή και την προσαρμοστικότητα σε αστοχίες, την αυτό-διατήρηση (nodes join and leave) και εξαρτάται από την τοπολογία του δικτύου επικάλυψης, την δόμησή του και την αρχιτεκτονική του. Το δίκτυο επικάλυψης μπορεί να είναι κεντρικοποιημένο, ιεραρχικό και αποκεντρωμένο. Στο κεντρικοποιημένο σύστημα υπάρχει ένας κεντρικός διακομιστής στον οποίο απευθύνουν οι κόμβοι τις ερωτήσεις τους για να πληροφορηθούν που βρίσκονται οι επιθυμητές πληροφορίες. Στο ιεραρχικό σύστημα οι κόμβοι οργανώνονται σε ιεραρχική δομή όπως γίνεται στην περίπτωση του DNS στο Διαδίκτυο. Στα ιεραρχικά ομότιμα συστήματα εισάγεται η έννοια των υπερκόμβων (superpeers) που η δομή τους μπορεί να είναι κεντρικοποιημένη ή μη. Στο αποκεντρωμένο σύστημα όπου οι κόμβοι συγκροτούν το δίκτυο επικάλυψης είτε δομημένα ακολουθώντας κανόνες για τον σχηματισμό του δικτύου, είτε αδόμητα όπου δεν υπάρχει ούτε κεντρικό κατάλογο ούτε ακριβείς οδηγίες για τον σχηματισμό τοπολογίας του δικτύου και την τοποθέτηση των περιεχομένων. Στην επόμενη ενότητα θα περιγράψουμε ενδεικτικά ορισμένα ευρέως γνωστά ομότιμα συστήματα όπως το Napster, η Gnutella, το CAN (Content Addressable Network) και το Chord. Το πρώτο είναι ένα κεντρικοποιημένο σύστημα, το δεύτερο αντιπροσωπεύει ένα αποκεντρωμένο αδόμητο σύστημα και τα δύο τελευταία είναι αποκεντρωμένα δομημένα συστήματα.

Υπάρχουν τέσσερις τεχνικές αναζήτησης και δρομολόγησης πάνω στα κεντρικοποιημένα, ιεραρχικά και αποκεντρώμενα ομότιμα συστήματα. Η πρώτη τεχνική είναι η κεντρικοποιημένη αναζήτηση. Σε αυτή υπάρχει μια κεντρική υπηρεσία καταλόγου που διατηρεί πληροφορίες τόσο για τα δεδομένα όσο και για τους κόμβους που τα διαθέτουν. Με άλλα λόγια σε μια κεντρική

βάση δεδομένων καταχωρούνται μια τιμή κλειδί για το στοιχείο (π.χ. τίτλος αρχείου) και η θέση (ο κόμβος) που το στοιχείο βρίσκεται αποθηκευμένο. Ένας ομότιμος κόμβος εκ μέρους του πελάτη στέλνει μια ερώτηση (π.χ. που μπορώ να βρώ το αρχείο) στην κεντρική υπηρεσία καταλόγου όπου αναζητά στην βάση της δεδομένων και επιστρέφει την θέση του κόμβου που κατέχει το αρχείο. Στην συνέχεια, οι δύο εμπλεκόμενοι κόμβοι επικοινωνούν μεταξύ τους απευθείας για την ανάκτηση της πληροφορίας. Αν η θέση είναι διαθέσιμη σε περισσότερες πηγές τότε επιλέγεται η ''καταλληλότερη'' ως προς το κόστος, την ταχύτητα, την διαθεσιμότητα σύμφωνα με τις ανάγκες του χρήστη. Ένα παράδειγμα ομότιμου συστήματος που ακολουθεί την παραπάνω τεχνική είναι το Napster. Η αναζήτηση με τον τρόπο αυτό είναι ταχύτατη - επιτυγχάνεται σε ένα βήμα - όμως δεν παύει η κεντρικοποιημένη δομή της, να αποτελεί περιορισμό στην κλιμάκωση (όταν αυξάνεται ο αριθμός των ερωτήσεων και χρηστών) και κεντρικό σημείο αποτυχίας για τα συστήματα αυτά.

Η δεύτερη τεχνική είναι η ιεραρχική αναζήτηση και βασίζεται στο παραδοσιακό σχήμα που χρησιμοποιείται στο Διαδίκτυο για τον εντοπισμό μιας διεύθυνσης IP στο Διαδίκτυο με την χρήση του DNS. Στα ομότιμα συστήματα ειδικοί κόμβοι, οι υπερκόμβοι, απαρτίζουν μια ιεραρχική δομή. Η αναζήτηση αρχίζει από την κορυφή της ιεραρχίας και διασχίζει ένα μονοπάτι μέχρι τον κόμβο που περιέχει την επιθυμητή πληροφορία. Όμως δεν υπάρχει καμιά εγγύηση ότι η πληροφορία θα βρεθεί. Παραδείγματα ομότιμων συστημάτων που ακολουθούν την ιεραρχική αναζήτηση είναι οι KaZaA, FastTrack, κλπ. Η αναζήτηση με αυτό τον τρόπο είναι ταχύτατη αλλά μια αποτυχία ή μια αποχώρηση ενός κόμβου μπορεί να δημιουργήσει σοβαρά προβλήματα ιδιαίτερα αν βρίκεται ψηλά στην ιεραρχία.

Η τρίτη τεχνική είναι η αναζήτηση πλημμύρας που εφαρμόζεται στα αποκεντρωμένα αδόμητα συστήματα που δεν υπάρχει καμιά δομή στο δίκτυο επικάλυψης και τα περιεχόμενα τοποθετούνται σε κόμβους στο δίκτυο χωρίς γνώση της τοπολογίας. Σύμφωνα με την μέθοδο της πλημμύρας, όταν ένας κόμβος ζητάει ένα δεδομένο τότε εκπέμπει μια ερώτηση στους άμεσα συνδεδεμένους γείτονες κόμβους. Στην συνέχεια, οι κόμβοι κοιτούν πρώτα την τοπική του βάση δεδομένων και αν το βρουν επιστρέφουν το δεδομένο, διαφορετικά προωθούν την ερώτηση στους γείτονες του. Αυτή η μέθοδος αναζήτησης δεν εγγυάται ότι το δεδομένο θα βρεθεί και σπαταλά πόρους τους συστήματος (εύρος ζώνης, υπολογιστική ισχύς κύκλους μηχανής, κλπ.). Πρέπει δε να υπάρχουν μηχανισμοί για την αποφυγή κύκλων, και το σταμάτημα της αναζήτησης μετά από κάποιο μέγιστο αριθμό βημάτων. Οι λύσεις που προτείνονται για το πρόωρο τερμα-

τισμό της αναζήτησης είναι η χρήση της παραμέτρου TTL (Time to Live) και Expanding Ring. Η παράμετρος TTL τίθεται σε μια τιμή αρχικά, συνήθως 9, και μειώνεται κατά ένα καθώς περνά η ερώτηση από κόμβο σε κόμβο. Η διάδοση των μηνυμάτων σταματά όταν αυτή η τιμή μηδενιστεί. Εναλλακτικά, αντί για προκαθορισμένη τιμή της παραμέτρου TTL, επιλέγεται η διαδοχική αύξηση της εφόσον δεν βρεθεί η ζητούμενη πληροφορία. Η μέθοδος αυτή είναι γνωστή ως Expanding Ring. Αρχικά, η αναζήτηση ξεκινά με ένα μικρό TTL. Αν δεν βρεθεί αποτέλεσμα, αρχίζει νέα αναζήτηση με αυξημένο TTL. Η διαδικασία επαναλαμβάνεται έως ότου βρεθεί το αποτέλεσμα. Παραδείγματα τέτοιων ομοτίμων συστημάτων είναι Gnutella κλπ.

Η τέταρτη τεχνική είναι η πληροφορημένη αναζήτηση που εφαρμόζεται σε αποκεντρωμένα δομημένα συστήματα που οι κόμβοι του διατηρούν τοπικά ή κατανεμημένα και σχηματίζουν ένα είδος δομής για το δίκτυο επικάλυψης. Η δομή αυτή χρησιμοποιείται για την δρομολόγηση των ερωτήσεων ώστε να αποφευχθούν τα μειονεκτήματα της πλημμύρας. Στα δομημένα συστήματα, η θέση των κόμβων και των δεδομένων πάνω στο σύστημα είναι συγκεκριμένη και ο προσδιορισμός της γίνεται με βάση ένα κατανεμημένο πίνακα κατακερματισμού (Distributed Hash Table - DHT). Τόσο οι κόμβοι όσο και τα δεδομένα πρέπει να έχουν ένα μοναδικό αναγνωριστικό ID. Το ID του κόμβου κατακερματίζεται από την IP του, και το ID του δεδομένου κατακερματίζεται από το όνομα του και έτσι προκύπτει η θέση τους πάνω στο σύστημα. Ο κόμβος που επιθυμεί να αποθηκεύσει ένα δεδομένο στο σύστημα εφαρμόζει μια συνάρτηση κατακερματισμού στο όνομα του δεδομένου και προκύπτει το αναγνωριστικό του. Το νέο δεδομένο αντιστοιχίζεται (και αποθηκεύεται) στον κόμβο που έχει αναγνωριστικό ID ίσο με το ID του δεδομένου. Κατά την διαδικασία της αναζήτησης, η θέση του επιθυμητού δεδομένου εντοπίζεται με μια συνάρτηση κατακερματισμού και η ερώτηση προωθείται στους κόμβους που τα αναγνωριστικά ID τους είναι πιο κοντά στο ID του δεδομένου. Τέλος, δρομολογείται η ανάκτηση του ζητούμενου δεδομένου. Η πληροφορημένη αναζήτηση έχει ένα πρόβλημα που απαιτεί τα αναγνωριστικά των δεδομένων να είναι γνωστά εκ των προτέρων πριν την δρομολόγηση της ερώτησης για ένα δεδομένο. Επίσης, υπάρχουν τέσσεροι βασικοί αλγόριθμοι που υλοποιούν την τεχνική αυτή: Chord, CAN, Taperstry και Pastry.

6.5.2 Ομοιότυπη Αντιγραφή

Για να αυξήσουμε την διαθεσιμότητα των περιεχομένων, για να βελτιώσουμε την απόδοση έτσι ώστε οι αναζητήσεις να πραγματοποιούνται αποδοτικότερα και για να βελτιώσουμε την ανοχή του ομότιμου συστήματος σε αστοχίες δημιουργούμε ομοιότυπα αντίγραφα της πληροφορίας. Αντίγραφα των αντικειμένων τοποθετούνται σε περισσότερους κόμβους έτσι ώστε οι αιτήσεις να ικανοποιούνται αποτελεσματικότερα με το μικρότερο κόστος (path length, latency, κλπ.) ή ακόμη και για να εντοπίζονται σπάνια δεδομένα. Άμεσα σχετιζόμενο θέμα που προκύπτει με την διατήρηση αντιγράφων είναι το πρόβλημα της ενημέρωσης για την διατήρηση της συνέπειας των δεδομένων. Όπως αναλύεται στη συνέχεια, κλασσικές προσεγγίσεις ή νέες προτάσεις διαχειρίζονται το ζήτημα αυτό.

Υπάρχουν δύο μηχανισμοί για την δημιουργία αντιγράφων, replication και caching. Αν και ο στόχος τους είναι κοινός ως προς την αποδοτικότητα του συστήματος και την διαθεσιμότητα των δεδομένων έχουν μια σημαντική διαφορά. Τα αντικείμενα που αποθηκεύονται με τον μηχανισμό replication αποθηκεύονται στον κόμβο σε μια βοηθητική μνήμη, ενώ με τον μηχανισμό caching αποθηκεύονται σε κάποια προσωρινή μνήμη. Συνεπώς, ανάλογα με το πού τοποθετούνται τα αντίγραφα διακρίνουμε τους μηχανισμούς:

- Caching: Δημιουργία των αντιγράφων από τους ομότιμους κόμβους καθώς κατεβάζουν δεδομένα στο δίσκο τους για να αυξήσουν την διαθεσιμότητα, ακόμη και όταν πολλοί προσπελαύνουν τα ίδια περιεχόμενα από διαφορετικές πηγές. Η μέθοδος αυτή αναφέρεται και ως παθητική αντιγραφή (passive replication). Τα δεδομένα αποθηκεύονται προσωρινά στην προσωρινή μνήμη και δεν υπάρχει εγγύηση για το χρονικό διάστημα που θα παραμείνουν εκεί, αφού μπορεί να αντικατασταθούν από πιο πρόσφατα δεδομένα από λειτουργίες ανάκτησης.

- Replication: Τα δεδομένα αντιγράφονται σε ένα κόμβο ακόμη και αν ο ίδιος δεν τα έχει ζητήσει. Μια στρατηγική που χρησιμοποιείται από την Gnutella προτείνει την owner replication όπου μόνο ο κόμβος που αναζητά το αντικείμενο το αποθηκεύει. Μια άλλη στρατηγική που χρησιμοποιείται από συστήματα όπως το Freenet προτείνει την path replication όπου όταν γίνει επιτυχή αναζήτηση, οι ενδιάμεσοι κόμβοι κατά μήκος του μονοπατιού (κόμβος αίτησης - κόμβος απόκρισης) διατηρούν αντίγραφο του δεδομένου.

Ανάλογα με τον αριθμό των αντιγράφων που επιθυμούμε, διακρίνουμε τις μεθόδους:

- Uniform replication: Δημιουργούμε τον ίδιο αριθμό αντιγράφων για κάθε δεδομένο του συστήματος. Πρόκειται για την απλούστερη τεχνική. Προφανώς, το γεγονός ότι δημιουργούμε αντίγραφα ακόμη και για δεδομένα που αναζητούνται σπάνια την καθιστά μη αποδοτική.

- Proportional replication: Για κάθε δεδομένο δημιουργείται αριθμός αντιγράφων ανάλογος της πιθανότητας αναζήτησης του. Με άλλα λόγια, όσο πιο δημοφιλές είναι ένα δεδομένο, τόσο πιο πολλά αντίγραφα του δημιουργούνται με αποτέλεσμα να πραγματοποιούνται αποδοτικότερα οι αναζητήσεις. Η μέθοδος αυτή καθιστά την αναζήτηση ευκολότερη για δημοφιλή δεδομένα και δυσκολότερη για λιγότερο δημοφιλή, αλλά και οδηγεί σε εξισορρόπηση φορτίου.

- Square-root replication: Για κάθε δεδομένο δημιουργείται αριθμός αντιγράφων ανάλογος της τετραγωνικής ρίζας της πιθανότητας αναζήτησης του. Αποδίδει καλύτερα από την παραπάνω μέθοδο, αφού το μέγεθος της αναζήτησης γίνεται βέλτιστο.

Για τον υπολογισμό καλύτερων εκτιμήσεων των πιθανοτήτων αναζήτησης δεδομένων μπορεί να χρησιμοποιηθεί η μέθοδος Replication with probe memory: οι κόμβοι διατηρούν το συνολικό αριθμό πρόσφατων ερωτήσεων για κάθε δεδομένο σε ολόκληρο το μονοπάτι αναζήτησης και ανάλογα με αυτόν υπολογίζουν καλύτερες εκτιμήσεις των πιθανοτήτων αναζήτησης. Τέλος, σε κάθε σύστημα είτε δομημένο είτε αδόμητο επιλέγει τη κατάλληλη στρατηγική δημιουργίας αντιγράφων[].

Ένα σημαντικό πρόβλημα που ανακύπτει με την χρήση αντιγράφων είναι η διαδικασία της ενημέρωσης, ο συγχρονισμός και η διατήρηση της συνέπειας αυτών. Έτσι, στα περισσότερα συστήματα που τα δεδομένα σπάνια αλλάζουν, δεν είναι απαραίτητη η συχνή ενημέρωση τους. Υπάρχουν, όμως, και εφαρμογές όπου τα δεδομένα τροποποιούνται συχνά, καθώς το σύστημα αλλάζει δυναμικά με την πάροδο του χρόνου, χωρίς οι κόμβοι να μπορούν να έχουν μια συνολική εικόνα του συστήματος. Στην περίπτωση αυτή, τα δεδομένα και τα αντίγραφά τους πρέπει να διατηρούνται ενημερωμένα για να παρέχονται οι σωστές πληροφορίες στους χρήστες. Επομένως, υπάρχουν διάφορες τεχνικές ενημέρωσης αντιγράφων σε αδόμητα και δομημένα συστήματα[].

6.5.3 Ασφάλεια

Σε αυτή την ενότητα αναπτύσσονται θέματα σχετικά με την ασφάλεια, ανωνυμία, εμπιστοσύνη και έλεγχο πρόσβασης στα ομότιμα συστήματα.

Η ασφάλεια είναι ιδιαίτερη απαίτηση στα ομότιμα συστήματα, καθώς υπάρχει ανάγκη για διαθεσιμότητα, μυστικότητα, εμπιστοσύνη, ακεραιότητα και αυθεντικότητα. Εξ αιτίας της αυτονομίας και της ανοικτής δομής τους τα συστήματα αυτά είναι ιδιαίτερα ευάλωτα σε επιθέσεις.

Το ενδιαφέρον επικεντρώνεται στην ασφαλή αποθήκευση δεδομένων και στην ασφαλή δρομολόγηση. Η ασφαλή αποθήκευση γίνεται με την χρήση παραδοσιακών και νέων πρωτοκόλλων - αλγορίθμων κρυπτογραφίας (ψηφιακές υπογραφές, multi-key encryption, τείχους προστασίας (firewalls)) με σκοπό να χτισθεί εμπιστοσύνη μεταξύ των κόμβων και των κοινόχρηστων αντικειμένων. Η ασφαλή δρομολόγηση επιτυγχάνεται με τεχνικές ελέγχου αυθεντικότητας και ακεραιότητας, την ασφαλή αντιστοίχηση ID, κλπ.

Το χαρακτηριστικό της ανωνυμίας επιτρέπει στους χρήστες να χρησιμοποιούν το ομότιμο σύστημα διατηρώντας όμως την ανωνυμία τους, αποφεύγοντας έτσι τυχόν νομικούς περιορισμούς (πρόβλημα που αντιμετώπισε το Napster). Σύμφωνα με τον Diegledine (et al. 2000) η ανωνυμία αναφέρεται στο συγγραφέα (ή εκδότη), στην ταυτότητα του κόμβου και του αντικειμένου και στις λεπτομέρειες της ερώτησης. Για την επίτευξη της ανωνυμίας χρησιμοποιούνται τεχνικές όπως πολυεκπομπή, Covert paths, ανώνυμες συνδέσεις κλπ.

Ο έλεγχος πρόσβασης, η πιστοποίηση, η διαχείριση ταυτοτήτων είναι θέματα που δεν έχουν δοθεί ιδιαίτερη σημασία. Αφού το περιβάλλον είναι κατανεμημένο είναι δυνατό οι ίδιες φυσικές οντότητες να εμφανίζονται με διαφορετικές ταυτότητες. Τελικά ο έλεγχος πρόσβασης και η πιστοποίηση προκύπτει από την κατανεμημένη δομή όπου η ευθύνη και ο έλεγχος περνά στους κόμβους.

6.6 Ενδιάμεσο Λογισμικό Εφαρμογών

Σε αυτή την ενότητα παρουσιάζουμε ορισμένα γνωστά εργαλεία τα οποία χρησιμοποιούνται για την ανάπτυξη ομότιμων υπηρεσιών ή εφαρμογών και εφαρμογές για κατανεμημένο υπολογισμό.

6.6.1 Ομότιμες Υπηρεσίες

Σχεδιάστηκε μια οικογένεια πρωτοκόλλων ειδικά για ομότιμες υπηρεσίες ή υπολογισμούς που είναι το λογισμικό JXTA. Το λογισμικό JXTA είναι ένα σύνολο ανοικτών και γενικευμένων πρωτοκόλλων ομότιμων, που ορίζονται ως μηνύματα XML, που επιτρέπει οποιαδήποτε συνδεδεμένη συσκευή στο δίκτυο από τηλέφωνα και ασύρματες PDAs μέχρι υπολογιστές και διακομιστές να επικοινωνήσει και να συνεργαστεί με έναν ομότιμο τρόπο. Χρησιμοποιώντας τα πρωτόκολλα JXTA, οι κόμβοι μπορούν να συνεργαστούν για να σχηματίσουν αυτο-διαμορφωμένες ομάδες ομοτίμων ή αλλιώς δικτύων επικάλυψης ανεξάρτητα από τις θέσεις τους στο δίκτυο (αντιπυρικές ζώνες) και χωρίς να απαιτείται υποδομή κεντρικής διαχείρισης. Οι κόμβοι χρησιμοποιούν τα πρωτόκολλα JXTA για να διαφημίσουν τους πόρους τους και για να ανακαλύψουν δικτυακούς πόρους (υπηρεσίες, σωληνώσεις, κ.λπ.) που είναι διαθέσιμες από άλλους κόμβους. Οι κόμβοι σχηματίζουν και ενώνουν ομάδες ομοτίμων για να δημιουργήσουν ειδικές σχέσεις. Οι κόμβοι συνεργάζονται για να δρομολογήσουν μηνύματα, επιτρέποντας με αυτό το τρόπο μια πλήρη συνδεσιμότητα των κόμβων. Τα πρωτόκολλα JXTA επιτρέπουν στους κόμβους να επικοινωνούν χωρίς να απαιτείται την κατανόηση ή διαχείριση ενδεχόμενων πολύπλοκων και δυναμικών τοπολογιών δικτύου. Αυτά τα χαρακτηριστικά συνθέτουν το JXTA ένα πρότυπο για υλοποίηση ομότιμων υπηρεσιών και εφαρμογών.

6.6.2 Κατανεμημένο Υπολογισμό

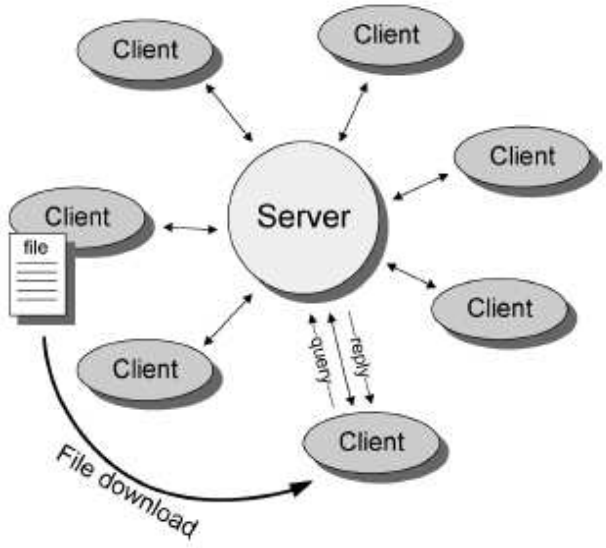
Υπάρχουν εργαλεία και πλατφόρμες οι οποίες δίνουν την δυνατότητα στους προγραμματιστές ή στους επιστήμονες να χρησιμοποιήσουν εύκολα τους διάσπαρτους και αδρανείς υπολογιστικούς πόρους για να επιλύσουν διάφορα υπολογιστικά προβλήματα μεγάλης κλίμακας. Τέτοια εργαλεία είναι το BOINC, XtremWeb, κλπ

6.7 Παραδείγματα Ομοτίμων Συστημάτων

Στην ενότητα αυτή περιγράφουμε συνοπτικά την λειτουργία ορισμένων γνωστών ομοτίμων συστημάτων όπως το Napster, η Gnutella, το CAN (Content Addressable Network) και το Chord. Το πρώτο είναι ένα κεντρικοποιημένο σύστημα, το δεύτερο αντιπροσωπεύει ένα αποκεντρωμένο αδόμητο σύστημα και τα δύο τελευταία είναι αποκεντρωμένα δομημένα συστήματα.

6.7.1 Σύστημα Napster

Στις αρχές του 1999, ο Shawn Fanning, φοιτητής στο Northeastern University των Η.Π.Α., δημιούργησε το Napster, που έμελλε να αποτελέσει κινητήριο μοχλό της εξέλιξης του φαινόμενου που ονομάζεται συστήματα ομότιμων κόμβων και χρησιμοποιήθηκε για ανταλλαγή μουσικών αρχείων. Το Napster παρουσιάζει μια κεντρικοποιημένη αρχιτεκτονική και η φιλοσοφία της λειτουργίας του είναι απλή. Σε έναν κεντρικό διακομιστή φιλοξενούνταν μια βάση δεδομένων με καταχωρίσεις ευρετηρίου για την θέση αποθήκευσης των μουσικών αρχείων. Ο χρήστης απεύθυνε εκεί την ερώτηση για το αρχείο που επιθυμούσε, ο διακομιστής του απαντούσε που θα βρει αντίγραφο και στη συνέχεια, ο χρήστης το κατέβαζε απευθείας από τον κόμβο που διατηρούσε αντίγραφο. Για την παραπάνω λειτουργία του Napster φαίνεται στο Σχήμα 6.2 Νομικοί περι-



Σχήμα 6.2: Παράδειγμα μηχανισμού αναζήτησης του Napster

ορισμοί επέβαλαν το τερματισμό λειτουργίας του Napster, όμως το πνεύμα του διαχέεται στα μετέπειτα ομότιμα συστήματα.

6.7.2 Σύστημα Gnutella

Η Gnutella είναι ένα από τα πιο δημοφιλή ομότιμα δίκτυα για διαμοιρασμό αρχείων. Η λειτουργία του στηρίζεται σε ένα μικρό πρόγραμμα που οι κόμβοι εγκαθιστούν για να έχουν την δυνατότητα δικτύωσης και που ουσιαστικά αποτελεί το πρωτόκολλο βάση του οποίου γίνεται η ανταλλαγή

των δεδομένων από κόμβο σε κόμβο (ή από υπολογιστή σε υπολογιστή).

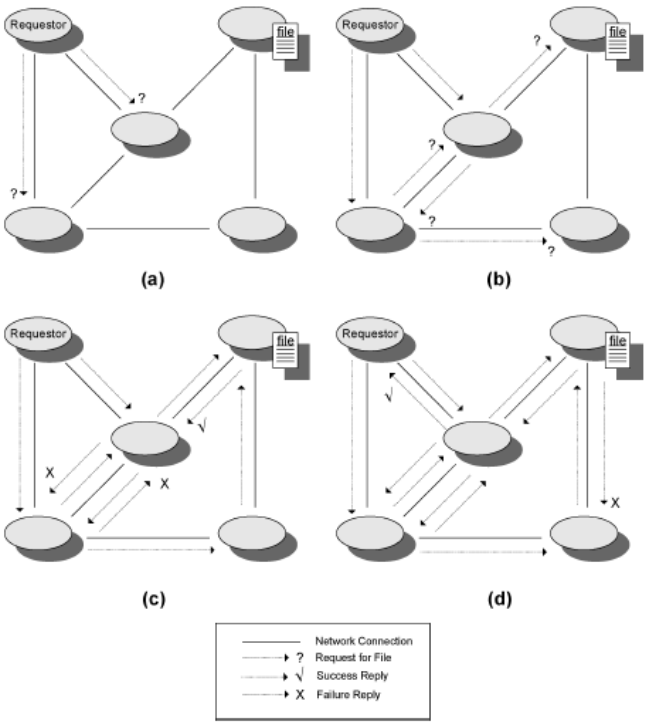
Η Gnutella λειτουργεί σαν ένα πρωτόκολλο ερωτήσεων. Για να το πετύχει αυτό χρησιμοποιεί πακέτα μηνυμάτων πέντε διαφορετικών τύπων (έκδοση 0.4):

- ping: για τον εντοπισμό κόμβων στο δίκτυο
- pong: απάντηση στο μήνυμα ping
- query: αναζήτηση αρχείου
- query hit: απάντηση στο ερώτημα της αναζήτησης
- push: μεταφόρτωση του αιτούμενου αρχείου

Η λειτουργία του Gnutella έχει ως εξής: Το δίκτυο απαρτίζεται από κόμβους (χρήστες) που έχουν εγκαταστήσει το λογισμικό του πελάτη και συνδέονται μεταξύ τους χωρίς συγκεκριμένη δομή. Αρχικά, για να συνδεθεί κανείς στο δίκτυο, αρκεί να εντοπίσει έναν κόμβο που συμμετέχει ήδη στο Gnutella. Εκείνος τον ενημερώνει με μια λίστα (την δική του) από άλλους κόμβους που μετέχουν στο δίκτυο. Ο κόμβος στέλνει τις ερωτήσεις του για αναζήτηση σε όλους τους ενεργά συνδεδεμένους κόμβους - συνήθως μέχρι πέντε - με εκείνον. Η ερώτηση εφόσον δεν απαντηθεί, προωθείται σε γειτονικούς κόμβους. Αν το επιθυμητό αρχείο εντοπισθεί σε περισσότερους κόμβους, τότε ο αιτών κόμβος μπορεί να κατεβάσει το αρχείο σε τμήματα από διαφορετικούς κόμβους, απευθείας όπως φαίνεται στο Σχήμα 6.3. Όταν ο κόμβος αποσυνδέεται η λίστα των κόμβων που ήταν ενεργά συνδεδεμένοι σε αυτόν αποθηκεύεται τοπικά για μελλοντική χρήση.

Για τον περιορισμό των μηνυμάτων που προκύπτουν από την δρομολόγηση των ερωτήσεων με την μέθοδο της πλημμύρας και τον τερματισμό της αναζήτησης χρησιμοποιείται ένα πεδίο στο μήνυμα που στέλνεται, το πεδίο Time - To - Live (TTL). Κάθε φορά που στέλνεται ένα μήνυμα από έναν κόμβο η τιμή του πεδίου μειώνεται κατά ένα. Ένας κόμβος που λαμβάνει την ερώτηση την προωθεί αν το πεδίο TTL έχει τιμή μεγαλύτερη του 0 και δεν έχει ξαναδεί το μήνυμα.

Ο μηχανισμός δρομολόγησης δημιουργεί υπερφόρτωση του δικτύου με μηνύματα και σπαταλά τους δικτυακούς πόρους.



Σχήμα 6.3: Παράδειγμα μηχανισμού αναζήτησης του Gnutella. Η αναζήτηση προέρχεται από τον αιτών κόμβο για ένα αρχείο που το κατέχει άλλο κόμβος. Διανέμονται μηνύματα αίτησης σε όλους τους γειτονικούς κόμβους και μεταδίδονται από κόμβο σε κόμβο όπως φαίνονται σε 4 διαδοχικά βήματα (α) έως (δ)

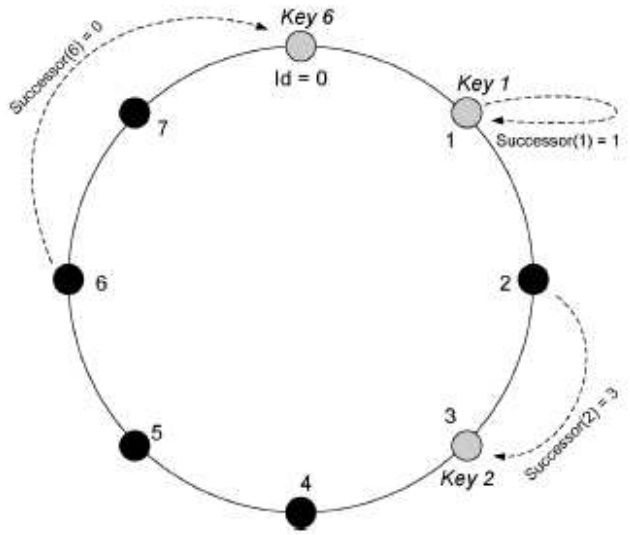
6.7.3 Σύστημα Chord

Το Chord είναι ένα κατανεμημένο πρωτόκολλο για τον εντοπισμό δεδομένων σε ένα ομότιμο σύστημα που αναπτύχθηκε από ομάδα του MIT και παρουσιάσθηκε το 2001. Βασίζεται στη λειτουργία: δεδομένου ενός κλειδιού, αυτό αντιστοιχίζεται σε έναν κόμβο, όπου αποθηκεύεται το ζεύγος κλειδί - τιμή. Πρόκειται για ένα σύστημα απλό στην κατασκευή του, ανεκτικό στις αλλαγές του ομότιμου δικτύου και εγγυάται την εύρεση των δεδομένων σε χρόνο $O(logN)$ όπου N το πλήθος των κόμβων στο δίκτυο.

Το Chord μπορεί να λειτουργήσει σε ένα δυναμικό περιβάλλον όπου οι κόμβοι εισέρχονται και αποχωρούν από το σύστημα αυθαίρετα με την απαίτηση όμως κάθε κόμβος να αποθηκεύει τμήμα της πληροφορία για επιτυχή δρομολόγηση.

Τόσο οι κόμβοι όσο και τα δεδομένα που έχουν μοναδικό αναγνωριστικό (ID) m-bit (160

bits) οργανώνονται σε έναν εικονικό δακτύλιο. Το ID του κόμβου κατακερματίζεται από την διεύθυνση IP του, και το ID του αντικειμένου (δεδομένο) κατακερματίζεται από το όνομα του και έτσι προκύπτει η θέση τους πάνω στο δακτύλιο (δακτύλιος των 0 έως $2^m - 1$ θέσεων). Οι κόμβοι κατέχουν πληροφορία για τον προηγούμενο και τον επόμενο κόμβο στο δακτύλιο. Ο κάθε κόμβος είναι υπεύθυνος για τα αντικείμενα που είναι μεταξύ του προηγούμενου κόμβου και του ίδιου. Στο Σχήμα 6.4 φαίνεται ένα παράδειγμα του συστήματος Chord με $m = 3$.



Σχήμα 6.4: Ένα σύστημα Chord που αποτελείται από τρεις κόμβους 0, 1 και 3. Σε αυτό το παράδειγμα, το κλειδί 1 βρίσκεται στο κόμβο 1, το κλειδί 2 στο κόμβο 3 και το κλειδί 6 στο κόμβο 0

Οι βασικές λειτουργίες που μπορούν να πραγματοποιηθούν σε ένα τέτοιο σύστημα είναι η εισαγωγή νέου κόμβου, η αποθήκευση και ανάκτηση δεδομένων και η αποχώρηση ενός κόμβου.

Όταν ένας κόμβος n επιθυμεί να εισέλθει στο σύστημα κατακερματίζει την διεύθυνση IP του για να προκύψει το αναγνωριστικό του και με κάποια διαδικασία (εξωτερικό μηχανισμό) μαθαίνει το αναγνωριστικό ενός κόμβου n" που ήδη ανήκει στο Chord. Ο νέος κόμβος χρησιμοποιεί τον n" κόμβο για να προσθέσει τον εαυτό του στο δίκτυο Chord και να αρχικοποιήσει την κατάσταση του που περιλαμβάνει και την μεταφορά κλειδιών από τον επόμενο του κόμβο, αν αυτά αντιστοιχίζονται τώρα σε αυτόν (ο κόμβος n είναι τώρα ο επόμενος τους). Η τελευταία διαδικασία απαιτεί το πολύ $O(1/N)$ μετακινήσεις κλειδιών.

Όταν ένας κόμβος αποχωρεί από το σύστημα, τότε τα κλειδιά που είχε στην ευθύνη του

αντιστοιχίζονται στον επόμενο του κόμβο. Η διαδικασία της ενημέρωσης καθώς οι κόμβοι εισέρχονται και αποχωρούν από το σύστημα απαιτεί $O(log^2 N)$ μηνύματα.

Ο κόμβος που επιθυμεί να αποθηκεύσει ένα αντικείμενο στο σύστημα εφαρμόζει μια συνάρτηση κατακερματισμού στο όνομα του αντικειμένου και προκύπτει το αναγνωριστικό του. Το νέο αντικείμενο αντιστοιχίζεται (και αποθηκεύεται) στον κόμβο που έχει ID ίσο με το ID του, αν υπάρχει ή του αμέσως επόμενου αν δεν υπάρχει.

Για την ανάκτηση του αντικειμένου χρησιμοποιείται παρόμοια διαδικασία. Εφαρμόζεται η συνάρτηση κατακερματισμού και προκύπτει η θέση του αντικειμένου και δρομολογείται η ανάκτησή του.

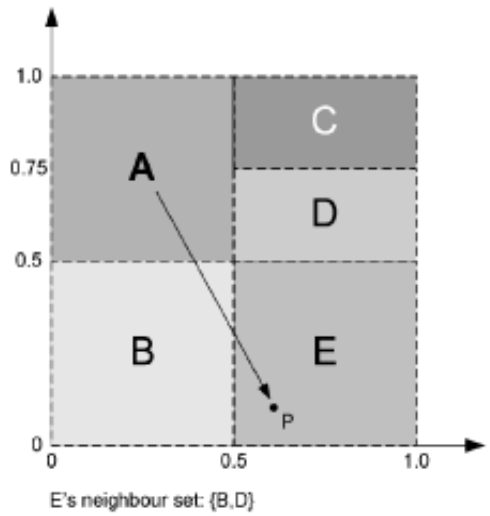
Η μόνη πληροφορία που είναι απαραίτητη στο σύστημα Chord για επιτυχή δρομολόγηση είναι η γνώση του επομένου. Όμως ένα τέτοιο σχήμα δεν διατηρεί την ιδιότητα της κλιμάκωσης. Για το λόγο αυτό κάθε κόμβος διατηρεί ένα τμήμα πληροφορίας για τους άλλους κόμβους βελτιώνοντας έτσι την απόδοση του αλγορίθμου. Η πληροφορία που διατηρεί ο κάθε κόμβος είναι ένας πίνακας m εγγραφών, **πίνακας δεικτών** (finger table), και περιέχει τους κόμβους που βρίσκονται σε απόσταση $2^0, 2^1, 2^2, ..., 2^{m-1}$ από αυτόν. Οι ερωτήσεις τώρα δρομολογούνται μέσω του πίνακα δεικτών και η αναζήτηση ολοκληρώνεται σε $O(logN)$ το πολύ βήματα.

6.7.4 Σύστημα CAN

Το σύστημα CAN (Content Addressable Network) είναι ένα ακόμη δομημένο σύστημα που ο σχηματισμός του δικτύου επικάλυψης βασίζεται σε κατανεμημένους πίνακες κατακερματισμού. Το CAN αναπτύχθηκε από ομάδα του πανεπιστημίου του Berkeley και παρουσιάσθηκε το 2001.

Η σχεδίαση του είναι επίσης απλή και κατανοητή. Βασίζεται στον εικονικό καρτεσιανό χώρο d-διαστάσεων. Για κάθε διάσταση υπάρχει μια συνάρτηση κατακερματισμού. Ο χώρος διαμερίζεται και αντιστοιχίζεται στους κόμβους που μετέχουν στο σύστημα. Το τμήμα του χώρου που κατέχει ένας κόμβος καλείται ζώνη (zone). Τα αντικείμενα (δεδομένα) αντιστοιχίζονται σε σημεία στο χώρο από μια συνάρτηση κατακερματισμού και αποθηκεύονται στους κόμβους που κατέχουν την ζώνη που ανήκουν τα σημεία, ως ζεύγη κλειδί - τιμή (K, V) με κλειδί K και τιμή V. Στο Σχήμα 6.5 φαίνεται ένα παράδειγμα συστήματος CAN.

Οι κόμβοι αποθηκεύουν τμήμα του κατανεμημένου πίνακα κατακερματισμού και πληροφορία για τους άμεσα γείτονες τους. Δύο κόμβοι θεωρούνται γείτονες αν οι $d-1$ διαστάσεις τους



Σχήμα 6.5: Ένα σύστημα CAN σε χώρο δύο διαστάσεων [0,1] × [0,1] διαμερισμένο σε 5 κόμβους

εκτεινόμενες συμπίπτουν και εφάπτονται στην άλλη διάσταση.

Ένα μήνυμα CAN περιλαμβάνει τις συντεταγμένες προορισμού. Έτσι δρομολογείται από έναν άπληστο αλγόριθμο (greedy algorithm) προς τον κόμβο που βρίσκεται πιο κοντά στον προορισμό του. Το μέσο μήκος μονοπατιού για ένα χώρο d διαστάσεων με n ίσες ζώνες είναι $(d/4)(n^{1/d})$ βήματα, η δε πληροφορία για τους γείτονες που διατηρεί κάθε κόμβος είναι $2d$.

Οι βασικές λειτουργίες που μπορούν να πραγματοποιηθούν σε ένα τέτοιο σύστημα είναι η εισαγωγή νέου κόμβου, η αποχώρηση ενός κόμβου και η αποθήκευση και ανάκτηση δεδομένων.

Η σύνδεση ενός νέου κόμβου γίνεται μέσω της διαδικασίας εκκίνησης. Ο νεοεισερχόμενος κόμβος εντοπίζει έναν κόμβο που ανήκει ήδη στο σύστημα CAN. Με την χρήση του μηχανισμού δρομολόγησης, επιλέγει ένα τυχαίο σημείο P στο χώρο και στέλνει ένα μήνυμα συνένωσης στον κόμβο που καλύπτει την ζώνη όπου ανήκει το P και ο οποίος μπορεί να διαμεριστεί. Ο νέος κόμβος αναλαμβάνει τη μισή ζώνη ενώ το άλλο μισό το αναλαμβάνει ο προκάτοχος του. Οι γείτονες κόμβοι ενημερώνονται έτσι ώστε να συμπεριλάβουν και τον νέο κόμβο.

Όταν ένας κόμβος αποχωρεί από το σύστημα τότε η ζώνη που καταλαμβάνει συνενώνεται ή αναλαμβάνεται από κάποιον γείτονα - αν δεν είναι δυνατή η συνένωση - όπως επίσης και το τμήμα του πίνακα κατακερματισμού που διατηρούσε ο κόμβος που αποχώρησε.

Για την εισαγωγή ενός αντικειμένου (K, V) με κλειδί K και τιμή V στο σύστημα CAN

έχει ως εξής: Ο κόμβος που επιθυμεί να εισάγει το αντικείμενο εφαρμόζει τις συναρτήσεις κατακερματισμού στο κλειδί K για την εύρεση των συντεταγμένων στο χώρο, σημείο P. Στη συνέχεια δρομολογείται στο σημείο αυτό και αποθηκεύεται το ζεύγος (K, V) στον κόμβο που κατέχει τη ζώνη.

Το σύστημα CAN έχει μηχανισμούς για την αποχώρηση κόμβου, την ανάκαμψη και την συντήρηση. Σε εθελούσια αποχώρηση ο κόμβος ενημερώνει και παραδίδει την πληροφορία που κατέχει σε έναν από τους γείτονές του. Οι κόμβοι ανταλλάσσουν περιοδικά μηνύματα για ανίχνευση-ενημέρωση αλλαγών που έχουν προκύψει (συντεταγμένες ζώνης, γείτονες, συντεταγμένες γειτόνων).

6.8 Σύγκριση Συστημάτων Υπολογιστών

Τα κατανεμημένα συστήματα, το υπολογιστικό πλέγμα και τα ομότιμα συστήματα μοιάζουν με τις συστοιχίες υπολογιστών στο γεγονός ότι όλοι οι υπολογιστές έχουν την δική του μνήμη και δεν υπάρχει κοινή μνήμη όπως οι συμμετρικοί πολυεπεξεργαστές. Ωστόσο, υπάρχουν ορισμένες διαφορές μεταξύ τους. Πρώτον, σε όλα τα υπολογιστικά συστήματα από συμμετρικούς πολυεπεξεργαστές μέχρι και ομότιμα συστήματα υπάρχει κλιμάκωση ως προς το αριθμό των κόμβων. Ιδιαίτερα τα μεγαλύτερα συστήματα όπως οι υπολογιστικές συστοιχίες, το υπολογιστικό πλέγμα και τα ομότιμα συστήματα είναι πολύ ανώτερα από τους συμμετρικούς πολυεπεξεργαστές ως προς την αυξητική και απόλυτη ικανότητα κλιμάκωσης. Δεύτερον, ότι οι κόμβοι μιας συστοιχίας μπορεί να είναι υπολογιστές που διαθέτουν επεξεργαστή, μνήμη, κάρτα δικτύου και σκληρό δίσκο ενώ οι κόμβοι ενός κατανεμημένου συστήματος, πλέγματος υπολογιστών και ομότιμου συστήματος μπορεί να είναι πλήρεις υπολογιστές με όλες τις περιφερειακές συσκευές, πολυεπεξεργαστή SMP ή ακόμα μπορεί να είναι μια συστοιχία υπολογιστών. Ενώ οι κόμβοι ενός συμμετρικού πολυεπεξεργαστή είναι απλοί επεξεργαστές και διαθέτει συστήματα μνήμης και Ε/Ε ως ξεχωριστά συστατικά. Επίσης, όλοι οι κόμβοι μιας συστοιχίας βρίσκονται σε σχετικά μικρές αποστάσεις ώστε να επικοινωνούν μέσω ενός αποκλειστικού δικτύου διασύνδεσης υψηλής ταχύτητας, ενώ οι κόμβοι ενός κατανεμημένου συστήματος, υπολογιστικού πλέγματος και ομότιμου συστήματος βρίσκονται σε απομακρυσμένες αποστάσεις. Επιπλέον, οι κόμβοι μιας συστοιχίας υπολογιστών εκτελούν το ίδιο λειτουργικό σύστημα, μοιράζονται το ίδιο σύστημα αρχείων και υπάγονται σε κοινή διαχείριση, ενώ οι κόμβοι ενός κατανεμημένου συστήματος, υπ-

ολογιστικού πλέγματος και ομότιμου συστήματος μπορούν να εκτελούν διαφορετικά λειτουργικά συστήματα, το καθένα με το δικό του σύστημα αρχείων και να υπάγονται σε διαφορετικούς διαχειριστές. Εδώ πρέπει να σημειωθεί ότι υπάρχει μια ενιαία και ευκολότερη διαχείριση στους συμμετρικούς πολυπεξεργαστές και συστοιχίες υπολογιστών από ότι τα μεγαλύτερα συστήματα όπως το πλέγμα και τα ομότιμα συστήματα. Τέλος, από την πλευρά των εφαρμογών, οι συμμετρικοί πολυεπεξεργαστές και συστοιχίες υπολογιστών χρησιμοποιούνται για την επίλυση των πιο απαιτητικών υπολογιστικών προβλημάτων ενώ τα κατανεμημένα και ομότιμα συστήματα χρησιμοποιούνται για να παρέχουν υπηρεσίες σε άλλους υπολογιστές ή προγράμματα κυρίως σε ανταλλαγές δεδομένων και τα πλέγματα υπολογιστών χρησιμοποιούνται σε ανταλλαγές δεδομένων και εκτέλεση υπολογισμών ταυτόχρονα. Επίσης, οι συμμετρικοί πολυεπεξεργαστές και συστοιχίες υπολογιστών μπορούν να χρησιμοποιηθούν για εφαρμογές διαθεσιμότητας κυρίως ως διακομιστές υψηλής απόδοσης. Οι συστοιχίες υπολογιστών είναι ανώτερα ως προς την διαθεσιμότητα σε σχέση με τους συμμετρικούς πολυεπεξεργαστές επειδή όλα τα συστατικά μέρη του συστήματος μπορούν εύκολα να αναπαραχθούν. Στον Πίνακα 6.1 παρουσιάζονται συνοπτικά οι διαφορές μεταξύ του συμμετρικού πολυεπεξεργαστή, της συστοιχίας υπολογιστών, του κατανεμημένου συστήματος, του υπολογιστικού πλέγματος και του ομότιμου συστήματος.

Πίνακας 6.1: Βασικά χαρακτηριστικά των παράλληλων και κατανεμημένων συστημάτων

Στοιχείο	Πολυεπεξεργαστής SMP	Συστοιχία Υπολογιστών	Κατανεμημένο Σύστημα	Πλέγμα Υπολογιστών	Ομότιμο Σύστημα
Πλήθος κόμβων	10 - 100	100 ή λιγότερα	10 - 1000	10000	1000000
Πολυπλοκότητα κόμβου	Επεξεργαστή	Ψπολογιστής, SMP	Πλήρης Υπολογιστής Συστοιχία	Υπολογιστής, SMP Συστοιχία	Υπολογιστής
Τοποθεσία	Κουτί	Δωμάτιο	Ευρεία	Παγκόσμιο	Παγκόσμιο
Επικοινωνία κόμβων	Κοινή μνήμη	Μεταβίβαση μηνυμάτων	Μεταβίβαση μηνυμάτων	Μεταβίβαση μηνυμάτων	Μεταβίβαση μηνυμάτων
Δίκτυο	Κοινή μνήμη	Τοπικό δίκτυο	Δίκτυο ευρείας	Δίκτυο ευρείας	Δίκτυο ευρείας
Λειτουργικό σύστημα	Ένα	Πολλά, ίδια	Πολλά, διαφορετικά	Πολλά, διαφορετικά	Πολλά, διαφορετικά
Διαχείριση	Ένας διαχειριστής	Ένας διαχειριστής	Πολλοί διαχειριστές	Πολλοί διαχειριστές	Πολλοί διαχειριστές
Εφαρμογή	Υπολογισμούς, Διαθεσιμότητα	Υπολογισμούς, Διαθεσιμότητα	Ανταλλαγή δεδομένων	Υπολογισμούς	Επικοινωνία, Υπολογισμούς

Κεφάλαιο 7

Μοντέλα Προγραμματισμού

Μέχρι στιγμής έχουμε εξετάζει τις αρχιτεκτονικές των συμμετρικών πολυεπεξεργαστών (SMP), της συστοιχίας υπολογιστών, τα κατανεμημένα συστήματα, το πλέγμα υπολογιστών και τα ομότιμα συστήματα υπολογιστών καθώς και το λογισμικό συστήματος που ήταν απαιραίτητο για την λειτουργία τους. Πρέπει να σημειώσουμε ότι τα παραπάνω συστήματα θα τα αναφέρουμε ως συστήματα υπολογιστών. Σε αυτό το κεφάλαιο θα εξετάσουμε τα μοντέλα προγραμματισμού και το λογισμικό εφαρμογών που απαιτούνται για το προγραμματισμό αυτών των συστημάτων.

7.1 Ενδιάμεσο Λογισμικό Εφαρμογών

Ακολουθούν ορισμοί για τους σκοπούς του βιβλίου. Έτσι, ένα **ακολουθιακό πρόγραμμα** (sequential program) είναι μια διεργασία που εκτελείται σε ένα σύστημα ενός επεξεργαστή και έχει μια μοναδική ροή ελέγχου. Ένα **συγχρονικό πρόγραμμα** (concurrent program) αποτελείται από ένα σύνολο διεργασιών που εκτελούνται συγχρονικά με χρονοκαταμερισμό σε ένα ή περισσότερους επεξεργαστές. Η επικοινωνία των διεργασιών ενός συγχρονικού προγράμματος γίνεται μέσω κοινής μνήμης. Ενώ, ένα **παράλληλο πρόγραμμα** (parallel program) περιέχει ένα σύνολο διεργασιών όσο είναι οι υπολογιστές μιας συστοιχίας υπολογιστών που όλες οι διεργασίες εκτελούν τον ίδιο κώδικα στους αντίστοιχους υπολογιστές αλλά συνεργάζονται παράλληλα. Η επικοινωνία των διεργασιών ενός παράλληλου προγράμματος γίνεται με μεταβίβαση μηνυμάτων. Τέλος, ένα **κατανεμημένο πρόγραμμα** (distributed program) είναι μια συλλογή ανεξαρτήτων διεργασιών που εκτελούνται σε απομακρυσμένους υπολογιστές.

Η επικοινωνία των διεργασιών ενός κατανεμημένου προγράμματος γίνεται με ανταλλαγή μηνυμάτων ή με απομακρυσμένη κλήση μεθόδων.

Η διαδικασία ανάπτυξης προγράμματος για συμμετρικό πολυεπεξεργαστή ή για συστοιχία υπολογιστών ή για κατανεμημένο σύστημα ή για πλέγμα υπολογιστών ονομάζεται συγχρονικό ή παράλληλο ή κατανεμημένο ή πλεγματικό προγραμματισμό αντίστοιχα.

Παρόλες τις ραγδαίες εξελίξεις των συστημάτων υπολογιστών, η διαδικασία προγραμματισμού στα συστήματα αυτά είναι μια πολύπλοκη εργασία. Πρώτα από όλα, η ανάπτυξη των παράλληλων κατανεμημένων εφαρμογών εξαρτάται σε μεγάλο βαθμό από την διαθεσιμότητα των εργαλείων λογισμικού εφαρμογών και από τα περιβάλλοντα προγραμματισμού. Επίσης, ο προγραμματιστής του παράλληλου και κατανεμημένου λογισμικού συναντά ορισμένα προβλήματα που δεν εμφανίζονται στην διαδικασία του ακολουθιακού προγραμματισμού όπως **επικοινωνία** (communication), **συγχρονισμός** (synchronization), **κατάτμηση δεδομένων και διανομή** (data partitioning and distribution), **ισορροπία φορτίου** (load balancing), **αδιέξοδα** (deadlocks) και **συνθήκες ανταγωνισμού** (race conditions).

Όμως γνωρίζουμε ότι στα παράλληλα κατανεμημένα συστήματα υπάρχει ετερογένεια ως προς το υλικό των υπολογιστών, τα δίκτυα, τα λειτουργικά συστήματα αλλά ακόμη και τις διαφορετικές παράλληλες γλώσσες προγραμματισμού που χρησιμοποιούνται ανάλογα με την αρχιτεκτονική των συστημάτων υπολογιστών. Αυτό έχει σαν συνέπεια ο προγραμματισμός στα συστήματα αυτά να είναι δύσκολος λόγω έλλειψη κοινού μοντέλου προγραμματισμού. Έτσι, ο προγραμματισμός για να είναι επιτυχής στα συστήματα αυτά, θα πρέπει το παράλληλο κατανεμημένο λογισμικό εφαρμογών να συνοδεύει τα παρακάτω χαρακτηριστικά όπως να:

- παρέχει διαφάνεια αρχιτεκτονικής και τύπου επεξεργαστή,
- παρέχει διαφάνεια δικτύου και επικοινωνίας,
- είναι εύκολο στην χρήση και αξιόπιστο,
- κρύψει την ετερογένεια και να είναι μεταφέρσιμο,
- μπορεί να συνεργαστεί και να υποστηρίξει ακολουθιακές γλώσσες προγραμματισμού υψηλού επιπέδου και
- παρέχει υψηλή απόδοση.

Για αυτό το λόγο ανάμεσα στο υλικό των συστημάτων υπολογιστών και στις παράλληλες κατανεμημένες εφαρμογές παρεμβάλλεται το ενδιάμεσο λογισμικό εφαρμογών. Με άλλα λόγια, το ενδιάμεσο λογισμικό παρέχει υπηρεσίες και αφαίρεση στους προγραμματιστές για την εύκολη ανάπτυξη παράλληλων κατανεμημένων εφαρμογών και επίσης έχουν στόχο να κρύψουν την ετερογένεια των συστημάτων.

Οι υπηρεσίες του ενδιάμεσου λογισμικού συνήθως βασίζεται σε κάποιο συγκεκριμένο μοντέλο παράλληλου κατανεμημένου προγραμματισμού. Το **μοντέλο** ή **υπόδειγμα** (paradigm) παρέχει μια λογική αφαίρεση και πρότυπο στους προγραμματιστές για το τρόπο με τον οποίο προγραμματίζονται οι παράλληλες κατανεμημένες εφαρμογές. Παρακάτω παρουσιάζεται μια ταξινόμηση των μοντέλων προγραμματισμού για παράλληλες και κατανεμημένες εφαρμογές:

- Νήματα (threads),
- Μεταβίβαση μηνυμάτων (message passing),
- Απομακρυσμένη κλήση διαδικασιών (remote procedure call),
- Κατανεμημένα αντικείμενα (distributed objects),
- Υπηρεσίες Ιστού και πλέγματος (web and grid services) και
- Ομότιμες υπηρεσίες (peer to peer services).

Οι λόγοι για τους οποίους υπάρχουν τόσα υποδείγματα ή μοντέλα προγραμματισμού είναι μεταξύ άλλων, οι εξής:

- Ο προγραμματισμός συστημάτων υπολογιστών είναι σχετικά νέα περιοχή και οι επιστήμονες δεν έχουν κατασταλάξει στο πως ακριβώς πρέπει να γίνεται,
- Οι προγραμματιστές επιθυμούν να υλοποιήσουν διαφορετικής φύσεως εφαρμογές και
- Οι νέες αρχιτεκτονικές υπολογιστών διαφέρουν σημαντικά μεταξύ τους.

Τέλος, το ενδιάμεσο λογισμικό εφαρμογών για να υποστηρίξει τα παραπάνω υποδείγματα προγραμματισμού συνοδεύονται από εργαλεία ή **διεπαφές προγραμματιστή εφαρμογών** (application programmer's interface) για την ανάπτυξη εφαρμογών. Οι διεπαφές αυτές απομομονώνει τον προγραμματιστή από πολύπλοκες λεπτομέρειες των πρωτοκόλλων και των

μηχανισμών μεταφοράς δεδομένων. Έτσι, μια διεπαφή ή εργαλείο παρέχει κλάσεις, μεθόδους και σταθερές που επιτρέπουν σε έναν προγραμματιστή να αναπτύξει γρήγορα μια εφαρμογή. Στην συνέχεια αυτού του βιβλίου θα περιγράψουμε διεπαφές Java για την υποστήριξη των παραπάνω μοντέλων προγραμματισμού.

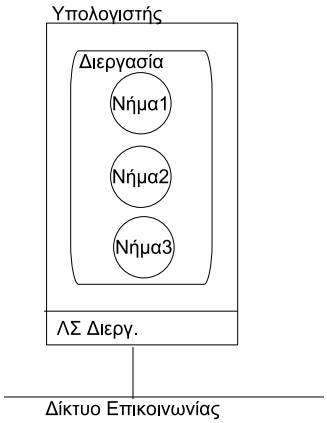
7.2 Μοντέλα Προγραμματισμού και Εργαλεία

Εδώ παρουσιάζουμε συνοπτικά τα πιο δημοφιλή μοντέλα ή πρότυπα προγραμματισμού και εργαλεία στα παράλληλα κατανεμημένα συστήματα.

7.2.1 Νήματα

Τα νήματα είναι ένα δημοφιλές μοντέλο για **συχρονικό προγραμματισμό** (concurrent programming) σε συμβατικό (ή ακολουθιακό) υπολογιστή και σε συστήματα συμμετρικής πολυεπεξεργασίας. Στα συστήματα πολυεπεξεργασίας, τα νήματα χρησιμοποιούνται κυρίως για να αξιοποιήσουν ταυτόχρονα όλους τους επεξεργαστές. Από την άλλη μεριά στα συμβατικά συστήματα, τα νήματα χρησιμοποιούνται για να αξιοποιήσουν αποτελεσματικά τους πόρους του συστήματος. Συγκεκριμένα, ο συγχρονικός προγραμματισμός υποστηρίζει δύο είδη εκτέλεσης: συγχρονική εκτέλεση νημάτων σε ένα σύστημα υπολογιστή και συγχρονική εκτέλεση νημάτων σε μια διεργασία. Η συγχρονική εκτέλεση νημάτων σε ένα σύστημα είναι πραγματικά παράλληλη αν το σύστημα είναι συμμετρικός πολυεπξεργαστής έτσι ώστε κάθε επεξεργαστής να εκτελεί ένα νήμα. Αντίθετα, η συγχρονική εκτέλεση νημάτων είναι εικονική αν το σύστημα υποστηρίζει έναν επεξεργαστή και σε αυτή την περίπτωση τα νήματα εκτελούνται ψευδοπαράλληλα. Η ψευδοπαράλληλη εκτέλεση νημάτων υλοποιείται από ένα πρόγραμμα χρονοδρομολογητή που ορίζει ποιο νήμα εκτελείται ανά πάσα στιγμή. Σε αυτή την περίπτωση, ο προγραμματισμός νημάτων γίνεται με την χρήση της κοινής μνήμης (ή των διαμοιραζόμενων μεταβλητών) καθώς τα νήματα δημιουργούνται μέσα από το χώρο διευθύνσεων της πατρικής διεργασίας όπως θα δούμε στο κεφάλαιο 8.

Η συγχρονική εκτέλεση νημάτων σε μια διεργασία χρησιμοποιείται περισσότερο στις κατανεμημένες εφαρμογές και είναι απαραίτητο για μια διεργασία που εκτελείται σε ένα υπολογιστή να δημιουργήσει νήματα που θα εκτελούνται συγχρονικά όπως φαίνεται στο Σχήμα 7.1. Για



Σχήμα 7.1: Συγχρονική εκτέλεση νημάτων μέσα σε μια διεργασία

παράδειγμα, μια διεργασία διακομιστή Ιστού που εκτελείται σε έναν υπολογιστή είναι ικανή να επικοινωνεί με πολλούς πελάτες ενώ συγχρόνως να περιμένει αιτήσεις από άλλους πελάτες. Η επικοινωνία του διακομιστή με ένα συγκεκριμένο πελάτη υλοποιείται από ένα νήμα. Αυτό είναι ίσως επιθυμητό για μια διεργασία να εκτελεί πολλά νήματα συγχρονικά για λόγους απόδοσης. Σε αυτή την περίπτωση, ο προγραμματισμός συγχρονικών διεργασιών γίνεται με την χρήση νημάτων όπως θα δούμε στο κεφάλαιο 10.

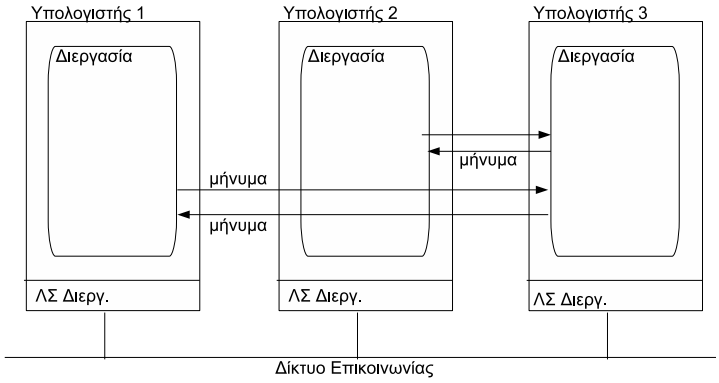
Ένα άλλο χαρακτηριστικό είναι ότι η δημιουργία ενός νήματος είναι φθηνότερη σε σχέση με την δημιουργία μιας διεργασίας. Η διεπαφή λογισμικού που υποστηρίζει το προγραμματισμό νημάτων περιλαμβάνει την δημιουργία και το συγχρονισμό ή συντονισμό των νημάτων. Παρακάτω αναφέρουμε ορισμένα εργαλεία λογισμικού.

Τα νήματα είναι μεταφέρσιμα καθώς υπάρχει το πρότυπο της IEEE POSIX (Portable Operating System Interface) που λέγεται pthreads [5]. Το εργαλείο αυτό βασίζεται σε μια βιβλιοθήκη συναρτήσεων για τις οποίες καλούνται μέσα από το παράλληλο πρόγραμμα. Επίσης, υπάρχουν τα πρότυπα OpenMP και JOMP που συνοδεύονται για ακολουθιακές γλώσσες προγραμματισμού C/C++, Fortran και Java αντίστοιχα. Τα εργαλεία OpenMP και JOMP βασίζονται σε ένα σύνολο οδηγιών προς το μεταγλωττιστή και ενσωματώνονται είτε στο ακολουθιακό είτε στο παράλληλο πρόγραμμα. Τα παραπάνω πρότυπα υποστηρίζουν σε πολλές πλατφόρμες όπως προσωπικούς υπολογιστές, σταθμούς εργασίας, SMPs και συστοιχίες υπολογιστών. Επίσης, η γλώσσα προγραμματισμού Java έχει ενσωματωμένη υποστήριξη πολυνημάτωσης (multithread-

ing) για την εύκολη ανάπτυξη πολυνηματικών εφαρμογών. Τα νήματα έχουν χρησιμοποιηθεί εκτενώς στην ανάπτυξη εφαρμογών και λογισμικού συστήματος. Τέλος, στο κεφάλαιο 8 θα παρουσιάζουμε περισσότερες λεπτομέρειες για το προγραμματισμό σε συστήματα κοινής μνήμης χρησιμοποιώντας τα εργαλεία Java Threads και JOMP.

7.2.2 Μεταβίβαση Μηνυμάτων

Η μεταβίβαση μηνυμάτων είναι ένα μοντέλο προγραμματισμού που βασίζεται σε ένα σύνολο διεργασιών που εκτελούνται σε ανεξάρτητους υπολογιστές ενός παράλληλου ή κατανεμημένου συστήματος που έχουν την δική του τοπική μνήμη. Οι δύο διεργασίες, αποστολέας και παραλήπτης, επικοινωνούν μεταξύ τους μέσω της χρήσης μεταβίβασης μηνυμάτων όπως φαίνεται στο Σχήμα 7.2. Το μοντέλο μεταβίβασης μηνυμάτων είναι ιδιαίτερα σημαντικό για παράλληλες



Σχήμα 7.2: Μοντέλο προγραμματισμού μεταβίβαση μηνυμάτων

εφαρμογές μεγάλης κλίμακας που απαιτεί την συνεργασία και την επικοινωνία μεταξύ των διεργασιών. Μια διεργασία στέλνει ένα μήνυμα που περιέχει δεδομένα του προβλήματος σε μια άλλη διεργασία. Η άλλη διεργασία παραλαμβάνει τα δεδομένα, εκτελεί απαιτητικούς υπολογισμούς και στέλνει ένα μήνυμα που περιέχει τα αποτελέσματα των υπολογισμών.

Επίσης, το μοντέλο μεταβίβασης μηνυμάτων είναι σημαντικό και για τις κατανεμημένες εφαρμογές. Οι διεργασίες ενός κατανεμημένου συστήματος ανταλλάσουν μηνύματα σύμφωνα με ένα πρωτόκολλο αίτησης - απάντησης. Συγκεκριμένα, μια διεργασία στέλνει ένα μήνυμα αίτησης που απαιτεί μια υπηρεσία από άλλη διεργασία. Η άλλη διεργασία λαμβάνει το μήνυμα,

επεξεργάζεται την αίτηση και στέλνει ένα μήνυμα απάντησης. Το μήνυμα - απάντησης ίσως πυροδοτήσει μια αλληλουχία μηνυμάτων αίτησης και απάντησης ανάμεσα στις δύο διεργασίες.

Τα εργαλεία που βασίζονται στην μεταβίβαση μηνυμάτων είναι πολλά όπως οι **υποδοχές** (sockets) για κατανεμημένες εφαρμογές και οι **βιβλιοθήκες επικοινωνίας** (communication libraries) για παράλληλες εφαρμογές. Οι υποδοχές χρησιμοποιούνται κυρίως στην συγγραφή προγραμμάτων για τα κατανεμημένα συστήματα. Οι υποδοχές είναι μια αφηρημένη έννοια χαμηλού επιπέδου η οποία χρησιμοποιείται για επικοινωνία σε δίκτυα βασισμένα σε πρωτόκολλα γενικού σκοπού όπως TCP/IP. Συνεπώς, η υποδοχή είναι ένα λογικό κανάλι επικοινωνίας που αποτελείται από μια διεύθυνση IP ενός υπολογιστή και την **θύρα** (port). Έτσι, δύο απομακρυσμένες διεργασίες μπορούν να ανταλλάσουν μηνύματα χρησιμοποιώντας τις αντίστοιχες υποδοχές τους και διάφορες συναρτήσεις αποστολής - λήψης μηνυμάτων ως εξής: Η διεργασία αποστολέα γράφει δεδομένα στην υποδοχή της διεργασίας παραλήπτη και η διεργασία παραλήπτη διαβάζει το μήνυμα από την υποδοχή της. Στο κεφάλαιο 10 θα εξετάσουμε τον προγραμματισμό υποδοχών σε Java που ανταλλάσουν μηνύματα μεταξύ δύο απομακρυσμένων διεργασιών ενός κατανεμημένου συστήματος χρησιμοποιώντας το ενσωματωμένο πακέτο `java.net`.

Οι βιβλιοθήκες μεταβίβασης μηνυμάτων μας επιτρέπουν να γράφουμε παράλληλα προγράμματα για παράλληλα συστήματα (όπως οι συστοιχίες υπολογιστών). Επίσης, οι βιβλιοθήκες προσφέρουν συναρτήσεις υψηλού επιπέδου για την διαχείριση του περιβάλλοντος μηνυμάτων και συναρτήσεις υψηλού επιπέδου για την αποστολή και λήψη μηνυμάτων. Στην παρούσα φάση υπάρχουν δύο δημοφιλείς βιβλιοθήκες μεταβίβασης μηνυμάτων για επιστημονικές και μηχανικές εφαρμογές που είναι το PVM (Parallel Virtual Machine) [3] από το Oak Ridge National Laboratory και το MPI (Message Passing Interface) [8, 7] που ορίστηκε από το MPI Forum.

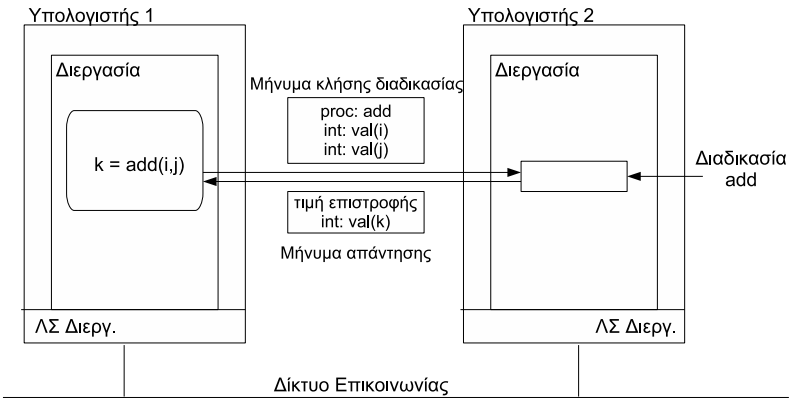
Το MPI είναι η πιο διαδεδομένη πρότυπη βιβλιοθήκη για μεταβίβαση μηνυμάτων που μπορεί να χρησιμοποιηθεί για την ανάπτυξη μεταφέρσιμων προγραμμάτων μεταβίβασης μηνυμάτων χρησιμοποιώντας τις ακολουθιακές γλώσσες προγραμματισμού C ή Fortran. Για την γλώσσα προγραμματισμού Java υποστηρίζεται από άλλο πρότυπο που λέγεται MPJ. Το πρότυπο MPI ορίζει συντακτικά και σημασιολογικά ένα σύνολο από συναρτήσεις βιβλιοθήκης που είναι χρήσιμες για την σύνταξη προγραμμάτων μεταβίβασης μηνυμάτων. Το MPI αναπτύχθηκε από μια κοινότητα ερευνητών οι οποίοι προέρχονται από πανεπιστήμια και βιομηχανίες και υποστηρίζεται από όλους τους προμηθευτές υλικού. Επίσης, το MPI είναι διαθέσιμο σε περισσότερα παράλληλα υπολογιστικά συστήματα (από υπολογιστές διαμοιραζόμενης μνήμης μέχρι και συστοιχίες υπολ-

ογιστών). Στο κεφάλαιο 9 θα παρουσιάσουμε τον προγραμματισμό μεταβίβασης μηνυμάτων σε συστοιχίες υπολογιστών χρησιμοποιώντας την βιβλιοθήκη MPJ.

7.2.3 Απομακρυσμένη Κλήση Διαδικασιών

Το μοντέλο μεταβίβασης μηνυμάτων λειτουργεί καλά για εφαρμογές με πολύ απλές ανάγκες επικοινωνίας. Καθώς οι ανάγκες επικοινωνίας των εφαρμογών γίνονται πολύπλοκες είναι απαραίτητο για περαιτέρω αφαιρέσεις ώστε να διεκολυνθεί ο προγραμματισμός της επικοινωνίας. Συγκεκριμένα, είναι επιθυμητό να έχουμε ένα μοντέλο που επιτρέπει το κατανεμημένο λογισμικό να προγραμματιστεί με τέτοιο τρόπο που να είναι παρόμοιο με τις σειριακές εφαρμογές. Μια τέτοια αφαίρεση είναι το μοντέλο απομακρυσμένη κλήση διαδικασιών που είναι μια τεχνολογία γνωστή από την αρχή της δεκαετίας του 1980. Χρησιμοποιώντας το μοντέλο αυτό, η επικοινωνία ανάμεσα στις δύο διεργασίες εκτελείται με ένα τρόπο παρόμοιο με εκείνο της τοπικής κλήσης διαδικασιών.

Στην ουσία, η κλήση απομακρυσμένων διαδικασιών (όπως φαίνεται στο Σχήμα 7.3) περιλαμβάνει δύο ανεξάρτητες διεργασίες οι οποίες εκτελούνται σε δύο ξεχωριστούς υπολογιστές. Η διεργασία 1 που εκτελείται σε έναν υπολογιστή καλεί μια διαδικασία `add` στο απομακρυσμένο υπολογιστή της διεργασίας 2 με παραμέτρους εισόδου τα ορίσματα της διαδικασίας. Στην συνέχεια, μεταβιβάζονται οι παράμετροι εισόδου στην απομακρυσμένη διαδικασία `add` που θα εκτελεστεί ως μήνυμα. Έπειτα εκτελείται η απομακρυσμένη διαδικασία `add` και η διεργασία 2 θα μεταβιβάζει το αποτέλεσμα της διαδικασίας στην διεργασία 1 που προκάλεσε την απομακρυσμένη κλήση.



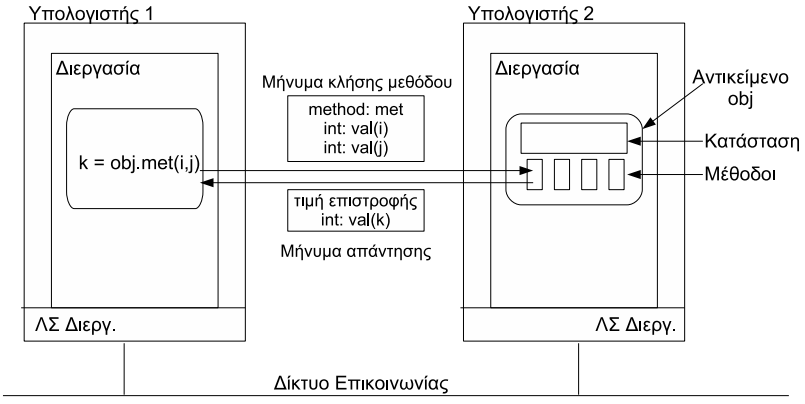
Σχήμα 7.3: Μοντέλο απομακρυσμένης κλήσης διαδικασιών

Τα σχήματα απομακρυσμένων διαδικασιών έχουν σχεδιαστεί έτσι ώστε ο προγραμματιστής να μην χρειάζεται να γνωρίζει πότε ένα πρόγραμμα που εκτελείται βρίσκεται σε διαφορετικό υπολογιστή.

Μια αρκετά ώριμη προγραμματιστική διεπαφή RPC είναι αυτό που σχεδιάστηκε για το Κατανεμημένο Περιβάλλον Επεξεργασίας (Distributed Computing Environment - DCE). Σε αυτό το βιβλίο δεν θα ασχοληθούμε σε βάθος το μοντέλο RPC επειδή το μοντέλο αυτό είναι διαδικασιοστρεφές και παλιό.

7.2.4 Κατανεμημένα Αντικείμενα

Το μοντέλο κατενεμημένων αντικειμένων είναι ένα μοντέλο που παρέχει περισσότερο αφαίρεση σε σχέση με το μοντέλο μεταβίβασης μηνυμάτων και ουσιαστικά επεκτείνει το μοντέλο απομακρυσμένης κλήσης διαδικασιών σε αντικείμενα που υπάρχουν σε ένα κατανεμημένο σύστημα. Είναι γνωστό ότι τα **αντικείμενα** (objects) συνδυάζουν τις **ιδιότητες** (properties) και τις **μεθόδους** (methods) σε μια οντότητα. Τα κατανεμημένα αντικείμενα είναι αντικείμενα που βρίσκονται διασκορπισμένα σε διαφορετικούς υπολογιστές (συνήθως φιλοξενούνται σε διακομιστές) και τα αντικείμενα επικοινωνούν μεταξύ τους μέσω της **κλήσης απομακρυσμένων μεθόδων** (remote method invocation). Κατά την κλήση μεθόδων σε απομακρυσμένα αντικείμενα ανταλάσσονται οι παράμετροι των μεθόδων ως μηνύματα. Στο Σχήμα 7.4 φαίνεται το μοντέλο της απομακρυσμένης κλήσης μεθόδων. Για να υλοποιηθεί η απομακρυσμένη



Σχήμα 7.4: Μοντέλο απομακρυσμένης κλήσης μεθόδων

κλήση μεθόδων πρέπει πρώτα η διεργασία 1 να εντοπίζει το αντικείμενο `obj` στο δίκτυο που

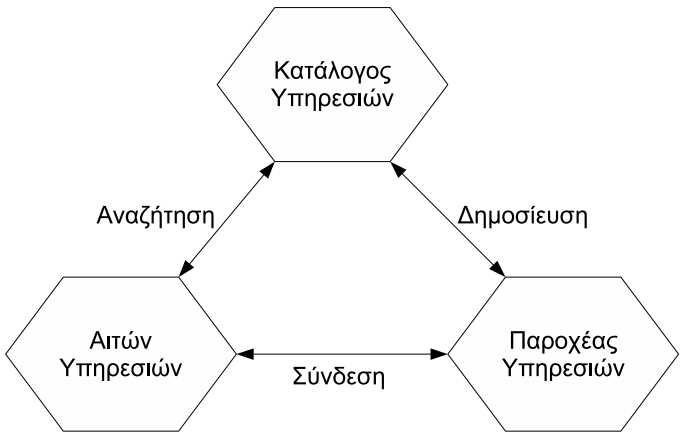
θέλει να καλέσει, στην συνέχεια η διεργασία 1 κάνει κλήση μεθόδου `met` στο απομακρυσμένο αντικείμενο `obj` της διεργασίας 2 μεταβιβάζοντας τις παραμέτρους της μεθόδου σε ένα μήνυμα, εκτελείται η μέθοδος `met` της διεργασίας 2 και επιστρέφει το αποτέλεσμα της μεθόδου πίσω στην διεργασία 1 που κάλεσε την μέθοδο σε ένα μήνυμα. Όλη αυτή η διαδικασία όμως δεν φαίνεται στον προγραμματιστή και πραγματοποιείται παρασκηνιακά. Έτσι, η θέση των αντικειμένων πρέπει να είναι διαφανής προς στον χρήστη όταν καλεί τις αντίστοιχες μεθόδους. Η διαφάνεια αυτή δεν περιορίζεται μόνο στην θέση αλλά περιλαμβάνει την διαφάνεια μετανάστευσης των αντικειμένων στο δίκτυο. Συνεπώς, η τεχνολογία κατανεμημένων αντικειμένων πρέπει να κρύβει τις λεπτομέρειες της επικοινωνίας του δικτύου από τον προγραμματιστή. Η διεπαφή λογισμικού που υποστηρίζει την τεχνολογία κατανεμημένων αντικειμένων είναι η Java RMI και θα την παρουσιάζουμε αναλυτικά στο κεφάλαιο 11.

7.2.5 Υπηρεσίες Ιστού και Πλέγματος

Σε αντίθεση με μια παραδοσιακή εφαρμογή Ιστού που σχεδιάζεται για να αλληλεπιδρά με χρήστη πελάτη, μια **υπηρεσία Ιστού** (web service) είναι εφαρμογή (ή τμήμα εφαρμογής) που διατίθεται για να χρησιμοποιείται από άλλη εφαρμογή μέσω Ιστού. Μια υπηρεσία Ιστού ή πλέγματος παρέχει μια διεπαφή υπηρεσίας που δίνει την δυνατότητα στα προγράμματα πελάτες να επικοινωνούν με τους διακομιστές σε ένα γενικό τρόπο από ότι με τους περιηγητές Ιστού. Με άλλα λόγια, τα προγράμματα πελάτες προσπελάζουν τις λειτουργίες που παρέχει η διεπαφή της υπηρεσίας Ιστού ή πλέγματος με μηνύματα εκφρασμένες σε μορφή XML και συνήθως τα μηνύματα αυτά μεταφέρονται μέσω του πρωτοκόλλου Διαδικτύου HTTP.

Στο Σχήμα 7.5 φαίνεται το μοντέλο προγραμματισμού για υπηρεσίες Ιστού και πλέγματος. Ο **παροχέας υπηρεσιών** (service provider) δημοσιεύσει τις υπηρεσίες του σε ένα **κατάλογο υπηρεσιών** (service registry) στο Διαδίκτυο ή στο πλέγμα μέσω της λειτουργίας δημοσίευσης. Η διεργασία ή ο **αιτών υπηρεσιών** (service requestor) που επιθυμεί να χρησιμοποιήσει μια υπηρεσία, πρέπει να επικοινωνήσει με τον κατάλογο υπηρεσιών μέσω της λειτουργίας αναζήτησης και αν είναι διαθέσιμη η υπηρεσία τότε ο αιτών υπηρεσιών θα αποκτήσει μια αναφορά προς στην υπηρεσία που θέλει να καλέσει. Με την χρήση της αναφοράς αυτής, ο αιτών υπηρεσιών αλληλεπιδρά με την υπηρεσία που θέλει μέσω της λειτουργίας σύνδεσης.

Αυτό το μοντέλο είναι ουσιαστικά μια επέκταση του μοντέλου κατανεμημένων αντικειμένων



Σχήμα 7.5: Μοντέλο υπηρεσιών Ιστού και πλέγματος

και λειτουργεί σε μεγαλύτερο και διαλειτουργικό δίκτυο όπως το Διαδίκτυο ή το πλέγμα. Με άλλα λόγια, τα αντικείμενα υπηρεσίες δημοσιεύονται σε ένα καθολικό κατάλογο υπηρεσιών που επιτρέπουν στους αιτούντες υπηρεσιών να αναζητήσουν και να προσπελάσουν τις υπηρεσίες αυτές σε ένα μεγαλύτερο δίκτυο. Επίσης, οι υπηρεσίες δημοσιεύονται και αναζητούνται με την χρήση ενός καθολικού και μοναδικού προσδιοριστή. Σε αυτή την περίπτωση το μοντέλο υπηρεσιών Ιστού και πλέγματος προσφέρει μια μεγαλύτερη αφαίρεση: την διαφάνεια θέσης. Η διαφάνεια θέσης επιστρέπει στο προγραμματιστή εφαρμογών να προσπελάζει ένα αντικείμενο ή μια υπηρεσία χωρίς να χρειάζεται να γνωρίζει την πραγματική θέση του αντικειμένου ή της υπηρεσίας.

Οι τεχνολογίες που βασίζονται σε αυτό το μοντέλο είναι οι SOAP (Simple Object Access Protocol), WSDL (Web Services Defintion Language) και UDDI (Universal Discovery Description Integration) για τις υπηρεσίες Ιστού που παρουσιάζουμε αναλυτικά στο κεφάλαιο 12. Επίσης, στο κεφάλαιο 13 παρουσιάζονται οι τεχνολογίες OGSI (Open Grid Service Infrastructure) και WSRF (Web Services Resource Framework) για τις υπηρεσίες πλέγματος.

7.2.6 Συγκρίσεις Μοντέλων Προγραμματισμού

Μια παράλληλη και κατανεμημένη εφαρμογή μπορεί να υλοποιηθεί χρησιμοποιώντας ένα από τα παραπάνω υποδείγματα προγραμματισμού. Το ερώτημα που ίσως προκύψει είναι το εξής: Δεδομένου των μοντέλων και των εργαλείων προγραμματισμού, ο προγραμματιστής ποιο

κατάλληλο μοντέλο πρέπει να επιλέξει για την υλοποίηση μιας εφαρμογής; Για να απαντήσουμε στην ερώτηση αυτή πρέπει να εξετάζουμε τα μοντέλα κάτω από ορισμένα κριτήρια. Είναι γνωστό ότι κάθε μοντέλο έχει τα δικά της πλεονεκτήματα και μειονεκτήματα.

- Πεδίο εφαρμογής. Για παράλληλες εφαρμογές που απαιτούν παραλληλοποίηση ή περιέχουν απαιτητικούς υπολογισμούς χρησιμοποιούνται τα μοντέλα νήματα και μεταβίβασης μηνυμάτων. Συγκεκριμένα, για εφαρμογές που περιέχουν απλές ανάγκες επικοινωνίας μπορεί να χρησιμοποιηθεί το μοντέλο μεταβίβασης μηνυμάτων ενώ σε πολύπλοκες εφαρμογές που περιέχουν πολύ επικοινωνία μπορεί να χρησιμοποιηθεί το μοντέλο νημάτων. Από την άλλη πλευρά, για κατανεμημένες εφαρμογές που απαιτούν να χρησιμοποιήσουν ή να αλληλεπιδρούν με άλλες υπηρεσίες που προσφέρονται από άλλες εφαρμογές χρησιμοποιούνται τα μοντέλα μεταβίβασης μηνυμάτων, κατανεμημένα αντικείμενα και υπηρεσίες Ίστού - πλέγματος. Το μοντέλο των νημάτων μπορεί να χρησιμοποιηθεί στις κατανεμημένες εφαρμογές μόνο σε περιπτώσεις εκείνες που μια διεργασία θέλει εκτελέσει πολλές δραστηριότητες ταυτόχρονα.

- Επίπεδο αφαίρεσης σε σχέση με την απόδοση. Είναι γνωστό ότι τα μοντέλα που παρουσιάζαμε έχουν κάποιο επίπεδο αφαίρεσης. Το μοντέλο μεταβίβασης μηνυμάτων είναι το χαμηλότερο επίπεδο αφαίρεσης σε σχέση με τα μοντέλα κατανεμημένα αντικείμενα και υπηρεσίες πλέγματος που βρίσκονται σε υψηλότερα επίπεδα αφαίρεσης. Το μοντέλο μεταβίβασης μηνυμάτων σε σχέση με τα άλλα, η βαθύτερη δομή του δικτύου είναι εμφανής μέσω των τρόπων επικοινωνίας που χρησιμοποιούνται για τη σύνδεση διεργασιών όπως διευθύνσεις IP ή υποδοχές και θύρες, που συμμετέχουν στη μεταφορά ενός μηνύματος από τον έναν υπολογιστή στον άλλο. Όμως, πρέπει να σημειώσουμε ότι το μοντέλο μεταβίβασης μηνυμάτων είναι μια αφαίρεση αυτού που πραγματικά συμβαίνει στο επίπεδο του υλισμικού και των γραμμών επικοινωνίας. Οι διευθύνσεις IP ή οι υποδοχές και θύρες είναι αφαιρέσεις μερικών αρκετά δύσχρηστων οντοτήτων χαμηλού επιπέδου. Από την άλλη πλευρά, τα μοντέλα κατανεμημένα αντικείμενα και υπηρεσίες πλέγματος κρύβουν τις βαθύτερες λεπτομέρειες της επικοινωνίας από τον προγραμματιστή και συνεπώς δεν υπάρχει αναφορά στις διευθύνσεις IP ή υποδοχές και θύρες. Τα μοντέλα που έχουν υψηλά επίπεδα αφαίρεσης περιλαμβάνουν κεκαλυμμένο κώδικα, ο οποίος ουσιαστικά πραγματοποιεί την αποστολή μηνυμάτων στις διεργασίες και την διαβίβαση δεδομένων σχετικών με αυτά τα

μηνύματα.

Τα μοντέλα με υψηλά επίπεδα αφαίρεσης προσφέρουν ταχεία υλοποίηση ανάπτυξης των πολύπλοκων κατανεμημένων εφαρμογών αλλά 'πληρώνεται' σε απόδοση. Αυτό συμβαίνει γιατί στα μοντέλα αυτά υπάρχει ο κεκαλυμένος κώδικας που διαχειρίζεται την διαδιεργασιακή επικοινωνία και απαιτεί πρόσθετους πόρους και χρόνο εκτέλεσης. Από την άλλη πλευρά, τα μοντέλα με χαμηλά επίπεδα αφαίρεσης όπως η μεταβίβαση μηνυμάτων χρησιμοποιείται σε εφαρμογές που απαιτούν υψηλή απόδοση ή ελάχιστο χρόνο εκτέλεσης.

- Κλιμάκωση. Η πολυπλοκότητα μιας παράλληλης ή κατανεμημένης εφαρμογής αυξάνεται σημαντικά καθώς αυξάνεται ο αριθμός των διεργασιών ή αντικειμένων. Έτσι στο μοντέλο μεταβίβασης μηνυμάτων, ο προγραμματιστής πρέπει να τροποποιήσει το κώδικα της εφαρμογής ώστε να διαχειριστεί τις διευθύνσεις και την επικοινωνία για κάθε επιπλέον διεργασία ξεχωριστά. Είναι προφανές ότι η πολυπλοκότητα συγγραφής κώδικα αυξάνεται καθώς αυξάνεται ο αριθμός των διεργασιών. Αντίθετα, στα μοντέλα βασισμένα σε υψηλά επίπεδα αφαίρεσης, η πολυπλοκότητα της διαχείρισης επιπλέον διεργασιών ή αντικειμένων αναλαμβάνεται από το εργαλείο λογισμικού. Συνεπώς, μια εφαρμογή μπορεί να συνοδευτεί από ένα μεγάλο αριθμό διεργασιών χωρίς να απαιτείται πρόσθετη πολυπλοκότητα στο κώδικα.

- Διαλειτουργική υποστήριξη. Τα υποδείγματα που είναι αφηρημένα είναι εγγενώς ανεξάρτητες από πλατφόρμες. Τα εργαλεία λογισμικού βασισμένα στα υποδείγματα αφ' ενός μερικά μπορούν να είναι εξαρτώμενες από πλατφόρμα και άλλα να μην είναι. Προκειμένου να παρασχεθεί η γενικότητα, ένα εργαλείο που υποστηρίζει ετερογενείς πλατφόρμες υφίστανται απαραιτήτως την πολυπλοκότητα σε σχέση με εκείνο που υποστηρίζει μια ενιαία και μοναδική πλατφόρμα. Για τον ίδιο λόγο, η σύνταξη προγραμματισμού τείνει επίσης να είναι πιο δυσκίνητη.

  Πολλές από τις τεχνολογίες Java όπως η Java RMI απαιτούν να εκτελεστούν σε πλατφόρμες που περιέχουν μόνο εικονικές μηχανές Java. Αυτό έχει σαν αποτέλεσμα, οι προγραμματιστές που μετέχουν σε μια εφαρμογή πρέπει να την γράψουν στην γλώσσα Java. Σε αντίθεση, με τις τεχνολογίες υπηρεσιών Ιστού και πλέγματος που υποστηρίζουν διαλειτουργική υποστήριξη πλατφόρμων. Συνεπώς, εργαλεία λογισμικού βασισμένα

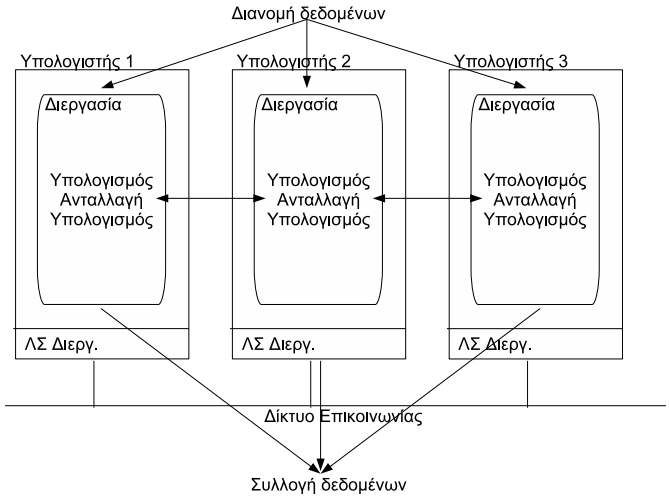
στις τεχνολογίες αυτές μπορούν να υποστηρίξουν προγράμματα που είναι γραμμένα σε διαφορετικές γλώσσες προγραμματισμού και επίσης μπορούν να υποστηρίξουν διεργασίες που εκτελούνται κάτω από διαφορετικές πλατφόρμες λειτουργικών συστημάτων.

7.3 Παράλληλα Υπολογιστικά Μοντέλα

Στη διεθνή βιβλιογραφία της παράλληλης επεξεργασίας υπάρχουν πολλά μοντέλα τα οποία χρησιμοποιούνται τόσο στο προγραμματισμό κοινής μνήμης όσο και στο προγραμματισμό μεταβίβασης μηνυμάτων. Τα περισσότερα παράλληλα προγράμματα βασίζονται σε κάποιο μοντέλο παραλληλισμού για την καλύτερη δόμηση και οργάνωση της δουλειάς τους. Περισσότερες λεπτομέρειες για τα μοντέλα παραλληλισμού αναφέρονται διεξοδικά στο βιβλίο [2]. Επομένως, στην ενότητα αυτή παρουσιάζουμε ορισμένα κοινά παράλληλα μοντέλα τα οποία χρησιμοποιούνται συχνά. Τέλος, πρέπει να σημειώσουμε ότι τα παράλληλα μοντέλα μπορούν να εφαρμοστούν σε οποιοδήποτε πεδίο εφαρμογής.

7.3.1 Μοντέλο Παράλληλων Δεδομένων

Το **μοντέλο παράλληλων δεδομένων** (data-parallel model) είναι ένα από τα πιο απλά παράλληλα μοντέλα. Σε αυτό το μοντέλο, οι εργασίες απεικονίζονται στατικά πάνω στις διεργασίες και κάθε εργασία εκτελεί ίδιες πράξεις πάνω σε διαφορετικά δεδομένα. Αυτό το είδος παραλληλισμού που έχει σαν αποτέλεσμα την εκτέλεση παρόμοιων πράξεων πάνω σε διαφορετικά δεδομένα ταυτόχρονα ονομάζεται **παραλληλισμός δεδομένων** (data parallelism). Αυτό το μοντέλο παραλληλισμού μπορεί να ονομάζεται και SPMD (Single Program Multiple Data). Στο Σχήμα 7.6 παρουσιάζεται το μοντέλο παράλληλων δεδομένων. Ο υπολογισμός μπορεί να εκτελείται σε διάφορες φάσεις και τα δεδομένα που επεξεργάζεται σε κάθε διαφορετική φάση μπορεί να είναι διαφορετικά. Έτσι, οι υπολογιστικές φάσεις εναλλάσονται σποραδικά με συγχρονισμό των εργασιών ή με αναμονή νέων δεδομένων για επεξεργασία. Η διάσπαση του προβλήματος σε εργασίες βασίζεται συνήθως στο σχήμα της **κατάτμησης δεδομένων** (data partitioning) επειδή μια ισομερής κατάτμηση των δεδομένων συνεπάγεται μια στατική απεικόνιση πάνω στις διεργασίες και μας παρέχει καλή **ισορροπία φορτίου** (load balance). Συνεπώς, το μοντέλο SPMD θα είναι αποτελεσματικό αν η κατάτμηση δεδομένων είναι καλά ισομερής και το



Σχήμα 7.6: Μοντέλο παράλληλων δεδομένων

σύστημα υπολογιστών είναι ομοιογενές. Σε περίπτωση που οι επεξεργαστές του συστήματος έχουν διαφορετικές ταχύτητες επεξεργασίας, τότε το μοντέλο SPMD απαιτεί ορισμένα σχήματα εξισορρόπησης φορτίου ώστε η διανομή δεδομένων στις διεργασίες να είναι δυναμική σε χρόνο εκτέλεσης.

Το μοντέλο παράλληλων δεδομένων μπορεί να υλοποιηθεί σε συστήματα διαμοιραζόμενης μνήμης και σε συστήματα συστοιχίας υπολογιστών. Η υλοποίηση του μοντέλου αυτού είναι κατάλληλη σε συστήματα υπολογιστικής συστοιχίας γιατί η υπολογιστική δραστηριότητα της κάθε διεργασίας επικεντρώνεται κυρίως στην δική της περιοχή τοπικών δεδομένων και η επικοινωνία των διεργασιών είναι σχετικά σπάνια. Επιπλέον, η υλοποίηση του μοντέλου παράλληλων δεδομένων είναι κατάλληλη ειδικά για συστήματα διαμοιραζόμενης μνήμης όταν η διανομή των δεδομένων είναι διαφορετική σε διαφορετικές φάσεις του μοντέλου.

7.3.2 Μοντέλο Γράφου Διεργασιών

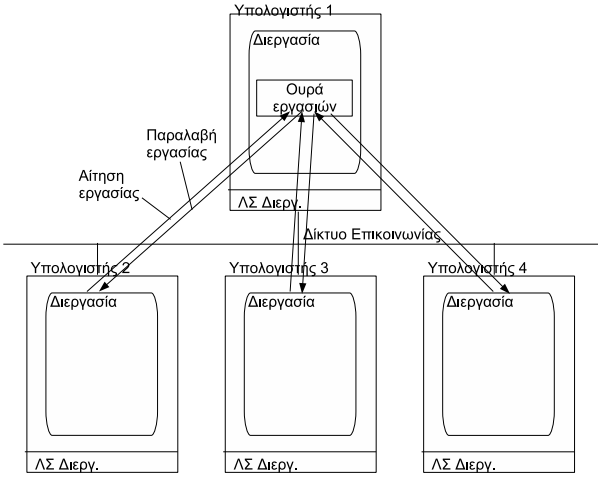
Ένας παράλληλος αλγόριθμος μπορεί να παρασταθεί σαν γράφος διεργασιών που σαφώς χρησιμοποιείται για την απεικόνιση των διεργασιών. Στο **μοντέλο γράφος διεργασιών** (task graph model), χρησιμοποιούνται οι σχέσεις μεταξύ των διεργασιών προκειμένου να μειωθεί το κόστος της επικοινωνίας. Το μοντέλο αυτό χρησιμοποιείται κυρίως για να λύσει προβλήματα στα οποία η ποσότητα των δεδομένων που συσχετίζεται με τις διεργασίες είναι μεγάλη σε σχέση με την ποσότητα υπολογισμού που συσχετίζεται με τις διεργασίες. Οι διεργασίες απεικονίζονται

στατικά έτσι ώστε το κόστος επικοινωνίας μεταξύ των διεργασιών να είναι ελάχιστο. Μερικές φορές χρησιμοποιείται η **αποκεντρωτική** (decentralized) δυναμική απεικόνιση που χρησιμοποιεί πληροφορίες σχετικά με την δομή του γράφου διεργασιών και την δομή επικοινωνίας των διεργασιών ώστε να ελαχιστοποιεί το κόστος της επικοινωνίας.

Υπάρχουν διάφορες τεχνικές για την μείωση της επικοινωνίας που εφαρμόζονται στο μοντέλο γράφου διεργασιών όπως η μείωση του όγκου και της συχνότητας της επικοινωνίας με την αύξηση της τοπικότητας των δεδομένων σε κάθε διεργασία ενώ η απεικόνιση των διεργασιών βασίζεται στην δομή επικοινωνίας μεταξύ των διεργασιών.

7.3.3 Μοντέλο Δεξαμενής Εργασίας

Το **μοντέλο δεξαμενής εργασίας** (work pool model) χρησιμοποιείται όταν οι εργασίες δεν είναι γνωστές εκ των προτέρων, αλλά δημιουργούνται δυναμικά καθώς εκτελείται η εφαρμογή. Για να πετύχουμε εξισορρόπηση φορτίου οι εργασίες πρέπει να απεικονίζονται σε διεργασίες δυναμικά καθώς δημιουργούνται κατά την διάρκεια εκτέλεσης της εφαρμογής. Έτσι, σε αυτό το μοντέλο δεν υπάρχει προ-απεικόνιση των εργασιών στις διεργασίες. Η απεικόνιση των εργασιών μπορεί να είναι **συγκεντρωτική** (centralized) ή **αποκεντρωτική** (decentralized). Στο Σχήμα 7.7 παρουσιάζεται το μοντέλο δεξαμενής εργασίας. Οι εργασίες μπορεί να αποθηκεύονται



Σχήμα 7.7: Μοντέλο δεξαμενής εργασίας

σε μια διαμοιραζόμενη λίστα όπως ουρά προτεραιότητας, πίνακας κατακερματισμού και δένδρο ή μπορούν να αποθηκεύονται σε μια κατανεμημένη δομή δεδομένων. Σε αυτό το μοντέλο, οι

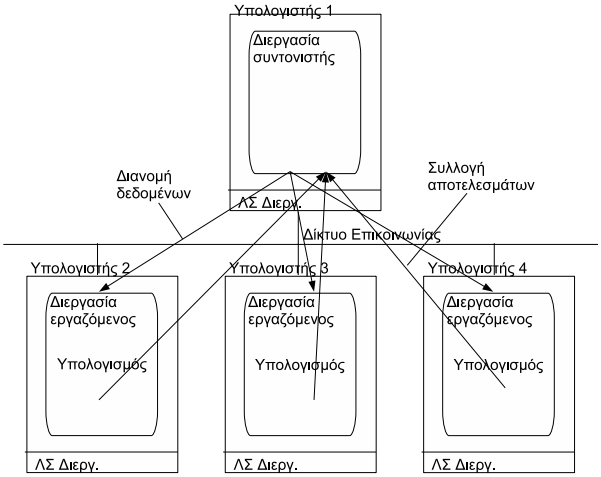
εργασίες αποθηκεύονται σε μια δεξαμενή εργασίας και κάθε διεργασία λαμβάνει μια εργασία από την δεξαμενή και στην συνέχεια πραγματοποιεί τον απαιτούμενο υπολογισμό. Κατά την διάρκεια της εκτέλεσης μιας εργασίας, οι διεργασίες μπορεί να δημιουργήσουν νέες εργασίες, οι οποίες προστίθενται στην δεξαμενή εργασίας. Τέλος, αν εφαρμόζεται στο μοντέλο δεξαμενής εργασίας αποκεντρωτική απεικόνιση, τότε απαιτείται ένας αλγόριθμος αχνίνευσης τερματισμού. Ο αλγόριθμος αυτός έχει σαν σκοπό όλες οι διεργασίες να ανιχνεύσουν την ολοκλήρωση της εφαρμογής (όταν είναι άδεια η δεξαμενή εργασίας) και να τερματίσουν τη λειτουργία τους.

7.3.4 Μοντέλο Συντονιστής - Εργαζόμενος

Το **μοντέλο συντονιστής - εργαζόμενος** (master - worker model) αποτελείται από δύο οντότητες: τον συντονιστή και τους πολλαπλούς εργαζόμενους. Ο συντονιστής είναι υπεύθυνος για την διάσπαση του προβλήματος σε μικρές εργασίες, για την διανομή των εργασιών αυτών στους εργαζόμενους και για την συλλογή αποτελεσμάτων από τους εργαζόμενους για να παράγει το τελικό αποτέλεσμα του προβλήματος. Οι εργαζόμενοι εκτελούν την ακόλουθη διαδικασία σε ένα απλό κύκλο χρόνου: λαμβάνουν μια εργασία, επεξεργάζονται την εργασία και αποστέλουν το αποτέλεσμα της επεξεργασίας στο συντονιστή. Έτσι, η επικοινωνία λαμβάνει χώρα μόνο ανάμεσα στον συντονιστή και στους εργαζόμενους.

Το μοντέλο συντονιστής - εργαζόμενος μπορεί να χρησιμοποιήσει είτε την **στατική διανομή** (static load balancing) ή την **δυναμική διανομή** (dynamic load balancing). Στην πρώτη περίπτωση, η διανομή των εργασιών γίνεται στην αρχή του υπολογισμού όπως φαίνεται στο Σχήμα 7.8. Απο την άλλη μεριά, η δυναμική διανομή χρησιμοποιείται όταν ο αριθμός των εργασιών είναι μεγαλύτερος από τον αριθμό των επεξεργαστών ή όταν δεν γνωρίζουμε από την αρχή της εφαρμογής τον αριθμό των εργασιών.

Το μοντέλο συντονιστής - εργαζόμενος μπορεί να γενικευτεί σε ένα **ιεραρχικό μοντέλο συντονιστής - εργαζόμενος** (hierachical master-worker model). Στο ανώτερο επίπεδο του ιεραρχικού μοντέλου ο συντονιστής διανέμει μεγάλα κομμάτια εργασιών στους συντονιστές του δευτέρου επιπέδου οι οποίοι διαμερίζουν και διανέμουν τις εργασίες στους αντίστοιχους εργαζομένους τους και επίσης κάθε συντονιστής του δευτέρου επιπέδου μπορεί να εκτελέσει κάποιες εργασίες. Το ιεραρχικό μοντέλο είναι κατάλληλο σε συστήματα διαμοιραζόμενης μνήμης ή κατανεμημένης μνήμης εφόσον η αλληλεπίδραση είναι δύο επιπέδων (two-way), δηλαδή ο συν-



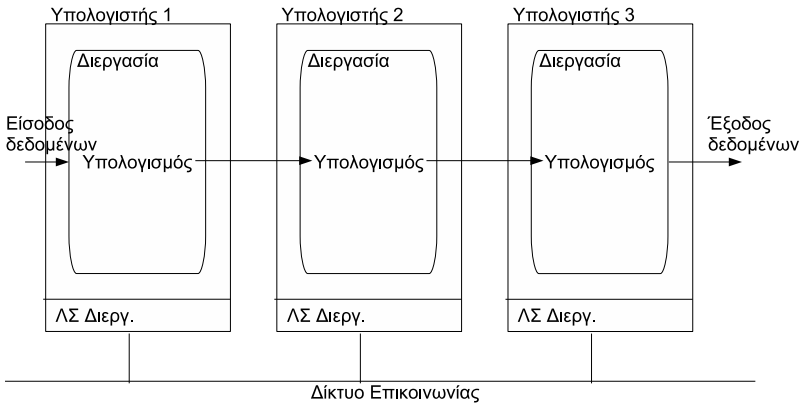
Σχήμα 7.8: Μοντέλο συντονιστής - εργαζόμενος

τονιστής γνωρίζει ότι πρέπει να διανέμει εργασίες και οι εργαζόμενοι γνωρίζουν να λαμβάνουν εργασίες από τον συντονιστή.

Αυτό το μοντέλο πραγματοποιεί υψηλές υπολογιστικές επιταχύνσεις και υψηλούς βαθμούς κλιμάκωσης. Επίσης, όταν χρησιμοποιούμε το μοντέλο συντονιστής - εργαζόμενος πρέπει να λαβούμε υπόψη ότι ο συντονιστής δεν πρέπει να υποστεί **συμφόρηση** (bottleneck) η οποία μπορεί να εμφανιστεί όταν οι εργασίες είναι πολύ μικρές ή όταν οι εργαζόμενοι είναι σχετικά γρήγοροι ή όταν το πλήθος των επεξεργαστών του συστήματος είναι μεγάλος. Σε αυτό το μοντέλο, η **διασπορά των εργασιών** (granularity) πρέπει να είναι τέτοια ώστε το κόστος του υπολογισμού να κυριαρχείται από το κόστος της διανομής των εργασιών και το κόστος του συγχρονισμού.

7.3.5 Μοντέλο Διασωλήνωσης

Στο **μοντέλο διασωλήνωσης** (pipeline model) οι επεξεργαστές ή διεργασίες οργανώνονται σε μερικές κανονικές δομές όπως αυτή του δακτυλίου ή του δισδιάστατου πλέγματος. Διαμέσου αυτής της κανονικής δομής τα δεδομένα μετακινούνται μέσα στη δομή, με κάθε επεξεργαστή να εκτελεί ένα συγκεριμένο τμήμα της συνολικής υπολογιστικής διαδικασίας. Γενικά σε αυτό το μοντέλο (όπως φαίνεται στο Σχήμα 7.9) μια σειρά από επεξεργαστές ή διεργασίες σχηματίζουν μια γραμμική ή πλεγματική διασωλήνωση όπου κάθε επεξεργαστής λαμβάνει μια σειρά από δεδομένα από τον αριστερό επεξεργαστή, τα μετατρέπει κατόπιν επεξεργασίας και μετά στέλνει



Σχήμα 7.9: Μοντέλο διασωλήνωσης

τα αποτελέσματα στον δεξιό επεξεργαστή. Κάθε αρχική τιμή που εισέρχεται στη διασωλήνωση υφίσταται μια σειρά από μετασχηματισμούς και εξέρχεται ως αποτέλεσμα από τη διασωλήνωση. Ο παραλληλισμός προέρχεται από το γεγονός ότι κάθε επεξεργαστής της διασωλήνωσης εκτελεί μετασχηματισμούς την ίδια ακριβώς στιγμή αλλά με διαφορετικά δεδομένα. Στο μοντέλο διασωλήνωσης η απεικόνιση των εργασιών πάνω στους επεξεργαστές είναι στατική.

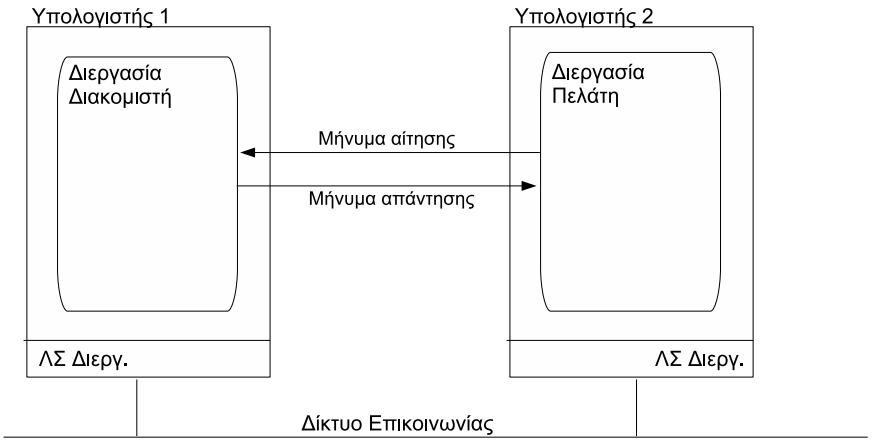
Η εξισορρόπηση του φορτίου είναι συνάρτηση της διασποράς των εργασιών. Έτσι, όσο μεγαλύτερη είναι η διασπορά τόσο αυξάνεται ο παραλληλισμός στην διασωλήνωση. Αντίθετα, αν η διασπορά είναι πολύ μικρή τότε αυξάνεται η επιβάρυνση της επικοινωνίας εφόσον οι επεξεργαστές χρειάζονται να επικοινωνήσουν για να λάβουν νέα δεδομένα μετά από μια μικρή διάρκεια υπολογισμού.

7.4 Κατανεμημένα Μοντέλα

Σε αυτή την ενότητα θα παρουσιάζουμε τα πιο αντιπροσωπευτικά μοντέλα επικοινωνίας που χρησιμοποιούνται για την ανάπτυξη κατανεμημένων εφαρμογών.

7.4.1 Μοντέλο Πελάτη - Διακομιστή

Το πιο συνηθισμένο μοντέλο επικοινωνίας που χρησιμοποιείται για την σύνδεση μεταξύ των υπολογιστών ενός κατανεμημένου συστήματος είναι το **μοντέλο πελάτη - διακομιστή**



Σχήμα 7.10: Μοντέλο πελάτη - διακομιστή

(client - server model). Σε αυτό το μοντέλο ο **διακομιστής** (server) είναι ένα πρόγραμμα που εκτελείται σε έναν αποκλειστικό υπολογιστή και παρέχει υπηρεσίες ενώ οι **πελάτες** (clients) είναι προγράμματα που εκτελούνται στους υπολογιστές χρηστών και λαμβάνουν τις υπηρεσίες. Ο πελάτης και ο διακομιστής μπορεί να εκτελούνται στο ίδιο υπολογιστή ή σε διαφορετικούς υπολογιστές. Η παραπάνω επικοινωνία μεταξύ πελάτη - διακομιστή γίνεται σύμφωνα με ένα γνωστό πρωτόκολλο αίτησης - απάντησης για την καλύτερη συνεργασία τους. Στο πρωτόκολλο αυτό ο πελάτης στέλνει ένα μήνυμα αίτησης προς το διακομιστή, ζητώντας να του παρασχεθεί κάποια υπηρεσία. Ο διακομιστής που περιμένει συνεχώς αιτήσεις από τους πελάτες, λαμβάνει το μήνυμα αυτό, επεξεργάζεται την αίτηση του πελάτη και του επιστρέφει ένα μήνυμα απάντησης. Στο Σχήμα 7.10 παρουσιάζεται το μοντέλο πελάτη - διακομιστή.

Ένα απλό παράδειγμα διακομιστή είναι ένας υπολογιστής που χρησιμοποιείται για την αποθήκευση ιστοσελίδων και σχετικών αρχείων. Αυτός αποστέλει ιστοσελίδες σε υπολογιστές που εκτελούν έναν περιηγητή και ζητούν αρχεία από το διακομιστή. Ένας τέτοιος διακομιστής λέγεται διακομιστής Ιστού και πελάτες του είναι οι υπολογιστές που 'εκτελούν' τους περιηγητές. Η επικοινωνία μεταξύ διακομιστή Ιστού και περιηγητή υλοποιείται από το γνωστό πρωτόκολλο HTTP.

Ο διακομιστής μπορεί να είναι **επαναληπτικός** (iterative) ή **ταυτόχρονος** (concurrent). Ο επαναληπτικός διακομιστής είναι ένας διακομιστής που δέχεται τις αιτήσεις των πελατών και τις επεξεργάζεται ακολουθιακά, ενώ ο ταυτόχρονος διακομιστής είναι ο διακομιστής που

επεξεργάζεται περισσότερες από μια αιτήσεις πελατών ταυτόχρονα.

Επίσης, ο διακομιστής μπορεί να είναι **καταστατικός** (stateful) ή **μη-καταστατικός** (stateless). Ο καταστατικός διακομιστής είναι ο διακομιστής που διατηρεί πληροφορίες ανάμεσα στις αιτήσεις των πελατών ενώ ο μη-καταστατικός διακομιστής είναι εκείνος που δεν έχει την δυνατότητα να διατηρεί τις καταστάσεις των πελατών.

Επίσης, οι διακομιστές μπορούν να διαιρεθούν σε διάφορους τύπους όπως είναι οι:

- **διακομιστές αρχείων** (file servers) που αποθηκεύουν και παρέχουν αρχεία στους πελάτες που αιτούνται.

- **διακομιστές βάσεων δεδομένων** (database servers) αποθηκεύουν μεγάλες δομημένες βάσεις δεδομένων και παρέχουν υπηρεσίες όπως η επεξεργασία των δεδομένων και η ανάκτηση των δεδομένων στα ερωτήματα που θέτουν οι πελάτες.

- **διακομιστές εφαρμογών** (application servers) εκτελούν εξειδικευμένο λογισμικό για κάποια εφαρμογή και μπορούν να απαντήσουν στις διάφορες απιτήσεις των πελατών.

- **διακομιστές Ιστού** (web servers) αποθηκεύουν διάφορα στοιχεία των ιστοσελίδων (όπως κείμενο, εικόνα, ήχο, βίντεο) και παρουσιάζουν τις ιστοσελίδες στους φυλλομετρητές όταν τις ζητούν οι πελάτες.

- **διακομιστές ταχυδρομείου** (mail servers) εκτελούν λειτουργίες όπως η παραλαβή, η αποθήκευση και η αποστολή μηνυμάτων ηλεκτρονικού ταχυδρομείου.

- **διακομιστές εκτυπωτών** (print servers) διαχειρίζονται τον εκτυπωτή ανάμεσα στις αιτήσεις των πελατών.

- **διακομιστές ονομάτων** (name servers) αποθηκεύουν έναν κατάλογο με ονόματα υπηρεσιών ώστε οι υπηρεσίες αυτές να εντοπίζονται εύκολα.

Οι παραπάνω διακομιστές μπορούν να εκτελεστούν σε ένα υπολογιστή ή να καταμεριστούν σε περισσότερους υπολογιστές. Για παράδειγμα, ένας υπολογιστής μπορεί να παρέχει σαν ένας διακομιστής εφαρμογών και ένας διακομιστής βάσεων δεδομένων.

Το μοντέλο πελάτη - διακομιστή μπορεί να διασπαστεί σε τρία λογικά επίπεδα: το **επίπεδο διεπαφής** (user interface level), το **επίπεδο επεξεργασίας** (processing level) και το

επίπεδο δεδομένων (data level). Το επίπεδο διεπαφής είναι υπεύθυνο για την αλληλεπίδραση μεταξύ χρήστη και υπολογιστή. Η αλληλεπίδραση αυτή μπορεί να παρουσιάζεται στο χρήστη σε κατάσταση κειμένου ή ακόμα καλύτερα σε γραφικό περιβάλλον. Το επίπεδο επεξεργασίας είναι υπεύθυνο για την επεξεργασία δεδομένων και το επίπεδο δεδομένων ασχολείται με την αποθήκευση των δεδομένων σε σχεσιακή βάση δεδομένων ή σε κάποια άλλη μορφή.

Ανάλογα με την τοποθέτηση των παραπάνω επιπέδων ανάμεσα στους πελάτες και στους διακομιστές έχουμε δυο δημοφιλείς αρχιτεκτονικές. Η μία είναι η **αρχιτεκτονική δύο επιπέδων** (two-tier architecture) και η άλλη είναι η **αρχιτεκτονική τριών επιπέδων** (three-tier architecture). Η αρχιτεκτονική δύο επιπέδων αποτελείται από δύο επίπεδα: το επίπεδο διεπαφής και επεξεργασίας που υλοποιούνται στην πλευρά του πελάτη και το επίπεδο δεδομένων που εκτελείται στην πλευρά του διακομιστή. Η αρχιτεκτονική αυτή χρησιμοποιείται όταν τα δεδομένα απαιτούν ελάχιστη επεξεργασία.

Η αρχιτεκτονική τριών επιπέδων αποτελείται από τρία επίπεδα: το επίπεδο διεπαφής που εκτελείται στο πελάτη, το επίπεδο επεξεργασίας μπορεί να εκτελεστεί στον ίδιο διακομιστή ή σε ξεχωριστό και το επίπεδο δεδομένων που εκτελείται στο διακομιστή. Με άλλα λόγια, το κάθε επίπεδο τοποθετείται σε διαφορετικούς υπολογιστές. Σε αυτή την αρχιτεκτονική, ο πελάτης στέλνει μια αίτηση στο διακομιστή επεξεργασίας (ή εφαρμογής) και ο διακομιστής αυτός για την ικανοποίηση της αίτησης του πελάτη στέλνει μια αίτηση στον επόμενο διακομιστή δηλαδή στο διακομιστή δεδομένων. Με αυτό αυτό το τρόπο, οι διάφοροι διακομιστές μπορούν να γίνουν πελάτες για άλλους διακομιστές. Η αρχιτεκτονική τριών επιπέδων είναι κατάλληλη για εφαρμογές που απαιτούν απαιτητική επεξεργασία δεδομένων και γι αυτό άλλωστε το επίπεδο επεξεργασίας τοποθετείται σε ξεχωριστό υπολογιστή. Επιπλέον, η αρχιτεκτονική αυτή απομονώνει την τεχνολογία της βάσης δεδομένων που χρησιμοποιείται για την εφαρμογή του τελικού στρώματος. Επίσης, μεταφέρει ένα μεγάλο τμήμα κώδικα από τους πελάτες και τον τοποθετεί στο διακομιστή.

Τέλος, το μοντέλο πελάτη - διακομιστή παρουσιάζει ορισμένα πλεονεκτήματα όπως:

- Ανοικτή αρχιτεκτονική. Το να συνδεθούν υπολογιστές μεταξύ τους χρησιμοποιώντας ένα πρωτόκολλο που όλοι καταλαβαίνουν σημαίνει ότι διάφοροι υπολογιστές, που χρησιμοποιούν διαφορετικά λειτουργικά συστήματα, μπορούν να λειτουργήσουν όλοι μαζί μέσα σε ένα δίκτυο: το μόνο που χρειάζεται είναι ένα λογισμικό σε κάθε υπολογιστή που να

καταλαβαίνει τα πρωτόκολλα που χρησιμοποιούνται.

- Κλιμάκωση. Πολλά συστήματα χρειάζεται να αναπτυχθούν σταδιακά: για παράδειγμα, ένα σύστημα υπολογιστών που χρησιμοποιείται από μία επιτυχημένη εταιρεία πρέπει να ανταποκριθεί σε όλο και μεγαλύτερο όγκο συναλλαγών. Το μοντέλο πελάτη-διακομιστή επιτρέπει την εύκολη προσθήκη διακομιστών, ώστε αυτοί ν' αντεπεξέλθουν σε πιθανή αύξηση των απαιτήσεων σε επεξεργαστή, μνήμη και αρχεία.

- Εξειδίκευση. Με τον σχεδιασμό ενός συστήματος που να περιλαμβάνει πολλούς σχετικά μικρούς υπολογιστές που λειτουργούν ως διακομιστές ένας σχεδιαστής μπορεί να εγγυηθεί ότι ένας συγκεκριμένος υπολογιστής βελτιστοποιείται για ένα συγκεκριμένο σκοπό όπως η διανομή αρχείων, η εκτέλεση υπολογισμών απαιτητικής επεξεργασίας ή η εκτύπωση αρχείων. Αν πολλές υπηρεσίες συγκεντρώνονταν σε έναν και μοναδικό υπολογιστή οι πιθανότητες μιας τέτοιας βελτιστοποίησης θα ήταν ελάχιστες.

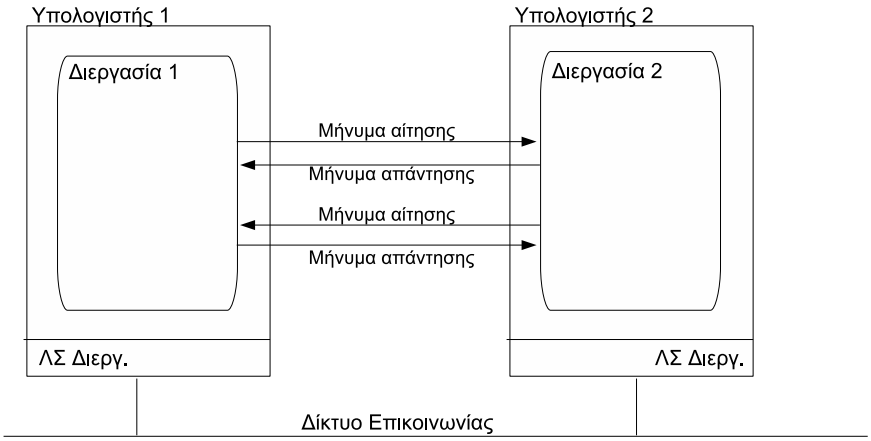
- Αξιοπιστία. Με την περίσσεια δεδομένων και υλισμικού μπορεί να επιτευχθεί υψηλή αξιοπιστία. Για παράδειγμα, η λειτουργία ενός διακομιστή με τέτοιο τρόπο που οι υπηρεσίες του ν' αναλαμβάνονται από έναν πανομοιότυπο διακομιστή, όταν αυτός πάθει βλάβη, επιτέπει τη χρήση συστημάτων πελάτη-διακομιστή στις πιο απαιτητικές εφαρμογές.

- Ευελιξία σχεδιασμού. Το γεγονός ότι τα στοιχεία υλισμικού ενός συστήματος διαμοιράζονται σε πολλούς υπολογιστές σημαίνει ότι ο σχεδιαστής ενός κατανεμημένου συστήματος έχει πολύ περισσότερες επιλογές σχεδιασμού. Για παράδειγμα, μία τεχνική που χρησιμοποιείται για ταχύτερη πρόσβαση σε ένα κατανεμημένο σύστημα είναι η διατήρηση των βάσεων δεδομένων κοντά στον υπολογιστή που τις χρησιμοποιεί. Αυτό συχνά επιτυγχάνεται με την πανομοιότυπη αντιγραφή των βάσεων δεδομένων και την αποθήκευσή τους στο ίδιο τοπικό δίκτυο στο οποίο βρίσκονται οι πελάτες που ζητούν αυτά τα δεδομένα. Αυτή η απόφαση σχεδιασμού δεν θα μπορούσε να ληφθεί μ' ένα σύστημα με συγκεντρωμένο υλισμικό.

7.4.2 Ομότιμο Μοντέλο

Ένα άλλο μοντέλο που χρησιμοποιείται για την οργάνωση κατανεμημένων εφαρμογών είναι το **ομότιμο μοντέλο** (peer-to-peer model) όπως φαίνεται στο Σχήμα 7.11. Στο μοντέλο αυτό

υπάρχουν μέλη ή διεργασίες που παίζουν το ίδιο ρόλο και επικοινωνούν μεταξύ τους ισότιμα σε σχέση με το προηγούμενο μοντέλο όπου υπάρχουν δύο προγράμματα πελάτη και διακομιστή που παίζουν διαφορετικό ρόλο. Έτσι, κάθε ομότιμο μέλος παίζει συγχρόνως το ρόλο του πελάτη και του διακομιστή όπου και οι δύο στέλνουν τις αιτήσεις, επεξεργάζονται τις εισερχόμενες αιτήσεις και στέλνουν τις απαντήσεις. Επίσης, στην αρχιτεκτονική τριών επιπέδων είδαμε ότι ένας διακομιστής μπορεί να είναι ως πελάτης σε ένα άλλο διακομιστή ενώ στο ομότιμο μοντέλο όλα τα μέλη παρέχουν τις ίδιες λογικές υπηρεσίες.



Σχήμα 7.11: Ομότιμο μοντέλο

Το μοντέλο πελάτη - διακομιστή είναι ένα ιδανικό μοντέλο για κεντρικοποιημένες δικτυακές εφαρμογές ενώ το ομότιμο μοντέλο είναι κατάλληλο και δημοφιλές για εφαρμογές διαμοιρασμού ή συνεργατικής εργασίας ή ανταλλαγή μουσικών αρχείων μέσω του Διαδικτύου. Σε αυτές τις εφαρμογές, οι χρήστες χρησιμοποιούν ένα πρόγραμμα που αναζητούν μουσικά αρχεία και όσοι χρήστες τα έχουν επικοινωνούν μαζί τους ώστε να κατεβάζουν τα αρχεία αυτά.

Το ομότιμο μοντέλο μπορεί να υλοποιηθεί με εργαλεία που παρέχουν την μεταβίβαση μηνυμάτων. Για πολύπλοκες εφαρμογές όπως διαμοιρασμό πόρων υπάρχει μια τάση να αναπτυχθούν πρωτόκολλα υψηλού επιπέδου και εργαλεία για την ανάπτυξη τους. Ένα παράδειγμα τέτοιων πρωτοκόλλων και εργαλείων είναι το έργο JXTA που θα παρουσιάζουμε στο κεφάλαιο 14.

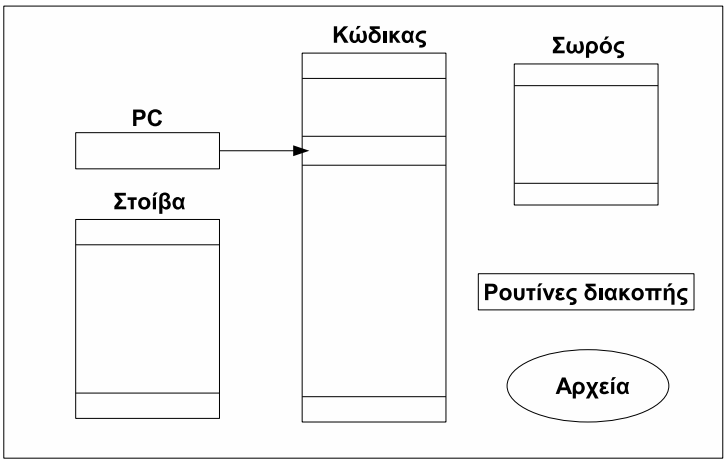
Κεφάλαιο 8

Πολυνηματικός Προγραμματισμός

Σε αυτό το κεφάλαιο, περιγράφουμε τεχνικές προγραμματισμού σε συστήματα κοινής μνήμης με τη χρήση των νημάτων και της ακολουθιακής γλώσσας με οδηγίες μεταγλωττιστή και συναρτήσεις βιβλιοθήκης. Πρώτα ξεκινάμε με την περιγραφή των εννοιών **διεργασία** (process) και **νήμα** (thread). Στην συνέχεια περιγράφουμε την Java Threads που χρησιμοποιείται σε συστήματα κοινής μνήμης, σε συστήματα διπλού-πυρήνα ακόμη και σε συστήματα ενός επεξεργαστή. Τέλος, περιγράφουμε το πρότυπο λογισμικού JOMP που χρησιμοποιείται για παράλληλο προγραμματισμό σε συστήματα κοινής μνήμης και το οποίο χρησιμοποιεί οδηγίες προς το μεταγλωττιστή.

8.1 Διεργασίες και Νήματα

Όπως είναι γνωστό τα λειτουργικά συστήματα βασίζονται πάνω στην έννοια της διεργασίας. Η διεργασία είναι ένα πρόγραμμα υπό εκτέλεση. Έτσι, η διεργασία αποτελείται από ένα **χώρο διευθύνσεων** (address space) και μια ροή ελέγχου του προγράμματος. Ο χώρος διευθύνσεων είναι ένα σύνολο σελίδων μνήμης που πάνω σε αυτές φορτώνονται ο κώδικας του προγράμματος, τα δεδομένα, ο σωρός και η στοίβα. Ενώ η ροή ελέγχου είναι μια ακολουθία εντολών που πρέπει να εκτελεί η διεργασία σε κάποια χρονική στιγμή. Με άλλα λόγια, η διεργασία κρατάει το **μετρητή προγράμματος** (program counter) που δείχνει την διεύθυνση της επόμενης εντολής που θα εκτελέσει. Στο Σχήμα 8.1 φαίνεται τα βασικά μέρη μιας διεργασίας. Από τα παραπάνω προκύπτει ότι όταν θέλουμε να εκτελέσουμε ένα πρόγραμμα ή μια εφαρμογή σε ένα

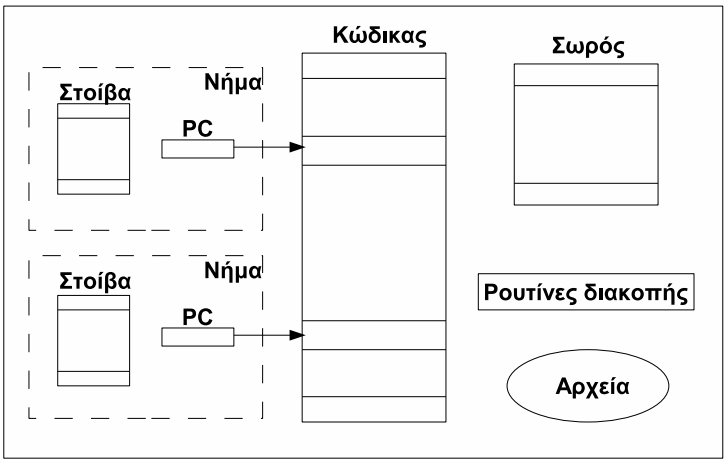


Σχήμα 8.1: Τα μέρη μιας διεργασίας

επεξεργαστή πρέπει να δημιουργηθεί η διεργασία.

Είναι γνωστό ότι στα συστήματα ενός επεξεργαστή εκτελούνται πολλές διεργασίες για την επίτευξη του **πολυπρογραμματισμού** (multiprogramming). Σε αυτή την περίπτωση οι διεργασίες εκτελούνται ψευδοπαράλληλα σε ένα μόνο επεξεργαστή εναλλάσοντας από μια διεργασία στην άλλη. Η εναλλαγή αυτή υλοποιείται από ένα σχήμα **χρονοδρομολόγησης** (scheduling) και συμβαίνει κατά τακτά χρονικά διαστήματα ή όταν μια διεργασία καθυστερείται. Στην περίπτωση ενός συστήματος πολυεπεξεργαστών οι διεργασίες εκτελούνται ταυτόχρονα σε διάφορους επεξεργαστές. Η δημιουργία και η επικοινωνία μεταξύ των διεργασιών μέσω του κοινού χώρου των διευθύνσεων (ή διαμοιραζόμενων μεταβλητών) μπορεί να επιτευχθεί με την χρήση κλήσεων συστήματος που παρέχει το λειτουργικό σύστημα. Για παράδειγμα, το λειτουργικό σύστημα UNIX παρέχει μια κλήση συστήματος `fork()` για την δημιουργία μιας διεργασίας και κλήσεις `wait()` και `exit()` για το τερματισμό της διεργασίας. Με τις κλήσεις αυτές μπορούμε να υλοποιήσουμε παράλληλα προγράμματα κοινής μνήμης σε συστήματα ενός επεξεργαστή και πολυεπεξεργαστών.

Όμως, η εκτέλεση πολλών διεργασιών σε ένα μόνο επεξεργαστή παρουσιάζει χαμηλή ταχύτητα εκτέλεσης. Αυτό συμβαίνει λόγω του υψηλού κόστους υλοποίησης των διεργασιών, δηλαδή έχει υψηλό κόστος για την δημιουργία και την εναλλαγή των διεργασιών. Έτσι, για την δημιουργία μίας νέας διεργασίας απαιτεί τη δημιουργία ενός νέου χώρου διευθύνσεων (δηλαδή μεταβλητές, στοίβα, σωρός κλπ) ενώ για την εναλλαγή των διεργασιών απαιτεί την εναλλαγή



Σχήμα 8.2: Δύο νήματα στην ίδια διεργασία

περιβάλλοντος. Η εναλλαγή περιβάλλοντος σημαίνει ότι πρέπει να αποθηκεύσει και να φορτώσει τα περιεχόμενα των καταχωρητών του επεξεργαστή, να τροποποιήσει τα περιεχόμενα των καταχωρητών του διαχειριστή μνήμης και να αλλάξει τις σελίδες μνήμης.

Η συχνή όμως δημιουργία ενός νέου χώρου διευθύνσεων για κάθε διεργασία δεν είναι επιθυμητή και για αυτό το λόγο είναι αποτελεσματικό να έχουμε περισσότερες της μιας ροές ελέγχου μέσα στην ίδια διεργασία, που λέγονται **νήματα** (threads) ή αλλιώς **ελαφριές διεργασίες** (lightweight processes). Συνεπώς, τα νήματα είναι ροές ελέγχου και διαμοιράζουν το κοινό χώρο διευθύνσεων της διεργασίας. Στο Σχήμα 8.2 φαίνονται δύο νήματα μιας διεργασίας και τα μέρη ενός νήματος. Κάθε νήμα έχει το δικό του μετρητή προγράμματος που δείχνει την επόμενη εντολή του νήματος που θα εκτελεστεί. Επίσης, κάθε νήμα έχει τη δική του στοίβα και κρατά πληροφορίες για τους καταχωρητές. Επίσης, όλα τα νήματα μιας διεργασίας διαμοιράζουν το κώδικα και τα δεδομένα αφού αξιοποιούν το κοινό χώρο των διευθύνσεων. Τέλος, ένα πρόγραμμα ονομάζεται **πολυνηματικό** (multithreading) όταν εκτελούνται πολλά νήματα **συγχρονικά** (concurrent) σε μια διεργασία.

Τα διάφορα νήματα όπως και στις διεργασίες μπορούν να εκτελούνται ψευδοπαράλληλα σε ένα μόνο επεξεργαστή ή να εκτελούνται ταυτόχρονα σε πολλούς επεξεργαστές ενός συστήματος κοινής μνήμης. Η ταχύτητα εκτέλεσης των νημάτων στα παραπάνω συστήματα σαφώς είναι καλύτερη διότι η δημιουργία των νημάτων παίρνει λιγότερο χρόνο σε σχέση με την δημιουργία των διεργασιών. Επίσης, ένα νήμα έχει άμεση πρόσβαση στα δεδομένα (ή στις διαμοιραζόμενες

μεταβλητές) της διεργασίας πράγμα που ταιριάζει καλύτερα με το μοντέλο προγραμματισμού κοινής μνήμης. Τέλος, η επικοινωνία και ο συγχρονισμός των νημάτων υλοποιείται πολύ αποτελεσματικά από ότι τις διεργασίες. Τα θέματα επικοινωνίας και συγχρονισμού νημάτων θα παρουσιάζουμε παρακάτω.

Οι κατασκευαστές έχουν χρησιμοποιήσει νήματα στα λειτουργικά τους συστήματα επειδή τα νήματα αυτά προσφέρουν μια ταχύτερη και απλή λύση για το χειρισμό συγχρονικών δραστηριοτήτων μέσα στο λειτουργικό σύστημα. Οτιδήποτε δραστηριότητα ενός νήματος καθυστερείται ή μπλοκάρεται όπως αναμονή για λειτουργία Εισόδου/Εξόδου (Ε/Ε), τότε το αντικαθιστά άλλο νήμα. Παραδείγματα πολυνηματικών λειτουργικών συστημάτων είναι οι SUN Solaris, IBM AIX, SGI IRIX και Windows XP. Τα λειτουργικά αυτά συστήματα προσφέρουν στους προγραμματιστές εργαλεία για να χρησιμοποιήσουν τα νήματα στις εφαρμογές τους αλλά κάθε σύστημα τα χειρίζεται διαφορετικά. Ευτυχώς, υπάρχουν διαθέσιμα πρότυπα λογισμικού ή ενδιάμεσου λογισμικού για τα νήματα όπως είναι το διαδεδομένο πρότυπο Pthreads (από την IEEE Portable Operating System Interface, POSIX), η Java Threads και JOMP. Σε αυτό το κεφάλαιο θα επικεντρωθούμε στο λογισμικό Java Threads και JOMP.

8.2 Χρήσεις των Νημάτων

Έχοντας εξηγήσει τι είναι τα νήματα και τη διαφορά τους με τις διεργασίες, θα αναφέρουμε εδώ τους λόγους για τους οποίους χρησιμοποιούμε τα νήματα. Ένας λόγος που χρησιμοποιούμε τα νήματα είναι ότι σε πολλές εφαρμογές υπάρχουν δραστηριότητες που εκτελούνται παράλληλα. Μερικές απο αυτές είναι δυνατόν να μπλοκάρονται περιστασιακά. Έτσι, χωρίζοντας την εφαρμογή σε πολλά σειριακά νήματα που εκτελούνται ψευδοπαράλληλα. Για παράδειγμα, αν σε μια εφαρμογή απαιτεί να διαβάζει δεδομένα από ένα αρχείο και ταυτόχρονα να επεξεργάζεται τα δεδομένα, μπορούμε να χρησιμοποιήσουμε δύο νήματα, το ένα νήμα για την ανάγνωση δεδομένων από το αρχείο και το άλλο για την επεξεργασία προκειμένουν να έχουμε μια ψευδοπαράλληλη εκτέλεση και των δύο δραστηριοτήτων.

Ένας άλλος λόγος είναι ότι τα νήματα εύκολα αξιοποιούνται σε συστήματα πολυεπεξεργαστών, όπου αναδεικνύει το πραγματικό παραλληλισμό στις πιο απαιτητικές εφαρμογές. Με αυτό το τρόπο επιταχύνουμε την εκτέλεση εφαρμογών που απαιτούν μεγάλο όγκο επεξεργασίας δεδομένων. Ο παραλληλισμός προκύπτει από την διάσπαση μιας εφαρμογής σε πολ-

λές μικρότερες εργασίες ή νήματα που θα εκτελούνται παράλληλα σε ένα υπολογιστή ή σε ένα σύστημα πολυεπεξεργαστών χωρίς καμία αλλαγή στην εφαρμογή. Συνεπώς, το προγραμματιστικό μοντέλο κοινής μνήμης που θα ασχοληθούμε σε αυτό κεφάλαιο με την βοήθεια των νημάτων γίνεται απλούστερο.

Τέλος, τα νήματα δεν έχουν στην κατοχή τους πόρους, οπότε η δημιουργία και η καταστροφή τους είναι ευκολότερη από ότι στις διεργασίες. Η ιδιότητα αυτή είναι εξαιρετικά χρήσιμη όταν χρειάζονται δυναμικές και γρήγορες αλλαγές στον αριθμό των νημάτων.

8.3 Υλοποίηση Νημάτων στην Java

Η Java Threads παρέχει πλήρη ενσωματωμένη υποστήριξη για τα νήματα. Κάθε εφαρμογή Java έχει τουλάχιστον ένα κύριο νήμα ελέγχου το οποίο υποστηρίζει την εκτέλεση της μεθόδου `main()` της εφαρμογής. Το κύριο αυτό νήμα μπορεί να δημιουργήσει άλλα νήματα ελέγχου που εκτελούν συγκεκριμένες εργασίες παράλληλα.

Στην Java υπάρχουν δύο τρόποι για να δημιουργήσουμε νήματα:

- Δημιουργούμε μια κλάση που επεκτείνει την κλάση `Thread` της Java που βρίσκεται στο πακέτο `java.lang`.
- Δημιουργούμε μια κλάση που υλοποιεί την διασύνδεση `Runnable`.

Παρακάτω θα παρουσιάζουμε τους δύο τρόπους αναλυτικά.

8.3.1 Δημιουργία Νήματος με την Κλάση `Thread`

Για την δημιουργία ενός νήματος με την κλάση `Thread` απαιτούνται τρία βήματα:

1. Στην Java τα νήματα υλοποιούνται σαν αντικείμενα που περιέχουν μια μέθοδο η οποία λέγεται `run()`. Για το λόγο αυτό δημιουργούμε μια νέα κλάση η οποία επεκτείνει την κλάση `Thread` και παρακάμπτουμε τη μέθοδο `run()`, βάζοντας τον κώδικα ή τις εντολές που θέλουμε να εκτελεί το νήμα. Για παράδειγμα, η τυπική δομή της κλάσης νήματος φαίνεται παρακάτω:

```
class MyThread extends Thread
```

```
{
    public void run()
    {
        // thread body of execution
    }
}
```

2. Έπειτα στην μέθοδο `main()` μιας άλλης εφαρμογής Java δημιουργούμε ένα αντικείμενο της κλάσης του νήματος που δημιουργήσαμε στο προηγούμενο βήμα. Για παράδειγμα, για να δημιουργήσουμε ένα αντικείμενο της κλάσης `MyThread` γράφουμε την παρακάτω δήλωση:

```
MyThread thr1 = new MyThread();
```

3. Τέλος, ξεκινάμε ένα νήμα (ή περισσότερα νήματα) με την κλήση της μεθόδου `start()` του αντικειμένου νήματος, η οποία η εικονική μηχανή της Java καλεί αυτόματα την μέθοδο `run()` του νήματος. Για παράδειγμα, η εκκίνηση της εκτέλεσης του νήματος `thr1` γίνεται με την δήλωση:

```
thr1.start();
```

Πρέπει να σημειώσουμε ότι η μέθοδος `run()` η οποία δηλώνεται πάντα ως `public`, δεν δέχεται παραμέτρους και δεν επιστρέφει τιμή. Παρακάτω παρουσιάζεται ένα πρόγραμμα μέσα στο οποίο δημιουργούνται τρία νήματα. Κάθε νήμα να εμφανίζει το όνομα του.

```
class MyThread1 extends Thread
{
        // K'wdikas pou ja ektele'i k'aje n'hma
      public void run()
      {
            // Emfan'isei to 'onoma tou k'aje n'hmatos
            System.out.println("My␣name␣is␣" + getName());
      }
}

class ThreadEx1
{
      public static void main(String[] args)
      {
            // Dhmiourg'ia nhm'atwn
            MyThread1 thr1 = new MyThread1();
            MyThread1 thr2 = new MyThread1();
            MyThread1 thr3 = new MyThread1();
```

```
            // Ekk'inhsh ekt'eleshs nhm'atwn
            thr1.start();
            thr2.start();
            thr3.start();
        }
}
```

Για την εμφάνιση του ονόματος κάθε νήματος χρησιμοποιήσαμε την μέθοδο `getName()`. Το όνομα ενός νήματος μπορεί να του ανατίθεται κατά τη δημιουργία του καθώς μερικές μέθοδοι κατασκευαστές νημάτων έχουν ένα όρισμα τύπου `string` το οποίο αναπαριστά το όνομα του νήματος. Για παράδειγμα, μπορούμε να δημιουργήσουμε ένα νήμα με όνομα ως '`Nhma1`' με την παρακάτω εντολή:

```
MyThread thr1 = new MyThread("Nhma1");
```

8.3.2 Δημιουργία Νήματος με την Διασύνδεση `Runnable`

Ο τρόπος αυτός είναι παρόμοιος με το τρόπο που περιγράψαμε στην προηγούμενη ενότητα με ελάχιστες διαφορές. Έτσι, για την δημιουργία ενός νήματος με την διασύνδεση `Runnable` απαιτούνται τέσσερα βήματα:

1. Δημιουργούμε μια νέα κλάση η οποία υλοποιεί την διασύνδεση `Runnable` η οποία περιέχει μόνο μια μέθοδο, τη μέθοδο `run()`. Για παράδειγμα, η τυπική δομή της κλάσης νήματος φαίνεται παρακάτω:

   ```
   class MyThread implements Runnable
   {
       public void run()
       {
           // thread body of execution
       }
   }
   ```

2. Έπειτα στην μέθοδο `main()` μιας άλλης εφαρμογής Java δημιουργούμε ένα αντικείμενο της κλάσης `Runnable` που δημιουργήσαμε στο προηγούμενο βήμα. Για παράδειγμα, για να δημιουργήσουμε ένα αντικείμενο της κλάσης `MyThread` γράφουμε την παρακάτω δήλωση:

   ```
   MyThread myObject = new MyThread();
   ```

3. Στην συνέχεια, δημιουργούμε ένα αντικείμενο της κλάσης `Thread` και περνάμε το αντικείμενο `Runnable` που δημιουργήσαμε στο προηγούμενο βήμα σαν όρισμα στον κατασκευαστή `Thread`. Για παράδειγμα, η δημιουργία αντικειμένου τύπου `Thread` γίνεται με την δήλωση:

   ```
   Thread thr1 = new Thread (myObject);
   ```

4. Τέλος, ξεκινάμε ένα νήμα καλώντας τη μέθοδο `start()` του αντικειμένου νήματος, η οποία καλεί αυτόματα την μέθοδο `run()` που περιέχει τον κώδικα για την εργασία του νήματος. Για παράδειγμα, η εκκίνηση της εκτέλεσης του νήματος `thr1` γίνεται με την δήλωση:

   ```
   thr1.start();
   ```

Παρακάτω παρουσιάζεται ένα παρόμοιο πρόγραμμα όπως στο προηγούμενο παράδειγμα.

```
class MyThread2 implements Runnable
{
        private int myid;

        // Kataskeuast'hs 'opou pern'ame thn taut'othta n'hmatos
        MyThread2(int myid)
        {
                this.myid = myid;
        }

        // K'wdikas pou ja ektele'i k'aje n'hma
        public void run()
        {
                // Emfan'isei to 'onoma kai thn taut'othta tou k'aje n'hmatos
                System.out.println("My␣name␣is␣" + Thread.currentThread().getName() + "␣and␣my␣identifier␣
        }
}

class ThreadEx2
{
        public static void main(String[] args)
        {
                // Dhmiourg'ia antikeim'enwn Runnable
                MyThread2 myObj1 = new MyThread2(0);
                MyThread2 myObj2 = new MyThread2(1);
                MyThread2 myObj3 = new MyThread2(2);
                // Dhmiourg'ia nhm'atwn
                Thread thr1 = new Thread(myObj1);
                Thread thr2 = new Thread(myObj2);
                Thread thr3 = new Thread(myObj3);
                // Ekk'inhsh ekt'eleshs nhm'atwn
```

```
                thr1.start();
                thr2.start();
                thr3.start();
        }
}
```

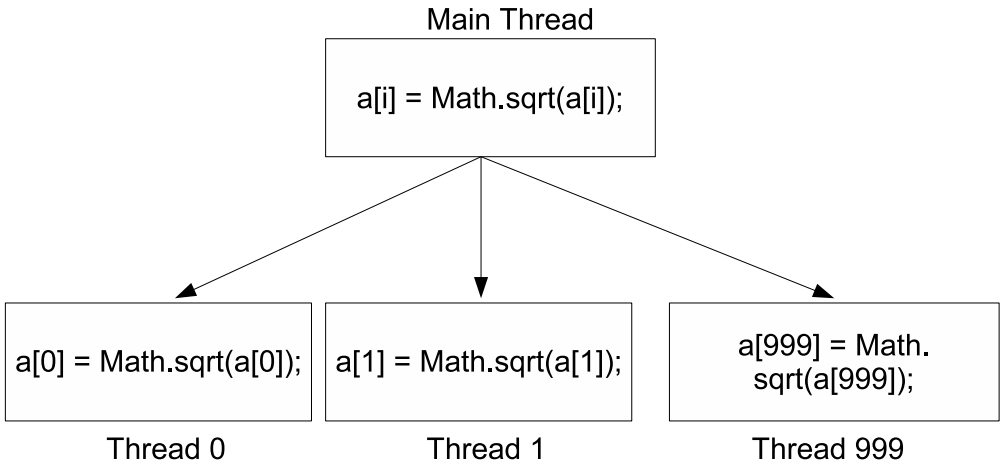
Στην κλάση `MyThread2` ορίζαμε ένα κατασκευαστή ώστε να περάσουμε ένα όρισμα ή μια σειρά παραμέτρων σε κάθε νήμα. Στο παράδειγμα μας, περνάμε ως όρισμα σε κάθε νήμα την ταυτότητα του νήματος. Αυτό είναι ιδιαίτερο χρήσιμο στο προγραμματιστή όπως θα δούμε στα επόμενα παραδείγματα. Επίσης, μέσα στην μέθοδο `run()` χρησιμοποιήσαμε την μέθοδο `currentThread()` η οποία επιστρέφει μια αναφορά προς το τρέχον (ή εκτελούμενο) νήμα.

8.4 Παραλληλισμός Δεδομένων και Ελέγχου

Σε αυτή την ενότητα θα περιγράψουμε δύο διαδεδομένες τεχνικές για την οργάνωση παράλληλων προγραμμάτων που υλοποιούνται σε συστήματα κοινής μνήμης ή σε νήματα κοινής μνήμης. Οι τεχνικές αυτές είναι ο **παραλληλισμός δεδομένων** (data parallelism) και ο **παραλληλισμός ελέγχου** (control parallelism).

8.4.1 Παραλληλισμός Δεδομένων

Η τεχνική του παραλληλισμού δεδομένων μπορεί να ορισθεί ως η ταυτόχρονη εκτέλεση της ίδιας λειτουργίας πάνω σε διαφορετικά δεδομένα. Με άλλα λόγια, οι επεξεργαστές ενός συστήματος κοινής μνήνης ή τα νήματα εκτελούν την ίδια λειτουργία πάνω σε διαφορετικά δεδομένα παράλληλα. Η τεχνική αυτή είναι πάρα πολλή απλή για δύο λόγους: Πρώτον, είναι εύκολη στο προγραμματισμό και δεύτερον, η τεχνική αυτή μπορεί εύκολα να κλιμακωθεί σε μεγαλύτερα προβλήματα που βασίζονται στην χρήση μεγάλων πινάκων και δομών δεδομένων. Το παράδειγμα που παρουσιάζαμε στην προηγούμενη ενότητα είναι ένα πολύ απλό παράδειγμα παραλληλισμού δεδομένων αφού όλα τα νήματα εκτελούν την ίδια λειτουργία όπως η εμφάνιση του ονόματος αλλά δεν επεξεργάζονται δεδομένα. Για ακόμα καλύτερη κατανόηση της τεχνικής του παραλληλισμού δεδομένων εισάγουμε ένα άλλο απλό παράδειγμα υπολογισμού. Θέλουμε το παρακάτω τμήμα ενός ακολουθιακού προγράμματος Java που υπολογίζει την τετραγωνική ρίζα κάθε στοιχείου ενός πίνακα να μετατραπεί σε παράλληλο πρόγραμμα χρησιμοποιώντας την τεχνική του



Σχήμα 8.3: Παραλληλισμός δεδομένων

παραλληλισμού δεδομένων:

```
for(i = 0; i < 1000; i++)
  a[i] = Math.sqrt(a[i]);
```

Η εντολή `a[i] = Math.sqrt(a[i])` θα μπορούσε να εκτελεστεί ταυτόχρονα από πολλαπλούς διαφορετικούς επεξεργαστές ή νήματα, που το καθένα θα χρησιμοποιεί ένα διαφορετική δείκτη i $(0 < i < 1000)$, όπως φαίνεται στο Σχήμα 8.3. Συνεπώς, το παραπάνω πρόγραμμα μπορεί να μετατραπεί από ακολουθιακή σε παράλληλη μορφή, υπολογίζοντας την τετραγωνική ρίζα και των 1000 στοιχείων του πίνακα δημιουργώντας 1000 παράλληλα νήματα:

```
class SqrtThread extends Thread
{
        private double [] table;
        private int index;

        // Kataskeuast'hs
        public SqrtThread(double [] array, int ind)
        {
                table = array;
                index = ind;
        }

        // K'aje n'hma upolog'izei thn tetragwnik'h en'os stoiqe'iou p'inaka
        public void run()
        {
                table[index] = Math.sqrt(table[index]);
        }
```

```
}

class ThreadParSqrt
{
        public static void main(String[] args)
        {
                // Dhmiourg'ia kai arqikopo'ihsh p'inaka
                double[] a = new double[1000];
        for(int i = 0; i < 1000; i++)
                a[i] = i;

                // Dhmiourg'ia p'inaka nhm'atwn
                SqrtThread threads[] = new SqrtThread[1000];

                // Dhmiourg'ia kai ekk'inhsh nhm'atwn
                for(int i = 0; i < 1000; i++)
                {
                        threads[i] = new SqrtThread(a,i);
                        threads[i].start();
                }

                // Emf'anish stoiqe'iwn p'inaka
                for(int i = 0; i < 1000; i++)
                {
                        System.out.println(a[i]);
                }
        }
}
```

Στο παραπάνω πρόγραμμα είναι παρόμοιο με τα προηγούμενα προγράμματα με την διαφορά είναι ότι όταν θέλουμε να δημιουργήσουμε πολλά νήματα μπορούμε να τα καταχωρήσουμε σε ένα πίνακα. Επίσης, το κύριο νήμα ελέγχου που αντιστοιχεί στην μέθοδο `main` δημιουργεί 1000 αντίγραφα της κλάσης `SqrtThread` που περιέχει την εντολή ανάθεσης και μετά ξεκινά ξεχωριστά παράλληλα νήματα που το καθένα έχει μια μοναδική τιμή του δείκτη i (όπου περνάμε το δείκτη i σαν παράμετρος) ώστε να χρησιμοποιηθεί για την πρόσβαση στο στοιχείο του πίνακα a που του αφορά. Επίσης, σε κάθε νήμα περνάμε σαν παράμετρος μια αναφορά του πίνακα a ο οποίος βρίσκεται στην κοινή μνήμη της διεργασίας (ή θα μπορούσε να βρίσκεται στην κύρια μνήμη ενός πραγματικού συστήματος πολυεπεξεργαστών) ώστε να είναι εύκολα προσβάσιμος από τα νήματα. Τα νήματα λειτουργούν παράλληλα και κάθε νήμα να επεξεργάζεται ένα διαφορετικό στοιχείο του πίνακα.

Στο παραπάνω πρόγραμμα όπως και άλλα προγράμματα που εκτελούν πάρα πολλά νήματα

περιορίζουν την ταχύτητα εκτέλεσης με αποτέλεσμα να πετύχουμε την ίδια ή ακόμα χειρότερη απόδοση σε σχέση με ένα νήμα. Αυτό συμβαίνει διότι υπάρχει σημαντική καθυστέρηση στην δημιουργία πολλών νημάτων από ότι ο χρόνος επεξεργασίας του κάθε νήματος αφού το κάθε νήμα επεξεργάζεται ένα μόνο στοιχείο του πίνακα (όπως στο προηγούμενο παράδειγμα). Για να αντιμετωπισθεί το πρόβλημα αυτό θα πρέπει ο χρόνος επεξεργασίας του κάθε νήματος να είναι μεγαλύτερος από τον χρόνο δημιουργίας των νημάτων. Με άλλα λόγια, θα πρέπει να αυξήσουμε το μέγεθος του νήματος δηλαδή να επεξεργάζεται πολλά στοιχεία δεδομένων ώστε να δημιουργήσουμε λιγότερα νήματα. Έτσι, στο προηγούμενο παράδειγμα το κάθε νήμα να υπολογίζει την τετραγωνική ρίζα για 100 στοιχεία του πίνακα ακολουθιακά αντί ένα στοιχείο και να δημιουργήσουμε συνολικά 10 νήματα. Συνεπώς, θα έχουμε 10 σχετικά μεγάλου μεγέθους νήματα αντί για 1000 μικρότερου μεγέθους νήματα. Παρακάτω παρουσιάζεται ο τροποποιημένος κώδικας του προηγούμενου παραδείγματος.

```
class SqrtGroupThread extends Thread
{
        private double [] table;
        private int index;
        private int myrank;

        // Kataskeuast'hs
        public SqrtGroupThread(double [] array, int rank, int ind)
        {
                table = array;
                index = ind;
                myrank = rank;
        }

        // K'aje n'hma upolog'izei thn tetragwnik'h r'iza twn 100 stoiqe'iwn
        // p'inaka seiriak'a
        public void run()
        {
                for(int i=index; i<(myrank*100)+100; i++)
                        table[i] = Math.sqrt(table[i]);
        }
}

class ThreadParGroupSqrt
{
        public static void main(String[] args)
        {
                // Dhmiourg'ia kai arqikopo'ihsh p'inaka
                double[] a = new double[1000];
```

```
                for(int i = 0; i < 1000; i++)
                        a[i] = i;

                // Dhmiourg'ia p'inaka nhm'atwn
                SqrtGroupThread threads[] = new SqrtGroupThread[10];

                // Dhmiourg'ia kai ekk'inhsh nhm'atwn
                for(int i = 0, j = 0; i < 10; i++,j += 100)
                {
                        threads[i] = new SqrtGroupThread(a,i,j);
                        threads[i].start();
                }

                // Emf'anish stoiqe'iwn p'inaka
                for(int i = 0; i < 1000; i++)
                {
                        System.out.println(a[i]);
                }
        }
}
```

Οι αλλαγές στο παραπάνω πρόγραμμα είναι οι εξής: Πρώτα, προσθέσαμε ένα μετρητή j που αυξάνεται κατά 100 στην δεύτερη εντολή `for`, ώστε να μετατρέπει τις τιμές του πίνακα νημάτων σε ομάδες των 100 στοιχείων ανά νήμα. Έτσι, δημιουργούνται μόνο 10 νήματα που το κάθε νήμα επεξεργάζεται για 100 στοιχεία. Δηλαδή, το πρώτο νήμα επαναλαμβάνεται ακολουθιακά για τις τιμές από 0 έως 99, το δεύτερο νήμα επαναλαμβάνεται για τις τιμές από 100 έως 199, το τρίτο νήμα επαναλαμβάνεται για τις τιμές από 200 έως 299 και ούτω καθεξής. Επίσης, προσθέσαμε μια εντολή επανάληψης `for` στον κώδικα της μεθόδου `run` ώστε το κάθε νήμα να υπολογίζει την τετραγωνική ρίζα για 100 στοιχεία του πίνακα.

8.4.2 Παραλληλισμός Ελέγχου

Η τεχνική του παραλληλισμού ελέγχου επικεντρώνεται στην εφαρμογή διαφορετικών λειτουργιών σε διαφορετικά δεδομένα. Με άλλα λόγια, εκτελούνται πολλά παράλληλα νήματα που το καθένα εκτελεί διαφορετική λειτουργία σε διαφορετικά δεδομένα. Παρακάτω, παρουσιάζουμε ένα απλό παράδειγμα παραλληλισμού ελέγχου με την βοήθεια των νημάτων της Java.

```
class ThreadAdd extends Thread
{
        private int num1, num2;
        // Kataskeuast'hs
```

```
        public ThreadAdd(int n1, int n2)
        {
                num1 = n1;
                num2 = n2;
        }

        // N'hma pou upolog'izei to 'ajroisma d'uo arijm'wn
        public void run()
        {
                System.out.println("Sum␣=␣" + (num1 + num2));
        }
}

class ThreadSub extends Thread
{
        private int num1, num2;
        // Kataskeuast'hs
        public ThreadSub(int n1, int n2)
        {
                num1 = n1;
                num2 = n2;
        }

        // N'hma pou upolog'izei thn afa'iresh d'uo arijm'wn
        public void run()
        {
                System.out.println("Sub␣=" + (num1 - num2));
        }
}

class ThreadMult extends Thread
{
        private int num1, num2, num3;
        // Kataskeuast'hs
        public ThreadMult(int n1, int n2, int n3)
        {
                num1 = n1;
                num2 = n2;
                num3 = n3;
        }

        // N'hma pou upolog'izei to gin'omeno tri'wn arijm'wn
        public void run()
        {
                System.out.println("Mult␣=␣" + (num1 * num2 * num3));
        }
}
```

```
class ThreadControl
{
        public static void main(String[] args)
        {
                // Dhmiourg'ia tri'wn diaforetik'wn nhm'atwn
                ThreadAdd thr1 = new ThreadAdd(2,3);
                ThreadSub thr2 = new ThreadSub(2,3);
                ThreadMult thr3 = new ThreadMult(2,2,1);
                // Ekk'inhsh tri'wn par'allhlwn nhm'atwn
                thr1.start();
                thr2.start();
                thr3.start();
        }
}
```

Στο παραπάνω παράδειγμα, δημιουργούνται τρία διαφορετικά παράλληλα νήματα. Το πρώτο νήμα εκτελεί την λειτουργία την πρόσθεση δύο ακέραιων αριθμών, το δεύτερο νήμα εκτελεί την λειτουργία την αφαίρεση δύο αριθμών και το τρίτο εκτελεί την λειτουργία του πολλαπλασιασμού τριών αριθμών.

8.5 Επικοινωνία Νημάτων

Στα προηγουμένα παραδείγματα που έχουν παρουσιαστεί είδαμε ότι τα παράλληλα νήματα εκτελούσαν ανεξάρτητα χωρίς να απαιτείται αλληλεπίδραση με τα υπόλοιπα νήματα. Υπάρχουν όμως περιπτώσεις για τις οποίες είναι απαραίτητη η αλληλεπίδραση ή επικοινωνία μεταξύ των νημάτων σε ένα σύστημα κοινής μνήμης για δύο λόγους:

1. Για το διαμοιρασμό κοινών δεδομένων.

2. Για το συγχρονισμό υπολογισμών.

Για κάθε έναν από τους παραπάνω λόγους θα αναφερθούμε εκτενώς παρακάτω.

8.5.1 Διαμοιρασμός Δεδομένων

Όταν τα ταυτόχρονα νήματα έχουν πρόσβαση σε διαμοιραζόμενα δεδομένα, τότε είναι ιδιαίτερα σημαντικό οι λειτουργίες που εκτελούνται πάνω σε αυτά τα δεδομένα να μην συγκρούονται μεταξύ τους. Έτσι, η ανάγνωση μιας μεταβλητής από διαφορετικά νήματα δεν προκαλεί

σύγκρουση αλλά προκαλεί όταν γίνεται εγγραφή νέων τιμών στην μεταβλητή. Για καλύτερη κατανόηση θεωρούμε δύο νήματα που το καθένα προσθέτει κατά μια μονάδα στην διαμοιραζόμενη μεταβλητή x. Η δήλωση της Java για την πρόσθεση κατά μια μονάδα στην διαμοιραζόμενη μεταβλητή x είναι η εξής:

$x = x + 1$ ή $x++$

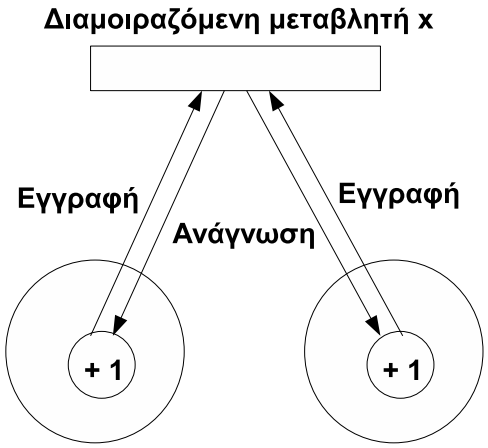
Η δήλωση αυτή στην αρχή για ένα νήμα προκαλεί την ανάγνωση της x, στην συνέχεια γίνεται ο υπολογισμός $x+1$ και τέλος το αποτέλεσμα εγγράφεται στην μεταβλητή x. Όμως, με δύο νήματα να προσπαθούν να εκτελέσουν την παραπάνω δήλωση ταυτόχρονα την ίδια στιγμή έχουμε το παρακάτω σενάριο:

Χρονική στιγμή	Νήμα 1	Νήμα 2
1	ανάγνωση x	ανάγνωση x
2	υπολογισμός $x+1$	υπολογισμός $x+1$
3	εγγραφή στη x	εγγραφή στη x

και παρουσιάζεται στο Σχήμα 8.4. Αν η αρχική τιμή της x είναι 20, τότε το τελικό επιθυμητό αποτέλεσμα μετά την εκτέλεση των δύο νημάτων θα πρέπει να είναι 22. Έστω ότι τα δύο νήματα διαβάζουν ταυτόχρονα την τιμή της μεταβλητής x που είναι 20 και τα δύο νήματα εκτελέσουν ταυτόχρονα την πράξη της πρόσθεσης τότε αυτά θα επιστρέψουν ταυτόχρονα το αποτέλεσμα 21. Έχουμε δηλαδή τελικά το εσφαλμένο αποτέλεσμα 21 αντί του σωστού 22. Έτσι προκύπτει πρόβλημα σύγκρουσης ή **συνθήκες ανταγωνισμού** (race conditions) σε μια διαμοιραζόμενη μεταβλητή. Παρακάτω παρουσιάζουμε ένα άλλο πιο σύνθετο παράδειγμα από την καθημερινή ζωή όπως εκείνου του τραπεζικού λογαριασμού που εμφανίζεται παρόμοιο πρόβλημα με το προηγούμενο παράδειγμα.

Έστω δύο αδέλφια οι Joe και John έχουν (ή διαμοιράζουν) ένα κοινό τραπεζικό λογαριασμό. Τα αδέλφια μπορούν ανεξάρτητα να πάρουν το υπόλοιπο, να κάνουν κατάθεση και ανάληψη χρημάτων. Έτσι μπορούμε να μοντελοποιήσουμε το τραπεζικό λογαριασμό σαν ένα αντικείμενο με την ακόλουθη κλάση της Java:

```
class Account {
        private double balance;

        // Kataskeuast'hs
```



Σχήμα 8.4: Σύγκρουση στην προσπέλαση σε μια διαμοιραζόμενη μεταβλητή

```
        public Account(double initialDeposit) {
                balance = initialDeposit;
        }

        // M'ejodos gia thn epistrof'h upolo'ipou
        public double getBalance() {
                return balance;
        }

        // M'ejodos gia thn kat'ajesh en'os poso'u sto log/sm'o
        public void deposit(double amount) {
                balance += amount;
        }

        // M'ejodos gia thn an'alhyh en'os poso'u ap'o to log/sm'o
        public void withdraw(double amount) {
                 // Den epitr'epetai arnhtik'o up'oloipo
                if ( balance >= amount ) { balance -= amount; }
        }
}
```

Στην συνέχεια μπορούμε να μοντελοποιήσουμε το κάτοχο ενός λογαριασμού που καταθέτει ένα ποσό ως ένα νήμα:

```
class AccountHolder extends Thread
{
        private Account acc;

        // Kataskeuast'hs
```

```
        public AccountHolder(Account a)
        {
                acc = a;
        }

        // K'aje n'hma 'h (k'atoqos logariasmo'u) na kataj'etei to
        // pos'o 100 Eur'w sto log/sm'o
        public void run()
        {
                acc.deposit(100);
        }
}
```

Το παρακάτω κομμάτι κώδικα δημιουργεί ένα αντικείμενο `Account` και δύο νήματα που παριστάνουν τους κατόχους λογαριασμών.

```
import java.io.*;

public class ThreadWithoutSync
{
        public static void main(String[] args)
        {
                // Dhmiourg'ia koino'u antikeim'enou - logariasmo'u
                Account accnt = new Account(150);
                System.out.println("Balance␣of␣shared␣account␣before␣deposit␣is␣" + accnt.getBalance());
                // Dhmiourg'ia d'uo nhm'atwn 'h kat'oqwn logariasm'wn
                AccountHolder John = new AccountHolder(accnt);
                AccountHolder Joe  = new AccountHolder(accnt);
                // Ekk'inhsh duo nhm'atwn pou epiqeiro'un na kataj'esoun
                // 'ena pos'o sto log/sm'o
                John.start();
                Joe.start();
                System.out.println("Balance␣of␣shared␣account␣after␣deposit␣is␣" + accnt.getBalance());
        }
}
```

Με την εκτέλεση του παραπάνω κώδικα παρουσιάζει ένα πρόβλημα. Δηλαδή οι δραστηριότητες των John και Joe μπορούν να συγκρουστούν και να προκύπτουν εσφαλμένες καταστάσεις. Έτσι, όταν τα δύο αδέλφια καταθέσουν το ποσό των 100 Ευρώ στην ίδια στιγμή τότε το τελικό υπόλοιπο του λογαριασμού θα είναι 250 Ευρώ αντί του σωστού που είναι 350 Ευρώ. Παρόμοιο πρόβλημα προκύπτει και με την μέθοδο `withdraw` όταν τα δύο αδέλφια προσπαθούν να κάνουν ανάληψη ενός ποσού ταυτόχρονα. Γενικά υπάρχει πρόβλημα σύγκρουσης ή ανταγωνισμού όταν δύο νήματα προσπαθούν να προσπελάζουν ένα κοινό αντικείμενο ή μια μέθοδο ταυτόχρονα.

Συγκεκριμένα, αν ένα νήμα προσπαθεί να διαβάζει δεδομένα και το άλλο νήμα προσπαθεί να ενημερώσει τα ίδια δεδομένα τότε προκύπτει μια απρόβλεπτη κατάσταση. Όμως, δεν υπάρχει πρόβλημα με τις μεθόδους που προσπελάζουν δεδομένα μόνο για ανάγνωση.

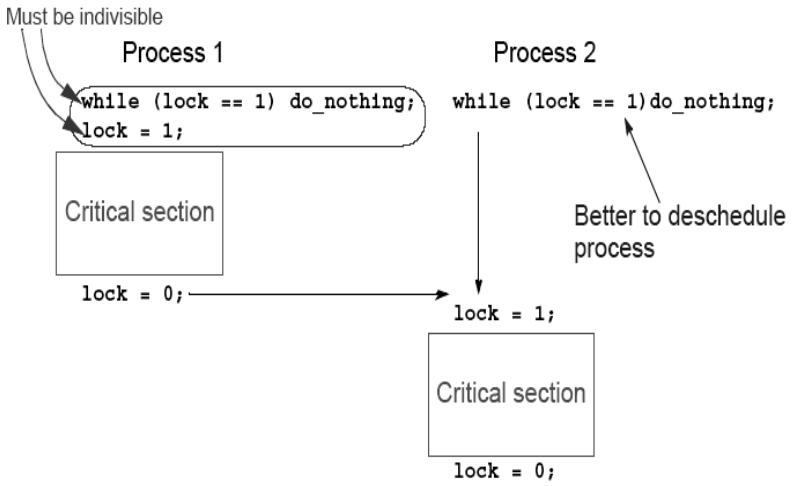
Επομένως, το πρόβλημα των διαμοιραζομένων δεδομένων μπορεί να γενικευτεί και σε διαμοιραζόμενους πόρους. Ο πόρος μπορεί να είναι μια φυσική συσκευή όπως μια συσκευή εισόδου/εξόδου, π.χ. εκτυπωτής. Ένας μηχανισμός για να εξασφαλίσουμε ότι ένα μόνο νήμα προσπελάζει ένα συγκεκριμένο πόρο κάθε φορά είναι ο ορισμός τμημάτων του κώδικα που περιλαμβάνουν το πόρο και ονομάζονται **κρίσιμα τμήματα** ή **κρίσιμες περιοχές** (critical sections). Έτσι, το πρώτο νήμα που φτάνει στο κρίσιμο τμήμα για ένα πόρο, εισάγεται και εκτελεί το κρίσιμο τμήμα. Το νήμα αυτό εμποδίζει στα άλλα νήματα να προσπελάζουν το κοινό πόρο. Μόλις το πρώτο νήμα τελειώσει την εκτέλεση του κρίσιμου τμήματος, τότε επιτρέπεται σε άλλο νήμα να εισαχθεί στο κρίσιμο τμήμα για το ίδιο πόρο. Ο μηχανισμός αυτός είναι γνωστός από τα λειτουγικά συστήματα ως **αμοιβαίος αποκλεισμός** (mutual exclusion). Παρακάτω παρουσιάζονται συνοπτικά κάποιες τεχνικές για την υλοποίηση του αμοιβαίου αποκλεισμού και πολλές από αυτές έχουν τις ρίζες τους στα λειτουργικά συστήματα.

Κλειδαριά

Η πιο απλή τεχνική για να εξασφαλίσουμε αμοιβαίο αποκλεισμό των κρίσιμων τμημάτων είναι η χρήση της **κλειδαριάς** (lock). Η κλειδαριά είναι μια μεταβλητή ενός bit που παίρνει τιμή 1 για να δείξει ότι ένα νήμα έχει εισαχθεί στο κρίσιμο τμήμα και την τιμή 0 για να δείξει ότι κανένα νήμα δεν βρίσκεται μέσα στο κρίσιμο τμήμα. Η λειτουργία της κλειδαριάς είναι ανάλογη με το κλειδί μιας πόρτας. Όταν ένα νήμα εισέρχεται στην ''πόρτα'' ενός κρίσιμου τμήματος και την βρίσκει ανοικτή τότε εισάγεται στο κρίσιμο τμήμα και κλειδώνει την πόρτα ώστε να εμποδίζει στα άλλα νήματα να εισαχθούν. Μόλις το νήμα ολοκληρώσει την εκτέλεση του κρίσιμου τμήματος, ξεκλειδώνει την πόρτα και αναχωρεί.

Υποθέτουμε ότι ένα νήμα φτάνει σε μια κλειδαριά που είναι ενεργοποιημένη και με αυτό το τρόπο το νήμα αποκλείεται από το κρίσιμο τμήμα και έτσι το νήμα μπαίνει σε κατάσταση αναμονής μέχρι ότου εισαχθεί στο τμήμα αυτό. Το νήμα πρέπει να ελέγχει συνεχώς την κατάσταση της κλειδαριάς μέσα σε μια επανάληψη καθυστέρησης όπως φαίνεται στο παρακάτω κώδικα:

```
while (lock == 1) do_nothing;  // no operation in while loop
lock = 1;                      // enter critical section
```



Σχήμα 8.5: Η ροή των κρίσιμων τμημάτων μέσω του μηχανισμού ενεργού αναμονής

```
   critical section
lock = 0;                       // leave critical section
```

Τέτοια κλειδαριά ονομάζεται **κλειδαριά spin** (spin lock) και ο μηχανισμός του ονομάζεται **ενεργός αναμονή** (busy waiting). Στο Σχήμα 8.5 παρουσιάζεται η σειριοποίηση των κρίσιμων τμημάτων από την χρήση της κλειδαριάς. Η ενεργός αναμονή αναγκάζει το νήμα να καταναλώνει άσκοπα το χρόνο του επεξεργαστή αφού εκτελεί ατέρμονη επανάληψη που περιλαμβάνει τον έλεγχο της μεταβλητής `lock` ενώ περιμένει να ανοίξει η κλειδαριά. Σε μερικές περιπτώσεις θα ήταν χρήσιμο να αντικαταστήσει το νήμα από το επεξεργαστή και να χρονοδρομολογήσει κάποιο άλλο νήμα. Αν περισσότερα από ένα νήματα περιμένουν να ανοίξει η κλειδαριά και η κλειδαριά ανοίξει τότε χρειάζεται ένας μηχανισμός (ή αλγόριθμος χρονοδρομολόγησης) για την επιλογή νήματος μεγαλύτερης προτεραιότητας ώστε να εισαχθεί πρώτο στο κρίσιμο τμήμα.

Είναι επίσης σημαντικό να εξασφαλίσουμε ότι δύο ή περισσότερα νήματα δεν μπορούν να εισαχθούν στο κρίσιμο τμήμα ταυτόχρονα. Παρόμοια, αν ένα νήμα βρίσκει την κλειδαριά ανοικτή αλλά δεν είναι ακόμα κλειστή και ένα άλλο νήμα βρίσκει επίσης την κλειδαριά ανοικτή τότε είναι απαραίτητο να εξασφαλίσουμε ότι και τα δύο νήματα δεν θα εισαχθούν στο κρίσιμο τμήμα στην ίδια στιγμή. Εδώ, οι λειτουργίες όπως η εξέταση αν η κλειδαριά είναι ανοικτή και το κλείδωμα πρέπει να γίνουν σαν μια **ενιαία ή αδιαίρετη λειτουργία** (atomic operation) ώστε κανένα άλλο νήμα να παρέμβει στην κλειδαριά. Ο μηχανισμός αυτός αποκλεισμού μπορεί να υλοποιηθεί

γενικά από το υλικό του υπολογιστή με ειδικές εντολές μηχανής όπως η εντολή που ονομάζεται test-and-set.

Σηματοφορείς

Για να εξαλειφθεί η σπατάλη χρόνου που συνεπάγεται η κλειδαριά ή ενεργός αναμονή ο Dijkstra πρότεινε ένα **σηματοφορέα** (semaphore) που είναι μια ακέραια θετική μεταβλητή και χρησιμοποιείται από δύο βασικές λειτουργίες που ονομάζονται P και V. Η λειτουργία P σε ένα σηματοφορέα s, γράφεται ως P(s), περιμένει μέχρι το s να είναι μεγαλύτερο από 0 και τότε μειώνει το s κατά μια μονάδα και επιτρέπει το νήμα να συνεχίζει. Η λειτουργία V αυξάνει το σηματοφορέα s κατά μια μονάδα και απελευθερώνει ένα από τα νήματα αναμονής (αν υπάρχουν). Οι λειτουργίες P και V εκτελούνται αδιαίρετα. Έαν δύο νήματα προσπαθούν να εκτελέσουν ταυτόχρονα τις λειτουργίες P και V, τότε αυτές θα εκτελεσθούν από ένα αλγόριθμο χρονοδρομολόγησης ο οποίος θα είναι δίκαιος (δεν το αναφέρουμε εδώ διότι είναι έξω από τους σκοπούς του βιβλίου).

Αμοιβαίος αποκλεισμός των κρίσιμων τμημάτων μπορεί να εξασφαλιστεί με ένα σηματοφορέα που έχει τιμή 0 ή 1 (που λέγεται **δυαδικός σηματοφορέας** (binary semaphore)) ο οποίος δουλεύει σαν μια μεταβλητή κλειδαριάς αλλά οι λειτουργίες P και V περιλαμβάνουν έναν αλγόριθμο χρονοδρομολόγησης νημάτων. Κάθε αμοιβαίο αποκλεισμό κρίσιμου τμήματος προηγείται από μια λειτουργία P(s) και τερματίζεται με μια λειτουργία V(s) πάνω σε ίδιο σηματοφορέα όπως φαίνεται παρακάτω:

Ο σηματοφορέας s έχει αρχική τιμή 1 που δείχνει ότι κανένα νήμα δεν βρίσκεται μέσα στο κρίσιμο τμήμα. Το πρώτο νήμα που φτάνει στην λειτουργία P(s) θέτει το σηματοφορέα σε 0 εμποδίζοντας τα υπόλοιπα νήματα να εισέλθουν στο κρίσιμο τμήμα αλλά οποιοδήποτε νήμα φτάσει στην λειτουργία P(s) καταγράφεται έτσι ώστε να επιλεγεί όταν απελευθερωθεί το κρίσιμο τμήμα. Όταν ένα νήμα φτάσει στην λειτουργία V(s) τότε θέτει το σηματοφορέα σε 1 και επιτρέπεται σε ένα από τα νήματα αναμονής να εισέλθει στο κρίσιμο τμήμα. Πρέπει να σημειώσουμε ότι η λύση του σηματοφορέα επεκτείνεται εύκολα για περισσότερα από ένα νήματα κάτι που δεν ήταν εύκολο να υλοποιηθεί με την χρήση της κλειδαριάς.

Μπορεί να υπάρξει ένας γενικός σηματοφορέας ή σηματοφορέας μετρητής που μπορεί να πάρει θετικές τιμές πέρα από τις τιμές 0 και 1. Τέτοιοι σηματοφορείς είναι χρήσιμοι γι-

Thread 1	Thread 2	Thread 3
Noncritical section	Noncritical section	Noncritical section
.	.	.
.	.	.
.	.	.
P(s)	P(s)	P(s)
Critical section	Critical section	Critical section
V(s)	V(s)	V(s)
.	.	.
.	.	.
.	.	.
Noncritical section	Noncritical section	Noncritical section

α παράδειγμα όταν θέλουμε να καταγράψουμε το πλήθος των πόρων που είναι διαθέσιμοι ή απασχολούμενοι και οι σηματοφορείς αυτοί μπορούν να χρησιμοποιηθούν για την επίλυση κλασικών προβλημάτων όπως εκείνου του παραγωγού - καταναλωτή.

Επόπτης

Είναι ευρέως γνωστό ότι οι σηματοφορείς είναι εύκολο να γίνουν ανθρώπινα σφάλματα ή οδηγούν σε δυσνόητο κώδικα παρόλο την καλή υλοποίηση του αμοιβαίου αποκλεισμού των κρίσιμων τμημάτων. Για κάθε λειτουργία P σε ένα συγκεκριμένο σηματοφορέα πρέπει να συνοδεύεται από την αντίστοιχη λειτουργία V στο ίδιο σηματοφορέα η οποία θα μπορούσε να εκτελεστεί από διαφορετικό νήμα. Έτσι, η παράλειψη μιας λειτουργίας P ή V ή ακόμα ορθογραφικό λάθος στο όνομα του σηματοφορέα προκαλεί σφάλμα. Για αυτό το λόγο ο Hoare πρότεινε μια τεχνική υψηλού επιπέδου την χρήση ενός **επόπτη** (monitor) ο οποίος είναι ένα σύνολο διαδικασιών που παρέχουν σε μια μόνο μέθοδο να έχει πρόσβαση σε ένα διαμοιραζόμενο πόρο. Ουσιαστικά τα δεδομένα και οι λειτουργίες ενθυλακώνονται σε μια δομή. Η ανάγνωση και εγγραφή μπορεί να γίνει με την χρήση της διαδικασίας επόπτης και μόνο ένα νήμα μπορεί να χρησιμοποιεί την διαδικασία επόπτη. Αν ένα νήμα ζητήσει την διαδικασία επόπτη ενώ χρησιμοποιείται από άλλο νήμα τότε το νήμα-αιτούντα μπαίνει σε κατάσταση αναμονής και τοποθετείται σε μια ουρά. Όταν

το ενεργό (ή τρέχων) νήμα τελειώσει την χρήση του επόπτη τότε το πρώτο νήμα στην ουρά (αν υπάρχει) μπορεί να χρησιμοποιήσει το επόπτη.

Η έννοια του επόπτη υπάρχει στην Java και συγκεκριμένα η λέξη κλειδί `synchronized` κάνει ένα τμήμα κώδικα μιας μεθόδου ασφαλές εμποδίζοντας περισσότερα από ένα νήματα να προσπελάζουν την μέθοδο. Έτσι, η λέξη κλειδί `synchronized` υποστηρίζει αμοιβαίο αποκλεισμό νημάτων. Επίσης, στην Java υπάρχουν δύο μηχανισμοί για αμοιβαίο αποκλεισμό: αμοιβαίο αποκλεισμό σε επίπεδο μεθόδων και αμοιβαίο αποκλεισμό σε επίπεδο εντολών.

Συγχρονισμός σε Επίπεδο Μεθόδων

Ο συγχρονισμός μεθόδων περιορίζει την προσπέλαση μιας μεθόδου σε ένα μόνο νήμα κάθε φορά. Για να δηλώσουμε μια ολόκληρη μέθοδο ως `synchronized` τοποθετούμε την λέξη κλειδί `synchronized` μπροστά στον ορισμό της μεθόδου όπως φαίνεται παρακάτω:

```
public synchronized void mymethod()
{
}
```

Όταν μια μέθοδος είναι δηλωμένη ως `synchronized`, η Java δημιουργεί ένα **αντικείμενο επόπτη** (monitor object) για αυτή τη μέθοδο. Την πρώτη φορά που το νήμα καλέσει τη μέθοδο, τότε το αντικείμενο επόπτης δίνει στο νήμα να εκτελέσει τη μέθοδο. Έτσι, το αντικείμενο επόπτης μοιάζει με την απόκτηση κλειδαριάς της μεθόδου. Όση ώρα το πρώτο νήμα κρατά το αντικείμενο επόπτη ή την κλειδαριά και τα υπόλοιπα νήματα που επιθυμούν να εκτελέσουν την μέθοδο `synchronized` μπαίνουν σε μια λίστα αναμονής. Όταν τελειώσει η εκτέλεση της `synchronized` μεθόδου, τότε απελευθερώνεται το αντικείμενο επόπτης και επιτρέπει στο πρώτο νήμα της λίστας αναμονής που είναι έτοιμο και έχει καλέσει μια μέθοδο `synchronized` να συνεχίσει την εκτέλεση του. Αν ένα αντικείμενο διαθέτει περισσότερες από μια μεθόδους `synchronized`, μόνο μια από αυτές μπορεί να είναι ενεργή κάθε φορά.

Το παράδειγμα του τραπεζικού λογαριασμού που είχαμε παρουσιάζει προηγουμένως αντιμετωπίζαμε το πρόβλημα της σύγκρουσης στο κοινό λογαριασμό. Μια λύση στο πρόβλημα αυτό είναι η υλοποίηση αμοιβαίου αποκλεισμού στα κρίσιμα τμήματα που πραγματοποιούν κατάθεση και ανάληψη χρημάτων από ένα κοινό λογαριασμό. Για να επιτύχουμε αυτό δηλώνουμε τις μεθόδους `getBalance`, `deposit` και `withdraw` ως `synchronized` ώστε να εκτελούνται από ένα νήμα κάθε

φορά. Με το τρόπο αυτό εξασφαλίσουμε το σωστό υπόλοιπο του λογαριασμού. Παρακάτω παρουσιάζεται ο σωστός κώδικας:

```
public class Account {
         private double balance;

        // Kataskeuast'hs
        public Account(double initialDeposit) {
                balance = initialDeposit;
        }

        // Kr'isimh m'ejodos gia thn epistrof'h upolo'ipou
        public synchronized double getBalance() {
                return balance;
        }

        // Kr'isimh m'ejodos gia thn kat'ajesh en'os poso'u sto log/sm'o
        public synchronized void deposit(double amount) {
                balance += amount;
        }

        // Kr'isimh m'ejodos gia thn an'alhyh en'os poso'u ap'o to log/sm'o
        public synchronized void withdraw(double amount) {
                 // Den epitr'epetai arnhtik'o up'oloipo
                if ( balance >= amount ) { balance -= amount; }
        }
}
```

Συγχρονισμός σε Επίπεδο Εντολών

Ο αμοιβαίος αποκλεισμός σε επίπεδο εντολών γίνεται χρησιμοποιώντας την λέξη - κλειδί `synchronized` και την τοποθετούμε γύρω από ένα τμήμα κώδικα αντί από μια ολόκληρη μέθοδο όπως φαίνεται παρακάτω:

```
synchronized (Object o)
{
   statements
}
```

Όταν ένα τμήμα κώδικα είναι δηλωμένη ως `synchronized`, η Java δημιουργεί ένα αντικείμενο επόπτη για αυτό το τμήμα κώδικα. Οποιοδήποτε νήμα επιχειρεί να εισέλθει σε αυτό το τμήμα κώδικα αποκτά το αντικείμενο επόπτη ή την κλειδαριά του αντικειμένου και εκτελεί τις εντολές

που περιέχει το τμήμα αυτό. Όταν το νήμα φτάσει στο τέλος του τμήματος τότε το αντικείμενο επόπτη απελευθερώνεται και είναι διαθέσιμο στα υπόλοιπα νήματα.

Επίσης, πρέπει να σημειώσουμε ότι ο αμοιβαίος αποκλεισμός σε επίπεδο εντολών χρησιμοποιείται για την περίπτωση που θέλουμε να ορίσουμε μικρότερα κρίσιμα τμήματα του προγράμματος αντί ολόκληροι μέθοδοι πράγμα που είναι επιθυμητό σε πολλά παράλληλα προγράμματα.

Παρακάτω παρουσιάζουμε ένα παράδειγμα που υπολογίζει το άθροισμα των 1000 στοιχείων του πίνακα. Συγκεκριμένα, δημιουργούνται δέκα νήματα που το κάθε νήμα υπολογίζει το τοπικό άθροισμα στο δικό τους τμήμα των 100 στοιχείων. Στη συνέχεια τα τοπικά αθροίσματα προστίθεται σε έναν γενικό αθροιστή με αμοιβαίο αποκλεισμό σε επίπεδο εντολών.

```
class ThreadSumSync
{
        public static int sum = 0;
        public static int[] array;

        public static void main(String[] args)
        {
                // Dhmiourg'ia kai arqikopo'ihsh p'inaka
                array = new int[1000];
                for(int i = 0; i < 1000; i++) array[i] = i;

                // Dhmiourg'ia p'inaka nhm'atwn
                ThreadSum threads[] = new ThreadSum[10];

                // Dhmiourg'ia kai ekk'inhsh nhm'atwn
                for(int i = 0; i < 10; i++)
                {
                        threads[i] = new ThreadSum(array,i);
                        threads[i].start();
                }

                // Anamon'h na telei'wsoun 'ola ta n'hmata
        for (int i = 0; i < threads.length; i++) {
                        try {
                                threads[i].join();
                        }
                        catch (InterruptedException e) {
                                System.err.println("this␣should␣not␣happen");
                        }
                }
                System.out.println("The␣sum␣of␣of␣the␣numbers␣is␣" + sum);
        }
```

```
        private static class ThreadSum extends Thread
        {
                int partial_sum=0;
                int [] array;
                int number;

                // Kataskeuast'hs
                public ThreadSum(int [] array, int number)
                {
                        this.array = array;
                        this.number = number;
                }

                // K'aje n'hma upolog'izei to topik'o 'ajroisma mias om'adas 100 stoiqe'iwn p'inaka
                public void run()
                {
                        int size = number * 100;
                        for(int i = size; i <= size + 99; i++)
                                partial_sum  = partial_sum + array[i];
                        System.out.println("Partial␣sum␣from␣thread␣" + number + "␣is␣" + partial_sum);
                        // Suggronism'os sthn koin'h metablht'h sum ('ajroisma)
                        synchronized(this)
                        {
                                sum = sum + partial_sum;
                        }
                }
        }
}
```

Μεταβλητές υπό Συνθήκη

Πολλές φορές, ένα κρίσιμο τμήμα μπορεί να εκτελείται με βάση κάποια συνθήκη. Για παράδειγμα στο κλασικό πρόβλημα παραγωγού - καταναλωτή, ο παραγωγός προσθέτει στοιχεία σε ένα **ενδιάμεσο ενταμιευτή** (buffer) και ο καταναλωτής τα αφαιρεί από εκεί, προκειμένου να τα επεξεργαστεί κατάλληλα. Ο ενδιάμεσος ενταμιευτής είναι το κρίσιμο τμήμα και η τροποποίηση του πρέπει να γίνεται σε συνθήκες αμοιβαίου αποκλεισμού. Έτσι, αν ο ενδιάμεσος ενταμιευτής είναι άδειος τότε ο καταναλωτής πρέπει να ανασταλεί να διαβάσει στοιχεία, ενώ όταν ο ενταμιευτής είναι γεμάτος τότε ο παραγωγός πρέπει να ανασταλεί να γράψει άλλα στοιχεία. Για να αντιμετωπιστεί η παραπάνω κατάσταση, η Java επιτρέπει στις `synchronized` μέθοδους ενός αντικειμένου να επικοινωνούν μεταξύ τους και να συγχρονίζονται με βάση τις συνθήκες που απαιτεί η εφαρμογή. Έτσι, υπάρχουν οι παρακάτω τρεις μέθοδοι στην Java για την υλοποίηση

του αμοιβαίου αποκλεισμού με βάση τη συνθήκη:

- `public final void wait()`. Προκαλεί την αναστολή της εκτέλεσης του νήματος μέχρις ότου ικανοποιηθεί η συνθήκη ή ένα άλλο νήμα καλέσει τη μέθοδο `notify()` ή `notifyAll()`.

- `public final void notify()`. Το πρώτο από τα νήματα που έχει ανασταλεί μέσω της `wait()` να αλλάζει την κατάσταση του νήματος αυτού σε έτοιμη δεδομένου ότι έχει αλλάξει η συνθήκη.

- `public final void notifyAll()`. Όλα τα νήματα που έχουν ανασταλεί μέσω της `wait()` να αλλάζουν την κατάσταση των νημάτων αυτών σε έτοιμη δεδομένου ότι έχει αλλάξει η συνθήκη.

Οι μέθοδοι `wait()`, `notify()` και `notifyAll()` πρέπει πάντα είτε να καλούνται από μεθόδους `synchronized` (ή εντολές) είτε να καλούνται έμμεσα από κάποια μέθοδο που χρησιμοποιείται μέσα σε τέτοιες μεθόδους ή εντολές. Δημιουργείται μια εξαίρεση τύπου `IllegalMonitorStateException` όταν ένα νήμα καλέσει μια από τις μεθόδους `wait()`, `notify()` ή `notifyAll()` χωρίς να έχει αποκτήσει την κλειδαριά του αντικειμένου.

Όταν καλείται η μέθοδος `wait()` μέσα από μία μέθοδο `synchronized`, το νήμα αναστέλεται και απελευθερώνει την κλειδαριά του αντικειμένου. Έτσι ένα άλλο νήμα μπορεί να αποκτήσει την κλειδαριά και να εκτελέσει την ίδια μέθοδο του αντικειμένου. Όταν το πρόγραμμα αναγνωρίζει την ικανοποίηση μιας συνθήκης τότε κάποιο από αυτά τα νήματα θα καλέσει τη μέθοδο `notify()` ή τη `notifyAll()`, με αποτέλεσμα μόλις το τρέχον νήμα απελευθερώσει την κλειδαριά του αντικειμένου, η κλειδαριά να παραχωρηθεί σε ένα από τα νήματα που έχουν ανασταλεί μέσω της `wait()`.

Για να υλοποιήσουμε αμοιβαίο αποκλεισμό με βάση κάποια συνθήκη, θα πρέπει η μέθοδος `wait()` να καλείται πάντα μέσα σε μια εντολή επανάληψης `while` η οποία να ελέγχει την συνθήκη. Έτσι ώστε κάθε φορά που το νήμα αφυπνίζεται θα ελέγχει ξανά τη συνθήκη και αν η συνθήκη δεν ικανοποιείται, το νήμα θα αναστέλεται ξανά.

Επίσης, πρέπει να σημειώσουμε ότι σε κάθε κλήση της μεθόδου `wait()` πρέπει να αντιστοιχεί μια κλήση της μεθόδου `notify()` ή `notifyAll()`, διαφορετικά μπορεί να δημιουργηθούν **αδιέξοδα** (deadlock). Γενικά, αδιέξοδα μπορεί να συμβούν όταν δύο νήματα έχουν αποκτήσει από μια από τις κλειδαριές δύο αντικειμένων και περιμένουν να αποκτήσουν την άλλη.

Για την χρήση του αμοιβαίου αποκλεισμού υπο συνθήκη, θα μελετήσουμε ένα παράδειγμα που είναι το κλασσικό πρόβλημα του παραγωγού - καταναλωτή ή και πρόβλημα του περιορισμένου ενδιάμεσου ενταμιευτή. Δύο νήματα μοιράζονται ένα κοινό ενδιάμεσο ενταμιευτή και υποθέτουμε ότι η χωρητικότητα του είναι ίση με $size$. Θεωρούμε ότι ο κοινός ενδιάμεσος ενταμιευτής είναι ένας πίνακας ακεραίων αριθμών οποιουδήποτε μεγέθους και χρησιμοποιείται για την επικοινωνία των νημάτων παραγωγού - καταναλωτή. Το νήμα παραγωγός παράγει ή γράφει τιμές στο ενταμιευτή και από τη άλλη το νήμα καταναλωτή καταναλώνει ή διαβάζει τιμές από το ενταμιευτή. Με αυτό το τρόπο ο ενταμιευτής επιτρέπει στα δύο νήματα να επικοινωνούν και να ανταλλάσουν δεδομένα. Εννοιολογικά, ο ενταμιευτής συμπεριφέρεται σαν μια ουρά που περιέχει τιμές. Οι τιμές γράφονται στο ενταμιευτή και αποθηκεύονται στην ουρά μέχρι να διαβαστεί από άλλο νήμα. Καθώς οι τιμές γράφονται και διαβάζονται, υπάρχει ένας διαμοιραζόμενης δείκτης που δείχνει κάθε φορά το τελευταίο στοιχείο του ενταμιευτή που πρόκειται να διαβαστεί ή να γραφτεί. Όμως, ο ενταμιευτής πρέπει να λάβει υπόψη δύο περιορισμούς: Πρώτον, στην περίπτωση που το νήμα παραγωγός επιχειρεί να γράψει μια τιμή σε ένα γεμάτο ενταμιευτή, τότε η εκτέλεση του νήματος αναβάλεται μέχρι ότου κάποιο άλλο νήμα καταναλωτής να διαβάζει μια τιμή από το ενταμιευτή, οπότε και το νήμα παραγωγός μπορεί να αφυπνιστεί. Δεύτερον, στην περίπτωση που το νήμα καταναλωτής επιχειρήσει να διαβάζει μια τιμή από ένα άδειο ενταμιευτή τότε η εκτέλεση του νήματος καθυστερεί μέχρι να γραφτεί μια τιμή στο ενταμιευτή από κάποιο άλλο νήμα παραγωγός. Οι παραπάνω περιορισμοί του ενταμιευτή υλοποιούνται μέσω του αμοιβαίου αποκλεισμού υπο συνθήκη.

Παρακάτω παρουσιάζεται μια κλάση `Buffer` που προσομοιώνει το κοινό ενδιάμεσο ενταμιευτή χωρητικότητας 5 στοιχείων και τις `synchronized` μεθόδους `put()` και `get()` που αντιστοιχούν στην προσθήκη και αφαίρεση ενός δεδομένου στο/από ενταμιευτή αντίστοιχα. Οι μέθοδοι `put()` και `get()` που παρουσιάζονται λαμβάνουν υπόψη στην περίπτωση που ο ενταμιευτής είναι γεμάτος ή άδειος και υλοποιούνται μέσω των μεθόδων `wait()` και `notify()`. Συγκεκριμένα, η μέθοδος `put()` πρώτα ελέγχει αν είναι γεμάτος ο ενταμιευτής. Σε αυτή την περίπτωση προκαλεί αναστολή της εκτέλεσης του νήματος που προσπαθεί να προσθέσει στοιχείο. Διαφορετικά, τοποθετεί το στοιχείο `data` στο ενταμιευτή και αυξάνει το δείκτη `counter` κατά ένα. Μετά την τοποθέτηση του στοιχείου ελέγχει αν έχει ή όχι γεμίσει το ενταμιευτή και στην συνέχεια αφυπνίσει κάποιο άλλο νήμα που τυχόν περίμενε να προστεθεί στοιχείο για να διαβάζει. Παρόμοια λειτουργεί και η μέθοδος `get()`.

```
public class Buffer
{
        private int[] contents = {0,0,0,0,0};
        private boolean buffernoEmpty = false;
        private boolean buffernoFull = true;
        private final int size = 5;
        private int counter = 0;

        // Kr'isimh m'ejodos pr'osjesh en'os dedom'enou sto entamieut'h
        public synchronized void put(int data)
        {
                while (!buffernoFull) // o entamieut'hs e'inai gem'atos
                {
                        try
                        {
                                // to n'hma pou k'alese th m'ejodo aut'hn pr'epei
                                // na perim'enei
                                wait();
                        }
                        catch (InterruptedException e)
                        {
                                System.err.println("Exception");
                        }
                }
                System.err.println("The␣producer␣adds␣the␣value␣" + data + "␣in␣the␣buffer");
                contents[counter] = data;
                counter++;
                buffernoEmpty = true;
                // el'egqei an 'eqei gem'isei o entamieut'hs
                if (counter==size)
                {
                        buffernoFull = false;
                        System.err.println("The␣buffer␣is␣full");
                }
                // af'upnise ton katanalwt'h pou pijan'on na perim'enei
                notifyAll();
        }

        // Kr'isimh m'ejodos afa'iresh en'os dedom'enou ap'o to entamieut'h
        public synchronized int get()
        {
                while (!buffernoEmpty) // o entamieut'hs e'inai 'adeios
                {
                        try
                        {
                                // to n'hma pou k'alese th m'ejodo aut'hn pr'epei
```

```
                        // na perim'enei
                        wait();
                }
                catch (InterruptedException e)
                {
                        System.err.println("Exception");
                }
        }
        counter--;
        int data = contents[counter];
        System.err.println("The␣consumer␣removes␣the␣value␣" + data + "␣by␣the␣buffer");
        // el'egqei an e'inai 'adeios o entamieut'hs
        buffernoFull = true;
        if (counter==0)
        {
                buffernoEmpty = false;
                System.err.println("The␣buffer␣is␣empty");
        }
        // af'upnish tou paragwgo'u pou pijan'on na 'eqei empodiste'i
        notifyAll();
        return data;
    }
}
```

Η κλάση `Buffer` είναι χρήσιμη για το προγραμματισμό επικοινωνίας νημάτων παραγωγού - καταναλωτή. Έστω το `buff` είναι ένα αντικείμενο τύπου `Buffer`, το νήμα παραγωγός για να γράψει μια τιμή `num` στο ενταμιευτή `buff` χρησιμοποιεί την εντολή `buff.put(num)` ή ακόμα το νήμα καταναλωτής για να διαβάζει μια τιμή `num` από το ενταμιευτή χρησιμοποιεί την εντολή `num = buff.get()`. Παρακάτω φαίνεται ο κώδικας των νημάτων παραγωγού - καταναλωτή που χρησιμοποιούν το κοινό ενταμιευτή `buff` για την επικοινωνία τιμών ανάμεσα στα δύο νήματα. Η κλάση `Producer` παράγει ή γράφει 10 τιμές στο ενταμιευτή και η κλάση `Consumer` καταναλώνει ή διαβάζει 10 τιμές από το ενταμιευτή. Τέλος, η κύρια κλάση `Producer_Consumer` δημιουργεί το κοινό ενταμιευτή `Buffer` και δημιουργεί και εκτελεί τα δύο νήματα παραγωγού - καταναλωτή.

```
public class Producer extends Thread
{
        private Buffer buff;

        // Kataskeuast'hs
        public Producer(Buffer b)
        {
                buff = b;
        }
```

```
        // o paragwg'os ektele'itai 10 for'es gia na prosj'esei
        // 'ena dedom'eno sto entamieut'h
        public void run()
        {
                for(int i = 0; i <= 9; i++)
                        buff.put(i);
                System.err.println("The␣producer␣produced␣the␣values");

        }
}

public class Consumer extends Thread
{
        private Buffer buff;

        // Kataskeuast'hs
        public Consumer(Buffer b)
        {
                buff = b;
        }

        // o katanalwt'hs ektele'itai 10 for'es gia na afair'esei
        // 'ena dedom'eno ap'o ton entamieut'h
        public void run()
        {
                int value;
                for(int i = 0; i <= 9; i++)
                        value = buff.get();
                System.err.println("The␣consumer␣removed␣the␣values");

        }
}

public class Producer_Consumer
{
        public static void main(String[] args)
        {
                // Dhmiourg'ia koino'u entamieut'h
                Buffer buff = new Buffer();

                // Dhmiourg'ia n'hmatos paragwgo'u
                Producer prod = new Producer(buff);

                // Dhmiourg'ia n'hmatos katanalwt'h
                Consumer cons = new Consumer(buff);
```

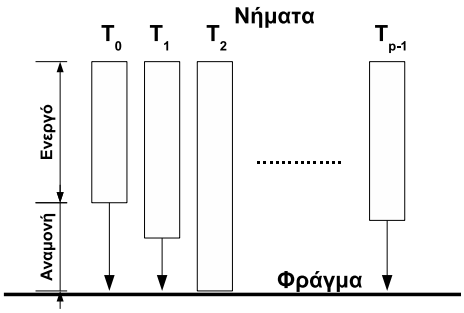
```
                // Ekk'inhsh nhm'atwn
                prod.start();
                cons.start();
        }
}
```

Συνοπτικά, η κλάση `Buffer` χρησιμοποιείται ως μια ενδιάμεση μνήμη για τις τιμές που στέλνονται από το νήμα παραγωγό στο νήμα καταναλωτή. Αν ο παραγωγός είναι γρηγορότερος, τότε ο ενταμιευτής θα αποθηκεύσει τις επιπλέον τιμές μέχρι ο καταναλωτής να είναι σε θέση να τις χρησιμοποιήσει. Αν όμως ο καταναλωτής είναι πιο γρήγορος τότε θα βρει ορισμένες φορές το κανάλι άδειο. Όποτε συμβαίνει κάτι τέτοιο, ο ενταμιευτής θα αναγκάζει τον καταναλωτή να περιμένει μέχρι να γραφτεί κάποια τιμή στο ενταμιευτή από τον παραγωγό. Το παράδειγμα επικοινωνίας παραγωγού - καταναλωτή μέσω του κοινού ενταμιευτή είναι πολύ χρήσιμη στον πολυνηματικό προγραμματισμό και θα μας επιτρέψει στην παρουσίαση σημαντικών αλγορίθμων ή μοντέλων όπως το μοντέλο διασωλήνωσης που θα δούμε αργότερα σε αυτό το κεφάλαιο.

8.5.2 Συγχρονισμός Νημάτων

Ο συγχρονισμός των νημάτων είναι αναγκαίος στα προγράμματα κοινής μνήμης και χρησιμοποιείται σε εφαρμογές όπου όλα τα νήματα συγχρονίζονται σε μερικά σημεία του προγράμματος. Με άλλα λόγια, χρησιμοποιείται σε εφαρμογές οι οποίες χωρίζονται σε επαναλήψεις ενός βρόχου. Στη διάρκεια κάθε επανάληψης πολλά νήματα δουλεύουν παράλληλα σε διαφορετικά δεδομένα, αλλά όταν το κάθε νήμα τελειώσει, περιμένει μέχρι να τελειώσουν και τα υπόλοιπα νήματα πριν προχωρήσει στην επόμενη επανάληψη. Αυτό συμβαίνει γιατί οι τιμές που παράγονται από ένα νήμα χρησιμοποιούνται από άλλα νήματα στην επόμενη επανάληψη. Από τα παραπάνω προκύπτει ότι είναι αναγκαίο να υπάρξει ένας μηχανισμός που να εμποδίζει σε ένα νήμα να προχωρήσει στην επόμενη επανάληψη μέχρι να τελειώσουν τα υπόλοιπα νήματα. Ο μηχανισμός αυτός για την αντιμετώπιση του συγχρονισμού ονομάζεται **φράγμα** (barrier).

Φράγμα είναι ένα σημείο του προγράμματος στο οποίο τα παράλληλα νήματα περιμένουν ο ένας τον άλλο. Το πρώτο νήμα που θα εκτελέσει την δήλωση του φράγματος απλά θα περιμένει μέχρι να φτάσουν όλα τα άλλα νήματα. Όταν όλα τα νήματα φτάσουν στη δήλωση φράγματος μπορούν όλα να συνεχίσουν την παράλληλη εκτέλεση. Η όλη διαδικασία του φράγματος παρουσιάζεται στο Σχήμα 8.6. Σε αυτό το Σχήμα, το νήμα 2 είναι το τελευταίο νήμα που φτάνει



Σχήμα 8.6: Τα νήματα που φτάνουν στο σημείο του φράγματος

στο φράγμα. Συνεπώς, όλα τα υπόλοιπα νήματα πρέπει να περιμένουν μέχρι το νήμα 2 να φτάσει στο φράγμα. Και τότε τα νήματα που είχαν ανασταλεί μπορούν να ξεκινήσουν την εκτέλεση τους από το σημείο του φράγματος.

Στην Java Threads δεν υπάρχει ρητή δήλωση για το φράγμα. Όμως, το φράγμα μπορεί εύκολα να υλοποιηθεί στην Java χρησιμοποιώντας τις μεταβλητές υπό συνθήκη. Πρέπει να σημειώσουμε ότι στην διεθνή βιβλιογραφία υπάρχουν πολλές τεχνικές για την υλοποίηση του φράγματος όπως το γραμμικό φράγμα (ή υλοποίηση μετρητή), το φράγμα δυαδικού δένδρου, το φράγμα πεταλούδα και τοπικός συγχρονισμός. Στην παρούσα ενότητα δεν θα αναφερθούμε τις λεπτομέρειες των παραπάνω τεχνικών παρά μόνο την πρώτη τεχνική που είναι η πιο απλούστερη.

Υλοποίηση Γραμμικού Φράγματος

Εδώ θα περιγράψουμε για το πώς υλοποιείται το γραμμικό φράγμα σε Java. Η προσέγγιση για την υλοποίηση του γραμμικού φράγματος είναι η χρήση ενός μετρητή. Συνεπώς, μια διαμοιραζόμενη μεταβλητή μετρητή χρησιμοποιείται για την καταγραφή του συνολικού αριθμού των νημάτων που φτάνουν στο σημείο του φράγματος. Καθώς κάθε νήμα που φτάνει στο φράγμα αυξάνει το μετρητή κατά ένα και ελέγχει αν ο μετρητής έχει φτάσει στον αριθμό των νημάτων ας πούμε p. Αν ο μετρητής δεν έχει φτάσει στον αριθμό p τότε το νήμα μπαίνει σε κατάσταση αναμονής μέσω της κλήσης μεθόδου `wait()`. Αν ο μετρητής έχει φτάσει στο p, τότε το τελευταίο νήμα που έφτασε στο φράγμα απελευθερώνει τα υπόλοιπα νήματα μέσω της μεθόδου `notifyAll()` και ο μετρητής παίρνει ξανά την τιμή 0 για την επόμενη φορά που θα χρησιμοποιηθεί το φράγμα. Παρακάτω, παρουσιάζεται η υλοποίηση του φράγματος σε Java:

```
public class LinearBarrier
```

```
{
        private int counter = 0;
        private final int size_threads = 10;
        private boolean waiting = true;

        // Kr'isimh m'ejodos metrht'hs
        public synchronized void mycounter()
        {
                counter++;
                // El'egqei an 'eqoun ft'asei 'ola ta n'hmata
                if (counter==size_threads)
                {
                        waiting = false;
                        counter = 0;
                }
                // Ta n'hmata mpa'inoun se kat'astash anamon'hs
                while (waiting)
                {
                        try
                        {
                                wait();
                        }
                        catch (InterruptedException e)
                        {
                                System.err.println("Exception");
                        }
                }
                // To teleuta'io n'hma afupn'isei 'ola ta up'oloipa n'hmata
                notifyAll();
        }
}
```

8.6 Υλοποίηση Υπολογιστικών Μοντέλων με Νήματα

Στην ενότητα αυτή παρουσιάζουμε την υλοποίηση των δύο κοινών μοντέλων υπολογισμών που χρησιμοποιούνται συχνά στις παράλληλες εφαρμογές σε Java Threads. Τα μοντέλα που θα εξετάσουμε είναι ο συντονιστής - εργαζόμενος και η διασωλήνωση.

8.6.1 Υλοποίηση Μοντέλου Συντονιστή - Εργαζόμενος

Όπως είδαμε στο κεφάλαιο 7, το μοντέλο συντονιστής - εργαζόμενος χρησιμοποιείται συχνά στην οργάνωση των παράλληλων προγραμμάτων. Για την υλοποίηση του μοντέλου συντονιστής - εργαζόμενος σε νήματα θα χρησιμοποιήσουμε ένα παράδειγμα όπως του υπολογισμού της τετραγωνικής ρίζας των στοιχείων ενός πίνακα. Θεωρούμε ότι τα στοιχεία του πίνακα ως μικρές εργασίες. Είναι γνωστό ότι για να εκτελεστούν οι εργασίες από τα νήματα - εργαζόμενους πρέπει να διανεμηθούν και υπάρχουν δύο τρόποι: ο στατικός και ο δυναμικός. Σε αυτή την ενότητα θα ασχοληθούμε με την υλοποίηση του συντονιστή - εργαζόμενου με δυναμική διανομή εργασιών εφόσον είναι πιο γενική από την στατική διανομή (δηλαδή, όταν το πλήθος των εργασιών είναι ίδιος με τον αριθμό των νημάτων προκύπτει στατική διανομή). Συνεπώς, η λειτουργία του είναι ως εξής: Το νήμα ως συντονιστής διανέμει τις εργασίες στα αδρανή νήματα ως εργαζόμενους και κάθε νήμα - εργαζόμενος λαμβάνει τις εργασίες, εκτελεί την επεξεργασία των εργασιών, επιστρέφει τα αποτελέσματα της επεξεργασίας πίσω στο νήμα - συντονιστής και τέλος λαμβάνει νέα εργασία αν υπάρχουν. Παρακάτω παρουσιάζεται ένα πρόγραμμα συντονιστή - εργαζόμενου με δυναμική διανομή εργασιών στους εργαζόμενους. Η εργασία που εκτελεί κάθε νήμα - εργαζόμενος είναι ο υπολογισμός της τετραγωνικής ρίζας ενός στοιχείου του πίνακα.

```
public class MasterWorker {

private static int totalTasks; // arijm'os ergasi'wn
private static int nThreads; // arijm'os nhm'atwn
private static int tasksAssigned = -1; // diamoiraz'omenh metablht'h metrht'h ergasi'wn

public static void main(String[] args)
{
        totalTasks = 20;
        nThreads = 10;

        // Dhmiourg'ia kai arqikopo'ihsh p'inaka
        double[] a = new double[totalTasks];
        for(int i=0; i<totalTasks; i++)
        {
                a[i] = i + 3;
                System.out.println(a[i]);
        }

        // Dhmiourg'ia nhm'atwn ergazom'enwn
        Thread[] threads = new Thread[nThreads];
        for (int i = 0; i < threads.length; ++i)
```

```
        {
                threads[i] = new Thread(new Worker(a,i));
        }

        // Ekk'inhsh nhm'atwn ergazom'enwn
        for (int i = 0; i < threads.length; ++i)
        {
                threads[i].start();
        }

        // Fr'agma
        for (int i = 0; i < threads.length; ++i)
        {
                try {
                        threads[i].join();
                }
                catch (InterruptedException e) {
                        System.err.println("this␣should␣not␣happen");
                }
        }

        // Emf'anish stoiqe'iwn p'inaka
        for(int i = 0; i < totalTasks; i++)
                System.out.println(a[i]);
}

//  Kr'isimh m'ejodos gia thn dianom'h ergasi'wn
private static synchronized int getTask()
{
        // Dian'emei mia ergas'ia ('h stoiqe'io p'inaka) an up'arqoun
        if (++tasksAssigned < totalTasks)
                return tasksAssigned;
        else
                return -1;
}

// K'wdikas pou ektele'i to k'aje n'hma - ergaz'omenos
private static class Worker implements Runnable
{

        private int myID;
        private double[] table;

        // Kataskeuast'hs
        public Worker(double[] array, int myID)
        {
```

```
                this.myID = myID;
                table = array;
        }

        public void run()
        {
                int element;
                // Lamb'anei ergas'ia ('h stoiqe'io p'inaka) an up'arqei
                while ((element = getTask()) >= 0)
                {
                        System.out.println("worker␣" + myID + "␣received␣" + element);
                        // Upolog'izei thn tetragwnik'h r'iza tou stoiqe'iou pou 'elabe
                        table[element]= Math.sqrt(table[element]);
                }
                System.out.println("worker␣" + myID + "␣done");
        }
}
}
```

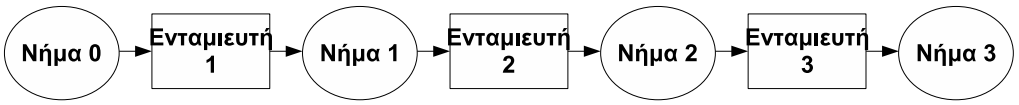
Οι εργασίες σε αυτό το παράδειγμα θεωρούνται τα στοιχεία του πίνακα. Η δυναμική διανομή των εργασιών (ή των στοιχείων του πίνακα) υλοποιείται με την διαμοιραζόμενη μέθοδο `getTask()`. Η μέθοδο αυτή χρησιμοποιεί ένα διαμοιραζόμενο μετρητή εργασιών (ή στοιχείων πίνακα) `tasksAssigned` που καταγράφει το πλήθος των εργασιών (ή στοιχείων πίνακα) που έχουν διανεμηθεί στους εργαζόμενους. Ο μετρητής αυξάνεται κάθε φορά που το νήμα ζητά εργασία (ή στοιχείο πίνακα) και ελέγχει αν έχει φτάσει στο προκαθορισμένο αριθμό εργασιών (ή το μέγεθος πίνακα) `totalTasks`. Στην περίπτωση που δεν φτάσει τότε επιστρέφει (ή διανέμει) την τρέχουσα τιμή του μετρητή όπου είναι η αντίστοιχη θέση του πίνακα που θα υπολογίζει το νήμα - εργαζόμενος την τετραγωνική ρίζα, διαφορετικά επιστρέφει μια τιμή τέλους π.χ. -1. Τέλος, η διανομή εργασιών (ή στοιχείων πίνακα) από το συντονιστή προς το εργαζόμενο και η επιστροφή αποτελέσματος από το εργαζόμενο προς το συντονιστή γίνεται μέσω του κοινού πίνακα.

8.6.2 Υλοποίηση Μοντέλου Διασωλήνωσης

Μια ειδική περίπτωση του παραδείγματος παραγωγού-καταναλωτή που έχουμε εξετάσει προηγουμένως είναι το μοντέλο της διασωλήνωσης, στην οποία μια σειρά από νήματα σχηματίζουν μια γραμμική διασωλήνωση όπου κάθε νήμα καταναλώνει δεδομένα από το αριστερό γειτονικό νήμα και παράγει δεδομένα για το δεξιό γειτονικό νήμα. Γενικά σε αλγόριθμους διασωλήνωσης, μια σειρά από τιμές δεδομένων ρέουν μέσα στη διασωλήνωση και μετασχηματίζονται από τα

νήματα σε κάθε στάδιο της διασωλήνωσης. Κάθε νήμα παραλαμβάνει μια σειρά από δεδομένα από τον αριστερό γείτονα, τα μετατρέπει κατόπιν επεξεργασίας και μετά στέλνει τα αποτελέσματα στο δεξιό γείτονα. Κάθε αρχική τιμή που εισέρχεται στη διασωλήνωση υφίσταται μια σειρά από μετασχηματισμούς και εξέρχεται ως αποτέλεσμα από τη διασωλήνωση. Ο παραλληλισμός προέρχεται από το γεγονός ότι κάθε νήμα της διασωλήνωσης εκτελεί μετασχηματισμούς την ίδια ακριβώς στιγμή αλλά με διαφορετικά δεδομένα.

Το Σχήμα 8.7 φαίνεται η γενική δομή της διασωλήνωσης νημάτων. Κάθε νήμα ενεργεί



Σχήμα 8.7: Γραμμική διασωλήνωση νημάτων

ταυτόχρονα και σαν παραγωγός αλλά και σαν καταναλωτής. Το νήμα καταναλώνει από τα αριστερά του από το ενταμιευτή και παράγει προς το δεξιό του στο ενταμιευτή. Για την υλοποίηση του μοντέλου διασωλήνωσης, χρησιμοποιείται ένας πίνακας που αποτελείται από ενταμιευτές. Η Java Threads επιτρέπει την δημιουργία πίνακα από ενταμιευτές και υλοποιείται με το παρακάτω κομμάτι κώδικα:

```
Buffer buf[] = new Buffer[nThreads];
for(int i = 0; i < buf.length; i++)
    buf[i] = new Buffer();
```

Η αναφορά οποιουδήποτε στοιχείου του πίνακα ενταμιευτών γίνεται ακριβώς όπως η αναφορά στα στοιχεία ενός οποιουδήποτε κοινού πίνακα. Το `buf[0]` είναι το πρώτο ενταμιευτή του πίνακα ενώ το `buf[1]` είναι το δεύτερο ενταμιευτή του πίνακα και ούτω καθεξής. Για παράδειγμα, όταν ένα νήμα θέλει να γράψει μια τιμή `value` στο ενταμιευτή `buf[0]` γίνεται με την εντολή `buf[0].put(value)`. Παρόμοια, όταν ένα νήμα θέλει να διαβάζει μια τιμή `value` από το ενταμιευτή `buf[0]` γίνεται με την εντολή `value = buf[0].get()`.

Η χρήση του πίνακα ενταμιευτών είναι ιδιαίτερα χρήσιμη στον προγραμματισμό της διασωλήνωσης νημάτων. Τα νήματα είναι αριθμημένα σειριακά, ξεκινώντας από τα αριστερά προς τα δεξιά. Το νήμα 1 διαβάζει από το `buf[1]` και γράφει στο `buf[2]`. Ομοίως, το νήμα 2 διαβάζει από το ενταμιευτή `buf[2]` που βρίσκεται από τα αριστερά του και γράφει στο `buf[3]` που βρίσκεται

στα δεξιά του. Γενικά ισχύει, το νήμα με ταυτότητα id διαβάζει από το ενταμιευτή `buf[id]` και γράφει στο ενταμιευτή `buf[id+1]`. Όμως τα δύο άκρα νήματα της διασωλήνωσης λειτουργούν διαφορετικά. Με άλλα λόγια, το νήμα 0 δεν διαβάζει από το ενταμιευτή `buf[0]` εφόσον είναι το πρώτο νήμα και τροφοδοτεί στοιχεία δεδομένων αλλά γράφει στο ενταμιευτή `buf[1]`. Παρόμοια, το τελευταίο νήμα δηλαδή $nthreads-1$ διαβάζει από το ενταμιευτή `buf[ntreads-1]` αλλά δεν γράφει στο ενταμιευτή `buf[nthreads]` εφόσον βγαίνουν τα αποτελέσματα από τη διασωλήνωση.

Για την κατανόηση της υλοποίησης του μοντέλου διασωλήνωσης σε νήματα θα χρησιμοποιήσουμε ένα παράδειγμα που είναι το εξής: Θεωρούμε ότι τα n στοιχεία ενός μονοδιάστατου πίνακα διαπερνούν στην διασωλήνωση από την αρχή μέχρι το τέλος. Κάθε νήμα στην διασωλήνωση θα αυξάνει την τιμή δεδομένων που λαμβάνει από αριστερά κατά $id + 1$ όπου id είναι η ταυτότητα του νήματος και το αποτέλεσμα της αύξησης θα την στέλνει στο επόμενο νήμα. Σε αυτό το παράδειγμα, το κάθε νήμα id της διασωλήνωσης θα λαμβάνει πρώτα ένα στοιχείο δεδομένων `num` από το αριστερό ενταμιευτή `buf[id]` με την μέθοδο `get()`, έπειτα υπολογίζει την αύξηση `num+(id+1)` και στέλνει το αποτέλεσμα στο δεξιό ενταμιευτή `buf[id+1]` με την μέθοδο `put()`. Παρακάτω παρουσιάζεται το πρόγραμμα διασωλήνωσης με την χρήση της κλάσης `Buffer` που είδαμε στο παράδειγμα παραγωγού-καταναλωτή.

```
class ThreadPipe
{
        private static int nThreads;

        public static void main(String[] args) throws Exception
        {
                nThreads = 5;
                // Dhmiourg'ia p'inaka entamieut'wn
                Buffer buf[] = new Buffer[nThreads];
                for(int i = 0; i < buf.length; i++)
                        buf[i] = new Buffer();
                // Dhmiourg'ia kai arqikopo'ihsh p'inaka stoiqe'iwn
                int a[] = new int[10];
                for(int i=0; i < a.length; i++)
                        a[i] = i;
                // Dhmiourg'ia p'inaka nhm'atwn
                Thread[] threads = new Thread[nThreads];
                for(int i = 0; i < threads.length; i++)
                        threads[i] = new CodeThread(a,buf,i);
                // Ekk'inhsh Nhm'atwn
                for(int i = 0; i< threads.length; i++)
                        threads[i].start();
```

```
        }

        private static class CodeThread extends Thread
        {
                private int myID;
                private int[] table;
                private Buffer[] buff;

                // Kataskeuast'hs
                public CodeThread(int[] table, Buffer[] buff, int myID)
                {
                        this.myID = myID;
                        this.table = table;
                        this.buff = buff;
                }

                // Ekt'elesh k'aje n'hmatos
                public void run()
                {
                        int num, result;
                        if (myID == 0)
                            // K'wdikas gia to pr'wto n'hma diaswl'hnwshs
                                for(int i = 0; i < table.length; i++){
                                        num = table[i];
                                        result = num + (myID + 1);
                                        buff[myID+1].put(result);
                                }
                        else if (myID == nThreads-1)
                                // K'wdikas gia to teleuta'io n'hma diaswl'hnwshs
                                for(int i = 0; i < table.length; i++){
                                        num = buff[myID].get();
                                        result = num + (myID + 1);
                                        System.out.println("Num␣=␣" + result);
                                }
                        else
                                // K'wdikas gia ta endi'amesa n'hmata diaswl'hnwshs
                                for(int i = 0; i < table.length; i++){
                                        num = buff[myID].get();
                                        result = num + (myID + 1);
                                        buff[myID+1].put(result);
                                }
                }
        }
}
```

Σε αυτό το πρόγραμμα περιλαμβάνει την μέθοδο `main()` που περιέχει την δημιουργία πίνακα

ενταμιευτών και νημάτων καθώς επίσης περιλαμβάνει το κώδικα κάθε νήματος. Στο κώδικα κάθε νήματος πρέπει να λάβουμε υπόψη τις λειτουργίες που εκτελούν κάθε ενδιάμεσο νήμα και τα δύο άκρα νήματα της διασωλήνωσης. Το ενδιάμεσο νήμα εκτελεί το κύκλο: ανάγνωση από το αριστερό ενταμιευτή, επεξεργασία και εγγραφή στο δεξιό ενταμιευτή. Αντίθετα, το ένα αριστερό άκρο νήμα εκτελεί εγγραφή στο δεξιό ενταμιευτή και το άλλο δεξιό άκρο νήμα εκτελεί ανάγνωση από το αριστερό ενταμιευτή. Όμως, και τα δύο αυτά νήματα εκτελούν την επεξεργασία αύξησης του στοιχείου δεδομένων. Τέλος, στο κώδικα κάθε νήματος γίνεται ο έλεγχος της ταυτότητας νήματος `myID` και με βάση αυτό εκτελεί τις κατάλληλες λειτουργίες.

8.7 Πρότυπο JOMP

Το JOMP είναι ένα πρότυπο API (Application Programming Interace) για παράλληλο προγραμματισμό που δίνει την δυνατότητα στον προγραμματιστή να αναπτύξει παράλληλα προγράμματα που είναι συμβατά για τα περισσότερα συστήματα κοινής μνήμης. Το πρότυπο αυτό προσφέρει επίσης στον προγραμματιστή ένα σύνολο **οδηγίες** (directives) προς το μεταγλωττιστή που ενσωματώνονται στο πρόγραμμα για να εκφράσει παραλληλισμό σε ορισμένα σημεία του προγράμματος. Το πρότυπο JOMP αποτελείται από ένα μικρό σύνολο οδηγιών προς τον μεταγλωττιστή, ένα σύνολο συναρτήσεων βιβλιοθήκης και μεταβλητές περιβάλλοντος χρησιμοποιώντας την γλώσσα προγραμματισμού Java.

Το μοντέλο εκτέλεσης του JOMP είναι βασισμένο στο πολυνηματικό μοντέλο για την παράλληλη εκτέλεση των νημάτων. Αρχικά, ένα πρόγραμμα σε JOMP ξεκινάει με ένα μόνο νήμα το οποίο ονομάζεται **κύριο νήμα** (master thread) και εκτελεί την ακολουθιακή ροή του προγράμματος. Όταν το κύριο νήμα εισέλθει σε μια περιοχή του προγράμματος που είναι **παράλληλη** (parallel region) που σημαίνει ότι είναι ένα τμήμα κώδικα που πρέπει να εκτελεστεί παράλληλα, τότε το κύριο νήμα δημιουργεί πολλαπλά νήματα (ή μια ομάδα νημάτων) που εκτελούνται παράλληλα. Ο ακριβές αριθμός των νημάτων προσδιορίζεται με πολλούς τρόπους όπως θα συζητήσουμε παρακάτω. Όταν ολοκληρωθεί η εκτέλεση της παράλληλης περιοχής όλα τα νήματα τερματίζουν και συνεχίζει μόνο το κύριο νήμα.

Συνεπώς, ένα πρόγραμμα JOMP έχει την παρακάτω τυπική δομή:

```
import jomp.runtime.*;

public class example {
```

```
public static void main(String[] args){
   Sequential code
   ...
   //omp directive_name {
      Parallel region executed by all threads
   }
   ...
   Resume sequential code
}
}
```

Με βάση την παραπάνω δομή κάθε πρόγραμμα JOMP πρέπει να ξεκινάει με την δήλωση εισαγωγής του πακέτου `jomp.runtime` που είναι απαραίτητο για την μεταγλώττιση και εκτέλεση του προγράμματος. Επίσης, στο πρόγραμμα JOMP όπως γνωρίζουμε περιέχουν οδηγίες προς το μεταγλωττιστή σε μορφή εντολών `//omp`. Οι οδηγίες JOMP `//omp` συντάσσονται με την παρακάτω γενική δομή:

```
//omp directive_name [clause,...]
```

όπου `directive_name` είναι λέξη-κλειδί της οδηγίας και `clause` είναι επιπλέον παράμετροι για κάθε οδηγία και είναι προαιρετική η ύπαρξη τους εφόσον περικλείονται σε αγκύλες. Πολλές από τις παραμέτρους είναι κοινές στις οδηγίες και αφορά την εμβέλεια των μεταβλητών που θα δούμε αμέσως παρακάτω. Τέλος, κάθε οδηγία περικλείονται σε άγκιστρα όπου εισάγεται ένα τμήμα κώδικα που περιέχει μια εντολή ή εντολές.

Πρέπει να σημειώσουμε ότι το μοντέλο μνήμης του JOMP μας επιτρέπει να έχουμε **διαμοιραζόμενα** (shared data) και **ιδιωτικά δεδομένα** (private data) στο πρόγραμμα. Τα διαμοιραζόμενα δεδομένα αποθηκεύονται στην κύρια μνήμη και έχουν πρόσβαση από όλα τα νήματα. Αντίθετα, τα ιδιωτικά δεδομένα αποθηκεύονται στην ιδιωτική μνήμη του κάθε νήματος όπου τα υπόλοιπα νήματα δεν έχουν πρόσβαση. Συνεπώς, σε κάθε οδηγία που ενσωματώνουμε στο πρόγραμμα JOMP μπορούμε να ορίσουμε διαμοιραζόμενες και ιδιωτικές μεταβλητές.

Έτσι, οι μεταβλητές μπορούν να δηλωθούν ως διαμοιραζόμενες για όλα τα νήματα με την παράμετρο `shared(variable_list)` στις οδηγίες, όπου `variable_list` είναι οι μεταβλητές που θέλουμε να ορίσουμε. Αυτό σημαίνει ότι αν ένα νήμα αλλάξει την τιμή μιας τέτοιας μεταβλητής, η αλλαγή θα είναι ορατή σε όλα τα υπόλοιπα νήματα. Επίσης, ο προγραμματιστής είναι υπεύθυνος

να φροντίσει να μην δημιουργηθούν προβλήματα από τις ταυτόχρονες αναγνώσεις και εγγραφές της ίδιας μεταβλητής από πολλά νήματα. Από την άλλη μεριά, μπορούν να δηλωθούν μεταβλητές ως ιδιωτικές για κάθε νήμα με την παράμετρο `private(variable_list)` στις οδηγίες. Αυτό σημαίνει ότι κάθε νήμα έχει ένα δικό της αντίγραφο της μεταβλητής στην ιδιωτική της μνήμη. Επίσης, τα τοπικά αντίγραφα των μεταβλητών καταστρέφονται μετά την έξοδο από την παράλληλη περιοχή του προγράμματος. Τέλος, στην περίπτωση που δεν οριστούν μεταβλητές ως διαμοιραζόμενες ή ιδιωτικές στις οδηγίες τότε οι μεταβλητές που χρησιμοποιούνται στο πρόγραμμα θεωρούνται εξ ορισμού ως διαμοιραζόμενες.

Στις παρακάτω ενότητες θα παρουσιάζουμε τις πιο κοινές οδηγίες που παρέχονται από το JOMP κατά κατηγορίες. Συγκεκριμένα, θα δούμε μια **οδηγία παράλληλη** (parallel directive), **οδηγίες διαμοιρασμού εργασίας** (work-sharing directives), **οδηγίες παράλληλης διαμοιρασμένης εργασίας** (combined parallel work-sharing) και **οδηγίες συγχρονισμού** (synchronization directives).

8.7.1 Οδηγία Παράλληλη

Η πιο βασική οδηγία είναι η οδηγία `parallel`

```
//omp parallel
  structured_block
```

η οποία δημιουργεί πολλαπλά νήματα τα οποία εκτελούν το ίδιο κώδικα που προσδιορίζεται από το `structured_block` παράλληλα. Το `structured_block` μπορεί να περιέχει μια εντολή ή εντολές που περικλείονται με άγκιστρα {}. Στο τέλος του κώδικα, υπάρχει ένα φράγμα που όλα τα νήματα περιμένουν και τερματίζουν εκτός από το κύριο νήμα το οποίο συνεχίζει την εκτέλεση του προγράμματος.

Ο αριθμός των νημάτων που θα δημιουργηθεί μπορεί να οριστεί με τη χρήση της συνάρτησης βιβλιοθήκης `OMP.setNumThreads(int numThreads)` στο πρόγραμμα ή με την χρήση της μεταβλητής περιβάλλοντος `Djomp.threads` στην γραμμή εντολών εκτέλεσης του προγράμματος. Ο αριθμός των νημάτων μπορεί να αλλάζει δυναμικά, να μην είναι ο ίδιος σε κάθε οδηγία `parallel` που υπάρχει στο πρόγραμμα ώστε να κάνει καλύτερη χρήση των πόρων του συστήματος. Συνήθως, δεν προκύπτει καλύτερη απόδοση όταν ο αριθμός των νημάτων είναι μεγαλύτερο από το αριθμό των

επεξεργαστών που διαθέτει ένα σύστημα κοινής μνήμης επειδή τα νήματα πρέπει να καταμερισ-τούν ανάμεσα στους επεξεργαστές. Έτσι, καλύτερη χρήση των πόρων γίνεται όταν ο αριθμός των νημάτων είναι ίδιος με τον αριθμό των διαθέσιμων επεξεργαστών. Ο δυναμικός ορισμός του πλήθους νημάτων μπορεί να ενεργοποιηθεί πριν από την εκτέλεση του προγράμματος ορί-ζοντας την μεταβλητή περιβάλλοντος `Djomp.dynamic` σε `TRUE`. Επίσης, μπορεί να ενεργοποιηθεί ή να απενεργοποιηθεί κατά την διάρκεια εκτέλεσης του προγράμματος εκτός παράλληλων περιοχών με την βοήθεια της συνάρτησης `OMP.setDynamic(int dynamicThreads)`. Στην περίπτωση που ενεργοποι-ηθεί, η κάθε παράλληλη περιοχή θα χρησιμοποιεί έναν αριθμό νημάτων που αξιοποιεί καλύτερα τους πόρους του συστήματος. Ενώ, όταν η παράμετρος `dynamicThreads` είναι ίση με μηδέν τότε απενεργοποιείται ο δυναμικός ορισμός. Τέλος, όταν χρησιμοποιούμε την επιλογή του δυναμικού ορισμού νημάτων πρέπει το πρόγραμμα να είναι γραμμένο τέτοιο ώστε να λειτουργεί με διαφορε-τικό αριθμό νημάτων στις παράλληλες περιοχές.

Παρακάτω παρουσιάζουμε ένα πρώτο απλό πρόγραμμα σε JOMP, το οποίο δημιουργεί πέντε παράλληλα νήματα και κάθε νήμα εμφανίζει ένα μήνυμα της μορφής '`Hello from thread i of n`', όπου `i` είναι η σειρά του νήματος και το `n` είναι το πλήθος νημάτων.

```
import jomp.runtime.*;
public class Hello {
  public static void main (String argv[]) {
    int myid, num_threads;
    // J'etei 5 n'hmata gia ekt'elesh
    OMP.setNumThreads(5);
    //omp parallel private(myid,num_threads)
    {
       // Epistrof'h ths seir'as n'hmatos
       myid = OMP.getThreadNum();
       // Epistrof'h tou pl'hjous nhm'atwn
       num_threads = OMP.getNumThreads();
       System.out.println("Hello␣from␣thread␣" + myid + "␣of␣" + num_threads);
    }
  }
}
```

Από το παραπάνω κώδικα υπάρχουν δυο νέες συναρτήσεις, την `OMP.getNumThreads()` η οποί-α επιστρέφει τον αριθμό των νημάτων που εκτελούνται μέσα στην παράλληλη περιοχή και η συνάρτηση `OMP.getThreadNum()` η οποία επιστρέφει την σειρά του νήματος. Η σειρά ενός νήματος είναι ένας ακέραιος αριθμός που κυμαίνεται από 0 μέχρι $n-1$ όπου 0 είναι το κύριο νήμα και n είναι το πλήθος των νημάτων. Επίσης, στο πρόγραμμα δηλώνονται οι μεταβλητές `myid` και

num_threads ως ιδιωτικές στα νήματα.

Ο παραπάνω κώδικας που είναι γραμμένος σε JOMP για να εκτελεστεί πρέπει πρώτα να διερμηνευτεί σε πηγαίο κώδικα Java και έπειτα να μεταγλωττιστεί σε εκτελέσιμο αρχείο. Η διερμηνεία και η μεταγλώττιση γίνεται από την γραμμή εντολών. Η διερμηνεία γίνεται χρησιμοποιώντας την εντολή `java jomp.compiler.Jomp Hello` και η μεταγλώττιση με την εντολή `javac Hello.java`. Μετά την μεταγλώττιση προκύπτει το εκτελέσιμο αρχείο το οποίο αντιγράφεται αυτόματα στους επεξεργαστές από το σύστημα κοινής μνήμης (π.χ. SMP) και έπειτα κάθε επεξεργαστή αρχίζει την εκτέλεση του. Έτσι, η εκκίνηση εκτέλεσης του παραπάνω προγράμματος σε ένα σύστημα κοινής μνήμης γίνεται με την εντολή `java Hello`. Όμως, στην περίπτωση που στο πρόγραμμα δεν ορίζει το αριθμό των νημάτων με την χρήση της συνάρτησης `OMP.setNumThreads()`, μπορούμε να ορίζουμε εμείς τον αριθμό των νημάτων στην γραμμή εντολών με την μεταβλητή περιβάλλοντος `Djomp.threads` ως εξής: `java -Djomp.threads=10 Hello`.

8.7.2 Οδηγίες Διαμοιρασμού Εργασίας

Σε αυτή την κατηγορία υπάρχουν τρεις οδηγίες `for`, `sections` και `single` οι οποίες διαμοιράζουν την εργασία μιας περιοχής κώδικα ανάμεσα στα νήματα που έχουν δημιουργηθεί από μια οδηγία `parallel`. Για τις τρεις παραπάνω οδηγίες υπάρχει ένα φράγμα στο τέλος του κώδικα της περιοχής εκτός και αν συμπεριληφθεί η παράμετρος `nowait`. Τέλος, οι τρεις οδηγίες δεν δημιουργούν μια καινούργια ομάδα νημάτων.

Οδηγία `for`

Η οδηγία `for` συντάσσεται ως εξής:

```
//omp for
  for_loop
```

που προκαλεί τις επαναλήψεις του `for` να χωρίζονται σε τμήματα και τα τμήματα διαμοιράζονται ανάμεσα στα διάφορα νήματα παράλληλα. Με άλλα λόγια, όλα τα νήματα εκτελούν το ίδιο κώδικα αλλά σε διαφορετικά δεδομένα. Είναι γνωστό ότι με την οδηγία `for` πετυχαίνουμε την τεχνική του παραλληλισμού δεδομένων στην οποία αναπτύξαμε στην ενότητα 8.3 του κεφαλαίου αυτού. Η σύνταξη της επανάληψης `for` πρέπει να είναι σε κανονική μορφή (ή σε απλή μορφή) όπως μια εντολή `for` στην Java δηλαδή, πρέπει να περιέχει μια αρχική έκφραση, μια λογική συνθήκη και

τέλος μια απλή αριθμητική έκφραση όπως έκφραση αύξησης. Ο τρόπος που η επανάληψη `for` διαμοιράζεται ή τεμαχίζεται ανάμεσα στα νήματα γίνεται από μια παράμετρο που λέγεται `schedule`. Η παράμετρος αυτή συνοδεύεται από τέσσερα ορίσματα:

- Η παράμετρος `schedule(static, chunk_size)` χωρίζει τις επαναλήψεις του `for` σε τμήματα μεγέθους που προσδιορίζεται από το όρισμα `chunk_size` και διανέμονται στατικά ανάμεσα στα νήματα εκ περιστροφής. Σε περίπτωση που δεν οριστεί το `chunk_size`, τότε γίνεται ισομεγέθη τεμαχισμός αν αυτό βέβαια είναι εφικτό.

- Η παράμετρος `schedule(dynamic, chunk_size)` χωρίζει τις επαναλήψεις του `for` σε τμήματα μεγέθους `chunk_size`. Κάθε φορά που ένα νήμα ολοκληρώνει την εκτέλεση του τμήματος που του έχει ανατεθεί, του διανέμεται δυναμικά ένα άλλο τμήμα.

- Η παράμετρος `schedule(guided, chunk_size)` χωρίζει τις επαναλήψεις του `for` σε τμήματα μεγέθους τα οποία μειώνονται εκθετικά, καθώς γίνονται οι διανομές των τμημάτων. Η τιμή `chunk_size` ορίζει το μικρότερο τμήμα στο οποίο μπορεί να τεμαχίσει. Η προκαθορισμένη τιμή είναι 1.

- Η παράμετρος `schedule(runtime)` αναβάλει την απόφαση χρονοδρομολόγησης μέχρι να αρχίσει να εκτελείται το πρόγραμμα. Με αυτή την παράμετρο δεν χρειάζεται να οριστεί το `chunk_size`.

Παρακάτω, παρουσιάζεται ένα παράδειγμα της οδηγίας `for` που θέλουμε την πρόσθεση δύο μονοδιάστατων πινάκων που περιέχουν 1000 ακέραιους αριθμούς. Επίσης, υποθέτουμε ότι έχουμε 10 νήματα και κάθε νήμα προσθέτει τα στοιχεία των δύο πινάκων σε δικό του τμήμα των 100 στοιχείων. Τέλος, η ανάθεση των τμημάτων 100 στοιχείων στα νήματα γίνεται στατικά με την οδηγία `//omp for schedule(static, 100)`.

```
import jomp.runtime.*;
public class AddArrays {
  public static void main (String argv[]) {
    int a[] = new int[1000];
    int b[] = new int[1000];
    int c[] = new int[1000];
    OMP.setNumThreads(10);
    // Arqikopo'ihsh pin'akwn
    for(int i = 0;i < 1000; i++)
         a[i] = b[i] = i;
    /* Dhmiourg'ia 10 par'allhlwn nhm'atwn twn 100 stoiqe'iwn */
```

```
    //omp parallel shared(a,b,c)
    {
       //omp for schedule(static,100)
       for(int i = 0;i < 1000; i++)
          c[i] = a[i] + b[i]; // Pr'osjesh pin'akwn
    }
    for(int i = 0;i < 1000; i++)
         System.out.println("c␣[␣" + i + "␣]␣=␣" + c[i]);
  }
}
```

Οδηγία sections

Η οδηγία sections συντάσσεται ως εξής:

```
//omp sections
{
  //omp section
   structured_block
  //omp section
   structured_block
}
```

και τα τμήματα κώδικα που προσδιορίζονται από το structured_block εκτελούνται από διαφορετικά νήματα παράλληλα. Με άλλα λόγια, κάθε section που ορίζεται μέσα στην οδηγία sections είναι ένα τμήμα κώδικα που εκτελείται από ένα νήμα μόνο. Έτσι, κάθε section θα εκτελεστεί από διαφορετικό νήμα και η εκτέλεση των section είναι παράλληλη. Επομένως, η οδηγία sections χρησιμοποιείται για να εκφράσουμε παραλληλισμό ελέγχου. Υπάρχει περίπτωση το πλήθος των section να μην είναι ίδιο με το πλήθος των νημάτων που έχουν δημιουργηθεί. Σε αυτή την περίπτωση, κάποια νήματα θα είναι αδρανής ή θα εκτελέσουν παραπάνω από ένα section. Παρακάτω παρουσιάζεται ένα παράδειγμα της οδηγίας sections και ορίζουμε δύο section, ένα section για την εύρευση ενός μεγαλύτερου στοιχείου από μονοδιάστατο πίνακα και το άλλο section για την εύρευση ενός μικρότερου στοιχείου από μονοδιάστατο πίνακα. Έτσι, δημιουργούνται δύο νήματα που το κάθε νήμα εκτελεί μια διαφορετική λειτουργία.

```
import jomp.runtime.*;
public class MultiOper {
  public static void main (String argv[]) {
    OMP.setNumThreads(2);
    int i, max, min;
```

```
    // Dhmiourg'ia kai arqikopo'ihsh p'inaka
    int a[] = new int[1000];
    for(i=0; i < 1000; i++)
       a[i] = i + 5;
    // Dhmiourg'ia d'uo par'allhlwn diaforetik'wn nhm'atwn
    //omp parallel private(i,max,min)
    {
       //omp sections
       {
          // To 'ena n'hma ektele'i thn e'uresh m'egistou stoiqe'iou p'inaka
          //omp section
          {
             max = -999;
             for(i=0; i < 1000; i++)
                if (a[i] > max) max = a[i];
             System.out.println("The␣max␣number␣is␣" + max);
          }
          // To 'allo n'hma ektele'i thn e'uresh el'aqistou stoiqe'iou p'inaka
          //omp section
          {
             min = 999;
             for(i=0; i < 1000; i++)
                if (a[i] < min) min = a[i];
             System.out.println("The␣min␣number␣is␣" + min);
          }
       }
    }
  }
}
```

Οδηγία `single`

Η οδηγία `single` συντάσσεται ως εξής:

```
//omp single
  structured_block
```

και το τμήμα κώδικα που προσδιορίζεται από το `structured_block` θα εκτελεστεί από ένα νήμα μόνο. Η οδηγία `single` χρησιμοποιείται για λειτουργίες εισόδου δεδομένων από αρχείο ή πληκτρολόγιο.

8.7.3 Οδηγίες Παράλληλης Διαμοιρασμένης Εργασίας

Αν μια οδηγία `parallel` ακολουθείται από μια μόνο οδηγία `for`, τότε οι δυο αυτές οδηγίες μπορούν να συνδυαστούν σε μια οδηγία όπως φαίνεται παρακάτω:

```
//omp parallel for
  for_loop
```

Επίσης, αν μια οδηγία `parallel` ακολουθείται από μια μόνο οδηγία `sections`, τότε οι δυο αυτές οδηγίες μπορούν να συνδυαστούν σε μια οδηγία όπως φαίνεται παρακάτω:

```
//omp parallel sections {
  //omp section
    structured_block
  //omp section
    structured_block
}
```

8.7.4 Οδηγίες Συγχρονισμού

Όταν σε ένα πρόγραμμα εκτελούνται πολλαπλά νήματα και έχουν πρόσβαση σε διαμοιραζόμενα δεδομένα πρέπει να υπάρχει συγχρονισμός μεταξύ των νημάτων ώστε να εξασφαλίσουμε την ακεραιότητα των δεδομένων. Έτσι, σε αυτήν την κατηγορία υπάρχουν πέντε οδηγίες όπως είναι οι `critical`, `barrier`, `atomic`, `flush` και `ordered`.

Οδηγία `critical`

Η οδηγία `critical` περιέχει ένα τμήμα κώδικα και επιτρέπει σε ένα μόνο νήμα να εκτελέσει το τμήμα αυτό. Όταν ένα ή περισσότερα νήματα φτάσουν στην οδηγία `critical` όπως φαίνεται παρακάτω

```
//omp critical name
  structured_block
```

και κάποιο άλλο νήμα εκτελεί το κρίσιμο κώδικα που προσδιορίζεται από το `structured_block` τότε θα περιμένουν το άλλο νήμα να ολοκληρώσει την εκτέλεση του κρίσιμου κώδικα. Μόλις ολοκληρωθεί η εκτέλεση του κρίσιμου κώδικα τότε επιλέγεται ένα άλλο νήμα που ήταν στην αναμονή να εκτελέσει το κώδικα `structured_block`. Συνεπώς, η οδηγία αυτή υλοποιεί το αμοιβαίο αποκλεισμό του κρίσιμου τμήματος όπως είδαμε προηγουμένως στο κεφάλαιο αυτό.

Παρακάτω παρουσιάζεται ένα απλό παράδειγμα που υπολογίζει την τιμή του π κατά προσέγγιση. Η μέθοδος που επιλέξαμε για το υπολογισμό του π είναι απλή. Η μέθοδος αυτή υπολογίζει το ολοκλήρωμα της συνάρτησης $\frac{4}{(1+x*x)}$ ανάμεσα στο διάστημα $-\frac{1}{2}$ και $\frac{1}{2}$. Το ολοκλήρωμα προσεγγίζεται από το άθροισμα $nsteps$ διαστημάτων· η προσέγγιση στο ολοκλήρωμα

σε κάθε διάστημα είναι $(\frac{1}{nsteps}) * \frac{4}{(1+x*x)}$. Η τιμή $nsteps$ είναι γνωστή εκ των προτέρων. Σε αυτό το πρόγραμμα κάθε νήμα υπολογίζει μερικό άθροισμα για κάθε $nsteps$ διάστημα (δηλαδή, $x = -\frac{1}{2} + \frac{rank}{n}, -\frac{1}{2} + \frac{rank}{n} + \frac{size}{n}, ...$). Τα τοπικά αθροίσματα που υπολογίζονται από κάθε νήμα προστίθεται σε ένα γενικό αθροιστή με την χρήση συγχρονισμού νημάτων ώστε να προκύπτει το σωστό άθροισμα.

```
import jomp.runtime.*;
public class NumInt {
  public static void main (String argv[]) {
    OMP.setNumThreads(10);
    int myid, nthreads;
    // arijm'os diasthm'atwn
    int nsteps = 100000000;
    double step = 1.0/(double)nsteps;
    double partsum, x, pi;
    double sum = 0.0;
    // Dhmiourge'i par'allhla n'hmata
    //omp parallel private(myid, nthreads, partsum, x)
    {
      myid = OMP.getThreadNum();
      nthreads = OMP.getNumThreads();
      partsum = 0;
      // K'aje n'hma upolog'izei to merik'o 'ajroisma
      //omp for schedule(static,nthreads)
      for(int i = 0; i < nsteps ; i++) {
          x = (i + 0.5) * step;
          partsum = partsum + 4.0/(1.0 + x * x);
      }
      // Kr'isimo tm'hma gia to sunolik'o 'ajroisma
      //omp critical
      {
        sum = sum + partsum;
      }
    }
    pi = sum * step;
    System.out.println("pi=␣" + pi);
  }
}
```

Στην περίπτωση που παραλείπαμε την οδηγία `critical` και υπήρχει μόνο η εντολή άθροισης της μεταβλητής sum τότε δεν θα είχαμε το σωστό αποτέλεσμα.

Οδηγία `barrier`

Η οδηγία αυτή υλοποιεί το φράγμα για όλα τα νήματα. Όταν ένα νήμα φτάσει στην οδηγία όπως φαίνεται παρακάτω

```
//omp barrier
```

πρέπει να περιμένει τα υπόλοιπα νήματα να φτάσουν την οδηγία `barrier` και τότε να συνεχιστεί η εκτέλεση του κώδικα. Υπάρχουν περιορισμοί σε ότι αφορά την τοποθέτηση της οδηγίας `barrier` μέσα στο πρόγραμμα. Συγκεκριμένα, θα πρέπει όλα τα νήματα να είναι σε θέση να φτάσουν το σημείο του φράγματος ή την οδηγία.

Οδηγία `atomic`

Η οδηγία `atomic` που συντάσσεται ως εξής:

```
//omp atomic
  expression_statement
```

υλοποιεί ένα κρίσιμο τμήμα σε επίπεδο αριθμητικής έκφρασης που προσδιορίζεται από το `expression_statement` και όχι σε επίπεδο τμήμα κώδικα. Όταν χρησιμοποιείται η οδηγία αυτή, η αριθμητική έκφραση που ακολουθεί εκτελείται ατομικά. Η αριθμητική έκφραση πρέπει να είναι σε μια απλή μορφή.

Οδηγία `flush`

Τα διαμοιραζόμενα δεδομένα αρχικά αποθηκεύονται στην κοινή μνήμη και όταν προσπελάζονται από τους επεξεργαστές μερικά από τα δεδομένα αυτά να βρίσκονται στις προσωρινές όψεις των μνημών των νημάτων (δηλαδή να βρίσκονται στην κρυφή μνήμη). Για αυτό το λόγο, η οδηγία `flush` είναι ένα σημείο συγχρονισμού που προκαλεί το νήμα να έχει μια συνεπή μορφή όλων των διαμοιραζόμενων δεδομένων στην κοινή μνήμη. Έτσι, πρέπει να ολοκληρωθούν όλες οι λειτουργίες ανάγνωσης και εγγραφής των διαμοιραζόμενων μεταβλητών και οι τιμές τους να αντιγραφούν στην κοινή μνήμη. Η οδηγία `flush` έχει την παρακάτω δομή:

```
//omp flush (variable_list)
```

όπου `variable_list` μας δίνει την δυνατότητα να ορίζουμε μια λίστα μεταβλητών και μόνο αυτές θα αντιγραφούν στη κύρια μνήμη. Η οδηγία `flush` εμφανίζεται αυτόματα στην αρχή και στο τέλος των οδηγιών `parallel` και `critical` καθώς επίσης στο τέλος των οδηγιών `for`, `sections` και `single`.

Οδηγία `ordered`

Η οδηγία `ordered` που φαίνεται παρακάτω:

```
//omp ordered
  structured_block
```

χρησιμοποιείται σε συνδυασμό με τις οδηγίες `for` και `parallel for` και υποχρεώνει ότι οι επαναλήψεις του κώδικα `structured_block` να εκτελεστούν με την ίδια σειρά όπως στην περίπτωση που ο βρόχος εκτελούνταν ακολουθιακά.

8.8 Υλοποίηση Υπολογιστικών Μοντέλων με JOMP

Σε αυτή την ενότητα όπως προηγουμένως παρουσιάζουμε υλοποιήσεις υπολογιστικών μοντέλων με το πρότυπο JOMP.

8.8.1 Υλοποίηση Μοντέλου Συντονιστή - Εργαζόμενος

Παρακάτω παρουσιάζεται η υλοποίηση του μοντέλου συντονιστή - εργαζόμενου με δυναμική διανομή χρησιμοποιώντας το παράδειγμα της τετραγωνικής ρίζας των στοιχείων ενός πίνακα. Η βασική ιδέα της υλοποίησης είναι παρόμοια με εκείνη της υλοποίησης σε νήματα με την διαφορά ότι στο JOMP η δυναμική διανομή γίνεται πιο αυτόματα. Αυτή η αυτόματη διανομή γίνεται απλά ενσωματώνοντας την οδηγία `schedule` με παράμετρος `dynamic` στην επανάληψη υπολογισμού τετραγωνικής ρίζας των στοιχείων του πίνακα. Επίσης, η επανάληψη υπολογισμού χωρίζεται σε τμήματα μεγέθους 1, ώστε να ανατίθεται ένα στοιχείο στο νήμα κάθε φορά.

```
import jomp.runtime.*;
public class MasterWorker {
  public static void main (String argv[]) {
    int totalTasks = 1000;
    int nThreads = 10;
    double a[] = new double[totalTasks];
    OMP.setNumThreads(nThreads);
    // Arqikopo'ihsh p'inaka
    for(int i = 0;i < totalTasks; i++)
         a[i] = i;
    // Dhmiourge'i par'allhla n'hmata
    //omp parallel shared(a)
    {
```

```
        int numtasks_thread = 0;
        // Dunamik'h dianom'h twn stoiqe'iwn p'inaka sta n'hmata
        //omp for schedule(dynamic,1)
        for(int i = 0;i < totalTasks; i++) {
           a[i] = Math.sqrt(a[i]);
           numtasks_thread++;
        }
        System.out.println("Thread␣" + OMP.getThreadNum() + "␣takes␣" + numtasks_thread + "␣tasks");
    }
    // Emf'anish stoiqe'iwn p'inaka
    for(int i = 0; i < totalTasks; i++)
       System.out.println("a␣[␣" + i + "␣]␣=␣" + a[i]);
  }
}
```

Στο παραπάνω πρόγραμμα χρησιμοποιήσαμε μια επιπλέον μεταβλητή `numtasks_thread` που καταγράφει τον αριθμό των εργασιών που έλαβε κάθε νήμα.

8.8.2 Υλοποίηση Μοντέλου Διασωλήνωσης

Παρακάτω παρουσιάζεται η υλοποίηση του μοντέλου διασωλήνωσης νημάτων χρησιμοποιώντας το παράδειγμα που είχαμε παρουσιάζει στην ενότητα 8.6.2. Το παρακάτω πρόγραμμα είναι μια απλή μετατροπή του προγράμματος που είχαμε παρουσιάζει από Java Threads σε JOMP. Επίσης, στο πρόγραμμα χρησιμοποιήσαμε την κλάση `Buffer` για την δήλωση πίνακα ενταμιευτών που απαραίτητο για την λειτουργία της διασωλήνωσης.

```
import jomp.runtime.*;
public class Pipeline {
  public static void main (String argv[]) {
    int totalTasks = 10;
    int nThreads = 5;
    int myid,num,result;
    OMP.setNumThreads(nThreads);
    // Dhmiourg'ia kai arqikopo'ihsh p'inaka
    int a[] = new int[totalTasks];
    for(int i = 0;i < a.length; i++)
         a[i] = i;
    // Dhmiourg'ia p'inaka entamieut'wn
    Buffer buf[] = new Buffer[nThreads];
    for(int i = 0; i< buf.length; i++)
        buf[i] = new Buffer();
    // Trofod'othsh dedom'enwn sthn arq'h diaswl'hnwshs
    for(int i = 0; i < a.length; i++)
```

```
        buf[0].put(a[i]);
    // Dhmiourg'ia par'allhlwn nhm'atwn
    //omp parallel private(myid)
    {
        myid = OMP.getThreadNum();
        if (myid == (nThreads - 1))
                // K'wdikas gia to teleuta'io n'hma
                for(int i = 0; i< totalTasks; i++) {
                    num = buf[myid].get();
                    result = num + myid + 1;
                    System.out.println("Num␣=␣" + result);
                }
        else
                // K'wdikas gia ta up'oloipa n'hmata
                for(int i = 0; i< totalTasks; i++) {
                    num = buf[myid].get();
                    result = num + myid + 1;
                    buf[myid+1].put(result);
                }
        }
    }
}
```

Κεφάλαιο 9

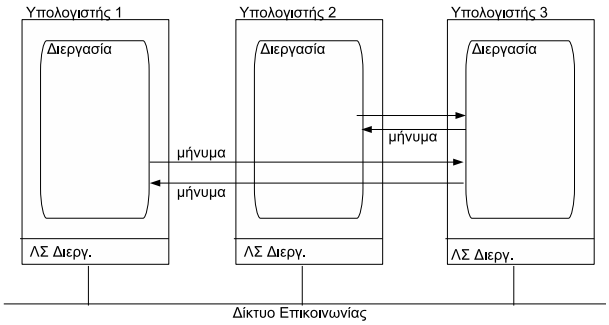
Προγραμματισμός Μεταβίβασης Μηνυμάτων

Στο κεφάλαιο αυτό πρώτα παρουσιάζουμε τις βασικές αρχές προγραμματισμού μεταβίβασης μηνυμάτων. Στην συνέχεια περιγράφουμε λεπτομέρειες για το πρότυπο βιβλιοθήκης επικοινωνίας MPJ που χρησιμοποιείται για προγραμματισμό σε συτοιχίες υπολογιστών.

9.1 Στοιχεία Προγραμματισμού Μεταβίβαση Μηνυμάτων

9.1.1 Μοντέλο

Γενικά, η μεταβίβαση μηνυμάτων είναι ένα μοντέλο παράλληλου προγραμματισμού που χρησιμοποιείται για την ανάπτυξη προγραμμάτων κατάλληλα για συστοιχίες υπολογιστών. Έτσι, η μεταβίβαση μηνυμάτων από την πλευρά του προγραμματιστή είναι ότι αποτελείται από διάφορους αυτόνομους υπολογιστές που το καθένα έχει την δική του τοπική μνήμη που αποθηκεύει δεδομένα ή ιδιωτικές μεταβλητές και επίσης κάθε υπολογιστής εκτελεί ένα πρόγραμμα ή μια διεργασία. Έτσι, ο υπολογιστής μπορεί να διαβάζει και να γράφει δεδομένα ελεύθερα, χρησιμοποιώντας την δική του τοπική μνήμη. Για την επίλυση ενός προβλήματος μεγάλης κλίμακας πολλές φορές απαιτεί την συνεργασία των διεργασιών που εκτελούνται στους αυτόνομους υπολογιστές και επομένως απαιτεί την επικοινωνία μεταξύ τους. Η επικοινωνία μεταξύ των διεργασιών επιτυγχάνεται με την ανταλλαγή μηνυμάτων (δηλαδή, αποστολή και παραλαβή μηνυμάτων) μέσω του δικτύου επικοινωνίας. Επίσης, η διεργασία έχει άμεση πρόσβαση μόνο στα δικά της δεδομένα τοπικής



Σχήμα 9.1: Μοντέλο προγραμματισμού μεταβίβασης μηνυμάτων

μνήμης και όχι στις μονάδες μνήμης των άλλων διεργασιών. Όμως, κάθε διεργασία μπορεί να διαβάζει τιμές δεδομένων από τη δική της τοπική μνήμη και να στέλνει αυτά τα δεδομένα σε οποιανδήποτε άλλη διεργασία. Έτσι, τα δεδομένα αυτά μπορούν να μοιράζονται και να ανταλλάσσονται ελεύθερα ανάμεσα στις διεργασίες. Στο Σχήμα 9.1 φαίνεται το μοντέλο προγραμματισμού μεταβίβασης μηνυμάτων. Σε αυτό το Σχήμα η διεργασία 2 που δρα ως αποστολέας στέλνει ένα μήνυμα στην διεργασία 3 που δρα ως παραλήπτης και αντίστροφα. Το μοντέλο μεταβίβασης μηνυμάτων έχει δύο μορφές επικοινωνίας: την **σημειακή** (point to point) και την **συλλογική** (collective) επικοινωνία. Η σημειακή επικοινωνία είναι όταν υπάρχει μεταβίβαση μηνυμάτων από μια διεργασία σε άλλη διεργασία όπως για παράδειγμα στο Σχήμα 9.1. Η συλλογική επικοινωνία είναι όταν υπάρχει μεταβίβαση μηνυμάτων από μια διεργασία σε μια ομάδα διεργασιών.

Σε πραγματικές εφαρμογές, τα προγράμματα μεταβίβασης μηνυμάτων χωρίζονται σε δύο μέρη: το μέρος του υπολογισμού που εκτελεί κάθε υπολογιστής το οποίο είναι γραμμένο σε μια από τις ακολουθιακές γλώσσες προγραμματισμού όπως C ή Java και το μέρος της επικοινωνίας που διαχειρίζεται την ανταλλαγή μηνυμάτων ανάμεσα στις διεργασίες χρησιμοποιούνται βιβλιοθήκες επικοινωνίας.

Με βάση τα παραπάνω για το μοντέλο προγραμματισμού μεταβίβασης μηνυμάτων προκύπτουν κάποια σημεία που πρέπει να εξεταστούν όπως ποιες διεργασίες εκτελούνται, πως γίνεται η μεταβίβαση μηνυμάτων ανάμεσα στις διεργασίες και ποιο είναι το περιεχόμενο των μηνυμάτων. Επομένως για το μοντέλο μεταβίβασης μηνυμάτων υπάρχουν δύο βασικά θέματα, το ένα είναι η μέθοδος δημιουργίας των διεργασιών που εκτελούνται σε διαφορετικούς υπολογιστές και

το άλλο είναι η μέθοδος αποστολής - παραλαβής μηνυμάτων καθώς και το συγχρονισμό των διεργασιών.

9.1.2 Δημιουργία Διεργασιών

Είναι γνωστό ότι για την επίλυση ενός προβλήματος σε λιγότερο χρόνο από ότι ένα ακολουθιακό πρόγραμμα απαιτεί την συγγραφή ενός παράλληλου προγράμματος. Το παράλληλο πρόγραμμα αποτελείται από έναν αριθμό διεργασιών που εκτελούνται σε διαφορετικούς υπολογιστές μιας συστοιχία υπολογιστών. Έτσι, αν υπάρχουν 10 διεργασίες και 10 υπολογιστές τότε μια διεργασία εκτελείται σε κάθε υπολογιστή. Όμως σε περίπτωση που υπάρχουν περισσότερες διεργασίες από τον αριθμό των διαθέσιμων υπολογιστών τότε σε κάθε υπολογιστή εκτελούνται περισσότερες από μια διεργασίες με τρόπο χρονοκαταμερισμού και αυτό έχει σαν αποτέλεσμα την επιβράδυνση του παράλληλου προγράμματος. Η επιβράδυνση του προγράμματος οφείλεται κυρίως στο υψηλό κόστος εναλλαγής περιβάλλοντος διεργασιών (όπως μετρητής προγράμματος, στοίβα κλπ). Υπάρχουν όμως περιπτώσεις να εκτελέσουμε το παράλληλο πρόγραμμα με περισσότερες από μια διεργασίες σε ένα μόνο υπολογιστή και γίνεται όταν θέλουμε να κάνουμε αποσφαλμάτωση του προγράμματος πριν την εκτέλεση του σε μια συστοιχία υπολογιστών. Σε γενικές γραμμές, στο παράλληλο προγραμματισμό υποθέτουμε ότι μια διεργασία εκτελείται σε κάθε υπολογιστή του παράλληλου συστήματος.

Για την εκτέλεση του παράλληλου προγράμματος είναι απαραίτητο λοιπόν να δημιουργηθούν οι διεργασίες και να αρχίζουν την εκτέλεση τους. Υπάρχουν δύο μέθοδοι δημιουργίας διεργασιών, η **στατική δημιουργία διεργασιών** (static process creation) και η **δυναμική δημιουργία διεργασιών** (dynamic process creation). Στην στατική δημιουργία διεργασιών, όλες οι διεργασίες είναι προκαθορισμένες πριν την εκτέλεση και το σύστημα θα εκτελέσει ένα σταθερό αριθμό διεργασιών. Αντίθετα, η δυναμική δημιουργία διεργασιών, οι διεργασίες μπορούν να δημιουργηθούν κατά την διάρκεια εκτέλεση του προγράμματος. Βέβαια, η δυναμική δημιουργία διεργασιών είναι η πιο ισχυρή τεχνική από την στατική δημιουργία διεργασιών αλλά εμφανίζει πρόσθετη επιβάρυνση λόγω του υψηλού κόστους δημιουργίας διεργασιών.

Σε περισσότερες εφαρμογές, όλες οι διεργασίες δεν είναι ούτε ίδιες αλλά ούτε διαφορετικές. Έτσι, συνήθως υπάρχει μια βασική διεργασία που ονομάζεται διεργασία ''συντονιστή'' (master process) και οι υπόλοιπες διεργασίες είναι ίδιες που ονομάζονται διεργασίες ''εργαζόμενοι''

(worker processess) οι οποίες διαφοροποιούνται με βάση την ταυτότητα ή σειρά διεργασίας (process identification). Η σειρά διεργασία είναι ένας ακέραιος αριθμός και οι διεργασίες είναι αριθμημένες από το 0 έως $p-1$ όταν υπάρχουν p διεργασίες. Με βάση την σειρά διεργασία μπορούμε να αλλάζουμε τις ενέργειες της διεργασίας ή να χρησιμοποιηθεί στο προσδιορισμό των διεργασιών παραληπτών σε περίπτωση που στέλνουμε μηνύματα. Οι διεργασίες ορίζονται από προγράμματα που είναι γραμμένα από τον προγραμματιστή.

Το πιο συνηθισμένο τρόπο προγραμματισμού για την περίπτωση στατικής δημιουργίας διεργασιών είναι το μοντέλο **μοναδικού προγράμματος, πολλαπλών δεδομένων** (Single Program Multiple Data - SPMD). Σύμφωνα με το μοντέλο αυτό διαφορετικά προγράμματα συγχωνεύονται σε ένα πρόγραμμα, δηλαδή το τμήμα κώδικα του συντονιστή και του εργαζόμενου είναι σε ένα πρόγραμμα. Μέσα σε αυτό το πρόγραμμα υπάρχουν εντολές ελέγχου ώστε να διαχωρίζει διαφορετικά μέρη κώδικα που θα εκτελεστεί για κάθε διεργασία. Μετά την ανάπτυξη του προγράμματος, το πρόγραμμα μεταγλωττίζεται σε εκτέλεσιμο κώδικα για κάθε υπολογιστή. Ο κάθε υπολογιστής θα φορτώσει ένα αντίγραφο του κώδικα αυτού στην τοπική του μνήμη για εκτέλεση και όλοι οι υπολογιστές μπορούν να ξεκινήσουν να εκτελούν τον κώδικα τους μαζί. Σε περίπτωση που οι υπολογιστές είναι ετερογενείς (δηλαδή διαφορετικούς τύπους επεξεργαστών), τότε ο πηγαίος κώδικας πρέπει να μεταγλωττιστεί για κάθε διαφορετικό τύπο επεξεργαστή ξεχωριστά. Σε αυτό το βιβλίο θα ακολουθήσουμε το μοντέλο SPMD, το οποίο είναι το πιο κοινό για την περίπτωση συστήματος μεταβίβασης μηνυμάτων MPI.

Αντίθετα, ο τρόπος προγραμματισμού για την δυναμική δημιουργία διεργασιών είναι το **μοντέλο πολλαπλών προγραμμάτων, πολλαπλών δεδομένων** (Multiple Program, Multiple Data - MPMD). Με το μοντέλο αυτό υπάρχουν ξεχωριστά και διαφορετικά προγράμματα για κάθε υπολογιστή. Έτσι υπάρχουν δυο διαφορετικά προγράμματα, ένα πρόγραμμα συντονιστής και ένα πρόγραμμα εργαζόμενος. Ο ένας επεξεργαστής εκτελεί το πρόγραμμα συντονιστής και οι υπόλοιποι υπολογιστές εκτελούν παρόμοια προγράμματα εργαζόμενος. Το μοντέλο αυτό το ακολουθεί ένα άλλο σύστημα πέρασμα μηνύματων, το PVM το οποίο υποστηρίζει την δυναμική δημιουργία διεργασιών. Όμως, το PVM δεν χρησιμοποιείται σήμερα πολύ συχνά.

9.1.3 Συναρτήσεις Σημειακής Επικοινωνίας

Η μεταβίβαση μηνυμάτων ανάμεσα σε δύο διεργασίες αποστολέα και παραλήπτη υποστηρίζεται από δύο βασικές σημειακές συναρτήσεις της μορφής:

```
send(parameter_list)

recv(parameter_list)
```

όπου η συνάρτηση `send()` τοποθετείται στην διεργασία - αποστολέα που θα στείλει το μήνυμα σε μια διεργασία - παραλήπτη και η συνάρτηση `recv()` τοποθετείται στην διεργασία παραλήπτη που θα παραλάβει το μήνυμα από μια διεργασία - αποστολέα. Οι πραγματικοί παράμετροι των συναρτήσεων εξαρτάται από το λογισμικό επικοινωνίας και σε μερικές περιπτώσεις μπορεί να είναι πολύπλοκοι. Οι πιο απλοί παράμετροι που θα μπορούσε να είναι στην συνάρτηση `send()` είναι η σειρά διεργασία παραλήπτη και το μήνυμα, ενώ στην συνάρτηση `recv()` να είναι η σειρά διεργασία αποστολέα και η θέση που θα αποθηκεύει το μήνυμα. Για την γλώσσα προγραμματισμού Java, θα μπορούσαμε να είχαμε την κλήση

```
send(x, dest_id);
```

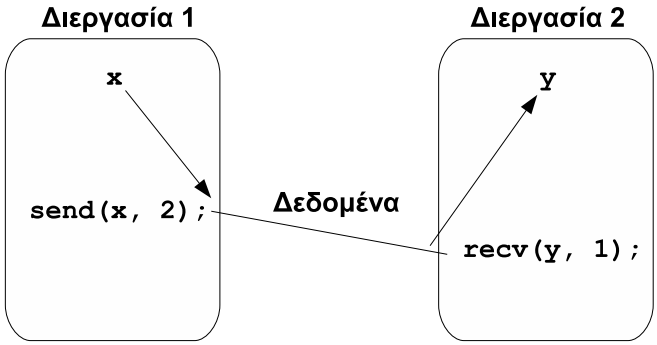
στην διεργασία αποστολέα και την κλήση

```
recv(y, source_id);
```

στην διεργασία παραλήπτη. Στο Σχήμα 9.2 παρουσιάζεται η διαδικασία αποστολής ενός δεδομένου x από την διεργασία αποστολέα με σειρά 1 στην διεργασία παραλήπτη με σειρά 2. Σε αυτό το παράδειγμα, η μεταβλητή x πρέπει να έχει φορτώσει την τιμή της πριν την αποστολή και οι x και y πρέπει να είναι ίδιου τύπου και μεγέθους. Η σειρά των παραμέτρων εξαρτάται από το σύστημα. Περισσότερες λεπτομέρειες και παραλλαγές παραμέτρων των πραγματικών συναρτήσεων μεταβίβασης μηνυμάτων θα αναφερθούμε παρακάτω αφού πρώτα παρουσιάζουμε τους βασικούς μηχανισμούς.

9.1.4 Σύγχρονη και Ασύγχρονη Μεταβίβαση Μηνυμάτων

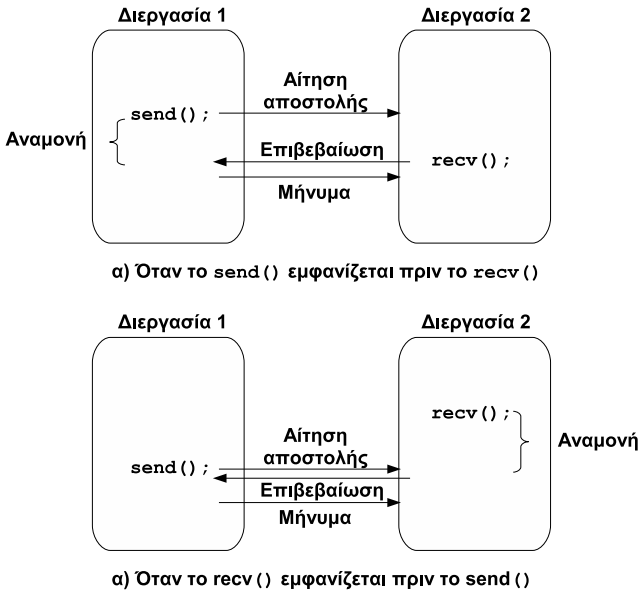
Η επικοινωνία ανάμεσα σε δύο διεργασίες αποστολέα και παραλήπτη μπορεί να είναι είτε σύγχρονη ή ασύγχρονη. Επομένως, οι συναρτήσεις `send/receive` υποστηρίζουν σύγχρονη και ασύγχρονη επικοινωνία. Ο όρος **σύγχρονος** (synchronous) χρησιμοποιείται για τις συναρτήσεις που επιστρέφουν τον έλεγχο τους μετά την ολοκλήρωση της μεταφοράς μηνύματος. Οι σύγχρονες συναρτήσεις δεν χρειάζονται ενταμιευτή μηνυμάτων. Μια σύγχρονη συνάρτηση `send` περιμένει



Σχήμα 9.2: Πέρασμα μηνύματος ανάμεσα σε δύο διεργασίες

μέχρι ολόκληρο το μήνυμα να γίνει αποδεκτό από την διεργασία παραλήπτη πριν επιστρέψει τον έλεγχο της. Μια σύγχρονη συνάρτηση `receive` θα περιμένει μέχρι να φτάσει και να αποθηκευτεί το μήνυμα. Έτσι ένα ζεύγος διεργασιών, μια διεργασία με την σύγχρονη λειτουργία `send` και μια διεργασία με την σύγχρονη λειτουργία `receive`, συγχρονίζονται με αποτέλεσμα ούτε η διεργασία αποστολέα ούτε η διεργασία παραλήπτη θα συνεχίσουν την εκτέλεση τους μέχρι ότου το μήνυμα να έχει μεταφερθεί από τον αποστολέα στον παραλήπτη. Συνεπώς, οι σύγχρονες συναρτήσεις έχουν την ιδιότητα να εκτελούν εσωτερικά δύο λειτουργίες: να μεταφέρουν το μήνυμα και να συγχρονίζουν τις δύο διεργασίες. Ο όρος **ραντεβού** (rendezvous) χρησιμοποιείται για να περιγράψει την συνάντηση και τον συγχρονισμό των διεργασιών που υπάρχει μέσω των σύγχρονων συναρτήσεων `send/receive`.

Οι σύγχρονες συναρτήσεις `send` και `receive` έχουν μια μορφή σηματοδότησης όπως ακολουθούν το **πρωτόκολλο τριών φάσεων** (three-way protocol). Έτσι, στην πρώτη φάση ο αποστολέας στέλνει ένα μήνυμα του τύπου ''αίτηση για αποστολή'' στο παραλήπτη. Στην συνέχεια, όταν ο παραλήπτης είναι έτοιμος να λάβει το μήνυμα τότε στέλνει ένα μήνυμα επιβεβαίωσης στον αποστολέα. Τέλος, με την λήψη του μηνύματος επιβεβαίωσης, ο αποστολέας στέλνει το πραγματικό μήνυμα. Στο Σχήμα 9.3 παρουσιάζεται η σύγχρονη μεταβίβαση μηνυμάτων χρησιμοποιώντας το πρωτόκολλο τριών φάσεων. Στο Σχήμα 9.3α, η διεργασία 1 φτάνει στην συνάρτηση `send()` πριν η διεργασία 2 φτάσει την αντίστοιχη συνάρτηση `recv()`. Σε αυτή την περίπτωση, η διεργασία 1 μπαίνει σε αναμονή μέχρι η διεργασία 2 να φτάσει στην `recv()`. Μέχρι να φτάσει, η διεργασία 2 στέλνει ένα μήνυμα επιβεβαίωσης σε μορφή σήματος



Σχήμα 9.3: Πρωτόκολλο τριών φάσεων

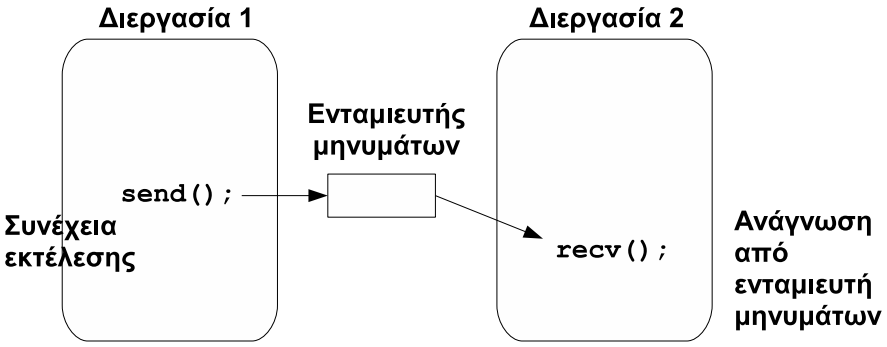
στην διεργασία 1 και τότε και οι δύο διεργασίες συμμετέχουν στην διαδικασία μεταφορά μηνύματος. Σημειώνουμε ότι στο Σχήμα 9.3α το μήνυμα είναι αποθηκευμένο στον αποστολέα μέχρι να αποσταλεί. Στο Σχήμα 9.3β, η διεργασία 2 φτάνει πρώτα στην συνάρτηση `recv()` πριν η διεργασία 1 φτάσει στην `send()`. Σε αυτή την περίπτωση, η διεργασία 2 μπαίνει σε αναμονή μέχρι να ξεκινήσει η διεργασία 1. Ο ακριβής μηχανισμός για αναμονή και εκτέλεση διεργασιών εξαρτάται από το σύστημα.

Ο όρος **ασύγχρονος** (asynchronous) χρησιμοποιείται στις συναρτήσεις που δεν περιμένουν να εκτελεστούν κάποιες λειτουργίες (όπως είδαμε στο σύγχρονο) πριν επιστρέψουν τον έλεγχο τους. Οι συναρτήσεις αυτές απαιτούν ενταμιευτή μηνυμάτων. Γενικά, οι συναρτήσεις αυτές δεν συγχρονίζουν τις διεργασίες αλλά επιτρέπουν να συνεχίσουν την εκτέλεση τους αμέσως. Σε αυτές τις περιπτώσεις χρειάζονται προσοχή γιατί μπορεί εύκολα να συμβούν σφάλματα στην διαδικασία ανταλλαγής μηνυμάτων.

Επίσης, υπάρχει ο όρος **ανασταλτικός** (blocking) που χρησιμοποιείται για να περιγράψει τις συναρτήσεις που δεν επιστρέφουν τον έλεγχο τους (δηλαδή, δεν επιτρέπουν την διεργασία να συνεχίσει την εκτέλεση του μέχρι να ολοκληρωθεί η μεταφορά). Έτσι, ο όρος αυτός είναι συνώνυμο με το όρο σύγχρονος. Αντίθετα, ο όρος **μη-ανασταλτικός** (non-blocking)

χρησιμοποιείται για να περιγράψει τις συναρτήσεις που επιστρέφουν το έλεγχο τους ανεξάρτητα αν έχει ολοκληρωθεί η μεταφορά μηνύματος. Όμως, οι όροι ανασταλτικός και μη-ανασταλτικός έχουν επανακαθοριστεί σε συστήματα όπως το MPJ που θα δούμε παρακάτω.

Γενικά, στην ασύγχρονη μεταβίβαση μηνυμάτων είναι αναγκαίο ένα ενταμιευτή ανάμεσα στον αποστολέα και στον παραλήπτη ώστε να αποθηκεύει τα μηνύματα, όπως φαίνεται στο Σχήμα 9.4. Εδώ, ο ενταμιευτής χρησιμοποιείται για να αποθηκεύσει τα μηνύματα που στέλνονται



Σχήμα 9.4: Χρήση ενταμιευτή μηνυμάτων

πριν ληφθούν από την συνάρτηση `recv()`. Για την συνάρτηση `recv()` πρέπει να λάβει το μήνυμα αν το χρειάζεται. Αν η συνάρτηση `recv()` φτάσει πρώτη πριν την `send()`, ο ενταμιευτής θα είναι άδειος και επομένως η συνάρτηση `recv()` περιμένει για το μήνυμα. Αλλά για την συνάρτηση `send()` μόλις τοποθετήσει το μήνυμα στον ενταμιευτή με ασφάλεια, τότε η διεργασία μπορεί να συνεχίσει την εκτέλεση του. Με αυτό τον τρόπο, χρησιμοποιώντας ασύγχρονες συναρτήσεις `send` μειώνεται ο συνολικός χρόνος εκτέλεσης. Επίσης, στην πράξη ο ενταμιευτής μηνυμάτων έχει περιορισμένη χωρητικότητα και η συνάρτηση `send` σταματά όταν ο χώρος του ενταμιευτή εξαντλείται. Έτσι, είναι απαραίτητο να γνωρίζουμε σε κάποιο σημείο αν έχει φτάσει το μήνυμα πράγμα το οποίο απαιτεί πρόσθετο πέρασμα μηνύματος.

Εδώ θα δώσουμε τους ορισμούς του MPJ για τους όρους ανασταλτικός και μη-ανασταλτικός. Ο όρος ανασταλτικός χρησιμοποιείται σε συναρτήσεις που χρησιμοποιούν ένα ενταμιευτή και επιστρέφουν τον έλεγχο τους αφού ολοκληρωθούν οι τοπικές λειτουργίες (π.χ. αντιγραφή του μηνύματος στο ενταμιευτή) ανεξάρτητα αν έχει ολοκληρωθεί η μεταφορά του μηνύματος. Ενώ οι συναρτήσεις που επιστρέφουν αμέσως τον έλεγχο τους είναι οι μη-ανασταλτικές.

9.1.5 Επιλογή Μηνυμάτων

Μέχρι στιγμή έχουμε περιγράψει μηνύματα τα οποία αποστέλονται σε μια συγκεκριμένη διεργασία παραλήπτη από μια συγκεκριμένη διεργασία αποστολέα, όπου η σειρά διεργασία παραλήπτη δίνεται ως παράμετρος στην συνάρτηση `send()` και η σειρά διεργασία αποστολέα ως παράμετρος στην συνάρτηση `recv()`. Έτσι, η συνάρτηση `recv()` θα δεχτεί μηνύματα από μια διεργασία αποστολέα που προσδιορίζεται ως παράμετρος στην συνάρτηση `recv()` και θα αγνοήσει τα υπόλοιπα μηνύματα. Στην μεταβίβαση μηνυμάτων παρέχεται ένας ειδικός χαρακτήρας ή αριθμός που λειτουργεί ως μια αδιάφορη διεύθυνση αποστολέα ώστε να επιτρέπει στην διεργασία παραλήπτη να δέχεται μηνύματα από οποιοδήποτε αποστολέα. Για παράδειγμα, ο αριθμός -1 μπορεί να χρησιμοποιηθεί ως μια αδιάφορη διεύθυνση αποστολέα.

Επίσης, για να υπάρχει ακόμα μεγαλύτερη ευελιξία τα μηνύματα μπορούν να φιλτράρονται ή να επιλέγονται με βάση μια **ετικέτα μηνύματος** (message tag) που είναι ενσωματωμένη μέσα στο μήνυμα. Η ετικέτα μηνύματος `msgtag` είναι ένας προκαθορισμένος θετικός ακέραιος αριθμός από τον προγραμματιστή που μπορεί να χρησιμοποιηθεί για να διακρίνει ανάμεσα στους διαφορετικούς τύπους μηνυμάτων τα οποία στέλνονται. Έτσι, μπορούμε να προσδιορίζουμε συγκεκριμένες συναρτήσεις `recv()` να δέχονται μόνο τα μηνύματα εκείνα που έχουν μια συγκεκριμένη ετικέτα μηνύματος και να αγνοούν τα υπόλοιπα μηνύματα. Η ετικέτα μηνύματος λογικά θα είναι μια πρόσθετη παράμετρος στις συναρτήσεις `send()` και `recv()`. Για παράδειγμα, για να στείλουμε ένα μήνυμα x με ετικέτα μηνύματος 5 από μια διεργασία αποστολέα 1 σε μια διεργασία παραλήπτη 2 που θα τοποθετηθεί στην y, τότε θα έχουμε την κλήση

```
send(x, 2, 5);
```

στην διεργασία αποστολέα και την κλήση

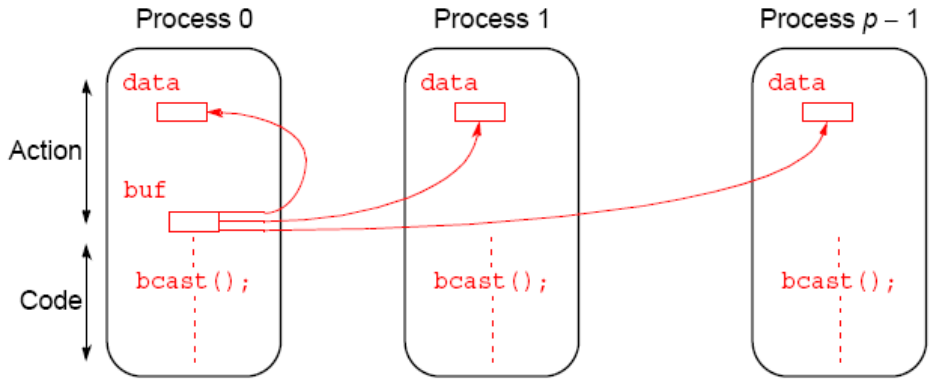
```
recv(y, 1, 5);
```

στην διεργασία παραλήπτη. Η ετικέτα του μηνύματος μεταφέρεται μαζί με το μήνυμα. Στην περίπτωση που δεν απαιτείται ταύτιση στους τύπους μηνυμάτων μπορεί στην θέση ετικέτα μηνύματος να χρησιμοποιηθεί ένα αδιάφορο χαρακτήρα έτσι ώστε η συνάρτηση `recv()` να ταυτίζεται με μια οποιαδήποτε `send()`.

9.1.6 Συναρτήσεις Συλλογικής Επικοινωνίας

Μέχρι τώρα είδαμε μεταβίβαση μηνυμάτων ανάμεσα σε δύο μόνο διεργασίες. Όμως, υπάρχει πολλές φορές ανάγκη μια διεργασία αποστολέα να στείλει το ίδιο μήνυμα σε περισσότερες από μία διεργασίες παραλήπτη. Η συλλογική επικοινωνία υποστηρίζεται από τις συναρτήσεις εκπομπή, διασκορπισμός, συλλογή και αναγωγή όπως θα δούμε παρακάτω.

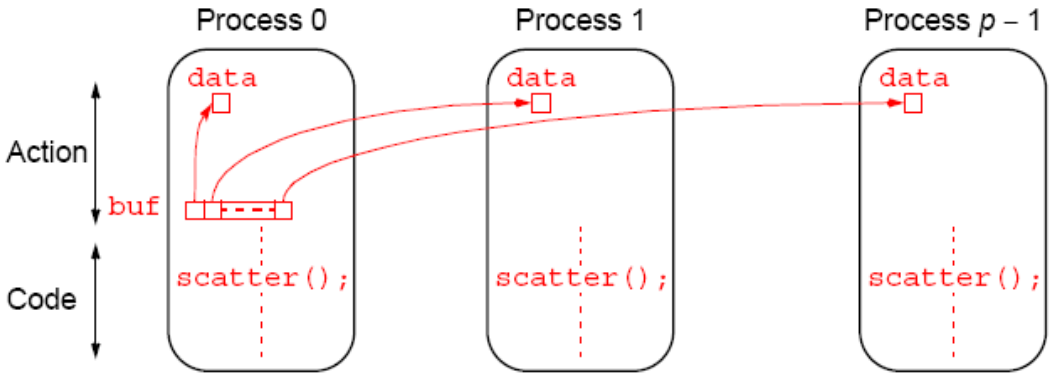
Ο όρος **εκπομπή** (broadcast) χρησιμοποιείται για να περιγράψει την αποστολή του ίδιου μηνύματος σε όλες τις υπόλοιπες διεργασίες. Στο Σχήμα 9.5 παρουσιάζεται ένα παράδειγμα εκπομπής. Οι διεργασίες που συμμετέχουν στην εκπομπή πρέπει πρώτα να αναγνωρίζονται και να σχηματιστεί ένα κανάλι επικοινωνίας (ή μια ομάδα διεργασιών) το οποίο θα χρησιμοποιηθεί ως παράμετρος στις συναρτήσεις εκπομπής. Στο Σχήμα 9.5, η διεργασία 0 αναγνωρίζεται ως η διεργασία ρίζα και η διεργασία ρίζα μπορεί να είναι οποιαδήποτε διεργασία στην ομάδα επικοινωνίας. Επίσης, στο Σχήμα όλες οι διεργασίες εκτελούν την ίδια συνάρτηση `bcast()` και αυτό έχει σαν αποτέλεσμα τα δεδομένα που είναι αποθηκευμένα στο `buff` της διεργασίας 0 να αντιγραφούν στις μεταβλητές `data` των αντίστοιχων διεργασιών.



Σχήμα 9.5: Εκπομπή

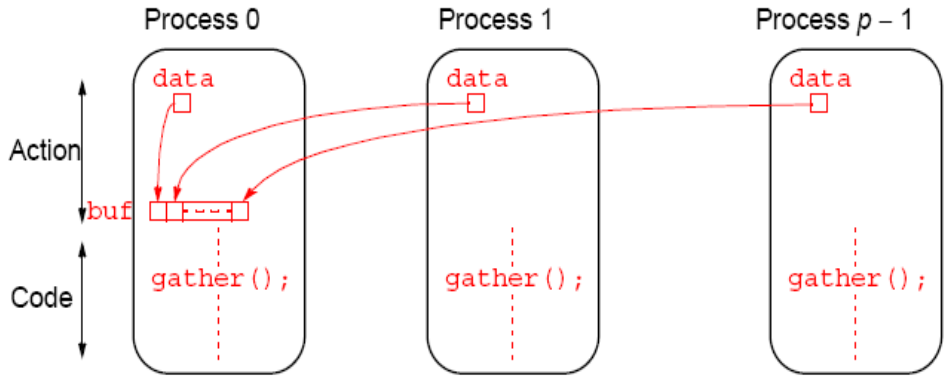
Ο όρος **διασκορπισμός** (scatter) χρησιμοποιείται για να περιγράψει την αποστολή κάθε ξεχωριστού στοιχείου ενός πίνακα δεδομένων που βρίσκεται στην διεργασία ρίζα προς κάθε διεργασία παραλήπτη. Δηλαδή, τα περιεχόμενα της θέσης i του πίνακα να σταλεί στην διεργασία i. Έτσι, η εκπομπή στέλνει ίδια δεδομένα σε όλες τις διεργασίες ενώ ο διασκορπισμός διανέμει διαφορετικά δεδομένα σε κάθε διεργασία. Στο Σχήμα 9.6 παρουσιάζεται η διαδικασία του διασκορπισμού. Όπως και στην εκπομπή, πρέπει να αναγνωριστεί μια ομάδα διεγασιών και

διεργασία ρίζα. Επίσης, στο Σχήμα όλες οι διεργασίες εκτελούν την ίδια συνάρτηση `scatter()` και αυτό έχει σαν αποτέλεσμα τα στοιχεία δεδομένων που είναι αποθηκευμένα στο `buff` της διεργασίας 0 να αντιγραφούν ξεχωριστά στις μεταβλητές `data` των αντίστοιχων διεργασιών.



Σχήμα 9.6: Διασκορπισμός

Ο όρος **συλλογή** (gather) χρησιμοποιείται για να περιγράψει μια διεργασία που συλλέγει ξεχωριστές τιμές από ένα σύνολο διεργασιών και τις αποθηκεύει σε ένα πίνακα. Η συλλογή συνήθως χρησιμοποιείται μετά τους υπολογισμούς που πραγματοποιούν οι διεργασίες. Ουσιαστικά, η συλλογή είναι ακριβώς αντίθετο του διασκορπισμού. Τα δεδομένα από την διεργασία i λαμβάνονται από την διεργασία ρίζα και τοποθετούνται στην θέση i του πίνακα που βρίσκεται στην διεργασία ρίζα. Στο Σχήμα 9.7 παρουσιάζεται η διαδικασία της συλλογής. Στο Σχήμα όλες

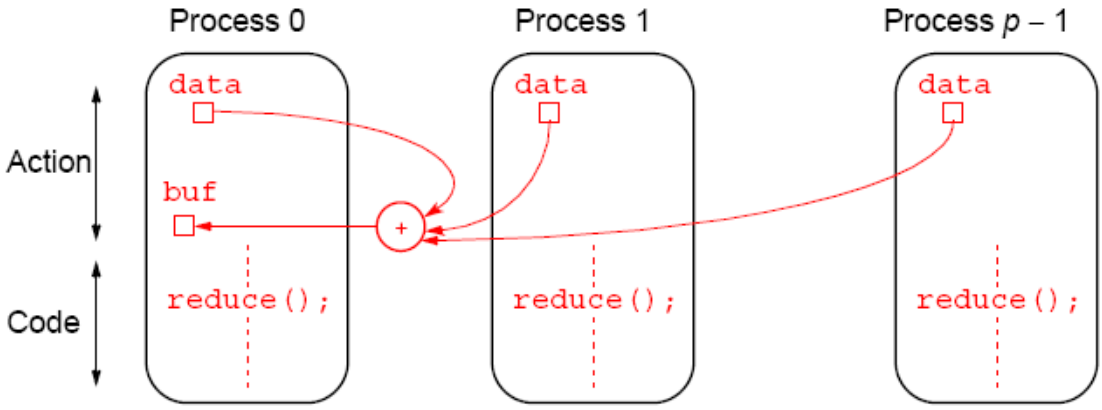


Σχήμα 9.7: Συλλογή

οι διεργασίες εκτελούν την ίδια συνάρτηση `gather()` και αυτό έχει σαν αποτέλεσμα τα δεδομένα

που είναι αποθηκευμένα στο `data` των διεργασιών να αντιγραφούν ξεχωριστά στις αντίστοιχες θέσεις του `buff` της διεργασίας 0.

Μερικές φορές η λειτουργία συλλογή μπορεί να συνδυαστεί με μια καθορισμένη αριθμητική ή λογική πράξη. Για παράδειγμα, οι τιμές που συλλέγονται θα μπορούσε να αθροίζονται όλα μαζί από την διεργασία ρίζα, όπως φαίνεται στο Σχήμα 9.8. Τέτοια λειτουργία ονομάζεται μερικές φορές **αναγωγή** (reduction). Στο Σχήμα όλες οι διεργασίες εκτελούν την ίδια συνάρτηση `reduce()` και αυτό έχει σαν αποτέλεσμα τα δεδομένα που είναι αποθηκευμένα στο `data` των διεργασιών να αθροιστούν και το αποτέλεσμα να αντιγραφεί στο `buff` της διεργασίας 0. Σε αυτό το



Σχήμα 9.8: Αναγωγή

παράδειγμα, η πράξη που εκτέλεσε η αναγωγή ήταν πρόσθεση αλλά θα μπορούσε να είναι μια άλλη αριθμητική ή λογική πράξη.

9.2 Βιβλιοθήκη MPJ

9.2.1 Εισαγωγή

Σε αυτή την ενότητα θα συσχετίζουμε τις αρχές της μεταβίβασης μηνυμάτων πάνω στις συστοιχίες υπολογιστών. Είναι γνωστό ότι υπάρχουν διάφορα πρότυπα εργαλεία λογισμικού για την διαχείριση της μεταβίβασης μηνυμάτων όπως το PVM και MPI. Στο βιβλίο αυτό θα επικεντρωθούμε στο λογισμικό MPJ για γλώσσα προγραμματισμού Java. Το MPJ είναι μια βιβλιοθήκη συναρτήσεων που μπορεί να χρησιμοποιηθεί για τη συγγραφή προγραμμάτων μεταβίβασης μηνυμάτων σε γλώσσα προγραμματισμού Java για μια ποικιλία παράλληλων υπολογιστικών συστη-

Πίνακας 9.1: Ένα σύνολο από συναρτήσεις MPJ

Συναρτήσεις MPJ	Σημασία
`MPI.Init()`	Εκκίνηση του MPJ
`MPI.Finalize()`	Τερματισμό του MPJ
`MPI.COMM_WORLD.Size()`	Προσδιορίζει τον αριθμό των διεργασιών
`MPI.COMM_WORLD.Rank()`	Προσδιορίζει την σειρά της καλούσας διεργασίας
`MPI.COMM_WORLD.Send()`	Στέλνει ένα μήνυμα
`MPI.COMM_WORLD.Recv()`	Λαμβάνει ένα μήνυμα

μάτων. Η βιβλιοθήκη MPJ αποτελείται πάνω από 125 συναρτήσεις αλλά ο αριθμός των βασικών εννοιών είναι πολύ μικρότερος. Οι συναρτήσεις αυτές διαχειρίζονται την επικοινωνία μεταξύ των υπολογιστών ή διεργασιών. Μπορούν να αναπτυχθούν πλήρη προγράμματα μεταβίβασης μηνυμάτων χρησιμοποιώντας μόνο έξι βασικές συναρτήσεις όπως φαίνεται στον Πίνακα 9.1. Το MPJ εξασφαλίζει μεταφερσιμότητα του πηγαίου κώδικα και επιτρέπει αποδοτική υλοποίηση σε διαφορετικές παράλληλες αρχιτεκτονικές.

Στις επόμενες ενότητες περιγράφουμε τις παραπάνω συναρτήσεις και τις βασικές έννοιες που είναι απαραίτητες για την συγγραφή σωστών και αποτελεσματικών προγραμμάτων μεταβίβασης μηνυμάτων.

9.2.2 Μοντέλο Διεργασιών

Η βασική μονάδα υπολογισμού στο MPJ είναι η διεργασία. Η δημιουργία και εκκίνηση των διεργασιών δεν ορίζεται στο πρότυπο MPJ αλλά εξαρτάται από την υλοποίηση. Το MPJ υποστηρίζει την στατική δημιουργία διεργασιών και ότι ένα σταθερό αριθμό διεργασιών ορίζεται από τον προγραμματιστή πριν την εκτέλεση του προγράμματος μέχρι την ολοκλήρωση. Επίσης στο MPJ υποστηρίζει το μοντέλο SPMD όπου υπάρχει ένα πρόγραμμα το οποίο εκτελείται από όλους τους υπολογιστές. Όμως το πρότυπο MPJ δεν προσφέρει μηχανισμούς για την φόρτωση εκτελέσιμων αρχείων στους υπολογιστές.

9.2.3 Εκκίνηση και Τερματισμός της βιβλιοθήκης MPJ

Κάθε πρόγραμμα MPJ πρέπει να περιέχει το πακέτο `mpi`. Το πακέτο `mpi` περιέχει ορισμούς και δηλώσεις που είναι απαραίτητες για την μεταγλώττιση ενός προγράμματος MPJ.

Το MPJ χρησιμοποιεί σταθερό σχήμα για την ονομασία των συναρτήσεων και σταθερών του MPJ. Όλα τα ονόματα των συναρτήσεων και σταθερών του MPJ αρχίζουν με το πρόθεμα `"MPI."`. Ο επόμενος πρώτος χαρακτήρας των ονομάτων συναρτήσεων είναι με κεφαλαίο γράμμα και οι υπόλοιποι χαρακτήρες με μικρά. Τα ονόματα των σταθερών του MPJ γράφονται με κεφαλαία γράμματα.

Η συνάρτηση `MPI.Init()` καλείται πριν οποιαδήποτε κλήση συναρτήσεων MPJ. Ο σκοπός της συνάρτησης αυτής είναι να εκκινήσει το περιβάλλον MPJ και να δημιουργήσει δομές δεδομένων, οι οποίες είναι απαραίτητες για την επικοινωνία των διεργασιών. Η κλήση της συνάρτησης `MPI.Init()` περισσότερο από μια φορά κατά την διάρκεια εκτέλεσης ενός προγραμμάτος προκαλεί σφάλμα. Η συνάρτηση `MPI.Finalize()` καλείται στο τέλος του υπολογισμού και εκτελεί διάφορες εργασίες εκκαθάρισης για να τερματίσει το περιβάλλον MPJ, δηλαδή καταστρέφει τις δομές που είχαν δημιουργηθεί από την κλήση της συνάρτησης `MPI.Init()`. Δεν εκτελούνται κλήσεις MPJ πριν την κλήση της συνάρτησης `MPI.Init()` και μετά την κλήση της συνάρτησης `MPI.Finalize()`. Οι συναρτήσεις `MPI.Init()` και `MPI.Finalize()` πρέπει να καλούνται από όλες τις διεργασίες διαφορετικά η συμπεριφορά του MPJ θα είναι απροσδιόριστη. Η σύνταξη των δύο παραπάνω συναρτήσεων για την Java είναι ως εξής:

```
int MPI.Init(String[] args)

int MPI.Finalize()
```

Το όρισμα `args` της συνάρτησης `MPI.Init()` είναι ορίσματα γραμμής - εντολών του προγράμματος Java. Η επιτυχής εκτέλεση των συναρτήσεων `MPI.Init()` και `MPI.Finalize()` επιστρέφει την σταθερά `MPI.SUCCESS` διαφορετικά επιστρέφει ένα προκαθορισμένο αριθμό σφάλματος. Συνεπώς, ένα πρόγραμμα MPJ έχει την ακόλουθη τυπική δομή:

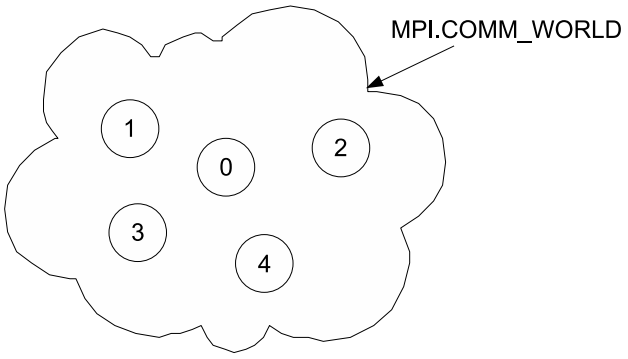
```
import mpi.*;

class example {

   static public void main(String[] args){

     MPI.Init(args);

        ...
```

```
        MPI.Finalize();
      }
    }
```

9.2.4 Κανάλια Επικοινωνίας

Μια **ομάδα διεργασιών** (processes group) είναι μια συλλογή από διεργασίες. Παρόλο που το μοντέλο διεργασιών του MPJ είναι στατικό, οι ομάδες διεργασιών είναι δυναμικές με την έννοια ότι μπορούν να δημιουργηθούν και να καταστραφούν. Επίσης, κάθε διεργασία μπορεί να ανήκει σε περισσότερες από μια ομάδες ταυτόχρονα. Συνεπώς, οι διεργασίες έχουν δύο βασικά χαρακτηριστικά: την ομάδα και την σειρά μέσα στην ομάδα. Έτσι, για μια ομάδα n διεργασιών, η κάθε διεργασία έχει μια σειρά που είναι ένας ακέραιος αριθμός από 0 μέχρι $n-1$. Η ομάδα διεργασιών συνδυάζεται σε ένα αντικείμενο που λέγεται **κανάλι επικοινωνίας** (communicator).

Τα κανάλια επικοινωνίας χρησιμοποιούνται ως παράμετροι στο MPJ για επικοινωνίες σημειακές και συλλογικές. Ένα κανάλι επικοινωνίας χρησιμοποιείται για να ορίζει ένα σύνολο από διεργασίες που μπορούν να επικοινωνούν μεταξύ τους. Το σύνολο των διεργασιών σχηματίζει μια **περιοχή επικοινωνίας** (communication domain). Πληροφορίες σχετικά με τις περιοχές επικοινωνίας αποθηκεύονται στις μεταβλητές του τύπου `MPI.Comm`. Υπάρχουν δύο τύποι καναλιών επικοινωνίας, ένα **κανάλι ενδοεπικοινωνίας** (intracommunicator) το οποίο χρησιμοποιείται για επικοινωνία μέσα σε μια ομάδα διεργασιών και το **κανάλι διαεπικοινωνίας** (intercommunicator) το οποίο χρησιμοποιείται για επικοινωνία ανάμεσα στις ομάδες διεργασιών. Για απλά προγράμματα, χρησιμοποιούνται πιο πολύ συχνά τα κανάλια ενδοεπικοινωνίας. Έτσι, στο MPJ υπάρχει ένα προκαθορισμένο κανάλι ενδοεπικοινωνίας που ονομάζεται `MPI.COMM_WORLD` το οποίο περιλαμβάνει όλες τις διεργασίες που υπάρχουν σε μια παράλληλη εφαρμογή όπως φαίνεται στο Σχήμα 9.9. Όμως, υπάρχουν περιπτώσεις που θέλουμε να πραγματοποιήσουμε επικοινωνία μόνο σε μια ορισμένη ομάδα διεργασιών, δηλαδή να ορίσουμε δικά μας κανάλια επικοινωνίας. Η δημιουργία καινούργιων καναλιών επικοινωνίας βασίζονται στα υπάρχοντα κανάλια επικοινωνίας. Προς το παρόν, παρακάτω χρησιμοποιούμε το `MPI.COMM_WORLD` ως όρισμα κανάλι επικοινωνίας σε όλες τις συναρτήσεις MPJ που απαιτούν ένα κανάλι επικοινωνίας.



Σχήμα 9.9: Κανάλι επικοινωνίας

9.2.5 Πλήθος Διεργασιών και Σειρά Διεργασίας

Οι συναρτήσεις `MPI.COMM_WORLD.Size()` και `MPI.COMM_WORLD.Rank()` χρησιμοποιούνται για να προσδιορίζουν το πλήθος των διεργασιών και την σειρά της καλούσας διεργασίας αντίστοιχα. Η σύνταξη των δύο παραπάνω συναρτήσεων είναι ως εξής:

```
int MPI.COMM_WORLD.Size()

int MPI.COMM_WORLD.Rank()
```

Η συνάρτηση `MPI.COMM_WORLD.Size()` επιστρέφει το πλήθος των διεργασιών που ανήκουν στο κανάλι επικοινωνίας `MPI.COMM_WORLD`. Κάθε διεργασία που ανήκει σε ένα κανάλι επικοινωνίας αναγνωρίζεται μοναδικά από την σειρά της `rank`. Η σειρά μιας διεργασίας είναι ένας ακέραιος αριθμός που κυμαίνεται από 0 μέχρι $n-1$, όταν υπάρχουν n διεργασίες. Μια διεργασία μπορεί να προσδιορίζει την σειρά της σε ένα κανάλι επικοινωνίας από την κλήση της συνάρτησης `MPI.COMM_WORLD.Rank()` που επιστρέφει τη σειρά της διεργασίας.

9.2.6 Όνομα Υπολογιστή

Η συνάρτηση `MPI.Get_processor_name()` χρησιμοποιείται για να προσδιορίζουμε το όνομα του υπολογιστή που εκτελείται κάθε διεργασία. Στις περισσότερες παράλληλες εφαρμογές μας ενδιαφέρει η σειρά μιας διεργασίας παρά σε ποιον υπολογιστή εκτελείται η διεργασία. Όμως, υπάρχουν εφαρμογές που μας ενδιαφέρει το όνομα του υπολογιστή. Η σύνταξη της παραπάνω συνάρτησης είναι ως εξής:

```
String MPI.Get_processor_name()
```

Η συνάρτηση αυτή επιστρέφει το όνομα του υπολογιστή.

9.2.7 Το Πρώτο Απλό Πρόγραμμα MPJ

Με βάση που παρουσιάσαμε προηγουμένως, δίνουμε παρακάτω ένα απλό παράδειγμα προγράμματος MPJ που κάθε διεργασία εμφανίζει ένα μήνυμα της μορφής ''`Hello from process i (or name) of n`'' όπου `i` είναι η σειρά διεργασίας, `name` είναι το όνομα υπολογιστή που εκτελείται η διεργασία και `n` είναι το πλήθος διεργασιών.

```
import mpi.*;
public class HelloWorld
{
    public static void main(String args[])
    {
        MPI.Init(args);
        int rank = MPI.COMM_WORLD.Rank();
        String name = MPI.Get_processor_name();
        int size = MPI.COMM_WORLD.Size();
        System.out.println("Hello␣from␣process␣" + rank + "␣(or␣" + name +")␣of␣" + size);
        MPI.Finalize();
    }
}
```

Το παραπάνω πρόγραμμα πρέπει να μεταγλωττιστεί με την βοήθεια του μεταλωττιστή Java και στην συνέχεια να συνδεθεί με την βιβλιοθήκη του MPJ (π.χ. με τον μεταγλωττιστή `javac`). Ο τρόπος που τα προγράμματα MPJ ξεκινάνε σε μια συστοιχία υπολογιστών δεν είναι μέρος του MPJ αλλά εξαρτάται από την υλοποίηση. Αφού μεταγλωττιστεί το πρόγραμμα προκύπτει ένα εκτελέσιμο αρχείο που πρέπει να εκτελεστεί στους αντίστοιχους υπολογιστές της συστοιχίας με την βοήθεια της γραμμής εντολών. Έτσι, το ίδιο εκτελέσιμο αρχείο (ή πρόγραμμα) αντιγράφεται σε κάθε υπολογιστή της συστοιχίας από την υλοποίηση του MPJ και κάθε υπολογιστής αρχίζει να εκτελεί το δικό του αντίγραφο του προγράμματος. Για παράδειγμα, το πρόγραμμα μπορεί να εκτελεστεί σε τρεις διαφορετικούς υπολογιστές της συστοιχίας ταυτόχρονα με την εντολή:

```
mpjrun.sh -np 3 HelloWorld
```

Με αυτό τον τρόπο δημιουργούνται τρεις διεργασίες που κάθε διεργασία εκτελείται σε διαφορετικό υπολογιστή της συστοιχίας. Η έξοδος της παραπάνω εντολής είναι η εξής:

```
Hello from process 1 (or localhost.localdomain) of 3
Hello from process 0 (or localhost.localdomain) of 3
Hello from process 3 (or localhost.localdomain) of 3
```

Το παραπάνω πρόγραμμα που είδαμε χρησιμοποιεί το μοντέλο SPMD αφού κάθε διεργασία εκτελεί τον ίδιο πρόγραμμα. Όμως στις περισσότερες εφαρμογές υπάρχει ανάγκη μια ή δύο διεργασίες να εκτελούν διαφορετικό κώδικα μέσα σε ένα πρόγραμμα. Αυτό υλοποιείται ενσωματώνοντας εντολές επιλογής σε ένα πρόγραμμα και με βάση την σειρά της διεργασίας να επιλέγουν τα τμήματα του κώδικα που θα εκτελούνται από κάθε διεργασία. Με άλλα λόγια, οι εντολές που εκτελούνται από την διεργασία με σειρά 0 είναι διαφορετικές από τις εντολές που εκτελούνται από τις υπόλοιπες διεργασίες παρόλο που όλες οι διεργασίες εκτελούν το ίδιο πρόγραμμα. Παρακάτω παρουσιάζεται ένα κομμάτι προγράμματος για το μοντέλο SPMD.

```
import mpi.*;

class example {

   static public void main(String[] args){

     MPI.Init(args);

     int rank = MPI.COMM_WORLD.Rank();

     if (rank == 0)

       master();

     else

       worker();

     MPI.Finalize();

   }
}
```

όπου `master()` και `worker()` είναι μέθοδοι (ή διαδικασίες) που εκτελούνται από την διεργασία συντονιστής και την διεργασία εργαζόμενος αντίστοιχα. Η παραπάνω προσέγγιση θα μπορούσε να χρησιμοποιηθεί για περισσότερα από δύο διαφορετικά τμήματα κώδικα. Παρακάτω παρουσιάζεται ένα παράδειγμα προγράμματος SPMD ώστε η διεργασία με σειρά 0 να εμφανίζει ένα μήνυμα ''`I am master process with rank 0`'' και οι υπόλοιπες διεργασίες να εμφανίζουν ένα μήνυμα ''`I am worker process with rank i`'' όπου `i` είναι η σειρά της διεργασίας.

```
import mpi.*;
public class HelloWorld1
{
    public static void main(String args[])
    {
        MPI.Init(args);
        int rank = MPI.COMM_WORLD.Rank();
```

```
        int size = MPI.COMM_WORLD.Size();
        if (rank == 0)
                System.out.println("I␣am␣master␣process␣with␣rank␣" + rank);
        else
                System.out.println("I␣am␣worker␣process␣with␣rank␣" + rank);
        MPI.Finalize();
    }
}
```

9.2.8 Μηνύματα

Το βασικό κομμάτι της επικοινωνίας του MPJ είναι η μεταφορά δεδομένων ή μηνυμάτων μεταξύ δύο διεργασιών. Έτσι, κάθε μήνυμα MPJ αποτελείται από δυο μέρη: από τον **φάκελο** (envelope) και από το **κυρίως μήνυμα** (message body). Ο φάκελος ενός μηνύματος περιέχει τα εξής στοιχεία:

- Η σειρά της διεργασίας αποστολέας.
- Η σειρά της διεργασίας παραλήπτης.
- Το κανάλι επικοινωνίας στον οποίο ανήκουν οι διεργασίες αποστολέας και παραλήπτης.
- Η ετικέτα ώστε η διεργασία παραλήπτης να διακρίνει τον τύπο του μηνύματος.

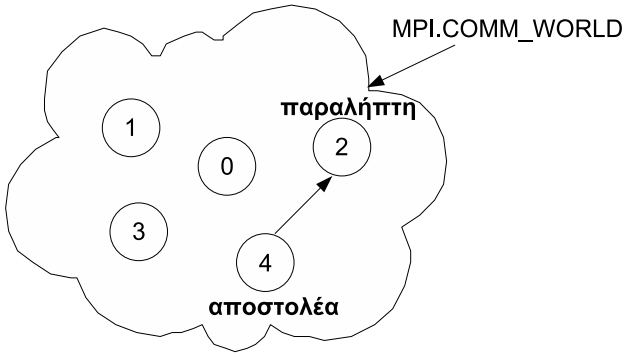
Το κυρίως μήνυμα αποτελείται από τα παρακάτω στοιχεία:

- Τη διεύθυνση αρχής του ενταμιευτή όπου μπορούν να αποθηκευτούν τα δεδομένα που αποστέλονται ή παραλαμβάνονται.
- Τον τύπο δεδομένων του μηνύματος.
- Τον αριθμό των στοιχείων του τύπου δεδομένων, ο οποίος προσδιορίστηκε προηγουμένως που πρόκειται να αποσταλούν ή να παραλαβούν.

9.2.9 Σημειακή Επικοινωνία

Η σημειακή επικοινωνία περιλαμβάνει πάντα δύο διεργασίες. Η μια διεργασία στέλνει ένα μήνυμα στην άλλη διεργασία. Η διεργασία αποστολέα για να στείλει ένα μήνυμα πρέπει να καλέσει μια

συνάρτηση αποστολής MPJ που να προσδιορίζει την διεργασία παραπλήπτη με βάση την σειρά της στο κανάλι επικοινωνίας (π.χ. το MPI.COMM_WORLD). Η διεργασία παραλήπτη για να παραλάβει το μήνυμα πρέπει να καλέσει την αντίστοιχη συνάρτηση παραλαβής MPJ. Και οι δύο διεργασίες αποστολέα και παραλήπτη πρέπει να προσδιορίζουν στο ίδιο κανάλι επικοινωνίας το οποίο πρέπει να ανήκουν όπως φαίνεται στο Σχήμα 9.10.



Σχήμα 9.10: Σημειακή επικοινωνία

Γνωρίζουμε ότι υπάρχουν διάφορες εκδόσεις για την αποστολή και λήψη μηνυμάτων. Έτσι, παρακάτω περιγράφουμε τις ανασταλτικές και μη ανασταλτικές συναρτήσεις για αποστολή και λήψη μηνυμάτων.

Ανασταλτικές Συναρτήσεις

Στο MPJ οι ανασταλτικές συναρτήσεις για αποστολή ή λήψη μηνημάτων επιστρέφουν τον έλεγχο τους όταν η τοπική λειτουργία έχει ολοκληρωθεί, δηλαδή στην περίπτωση αποστολή έχει αντιγραφεί το μήνυμα στο ενταμιευτή μηνυμάτων και στην περίπτωση λήψη έχει παραληφθεί το μήνυμα από το ενταμιευτή. Οι ανασταλτικές συναρτήσεις για αποστολή και λήψη μηνυμάτων στο MPJ είναι οι `Comm.Send()` και `Comm.Recv()`, αντίστοιχα. Η ανασταλτική συνάρτηση `Comm.Send()` επιστρέφει όταν το μήνυμα έχει αντιγραφεί στον ενταμιευτή. Η σύνταξη της συνάρτησης `Comm.Send()` έχει ως εξής:

```
void Comm.Send(Object buf, int offset, int count, Datatype datatype, int dest, int tag)
```

όπου

- `Comm` είναι το κανάλι επικοινωνίας που διαμοιράζεται από τις διεργασίες αποστολέα και

παραλήπτη.

- `buf` είναι αντικείμενο του ενταμιευτή που περιέχει δεδομένα που πρόκειται να αποσταλούν.
- `offset` είναι η αρχική θέση μέσα στο ενταμιευτή `buf`.
- `count` είναι ο αριθμός των στοιχείων του τύπου δεδομένων που ορίζεται από την παράμετρο `datatype` που περιέχονται στον ενταμιευτή `buf`.
- `datatype` είναι ο τύπος δεδομένων του MPJ των στοιχείων του `buf`. Στον Πίνακα 9.2 παρουσιάζεται η αντιστοίχηση ανάμεσα στους τύπους δεδομένων που υποστηρίζει το MPJ και τους τύπους δεδομένων που υποστηρίζει η γλώσσα προγραμματισμού Java. Σημειώνουμε ότι στο MPJ υπάρχουν δύο τύποι δεδομένων που δεν υποστηρίζει Java. Οι τύποι αυτοί είναι οι `MPI.BYTE` και `MPI.PACKED`. Ο τύπος `MPI.BYTE` αντιστοιχεί σε ένα byte (8 bits). Ο τύπος `MPI.PACKED` αντιστοιχεί σε μια συλλογή δεδομένων που έχει δημιουργηθεί με συννένωση.
- `dest` είναι η διεργασία παραλήπτη που θα παραλάβει το μήνυμα. Η παράμετρος αυτή είναι η σειρά της διεργασίας παραλήπτη στην περιοχή επικοινωνίας που ορίζεται από το κανάλι επικοινωνίας `Comm`.
- `tag` είναι μια ετικέτα που έχει μια ακέραια τιμή και χρησιμοποιείται για να διακρίνει τους διαφορετικούς τύπους μηνυμάτων. Η ετικέτα `tag` παίρνει τιμές από 0 μέχρι `MPI.TAG_UB` που είναι μια προκαθορισμένη σταθερά του MPJ. Η τιμή της σταθεράς `MPI.TAG_UB` είναι τουλάχιστον 32,767.

Η ανασταλτική συνάρτηση `Comm.Recv()` επιστρέφει όταν έχει παραλάβει το μήνυμα από το ενταμιευτή. Η σύνταξη της συνάρτησης `Comm.Recv()` έχει ως εξής:

```
Status Comm.Recv(Object buf, int offset, int count, Datatype datatype, int source, int tag)
```

όπου

- `Comm` είναι το κανάλι επικοινωνίας που διαμοιράζεται από τις διεργασίες αποστολέα και παραλήπτη.
- `buf` είναι αντικείμενο του ενταμιευτή που περιέχει δεδομένα μετά την λήψη του μηνύματος.

Πίνακας 9.2: Αντιστοίχηση ανάμεσα στους τύπους δεδομένων που υποστηρίζει το MPJ και τους τύπους δεδομένων που υποστηρίζει η Java

Τύπος δεδομένων του MPJ	Τύπος δεδομένων της Java
MPI.CHAR	char
MPI.SHORT	short
MPI.BOOLEAN	boolean
MPI.INT	int
MPI.LONG	long
MPI.FLOAT	float
MPI.DOUBLE	double
MPI.BYTE	byte
MPI.OBJECT	Object

- offset είναι η αρχική θέση μέσα στο ενταμιευτή buf.

- count είναι ο μέγιστος αριθμός των στοιχείων του τύπου δεδομένων που ορίζεται από την παράμετρο datatype που μπορεί να περιέχονται στον ενταμιευτή buf. Ο αριθμός των δεδομένων που πραγματικά λαμβάνει πρέπει να είναι λιγότερο από τον αριθμό count. Άν όμως ο αριθμός των δεδομένων που λαμβάνει είναι μεγαλύτερος από το μέγεθος του ενταμιευτή (count) τότε εμφανίζεται σφάλμα υπερχείλισης και η συνάρτηση επιστρέφει το σφάλμα MPI.ERR_TRUNCATE.

- datatype είναι ο τύπος δεδομένων του MPJ των στοιχείων του buf. Ο τύπος αυτός πρέπει να ταυτίζεται με τον τύπο δεδομένων που ορίζεται στην συνάρτηση Comm.Send.

- source είναι η σειρά της διεργασίας αποστολέα του μηνύματος στην περιοχή επικοινωνίας που ορίζεται από το κανάλι επικοινωνίας Comm. Στην περίπτωση που η παράμετρος source έχει οριστεί με τιμή MPI.ANY_SOURCE τότε λαμβάνει μηνύματα από οποιοδήποτε αποστολέα.

- tag είναι μια ετικέτα που δηλώνει τι είδους μηνύματα μπορεί να λάβει η διεργασία. Αν η ετικέτα αυτή έχει τιμή MPI.ANY_TAG τότε λαμβάνει μηνύματα με οποιαδήποτε τιμή ετικέτας.

Τέλος, η συνάρτηση επιστρέφει ένα αντικείμενο τύπου `Status` που περιέχει πληροφορίες σχετικά με την λήψη του μηνύματος.

Παρακάτω παρουσιάζεται ένα πρόγραμμα MPJ που η διεργασία 0 στέλνει ένα μήνυμα χαιρετισμού ''`Hello World`'' στην διεργασία 1 και επίσης οι δύο διεργασίες εμφανίζουν στην οθόνη ένα διαγνωστικό μήνυμα.

```
import mpi.*;
public class HelloWorldSR1
{
    public static void main(String args[])
    {
        char[] message = "Hello␣World".toCharArray();
        MPI.Init(args);
        int rank = MPI.COMM_WORLD.Rank();
        if (rank == 0) {
                System.out.println("Sending␣message␣to␣proc␣#1␣" + message);
                MPI.COMM_WORLD.Send(message,0,message.length,MPI.CHAR,1,99);
        }
        else if (rank == 1) {
                MPI.COMM_WORLD.Recv(message,0,20,MPI.CHAR,0,99);
                System.out.println("Received␣message␣from␣proc␣#0␣" + message);
        }
        MPI.Finalize();
    }
}
```

Στο προηγούμενο παράδειγμα που είδαμε η επικοινωνία ήταν μονόπλευρη δηλαδή ένα μήνυμα στάλθηκε από την διεργασία αποστολέα προς την διεργασία παραλήπτη. Στο επόμενο παράδειγμα προγράμματος MPJ παρουσιάζεται μια αμφίδρομη επικοινωνία, δηλαδή η διεργασία 0 στέλνει ένα μήνυμα χαιρετισμού ''`Hello World`'' στην διεργασία 1 και στην συνέχεια η διεργασία 1 στέλνει πίσω ένα μήνυμα χαιρετισμού ''`Good Bye`'' στην διεργασία 0.

```
import mpi.*;
public class HelloWorldSR2
{
    public static void main(String args[])
    {
        char[] message1 = "Hello␣World".toCharArray();
        char[] message2 = "Good␣Bye".toCharArray();
        char[] reply = new char[20];
        MPI.Init(args);
        int rank = MPI.COMM_WORLD.Rank();
        if (rank == 0) {
                System.out.println("Sending␣message␣to␣proc␣#1␣" + message1);
```

```
                MPI.COMM_WORLD.Send(message1,0,message1.length,MPI.CHAR,1,99);
                MPI.COMM_WORLD.Recv(reply,0,20,MPI.CHAR,1,99);
                System.out.println("Received␣message␣from␣proc␣#1␣" + reply);
        }
        else if (rank == 1) {
                MPI.COMM_WORLD.Recv(reply,0,20,MPI.CHAR,0,99);
                System.out.println("Received␣message␣from␣proc␣#0␣" + reply);
                System.out.println("Sending␣message␣to␣proc␣#0:␣" + message2);
                MPI.COMM_WORLD.Send(message2,0,message2.length,MPI.CHAR,0,99);
        }
        MPI.Finalize();
    }
}
```

Τέλος, στο επόμενο πρόγραμμα παρουσιάζεται μια επέκταση του προηγούμενου προγράμματος έτσι ώστε οι δύο διεργασίες να ανταλλάσουν επαναληπτικά μηνύματα χαιρετισμού 10 φορές.

```
import mpi.*;
public class HelloWorldSR3
{
    public static void main(String args[])
    {
        char[] message1 = "Hello␣World".toCharArray();
        char[] message2 = "Good␣Bye".toCharArray();
        char[] reply = new char[20];
        MPI.Init(args);
        int rank = MPI.COMM_WORLD.Rank();
        if (rank == 0) {
                for (int i = 0; i < 10; i++) {
                      System.out.println("Sending␣message␣to␣proc␣#1␣" + message1);
                      MPI.COMM_WORLD.Send(message1,0,message1.length,MPI.CHAR,1,99);
                      MPI.COMM_WORLD.Recv(reply,0,20,MPI.CHAR,1,99);
                      System.out.println("Received␣message␣from␣proc␣#1␣" + reply);
                }
        }
        else if (rank == 1) {
                for(int i = 0; i < 10; i++) {
                     MPI.COMM_WORLD.Recv(reply,0,20,MPI.CHAR,0,99);
                     System.out.println("Received␣message␣from␣proc␣#0␣" + reply);
                     System.out.println("Sending␣message␣to␣proc␣#0:␣" + message2);
                     MPI.COMM_WORLD.Send(message2,0,message2.length,MPI.CHAR,0,99);
                }
        }
        MPI.Finalize();
    }
```

```
}
```

Η χρήση των ανασταλτικών λειτουργιών επικοινωνίας `Comm.Send()` και `Comm.Recv()` πρέπει να είναι ιδιαίτερα προσεκτική και κάθε κλήση ανασταλτικής λειτουργίας `Comm.Send()` πρέπει να συνοδεύεται από την αντίστοιχη ανασταλτικής κλήσης `Comm.Recv()` διαφορετικά μπορεί να προκαλέσει **αδιέξοδο** (deadlock) το εμποδίζει τη λειτουργία του προγράμματος. Αδιέξοδο συμβαίνει στην περίπτωση στην οποία δύο ή περισσότερες ενέργειες περιμένουν τις υπόλοιπες για να τερματίσουν και έτσι καμμία από αυτές δεν το καταφέρνει. Για την καλύτερη κατανόηση του προβλήματος αδιεξόδου θεωρούμε το παρακάτω παράδειγμα κώδικα:

```
int rank = MPI.COMM_WORLD.Rank();
if (rank == 0) {
    MPI.COMM_WORLD.Recv(reply, 0, 20, MPI.CHAR, 1, 99);
    MPI.COMM_WORLD.Send( message1, 0, message1.length, MPI.CHAR, 1, 99);
}
else if (rank == 1) {
    MPI.COMM_WORLD.Recv(reply, 0, 20, MPI.CHAR, 0, 99);
    MPI.COMM_WORLD.Send(message2, 0, message2.length, MPI.CHAR, 0, 99);
}
```

Η συνάρτηση λήψης της πρώτης διεργασίας πρέπει να ολοκληρωθεί για να είναι δυνατή η κλήση της συνάρτησης αποστολής της. Για να γίνει αυτό πρέπει η δεύτερη διεργασία να καλέσει τη αντίστοιχη δική της συνάρτηση αποστολής ώστε να παραλάβει το μήνυμα. Παρόμοια, η συνάρτηση λήψης της δεύτερης διεργασίας πρέπει να ολοκληρωθεί πριν τη συνάρτηση αποστολής της και για να γίνει αυτό θα πρέπει η πρώτη διεργασία να εκτελέσει μια αντίστοιχη συνάρτηση αποστολής. Έτσι, το πρόγραμμα εκτελείται αρτέμονα και εμφανίζεται αδιέξοδο.

Μη Ανασταλτικές Συναρτήσεις

Για να λυθεί το παραπάνω πρόβλημα του αδιεξόδου μπορούν να χρησιμοποιηθούν οι μη ανασταλτικές συναρτήσεις αποστολής και λήψης. Μια μη ανασταλτική συνάρτηση επιστρέφει αμέσως και συνεχίζει να εκτελεί την επόμενη εντολή του προγράμματος. Σε κάποια μεταγενέστερη χρονική στιγμή η συνάρτηση ελέγχει για την ολοκλήρωση της επικοινωνίας. Οι μη ανασταλτικές συναρτήσεις για αποστολή και λήψη μηνυμάτων στο MPJ είναι οι `Comm.Isend()` και `Comm.Irecv()`, αντίστοιχα. Η συνάρτηση `Comm.Isend()` επιστρέφει ανεξάρτητα με την ολοκλήρωση της επικοινωνίας. Ενώ η συνάρτηση `Comm.Irecv()` επιστρέφει ακόμα και αν δεν υπάρχει μήνυμα να παραλάβει. Η σύνταξη των δύο παραπάνω συναρτήσεων είναι ως εξής:

```
Request Comm.Isend(Object buf, int offset, int count, Datatype datatype, int dest, int tag)

Request Comm.Irecv(Object buf, int offset, int count, Datatype datatype, int source, int tag)
```

Παρατηρούμε από τις παραπάνω συναρτήσεις ότι έχουν παρόμοια ορίσματα με τις ανασταλτικές συναρτήσεις. Όμως, οι δυο αυτές συναρτήσεις επιστρέφουν το `request` που χρησιμοποιείται για τον έλεγχο ολοκλήρωσης της επικοινωνίας. Οι δύο παραπάνω συναρτήσεις μπορούν να ελέγξουν για το αν ολοκληρώθηκε η επικοινωνία με την χρήση των συναρτήσεων `Request.Wait()` και `Request.Test()`. Η συνάρτηση `Request.Wait()` περιμένει μέχρι να ολοκληρωθεί η επικοινωνία και μετά επιστρέφει. Η συνάρτηση `Request.Test()` εξετάζει αν η επικοινωνία εκείνη τη στιγμή που καλείται η συνάρτηση έχει ολοκληρωθεί και επιστρέφει αμέσως με μια λογική απάντηση `TRUE` ή `FALSE`. Οι παραπάνω συναρτήσεις χρησιμοποιούνται για την επικάλυψη μεταξύ επικοινωνίας και υπολογισμού στις περιπτώσεις που το κόστος της επικοινωνίας είναι υψηλό.

Παρακάτω παρουσιάζεται ένα πρόγραμμα MPJ ασύγχρονης ανταλλαγής μηνυμάτων.

```
import mpi.*;
class HelloWorldISR {
    static public void main(String[] args) throws MPIException {
        MPI.Init(args) ;
        int rank = MPI.COMM_WORLD.Rank();
        if (rank == 0) {
            char[] message = "Hello␣World".toCharArray();
            Request req = MPI.COMM_WORLD.Isend(message, 0, message.length, MPI.CHAR, 1, 99);
            Status st = req.Wait();
        }
        else if (rank == 1) {
            char[] message = new char[20];
            MPI.COMM_WORLD.Recv(message, 0, 20, MPI.CHAR, 0, 99);
            System.out.println("Process␣" + rank + "␣received␣the␣message:" + new String(message));
        }
        MPI.Finalize() ;
    }
}
```

Η διεργασία 0 στέλνει ένα μήνυμα χαιρετισμού στην διεργασία 1 και η διεργασία 0 μπορεί να συνεχίζει την εκτέλεση της.

9.2.10 Καταστάσεις Επικοινωνίας Αποστολής

Οι συναρτήσεις αποστολής του MPJ μπορούν να έχουν μια από τις τέσσερις **καταστάσεις επικοινωνίας** (communication modes) που ορίζει το πρωτόκολλο αποστολής και λήψης. Οι

καταστάσεις είναι ο **σύχρονος** (synchronous), ο **ενδιάμεσος ενταμιευτής** (buffered), ο **κανονικός** (standard) και η **ετοιμότητα** (ready).

Η κατάσταση σύγχρονος είναι όπου ο αποστολέας δεν μπορεί να αρχίσει να στέλνει πριν καλέσει ο παραλήπτης την συνάρτηση `MPI.Recv()`. Η κατάσταση ενδιάμεσος ενταμιευτής είναι ότι το προς αποστολή μήνυμα πρέπει να αποθηκευτεί στον ενδιάμεσο ενταμιευτή (ο οποίος παρέχεται από την εφαρμογή). Θα πρέπει ο χώρος του ενδιάμεσου ενταμιευτή να είναι αρκετά μεγάλος. Η κατάσταση κανονικός η οποία εξαρτάται από την υλοποίηση και μπορεί να είναι είτε σύγχρονος είτε με ενδιάμεσο ενταμιευτή. Τέλος, η κατάσταση ετοιμότητα είναι ότι ο αποστολέας ισχυρίζεται ότι ο παραλήπτης είναι έτοιμος αλλά δεν γίνεται κανένας έλεγχος.

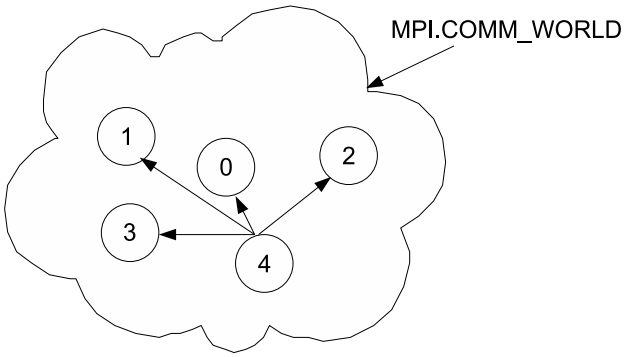
Κάθε μια από τις τέσσερις καταστάσεις επικοινωνίας υπάρχει σε δύο εκδόσεις, ανασταλτική και μη ανασταλτική και έτσι έχουμε συνολικά οκτώ συναρτήσεις. Οι τρεις μη κανονικές καταστάσεις αναγνωρίζονται από ένα γράμμα, δηλαδή το γράμμα `b` για buffered, το `s` για synchronous και το `r` για ready. Για παράδειγμα, η `MPI.Issend()` είναι μια μη ανασταλτική σύγχρονη συνάρτηση αποστολής. Η συνάρτηση λήψη είναι διαθέσιμη μόνο σε κατάσταση κανονικός με δύο εκδόσεις, ανασταλτική και μη ανασταλτική.

9.2.11 Συλλογική Επικοινωνία

Η συλλογική επικοινωνία περιλαμβάνει ένα σύνολο από διεργασίες σε σχέση με την σημειακή επικοινωνία που περιλαμβάνει μόνο δύο διεργασίες. Το σύνολο των διεργασιών που ορίζει το κανάλι επικοινωνίας συμμετέχουν στην συλλογική επικοινωνία. Ένα παράδειγμα είναι ίσως μια απλή συνάρτηση εκπομπή όπου μια διεργασία στέλνει τα ίδια δεδομένα σε όλες τις διεργασίες στην ομάδα ή κανάλι επικοινωνίας όπως φαίνεται στο Σχήμα 9.11. Η συλλογική επικοινωνία μπορεί να υλοποιηθεί από τον προγραμματιστή χρησιμοποιώντας τις συναρτήσεις `Comm.Send()` και `Comm.Recv()`. Όμως, η βιβλιοθήκη MPJ παρέχει ένα σύνολο από ειδικές συναρτήσεις συλλογικής επικοινωνίας. Οι συναρτήσεις αυτές είναι οι παρακάτω:

Φράγμα (Barrier)

Η συνάρτηση για το φράγμα είναι η `Intracomm.Barrier()`. Η συνάρτηση αυτή συγχρονίζει την καλούσα διεργασία με όλες τις διεργασίες που πρέπει να καλέσουν την συνάρτηση αυτή. Με άλλα λόγια συγχρονίζει όλες τις διεργασίες. Η συνάρτηση αυτή επιστρέφει τον έλεγχο της,



Σχήμα 9.11: Συλλογική επικοινωνία: εκπομπή

μόνο όταν την καλέσουν όλες οι διεργασίες. Η σύνταξη της είναι ως εξής:

```
void Intracomm.Barrier()
```

Εκπομπή (Brodacast)

Η συνάρτηση για την εκπομπή είναι η `Intracomm.Bcast()`. Η συνάρτηση αυτή στέλνει δεδομένα από μια διεργασία ρίζα προς όλες τις διεργασίες που μετέχουν στο κανάλι επικοινωνίας. Η σύνταξη της παραπάνω συνάρτησης έχει ως εξής:

```
void Intracomm.Bcast(Object buf, int offset, int count, Datatype datatype, int root)
```

όπου τα ορίσματα `buf`, `offset`, `count` και `datatype` είναι παρόμοια με τα ορίσματα των συναρτήσεων `Comm.Send` και `Comm.Recv`. Το όρισμα `root` είναι η σειρά της διεργασίας ρίζας και το `IntraComm` είναι το κανάλι επικοινωνίας.

Στο επόμενο πρόγραμμα MPJ φαίνεται η χρήση της συνάρτησης `Intracomm.Bcast`.

```
import mpi.*;
public class BroadData
{
    public static void main(String args[]) throws MPIException
    {
        MPI.Init(args);
        int rank = MPI.COMM_WORLD.Rank();
        int size = MPI.COMM_WORLD.Size();
        int number = 0;
        int[] num = new int[]{number};
        do {
                if (rank == 0) {
                        number++;
```

```
                num[0] = number;
            }
            MPI.COMM_WORLD.Bcast(num,0,1,MPI.INT,0);
            number = num[0];
            System.out.println("Process␣" + rank + "␣got␣" + number);
        } while (number < 5);
        MPI.Finalize();
    }
}
```

Η διεργασία 0 έχει ένα μετρητή που ξεκινάει από τιμή 1 και κάθε φορά που αυξάνεται ο μετρητής διανέμεται η τιμή του στις υπόλοιπες διεργασίες. Κάθε διεργασία εμφανίζει στην οθόνη την σειρά της και την τιμή που έλαβε. Η διαδικασία της αύξησης του μετρητή συνεχίζεται μέχρι να φτάσει την τιμή 5.

Συλλογή (Gather)

Η συνάρτηση για την συλλογή είναι η `Intracomm.Gather()`. Με τη συνάρτηση αυτή συλλέγονται δεδομένα από όλες τις διεργασίες που μετέχουν στο κανάλι επικοινωνίας στη διεργασία ρίζα. Η σύνταξη της συνάρτησης αυτής είναι ως εξής:

```
void Intracomm.Gather(Object sendbuf, int sendoffset, int sendcount, Datatype sendtype, Object recvbuf, int recvoffset, int recvcount, Datatype recvtype, int root)
```

όπου

- `sendbuf` είναι η διεύθυνση του ενταμιευτή δεδομένων της κάθε διεργασίας.
- `sendoffset` είναι η αρχική θέση μέσα στο `sendbuf`.
- `sendcount` είναι ο αριθμός των στοιχείων του τύπου δεδομένων που ορίζεται από την παράμετρο `sendtype` που περιέχονται στο `sendbuf` της κάθε διεργασίας.
- `sendtype` είναι ο τύπος δεδομένων του MPJ των στοιχείων του `sendbuf` της κάθε διεργασίας.
- `recvbuf` είναι η διεύθυνση του ενταμιευτή της διεργασίας ρίζας που θα περιέχει τα δεδομένα τα οποία συλλέγονται από όλες τις διεργασίες.
- `recvoffset` είναι η αρχική θέση μέσα στο `recvbuf`.

- `recvcount` είναι ο αριθμός των στοιχείων του τύπου δεδομένων `recvtype` τα οποία συλλέγονται από κάθε διεργασία.

- `recvtype` είναι ο τύπος δεδομένων των στοιχείων του `recvbuf` που έλαβε από κάθε διεργασία.

- `root` είναι η σειρά της διεργασίας ρίζας.

- `Intracomm` είναι το κανάλι ενδοεπικοινωνίας που περιέχει όλες τις διεργασίες που μετέχουν στην συλλογική επικοινωνία.

Παρακάτω παρουσιάζεται ένα πρόγραμμα MPJ που φαίνεται η χρήση της συνάρτησης `Intracomm.Gather`.

```
import mpi.*;
public class GatherData
{
    public static void main(String args[]) throws MPIException
    {
        MPI.Init(args);
        int rank = MPI.COMM_WORLD.Rank();
        int size = MPI.COMM_WORLD.Size();
        int number = 0;
        int numbers[] = new int[size];
        number = (rank + 1) * 10;
        MPI.COMM_WORLD.Gather(new int[]{number},0,1,MPI.INT,numbers,0,1,MPI.INT,0);
        if (rank == 0) {
             System.out.print("Process␣" + rank + "␣got␣the␣values:␣");
             for(int i = 0; i < size; i++) System.out.print(numbers[i] + "␣");
             System.out.println();
        }
        MPI.Finalize();
    }
}
```

Η κάθε διεργασία υπολογίζει την σχέση `(i + 1) * 10` όπου `i` είναι η σειρά διεργασίας. Επίσης, η διεργασία με σειρά 0 συλλέγει τις τιμές αυτές που έχουν υπολογίζει οι διεργασίες και στην συνέχεια τις εμφανίζει στην οθόνη.

Διασκορπισμό (Scatter)

Η συνάρτηση για το διασκορπισμό είναι η `Intracomm.Scatter()`. Η `Intracomm.Scatter` κάνει την αντίστροφη δουλειά από την `Intracomm.Gather()`. Με άλλα λόγια, η διεργασία ρίζα διασκορπίζει τα

δεδομένα της σε όλες τις διεργασίες, κάθε μια απο τις οποίες λαμβάνει ένα μέρος των δεδομένων. Η σύνταξη της συνάρτησης αυτής είναι ως εξής:

```
void Intracomm.Scatter(Object sendbuf, int sendoffset, int sendcount, Datatype sendtype, Object recvbuf, int recvoffset, int recvcount, Datatype recvtype, int root)
```

όπου τα ορίσματα της συνάρτησης αυτής είναι παρόμοια με τα ορίσματα της συνάρτησης `Intracomm.Gather()`.

Στο επόμενο παράδειγμα προγράμματος MPJ φαίνεται η χρήση της συνάρτησης `Intracomm.Scatter`.

```
import mpi.*;
public class ScatterTable
{
    public static void main(String args[]) throws MPIException
    {
        MPI.Init(args);
        int rank = MPI.COMM_WORLD.Rank();
        int size = MPI.COMM_WORLD.Size();
        int[] number = new int[size];
        int[] numbers = new int[size*size];
        if (rank == 0)
        {
           for(int i = 0; i < numbers.length; i++)
             numbers[i] = (i + 1) * 10;
           System.out.println("The␣table␣before␣scatter␣is␣");
           for(int i = 0; i < numbers.length; i++)
               System.out.print(numbers[i] + "␣");
            System.out.println();
        }
        MPI.COMM_WORLD.Scatter(numbers,0,size,MPI.INT,number,0,size,MPI.INT,0);
        System.out.print("Process␣" + rank + "␣got␣the␣values␣");
        for(int i = 0; i < size; i++)
          System.out.print(number[i] + "␣");
        System.out.println();
        MPI.Finalize();
    }
}
```

Η διεργασία 0 αρχικοποιεί ένα μονοδιάσταστο πίνακα μεγέθους $n \times n$ (όπου n είναι το πλήθος διεργασιών) και η τιμή του κάθε στοιχείου υπολογίζεται από την σχέση `(i + 1) * 10` όπου `i` είναι το `i`-στοιχείο του πίνακα. Επίσης, το πρόγραμμα διασκορπεί n στοιχεία του πίνακα σε κάθε διαφορετική διεργασία.

Πίνακας 9.3: Προκαθορισμένοι τελεστές αναγωγής

Τελεστές MPJ	Σημασία
`MPI.MAX`	Μεγαλύτερη τιμή
`MPI.MIN`	Μικρότερη τιμή
`MPI.SUM`	Άθροισμα
`MPI.PROD`	Γινόμενο
`MPI.LAND`	Λογικό AND
`MPI.BAND`	Δυαδικό AND
`MPI.LOR`	Λογικό OR
`MPI.BOR`	Δυαδικό OR
`MPI.LXOR`	Λογικό αποκλειστικό OR
`MPI.BXOR`	Δυαδικό αποκλειστικό OR
`MPI.MINLOC`	Ελάχιστο και η θέση του ελαχίστου
`MPI.MAXLOC`	Μέγιστο και η θέση του μεγίστου

Αναγωγή (Reduction)

Η συνάρτηση για την αναγωγή είναι η `Intracomm.Reduce()`. Η συνάρτηση αυτή συνδυάζει τα δεδομένα όλων των διεργασιών χρησιμοποιώντας έναν τελεστή πράξης σε μια τιμή που αποθηκεύεται στην διεργασία ρίζα. Οι πράξεις μπορεί να είναι πρόσθεση, πολλαπλασιασμός, μεγαλύτερη τιμή, μικρότερη τιμή, το λογικό AND, κλπ. Η σύνταξη της συνάρτησης είναι ως εξής:

```
void Intracomm.Reduce(Object sendbuf, int sendoffset, Object recvbuf, int recvoffset, int count, Datatype datatype,
Op op, int root)
```

όπου τα ορίσματα `sendbuf, sendoffset, recvbuf, recvoffset, count, datatype` και `root` είναι παρόμοια με τα ορίσματα των συναρτήσεων που έχουμε παρουσιάσει μέχρι τώρα. Το όρισμα `op` είναι ο τελεστής πράξης ή αναγωγής που συνδυάζει τα δεδομένα που περιέχονται στο `sendbuf` από κάθε διεργασία σε μια τιμή που αποθηκεύεται στο `recvbuf` της διεργασίας ρίζας. Στο MPJ υπάρχουν προκαθορισμένοι τελεστές αναγωγής μερικούς από τους οποίους παρουσιάζουμε στον Πίνακα 9.3.

Το παρακάτω πρόγραμμα φαίνεται η χρήση της συνάρτησης `Intracomm.Reduce()`.

```
import mpi.*;
public class GlobalSum
```

```
{
    public static void main(String args[]) throws MPIException
    {
        MPI.Init(args);
        int rank = MPI.COMM_WORLD.Rank();
        int size = MPI.COMM_WORLD.Size();
        int globalsum = 0;
        int number = 0;
        int[] sum = new int[]{globalsum};

        number = rank;
        MPI.COMM_WORLD.Reduce(new int[]{number},0,sum,0,1,MPI.INT,MPI.SUM,0);
        globalsum = sum[0];
        if (rank==0) System.out.println("The sum of the values is " + globalsum);
        MPI.Finalize();
    }
}
```

Το πρόγραμμα υπολογίζει το συνολικό άθροισμα με βάση την σειρά των διεργασιών. Συγκεκριμένα, κάθε διεργασία αποθηκεύει την σειρά της σε μια ακέραια μεταβλητή και η διεργασία 0 να εμφανίζει το συνολικό άθροισμα.

9.3 Υλοποίηση Υπολογιστικών Μοντέλων με MPJ

Σε αυτή την ενότητα όπως και στο προηγούμενο κεφάλαιο παρουσιάζουμε υλοποιήσεις υπολογιστικών μοντέλων με την τεχνική της μεταβίβασης μηνυμάτων.

9.3.1 Υλοποίηση Μοντέλου Παράλληλων Δεδομένων

Σε αυτή την ενότητα θα παρουσιάζουμε την υλοποίηση του μοντέλου παράλληλων δεδομένων σε MPJ. Αυτό το μοντέλο βασίζεται στην διάσπαση του προβλήματος σε έναν αριθμό ανεξαρτήτων τμημάτων δεδομένων τα οποία εκτελούνται παράλληλα και δεν απαιτούν καμιά επικοινωνία μεταξύ τους. Με άλλα λόγια, κάθε διεργασία εκτελεί τους ίδιους υπολογισμούς σε διαφορετικά τμήματα δεδομένων χωρίς να απαιτεί αποτελέσματα ή δεδομένα από άλλες διεργασίες. Για τους σκοπούς της υλοποίησης του μοντέλου παράλληλων δεδομένων, θα χρησιμοποιήσουμε το παράδειγμα υπολογισμού παραγοντικού ενός αριθμού `n`.

Για το υπολογισμό του συνολικού παραγοντικού με την χρήση της μεταβίβασης μηνυμάτων, θα πρέπει ο αριθμός `n` να διαιρεθεί σε `p` τμήματα δεδομένων (π.χ. `n / p`) και κάθε διεργασία να

υπολογίζει το τοπικό της παραγοντικό ή γινόμενο στο δικό της τμήμα δεδομένων. Κάθε τμήμα δεδομένων θα οριοθετείται από έναν αρχικό αριθμό `start = 1 + (i * n / p)` όπου `i` είναι η τάξη της διεργασίας και από ένα τελικό αριθμό `end = start + (n / p - 1)`. Αφού υπολογίζουν όλες οι διεργασίες το τοπικό τους γινόμενο, πρέπει να συλλεχθούν τα γινόμενα αυτά από την διεργασία 0 ώστε να υπολογίζει το συνολικό παραγοντικό. Για την συλλογή των τοπικών γινομένων και το υπολογισμό του συνολικού παραγοντικού χρησιμοποιούμε την συνάρτηση `Reduce()` του MPJ με το τελεστή αναγωγή γινομένου `MPI.PROD`. Παρακάτω παρουσιάζεται η υλοποίηση των παράλληλων δεδομένων σε MPJ.

```
import mpi.*;
public class Factorial
{
    public static void main(String args[]) throws MPIException
    {
        MPI.Init(args);
        int rank = MPI.COMM_WORLD.Rank();
        int size = MPI.COMM_WORLD.Size();
        int n = 10;
        int start, end, block;
        block = n / size;
        double globalfactorial = 1.0;
        double localfactorial = 1.0;
        double[] gf = new double[]{globalfactorial};

        start = 1 + (rank * block);
        end = start + (block - 1);
        for(int i = start; i <= end; i++)
                localfactorial = localfactorial * i;
        MPI.COMM_WORLD.Reduce(new double[]{localfactorial},0,gf,0,1,MPI.DOUBLE,MPI.PROD,0);
        globalfactorial = gf[0];
        if (rank == 0) System.out.println("The␣factorial␣is␣" + globalfactorial);
        MPI.Finalize();
    }
}
```

9.3.2 Υλοποίηση Μοντέλου Συντονιστής - Εργαζόμενος

Για την υλοποίηση του μοντέλου συντονιστής - εργαζόμενος σε MPJ θα χρησιμοποιήσουμε πάλι το παράδειγμα της τετραγωνικής ρίζας των στοιχείων του πίνακα όπως στο μοντέλο κοινής μνήμης. Θεωρούμε τα στοιχεία του πίνακα ως εργασίες και η διανομή των εργασιών στους υπ-

ολογιστές της συστοιχίας θα είναι δυναμική. Επομένως, η λειτουργία του μοντέλου συντονιστή - εργαζόμενου στη μεταβίβαση μηνυμάτων είναι η εξής: Ο συντονιστής στέλνει τα p πρώτα στοιχεία του πίνακα στους p αντίστοιχους υπολογιστές της συστοιχίας. Στην συνέχεια, κάθε εργαζόμενος υπολογίζει την τετραγωνική ρίζα του στοιχείου που έλαβε, έπειτα επιστρέφει το αποτέλεσμα πίσω στον συντονιστή και λαμβάνει νέα εργασία (ή στοιχείο) από τον πίνακα. Τέλος, όταν όλα τα στοιχεία του πίνακα εξαντληθούν από τους εργαζόμενους, τότε τερματίζεται η εφαρμογή. Παρακάτω φαίνεται η παραλληλοποίηση με δυναμική διανομή σε MPJ.

```
import mpi.*;
class MasterWorker {
    static public void main(String[] args) throws MPIException {
        int totalTasks = 100;
        double a[] = new double[totalTasks];
        double num[] = new double[1];
        double outs[] = new double[1];
        int sender, assignedTasks, index;
        Status status;
        MPI.Init(args) ;
        int myrank = MPI.COMM_WORLD.Rank();
        int size = MPI.COMM_WORLD.Size();
        if (myrank == 0) {
            // Arxikopoihsh pinaka
            for(int i = 0; i < totalTasks; i++)
                a[i] = i + 1;
            assignedTasks = 0;
            index = 0;
            // Anathesi ton stoixeion tou pinaka stis protes diergasies ergazomenoys
            for(int i = 1; i < size; i++) {
                num[0] = a[index];
                MPI.COMM_WORLD.Send(num, 0, 1, MPI.DOUBLE, i, 99);
                assignedTasks++;
                index++;
            }
            do {
                // Lipsi ths apanthshs (to apotelesma) apo thn diergasia ergazomenos
                status = MPI.COMM_WORLD.Recv(outs, 0, 1, MPI.DOUBLE, MPI.ANY_SOURCE, 99);
                // Prosdiorismos apostoleas
                sender = (status.source);
                System.out.println("Received␣the␣result␣" + outs[0] + "␣from␣process␣" + sender);
                assignedTasks--;
                // Elegxos an yparxoyn alles ergasies gia anathesi stis diergasies ergazomenoys
                if (index < totalTasks) {
                     num[0] = a[index];
                     MPI.COMM_WORLD.Send(num, 0, 1, MPI.DOUBLE, sender, 99);
```

```
                    assignedTasks++;
                    index++;
                }
                else { // Stelnei shma termatismoy stis diergasies ergazomenoys
                    num[0] = 0;
                    MPI.COMM_WORLD.Send(num, 0, 1, MPI.DOUBLE, sender, 100);
                }
            } while (assignedTasks > 0);
        }
        else {
            boolean state = true;
            while (state) {
                // Lipsi toy arithmoy apo thn diergasia syntonisths gia ypologismo
                status = MPI.COMM_WORLD.Recv(num, 0, 1, MPI.DOUBLE, 0, MPI.ANY_TAG);
                if (status.tag == 100) break;
                // Ypologizei thn tetragonikh riza
                outs[0] = Math.sqrt(num[0]);
                // Stelnei to apotelesma piso sthn diergasia syntonisths
                MPI.COMM_WORLD.Send(outs, 0, 1, MPI.DOUBLE, 0, 99);
            }
        }
        MPI.Finalize();
    }
}
```

9.3.3 Υλοποίηση Μοντέλου Διασωλήνωσης

Η λειτουργία του μοντέλου διασωλήνωσης είναι παρόμοια με εκείνη που είδαμε στην ενότητα 8.6.2. Για την υλοποίηση του μοντέλου διασωλήνωσης σε MPJ θα χρησιμοποιήσουμε το παράδειγμα του υπολογισμού μερικών αθροισμάτων και συνολικού αθροίσματος μια σειράς στοιχείων όπου κάθε στοιχείο είναι η τάξη της διεργασίας. Με άλλα λόγια, οι διεργασίες είναι οργανωμένες σε μορφή δακτυλίου και κάθε διεργασία αποθηκεύει την τάξη της σε μια μεταβλητή και την στέλνει στην επόμενη διεργασία. Με τη χρήση της διασωλήνωσης, οι διεργασίες υπολογίζουν συλλογικά το μερικό τους άθροισμα και επίσης συνεχίζουν να μεταβιβάζουν τις τιμές που λαμβάνουν μέχρι να λάβουν πίσω την δικιά τους τάξη. Κάθε διεργασία εμφανίζει το μερικό άθροισμα και η διεργασία με τάξη 0 εμφανίζει το συνολικό άθροισμα. Συνεπώς, η λειτουργία του μοντέλου διασωλήνωσης στη μεταβίβαση μηνυμάτων είναι η εξής: Κάθε διεργασία αποθηκεύει την τάξη της σε μια μεταβλητή `value`, την στέλνει στην επόμενη διεργασία, στην συνέχεια λαμβάνει ένα στοιχείο (ή την τάξη) της προηγούμενης διεργασίας και τέλος,

προσθέτει το στοιχείο που έλαβε στο μερικό άθροισμα της sum. Αφού ολοκληρωθεί ο κύκλος, τότε κάθε διεργασία εμφανίζει το μερικό άθροισμα και η διεργασία 0 εμφανίζει το συνολικό άθροισμα. Παρακάτω παρουσιάζεται ο κώδικας της υλοποίησης της διασωλήνωσης με δακτύλιο σε MPJ.

```
import mpi.*;
public class Pipeline
{
    public static void main(String args[]) throws MPIException
    {
        int left, right, sum, value;
        int new_value[] = new int[1];

        MPI.Init(args);
        int rank = MPI.COMM_WORLD.Rank();
        int size = MPI.COMM_WORLD.Size();

        right = rank + 1;
        if (right == size) right = 0;
        left = rank - 1;
        if (left == -1) left = size - 1;

        sum = 0;
        value = rank;
        for(int i = 0; i < size; i++) {
                // Apostolh sta dexia
                MPI.COMM_WORLD.Send(new int[]{value}, 0, 1, MPI.INT, right, 99);
                // Lipsi apo ta aristera
                MPI.COMM_WORLD.Recv(new_value, 0, 1, MPI.INT, left, 99);

                // Athroish ths timhs new_value
                sum = sum + new_value[0];
                // Enhmerosi ths timhs value poy tha apostalei
                value = new_value[0];

                // Emfanisi merikon athroismaton se kathe bhma
                System.out.println("Process␣" + rank + ":␣Partial␣sum␣=␣" + sum);
        }
        // Emfanisi synolikoy athroismatos
        if (rank == 0) {
                System.out.println("Sum␣of␣all␣processes␣numbers␣=␣" + sum);
        }
        MPI.Finalize();
    }
}
```

Κεφάλαιο 10

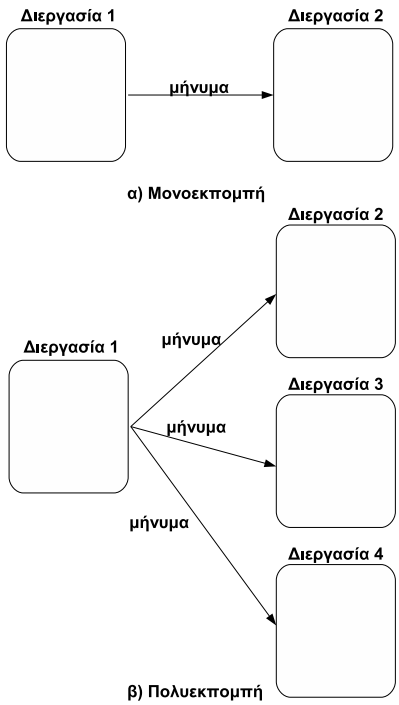
Προγραμματισμός Υποδοχών

Είναι γνωστό ότι ένα κατανεμημένο σύστημα αποτελείται από ένα σύνολο διασκορπισμένων διεργασιών (όπως πελάτες και διακομιστές) μέσω ενός αναξιόπιστου δικτύου όπως το Διαδίκτυο και οι διεργασίες επικοινωνούν μεταξύ τους μέσω πέρασμα μηνυμάτων χαμηλού επιπέδου. Το πέρασμα μηνυμάτων στα κατανεμημένα συστήματα βασίζεται στην ιδέα του πρωτοκόλλου όπως η ομάδα πρωτοκόλλων TCP/IP που χρησιμοποιείται για το Διαδίκτυο. Έτσι, σε αυτό το κεφάλαιο θα παρουσιάζουμε πρώτα τα βασικά στοιχεία για τα πρωτόκολλα και ιδιαίτερα το TCP/IP που χρησιμοποιείται στα κατανεμημένα συστήματα. Στην συνέχεια παρουσιάζουμε τον προγραμματισμό δικτυακών εφαρμογών που χρησιμοποιούν τα πρωτόκολλα TCP/IP μέσω της χρήσης υποδοχών της Java.

10.1 Διαδιεργασιακή Επικοινωνία

Είναι γνωστό ότι οι διεργασίες που εκτελούνται σε ανεξάρτητους υπολογιστές ενός κατανεμημένου συστήματος λαμβάνουν χώρα **διαδιεργασιακή επικοινωνία** (interprocess communication) δηλαδή δύο ανεξάρτητες διεργασίες επικοινωνούν μεταξύ τους ανταλλάζοντας μηνύματα μέσω του δικτύου επικοινωνίας όπως το Διαδίκτυο. Στα πλαίσια κατανεμημένων συστημάτων, δύο ή περισσότερες διεργασίες επικοινωνούν μεταξύ τους σύμφωνα με ένα πρωτόκολλο.

Υπάρχουν δύο βασικές μορφές διαδιεργασιακής επικοινωνίας σε ένα κατανεμημένο σύστημα: η **μονοεκπομπή** (unicast) και η **πολυεκπομπή** (multicast). Η επικοινωνία μονοεκπομπή είναι όταν μια διεργασία στέλνει ένα μήνυμα σε μια άλλη διεργασία όπως φαίνεται στο Σχή-



Σχήμα 10.1: Μονοεκπομπή σε σχέση με την πολυεκπομπή

μα 10.1α ενώ η επικοινωνία πολυεκπομπή είναι όταν μια διεργασία στέλνει ένα μήνυμα σε μια ομάδα διεργασιών όπως φαίνεται στο Σχήμα 10.1β.

Στις επόμενες ενότητες θα εξετάσουμε τις λειτουργίες που υποστηρίζει η διαδιεργασιακή επικοινωνία, πως αυτές οι λειτουργίες συγχρονίζουν τις διεργασίες και τέλος θα δούμε το τρόπο καθορισμού διεργασιών αποστολέα και παραλήπτη κατά την ανταλλαγή μηνυμάτων σε ένα κατανεμημένο σύστημα όπως το Διαδίκτυο.

10.1.1 Λειτουργίες Επικοινωνίας

Η διαδιεργασιακή επικοινωνία ή το πέρασμα μηνυμάτων ανάμεσα σε ένα ζεύγος διεργασιών υποστηρίζεται από τις τέσσερις λειτουργίες επικοινωνίας `send`, `receive`, `connect` και `disconnect`. Η λειτουργία `send` εκτελείται από την διεργασία αποστολέα για να στείλει μήνυμα ή δεδομένα σε μια διεργασία παραλήπτη. Η λειτουργία `receive` εκτελείται από την διεργασία παραλήπτη με σκοπό να παραλάβει το μήνυμα ή τα δεδομένα από μια διεργασία αποστολέα. Η λειτουργία `connect` δημιουργεί μια λογική σύνδεση ανάμεσα σε δύο διεργασίες: η μια διεργασία εκτελεί μια λειτουργία αίτηση για σύνδεση ενώ η άλλη διεργασία εκτελεί μια λειτουργία αποδοχής σύνδεσης.

Τέλος, η λειτουργία `disconnect` διακόπτει μια προηγούμενη λογική σύνδεση ανάμεσα στις δύο διεργασίες.

Οι παραπάνω λειτουργίες συνοδεύονται από παραμέτρους οι οποίες εξαρτώνται από το λογισμικό ή εργαλείο διαδιεργασιακής επικοινωνίας όπως θα εξετάζουμε διεξοδικά παρακάτω.

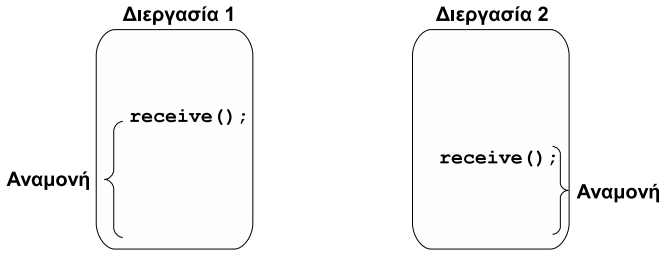
10.1.2 Σύγχρονη και Ασύγχρονη Επικοινωνία

Όταν μια διεργασία θέλει να επικοινωνήσει με μια άλλη διεργασία, η πρώτη διεργασία καλεί μια λειτουργία `send` ώστε να στείλει ένα μήνυμα στην άλλη διεργασία και η δεύτερη διεργασία καλεί μια λειτουργία `receive` ώστε να παραλάβει το μήνυμα. Αυτή η δραστηριότητα περιλαμβάνει την επικοινωνία δεδομένων από την διεργασία αποστολέα στην διεργασία παραλήπτη και ίσως περιλαμβάνει το συγχρονισμό των δύο διεργασιών. Έτσι, η επικοινωνία ανάμεσα στις διεργασίες αποστολέα και παραλήπτη μπορεί να είναι είτε σύγχρονη είτε ασύγχρονη.

Στην σύγχρονη μορφή επικοινωνίας, οι διεργασίες αποστολέα και παραλήπτη συγχρονίζονται σε κάθε μήνυμα. Σε αυτή την περίπτωση, οι λειτουργίες `send` και `receive` είναι ανασταλτικές. Όταν μια διεργασία αποστολέα καλέσει μια λειτουργία `send` εμποδίζεται μέχρι ότου εκτελεστεί μια αντίστοιχη λειτουργία `receive` από την διεργασία παραλήπτη. Επίσης, όταν μια διεργασία παραλήπτη εκτελέσει μια λειτουργία `receive` εμποδίζεται μέχρι ότου παραλάβει το μήνυμα.

Στην ασύγχρονη μορφή επικοινωνίας, η χρήση της λειτουργίας `send` είναι μη-ανασταλτική δηλαδή η διεργασία αποστολέα συνεχίζει την εκτέλεση της μόλις αντιγράψει το μήνυμα σε ένα τοπικό ενταμιευτή μηνυμάτων και η μεταφορά του μηνύματος γίνεται ανεξάρτητα ή παράλληλα με την εκτέλεση της διεργασίας αποστολέα. Επίσης, η λειτουργία `receive` είναι μη-ανασταλτική, δηλαδή η διεργασία παραλήπτη συνεχίζει την εκτέλεση του προγράμματος και το παρασκήνιο το μήνυμα μεταφέρεται στο τοπικό ενταμιευτή. Εδώ πρέπει να σημειώσουμε ότι όταν ο ενταμιευτής του παραλήπτη γεμίσει πρέπει η διεργασία αυτή να ειδοποιηθεί είτε με διακοπή είτε με ερώτηση.

Η χρήση των ανασταλτικών λειτουργιών επικοινωνίας `send` και `receive` πρέπει να είναι ιδιαίτερα προσεκτική και κάθε κλήση ανασταλτικής λειτουργίας `send` πρέπει να συνοδεύεται από την αντίστοιχη ανασταλτικής κλήσης `receive` διαφορετικά μπορεί να προκαλέσει **αδιέξοδο** (deadlock). Για παράδειγμα, στο Σχήμα 10.2 παρουσιάζεται μια περίπτωση αδιεξόδου από τις αναντιστοιχίες των ανασταλτικών λειτουργιών επικοινωνίας. Στην διεργασία 1 καλεί μια ανασταλτική λειτουργία `receive` για να παραλάβει δεδομένα από την διεργασία 2. Στην συνέχεια, η διεργασία 2



Σχήμα 10.2: Αδιέξοδο από την χρήση ανασταλτικών λειτουργιών επικοινωνίας

καλεί μια άλλη ανασταλτική λειτουργία `receive` για να παραλάβει δεδομένα από την διεργασία 1. Αυτό έχει σαν αποτέλεσμα και οι δύο διεργασίες να περιμένουν αρτέμονα μέχρι ότου κάποια από τις διεργασίες να στείλει δεδομένα το οποίο δεν πρόκειται να συμβεί ποτέ στην προκειμένη περίπτωση. Έτσι, προκύπτει το φαινόμενο κάθε μια από τις διεργασίες να εκτελούν μια ατέρμονη ανασταλτική λειτουργία. Αυτό το φαινόμενο μπορεί να εξαλειφθεί με δύο τρόπους. Ο ένας τρόπος είναι με την χρήση χρονοδιακοπή (timeout) που θέτει ένα μέγιστο χρονικό διάστημα για την ανασταλτική λειτουργία. Συνεπώς, αν μια διεργασία που εκτελεί μια ανασταλτική λειτουργία δεν ολοκληρωθεί από μια αντίστοιχη λειτουργία επικοινωνίας μέσα σε ένα λογικό χρονικό διάστημα, τότε το λειτουργικό σύστημα τερματίζει την διεργασία αυτή. Ο δεύτερος τρόπος είναι με την χρήση δημιουργίας μιας διεργασίας ή νήματος. Με άλλα λόγια, η διεργασία που εκτελεί μια ανασταλτική λειτουργία μπορεί να δημιουργήσει μια θυγατρική διεργασία ή νήμα που θα είναι σε κατάσταση αναμονής ενώ η πατρική διεργασία να συνεχίζει την εκτέλεση της.

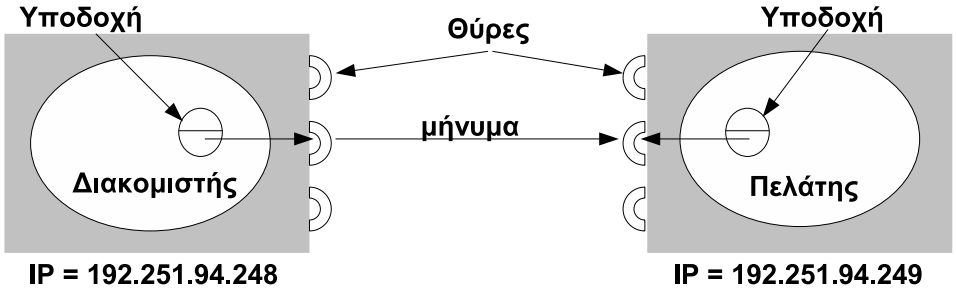
10.1.3 Προορισμούς Μηνυμάτων

Μέχρι τώρα έχουμε δεί τις λειτουργίες για την αποστολή και παραλαβή μηνυμάτων αλλά δεν έχουμε δεί το τρόπο προσδιορισμού αποστολέα και παραλήπτη κατά την διαδικασία ανταλλαγής μηνυμάτων στα κατανεμημένα συστήματα. Όπως έχουμε δεί στα πρωτόκολλα Διαδικτύου, τα μηνύματα στο Διαδίκτυο στέλνονται στην μορφή ζεύγους διεύθυνση IP και θύρας γνωστή ως υποδοχή που εξετάζαμε προηγουμένως. Μια τοπική θύρα είναι ένας προορισμός μηνύματος σε ένα υπολογιστή που προσδιορίζεται από έναν ακέραιο αριθμό. Μια θύρα έχει ακριβώς ένα παραλήπτη και πολλούς αποστολείς. Οι διεργασίες ίσως χρησιμοποιούν πολλές θύρες μέσα από τις οποίες λαμβάνουν μηνύματα. Έτσι, μια διεργασία για να στείλει ένα μήνυμα σε μια άλλη

διεργασία πρέπει η πρώτη διεργασία να γνωρίζει την διεύθυνση IP και το αριθμό θύρας της δεύτερης διεργασίας. Για αυτό το λόγο, στα κατανεμημένα συστήματα οι διεργασίες διακομιστές δημοσιοποιούν τους αριθμούς θυρών ώστε να μπορούν να χρησιμοποιηθούν από τις διεργασίες πελάτες.

10.2 Υποδοχές Επικοινωνίας

Γενικά, οι δύο ανεξάρτητες διεργασίες ενός κατανεμημένου συστήματος ανταλλάσουν μηνύματα μέσω μιας λογικής αφαίρεσης υποδοχής. Συνεπώς, η διαδιεργασιακή επικοινωνία μέσω υποδοχών αποτελείται η μετάδοση ενός μηνύματος ανάμεσα την υποδοχή της μιας διεργασίας και την υποδοχή της άλλης διεργασίας όπως φαίνεται στο Σχήμα 10.3. Από το Σχήμα παρατηρούμε ότι μια διεργασία που θέλει να επικοινωνήσει με μια άλλη διεργασία πρέπει πρώτα να δημιουργήσει μια υποδοχή. Η διεργασία που θέλει να λαμβάνει μηνύματα, πρέπει η υποδοχή της να είναι συνδεδεμένη σε μια τοπική θύρα και μια διεύθυνση IP του υπολογιστή της διεργασίας που εκτελείται. Επίσης, όταν μια διεργασία στέλνει μηνύματα σε μια υποδοχή δηλαδή σε μια συγκεκριμένη διεύθυνση IP και θύρα τότε τα μηνύματα μπορούν να παραληφθούν μόνο από τη διεργασία της οποίας η υποδοχή της είναι συσχετισμένη με αυτή τη διεύθυνση IP και θύρα.



Σχήμα 10.3: Υποδοχές και θύρες

Οι υποδοχές εκτός από το να παρέχουν μια απλή διεπαφή επικοινωνίας μέσω δικτύου είναι επίσης συσχετισμένες με τα δύο συγκεκριμένα πρωτόκολλα μεταφοράς UDP και TCP. Το πρωτόκολλο UDP επιτρέπεται να μεταφέρεται (δηλαδή, να στέλνεται ή να λαμβάνεται στο επίπεδο μεταφοράς) ένα πακέτο δεδομένων με την χρήση της ασυνδεσιοστρεφής επικοινωνίας. Το πακέτο δεδομένων που μεταφέρεται λέγεται **δεδομενόγραμμα** (datagram). Σύμφωνα

με την ασυνδεσιοστρεφή επικοινωνία, κάθε δεδομενόγραμμα πακέτο διευθυνσιοδοτείται και δρομολογείται ανεξάρτητα και ίσως φτάσει στο παραλήπτη με διαφορετική σειρά. Για παράδειγμα, αν η διεργασία 1 που εκτελείται στον υπολογιστή A στείλει μηνύματα που μεταφέρονται ως δεδομενογράμματα m1, m2 διαδοχικά στην διεργασία 2 που εκτελείται στον υπολογιστή B, τότε τα δεδομενογράμματα μεταφέρονται στο δίκτυο μέσω διαφορετικών δρομολογίων και φτάσουν στην διεργασία 2 σε μια από τις πιθανές σειρές: m1 - m2 ή m2 - m1.

Αντίθετα το πρωτόκολλο TCP είναι συνδεσιοστρεφή και μεταδίδει μια σειρά από δεδομένα πάνω σε μια λογική σύνδεση που είναι εγκατεστημένη ανάμεσα στο αποστολέα και παραλήπτη. Με αυτό το τρόπο, τα δεδομένα που στέλνει ο αποστολέας στο παραλήπτη εγγυάται ότι θα παραληφθούν με την ίδια σειρά που στάλθηκαν. Για παράδειγμα, αν η διεργασία 1 που εκτελείται στον υπολογιστή A στείλει μηνύματα m1, m2 διαδοχικά στην διεργασία 2 που εκτελείται στον υπολογιστή B τότε η διεργασία 2 θα παραλάβει τα μηνύματα με την σειρά m1 - m2 και όχι m2 - m1.

Οι υποδοχές που χρησιμοποιούν τα πρωτόκολλα μεταφοράς UDP και TCP αντιστοιχούν σε δύο διαφορετικούς τύπους υποδοχών. Οι υποδοχές που χρησιμοποιούν το πρωτόκολλο UDP για μεταφορά λέγονται **υποδοχές δεδομενογραμμάτων** (datagram sockets) ενώ οι υποδοχές που χρησιμοποιούν το πρωτόκολλο TCP ονομάζονται **υποδοχές ρεύματος** (stream sockets). Οι παρακάτω υποενότητες που ακολουθούν θα περιγράψουμε συνοπτικά την επικοινωνία για το κάθε ένα από τους δύο τύπους υποδοχών.

10.2.1 Υποδοχές Δεδομενογραμμάτων

Σε αυτή την ενότητα θα περιγράψουμε την επικοινωνία ανάμεσα στις δύο διεργασίες πελάτη - διακομιστή μέσω υποδοχών δεδομενογραμμάτων. Μια διεργασία για να στείλει ή να λάβει ένα μήνυμα θα πρέπει να δημιουργήσει μια υποδοχή που θα είναι συνδεδεμένη σε μια διεύθυνση IP του τοπικού υπολογιστή (που εκτελείται η διεργασία) και μια τοπική θύρα UDP. Έτσι, η διεργασία διακομιστή θα συνδέσει την υποδοχή σε μια θύρα διακομιστή που να είναι γνωστή σε όλες τις διεργασίες πελάτες έτσι ώστε να μπορούν να στέλνουν μηνύματα στο διακομιστή. Η διεργασία πελάτη θα συνδέσει την υποδοχή της σε μια τοπική ελεύθερη θύρα.

Στην συνέχεια, μια διεργασία για να στείλει μήνυμα σε μια άλλη διεργασία θα πρέπει επίσης να κατασκευάσει ένα πακέτο δεδομενόγραμμα. Το πακέτο αυτό θα περιέχει το μήνυμα και

μια διεύθυνση παραλήπτη (η διεύθυνση θα είναι ένα ζεύγος διεύθυνση IP και θύρα). Μόλις κατασκευαστεί το πακέτο αυτό, η διεργασία αποστολέα φορτώνει σε αυτό το μήνυμα και την διεύθυνση παραλήπτη. Η διεργασία αποστολέα τότε καλεί μια συνάρτηση `send()` με παράμετρο το δεδομενόγραμμα πακέτο ώστε να μεταφερθεί το πακέτο αυτό στον παραλήπτη.

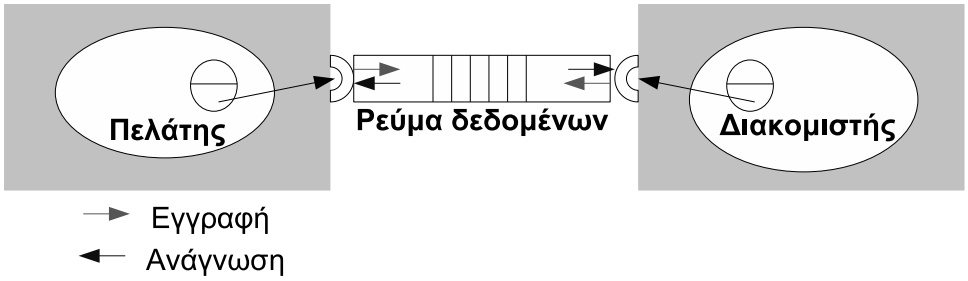
Στην άλλη πλευρά, η διεργασία παραλήπτη αφού η υποδοχή της είναι συνδεδεμένη σε μια τοπική θύρα πρέπει επίσης να κατασκευάσει το δεδομενόγραμμα πακέτο ώστε να τοποθετήσει το μήνυμα από το αποστολέα. Η διεργασία παραλήπτη καλεί μια συνάρτηση `receive()` με παράμετρο το δεδομενόγραμμα πακέτο ώστε να παραλάβει το μήνυμα που θα τοποθετηθεί μέσα στο πακέτο. Κατά την κλήση της συνάρτησης `receive()` ίσως επιστρέψει την διεύθυνση IP και την θύρα της διεργασίας αποστολέας, που θα επιτρέψει την διεργασία παραλήπτη να στείλει μια απάντηση.

Τέλος, οι υποδοχές δεδομενογραμμάτων υποστηρίζουν μη-ανασταλτικές συναρτήσεις `send()` και ανασταλτικές συναρτήσεις `receive()`. Η μη ανασταλτική συνάρτηση `send()` θα συνεχίζει την εκτέλεση ακόμα και μετά την κλήση της. Η ανασταλτική συνάρτηση `receive()` αναστέλει την διεργασία που την κάλεσε μέχρι να παραλάβει το δεδομενόγραμμα πακέτο. Επίσης, για αποφυγή αρτέμονη αναστολή η διεργασία παραλήπτη μπορεί να χρησιμοποιεί μια τιμεουτ στην υποδοχή της ώστε να ανασταλεί για ορισμένο χρονικό διάστημα.

10.2.2 Υποδοχές Ρεύματος

Στις υποδοχές δεδομενογραμμάτων υποστηρίζει την ανταλλαγή διακριτών πακέτων δεδομένων ανάμεσα στις δύο υποδοχές των διεργασιών ενώ οι υποδοχές ρεύματος υποστηρίζει την λογική αφαίρεση ενός ρεύματος δεδομένων εισόδου - εξόδου ανάμεσα στις δύο υποδοχές. Σε ένα ρεύμα δεδομένων εισόδου - εξόδου ανάμεσα στις δύο υποδοχές όπως φαίνεται στο Σχήμα 10.4, τα δεδομένα ρέουν μέσω συνεχούς ρεύματος από την διεργασία πελάτη (ή αποστολέα) στην διεργασία διακομιστή (ή παραλήπτη). Τα δεδομένα γράφονται στο ρεύμα δεδομένων από την διεργασία αποστολέα και επίσης τα δεδομένα διαβάζονται από το ρεύμα από την διεργασία παραλήπτη. Σημειώνουμε ότι η φύση του συνεχούς ρεύματος επιτρέπει τα δεδομένα να γράφονται στο ρεύμα και να διαβάζονται από το ρεύμα με διαφορετικούς ρυθμούς.

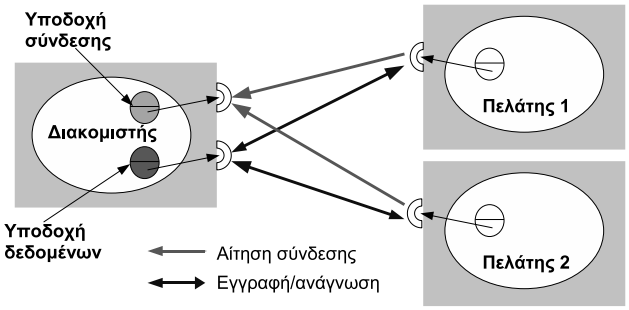
Συνεπώς για να επικοινωνήσουν δυο διεργασίες μέσω ρεύματος πρέπει πρώτα να δημιουργήσουν τις υποδοχές τους και έπειτα να εγκατασταθεί μια σύνδεση ανάμεσα στις δύο αυτές υποδοχές. Η εγκατάσταση της σύνδεσης γίνεται μέσω της χειραψίας τριών φάσεων σύμφωνα με το



Σχήμα 10.4: Υποδοχές ρεύματος για μεταφορά δεδομένων

πρωτόκολλο TCP. Μόλις εγκατασταθεί η σύνδεση χρησιμοποιούνται οι υποδοχές ως ρεύματα δεδομένων για ανταλλαγή μηνυμάτων ή δεδομένων ανάμεσα στις δύο διεργασίες. Σε αυτή την περίπτωση, οι διεργασίες απλά γράφουν δεδομένα στο ρεύμα και διαβάζουν από το ρεύμα χωρίς να χρησιμοποιήσουν κάθε φορά την διεύθυνση IP και θύρα παρά μόνο στην αρχή της σύνδεσης.

Σημειώνουμε ότι κατά την επικοινωνία πελάτη - διακομιστή μέσω υποδοχών ρεύματος υπάρχουν δύο τύποι υποδοχών όπως φαίνεται στο Σχήμα 10.5: η **υποδοχή σύνδεσης** (connection socket) η οποία χρησιμοποιείται στην πλευρά του διακομιστή για αποδοχή συνδέσεων από τους πελάτες και η **υποδοχή δεδομένων** (data socket) η οποία χρησιμοποιείται και στις δυο πλευρές (πελάτη και διακομιστή) για ανταλλαγή (δηλαδή εγγραφή και ανάγνωση) δεδομένων.



Σχήμα 10.5: Υποδοχές σύνδεσης και δεδομένων

Με βάση τα παραπάνω η υλοποίηση της επικοινωνίας μεταξύ πελάτη - διακομιστή με την χρήση υποδοχών ρεύματος είναι η εξής: Ένας διακομιστής εκτελείται σε ένα συγκεκριμένο υπολογιστή και δημιουργεί μια υποδοχή σύνδεσης που είναι συνδεδεμένη σε μια συγκεκριμένη τοπική θύρα. Η διεύθυνση IP του υπολογιστή και η θύρα είναι ευρέως γνωστές στους πελάτες. Ο διακομιστής περιμένει και ακούει την υποδοχή σύνδεσης για αιτήσεις σύνδεσης από τους πελάτες.

Στην συνέχεια, ο πελάτης πρέπει να δημιουργήσει μια υποδοχή που να περιέχει την διεύθυνση IP του υπολογιστή διακομιστή καθώς και το αριθμό θύρας στην οποία είναι συνδεδεμένος ο διακομιστής. Όταν δημιουργηθεί η υποδοχή αυτή εγκαθιστά μια σύνδεση με τον διακομιστή. Με βάση τα στοιχεία αυτά, ο πελάτης μέσω της υποδοχής σύνδεσης του διακομιστή στέλνει μια αίτηση σύνδεσης στον διακομιστή.

Αν όλα πάνε καλά, ο διακομιστής κάνει αποδεκτή την αίτηση του πελάτη με αποτέλεσμα τη δημιουργία μιας νέας υποδοχής δεδομένων η οποία είναι συνδεδεμένη σε μια διαφορετική θύρα. Η νέα αυτή υποδοχή δεδομένων είναι απαραίτητη ώστε ο διακομιστής να ανταλλάσει δεδομένα με ένα συγκεκριμένο πελάτη ενώ η αρχική υποδοχή σύνδεσης παραμένει συνδεδεμένη στην θύρα διακομιστή ώστε να ακούει για νέες εισερχόμενες αιτήσεις από άλλους πελάτες.

Στην συνέχεια, ακολουθεί η ανταλλαγή δεδομένων ανάμεσα στο ζεύγος υποδοχών δεδομένων του πελάτη και του διακομιστή. Οι υποδοχές δεδομένων συνδέονται από ένα ζεύγος ρευμάτων, ένα σε κάθε κατεύθυνση. Δηλαδή κάθε υποδοχή έχει ένα ρεύμα εισόδου και ένα ρεύμα εξόδου. Έτσι, όταν μια από τις δύο διεργασίες θέλει να στείλει δεδομένα στην άλλη διεργασία εκτελεί μια λειτουργία εγγραφής στο ρεύμα της εξόδου και η άλλη διεργασία εκτελεί μια λειτουργία ανάγνωσης από το ρεύμα της εισόδου για να λάβει τα δεδομένα. Σημειώνουμε ότι όταν μια διεργασία επιχειρεί να διαβάζει δεδομένα από ένα άδειο ρεύμα εισόδου τότε η διεργασία αναστέλεται μέχρι να γίνουν διαθέσιμα τα δεδομένα. Επίσης, όταν μια διεργασία επιχειρεί να γράψει δεδομένα στο ρεύμα εξόδου, ίσως η διεργασία ανασταλεί από το μηχανισμό ροής ελέγχου του TCP όταν η άλλη διεργασία διαβάζει τα δεδομένα με αργό ρυθμό.

Τέλος, όταν ολοκληρωθεί η επικοινωνία μεταξύ των διεργασιών διακομιστή - πελάτη, ο διακομιστής κλείνει την υποδοχή δεδομένων και η υποδοχή σύνδεσης παραμένει ανοιχτή για να δεχτεί επόμενη σύνδεση από άλλο πελάτη. Στην πλευρά του πελάτη κλείνει την υποδοχή δεδομένων του.

10.3 Διεπαφή Υποδοχών σε Java

Στην ενότητα αυτή θα παρουσιάζουμε τον προγραμματισμό δικτυακών εφαρμογών που χρησιμοποιούν τα πρωτόκολλα IP, TCP και UDP με την απλοποιημένη διεπαφή υποδοχών Java. Συγκεκριμένα, η Java διαθέτει το πακέτο `java.net` που περιέχει σημαντικές κλάσεις οι οποίες μας επιτρέπουν να αναπτύξουμε δικτυακές εφαρμογές και δίνουν την δυνατότητα στον προ-

γραμματιστή να εγκαταστήσει επικοινωνία ανάμεσα στις υποδοχές πελάτη - διακομιστή με απλό τρόπο. Οι κλάσεις αυτές περιέχουν τις διευθύνσεις IP, τις υποδοχές δεδομενογραμμάτων UDP, τις υποδοχές ρεύματος TCP, και υποδοχές πολυεκπομπής. Χρησιμοποιώντας τις κλάσεις αυτές μπορούμε να αναπτύξουμε εφαρμογές που επικοινωνούν ανάμεσα στο πελάτη και διακομιστή ή εφαρμογές που επικοινωνούν με ένα διακομιστή του Διαδικτύου. Στις παρακάτω ενότητες θα περιγράψουμε τις προαναφερθείσες σημαντικές κλάσεις.

10.3.1 Διευθύνσεις: Κλάση `InetAddress`

Η κλάση `InetAddress` χειρίζεται τις διευθύνσεις Διαδικτύου ως ονόματα υπολογιστών και ως διευθύνσεις IP. Η κλάση `InetAddress` έχει δύο υποκλάσεις τις `Inet4Address` και `Inet6Address` οι οποίες παριστάνουν τις 32-bit διευθύνσεις IPv4 και 128-bit διευθύνσεις IPv6, αντίστοιχα. Κάθε αντικείμενο τύπου `InetAddress` περιέχει μια διεύθυνση IP και πιθανώς το αντίστοιχο όνομα υπολογιστή. Η κλάση αυτή δεν έχει κατασκευαστές και παρέχει μεθόδους για την αντιστοίχιση ονομάτων υπολογιστών στις διευθύνσεις IP τους και αντίστροφα. Μερικές από τις κοινές μεθόδους της κλάσης `InetAddress` παρουσιάζονται παρακάτω.

Μέθοδοι `InetAddress`

- `public static InetAddress getByName (String host) throws UnknownHostException` όπου η παράμετρος `host` είναι ένα αλφαριθμητικό που μπορεί να είναι είτε το όνομα ενός υπολογιστή, π.χ. `www.uom.gr` είτε η διεύθυνση IP που έχει σε μορφή αλφαριθμητικού, π.χ. '195.251.197.55'.

  Η μέθοδος `getByName` χρησιμοποιεί την υπηρεσία ονομασίας DNS που επιστρέφει ένα αντικείμενο τύπου `InetAddress` (π.χ. τη διεύθυνση IP ενός υπολογιστή) με βάση το όνομα του που ορίζεται από την παράμετρο `host`. Για παράδειγμα, όταν θέλουμε να πάρουμε τη διεύθυνση IP που αντιστοιχεί στο όνομα υπολογιστή `www.uom.gr` ορίζουμε τη μεταβλητή `address` που κρατά ένα αντικείμενο `InetAddress` ως εξής:

  ```
  InetAddress address = InetAddress.getByName("www.uom.gr");
  ```

  Η μέθοδος αυτή μπορεί να δημιουργήσει μια εξαίρεση τύπου `UnknownHostException` στην περίπτωση που δεν αναγνωριστεί το όνομα του υπολογιστή `host`. Έχουμε την δυνατότητα να παράγουμε μέσα από το πρόγραμμα αυτή την εξαίρεση ή να χειριστούμε την εξαίρεση αυτή με δύο

τμήματα κώδικα `try/catch`.

- `public static InetAddress getAllByName (String host) throws UnknownHostException` όπου η παράμετρος `host` είναι ακριβώς ίδια όπως στην μέθοδο `getByName`.

  Η μέθοδος `getAllByName` επιστρέφει έναν πίνακα με αντικείμενα τύπου `InetAddress` που το κάθε αντικείμενο περιέχει μια δικτυακή διεύθυνση (π.χ. τη διεύθυνση IP ή το όνομα ενός υπολογιστή) με βάση το όνομα του που ορίζεται από την παράμετρο `host`. Η μέθοδος αυτή χρησιμοποιείται όταν θέλουμε να πάρουμε πολλές διευθύνσεις IP (αν υπάρχουν) οι οποίες αντιστοιχούν σε ένα υπολογιστή. Για παράδειγμα, εάν ένας υπολογιστής με όνομα `www.uom.gr` έχει πολλές διευθύνσεις IP μπορούμε να ορίζουμε έναν πίνακα `address` ως εξής:

  ```
  InetAddress address[] = InetAddress.getAllByName("www.uom.gr");
  ```

  Η μέθοδος αυτή δημιουργεί μια εξαίρεση τύπου `UnknownHostException` όπως ακριβώς ισχύουν στην μέθοδο `getByName`.

- `public InetAddress getLocalHost () throws UnknownHostException`

  με καμία παράμετρο.

  Η μέθοδος `getLocalHost` επιστρέφει ένα αντικείμενο τύπου `InetAddress` που αντιστοιχεί στον τοπικό υπολογιστή, δηλαδή επιστρέφει τη διεύθυνση IP του τοπικού υπολογιστή. Για παράδειγμα, όταν θέλουμε να βρούμε τη διεύθυνση IP ενός τοπικού υπολογιστή γράφουμε ως εξής:

  ```
  InetAddress address = InetAddress.getLocalHost();
  ```

  Τέλος, η μέθοδος αυτή δημιουργεί εξαίρεση τύπου `UnknownHostException` όπως και στις παραπάνω μεθόδους.

- `public String getHostAddress ()`

  Η μέθοδος `getHostAddress` επιστρέφει μια διεύθυνση IP σε μορφή αλφαριθμητικού. Για να χρησιμοποιήσουμε τη μέθοδο αυτή πρέπει πρώτα να έχουμε δημιουργήσει αντικείμενο της τάξης `InetAddress` με βάση το όνομα ενός υπολογιστή, π.χ. να χρησιμοποιήσουμε την μέθοδο `getByName` ή την μέθοδο `getAllByName`. Για παράδειγμα, όταν θέλουμε να πάρουμε τη διεύθυνση IP ενός υπολογιστή `www.uom.gr` σε μορφή αλφαριθμητικού γράφουμε το παρακάτω τμήμα κώδικα:

```
InetAddress address = InetAddress.getByName("www.uom.gr");

System.out.println("IP address:  " + address.getHostAddress());
```

- `public String getHostName ()`

  Η μέθοδος `getHostName` επιστρέφει το όνομα του υπολογιστή σε μορφή αλφαριθμητικού. Για να χρησιμοποιήσουμε τη μέθοδο αυτή πρέπει πρώτα να έχουμε δημιουργήσει αντικείμενο της τάξης `InetAddress` με βάση το όνομα ενός υπολογιστή, π.χ. να χρησιμοποιήσουμε την μέθοδο `getByName` ή την μέθοδο `getAllByName`. Όμως, στην περίπτωση που το αντικείμενο έχει δημιουργηθεί με βάση τη διεύθυνση IP, το όνομα εντοπίζεται μέσω της υπηρεσίας DNS και επιστρέφεται, διαφορετικά επιστρέφεται η ίδια η διεύθυνση IP σε μορφή αλφαριθμητικού. Για παράδειγμα, όταν θέλουμε να πάρουμε το όνομα ενός υπολογιστή `www.uom.gr` σε μορφή αλφαριθμητικού γράφουμε το παρακάτω τμήμα κώδικα:

  ```
  InetAddress address = InetAddress.getByName("www.uom.gr");

  System.out.println("IP address:  " + address.getHostName());
  ```

Παρακάτω παρουσιάζεται ένα πρώτο απλό πρόγραμμα που εντοπίζει και εμφανίζει το όνομα και τη διεύθυνση IP του τοπικού μας υπολογιστή.

```
import java.io.*;
import java.net.*;
public class GetLocalNameIP
{
        public static void main(String[] args) throws IOException
        {
                // Epistrofi antikeimenou InetAddress
                InetAddress address = InetAddress.getLocalHost();
                // Emfaniseis mhnymaton
                System.out.println("Local␣Host␣Name:␣" + address.getHostName());
                System.out.println("Local␣IP␣Address:␣" + address.getHostAddress());
        }
}
```

Επίσης, παρακάτω παρουσιάζεται ένα άλλο πρόγραμμα που διαβάζει το όνομα του υπολογιστή από το πληκτρολόγιο και επιστρέφει (ή εμφανίζει) το όνομα υπολογιστή και τη διεύθυνση IP που αντιστοιχεί.

```
import java.io.*;
import java.net.*;
```

```
import java.util.Scanner;
public class NsLookup
{
        public static void main(String[] args) throws IOException
        {
                String hostname;
                // Anagnosi onomatos ypologisth
                Scanner input = new Scanner(System.in);
                System.out.print("Enter␣host␣name:␣");
                hostname = input.nextLine();
                // Epistrofi antikeimenou InetAddress
                InetAddress address = InetAddress.getByName(hostname);
                // Emfaniseis mhnymaton
                System.out.println("Host␣Name:␣" + address.getHostName());
                System.out.println("IP␣address:␣" + address.getHostAddress());
        }
}
```

10.3.2 Υποδοχές Δεδομενογραμμάτων UDP

Η υποδοχή είναι το βασικό μέσο που χρησιμοποιεί η Java στην τεχνολογία του δικτύου. Σ αυτή την ενότητα θα δούμε υποδοχές που είναι βασισμένες στο πρωτόκολλο UDP και στην επόμενη ενότητα θα περιγράψουμε υποδοχές που είναι βασισμένες στο πρωτόκολλο TCP.

Η Java υποστηρίζει υποδοχές δεδομενογραμμάτων UDP κάτω από δύο κλάσεις, την κλάση πακέτων δεδομενογραμμάτων `DatagramPacket` και την κλάση υποδοχών `DatagramSocket` οι οποίες βρίσκονται στο πακέτο `java.net`. Η κλάση `DatagramPacket` παριστάνει ένα πακέτο δεδομένων για αποστολή ή λήψη μέσω του πρωτοκόλλου UDP. Τα πακέτα δεδομένων περιέχουν μια σειρά από bytes καθώς και πληροφορίες διεύθυνσης όπως η διεύθυνση IP και η θύρα. Η κλάση `DatagramSocket` υλοποιεί υποδοχές UDP ώστε τα πακέτα UDP να μεταφέρονται δηλαδή να στέλνονται ή λαμβάνονται. Πρώτα θα ξεκινήσουμε να περιγράψουμε τις κλάσεις `DatagramPacket` και `DatagramSocket` και έπειτα τα βήματα που απαιτούνται για την δημιουργία ενός προγράμματος πελάτη και διακομιστή.

Κατασκευαστές `DatagramPacket`

Υπάρχουν δύο λόγοι για να δημιουργήσουμε ένα αντικείμενο τύπου `DatagramPacket`. Πρώτον, για να στείλουμε δεδομένα σε ένα απομακρυσμένο υπολογιστή μέσω του πρωτοκόλλου UDP και δεύτερον για να λάβουμε δεδομένα από ένα απομακρυσμένο υπολογιστή. Για αυτό τον λόγο θα

παρουσιάζουμε παρακάτω δύο κοινούς κατασκευαστές της κλάσης `DatagramPacket`.

- `public DatagramPacket(byte[] buff, int length, InetAddress host, int port)`. Δημιουργεί ένα αντικείμενο `DatagramPacket` για την αποστολή πακέτου UDP το οποίο περιέχει το πίνακα `buff` (που αποτελείται το μήνυμα) μεγέθους `length` στο απομακρυσμένο υπολογιστή προορισμού με διεύθυνση `host` στην θύρα `port`.

- `public DatagramPacket(byte[] buff, int length)`. Δημιουργεί ένα αντικείμενο `DatagramPacket` για τη λήψη εισερχόμενου πακέτου UDP μεγέθους `length` και το μήνυμα που λαμβάνεται τοποθετείται στο πίνακα `buff`. Η τιμή της παραμέτρου `length` πρέπει να είναι μικρότερη ή ίση με το μέγεθος του πίνακα `buff`.

Μεθόδους `DatagramPacket`

Η κλάση `DatagramPacket` διαθέτει μερικές σημαντικές μεθόδους για την χρησιμοποίηση των πακέτων. Στην ενότητα αυτή θα περιγράψουμε έξι κοινές μεθόδους:

- `InetAdress getAddress()`. Επιστρέφει τη διεύθυνση IP του υπολογιστή στην οποία στέλνεται ή από την οποία προέρχεται το πακέτο δεδομενόγραμμα.

- `public byte[] getData()`. Επιστρέφει τα περιεχόμενα ή το μήνυμα του πακέτου `DatagramPacket` που παριστάνεται από ένα πίνακα bytes.

- `public int getPort()`. Επιστρέφει τον αριθμό θύρας του υπολογιστή στην οποία στέλνεται ή από την οποία προέρχεται το πακέτο δεδομενόγραμμα.

- `public void setAddress(InetAddress addr)`. Θέτει τη διεύθυνση IP του υπολογιστή σε ένα πακέτο δεδομενόγραμμα που πρόκειται να αποσταλεί.

- `public void setData(byte[] buff)`. Θέτει τα δεδομένα στο πακέτο `DatagramPacket`.

- `public void setPort(int port)`. Θέτει το αριθμό της θύρας του υπολογιστή στο πακέτο `DatagramPacket` που πρόκειται να αποσταλεί.

Κατασκευαστές `DatagramSocket`

Η κλάση `DatagramSocket` συνήθως περιέχει δύο κατασκευαστές.

- `public DatagramSocket() throws SocketException.` Δημιουργεί μια υποδοχή πελάτη και τη συνδέει σε οποιαδήποτε ελεύθερη θύρα στον υπολογιστή. Στην περίπτωση που συμβεί σφάλμα κατά τη δημιουργία της υποδοχής τότε δημιουργείται μια εξαίρεση τύπου `SocketException`.

- `public DatagramSocket(int port) throws SocketException.` Δημιουργεί μια υποδοχή διακομιστή και τη συνδέει σε μια συγκεκριμένη θύρα `port` του υπολογιστή. Στην περίπτωση που συμβεί σφάλμα κατά τη δημιουργία της υποδοχής ή δεν μπορεί να δεσμευτεί στην θύρα `port` τότε δημιουργείται μια εξαίρεση τύπου `SocketException`.

Μεθόδους `DatagramSocket`

Η κλάση `DatagramSocket` διαθέτει μεθόδους για την αποστολή και λήψη πακέτων δεδομενογραμμάτων UDP ανάμεσα σε ένα ζεύγος υποδοχών.

- `public void send(DatagramPacket packet) throws IOException.` Στέλνει ένα δεδομενόγραμμα πακέτο μέσω της υποδοχής UDP. Είναι γνωστό ότι το πακέτο `packet` περιέχει το μήνυμα και τις πληροφορίες διεύθυνσης παραλήπτη (δηλαδή διεύθυνση IP και αριθμό θύρας).

- `public void receive(DatagramPacket packet) throws IOException.` Διαβάζει ένα πακέτο UDP και τοποθετεί τα περιεχόμενα του στο πακέτο `packet`. Επομένως, το πακέτο θα περιέχει τα δεδομένα και το μήκος τους αλλά και τη διεύθυνση και τον αριθμό θύρας υπολογιστή του αποστολέα. Το πεδίο `length` του αντικειμένου `packet` περιέχει το μήκος των δεδομένων που παρέλαβε. Αν τα δεδομένα που παρελήφθησαν είχαν μεγαλύτερο μήκος από τον ενταμιευτή του πακέτου `packet` τότε περικόπτονται.

- `public void setSoTimeout(int timeout).` Θέτει μια χρονική περίοδο `timeout` σε milliseconds για την ανασταλτική μέθοδο `receive`.

- `public void close().` Κλείνει την υποδοχή UDP και τη αποσυνδέει από την θύρα.

Βήματα Δημιουργίας Προγράμματος Πελάτη

Παρακάτω θα αναπτύξουμε τα βήματα για τη δημιουργία προγραμμάτων πελάτη και διακομιστή ξεχωριστά για υποδοχές UDP. Πρώτα, θα περιγράψουμε τα βήματα που απαιτούνται για την δημιουργία ενός προγράμματος πελάτη. Η διαδικασία αυτή περιλαμβάνει οκτώ βήματα.

1. Πρώτα δημιουργούμε ένα αντικείμενο τύπου `DatagramSocket`. Αυτό σημαίνει ότι χρησιμοποιούμε το κατασκευαστή `DatagramSocket` που δεν απαιτεί καμία παράμετρος. Για παράδειγμα,

   ```
   DatagramSocket dgramSocket = new DatagramSocket();
   ```

2. Δημιουργούμε ένα εξερχόμενο δεδομενόγραμμα πακέτο (ή πακέτο προς αποστολή). Δηλαδή δημιουργούμε ένα αντικείμενο `DatagramPacket` ο οποίος παίρνει τέσσερις παραμέτρους:

   - το πίνακα τύπου `byte` που περιέχει το μήνυμα αποστολής.
   - το μέγεθος του μηνύματος.
   - την διεύθυνση του πελάτη.
   - τον αριθμό θύρας του πελάτη.

   Η πρώτη παράμετρος από τους τέσσερις επιστρέφεται από την μέθοδο `getBytes` της κλάσης `String`. Για παράδειγμα,

   ```
   DatagramPacket outPacket = new DatagramPacket(message.getBytes(),message.length(),host,PORT);
   ```

3. Έπειτα στέλνουμε το μήνυμα του δεδομενόγραμμου πακέτου. Αυτό πραγματοποιείται με την κλήση της μεθόδου `send` του αντικειμένου `DatagramSocket` παίρνοντας ως παράμετρος το εξερχόμενο αντικείμενο `DatagramPacket` που δημιουργήσαμε στο προηγούμενο βήμα. Για παράδειγμα,

   ```
   dgramSocket.send(outPacket);
   ```

4. Στην συνέχεια δημιουργούμε ένα ενταμιευτή για τα εισερχόμενα πακέτα δεδομενογραμμάτων. Αυτό υλοποιείται δημιουργώντας ένα πίνακα τύπου `bytes`. Για παράδειγμα,

   ```
   byte[] buffer = new byte[256];
   ```

5. Δημιουργούμε ένα αντικείμενο `DatagramPacket` για τα εισερχόμενα πακέτα δεδομενογραμμάτων. Ο κατασκευαστής του αντικειμένου αυτού απαιτούν δύο παραμέτρους:

- το πίνακα `byte` που δημιουργήσαμε προηγουμένως.
- το μέγεθος του παραπάνω πίνακα.

Για παράδειγμα,

```
DatagramPacket inPacket= new DatagramPacket(buffer,buffer.length);
```

6. Αποδοχή του εισερχόμενου πακέτου δεδομενογραμμάτων. Αυτό υλοποιείται με την κλήση της μεθόδου `receive` του αντικειμένου `DatagramSocket` χρησιμοποιώντας ως παράμετρος το αντικείμενο `DatagramPacket` που δημιουργήσαμε προηγουμένως. Για παράδειγμα,

   ```
   dgramSocket.receive(inPacket);
   ```

7. Ανακτούμε τα δεδομένα από τον ενταμιευτή. Τα δεδομένα θα ανακτηθούν ως αλφαριθμητικό χρησιμοποιώντας μια μορφή πολυμορφισμού του κατασκευαστή `String` που απαιτεί τρεις παραμέτρους:

   - ένα πίνακα τύπου `byte`.
   - η αρχική θέση του πίνακα (δηλαδή το 0).
   - το αριθμό των `bytes` (δηλαδή το μέγεθος του ενταμιευτή).

   Για παράδειγμα,

   ```
   String response = new String(inPacket.getData(),0,inPacket.getLength());
   ```

8. Τέλος, τερματίζουμε την υποδοχή `DatagramSocket`. Αυτό πραγματοποιείται με την κλήση της μεθόδου `close` του αντικειμένου `DatagramSocket`. Για παράδειγμα,

   ```
   dgramSocket.close();
   ```

Στο παρακάτω παράδειγμα παρουσιάζεται ένα πρόγραμμα πελάτη που στέλνει ένα μήνυμα χαιρετισμού ''Hello from client!'' στον διακομιστή και λαμβάνει ένα μήνυμα επιβεβαίωσης από τον διακομιστή.

```
import java.net.*;
import java.io.*;
public class SimpleClientUDP {
        private static final int PORT = 1234;
        public static void main(String args[]) throws IOException {
                // Bhma 1o: Dhmioyrgia antikeimenou ypodoxhs DatagramSocket
```

```
                InetAddress host = InetAddress.getLocalHost();
                DatagramSocket dgramSocket = new DatagramSocket();
                System.out.println("Connection␣established");
                // Bhma 2o: Dhmioyrgia antikeimenoy DatagramPacket gia apostolh
                String message = "Hello␣from␣Client!";
                DatagramPacket outPacket = new DatagramPacket(message.getBytes(),message.length(),host,POR
                // Bhma 3o: Apostolh paketoy
                dgramSocket.send(outPacket);
                System.out.println("Message␣sent:␣Hello␣from␣Client!");
                // Bhma 4o:     Dhmioyrgia entamieyth
                byte[] buffer = new byte[256];
                // Bhma 5o: Dhmioyrgia antikeimenoy DatagramPacket gia lhpsi
                DatagramPacket inPacket= new DatagramPacket(buffer,buffer.length);
                // Bhma 6o: Lhpsi paketoy
                dgramSocket.receive(inPacket);
                // Bhma 7o: Anaktisi dedomenon apo to entamieyth
                String response = new String(inPacket.getData(),0,inPacket.getLength());
                System.out.println("Received␣message␣from␣server:" + response);
                // Bhma 8o: Kleisimo ths ypodoxhs
                dgramSocket.close();
                System.out.println("Datagram␣socket␣closed");
        }
}
```

Βήματα Δημιουργίας Προγράμματος Διακομιστή

Παρόμοια διαδικασία ακολουθούμε για την δημιουργία προγράμματος διακομιστή το οποίο περιλαμβάνει εννιά βήματα.

1. Πρώτα δημιουργούμε ένα αντικείμενο `DatagramSocket`. Αυτό το βήμα είναι παρόμοιο όπως στο πρόγραμμα πελάτη με την μόνη διαφορά ότι ο κατασκευαστής απαιτεί μια παράμετρο τον αριθμό θύρας. Για παράδειγμα,

   ```
   DatagramSocket dgramSocket = new DatagramSocket(1234);
   ```

   Τα βήματα 2-4 είναι ακριβώς ίδια όπως στα βήματα 4-6 του προγράμματος πελάτη.

2. Δημιουργούμε ένα ενταμιευτή για τα εισερχόμενα πακέτα δεδομενογραμμάτων. Για παράδειγμα,

   ```
   byte[] buffer = new byte[256];
   ```

3. Δημιουργούμε ένα αντικείμενο `DatagramPacket` για τα εισερχόμενα πακέτα δεδομενογραμμάτων. Για παράδειγμα,

```
DatagramPacket inPacket = new DatagramPacket(buffer, buffer.length);
```

4. Αποδοχή του εισερχόμενου πακέτου δεδομενογραμμάτων. Για παράδειγμα,

   ```
   dgramSocket.receive(inPacket);
   ```

5. Αποδοχή την διεύθυνση του αποστολέα και την θύρα από το εισερχόμενο πακέτο. Αυτό πραγματοποιείται με την χρήση των μεθόδων `getAddress` και `getPort` του αντικειμένου `DatagramPacket`. Για παράδειγμα,

   ```
   InetAddress clientAddress = inPacket.getAddress();

   int clientPort = inPacket.getPort();
   ```

6. Ανακτούμε τα δεδομένα από τον ενταμιευτή. Αυτό το βήμα είναι ίδιο με το βήμα 7 του προγράμματος πελάτη. Για παράδειγμα,

   ```
   String message = new String(inPacket.getData(),0,inPacket.getLength());
   ```

7. Δημιουργούμε ένα δεδομενόγραμμο πακέτο απάντησης. Αυτή η διαδικασία είναι ακριβώς ίδια όπως στο βήμα 2 του προγράμματος πελάτη. Για παράδειγμα,

   ```
   DatagramPacket outPacket = new DatagramPacket(response.getBytes(),response.length(),clientAddress,clientPort);
   ```

   Το `response` είναι μεταβλητή `String` που αποθηκεύει το εξερχόμενο μήνυμα.

8. Στέλνουμε το δεδομενόγραμμο πακέτο απάντησης. Αυτό πραγματοποιείται με την κλήση της μεθόδου `send` του αντικειμένου `DatagramSocket` παίρνοντας ως παράμετρος το εξερχόμενο αντικείμενο `DatagramPacket` που δημιουργήσαμε στο προηγούμενο βήμα. Για παράδειγμα,

   ```
   dgramSocket.send(outPacket);
   ```

   Τα βήματα 4-8 εκτελούνται μέσα σε ένα ατέρμονα βρόχο.

9. Τέλος, τερματίζουμε την υποδοχή `DatagramSocket`. Αυτό πραγματοποιείται με την κλήση της μεθόδου `close` του αντικειμένου `DatagramSocket`. Για παράδειγμα,

   ```
   dgramSocket.close();
   ```

Στο παρακάτω παράδειγμα παρουσιάζεται ένα πρόγραμμα διακομιστή που δέχεται ένα μήνυμα χαιρετισμού από τον πελάτη και του επιστρέφει ένα μήνυμα επιβεβαίωσης ''Goodbye!''.

```
import java.net.*;
import java.io.*;
public class SimpleServerUDP {
        private static final int PORT = 1234;
        public static void main(String args[]) throws IOException {
                // Bhma 1o: Dhmioyrgia antikeimenoy ypodoxhs DatagramSocket
                DatagramSocket dgramSocket = new DatagramSocket(1234);
                System.out.println("Server started");
                // Bhma 2o: Dhmioyrgia entamieyth
                byte[] buffer = new byte[256];
                // Bhma 3o: Dhmioyrgia antikeimenoy DatagramPacket gia lhpsi
                DatagramPacket inPacket= new DatagramPacket(buffer,buffer.length);
                // Bhma 4o: Lipsi paketoy
                dgramSocket.receive(inPacket);
                // Bhma 5o: Apodoxh ths aithshs
                InetAddress clientAddress = inPacket.getAddress();
                int clientPort = inPacket.getPort();
                // Bhma 6o: Anaktisi dedomenon apo to entamieyth
                String message = new String(inPacket.getData(),0,inPacket.getLength());
                System.out.println("Received request from " + clientAddress + ":" + clientPort);
                System.out.println("Received message from client:" + message);
                // Bhma 7o: Dhmioyrgia antikeimenoy DatagramPacket gia apostolh
                String response = "Good Bye!";
                DatagramPacket outPacket = new DatagramPacket(response.getBytes(),response.length(),client
                // Bhma 8o: Apostolh paketoy
                dgramSocket.send(outPacket);
                System.out.println("Message sent: Good Bye!");
                // Bhma 9o: Kleisimo ths ypodoxhs
                dgramSocket.close();
                System.out.println("Datagram socket closed");
        }
}
```

10.3.3 Υποδοχές Ρεύματος TCP

Η Java υποστηρίζει υποδοχές ρεύματος TCP κάτω από δύο κλάσεις υποδοχών, την κλάση `ServerSocket` και την κλάση `Socket` οι οποίες βρίσκονται στο πακέτο `java.net`. Η κλάση `ServerSocket` υλοποιεί **υποδοχές διακομιστών** (server sockets) για εγκατάσταση (ή αποδοχή) συνδέσεων και η κλάση `Socket` υλοποιεί **υποδοχές πελατών** (client sockets) για ανταλλαγή δεδομένων. Πρώτα θα ξεκινήσουμε να περιγράψουμε για την κλάση `Socket` και τα βήματα που απαιτούνται για την δημιουργία ενός προγράμματος πελάτη.

Κατασκευαστές `Socket`

Η κλάση `Socket` αφορά την υποδοχή δεδομένων και περιέχει πολλούς κατασκευαστές. Οι πιο κοινοί κατασκευαστές είναι οι εξής:

- `public Socket(InetAddress addr, int port) throws IOException.` Δημιουργεί μια υποδοχή ρεύματος και τη συνδέει με έναν απομακρυσμένο υπολογιστή στη διεύθυνση IP που ορίζεται από την παράμετρο `addr` και στην θύρα `port`. Σε περίπτωση που συμβεί σφάλμα εισόδου/εξόδου κατά τη δημιουργία της υποδοχής τότε δημιουργείται μια εξαίρεση τύπου `IOException`.

- `public Socket(String host, int port) throws UnknownHostException, IOException.` Δημιουργεί μια υποδοχή ρεύματος και τη συνδέει με έναν απομακρυσμένο υπολογιστή με συμβολικό όνομα `host` στην θύρα `port`. Σε περίπτωση αποτυχίας επίλυσης του ονόματος υπολογιστή `host` δημιουργείται μια εξαίρεση τύπου `UnknownHostException`, ενώ αν συμβεί σφάλμα εισόδου/εξόδου κατά τη δημιουργία της υποδοχής τότε δημιουργείται μια εξαίρεση τύπου `IOException`.

Μεθόδους `Socket`

Επίσης, η κλάση `Socket` διαθέτει μερικές μεθόδους για την χρησιμοποίηση των υποδοχών του πελάτη. Σε αυτή την ενότητα περιγράφουμε τρεις κοινές μεθόδους.

- `public InputStream getInputStream() throws IOException.` Επιστρέφει ένα αντικείμενο `InputStream` δηλαδή ένα ρεύμα εισόδου για την ανάγνωση δεδομένων από τη συγκεκριμένη υποδοχή.

- `public OutputStream getOutputStream() throws IOException.` Επιστρέφει ένα αντικείμενο `OutputStream` δηλαδή ένα ρεύμα εξόδου για την εγγραφή δεδομένων στη συγκεκριμένη υποδοχή.

- `public void close() throws IOException.` Κλείνει μια συγκεκριμένη υποδοχή.

Βήματα Δημιουργίας Προγράμματος Πελάτη

Παρακάτω περιγράφουμε τα βήματα που απαιτούνται για την δημιουργία ενός προγράμματος πελάτη.

1. Πρώτα ο πελάτης εγκαθιστά μια σύνδεση με τον απομακρυσμένο διακομιστή. Συγκεκριμένα, δημιουργούμε ένα αντικείμενο `Socket` μέσω του αντίστοιχου κατασκευαστή να απαιτεί

δύο ορίσματα: την διεύθυνση IP του διακομιστή και τον κατάλληλο αριθμό θύρας. Εδώ πρέπει να σημειώσουμε ότι ο αριθμός θύρας πρέπει να είναι το ίδιο και στα δύο προγράμματα δηλαδή του διακομιστή και του πελάτη. Για λόγους ευκολίας, υποθέτουμε ότι τα προγράμματα διακομιστή και πελάτη θα εκτελεστούν στον ίδιο υπολογιστή και για αυτό το λόγο θα κάνουμε κλήση της μεθόδου `getLocalHost` για να ανακτήσουμε την τοπική διεύθυνση IP. Για παράδειγμα,

```
Socket link = new Socket(InetAddress.getLocalHost(), 1234);
```

2. Έπειτα εγκαθιστούμε τα ρεύματα εισερχόμενων και εξερχόμενων δεδομένων τα οποία παρέχουν τη δυνατότητα στον πελάτη να επικοινωνεί με τον διακομιστή. Δηλαδή, δημιουργούμε αντικείμενα InputStream και OutputStream με τη βοήθεια των μεθόδων `getInputStream()` και `getOutputStream()` οι οποίες συσχετίζονται με την υποδοχή που έχει δημιουργηθεί από το βήμα 1 και ουσιαστικά αποτελούν τα ρεύματα εισερχόμενων και εξερχόμενων δεδομένων της υποδοχής. Στην συνέχεια δημιουργούμε ένα αντικείμενο `BufferedReader` το οποίο διευκολύνει την ανάγνωση χαρακτήρων από ένα ρεύμα εισόδου όπως αυτό που αντιπροσωπεύει το InputStream. Για παράδειγμα,

```
InputStream is = link.getInputStream();

BufferedReader in = new BufferedReader(new InputStreamReader(is));
```

Με παρόμοιο τρόπο δημιουργούμε ένα αντικείμενο `PrintWriter` το οποίο είναι το ανάλογο του `BufferedReader` αλλά χρησιμοποιείται για αποστολή δεδομένων. Για παράδειγμα,

```
OutputStream os = link.getOutputStream();

PrintWriter out = new PrintWriter(os,true);
```

3. Έπειτα πραγματοποιούμε λειτουργίες αποστολής και λήψης δεδομένων με τον διακομιστή. Εφόσον, έχουμε δημιουργήσει τα αντικείμενα `BufferedReader` και `PrintWriter`, η διαδικασία αποστολής και λήψης δεδομένων είναι απλή. Έτσι, χρησιμοποιούμε την μέθοδο `readLine()` του αντικειμένου `BuferredReader` για την λήψη δεδομένων και την μέθοδο `println()` του αντικειμένου `PrintWriter` για την αποστολή δεδομένων. Για παράδειγμα,

```
out.println("Sending data...");

String input = in.readLine();
```

4. Τέλος, τερματίζουμε την σύνδεση μετά την ολοκλήρωση της επικοινωνίας μεταξύ διακομιστή και πελάτη. Αυτή η λειτουργία υλοποιείται με την μέθοδο `close()`. Για παράδειγμα,

```
link.close();
```

Παρακάτω παρουσιάζεται ένα πρόγραμμα πελάτη που είναι παρόμοιο με το πρόγραμμα πελάτη που χρησιμοποιούσε υποδοχές UDP αλλά χρησιμοποιεί υποδοχές TCP.

```
import java.net.*;
import java.io.*;
public class SimpleClient {
        private static final int PORT = 1234;
        public static void main(String args[]) throws IOException {
                // Bhma 1o: Dhmioyrgia ypodoxhs reymatos kai syndesi me to diakomisth sthn thyra PORT
                Socket dataSocket = new Socket(InetAddress.getLocalHost(),PORT);
                System.out.println("Connection␣established");
                // Bhma 2o: Dhmioyrgia reymatos eiserxomenon dedomenon ths ypodoxhs dataSocket
                InputStream is = dataSocket.getInputStream();
                BufferedReader in = new BufferedReader(new InputStreamReader(is));
                // Bhma 2o: Dhmioyrgia reymatos exerxomenon dedomenon ths ypodoxhs dataSocket
                OutputStream os = dataSocket.getOutputStream();
                PrintWriter out = new PrintWriter(os, true);
                // Bhma 3o: Eggrafh mhnymatos Hello from Client! sto diakomisth
                out.println("Hello␣from␣Client!");
                System.out.println("Message␣sent:␣Hello␣from␣Client!");
                // Bhma 3o: Anagnosi mhnymatos apo to diakomisth
                String line = in.readLine();
                System.out.println("Received␣message␣from␣server:␣" + line);
                // Bhma 4o: Kleisimo ypodoxhs
                dataSocket.close();
                System.out.println("Data␣socket␣closed");
        }
}
```

Κατασκευαστές `ServerSocket`

Η κλάση `ServerSocket` αφορά την υποδοχή σύνδεσης και περιέχει πολλούς κατασκευαστές. Οι πιο κοινοί κατασκευαστές είναι οι εξής:

- `public ServerSocket(int port) throws IOException.` Δημιουργεί μια υποδοχή διακομιστή η οποία είναι συνδεδεμένη στη θύρα `port`. Αν η παράμετρος `port` έχει την τιμή 0, τότε η υποδοχή συνδέεται με οποιαδήποτε ελεύθερη θύρα. Επίσης, ο κατασκευαστής αυτός θέτει εξ ορισμού το μήκος της ουράς των εισερχόμενων αιτήσεων σε 50, αλλά υπάρχει ένας

εναλλακτικός κατασκευαστής που μας επιτρέπει να αλλάζουμε το μήκος της ουράς. Σε περίπτωση που συμβεί σφάλμα εισόδου/εξόδου κατά τη δημιουργία της υποδοχής τότε δημιουργείται μια εξαίρεση τύπου `IOException`.

- `public ServerSocket(int port, int numberOfClients) throws IOException`. Δημιουργεί μια υποδοχή διακομιστή η οποία είναι συνδεδεμένη στη θύρα `port` και θέτει το μήκος της ουράς των εισερχόμενων αιτήσεων σύνδεσης από πελάτες σε `numberOfClients`. Οι αιτήσεις σύνδεσης μπαίνουν σε μια ουρά και εφόσον η ουρά είναι γεμάτη κάθε νέα αίτηση για σύνδεση απορρίπτεται. Σε περίπτωση που συμβεί σφάλμα εισόδου/εξόδου κατά τη δημιουργία της υποδοχής τότε δημιουργείται μια εξαίρεση τύπου `IOException`.

Μεθόδους `ServerSocket`

Επίσης, η κλάση `ServerSocket` διαθέτει μερικές μεθόδους για την χρησιμοποίηση των υποδοχών διακομιστή. Σε αυτή την ενότητα περιγράφουμε δυο κοινές μεθόδους.

- `public Socket accept() throws IOException`. Η οποία αναμένει και αποδέχεται την αίτηση σύνδεση από κάποιο πελάτη σε μια νέα υποδοχή δεδομένων η οποία μπορεί να χρησιμοποιηθεί για σύνδεση και επικοινωνία με τον πελάτη. Επίσης, η μέθοδος αυτή είναι ανασταλτική μέχρι ότου φτάσει μια αίτηση σύνδεσης στη θύρα με την οποία συνδέθηκε η υποδοχή κατά τη δημιουργία της.

- `public void close() throws IOException`. Η οποία κλείνει μια υποδοχή του διακομιστή.

Βήματα Δημιουργίας Προγράμματος Διακομιστή

Παρόμοια διαδικασία ακολουθούμε για την δημιουργία ενός προγράμματος διακομιστή. Συγκεκριμένα, παρακάτω περιγράφουμε τα βήματα για την δημιουργία ενός προγράμματος διακομιστή.

1. Πρώτα δημιουργούμε ένα αντικείμενο τύπου `ServerSocket`. Ο κατασκευαστής `ServerSocket` απαιτεί ένα όρισμα τον αριθμό της θύρας. Παράδειγμα,

   ```
   ServerSocket servsock = new ServerSocket(1234);
   ```

   Σε αυτό το παράδειγμα, ο διακομιστής περιμένει μια σύνδεση από ένα πρόγραμμα πελάτη στη θύρα 1234.

2. Ο διακομιστής βρίσκεται σε κατάσταση αναμονής και περιμένει να δεχτεί μια αίτηση σύνδεσης από ένα πελάτη. Αυτό πραγματοποιείται με την κλήση της μεθόδου `accept()` της κλάσης `ServerSocket` η οποία επιστρέφει ένα αντικείμενο `Socket` όταν εγκαθίστανται μια σύνδεση. Για παράδειγμα

   ```
   Socket link = servsock.accept();
   ```

3. Στην συνέχεια εγκαθιστούμε τα ρεύματα εισερχόμενων και εξερχόμενων δεδομένων τα οποία παρέχουν τη δυνατότητα στον διακομιστή να επικοινωνεί με τον πελάτη. Η διαδικασία αυτή είναι ακριβώς ίδια όπως στο βήμα 2 του προγράμματος πελάτη.

4. Στην συνέχεια ορίζουμε λειτουργίες αποστολής και λήψης δεδομένων με το πελάτη. Στην πλευρά του διακομιστή το αντικείμενο `BufferedReader` θα λαμβάνει μηνύματα που απεστάλησαν από το αντικείμενο `PrintWriter` από την πλευρά του πελάτη. Αντίθετα, στην πλευρά του διακομιστή το αντικείμενο `PrintWriter` αποστέλονται μηνύματα που λαμβάνονται από το αντικείμενο `BufferedReader` στην πλευρά του πελάτη.

5. Τέλος, τερματίζουμε την σύνδεση. Το βήμα αυτό είναι ακριβώς ίδιο όπως στο βήμα 4 του προγράμματος πελάτη.

Παρακάτω παρουσιάζεται ένα πρόγραμμα διακομιστή που είναι παρόμοιο με το πρόγραμμα διακομιστή που χρησιμοποιούσε υποδοχές UDP αλλά χρησιμοποιεί υποδοχές TCP.

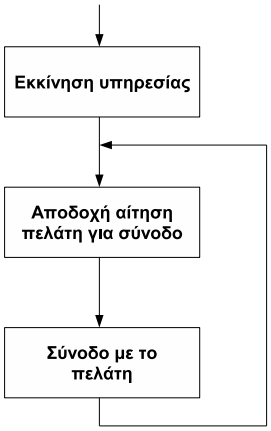
```
import java.net.*;
import java.io.*;
public class SimpleServer {
        private static final int PORT = 1234;
        public static void main(String args[]) throws IOException {
                // Bhma 1o: Dhmioyrgia ypodoxhs reymatos ServerSocket sthn thyra PORT
                ServerSocket connectionSocket = new ServerSocket(PORT);
                System.out.println("Server␣started");
                // Bhma 2o: Anamoni kai apodoxh aithshs syndeshs apo pelath
                Socket dataSocket = connectionSocket.accept();
                System.out.println("Received␣request␣from␣" + connectionSocket.getInetAddress());
                // Bhma 3o: Dhmioyrgia reymatos eiserxomenon dedomenon ths ypodoxhs dataSocket
                InputStream is = dataSocket.getInputStream();
                BufferedReader in = new BufferedReader(new InputStreamReader(is));
                // Bhma 3o: Dhmioyrgia reymatos exerxomenon dedomenon ths ypodoxhs dataSocket
                OutputStream os = dataSocket.getOutputStream();
                PrintWriter out = new PrintWriter(os, true);
                // Bhma 4o: Anagnosi mhnymatos apo to pelath
```

```
                String line = in.readLine();
                System.out.println("Received␣message␣from␣client:␣" + line);
                // Bhma 4o: Eggrafi mhnymatos Good Bye! sto pelath
                out.println("Good␣Bye!");
                System.out.println("Message␣sent:␣Good␣Bye!");
                // Bhma 5o: Kleisimo ths syndeshs pelaths
                dataSocket.close();
                System.out.println("Data␣socket␣closed");
                // Kleisimo ths ypodoxhs diakomisth
                connectionSocket.close();
                System.out.println("Connection␣socket␣closed");
        }
}
```

10.4 Επαναληπτικός και Συγχρονικός Διακομιστής

Η διεπαφή των υποδοχών μπορεί να χρησιμοποιηθεί για την επικοινωνία μεταξύ πελάτη και διακομιστή σε συνδυασμό είτε με το πρωτόκολλο TCP είτε με το πρωτόκολλο UDP ανάλογα με τις απαιτήσεις της εφαρμογής. Στα πλαίσια του μοντέλου πελάτη - διακομιστή, θα χρησιμοποιήσουμε το όρο **σύνοδο** (session) για να αναφερθούμε το σύνολο των αλληλεπιδράσεων ανάμεσα στο πελάτη και στο διακομιστή. Έτσι, η υπηρεσία που παρέχει ένας διακομιστής προσπελάζεται μερικές φορές παράλληλα από πολλούς πελάτες που θέλουν την υπηρεσία αυτή. Κάθε πελάτης, όταν εξυπηρετείται από το διακομιστή, εντάσσεται σε μια ξεχωριστή και ανεξάρτητη σύνοδο με το διακομιστή κατά την διάρκεια της οποίας ο πελάτης πραγματοποιεί ένα διάλογο ή επικοινωνία με το διακομιστή μέχρι ότου ο πελάτης να αποκτήσει το αποτέλεσμα της υπηρεσίας.

Η βασική ροή εκτέλεσης μιας διεργασίας διακομιστή παρουσιάζεται στο Σχήμα 10.6. Μόλις η διεργασία διακομιστής ξεκινήσει εκτελείται μέσα σε μια ατέρμονη επαναληπτική διαδικασία. Μέσα στην επαναληπτική διαδικασία, ο διακομιστής λαμβάνει τις αιτήσεις για σύνοδο από τους πελάτες και πραγματοποιεί μια σύνοδο υπηρεσίας για κάθε πελάτη. Η ροή που λαμβάνει στην σύνοδο μεταξύ πελάτη και διακομιστή είναι η εξής: ο διακομιστής λαμβάνει τις απαιτήσεις υπηρεσίας από το πελάτη, στην συνέχεια τις εκτελεί και τέλος να ενημερώνει το πελάτη για τα αποτελέσματα. Ένας διακομιστής γενικά παραμένει ενεργός για μεγάλα χρονικά διαστήματα, για να διεκπεραιώσει τις αιτήσεις πολλών πελατών για σύνοδο. Αν πολλοί πελάτες υποβάλουν ταυτόχρονα αιτήσεις στον διακομιστή, ο διακομιστής μπορεί να τις αντιμετωπίσει είτε επαναληπτικά είτε συγχρονικά.



Σχήμα 10.6: Ροή εκτέλεσης μια διεργασίας διακομιστή

Όπως είδαμε στο Σχήμα 10.6 δεν υπάρχει επικάλλυψη μεταξύ των συνόδων πελατών αφού ο διακομιστής είναι περιορισμένος να ανταλλάσει μηνύματα με ένα πελάτη του οποίου η σύνδεση του έχει αποδεκτεί. Τέτοιος διακομιστής ονομάζεται **επαναληπτικός διακομιστής** (iterative server) αφού επικοινωνεί μόνο με έναν πελάτη ανά πάσα στιγμή. Προφανώς σε αυτή την περίπτωση μία μόνο διεργασία επαρκεί για την διεκπεραίωση όλων των αιτήσεων πελατών. Ωστόσο σε έναν επαναληπτικό διακομιστή απαιτείται η δυνατότητα προσθήκης των αιτήσεων που φτάνουν σε μια ουρά, όταν ο διακομιστής είναι απασχολημένος, ώστε αυτές να διεκπεραιωθούν αργότερα. Συνεπώς, η προγραμματιστική υλοποίηση του επαναληπτικού διακομιστή είναι η προσθήκη μιας αρτέμονης επανάληψης `while` μεταξύ της αποδοχής της αίτησης πελάτη μέχρι το τερματισμό της επικοινωνίας με το πλεάτη. Παρακάτω παρουσιάζεται ένα τροποποιημένο πρόγραμμα επαναληπτικού διακομιστή σύμφωνα με το προηγούμενο πρόγραμμα διακομιστή με υποδοχές TCP. Επίσης, σημειώνουμε ότι καμία αλλαγή δεν πραγματοποιείται στο κώδικα του πελάτη.

```
import java.net.*;
import java.io.*;
public class SimpleIterServer {
        private static final int PORT = 1234;
        public static void main(String args[]) throws IOException {
                int connectionCount = 0;
                // Bhma 1o: Dhmioyrgia ypodoxhs reymatos ServerSocket sthn thyra PORT
                ServerSocket connectionSocket = new ServerSocket(PORT);
                System.out.println("Server␣started");
                while (true) { // Anamoni gia pelates syndeshs
```

```
                    // Bhma 2o: Anamoni kai apodoxh aithshs syndeshs apo pelath
                    Socket dataSocket = connectionSocket.accept();
                    connectionCount++;
                    System.out.println("Received " + connectionCount + " request from " + connectionSo
                    // Bhma 3o: Dhmioyrgia reymatos eiserxomenon dedomenon ths ypodoxhs dataSocket
                    InputStream is = dataSocket.getInputStream();
                    BufferedReader in = new BufferedReader(new InputStreamReader(is));
                    // Bhma 3o: Dhmioyrgia reymatos exerxomenon dedomenon ths ypodoxhs dataSocket
                    OutputStream os = dataSocket.getOutputStream();
                    PrintWriter out = new PrintWriter(os, true);
                    // Bhma 4o: Anagnosi mhnymatos apo to pelath
                    String line = in.readLine();
                    System.out.println("Received message from client: " + line);
                    // Bhma 4o: Eggrafi mhnymatos Good Bye! sto pelath
                    out.println("Good Bye!");
                    System.out.println("Message sent: Good Bye!");
                    // Bhma 5o: Kleisimo ths syndeshs pelaths
                    dataSocket.close();
                    System.out.println("Data socket closed");
                }
            }
}
```

Σε περίπτωση αίτησης ενός πελάτη από έναν επαναληπτικό διακομιστή ίσως ο πελάτης μπεί σε μια ουρά αναμονής μέχρι ο διακομιστής να εξυπηρετήσει όλους τους προηγουμένους πελάτες. Αυτό ίσως έχει σαν αποτέλεσμα ο χρόνος αναμονής του τελευταίου πελάτη στην ουρά να είναι σημαντικός αν η διάρκεια της συνόδου κάθε πελάτη είναι μεγάλη και το πλήθος των πελατών στην ουρά είναι επίσης μεγάλος. Ο περιορισμός αυτός δεν είναι ρεαλιστικός στις περισσότερες κατανεμημένες εφαρμογές. Μια λύση σε αυτό το περιορισμό είναι ο **συγχρονικός διακομιστής** (concurrent server). Ο συγχρονικός διακομιστής επικοινωνεί ταυτόχρονα με πολλούς πελάτες, δηλαδή έχει τη δυνατότητα να διατηρεί πολλές ενεργές συνόδους πελατών ταυτόχρονα. Για την υλοποίηση του συγχρονικού διακομιστή, είναι ότι για κάθε νέα αίτηση που λαμβάνει μέσω μιας υποδοχής να δημιουργήσει ένα αντίγραφο της υποδοχής αυτής. Με αυτό τον τρόπο, ο διακομιστής πραγματοποιεί σύνοδο με το πελάτη χρησιμοποιώντας το αντίγραφο της υποδοχής, ενώ συγχρόνως μπορεί να συνεχίσει να δέχεται επομένες αιτήσεις πελατών από την αρχική υποδοχή. Επομένως, ο διακομιστής δημιουργεί για κάθε αίτηση που λαμβάνει ένα νέο νήμα, με το νέο νήμα να χειρίζεται τη σύνοδο με το πελάτη και το νήμα τερματίζεται μετά την ολοκλήρωση της συνόδου.

Συνεπώς, η προγραμματιστική υλοποίηση του συγχρονικού διακομιστή περιλαμβάνει δύο

βήματα: Πρώτον, το βασικό νήμα ελέγχου (δηλαδή η μέθοδος `main()`) δημιουργεί ξεχωριστά νήματα για κάθε εισερχόμενη αίτηση πελάτη και δεύτερον το νέο νήμα που δημιουργείται χειρίζεται την αλληλεπίδραση μεταξύ πελάτη - διακομιστή (ή σύνοδο) μέσω της μεθόδου `run()` του νήματος. Πρέπει να σημειωθεί ότι για κάθε νέο νήμα που δημιουργείται πρέπει να περάσουμε ως παράμετρο την υποδοχή του πελάτη. Τέλος, τα παραπάνω δύο βήματα βρίσκονται μέσα σε μια ατέρμονη επαναληπτική δομή. Παρακάτω παρουσιάζονται δύο προγράμματα Java, ένα για το συγχρονικό διακομιστή και ένα για την κλάση νήματος που υλοποιεί τη σύνοδο με το πελάτη. Επίσης, σημειώνουμε ότι καμία αλλαγή δεν πραγματοποιείται στο κώδικα του πελάτη.

```
import java.net.*;
import java.io.*;
public class SimpleConcurServer {
        private static final int PORT = 1234;
        public static void main(String args[]) throws IOException {
                int connectionCount = 0;
                // Bhma 1o: Dhmioyrgia ypodoxhs reymatos ServerSocket sthn thyra PORT
                ServerSocket connectionSocket = new ServerSocket(PORT);
                System.out.println("Server␣started");
                while (true) { // Anamoni gia pelates syndeshs
                        // Bhma 2o: Anamoni kai apodoxh aithshs syndeshs apo pelath
                        Socket dataSocket = connectionSocket.accept();
                        connectionCount++;
                        System.out.println("Received␣" + connectionCount + "␣request␣from␣" + connectionSo
                        // Dhmioyrgia nhmatos gia kathe syndesh pelath
                        ClientThread cthread = new ClientThread(dataSocket);
                        cthread.start();
                }
        }
}

import java.io.*;
import java.net.*;
class ClientThread extends Thread
{
        private Socket dataSocket;
        BufferedReader in;
        PrintWriter out;
        public ClientThread(Socket socket)
        {
        dataSocket = socket;
        try
        {
                // Bhma 3o: Dhmioyrgia reymatos eiserxomenon dedomenon ths ypodoxhs dataSocket
                InputStream is = dataSocket.getInputStream();
```

```
                        in = new BufferedReader(new InputStreamReader(is));
                        // Bhma 3o: Dhmioyrgia reymatos exerxomenon dedomenon ths ypodoxhs dataSocket
                        OutputStream os = dataSocket.getOutputStream();
                        out = new PrintWriter(os,true);
        }
        catch (IOException e)
        {
                        System.out.println("Error " + e);
        }
    }

        public void run()
        {
                try
                {
                        // Bhma 4o: Anagnosi mhnymatos apo to pelath
                String msg = in.readLine();
                System.out.println("Received message from client:" + msg);
                // Bhma 4o: Eggrafi mhnymatos Good Bye! sto pelath
                out.println("Good Bye!");
                System.out.println("Message sent: Good Bye!");
                // Bhma 5o: Kleisimo ths syndeshs pelaths
                dataSocket.close();
                System.out.println("Data socket closed");
                }
                catch (IOException e)
                {
                System.out.println("Error " + e);
                }
        }
}
```

10.5 Πρωτόκολλα Αίτησης - Απάντησης

Στις σύγχρονες κατανεμημένες εφαρμογές, η ανταλλαγή μηνυμάτων μεταξύ πελάτη και διακομιστή υλοποιείται με υποδοχές είτε UDP είτε TCP η οποία βασίζεται στην έννοια ενός κοινού πρωτοκόλλου. Πρωτόκολλο είναι ένα σύνολο κανόνων που πρέπει ακολουθήσουν ο πελάτης και ο διακομιστής κατά την διάρκεια της συνόδου. Τέτοιοι κανόνες περιλαμβάνουν συνήθως προδιαγραφές σε τρία ζητήματα όπως ο τρόπος εντοπισμού της υπηρεσίας που φιλοξενείται σε ένα διακομιστή, η σειρά μηνυμάτων της διαδιεργασιακής επικοινωνίας και η αναπαράσταση και ερμηνεία των μηνυμάτων. Ο μηχανισμός που χρησιμοποιείται ώστε η διεργασία πελάτη να καλέσει

μια υπηρεσία που βρίσκεται σε ένα διακομιστή είναι η χρήση της διεύθυνσης IP του υπολογιστή διακομιστή και το αριθμό θύρας UDP ή TCP της διεργασίας στην οποία αναμένει αιτήσεις ο διακομιστής. Αυτός ο μηχανισμός χρησιμοποιείται για τις υπηρεσίες Διαδικτύου όπου κάθε υπηρεσία Διαδικτύου αναγνωρίζεται από ένα ευρέως γνωστό αριθμό θύρας. Για παράδειγμα, οι διακομιστές Ιστού συνήθως χρησιμοποιούν την θύρα 80 του TCP.

Η σειρά μηνυμάτων της διαδιεργασιακής επικοινωνίας στο μοντέλο πελάτη - διακομιστή ακολουθεί το **πρωτόκολλο αίτησης - απάντησης** (request - reply protocol). Σύμφωνα με αυτό το πρωτόκολλο, κατά την διάρκεια της συνόδου ένας πελάτης στέλνει ένα μήνυμα αίτησης (που απαιτεί μια υπηρεσία) στον διακομιστή και περιμένει ένα μήνυμα απάντησης (το αποτέλεσμα της υπηρεσίας) από το διακομιστή. Στην συνέχεια, ο ίδιος πελάτης μπορεί να στείλει επόμενο μήνυμα αίτησης και να περιμένει την απάντηση. Το πρωτόκολλο αυτό επεξεργάζεται σε μια επανάληψη μηνυμάτων αίτησης - απάντησης μέχρι να ολοκληρωθεί η σύνοδος μεταξύ πελάτη και διακομιστή. Είναι γνωστό ότι τα περισσότερα πρωτόκολλα Διαδικτύου (όπως HTTP, SMTP, FTP κλπ) ακολουθούν το πρωτόκολλο αίτησης - απάντησης.

Το επόμενο ζήτημα είναι η επεξεργασία των μηνυμάτων που ανταλλάζονται σε κάθε σύνοδο. Για την εύκολη επεξεργασία μηνυμάτων, θα πρέπει να υπάρχει μια σειρά μηνυμάτων στην διαδιεργασιακή επικοινωνία μεταξύ πελάτη και διακομιστή. Επίσης, τα μηνύματα αίτησης και απάντησης ακολουθούν κάποιους συντακτικούς και σημασιολογικούς κανόνες. Οι συνταντικοί κανόνες αναφέρονται στην δομή και την γραμματική των μηνυμάτων ενώ οι σημασιολογικοί κανόνες αναφέρονται στην ερμηνεία των μηνυμάτων. Τέλος, κάθε ένα από τα μηνύματα αίτησης ή απάντησης που λαμβάνει υλοποιεί ορισμένες ενέργειες σε κάθε πλευρά του μοντέλου πελάτη - διακομιστή.

Τέλος, η επιλογή της αναπαράστασης μηνυμάτων εξαρτάται από την φύση και τις ανάγκες του πρωτοκόλλου. Συνήθως, τα πρωτόκολλα είναι βασισμένα σε κείμενο που σημαίνει ότι τα μηνύματα αίτησης και απάντησης είναι σε μορφή αλφαριθμητικών ASCII. Το πλεονέκτημα σε αυτή την μορφή αναπαράστασης μηνυμάτων είναι ότι τα μηνύματα εύκολα επεξεργάζονται από προγράμματα και επίσης είναι ευανάγνωστα από ανθρώπους.

Παρακάτω θα περιγράψουμε δύο παραδείγματα πρωτοκόλλων αίτησης - απάντησης, το ένα πρωτόκολλο συσχετίζεται με το ενδιάμεσο λογισμικό και το άλλο αφορά σε επίπεδο εφαρμογής.

10.5.1 Πρωτόκολλο HTTP

Το HTTP (HyperText Transfer Protocol) είναι ένα πρωτόκολλο Διαδικτύου που χρησιμοποιεί τις υποδοχές TCP ως ένας μηχανισμός μεταφοράς εγγράφων υπερκειμένου ανάμεσα σε έναν πελάτη και ένα διακομιστή ιστού. Συνεπώς, το HTTP είναι ένα παράδειγμα πρωτοκόλλου αίτησης - απάντησης βασισμένο σε κείμενο. Όταν ένας πελάτης HTTP (συνήθως είναι ένας περιηγητής Ιστού) απαιτεί ένα πόρο (π.χ. ένα αρχείο html) χρησιμοποιώντας το URL, τότε εγκαθιστά μια σύνδεση TCP με ένα διακομιστή HTTP (συνήθως είναι ένας διακομιστή Ιστού αλλά μπορεί να είναι ένας διακομιστής πληρεξούσιος (proxy server)) στην εξ ορισμού θύρα 80. Στην συνέχεια, ο πελάτης στέλνει ένα **μήνυμα αίτησης** (request message) για ανάκτηση ενός πόρου στο διακομιστή. Ο διακομιστής επεξεργάζεται την αίτηση του πελάτη και στέλνει ένα **μήνυμα απάντησης** (response message) που περιέχει τα περιεχόμενα του πόρου (π.χ. του αρχείου html).

Από την λειτουργία του πρωτοκόλλου HTTP προκύπτει ότι αποτελείται από δύο σύνολα μηνυμάτων: το σύνολο μηνυμάτων αίτησης που αποστέλλονται από τον πελάτη που εκτελεί τον περιηγητή και το σύνολο μηνυμάτων απάντησης του διακομιστή. Η δομή του μηνύματος αίτησης HTTP αποτελείται από δύο μέρη:

- μια γραμμή αίτησης που είναι της μορφής: <εντολή HTTP> <URL ενός πόρου> <έκδοση του HTTP>
- τα πεδία επικεφαλίδας που κάθε πεδίο αποτελείται από μια λέξη κλειδί ακολουθούμενη από άνω κάτω τελεία και ένα όρισμα που αντιστοιχεί στην τιμή της λέξης κλειδί. Δηλαδή κάθε πεδίο επικεφαλίδας είναι της μορφής: <λέξη-κλειδί>:<τιμή>

Το HTTP διαθέτει εντολές, οι οποίες είναι επίσης γνωστές ως **μέθοδοι** (methods) για την υποστήριξη διαφόρων τύπων αιτήσεων που μπορεί να στείλει μια εφαρμογή πελάτη στο διακομιστή Ιστού. Στον πίνακα 10.1 παρουσιάζονται μερικές από αυτές τις εντολές, οι πιο συνηθισμένες. Η πιο συνηθισμένη εντολή αίτησης στον Παγκόσμιο Ιστό είναι η GET για να ανακτήσουμε ένα έγγραφο υπερκειμένου ή άλλο πόρο από ένα διακομιστή Ιστού. Μετά την γραμμή αίτηση ενός μηνύματος HTTP ακολουθούν τα πεδία επικεφαλίδας. Στον πίνακα 10.2 φαίνονται τα πιο συνηθισμένα πεδία επικεφαλίδας σε μια αίτηση HTTP.

Έτσι για παράδειγμα, όταν ένα πρόγραμμα πελάτη όπως ο περιηγητής Ιστού θέλει να ανακ-

Εντολή	Σημασία
GET	Ανάκτηση ενός πόρου από το διακομιστή και το επιστρέφει στον πελάτη
HEAD	Ίδια με τη εντολή GET, αλλά επιστρέφει μόνο πληροφορία για έναν πόρο και όχι το περιεχόμενο
POST	Στέλνει δεδομένα από τον πελάτη στον διακομιστή για εκτέλεση εφαρμογών διακομιστή

Πίνακας 10.1: Εντολές του πρωτοκόλλου HTTP

Πεδίο επικεφαλίδας	Σημασία
User agent	Τύπος του προγράμματος πελάτη HTTP που στέλνει την αίτηση
Connection	Τύπος σύνδεσης
Host	Όνομα υπολογιστή - διακομιστή που βρίσκεται ο πόρος
Accept	Τύπος του περιεχομένου ενός πόρου δηλαδή ο τύπος MIME

Πίνακας 10.2: Πεδία επικεφαλίδας του πρωτοκόλλου HTTP

τήσει ένα αρχείο `index.hmtl` από τον διακομιστή ιστού `eos.uom.gr`, τότε ο περιηγητής θα στείλει το παρακάτω μήνυμα αίτησης GET:

```
GET /index.html HTTP/1.1
User-Agent:  Mozilla/1.5
Connection:  Keep-Alive
Host:  eos.uom.gr
Accept:  text/html
```

Στην άλλη πλευρά του πρωτοκόλλου HTTP είναι το σύνολο των μηνυμάτων απάντησης του διακομιστή στον πελάτη ο οποίος έστειλε μήνυμα αίτησης. Έτσι, η δομή του μηνύματος απάντησης αποτελείται από τρία μέρη:

- μια γραμμή κατάστασης που είναι της μορφής: <έκδοση HTTP> <κωδικός κατάστασης> <μήνυμα κειμένου>
- τα πεδία επικεφαλίδας απάντησης που έχει την ίδια μορφή όπως στα πεδία επικεφαλίδας αίτησης ενός πελάτη
- το σώμα μηνύματος που περιέχει τα περιεχόμενα του πόρου (ή αρχείου) που ζητήθηκε

Η γραμμή κατάστασης δείχνει αν η αίτηση του πελάτη εκτελέστηκε επιτυχώς ή αν προκάλεσε κάποιο σφάλμα. Αυτή η κατάσταση προσδιορίζεται από τον κωδικό κατάστασης που είναι ένας τριψήφιος δεκαδικός αριθμός. Το πρώτο ψηφίο αντιστοιχεί σε μια γενική κατάσταση σφάλματος και τα άλλα δυο ψηφία προσδιορίζουν τη συγκεκριμένη κατάσταση σφάλματος. Με βάση το πρώτο ψηφίο οι κωδικοί κατάστασης χωρίζονται σε πέντε γενικές κατηγορίες όπως παρουσιάζεται στον Πίνακα 10.3.

Κατηγορία κατάστασης	Όνομα κατηγορίας σφάλματος
1xx	Πληροφοριακός
2xx	Επιτυχία
3xx	Ανακατεύθυνση
4xx	Σφάλμα πελάτη
5xx	Σφάλμα διακομιστή

Πίνακας 10.3: Κωδικοί κατάστασης του πρωτοκόλλου HTTP

Ο διακομιστής προσθέτει ορισμένα πεδία επικεφαλίδας στο μήνυμα απάντησης και τα πεδία αυτά δεν παρουσιάζονται πάντα σε κάθε μήνυμα. Όμως, υπάρχουν μερικά πεδία επικεφαλίδας που εμφανίζονται πάντα όπως τα πεδία `Content-Type` και `Last Modified`. Έτσι, σε αυτή την ενότητα είναι δύσκολο να καλύψουμε όλα τα πεδία επικεφαλίδας και για το λόγο αυτό στον Πίνακα 10.4 παρουσιάζονται τα πιο συνηθισμένα πεδία επικεφαλίδας που συναντάμε στο HTTP. Αξίζει να σημειώσουμε ότι ο τύπος περιεχομένου MIME δηλώνει στον πελάτη το περιεχόμενου του αρχείου. Για παράδειγμα, ο τύπος MIME text/html δηλώνει κείμενο σε μορφή html, ο

Πεδίο επικεφαλίδας	Σημασία
Date	Ημέρα και χρόνος επεξεργασίας της αίτησης
Server	Τύπος και έκδοση του διακομιστή που επεξεργάζεται την αίτηση
Content-Type	Τύπος περιεχομένου MIME του σώματος μηνύματος
Content-Length	Μέγεθος σε bytes του σώματος μηνύματος
Last-Modified	Πότε ενημερώθηκε για τελευταία φορά ο πόρος

Πίνακας 10.4: Πεδία επικεφαλίδας του πρωτοκόλλου HTTP

τύπος text/plain δηλώνει κείμενο σε μορφή απλού κειμένου, ο image/gif δηλώνει μια εικόνα σε μορφή gif, ο τύπος image/jpeg δηλώνει εικόνα σε μορφή jpg κλπ. Οι τύποι MIME αρχικά χρησιμοποιούνταν για την επισύναψη σε μηνύματα ηλεκτρονικού ταχυδρομείου περιεχομένου που δεν ήταν απλό κείμενο.

Τέλος, το μήνυμα απάντησης τελειώνει με το σώμα μηνύματος που περιέχει το πραγματικό περιεχόμενο του πόρου. Αυτού προηγείται μια κενή γραμμή. Έτσι, αν ο πόρος που ζητήθηκε είναι ένα αρχείο html τότε το περιεχόμενο του μηνύματος είναι ο κώδικας html. Τα μετα-δεδομένα σχετικά με το σώμα μηνύματος προσδιορίζονται από τα πεδία επικεφαλίδας όπως `Content-Type` και `Content-Length`. Αφού ο περιηγητής λάβει την γραμμή κατάστασης, τα πεδία επικεφαλίδας και τον κώδικα html θα εμφανίσει στην οθόνη του χρήστη το περιεχόμενο μορφοποιημένο, αφού πρώτα ερμηνεύσει τα επιθέματα της html που υπάρχουν στο κείμενο.

Έτσι για παράδειγμα, όταν ο διακομιστής Ιστού λάβει ένα μήνυμα αίτησης από το πελάτη που αφορά την ανάκτηση ενός αρχείου html όπως είδαμε στο προηγούμενο παράδειγμα το οποίο υπάρχει στα αρχεία του διακομιστή Ιστού, τότε ο διακομιστής θα στείλει το παρακάτω μήνυμα απάντησης:

```
HTTP/1.1 200 OK
Date:  Thu, 22 July 1998 18:40:55 GMT
Server:  Apache 1.3.5 (Unix) PHP/3.0.6
Last-Modified:  Mon, 19 July 1997 16:03:22 GMT
Content-Type:  text/html
Content-Length:  12987
(κενή γραμμή)
<html>
....
</html>
```

Παρακάτω παρουσιάζουμε ένα απλό πρόγραμμα πελάτη HTTP χρησιμοποιώντας τις υποδοχές TCP ώστε να στείλει ένα μήνυμα αίτησης GET στο διακομιστή Ιστού (π.χ. στο διακομιστή Google) σύμφωνα με το πρωτόκολλο HTTP και στην συνέχεια το πρόγραμμα εμφανίζει την απάντηση του διακομιστή γραμμή προς γραμμή.

```
import java.net.*;
import java.io.*;
```

```
public class HTTPClient {
        private static final int PORT = 80;
        public static void main(String args[]) throws IOException {
                // Bhma 1o: Dhmioyrgia ypodoxhs reymatos kai syndesi me to diakomisth sthn thyra PORT
                Socket dataSocket = new Socket("www.google.com",PORT);
                System.out.println("Connection␣established");
                String request = "GET␣index.html␣HTTP/1.1\r\n";
                // Bhma 2o: Dhmioyrgia reymatos eiserxomenon dedomenon ths ypodoxhs dataSocket
                InputStream is = dataSocket.getInputStream();
                BufferedReader in = new BufferedReader(new InputStreamReader(is));
                // Bhma 2o: Dhmioyrgia reymatos exerxomenon dedomenon ths ypodoxhs dataSocket
                OutputStream os = dataSocket.getOutputStream();
                PrintWriter out = new PrintWriter(os, true);
                // Bhma 3o: Apostolh mhnymatos aithshs sto diakomisth
                out.println(request);
                // Bhma 3o: Anagnosi mhnymatos apantisi apo to diakomisth
                String response;
                response = in.readLine();
                while (response != null){
                        System.out.println(response);
                        response = in.readLine();
                }
                // Bhma 4o: Kleisimo ypodoxhs
                dataSocket.close();
                System.out.println("Data␣socket␣closed");
        }
}
```

Σε σχέση με τους σύγχρονους περιηγητές Ιστού, το παραπάνω απλό πρόγραμμα περιηγητή δεν υλοποιεί την ερμηνεία της απάντησης (δηλαδή του αρχείου html) και εμφανίζει το κείμενο όπως το λαμβάνει.

10.5.2 Πρωτόκολλο Εφαρμογής

Σε την ενότητα για να περιγράψουμε την έννοια του πρωτόκολλου εφαρμογής θα χρησιμοποιήσουμε ένα απλό παράδειγμα: εκείνου του διακομιστή που θα παρέχει δύο δικτυακές υπηρεσίες, η μία να μετατρέπει ένα αλφαριθμητικό σε πεζά γράμματα και η άλλη να μετατρέπει ένα αλφαριθμητικό σε κεφαλαία γράμματα, όπου ο πελάτης και διακομιστής θα επικοινωνούν χρησιμοποιώντας μια σειρά μηνυμάτων τα οποία περιέχουν ενδείξεις για την απαιτούμενη λειτουργικότητα και για τα δεδομένα που συσχετίζονται με τη λειτουργία που πρόκειται να εκτελεστεί.

Για παράδειγμα, αν ο πελάτης επιθυμεί να μετατρέψει ένα αλφαριθμητικό σε πεζά γράμμα-

τα θα στείλει ένα μήνυμα το οποίο θα αποτελείται από το γράμμα L ακολουθούμενο από το αλφαριθμητικό, δηλαδή της μορφής

```
L <string>
```

ή αν ο πελάτης επιθυμεί να μετατρέψει το αλφαριθμητικό σε κεφαλαία θα στείλει ένα μήνυμα της μορφής

```
U <string>
```

όπου τα L και U καθορίζουν το γεγονός ότι ο πελάτης επιθυμεί να μετατρέψει σε πεζά και κεφαλαία γράμματα για ένα συγκεκριμένο αλφαριθμητικό <string> αντίστοιχα. Ο διακομιστής θα επεξεργαστεί το μήνυμα και θα εκτελέσει την κατάλληλη υπηρεσία μετατροπής αλφαριθμητικού ώστε να επιστρέψει το αλφαριθμητικό μετά την μετατροπή. Το μήνυμα που επιστρέφεται μπορεί να έχει τη μορφή

```
R <string>
```

όπου το γράμμα R υποδεικνύει ότι έχει αποσταλεί μια επιτυχή απάντηση που συνοδεύει το αλφαριθμητικό που μετατράπηκε. Στην περίπτωση που ο διακομιστής επεξεργάζεται ένα λανθασμένο μήνυμα (δηλαδή, δεν αναγνωρίζει ποια υπηρεσία θα εκτελέσει) τότε θα στείλει ένα μήνυμα που θα αποτελείται μόνο από το γράμμα E που υποδεικνεύει κάποιο σφάλμα. Στην συνέχεια, ο πελάτης θα επεξεργαστεί το μήνυμα που στέλνει ο διακομιστής και θα εμφανίζει το κατάλληλο μήνυμα, δηλαδή το αλφαριθμητικό αν έχει μετατραπεί επιτυχώς ή ένα μήνυμα λάθους.

Πρέπει να σημειώσουμε ότι οποιοδήποτε μηχανισμός και να χρησιμοποιείται για την επικοινωνία μεταξύ των πελατών και των διακομιστών οπωσδήποτε κάποιο πρωτόκολλο θα αναμειγνύεται. Παρακάτω παρουσιάζουμε τα δύο πρόγραμματα πελάτη και συγχρονικού διακομιστή που υλοποιούν το πρωτόκκολλο εφαρμογής που περιγράψαμε προηγουμένως.

```
import java.net.*;
import java.io.*;
public class ChangeCaseClient {
        private static final int PORT = 1234;
        public static void main(String args[]) throws IOException {
                // Dhmioyrgia ypodoxhs reymatos kai syndesi me to diakomisth sthn thyra PORT
                Socket dataSocket = new Socket(InetAddress.getLocalHost(),PORT);
                // Dhmioyrgia reymatos eiserxomenon dedomenon ths ypodoxhs dataSocket
                InputStream is = dataSocket.getInputStream();
                BufferedReader in = new BufferedReader(new InputStreamReader(is));
                // Dhmioyrgia reymatos exerxomenon dedomenon ths ypodoxhs dataSocket
                OutputStream os = dataSocket.getOutputStream();
                PrintWriter out = new PrintWriter(os, true);
```

```
                // Dhmioyrgia reymatos gia to pliktrologio
                BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
                // Anagnosi enos alfarithmhtikoy apo to pliktrologio
                System.out.print("Give␣a␣string:");
                String stringclient = input.readLine();
                // Apostolh aithshs gia metatroph se kefalaia grammata
                out.println("L" + stringclient);
                // Anagnosi apanthshs apo to diakomhsths
                String reply = in.readLine();
                // Epexergasia toy mhnymatos apanthshs
                if (reply.charAt(0) == 'E')
                        System.out.println("Error!!!");
                else
                        System.out.println("Server␣Reply:␣" + reply);
                // Kleisimo ypodoxhs
                dataSocket.close();
        }
}

import java.net.*;
import java.io.*;
public class ChangeCaseServer {
        private static final int PORT = 1234;
        public static void main(String args[]) throws IOException {
                // Dhmioyrgia ypodoxhs reymatos ServerSocket sthn thyra PORT
                ServerSocket connectionSocket = new ServerSocket(PORT);
                while (true) {
                        // Anamoni kai apodoxh aithshs syndeshs apo pelath
                        Socket dataSocket = connectionSocket.accept();
                        // Dhmioyrgia nhmatos gia kathe syndesh pelath
                        ClientThread cthread = new ClientThread(dataSocket);
                        cthread.start();
                }
        }
}

import java.io.*;
import java.net.*;
class ClientThread extends Thread
{
        private Socket dataSocket;
        BufferedReader in;
        PrintWriter out;
        public ClientThread(Socket socket)
        {
        dataSocket = socket;
        try
```

```
        {
                // Dhmioyrgia reymatos eiserxomenon dedomenon ths ypodoxhs dataSocket
                InputStream is = dataSocket.getInputStream();
                        in = new BufferedReader(new InputStreamReader(is));
                        // Dhmioyrgia reymatos exerxomenon dedomenon ths ypodoxhs dataSocket
                        OutputStream os = dataSocket.getOutputStream();
                        out = new PrintWriter(os,true);
        }
        catch (IOException e)
        {
                        System.out.println("Error␣" + e);
        }
    }

        public void run()
        {
                try
                {
                        // Anagnosi mhnymatos aithshs apo to pelath
                        String clientreq = in.readLine();
                        // Epexergasia ths aithshs gia thn epilogh ektelesh mias katallhlhs yphresias
                        switch (clientreq.charAt(0)) {
                                case 'L': { // yphresia metatrophs alfarithmhtikoy se peza
                                                        // Pairnoume to alfarithmhtiko
                                                        String clientstring = clientreq.substring(1, client
                                                        // Apostoli mhnymatos apantishs
                                                        out.println("R" + clientstring.toLowerCase());
                                                        break;
                                }
                                case 'U': { // yphresia metatrophs alfarithmhtikoy se kefalaia
                                                        // Pairnoume to alfarithmhtiko
                                                        String clientstring = clientreq.substring(1, client
                                                        // Apostoli mhnymatos apantishs
                                                        out.println("R" + clientstring.toUpperCase());
                                                        break;
                                }
                                default: {
                                                        // Apostoli mhnymatos lathoys se periptosi lathasm
                                                        out.println("E");
                                                        break;
                                }
                        }
                        // Kleisimo ths syndeshs pelaths
                        dataSocket.close();
                }
                catch (IOException e)
```

```
                {
                System.out.println("Error␣" + e);
                }
        }
}
```

Παράδειγμα 2

Σε αυτό το παράδειγμα, ο διακομιστής δέχεται μηνύματα αλφαριθμητικών από το πελάτη, βρίσκει το πλήθος των μηνυμάτων και επιστρέφει πίσω στο πελάτη τα μηνύματα με αρίθμηση. Το βασικό πρωτόκολλο για την υπηρεσία αυτή είναι ότι ο πελάτης και ο διακομιστής πρέπει να εναλλάσονται ανάμεσα στην αποστολή και λήψη μηνυμάτων. Τέλος, ο διακομιστής τερματίζει την σύνδεση με τον πελάτη όταν δεχτεί το μήνυμα CLOSE.

```
import java.net.*; import java.io.*; public class EchoClient2 {
    private static final int PORT = 1234;
    public static void main(String args[])
    {
        try
        {
            InetAddress host = InetAddress.getLocalHost();
            Socket link = new Socket(host,PORT);
            System.out.println("Connection established");
            InputStream is = link.getInputStream();
            BufferedReader in = new BufferedReader(new InputStreamReader(is));
            BufferedReader user = new BufferedReader(new InputStreamReader(System.in));
            OutputStream os = link.getOutputStream();
            PrintWriter out = new PrintWriter(os,true);
            System.out.print("Enter message:");
            String msg=user.readLine();
            while(!msg.equals("CLOSE")) {
                out.println(msg);
                System.out.println("Received message from server: " + in.readLine());
```

```
                System.out.print("Enter message:");
                msg=user.readLine();
            }
            link.close();
            System.out.println("Closing connection");
        }
        catch (IOException e)
        {
            System.out.println("Error " + e);
        }
    }
}
```

```
import java.net.*; import java.io.*; public class EchoServer2 {
    private static final int PORT = 1234;
    public static void main(String args[])
    {
        System.out.println("Server started");
        try
        {
            ServerSocket servsock = new ServerSocket(PORT);
            Socket link = servsock.accept();
            System.out.println("Received request from " + link.getInetAddress() + ":" +
            InputStream is = link.getInputStream();
            BufferedReader in = new BufferedReader(new InputStreamReader(is));
            OutputStream os = link.getOutputStream();
            PrintWriter out = new PrintWriter(os,true);
            int num_msgs=0;
            String msg=in.readLine();
            while(!msg.equals("CLOSE")) {
                System.out.println("Received message from client: " + msg);
```

```
                num_msgs++;
                out.println("Message " + num_msgs);
                msg=in.readLine();
            }
            link.close();
            System.out.println("Closing connection");
        }
        catch (IOException e)
        {
            System.out.println("Error " + e);
        }
    }
}
```

10.6 Ομαδική Επικοινωνία

Στις προηγουμένες ενότητες είδαμε την διαδιεργασιακή επικοινωνία ως μια ανταλλαγή δεδομένων ανάμεσα στις δύο διεργασίες. Συγκεκριμένα, μια διεργασία αποστολέα στέλνει δεδομένα σε μια διεργασία παραλήπτη. Αυτή η μορφή της διαδιεργασιακής επικοινωνίας ονομάζεται μονοεκπομπή σε αντίθεση με την πολυεκπομπή όπου μια διεργασία αποστολέα στέλνει δεδομένα σε πολλαπλές διεργασίες παραλήπτες.

Είναι γνωστό ότι στην μεγάλη πλειοψηφία δικτυακών υπηρεσιών και εφαρμογών χρησιμοποιούν μονοεκπομπή ενώ η πολυεκπομπή είναι χρήσιμη σε εφαρμογές όπως ομαδική ανταλλαγή μηνυμάτων, τηλεδιάσκεψη, συνεργατικά περιβάλλοντα για εκπαίδευση και δημοπρασίες πραγματικού χρόνου.

Σε μια δικτυακή εφαρμογή που χρησιμοποιεί πολυεκπομπή, ένα σύνολο από διεργασίες σχηματίζουν μια ομάδα που ονομάζεται **ομάδα πολυεκπομπής** (multicast group). Η ομάδα πολυεκπομπής προσδιορίζεται από μια διεύθυνση IP και σε αυτήν εγγράφονται όσοι διεργασίες παραλήπτες θέλουν να λαμβάνουν μήνυμα πολυεκπομπής. Κάθε διεργασία που είναι στην ομάδα μπορούν να στέλνουν και να λαμβάνουν μηνύματα. Όταν κάποια διεργασία στέλνει ένα μή-

νυμα σε μια ομάδα πολυεκπομπής, όλοι οι διεργασίες παραλήπτες που είναι μέλη της ομάδας πολυεκπομπής λαμβάνουν το μήνυμα αυτό. Δεν είναι ανάγκη να είναι κανείς μέλος μιας ομάδας πολυεκπομπής για να στείλει μηνύματα προς αυτήν. Τέλος, μια διεργασία μπορεί να αποχωρήσει από μια ομάδα πολυεκπομπής.

Στις επόμενες ενότητες θα εξετάζουμε τις λειτουργίες επικοινωνίας που υποστηρίζει η ομαδική επικοινωνία και την διεπαφή υποδοχών πολυεκπομπής σε Java που σχετίζεται με την ομαδική επικοινωνία.

10.6.1 Λειτουργίες Επικοινωνίας Πολυεκπομπής

Η ομαδική επικοινωνία υποστηρίζεται από τις τέσσερις λειτουργίες επικοινωνίας `join`, `leave`, `send` και `receive`. Η λειτουργία `join` επιτρέπει σε μια διεργασία να εγγράφεται σε μια συγκεκριμένη ομάδα πολυεκπομπή. Μια διεργασία που εγγράφεται στην ομάδα είναι μέλος της ομάδας και έχει τη δυνατότητα να λαμβάνει όλα τα μηνύματα που στέλνονται στην ομάδα. Επίσης, μια διεργασία μπορεί να είναι μέλος ενός ή περισσοτέρων ομάδων πολυεκπομπής κάθε φορά. Η λειτουργία `leave` επιτρέπει σε μια διεργασία να τερματίζει την συμμετοχή της στην ομάδα πολυεκπομπής. Μια διεργασία που έχει αποχωρήσει από την ομάδα παύει να είναι μέλος της ομάδας και συνεπώς δεν μπορεί να λαμβάνει μηνύματα. Επίσης, η διεργασία μπορεί να παραμείνει μέλος κάποιας άλλης ομάδας πολυεκπομπής. Η λειτουργία `send` επιτρέπει σε μια διεργασία να στέλνει ένα μήνυμα σε όλες τις τρέχουσες διεργασίες που μετέχουν στην ομάδα πολυεκπομπής. Τέλος, η λειτουργία `receive` επιτρέπει σε μια διεργασία μέλους να λαμβάνει μηνύματα που στέλνονται στην ομάδα πολυεκπομπής.

Για τις παραπάνω λειτουργίες θα εξετάζουμε διεξοδικά παρακάτω από ένα λογισμικό API της Java για πολυεκπομπή.

Ο βασικός μηχανισμός επικοινωνίας πολυεκπομπής είναι ασυνδεσιοστρεφής και όχι συνδεσιοστρεφής. Αυτό συμβαίνει γιατί σε εφαρμογές όπως τα ηλεκτρονικά ομαδικά παιχνίδια που η μετάδοση δεδομένων ήχου ή βίντεο μέσω του δικτύου πρέπει να ακούγονται ή να εμφανίζονται σε πραγματικό χρόνο, πράγμα που δεν μπορεί να υλοποιηθεί με το να αποστέλονται ξανά τα καθυστερημένα μηνύματα με όλα τα μέλη μιας πολυεκπομπής, όπως γίνεται στην συνδεσιοστρεφή επικοινωνία. Στην πραγματικότητα η συνδεσιοστρεφή επικοινωνία δεν μπορεί να χρησιμοποιηθεί για πολυεκπομπή, επειδή απαιτεί συνεχή ανταλλαγή κατάστασης μεταξύ αποστολέα και

παραλήπτη, πράγμα προβληματικό στην πολυεκπομπή που επιτρέπει απεριόριστους παραλήπτες για κάθε μήνυμα που πολυεκπέμπει ο αποστολέας.

10.6.2 Διεπαφή Υποδοχών Πολυεκπομπής σε Java

Σε επίπεδο μεταφοράς, η βασική πολυεκπομπή που υποστηρίζεται από την Java είναι μια επέκταση του πρωτοκόλλου UDP το οποίο όπως είπαμε υποστηρίζει την ασυνδεσιοστρεφή επικοινωνία. Η διεπαφή Java για πολυεκπομπή παρέχει ένα σύνολο κλάσεων που είναι παρόμοιες με εκείνες τις κλάσεις των υποδοχών δεδομενογραμμάτων που είδαμε στις προηγούμενες ενότητες. Βασικά υπάρχουν τρεις κλάσεις τις οποίες εξετάζαμε στα πλαίσια των υποδοχών δεδομενογραμμάτων. Οι κλάσεις αυτές είναι οι εξής:

- `InetAddress`. Στην διεπαφή υποδοχών δεδομενογραμμάτων, αυτή η κλάση αναπαριστά την διεύθυνση IP του αποστολέα ή του παραλήπτη. Στις υποδοχές πολυεκπομπής, αυτή η κλάση μπορεί να χρησιμοποιηθεί για να αναγνωρίζει μια ομάδα πολυεκπομπής.

- `DatagramPacket`. Στις υποδοχές δεδομενογραμμάτων, ένα αντικείμενο αυτής της κλάσης αναπαριστά ένα δεδομενόγραμμα πακέτο. Στην πολυεκπομπή, ένα αντικείμενο `DatagramPacket` αναπαριστά ένα πακέτο δεδομένων που στέλνεται σε όλες τις διεργασίες ή λαμβάνεται από κάθε διεργασία της ομάδας πολυεκπομπής.

- `MulticastSocket`. Η κλάση `MulticastSocket` επεκτείνεται από την κλάση `DatagramSocket` με επιπλέον δυνατότητες για εγγραφή και αποχώρηση σε/από μια ομάδα πολυεκπομπής. Ένα αντικείμενο της κλάσης `MulticastSocket` μπορεί να χρησιμοποιηθεί για αποστολή και λήψη πακέτων πολυεκπομπής IP.

Παρακάτω συζητούμε τις διευθύνσεις πολυεκπομπής IP και περιγράφουμε τους κατασκευαστές και τις μεθόδους που υποστηρίζει η κλάση `MulticastSocket`. Για την κλάση `DatagramPacket` δεν περιγράφουμε αφού είναι παρόμοια με εκείνη τη κλάση που είδαμε στις υποδοχές δεδομενογραμμάτων.

Διευθύνσεις Πολυεκπομπής IP

Είναι γνωστό ότι στην διεπαφή υποδοχών Java που εξετάζαμε στις προηγούμενες ενότητες, μια διεργασία αποστολέα αναγνωρίζει μια διεργασία παραλήπτη προσδιορίζοντας το όνομα υπολο-

γιστή που εκτελείται η διεργασία παραλήπτης και το αριθμό θύρας στην οποία είναι συνδεδεμένη η διεργασία παραλήπτη.

Στην περίπτωση πολυεκπομπής, ο αποστολέας πολυεκπομπής χρειάζεται μια διεύθυνση ώστε να στείλει ένα μήνυμα σε μια ομάδα. Συνεπώς, ένα δεδομενόγραμμα πακέτο πολυεκπομπής πρέπει να ληφθεί από όλες τις διεργασίες που είναι μέλη σε μια συγκεκριμένη ομάδα πολυεκπομπής. Έτσι, κάθε δεδομενόγραμμα πακέτο πολυεκπομπής πρέπει να διεθυνσιοδοτηθεί σε μια ομάδα διεργασιών αντί σε μια διεργασία. Για αυτό το λόγο, μια ομάδα πολυεκπομπής προσδιορίζεται από μια διεύθυνση IP κλάσης D και από έναν αριθμό θύρας UDP.

Μια εφαρμογή που χρησιμοποιεί τη διεπαφή Java για πολυεκπομπή πρέπει να προσδιορίζει τουλάχιστον μια διεύθυνση πολυεκπομπής. Ένας τρόπος για να επιλέξουμε μια διεύθυνση πολυεκπομπής για μια εφαρμογή είναι να χρησιμοποιήσουμε μια στατική διεύθυνση κλάσης D όπως η 224.0.0.1 που αντιστοιχεί στην περιοχή ονομασίας all-systems.mcast.net για διεργασίες που εκτελούνται σε όλους τους υπολογιστές ενός τοπικού δικτύου υπολογιστών. Ένας άλλος τρόπος είναι να χρησιμοποιήσουμε μια αφηρημένη διεύθυνση που δεν έχει ανατεθεί όπως ένας αριθμός στο διάστημα 239.*.*.* (π.χ. 239.1.2.3).

Κατασκευαστές `MulticastSocket`

Ο κοινός κατασκευαστής για υποδοχή πολυεκπομπής είναι ο εξής:

- `public MulticastSocket(int port) throws IOException.` Δημιουργεί μια υποδοχή πολυεκπομπής και τη συνδέει με τη θύρα `port`. Στην περίπτωση που συμβεί σφάλμα εισόδου/εξόδου κατά τη δημιουργία της υποδοχής τότε δημιουργείται μια εξαίρεση τύπου `IOException`.

Μεθόδους `MulticastSocket`

Η κλάση `MulticastSocket` διαθέτει μερικές μεθόδους για τον χειρισμό υποδοχών πολυεκπομπής. Οι πιο κοινοί μέθοδοι είναι οι εξής:

- `public void joinGroup(InetAddress addr) throws IOException.` Εκτελεί την εγγραφή ενός παραλήπτη στην ομάδα πολυεκπομπής που προσδιορίζεται από την διεύθυνση `addr`.

- `public void leaveGroup(InetAddress addr) throws IOException.` Εκτελεί τη αποχώρηση ή διαγραφή ενός παραλήπτη από την ομάδα πολυεκπομπής που προσδιορίζεται από την διεύθυνση `addr`.

- `public int getTimeToLive() throws IOException.` Επιστρέφει τον εξ ορισμού χρόνο ζωής των πακέτων πολυεκπομπής που στέλνονται μέσω της υποδοχής αυτής.

- `public void setTimeToLive(int time) throws IOException.` Θέτει τον εξ ορισμού χρόνο ζωής των πακέτων πολυεκπομπής που στέλνονται μέσω της υποδοχής αυτής σε `time` με σκοπό τον έλεγχο των πολυεκπομπών. Η τιμή της παραμέτρου `time` πρέπει να βρίσκεται στο διάστημα [0,255] διαφορετικά δημιουργείται μια εξαίρεση τύπου `IllegalArgumentException`.

Για την αποστολή και λήψη πακέτων πολυεκπομπής, χρησιμοποιούνται οι μέθοδοι `send` και `receive` της κλάσης `DatagramSocket`.

Τα βήματα δημιουργίας προγραμμάτων πελάτη και διακομιστή για υποδοχές πολυεκπομπής είναι παρόμοια με εκείνα τις υποδοχές δεδομενογραμμάτων πακέτων UDP. Για το λόγο αυτό δεν παρουσιάζουμε τα βήματα ανάπτυξης προγραμμάτων πελάτη και διακομιστή.

Παρακάτω παρουσιάζονται τα προγράμματα πελάτη και διακομιστή που χρησιμοποιούν υποδοχές πολυεκπομπής και υλοποιούν μια μονόδρομη επικοινωνία. Συγκεκριμένα, το πρόγραμμα πελάτη στέλνει ένα μήνυμα χαιρετισμού ''`Hello`'' στην ομάδα πολυεκπομπής.

```
import java.io.*;
import java.net.*;
public class MulticastClient
{
        public static final String MCAST_ADDR = "239.1.2.3";
        public static final int MCAST_PORT = 3456;
        public static void main(String[] args)
        {
                String msg = "Hello";
                try
                {
                        InetAddress group = InetAddress.getByName(MCAST_ADDR);
                        MulticastSocket socket = new MulticastSocket(MCAST_PORT);
                        socket.joinGroup(group);
                        DatagramPacket outPacket = new DatagramPacket(msg.getBytes(), msg.length(),group,
                        System.out.println("Sending:␣" + msg);
                        socket.send(outPacket);
                        socket.leaveGroup(group);
                        socket.close();
                }
                catch (IOException e)
                {
                        System.out.println("Error␣" + e);
                }
```

```
        }
}
```

Από την άλλη πλευρά, το πρόγραμμα διακομιστή περιμένει και λαμβάνει ένα μήνυμα από την ομάδα πολυεκπομπής.

```
import java.io.*;
import java.net.*;
public class MulticastServer
{
        public static final String MCAST_ADDR = "239.1.2.3";
        public static final int MCAST_PORT = 3456;
        public static final int DGRAM_BUF_LEN = 512;
        public static void main(String[] args)
        {
                try
                {
                        InetAddress group = InetAddress.getByName(MCAST_ADDR);
                        MulticastSocket socket = new MulticastSocket(MCAST_PORT);
                        socket.joinGroup(group);
                        byte[] buf = new byte[DGRAM_BUF_LEN];
                        DatagramPacket inPacket = new DatagramPacket(buf, buf.length);
                        socket.receive(inPacket);
                        System.out.println("Received:␣" + new String(buf));
                        socket.leaveGroup(group);
                        socket.close();
                }
                catch(IOException e)
                {
                        System.out.println("Error␣" + e);
                }
        }
}
```

Κεφάλαιο 11

Προγραμματισμός Κατανεμημένων Αντικειμένων

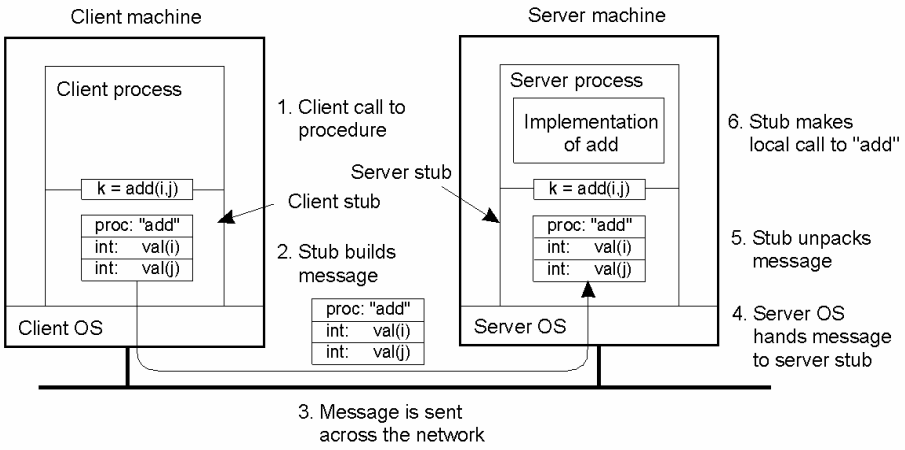
Στο κεφάλαιο αυτό ξεκινά με το βασικό μοντέλο της απομακρυσμένης κλήσης διαδικασιών και έπειτα της επέκτασης του μοντέλου αυτού σε αντικείμενα. Στην συνέχεια, θα παρουσιάζουμε την ανάπτυξη μιας απλής εφαρμογής κατανεμημένων αντικειμένων χρησιμοποιώντας το ενδιάμεσο λογισμικό Java RMI.

11.1 Απομακρυσμένη Κλήση Διαδικασιών

Μέχρι στιγμή έχουμε παρουσιάζει το μοντέλο πέρασμα μηνυμάτων το οποίο χρησιμοποιείται στα παράλληλα και κατανεμημένα συστήματα. Το μοντέλο αυτό με τις συναρτήσεις `send()` και `recv()` δεν κρύβει τις λεπτομέρειες της επικοινωνίας από το προγραμματιστή εφόσον για να στείλουμε ένα μήνυμα πρέπει να προσδιορίζουμε τον υπολογιστή ή τη διεύθυνση IP και αριθμός θύρας. Συνεπώς, το μοντέλο αυτό δεν παρέχει διαφάνεια πρόσβασης που είναι μια σημαντική ιδιότητα για τα κατανεμημένα συστήματα. Γι αυτό το λόγο υπάρχει μια μέθοδο για την απόκρυψη της επικοινωνίας και είναι γνωστή ως το μοντέλο της **απομακρυσμένης κλήσης διαδικασιών** (Remote Procedure Call - RPC). Το μοντέλο RPC είναι ουσιαστικά επέκταση του μοντέλου της τοπικής κλήσης μιας διαδικασίας και η βασική ιδέα της RPC είναι να αντικαταστήσει το μοντέλο πέρασμα μηνυμάτων με ένα μοντέλο που να επιτρέπει μια κλήση διαδικασία η οποία βρίσκεται σε ένα απομακρυσμένο υπολογιστή.

Έτσι όταν μια διεργασία πελάτης που εκτελείται σε ξεχωριστό υπολογιστή καλεί μια διαδικασία στον απομακρυσμένο υπολογιστή όπου εκτελείται η διεργασία διακομιστής, ο πελάτης αναστέλλεται και η εκτέλεση της διαδικασίας γίνεται στον διακομιστή. Οι παράμετροι της διαδικασίας μεταφέρονται από τον υπολογιστή πελάτη στον υπολογιστή διακομιστή μέσω των μηνυμάτων. Όταν η διεργασία διακομιστή λάβει ένα μήνυμα, πυροδοτεί την έναρξη εκτέλεσης της διαδικασίας και επιστρέφει το αποτέλεσμα από την εκτέλεση της διαδικασίας πίσω στην διεργασία πελάτης σε ένα μήνυμα. Συνεπώς, για να γίνει η απομακρυσμένη κλήση διαδικασιών θα πρέπει το πρόγραμμα - πελάτης να είναι συνδεδεμένο με μια μικρή διαδικασία βιβλιοθήκης που ονομάζεται **κορμός πελάτη** (client stub), το οποίο αναπαριστά τη διαδικασία-διακομιστή στο χώρο διευθύνσεων του πελάτη. Παρόμοια, ο διακομιστής είναι συνδεδεμένος με μια διαδικασία που ονομάζεται **κορμός διακομιστή** (server stub). Όταν ένας προγραμματιστής χρησιμοποιεί την RPC απλά εκτελεί σαν μια τοπική κλήση διαδικασίας με την βοήθεια των κορμών πελάτη και διακομιστή, ενώ στο παρασκήνιο ανταλλάσονται μηνύματα ανάμεσα στους υπολογιστές πελάτη και διακομιστή, δηλαδή δεν είναι ορατή το πέρασμα μηνυμάτων στον προγραμματιστή.

Στο Σχήμα 11.1 φαίνονται τα βήματα που πραγματοποιούνται όταν ο πελάτης καλεί μια διαδικασία η οποία βρίσκεται στο διακομιστή. Οι αριθμοί του Σχήματος 11.1 αντιστοιχούν στα παρακάτω βήματα (τα βήματα 7 έως 10 δεν εμφανίζονται στο Σχήμα):



Σχήμα 11.1: Μια απομακρυσμένη κλήση διαδικασίας

1. Το πρόγραμμα πελάτη καλεί τον κορμό πελάτη με κανονικό τρόπο.

2. Ο κορμός πελάτη τοποθετεί τις παραμέτρους της κλήσης σε δομή δεδομένων μηνύματος.

3. Ο κορμός πελάτη καλεί τη συνάρτηση συστήματος `send()`, ζητώντας από τον πυρήνα να στείλει το μήνυμα αυτό προς τον απομακρυσμένο πυρήνα του διακομιστή. Επίσης, σε αυτό το βήμα ο κορμός πελάτης καλεί τη συνάρτηση `receive()` και προκαλεί την αναστολή του μέχρις ότου λάβει την απάντηση από τον διακομιστή.

4. Ο πυρήνας του απομακρυσμένου διακομιστή μεταβιβάζει το μήνυμα σε μια διαδικασία κορμού διακομιστή.

5. Ο κορμός διακομιστής ανακτά τις παραμέτρους από το μήνυμα και καλεί την διαδικασία του διακομιστή με το κανονικό τρόπο.

6. Η διαδικασία του διακομιστή εκτελείται και επιστρέφει το αποτέλεσμα στο κορμό του διακομιστή.

7. Ο κορμός διακομιστής τοποθετεί τα αποτελέσματα σε ένα μήνυμα απάντησης.

8. Ο κορμός διακομιστής καλεί τη συνάρτηση συστήματος `send()` ζητώντας από τον πυρήνα να στείλει το μήνυμα στον πυρήνα του πελάτη.

9. Ο πυρήνας του πελάτη στέλνει το μήνυμα στο κορμό πελάτη και ο κορμός του πελάτη ενημερώνεται ότι η κλήση `receive()` έχει ολοκληρωθεί.

10. Ο κορμός του πελάτη ανακτά τα αποτελέσματα από το μήνυμα που έλαβε και τα μεταβιβάζει με το συνηθισμένο τρόπο στο πρόγραμμα πελάτη.

Μια από τις εργασίες του μοντέλου RPC που κρύβει από τους προγραμματιστές είναι η τοποθέτηση των παραμέτρων σε μηνύματα η οποία είναι γνωστή ως **πρόταξη παραμέτρων** (parameter marshalling) και η ανάκτηση των αποτελεσμάτων από τα μηνύματα η οποία είναι γνωστή ως **αποπρόταξη** (unmarshalling). Οι διαδικασίες της πρόταξης και της αποπρόταξης εκτελούνται στους κορμούς πελάτη και διακομιστή αντίστοιχα. Από τα παραπάνω βήματα της απομακρυσμένης κλήσης διαδικασιών, προκύπτουν ορισμένα ζητήματα που έχουν να κάνουν με την επικοινωνία (ή πέρασμα παραμέτρων) μεταξύ του πελάτη και διακομιστή οι οποίοι μπορεί

να εκτελούνται σε διαφορετικές αρχιτεκτονικές μεταξύ τους. Οι αρχιτεκτονικές αυτές περιλαμβάνουν διαφορετικές εσωτερικές αναπαραστάσεις δεδομένων, διαφορετικές διατάξεις byte και προβλήματα σχετικά με την μεταφορά δεικτών.

Ένα σημαντικό καθήκον κατά την διαδικασία πρόταξης είναι η μετατροπή των δεδομένων σε ένα μορφότυπο που να είναι κατανοητό από το παραλήπτη. Γενικά, οι διαφορές στο μορφότυπο μπορούν να αντιμετωπιστούν ορίζοντας ένα πρότυπο **μορφότυπο δικτύου** (network format) στο οποίο θα μετατρέπονται όλα τα δεδομένα. Όμως, η μετατροπή αυτή μπορεί να είναι περιττή στην περίπτωση που οι υπολογιστές πελάτη και διακομιστή χρησιμοποιούν το ίδιο εσωτερικό μορφότυπο ακόμα και αν το μορφότυπο αυτό διαφέρει από το μορφότυπο δικτύου. Για να αποφύγουμε το πρόβλημα αυτό, μια εναλλακτική λύση είναι να προσδιορίζουμε το μορφότυπο που χρησιμοποιείται σε ένα πεδίο του μηνύματος έτσι ώστε ο παραλήπτης ή ο διακομιστής με βάση το πεδίο αυτό να αποφασίσει αν θα γίνονται οι απαιτούμενες μετατροπές.

Ένα άλλο πρόβλημα που αντιμετωπίζει η RPC είναι η χρήση των δεικτών. Οι δείκτες δεν μπορούν να διαμοιράζουν ανάμεσα σε απομακρυσμένους υπολογιστές, δηλαδή ο χώρος διευθύνσεων του πελάτη δεν μπορεί να μεταφερθεί στο χώρο διευθύνσεων του διακομιστή εφόσον δεν θα υπάρχει σωστή αντιστοίχηση των διευθύνσεων και θα είναι άχρηστοι για απομακρυσμένη κλήση. Μια απλή λύση είναι η απαραίτητη σειριοποίηση όλων των δεικτών (που δείχνουν σε δομές δεδομένων) όταν αυτοί μεταβιβάζονται στο κορμό πελάτη. Στην πλευρά του κορμού διακομιστή ανακτά όλες τις σειριοποιημένες δομές δεδομένων στο χώρο διευθύνσεων του παραλήπτη. Δυστυχώς, η προσέγγιση αυτή παρουσιάζει προβλήματα με τις σύνθετες (ή δυναμικές) δομές δεδομένων. Μια άλλη προσέγγιση για την αντιμετώπιση του προβλήματος των δεικτών είναι ο διακομιστής να στέλνει μια αίτηση για προσπέλαση δεδομένων στο πελάτη κάθε φορά που συναντά ένα δείκτη.

Με τα βάση τα παραπάνω, παρατηρούμε το σημαντικό ρόλο που παίζουν οι κορμοί πελάτη και διακομιστή οι οποίοι αποκρύπτουν από τις διαδικασίες το μηχανισμό RPC. Έτσι, για να μπορέσουμε μια διαδικασία να είναι διαθέσιμη για απομακρυσμένη κλήση πρέπει να έχουμε στην διάθεση μας τους κορμούς πελάτη και διακομιστή. Για την δημιουργία κορμών πελάτη και διακομιστή έχουν αυτοματοποιηθεί με τη βοήθεια κατάλληλων εργαλείων. Συνεπώς, όταν ένας διακομιστής παρέχει υπηρεσίες απομακρυσμένης κλήσης διαδικασιών πρέπει να ορίζει τις διαθέσιμες διαδικασίες που θα καλούνται από τους πελάτες σε μια **διεπαφή υπηρεσίας** (service interface). Ο όρος διεπαφή υπηρεσίας αναφέρεται στον ορισμό των διαδικασιών που παρέχον-

ται από τον διακομιστή. Μια διεπαφή υπηρεσίας ορίζεται γενικά σε μια **Γλώσσα Ορισμού Διεπαφών** (Interface Definition Language - IDL), η οποία είναι μια απλοποιημένη γλώσσα προγραμματισμού κατάλληλη για ορισμούς τύπων δεδομένων και υπογραφές διαδικασιών (δηλαδή, η δήλωση και η σειρά των παραμέτρων της κάθε διαδικασίας κλπ) αλλά όχι για συγγραφή εκτελέσιμου κώδικα. Ο ορισμός της διεπαφής υπηρεσίας σε IDL χρησιμοποιείται για την αυτόματη παραγωγή κώδικα κορμού πελάτη και διακομιστή με την βοήθεια μιας γεννήτριας κορμών. Στην συνέχεια, ο κώδικας των κορμών πελάτη και διακομιστή μεταγλωττίζονται και συνδέονται με το πρόγραμμα του πελάτη και την υλοποίηση της διαδικασίας αντίστοιχα.

Επίσης, ο μηχανισμός RPC είναι σύγχρονος ή υποστηρίζει ανασταλτική επικοινωνία. Αυτό σημαίνει ότι οι πελάτες που πραγματοποιούν κλήσεις RPC περιμένουν μέχρι να εκτελεστεί η απομακρυσμένη διαδικασία και να έχει επιστρέψει μια απάντηση. Παρόλο που είναι επιθυμητή αυτή η συμπεριφορά, μερικές φορές η αναμονή του πελάτη δεν είναι απαραίτητη. Για παράδειγμα, αν η διαδικασία δεν επιστρέψει τιμές τότε δεν είναι απαραίτητο ο πελάτης να περιμένει μια απάντηση. Έτσι, σε αυτή την περίπτωση είναι καλύτερο για την RPC να επιστρέψει αμέσως μόλις ο διακομιστής επιβεβαιώσει την λήψη του μηνύματος. Αυτή η διαδικασία λέγεται ασύγχρονη επικοινωνία RPC.

Τέλος, ένα ζήτημα που πρέπει να εξετάσουμε είναι πως κορμός του πελάτη γνωρίζει που θα στείλει ένα μήνυμα RPC. Σε μια κανονική ή τοπική κλήση διαδικασία, η διεύθυνση της διαδικασίας προσδιορίζεται κατά το χρόνο της μεταγλώττισης και γίνεται άμεσα η κλήση. Στην RPC η πληροφορία αυτή ζητείται από μια **υπηρεσία δέσμευσης** (binding service). Η υπηρεσία αυτή επιτρέπει την εγγραφή και την αναζήτηση υπηρεσιών. Μια υπηρεσία δέσμευσης τυπικά παρέχει μια διεπαφή παρόμοια με την παρακάτω:

- `register(name, version, handle, UID)`
- `deregister(name, version, UID)`
- `lookup(name, version)` $--> $ `(handle,UID)`

όπου `handle` είναι μια φυσική διεύθυνση (διεύθυνση IP, ταυτότητα διεργασία (ID), κλπ) και `UID` χρησιμοποιείται για την διάκριση ανάμεσα στους διακομιστές που προσφέρουν την ίδια υπηρεσία (ή διαδικασία). Επίσης, είναι σημαντικό να περιλαμβάνονται πληροφορίες έκδοσης αφού η α-

παίτηση της ευελιξίας για ένα κατανεμημένο σύστημα απαιτεί σε εμάς να αντιμετωπίζουμε τις διαφορετικές εκδόσεις του ίδιου λογισμικού μέσα σε ένα ετερογενές περιβάλλον.

Από τις αρχές της δεκαετίας του 80, το μοντέλο κλήση απομακρυσμένων διαδικασιών έχει χρησιμοποιηθεί ευρύτατα για τη δημιουργία κατανεμημένων εφαρμογών. Μια αρκετά ώριμη προγραμματιστική διεπαφή RPC είναι αυτό που σχεδιάστηκε για το Κατανεμημένο Περιβάλλον Επεξεργασίας (Distributed Computing Environment - DCE). Σε αυτό το βιβλίο δεν θα ασχοληθούμε σε βάθος το μοντέλο RPC επειδή το μοντέλο αυτό είναι διαδικασιοστρεφές και είναι περισσότερο κατάλληλο για μια διαδικασιακή γλώσσα προγραμματισμού C. Επίσης, το μοντέλο RPC δεν είναι κατάλληλο για προγράμματα σε αντικειμενοστρεφή γλώσσα Java όπου τη υιοθετούμε σε αυτό το βιβλίο. Γι αυτό το λόγο στο υπόλοιπο του βιβλίου θα ασχοληθούμε το αντίστοιχο μοντέλο RPC που υποστηρίζει η Java και είναι η **επίκληση απομακρυσμένων μεθόδων** (remote method invocation - RMI) ή **κατανεμημένα αντικείμενα** (distributed objects).

11.2 Κατανεμημένα Αντικείμενα

Η RPC περιλαμβάνει την κλήση της διαδικασίας σε απομακρυσμένο διακομιστή και το μοντέλο αυτό μπορεί να επεκταθεί και σε αντικείμενα. Με άλλα λόγια, υπάρχει η δυνατότητα επίκληση μεθόδων ενός αντικειμένου που εκτελείται σε ξεχωριστό απομακρυσμένο υπολογιστή. Πριν προχωρήσουμε περισσότερο για την επίκληση απομακρυσμένων μεθόδων να εξετάσουμε πρώτα ορισμένα βασικά στοιχεία για τα αντικείμενα.

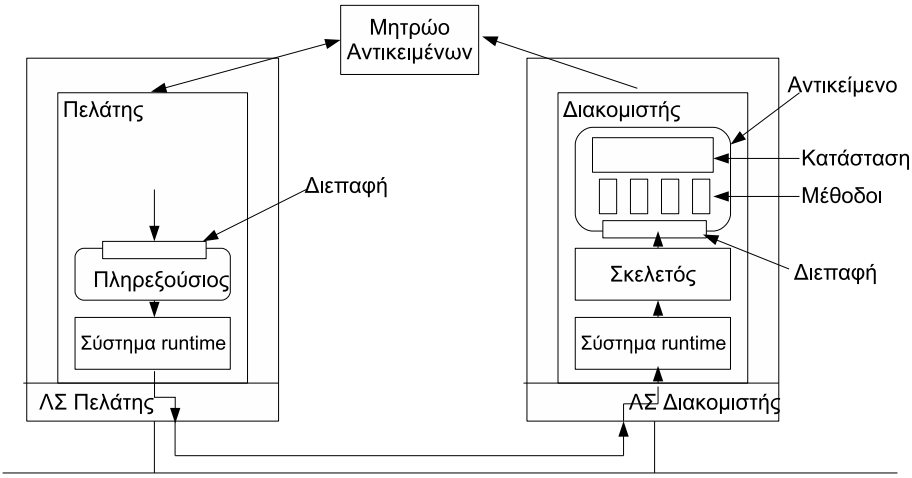
Στον αντικειμενοστραφή προγραμματισμό συνήθως υπάρχουν οι έννοιες **κλάση** (class) και **αντικείμενα** (objects). Η κλάση είναι ένα πρότυπο που χρησιμοποιείται για τη δημιουργία αντικειμένων ενός ορισμένου τύπου και ένα αντικείμενο είναι ένα στιγμιότυπο μιας συγκεκριμένης κλάσης. Τα δεδομένα ενός αντικειμένου ονομάζονται **μεταβλητές στιγμιότυπου** (instance variables) ενώ οι λειτουργίες που εφαρμόζονται στα δεδομένα αυτά ονομάζονται **μέθοδοι** (methods). Ένα αντικείμενο συνδυάζει τα περιεχόμενα των μεταβλητών που ονομάζεται **κατάσταση** (state) και τις **λειτουργίες** (operations) πάνω στην κατάσταση αυτή, δηλαδή τις μεθόδους ή την συμπεριφορά του αντικειμένου. Επίσης, μια κλάση ορίζει μια **διεπαφή** (interface) που ορίζει τις μεθόδους που υλοποιούν τα αντικείμενα της κλάσης αυτής. Συγκεκριμένα, η διεπαφή ορίζει υπογραφές ενός συνόλου μεθόδων (δηλαδή τους τύπους των παραμέτρων, τις

επιστρεφόμενες τιμές και τις εξαιρέσεις) χωρίς την υλοποίηση τους. Ένα αντικείμενο μπορεί να έχει περισσότερες από μια διεπαφές. Επίσης, **τοπικά αντικείμενα** (local objects) είναι τα αντικείμενα των οποίων οι μέθοδοι της καλούνται μόνο από μια τοπική διεργασία, η οποία εκτελείται στον ίδιο υπολογιστή που βρίσκεται η υλοποίηση του αντικειμένου. Τέλος, τα τοπικά αντικείμενα προσπελάζονται μέσω **αναφορών αντικειμένων** (object references). Έτσι, μια μεταβλητή που εμφανίζεται να αποθηκεύει ένα αντικείμενο στην πράξη αποθηκεύει μια αναφορά προς το αντικείμενο αυτό και βρίσκεται μέσα στο μοναδικό χώρο διευθύνσεων. Επομένως, για να γίνει κλήση σε μια μέθοδο ενός αντικειμένου πρέπει πρώτα να υπάρχει αναφορά αντικειμένου και το όνομα της μεθόδου μαζί με τις παραμέτρους.

Στα κατανεμημένα συστήματα υπάρχει μια διάκριση ανάμεσα στην διεπαφή και την υλοποίηση του αντικειμένου. Η διάκριση αυτή επιτρέπει την τοποθέτηση της διεπαφής σε ένα υπολογιστή ενώ η υλοποίηση του αντικειμένου σε ένα διαφορετικό υπολογιστή του κατανεμημένου συστήματος. **Απομακρυσμένο αντικείμενο** (remote object) είναι η υλοποίηση του αντικειμένου που είναι εγκατεστημένη σε ένα υπολογιστή και οι μέθοδοι του αντικειμένου αυτού καλούνται από άλλες απομακρυσμένες διεργασίες που εκτελούνται σε διαφορετικούς υπολογιστές, δηλαδή οι υπολογιστές που διαθέτουν την διεπαφή του αντικειμένου. Μια γενικότερη μορφή είναι το κατανεμημένο αντικείμενο το οποίο μπορεί να έχει την υλοποίηση του αντικειμένου σε περισσότερους από έναν υπολογιστές. Σε αυτό το βιβλίο θα ασχοληθούμε κυρίως με τα απομακρυσμένα αντικείμενα που είναι πιο συνηθισμένα. Η διαδικασία κλήσης μιας μεθόδου ενός απομακρυσμένου αντικειμένου ονομάζεται επίκληση απομακρυσμένων αντικειμένων ή μεθόδων. Όπως ένα τοπικό αντικείμενο έχει περισσότερες από μια διεπαφές έτσι και στο απομακρυσμένο αντικείμενο υλοποιεί μια ή περισσότερες **απομακρυσμένες διεπαφές** (remote interfaces). Οι απομακρυσμένες διεπαφές περιέχουν τις υπογραφές των μεθόδων ενός αντικειμένου των οποίων μπορούν να κληθούν από αντικείμενα άλλων υπολογιστών. Τέλος, τα αντικείμενα για να μπορούν να καλέσουν τις μεθόδους ενός απομακρυσμένου αντικειμένου πρέπει πρώτα να αποκτήσουν μια **απομακρυσμένη αναφορά αντικειμένου** (remote object reference). Η απομακρυσμένη αναφορά αντικειμένου είναι ένας προσδιοριστής που μπορεί να χρησιμοποιηθεί μέσα στο κατανεμημένο σύστημα για να αναφέρεται σε ένα συγκεκριμένο μοναδικό αντικείμενο.

11.2.1 Αρχιτεκτονική Κατανεμημένων Αντικειμένων

Στο Σχήμα 11.2 παρουσιάζεται μια γενική αρχιτεκτονική που υποστηρίζει το μοντέλο των κατανεμημένων αντικειμένων. Η υλοποίηση του αντικειμένου που φιλοξενείται από μια διερ-



Σχήμα 11.2: Αρχιτεκτονική κατανεμημένων αντικειμένων

γασία ονομάζεται **διακομιστή αντικειμένων** (object server) και η διεργασία που κατέχει την διεπαφή ώστε να προσπελάζει το απομακρυσμένο αντικείμενο ονομάζεται **πελάτη αντικειμένου** (object client) ή απλά πελάτης. Επίσης, σε αυτή την αρχιτεκτονική παρουσιάζεται μια τρίτη οντότητα που λέγεται **μητρώου αντικειμένων** (object registry) το οποίο είναι μια υπηρεσία ονομασίας που συσχετίζει τα αντικείμενα με λογικά ονόματα. Έτσι, τα αντικείμενα καταχωρούνται στο μητρώο κάτω από μοναδικό όνομα. Όταν μια διεργασία πελάτη θέλει να προσπελάζει ένα απομακρυσμένο αντικείμενο, πρώτα αναζητά στο μητρώο αντικειμένων για μια απομακρυσμένη αναφορά αντικειμένου με την χρήση του ονόματος. Η αναφορά αυτή θα χρησιμοποιηθεί από το πελάτη για την κλήση των μεθόδων του απομακρυσμένου αντικειμένου. Έτσι, για την κλήση των απομακρυσμένων μεθόδων πρέπει ο πελάτης να συνδεθεί με το **αντικείμενο πληρεξούσιο** (proxy object) του τοπικού χώρου διευθύνσεων της. Το αντικείμενο πληρεξούσιο προτάσει τους παραμέτρους της κλήσης και με την βοήθεια του συστήματος runtime στέλνει την αίτηση κλήσης στο διακομιστή αντικειμένων. Αντίστοιχα, το σύστημα runtime του διακομιστή λαμβάνει μηνύματα κλήσεις και διανέμει την κλήση σε ένα κατάλληλο **αντικείμενο σκελετός** (skeleton) ο οποίος είναι υπεύθυνος για την αποπρόταξη της αίτησης και την κλήση κατάλληλων μεθόδων στο τοπικό αντικείμενο. Τα αποτελέσματα της εκτέλεσης των

μεθόδων επιστρέφονται με παρόμοιο τρόπο αλλά σε αντίθετη σειρά όπως παρουσιάζαμε για την περίπτωση RPC.

Σύμφωνα με την παραπάνω αρχιτεκτονική για να λειτουργήσει το μοντέλο των κατανεμημένων αντικειμένων πρέπει απαραίτητα να περιλαμβάνει ένα αριθμό στοιχείων όπως μια διεπαφή, ένα αντικείμενο πληρεξούσιο, ένα αντικείμενο σκελετό και ένα μητρώο αντικειμένων. Τα στοιχεία αυτά θα τα αναλύσουμε παρακάτω.

Διεπαφή

Μια διεπαφή ορίζει τις μεθόδους που υλοποιεί ένα απομακρυσμένο αντικείμενο. Επίσης, μια διεπαφή καθορίζει τη δομή των μεθόδων που είναι διαθέσιμες στον απομακρυσμένο πελάτη, καθώς και τη δομή των μεταβλητών που περιγράφουν την κατάσταση του αντικειμένου δηλαδή την σύνταξη των μεθόδων και όχι την υλοποίηση τους. Ένα αντικείμενο μπορεί να υλοποιεί περισσότερες από μια διεπαφές. Όπως στο μοντέλο RPC, οι διεπαφές ορίζονται χρησιμοποιώντας μια ειδική γλώσσα όπως η Γλώσσα ορισμού διεπαφών IDL ή μπορεί να είναι τμήμα μιας γλώσσας προγραμματισμού όπως η διεπαφή Java στην RMI. Η IDL για κατανεμημένα αντικείμενα είναι συνήθως εμπλουτισμένη από ότι τα συστήματα RPC και επιτρέπει τον ορισμό των κλάσεων, χαρακτηριστικά κλάσης, μεθόδους public, εξαιρέσεις και άλλους σύνθετους τύπους δεδομένων (για παράδειγμα, πίνακες, εγγραφές).

Αντικείμενο Πληρεξούσιο

Προκειμένου ο πελάτης να χρησιμοποιήσει ένα απομακρυσμένο αντικείμενο πρέπει πρώτα να αποκτήσει μια αναφορά προς το απομακρυσμένο αντικείμενο δηλαδή ο πελάτης να δεσμευτεί με το απομακρυσμένο αντικείμενο. Αυτό έχει σαν αποτέλεσμα την δημιουργία του πληρεξούσιου στο χώρο διευθύνσεων του πελάτη και την φόρτωση της αναφοράς στο πληρεξούσιο από το μητρώο αντικειμένων. Μετά την δέσμευση, ο πελάτης μπορεί να καλέσει μια τοπική μέθοδο στο πληρεξούσιο. Ο πληρεξούσιος είναι ανάλογος με τον κορμό πελάτη του μοντέλου RPC, δηλαδή είναι υπεύθυνος για την πρόταξη παραμέτρων κατά τις κλήσεις των μεθόδων και την αποστολή τους μέσω μηνυμάτων στον απομακρυσμένο διακομιστή αντικειμένων. Επίσης, ο πελάτης πρέπει να χρησιμοποιεί ένα διαφορετικό πληρεξούσιο για κάθε απομακρυσμένο αντικείμενο που θέλει να προσπελάζει. Ο κώδικας του πληρεξούσιου συνήθως δημιουργείται αυτόματα με το πέρασμα

της γλώσσας ορισμού διεπαφής μέσα από έναν ειδικό επεξεργαστή και τη σύνδεση αυτού του κώδικα με τον κώδικα του πελάτη. Το σύστημα υποστήριξης στην πλευρά του πελάτη παρέχει ορισμένες υπηρεσίες που είναι απαραίτητες για τον πληρεξούσιο ώστε να επικοινωνήσει με τον διακομιστή αντικειμένων. Επίσης, το σύστημα υποστήριξης παρέχει υπηρεσίες άμεσες προς το πελάτη όπως η αρχικοποίηση του συστήματος, λειτουργίες για την δέσμευση σε αντικείμενα και λειτουργίες για την διαχείριση απομακρυσμένων αναφορών.

Αντικείμενο Σκελετός

Η βασική εργασία του διακομιστή αντικειμένων είναι η φιλοξενία των υλοποιήσεων του αντικειμένου και επιτρέπει στους απομακρυσμένους πελάτες την απομακρυσμένη κλήση μεθόδων. Το αντικείμενο που φιλοξενείται στο διακομιστή περιέχει μια κατάσταση και την υλοποίηση των μεθόδων που λειτουργούν πάνω στην κατάσταση αυτή. Έτσι, το αντικείμενο υλοποιεί τις μεθόδους που ορίζεται από την απομακρυσμένη διεπαφή. Η υλοποίηση του αντικειμένου προσπελάζεται από το αντικείμενο σκελετός ο οποίος είναι ανάλογος με τον κορμό του διακομιστή του RPC, δηλαδή ανακτά τις παραμέτρους από το μήνυμα και καλεί τοπικά τις μεθόδους του αντικειμένου. Επίσης, όπως στον κώδικα του πληρεξούσιου έτσι και στον κώδικα σκελετός δημιουργείται αυτόματα με το πέρασμα της γλώσσας ορισμού διεπαφής μέσα από έναν ειδικό επεξεργαστή και τη σύνδεση αυτού του κώδικα με τον κώδικα του διακομιστή (ή με τον κώδικα υλοποίησης αντικειμένου). Το σύστημα υποστήριξης στην πλευρά του διακομιστή είναι υπεύθυνο για την λήψη κλήσεων απομακρυσμένων μεθόδων από τους πελάτες και την διανομή των κλήσεων αυτών στους κατάλληλους σκελετούς οι οποίοι αναλαμβάνουν την περαιτέρω διαχείριση κάθε κλήσης. Οι εργασίες του συστήματος υποστήριξης χωρίζονται σε βασικό σύστημα υποστήριξης και σε **προσαρμογέα αντικειμένων** (object adapter). Το βασικό σύστημα υποστήριξης αναλαμβάνει την επικοινωνία και την βασική διανομή των κλήσεων ενώ ο προσαρμογέας αντικειμένων αναλαμβάνει τις πολιτικές ενεργοποίησης, την δημιουργία αντικειμένων και την διανομή σε συγκεκριμένα αντικείμενα. Επίσης, ο προσαρμογέας αντικειμένων είναι υπεύθυνος να παρέχει μια αντικειμενοστρεφή διεπαφή στο πιθανόν διαδικαστικό κώδικα περιτυλίγοντας τον κώδικα αυτό μέσα σε ένα αντικείμενο.

Μητρώο Αντικειμένων

Για να μπορέσει ο πελάτης να καλέσει μεθόδους απομακρυσμένων αντικειμένων απαιτείται κάποιος τρόπος αναφοράς στα απομακρυσμένα αυτά αντικείμενα που υπάρχουν σε ένα κατανεμημένο σύστημα. Μια αναφορά απομακρυσμένου αντικειμένου πρέπει να επιτρέψει στον πελάτη να δημιουργήσει ένα πληρεξούσιο στο χώρο διευθύνσεων της και να επιτρέψει στον πληρεξούσιο να εντοπίζει και να δεσμευθεί ή (συνδεθεί) με το κατάλληλο διακομιστή αντικειμένων όπου φιλοξενεί την υλοποίηση του αντικειμένου ώστε να επικοινωνήσει με τον αντίστοιχο σκελετό. Για να είναι επιτυχής η δέσμευση με ένα αντικείμενο πρέπει να εντοπιστεί ο διακομιστής όπου εκτελείται το αντικείμενο στο οποίο γίνεται αναφορά. Για τον εντοπισμό του διακομιστή χρειαζόμαστε ένα μητρώο αντικειμένων. Ένας μητρώο αντικειμένων σε ένα κατανεμημένο σύστημα είναι μια υπηρεσία που συντηρεί έναν πίνακα που περιέχει απεικονίσεις συμβολικών ονομάτων προς αναφορές απομακρυσμένων αντικειμένων. Το μητρώο χρησιμοποιείται από τους διακομιστές για να εγγράφουν τα απομακρυσμένα αντικείμενα τους με χρήση του ονόματος των τελευταίων και από τους πελάτες για να αναζητούν τα αντικείμενα αυτά. Έτσι, για να μπορέσουμε να εντοπίσουμε ένα αντικείμενο, μια αναφορά προς αυτό θα πρέπει να περιλαμβάνει αρκετές πληροφορίες ώστε να επιτρέπει τη δέσμευση του πελάτη με το απομακρυσμένο αντικείμενο. Μια λύση είναι η αναφορά απομακρυσμένου αντικειμένου να περιέχει τη διεύθυνση διακομιστή αντικειμένων, τη θύρα του διακομιστή, το αναγνωριστικό του αντικειμένου και μια αναφορά προς ένα αντικείμενο πληρεξούσιο.

Μετά την μελέτη της αρχιτεκτονικής κατανεμημένων αντικειμένων, πρέπει να σημειωθεί ότι οι λειτουργίες του μοντέλου RPC είναι άμεσα συσχετισμένες με ένα διακομιστή ενώ οι λειτουργίες του μοντέλου RMI είναι μόνο συσχετισμένες με ένα αντικείμενο. Αυτό σημαίνει ότι η πραγματική θέση του αντικειμένου (π.χ. το διακομιστή το οποίο φιλοξενεί το αντικείμενο) είναι διαφανής προς το πελάτη για κλήση των μεθόδων.

11.2.2 Ανάπτυξη Απομακρυσμένων Αντικειμένων

Οι κατανεμημένες εφαρμογές που υλοποιούν απομακρυσμένα αντικείμενα συνήθως βασίζονται στο μοντέλο πελάτη - διακομιστή. Στην περίπτωση αυτήν οι διακομιστές διαχειρίζονται τα απομακρυσμένα αντικείμενα και οι πελάτες καλούν τις μεθόδους των τελευταίων. Συνεπώς, για την ανάπτυξη του κώδικα του διακομιστή και του πελάτη ακολουθούμε μια σειρά από τυποποιημένα

βήματα.

Το πρώτο βήμα στη διαδικασία είναι η σχεδίαση του συστήματος σε όρους των κλάσεων που καθορίζουν τα εμπλεκόμενα αντικείμενα. Αυτό το βήμα θα είναι το ίδιο είτε το σύστημα είναι κατανεμημένο είτε όχι. Το αποτέλεσμα αυτού του πρώτου βήματος θα είναι ένας ορισμός των κλάσεων και των υπηρεσιών που σχετίζονται με αυτές τις κλάσεις σε όρους των μεθόδων κάθε κλάσης.

Το επόμενο βήμα είναι ο ορισμός αυτών των κλάσεων σε όρους κάποιας γλώσσας ορισμού διεπαφής. Τα αρχεία που περιέχουν τους ορισμούς των διεπαφών θα υποστούν κατόπιν επεξεργασία από κάποιο εργαλείο ή μεταγλωττιστή που παράγει κώδικα πληρεξούσιου του πελάτη και κώδικα σκελετού του διακομιστή οι οποίοι υλοποιούν διαδικασίες όπως η πρόταξη και η αποπρόταξη των δεδομένων καθώς και η αποστολή τους μέσω κάποιας γραμμής επικοινωνίας. Αυτοί οι κώδικες τελικά θα συγχωνευτούν με τον κώδικα εφαρμογής.

Το επόμενο βήμα είναι ο προγραμματισμός του διακομιστή που περιέχει την υλοποίηση των κλάσεων των απομακρυσμένων αντικειμένων. Ο διακομιστής θα κάνει τα απομακρυσμένα αντικείμενα διαθέσιμα στους πελάτες. Για αυτό το λόγο, στο τμήμα αρχικοποίησης του προγράμματος διακομιστή δηλαδή στη μέθοδο `main()` της Java είναι υπεύθυνο για τη δημιουργία αντικειμένων που φιλοξενεί ο διακομιστής. Τα αντικείμενα θα δημιουργηθούν είτε στατικά, δηλαδή δημιουργούνται μόνο μια φορά και παραμένουν στο διακομιστή - είτε δυναμικά, όπου θα δημιουργούνται κατ΄ απαίτηση ανάλογα με τη ζήτηση και θα απενεργοποιούνται όταν δε θα χρειάζονται. Ο κώδικας του διακομιστή θα χρησιμοποιεί το βοηθητικό κώδικα. Επίσης, στο τμήμα αρχικοποίησης του διακομιστή πρέπει επίσης να καταχωρήσει τα αντικείμενα σε μια υπηρεσία μητρώου ώστε οι πελάτες να μπορούν να τα βρούν. Αυτό συνήθως περιλαμβάνει την καταχώρηση των ονομάτων σε ένα διακομιστή μητρώου που σχετίζεται με το σχήμα κατανεμημένων αντικειμένων. Ο τρόπος ονοματοδοσίας εξαρτάται από την κατανεμημένη τεχνολογία που χρησιμοποιείται: η Java RMI έχει ένα απλό σύστημα ονοματοδοσίας, ενώ η CORBA έχει ένα ιεραρχικό σύστημα ονοματοδοσίας παρόμοιο με αυτό του συστήματος ονοματοδοσίας πεδίων του Διαδικτύου.

Τέλος, πρέπει να αναπτυχθεί και ο κώδικας για τον πελάτη. Και πάλι, αυτός ο κώδικας θα επικοινωνεί με το βοηθητικό κώδικα που έχει παραχθεί. Ο κώδικας του πελάτη στην πράξη αναφέρεται στα απομακρυσμένα αντικείμενα μέσω της υπηρεσίας μητρώου, καλεί μεθόδους στα απομακρυσμένα αντικείμενα και επεξεργάζεται τα δεδομένα που επιστρέφονται ως αποτέλεσμα

αυτών των κλήσεων.

Αυτή είναι λοιπόν η διαδικασία ανάπτυξης κώδικα κατανεμημένων αντικειμένων. Υπάρχουν μερικά σημεία που πρέπει να τονίσουμε:

- Το πρώτο βήμα είναι το ίδιο σε κάθε διαδικασία ανάπτυξης αντικειμένων, ανεξάρτητα αν αυτά θα είναι κατανεμημένα ή όχι.
- Χρησιμοποιείται πάντα μια υπηρεσία μητρώου αντικειμένων.
- Οι κώδικες πελάτη και διακομιστή θα πρέπει να περιέχουν ελάχιστες αναφορές στο γεγονός ότι χρησιμοποιείται κάποιο απομακρυσμένο αντικείμενο. Το μόνο που θα βλέπει όποιος διαβάζει τον κώδικα θα είναι το γεγονός ότι χρησιμοποιείται μια υπηρεσία μητρώου.

11.3 Τεχνολογίες Κατανεμημένων Αντικειμένων

Το υπόδειγμα κατανεμημένων αντικειμένων έχει ευρύτατα υιοθετηθεί σε κατανεμημένες εφαρμογές για το οποίο ένας μεγάλος αριθμός εργαλείων είναι διαθέσιμος. Μεταξύ αυτών τα πιο δημοφιλή εργαλεία είναι τα εξής:

- Java Remote Method Invocation (RMI),
- συστήματα βασισμένα σε Common Object Request Broker Architecture (CORBA) και
- Distributed Component Object Model (DCOM).

Φυσικά, η πιο απλή τεχνολογία είναι η Java RMI που θα ασχοληθούμε περισσότερο σε αυτό βιβλίο και παρέχει τα βασικά για την εύκολη εκμάθηση των βασικών εννοιών του μοντέλου κατανεμημένων αντικειμένων. Επίσης, η Java RMI είναι μια καθαρά Java τεχνολογία σε σχέση με τις δύο άλλες τεχνολογίες, που περιορίζεται στην ανάπτυξη κατανεμημένων αντικειμένων Java. Οσο αφορά τις άλλες δύο τεχνολογίες είναι αδύνατο να τις καλύψουμε σε αυτό το βιβλίο αλλά αν ενδιαφέρεστε μπορείτε να συμβουλείτε τις αναφορές.

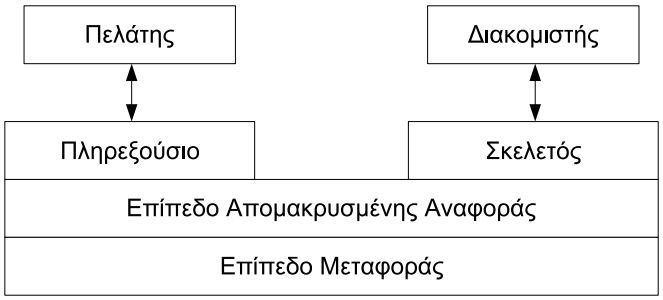
11.4 Java RMI

Στην ενότητα αυτή θα εξετάσουμε το ενδιάμεσο λογισμικό Java RMI το οποίο είναι μια αντικειμενοστραφής υλοποίηση του μοντέλου RPC και έχει ενσωματωθεί στην γλώσσα προγραμ-

ματισμού Java. Στην συνέχεια θα παρουσιάζουμε την διαδικασία ανάπτυξης κατανεμημένων εφαρμογών με την βοήθεια του συστήματος Java RMI. Πρέπει να σημειώσουμε εδώ ότι η κατανεμημένη εφαρμογή που χρησιμοποιεί την Java RMI πρέπει να είναι γραμμένη σε γλώσσα Java και πρέπει να χρησιμοποιεί τα εργαλεία που παρέχονται από την Εργαλειοθήκη Ανάπτυξης της Java (Java Development Kit -JDK).

11.4.1 Αρχιτεκτονική Java RMI

Είδαμε ότι το σύστημα runtime του Σχήματος 11.2 είναι μια υλοποίηση που προσφέρεται από μια τεχνολογία κατανεμημένων αντικειμένων όπως η Java RMI που θα δούμε παρακάτω. Το σύστημα runtime είναι υπεύθυνο για το μετασχηματισμό των επικλήσεων απομακρυσμένων μεθόδων σε τοπικές κλήσεις μεθόδων και χειρίζεται τις λεπτομέρειες της διαδιεργασιακής επικοινωνίας. Το σύστημα Java RMI βασίζεται σε μια αρχιτεκτονική επιπέδων. Η αρχιτεκτονική αυτή φαίνεται στο Σχήμα 11.3. Αποτελείται από τρία επίπεδα:



Σχήμα 11.3: Τα επίπεδα του Java RMI

- Το **επίπεδο πληρεξούσιου / σκελετού** (proxy/skeleton layer): Πρόκειται για το επίπεδο με το οποίο επικοινωνούν τα απομακρυσμένα αντικείμενα ενός πελάτη και ενός διακομιστή. Συγκεκριμένα, για να μπορέσει ο πελάτης που βρίσκεται στο ένα υπολογιστή να καλέσει μια μέθοδο του αντικειμένου που βρίσκεται στον απομακρυσμένο υπολογιστή δηλαδή στο διακομιστή και επίσης να δεχτεί τα αποτελέσματα από το απομακρυσμένο αντικείμενο, πρέπει να δημιουργηθούν οι κλάσεις πληρεξούσιος και σκελετός. Ο πληρεξούσιος σχετίζεται μ΄ έναν πελάτη και ο σκελετός μ΄ έναν διακομιστή. Στην περίπτωση που ο πελάτης καλέσει μια μέθοδο του απομακρυσμένου αντικειμένου τότε ο πληρεξούσιος το

οποίο αποτελεί το ανώτερο επίπεδο του συστήματος εμφανίζει το απομακρυσμένο αντικείμενο προς τον πελάτη σαν να ήταν τοπικά και προωθεί την κλήση και τις παραμέτρους της μεθόδου στο σκελετό του διακομιστή. Από την άλλη μεριά, ο σκελετός του διακομιστή δέχεται την κλήση της απομακρυσμένης μεθόδου καθώς και τις παραμέτρους που μεταφέρονται από το πληρεξούσιο του πελάτη. Στην συνέχεια, ο σκελετός εκτελεί την κλήση της μεθόδου προς το αντικείμενο του διακομιστή και τέλος δέχεται τα αποτελέσματα της μεθόδου τα οποία επιστρέφει πίσω στο πληρεξούσιο του πελάτη. Τέλος, ο πληρεξούσιος και ο σκελετός δημιουργούνται αυτόματα από έναν μεταγλωττιστή της Java RMI ο οποίος ονομάζεται `rmic`.

- Το **επίπεδο απομακρυσμένης αναφοράς** (remote reference layer): Αυτό το επίπεδο ασχολείται με τα πρωτόκολλα που πρέπει να χρησιμοποιηθούν για την επικοινωνία σκελετού-πληρεξούσιου. Για παράδειγμα το επίπεδο αυτό μπορεί να υποστηρίξει ένα πρωτόκολλο που συντελεί μόνο στην κλήση μεθόδων σ΄ ένα αντικείμενο (πρωτόκολλο απλής εκπομπής) ή μπορεί να υποστηρίξει ένα πρωτόκολλο μέσω του οποίου αποστέλλονται κλήσεις σε πληθώρα αντικειμένων (πρωτόκολλο πολυεκπομπής).

- Το **επίπεδο μεταφοράς** (transport layer): Το επίπεδο αυτό ασχολείται με την αποστολή δεδομένων που σχετίζονται με τη κλήση της μεθόδου σ΄ ένα απομακρυσμένο αντικείμενο. Το επίπεδο αυτό υποστηρίζει ορισμένους μηχανισμούς μεταφοράς: ο προκαθορισμένος μηχανισμός είναι το TCP/IP, αλλά δεν υπάρχει κάποιος λόγος για τον οποίο να μην μπορεί να χρησιμοποιηθεί κάποιο άλλο κατάλληλο πρωτόκολλο.

Για να κατανοήσουμε την λειτουργία καθενός από τα επίπεδα, αξίζει να ρίξουμε μια ματιά στην επίκληση της μεθόδου `remoteMethod`, η οποία περιλαμβάνει δύο ορίσματα που εφαρμόζονται στο απομακρυσμένο αντικείμενο `remoteObject`.

```
remoteObject.remoteMethod(arg1, arg2)
```

Το αντικείμενο αυτό βρίσκεται στον διακομιστή και ο πελάτης περιλαμβάνει τον κώδικα που παρουσιάζεται στην παραπάνω γραμμή.

Αρχικά, καλείται η απομακρυσμένη μέθοδος που αντιστοιχεί στο `remoteObject` στο επίπεδο πληρεξούσιου/σκελετού, το πληρεξούσιος δηλαδή, συλλέγει τα ορίσματα που χρησιμοποιούνται για την επίκληση της μεθόδου, ενημερώνει το επίπεδο απομακρυσμένης αναφοράς ότι ο κώδικας πρέπει να εκτελεστεί και αποστέλλει τα απαιτούμενα δεδομένα σε αυτό το επίπεδο.

Το επίπεδο απομακρυσμένης αναφοράς θα αποφασίσει αν το αντικείμενο που αναφέρεται βρίσκεται στον τοπικό υπολογιστή ή σε κάποιον απομακρυσμένο. Αν βρίσκεται σε απομακρυσμένο υπολογιστή θα περάσει στο επίπεδο μεταφοράς τα δεδομένα που απαιτούνται για την επίκληση της μεθόδου στον απομακρυσμένο υπολογιστή.

Το επίπεδο μεταφοράς θα εκτελέσει την σύνδεση και την φυσική μεταφορά των δεδομένων που σχετίζονται με την επίκληση της μεθόδου, για παράδειγμα τις τιμές των ορισμάτων της μεθόδου. Όταν το απομακρυσμένο αντικείμενο παραλάβει το μήνυμα, τα δεδομένα που έχουν παραχθεί αποστέλλονται με ανάλογο τρόπο πίσω στον πελάτη που κάλεσε τη μέθοδο. Στο τέλος, το πληρεξούσιο του πελάτη αποκωδικοποιεί τα αποτελέσματα που επέστρεψε η μέθοδος και τα περνά στο πρόγραμμα που προκάλεσε την απομακρυσμένη κλήση.

11.4.2 Μητρώο Αντικειμένων Java RMI

Με βάση την αρχιτεκτονική Java RMI προκύπτει ένα ερώτημα πώς ο πελάτης θα συνδεθεί με ένα απομακρυσμένο αντικείμενο ώστε να καλέσει μια μέθοδος του. Η απάντηση στο ερώτημα αυτό είναι ότι ο πελάτης μπορεί να εντοπίζει το αντικείμενο χρησιμοποιώντας το μητρώο αντικειμένων που διαθέτει η Java RMI. Έτσι, το μητρώο αντικειμένων Java RMI είναι μια υπηρεσία ονομασίας ενός κατανεμημένου συστήματος που βασίζεται στο Java RMI. Το μητρώο πρέπει να εκτελείται σε κάθε διακομιστή απομακρυσμένων αντικειμένων. Επίσης, το μητρώο διατηρεί έναν πίνακα που απεικονίζει ονόματα σε μορφή URL που δείχνουν σε αναφορές απομακρυσμένων αντικειμένων που φιλοξενούνται στον διακομιστή. Οι πελάτες χρησιμοποιούν τις κλάσεις του πακέτου `java.rmi` για να επικοινωνήσουν με το μητρώο του διακομιστή και να εντοπίσουν τα απομακρυσμένα αντικείμενα που φιλοξενεί. Συνεπώς, επιτρέπεται στους πελάτες να προσπελάζουν τα απομακρυσμένα αντικείμενα με την χρήση ενός URL που ακολουθεί το πρωτόκολλο RMI, δηλαδή την χρήση του ονόματος του απομακρυσμένου αντικειμένου. Το URL είναι ένα αλφαριθμητικό της μορφής:

```
rmi://hostname:port/object_name
```

όπου `hostname` είναι το όνομα ή την διεύθυνση IP του υπολογιστή που εκτελείται το μητρώο αντικειμένων Java RMI, `port` είναι ο αριθμός της θύρας στο οποίο ακούει τις αιτήσεις το μητρώο (η προκαθορισμένη τιμή για την θύρα είναι 1099) και `object_name` είναι το όνομα του απομακρυσμένου αντικειμένου εντός του μητρώου αντικειμένων.

Για να μπορέσει ο διακομιστής αντικειμένων να καταχωρήσει και να φιλοξενήσει τα απομακρυσμένα αντικείμενα, πρέπει πρώτα να ξεκινήσει την εκτέλεση του μητρώου αντικειμένων Java RMI στο παρασκήνιο με την χρήση της εντολής `rmiregistry`. Η παραπάνω εντολή το μητρώο ακούει αιτήσεις στην προκαθορισμένη θύρα 1099 του διακομιστή. Στην περίπτωση που επιθυμούμε να χρησιμοποιηθεί κάποια άλλη τιμή για την θύρα στο `rmiregisty` θα πρέπει να ξεκινήσει σε αυτή την επιθυμητή θύρα. Αυτό γίνεται προσθέτοντας στο τέλος της παραπάνω εντολής τον αριθμό της θύρας όπως `rmiregistry 3406`.

Μετά την επιτυχή εκτέλεση του μητρώου αντικειμένων στον διακομιστή, είναι δυνατό να καταχωρήσουμε σε αυτό αντικείμενα με το όνομα τους χρησιμοποιώντας την κλάση `java.rmi.Naming`. Η κλάση αυτή προσφέρει μεθόδους για την αντιστοιχία ονομάτων με απομακρυσμένα αντικείμενα. Οι μέθοδοι είναι οι εξής:

- `public static void bind(String url, Remote obj) throws AlreadyBoundException, MalformedURLException`
- `public static void rebind(String url, Remote obj) throws RemoteException`
- `public static void unbind(String url) throws RemoteException, NotBoundException`
- `public static Remote lookup(String url) throws NotBoundException, RemoteException, MalformedURLException`

Από τους παραμέτρους, το `url` αναφέρεται η διεύθυνση (σε αλφαριθμητική μορφή) του απομακρυσμένου αντικειμένου που χρησιμοποιεί το πρωτόκολλο `rmi` και το `obj` είναι μια αναφορά σε αυτό το αντικείμενο. Η μέθοδος `bind()` συνδέει το όνομα `url` με το απομακρυσμένο αντικείμενο `obj`. Εάν όμως το όνομα χρησιμοποιείται ήδη στο μητρώο, δημιουργείται μια εξαίρεση τύπου `AlreadyBoundException`. Η μέθοδος `rebind()` συνδέει επίσης ένα όνομα με ένα απομακρυσμένο αντικείμενο. Εάν το όνομα χρησιμοποιείται στο μητρώο, η υπάρχουσα σύνδεση αντικαθίσταται. Η μέθοδος `unbind()` αφαιρεί τη σύνδεση ονόματος και απομακρυσμένου αντικειμένου. Εάν το όνομα δεν είχε συνδεθεί με κάποιο αντικείμενο, δημιουργείται μια εξαίρεση τύπου `NotBoundException`. Τέλος, η μέθοδος `lookup()` επιστρέφει μια αναφορά προς το απομακρυσμένο αντικείμενο που είναι συνδεδεμένο με το όνομα που προσδιορίζεται από την παράμετρο `url`. Εάν το όνομα δεν είχε συνδεθεί με ένα αντικείμενο, δημιουργείται μια εξαίρεση τύπου `NotBoundException`.

11.4.3 Ανάπτυξη εφαρμογής Java RMI

Στην ενότητα αυτή θα περιγράψουμε τα βασικά βήματα για την υλοποίηση μιας εφαρμογής πελάτη - διακομιστή βασισμένη στην τεχνολογία Java RMI. Για την καλύτερη κατανόηση της διαδικασίας ανάπτυξης εφαρμογής θα παρουσιάζουμε ένα απλό παράδειγμα προκειμένου ένα πρόγραμμα σε έναν πελάτη να έχει πρόσβαση σε κάποιο απομακρυσμένο αντικείμενο σε ένα διακομιστή. Το απομακρυσμένο αντικείμενο που θα χρησιμοποιήσουμε έχει μια πολύ απλή λειτουργία και το μόνο που κάνει είναι να επιστρέφει τον παραγοντικό ενός αριθμού. Επομένως, τα βήματα για την ανάπτυξη της εφαρμογής είναι τα εξής:

1. Η δημιουργία της απομακρυσμένης διεπαφής. Η διεπαφή Java είναι μια κλάση που λειτουργεί ως πρότυπο για τις υπόλοιπες κλάσεις. Η διεπαφή περιέχει υπογραφές των μεθόδων των οποίων οι υλοποιήσεις τους ορίζονται από άλλες κλάσεις που υλοποιούν την διεπαφή. Η απομακρυσμένη διεπαφή πρέπει να επεκτείνει την διεπαφή `Remote` που βρίσκεται στο πακέτο `java.rmi` και δεν περιέχει σταθερές ούτε μεθόδους. Η διεπαφή `Remote` υπάρχει μόνο για να σημειώσει ποιες μέθοδοι είναι απομακρυσμένες για το σύστημα RMI. Η απομακρυσμένη διεπαφή πρέπει να δηλωθεί ως δημόσια και πρέπει να ορίζει όλες τις μεθόδους των απομακρυσμένων αντικειμένων που να είναι προσβάσιμες σε όλους τους πελάτες. Τέλος, όλες οι μέθοδοι της απομακρυσμένης διεπαφής πρέπει να ορίζουν ότι δημιουργούν μια εξαίρεση τύπου `RemoteException`. Η εξαίρεση αυτή παράγεται όταν εμφανιστεί κάποιο σφάλμα κατά τη διάρκεια της απομακρυσμένης επικοινωνίας. Οι πιο πιθανές αιτίες τέτοιων εξαιρέσεων είναι τα προβλήματα που ίσως εμφανιστούν κατά την διάρκεια της διαδιεργασιακής επικοινωνίας (όπως αστοχίες πρόσβασης, αστοχίες σύνδεσης) και προβλήματα που έχουν σχέση με τις επικλήσεις απομακρυσμένων μεθόδων (όπως δεν βρίσκει το απομακρυσμένο αντικείμενο, το πληρεξούσιο ή το σκελετό). Για παράδειγμα, ο τυπικός κώδικας για το βήμα αυτό παρουσιάζεται παρακάτω:

```
import java.rmi.*;
public interface Factorial extends Remote
{
        // ypografh ths apomakrysmenhs methodoy
        public int fact(int number) throws RemoteException;
}
```

2. Η δημιουργία κλάσης για την υλοποίηση της απομακρυσμένης διεπαφής. Η κλάση αυτή

που υλοποιεί την απομακρυσμένη διεπαφή του αντικειμένου που ορίσαμε στο προηγούμενο βήμα επεκτείνει την κλάση `RemoteObject` που βρίσκεται στο πακέτο `java.rmi.server`. Στην πράξη, οι περισσότερες υλοποιήσεις επεκτείνουν την υποκλάση `UnicastRometeObject` αφού η κλάση αυτή υποστηρίζει την σημειακή επικοινωνία δηλαδή επιτρέπει την επικοινωνία μεταξύ ενός πελάτη και ενός απλού αντικειμένου εγκατεστημένου σ΄ έναν διακομιστή. Στην συνέχεια, στην κλάση υλοποίησης πρέπει να ορίσουμε έναν κατασκευαστή για την δημιουργία του απομακρυσμένου αντικειμένου. Επίσης, από την στιγμή που ο κατασκευαστής αυτός αντιπροσωπεύει ένα απομακρυσμένο αντικείμενο πρέπει να δηλωθεί ότι μπορεί να δημιουργήσει μια εξαίρεση τύπου `RemoteException` όπως στις μέθοδους που δηλώσαμε στην απομακρυσμένη διεπαφή. Τέλος, στην ίδια κλάση ορίζουμε τον κώδικα υλοποίησης για κάθε μέθοδο που ορίζεται στην απομακρυσμένη διεπαφή. Οι επικεφαλίδες των μεθόδων πρέπει να ταυτίζονται με τις υπογραφές που ορίστηκαν στην διεπαφή. Επίσης, οι μέθοδοι πρέπει να δηλωθούν ότι μπορούν να δημιουργούν εξαίρεση `RemoteException` εφόσον οι μεθόδοι αυτοί κληθούν απομακρυσμένα. Για παράδειγμα, ο τυπικός κώδικας για το βήμα αυτό παρουσιάζεται παρακάτω:

```
import java.rmi.*;
import java.rmi.server.*;

// H klash ayth ylopoiei thn apomakrysmenh diepafh Factorial
public class FactorialImpl extends UnicastRemoteObject implements Factorial
{
        // Kataskeyasths
        public FactorialImpl() throws RemoteException
        {
                super();
        }

        // Kodikas ylopoihshs ths apomakrysmenhs methodoy
        public int fact(int number) throws RemoteException
        {

                int result = 1;

                for(int i = 1; i <= number; i++)
                        result = result * i;
                return (result);
        }
}
```

3. Δημιουργία προγράμματος διακομιστή. Το πρόγραμμα διακομιστή δημιουργεί ένα ή περισσότερα απομακρυσμένα αντικείμενα από την κλάση υλοποίηση του προηγούμενου βήματος και καταχωρεί τα αντικείμενα στο μητρώο αντικειμένων του RMI. Το μητρώο αντικειμένων του RMI όπως είναι γνωστό είναι ένα πρόγραμμα που τρέχει στο παρασκήνιο, παρακολουθεί όλα τα κατανεμημένα αντικείμενα που υπάρχουν και συσχετίζει καθένα από αυτά με ένα όνομα. Η καταχώριση και η συσχέτιση του απομακρυσμένου αντικειμένου με ένα όνομα γίνεται με την στατική μέθοδο `rebind` της κλάσης `Naming` (που βρίσκεται στο πακέτο `java.rmi`). Η μέθοδος αυτή παίρνει δύο παραμέτρους: ένα αλφαριθμητικό που αναφέρεται στο όνομα του απομακρυσμένου αντικειμένου της μορφής URL με πρωτόκολλο `rmi` και μια αναφορά στο απομακρυσμένο αντικείμενο. Μετά από αυτή την διαδικασία, τα απομακρυσμένα αντικείμενα είναι έτοιμα να δεχτούν κλήσεις μεθόδων από προγράμματα πελάτες. Για παράδειγμα, ο τυπικός κώδικας για το βήμα αυτό που περιέχει μια μέθοδο την `main` παρουσιάζεται παρακάτω:

```
import java.rmi.*;
public class FactorialServer
{
        private static final String HOST = "localhost";
        public static void main(String[] args) throws Exception
        {
                // Dhmioyrgia antikeimenoy
                FactorialImpl robj = new FactorialImpl();
                // Eggrafh toy antikeimenoy sthn yphresia onomasias RMI
                // kato apo to onoma Factorial
                String rmiObjectName = "rmi://" + HOST + "/Factorial";
                Naming.rebind(rmiObjectName,robj);
        }
}
```

4. Το τελευταίο βήμα είναι η δημιουργία προγράμματος πελάτη η οποία αποκτά αναφορά στο απομακρυσμένο αντικείμενο για να καλέσει τις μεθόδους. Για να αποκτήσει μια αναφορά στο απομακρυσμένο αντικείμενο πρέπει να συνδεθεί στο μητρώο του RMI και να ζητήσει το αντικείμενο με το όνομα του. Για να γίνει αυτό, χρησιμοποιείται η στατική μέθοδο `lookup` της κλάσης `Naming` και παίρνει το όνομα του αντικειμένου ως όρισμα. Το όνομα πρέπει να έχει την ίδια μορφή με το URL όπως στην εγγραφή του αντικειμένου στο μητρώο του προγράμματος διακομιστή. Η μέθοδος `lookup` θα επιστρέψει μια αναφορά `Remote`, η οποία πρέπει να μετατραπεί σε αναφορά τύπου ονόματος απομακρυσμένης διεπαφής που ορίζαμε

στο βήμα 1. Τέλος, απομένει η κλήση της μεθόδου στο απομακρυσμένο αντικείμενο μέσω της απομακρυσμένης αναφοράς. Για παράδειγμα, ο τυπικός κώδικας για το βήμα αυτό που περιέχει μια μέθοδο την `main` παρουσιάζεται παρακάτω:

```
import java.rmi.*;
public class FactorialClient
{
        private static final String HOST = "localhost";
        public static void main(String[] args)
        {
                try
                {
                        // Anazhtish toy apomakrysmenoy antikeimenoy kai metatropi thn
                        // anafora toy se klash apomakrysmenhs diepafhs Factorial
                        Factorial ref = (Factorial) Naming.lookup("rmi://" + HOST + "/Factorial");
                        // Klhsh ths apomakrysmenhs methodoy
                        int result = ref.fact(3);
                        System.out.println("The␣factorial␣of␣3␣is␣" + result);
                }
                catch (RemoteException re)
                {
                        System.out.println("Remote␣Exception");
                        re.printStackTrace();
                }
                catch (Exception e)
                {
                        System.out.println("Other␣Exception");
                        e.printStackTrace();
                }
        }
}
```

Επίσης, παρακάτω παρουσιάζουμε τα βήματα που πρέπει ακόμα να ακολουθήσουμε για την μεταγλώττιση και εκτέλεση μιας εφαρμογής RMI.

1. Μεταγλώττιση όλων των αρχείων java.

2. Μεταγλώττιση του αρχείου της κλάσης που υλοποιεί την διεπαφή (π.χ. `FactorialImpl`) με την βοήθεια του μεταγλωττιστή RMI. Η διαδικασία αυτή δημιουργεί τα αρχεία στελέχους και σκελετού που είναι απαραίτητα για την εκτέλεση της εφαρμογής σύμφωνα με την αρχιτεκτονική του συστήματος RMI. Το εργαλείο του μεταγλωττιστή RMI είναι γνωστό ως `rmic`. Έτσι, εκτελώντας την εντολή

```
rmic FactorialImpl
```

θα παράγει δύο αρχεία με ονόματα `FactorialImpl_skel.class` και `FactorialImpl_stub.class`. Επίσης, πρέπει να σημειώσουμε ότι ο μεταγλωττιστής `rmic` προσθέτει τα επιθέματα `_skel` και `_stub` στο τέλος των ονομάτων αρχείων. Ο σκελετός της κλάσης θα πρέπει να τοποθετηθεί στο διακομιστή και το πληρεξούσιο να τοποθετηθεί στον πελάτη.

3. Εκκίνηση την υπηρεσία ονομασίας ή μητρώου του RMI στον διακομιστή εκτελώντας την εντολή `rmiregistry` και θα έχει σαν αποτέλεσμα να εκτελείται η υπηρεσία ονομασίας στο παρασκήνιο.

4. Εκτέλεση του αρχείου που περιέχει τον κώδικα του διακομιστή. Το απομακρυσμένο αντικείμενο που σχετίζεται με τον διακομιστή είναι τώρα έτοιμο να δεχτεί τις κλήσεις μεθόδων από τον κώδικα που τρέχει σε άλλους υπολογιστές.

5. Εκτέλεση του αρχείου που περιέχει τον κώδικα του πελάτη. Αυτός θα συνδεθεί με το απομακρυσμένο αντικείμενο που είναι πλέον διαθέσιμο και θα εκτελέσει την απαιτούμενη επεξεργασία.

11.4.4 Επιχειρηματική Εφαρμογή RMI

Η τεχνολογία RMI είναι μια καλή υποψήφια λύση για ένα συστατικό λογισμικού σε επίπεδο υπηρεσιών. Μια πιο σύνθετη και επιχειρηματική εφαρμογή είναι για παράδειγμα η υπηρεσία διαχείριση καταλόγου διευθύνσεων ηλεκτρονικού ταχυδρομείου. Γενικά, σε μια επιχειρηματική εφαρμογή ο διακομιστής αντικειμένων παρέχει απομακρυσμένους μεθόδους έτσι ώστε να επιτρέψει στα προγράμματα πελάτες να αναζητήσουν ή να ενημερώσουν δεδομένα σε μια βάση δεδομένων. Συγκεκριμένα, στην υπηρεσία διαχείριση καταλόγου διευθύνσεων, ο διακομιστής διαθέτει μια βάση δεδομένων που αποθηκεύει ζεύγη τιμών ονόματα πελατών/διευθύνσεις ηλεκτρονικού ταχυδρομείου και ο διακομιστής παρέχει απομακρυσμένους μεθόδους ώστε οι πελάτες να αναζητήσουν την διεύθυνση ηλεκτρονικού ταχυδρομείου ενός πελάτη, να προσθέσουν μια καινούργια διεύθυνση ή να διαγράψουν μια διεύθυνση ηλεκτρονικού ταχυδρομείου από την βάση δεδομένων.

Παρακάτω παρουσιάζουμε τμήματα κώδικα για την υλοποίηση της απομακρυσμένης υπηρεσίας διαχείρισης καταλόγου. Το επόμενο τμήμα κώδικα είναι ο ορισμός της διεπαφής υπηρεσίας που

ορίζει τις υπογραφές τριών μεθόδων: μια μέθοδο `getEmail` η οποία με δεδομένο το όνομα ενός πελάτη σε μορφή αλφαριθμητικού επιστρέφει τη διεύθυνση ηλεκτρονικού ταχυδρομείου που συνδέεται με το όνομα, μια μέθοδο `putEmail` που δέχεται ένα ζεύγος ονόματος/διεύθυνσης και το αποθηκεύσει στην βάση δεδομένων και μια μέθοδο `removeEmail` που δέχεται το όνομα και απομακρύνει από την βάση δεδομένων το αντίστοιχο ζεύγος τιμών που αντιστοιχεί στο όνομα.

```
import java.rmi.*;
public interface ManagementEmails extends Remote
{
        // ypografes ton apomakrysmenon methodon
        String getEmail(String name) throws RemoteException;
        void putEmail(String name, String email) throws RemoteException;
        void removeEmail(String name) throws RemoteException;
}
```

Στην συνέχεια δίνεται ο κώδικας για την υλοποίηση της απομακρυσμένης υπηρεσίας. Αποθηκεύει τα ζεύγη όνομα/διεύθυνση σε μια βάση δεδομένων (π.χ. σε ένα πίνακα κατακερματισμού) και ορίζει τις υλοποιήσεις των μεθόδων της απομακρυσμένης υπηρεσίας.

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.Hashtable;

// H klash ayth ylopoiei thn apomakrysmenh diepafh ManagementEmails
public class ManagementEmailsImpl extends UnicastRemoteObject implements ManagementEmails
{
        private Hashtable storeEmails;

        // Kataskeyasths
        public ManagementEmailsImpl() throws RemoteException
        {
                // Dhmioyrgia bashs dedomenon e-mails
                storeEmails = new Hashtable();
                storeEmails.put("Panos","panosm@uom.gr");
                storeEmails.put("John","johnf@gmail.com");
        }

        // Kodikas ylopoihshs ths apomakrysmenhs methodoy getEmail
        public String getEmail(String name) throws RemoteException
        {
                return (String) storeEmails.get(name);
        }
```

```
        // Kodikas ylopoihshs ths apomakrysmenhs methodoy putEmail
        public void putEmail(String name, String email) throws RemoteException
        {
                storeEmails.put(name,email);
        }

        // Kodikas ylopoihshs ths apomakrysmenhs methodoy removeEmail
        public void removeEmail(String name) throws RemoteException
        {
                storeEmails.remove(name);
        }
}
```

Παρακάτω παρουσιάζεται το τμήμα κώδικα του διακομιστή που κατασκευάζει ένα απομακρυσμένο αντικείμενο και καταχωρεί το αντικείμενο αυτό στο μητρώο RMI ώστε να μπορούν να το προσπελάσουν οι πελάτες.

```
import java.rmi.*;
public class ManagementEmailsServer
{
        private static final String HOST = "localhost";
        public static void main(String[] args) throws Exception
        {
                // Dhmioyrgia antikeimenoy
                ManagementEmailsImpl robj = new ManagementEmailsImpl();
                // Eggrafh toy antikeimenoy sthn yphresia onomasias RMI
                // kato apo to onoma ManagementEmails
                String rmiObjectName = "rmi://" + HOST + "/ManagementEmails";
                Naming.rebind(rmiObjectName,robj);
        }
}
```

Τέλος, δίνεται παρακάτω το τμήμα κώδικα του πελάτη που δημιουργεί μια αναφορά στο απομακρυσμένο αντικείμενο του διακομιστή και καλεί τις τρεις μεθόδους του αντικειμένου.

```
import java.rmi.*;
public class ManagementEmailsClient
{
        private static final String HOST = "localhost";
        public static void main(String[] args)
        {
                try
                {
                        // Anazhtish toy apomakrysmenoy antikeimenoy kai metatropi thn
                        // anafora toy se klash apomakrysmenhs diepafhs ManagementEmails
                        ManagementEmails ref = (ManagementEmails) Naming.lookup("rmi://" + HOST + "/Manage
```

```
                        // Klhsh ton apomakrysmenon methodon
                        String result = ref.getEmail("Panos");
                        System.out.println("The email of the Panos is " + result);
                        ref.putEmail("George","george@yahoo.com");
                        System.out.println("The insertion of the George's email is OK");
                        ref.removeEmail("John");
                        System.out.println("The deletion of the John' email is OK");
                        System.out.println("The email of the George is " + ref.getEmail("George"));
                        System.out.println("The email of the John is " + ref.getEmail("John"));
                }
                catch (RemoteException re)
                {
                        System.out.println("Remote Exception");
                        re.printStackTrace();
                }
                catch (Exception e)
                {
                        System.out.println("Other Exception");
                        e.printStackTrace();
                }
        }
}
```

11.4.5 Βήματα για την Ανάπτυξη μιας Εφαρμογής RMI

Έχοντας εξετάσει αναλυτικά τα βήματα για την ανάπτυξη μιας εφαρμογής κατανεμημένων αντικειμένων με το Java RMI, θα περιγράψουμε παρακάτω τα βήματα ανάπτυξης προγραμμάτων για τις δυο πλευρές (πελάτης και διακομιστής) συνοπτικά. Τα βήματα που περιγράφουμε παρακάτω αναφέρεται σε μια εφαρμογή με όνομα `Sample`. Συνεπώς τα παρακάτω βήματα μπορούν να εφαρμοστούν σε μια οποιαδήποτε εφαρμογή αντικαθιστώντας το όνομα `Sample` με το όνομα της εφαρμογής. Η ανάπτυξη ενός συστήματος απομακρυσμένων αντικειμένων με την χρήση του Java RMI περιλαμβάνει τα ακόλουθα στάδια:

1. Ανάπτυξη της διεπαφής που περιγράφει τις υπηρεσίες που παρέχονται από τα απομακρυσμένα αντικείμενα σε αρχείο `SampleInterface.java`. Μεταγλώττιση του κώδικα.

2. Ανάπτυξη μίας κλάσης που υλοποιεί την απομακρυσμένη διεπαφή σε αρχείο `SampleImpl.java`. Μεταγλώττιση του κώδικα.

3. Εκτέλεση του μεταγλωττιστή RMI (`rmic`) στην απομακρυσμένη κλάση ώστε να παραχθούν

τα στελέχη και τα πληρεξούσια και στην συνέχεια διάθεσή τους στον πελάτη και στον διακομιστή. Δηλαδή εκτελούμε την εντολή: `rmic SampleImpl`.

4. Υλοποίηση του κώδικα διακομιστή σε αρχείο `SampleServer.java` που δημιουργεί και εγγράφει τα κατανεμημένα αντικείμενα στο μητρώο του RMI. Μεταγλώττιση του κώδικα.

5. Υλοποίηση του κώδικα για τον πελάτη σε αρχείο `SampleClient.java` που χρησιμοποιεί τον κατάλογο RMI για να συνδεθεί με το απομακρυσμένο αντικείμενο. Μεταγλώττιση του κώδικα.

6. Εκτέλεση του μητρώου RMI στον υπολογιστή διακομιστή.

7. Εκτέλεση του αρχείου διακομιστή.

8. Εκτέλεση του αρχείου πελάτη.

11.4.6 Σύγκριση μεταξύ Java RMI και Java Socket APIs

Το ενδιάμεσο λογισμικό Java RMI ως εκπρόσωπος του μοντέλου κατανεμημένων αντικειμένων είναι ένα αποτελεσματικό εργαλείο για την ανάπτυξη κατανεμημένων εφαρμογών. Όμως αντί αυτού μπορεί να χρησιμοποιηθεί το λογισμικό Java Socket ως εκπρόσωπος του μοντέλου μεταβίβασης μηνυμάτων για ταχεία ανάπτυξη εφαρμογών. Γι αυτό τα δύο ενδιάμεσα λογισμικά έχει τα δικά του πλεονεκτήματα και μειονεκτήματα που θα παρουσιάζουμε παρακάτω.

Το λογισμικό Java Socket έχει τα παρακάτω πλεονεκτήματα:

- Το λογισμικό υποδοχών χρησιμοποιεί απλούς και αποτελεσματικούς μηχανισμούς μεταβίβασης μηνυμάτων και αυτό έχει σαν αποτέλεσμα λιγότερη επιβάρυνσης εκτέλεσης. Για εφαρμογές που απαιτούν υψηλή απόδοση, το λογισμικό υποδοχών είναι η μόνη λύση.

- Το λογισμικό υποδοχών διεκολύνει την ταχεία ανάπτυξη κατανεμημένων εφαρμογών.

- Επειδή το λογισμικό υποδοχών λειτουργεί σε χαμηλό επίπεδο (σε επίπεδο λειτουργικού συστήματος) υποστηρίζει ανεξαρτησία πλατφόρμας και γλώσσας.

Είδαμε στο προηγούμενο κεφάλαιο να υλοποιούμε ορισμένα πρωτόκολλα μέσω λογισμικού υποδοχών που οδηγούν σε αποτελεσματικές εφαρμογές, ωστόσο όμως έχουν ορισμένα μειονεκτήματα:

- Όταν η λειτουργία μιας εφαρμογής αλλάζει, το βάρος της προσπάθειας για την προσθήκη καινούριου κώδικα που ανταποκρίνεται σε ένα συγκεκριμένο καινούριο μήνυμα στο πρωτόκολλο, μπορεί να είναι πολύ μεγάλο. Επιπλέον, αυτός ο κώδικας θα πρέπει να αλλάξει τόσο στον πελάτη όσο και στον διακομιστή.

- Είναι δύσκολη η εισαγωγή τέτοιων πρωτοκόλλων σε ένα αντικειμενοστρεφές μοντέλο.

- Η χρήση ενός πρωτοκόλλου ειδικού σκοπού συχνά συνεπάγεται πολύ κώδικα στον πελάτη: ο κώδικας επιφορτίζεται με την αποστολή και την επεξεργασία συμβολοσειρών που αντιπροσωπεύουν μεμονωμένες εντολές του πρωτοκόλλου.

- Πρωτόκολλα μέσω λογισμικού υποδοχών είναι εμφανής στο προγραμματιστή ο τρόπος επικοινωνίας που χρησιμοποιείται για τη σύνδεση πελατών και διακομιστών όπως υποδοχές και θύρες διακομιστών που μετέχουν στη μεταφορά ενός μηνύματος από το έναν υπολογιστή στον άλλο.

Από την άλλη πλευρά το λογισμικό Java RMI παρέχει τα παρακάτω πλεονεκτήματα:

- Η τεχνολογία κατανεμημένων αντικειμένων κρύβει τις επικοινωνίες χαμηλού επιπέδου από το προγραμματιστή μέσω της αυτόματης παραγωγής προγραμμάτων πληρεξούσιου και σκελετού.

- Το λογισμικό RMI παρέχει μια αφαίρεση που διεκολύνει την διαδικασία ανάπτυξης λογισμικού. Έτσι, προγράμματα αναπτυγμένα με ένα υψηλότερο επίπεδο αφαίρεσης είναι περισσότερα κατανοητά και εύκολα συντηρούνται.

Τα μειονεκτήματα του λογισμικού κατανεμημένων αντικειμένων είναι τα εξής:

- Η απόδοση των κατανεμημένων αντικειμένων σε σχέση με τις τεχνολογίες υποδοχών είναι χαμηλή. Αυτό οφείλεται στο γεγονός ότι το RMI απαιτεί πρόσθετη υποστήριξη λογισμικού περιλαμβάνοντας τα προγράμματα πληρεξούσιο, σκελετό και μητρώο αντικειμένων που έχουν σαν αποτέλεσμα την επιβάρυνση του χρόνου εκτέλεσης.

- Το λογισμικό Java RMI απαιτεί στενή υποστήριξη Java. Αυτό σημαίνει ότι μια εφαρμογή που θα υλοποιηθεί με την χρήση Java RMI πρέπει να γραφτεί σε Java και μπορεί να εκτελεστεί μόνο σε πλατφόρμες Java.

- Η τεχνολογία Java RMI είναι δύσκολη και ακατάλληλη για ανάπτυξη ετερογενών συστημάτων επειδή δεν βασίζεται σε ανοικτά πρότυπα

Η επιλογή ενός κατάλληλου ενδιάμεσου λογισμικού είναι μια βασική απόφαση για την σχεδίαση και ανάπτυξη των κατανεμημένων εφαρμογών. Εξαιτίας της ευκολίας ανάπτυξης εφαρμογών με την χρήση RMI, το RMI είναι μια καλή λύση για την ταχεία ανάπτυξη ενός πρωτοτύπου για μια εφαρμογή.

Κεφάλαιο 12

Προγραμματισμός Υπηρεσιών Ιστού

Σε αντίθεση με μια παραδοσιακή εφαρμογή Ιστού που σχεδιάζεται για να αλληλεπιδρά με χρήστη πελάτη, μια **υπηρεσία Ιστού** (web service) είναι εφαρμογή (ή τμήμα εφαρμογής) που διατίθεται για να χρησιμοποιείται από άλλη εφαρμογή μέσω Ιστού. Μια υπηρεσία Ιστού παρέχει μια διεπαφή υπηρεσίας που δίνει την δυνατότητα στα προγράμματα πελάτες να επικοινωνούν με τους διακομιστές σε ένα γενικό τρόπο από ότι με τους περιηγητές Ιστού. Με άλλα λόγια, τα προγράμματα πελάτες προσπελάζουν τις λειτουργίες που παρέχει η διεπαφή της υπηρεσίας Ιστού με μηνύματα αίτησης και απάντησης εκφρασμένες σε μορφή XML και συνήθως τα μηνύματα αυτά μεταφέρονται μέσω του πρωτοκόλλου Διαδικτύου HTTP. Συνεπώς, παρατηρούμε ότι οι υπηρεσίες Ιστού επεκτείνει το μοντέλο κατανεμημένων αντικειμένων σε ένα περισσότερο γενικό τρόπο και επίσης μπορούν να χρησιμοποιηθούν εύκολα στην ανάπτυξη εφαρμογών για το Διαδίκτυο. Έτσι, οι υπηρεσίες Ιστού διαφέρουν από τις τεχνολογίες κατανεμημένων αντικειμένων όπως Java RMI, CORBA και DCOM στο γεγονός ότι βασίζονται σε απλά ανοικτά πρότυπα όπως XML και HTTP που έχουν διαλειτουργική υποστήριξη για την ανάπτυξη κατανεμημένων εφαρμογών σε ετερογενή περιβάλλοντα. Στο υπόλοιπο του κεφαλαίου θα παρουσιάζουμε την αρχιτεκτονική και τις βασικές τεχνολογίες που χρησιμοποιούνται στις υπηρεσίες Ιστού. Τέλος, θα παρουσιάζουμε το ενδιάμεσο λογισμικό που χρησιμοποιείται για την ανάπτυξη εφαρμογών υπηρεσιών Ιστού.

12.1 Ορισμός

Μια υπηρεσία Ιστού είναι μια διεπαφή με κάποια εφαρμογή που επιτυγχάνεται με τη βοήθεια τεχνολογιών Διαδικτύου, συνήθως τεχνολογιών Ιστού. Γενικά, μια διεπαφή υπηρεσία Ιστού αποτελείται από μια συλλογή λειτουργιών που μπορούν να χρησιμοποιηθούν από ένα πρόγραμμα πελάτη στο Διαδίκτυο. Τυπικά παραδείγματα υπηρεσιών Ιστού είναι:

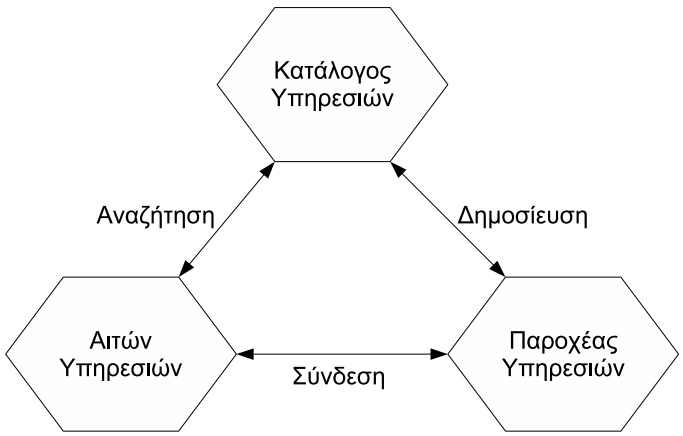
- ένα αίτημα για την τιμή μιας μετοχής ή ενός μερίσματος,
- ένα αίτημα για αναφορές πρόβλεψης καιρού,
- ένα αίτημα για την παραγγελία και αποστολή ενός βιβλίου.

Διάφοροι οργανισμοί εμπλέκονται στον καθορισμό προτύπων για τις υπηρεσίες Ιστού με αποτέλεσμα αρκετοί ορισμοί να έχουν διατυπωθεί, με κυρίαρχο αυτόν της ομάδας World Wide Web Consortium (W3C) Web Services Architecture Working Group:

'' Η υπηρεσία ιστού (web service) είναι ένα λογισμικό συστήματος της οποίας οι δημόσιες διεπαφές και οι συνδέσεις της είναι δυνατόν να οριστούν, να περιγράφουν και να εντοπισθούν από τεχνουργήματα XML και υποστηρίζουν απευθείας αλληλεπίδραση με άλλα συστήματα λογισμικού χρησιμοποιώντας μηνύματα XML που μεταφέρονται μέσω των πρωτοκόλλων Διαδικτύου.''

Σύμφωνα με τον ορισμό, η υπηρεσία Ιστού ορίζεται σαν ένας πόρος, όχι μόνο μέσα στον Διαδίκτυο αλλά και σε επίπεδο ενός τοπικού εσωτερικού δικτύου ή **εσωδικτύου** (intranet) ο οποίος παρέχεται από κάποιον οργανισμό. Σημαντικό είναι το γεγονός της διαλειτουργικότητας ανάμεσα στις διαφορετικές υπολογιστικές πλατφόρμες από τη μεριά του παροχέα της υπηρεσίας και του τελικού χρήστη αυτής.

Ένα άλλο σημαντικό σημείο, είναι ότι οι λεπτομέρειες υλοποίησης και ανάπτυξης μιας πλατφόρμας υπηρεσιών Ιστού, δεν είναι σχετικές με ένα πρόγραμμα που επικαλείται την υπηρεσία. Μια υπηρεσία Ιστού είναι διαθέσιμη μέσω των δημοσίων διεπαφών της, στις οποίες είναι δυνατή η κλήση. Η σχέση αυτή είναι ανάλογη με αυτή του προγράμματος **περιηγητή Ιστού** (web browser) και του **διακομιστή εφαρμογών Ιστού** (web application server). Το πρόγραμμα περιηγητή Ιστού αγνοεί εάν ο διακομιστής Ιστού είναι ο Apache Tomcat ή ο IIS της Microsoft, αρκεί το αποτέλεσμα να είναι εκφρασμένο σε γλώσσα HTML ή με ένα περιορισμένο σύνολο τύπων MIME. Παρόμοια, ο διακομιστής Ιστού αγνοεί ποιος χρήστης τον χρησιμοποιεί.



Σχήμα 12.1: Υπηρεσιοστρεφή αρχιτεκτονική

12.2 Υπηρεσιοστρεφή Αρχιτεκτονική

Το πρότυπο που χρησιμοποιείται για την περιγραφή των υπηρεσιών Ιστού βασίζεται κυρίως στο κλασικό επιχειρηματικό μοντέλο το οποίο έχουν υιοθετήσει οι διάφοροι ιδιωτικοί οργανισμοί. Αυτό το πρότυπο ονομάζεται **υπηρεσιοστρεφή αρχιτεκτονική** (service oriented architecture - SOA) και παρουσιάζεται στο Σχήμα 12.1. Από το παραπάνω σχήμα απεικονίζει τους κύριους ρόλους και τις λειτουργίες σε ένα πρότυπο SOA.

Έτσι, στην υπηρεσιοστρεφή αρχιτεκτονική υπάρχουν τρεις ρόλοι: έναν **αιτούντα υπηρεσιών** (service requestor), έναν **παροχέα υπηρεσιών** (service provider) και έναν **κατάλογο υπηρεσιών** (service registry). Πιο αναλυτική περιγραφή των ρόλων παρουσιάζεται παρακάτω.

- Ο αιτών υπηρεσιών (ή μερικές φορές ονομάζεται πελάτης) είναι εκείνος που αρχίζει όλη την διαδικασία. Έτσι, είναι υπεύθυνος για την αναζήτηση μιας περιγραφής υπηρεσιών Ιστού που έχει δημοσιευτεί σε έναν ή περισσότερους καταλόγους υπηρεσιών. Στην συνέχεια, ενώ έχει εντοπίσει τον κατάληλλο παροχέα, δημιουργεί σύνδεση με αυτόν, καλεί την υπηρεσία και λαμβάνει τα αποτελέσματα.

- Ο παροχέας υπηρεσιών (ή διακομιστής) ο οποίος είναι υπεύθυνος για τη δημιουργία μιας περιγραφής υπηρεσιών Ιστού, δημοσιεύοντας της σε έναν ή περισσότερους καταλόγους υπηρεσιών μέσω του οποίου γίνονται διαθέσιμες αυτές οι υπηρεσίες. Επιπλέον, είναι υπ-

εύθυνος για την λήψη των μηνυμάτων κλήσης για την υπηρεσία από έναν ή περισσότερους αιτούντες και της παροχής των απαραίτητων αποτελεσμάτων.

- Ο κατάλογος υπηρεσιών αναλαμβάνει το ρόλο της διασύνδεσης ανάμεσα στον παροχέα και τον αιτών υπηρεσιών. Συγκεκριμένα, ο κατάλογος είναι υπεύθυνος για τη δημοσίευση μιας περιγραφής υπηρεσιών από τον παροχέα υπηρεσιών και επιτρέπει στον αιτών υπηρεσιών να αναζητήσει ένα σύνολο από περιγραφές υπηρεσιών που περιέχονται μέσα στο κατάλογο.

Οι λειτουργίες που παρέχονται μέσω αυτής της αρχιτεκτονικής είναι οι ακόλουθες:

- Η **δημοσίευση** (publish) η οποία είναι μια ενέργεια καταλόγου υπηρεσιών κατά την οποία ένας παροχέας υπηρεσιών δημοσιεύσει μια περιγραφή υπηρεσίας σε ένα κατάλογο, κοινοποιώντας όλες τις λεπτομέρειες που την αφορούν.

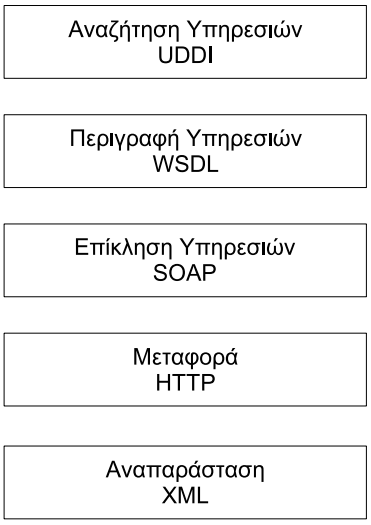
- Η **αναζήτηση** (find) η οποία πραγματοποιείται από τους αιτών και καταλόγου υπηρεσιών. Έτσι, ο αιτών υπηρεσιών δηλώνει τα κριτήρια της αναζήτησης όπως για παράδειγμα τον τύπο και ποιότητα της υπηρεσίας. Στην συνέχεια, ο κατάλογος υπηρεσιών προσπαθεί να ταιριάξει τα κριτήρια αναζήτησης με τις περιγραφές υπηρεσιών που έχουν δημοσιευθεί στον κατάλογο. Το αποτέλεσμα της αναζήτησης αυτής είναι μια λίστα από περιγραφές υπηρεσιών που ταιριάζουν στα συγκεκριμένα κριτήρια αναζήτησης.

- Η **σύνδεση** (bind) η οποία λαμβάνει μια σχέση πελάτη - διακομιστή ανάμεσα στον αιτών και τον παροχέα υπηρεσιών. Οι δύο πλευρές κάνουν τις απαραίτητες διαπραγματεύσεις έτσι ώστε ο αιτών να έχει πρόσβαση και να καλέσει τις υπηρεσίες του παροχέα και ο παροχέας να επιστρέψει τα αποτελέσματα.

Το σημαντικό στοιχείο της παραπάνω αρχιτεκτονικής είναι η περιγραφή της υπηρεσίας Ιστού. Στην ουσία όλα περιστρέφονται γύρω από αυτή. Αυτή είναι που δημοσιεύεται, αυτή αναζητά κανείς όταν ψάχνει μια υπηρεσία Ιστού και με βάση αυτή γίνεται η κλήση.

12.3 Στοίβα Υπηρεσιών Ιστού

Η **στοίβα υπηρεσιών Ιστού** (web services stack) περιλαμβάνει πολλές αλληλοσυσχετιζόμενες τεχνολογίες. Υπάρχουν πολλοί τρόποι για να αναπαρασταθούν οι τεχνολογίες αυτές



Σχήμα 12.2: Στοίβα υπηρεσιών Ιστού

όπως πολλοί είναι και οι τρόποι για να κατασκευαστούν και να χρησιμοποιηθούν οι υπηρεσίες Ιστού. Στο Σχήμα 12.2 παρουσιάζεται η στοίβα των τεχνολογιών που χρησιμοποιείται στις υπηρεσίες Ιστού και η οποία αποτελείται από πέντε επίπεδα:

- Το επίπεδο αναπαράστασης. Σε αυτό το επίπεδο χρησιμοποιείται η γλώσσα XML (eXtensible Markup Language) που είναι η πιο διαδεδομένη για την αναπάρασταση των δεδομένων και των μηνυμάτων που ανταλάσσονται ανάμεσα στους πελάτες και στους παροχείς υπηρεσιών ή υπηρεσιών Ιστού. Συγκεκριμένα, η γλώσσα XML περιγράφει δεδομένα, μηνύματα, γλώσσες και μεταδεδομένα. Τέλος, η XML είναι το βασικό στοιχείο για την ανάπτυξη υπηρεσιών Ιστού.

- Το επίπεδο μεταφοράς. Το επίπεδο αυτό περιλαμβάνει τεχνολογίες που χρησιμοποιούνται για την μεταφορά μηνυμάτων και δεδομένων που είναι εκφρασμένοι σε XML μέσω του Διαδικτύου. Οι τεχνολογίες αυτές είναι TCP, HTTP, SMTP και FTP. Το πρωτόκολλο HTTP χρησιμοποιείται πιο συχνά στις υπηρεσίες Ιστού για μεταφορά μηνυμάτων.

- Το επίπεδο επίκλησης υπηρεσιών. Η επίκληση μιας υπηρεσίας Ιστού περιλαμβάνει την μεταβίβαση μηνυμάτων μεταξύ του αιτών (ή πελάτη) και του παροχέα υπηρεσιών (ή διακομιστή). Το επίπεδο αυτό ασχολείται με την περιγραφή της δομής των μηνυμάτων που χρησιμοποιούνται κατά την φάση αποστολής ή λήψης προς ή από μια υπηρεσία Ιστού. Με

άλλα λόγια, ποια στοιχεία θα περιέχονται σε μια αίτηση υπηρεσία και ποια στοιχεία θα περιέχονται σε μια απάντηση. Συνεπώς, στο επίπεδο αυτό χρησιμοποιείται το πρωτόκολλο SOAP (Simple Object Access Protocol).

- Το επίπεδο περιγραφής υπηρεσιών. Στο επίπεδο αυτό γίνεται η περιγραφή της υπηρεσίας Ιστού που περιλαμβάνει τον ορισμό της διεπαφής (δηλαδή ποιες μέθοδοι υποστηρίζει και πως καλούνται) και άλλες πληροφορίες όπως η διεύθυνση της υπηρεσίας (δηλαδή που βρίσκεται). Αυτή η περιγραφή μπορεί να χρησιμοποιηθεί σαν οδηγός για την σύνταξη μηνυμάτων κλήσης μιας υπηρεσίας (ή κλήση μεθόδων) και απάντησης που λαμβάνουν χώρα ανάμεσα στον αιτών και τον παροχέα υπηρεσιών. Στο επίπεδο περιγραφής υπηρεσιών περιλαμβάνεται η γλώσσα WSDL (Web Services Defintion Language).

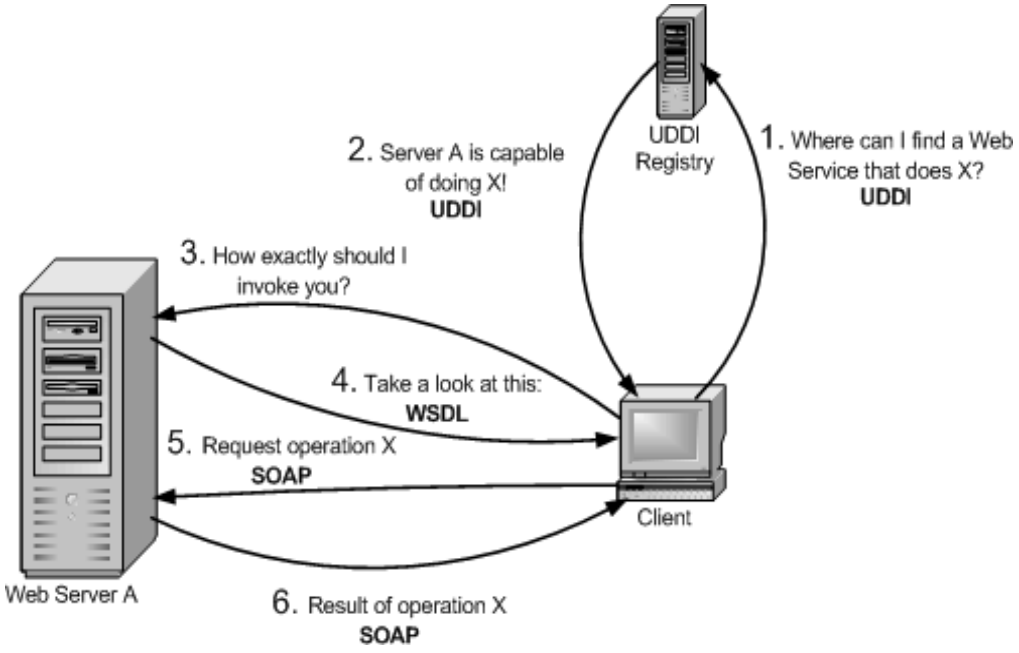
- Το επίπεδο αναζήτησης υπηρεσιών. Αυτό το επίπεδο δίνει την δυνατότητα στον αιτών ο οποίος για να καλέσει μια υπηρεσία Ιστού πρέπει να αναζητήσει τις υπηρεσίες ή τις περιγραφές των υπηρεσιών Ιστού με βάση κάποια κριστήρια από τους καταλόγους υπηρεσιών. Στο επίπεδο αναζήτησης χρησιμοποιείται ο μηχανισμός UDDI (Universal Discovery Description Integration).

Μετά την εξέταση των τεχνολογιών που χρησιμοποιούνται στις υπηρεσίες Ιστού, ας δούμε τώρα όλα τα βήματα που περιλαμβάνονται σε μια τυπική κλήση υπηρεσίας Ιστού, όπως φαίνεται στο Σχήμα 12.3.

1. Όπως έχουμε πει πριν, ο πελάτης ίσως να μην έχει γνώση τι είναι υπηρεσία Ιστού μέχρι να την καλέσει. Έτσι, το πρώτο βήμα θα είναι η αναζήτηση μιας υπηρεσίας Ιστού που να ικανοποιεί τα κριτήρια μας. Για παράδειγμα, θέλουμε να βρούμε που βρίσκεται η δημόσια υπηρεσία Ιστού που να μας δίνει τις θερμοκρασίες των πόλεων της Ελλάδας. Για να γίνει αυτό, ο πελάτης θα έλθει σε επαφή με το κατάλογο υπηρεσιών UDDI.

2. Ο κατάλογος UDDI θα απαντήσει και θα λέει στο πελάτη ποιοι είναι οι διακομιστές οι οποίοι παρέχουν την υπηρεσία που ζητήσαμε (π.χ. τις θερμοκρασίες των πόλεων της Ελλάδας).

3. Γνωρίζουμε την θέση της υπηρεσίας Ιστού αλλά δεν έχουμε ιδέα για το πως πραγματικά θα την καλέσουμε. Έτσι, η υπηρεσία είναι σίγουρο ότι θα μας δώσει την θερμοκρασία



Σχήμα 12.3: Διαδικασία κλήσης μιας υπηρεσίας Ιστού

μιας πόλης της Ελλάδος αλλά πως θα γίνει η κλήση της υπηρεσίας; Μπορεί η μέθοδος που πρέπει να καλέσουμε ίσως να ονομάζεται `Temperature getCityTemperature(int CityPostalCode)` αλλά θα μπορούσε να λέγεται και ως `int getUSCityTemp(string cityName, bool isFarenheit)`. Άρα, πρέπει να ζητήσουμε την περιγραφή της υπηρεσίας Ιστού (π.χ. θα μας λέει πως ακριβώς θα καλέσουμε την υπηρεσία).

4. Η υπηρεσία Ιστού απαντά σε μια γλώσσα που λέγεται WSDL.

5. Τώρα γνωρίζουμε που βρίσκεται η υπηρεσία Ιστού και πως θα την καλέσουμε. Η κλήση της υπηρεσίας γίνεται σε μια γλώσσα που λέγεται SOAP. Συνεπώς, θα στείλουμε πρώτα μια αίτηση SOAP σε μορφή XML που θα ζητάει την θερμοκρασία μιας πόλης.

6. Η υπηρεσία Ιστού θα απαντήσει με μια απάντηση SOAP σε μορφή XML που θα περιλαμβάνει την θερμοκρασία που ζητήσαμε ή ίσως να περιέχει ένα μήνυμα σφάλματος αν η αίτηση μας SOAP ήταν συντακτικά λάθος.

Πρέπει να σημειώσουμε ότι στο βήμα 2 ο κατάλογος UDDI δίνει στον πελάτη μια πληροφορία για το που βρίσκεται η υπηρεσία Ιστού που αναζητά. Έτσι, για τον εύκολο εντοπισμό των υπηρεσιών Ιστού εισάγεται ένα απλό σχήμα διευθυνσιοδότησης URI (Uniform Resource

Identifier) όπως ακριβώς οι ιστοσελίδες που έχουν μια διεύθυνση URL. Γενικά, κάθε υπηρεσία Ιστού έχει μια URI η οποία χρησιμοποιείται από τους πελάτες για την προσπέλαση της. Η μορφή της διεύθυνσης URI είναι της μορφής `http://webservices.mysite.com/WeatherService`.

12.4 Πλεονεκτήματα και Μειονεκτήματα Υπηρεσιών Ιστού

Πριν εξετάσουμε τις τεχνολογίες των υπηρεσιών Ιστού ξεχωριστά, να παρουσιάσουμε τα πλεονεκτήματα και τα μειονεκτήματα που παρουσιάζουν οι τεχνολογίες αυτές. Οι υπηρεσίες Ιστού έχουν πολλά πλεονεκτήματα όπως:

- Προωθούν τη **διαλειτουργικότητα** (interoperability): Η αλληλεπίδραση μεταξύ ενός παροχέα υπηρεσιών και ενός αιτών υπηρεσιών έχει ως σκοπό να είναι εντελώς ανεξάρτητη πλατφορμών και γλώσσας προγραμματισμού. Έτσι ο παροχέας υπηρεσιών και ο αιτών υπηρεσιών δεν έχουν καμία ιδέα ποιες πλατφόρμες ή γλώσσες προγραμματισμού χρησιμοποιεί ο καθένας τους, η διαλειτουργικότητα είναι δεδομένη. Με άλλα λόγια, το πρόγραμμα του αιτών μπορεί να έχει υλοποιηθεί σε γλώσσα C++ και να εκτελείται κάτω από Windows, ενώ η υπηρεσία Ιστού να έχει υλοποιηθεί σε Java και να εκτελείται κάτω από Linux.

- Συνδυάζουν διαφορετικές υπηρεσίες Ιστού: Έτσι, μια διεπαφή υπηρεσίας επιτρέπει τις μεθόδους της να συνδυάζονται με αντίστοιχες μεθόδους άλλων υπηρεσιών ώστε να παρέχουν μια νέα λειτουργικότητα στο αιτών υπηρεσιών. Για παράδειγμα, ας υποθέσουμε ότι ένας χρήστης - ταξιδιώτης για να πάει διακοπές σε ένα μεγάλο νησί της Ελλάδας πρέπει να εξασφαλίζει κράτηση για αεροπλάνο, κράτηση για ξενοδοχείο και ενοικίαση αυτοκινήτου. Οι λειτουργίες αυτές μπορεί να τις πραγματοποιήσει ο ταξιδιώτης μέσω του περιηγητή Ιστού επιλέγοντας για κάθε λειτουργία ένα διαφορετικό ιστοχώρο δηλαδή θα επιλέξει 3 ιστοχώρους. Όμως, αν κάθε από τους ιστοχώρους αυτούς διέθεταν μια υπηρεσία Ιστού, τότε μια νέα υπηρεσία Ιστού θα μπορούσε να συνδυάζει τις αντίστοιχες λειτουργίες των τριών υπηρεσιών προκειμένου να προσφέρει στον ταξιδιώτη ένα συνδυασμός των υπηρεσιών αυτών.

- Διασχίζουν μέσω **αντιπυρηνικών ζώνων** (firewalls): Οι περισσότερες υπηρεσίες Ιστού χρησιμοποιούν το πρωτόκολλο HTTP για ανταλλαγή μηνυμάτων (όπως αίτηση υπηρεσίας και απάντηση). Αυτό είναι το βασικό πλεονέκτημα στην περίπτωση που θέλουμε να αναπτύξουμε μια εφαρμογή για Διαδίκτυο αφού οι περισσότερες αντιπυρηνικές ζώνες του Διαδικτύου δεν εμποδίζουν την θύρα HTTP σε σχέση με τα πρωτόκολλα μεταφοράς που χρησιμοποιούν η Java RMI όπου δεν επιτρέπουν να περάσουν μέσα από την αντιπυρηνική ζώνη.

- Προσφέρουν διαφάνεια: Το ενδιάμεσο λογισμικό των υπηρεσιών Ιστού συνήθως προστατεύει τον προγραμματιστή από τις λεπτομέρειες αναπαράστασης και της πρόταξης καθώς επίσης και με την κλήση απομακρυσμένων υπηρεσιών Ιστού. Σε απλό επίπεδο, ο αιτών και ο παροχέας υπηρεσιών ανταλλάσουν μηνύματα SOAP με την χρήση της XML. Συνεπώς, οι λεπτομέρειες του SOAP, XML καθώς και άλλων τεχνολογιών όπως WSDL είναι γενικά κρυμμένες από ένα ενδιάμεσο λογισμικό που ενσωματώνεται σε μια γλώσσα προγραμματισμού όπως η Java.

Όμως, οι υπηρεσίες Ιστού παρουσιάζουν μερικά μειονεκτήματα:

- Επιβάρυνση: Γνωρίζουμε ότι τα μηνύματα και τα δεδομένα που ανταλλάσονται μεταξύ αιτών και παροχέα υπηρεσιών είναι σε μορφή XML. Η αναπαράσταση XML είναι ουσιαστικά αναπαράσταση κειμένου που απαιτεί περισσότερο χώρο από ότι το εκτελέσιμο κώδικα και επομένως απαιτεί περισσότερο χρόνο για επξεργασία. Συνεπώς, η μετάδοση όλων των δεδομένων σε XML δεν είναι προφανώς τόσο αποτελεσματική όσο εκείνου του εκτελέσιμου κώδικα. Όμως κερδίσουμε την μεταφερσιμότητα αλλά χάνουμε στην απόδοση. Ακόμα και έτσι, η επιβάρυνση αυτή είναι συνήθως αποδεκτή για τις περισσότερες εφαρμογές αλλά πιθανώς να μην βρούμε μια εφαρμογή πραγματικού χρόνου που να χρησιμοποιεί υπηρεσίες Ιστού.

- Έλλειψη ευστροφίας. Οι σημερινές υπηρεσίες Ιστού δεν είναι πολύ εύστοχες αφού επιτρέπουν μόνο για βασικές μορφές επίκλησης υπηρεσιών. Για παράδειγμα, το λογισμικό CORBA προσφέρει στους προγραμματιστές πολλές υπηρεσίες υποστήριξης (όπως επίμονα, διαχείριση κύκλου ζωής, συναλλαγές κλπ).

12.5 SOAP

Το SOAP είναι απλό πρωτόκολλο που ορίζει κανόνες για την οργάνωση και επεξεργασία των μηνυμάτων XML καθώς και την μεταφορά των μηνυμάτων αυτών πάνω στο πρωτόκολλο HTTP. Συνεπώς, το πρωτόκολλο SOAP ορίζει ένα σχήμα για χρήση της γλώσσας XML για την αναπαράσταση των μηνυμάτων αποστολής και λήψης καθώς επίσης ορίζει ένα σχήμα για επικοινωνία των μηνυμάτων μέσω του πρωτοκόλλου HTTP. Αρχικά το SOAP χρησιμοποιήθηκε μόνο για HTTP αλλά η τρέχουσα έκδοση έχει σχεδιαστεί με τέτοιο τρόπο ώστε να χρησιμοποιεί μια ποικιλία πρωτοκόλλων μεταφοράς όπως SMTP, TCP ή UDP. Σε αυτή την ενότητα η περιγραφή του SOAP βασίζεται στην έκδοση 1.2 η οποία είναι μια σύσταση του οργανισμού W3C.

Το SOAP παρέχει τα παρακάτω χαρακτηριστικά και μηχανισμούς:

- Μηχανισμός για τον ορισμό της πληροφορίας στην επικοινωνία. Στο SOAP, όλη η πληροφορία είναι καταχωρημένη σε ένα ξεκάθαρο και ταυτοποιήσιμο μήνυμα SOAP (SOAP message). Αυτό γίνεται μέσω ενός φακέλου SOAP (SOAP envelope), που περικλείει όλες τις απαραίτητες πληροφορίες. Ένα μήνυμα SOAP, μπορεί να έχει σώμα (body), το οποίο περιέχει πληροφορία σε δομή XML. Επίσης, μπορεί να διαθέτει έναν αριθμό από επικεφαλίδες (headers), οι οποίες ενσωματώνουν επιπλέον πληροφορίες έξω από το σώμα του μηνύματος.

- Διεργασιακό μοντέλο. Αυτό το μοντέλο ορίζει ένα σύνολο κανόνων βάσει των οποίων γίνονται οι διαπραγματεύσεις μεταξύ των μηνυμάτων SOAP και του λογισμικού. Διακρίνεται από την απλότητά του.

- Μηχανισμός για αντιμετώπιση σφαλμάτων. Το SOAP παρέχει το μηχανισμό αυτό με τη μορφή των SOAP faults, τα οποία όταν χρησιμοποιούνται μπορεί να προσδιοριστεί η πηγή που προκάλεσε το σφάλμα. Επιπλέον, παρέχουν τη δυνατότητα ανταλλαγής διαγνωστικών πληροφοριών μεταξύ των μελών της επικοινωνίας.

- Μοντέλο επεκτασιμότητας. Το μοντέλο αυτό χρησιμοποιεί επικεφαλίδες SOAP για την υλοποίηση προεκτάσεων πάνω στο SOAP. Οι επικεφαλίδες περιέχουν κομμάτια από δεδομένα με δυνατότητα επέκτασης, τα οποία ταξιδεύουν μαζί με το μήνυμα και μπορούν να γίνουν στόχος για επέκταση σε συγκεκριμένους κόμβους του δικτύου.

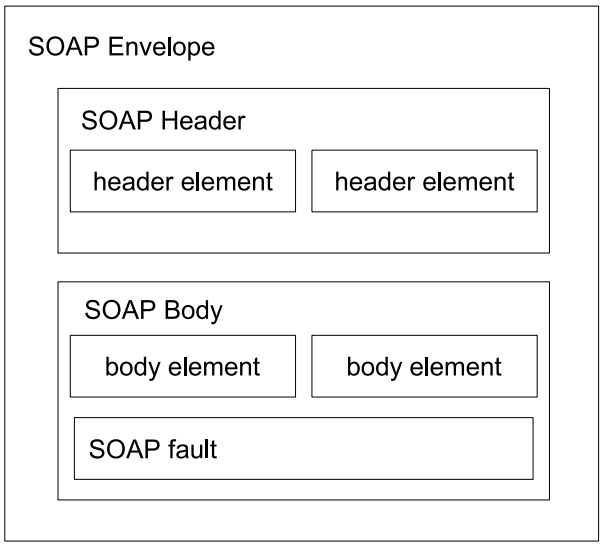
- Ευέλικτος μηχανισμός για αναπαράσταση δεδομένων. Ο μηχανισμός αυτός επιτρέπει στα δεδομένα σε οποιαδήποτε σειριακή μορφή κι αν βρίσκονται να αναπαρασταθούν σε μορφή XML.

- Σύμβαση για αναπαράσταση των RPCs και των απαντήσεων τους ως μηνύματα SOAP. Οι απομακρυσμένες κλήσεις διαδικασιών είναι αρκετά διαδεδομένες στον κατανεμημένο προγραμματισμό/ υπολογισμό και μπορούν να αναπαρασταθούν καλά μέσω του μηχανισμού αυτού ως μηνύματα SOAP.

- Πρωτόκολλο εγκατάστασης σύνδεσης. Αυτό το πρωτόκολλο ορίζει μια αρχιτεκτονική για την οικοδόμηση συνδέσεων επικοινωνίας ώστε να είναι εφικτή η ανταλλαγή μηνυμάτων SOAP πάνω σε μέσα μεταφοράς και επικοινωνιακά κανάλια. Χρησιμοποιείται το πρωτόκολλο HTTP, καθώς είναι το πιο διαδεδομένο και ευρέως χρησιμοποιούμενο στο Διαδίκτυο.

12.5.1 Δομή ενός Μηνύματος SOAP

Ένα μήνυμα SOAP είναι παρόμοιο με ένα φάκελο, όπου απέξω περιέχουν πληροφορίες δρομολόγησης (δηλαδή διευθύνσεις αποστολέα και παραλήπτη) και μέσα στο φάκελο είναι το περιεχόμενο του. Ένα μήνυμα SOAP είναι ένα απλό αρχείο XML και περιέχει τα ακόλουθα στοιχεία, όπως φαίνεται στο Σχήμα 12.4:



Σχήμα 12.4: Η δομή του μηνύματος SOAP

1. Τον **φάκελο** (envelope) το οποίο είναι υποχρεωτικό και αναγνωρίζει το αρχείο XML ως ένα μήνυμα SOAP, δηλαδή οριοθετεί την αρχή και το τέλος του μηνύματος. Τέλος, περιέχει πληροφορίες για το περιεχόμενο του μηνύματος και πως να γίνει η επεξεργασία του.

2. Την **επικεφαλίδα** (header) η οποία είναι προαιρετική και περιέχει πληροφορίες που σχετίζονται με τον περιεχόμενο, την ασφάλεια και την ποιότητα των υπηρεσιών.

3. Το **σώμα** (body) το οποίο είναι υποχρεωτικό και περιέχει τα δεδομένα του μηνύματος σε μορφή XML. Στο σώμα διακρίνονται δύο κατηγορίες μηνυμάτων: μηνύματα που περιέχουν μόνο κείμενο και μηνύματα που περιέχουν πληροφορίες κλήσης και απάντησης απομακρυσμένων διαδικασιών RPC.

4. Το **σφάλμα** (fault) το οποίο είναι προαιρετικό και περιέχει πληροφορίες σχετικές με λάθη που γίνονται κατά την επεξεργασία του μηνύματος.

Παρακάτω παρουσιάζεται η γενική δομή του μηνύματος SOAP σε μορφή XML:

```
<soap:envelope
      xmlns:soap = namespace URI for SOAP envelopes>
<soap:header>
     .....
     .....
</soap:header>
<soap:body>
     .....
    <soap:fault>
        .....
    </soap:fault>
</soap:body>
</soap:envelope>
```

Η δομή του σώματος καθορίζεται από δύο βασικά πρότυπα ανταλλαγής μηνυμάτων: τα μηνύματα SOAP που περιέχουν κείμενο και τα μηνύματα SOAP που συμπεριφέρονται ως κλήση απομακρυσμένης διαδικασίας. Στις επόμενες ενότητες παρουσιάζουμε για αυτά τα δύο πρότυπα ανταλλαγής μηνυμάτων.

Μηνύματα SOAP σε Κείμενο

Σε αυτή την κατηγορία μηνυμάτων στο σώμα του SOAP περιέχουν πληροφορίες περιγραφής της δομής του κειμένου. Επίσης, σε αυτή την κατηγορία, ο αποστολέας και ο παραλήπτης έχουν

συμφωνήσει την δομή του κειμένου τα οποία ανταλλάσονται και τα οποία περικλείονται στο σώμα των μηνυμάτων. Το κείμενο βρίσκεται μέσα στο φάκελο. Για καλύτερη κατανόηση θα παρουσιάζουμε ένα παράδειγμα που θα αναφερθεί στο υπόλοιπο του κεφαλαίου για την χρήση του SOAP για μια συγκεκριμένη μέθοδο. Για παράδειγμα, θέλουμε να πάρουμε την τιμή ενός βιβλίου από ένα συγκεκριμένο βιβλιοπωλείο. Στην Java αυτό θα μπορούσε να γραφεί σαν μέθοδο όπως φαίνεται παρακάτω:

```
public Double getPrice(String title);
```

Έτσι, η παραπάνω μέθοδος παίρνει μια παράμετρος το τίτλο του βιβλίου και επιστρέφει την τιμή του. Παρακάτω φαίνεται ένα παράδειγμα μηνύματος SOAP που περιγράφει την μέθοδο `getPrice`:

```
<soap:envelope>
  <soap:body>
    <GetPrice>
      <title>
         Distributed Systems
      </title>
    </GetPrice>
  </soap:body>
</soap:envelope>
```

Το παραπάνω παράδειγμα είναι σύμφωνα με την δομή ενός μηνύματος SOAP αλλά δεν περιέχει αρκετές πληροφορίες για μια κλήση απομακρυσμένης διαδικασίας όπως για παράδειγμα, δεν περιέχει το τύπο της παραμέτρου. Έτσι, στην επόμενη ενότητα εξετάζουμε τα μηνύματα SOAP ώστε να μπορούν να λειτουργήσουν ως κλήση RPC.

Μηνύματα SOAP σε Κλήση RPC

Σε αυτή την κατηγορία έχουμε μηνύματα που αποτελούν αιτήσεις για την εκτέλεση μιας λειτουργίας και μηνύματα απαντήσεις που μεταφέρουν τα αποτελέσματα της αίτησης. Η διαφορά στις δύο αυτές περιπτώσεις έγκειται στον τρόπο με τον οποίο τα μηνύματα αυτά κατασκευάζονται. Το κυρίως μέρος του μηνύματος αίτησης περιλαμβάνει την κλήση. Επίσης, περιλαμβάνει το όνομα της μεθόδου την οποία θα καλέσει και τις παραμέτρους εισόδου. Το κυρίως μέρος του μηνύματος απάντησης περιλαμβάνει το αποτέλεσμα και τις παραμέτρους εξόδου. Και οι δυο εφαρμογές που αλληλεπιδρούν θα πρέπει να συμφωνήσουν για την υπογραφή της μεθόδου RPC.

Η ενθυλάκωση των κλήσεων απομακρυσμένων διαδικασιών είναι ένα σημαντικό χαρακτηριστικό που χρησιμοποιείται στο SOAP. Για το σκοπό αυτό, ορίζουμε ένα καθολικό χαρακτηριστικό

που λέγεται ''`encodingStyle`'' το οποίο χρησιμοποιείται για την σειριοποίηση των δεδομένων και μπορεί να χρησιμοποιηθεί για την πρόταξη των παραμέτρων κατά τις κλήσεις απομακρυσμένων διαδικασιών. Το χαρακτηριστικό αυτό μπορεί να εμφανιστεί σε οποιοδήποτε στοιχείο του SOAP. Τα στοιχεία παιδιά του SOAP που δεν έχουν δικό του χαρακτηριστικό `encodingStyle` ανήκουν στην εμβέλεια του χαρακτηριστικού `encodingStyle` των πατρικών στοιχείων. Παρόλο αυτά οι κανόνες που ορίζονται από την XML Schema είναι επαρκείς και το SOAP χρησιμοποιεί ένα υποσύνολο αυτών των κανόνων.

Όπως οι γλώσσες προγραμματισμού, έτσι και το SOAP παρέχει βασικούς τύπους δεδομένων όπως `short, int, float, string, enumeration, array` και σύνθετους τύπους δεδομένων. Το καθένα από τα στοιχεία του σύνθετου τύπου δεδομένων περιέχει επίσης τύπος δεδομένων. Οι τύποι δεδομένων υιοθετήθηκαν από τον ορισμό της XML Schema. Οι κλήσεις RPC κωδικοποιούνται μέσα στο σώμα του μηνύματος SOAP. Τα παρακάτω στοιχεία είναι απαραίτητα για την κλήση μιας RPC:

- Ένα URI του αντικειμένου-στόχου
- Το όνομα της μεθόδου
- Μια προαιρετική υπογραφή της μεθόδου
- Οι παράμετροι της μεθόδου
- Προαιρετικές πληροφορίες επικεφαλίδας

Η URI του αντικειμένου-στόχου δεν προσδιορίζεται μέσα στο φάκελο του μηνύματος αλλά είναι υποχρέωση του πρωτοκόλλου. Στην περίπτωση που χρησιμοποιούμε το πρωτόκολλο HTTP, τότε η αίτηση HTTP προσδιορίζει την URI του αντικειμένου όπως θα δούμε παρακάτω.

Κάθε μέθοδο SOAP περιέχει μόνο μια κλήση μεθόδου. Η κλήση RPC περιγράφεται ως μια δομή. Το όνομα της δομής (δηλαδή το όνομα του πρώτου στοιχείου στο σώμα SOAP), είναι ίδιο με αυτό της μεθόδου. Επιπλέον, υπάρχει μια προσθήκη για κάθε παράμετρο με το ίδιο όνομα και τύπου όπως στην παράμετρο μιας μεθόδου. Τέλος, η σειρά των παραμέτρων πρέπει να είναι ίδια όπως στην μέθοδο. Παρακάτω, παρουσιάζεται ένα παράδειγμα μηνύματος SOAP αίτησης χρησιμοποιώντας namespaces.

```
<soap:envelope xmlns:soap = "namespace URI for SOAP envelope"
 soap:encodingStyle = "namespace URI for encoding">
<soap:body>
  <m:GetPrice xmlns:m = "namespace URI of the service description">
    <title xsi:type = "xsd:string">
      Distributed Systems
    </title>
  </m:GetPrice>
</soap:body>
</soap:envelope>
```

Στο μοντέλο RPC υπάρχει και ένα μήνυμα SOAP απάντησης που στέλνεται από τον παροχέα υπηρεσιών στο αιτούντα. Έτσι και στο μήνυμα απάντησης RPC, περιγράφεται επίσης ως μια δομή. Το όνομα της είναι ίδιο με αυτό της μεθόδου, με μοναδική διαφορά την προσθήκη της λέξης ''`Response`''. Επίσης, υπάρχει μια προσθήκη για μια επιστρεφόμενη τιμή για κάθε παράμετρο επιστροφής. Η σειρά των τιμών είναι ίδια όπως στο μήνυμα αίτησης. Παρακάτω, φαίνεται ένα παράδειγμα μηνύματος απάντησης σε σχέση με το μήνυμα αίτησης:

```
<soap:envelope xmlns:soap = "namespace URI for SOAP envelope"
 soap:encodingStyle = "namespace URI for encoding">
<soap:body>
  <m:GetPriceResponse xmlns:m = "namespace URI of the service description">
    <return xsi:type = "xsd:float">
      55
    </return>
  </m:GetPriceResponse>
</soap:body>
</soap:envelope>
```

Αν για κάποιο λόγο αποτύχει η αίτηση, τότε στο σώμα του μηνύματος πρέπει να περιέχει ένα στοιχείο σφάλματος που θα εξηγεί την αποτυχία. Έτσι, το στοιχείο `fault` του SOAP έχει κάποια συγκεκριμένη και ορισμένη δομή ώστε να παρέχει πληροφορία σχετική με το λάθος που συνέβη. Επομένως, το στοιχείο αυτό περιέχει 4 υποστοιχεία:

- `faultcode`. Το στοιχείο αυτό προσδιορίζει τους παρακάτω τέσσερις κωδικούς για την περιγραφή του τύπου αποτυχίας:

  1. VersionMismatch. Το μήνυμα δεν περιέχει το namespace `http://schemas.xmlsoap.org/soap/envelope/`

2. MustUnderstand. Ένα στοιχείο με το χαρακτηριστικό mustUnderstand να έχει τιμή 1 σημαίνει ότι δεν έγινε σωστή επεξεργασία.
3. Client. Το μήνυμα που παράγεται δεν είναι σε σωστή μορφή.
4. Server. Το μήνυμα δεν μπορούσε να επεξεργαστεί σωστά αλλά το σφάλμα δεν ανήκει στα περιεχόμενα του μηνύματος.

- `faultstring`. Παρέχει μια ευανάγνωστη εξήγηση.
- `faultactor`. Προσδιορίζει την οντότητα που προκάλεσε την αποτυχία στο τρόπο του μηνύματος προς το τελικό προορισμό.
- `detail`. Μεταφέρει συγκεκριμένες πληροφορίες σφάλματος της εφαρμογής.

Παρακάτω φαίνεται ένα πλήρες μήνυμα SOAP με μια πιθανή εγγραφή σφάλματος στο παράδειγμα μας για την περίπτωση που δεν βρεθεί η τιμή του βιβλίου:

```
<soap:envelope xmlns:soap = "namespace URI for SOAP envelope">
  <soap:body>
    <soap:fault>
      <faultcode>
        soap:Server
      </faultcode>
      <faultstring>
        Server Error
      </faultstring>
      <detail>
        <d:message xmlns:d="some URI">
          No Symbol found
        </d:message>
      </detail>
    </soap:fault>
  </soap:body>
</soap:envelope>
```

12.5.2 Μεταφορά Μηνυμάτων SOAP

Απαιτείται ένα πρωτόκολλο μεταφοράς για την μεταφορά ενός μηνύματος SOAP προς τον προορισμό. Τα μηνύματα SOAP είναι ανεξάρτητα από τον τύπο μεταφοράς που θα χρησιμοποιηθεί και οι φάκελοι δεν περιέχουν αναφορά για την διεύθυνση προορισμού. Αλλά είναι υποχρέωση

του πρωτοκόλλου HTTP για τον προσδιορισμό της διεύθυνσης προορισμού. Παρακάτω φαίνεται ένα παράδειγμα για την μεταφορά ενός μηνύματος αίτησης SOAP χρησιμοποιώντας την μέθοδο POST του HTTP.

```
POST /GetPrice
Host: www.amazon.com
Content-Type: application/soap+xml
Action: http://www.amazon.com/GetPrice

<soap:envelope xmlns:env = "namespace URI for SOAP envelope">
<soap:body>
  <m:GetPrice xmlns:m = "namespace URI of the service description">
    <return xsi:type = "xsd:float">
      55
    </return>
  </m:GetPriceResponse>
</soap:body>
</soap:envelope>
```

Η επικεφαλίδα HTTP προσδιορίζει την διεύθυνση ή URI του τελικού παραλήπτη και την λειτουργία που θα εκτελέσει. Η παράμετρος `Action` στην επικεφαλίδα HTTP χρησιμοποιείται για την σωστή επιλογή της μεθόδου που θα κληθεί χωρίς να χρειαστεί ανάλυση του μηνύματος SOAP στο σώμα HTTP. Γι αυτό το λόγο στην παράμετρο `Action` δίνεται το όνομα της μεθόδου που πρόκειται να κληθεί. Αντίθετα, το σώμα HTTP μεταφέρει το μήνυμα SOAP.

Μετά την αποστολή του μηνύματος αίτησης SOAP μέσω του πρωτόκολλου HTTP και λόγω ότι το πρωτόκολλο αυτό είναι σύγχρονο τότε θα επιστραφεί μια απάντηση που θα περιέχει το μήνυμα απάντησης SOAP όπως για παράδειγμα είδαμε προηγουμένως. Στην περίπτωση που θέλουμε να μεταφέρουμε ένα μήνυμα αίτησης SOAP που περιέχει αίτηση για επιστροφή πληροφοριών χωρίς παραμέτρους τότε χρησιμοποιούμε την μέθοδο GET του HTTP.

Τέλος, ο προγραμματιστής δεν χρειάζεται να γνωρίζει όλες τις παραπάνω λεπτομέρειες που αφορά για το SOAP αφού η διαδικασία μετάφρασης της κλήσης των απομακρυσμένων μεθόδων σε μηνύματα SOAP έχει υλοποιηθεί από ένα ενδιάμεσο λογισμικό όπως ο Apache Axis. Το λογισμικό αυτό είναι μια μηχανή SOAP ανοικτού λογισμικού και θα το δούμε παρακάτω στην διαδικασία ανάπτυξης εφαρμογών υπηρεσιών Ιστού.

12.6 WSDL

Είδαμε προηγουμένως ότι το SOAP προσδιορίζει την επικοινωνία ανάμεσα στο αιτών και τον παροχέα. Συγκεκριμένα, στο πρωτόκολλο SOAP μπορούν να γίνουν αιτήσεις για απομακρυσμένες κλήσεις διαδικασιών από το αιτών σε κάποιο παροχέα. Από τη μεριά του αιτών όμως υπάρχουν δυσκολίες στη σύνταξη του μηνύματος SOAP, καθώς ο πελάτης δεν μπορεί να ξέρει τι ακριβώς μήνυμα να στείλει. Το SOAP καθορίζει κάποιους κανόνες και προσφέρει μια συγκεκριμένη δομή για τα μηνύματα, αλλά πρέπει να βρεθεί κάποιος άλλος τρόπος ώστε ο αιτών να είναι σε θέση να γνωρίζει τι μήνυμα να στείλει στον παροχέα της υπηρεσίας. Επίσης, ο αιτών πρέπει να γνωρίζει αρκετές λεπτομέρειες προτού στείλει το μήνυμα. Πρέπει να γνωρίζει καταρχάς που ακριβώς να το στείλει, ή ποιες μεθόδους θέλει να καλέσει από την υπηρεσία, ή ποια πρωτόκολλα επικοινωνίας υποστηρίζονται από τον παροχέα της υπηρεσίας. Προκειμένου ο αιτών να γνωρίζει όλες τις απαραίτητες πληροφορίες για την κλήση μιας υπηρεσίας Ιστού, πρέπει να έχει στην κατοχή του μια περιγραφή αυτής. Έτσι απαιτείται ένας ορισμός της διεπαφής υπηρεσίας ή περιγραφή της υπηρεσίας Ιστού ώστε να επιτρέψει στο αιτών να επικοινωνήσει με την αντίστοιχη υπηρεσία. Για το σκοπό αυτό έχει προταθεί η καθεριωμένη γλώσσα περιγραφής υπηρεσιών Ιστού που είναι η WSDL (Web Services Description Language) από τον οργανισμό W3C.

Η γλώσσα WSDL χρησιμοποιείται για την περιγραφή και την δήλωση της τοποθεσίας των υπηρεσιών Ιστού χρησιμοποιώντας την γλώσσα XML. Επίσης, η WSDL περιγράφει τρεις σημαντικές ιδιότητες της υπηρεσίας:

- Τι κάνει η υπηρεσία; Εδώ περιγράφονται οι μέθοδοι της υπηρεσίας, οι παράμετροι της και τα αποτελέσματα που επιστρέφει.
- Πώς γίνεται η πρόσβαση στην υπηρεσία; Εδώ παρέχονται όλες οι απαραίτητες πληροφορίες για τη σύνταξη των μηνυμάτων SOAP που θα ανταλλαγούν, καθώς και τα πρωτόκολλα που υποστηρίζονται από τον παροχέα.
- Που βρίσκεται η υπηρεσία; Εδώ παρέχονται πληροφορίες για τη δικτυακή διεύθυνση της υπηρεσίας. Συνήθως είναι ένα URI.

Γενικά, η γλώσσα WSDL περιγράφει τις υπηρεσίες που προσφέρει ο παροχέας και μπορεί να χρησιμοποιηθεί ως συνταγή για την παραγωγή κατάλληλων μηνυμάτων SOAP για την πρόσβαση

των υπηρεσιών Ιστού. Ένα έγγραφο σε WSDL έχει παρόμοιο ρόλο με εκείνο τον ορισμό της απομακρυσμένης διεπαφής σε υλοποίηση Java RMI ή το αρχείο σε IDL της CORBA.

12.6.1 Δομή ενός Εγγράφου WSDL

Το πρώτο στοιχείο ενός οποιοδήποτε εγγράφου WSDL είναι το στοιχείο `<definitions>`. Το στοιχείο αυτό περιλαμβάνει έξι στοιχεία τα οποία διακρίνονται σε δύο κατηγορίες: **αφαιρετική περιγραφή** (abstract definition) και **αυστηρή περιγραφή** (concrete defintion).

Η αφαιρετική περιγραφή περιλαμβάνει τα στοιχεία:

- `types` είναι το σύστημα τύπων που χρησιμοποιείται για την περιγραφή των μηνυμάτων.
- `messages` είναι τα μηνύματα που σχετίζονται με την κλήση της υπηρεσίας.
- `portType` είναι ξεχωριστές λειτουργίες που αποτελούνται από διαφορετικά πρότυπα ανταλλαγής μηνυμάτων.

Η αυστηρή περιγραφή περιλαμβάνει τα παρακάτω στοιχεία:

- `bindings` είναι οι συνδέσεις των λειτουργιών με ένα πρωτόκολλο μεταφοράς (ή επικοινωνίας).
- `services` είναι ως μια συλλογή των σχετικών συνδέσεων.

Παρακάτω φαίνεται η δομή του εγγράφου WSDL σε μορφή XML.

```
<definitions>
   <import/>
   <types>
   </types>
   <messages>
   </messages>
   <portType>
   </portType>
   <binding>
   </binding>
   <service>
   </service>
</definitions>
```

Κάθε στοιχείο του εγγράφου μπορεί να περιέχει ένα στοιχείο `<documentation>` που παρέχει μια ευανάγνωστη πληροφορία για το στοιχείο. Επίσης, όλα τα στοιχεία περιέχουν ένα χαρακτηριστικό που λέγεται ''`name`'' και λειτουργεί ως αναγνωριστής (identifier).

Στοιχείο import

Η δήλωση του στοιχείου `import` διαχωρίζει ένα έγγραφο WSDL σε πολλαπλά ανεξάρτητα έγγραφα. Έτσι, το στοιχείο αυτό δίνει τη δυνατότητα επαναχρησιμοποίησης ήδη υπαρχόντων εγγράφων WSDL επιτρέποντας σε διάφορα στοιχεία να έχουν αναφορές σε άλλα ήδη υπάρχοντα που βρίσκονται σε διαφορετικό αρχείο. Με αυτό τον τρόπο η δομή του εγγράφου WSDL γίνεται πιο κατανοητή και πιο εύκολη στη διαχείριση του. Η σύνταξη για το στοιχείο `import` είναι ως εξής:

```
<import namespace = "an uri" location = "an uri"/>
```

Σε κάθε έγγραφο μπορεί να έχει πολλές δηλώσεις `import`.

Στοιχείο types

Το στοιχείο αυτό μπορεί να χρησιμοποιηθεί για να περιγράψει όλους τους τύπους δεδομένων που μπορούν να χρησιμοποιηθούν στα διάφορα μηνύματα που ανταλλάσσονται για την κλήση της υπηρεσίας. Το σύστημα των τύπων δεδομένων στην WSDL βασίζεται στους ορισμούς της XML schema definitions - XSD.

Παρακάτω παρουσιάζεται ένα έγγραφο WSDL με την δήλωση τύπων για το παράδειγμα που χρησιμοποιούμε σε αυτό το κεφάλαιο.

```
<types>
  <shema>
    targetNamespace = "uri"
    xmlns = "uri for XML schema">
    <element name = "PriceResponseType">
      <complexType>
      <all>
        <element name = "price" type = "float"/>
      </all>
      </complexType>
    </element>
    <element name = "TitleRequestType">
      <complexType>
      <all>
        <element name = "title" type = "string"/>
      </all>
      </complexType>
    </element>
  </schema>
</types>
```

Στοιχείο message

Το στοιχείο `message` προσδιορίζουμε το τύπο δεδομένων (ή των παραμέτρων) που θα ανταλλάξουν ανάμεσα στον αιτών υπηρεσιών και τον παροχέα υπηρεσιών. Με άλλα λόγια, το στοιχείο `message` είναι ανάλογη με την παράμετρο σε μια μέθοδο της Java. Ένα στοιχείο `message` μπορεί να εμφανιστεί πολλές φορές στο έγγραφο και αποτελείται από ένα χαρακτηριστικό `name` το οποίο πρέπει να είναι μοναδικό ώστε να ξεχωρίζει από τα υπόλοιπα. Επίσης, το στοιχείο `message` χωρίζεται σε ένα ή περισσότερα στοιχεία `<part>`, που το καθένα από τα οποία είναι μια δομή δεδομένων σε μορφή XML. Καθένα από τα στοιχεία part πρέπει να έχει κάποιο τύπο (είτε βασικό ή σύνθετο) που έχει οριστεί προηγουμένως στο έγγραφο WSDL. Ουσιαστικά, τα στοιχεία αυτά δεν παρουσιάζουν ιδαίτερο ενδιαφέρον αφού δεν είναι τίποτε άλλο από μια συλλογή από parts και κάθε `part` έχει ένα όνομα που συνήθως αντικατοπτρίζει την πληροφορία που περιέχεται στο `part`. Έτσι παρακάτω παρουσιάζουμε δύο μηνύματα για την λειτουργία ή μέθοδο `GetPrice` - ένα για την αίτηση και ένα για την απάντηση. Το μήνυμα αίτησης περιέχει μια παράμετρο εισόδου και το μήνυμα απάντησης περιέχει μια παράμετρο εξόδου.

```
<message name="GetPriceInput">
  <part name="inputparam" element="xsd1:PriceRequestType"/>
</message>
<message name="GetPriceOutput">
  <part name="returnvalue" element="xsd1:PriceResponseType"/>
</message>
```

Στοιχείο portType

Το στοιχείο `portType` (μερικές φορές λέγεται interface) ορίζει την διεπαφή της υπηρεσίας Ιστού. Συνεπώς, ο σκοπός του είναι η περιγραφή αυτής της διεπαφής. Το στοιχείο αυτό σε όρους προγραμματισμού είναι ανάλογο με τον ορισμό της κλάσης Java. Έτσι, στον ορισμό του στοιχείου `portType` αποτελείται από ένα σύνολο συσχετιζόμενων λειτουργιών που υποστηρίζει η υπηρεσία Ιστού. Η λειτουργία σε προγραμματιστικούς όρους είναι ανάλογη με την μέθοδο σε μια κλάση. Ένα στοιχείο `<operation>` έχει ένα χαρακτηριστικό όνομα (`name`) το οποίο πιθανόν να σχετίζεται με τη λειτουργία που παρέχει η υπηρεσία και έχει ένα άλλο χαρακτηριστικό που προσδιορίζει την σειρά των παραμέτρων (ή των μηνυμάτων) που χρησιμοποιούνται στην λειτουργία. Επίσης, για κάθε λειτουργία πρέπει να προσδιοριστεί το πρότυπο ανταλλαγής μηνυμάτων ανάμεσα στο

αιτών και τον παροχέα υπηρεσιών. Έτσι, υπάρχουν τέσσερα πρότυπα ανταλλαγής παραμέτρων για το προσδιορισμό του τύπου μιας λειτουργίας:

- Μονόδρομη (one-way): Η υπηρεσία λαμβάνει μια αίτηση και δεν στέλνει απάντηση.
- Αίτηση/απάντηση (request/response): Η υπηρεσία λαμβάνει μια αίτηση και στέλνει μια απάντηση.
- Αίτηση για απάντηση (solicit response): Η υπηρεσία στέλνει ένα μήνυμα και λαμβάνει μια απάντηση.
- Γνωστοποίηση (notification): Η υπηρεσία στέλνει μια απάντηση και δεν λαμβάνει αίτηση.

Τα παραπάνω πρότυπα ανταλλαγής περιγράφονται από τα τρία παρακάτω υποστοιχεία μέσα στο στοιχείο `<operation>`:

- `<input ... />`
- `<output ... />`
- `<fault ... />`

Καθένα από τα τέσσερα υποστοιχεία περιέχει ένα χαρακτηριστικό `name` και ένα χαρακτηριστικό `message` το οποίο έχει οριστεί προηγουμένα στο στοιχείο `<message>`. Ακολουθούμε τους παρακάτω κανόνες για την χρήση των παραπάνω υποστοιχείων:

- Μια λειτουργία μονόδρομη προσδιορίζεται μόνο από το στοιχείο `input`.
- Μια λειτουργία αίτησης και απάντησης προσδιορίζεται πρώτα από το στοιχείο `input`, ακολουθούμενο από το στοιχείο `output` και έπειτα ακολουθούν μερικά προαιρετικά στοιχεία `fault`.
- Μια λειτουργία αίτησης για απάντηση προσδιορίζεται πρώτα από το στοιχείο `output`, ακολουθούμενο από το στοιχείο `input` και έπειτα ακολουθούν μερικά προαιρετικά στοιχεία `fault`.
- Μια λειτουργία γνωστοποίησης προσδιορίζεται μόνο από το στοιχείο `output`.

Παρακάτω παρουσιάζεται ένα παράδειγμα για την λειτουργία `GetPrice` χρησιμοποιώντας τους ορισμούς `message` που είδαμε στο προηγούμενο παράδειγμα.

```
<portType name="PricePortType">
   <operation name="GetPrice">
     <input message="tns:GetPriceInput" name="GetPriceInput"/>
     <output message="tns:GetPriceOutput" name="GetPriceOutput"/>
   </operation>
</portType>
```

Στοιχείο binding

Τα στοιχεία που παρουσιάζαμε μέχρι τώρα περιγράφουν μια λειτουργία σε ένα γενικό και αφηρημένο τρόπο. Κανένα από τα προηγούμενα στοιχεία δεν ασχολείται με την υλοποίηση της λειτουργίας ή την σύνδεση με κάποιο πρωτόκολλο. Έτσι, ο στόχος μας είναι να συνδέσουμε το στοιχείο `portType` με κάποια πρωτόκολλα που θα χρησιμοποιηθούν για την επικοινωνία μεταξύ του αιτών (ή πελάτη) και της υπηρεσίας. Για το σκοπό αυτό, το πρότυπο WSDL εισάγει το στοιχείο `<binding>`. Ουσιαστικά είναι το στοιχείο που λέει στον πελάτη πως ακριβώς να μορφοποιήσει το μήνυμα που θα στείλει για την κλήση της υπηρεσίας. Με άλλα λόγια, το στοιχείο αυτό ορίζει μορφοποιήσεις μηνυμάτων ενός στοιχείου `portType`. Κάθε στοιχείο `portType` μπορεί να έχει ένα ή περισσότερα στοιχεία `binding` συσχετισμένα με αυτό. Για ένα συγκεκριμένο στοιχείο `portType`, ένα στοιχείο `binding` περιγράφει τον τρόπο που πρέπει να γίνει η κλήση των μεθόδων της υπηρεσίας χρησιμοποιώντας κάποιο καθορισμένο πρωτόκολλο, όπως το SOAP, πάνω σε κάποιο επίσης καθορισμένο πρωτόκολλο μεταφοράς, όπως το HTTP. Μέσα στο στοιχείο `binding` υπάρχει το χαρακτηριστικό `name` που πρέπει να είναι μοναδικό και το χαρακτηριστικό `type` που αναφέρεται στο όνομα ενός `portType`. Η σύνταξη του είναι παρόμοια με την σύνταξη του `portType` και περιγράφεται αναλυτικά στην επόμενη υποενότητα.

Σύνδεση με το SOAP

Για να συνδεθεί η διεπαφή `portType` με το πρωτόκολλο SOAP, υπάρχει ένα στοιχείο επέκταση SOAP που λέγεται `<soap:binding>`. Το στοιχείο αυτό συνοδεύεται με δύο παραμέτρους, το πρότυπο αίτησης (request style) και το πρωτόκολλο μεταφοράς. Για το πρότυπο αίτησης μπορεί να πάρει δύο τιμές την τιμή `RPC` ή την τιμή `document`. Στην περίπτωση `RPC`, τα μηνύματα περιέχουν παραμέτρους και τιμές επιστροφής ενώ στην περίπτωση του `document`, τα μηνύματα περιέχουν κείμενο. Η παράμετρος `transport` προσδιορίζει την URL ενός πρωτόκολλου για την μεταφορά των μηνυμάτων SOAP. Παράδειγμα, θέλουμε να μεταφέρουμε ένα μήνυμα SOAP για την κλήση μιας

αίτησης RPC πάνω στο πρωτόκολλο HTTP, τότε το στοιχείο <soap:binding> μπορεί να οριστεί ως εξής:

```
<soap:binding style = "rpc"
transport = "an URI for http" />
```

Στην συνέχεια πρέπει να δώσουμε πληροφορίες για την λειτουργία. Για το σκοπό αυτό πρέπει να χρησιμοποιήσουμε το στοιχείο <soap:operation>. Το στοιχείο αυτό περιέχει το χαρακτηριστικό ''style'' για να παρακάμψει το προκαθορισμένο πρότυπο για την συγκεκριμένη λειτουργία και το χαρακτηριστικό ''soapAction'' που χρησιμοποιείται για την επικεφαλίδα HTTP ενός μηνύματος SOAP. Για το προηγούμενο παράδειγμα, το στοιχείο <soap:operation> μπορεί να οριστεί ως εξής:

```
<soap:operation style = "rpc"
soapAction = " " />
```

Για κάθε λειτουργία εισάγουμε το στοιχείο <soap:body> μέσα στο έγγραφο WSDL που ορίζει πως θα εμφανίζεται ένα μήνυμα μέσα στο σώμα ενός μηνύματος SOAP. Το στοιχείο <soap:body> εμφανίζεται είτε μέσα στο στοιχείο <input> ή στο <output> του στοιχείου binding. Το σώμα SOAP περιέχει τέσσερα στοιχεία:

- parts. Το στοιχείο αυτό είναι προαιρετικό και δηλώνει ποια μέρη πρέπει να εμφανίζονται μέσα στο μήνυμα.

- use. Το στοιχείο αυτό είναι υποχρεωτικό και δηλώνει αν τα μέρη του μηνύματος είναι κωδικοποιημένοι χρησιμοποιώντας ένα πρότυπο κωδικοποίησης (encoding style) (παίρνει την τιμή ''encoded'') ή αν ένα στοιχείο αναφέρει ένα ορισμό σχήματος (παίρνει την τιμή ''literal'').

- encodingStyle. Το στοιχείο έχει ακριβώς την ίδια σημασιολογία όπως στο πρότυπο SOAP.

- namespace. Το στοιχείο αυτό μας δίνει την δυνατότητα να ορίζουμε άλλο namespace για την αποφυγή συγκρούσεων ονομάτων.

Παρακάτω ένα παράδειγμα που φαίνονται όλες οι λεπτομέρειες του στοιχείου binding για την λειτουργία GetPrice.

```
<binding name="PriceSoapBinding" type="tns:PricePortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetPrice">
     <soap:operation style = ‘‘rpc’’ soapAction=" "/>
     <input name="GetPriceInput">
        <soap:body use="encoded"
           encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
     </input>
     <output name="GetPriceOutput">
        <soap:body use="encoded"
           encodingStyle= "http://schemas.xmlsoap.org/soap/encoding/" />
     </output>
  </operation>
</binding>
```

Στοιχείο port

Περιγράφει πως ακριβώς εκφράζεται ένα στοιχείο `binding` σε μια φυσική θύρα του δικτύου. Ο ρόλος της είναι πολύ απλός αφού δεν κάνει τίποτε άλλο από μια απλή αντιστοιχία. Το στοιχείο αυτό, όπως και τα προηγούμενα, πρέπει να έχει ένα μοναδικό όνομα μέσα στο WSDL έγγραφο. Συνήθως, το στοιχείο αυτό περιέχει το URI στο οποίο πρέπει να σταλούν τα μηνύματα SOAP ώστε να γίνει η κλήση της υπηρεσίας. Το στοιχείο αυτό δεν εμφανίζει μόνο του μέσα στο έγγραφο της περιγραφής της υπηρεσίας. Είναι στοιχείο - παιδί του συστατικού `service` που θα δούμε παρακάτω.

Στοιχείο service

Το στοιχείο `service` είναι ένα σύνολο από θύρες. Το στοιχείο αυτό περιέχει το χαρακτηριστικό `name` το οποίο πρέπει να είναι μοναδικό και ένα υποστοιχείο `<port>` που μπορεί να εμφανιστεί πολλές φορές. Μπορούν να οριστούν διαφορετικές θύρες αν υπάρχουν πολλαπλές υλοποιήσεις του ίδιου `binding`. Για κάθε στοιχείο `<port>` αναφέρεται στο όνομα του `binding` που χρησιμοποιεί και επίσης χρησιμοποιεί το στοιχείο `soap:address` για να προσδιορίζει την URI της υπηρεσίας Ιστού. Παρακάτω παρουσιάζεται το στοιχείο `<service>` για το παράδειγμα που χρησιμοποιήσαμε στα προηγούμενα στοιχεία της WSDL.

```
<service name="PriceService">
   <port name="PricePort" binding="tns:PriceSoapBinding">
      <soap:address location="http://amazon.com/price"/>
```

```
  </port>
</service>
```

Πρέπει να σημειώσουμε ότι υπάρχουν διάφορα εργαλεία για την αυτόματη παραγωγή εγγράφων WSDL χωρίς να απαιτείται από τους προγραμματιστές να ασχολούνται με τις πολύπλοκες λεπτομέρειες και την δομή του WSDL. Σε αυτό το βιβλίο θα δούμε την παραγωγή εγγράφων WSDL από τον ορισμό διεπαφής σε γλώσσα Java χρησιμοποιώντας το λογισμικό Apache-Axis. Τέλος, το έγγραφο WSDL μπορεί να χρησιμοποιηθεί ως είσοδο σε μεταγλωττιστές κορμούς stubs και άλλα εργαλεία τα οποία παράγουν κορμούς.

12.7 UDDI

Το πρότυπο του UDDI προήλθε από την συνεργασία των IBM, Microsoft και Ariba στο Σεπτέμβριο του 2000 και η υποστήριξή του επεκτάθηκε πέρα από αυτές τις τρεις επιχειρήσεις (σήμερα περιλαμβάνει μια κοινότητα περισσότερων από 310 επιχειρήσεις). Στόχος του προτύπου UDDI είναι να παρέχει την αναγκαία υποδομή για την περιγραφή και αναζήτηση υπηρεσιών Ιστού. Έτσι, παριστάνει το κατάλογο υπηρεσιών που δίνει την δυνατότητα στους αιτούντες υπηρεσιών να αναζητήσουν το κατάλληλο παροχέα υπηρεσιών ή την περιγραφή υπηρεσίας Ιστού. Με πολλούς τρόπους, το UDDI λειτουργεί ως ένας ηλεκτρονικός κατάλογος όπως οι σελίδες του χρυσού οδηγού που προσφέρει πληροφορίες δυναμικά σχετικά με τις επιχειρήσεις και τις υπηρεσίες που παρέχουν.

Στα πλαίσια του μοντέλου υπηρεσιοστρεφή αρχιτεκτονική SOA, ο παροχέας υπηρεσιών δημοσιεύει μια υπηρεσία Ιστού στον κατάλογο UDDI. Για την δημοσίευση της υπηρεσίας απαιτεί μια εγγραφή στο κατάλογο UDDI, που δεν είναι τίποτα άλλο από ένα έγγραφο XML για την περιγραφή της υπηρεσίας Ιστού. Επίσης, ένα έγγραφο XML αποτελείται από τέσσερις βασικές δομές δεδομένων:

- businessEntity είναι μια δομή δεδομένων που περιέχει πληροφορίες σχετικά με το οργανισμό που παρέχει υπηρεσίες Ιστού. Οι πληροφορίες μπορεί να είναι το όνομα, η διεύθυνση και δραστηριότητες, κλπ

- businessService είναι μια δομή δεδομένων για να περιγράψει μια λίστα όλων των υπηρεσιών Ιστού που προσφέρονται από την businessEntity. Συνήθως περιλαμβάνει το όνομα και

μια περιγραφή του σκοπού της κάθε υπηρεσίας Ιστού.

- bindingTemplate είναι μια δομή που αποθηκεύει την διεύθυνση της κάθε υπηρεσίας Ιστού που αναφέρεται από την bussinessService και τις αναφορές σε περιγραφές υπηρεσίας.

- tModel αποθηκεύει τις περιγραφές υπηρεσιών, συνήθως έγγραφα WSDL τα αποία αποθηκεύονται εξωτερικά σε κάποια βάση δεδομένων και προσπελάζονται με βάση την διεύθυνση URLs.

Οι τρεις πρώτες δομές δεδομένων προσπελάζονται ξεχωριστά με βάση ένα αναγνωριστή που λέγεται κλειδί ενώ η τελευταία δομή προσπελάζεται με βάση το URL.

Οι παραπάνω δομές δεδομένων χρησιμοποιούνται ώστε το UDDI να προσφέρει τις παρακάτω υπηρεσίες πληροφοριών:

- **Λευκές σελίδες** (white pages): περιέχει γενικές πληροφορίες σχετικά με τον παροχέα της υπηρεσίας περιλαμβάνοντας το όνομα, την διεύθυνση, το πρόσωπο επικοινωνίας κλπ

- **Κίτρινες σελίδες** (yellow pages): περιέχει τι είδους υπηρεσίες προσφέρονται, καθώς και μια λίστα όλων των διαφορετικών κατηγοριοποιημένων υπηρεσιών.

- **Πράσινες σελίδες** (green pages): περιέχει τεχνικές λεπτομέρειες σχετικά με το πώς χρησιμοποιείται καθεμιά από τις προσφερόμενες υπηρεσίες ιστού, συμπεριλαμβάνοντας αναφορές προς τις διεπαφές όπως η περιγραφή τους WSDL (η οποία δεν βρίσκεται στο κατάλογο UDDI).

Με αυτό τον τρόπο οι αιτούντες (ή πελάτες) μπορούν να αποκτήσουν περιγραφές υπηρεσίας WSDL χρησιμοποιώντας τις λευκές σελίδες αναζητώντας με βάση το όνομα του οργανισμού ή τις κίτρινες σελίδες αναζητώντας μια συγκεκριμένη κατηγορία υπηρεσιών.

Εκτός από τον ορισμό των δομών δεδομένων που θα αποθηκεύονται στον κατάλογο, οι προδιαγραφές του προτύπου UDDI παρέχουν και δύο βασικούς τύπους προγραμματιστικών διεπαφών APIs: την προγραμματιστική διεπαφή ερώτησης και την προγραμματιστική διεπαφή δημοσίευσης. Η προγραμματιστική διεπαφή ερώτησης αποτελείται από ένα σύνολο λειτουργιών που επιτρέπουν στους χρήστες να λαμβάνουν τις πληροφορίες που αναζητούν από τον κατάλογο UDDI. Έτσι, η αναζήτηση υπηρεσιών μπορεί να γίνει με βάση δύο λίστες λειτουργιών ερώτησης:

- την λίστα λειτουργιών αναζήτησης λεπτομερειών `get_xxx` που περιλαμβάνει τις `get_BussinessDetail`, `get_ServiceDetail`, `get_bindingDetail` και `get_tModelDetail` οι οποίες ανακτούν λεπτομερείς πληροφορίες για κάθε ένα από τις 4 δομές δεδομένων αντίστοιχα με βάση το κλειδί που έχει δοθεί ως είσοδο.

- την λίστα λειτουργιών αναζήτησης `find_xxx` που περιλαμβάνει τις `find_bussiness`, `find_service`, `find_binding` και `find_tModel` οι οποίες ανακτούν ένα σύνολο εγγραφών που ικανοποιούν τα κριτήρια αναζήτησης που έχουν τεθεί ως είσοδο.

Με βάση τις παραπάνω λειτουργίες οι αιτούντες μπορούν να εκτελέσουν την αναζήτηση σε δύο στάδια. Πρώτα η αναζήτηση γίνεται με τη χρήση επόπτη διαδικτύου (browser) για την εύρεση γενικών πληροφοριών η οποία συνήθως ακολουθείται από ένα δεύτερο στάδιο αναζήτησης σε βάθος, με βάση της οποίας αναζητούνται συγκεκριμένες λεπτομερείς πληροφορίες. Για παράδειγμα, οι αιτούντες μπορούν να χρησιμοποιήσουν την λειτουργία `find_bussiness` προκειμένου να αποκτήσει μια λίστα που περιέχει γενικές πληροφορίες για τους παροχείς. Από αυτές τις πληροφορίες ο χρήστης χρησιμοποιεί την λειτουργία `find_service` για να εμφανίζει τις υπηρεσίες εκείνες που ικανοποιούν με το τύπο υπηρεσίας που θέλουμε. Με αυτό τον τρόπο στενεύται η αναζήτηση. Στην συνέχεια αφού χρήστης επιλέξει μια κατάλληλη υπηρεσία παίρνει το αντίστοιχο κλειδί και με βάση αυτό εκτελεί την λειτουργία `get_bindingDetail`. Έπειτα με βάση το κλειδί του κατάλληλου bindingTemplate εκτελεί την λειτουργία `get_tModelDetail` που εμφανίζει πληροφορίες όπως το URL για την ανάκτηση εγγράφου WSDL μιας συγκεκριμένης υπηρεσίας που θέλουμε. Με αυτό το παράδειγμα, φαίνεται η διαδικασία αναζήτησης από το γενικό προς το ειδικό.

Τέλος, η προγραμματιστική διεπαφή δημοσίευσης ορίζει ένα πιστοποιημένο σύνολο λειτουργιών που επιτρέπει στους οργανισμούς να δημοσιεύουν πληροφορία για εταιρίες, υπηρεσίες ή είδη υπηρεσιών στον κατάλογο UDDI. Το UDDI δεν καθορίζει κάποια μέθοδο πιστοποίησης, εκτός από την απαίτηση για ύπαρξη **κουπονιού πιστοποίησης** (authorization token) και τη χρήση του πρωτοκόλλου HTTPS κατά τις διαδικασίες πιστοποίησης. Έτσι ο κάθε διαχειριστής θα πρέπει να έχει το δικό του μηχανισμό πιστοποίησης.

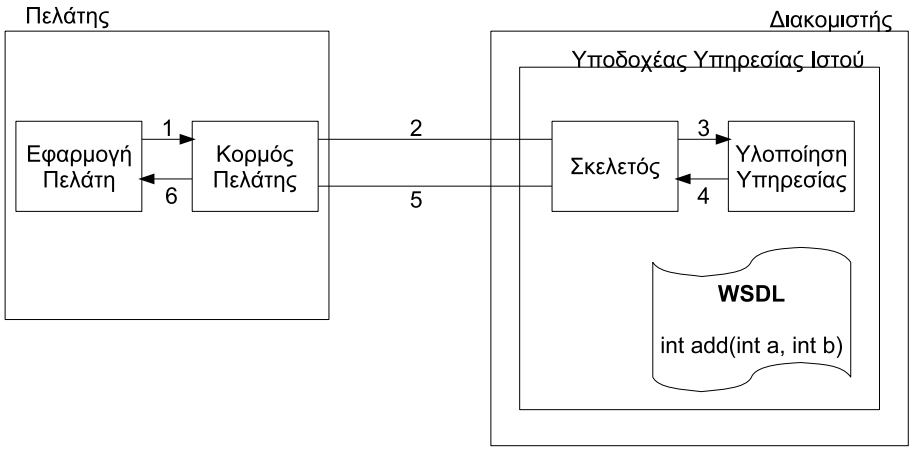
12.8 Ανάπτυξη Εφαρμογής Υπηρεσιών Ιστού

Στην ενότητα αυτή θα δούμε λίγο την διαδικασία προγραμματισμού των υπηρεσιών Ιστού και πως είναι δομημένες οι εφαρμογές υπηρεσιών Ιστού. Ο προγραμματισμός υπηρεσιών Ιστού είναι σχεδόν παρόμοιο με τον προγραμματισμό με Java RMI που παρουσιάζαμε στο προηγούμενο κεφάλαιο.

Πρώτα από όλα γνωρίζουμε ότι υπάρχουν παρά πολλά πρωτόκολλα και γλώσσες που οι προγραμματιστές των υπηρεσιών Ιστού δεν χρειάζονται να γράψουν μια γραμμή κώδικα του SOAP ή της WSDL. Όταν φτάσουμε σε ένα σημείο που η εφαρμογή πελάτη χρειαστεί να καλέσει μια υπηρεσία Ιστού συνήθως εξουσιοδοτούμε την εργασία της κλήσης σε ένα τμήμα λογισμικού που λέγεται κορμός πελάτης (ή πληρεξούσιο). Ο ρόλος του κορμού πελάτη είναι να μεταφράσει την κλήση αίτηση του πελάτη και την δημιουργία ενός μηνύματος αίτησης SOAP. Επίσης, ο κορμός είναι υπεύθυνος να λαμβάνει τα μηνύματα απάντησης SOAP και να τα προωθεί στο πρόγραμμα πελάτη. Έτσι, υπάρχουν πολλά εργαλεία για την αυτόματη παραγωγή κορμού πελάτη και συνήθως παίρνουν ως είσοδος την περιγραφή υπηρεσίας Ιστού WSDL.

Επίσης, ο προγραμματισμός στην πλευρά του παροχέα υπηρεσιών (ή διακομιστή) είναι απλός. Έτσι δεν χρειάζεται να γράψουμε ένα πολύπλοκο πρόγραμμα διακομιστή το οποίο δυναμικά μεταφράζει τις εισερχόμενες αιτήσεις SOAP και παράγει τις απαντήσεις SOAP. Για αυτό το λόγο παράγουμε ένα κορμό διακομιστή (ή σκελετός) ο οποίος είναι υπεύθυνος για την μετάφραση των αιτήσεων και την προώθηση των αιτήσεων αυτών στην υλοποίηση της υπηρεσίας. Όταν η υλοποίηση της υπηρεσίας προκύψει ένα αποτέλεσμα, τότε το διανέμει στον σκελετό ο οποίος θα παράγει το αντίστοιχο μήνυμα απάντησης SOAP. Ο σκελετός επίσης παράγεται από μια περιγραφή WSDL. Επιπλέον, η υλοποίηση της υπηρεσίας και ο σκελετός φιλοξενούνται από ένα τμήμα λογισμικού που λέγεται **υποδοχέας υπηρεσίας Ιστού** (Web Service container) ο οποίος θα λαμβάνει τις εισερχόμενες αιτήσεις HTTP για μια υπηρεσία Ιστού και θα τις προωθεί στο σκελετό. Υπάρχουν διάφορα περιβάλλοντα λογισμικού τα οποία φιλοξενούν τις υπηρεσίες Ιστού όπως J2EE, Apache Axis (Apache eXtensible Interaction System), IBM Websphere και Microsoft .NET. Σε αυτό το βιβλίο θα επικεντρωθούμε το λογισμικό Apache Axis για την φιλοξενία των υπηρεσιών Ιστού όπως θα δούμε στην επόμενη ενότητα.

Συνεπώς, στο Σχήμα 12.5 παρουσιάζονται τα ολοκληρωμένα βήματα που είναι απαραίτητα για την κλήση μιας υπηρεσίας Ιστού. Υποθέτουμε ότι έχουμε ήδη εντοπίσει την υπηρεσία Ιστού και



Σχήμα 12.5: Κλήσης υπηρεσίας Ιστού

έχουμε παράγει το κορμό πελάτη και σκελετός από την περιγραφή WSDL που είναι απαραίτητα για την εφαρμογή μας.

1. Η εφαρμογή πελάτη χρειάζεται να καλέσει μια υπηρεσία Ιστού και επομένως καλεί το κορμό πελάτη. Ο κορμός πελάτης θα μετατρέψει αυτή την τοπική κλήση σε ένα μήνυμα αίτησης SOAP. Η διαδικασία αυτή συνήθως ονομάζεται πρόταξη.

2. Το μήνυμα αίτησης SOAP μεταφέρεται στο δίκτυο μέσω του πρωτοκόλλου HTTP. Ο υποδοχέας υπηρεσίας Ιστού θα λάβει την αίτηση HTTP, θα εξάγει το όνομα της μεθόδου από την επικεφαλίδα `action` της αίτησης HTTP και προωθεί το φάκελο SOAP που περιέχει μέσα στην αίτηση HTTP στο κατάλληλο σκελετό. Ο σκελετός θα μετατρέψει την αίτηση SOAP σε κάποια μορφή που να είναι κατανοητή από την υλοποίηση της υπηρεσίας. Η διαδικασία αυτή ονομάζεται αποπρόταξη.

3. Η υλοποίηση υπηρεσίας θα λάβει την αίτηση από τον σκελετό και εκτελεί την εργασία που έχει ζητηθεί. Για παράδειγμα, αν καλέσαμε την μέθοδο `int add(int a, int b)`, τότε η υλοποίηση υπηρεσίας θα εκτελέσει την πράξη της πρόσθεσης δύο αριθμών.

4. Το αποτέλεσμα της πράξης μεταβιβάζεται στο σκελετό ο οποίος θα μετατρέψει πάλι το αποτέλεσμα σε ένα μήνυμα απάντησης SOAP.

5. Το μήνυμα απάντησης SOAP μεταφέρεται στο δίκτυο μέσω του πρωτοκόλλου HTTP.

Στην συνέχεια, ο κορμός πελάτης θα λάβει την απάντηση SOAP και θα την μετατρέψει σε κάποια μορφή που να είναι κατανοητή από την εφαρμογή πελάτης.

6. Τέλος, η εφαρμογή θα λάβει το αποτέλεσμα της κλήσης υπηρεσίας Ιστού και θα το χρησιμοποιήσει για άλλο σκοπό.

Γενικά, για την ανάπτυξη μιας εφαρμογής υπηρεσίας Ιστού βασισμένη στο μοντέλο πελάτη - διακομιστή ακολουθούμε μια σειρά από βήματα. Το πρώτο βήμα είναι η συγγραφή της κλάσης που περιέχει την υλοποίηση υπηρεσίας Ιστού σε κώδικα Java. Στην κλάση αυτή δεν ορίζεται η μέθοδος `main()` παρά μόνο η υλοποίηση των μεθόδων που θα προσφέρει η υπηρεσία Ιστού. Συνεπώς, η υπηρεσία Ιστού είναι ένα αντικείμενο που προσφέρει ένα σύνολο λειτουργιών. Στην συνέχεια, η υλοποίηση της υπηρεσίας τοποθετείται στον υποδοχέα υπηρεσιών Ιστού ενός υπολογιστή διακομιστή ώστε να αρχίζει να εκτελείται. Η διεύθυνση URL της υπηρεσίας Ιστού αποτελείται από την συνένωση της URL υποδοχέα του διακομιστή και το όνομα της υπηρεσίας.

Το επόμενο βήμα είναι η παραγωγή της περιγραφής υπηρεσίας Ιστού WSDL. Η παραγωγή του αρχείου WSDL γίνεται με την βοήθεια διαφόρων εργαλείων ή του υποδοχέα υπηρεσιών Ιστού παίρνοντας ως είσοδο τον κώδικα υλοποίησης της υπηρεσίας Ιστού. Στην συνέχεια, το αρχείο WSDL θα υποστεί επεξεργασία από κάποιο μεταγλωττιστή προκειμένου να παράγει τους κορμούς πελάτη και διακομιστή οι οποίες είναι απαραίτητες για την πρόταξη και αποπρόταξη των δεδομένων αλλά και την επικοινωνία μεταξύ εφαρμογή πελάτη και υπηρεσία Ιστού. Επίσης, από την παραπάνω μεταγλώττιση προκύπτει και ένα άλλο αρχείο που περιέχει τον ορισμό της διεπαφής της υπηρεσίας Ιστού σε Java.

Το τελευταίο βήμα είναι ο προγραμματισμός του πελάτη που περιέχει την προσπέλαση της υπηρεσίας Ιστού. Με άλλα λόγια, ο κώδικας του πελάτη θα επικοινωνεί με τα βοηθητικά αρχεία που έχουν παραχθεί από τον προηγούμενο βήμα. Επίσης, ο κώδικας θα περιέχει και κλήσεις μεθόδων της απομακρυσμένης υπηρεσίας Ιστού.

12.9 Μεθοδολογίες Ανάπτυξης Υπηρεσιών Ιστού

Σε αυτή την ενότητα θα περιγράψουμε δύο βασικές μεθοδολογίες που χρησιμοποιούνται για την ανάπτυξη υπηρεσιών Ιστού. Η μία μεθοδολογία βασίζεται στην χρήση του εργαλείου Apache Axis Java Web Service (JWS) και η άλλη βασίζεται στην χρήση του Apache Axis Web Service

Deployment Descriptor (WSDD).

12.9.1 Χρήση Apache Axis Java Web Service (JWS)

Τα βήματα για την ανάπτυξη υπηρεσιών Ιστού σε Java χρησιμοποιώντας το εργαλείο JWS είναι τα εξής:

1. Δημιουργούμε πηγαίο κώδικα Java που θα υλοποιεί μια υπηρεσία Ιστού σαν αρχείο jws (Java Web Service).

2. Παράγουμε όλα τα απαραίτητα αρχεία πηγαίου κώδικα Java που παριστάνουν την υπηρεσία από το αρχείο jws με την χρήση του εργαλείου Apache Axis.

3. Μεταγλωττίζουμε τα αρχεία πηγαίου κώδικα που δημιουργήθηκαν στο προηγούμενο βήμα.

4. Δημιουργούμε ένα πρόγραμμα πελάτη και το μεταγλωττίζουμε.

5. Εκτελούμε το πρόγραμμα πελάτη για την προσπέλαση ή κλήση της υπηρεσίας.

Για την καλύτερη κατανόηση της διαδικασίας ανάπτυξης υπηρεσίας Ιστού θα αναπτύξουμε ένα απλό παράδειγμα. Έτσι, σε αυτό το παράδειγμα θα αναπτύξουμε μια απλή μαθηματική υπηρεσία που θα εκτελεί μια αριθμητική πράξη όπως π.χ. την πρόσθεση δύο ακέραιων αριθμών για ένα πελάτη. Η μαθηματική αυτή υπηρεσία δεν είναι καταστατική που σημαίνει ότι η υπηρεσία δεν κρατά το αποτέλεσμα της προηγούμενης αριθμητικής πράξης.

1. Δημιουργία της υπηρεσίας σε Java. Σε αυτό το βήμα γράφουμε το πρόγραμμα Java που θα υλοποιεί την υπηρεσία πρόσθεσης δύο αριθμών. Έτσι, παρακάτω παρουσιάζεται το πρόγραμμα Java:

```
public class MyMath {
        public int add(int x, int y) {
                return x + y;
        }
}
```

Στην συνέχεια αποθηκεύουμε το παραπάνω πρόγραμμα με όνομα `MyMath` και επέκταση `jws`, δηλαδή ως `MyMath.jws`. Το εργαλείο Axis αναμένει το αρχείο jws να βρίσκεται σε μια από τις διάφορες πρότυπες θέσεις. Για να ακολουθήσουμε την απαίτηση αυτή πρέπει να

δημιουργήσουμε ένα κατάλογο κάτω από το κατάλογο `axis` της Jakarta Tomcat. Δηλαδή, πρέπει να δημιουργήσουμε ένα κατάλογο με όποιο όνομα θελουμε (π.χ. το username μας) κάτω από το κατάλογο `$CATALINA_HOME/webapps/axis/yourusername/`. Η `CATALINA_HOME` είναι μια μεταβλητή περιβάλλοντος που προσδιορίζει την διαδρομή του αρχικού καταλόγου της μηχανής Jakarta Tomcat Java servlet. Στην συνέχεια αντιγράφουμε το αρχείο `jws` κάτω από το κατάλογο που δημιουργήσαμε προηγουμένως έτσι ώστε το Apache Axis να είναι σε θέση να βρίσκει το αρχείο αυτό.

2. Παραγωγή των απαραίτητων αρχείων Java για την υπηρεσία Ιστού. Σε αυτό το βήμα γίνεται ο ορισμός της διεπαφής για την μαθηματική υπηρεσία. Όπως, είναι γνωστό η διεπαφή ορίζεται χρησιμοποιώντας την γλώσσα WSDL που προσδιορίζει ποιές πράξεις μέσω της υπηρεσίας Ιστού θα είναι διαθέσιμες στα προγράμματα πελάτες. Σε αυτό το παράδειγμα, πρέπει να έχουμε την διεπαφή σε Java από το έγγραφο WSDL. Για το λόγο αυτό, το Apache Axis διαθέτει το εργαλείο WSDL2Java που παράγει τα απαραίτητα αρχεία Java για την υλοποίηση της υπηρεσίας από το αρχείο `MyMath.jws` που δημιουργήσαμε στο προηγούμενο βήμα. Έτσι, χρησιμοποιούμε την εντολή:

```
java -classpath \$AXISCLASSPATH org.apache.axis.wsdl.WSDL2Java \
 http://localhost:8080/axis/yourusername/MyMath.jws?wsdl
```

Η δεύτερη παράμετρος της παραπάνω εντολής είναι μια διεύθυνση URL που προσδιορίζει το διακομιστή Ιστού (στο παράδειγμα μας είναι ο localhost) που βρίσκεται το αρχείο `MyMath.jws`.

Το αποτέλεσμα της εκτέλεσης του προγράμματος WSDL2Java είναι η δημιουργία ενός καταλόγου με όνομα `localhost` μέσα στον τρέχοντα κατάλογο. Μέσα στον κατάλογο `localhost` υπάρχει ένας υποκατάλογος με όνομα `axis`, ο οποίος έχει έναν άλλο υποκατάλογο με όνομα το `yourusername` που πάλι έχει ακόμα έναν υποκατάλογο με όνομα `MyMath_jws`. Παρατηρούμε λοιπόν ότι δημιουργήθηκαν μια σειρά καταλόγων σύμφωνα με τα μέρη της διεύθυνσης `URL http://localhost:8080/axis/yourusername/MyMath.jws?wsdl`. Μέσα στον κατάλογο `MyMath_jws` υπάρχουν τέσσερα αρχεία πηγαίου κώδικα Java τα οποία είναι τα εξής:

- `MyMath.java`: διεπαφή Java για την κλάση `MyMath`
- `MyMathService.java`: διεπαφή Java που περιλαμβάνει την μέθοδο `getMyMath`

- `MyMathServiceLocator.java`: κλάση Java `MyMathServiceLocator`
- `MyMathSoapBindingStub.java`: κλάση Java `MyMathSoapBindingStub`

Τα παραπάνω αρχεία είναι απαραίτητα ώστε η `MyMath` να μπορεί να χρησιμοποιηθεί ως υπηρεσία Ιστού.

3. Μεταγλώττιση των παραγόμενων αρχείων Java. Στο βήμα αυτό μεταγλωττίζουμε τα τέσσερα αρχεία Java που δημιουργήθηκαν από το προηγούμενο βήμα με την παρακάτω εντολή:

```
javac -classpath \$AXISCLASSPATH   \
 localhost/axis/yourusername/MyMath_jws/*.java
```

4. Σύνταξη προγράμματος πελάτη μιας υπηρεσίας Ιστού. Έτσι, στο βήμα αυτό γράφουμε ένα πρόγραμμα πελάτη σε Java που θα υλοποιεί την κλήση μιας υπηρεσίας Ιστού. Παρακάτω, παρουσιάζεται το πρόγραμμα αυτό:

```
import localhost.axis.MyMath_jws.MyMathServiceLocator;
import localhost.axis.MyMath_jws.MyMathService;
import localhost.axis.MyMath_jws.MyMath;

public class MyMathClient {
  public static void main(String args[]) throws Exception {
    MyMathService service = new MyMathServiceLocator();
    MyMath myMath = service.getMyMath();
    int x = (new Integer(args[0])).intValue();
    int y = (new Integer(args[1])).intValue();
    System.out.println("The␣add␣of␣" + args[0] + "␣and␣" + args[1]
                +  "␣is␣" + myMath.add(x,y));
  }
}
```

Στο παραπάνω πρόγραμμα εισάγουμε τρείς δηλώσεις `import` όπου καλέσαμε το πρόγραμμα WSDL2Java χρησιμοποιώντας το υπολογιστή localhost. Τέλος, το παραπάνω πρόγραμμα αποθηκεύεται ως `MyMathClient.java`.

5. Μεταγλώττιση του προγράμματος πελάτη. Στην συνέχεια, μεταγλωττίζουμε το πρόγραμμα πελάτη με την παρακάτω εντολή:

```
javac -classpath \$AXISCLASSPATH:. MyMathClient.java
```

6. Εκτέλεση προγράμματος υπηρεσίας Ιστού. Εκτελούμε το πρόγραμμα πελάτη με την εντολή:

```
java -classpath \$AXISCLASSPATH:. MyMathClient 6 5
```

Μετά την εκτέλεση του προγράμματος `MyMathClient`, παίρνουμε την παρακάτω έξοδο:

```
The add of 6 and 5 is 11
```

12.9.2 Χρήση Apache Axis Web Service Deployment Descriptor (WSDD)

Η ανάπτυξη υπηρεσίας Ιστού με την προσέγγιση JWS είναι απλή διότι το Apache Axis αυτόματα εντοπίζει το αρχείο jws, μεταγλωττίζει την κλάση και μετατρέπει σωστά τις κλήσεις SOAP σε κλήσεις Java της υπηρεσίας. Όμως, η προσέγγιση αυτή έχει περιορισμούς όπως μπορεί να αναπτύξει μόνο απλές υπηρεσίες Ιστού και δεν υποστηρίζει σύνθετους τύπους δεδομένων. Για το λόγο αυτό υπάρχει η προσέγγιση WSDD που μπορεί να αναπτύξει υπηρεσίες Ιστού που υποστηρίζουν σύνθετους τύπους δεδομένων. Τα βήματα για την ανάπτυξη της υπηρεσίας Ιστού με την προσέγγιση WSDD είναι παρόμοια με εκείνη την JWS. Επομένως, τα βήματα για την ανάπτυξη υπηρεσιών Ιστού σε Java είναι τα εξής:

1. Δημιουργούμε πηγαίο κώδικα Java που θα υλοποιεί μια υπηρεσία Ιστού.
2. Γράφουμε ένα αρχείο WSDD για την υπηρεσία.
3. Εγκατάσταση του αρχείου WSDD στο Apache Axis.
4. Παράγουμε όλα τα απαραίτητα αρχεία πηγαίου κώδικα Java που παριστάνουν την υπηρεσία από το έγγραφο WSDL με την χρήση του εργαλείου Apache Axis.
5. Μεταγλωττίζουμε τα αρχεία πηγαίου κώδικα που δημιουργήθηκαν στο προηγούμενο βήμα.
6. Δημιουργούμε ένα πρόγραμμα πελάτη και το μεταγλωττίζουμε.
7. Εκτελούμε το πρόγραμμα πελάτη για την προσπέλαση ή κλήση της υπηρεσίας.

Θα αναλύσουμε τα παραπάνω βήματα με το παράδειγμα της μαθηματικής υπηρεσίας που είχαμε αναπτύξει για την προηγούμενη προσέγγιση JWS.

1. Δημιουργία της υπηρεσίας σε Java. Σε αυτό το βήμα γράφουμε το πρόγραμμα Java που θα υλοποιεί την υπηρεσία πρόσθεσης δύο αριθμών. Εδώ, δεν παρουσιάζουμε το πρόγραμμα Java αφού είναι ακριβώς ίδιο με το πρόγραμμα στο βήμα 1 της προηγούμενης προσέγγισης. Όμως, αποθηκεύουμε το πρόγραμμα με όνομα `MyMath` και επέκταση `java`, δηλαδή ως `MyMath.java`. Στην συνέχεια, μεταγλωττίζουμε το αρχείο αυτό ώστε να παράγουμε τα αρχεία `class`. Σύμφωνα με το εργαλείο Axis πρέπει τα αρχεία `class` να τοποθετηθούν στις κατάλληλες θέσεις. Για το λόγο αυτό πρέπει να δημιουργήσουμε ένα κατάλογο με όποιο όνομα θέλουμε (π.χ. το username μας) κάτω από το κατάλογο `$CATALINA_HOME/webapps/axis/WEF-INF/classes/` της Jakarta Tomcat. Τέλος, αντιγράφουμε το αρχείο `class` κάτω από το κατάλογο που δημιουργήσαμε προηγουμένως έτσι ώστε το Apache Axis να είναι σε θέση να βρίσκει το αρχείο αυτό.

2. Δημιουργία αρχείου WSDD. Για να αναπτύξουμε την υπηρεσία πρέπει να παραμετροποιήσουμε το αρχείο WSDD για την περιγραφή της υπηρεσίας. Έτσι, παρακάτω φαίνονται τα περιεχόμενα του αρχείου `deploy.wsdd` για την μαθηματική υπηρεσία που θέλουμε να αναπτύξουμε:

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
            xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
<service name="MyMath" provider="java:RPC">
 <parameter name="className" value="MyMath"/>
 <parameter name="allowedMethods" value="*"/>
</service>
</deployment>
```

   Η δομή του αρχείου WSDD περιέχει ένα στοιχείο XML το `deployment`. Το στοιχείο αυτό ορίζει το όνομα περιοχής Java. Μέσα στο στοιχείο αυτό υπάρχει το στοιχείο `service` που ορίζει την υπηρεσία. Το στοιχείο `service` μπορεί να είναι ένα ή όλα από τα παρακάτω:

   - ένα request flow
   - ένα pivot Handler το οποίο είναι το service provider
   - ένα response flow

Στην περίπτωση μας, το provider είναι η ''Java:RPC'', δηλαδή κλήση απομακρυσμένων διαδικασιών. Η ετικέτα `parameter` προσδιορίζει ποια κλάση (π.χ. `MyMath`) πρέπει να φορτώσει ο παροχέας και ποιοι μέθοδοι της κλάσης πρέπει να καλούνται. Στο παράδειγμα μας, προσδιορίζουμε ότι θα φορτωθεί η κλάση `MyMath` και ότι θα καλείται οποιαδήποτε μέθοδο της κλάσης. Πρέπει να σημειώσουμε ότι τα αρχεία WSDD είναι επαναχρησιμοποιήσιμα και ότι απαιτούν ελάχιστες τροποποιήσεις στην περίπτωση που θέλουμε να χρησιμοποιήσουμε άλλες υπηρεσίες όπως π.χ. το όνομα της υπηρεσίας και το όνομα της κλάσης.

3. Εγκατάσταση του αρχείου WSDD στο Apache Axis. Στο βήμα αυτό πρέπει να περάσουμε το αρχείο `wsdd` που δημιουργήσαμε από το προηγούμενο βήμα στο εργαλείο AdminClient του Apache Axis. Το οποίο θα μεταγλωττίζει την υπηρεσία Ιστού με βάση τις παραμέτρους που περιγράφει το αρχείο `wsdd` και θα εγκατασταθεί η υπηρεσία στην κατάλληλη θέση με την παρακάτω εντολή:

```
java -cp $AXISCLASSPATH org.apache.axis.client.AdminClient /
  -ltthp://localhost/axis/services/AdminService deploy.wsdd
```

Για να επιβεβαιώσουμε ότι η υπηρεσία Ιστού εγκαταστάθηκε σωστά, μπορούμε να ελέγξουμε την λίστα των εγκατεστημένων υπηρεσιών του περιβάλλοντος Axis. Για να γίνει αυτό χρησιμοποιούμε ένα περιηγητή και κάνουμε κλικ στην επιλογή ''View the list of deployed Web services'' πάνω στην αρχική σελίδα του Axis. Στην συνέχεια, εμφανίζεται η σελίδα των υπηρεσιών όπου θα πρέπει να φαίνεται και η υπηρεσία μας MyMath αν εγκαταστάθηκε επιτυχώς.

4. Παραγωγή των απαραίτητων αρχείων Java για την υπηρεσία Ιστού. Σε αυτό το βήμα είναι παρόμοιο με το βήμα 2 της προσέγγισης JWS. Έτσι, χρησιμοποιούμε την παρακάτω εντολή:

```
java -classpath $AXISCLASSPATH org.apache.axis.wsdl.WSDL2Java \
 http://localhost:8080/axis/services/MyMath.wsdl
```

Το αποτέλεσμα της εκτέλεσης του προγράμματος WSDL2Java είναι ότι δημιουργεί τέσσερα αρχεία πηγαίου κώδικα Java στον κατάλογο `localhost/axis/services/MyMath` τα οποία είναι παρόμοια με εκείνα του βήματος 2 της προσέγγισης JWS.

5. Μεταγλώττιση των παραγόμενων αρχείων Java. Στο βήμα αυτό μεταγλωττίζουμε τα τέσσερα αρχεία Java που δημιουργήθηκαν από το προηγούμενο βήμα με την παρακάτω εντολή:

```
javac -classpath \$AXISCLASSPATH   \
 localhost/axis/services/MyMath/*.java
```

6. Σύνταξη προγράμματος πελάτη μιας υπηρεσίας Ιστού. Έτσι, στο βήμα αυτό γράφουμε ένα πρόγραμμα πελάτη σε Java που θα υλοποιεί την κλήση μιας υπηρεσίας Ιστού. Παρακάτω, παρουσιάζεται το πρόγραμμα αυτό:

```
import localhost.*;

public class MyMathClient {
  public static void main(String args[]) throws Exception {
    MyMathService service = new MyMathServiceLocator();
    MyMath myMath = service.getMyMath();
    int x = (new Integer(args[0])).intValue();
    int y = (new Integer(args[1])).intValue();
    System.out.println("The␣add␣of␣" + args[0] + "␣and␣" + args[1]
                +  "␣is␣" + myMath.add(x,y));
  }
}
```

Τέλος, το παραπάνω πρόγραμμα αποθηκεύεται ως `MyMathClient.java`.

7. Μεταγλώττιση του προγράμματος πελάτη. Στην συνέχεια, μεταγλωττίζουμε το πρόγραμμα πελάτη με την παρακάτω εντολή:

```
javac -classpath \$AXISCLASSPATH:. MyMathClient.java
```

8. Εκτέλεση προγράμματος υπηρεσίας Ιστού. Εκτελούμε το πρόγραμμα πελάτη με την εντολή:

```
java -classpath \$AXISCLASSPATH:. MyMathClient 6 5
```

Μετά την εκτέλεση του προγράμματος `MyMathClient`, παίρνουμε την παρακάτω έξοδο:

```
The add of 6 and 5 is 11
```

12.10 Σύγκριση Υπηρεσιών Ιστού με τα Κατανεμημένα Αντικείμενα

Μια υπηρεσία Ιστού είναι μια διεπαφή υπηρεσίας η οποία παρέχει λειτουργίες ή μέθοδους για να προσπελάσουν δεδομένα. Σε απλοικό επίπεδο μπορούμε να πούμε ότι η αλληλεπίδραση μεταξύ πελάτη και διακομιστή είναι σχεδόν παρόμοια με το μοντέλο RMI, όπου ο πελάτης χρησιμοποιεί μια απομακρυσμένη αναφορά αντικειμένου για να καλέσει μια μέθοδο στο απομακρυσμένο αντικείμενο. Στην υπηρεσία Ιστού ο πελάτης χρησιμοποιεί την διεύθυνση URI για να καλέσει μια μέθοδο στο πόρο που προσδιορίζεται από την URI.

Έτσι, η URI της υπηρεσίας Ιστού μπορεί να συγκριθεί με την αναφορά απομακρυσμένου αντικειμένου. Όμως, στο μοντέλο Java RMI το οποίο είναι ένα παράδειγμα συστήματος κατανεμημένων αντικειμένων, τα αντικείμενα που βρίσκονται στο διακομιστή μπορούν να δημιουργήσουν στιγμιότυπα απομακρυσμένων αντικειμένων δυναμικά και επιστρέφουν απομακρυσμένες αναφορές για αυτά τα οποία αποθηκεύονται στον πληρεξούσιο του πελάτη. Οι πελάτες μπορούν να καλέσουν μεθόδους στα απομακρυσμένα αντικείμενα χρησιμοποιώντας το πληρεξούσιο όπου διατηρεί τις απομακρυσμένες αναφορές στα στιγμιότυπα των απομακρυσμένων αντικειμένων. Χρησιμοποιώντας το πληρεξούσιο αυτό, ο πελάτης μπορεί να εναλλάσει την κατάσταση του απομακρυσμένου αντικειμένου αφού παραμένει μέσα στο κατανεμημένο περιβάλλον. Αντίθετα, τίποτα από τα παραπάνω δεν γίνεται με τις υπηρεσίες Ιστού όπου δεν μπορούν να δημιουργήσουν στιγμιότυπα απομακρυσμένων αντικειμένων. Συνεπώς, οι υπηρεσίες Ιστού δεν υποστηρίζουν την έννοια της κατάστασης διότι καλώντας μια ίδια υπηρεσία Ιστού δύο φορές διαδοχικά θα αντιμετωπιστεί ως δυο ανεξάρτητες κλήσεις σε ξεχωριστά στιγμιότυπα της ίδιας υπηρεσίας. Μόλις η κλήση ολοκληρωθεί, η σύνδεση μεταξύ πελάτη - διακομιστή τερματίζεται και καταστρέφονται όλες οι τοπικές πληροφορίες. Φυσικά, θα μπορούσε να αναπτυχθεί η εφαρμογή με τέτοιο τρόπο ώστε να διατηρείται η κατάσταση αλλά αυτό θα περιλαμβάνει ένα επίπεδο πάνω από τις βασικές τεχνολογίες υπηρεσιών Ιστού π.χ. αριθμοί αναγνώρισης θα μπορούσαν να χρησιμοποιηθούν για ενημέρωση απομακρυσμένων βάσεων δεδομένων. Αυτή την λειτουργία υποστηρίζουν οι υπηρεσίες πλέγματος όπως θα δούμε στο επόμενο κεφάλαιο.

Επίσης, ένα άλλο σημείο για το υπάρχει διαφορά ανάμεσα στα μοντέλα υπηρεσιών Ιστού και κατανεμημένα αντικείμενα είναι η εμβέλεια των εφαρμογών. Συγκεκριμένα, το λογισμικό των

κατανεμημένων αντικειμένων επιτρέπει την ανάπτυξη εφαρμογών ειδικά για το εσωδίκτυο ενός οργανισμού ενώ οι υπηρεσίες Ιστού μπορούν να αναπτυχθούν εφαρμογές για το Διαδίκτυο.

Κεφάλαιο 13

Προγραμματισμός Υπηρεσιών Πλέγματος

Στα τελευταία χρόνια, οι υπηρεσίες Ιστού έχουν μια ευρεία αποδοχή από τους επιστήμονες ως ένα κατανεμημένο υπολογιστικό μοντέλο. Οι υπηρεσίες Ιστού βασίζονται σε ανοικτά πρότυπα Διαδικτύου όπως η XML (eXtensible Markup Language) για να περιγράψει απομακρυσμένα συστατικά λογισμικού, μεθόδους με τους οποίους προσπελάζουν τα συστατικά και διαδικασίες με τις οποίες αναζητούνται οι μέθοδοι. Τα προσβάσιμα εκείνα συστατικά λογισμικού ονομάζονται υπηρεσίες και είναι διαθέσιμες από τους παροχείς υπηρεσιών.

Το πλέγμα μπορεί να ενισχυθεί από την τεχνολογία υπηρεσιών Ιστού για διάφορους λόγους: Ένας από τους κύριους λόγους είναι ότι οι υπηρεσίες Ιστού έχουν τη δυνατότητα να υποστηρίξουν τη δυναμική αναζήτηση και σύνθεση των υπηρεσιών σε ετερογενή περιβάλλοντα. Οι υπηρεσίες Ιστού έχουν τους μηχανισμούς για καταχώριση (ή δημοσίευση) και αναζήτηση ορισμών διεπαφών, περιγραφές και για δυναμική παραγωγή πληρεξούσιων υπηρεσιών. Η γλώσσα WDSL παρέχει ένα πρότυπο μηχανισμό για τον ορισμό διεπαφών ξεχωριστά από την ενσωμάτωση τους μέσα σε μια συγκεκριμένη σύνδεση (πρωτόκολλο μεταφοράς και μορφή κωδικοποίησης δεδομένων). Ένας άλλος λόγος είναι ότι οι τεχνολογίες υπηρεσιών Ιστού βασίζονται σε αναγνωρισμένα διεθνή πρότυπα. Η ευρεία υιοθέτηση των υπηρεσιών Ιστού σημαίνει ότι θα επιτρέψει ένα μεγαλύτερο επίπεδο διαλειτουργικότητας και της ικανότητας να εκμεταλλευτεί καινούργια εργαλεία και υπηρεσίες όπως οι Apache Axis και Microsoft .NET.

13.1 OGSA

Το πλαίσιο Ανοιχτή Αρχιτεκτονική Υπηρεσιών Πλέγματος (Open Grid Services Architecture - OGSA) είναι ιδέα του Globus και της IBM σε μια προσπάθεια για σύγκλιση των τεχνολογιών υπηρεσιών Ιστού με το υπολογιστικό πλέγμα. Το OGSA παρουσιάστηκε στην συνάντηση GGF (Global Grid Forum) το Φεβρουάριο 2002 και περιγράφεται στην εργασία τους. Το πλαίσιο OGSA περιγράφει μια καινούργια αρχιτεκτονική για ενσωμάτωση στα κατανεμημένα συστήματα. Ο οργανισμός GGF έχει ιδρύσει μια ομάδα εργασίας Ανοιχτών Υπηρεσιών Πλέγματος για να εξετάζει και να ορίζει την αρχιτεκτονική υπηρεσιών πλέγματος.

Το πλαίσιο OGSA είναι μια προδιαγραφή ανοιχτής αρχιτεκτονικής που ορίζει σημασιολογικά και μηχανισμούς για την δημιουργία, αναζήτηση, προσπέλαση, χρήση, συντήρηση και καταστροφή των υπηρεσιών πλέγματος. Ο στόχος του OGSA είναι να προτυποποιήσει όλες τις υπηρεσίες εκείνες που συναντά σε μια εφαρμογή πλέγματος (όπως οι υπηρεσίες διαχείρισης εργασιών, υπηρεσίες διαχείρισης πόρων, υπηρεσίες ασφάλειας κλπ) ορίζοντας ως ένα σύνολο διεπαφών για τις υπηρεσίες αυτές. Το OGSA ορίζει ως μοντέλο υπηρεσιών πλέγματος που βλέπει το πλέγμα ως ένα εκτεκταμένο σύνολο υπηρεσιών πλέγματος που μπορούν να συναθροιστούν με τις ανάγκες των χρηστών δηλαδή εικονικές επιχειρήσεις και εικονικούς οργανισμούς.

Στην καρδιά του πλαίσιου OGSA είναι η υπηρεσία πλέγματος. Η υπηρεσία πλέγματος είναι απλά μια υπηρεσία Ιστού με πρόσθετα χαρακτηριστικά ώστε να είναι επαρκή για μια εφαρμογή πλέγματος. Έτσι, η υπηρεσία πλέγματος είναι μια υπηρεσία Ιστού που παρέχει ένα σύνολο ορισμένων διεπαφών οι οποίες ακολουθούν συγκεκριμένες συμβάσεις. Συνεπώς το OGSA επεκτείνει τις υπηρεσίες Ιστού εισάγοντας τις παρακάτω διεπαφές και συμβάσεις:

- Αναζήτηση. Οι πελάτες απαιτούν μηχανισμούς για αναζήτηση διαθέσιμων υπηρεσιών και για τον καθορισμό των χαρακτηριστικών εκείνων των υπηρεσιών έτσι ώστε οι αιτήσεις τους να διαμορφωθούν στις κατάλληλες εκείνες υπηρεσίες.

- Δυναμική δημιουργία παροδικών υπηρεσιών. Μέσα σε ένα περιβάλλον πλέγματος οι υπηρεσίες είναι δυναμικές και παροδικές. Δηλαδή οι υπηρεσίες δημιουργούνται και καταστρέφονται δυναμικά καθώς οι πόροι διαμορφώνονται και αλλάζει η κατάσταση του συστήματος. Συνεπώς, οι υπηρεσίες πλέγματος χρειάζονται διεπαφές για να διαχειριστεί την δημιουργία, την καταστροφή και το κύκλο ζωής τους.

- Καταστατικές υπηρεσίες. Οι υπηρεσίες σε ένα πλέγμα πρέπει να έχουν ιδιότητες και δεδομένα που συσχετίζονται με αυτές. Η ιδέα αυτή είναι παρόμοια με την παραδοσιακή μορφή των αντικειμένων στον αντικειμενοστρεφή προγραμματισμό. Έτσι, τα αντικείμενα έχουν συμπεριφορά και δεδομένα.

- Διαχείριση κύκλου ζωής. Σε ένα σύστημα που ενσωματώνει τις παροδικές και καταστατικές υπηρεσίες, οι μηχανισμοί πρέπει να παρέχουν για τη λήψη των υπηρεσιών και κατάστασης που συσχετίζονται με τις αποτυχημένες διαδικασίες.

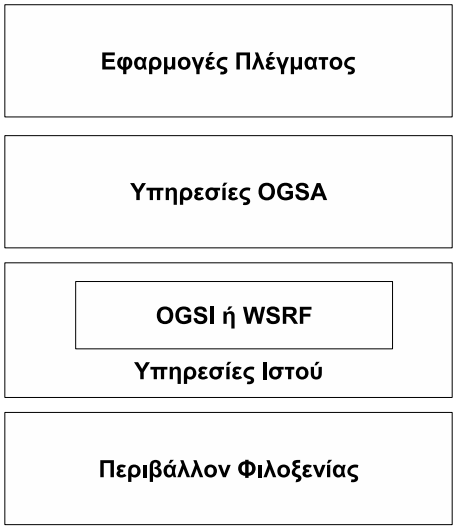
- Ανακοίνωση. Αρχικά οι πελάτες εγγράφονται ως συνδρομητές σε κάποιες υπηρεσίες. Έτσι, μια συλλογή δυναμικών και κατανεμημένων υπηρεσιών πρέπει να είναι σε θέση να ειδοποιήσουν ασύγχρονα στους πελάτες τις ενδιαφέρουσες αλλαγές που συμβαίνουν στην κατάσταση τους.

- Διαχειρισιμότητα. Παρέχονται διαδικασίες που αφορά τη διαχείριση και τη παρακολούθηση ενός μεγάλου αριθμού στιγμιοτύπων υπηρεσιών πλέγματος.

- Απλό περιβάλλον φιλοξενίας. Ένα απλό περιβάλλον εκτέλεσης είναι ένα σύνολο πόρων τοποθετημένων μέσα σε μια ενιαία διαχειριστική περιοχή και υποστηρίζει ευκολίες για τη διαχείριση υπηρεσιών: για παράδειγμα, ένας διακομιστής εφαρμογής J2EE, ένα σύστημα της Microsoft .NET, ή μια συστοιχία Linux.

Όπως φαίνεται στο Σχήμα 13.1, οι εφαρμογές πλέγματος μπορούν να αναπτυχθούν από τις υπηρεσίες OGSA. Οι υπηρεσίες του OGSA αποτελούνται από δύο μέρη: υπηρεσίες πλατφόρμας OGSA και βασικές υπηρεσίες. Οι υπηρεσίες πλατφόρμας OGSA είναι υπηρεσίες πλέγματος που αφορά στην εξουσιοδότηση και αυθεντικότητα χρηστών, ανοχή σφαλμάτων, υποβολή εργασίας, διαχείριση πόρων, παρακολούθηση και πρόσβαση δεδομένων. Οι βασικές υπηρεσίες περιλαμβάνουν κυρίως δημιουργία υπηρεσίας, καταστροφή υπηρεσίας, διαχείριση του κύκλου ζωής υπηρεσίας, καταχώριση υπηρεσίας, αναζήτηση υπηρεσίας και ανακοίνωση υπηρεσίας.

Το OGSA ορίζει διάφορες πλευρές που αφορά σε μια υπηρεσία πλέγματος όπως τα χαρακτηριστικά των υπηρεσιών πλέγματος και ποιες διεπαφές είναι απαραίτητες. Όμως, το OGSA δεν ασχολείται πως οι διεπαφές αυτές πρέπει να υλοποιηθούν. Αυτό το κομμάτι είναι εργασία των OGSI (Open Grid Service Infrastructure) και WSRF (Web Services Resource Framework) οι



Σχήμα 13.1: Ανοιχτή αρχιτεκτονική υπηρεσιών πλέγματος

οποίες είναι δύο τεχνικές προδιαγραφές για το πως θα υλοποιηθούν οι βασικές υπηρεσίες πλέγματος όπως ορίστηκαν στο πρότυπο OGSA στα πλαίσια των υπηρεσιών Ιστού. Στις επόμενες ενότητες θα περιγράψουμε συνοπτικά τις δύο παραπάνω προδιαγραφές.

13.2 OGSI

Η προδιαγραφή αυτή ορίζει ένα πρότυπο χρησιμοποιώντας επεκταμένους ορισμούς σχημάτων WSDL και XML. Έτσι, η OGSI ασχολείται κυρίως με την δημιουργία, διευθυνσιοδότηση και διαχείριση κύκλου ζωής των καταστατικών υπηρεσιών πλέγματος. Με άλλα λόγια, η OGSI εισάγει τις διεπαφές **στιγμιότυπου υπηρεσίας** (service instance) και **δεδομένα υπηρεσίας** (service data) οι οποίες συσχετίζονται σε κάθε υπηρεσία πλέγματος ώστε να υποστηρίξει τις παροδικές και καταστατικές υπηρεσίες πλέγματος αντίστοιχα. Επίσης, η OGSI εισάγει τις διεπαφές `GridService`, `Factory`, `Registration`, `HandleResolver` και `Notification` για να υποστηρίξει τις υπόλοιπες βασικές υπηρεσίες. Συγκεκριμένα, η OGSI ορίζει τις παρακάτω διεπαφές:

13.2.1 Στιγμιότυπα Υπηρεσίας Πλέγματος

Γνωρίζουμε ότι οι υπηρεσίες Ιστού είναι επίμονες ενώ οι υπηρεσίες πλέγματος μπορεί να είναι παροδικές. Οι υπηρεσίες Ιστού που αναφέρονται ως επίμονες επειδή ο κύκλος ζωής τους

είναι στενά συνδεδεμένο με τον υποδοχέα των υπηρεσιών Ιστού, δηλαδή μια υπηρεσία Ιστού είναι διαθέσιμη από την στιγμή που ξεκινά ο διακομιστής και δεν τερματίζεται μέχρι ότου τερματιστεί ο διακομιστής. Προκειμένουν οι υπηρεσίες πλέγματος να είναι παροδικές (δηλαδή να δημιουργούνται και να τερματίζονται σε οποιαδήποτε στιγμή) η OGSI εισήγαγε την έννοια του στιγμιότυπου υπηρεσίας. Ένα στιγμιότυπο υπηρεσίας πλέγματος είναι μια στιγμιοποίηση μιας υπηρεσίας πλέγματος η οποία δημιουργείται και τερματίζεται δυναμικά. Μια υπηρεσία πλέγματος που δημιουργεί ένα στιγμιότυπο ονομάζεται εργοστασιακή υπηρεσία (factory). Γι αυτό το λόγο η OGSI εισάγει την διεπαφή `Factory`. Όταν ένας πελάτης θέλει να δημιουργήσει στιγμιότυπο υπηρεσίας θα ζητήσει την εργοστασιακή υπηρεσία. Όταν όμως ο πελάτης θέλει να καλέσει μια λειτουργία θα προσπελάζει τα στιγμιότυπα υπηρεσίας. Τέλος, κάθε στιγμιότυπο υπηρεσίας πλέγματος προσδιορίζεται μοναδικά από ένα καθολικό χειριστής υπηρεσιών πλέγματος (Global Service Handle - GSH) ώστε να ξεχωρίζει από τα υπόλοιπα στιγμιότυπα υπηρεσιών πλέγματος.

13.2.2 Αναβάθμιση και Επικοινωνία

Η GSH παρέχει μόνο ένα μοναδικό αναγνωριστικό για κάθε στιγμιότυπο και δεν περιλαμβάνει το πρωτόκολλο ή ειδικές πληροφορίες στιγμιοτύπου που απαιτούνται για την αλληλεπίδραση ή επικοινωνία με ένα συγκεκριμένο στιγμιότυπο. Οι πληροφορίες αυτές περιέχονται μέσα σε μια άλλη οντότητα που ονομάζεται **αναφορά υπηρεσίας πλέγματος** (Grid Service Reference - GSR). Ο διαχωρισμός των δύο παραπάνω οντοτήτων πληροφοριών επιτρέπουν στο στιγμιότυπο να αλλάζει ή να αναβαθμίζεται κατά το κύκλο ζωής του χωρίς να απαιτείται ένα νέο μοναδικό αναγνωριστικό. Ενώ η GSH του στιγμιοτύπου είναι στατική, η αναφορά GSR μπορεί να αλλάζει. Για να αποκτήσει μια έγκυρη αναφορά GSR από μια GSH, το OGSA ορίζει μια διεπαφή `HandleMap`.

13.2.3 Δεδομένα Υπηρεσίας

Γνωρίζουμε ότι οι υπηρεσίες Ιστού είναι μη-καταστατικές δηλαδή δεν αποθηκεύουν την κατάσταση των λειτουργιών που εκτέλεσε από μια κλήση σε άλλη ενώ οι υπηρεσίες πλέγματος είναι καταστατικές. Προκειμένουν οι υπηρεσίες πλέγματος να είναι καταστατικές, η OGSI όρισε την έννοια δεδομένα υπηρεσίας. Συγκεκριμένα, σε κάθε στιγμιότυπο υπηρεσίας είναι συσχετισμένο με δεδομένα υπηρεσίας τα οποία είναι μια συλλογή από στοιχεία XML. Τα δεδομένα υπηρεσίας χρησιμοποιούνται για να περιγράψουν πληροφορίες κατάστασης και μεταδεδομένα της υπηρεσίας.

Οι πληροφορίες κατάστασης αφορούν σχετικά με την τρέχουσα κατάσταση μιας υπηρεσίας ενώ τα μεταδεδομένα αφορούν πληροφορίες σχετικά με την περιγραφή του στιγμιότυπου υπηρεσίας.

13.2.4 Αναζήτηση Υπηρεσίας

Οι GSHs μπορούν αρχικά να ληφθούν μέσω ενός καταλόγου ή μητρώου, του οποίου είναι μια υπηρεσία πλέγματος που υποστηρίζει αναζήτηση υπηρεσιών. Μια υπηρεσία καταλόγου ορίζονται από τη διεπαφή `Registry` και από δεδομένα υπηρεσίας που περιέχει πληροφορίες σχετικά με καταχωρημένες GSHs. Η διεπαφή `Registry` παρέχει ένα σύνολο λειτουργιών που επιτρέπουν την εγγραφή μιας GSH. Ένας πελάτης μπορεί να χρησιμοποιήσει την λειτουργία FindServiceData της διεπαφής `GridService` για να ανακτήσει πληροφορίες ή δεδομένα υπηρεσίας σχετικά με την υπηρεσία πλέγματος που είναι καταχωρημένη σε ένα μητρώο, π.χ. η GSH του στιγμιότυπου υπηρεσίας, την θέση της εργοστασιακής υπηρεσίας, κλπ.

13.2.5 Ανακοίνωση

Μια υπηρεσία πλέγματος μπορεί να διαμορφωθεί ως **πηγή ανακοίνωσης** (notification source) και συγκεκριμένοι πελάτες να είναι **παραλήπτες ανακοίνωσης** (notification sinks) (ή αλλιώς συνδρομητές). Αυτό σημαίνει ότι αν μια αλλαγή εμφανιστεί σε μια υπηρεσία πλέγματος, η αλλαγή αυτή ανακοινώνεται στους συνδρομητές. Σημειώνουμε ότι δεν είναι απαραίτητο να ανακοινώνονται όλες οι αλλαγές αλλά μόνο εκείνες τις υπηρεσίες πλέγματος που ενδιαφέρεται ο προγραμματιστής. Έτσι, ένας πελάτης μπορεί να καλέσει την υπηρεσία ανακοίνωσης χρησιμοποιώντας την λειτουργία συνδρομής και έχοντας τη GSH της πηγής ανακοίνωσης. Επίσης, ο συνδρομητής απαιτείται να στείλει στην πηγή περιοδικά μηνύματα ώστε να συνεχίσει να λαμβάνει τις ανακοινώσεις. Τα μηνύματα αυτά χρησιμοποιούνται επίσης για να διαχειριστούν τη διάρκεια ζωής μιας παροδικής υπηρεσίας.

13.2.6 Διαχείριση Κύκλου Ζωής Υπηρεσίας

Η εισαγωγή των παροδικών υπηρεσιών δημιουργεί το πρόβλημα διαχείρισης και προσδιορισμού διάρκειας ζωής υπηρεσιών. Δεδομένου ότι το πλέγμα είναι ανοικτό και δυναμικό, τα διάφορα συστατικά μπορούν να αποτύχουν και μια δημιουργημένη υπηρεσία ίσως να μην μπορεί να

ολοκληρωθεί ρητά από τον πελάτη. Η OGSI λύνει το πρόβλημα αυτό μέσω μιας προσέγγισης υλισμικού κατάστασης η οποία αποτελούνται από λειτουργίες για τη διαπραγμάτευση μιας αρχικής διάρκεια ζωής, αιτήματα επέκτασης και αποκομιδή στιγμιοτύπων υπηρεσιών μετά από τη λήξη διάρκειας ζωής. Κατά τη διάρκεια της διαπραγμάτευσης για την αρχική διάρκειας ζωής, ο πελάτης προσδιορίζει ένα ελάχιστο και μέγιστο αρχικό χρόνο ζωής. Από αυτό το διάστημα, το εργοστάσιο επιλέγει μια αρχική διάρκεια ζωής η οποία επιστρέφεται στο πελάτη. Ένας πελάτης μπορεί να επεκτείνει μια διάρκεια ζωής προσδιορίζοντας μια νέα ελάχιστη και μέγιστη διάρκεια ζωής χρησιμοποιώντας το μήνυμα `SetTerminationTime`. Εάν η διάρκεια ζωής ενός στιγμιοτύπου υπηρεσίας λήγει, το περιβάλλον φιλοξενίας τερματίζει την υπηρεσία και αφήνει τους συσχετισμένους πόρους.

13.2.7 Ομάδες Υπηρεσίας

Μια υπηρεσία μπορεί να διαμορφωθεί ως ομάδα υπηρεσίας η οποία συναθροίζει άλλες υπηρεσίες. Με αυτό το τρόπο μπορούμε εύκολα να εκτελέσουμε τις λειτουργίες όπως «προσθήκη νέας υπηρεσίας στην ομάδα», «αφαίρεση μιας υπηρεσίας από την ομάδα» και (η πιο σημαντική) «αναζήτηση μιας υπηρεσίας στην ομάδα που ικανοποιεί την συνθήκη FOOBAR». Αν και η λειτουργικότητα που παρέχεται από αυτήν την προδιαγραφή είναι πολύ βασική, είναι εν τούτοις η βάση των ισχυρών υπηρεσιών αναζήτησης (όπως GT3 IndexService) οι οποίες μας επιτρέπουν να ομαδοποιήσουμε διαφορετικές υπηρεσίες μαζί και να προσπελάζονται σε αυτές μέσω ενός ενιαίου σημείου εισόδου (ή ομάδα υπηρεσιών).

13.2.8 Επέκταση portType

Στο προηγούμενο κεφάλαιο είδαμε ότι μια υπηρεσία Ιστού ορίζει την διεπαφή του (δηλαδή, τις λειτουργίες που θα εκτελέσει) μέσω ενός εγγράφου WSDL. Η διεπαφή συνήθως ονομάζεται `portType`. Μια κανονική υπηρεσία Ιστού μπορεί να έχει ένα `portType`. Από την άλλη μεριά, οι υπηρεσίες πλέγματος υποστηρίζουν επέκταση `portType` το οποίο σημαίνει ότι μπορούμε να ορίζουμε ένα `portType` ως επέκταση του προηγουμένου `portType`.

Το λογισμικό Globus Toolkit έκδοση 3 είναι μια υλοποίηση αναφοράς της προδιαγραφής OGSI. Όμως, η προδιαγραφή OGSI έχει προκαλέσει μεγάλη ανησυχία στην κοινότητα των υπηρεσιών Ιστού για διάφορους λόγους όπως θα δούμε παρακάτω κυρίως στο γεγονός ότι

οι υπηρεσίες πλέγματος δεν ήταν συμβατά με τα πρότυπα υπηρεσιών Ιστού. Με βάση τους λόγους αυτούς, η προδιαγραφή OGSI αποδοκιμάστηκε και αντικαταστάθηκε από μια καινούργια προδιαγραφή που ονομάζεται WSRF. Συνεπώς, η κοινότητα των υπηρεσιών Ιστού προσδιόρισε τρία βασικά προβλήματα στην προδιαγραφή OGSI:

- Όλα σε μια μόνο προδιαγραφή. Η OGSI όριζε πολλές περιοχές ή υπηρεσίες σε μια μόνο προδιαγραφή. Ενώ η προδιαγραφή WSRF χωρίζει τις λειτουργίες του OGSI σε ένα σύνολο προδιαγραφών.

- Ασυμβατότητα με τα εργαλεία υπηρεσιών Ιστού. Η OGSI χρησιμοποιεί εκτενώς το σχήμα XML, δηλαδή υπάρχει η συχνή χρήση των χαρακτηριστικών `xsd:any`. Το χαρακτηριστικό αυτό προκαλεί προβλήματα με π.χ το JAX-RPC. Στην προδιαγραφή WSRF μειώνει την χρήση του σχήματος XML.

- Αντικειμενοστρεφής προσέγγιση. Η OGSI μοντελοποιεί μια καταστατική υπηρεσία πλέγματος ως μια υπηρεσία Ιστού που ενθυλακώνει την κατάσταση των πόρων δηλαδή οι καταστάσεις υπηρεσίας και πόρου είναι μαζί. Στην WSRF, διαχωρίζονται η υπηρεσία από την κατάσταση του πόρου όπως θα δούμε παρακάτω.

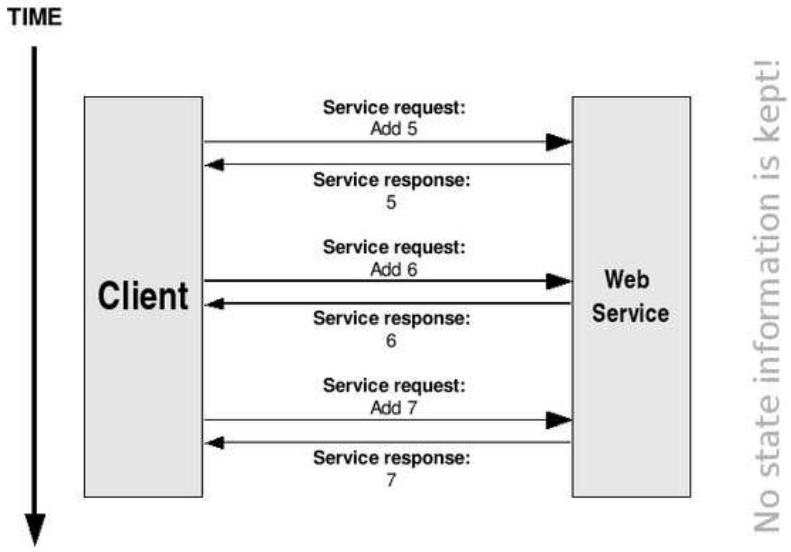
13.3 WSRF

Γνωρίζουμε ότι η τεχνολογία υπηρεσιών Ιστού είναι η φυσική επιλογή για την κατασκευή επόμενης γενιάς εφαρμογών πλέγματος. Όμως, οι υπηρεσίες Ιστού έχουν ορισμένους περιορισμούς. Στην πραγματικότητα, οι πραγματικές υπηρεσίες Ιστού (όπως ορίστηκε από W3C) δεν θα μπορούσε να ήταν πολύ χρήσιμες για την ανάπτυξη μιας εφαρμογής πλέγματος. Η εισαγωγή του πλαίσιου WSRF που αναπτύχθηκε από τον οργανισμό OASIS (Organization for the Advancement of Structured Information Standards) και που αντικατέστησε την OGSI ώστε να υλοποιήσει τις καταστατικές υπηρεσίες Ιστού. Έτσι, η WSRF βελτιώνει διάφορες πλευρές των υπηρεσιών Ιστού και τις καθιστά περισσότερο επαρκείς για τις εφαρμογές πλέγματος.

Σε αυτή την ενότητα θα εξετάζουμε μια συνοπτική ματιά στα διαφορετικά μέρη της προδιαγραφής WSRF. Όμως, πριν προχωρήσουμε σε αυτό αξίζει να δούμε μια ματιά στην βασική βελτίωση της προδιαγραφής WSRF: την καταστατικότητα.

13.3.1 WSRF: Κατάσταση

Οι υπηρεσίες Ιστού είναι συνήθως μη καταστατικές (ακόμα κι αν, θεωρητικά, δεν υπάρχει τίποτα στην αρχιτεκτονική υπηρεσιών Ιστού που να λέει ότι δεν μπορούν να είναι καταστατικές). Αυτό σημαίνει ότι η υπηρεσία Ιστού δεν μπορεί «να θυμάται» πληροφορίες, ή να κρατήσει την κατάσταση από μια κλήση σε άλλη. Για παράδειγμα, φανταστείτε ότι θέλουμε να προγραμματίσουμε μια πολύ απλή υπηρεσία Ιστού που ενεργεί απλά ως ένας αθροιστής ακέραιων αριθμών. Ο αθροιστής αυτός αρχικά έχει τιμή μηδέν, και θέλουμε να προσθέσουμε τις διάφορες τιμές σε αυτό τον αθροιστή. Υποθέτουμε ότι έχουμε μια λειτουργία add η οποία λαμβάνει την τιμή που προσθέτει και επιστρέφει την τρέχουσα τιμή του αθροιστή. Όπως φαίνεται στον Σχήμα 13.2, η πρώτη κλήση λειτουργίας add φαίνεται να λειτουργεί (ζητάμε να προστεθεί η τιμή 5 και λαμβάνουμε το 5 ως άθροισμα). Όμως, δεδομένου ότι μια υπηρεσία Ιστού είναι μη καταστατική, οι ακόλουθες κλήσεις δεν έχουν καμία ιδέα για τις λειτουργίες που εκτελέστηκαν στις προηγούμενες κλήσεις. Έτσι, στη δεύτερη κλήση στην λειτουργία add παίρνουμε πίσω 6, αντί 11 (που θα ήταν η αναμενόμενη τιμή εάν η υπηρεσία Ιστού ήταν σε θέση να κρατήσει την κατάσταση).

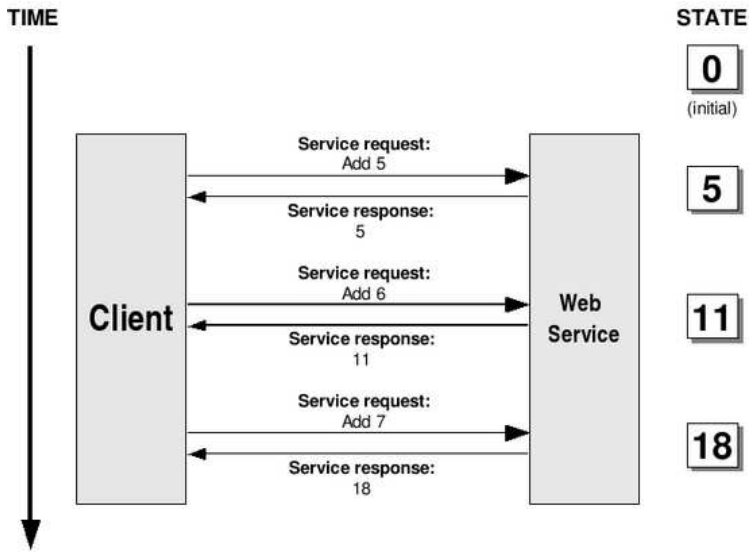


Σχήμα 13.2: Μια κλήση μη-καταστατικής υπηρεσίας Ιστού

Το γεγονός ότι οι υπηρεσίες Ιστού δεν κρατούν πληροφορίες κατάστασης δεν είναι απαραίτητα κακή πρακτική. Υπάρχει μια ποικιλία εφαρμογών που δεν χρειάζονται την καταστατικότητα. Για παράδειγμα, η υπηρεσία Ιστού καιρού που είδαμε στο προηγούμενο κεφάλαιο είναι

πραγματική η οποία δεν χρειάζεται να γνωρίζει τι συνέβη στις προηγούμενες κλήσεις.

Όμως, οι εφαρμογές πλέγματος γενικά απαιτούν κατάσταση. Έτσι, ιδανικά θα επιθυμούσαμε η παραπάνω υπηρεσία Ιστού να κρατήσει τις πληροφορίες κατάστασης:



Σχήμα 13.3: Μια κλήση καταστατικής υπηρεσίας Ιστού

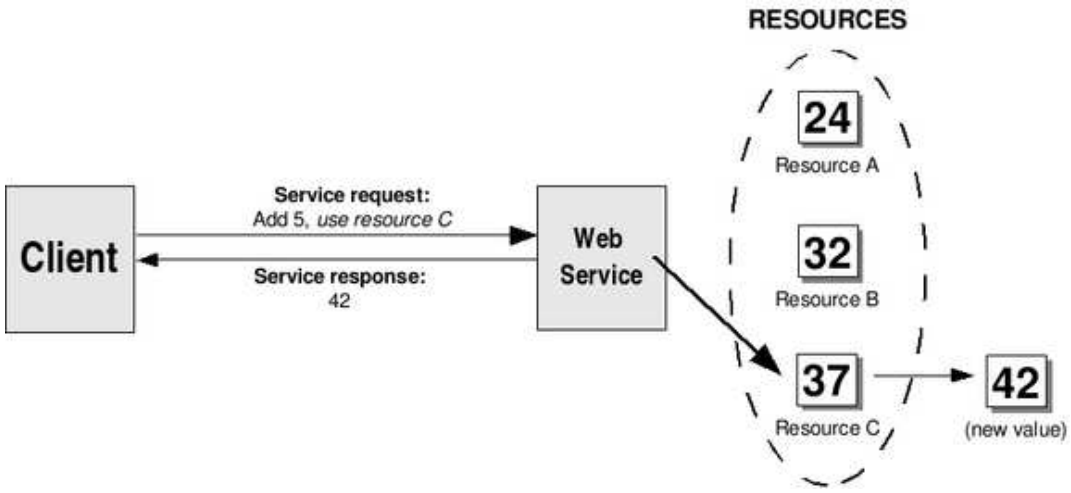
Όμως, αυτό είναι ένα αρκετά ιδιαίτερο δίλημμα δεδομένου ότι μια υπηρεσία Ιστού είναι συνήθως μια οντότητα μη-καταστατική.

13.3.2 Η Προσέγγιση Πόρων για την Κατάσταση

Δίνοντας την δυνατότητα στις υπηρεσίες Ιστού να αποθηκεύουν τις πληροφορίες κατάστασης ενώ διατηρώντας ακόμα τις υπηρεσίες μη-καταστατικές φαίνεται σαν ένα πολύπλοκο πρόβλημα. Ευτυχώς, είναι ένα πρόβλημα με μια πολύ απλή λύση: απλά αποθηκεύουμε την υπηρεσία Ιστού και τις πληροφορίες κατάστασης ξεχωριστά.

Αντί να τοποθετήσουμε την κατάσταση μέσα στην υπηρεσία Ιστού (που καθιστά έτσι καταστατική, το οποίο θεωρείται γενικά ως κακή τεχνική) θα αποθηκεύσουμε την κατάσταση σε μια ξεχωριστή οντότητα που ονομάζεται **πόρος** (resource), ο οποίος θα αποθηκεύσει όλες τις πληροφορίες κατάστασης. Κάθε πόρος θα έχει ένα μοναδικό κλειδί, έτσι όποτε θέλουμε μια καταστατική αλληλεπίδραση με μια υπηρεσία Ιστού πρέπει απλά να καθοδηγήσουμε την υπηρεσία Ιστού ώστε να χρησιμοποιήσει έναν συγκεκριμένο πόρο.

Επιστρέφουμε στο παράδειγμα του αθροιστή. Όπως φαίνεται στο Σχήμα 13.4, η υπηρεσία Ιστού μας θα μπορούσε να έχει τρεις διαφορετικούς πόρους (A, B, C) για να επιλέξει. Εάν θέλουμε να αποθηκεύεται η ακέραια τιμή του αθροιστή από κλήση σε κλήση τότε ο πελάτης πρέπει απλά να προσδιορίζει την μέθοδο που θέλει να καλέσει με έναν συγκεκριμένο πόρο.

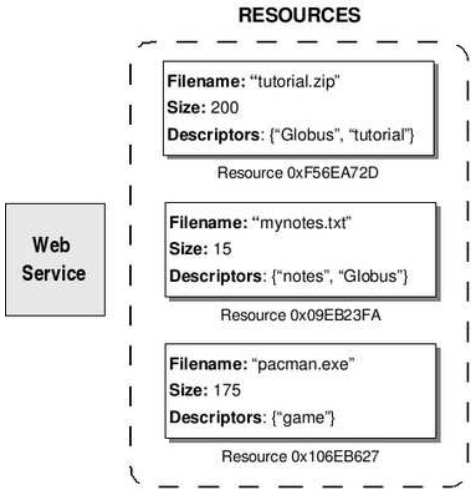


Σχήμα 13.4: Η προσέγγιση πόρων για την καταστατικότητα

Στο Σχήμα 13.4 μπορούμε να παρατηρήσουμε ότι ο πελάτης θέλει να καλέσει την λειτουργία `add` με τον πόρο C. Όταν η υπηρεσία Ιστού λάβει την αίτηση `add`, θα ανακτήσει τον πόρο C έτσι ώστε η λειτουργία `add` να εκτελεστεί πραγματικά σε εκείνο τον πόρο. Ο πόρος μπορεί να αποθηκευτεί στη μνήμη, στη δευτερεύουσα αποθήκευση, ή ακόμα και σε μια βάση δεδομένων. Επίσης, σημειώνουμε ότι μια υπηρεσία Ιστού μπορεί να έχει πρόσβαση σε περισσότερους από έναν πόρους.

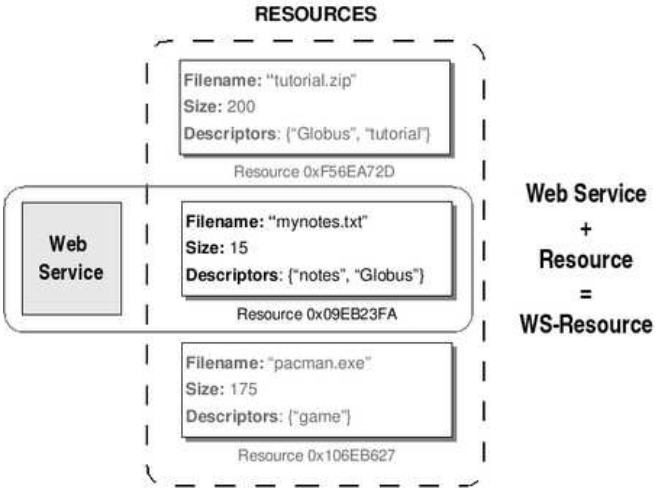
Φυσικά, οι πόροι μπορούν να είναι σε διαφορετικές μορφές και μεγέθη. Ένας πόρος μπορεί να αποθηκεύσει πολλές τιμές (όχι μόνο μια τιμή, όπως είδαμε στον προηγούμενο Σχήμα). Για παράδειγμα, οι πόροι μας θα μπορούσαν να αντιπροσωπεύσουν αρχεία όπως φαίνεται στο Σχήμα 13.5:

Προκύπτει ίσως το ερώτημα: Πως ακριβώς ο πελάτης προσδιορίζει ποιος πόρος πρέπει να χρησιμοποιηθεί; Υπάρχουν πραγματικά διαφορετικοί τρόποι για αυτό. Όπως θα δούμε αργότερα, ο καλύτερος τρόπος είναι να χρησιμοποιηθεί μια σχετικά νέα προδιαγραφή που ονομάζεται WS-Addressing η οποία παρέχει έναν πιο συμβατό τρόπο διευθυνσιοδότηση των υπηρεσιών Ιστού (σε σχέση με τα URIs).



Σχήμα 13.5: Μια υπηρεσία Ιστού με πολλούς πόρους. Κάθε πόρος αντιπροσωπεύει ένα αρχείο

Τέλος, ένα ζεύγος μιας υπηρεσίας Ιστού με έναν πόρο ονομάζεται WS-Resource όπως φαίνεται στο Σχήμα 13.6. Η διεύθυνση ενός συγκεκριμένου WS-Resource ονομάζεται αναφορά endpoint.



Σχήμα 13.6: WS-Resource

13.3.3 Η προδιαγραφή WSRF

Η WSRF είναι μια συλλογή πέντε διαφορετικών προδιαγραφών. Φυσικά, όλα αφορούν (με κάποιο τρόπο ή διαφορετικά) τη διαχείριση των WS-Resources. Οι προδιαγραφές αυτές είναι οι

εξής:

- WS-ResourceProperties: Ένας πόρος αποτελείται από μηδέν ή περισσότερες ιδιότητες πόρων. Οι ιδιότητες αυτές παριστάνουν την κατάσταση του WS-Resource. Για παράδειγμα, στο Σχήμα 13.5 που παρουσιάζαμε προηγουμένως κάθε πόρος έχει τρεις ιδιότητες των πόρων: Όνομα αρχείου, μέγεθος και περιγραφείς (ή αλλιώς λέξεις κλειδιά). WS-ResourceProperties προσδιορίζει πώς οι ιδιότητες πόρων ορίζονται και προσπελάζονται. Όπως θα δούμε αργότερα στον προγραμματισμό, οι ιδιότητες πόρων ορίζονται στην περιγραφή διεπαφών WSDL της υπηρεσίας Ιστού.

- WS-ResourceLifetime: Οι πόροι έχουν νον-τριιαλ κύκλους ζωής. Με άλλα λόγια, οι πόροι δεν είναι στατικές οντότητες οι οποίες δημιουργούνται όταν ξεκινά ο διακομιστής και καταστρέφονται όταν ο διακομιστής τερματίζεται. Οι πόροι δημιουργούνται και καταστρέφονται σε οποιαδήποτε στιγμή. Η WS-ResourceLifetime παρέχει μερικούς βασικούς μηχανισμούς για να διαχειριστεί τον κύκλο ζωής των πόρων.

- WS-ServiceGroup: Θα ενδιαφερθούμε συχνά για τη διαχείριση ομάδων υπηρεσιών Ιστού ή ομάδων WS-Resourses, και εκτελώντας λειτουργίες όπως «προσθήκη νέας υπηρεσίας στην ομάδα», «αφαίρεση μιας υπηρεσίας από την ομάδα» και (η πιο σημαντική) «αναζήτηση μιας υπηρεσίας στην ομάδα που ικανοποιεί την συνθήκη FOOBAR». Η WS-ServiceGroup προσδιορίζει πόσο ακριβώς πρέπει να ομαδοποιήσουμε τις υπηρεσίες ή τους WS-Resources μαζί. Μια WS-ServiceGroup είναι ένας WS-Resource που περιέχει μια συλλογή υπηρεσιών Ιστού. Οι ξεχωριστές υπηρεσίες μέσα στην WS-ServiceGroup είναι μέλη. Αν και η λειτουργικότητα που παρέχεται από αυτήν την προδιαγραφή είναι πολύ βασική, είναι εν τούτοις η βάση των ισχυρών υπηρεσιών αναζήτησης (όπως GT4 IndexService) οι οποίες μας επιτρέπουν να ομαδοποιήσουμε διαφορετικές υπηρεσίες μαζί και να προσπελάζονται σε αυτές μέσω ενός ενιαίου σημείου εισόδου (ή ομάδα υπηρεσιών).

- WS-BaseFaults: Η προδιαγραφή αυτή στοχεύει να παρέχει έναν τυποποιημένο τρόπο αναφοράς σφαλμάτων όταν συμβαίνει κάτι λάθος κατά τη διάρκεια μιας κλήσης WS-Service.

- WS-Notification: Η WS-Notification είναι μια άλλη συλλογή προδιαγραφών που παρόλο δεν είναι μέρος του WSRF, είναι στενά συνδεδεμένο. Η προδιαγραφή αυτή επιτρέπει σε μια

υπηρεσία Ιστού να διαμορφωθεί ως παραγωγός ανακοίνωσης και συγκεκριμένοι πελάτες να είναι καταναλωτές ανακοίνωσης (ή συνδρομητές). Αυτό σημαίνει ότι εάν μια αλλαγή εμφανίζεται στην υπηρεσία Ιστού (ή πιο συγκεκριμένα, σε ένας από τους WS-Resources), η αλλαγή αυτή ανακοινώνεται σε όλους τους συνδρομητές (δεν ανακοινώνονται όλες οι αλλαγές αλλά μόνο αυτές που ο προγραμματιστής υπηρεσιών Ιστού θέλει).

- WS-Addressing: Όπως αναφερθήκαμε πριν, η προδιαγραφή WS-Addressing παρέχει έναν μηχανισμό για την διευθυνσιοδότηση των υπηρεσιών Ιστού που είναι πιο συμβατό από τα URIs. Συγκεκριμένα, μπορούμε να χρησιμοποιήσουμε την WS-Addressing για να αναφερθούμε ένα ζευγάρι υπηρεσία Ιστού + πόρος (ένας WS-Resource).

Η WSRF υλοποιείται από το ανοιχτό λογισμικό Globus Toolkit έκδοση 4.

13.4 Μεθοδολογία Ανάπτυξης Υπηρεσιών Πλέγματος

Στην ενότητα αυτή θα περιγράψουμε τα βήματα που απαιτούνται για την ανάπτυξη υπηρεσιών πλέγματος σύμφωνα με την προδιαγραφή WSRF. Συνεπώς τα βήματα είναι τα εξής:

1. Πρώτα ορίζουμε την διεπαφή της υπηρεσίας με την χρήση της γλώσσας περιγραφής WS-DL.

2. Υλοποιούμε την υπηρεσία σε γλώσσα Java.

3. Ορίζουμε τις παραμέτρους ανάπτυξης με την χρήση του WSDD.

4. Μεταγλωττίζουμε όλα τα προηγούμενα αρχεία προκειμένουν να παραχθεί ένα αρχείο GAR. Αυτό γίνεται με την βοήθεια του εργαλείου Ant.

5. Εγκαθιστούμε την υπηρεσία με το εργαλείο Globus 4.

6. Δημιουργούμε ένα πρόγραμμα πελάτη ώστε να καλεί την υπηρεσία και το μεταγλωττίζουμε.

7. Ξεκινούμε το υποδοχέα Globus και εκτελούμε το πρόγραμμα πελάτη.

Για την καλύτερη κατανόηση της διαδικασίας ανάπτυξης μιας καταστατικής υπηρεσίας πλέγματος θα αναπτύξουμε μια απλή μαθηματική υπηρεσία. Η μαθηματική αυτή υπηρεσία που θα αναφέρεται ως `MathService` θα επιτρέψει στους πελάτες να εκτελέσουν κάποιες αριθμητικές πράξεις. Η υπηρεσία αυτή θα προσπελάζει ένα πόρο με δύο ιδιότητες:

- Μια ακέραια τιμή που θα λέγεται `a`.
- Μια τιμή αλφαριθμητικού που θα αποθηκεύει ένα αλφαριθμητικό το οποίο περιγράφει την τελευταία πράξη.

Επίσης, η μαθηματική αυτή υπηρεσία θα προσφέρει τρεις προσβάσιμες λειτουργίες στην τιμή `a`:

- την λειτουργία `add` η οποία προσθέτει μια τιμή στην ιδιότητα πόρου `a`.
- την λειτουργία `substract` η οποία αφαιρεί μια τιμή από την ιδιότητα πόρου `a`.
- την λειτουργία `getValueRP` η οποία επιστρέφει την τρέχουσα τιμή της ιδιότητας πόρου `a`.

1. Ορισμός διεπαφής της υπηρεσίας. Σε αυτό το βήμα δίνουμε τον ορισμό διεπαφής της μαθηματικής υπηρεσίας. Στον ορισμό διεπαφής περιλαμβάνει τις λειτουργίες που θα είναι διαθέσιμες στους πελάτες και επίσης σε κάθε λειτουργία δίνεται το όνομα της, τα ορίσματα της και το τύπο επιστροφής της, δηλαδή την υπογραφή της λειτουργίας. Ο ορισμός της διεπαφής γίνεται στην γλώσσα WSDL όπως στις υπηρεσίες Ιστού. Σαν αρχή θα δώσουμε τον ορισμό της διεπαφής σε γλώσσα Java που ίσως είναι περισσότερο φιλικό από ότι την άμεση κωδικοποίηση σε WSDL. Έτσι παρακάτω παρουσιάζεται η διεπαφή Java της μαθηματικής υπηρεσίας:

```
public interface Math
{
  public void add(int a);
  public void subtract(int a);
  public int getValueRP();
}
```

   Στην συνέχεια πρέπει να παραχθεί η περιγραφή WSDL της διεπαφής χρησιμοποιώντας το εργαλείο Java2WSDL όπως είδαμε στην περίπτωση των υπηρεσιών Ιστού. Όμως, το αρχείο WSDL για το Globus 4 πρέπει να συμπεριληφθούν πληροφορίες σχετικά με τις ιδιότητες των πόρων σύμφωνα με την προδιαγραφή WSRF. Για τους λόγους αυτούς

παρουσιάζουμε παρακάτω τα περιεχόμενα του αρχείου WSDL και εξετάζουμε την δομή του.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MathService"
    targetNamespace="http://www.globus.org/namespaces/examples/core/MathService_instance"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://www.globus.org/namespaces/examples/core/MathService_instance"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd"
    xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.wsdl
    xmlns:wsdlpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<wsdl:import
    namespace=
    "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"
    location="../../wsrf/properties/WS-ResourceProperties.wsdl" />


<!--============================================================

                    T Y P E S

  ============================================================-->
<types>
<xsd:schema targetNamespace="http://www.globus.org/namespaces/examples/core/MathService_instance"
    xmlns:tns="http://www.globus.org/namespaces/examples/core/MathService_instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">


        <!-- REQUESTS AND RESPONSES -->

        <xsd:element name="add" type="xsd:int"/>
        <xsd:element name="addResponse">
                <xsd:complexType/>
        </xsd:element>

        <xsd:element name="subtract" type="xsd:int"/>
        <xsd:element name="subtractResponse">
                <xsd:complexType/>
        </xsd:element>

        <xsd:element name="getValueRP">
                <xsd:complexType/>
        </xsd:element>
        <xsd:element name="getValueRPResponse" type="xsd:int"/>
```

```
        <!-- RESOURCE PROPERTIES -->

        <xsd:element name="Value" type="xsd:int"/>
        <xsd:element name="LastOp" type="xsd:string"/>

        <xsd:element name="MathResourceProperties">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element ref="tns:Value" minOccurs="1" maxOccurs="1"/>
                        <xsd:element ref="tns:LastOp" minOccurs="1" maxOccurs="1"/>
                </xsd:sequence>
        </xsd:complexType>
        </xsd:element>

</xsd:schema>
</types>


<!--============================================================

                        M E S S A G E S

  ============================================================-->
<message name="AddInputMessage">
        <part name="parameters" element="tns:add"/>
</message>
<message name="AddOutputMessage">
        <part name="parameters" element="tns:addResponse"/>
</message>

<message name="SubtractInputMessage">
        <part name="parameters" element="tns:subtract"/>
</message>
<message name="SubtractOutputMessage">
        <part name="parameters" element="tns:subtractResponse"/>
</message>

<message name="GetValueRPInputMessage">
        <part name="parameters" element="tns:getValueRP"/>
</message>
<message name="GetValueRPOutputMessage">
        <part name="parameters" element="tns:getValueRPResponse"/>
</message>
```

```
<!--============================================================

                    P O R T T Y P E

  ============================================================-->
<portType name="MathPortType"
    wsdlpp:extends="wsrpw:GetResourceProperty"
    wsrp:ResourceProperties="tns:MathResourceProperties">

        <operation name="add">
                <input message="tns:AddInputMessage"/>
                <output message="tns:AddOutputMessage"/>
        </operation>

        <operation name="subtract">
                <input message="tns:SubtractInputMessage"/>
                <output message="tns:SubtractOutputMessage"/>
        </operation>

        <operation name="getValueRP">
                <input message="tns:GetValueRPInputMessage"/>
                <output message="tns:GetValueRPOutputMessage"/>
        </operation>

</portType>

</definitions>
```

Η δομή του αρχείου WSDL αποτελείται από τέσσερα βασικά στοιχεία:

- Στην αρχή του αρχείου υπάρχει ένα στοιχείο που λέγεται <definitions>. Το στοιχείο περιέχει δύο ιδιότητες: Η ιδιότητα name και η ιδιότητα targetNamespace. Η ιδιότητα name θέτει το όνομα της υπηρεσίας που πρόκειται να υλοποιήσουμε δηλαδή το όνομα του αρχείου WSDL. Η ιδιότητα targetNamespace προσδιορίζει το namespace του αρχείου WSDL. Αυτό σημαίνει ότι όλοι οι PortTypes και λειτουργίες που ορίζονται μέσα στο αρχείο WSDL θα ανήκουν σε αυτό το namespace.
- Έπειτα ακολουθεί το στοιχείο <types> που μέσα σε αυτό ορίζονται οι υπογραφές των λειτουργιών (δηλαδή, το όνομα της λειτουργίας, το τύπο δεδομένων του ορίσματος και το τύπο επιστροφής) καθώς και τις ιδιότητες των πόρων. Οι ιδιότητες των

πόρων χρησιμοποιούνται για να αποθηκεύουν πληροφορίες κατάστασης. Η δήλωση υπογραφών των λειτουργιών και των ιδιοτήτων πόρων γίνονται με την βοήθεια του σχήματος XML Schema.

- Στην συνέχεια ορίζουμε στοιχεία <message> όπου δηλώνουμε μηνύματα εισόδου και εξόδου για κάθε λειτουργία της μαθηματικής υπηρεσίας.
- Τέλος, υπάρχει το στοιχείο <PortType> που ορίζει την διεπαφή της μαθηματικής υπηρεσίας. Μέσα στο στοιχείο <PortType> ορίζονται στοιχεία <operation> για κάθε λειτουργία ή μέθοδο που παρέχεται η μαθηματική υπηρεσία. Σε κάθε στοιχείο <operation> έχει δυο υποστοιχεία <input> και <output> τα οποία αντιστοιχούν στα μηνύματα εισόδου και εξόδου που ορίστηκαν προηγουμένως στο στοιχείο <message>.

Επίσης, σημειώνουμε ότι το αρχείο WSDL για το Globus 4 δεν υπάρχουν στοιχεία σύνδεσης όπως υπήρχαν σε αντίστοιχο αρχείο WSDL στις υπηρεσίες Ιστού. Ο λόγος είναι ότι τα στοιχεία αυτά είναι ειδικά και δημιουργούνται αυτόματα από το εργαλείο Globus 4 όταν κατασκευαστεί η υπηρεσία.

2. Υλοποίηση της υπηρεσίας σε Java. Το επόμενο βήμα μετά τον ορισμό της διεπαφής υπηρεσίας είναι η υλοποίηση της υπηρεσίας δηλαδή ορίζουμε το κώδικα για κάθε λειτουργία της μαθηματικής υπηρεσίας. Γνωρίζουμε ότι η υπηρεσία και οι πόροι είναι δύο ξεχωριστές οντότητες. Όμως σε αυτό το απλό παράδειγμα γράφουμε το κώδικα για τις λειτουργίες της υπηρεσίας και τις ιδιότητες των πόρων μέσα στην ίδια κλάση Java. Παρακάτω παρουσιάζεται ο κώδικας για την υπηρεσία και τις ιδιότητες πόρων:

```
package org.globus.examples.services.core.first.impl;

import java.rmi.RemoteException;

import org.globus.wsrf.Resource;
import org.globus.wsrf.ResourceProperties;
import org.globus.wsrf.ResourceProperty;
import org.globus.wsrf.ResourcePropertySet;
import org.globus.wsrf.impl.ReflectionResourceProperty;
import org.globus.wsrf.impl.SimpleResourcePropertySet;
import org.globus.examples.stubs.MathService_instance.AddResponse;
import org.globus.examples.stubs.MathService_instance.SubtractResponse;
import org.globus.examples.stubs.MathService_instance.GetValueRP;
```

```
public class MathService implements Resource, ResourceProperties {

        /* Resource Property set */
        private ResourcePropertySet propSet;

        /* Resource properties */
        private int value;
        private String lastOp;

        /* Constructor. Initializes RPs */
        public MathService() throws RemoteException {
                /* Create RP set */
                this.propSet = new SimpleResourcePropertySet(
                                MathQNames.RESOURCE_PROPERTIES);

                /* Initialize the RP's */
                try {
                        ResourceProperty valueRP = new ReflectionResourceProperty(
                                        MathQNames.RP_VALUE, "Value", this);
                        this.propSet.add(valueRP);
                        setValue(0);

                        ResourceProperty lastOpRP = new ReflectionResourceProperty(
                                        MathQNames.RP_LASTOP, "LastOp", this);
                        this.propSet.add(lastOpRP);
                        setLastOp("NONE");
                } catch (Exception e) {
                        throw new RuntimeException(e.getMessage());
                }
        }

        /* Get/Setters for the RPs */
        public int getValue() {
                return value;
        }

        public void setValue(int value) {
                this.value = value;
        }

        public String getLastOp() {
                return lastOp;
        }

        public void setLastOp(String lastOp) {
                this.lastOp = lastOp;
```

```
        }

        /* Remotely-accessible operations */

        public AddResponse add(int a) throws RemoteException {
                value += a;
                lastOp = "ADDITION";

                return new AddResponse();
        }

        public SubtractResponse subtract(int a) throws RemoteException {
                value -= a;
                lastOp = "SUBTRACTION";

                return new SubtractResponse();
        }

        public int getValueRP(GetValueRP params) throws RemoteException {
                return value;
        }

        /* Required by interface ResourceProperties */
        public ResourcePropertySet getResourcePropertySet() {
                return this.propSet;
        }
}
```

Πρέπει να σημειώσουμε ότι η κλάση `MathService` υλοποιεί τις κλάσεις `Resource` και `ResourceProperties`. Οι κλάσεις αυτές είναι μέρος του πακέτου `org.globus.wsrf` που παρέχει το Globus 4. Στην συνέχεια δηλώνουμε τις δυο ιδιότητες πόρων για την μαθηματική υπηρεσία, ορίζουμε το κατασκευαστή `MathService` ώστε να αρχικοποιεί τις ιδιότητες πόρων, έπειτα ορίζουμε μεθόδους set/get για την πρόσβαση ιδιότητα κάθε πόρου και τέλος ορίζουμε το απλό κώδικα για κάθε λειτουργία που υποστηρίζει η μαθηματική υπηρεσία.

3. Διαμόρφωση εγκατάστασης υπηρεσίας σε WSDD. Σε αυτό το βήμα τοποθετούμε όλα εκείνα τα αρχεία που έχουμε γράψει στο υποδοχέα υπηρεσιών Ιστού ώστε η υπηρεσία να είναι διαθέσιμη στους πελάτες. Το βήμα αυτό ονομάζεται εγκατάσταση της υπηρεσίας Ιστού. Ένα από τα βασικά συστατικά της φάσης εγκατάστασης είναι ένα αρχείο που λέγεται περιγραφέας εγκατάστασης. Το αρχείο αυτό προσδιορίζει στον υποδοχέα υπηρεσιών Ιστού που πρέπει να δημοσιευτεί η υπηρεσία Ιστού (δηλαδή, προσδιορίζουμε το URI

της υπηρεσίας Ιστού). Ο περιγραφέας εγκατάστασης είναι γραμμένο σε μορφή WSDD και παρουσιάζεται παρακάτω το περιεχόμενο του:

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultServerConfig"
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <service name="examples/core/first/MathService" provider="Handler" use="literal" style="document"
        <parameter name="className" value="org.globus.examples.services.core.first.impl.MathService"/
        <wsdlFile>share/schema/examples/MathService_instance/Math_service.wsdl</wsdlFile>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
        <parameter name="scope" value="Application"/>
        <parameter name="providers" value="GetRPProvider"/>
        <parameter name="loadOnStartup" value="true"/>
    </service>

</deployment>
```

Από το παραπάνω αρχείο παρατηρούμε ότι το στοιχείο `service name` προσδιορίζει την θέση της υπηρεσίας Ιστού που μπορεί να βρεθεί. Αν συνδυάσουμε αυτό το στοιχείο με την βασική διεύθυνση του υποδοχέα υπηρεσιών Ιστού, παίρνουμε την πλήρη διεύθυνση URI της μαθηματικής μας υπηρεσίας η οποία είναι η εξής: `http://localhost:8080/wsrf/services/examples/core/first/MathS` Τέλος, τα στοιχεία `className` και `<wsdlFile>` δείχνουν στον υποδοχέα υπηρεσιών Ιστού τη θέση της κλάσης και περιγραφής WSDL για την μαθηματική υπηρεσία.

4. Δημιουργία αρχείου GAR με το εργαλείο Ant. Σε αυτό το βήμα χρησιμοποιούμε τα τρία αρχεία που δημιουργήσαμε στα προηγούμενα βήματα (δηλαδή το αρχείο WSDL, το αρχείο με την υλοποίηση της υπηρεσίας και το αρχείο WSDD) ώστε να δημιουργήσουμε ένα νέο αρχείο που λέγεται GAR (Grid Archive). Το αρχείο GAR είναι ένα ενιαίο αρχείο το οποίο περιέχει όλα τα αρχεία και πληροφορίες που πραγματικά χρειάζεται ο υποδοχέας υπηρεσιών Ιστού ώστε να εγκαταστήσει την μαθηματική υπηρεσία και να είναι διαθέσιμη στους πελάτες. Για την δημιουργία του αρχείου GAR πρέπει να λάβουν μια σειρά από βήματα όπως:

   - Επεξεργασία του αρχείου WSDL ώστε να προσθέσει ορισμένα κομμάτια όπως οι συνδέσεις.

- Δημιουργία κλάσεων πληρεξούσιων ή σκελετών από το αρχείο WSDL.
- Μεταγλώττιση των κλάσεων πληρεξούσιων/σκελετών.
- Μεταγλώττιση της υλοποίησης υπηρεσίας.
- Οργάνωση όλων των παραγόμενων αρχείων σε μια συγκεκριμένη δομή καταλόγου.

Όμως τα παραπάνω βήματα αυτοματοποιούνται και παράγεται το αρχείο GAR σε ένα μόνο βήμα με την βοήθεια του εργαλείου Ant. Έτσι, το εργαλείο αυτό είναι παρόμοιο με την εντολή `make` του UNIX. Παρόλα αυτά εμείς θα χρησιμοποιούμε ένα πιο ευέλικτο εργαλείο που βασίζεται στο σενάριο `globus-build-service` το οποίο θα παράγει ένα αρχείο GAR με ελάχιστη προσπάθεια χωρίς καθε φορά να τροποιούμε το αρχείο του Ant. Το αρχείο `globus-build-service` συνοδεύεται με τα υπόλοιπα αρχεία του παρόντος παραδείγματος ή ακόμα μπορεί να βρεθεί στην διεύθυνση `http://gsbt.sourceforge.net/`. Για να παράγουμε το αρχείο GAR εισάγουμε την παρακάτω εντολή:

```
.globus-build-service.sh -d <service base directory> -s <service WSDL file>
```

Ο κατάλογος `service base directory` είναι ο κατάλογος που τοποθετήσαμε το αρχείο `deploy-server.wsdd` και που βρίσκονται τα αρχεία java. Έτσι, στο παράδειγμα μας θα εισάγουμε την παρακάτω εντολή:

```
./globus-build-service.sh -d org/globus/examples/services/core/first/  -s schema/examples/MathService_instance/Math.wsdl
```

5. Εγκατάσταση υπηρεσίας στον υποδοχέα υπηρεσιών Ιστού. Είναι γνωστό ότι το αρχείο GAR περιέχει όλα τα αρχεία και πληροφορίες που χρειάζεται ο διακομιστής Ιστού για να εγκαταστήσει την μαθηματική υπηρεσία. Η εγκατάσταση της υπηρεσίας γίνεται με ένα εργαλείο του Globus 4 και χρησιμοποιώντας το Ant για αποσυμπίεση του αρχείου GAR και αντιγραφή των αρχείων (δηλαδή, WSDL, μεταγλωττισμένους κορμούς, μεταγλωττισμένη υλοποίηση, WSDD) μέσα στις βασικές θέσεις του καταλόγου Globus 4. Το εργαλείο Globus που χρησιμοποιείται για την εγκατάσταση ονομάζεται `globus-deploy-gar`. Για να εκτελεστεί η εντολή αυτή θα πρέπει στον προσωπικό κατάλογο του χρήστη να έχει δικαιώματα ανάγνωσης. Συνεπώς, για την εγκατάσταση της μαθηματικής υπηρεσίας εισάγουμε την παρακάτω εντολή:

```
globus-deploy-gar /org_globus_examples_services_core_first.gar
```

6. Δημιουργία του προγράμματος πελάτη και μεταγλώττιση. Σε αυτό το βήμα αναπτύσσουμε το πρόγραμμα πελάτη το οποίο θα επικοινωνεί με την μαθηματική υπηρεσία δηλαδή θα καλεί τις λειτουργίες `add`, `substract` και την `getValueRP`. Παρακάτω παρουσιάζεται το πρόγραμμα πελάτη:

```
package org.globus.examples.clients.MathService_instance;

import org.apache.axis.message.addressing.Address;
import org.apache.axis.message.addressing.EndpointReferenceType;

import org.globus.examples.stubs.MathService_instance.MathPortType;
import org.globus.examples.stubs.MathService_instance.GetValueRP;
import org.globus.examples.stubs.MathService_instance.service.MathServiceAddressingLocator;

public class Client {

        public static void main(String[] args) {
                MathServiceAddressingLocator locator = new MathServiceAddressingLocator();

                try {
                        String serviceURI=args[0];

                        EndpointReferenceType endpoint = new EndpointReferenceType();
                        endpoint.setAddress(new Address(serviceURI));

                        MathPortType math = locator.getMathPortTypePort(endpoint);

                        // Perform an addition
                        math.add(10);

                        // Perform another addition
                        math.add(5);

                        // Access value
                        System.out.println("Current value:" + math.getValue(new GetValueRP()));

                        // Perform a subtraction
                        math.subtract(5);

                        // Access value
                        System.out.println("Current value:" + math.getValue(new GetValueRP()));
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
```

```
}
```

Πρέπει να σημειώσουμε εδώ ότι όταν εκτελείται το πρόγραμμα πελάτη από την γραμμή εντολών πρέπει να περάσουμε μια παράμετρος. Η παράμετρος αυτή θα είναι η GSH η οποία είναι μια URL που προσδιορίζει που βρίσκεται η υπηρεσία. Η θέση της υπηρεσίας είναι σχετική ως προς το namespace της `MathService` που εγκαταστάθηκε. Στην συνέχεια μεταγλωττίζουμε το πρόγραμμα πελάτη με την παρακάτω εντολή:

```
javac -classpath ./build/stubs/classes/:$CLASSPATH
org/globus/examples/clients/MathService_instance/Client.java
```

7. Εκκίνηση του υποδοχέα και εκτέλεση του προγράμματος πελάτη. Πριν εκτελέσουμε το πρόγραμμα πελάτη, πρέπει πρώτα να ξεκινήσουμε το υποδοχέα Globus. Ο υποδοχέας Globus περιέχει όλες τις υπηρεσίες που έχουν εγκατασταθεί. Για να εκκινήσουμε το υποδοχέα χρησιμοποιούμε την εντολή `globus-start-container`. Έτσι για να ξεκινήσουμε τον υποδοχέα εισάγουμε την παρακάτω εντολή:

```
globus-start-container -nosec
```

Μόλις ξεκινήσει ο υποδοχέας θα εμφανιστεί μια λίστα από URLs των εγκαταστημένων υπηρεσιών. Αν εγκατασταθεί η μαθηματική υπηρεσία επιτυχώς μπορούμε να εκτελέσουμε το πρόγραμμα πελάτη περνώντας ως παράμετρος την GSH της υπηρεσίας. Για την εκτέλεση του προγράμματος πελάτη εισάγουμε την παρακάτω εντολή:

```
java -classpath ./build/stubs/classes/:$CLASSPATH
org.globus.examples.clients.MathService_instance.Client
http://localhost:8080/wsrf/services/examples/core/first/MathService
```

Μετά την εκτέλεση του προγράμματος πελάτη `Client` παίρνουμε την παρακάτω έξοδο:

```
Current value:  15
Current value:  10
```

Κεφάλαιο 14

Προγραμματισμός JXTA

14.1 Εισαγωγή

Το ενδιάμεσο λογισμικό JXTA είναι ένα σύνολο ανοιχτών και γενικευμένων πρωτοκόλλων ομότιμων που επιτρέπουν σε οποιαδήποτε συσκευή (από προσωπικό υπολογιστή μέχρι διακομιστή, από τηλέφωνο μέχρι υπολογιστή παλάμης) στο δίκτυο να επικοινωνεί και να συνεργαστεί. Το όνομα JXTA προέρχεται από τη λέξη Juxtapose που σε ελεύθερη μετάφραση σημαίνει ''Δίπλα - Δίπλα''. Ο χαρακτηρισμός αυτός αναφέρεται στη δυνατότητα που προσφέρει η πλατφόρμα αυτή να συνυπάρχουν ''Δίπλα - Δίπλα'' σε δίκτυο όλα τα είδη κόμβων, είτε πρόκειται για απλούς προσωπικούς υπολογιστές, είτε για διακομιστές ειδικού σκοπού, φορητά, κινητά τηλέφωνα ή ηλεκτρονικές συσκευές παλάμης. Το JXTA είναι μια ανοιχτή πλατφόρμα δικτύου σχεδιασμένη για την ανάπτυξη ομότιμων εφαρμογών.

Σκοπός της τεχνολογίας JXTA είναι η ανάπτυξη και προτυποποίηση βασικών υπηρεσιών που θα επιτρέπουν στους προγραμματιστές την ανάπτυξη διαλειτουργικών ομότιμων υπηρεσιών και εφαρμογών. Το JXTA παρέχει ένα σύνολο από ανοιχτά πρωτόκολλα και μια υλοποίηση ανοιχτού λογισμικού για την ανάπτυξη ομότιμων εφαρμογών.

Εξετάζοντας την πλατφόρμα JXTA πιο αφαιρετικά θα λέγαμε ότι πρόκειται για μια πολύ αξιόλογη προσπάθεια για να ξεπεραστούν βασικές αδυναμίες των μέχρι τώρα υπαρχόντων ομότιμων συστημάτων. Αδυναμίες που έχουν να κάνουν με τη διαλειτουργικότητα, την δυνατότητα δηλαδή, να παρέχεται ένα σύνολο υπηρεσιών για ανεύρεση κόμβων που βρίσκονται στη ίδια ομάδα και τη δυνατότητα επικοινωνίας μεταξύ τους και την ανεξαρτησία όσων αφορά την πλατ-

φόρμα συστήματος. Να μην εξαρτάται από πλατφόρμες λειτουργικών συστημάτων, γλώσσες προγραμματισμού και πρωτόκολλα μεταφοράς. Και το τελευταίο και πιο σημαντικό, να είναι προσβάσιμη από κάθε είδους συσκευή η οποία να μπορεί να συνδεθεί στο δίκτυο.

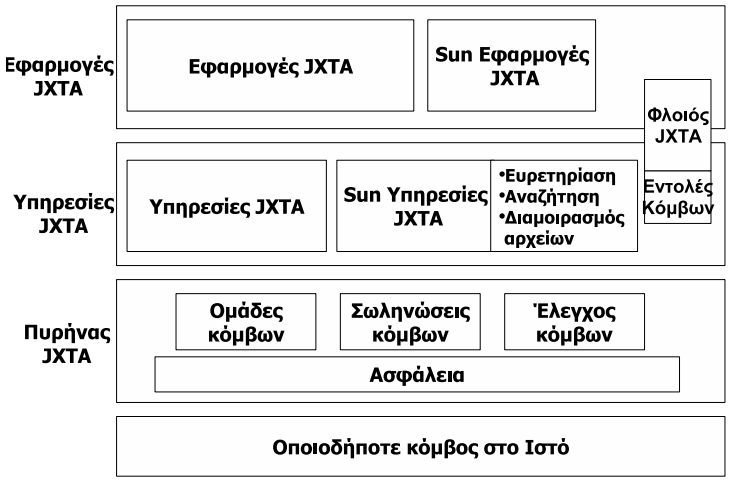
14.2 Αρχιτεκτονική JXTA

Τα πρωτόκολλα του JXTA προτυποποιούν τον τρόπο με τον οποίο οι κόμβοι:

- Ανακαλύπτουν ο ένας τον άλλο.
- Αυτο-οργανώνονται σε ομάδες.
- Δημοσιεύουν και ανακαλύπτουν δικτυακές υπηρεσίες.
- Επικοινωνούν μεταξύ τους.
- Παρακολουθούν και ελέγχουν ο ένας τον άλλο.

Τα πρωτόκολλα του JXTA δεν απαιτούν την χρήση μιας συγκεκριμένης γλώσσας προγραμματισμού ή λειτουργικού συστήματος ή πρωτοκόλλου μεταφοράς ή μοντέλου ασφάλειας. Συνεπώς, τα πρωτόκολλα JXTAεπιτρέπουν στις ετερογενείς συσκευές με διαφορετικές στοίβες λογισμικού να διαλειτουργούν. Στο Σχήμα 14.1 παρουσιάζεται η αρχιτεκτονική του λογισμικού της πλατφόρμας JXTA χωρισμένη σε τρία επίπεδα:

1. Επίπεδο πλατφόρμας: Το χαμηλότερο αυτό επίπεδο το οποίο ονομάζεται αλλιώς και **πυρήνας** (Core) περιλαμβάνει βασικά συσταστικά τα οποία είναι κοινά στα ομότιμα συστήματα. Τα συσταστικά αυτά έχουν να κάνουν με την δημιουργία κόμβων και ομάδων, διαχείριση επικοινωνίας (όπως σωληνώσεις, ανακάλυψη κόμβων) καθώς και ζητήματα ασφάλειας.
2. Επίπεδο υπηρεσιών: Ψπάρχει το επίπεδο υπηρεσιών που περιλαμβάνει υπηρεσίες σχετικές με την ευρετηρίαση, την αναζήτηση και τον διαμοιρασμό αρχείων.
3. Επίπεδο εφαρμογών: Στο υψηλότερο επίπεδο υπάρχει το επίπεδο εφαρμογών όπως διαμοιρασμό αρχείων, συστήματα αποθήκευσης και δημοπρασίας.



Σχήμα 14.1: Αρχιτεκτονική λογισμικού JXTA

Συνοπτικά, τα πρωτόκολλα JXTA είναι τα εξής: το **πρωτόκολλο επιλυτής κόμβου** (Peer Resolver Protocol - PRP) είναι ένας μηχανισμός με το οποίο ένας ομότιμος στέλνει μια ερώτηση σε ένα ή περισσότερους ομότιμους κόμβους και λαμβάνει μια απάντηση στην ερώτηση. Το **πρωτόκολλο ανακάλυψης κόμβου** (Peer Discovery Protocol - PDP) είναι ένας μηχανισμός με το οποίο ένας ομότιμος κόμβος δημοσιεύσει τους δικούς τους πόρους και αποκαλύπτει πόρους από άλλους ομότιμους (ομάδες ομότιμων, υπηρεσίες, σωληνώσεις και επιπλέον ομότιμους). Το **πρωτόκολλο πληροφοριών κόμβου** (Peer Information Protocol - PIP) είναι ένας μηχανισμός με τον οποίο ένας ομότιμος μπορεί να αποκτήσει πληροφορίες κατάστασης σχετικά με τους άλλους ομότιμους κόμβους, όπως κατάσταση, χρόνος, φορτίο κίνησης και άλλα χαρακτηριστικά. Το **πρωτόκολλο σύνδεσης σωλήνωσης** (Pipe Binding Protocol - PBP) χρησιμοποιείται για να συνδέσει σωληνώσεις ανάμεσα στους ομότιμους κόμβους. Το **πρωτόκολλο δρομολόγησης κατάληξης** (Endpoint Routing Protocol - ERP) χρησιμοποιείται για να δρομολογήσει μηνύματα JXTA. Τέλος, το **πρωτόκολλο ραντεβού** (Rendezvous Protocol - RVP) είναι ένας μηχανισμός με το οποίο οι ομότιμοι κόμβοι μπορούν να εγγραφούν ως συνδρομητές σε μια υπηρεσία εκπομπής.

14.2.1 Κόμβοι JXTA

Ένας **κόμβος** (peer) είναι μια οποιαδήποτε δικτυακή συσκευή που υλοποιεί ένα ή περισσότερα πρωτόκολλα του JXTA. Κάθε κόμβος λειτουργεί ανεξάρτητα και ασύγχρονα από όλους τους

άλλους κόμβους. Είναι γνωστός στους υπόλοιπους από ένα μοναδικό αναγνωριστικό του, το Peer ID. Οι κόμβοι για την μεταξύ τους σημειακή επικοινωνία δημοσιεύουν στο δίκτυο διεπαφές ως σημεία κατάληξης. Εκτός από την απευθείας επικοινωνία που μπορεί να συνάψουν μπορεί να χρησιμοποιηθούν και άλλοι κόμβοι με αυξημένες δυνατότητες για να συνδράμουν στην επικοινωνία μεταξύ κόμβων σε διαφορετικά δίκτυα ή πίσω από συστήματα ασφαλείας, για παράδειγμα τείχους προστασίας.

14.2.2 Αναγνωριστικά

Το JXTA χρησιμοποιεί **αναγνωριστικό** (identifier) 128-bit για να αναφερθεί σε μια οντότητα όπως ένα κόμβο, μια υπηρεσία, μια διαφήμιση κλπ. Έτσι είναι σχετικά εύκολο για κάθε οντότητα να έχει ένα μοναδικό αναγνωριστικό μέσα σε ένα τοπικό περιβάλλον εκτέλεσης αλλά δύσκολο να υπάρχει μοναδικό αναγνωριστικό σε μια ολόκληρη κοινότητα η οποία αποτελείται από εκατομμύρια κόμβους. Συνεπώς, το JXTA λύνει το πρόβλημα αυτό συνδέοντας το αναγνωριστικό με επιπλέον πληροφορίες όπως το όνομα και μια διεύθυνση δικτύου.

14.2.3 Διαφημίσεις

Μια **διαφήμιση** (advertisement) είναι μια δομή μεταδεδομένων που αναπαρίστανται ως ένα δομημένο έγγραφο XML το οποίο ονομάζει, περιγράφει και δημοσιεύει την ύπαρξη ενός πόρου όπως ένας ομότιμος κόμβος, μια ομάδα ομότιμων, μια σωλήνωση ή μια υπηρεσία. Η τεχνολογία JXTA ορίζει ένα σύνολο βασικών τύπων διαφημίσεων αλλά μπορούν να δημιουργηθούν ακόμα περισσότερους τύπους διαφημίσεων από τους βασικούς τύπους χρησιμοποιώντας το σχήμα XML schema.

14.2.4 Μηνύματα

Ένα **μήνυμα** (message) είναι ένα αντικείμενο που στέλνεται ανάμεσα στους κόμβους JXTA και αναπαριστάνεται είτε σε μορφή XML είτε σε δυαδική μορφή. Έτσι, πρόκειται για τη βασική μονάδα ανταλλαγής δεδομένων μεταξύ των ομότιμων κόμβων. Τα μηνύματα στέλνονται και λαμβάνονται από την υπηρεσία σωλήνωσης και από την υπηρεσία σημείου κατάληξης. Μια εφαρμογή χρησιμοποιεί την υπηρεσία σωλήνωσης για να δημιουργήσει, να στείλει και να λάβει μηνύματα.

Το μήνυμα είναι μια διατεταγμένη ακολουθία από ονόματα και περιεχόμενα που ονομάζονται στοιχεία του μηνύματος. Έτσι το μήνυμα δεν είναι τίποτα άλλο από ένα σύνολο από ζεύγη ονόματος - τιμής. Το περιεχόμενο μπορεί να είναι αφηρημένου ή αυθαίρετου τύπου. Τα πρωτόκολλα της τεχνολογίας JXTA καθορίζονται σαν ένα σύνολο από μηνύματα που ανταλλάσονται μεταξύ των ομότιμων κόμβων.

14.3 Δίκτυο Επικάλυψης JXTA

Η τεχνολογία JXTA αποτελείται από ένα σύνολο ομότιμων κόμβων που αλληλεπιδρούν και οργανώνονται με διάφορους τρόπους. Η οργάνωση των κόμβων JXTA είναι ανεξάρτητη από τις υπάρχουσες φυσικές συσκευές και συνδεσιμότητα. Έτσι, μπορούμε να πούμε ότι οι κόμβοι JXTA οργανώνονται σε ένα **εικονικό δίκτυο επικάλυψης** (virtual network overlay) το οποίο ''κάθεται'' πάνω από τις φυσικές συσκευές όπως φαίνεται στο Σχήμα. Οι κόμβοι δεν απαιτούνται να έχουν άμεσες και απευθείες (ή σημειακές) συνδέσεις μεταξύ τους. Επίσης, οι κόμβοι μπορούν να εντοπίσουν άλλους στο δίκτυο προκειμένουν να σχηματιστούν παροδικές ή επίμονες σχέσεις οι οποίες λέγονται ομάδες κόμβων. Οι παραπάνω έννοιες θα αναλυθούν παρακάτω.

14.3.1 Ομάδες Κόμβων

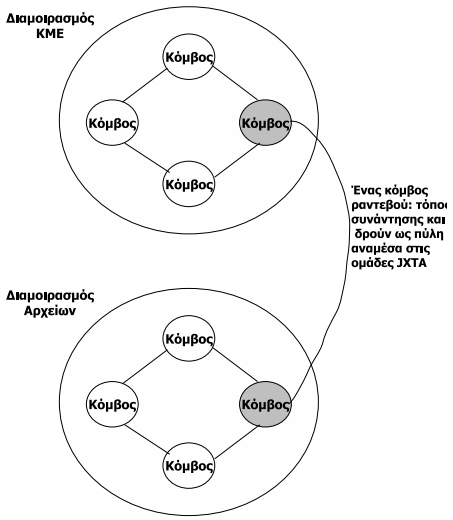
Η **ομάδα κόμβων** (peer group) είναι μια συλλογή από κόμβους η οποία παρέχει ένα σύνολο υπηρεσιών για παράδειγμα, θα μπορούσαμε να έχουμε μια ομάδα κόμβων διαμοιρασμού αρχείων ή μια ομάδα κόμβων διαμοιρασμού κύκλων ΚΜΕ. Συνεπώς, οι ομάδες κόμβων σχηματίζονται και αυτοργανώνονται με βάση τα αμοιβαία ενδιαφέροντα των κόμβων και κάθε ομάδα ορίζει ένα περιβάλλον εμβέλειας. Τα όρια της ομάδας κόμβων ορίζουν την εμβέλεια αναζήτησης όταν αναζητούμε τα περιεχόμενα μιας ομάδας.

Επίσης, οι ομάδες κόμβων μπορούν να χρησιμοποιηθούν για να δημιουργήσουμε ένα περιβάλλον παρακολούθησης, δηλαδή παρακολούθηση ενός συνόλου κόμβων για ένα ειδικό σκοπό. Οι ομάδες κόμβων μπορούν να προστατευθούν με κωδικό πρόσβασης και να υλοποιήσουν τοπικές πολιτικές ασφάλειας αν επιθυμούμε ασφαλείς ομάδες. Τέλος, υπάρχει μια ειδική ομάδα η οποία λέγεται World Peer Group (η εξ' ορισμού ομάδα κόμβων) που περιλαμβάνει όλους τους κόμβους

JXTA.

14.3.2 Κόμβους Ραντεβού

Για να αντισταθμίσει την έλλειψη μιας κεντρικής υπηρεσίας (όπως ένας διακομιστής ονομάτων περιοχής), το δίκτυο JXTA χρησιμοποιεί **κόμβους ραντεβού** (rendezvous nodes). Οι κόμβοι ραντεβού είναι εθελοντές που έχουν συμφωνήσει για να δρούν ως σημείο συνάντησης για άλλους κόμβους. Οι κόμβοι ραντεβού συχνά διατηρούν μια μόνιμη (συνδεδεμένη) διεύθυνση IP, έτσι ώστε οι άλλοι κόμβοι να μπορούν να έρθουν σε επαφή με τους κόμβους ραντεβού ώστε να ελέγξουν τις τρέχουσες συνδέσεις των δυναμικά (συνδεδεμένων) κόμβων σημείων κατάληξης. Οι κόμβοι ραντεβού μπορούν επίσης να αποθηκεύσουν μια εγγραφή των άλλων κόμβων ραντεβού. Συνεπώς, εάν γνωρίζετε ένα σημείο ραντεβού και ο φίλος σας επίσης γνωρίζει ένα σημείο ραντεβού, και τα δύο σημεία ραντεβού γνωρίζουν ο ένας στο άλλο (άμεσα ή μέσω άλλων κόμβων ραντεβού), εσείς και ο φίλος σας μπορείτε να βρείτε και να συναντηθείτε, όπως φαίνεται στο Σχήμα 14.2.



Σχήμα 14.2: Η σχέση ανάμεσα στις ομάδες κόμβων JXTA μέσω των κόμβων ραντεβού

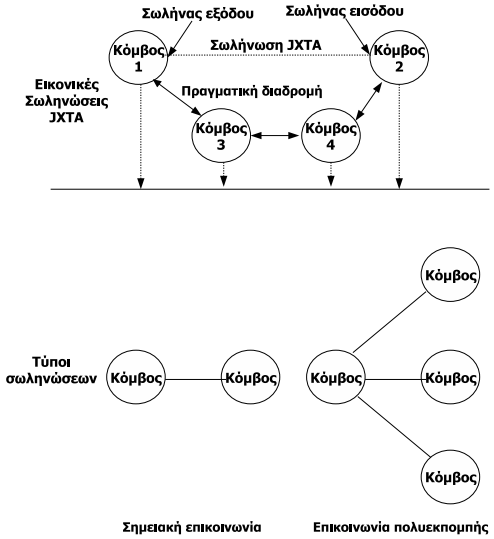
Η υλοποίηση του πρωτοκόλλου ραντεβού JXTA ελέγχει αυτό. Οι εφαρμογές δεν ειναι απαραίτητο να γνωρίζουν για αυτές τις λεπτομέρειες χαμηλού επιπέδου. Οι κόμβοι στέλνουν μηνύματα αναζήτησης στους κόμβους ραντεβού που γνωρίζουν και το δίκτυο JXTA θα κάνει τις απαραίτητες ερωτήσεις.

14.3.3 Σωληνώσεις

Οι κόμβοι JXTA χρησιμοποιούν **σωληνώσεις** (pipes) για να στείλουν μηνύματα μεταξύ τους οι κόμβοι. Οι σωληνώσεις είναι ασύγχρονοι και μονής κατεύθυνσης μηχανισμούς μεταφοράς μηνυμάτων που χρησιμοποιούνται για επικοινωνία. Οι σωληνώσεις υποστηρίζουν την μεταφορά ενός οποιοδήποτε αντικειμένου περιλαμβάνοντας δυαδικός κώδικας, αλφαριθμητικά δεδομένων και αντικείμενα τεχνολογίας Java.

Τα σημεία κατάληξης της σωλήνωσης αναφέρονται ως **σωλήνας εισόδου** (input pipe) και **σωλήνας εξόδου** (output pipe). Τα μηνύματα ρέουν από το σωλήνα εξόδου στους σωλήνες εισόδου.

Οι σωληνώσεις είναι εικονικά κανάλια επικοινωνίας και μπορούν να συνδεσούν κόμβους που δεν έχουν άμεση φυσική σύνδεση. Στο παράδειγμα που δίνεται εδώ όπως φαίνεται στο Σχήμα 14.3, ο κόμβος 1 έχει δημιουργήσει μια εικονική σωλήνωση JXTA μεταξύ του ίδιου και του κόμβου 2. Η πραγματική φυσική διαδρομή όμως περνάει μέσω μιας αντιπυρικής ζώνης, μέσω ενός υπολογιστή γραφείου (κόμβος 3) και ενός διακομιστή (κόμβος 4). Ο αριθμός των ενδιάμεσων κόμβων που χρησιμοποιούνται για να επικοινωνήσουν ένα μήνυμα είναι γνωστός ως αριθμός βημάτων (hops).



Σχήμα 14.3: Το πάνω μέρος του Σχήματος φαίνεται η διάχιση μέσω πολλαπλών βημάτων πριν το μήνυμα φτάσει στο παραλήπτη και στο κάτω μέρος φαίνονται οι δύο τύποι σωληνώσεων, σημειακή και πολυεκπομπή

Επίσης, σε κάθε βήμα του δικτύου μπορεί να χρησιμοποιηθεί ένα διαφορετικό πρωτόκολλο μεταφοράς δεδομένου ότι οι σωληνώσεις είναι ανεξάρτητοι από οποιοδήποτε συγκεκριμένο μηχανισμό επικοινωνίας. Την περίοδο αυτή, υπάρχουν τρεις υλοποιήσεις: HTTP, TCP και Bluetooth. Οι σωληνώσεις αλλάζουν δυναμικά σε χρόνο εκτέλεσης ανάλογα με αυτό που είναι διαθέσιμο για χρήση. Με αυτήν την έννοια, μια σωλήνωση μπορεί να αντιμετωπισθεί ως αφηρημένος, που ονομάζεται ουρά μηνυμάτων η οποία υποστηρίζει μια σειρά αφηρημένων λειτουργιών όπως δημιουργία, άνοιγμα, κλείσιμο, διαγραφή, αποστολή και λήψη. Οι σωληνώσεις παρέχουν δύο τρόπους επικοινωνίας, σημειακή και πολυεκπομπή, όπως φαίνεται στο Σχήμα 14.3.

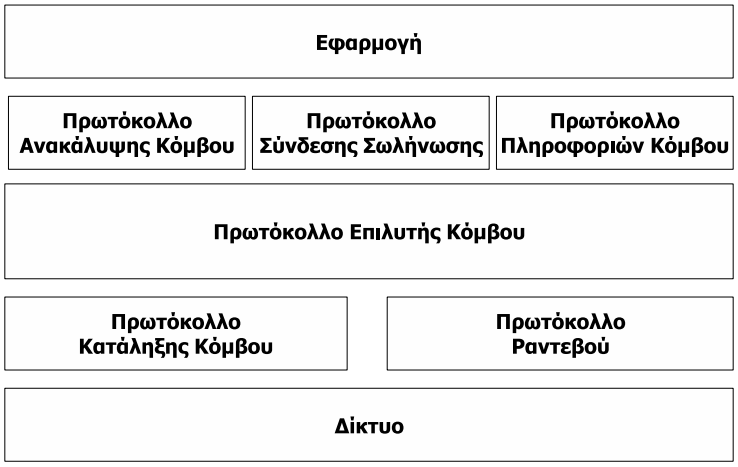
Μια σημειακή σωλήνωση συνδέει ακριβώς δύο σημεία κατάληξης σωλήνωσης: ένας σωλήνας εισόδου σε ένα κόμβο για να λαμβάνει μηνύματα που στέλνονται από το σωλήνα εξόδου ενός άλλου κόμβου. Μια σωλήνωση πολυεκπομπή συνδέει έναν σωλήνα εξόδου με πολλαπλούς σωλήνες εισόδου. Ο πυρήνας JXTA παρέχει επίσης ασφαλείς σωληνώσεις μονής κατεύθυνσης, μια ασφαλή παραλλαγή της σημειακής σωλήνωσης που παρέχει έναν ασφαλές κανάλι επικοινωνίας.

14.3.4 Κόμβοι Αναμετάδοσης

Ένας **κόμβος αναμετάδοσης** (relay node) JXTA είναι ένα είδος δρομολογητής JXTA για μηνύματα JXTA. Υπάρχουν πολλά παραδείγματα γιατί αυτά απαιτούνται. Μπορούν να χρησιμοποιηθούν για να βοηθήσουν να διασχίσουν τις αντιπυρικές ζώνες ή να βοηθήσουν τους κόμβους μίκρο-JXTA. Για παράδειγμα, η έκδοση της Java JXTA για μικροσυσκευές (π.χ. οι PDAs χρησιμοποιούν J2ME) επικοινωνούν απλά με έναν αναμεταδότη JXTA (ένας κόμβος ανεμετάδοσης μηνυμάτων), ο οποίος αναλαμβάνει το μεγαλύτερο μέρος της επεξεργασίας μηνυμάτων (όπως συγγραφή XML για διαφημίσεις, αποστολή μηνυμάτων αναζήτησης μέσω του δικτύου JXTA, και ούτω καθεξής) και αναμετάδοσης φορτίου. Ένας κόμβος βασισμένος στο J2ME, μαζί με έναν αναμεταδότη JXTA είναι λειτουργικά ισοδύναμος με ένα κανονικό κόμβο JXTA. Συνεπώς, οι κόμβοι J2ME δρουν ως συσκευή άκρου, πάνω στην περίμετρο ενός δικτύου JXTA.

14.4 Πρωτόκολλα JXTA

Τα πρωτόκολλα JXTA είναι ένα σύνολο έξι πρωτοκόλλων που έχουν σχεδιαστεί για ad hoc και ομότιμα δίκτυα. Χρησιμοποιώντας τα πρωτόκολλα JXTA, οι ομότιμοι κόμβοι μπορούν να συνεργαστούν για να σχηματίζουν αυτο-οργανωμένες και αυτο-διαχειρίσιμες ομάδες ομότιμων ανεξάρτητα από τις θέσεις τους στο δίκτυο (άκρες, αντιπυρικές ζώνες) και χωρίς την ανάγκη υποδομή κεντρικής διαχείρισης. Στο Σχήμα 14.4 φαίνεται η ιεραρχία των έξι πρωτοκόλλων που δεν ανεξάρτητοι μεταξύ τους. Κάθε επίπεδο στην στοίβα πρωτοκόλλων βασίζεται στο επίπεδο που ακολουθεί ώστε να παρέχει συνδεσιμότητα στους άλλους κόμβους. Στις επόμενες έξι υποενότητες περιγράφονται τα πρωτόκολλα.



Σχήμα 14.4: Ιεραρχία των έξι πρωτοκόλλων JXTA

14.4.1 Πρωτόκολλο Ανακάλυψης Κόμβου

Ένας κόμβος χρησιμοποιεί το πρωτόκολλο ανακάλυψης κόμβου για να ανακαλύψει έναν πόρο JXTA. Οι πόροι JXTA περιγράφονται από τις διαφημίσεις XML και μπορούν να είναι διαφορετικών τύπων π.χ. υπηρεσίες, σωληνώσεις, κόμβοι, ομάδες κόμβων, ή οποιεσδήποτε άλλες διαφημίσεις. Χρησιμοποιώντας το πρωτόκολλο αυτό, οι κόμβοι μπορούν να διαφημίσουν τους δικούς πόρους τους και να ανακαλύπτουν τους πόρους από άλλους κόμβους.

Η JXTA δεν εξουσιοδοτεί ακριβώς πώς γίνεται η ανακάλυψη. Έτσι, η ανακάλυψη μπορεί να είναι αποκεντρωμένη, κεντρική ή υβριδική των δύο. Υπάρχουν δύο επίπεδα ανακάλυψης: η

εγγραφή σε ένα δίκτυο JXTA και η ανακάλυψη ενός πόρου JXTA μέσα σε ένα δίκτυο JXTA. Επιπλέον, υπάρχουν δύο μέθοδοι για την εγγραφή σε ένα δίκτυο JXTA:

- Πολυεκπομπή. Η οποία χρησιμοποιεί πρωτοκόλλο πολυεκπομπής σε ένα τοπικό δίκτυο δηλαδή, μόλις ένας κόμβος JXTA συνδεθεί στο δίκτυο, εκπέμπει ή μεταδίδει μια ανακοίνωση παρουσία με την αποστολή ενός πακέτου πολυεκπομπής σε μια ευρέως γνωστή θύρα. Οι υπόλοιποι κόμβοι παρακολουθούν την θύρα αυτή χρησιμοποιώντας αιτήσεις πολυεκπομπής για κατάλληλα πακέτα και όταν λαμβάνονται έρχονται σε επαφή με τους άλλους κόμβους.

- Σημειακή. Το οποίο χρησιμοποιείται όταν ο κόμβος γνωρίζει τη θέση ενός κόμβου JXTA, χαρακτηριστικά ένας κόμβος ραντεβού (ή μια συλλογή γνωστών κόμβων) και επομένως μπορεί να έρθει σε επαφή άμεσα. Τυπικά, οι κόμβοι αναζητούν για διάφορους κόμβους ραντεβού για πλεονασμό.

Μόλις ένας κόμβος εγγραφεί στο δίκτυο JXTA, μπορεί να ανακαλύψει με διάφορους τρόπους τους πόρους JXTA. Για παράδειγμα, ένας κόμβος μπορεί να χρησιμοποιήσει τη ανακάλυψη σειρά, δηλαδή εάν ένας κόμβος ανακαλύπτει ένα δεύτερο κόμβο, ο πρώτος κόμβος μπορεί να δει τον ορίζοντα του δεύτερου κόμβου, ανακαλύπτοντας νέους κόμβους, ομάδες και υπηρεσίες. Διαφορετικά, ανακαλύπτει άλλους κόμβους (ή πόρους) μέσω των σημείων ραντεβού.

14.4.2 Πρωτόκολλο Επιλυτής Κόμβου

Το πρωτόκολλο επιλυτής κόμβου επιτρέπει σε έναν κόμβο να υλοποιήσει δυνατότητες αναζήτησης υψηλού επιπέδου, επιτρέποντας στο ομότιμο κόμβο να στείλει και να λαμβάνει γενικές ερωτήσεις ώστε να εντοπίσει ή να αναζητήσει κόμβους, ομάδες κόμβων, σωληνώσεις και άλλες πληροφορίες.

14.4.3 Πρωτόκολλο Πληροφοριών Κόμβου

Το πρωτόκολλο πληροφοριών κόμβου επιτρέπει στους κόμβους να πληροφορηθούν για τις δυνατότητες και την κατάσταση των άλλων κόμβων π.χ. uptime, φορτίο κυκλοφορίας, κλπ. Για παράδειγμα, ένας κόμβος θα μπορούσε να στείλει ένα μήνυμα `ping` για να διαπιστώσει αν ένας

άλλος κόμβος είναι σε σύνδεση (ή ζωντανός). Επίσης, μπορεί κάποιος κόμβος να ρωτήσει τις ιδιότητες ενός κόμβου όπου κάθε ιδιότητα είναι ένα ζεύγος ονόματος και τιμή αλφαριθμητικού.

14.4.4 Πρωτόκολλο Σύνδεσης Σωλήνωσης

Το πρωτόκολλο σύνδεσης σωλήνωσης επιτρέπει σε έναν κόμβο να εγκαταστήσει ένα εικονικό κανάλι επικοινωνία (δηλαδή, μια σωλήνωση) μεταξύ ενός ή περισσότερων κόμβων. Το πρωτόκολλο επιτρέπει τη σύνδεση των δύο ή περισσοτέρων άκρων σωλήνωσης για να διαμορφώσει τη σύνδεση. Συγκεκριμένα, ένας κόμβος συνδέει μια σωλήνωση διαφήμιση με μια σωλήνωση κατάληξης για να δημιουργήσει μια εικονική σύνδεση. Η σύνδεση εμφανίζεται κατά τη διάρκεια της λειτουργίας άνοιγμα, ενώ η αποσύνδεση εμφανίζεται κατά τη διάρκεια της λειτουργίας κλείσιμο.

14.4.5 Πρωτόκολλο Δρομολόγησης Κατάληξης

Το πρωτόκολλο δρομολόγησης κατάληξης επιτρέπει σε έναν κόμβο να αναζητήσει πληροφορίες σχετικά με τις διαθέσιμες διαδρομές για την αποστολή ενός μηνύματος στον κόμβο παραλήπτη. Οι κόμβοι που υλοποιούν το πρωτόκολλο δρομολόγησης κατάληξης ανταποκρίνονται σε ερωτήσεις με διαθέσιμες πληροφορίες διαδρομών δίνοντας έναν κατάλογο πυλών (gateways) κατά μήκος της διαδρομής.

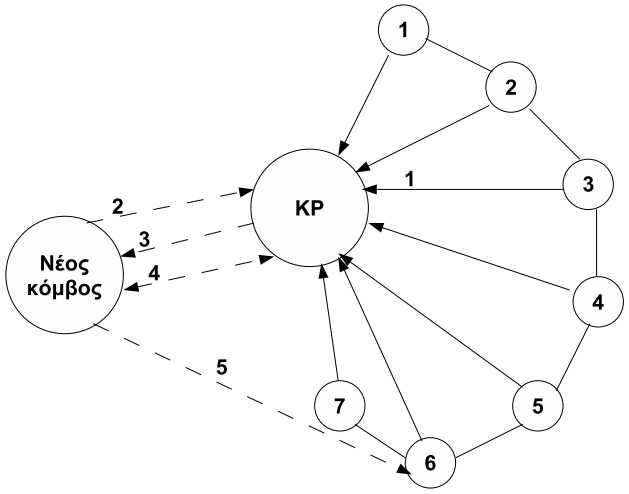
14.4.6 Πρωτόκολλο Ραντεβού

Το πρωτόκολλο ραντεβού επιτρέπει σε έναν κόμβο να στείλει μηνύματα σε όλους τους ακροατές της υπηρεσίας. Εξ ορισμού, τα μηνύματα ερώτησης φθάνουν μόνο στους κόμβους εντός του ίδιου φυσικού δίκτυου. Το πρωτόκολλο ραντεβού καθορίζει πώς ένας κόμβος μπορεί εγγραφεί ή να είναι συνδρομητής σε μια υπηρεσία πολυεκπομπής για να δημιουργήσει μεγαλύτερες κοινότητες. Η εμβέλεια του κόμβου ραντεβού είναι η ομάδα κόμβων. Το πρωτόκολλο ραντεβού επιτρέπει σε έναν κόμβο να εκπέμπει μηνύματα σε όλους τους ακροατές της υπηρεσίας.

14.5 Εφαρμογή JXTA

Σε αυτή την ενότητα θα αναφερθούμε πως σχετίζονται με τα προηγούμενα πρωτόκολλα JXTA στην ανάπτυξη μιας ομότιμης εφαρμογής. Θεωρούμε ένα ομότιμο κόμβο JXTA όπως ένας

κόμβος Gnutella που μπορεί να είναι πελάτης και διακομιστής συγχρόνως. Επίσης, οι κόμβοι JXTA μπορούν να δρούν ως κόμβοι ραντεβού οι οποίοι λειτουργούν ως σημεία συνάντησης (ή διακομιστές αναζήτησης) για τους άλλους κόμβους JXTA. Στο Σχήμα 14.5 φαίνεται ένα σενάριο για το πώς ένας κόμβος εγγράφεται ή συνδέεται σε ένα δίκτυο JXTA, εκτελεί ένα ερώτημα αναζήτησης και εντοπίζει ένα αρχείο για μεταφόρτωση. Τα διάφορα πρωτόκολλα JXTA που



Σχήμα 14.5: Σενάριο σύνδεσης για απλή αναζήτηση αρχείου χρησιμοποιώντας JXTA

περιλαμβάνονται στις λειτουργίες που συμβαίνουν σε κάθε στάδιο περιγράφονται ως εξής:

1. Ο κόμβος ραντεβού, KP, δέχεται μια σύνδεση για τους κόμβους 1 - 7 και αποθηκεύει τις διαφημίσεις τους τοπικά.

2. Ένας νέος κόμβος έρχεται σε επαφή με το KP χρησιμοποιώντας ένα μηχανισμό ανακάλυψης όπως μονοεκπομπή/πολυεκπομπή μέσω του πρωτοκόλλου ανακάλυψης κόμβου.

3. Ο KP πιστοποιεί το νέο κόμβο και το προσθέτει στην ομάδα.

4. Ο νέος κόμβος εκτελεί ένα ερώτημα αναζήτησης αρχείου στο KP έτσι ώστε να αναζητήσει για τοπική ταύτιση ή να εκπέμψει το ερώτημα αυτό στα υπόλοιπα μέλη της ομάδας. Το αρχείο εντοπίζεται τελικά στο κόμβο 6. Σε αυτό το βήμα χρησιμοποιεί το πρωτόκολλο ανακάλυψης κόμβου, το πρωτόκολλο επιλυτής κόμβου και το πρωτόκολλο δρομολόγησης κατάληξης.

5. Εγκαθίστανται μια σύνδεση ανάμεσα στο νέο κόμβο και το κόμβο 6 για άμεση επικοινωνία μέσω της σωλήνωσης JXTA. Αυτή η σύνδεση είναι εικονική και ίσως διασχίσει μέσω των κόμβων KP και 7. Αυτό το βήμα χρησιμοποιεί το πρωτόκολλο σύνδεσης σωλήνωσης, το πρωτόκολλο επιλυτής κόμβου και το πρωτόκολλο δρομολόγησης κατάληξης.

14.6 Java για JXTA

14.6.1 Βασική Δομή μιας Εφαρμογής JXTA

Για όλες τις εφαρμογές JXTA ακολουθούμε μια τυπική δομή προγράμματος. Η τυπική δομή κάθε εφαρμογής JXTA περιλαμβάνει μια κλάση και μια μέθοδο `main()`. Μέσα στην μέθοδο `main()` καλεί μια άλλη μέθοδο `startJxta()` η οποία εκτελεί κάποια βασική αρχικοποίηση της πλατφόρμας JXTA. Παρακάτω φαίνεται η δομή μιας εφαρμογής JXTA σε Java.

```
public class SimpleJxtaApp {
    public static void main(String args[]) {
        System.out.println("Starting JXTA ....");
        SimpleJxtaApp myapp = new SimpleJxtaApp();
        myapp.startJxta();
        System.exit(0);
    }
    private void startJxta() {
    }
}
```

Η κάθε εφαρμογή JXTA για να γίνει μέλος κόμβου μιας ομάδας και κατόπιν να συνδεθεί στο δίκτυο JXTA πρέπει να εισάγουμε κώδικα μέσα στην μέθοδο `startJxta()`. Έτσι, όλες οι εφαρμογές JXTA (ή κόμβοι) που θέλουν να είναι μέλη μιας ομάδας κόμβων πρέπει να χρησιμοποιούν την κλάση `PeerGroupFactory` που επιτρέπει να δημιουργήσουν ένα αντικείμενο `PeerGroup` που αντιστοιχεί σε μια ομάδα κόμβων. Η κλάση `PeerGroupFactory` διαθέτει πολλές μεθόδους για την δημιουργία αντικειμένου `PeerGroupFactory`. Η πιο κοινή μέθοδος είναι η `newNetPeerGroup()` και η υπογραφή της φαίνεται παρακάτω:

- `static PeerGroup newNetPeerGroup()`. Δημιουργεί μια προκαθορισμένη ομάδα κόμβων με όνομα NetPeerGroup.

Σαν γενικός κανόνας, όλες οι εφαρμογές JXTA θα ανήκουν στην ομάδα NetPeerGroup. Η ομάδα αυτή παρέχει ένα αριθμό υπηρεσιών για εύρεση και δημιουργία άλλων ομάδων κόμβων.

Επομένως, όταν ένας κόμβος ή μια εφαρμογή JXTA καλέσει επιτυχώς την μέθοδο `newNetPeerGroup` τότε ο τρέχων κόμβος συνδέεται με ένα κόμβο δρομολογητή (ή ραντεβού) και στην ομάδα NetPeerGroup. Σε διαφορετική περίπτωση δημιουργεί μια εξαίρεση τύπου `PeerGroupException`. Κατά την σύνδεση του κόμβου στην ομάδα, ανατίθεται ορισμένες πληροφορίες όπως το ID της ομάδας, το όνομα του κόμβου κλπ.

Για να αποκτήσουμε πληροφορίες σχετικά με το κόμβο που ανήκει στην προκαθορισμένη ομάδα NetPeerGroup υπάρχουν μερικές μεθόδους από το αντικείμενο `PeerGroup` το οποίο επιστρέφεται από την μέθοδο `newNetPeerGroup()`. Οι μέθοδοι αυτοί είναι οι εξής:

- `getPeerGroupID()`. Επιστρέφει το id της ομάδας που συσχετίζεται με το αντικείμενο `PeerGroup`.
- `getPeerGroupName()`. Επιστρέφει το όνομα της ομάδας που συσχετίζεται με το αντικείμενο `PeerGroup`.
- `getPeerID()`. Επιστρέφει το id του κόμβου που συσχετίζεται με το αντικείμενο `PeerGroup`.
- `getPeerName()`. Επιστρέφει το όνομα του κόμβου που συσχετίζεται με το αντικείμενο `PeerGroup`.

Παρακάτω παρουσιάζεται ένα παράδειγμα προγράμματος που ένας κόμβος συνδέεται στο δίκτυο JXTA και εμφανίζουμε πληροφορίες σχετικά με το κόμβο.

```
import net.jxta.peergroup.PeerGroup;
import net.jxta.peergroup.PeerGroupFactory;
import net.jxta.exception.PeerGroupException;
public class SimpleJxtaApp {
    static PeerGroup netPeerGroup = null;
    public static void main(String args[]) {
        System.out.println("Starting JXTA ....");
        SimpleJxtaApp myapp = new SimpleJxtaApp();
        myapp.startJxta();
        System.out.println("Hello from JXTA group " +
                           netPeerGroup.getPeerGroupName() );
        System.out.println("  Group ID = " +
                           netPeerGroup.getPeerGroupID().toString());
        System.out.println("  Peer name = " +
                           netPeerGroup.getPeerName());
        System.out.println("  Peer ID = " +
                           netPeerGroup.getPeerID().toString());
```

```
        System.out.println( "Good Bye ....");
        myapp.netPeerGroup.stopApp();
        System.exit(0);
    }

    private void startJxta() {
        try {
            // create and start the default JXTA NetPeerGroup
            netPeerGroup = PeerGroupFactory.newNetPeerGroup();
        } catch (PeerGroupException e) {
            // could not instantiate the group, print the stack and exit
            System.out.println("fatal error : group creation failure");
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

14.6.2 Ανακάλυψη JXTA

Οι κλάσεις που είναι απαραίτητες για την υλοποίηση της υπηρεσίας ανακάλυψης μέσα σε μια ομάδα ομότιμων κόμβων και τελικά ένας κόμβος είναι οι εξής:

- `net.jxta.discovery.DiscoveryService`
- `net.jxta.discovery.DiscoveryListener`
- `net.jxta.impl.discovery.DiscoveryServiceImpl`
- `net.jxta.impl.discovery.DiscoveryServiceInterface`

Υπηρεσία Ανακάλυψης

Το βασικό σημείο για την υλοποίησης της ανακάλυψης είναι η υπηρεσία ανακάλυψης. Η υπηρεσία αυτή έχει διπλό στόχο: την δημοσίευση νέων διαφημίσεων και αναζήτηση για τις διαφημίσεις. Έτσι, η υπηρεσία ανακάλυψης είναι μια βασική υπηρεσία που παρέχεται στην ομάδα NetPeerGroup ή σε οποιαδήποτε άλλη ομάδα που δημιουργείται. Για να αποκτήσουμε μια αναφορά σε ένα αντικείμενο υπηρεσίας ανακάλυψης JXTA χρησιμοποιούμε την μέθοδο `getDiscoveryService()` του αντικειμένου `PeerGroup` του οποίου αποκτάται όταν ο κόμβος συνδέεται στην ομάδα. Για παράδειγμα,

```
PeerGroup netPeerGroup;

myDiscoveryService DiscoveryService = netPeerGroup.getDiscoveryService();
```

Δημοσίευση Διαφημίσεων

Πριν αναζητηθούν οι διαφημίσεις, θα πρέπει μερικές διαφημίσεις να υπάρχουν σε μια ομάδα κόμβων. Όλοι οι κόμβοι θα αποθηκεύουν διαφημίσεις σχετικές με τους ευατούς τους και την ομάδα τους κόμβων. Επίσης, μια εφαρμογή θα χρειαστεί να δημοσιεύσει διαφημίσεις σχετικά με τους πόρους που είναι διαθέσιμοι στην ομάδα. Η δημοσίευση των διαφημίσεων μπορεί να είναι είτε τοπική είτε απομακρυσμένη. Όλες οι δημοσιευμένες διαφημίσεις έχουν μια προκαθορισμένη διάρκεια ζωής και μια προκαθορισμένη λήξη που είναι εκφρασμένες σε χιλιοστά του δευτερολέπτου. Η JXTA υποστηρίζει τις δύο παραπάνω τιμές από τις δύο παρακάτω προκαθορισμένες σταθερές:

```
public final static long DEFAULT_LIFETIME = 1000 * 60 * 60 * 24 * 365;

public final static long DEFAULT_EXPIRATION = 1000 * 60 * 60 * 2;
```

Μια τοπική δημοσίευση διαφήμισης σημαίνει ότι η διαφήμιση θα τοποθετηθεί στην μνήμη του υπό εκτέλεση κόμβου και χρησιμοποιεί μια πολυεκπομπή εάν είναι διαθέσιμη, για να διαδώσει την διαφήμιση στους υπόλοιπους κόμβους. Εάν είναι διαθέσιμη η μεταφορά TCP τότε όλοι οι υπόλοιποι κόμβοι του τοπικού δικτύου θα λάβουν ένα αντίγραφο της δημοσιευμένης διαφήμισης. Οι μέθοδοι για τοπική δημοσίευση είναι οι εξής:

- `public void publish(Advertisement advertisement, int type)`. Η διαφήμιση θα δημοσιευθεί χρησιμοποιώντας τις προκαθορισμένες τιμές διάρκεια ζωής και λήξης.

- `public void publish(Advertisement adv, int type, long lifetime, long lifetimeForOthers)`. Η διαφήμιση θα δημοσιευθεί εισάγοντας νέες τιμές διάρκειας ζωής και λήξη σε χιλιοστά του δευτερολέπτου.

Μια απομακρυσμένη δημοσίευση διαφήμιση σημαίνει ότι η διαφήμιση θα τοποθετηθεί στην τοπική μνήμη όπως και τη εκπομπή στην τρέχουσα ομάδα κόμβων που χρησιμοποιεί όλα τα διαθέσιμα πρωτόκολλα μεταφοράς. Αυτό σημαίνει ότι η διαφήμιση θα διαδοθεί σε όλους τους απομακρυσμένους κόμβους ραντεβού (ή δρομολογητές) χρησιμοποιώντας το HTTP ή το TCP (ανάλογα ποιο πρωτόκολλο είναι διαθέσιμο). Προφανώς, οι διαφημίσεις που δημοσιεύονται

απομακρυσμένα θα έχουν ευρύτερο κοινό. Οι μέθοδοι για απομακρυσμένη δημοσίευση είναι οι εξής:

- `public void remotePublish(Advertisement adv, int type).`
- `public void remotePublish(Advertisement adv, int type, long lifetime).`

Οι παραπάνω μέθοδοι έχουν παρόμοια σύνταξη με εκείνες τις μεθόδους τοπικής δημοσίευσης.

Ανακάλυψη Διαφημίσεων

Η δεύτερη λειτουργία που παρέχεται στο αντικείμενο DiscoveryService είναι η αναζήτηση για διαφημίσεις που ικανοποιεί ένα συγκεκριμένο κριτήριο. Οι μέθοδοι για την αναζήτηση διαφημίσεων σε μια ομάδα κόμβων είναι οι εξής:

- `public Enumeration getLocalAdvertisements(int type, String attribute, String value )`. Η μέθοδος αυτή είναι υπεύθυνη για την διαλογή διαφημίσεων από την τοπική μνήμη του τρέχοντος κόμβου μόνο. Όλες από τις διαφημίσεις που θα εντοπίζει και οι οποίες ταιριάζουν με τις παραμέτρους `type`, `attributed` και `value` της μεθόδου θα επιστρέφονται σε ένα αντικείμενο `Emumeration`.

- `public int getRemoteAdvertisements(String peerid, int type, String attribute, String value, int threshold).` Η μέθοδος αυτή θα στείλει ένα μήνυμα ερώτηση σε όλους τους πιθανούς κόμβους για να εντοπίζει το πολύ `threshold` διαφημίσεις που ταιριάζουν με τις παραμέτρους της μεθόδου. Όταν ένας απομακρυσμένος κόμβος εντοπίζει μια ταύτιση, η διαφήμιση επιστρέφεται στο κόμβο αιτών. Η επιστρεφόμενη διαφήμιση θα τοποθετηθεί στην τοπική μνήμη και μπορεί να αναζητηθεί από μια κλήση `getLocalAdvertisements()`.

- `public void getRemoteAdvertisements(String peerid, int type, String attribute, String value, int threshold, DiscoveryListener listener).` Η μέθοδος αυτή είναι παρόμοια με την προηγούμενη απλά προσδιορίζει ένα αντικείμενο ακροατή `listener` το οποίο θα καλείται όταν επιστρέφεται μια οποιαδήποτε απομακρυσμένη διαφήμιση που σχετίζεται με την αναζήτηση.

Είδαμε στην τελευταία μέθοδο `getRemoteAdvertisements()` ότι τοποθετεί μια επιστρεφόμενη διαφήμιση στην τοπική μνήμη με την δυαντότητα να καλεί μια μέθοδος με ασύγχρονο τρόπο κάθε

φορά που επιστρέφει νέα διαφήμιση. Αυτή η δυνατότητα υλοποιείται με το αντικείμενο ακροατή `listener` το οποίο λειτουργεί με ένα παρόμοιο τρόπο στο γεγονοστρεφής προγραμματισμός της Java. Ο μηχανισμός αυτός λέγεται επανάκληση (callback). Παρακάτω περιγράφεται ο τρόπος υλοποίησης του μηχανισμού επανάκλησης. Πρώτα, πρέπει να προσδιορίζουμε στην γραμμή ορισμού κλάσης ότι υλοποιεί την διεπαφή `DiscoveryListener`, διεπαφή που χρησιμοποιείται για να υλοποιήσει το απομακρυσμένο μηχανισμό επανάκλησης αναζήτησης. Έτσι, η δήλωση κλάση θα φαίνεται ως εξής:

```
public class Client implements DiscoveryListener {

// code to implement listener goes here

}
```

Η διεπαφή `DiscoveryListener` απαιτεί να περιέχει μια μοναδική μέθοδος με την παρακάτω υπογραφή:

```
public void discoveryEvent(DiscoveryEvent e)
```

Η μέθοδος `DiscoveryEvent()` καλείται κάθε φορά που λαμβάνει μια νέα απάντηση διαφήμισης. Η παράμετρος που μεταβιβάζεται στην μέθοδο είναι ένα αντικείμενο `DiscoveryEvent` το οποίο περιέχει ένα αντικείμενο `DiscoveryResponseMsg`. Το αντικείμενο `Msg` διαλέγεται από το `DiscoveryEvent` με την χρήση της μεθόδου `getResponse()`. Μια συλλογή απαντήσεων - διαφημίσεων προκύπτει από την απομακρυσμένη αναζήτηση καλώντας την μέθοδο `getResponse()` του αντικειμένου `DiscoveryResponseMsg`. Στην συνέχεια, οι διαφημίσεις που επιστρέφονται ελέγχονται και επεξεργάζονται.

Μετά την δημιουργία του αντικειμένου ακροατή, πρέπει το αντικείμενο αυτό να συνδεθεί με το αντικείμενο `DiscoveryService`. Βασικά, υπάρχουν δύο τρόποι για αυτή την σύνδεση. Ο πρώτος τρόπος είναι η υπερφόρτωση της μεθόδου `getRemoteAdvertisements()`. Για παράδειγμα, η μέθοδος

```
myDiscoveryService.getRemoteAdvertisements(null, DiscoveryService.ADV, searchKey, searchValue, 1, myDiscoveryListener);
```

μεταβιβάζουμε το αντικείμενο `DiscoveryListener` στην υπηρεσία ανακάλυψης ως τελευταία παράμετρο.

Ο δεύτερος τρόπος για να συνδέσουμε το ακροατή είναι να χρησιμοποιήσουμε την μέθοδο `addDiscoveryListener()` του αντικειμένου `DiscoveryService`. Η μέθοδος είναι η εξής:

```
public void addDiscoveryListener(DiscoveryListener listener)
```

Παρακάτω παρουσιάζεται ένα τμήμα κωδικα για την ανακάλυψη.

```
public class DiscoveryDemo implements Runnable, DiscoveryListener {
    static PeerGroup netPeerGroup  = null;
    private DiscoveryService discovery;
```

```
//start the JXTA platform
 private void startJxta() {
     try {
         netPeerGroup = PeerGroupFactory.newNetPeerGroup();
     } catch ( PeerGroupException e) {
         // could not instantiate the group, print the stack and exit
         System.out.println("fatal error : group creation failure");
         e.printStackTrace();
         System.exit(1);
     }

     // Get the discovery service from our peer group
     discovery = netPeerGroup.getDiscoveryService();
 }

 /**
  * This thread loops forever discovering peers
  * every minute, and displaying the results.
  */

 public void run() {
     try {
         // Add ourselves as a DiscoveryListener for DiscoveryResponse events
         discovery.addDiscoveryListener(this);
         while (true) {
             System.out.println("Sending a Discovery Message");
             // look for any peer
             discovery.getRemoteAdvertisements(null, DiscoveryService.PEER,
                                               null, null, 5);
             // wait a bit before sending next discovery message
             try {
                 Thread.sleep(60 * 1000);
             } catch(Exception e) {}

         } //end while
     } catch(Exception e) {
         e.printStackTrace();
     }
 }

 /**
  * by implementing DiscoveryListener we must define this method
  * to deal to discovery responses
  */

 public void discoveryEvent(DiscoveryEvent ev) {
```

```
        DiscoveryResponseMsg res = ev.getResponse();
        String name = "unknown";

        // Get the responding peer's advertisement
        PeerAdvertisement peerAdv = res.getPeerAdvertisement();
        // some peers may not respond with their peerAdv
        if (peerAdv != null) {
            name = peerAdv.getName();
        }

        System.out.println("Got a Discovery Response [" +
                           res.getResponseCount() + " elements] from peer: " +
                           name);
        //printout each discovered peer
        PeerAdvertisement adv = null;
        Enumeration en = res.getAdvertisements();
        if (en != null ) {
            while (en.hasMoreElements()) {
                adv = (PeerAdvertisement) en.nextElement();
                System.out.println (" Peer name = " + adv.getName());
            }
        }
    }

    static public void main(String args[]) {
        DiscoveryDemo myapp  = new DiscoveryDemo();
        myapp.startJxta();
        myapp.run();
    }
}
```

Bibliography

[1] R. Buyya. *High Performance Cluster Computing: Architectures and Systems*, volume 1. Prentice Hall, 1999.

[2] R. Buyya. *High Performance Cluster Computing: Programming and Applications*, volume 2. Prentice Hall, 1999.

[3] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing*. Massachusetts: The MIT Press, 1994.

[4] D. Loveman. High-performance fotran. *IEEE Parallel and Distributed Technology*, 1(1), 1993.

[5] B. Nichols, D. Buttlar, and J. P. Farrell. *Pthreads Programming*. O'Reilly, 1996.

[6] G. Pfister. *In Search of Clusters*. Prentice Hall, 1998.

[7] M. Snir, S. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. *MPI: The complete reference*. Massachusetts: The MIT Press, 1996.

[8] E. Lusk W. Gropp and A. Skjellum. *Using MPI: Portable parallel programming with the message passing interface*. Massachusetts: The MIT Press, 1994.