Πολυτεχνειο Κρητης
Τμημα Ηλεκτρονικων Μηχανικων & Μηχανικων Η/Υ

Διπλωματικη Εργασια

Σχεδίαση και Υλοποίηση Ενδιάμεσου Λογισμικού σε P-Grid Δίκτυα Ανεκτικά σε Λάθη

Συγγραφέας:
Νικόλας Βουρλάκης

Επιβλέπων:
Βασίλης Σαμολαδάς

14 Ιανουαρίου 2013

Περίληψη

Τα peer-to-peer συστήματα εμπεριέχουν διάφορες προκλήσεις στον σχεδιασμό τους. Η πολυπλοκότητα τους μπορεί να αντιμετωπιστεί μέσω μιας καλής σχεδίασης. Ωστόσο, πολλά peer-to-peer συστήματα αναπτύσσονται με τρόπο που επιλύει το συγκεκριμένο πρόβλημα που αντιμετωπίζουν. Ένα θέμα, επομένως, είναι μια γενίκευση και τυποποίηση τέτοιων αρχιτεκτονικών. Ωστόσο, στη σχετική βιβλιογραφία βρίσκουμε λίγες αναφορές που ασχολούνται με αυτό το θέμα. Στην παρούσα διπλωματική εργασία, βασικός στόχος είναι η σχεδίαση της αρχιτεκτονικής και η υλοποίηση ενδιάμεσου λογισμικού ωφέλιμο για τη δημιουργία συστημάτων που εκμεταλλεύονται το peer-to-peer μοντέλο. Τα υπηρεσιοκεντρικά συστήματα και η θεωρία γύρω από αυτά ήταν η έμπνευση για τη σχεδίαση. Βάσει αυτής της αρχιτεκτονικής η προσθήκη νέων πολύπλοκων πρωτοκόλλων απλοποιείται. Η οργάνωση που ακολουθούν οι peer του δικτύου είναι αυτή του P-Grid. Το δίκτυο που σχηματίζεται είναι αναγκαίο να είναι ανεκτικό σε λάθη και να συνεχίζει να λειτουργεί σωστά υπό την παρουσία αποτυχιών. Ο δεύτερος στόχος της εργασίας είναι η δημιουργία ενός fault-tolerant πρωτοκόλλου, βάσει του οποίου το σύστημα μπορεί να διατηρήσει την τοπολογία του, όχι μόνο στην περίπτωση αποτυχιών αλλά και στις μεταβολές που παρουσιάζονται λόγω του φαινομένου churn.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή κ. Βασίλη Σαμολαδά, για την καθοδήγηση και τη βοήθεια που πρόσφερε κατά τη διάρκεια εκπόνησης της παρούσας διπλωματικής εργασίας. Επίσης, την οικογένεια μου και τη σύντροφό μου, για την υπομονή και υποστήριξη που έδειξαν κατά τη διάρκεια των σπουδών μου.

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος αλγορίθμων

Κεφάλαιο 1

Εισαγωγή

1.1 Πρόβλημα και Πλαίσιο Έρευνας

Τα peer-to-peer συστήματα έχουν αναδειχθεί τα τελευταία χρόνια όχι μόνο ως ερευνητικό αντικείμενο, αλλά και ως ένα επιτυχημένο μοντέλο ανάπτυξης εφαρμογών. Αρχικά έγιναν γνωστά από εφαρμογές διαμοιρασμού αρχείων. Στην πορεία όμως, αναπτύχθηκαν συστήματα σε ευρύτερο φάσμα εφαρμογών. Συγκεντρωτικά, υπάρχουν διάφοροι τομείς που εκμεταλλεύονται το μοντέλο των peer-to-peer συστημάτων, όπως αναφέρεται στην [1]. Έχουμε συστήματα στον τομέα της επικοινωνίας (communication & collaboration),όπως chat και υπηρεσίες άμεσων μηνυμάτων (instant messaging).Στο χώρο του κατανεμημένου υπολογισμού (distributed computation) παράδειγμα αποτελεί το σύστημα Seti@home. Έχουν αναπτυχθεί συστήματα που παρέχουν υπηρεσίες υποστήριξης, όπως για παράδειγμα εφαρμογές που αφορούν την ασφάλεια συστημάτων. Σημαντική εργασία έχει γίνει στο χώρο των κατανεμημένων βάσεων δεδομένων, όπως NoSQL συστημάτων. Τέλος, έχουμε συστήματα διανομής περιεχομένου (content distribution) που είναι και η πιο συνηθισμένη εφαρμογή των peer-to-peer συστημάτων.

Με την εξέλιξη της τεχνολογίας έχουμε τη δημιουργία νέων αναγκών και απαιτήσεων όσον αφορά τα τεχνολογικά χαρακτηριστικά που προσφέρουν τα peer-to-peer συστήματα. Η αρχιτεκτονική λογισμικού που ακολουθεί ένα τέτοιο σύστημα είναι αυτή που θέτει τα όρια στον προγραμματιστή αυτών. Την τελευταία δεκαετία έχουν γίνει διάφορες δοκιμές και πειραματισμοί στην προσπάθεια τυποποίησης αυτών. Η άνοδος του μοντέλου της υπηρεσιοκεντρικής αρχιτεκτονικής (Service-Oriented Architecture) μας δίνει την έμπνευση και το θεωρητικό υπόβαθρο για την σχεδίαση του λογισμικού που πραγματεύεται η παρούσα διπλωματική εργασία.

Ένα δεύτερο ζήτημα που τίθεται, είναι η διατήρηση του δικτύου και της τοπολογίας των peer-to-peer συστημάτων κατά τη διάρκεια ζωής τους. Όπως θα δούμε και στην ενότητα 5.3, αναφερόμαστε σε προβλήματα που δημιουργούνται από το φαινόμενο churn και την αποτυχία των peer. Ένας peer μπορεί να αποτύχει για δυο λόγους, είτε έχει πρόβλημα ο ίδιος (έχει κρασάρει κλπ), είτε

υπάρχει πρόβλημα στη σύνδεση του. Οπότε, σημαντικό χαρακτηριστικό είναι να συνεχίσει ένα peer-to-peer σύστημα να λειτουργεί σωστά, καθώς και οι υπηρεσίες που παρέχονται να είναι διαθέσιμες. Υπάρχει η ανάγκη ενός πρωτοκόλλου, με τη βοήθεια του οποίου προλαμβάνεται το προηγούμενο πρόβλημα. Είναι απαραίτητος αυτός ο μηχανισμός, ο οποίος να είναι σε θέση να αναγνωρίσει γρήγορα και με ακρίβεια peer που αστοχούν ή αποκλίνουν από την αναμενόμενη συμπεριφορά. Γενικά, ένας peer μπορεί να μαθαίνει πληροφορίες για τους υπόλοιπους, με δυο τρόπους, όπως έχει αναλυθεί και στη δημοσίευση [2]. Έχει την ικανότητα να ρωτήσει απευθείας τους peer που τον ενδιαφέρουν άμεσα, όπως αυτοί που είναι αποθηκευμένοι στο routing table του, περιμένοντας κάποια απάντηση από αυτούς ως απόδειξη ότι λειτουργούν σωστά. Επίσης, από τη στιγμή που κάποιοι peer απλά δρομολογούν τα μηνύματα, δεδομένου ότι δεν έχουν φτάσει ακόμα στον τελικό παραλήπτη, μπορεί να γίνει γνωστό στο δίκτυο ποιοι peer είναι ενεργοί. Χρησιμοποιώντας αυτές τις πληροφορίες οι peer μπορούν να διορθώσουν τις προβληματικές αναφορές. Μεγάλο ρόλο παίζει ο χρόνος που χρειάζεται για να αναγνωριστεί μια αστοχία. Όσο πιο γρήγορα εντοπιστεί το πρόβλημα, τόσο πιο γρήγορα θα διορθωθεί, επομένως το δίκτυο ξοδεύει λιγότερους πόρους για να στείλει μηνύματα σε peer που έχουν πέσει. Παρόμοια μοντέλα έχουν περιγραφεί στη δημοσίευση [3].

1.2 Στόχοι

Η διπλωματική χωρίζεται σε δυο κομμάτια. Το πρώτο κομμάτι αφορά την αρχιτεκτονική του ενδιάμεσου λογισμικού. Το λογισμικό που θα αναπτυχθεί θα καθιστά δυνατή τη δημιουργία εφαρμογών που στηρίζονται στο peer-to-peer μοντέλο. Η σχεδίαση καθοδηγείται από τις εξής απαιτήσεις:

- Ένα peer-to-peer σύστημα περιέχει πρωτόκολλα, βάσει αυτών οι peer αλληλεπιδρούν. Ανάγκη μιας κατηγορίας χρηστών είναι η επέκταση της βιβλιοθήκης με νέα πρωτόκολλα καθώς και ο μεταξύ τους συνδυασμός ώστε να προκύψουν πιο πολύπλοκα. Η διαδικασία αυτή, δηλαδή η δημιουργία και η σύνθεση πολύπλοκων πρωτοκόλλων καθώς και η ευκολία με την οποία επιτυγχάνεται αυτό, πρέπει να αποτυπωθεί στην αρχιτεκτονική.
- Παροχή μηχανισμού ανοχής λαθών.

Το δεύτερο κομμάτι της εργασίας, είναι η δημιουργία αυτού του μηχανισμού ανοχής λαθών. Επιδιώκουμε την βελτιστοποίηση της συμπεριφοράς του δικτύου κατά την αποτυχία των κόμβων του (fault-tolerance). Στόχος είναι η υλοποίηση πρωτοκόλλου, βάσει του οποίου το δίκτυο να μπορεί να αναγνωρίσει αστοχίες peer, να τις διορθώσει και να συνεχίσει να λειτουργεί ομαλά.

1.3 Οργάνωση Διπλωματικής

Στο κεφάλαιο 2 εξηγείται το μοντέλο που ακολουθούν τα peer-to-peer συστήματα. Επίσης, αναλύεται η δομή και ο τρόπος λειτουργίας του P-Grid [4--11] συστήματος.

Στο κεφάλαιο 3 γίνεται μια επισκόπηση της βιβλιογραφίας που ασχολείται με την αρχιτεκτονική των peer-to-peer συστημάτων. Αναλύεται η αρχιτεκτονική των υπηρεσιοκεντρικών συστημάτων και ο ρόλος που παίζει η υπηρεσία ως έννοια σε αυτά. Το συγκεκριμένο μοντέλο χρησιμοποιήθηκε ως βάση για την ανάπτυξη της δικής μας αρχιτεκτονικής.

Στο κεφάλαιο 4 περιγράφεται η γενική δομή της προτεινόμενης αρχιτεκτονικής. Παρουσιάζονται τα διάφορα αρχιτεκτονικά επίπεδα και η λειτουργικότητα που ενθυλακώνει το καθένα. Αναλύεται ο τρόπος που λύθηκαν κάποια βασικά θέματα και καταλήγει με τη γενική παρουσίαση των επιπέδων.

Στο κεφάλαιο 5 αναλύονται τα πρωτόκολλα που υλοποιούνται στο σύστημα. Το πρώτο, είναι τα βήματα που ακολουθούν δυο peer όταν επικοινωνούν, που περιλαμβάνει την εκτέλεση του αλγορίθμου exchange όπως περιγράφεται στη δημοσίευση [4]. Τέλος, αναλύεται το πρωτόκολλο ανοχής λαθών, ο τρόπος με τον οποίο οι peer διορθώνουν αποτυχίες στο δίκτυο.

Στο κεφάλαιο 6 αναλύεται ο τρόπος με τον οποίο υλοποιήθηκε η προτεινόμενη αρχιτεκτονική. Εξηγούνται οι δυνατότητες που πρόσφερε η χρήση της CORBA. Επίσης, αναλύονται διεξοδικά τα συστατικά κομμάτια κάθε επιπέδου.

Στο κεφάλαιο 7 παρουσιάζεται ο τρόπος με τον οποίο αναπτύσσει κάποιος εφαρμογές με τη χρήση της βιβλιοθήκης που υλοποιήθηκε. Συγκεκριμένα εισάγουμε λειτουργικότητα διαμοιρασμού αρχείων στο σύστημα. Αναλύοντας το πρόβλημα εξηγείται ο τρόπος σκέψης και υλοποίησης της λύσης.

Τέλος, στο κεφάλαιο 8 παρουσιάζονται τα συμπεράσματα αυτής της εργασίας και δίνονται μερικές ιδέες σαν αντικείμενο μελλοντικής εργασίας.

Κεφάλαιο 2

Συστήματα Peer-to-Peer

2.1 Εισαγωγή

Ένα peer-to-peer σύστημα είναι ένα δίκτυο αποτελούμενο από ισότιμους συμμετέχοντες, τους peer. Αυτοί διαμοιράζονται τους πόρους που έχει ο καθένας μεταξύ τους, όπως υπολογιστική ισχύ ή αποθηκευτικό χώρο. Χαρακτηριστικό του συστήματος, που το διαχωρίζει από την κλασική αρχιτεκτονική client-server, είναι πως ένας peer είναι πάροχος υπηρεσιών και πόρων και την ίδια στιγμή αιτείται αντίστοιχη χρήση από απομακρυσμένους peer. Τα peer-to-peer συστήματα χωρίζονται σε δομημένα (structured) και μη (unstructured).

Στα δομημένα peer-to-peer συστήματα η τοπολογία του δικτύου κατασκευάζεται ντετερμινιστικά με μια κλασική οργάνωση να είναι οι κατανεμημένοι πίνακες κατακερματισμού (DHT). Υπάρχει ένας ευρύτερος χώρος αναγνωριστικών από όπου τα δεδομένα και οι peer αντιστοιχούνται σε κλειδιά. Ένα σημαντικό ζήτημα του συστήματος είναι για ποια δεδομένα είναι υπεύθυνος ένας peer βάσει των κλειδιών τους. Παράδειγμα τέτοιου συστήματος είναι το Chord [12]. Οι peer οργανώνονται πάνω σε ένα κύκλο. Κάθε peer είναι υπεύθυνος για εκείνα τα δεδομένα που τους αναλογούν κλειδιά μικρότερα σε σχέση με αυτό του peer. Επίσης, ένας peer, με βάση την τοποθεσία του, διατηρεί μια λίστα από γείτονες peer οι οποίοι βρίσκονται σε συγκεκριμένες αποστάσεις πάνω στον κύκλο. Το αποτέλεσμα αυτής της τοπολογίας του Chord είναι πως ένα lookup εξυπηρετείται σε $O(logN)$ βήματα.

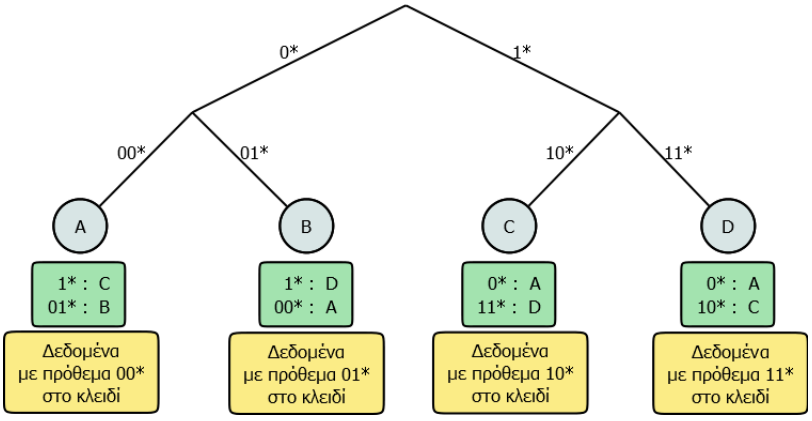
Η δεύτερη περίπτωση είναι τα μη δομημένα συστήματα. Αυτά στηρίζονται συνήθως σε τυχαιοκρατικούς αλγορίθμους για την κατασκευή της τοπολογίας. Δεν υπάρχει ένας συγκεκριμένος σχηματισμός και κάθε peer σχετίζεται με γείτονες που έχουν επιλεγεί τυχαία. Μια συνηθισμένη πρακτική σε αυτή την κατηγορία συστημάτων είναι η ύπαρξη των υπερκόμβων (supernode). Όσο το δίκτυο επεκτείνεται, τόσο περισσότερο αυξάνεται ο χρόνος εξυπηρέτησης ενός lookup. Ταυτόχρονα έχουμε και αυξημένη χρήση του bandwidth του δικτύου μιας και η πρακτική που ακολουθείται είναι αυτή του flooding. Οι υπερκόμβοι λύνουν αυτό το πρόβλημα είτε διατηρώντας ένα ευρετήριο με τους πόρους και τους peer του

δικτύου είτε λειτουργώντας ως διαμεσολαβητές στη δρομολόγηση των μηνυμάτων (broker). Αφού είναι και αυτοί μέρος του δικτύου, τότε το σύστημα οδηγείται σε μια ιεραρχική δομή.

2.2 Το P-Grid Σύστημα

Το P-Grid [4--11] είναι ένα πλήρως αποκεντρωμένο και δομημένο (structured) peer-to-peer σύστημα. Χρησιμοποιεί τυχαιοκρατικούς αλγορίθμους για την κατασκευή και την λειτουργία του. Η τοπολογία που δημιουργείται είναι ένα δυαδικό trie δέντρο και χρησιμοποιούνται προθέματα κλειδιών για την δρομολόγηση των μηνυμάτων μέσα στο δίκτυο.

Οι peer αποφασίζουν κατά ζεύγη τον τρόπο με τον οποίο θα χωρίσουν το χώρο κλειδιών (key space). Κάθε κατάτμηση του χώρου αντιστοιχεί σε μια συγκεκριμένη δυαδική συμβολοσειρά ή οποία αποτελεί και το μονοπάτι ενός peer. Στο σχήμα 2.1 φαίνεται η τελική διαμόρφωση του χώρου κλειδιών και της τοπολογίας που έχουν σχηματίσει οι peer. Το δίκτυο αποτελείται από τέσσερις peer κάθε ένας εκ των οποίων έχει συσχετιστεί με ένα μονοπάτι. Η ρίζα του δέντρου αποτελεί το συνολικό χώρο των κλειδιών. Οι διακλαδώσεις σε κάθε ύψος αναπαριστούν κατατμήσεις του χώρου ένα ψηφίο τη φορά. Παράδειγμα, μετά τη ρίζα, έχουν δημιουργηθεί δύο μεγάλα υπόδεντρα. Το αριστερό περιλαμβάνει όλα τα κλειδιά τα οποία έχουν πρόθεμα το *'0'*, ενώ το αριστερό αυτά που έχουν πρόθεμα το *'1'*. Συνεπώς, οι peer ανάλογα με το μονοπάτι με το οποίο έχουν συσχετιστεί, *γνωρίζουν* ένα συγκεκριμένο πλήθος κλειδιών.



Σχήμα 2.1: Η τοπολογία ενός δικτύου P-Grid

Για κάθε επίπεδο του δέντρου, αποθηκεύονται αναφορές στον πίνακα δρομολόγησης προς τους peer των οποίων το μονοπάτι ανήκει στο συζυγές υπόδεντρο. Παράδειγμα, ο peer A έχει μονοπάτι *'00'* και επομένως θα αποθηκεύσει αναφορές για το επίπεδο με μονοπάτι *'0'* έναν peer από το συζυγές του που είναι το υπόδεντρο με μονοπάτι *'1'*. Αντίστοιχα για το *'00'* είναι το *'01'*. Στο σχήμα 2.1 έχει συσχετίσει το *'1'* με τον peer C και το *'01'* με τον B. Η επιλογή των peer που θα

επιλέξει να κρατήσει αναφορές είναι τυχαία. Επίσης, οι peer που αποθηκεύονται ανά επίπεδο μπορεί να είναι περισσότεροι του ενός. Βάσει αυτής της τοπολογίας, μια αναζήτηση θα κάνει O(logN) βήματα μέχρι να βρεθεί το αποτέλεσμα της. Άλλο πλεονέκτημα είναι και η δυνατότητα εκτέλεσης αναζητήσεων με δεδομένο ένα εύρος κλειδιών (key range).

Όσον αφορά τα δεδομένα που μπορεί να αποθηκεύσει κάθε peer, ακολουθείται η ίδια λογική. Υπάρχει μια συνάρτηση κατακερματισμού η οποία αντιστοιχεί κλειδιά σε δεδομένα. Κάθε peer είναι υπεύθυνος για εκείνο το κομμάτι των δεδομένων που έχει πρόθεμα το μονοπάτι του.

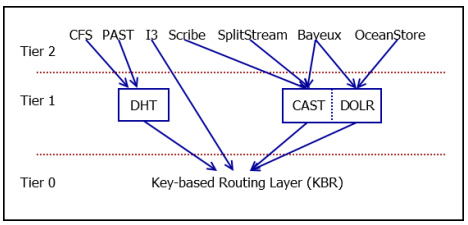
Τέλος, το σύστημα υποστηρίζει διάφορα πρωτόκολλα. Αυτό που μας ενδιαφέρει στην παρούσα εργασία είναι αυτό της ανοχής λαθών (fault-tolerance). Αυτό που προτείνεται και είναι ήδη υλοποιημένο είναι η τεχνική της αντιγραφής (replication). Κάθε μονοπάτι του δέντρου αντιστοιχεί σε πολλαπλούς peer. Αυξάνει τη γενική αντοχή του συστήματος σε αποτυχίες peer. Το κόστος είναι η μείωση της χωρητικότητας του δικτύου, αφού μέρος των peer είναι αντίγραφα άλλων. Επίσης γίνεται πιο πολύπλοκη η διατήρηση και ο συγχρονισμός του συστήματος.

Κεφάλαιο 3

Σχετική Εργασία και Θεωρία στην Αρχιτεκτονική

3.1 Σχετική Εργασία

Τα τελευταία χρόνια γίνεται μια προσπάθεια να τυποποιηθούν τα peer-to-peer συστήματα σε μια ενιαία αρχιτεκτονική. Πιο γενικά χρειάζεται ένα μοντέλο αναφοράς όπου περιγράφονται βασικές οντότητες και αλληλεπιδράσεις και βάσει αυτού μπορούν να κατασκευαστούν τέτοια συστήματα.

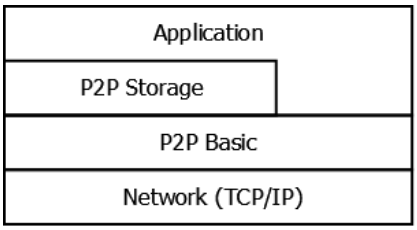


Σχήμα 3.1: Towards a common API

Στη δημοσίευση [13] έχουμε το διαχωρισμό του συστήματος σε τρία επίπεδα. Το πρώτο επίπεδο αφορά τη δρομολόγηση, η οποία βασίζεται σε κλειδιά (key-based routing). Πάνω σε αυτό χτίζονται τα πρωτόκολλα του συστήματος, όπως κατανεμημένοι πίνακες κατακερματισμού (DHT), αποκεντρωμένη τοποθεσία αντικειμένου και δρομολόγησης (Decentralized Object Location and Routing) καθώς και anycast/multicast λειτουργικότητα. Οι εφαρμογές έχουν πρόσβαση που αποτελούν το ανώτερο αρχιτεκτονικό επίπεδο, έχουν πρόσβαση απευθείας στα δυο κατώτερα επίπεδα. Οι διεπαφές που προτείνονται όμως δεν μπορούν να καλύψουν όλες τις ανάγκες. Στο σχήμα 3.1 φαίνονται τα επίπεδα της προτεινόμενης αρχιτεκτονικής.

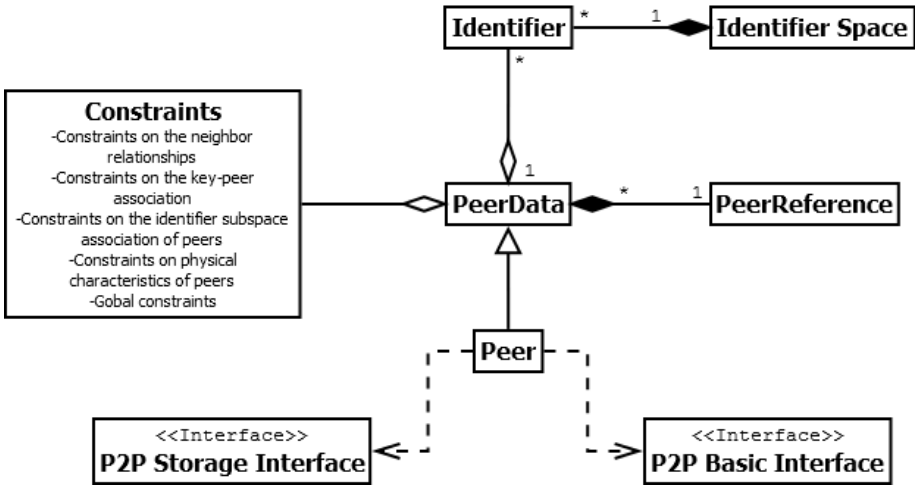
Ο εμπνευστής του P-Grid συστήματος με την ομάδα του εισάγει ένα μοντέλο αναφοράς [14] στο οποίο μπορούν να σχεδιαστούν διάφορα peer-to-peer συστήματα. Στο σχήμα 3.2 παρουσιάζονται τα αρχιτεκτονικά επίπεδα του μοντέλου. Στηρίζεται πάνω στο επίπεδο του δικτύου το οποίο συγκεντρώνει τα πρωτόκολλα TCP/IP. Αυτό παρέχει τις λειτουργίες του στο βασικό επίπεδο (P2P basic) το οποίο και υλοποιεί το peer-to-peer δίκτυο. Οι λειτουργίες που παρέχει εί-

ναι η σύνδεση (join) και η αποχώρηση (leave) στο δίκτυο, αναζήτησης (lookup) και δρομολόγησης (route). Στη συνέχεια έχουμε το επίπεδο αποθήκευσης (P2P storage) που παρέχει λειτουργίες εισαγωγής (insert), διαγραφής (delete), ανανέωσης (update) δεδομένων στο δίκτυο καθώς και τη διενέργεια ερωτήσεων πάνω σε αυτά (query). Για την ανάπτυξη μιας εφαρμογής, χρησιμοποιούνται οι διεπαφές που προσφέρονται από το βασικό επίπεδο και αυτό της αποθήκευσης. Στο σχήμα 3.3 παρουσιάζεται το προτεινόμενο βασικό μοντέλο.



Σχήμα 3.2: Προτεινόμενη αρχιτεκτονική από Aberer et al.

Σε αυτό φαίνονται τα απαραίτητα συστατικά που προκύπτουν από την ανάλυση, οι διάφοροι περιορισμοί που τίθενται και οι διεπαφές που πρέπει να υποστηρίζει κάθε peer-to-peer σύστημα. Στην υλοποίηση του P-Grid [1] από τον Schmidt [15], ακολουθήθηκε το παραπάνω μοντέλο. Η όλη λογική είναι ένα peer-to-peer σύστημα να είναι χαλαρά συνδεδεμένο με τις υπηρεσίες που προσφέρουν το βασικό επίπεδο και αυτό της αποθήκευσης. Στόχος είναι η ευκολία αλλαγής συμπεριφοράς του ίδιου συστήματος εναλλάσσοντας τις διάφορες υλοποιήσεις. Παρόλα αυτά όπως προκύπτει από την εξέταση της υλοποίησης, είναι δύσκολο να επεκταθεί μια υπάρχουσα υλοποίηση με άλλα πρωτόκολλα πιο πολύπλοκα.

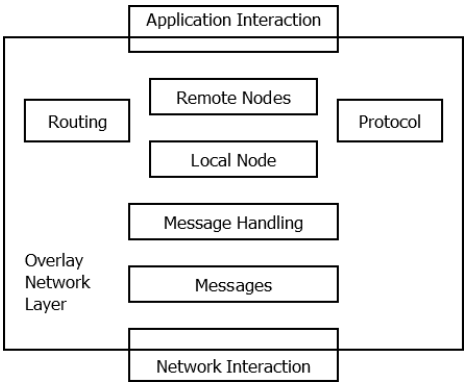


Σχήμα 3.3: Προτεινόμενο μοντέλο αναφοράς από Aberer et al.

Για την εσωτερική δομή επιπέδων, όπως αυτό του βασικού της παραπάνω αρχιτεκτονικής , συντελείται μια προσπάθεια να δημιουργηθούν σχεδιαστικά πρό-

[1] http://www.p-grid.org

τυπα. Στις δημοσιεύσεις [16, 17] αναλύονται ένα σύνολο peer-to-peer συστημάτων και ενδιάμεσου λογισμικού. Από αυτό προκύπτουν σχεδιαστικά πρότυπα που έχουν να κάνουν με ένα βασικό στοιχείο αυτών των συστημάτων που είναι η δρομολόγηση των μηνυμάτων μέσα στο δίκτυο. Στο σχήμα 3.4 φαίνεται η αρχιτεκτονική δομή του συγκεκριμένου επιπέδου. Κάθε στοιχείο του επιπέδου περιλαμβάνει μια σειρά σχεδιαστικών προτύπων.



Σχήμα 3.4: A Pattern Language

Στη δημοσίευση [18] εισάγεται η έννοια του Peer ως σχεδιαστικό πρότυπο. Το πρόβλημα που προσπαθεί να επιλυθεί είναι η αποτελεσματική κατανομή του φόρτου εργασίας και η υψηλή διαθεσιμότητα των πόρων. Η λογική είναι παρόμοια, μόνο που πλέον οι πόροι και οι λειτουργίες που προσφέρει ένας peer παρουσιάζονται ως υπηρεσίες. Ενδιαφέρει ο τρόπος με τον οποίο περιγράφονται οι υπηρεσίες και προτείνεται η ύπαρξη είτε μιας WSDL διεπαφής όπου μπορεί να ανακτηθεί πλήρως η λειτουργικότητά της είτε OWL-S περιγραφών σε περίπτωση που είναι αναγκαία η σημασιολογική περιγραφή (semantic) της πληροφορίας. Προτείνεται, μάλιστα, η χρήση Web Service αρχιτεκτονικής δεδομένου των πλεονεκτημάτων που μπορεί να φέρει μια τέτοια υπηρεσία.

Από την πρακτική εφαρμογή του προτύπου προκύπτει ένα πλαίσιο ανάπτυξης peer-to-peer συστημάτων, το SP2A [19]. Χρησιμοποιήθηκαν τεχνολογίες όπως η JXTA και Web Services. Για την ανάπτυξη των τελευταίων χρησιμοποιήθηκαν τα JXTA-SOAP και OWL-S. Η περιγραφή των web service μέσω OWL-S κάνει εύκολη την ανακάλυψη, την εκτέλεση, την σύνθεση αυτών και την παρακολούθησή τους (monitoring). Παρόλα αυτά, το συγκεκριμένο framework υποστηρίζει υβριδικά και καθαρά peer-to-peer δίκτυα χωρίς όμως να υπάρχει συγκεκριμένη τοπολογία. Επίσης, δεν υπάρχει η έννοια της σύνθεσης των υπηρεσιών στο ίδιο το λογισμικό του peer σε πιο πολύπλοκες διαδικασίες/πρωτόκολλα.

Σημαντική δουλειά είναι τα JXTA πρωτόκολλα [20]. Αυτά αποτελούν μια ανοιχτή διαδικτυακή πλατφόρμα για peer-to-peer συστήματα. Τυποποιείται ο τρόπος με τον οποίο ένας peer ανακαλύπτει άλλους, το πώς οργανώνεται σε ομάδες peer και πώς επικοινωνεί. Επίσης, ορίζονται πρωτόκολλα για τη διαφήμιση και ανακάλυψη διαδικτυακών πόρων. Υπάρχουν τρία αρχιτεκτονικά επίπεδα, με πρώτο το επίπεδο της πλατφόρμας όπου ορίζονται κλασικοί τύποι για ένα peer-to-peer δίκτυο. Στη συνέχεια, έχουμε το επίπεδο των υπηρεσιών το οποίο περιλαμβάνει διαδικτυακές υπηρεσίες, όπως αναζήτησης και ευρετηριοποίησης, αποθηκευτικού χώρου, διαμοιρασμού αρχείων, κατανεμημένα συστήματα αρχείων και άλλα. Το τελικό επίπεδο είναι αυτό της εφαρμογής που εκμεταλλεύεται τις προσφερόμενες υπηρεσίες. Δεν υπάρχει όμως καθορισμός της τοπολογίας των peer-to-peer δικτύων

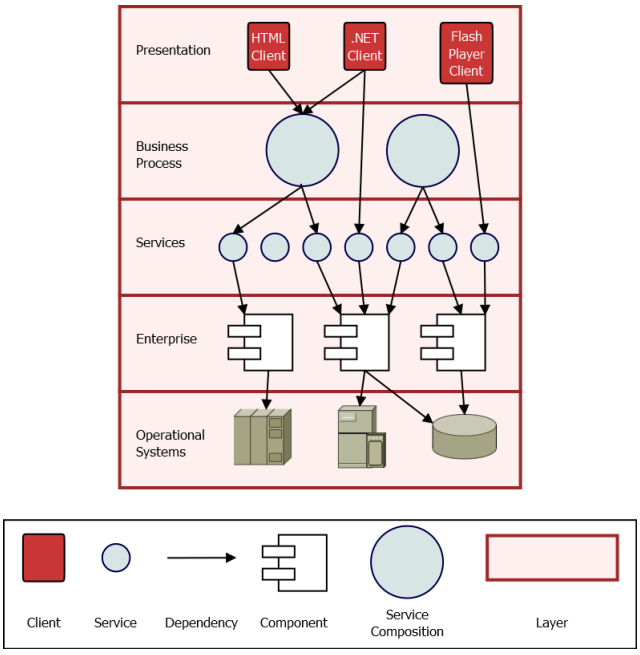
που προκύπτουν. Επιπλέον, τα πρωτόκολλα είναι ορισμένα ως ένα σύνολο από XML μηνύματα προκειμένης της ανεξαρτητοποίησης από λειτουργικά συστήματα και γλώσσες προγραμματισμού.

3.2 Υπηρεσιοκεντρική Αρχιτεκτονική (Service - Oriented Architecture - SOA)

3.2.1 Η Υπηρεσία ως Δομικό Στοιχείο

Η αρχιτεκτονική που εισάγουμε έχει επηρεαστεί από τα συστήματα που ακολουθούν το μοντέλο της υπηρεσιοκεντρούς αρχιτεκτονικής (SOA). Θα ακολουθήσει μια μικρή ανάλυση τέτοιων συστημάτων και των χαρακτηριστικών τους πριν την εξήγηση της αρχιτεκτονικής. Θα δούμε γιατί η έννοια της υπηρεσίας και της θεωρίας που έχει αναπτυχθεί είναι χρήσιμη και ποιά είναι τα πλεονεκτήματα που απορρέουν από μια τέτοια σχεδίαση. Το υπηρεσιοκεντρικό μοντέλο είναι ένα πρότυπο για την οργάνωση και τη χρήση κατανεμημένων λειτουργιών [21].

Στο σχήμα 3.5 παρουσιάζονται τα αρχιτεκτονικά επίπεδα ενός υπηρεσιοκεντρικού συστήματος [22]. Τα επίπεδα όπως παρουσιάζονται είναι:



Σχήμα 3.5: Αρχιτεκτονικά επίπεδα ενός υπηρεσιοκεντρικού συστήματος

1. **Business Process.** Οι υπηρεσίες που παρέχονται στο επίπεδο των υπηρεσιών, συνήθως συντίθενται σε ροές εργασίας (workflow) και βοηθούν στην ανάπτυξη εφαρμογών.

2. **Services.** Στο επίπεδο αυτό υπάρχουν οι υπηρεσίες. Η κλήση των υπηρεσιών καθορίζεται είτε κατά τη διάρκεια της σχεδίασης είτε μέσω μιας υπηρεσίας μητρώου (service registry) κατά τη διάρκεια εκτέλεσης.

3. **Enterprise.** Τα διάφορα κομμάτια που ανήκουν στο επίπεδο περιέχουν κώδικα που εξυπηρετεί τις ανάγκες των υπηρεσιών. Επίσης, παρέχεται πρόσβαση στα συστήματα που βρίσκονται στο κατώτερο επίπεδο.

4. **Operational Systems.** Εδώ ανήκουν όλα τα λειτουργικά εμπορικά και μη συστήματα.

Η δομική μονάδα του μοντέλου είναι η υπηρεσία. Η υπηρεσία είναι ένας μηχανισμός με τον οποίο επιτρέπεται πρόσβαση σε μια ή περισσότερες λειτουργίες. Η πρόσβαση παρέχεται μέσω μιας προκαθορισμένης διεπαφής και χρησιμοποιείται σύμφωνα με τους περιορισμούς και τις πολιτικές που ορίζονται από την περιγραφή της υπηρεσίας. Σύμφωνα με το αρχιτεκτονικό μοντέλο της OASIS [23] έχουμε διάφορους ρόλους σε μια υπηρεσία. Έχουμε αυτόν που προσφέρει την υπηρεσία και ονομάζεται Service Provider. Υπάρχει αυτός που ζητά την υπηρεσία για να την χρησιμοποιήσει/καταναλώσει και ονομάζεται Service Consumer. Έχουμε τον ρόλο του διαμεσολαβητή ο οποίος διευκολύνει την αλληλεπίδραση και τη συνδεσιμότητα στην προσφορά και χρήση της υπηρεσίας και ονομάζεται Service Mediator. Καταλήγοντας, έχουμε τον κάτοχο της υπηρεσίας ο οποίος ονομάζεται Service Owner.

Η κλήση μιας υπηρεσίας έχει το αποτέλεσμα είτε της επιστροφής πληροφορίας αν η αίτηση είχε τέτοιο σκοπό είτε της αλλαγής της κατάστασης του συστήματος. Σε καμία περίπτωση τον καταναλωτή της υπηρεσίας δεν τον ενδιαφέρει ο τρόπος παραγωγής της πληροφορίας ή πως έγινε η αλλαγή της κατάστασης. Επιπλέον, δεν γνωρίζει πώς υλοποιείται η υπηρεσία που χρησιμοποιεί. Παρακάτω αναφέρονται οι γενικές σχεδιαστικές αρχές που διέπουν μια υπηρεσία.

3.2.2 Χαρακτηριστικά Υπηρεσιών

Συνολικά έχουμε δυο κατηγορίες αρχών [24]. Η πρώτη περιλαμβάνει τις αρχές που αφορούν τα σχεδιαστικά χαρακτηριστικά των υπηρεσιών. Αυτά είναι:

1. Τυποποιημένη σύμβαση υπηρεσίας (service contract)

2. Επαναχρησιμοποίηση υπηρεσίας (Service reusability)

3. Αυτονομία υπηρεσίας (Service autonomy)

4. Υπηρεσία χωρίς κατάσταση (Service statelessness)

5. Δυνατότητα ανακάλυψης υπηρεσίας (Service discoverability)

Η δεύτερη κατηγορία περιλαμβάνει τις αρχές που ρυθμίζουν τον τρόπο με τον οποίο θα εφαρμοστούν οι παραπάνω αρχές. Αυτές είναι:

1. Χαλαρή σύνδεση (Service loose coupling)
2. Αφαιρετικότητα (Service abstraction)
3. Δυνατότητα σύνθεσης (Service composability).

Στη συνέχεια αναλύονται εκείνες οι αρχές που έχουν νόημα για το επίπεδο και το μέγεθος του συστήματος που χτίζουμε.

Τυποποιημένη σύμβαση υπηρεσίας

Είναι πολύ σημαντικό και αναγκαίο η ύπαρξη σαφών και καλώς ορισμένων συμβάσεων και διεπαφών που ενθυλακώνουν τα στοιχεία και τις λειτουργίες του συστήματος. Οι πόροι που προσφέρονται από μια υπηρεσία πρέπει να διαχειρίζονται μέσω αυτών των συμβάσεων. Δεν πρέπει δηλαδή να υπάρχει απευθείας σύνδεση των καταναλωτών της υπηρεσίας με τους υποκείμενους πόρους της. Η ανάγκη μας οδηγεί σε αυτή την αρχή είναι η χαλαρή συνδεσιμότητα μεταξύ της υπηρεσίας και του χρήστη της.

Ο καταναλωτής της υπηρεσίας κατ' επέκταση δεσμεύεται από την σύμβαση αυτή προκειμένου να έχει πρόσβαση στην υπηρεσία. Η δέσμευση υπάρχει σε τέσσερα σημεία:

1. Την περιγραφή των λειτουργιών που προφέρει η υπηρεσία.
2. Την περιγραφή του μοντέλου των δεδομένων που θα χρησιμοποιηθεί για την ανταλλαγή δεδομένων μεταξύ υπηρεσίας και χρήστη.
3. Θέματα που αφορούν την ποιότητα των παρεχόμενων υπηρεσιών (QoS)
4. Τον καθορισμό του τρόπου σύνδεσης του χρήστη με την υπηρεσία.

Επαναχρησιμοποίηση υπηρεσίας

Το να είναι επαναχρησιμοποιήσιμη μια μονάδα λογισμικού είναι πολύ σημαντικό χαρακτηριστικό. Τα κατάλληλα επίπεδα αφαιρετικότητας βοηθούν στο να επιτευχθεί η αρχή αυτή.

Αυτονομία υπηρεσίας

Οι υπηρεσίες που προσφέρει το σύστημα πρέπει να είναι αυτόνομες μεταξύ τους. Η σχεδίαση αυτών πρέπει να γίνει με τέτοιο τρόπο ώστε να αποφευχθεί οποιαδήποτε επικάλυψη λειτουργικότητας μεταξύ τους. Επίσης, οι υπηρεσίες πρέπει να είναι αξιόπιστες ώστε να έχει νόημα η επαναχρησιμοποίηση τους. Για να επιτευχθεί αυτό η υπηρεσία έχει τον πλήρη έλεγχο στο πώς λειτουργεί. Εξωτερικοί-απομακρυσμένοι πόροι στους οποίους δεν υπάρχει έλεγχος δεν πρέπει να επηρεάζουν αρνητικά.

Υπηρεσία χωρίς κατάσταση

Το να μην αποθηκεύεται κατάσταση σε μια υπηρεσία είναι θεμιτό αφού προωθεί την επαναχρησιμοποίηση της.

Χαλαρή σύνδεση

Τα κίνητρα πίσω από αυτήν την αρχή είναι η δυνατότητα επαναχρησιμοποίησης και συντήρησης. Ανεξάρτητες μονάδες λογισμικού καλύπτουν εύκολα αυτά τα χαρακτηριστικά. Όσο πιο σφιχτά συνδεδεμένα είναι τα στοιχεία του συστήματος όσον αφορά λειτουργικές εξαρτήσεις, τόσο περισσότερο αυξάνει η πιθανότητα να εξαπλωθεί και να επηρεάσει μια αλλαγή [25]. Το μειονέκτημα που εισάγεται όμως, είναι η προσθήκη επιπλέον επιπέδων αφαιρετικότητας με τις όποιες συνέπειες που έχει αυτό στην απόδοση του συστήματος.

Αφαιρετικότητα

Για να υλοποιηθεί σωστά η σύμβαση που προσφέρει η υπηρεσία, η σχεδίαση πρέπει να έχει τα απαραίτητα επίπεδα αφαιρετικότητας. Παράδειγμα, στη σύνδεση μεταξύ του παρόχου μιας υπηρεσίας και του καταναλωτή της διαμεσολαβεί η σύμβαση/διεπαφή αυτής. Αυτό έχει ως συνέπεια την απομόνωση εκείνου του κομματιού της υπηρεσίας, όπως είναι η υλοποίησή της, το οποίο δεν αποτελεί χρήσιμη γνώση στον χρήστη της.

Υπάρχουν και άλλα σημεία στο σύστημα που επωφελούνται από αυτή την αρχή. Η πλατφόρμα, τα διάφορα πρωτόκολλα, τα δεδομένα και η δομή τους αποτελούν όλα διαφορετικά επίπεδα αφαρετικότητας.

Δυνατότητα σύνθεσης

Οι υπηρεσίες είναι σχεδιασμένες έτσι ώστε να μπορούν να συνδυαστούν και να προκύψουν διαδικασίες που έχουν νόημα για την εφαρμογή. Στα υπηρεσιοκετρικά συστήματα η σύνθεση μπορεί να επιτευχθεί με την ενορχήστρωσης όπου υπάρχει ένας καθοδηγητής που καθορίζει τον ρόλο της κάθε υπηρεσίας. Μπορεί να γίνει επίσης με την χορογραφία όπου κάθε υπηρεσία παίζει τον ρόλο της παράγοντας και εκδίδοντας γεγονότα που προκύπτουν κατά την εκτέλεσή της και λαμβάνει τα απαραίτητα γεγονότα για να επιτελέσει τον ρόλο της.

Κεφάλαιο 4

Γενική Δομή Αρχιτεκτονικής

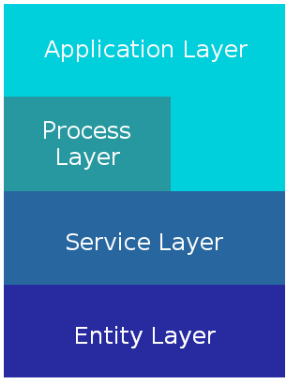
4.1 Επίπεδα Συστήματος

Με οδηγό το υπηρεσιοκεντρικό μοντέλο που περιγράφηκε στην ενότητα 3.2, το σύστημα δομείται σε αρχιτεκτονικά επίπεδα. Όπως αναφέρθηκε ένα SOA σύστημα χωρίζεται σε:

1. **Presentation.** Αφορά εφαρμογές που χρησιμοποιούν το σύστημα.
2. **Business Process.** Εκτελεί συγκεκριμένη λειτουργία που έχει να κάνει με τον τύπο της εφαρμογής και συντίθεται από τις υπάρχουσες υπηρεσίες.
3. **Services.** Συμπεριλαμβάνει όλες τις υπηρεσίες.
4. **Enterprise.** Εξυπηρετεί απευθείας τις ανάγκες των υπηρεσιών και προσφέρει πρόσβαση στα συστήματα της επιχείρησης.
5. **Operational Systems.** Αποτελείται από τις υπάρχουσα εμπορικά και μη εξωτερικά συστήματα ή κάποιο συνδυασμό αυτών.

Με βάση την παραπάνω κατηγοριοποίηση το σύστημα χωρίζεται σε τέσσερα επίπεδα. Ξεκινώντας από την κορυφή όπως φαίνεται στο σχήμα 4.1 έχουμε:

- **Επίπεδο Εφαρμογής (Application Layer).** Το επίπεδο αυτό περιλαμβάνει την λογική οποιασδήποτε εφαρμογής που χρησιμοποιήσει την παρούσα αρχιτεκτονική. Δεν αφορά την προκείμενη αρχιτεκτονική.
- **Επίπεδο Διαδικασιών (Process Layer).** Το επίπεδο αυτό περιέχει όλες τις διαδικασίες που στην ουσία είναι συνδυασμός υπηρεσιών ώστε να παράγουν ουσιώδεις λειτουργίες προς διευκόλυνση της ανάπτυξης εφαρμογών. Στην ουσία οι διαδικασίες είναι κι αυτές υπηρεσίες.
- **Επίπεδο Υπηρεσιών (Service Layer).** Το επίπεδο των υπηρεσιών περιλαμβάνει ένα σύνολο από διαφορετικές υπηρεσίες. Με τον όρο υπηρεσία (service)



Σχήμα 4.1: Τα επίπεδα της αρχιτεκτονικής

αναφερόμαστε σε όλες τις σαφώς οριζόμενες και αυτόνομες λειτουργίες που έχουν νόημα για ένα peer-to-peer σύστημα. Είναι στην ουσία αναπαράσταση αλγορίθμων, πρωτοκόλλων και λειτουργικότητας που προσφέρει το σύστημα. Μπορούμε να το δούμε ως μια ενέργεια που επιδρά πάνω στο επίπεδο των οντοτήτων. Απόρροια αυτού μπορεί να είναι η ανανέωση της αποθηκευμένης πληροφορίας όχι μόνο τοπικά σε κάποιον peer αλλά και συνολικά στο σύστημα. Μπορεί να οδηγήσει στην αλλαγή της κατάστασης του τοπικού peer ακόμα και μέρους του δικτύου. Επίσης, μπορεί να επιστρέψει χρήσιμες πληροφορίες για το σύστημα ή για τα δεδομένα που αποθηκεύονται σε αυτό. Η υπηρεσία είναι το δομικό στοιχείο της αρχιτεκτονικής βάσει της οποίας ο χρήστης μπορεί είτε να δομήσει την εφαρμογή του είτε να προσθέσει επιπλέον λειτουργικότητα στο σύστημα.

- **Επίπεδο Οντοτήτων (Entity Layer).** Το επίπεδο των οντοτήτων είναι το κατώτερο επίπεδο της αρχιτεκτονικής πάνω στο οποίο επιδρούν και στηρίζονται όλα τα υπόλοιπα. Με τον όρο οντότητα ονομάζουμε όλα τα αντικείμενα τα οποία διατηρούν κατάσταση ή μπορούν να προσφέρουν πρόσβαση σε πόρους του συστήματος. Επίσης συμπεριλαμβάνονται και αυτά που προσφέρουν λειτουργικότητα από το λειτουργικό σύστημα στο οποίο τρέχει η εφαρμογή. Παράδειγμα είναι η ικανότητα επικοινωνίας με άλλες εφαρμογές μέσω απομακρυσμένων κλήσεων συναρτήσεων (rpc) ή με τη χρήση socket.

4.2 Σχεδιαστικό Πρότυπο Dependency Injection

4.2.1 Εξήγηση προτύπου

Η αρχιτεκτονική έχει χωριστεί σε διακριτά επίπεδα. Κάθε επίπεδο περιέχει τα δικά του domain object τα οποία ενθυλακώνουν σαφώς ορισμένη και ξεχωριστή λειτουργικότητα [26]. Τα domain object κατηγοριοποιούνται περισσότερο, με δυο κατηγορίες να συμπίπτουν με την παρούσα αρχιτεκτονική. Αυτές είναι το application service το οποίο έχει την ίδια έννοια με την υπηρεσία όπως έχει οριστεί εδώ και το component το οποίο αντιστοιχεί στις οντότητες του επιπέδου οντοτήτων όπως περιγράφηκε. Όλα αυτά θέτουν κάποια κοινά προβλήματα, όπως ο τρόπος σύνδεσης διεπαφών και υλοποιήσεων, ο τρόπος που θα κατασκευάζονται, ο έλεγχος του κύκλου ζωής και η παραμετροποίηση τους είτε κατά την εκκίνηση του συστήματος είτε κατά τον χρόνο εκτέλεσης. Λύσεις σε αυτά τα ζητήματα δίνονται μεμονωμένα από διάφορα σχεδιαστικά πρότυπα. Παρακάτω δικαιολογείται η επιλογή του σχεδιαστικού προτύπου Dependency Injection [27] για την λύση των παραπάνω ζητημάτων στο σύνολό τους.

Μια κλάση μπορεί να χρησιμοποιεί διάφορες διεπαφές οι οποίες με τη σειρά τους έχουν κάποιες υλοποιήσεις. Η βασική ιδέα του σχεδιαστικού προτύπου dependency injection είναι η ύπαρξη ενός ξεχωριστού αντικειμένου που είναι υπεύθυνο να προσφέρει στις κλάσεις υλοποιήσεις των διεπαφών που χρησιμοποιούν. Προγραμματιστικά σημαίνει πως η κλάση δεν περιέχει κώδικα που αρχικοποιεί αντικείμενα από τα οποία εξαρτάται. Η λειτουργία αυτή έχει μεταφερθεί στο ξεχωριστό αντικείμενο που αναφέρθηκε και το οποίο ονομάζεται dependency injector. Το αποτέλεσμα που προκύπτει είναι η αφαίρεση των εξαρτήσεων από τις υλοποιήσεις των διεπαφών που χρησιμοποιεί μια κλάση. Οι εξαρτήσεις πλέον γίνονται εμφανείς ως ορίσματα στον constructor ή σε μεθόδους που αρχικοποιούν τα πεδία της κλάσης (getter/setter μέθοδοι). Ο injector, επομένως, κατασκευάζει το ζητούμενο αντικείμενο ικανοποιώντας όλες τις εξαρτήσεις του.

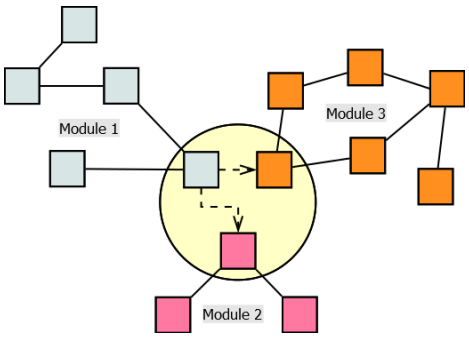
Αντίστοιχη λύση μπορεί να δοθεί και από το σχεδιαστικό πρότυπο Service Locator [28] που είναι μια παραλλαγή του προτύπου Inversion of Control. Παρέχεται ένα γενικό σημείο πρόσβασης προς όλες τις υπηρεσίες χωρίς να ενδιαφέρει η υλοποίηση που χρησιμοποιείται και ο τρόπος με τον οποίο αναζητείται από τον service locator. Η διαφορά και το μειονέκτημα σε σύγκριση με το πρότυπο Dependency Injection είναι πως η μοναδική εξάρτηση που έχει πλέον μια κλάση είναι το αντικείμενο service locator.

Λαμβάνουμε διάφορα οφέλη από την χρήση του σχεδιαστικού προτύπου Dependency Injection. Ο κώδικας χωρίζεται σε ευδιάκριτες μονάδες (module) και έχουμε πλήρη διαχωρισμό διεπαφών και υλοποιήσεων. Οι κλάσεις δεν γνωρίζουν τίποτα για τις υλοποιήσεις των διεπαφών που χρησιμοποιούν. Διαχωρίζεται δηλαδή, η συμπεριφορά και η λειτουργικότητα που προσφέρεται από την λύση των εξαρτήσεων. Επίσης γίνεται περισσότερο επαναχρησιμοποιήσιμος (reusable) και είναι πιο ευανάγνωστος αφού οι εξαρτήσεις είναι ορατές. Τέλος, διευκολύνεται ο έλεγχος ορθής λειτουργίας του κώδικα όπως η χρήση Mock Object [29] το

οποίο παρέχεται στη θέση μιας πραγματικής υλοποίησης μιας διεπαφής από την οποία εξαρτάται η κλάση.

4.2.2 Guice

Στην υλοποίηση της βιβλιοθήκης χρησιμοποιήθηκε η Guice, ένα ελαφρύ πλαίσιο που υλοποιεί το πρότυπο dependency injection για την Java κατασκευασμένο από την Google. Όλες οι πληροφορίες παραμετροποίησης των συνδέσεων μεταξύ διεπαφών και υλοποιήσεων περιέχονται κυρίως σε ένα Module. Γενικά μια εφαρμογή που ακολουθεί αυτό το πρότυπο αποτελείται από διάφορα module και κώδικα που αφορά την αρχικοποίηση της.



Σχήμα 4.2: Σύνθεση module μέσω της Guice

Η κλάση που είναι υπεύθυνη για την κατασκευή των αντικειμένων είναι ο Injector. Για την αρχικοποίησή του χρειάζονται τα module. Σε γενικές γραμμές, όταν ζητηθεί η κατασκευή ενός αντικειμένου ενός τύπου, αρχικά βρίσκει τι να κατασκευάσει αφού μπορεί να υπάρχουν πολλές υλοποιήσεις. Στη συνέχεια λύνει τις εξαρτήσεις και αρχικοποιεί το αντικείμενο.

Κατά την εκκίνηση του συστήματος, όλη η παραμετροποίηση όσον αφορά τις υλοποιήσεις που θα χρησιμοποιηθούν παρέχεται από τα module όπως περιγράφηκε. Επιπλέον, έχουμε τη δυνατότητα αλλαγών κατά τη διάρκεια εκτέλεσης (runtime) του προγράμματος.

Ένα άλλο σημαντικό ζήτημα είναι και ο κύκλος ζωής των αντικειμένων που παράγονται. Αυτός ελέγχεται από τα λεγόμενα Scope. Αυτά περιγράφουν πόσο θα ζήσει ένα αντικείμενο κι αν είναι σε θέση να επαναχρησιμοποιηθεί. Οι λόγοι είναι επειδή είτε κρατάνε κατάσταση, είτε είναι ακριβά στην κατασκευή ή στη διατήρηση τους. Στην Guice προσφέρονται κάποια έτοιμα, τα οποία φαίνονται παρακάτω, αλλά παρέχεται και όλο το πλαίσιο να υλοποιηθούν νέα με την απαραίτητη λειτουργικότητα.

1. Unscoped: Το αντικείμενο παράγεται εκ νέου μετά από κάθε αίτηση κατασκευής στον injector.

2. Singleton: Το αντικείμενο που επιστρέφει ο injector είναι το ίδιο για όλη την εφαρμογή.

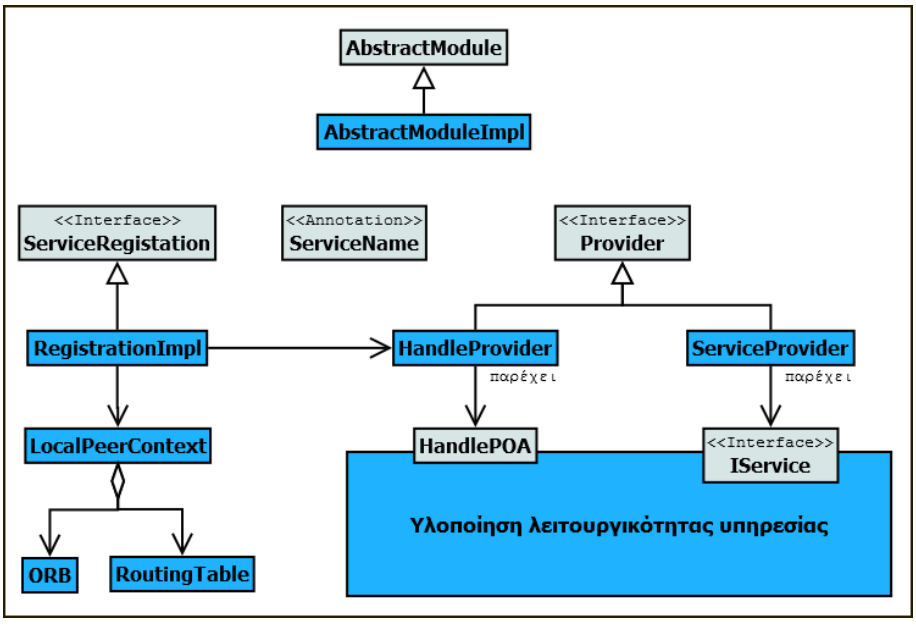
3. RequestScoped: Το αντικείμενο διαρκεί για μια web/rpc αίτηση (αφορά web εφαρμογές και servlet).

4. SessionScoped: Το αντικείμενο διαρκεί για μια HTTP συνεδρία. (αφορά web εφαρμογές και servlet).

4.3 Δομή Υπηρεσίας

Πλέον έχουμε την κατάλληλη βάση για την περιγραφή της δομής της υπηρεσίας που εισάγει η παρούσα αρχιτεκτονική. Οι αρχές που διέπουν τις υπηρεσίες και αναφέρθηκαν στην περιγραφή των υπηρεσιοκεντρικών συστημάτων είναι σημαντικό να εφαρμοστούν και στην παρούσα αρχιτεκτονική. Είναι απαραίτητο να υπάρχει μια ενιαία δομή αφού αυτό επιβάλλει μια ομοιομορφία στο επίπεδο. Το αποτέλεσμα είναι οι διάφορες υπηρεσίες να είναι εύκολα κατανοητές και το πιο σημαντικό είναι η ανάπτυξη ενός πλάνου για την υλοποίηση ή την τροποποίηση αυτών. Στο σχήμα 4.3 φαίνεται η δομή και ακολουθεί η εξήγηση καθώς και η αιτιολόγηση της ύπαρξης των στοιχείων από τα οποία αποτελείται.

Να σημειωθεί πως για την κλήση απομακρυσμένων συναρτήσεων έχει χρησιμοποιηθεί η CORBA προσθέτοντας κάποιες απαιτήσεις όσον αφορά την δομή της υπηρεσίας. Αυτό δε μας περιορίζει στο να αντικατασταθεί με κάποια άλλη βιβλιοθήκη. Οι αλλαγές που θα προκύψουν θα έχουν να κάνουν μόνο με τις διεπαφές και τις κλάσεις της CORBA.



Σχήμα 4.3: Γενική δομή υπηρεσίας

Διεπαφή Υπηρεσίας. Όπως αναφέρθηκε στην ανάλυση των SOA συστημάτων, η υπηρεσία προσφέρει πρόσβαση στον χρήστη σε μια ή περισσότερες λειτουργίες.

Είναι φυσικό να υπάρχει η διεπαφή που καθορίζει τις ενέργειες που μπορούν να γίνουν. Μέσω αυτής της διεπαφής πρέπει να περιγράφεται όλο εκείνο το πλαίσιο στο οποίο εκτελείται η υπηρεσία εννοώντας τους περιορισμούς που τίθενται. Η διεπαφή IService του σχήματος 4.3 αντιπροσωπεύει μια τέτοια διεπαφή. Γύρω από αυτήν μπορεί να υπάρξουν και άλλες βοηθητικές κλάσεις ή διεπαφές. Για παράδειγμα είναι πιθανό κατά την εκτέλεση μιας υπηρεσίας να αλλάξει μονοπάτι ο peer. Αυτή η πληροφορία μπορεί να αφορά άμεσα την εφαρμογή και θα πρέπει ως αποτέλεσμα να υπάρξει συγκεκριμένη αντίδραση. Το πρόβλημα αυτό λύνεται με το σχεδιαστικό πρότυπο Observer [30] ή Listener όπως ονομάζεται αλλιώς στον κόσμο της Java.

Java IDL (Servant HandlePOA). Η δυνατότητα εκτέλεσης της υπηρεσίας ως απομακρυσμένης κλήσης συνάρτησης (rpc) ικανοποιείται από την CORBA. Κατ' επέκταση είναι αναγκαία η περιγραφή της υπηρεσίας σε IDL. Από αυτήν παράγονται οι απαραίτητοι τύποι. Ανάμεσα σε αυτούς είναι και ο Servant ο οποίος παρέχει την ίδια λειτουργικότητα με την υπηρεσία. Είναι εκείνο το αντικείμενο που θα κληθεί όταν ένας peer λάβει μια αίτηση εκτέλεσης της υπηρεσίας.

Διεπαφή Provider. Η κατασκευή αντικειμένων των υλοποιήσεων είτε της υπηρεσίας είτε του Servant μπορεί να είναι μια πολύπλοκη διαδικασία η οποία δε θα έπρεπε να απασχολεί τον χρήστη. Στην προκειμένη περίπτωση, η Guice έχει αναλάβει την κατασκευή τους. Οι κλάσεις που υλοποιούν την διεπαφή Provider είναι ικανές να παρέχουν συγκεκριμένα αντικείμενα. Περιγράφουν στην Guice τον τρόπο με τον οποίο κατασκευάζονται πολύπλοκοι τύποι. Επίσης, μέσω των Provider μπορούμε να ελέγξουμε τον κύκλο ζωής τον αντικειμένων που επιστρέφουν. Παράδειγμα η υλοποίηση του Servant δεν έχει νόημα να κατασκευάζεται συνεχώς εκ νέου. Έχει επιλεχθεί επομένως ο Provider του να συμπεριφέρεται ως singleton και έτσι να επιστρέφει το ίδιο αντικείμενο όταν του ζητείται. Σε αντίθεση με την υλοποίηση της διεπαφής της υπηρεσίας όπου εκεί μπορούμε να κατασκευάζουμε νέο κάθε φορά. Βέβαια υπάρχουν και περιπτώσεις όπου, όπως περιγράφηκε στην ενότητα του Dependency Injection, μπορεί να είναι ακριβό η συνεχής κατασκευή.

Εγκατάσταση υπηρεσίας. Τον χρήστη της υπηρεσίας δεν τον απασχολεί ο τρόπος με τον οποίο ένας peer ξεκινά να εξυπηρετεί απομακρυσμένες αιτήσεις μιας υπηρεσίας. Αυτό καλύπτεται από την διεπαφή ServiceRegistration. Στην παρούσα φάση όπου εξαρτόμαστε από την CORBA, τα βήματα είναι η δημιουργία ενός Servant της υπηρεσίας που θέλουμε και η εγκατάστασή του στον POA. Ο Portable Object Adapter είναι κομμάτι της CORBA και θα αναλυθεί στην ενότητα 6. Για να επιτευχθεί αυτό χρειάζεται μια αναφορά προς το ORB του peer. Αυτή μπορεί να ληφθεί μέσω του LocalPeerContext που κρατά πληροφορίες για τον τοπικό peer, αναφορές προς τον πίνακα δρομολόγησης του και το ORB του.

Η διεπαφή ServiceRegistration πρόκειται να έχει τόσες υλοποιήσεις όσες και οι υπηρεσίες του συστήματος. Είναι αναγκαίος ένας τρόπος ώστε η Guice να ξέρει

ποια να επιστρέφει όταν ζητείται μιας συγκεκριμένης υπηρεσίας. Αυτό επιτυγχάνεται με τη δημιουργία ενός annotation. Στην εικόνα 4.3 το ServiceName παίζει αυτόν τον ρόλο.

Λειτουργικότητα Υπηρεσίας. Μετά τον καθορισμό των παραπάνω έχουμε την υλοποίηση της υπηρεσίας όπου προσφέρεται η λειτουργικότητά της. Εδώ έχουμε τον χώρο για την δήλωση όλων των βοηθητικών διεπαφών και κλάσεων καθώς και των υλοποιήσεών του. Είναι προφανές πως η λογική που θα ακολουθηθεί εδώ κρίνει και κατά πόσο η υπηρεσία θα καταστεί επεκτάσιμη και τροποποιήσιμη. Στην εικόνα 4.3 φαίνεται αυτό το κομμάτι καθώς και οι διεπαφές που το καλύπτουν.

Παραμετροποίηση Module. Τελικά, καταλήγουμε να έχουμε υλοποιήσεις όλων των αναγκαίων κλάσεων και διεπαφών καθώς και εκείνου του κομματιού που έχει σχέση με την αυτή καθεαυτή λειτουργικότητα της υπηρεσίας. Αυτό που απομένει είναι η δήλωση όλων των αντιστοιχίσεων διεπαφών/κλάσεων με τις υλοποιήσεις τους ώστε να γνωρίζει η Guice τον τρόπο με τον οποίο θα παράγει αντικείμενα αυτών λύνοντας τις εξαρτήσεις τους. Η υλοποίηση της κλάσης AbstractModule που προσφέρεται από την Guice εξυπηρετεί αυτόν τον σκοπό. Να σημειωθεί πως μπορεί να έχουμε αρκετές υλοποιήσεις της ίδιας υπηρεσίας. Κατά την εκκίνηση του προγράμματος θα έχουμε μια συσχέτιση με συγκεκριμένη υλοποίηση. Παρόλα αυτά μπορεί να είναι αναγκαίο κατά τον χρόνο εκτέλεσης του συστήματος να γίνει αλλαγή σε άλλη υλοποίηση της υπηρεσίας. Η συμπεριφορά αυτή δηλώνεται στο Module ως multibindings όσον αφορά τον τρόπο με τον οποίο το επιτυγχάνει η Guice.

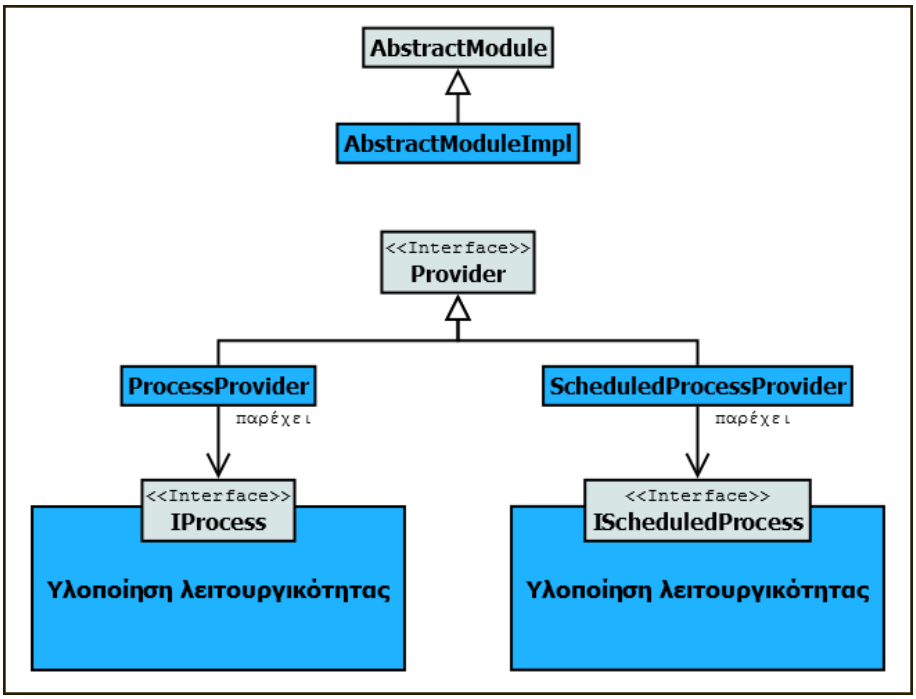
4.4 Δομή Διαδικασιών

Όπως αναφέρθηκε η διαδικασία είναι αποτέλεσμα της σύνθεσης διάφορων υπηρεσιών. Στόχος είναι να παραχθεί λειτουργικότητα που έχει νόημα και είναι χρήσιμη στην εφαρμογή. Στην ουσία η διαδικασία είναι και αυτή ένας τύπος υπηρεσίας. Εδώ υπάρχει μεγαλύτερη ελευθερία αφού η γενική δομή που απεικονίζεται στο σχήμα 4.4 είναι απλά ενδεικτική.

Διεπαφή διαδικασίας. Η περιγραφή της λειτουργικότητας και των περιορισμών που θέτει η διαδικασία γίνεται μέσω της διεπαφής της.

Provider. Όπως και στις υπηρεσίες, έτσι κι εδώ, μια διαδικασία μπορεί να είναι πολύπλοκη στην κατασκευή. Ένας Provider αναλαμβάνει να κατασκευάσει τέτοια αντικείμενα αποκρύπτοντας τον τρόπο από τον χρήστη.

Παραμετροποίηση Module. Αντίστοιχα, οι αντιστοιχίσεις διεπαφών/κλάσεων με τις υλοποιήσεις τους που αφορούν την Guice βρίσκονται στο Module της διαδικασίας.



Σχήμα 4.4: Γενική δομή διαδικασίας

Πέρα από την απλή σύνθεση των διαδικασιών, μπορούν να εισαχθούν νέες έννοιες (semantics). Στην παρούσα φάση, πέρα από την απλή, έχουμε την έννοια της προγραμματιζόμενης διαδικασίας. Για να γίνει μια διαδικασία προγραμματισμένη ώστε να εκτελείται ανά τακτά χρονικά διαστήματα, πρέπει να υλοποιηθεί η κλάση TimerTask της Java. Όσον αφορά την Guice, για τον διαχωρισμό τον υλοποιήσεων, χρειάζεται η ανάγκη ενός annotation που να δηλώνει την παραπάνω έννοια.

Κεφάλαιο 5

Πρωτόκολλα Αρχιτεκτονικής

5.1 Εισαγωγή

Στο παρόν κεφάλαιο περιγράφονται τα πρωτόκολλα του συστήματος. Η βιβλιοθήκη έχει βασιστεί στο P-Grid και επομένως υλοποιείται ο αλγόριθμος exchange όπως έχει οριστεί στο [4]. Θα γίνει μια μικρή περιγραφή του. Στη συνέχεια αναλύεται διεξοδικά το πρωτόκολλο ανοχής λαθών που η ανάπτυξη του είναι και ένας από τους στόχους της διπλωματικής εργασίας.

5.2 Πρωτόκολλο Exchange

Το P-Grid δίκτυο κατασκευάζεται μέσω των αλληλεπιδράσεων μεταξύ των peer και μέσω του αλγορίθμου exchange. Στα σχήματα 5.1 και 5.2 παρουσιάζεται ο αλγόριθμος.

Αρχικά, κατά τα πρώτα στάδια εκκίνησης του δικτύου, οι peer είναι υπεύθυνοι για όλο τον χώρο κλειδιών. Δεν υπάρχουν κατατμήσεις και όλοι οι peer αντιστοιχούν στο ριζικό μονοπάτι του δέντρου. Ο αλγόριθμος exchange καθορίζει τον τρόπο με τον οποίο ο χώρος κλειδιών θα διασπαστεί σε περισσότερες εξειδικεύσεις. Κάθε μια από αυτές τίθεται υπό την ευθύνη ενός peer. Ο αλγόριθμος εκτελείται με την πρώτη ευκαιρία που δύο peer θα συναντηθούν.

Οι peer ελέγχονται πάνω στο μονοπάτι για το οποίο είναι υπεύθυνοι. Σε κάθε εκτέλεση του αλγορίθμου, οι δυο peer ανανεώνουν τους πίνακες δρομολόγησης τους. Αρχικά, ανταλλάσσουν τυχαία αναφορές προς peer για όλα τα επίπεδα του κοινού προθέματος.

Περίπτωση 1. Η πρώτη περίπτωση του αλγορίθμου προκύπτει όταν αυτοί αντιστοιχούν στο ίδιο μονοπάτι, έστω x, και είναι στην γραμμή 10. Τότε και οι δύο peer εξειδικεύονται περαιτέρω προς αντίθετες κατευθύνσεις. Ο ένας θα αναλάβει το μονοπάτι 0 και ο άλλος το συζυγές 1. Κάθε peer προσθέτει τον άλλον στον πίνακα δρομολόγησης ως υπεύθυνο για το νέο επίπεδο που δημιουργήθηκε στο μονοπάτι.

Περιπτώσεις 2 και 3. Η δεύτερη και η τρίτη περίπτωση είναι παρόμοιες. Το μονοπάτι του ενός peer είναι μικρότερο από αυτό του άλλου και μάλιστα είναι και πρόθεμα του. Διακρίνονται στις γραμμές 16 και 21. Στην περίπτωση αυτή, ο λιγότερο εξειδικευμένος peer προχωρά προς την αντίθετη κατεύθυνση. Αν παράδειγμα αυτός είχε το μονοπάτι x και ο άλλος peer το $x1$, τότε ο πρώτος εξειδικεύεται προς το $x0$. Επίσης, ανανεώνονται και οι αναφορές στους πίνακες δρομολόγησης. Παρατηρούμε, πως αν ο peer $a1$ αντιστοιχεί το μικρότερο μονοπάτι, τότε κατά την εκτέλεση του αλγορίθμου αυτός θα είναι στην δεύτερη περίπτωση ενώ ο $a2$ στην τρίτη.

Περίπτωση 4. Τέλος, στην γραμμή 26 έχουμε την περίπτωση όπου οι δυο peer μοιράζονται ένα κοινό πρόθεμα αλλά από εκεί και μετά έχουν ακολουθήσει διαφορετικούς δρόμους στο δέντρο. Εδώ ο αλγόριθμος εκτελείται αναδρομικά. Οι peer $a1$ και $a2$ προωθούν αιτήσεις για εκτέλεση του αλγορίθμου στους peer που έχουν αποθηκευμένους στους πίνακες δρομολόγησης τους. Όταν το δίκτυο φτάσει στην τελική του μορφή, ο χώρος κλειδιών θα έχει διασπαστεί σε τόσα κομμάτια όσος και ο αριθμός των peer. Τότε θα προκύπτει συνεχώς η τέταρτη περίπτωση κατά την εκτέλεση του αλγορίθμου. Η σταθερά $recmax$ θέτει ένα άνω όριο στον αριθμό των αναδρομών που θα εκτελεστούν και είναι μια συνθήκη τερματισμού.

Algorithm 5.1 Αλγόριθμος Exchange, μέρος 1ο

```
 1: procedure EXCHANGE(a1, a2, r)
 2:     commonpath = common_prefix_of(path(a1), path(a2));
 3:     lc = length(commonpath);
 4:     if (lc > 0) then      ▷ exchange references at the level where the paths agree
 5:         commonrefs = union(refs(lc, a1), refs(lc, a2));
 6:         refs(lc, a1) = random_select(refmax, commonrefs);
 7:         refs(lc, a2) = random_select(refmax, commonrefs);
 8:         l1 = length(sub_path(path(a1), lc + 1, length(path(a1))));
 9:         l2 = length(sub_path(path(a2), lc + 1, length(path(a2))));
    ▷ Case 1: if both remaining paths are empty introduce a new level
10:         if (l1 = 0 & l2 = 0) then
11:             path(a1) = append(path(a1), 0);
12:             path(a2) = append(path(a2), 1);
13:             refs(lc + 1, a1) = a2;
14:             refs(lc + 1, a2) = a1;
15:         end if
    ▷ Case 2: if one remaining path is empty split the shorter path
16:         if (l1 = 0 & l2 > 0) then
17:             path(a1) = append(path(a1), ¬value(lc + 1, path(a2)));
18:             refs(lc + 1, a1) = a2;
19:             refs(lc + 1, a2) = random_select(refmax, union(a1, refs(lc + 1, a2)));
20:         end if
```

Algorithm 5.2 Αλγόριθμος Exchange, μέρος 2ο

```
                                                    ▷ Case 3: like case 2
21:         if (l1 > 0 & l2 = 0) then
22:             path(a2) = append(path(a2), ¬value(lc + 1, path(a1)));
23:             refs(lc + 1, a2) = a1;
24:             refs(lc + 1, a1) = random_select(refmax, union(a2, refs(lc + 1, a1)));
25:         end if
    ▷ Case 4: recursively perform exchange with referenced peers
26:         if (l1 > 0 & l2 > 0 & r < recmax) then
27:             refs1 = refs(lc + 1, a1) \ a2;
28:             refs2 = refs(lc + 1, a2) \ a1;
29:             for r1 ∈ refs1 do
30:                 if online(peer(r1)) then
31:                     exchange(a2, peer(r1), r + 1)
32:                 end if
33:             end for
34:             for r2 ∈ refs2 do
35:                 if online(peer(r2)) then
36:                     exchange(a1, peer(r2), r + 1)
37:                 end if
38:             end for
39:         end if
40:     end if
41: end procedure
```

5.3 Πρωτόκολλο Ανοχής Λαθών (Fault-Tolerance)

5.3.1 Γενικά

Για να κατανοήσουμε τι είναι η ανοχή λαθών ας δούμε τα χαρακτηριστικά ενός αξιόπιστου συστήματος.

Διαθεσιμότητα (Availability). Αντιπροσωπεύει την ετοιμότητα του συστήματος. Συγκεκριμένα είναι η πιθανότητα το σύστημα να λειτουργεί σωστά σε κάποιες χρονικές στιγμές και να εκτελεί σωστά τη λειτουργικότητα που προσφέρει.

Αξιοπιστία (Reliability). Είναι η ικανότητα του συστήματος να λειτουργεί συνεχόμενα χωρίς αποτυχίες. Είναι η πιθανότητα να λειτουργεί σε ένα χρονικό διάστημα.

Ασφάλεια (Safety). Η κατάσταση όπου το σύστημα αποτυγχάνει προσωρινά χωρίς να συμβεί κάτι καταστροφικό.

Δυνατότητα συντήρησης (Maintainability). Η ευκολία με την οποία επιδιορθώνεται το σύστημα.

Η εκπλήρωση των παραπάνω χαρακτηριστικών επιτυγχάνεται με τη μείωση των αποτυχιών που συμβαίνουν στο σύστημα. Ακόμα όμως και σε περιπτώσεις αποτυχίας το σύστημα πρέπει να λειτουργεί σωστά και να συνεχίσει να παρέχει τις υπηρεσίες για τις οποίες έχει σχεδιαστεί. Αυτή είναι και η ουσία της έννοιας «ανοχή λαθών». Οι αποτυχίες χωρίζονται σε διάφορες κατηγορίες. Μπορεί να είναι παροδική, επαναλαμβανόμενη ή μόνιμη όσον αφορά τη συμπεριφορά της σε σχέση με τον χρόνο. Οι λόγοι που αναγνωρίζεται κάτι ως αποτυχία είναι η διακοπή λειτουργίας μέρους του δικτύου (crash). Μπορεί όμως διάφοροι peer να λειτουργούν κανονικά και σωστά αλλά να απαντούν σε μηνύματα μετά από κάποιο χρόνο timeout. Αντίστοιχο πρόβλημα είναι η παράληψη αποστολής ή παραλαβής μηνυμάτων το οποίο μπορεί να αντιμετωπιστεί ως αποτυχία. Επίσης, είναι πιθανό να στέλνονται λάθος απαντήσεις είτε κατά λάθος είτε κακόβουλα (Byzantine). Να σημειωθεί πως κάποια από τα προβλήματα αντιμετωπίζονται απλά. Παράδειγμα στην περίπτωση αργοπορίας απάντησης λόγω timeout μπορεί να δοθούν επιπλέον ευκαιρίες ώστε να εξακριβωθεί η κατάσταση του peer.

Τα παραπάνω είναι μια κατηγορία προβλημάτων που επηρεάζουν την κατάσταση του δικτύου. Επιπλέον, έχουμε και τη συνεχή σύνδεση και αποχώρηση των peer από το δίκτυο χωρίς να μπορούμε να προβλέψουμε τον χρόνο συμμετοχής. Αυτό το φαινόμενο ονομάζεται churn [31].

Συνολικά, οι δομές που διατηρεί κάθε peer, όπως τον πίνακα δρομολόγησης, αν δεν ανανεωθούν τότε αυξάνεται η καθυστέρηση του συστήματος. Ο αριθμός των μηνυμάτων αυξάνεται αφού πολλά στέλνονται σε μη διαθέσιμους peer και αυτή η καθυστέρηση είναι συνέπεια των timeout των απαντήσεων. Επομένως, χρειαζόμαστε μια συνολική στρατηγική για τη συντήρηση του δικτύου. Το πρόβλημα είναι το πώς θα ανιχνευτεί μια αποτυχία, πώς θα την διορθώσει το σύστημα και γενικά πώς θα δράσει απέναντι στο φαινόμενο churn.

Κάθε peer είτε αποθηκεύει στον πίνακα δρομολόγησης τους γείτονες του είτε έχει επιπλέον δομές και κρατάει επιπλέον πληροφορίες. Έχουμε διάφορες προσεγγίσεις στο πότε θα εκκινηθεί η διαδικασία συντήρησης:

1. Ενεργή συντήρηση. Όταν ένας peer αποτυγχάνει ή αποχωρεί από το δίκτυο τότε κάποιος από τους γείτονες του ενημερώνει τους υπόλοιπους ανταλλάσοντας πληροφορίες για τους ενεργούς. Είτε μπορεί να στείλει τις λίστες με τους γείτονες που αποθηκεύει είτε μόνο αυτούς που έχουν αλλάξει σε σύγκριση με μια προηγούμενη κατάσταση. Όπως θα δούμε παρακάτω αυτή την τακτική ακολουθούν τα συστήματα Pastry και Tapestry.

2. Ευκαιριακή συντήρηση. Περιοδικά ο peer ανταλλάσει τις δομές όπου αποθηκεύει τους γείτονες του επιλέγοντας έναν peer από αυτές. Αντίστοιχα ελέγχουν και ανανεώνουν την κατάσταση των γειτόνων τους. Το Chord ακολουθεί αυτό το παράδειγμα.

Η ανίχνευση της αποτυχίας μπορεί να γίνει μέσω περιοδικών ping ή εφαρμογή πρωτοκόλλων gossip. Μπορεί να ακολουθηθεί και πιο παθητική στάση αναμένοντας μηνύματα ενημέρωσης αποτυχίας. Στις επόμενες παραγράφους περιγράφεται συνοπτικά η διαδικασία συντήρησης γνωστών peer-to-peer συστημάτων.

Στο Pastry [32] χρησιμοποιείται ένας gossip αλγόριθμος μέσω του οποίου ανανεώνεται ο πίνακας δρομολόγησης. Όταν ανακαλυφθεί ένα πιθανό πρόβλημα και διαπιστωθεί πως πρόκειται για αποτυχία, καταγράφεται και εκκινείται η διαδικασία επιδιόρθωσης. Στο Tapestry [33] ακολουθείται η τακτική του πλεονασμού (redundancy) όσον αφορά τις αναφορές που αποθηκεύονται στον πίνακα δρομολόγησης. Οπότε στην είσοδο/έξοδο ενός peer καθώς και σε αποτυχία του, ενημερώνεται εκείνο το κομμάτι του δικτύου που γνωρίζει τον peer και αντίστοιχα ανανεώνονται οι πίνακες δρομολόγησης. Στο Chord [34] κάθε peer κρατά μια λίστα με peer διαδόχους (successor). Στην περίπτωση αποτυχίας επικοινωνεί με τον πιο κοντινό του τοπολογικά για να λύσει το πρόβλημα.

Στην περίπτωση του P-Grid έχουμε την εφαρμογή της τεχνικής της αντιγραφής (replication). Υπάρχουν peer αντίγραφα άλλων που ο ρόλος τους είναι να πάρουν την θέση τους στην περίπτωση που αποτύχουν. Η τεχνική του replication όμως απαιτεί ένα σύνολο των peer του δικτύου να λειτουργούν ως αντίγραφα. Συνεπώς, μειώνεται η χωρητικότητα του συστήματος και έχουμε το επιπλέον κόστος της συντήρησης και του συγχρονισμού. Όλοι οι peer αντίγραφα πρέπει να έχουν όχι μόνο ενημερωμένες τις δομές τους αλλά και όλα τα δεδομένα που αποθηκεύουν. Επίσης, αν αποτύχει ένας peer καθώς και όλα τα αντίγραφά του τότε δημιουργείται ένα κενό στη τοπολογία του P-Grid. Προκύπτουν μονοπάτια τα οποία δεν έχουν αντίστοιχα συζυγή το οποίο έχει συνέπειες στην συνολική απόδοση του συστήματος. Είναι επομένως αναγκαία η δημιουργία ενός πρωτοκόλλου που θα διατηρεί την δομή του δικτύου χωρίς να δημιουργούνται κενά λόγω αποτυχιών ή αποχώρησης των peer. Ένα σημαντικό χαρακτηριστικό το οποίο βοηθάει στην απόδοση του πρωτοκόλλου είναι η αποθήκευση πολλαπλών αναφορών για κάθε επίπεδο του πίνακα δρομολόγησης ενός peer.

5.3.2 Ανάλυση Αλγορίθμων Πρωτοκόλλου

Το πρωτόκολλο αποτελείται από τους αλγόριθμους 5.3 FindContinuation, 5.4 Replace και 5.5 Broadcast. Για την καλύτερη κατανόηση κρίνεται σκόπιμο να αναλυθούν αυτοί πρώτα. Στην επόμενη ενότητα θα εξηγηθεί ο τρόπος και η σειρά με τον οποίο χρησιμοποιούνται οι αλγόριθμοι αυτοί ώστε να λυθούν τα προβλήματα που παρουσιάζονται στο δίκτυο.

Αλγόριθμος FindContinuation

Για να βρεθεί εκείνος ο peer που θα δώσει την λύση στην αποτυχία πρέπει να διασχίσουμε την τοπολογία του P-Grid δικτύου με συγκεκριμένο τρόπο. Ο αλγόριθμος 5.3 (FindContinuation) επιτελεί αυτό το έργο. Χρειάζεται ένα όρισμα, το ίχνος *pathTrace*. Αυτό αντιπροσωπεύει την διαδρομή που ακολουθεί η επιδιόρθωση μέσα στο δίκτυο. Ο αλγόριθμος εκτελείται εξολοκλήρου τοπικά από τους peer του

Algorithm 5.3 Αλγόριθμος FindContinuation

```
1:  procedure p.FindContinuation(pathTrace)
2:      if path(p) ≡ pathTrace then
3:          return p
4:      end if
5:      if p.isPrefixOf(pathTrace) ∪ (path(p).length() ≤ pathTrace.length()) then
6:          return p.closestLevelTo(pathTrace)
7:      end if
8:      route1 ← pathTrace + ¬pathTrace.lastBit()
9:      route2 ← pathTrace + pathTrace.lastBit()
10:     if p.hasPrefix(route1) then
11:         if path(p) ≡ route1 then
12:             if p has conjugate peer and not a subtree then
13:                 return p
14:             else
15:                 return p.closestLevelTo(route2)
16:             end if
17:         else
18:             return p.FindContinuation(route1)
19:         end if
20:     else if p.hasPrefix(route2) then
21:         if path(p) ≡ route2 then
22:             if p has conjugate peer and not a subtree then
23:                 return p
24:             else
25:                 return p.closestLevelTo(route1)
26:             end if
27:         else
28:             return p.FindContinuation(route2)
29:         end if
30:     end if
31: end procedure
```

δικτύου. Κατά τον τερματισμό της εκτέλεσης τους θα επιστραφεί εκείνος ο peer που είναι υπεύθυνος να συνεχίσει την εκτέλεση του πρωτοκόλλου επιδιόρθωσης. Να σημειωθεί πως μπορεί να μην επιστραφεί κάποιος peer. Ο λόγος είναι η απουσία ύπαρξης αναφοράς στον πίνακα δρομολόγησης που να ικανοποιεί τον αλγόριθμο.

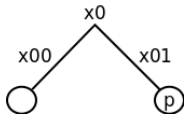
Ο αλγόριθμος ξεκινά από ένα μονοπάτι το οποίο είναι πρόθεμα σε ένα υπόδεντρο. Το ποθητό αποτέλεσμα είναι να καταλήξουμε σε ένα μονοπάτι το οποίο είναι φύλλο μέσα στο P-Grid δέντρο, να αντιστοιχεί δηλαδή σε peer. Για να μπορέσει αυτός ο peer να λύσει το πρόβλημα, όπως θα δούμε και παρακάτω στον αλγόριθμο Replace, θα πρέπει στο συζυγές μονοπάτι του να βρίσκεται ένας peer και όχι υπόδεντρο. Από την ανάλυση του αλγορίθμου 5.3 προκύπτουν κάποιες περιπτώσεις. Παρακάτω εξετάζονται αυτές και αναλύεται τι επιστρέφει ο αλγόριθμος. Να σημειωθεί πως ο αλγόριθμος εκτελείται και τερματίζει τοπικά χωρίς κάποια απομακρυσμένη κλήση συνάρτησης. Όμως, ο peer που θα επιστραφεί από αυτόν είναι και αυτός που θα συνεχίσει το πρωτόκολλο και θα εκτελέσει εκ νέου τον αλγόριθμο 5.3. Αν το δούμε μακροσκοπικά, μέρος του δικτύου συνεργάζεται εκτελώντας τον ώστε να καταλήξει σε συγκεκριμένο ζευγάρι peer. Το ζευγάρι αυτό θα λύσει και την αποτυχία.

Στην γραμμή 2 έχουμε την περίπτωση όπου το ίχνος είναι όμοιο με το μονοπάτι του peer ο οποίος και επιστρέφεται.

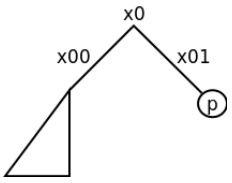
Στην γραμμή 5 ελέγχεται αν ο peer που εκτελεί τον αλγόριθμο είναι σε μικρότερο μονοπάτι από το ίχνος. Το μονοπάτι και το ίχνος μπορεί να έχουν κάποια σχέση αν ο peer ανήκει στο ίδιο υπόδεντρο με αυτό του ίχνους. Στην περίπτωση αυτή δεν μπορεί να αποφασίσει για την κατεύθυνση. Επιλέγει έναν peer από τον πίνακα δρομολόγησης του που είναι πιο κοντά στο ίχνος και τον επιστρέφει με στόχο να συνεχίσει εκεί το πρωτόκολλο επιδιόρθωσης.

Δεδομένου ότι δεν ισχύει τίποτα από τα παραπάνω, αποφασίζεται η διαδρομή που πρέπει να ακολουθηθεί. Με βάση το ίχνος οι δυνατές κατευθύνσεις είναι δύο και προκύπτουν από το τελικό bit του. Στην γραμμή 8 το πρώτο μονοπάτι *route1* προκύπτει με την αντιστροφή του τελευταίου bit και την προσθήκη του στο τέλος του ίχνους. Αντίστοιχα και για το δεύτερο *route2* στην γραμμή 9 χωρίς την αντιστροφή. Στην ουσία τα δυο μονοπάτια είναι η κατεύθυνση προς τα υπόδεντρα αριστερά και δεξιά του ίχνους 'x', 'x0' και 'x1'. Στα σχήματα 5.1 και 5.2, το ίχνος έχει τιμή 'x0' και συνεπώς έχουμε $route1 =$ 'x01' και $route2 =$ 'x00'. Αντίστοιχα για την περίπτωση όπου το ίχνος είναι 'x1' έχουμε $route1 =$ 'x10' και $route2 =$ 'x11'. Και οι δύο περιπτώσεις είναι όμοιες στην διαχείριση. Για την εξήγηση του αλγορίθμου χρησιμοποιούμε για ίχνος το 'x0' για τον peer *p* που εκτελεί τον αλγόριθμο τοπικά.

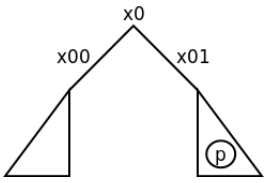
Αρχικά ελέγχουμε το μονοπάτι του peer που εκτελεί τον αλγόριθμο με τη πρώτη διαδρομή *route1*. Στη γραμμή 10 αποφασίζεται αν ο peer έχει πρόθεμα το *route1*. Αν η απάντηση είναι αληθής τότε διακρίνουμε τρεις περιπτώσεις που πρέπει να ελέγξουμε. Οι περιπτώσεις αυτές είναι που βρίσκεται ο peer που εκτελεί τον αλγόριθμο και τι συμβαίνει στο συζυγές μονοπάτι του *route1*. Ένας πρώτος διαχωρισμός είναι αν το μονοπάτι του peer είναι όμοιο με το *route1* ή όχι. Ο έλεγχος φαίνεται στη γραμμή 11.



(α′) Peer p στο μονοπάτι route1 και peer στο συζυγές μονοπάτι



(β′) Peer p στο μονοπάτι route1 και υπόδεντρο στο συζυγές μονοπάτι

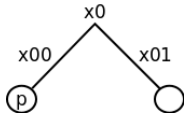


(γ′) Το μονοπάτι route1 είναι πρόθεμα σε υπόδεντρο που περιέχει τον peer p

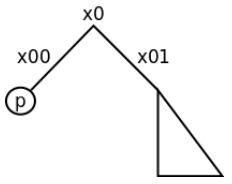
Σχήμα 5.1: Περιπτώσεις κατά την επιλογή της κατεύθυνσης *route1* ≡ *'x01'*

Για την περίπτωση της ομοιότητας για να αποφανθούμε τι τελικά θα ακολουθήσουμε πρέπει να εξετάσουμε το συζυγές μονοπάτι του *route1*. Εδώ έχουμε δυο περιπτώσεις, η πρώτη είναι το συζυγές να αντιστοιχεί σε μόνο ένα peer και εξετάζεται στη γραμμή 12. Η κατάσταση αυτή φαίνεται ξεκάθαρα στο σχήμα5.1α′. Ο αλγόριθμος τερματίζει και επιστρέφεται ο peer *p* που τον εκτέλεσε. Η διάσχιση οδήγησε σε δυο φύλλα του δέντρου του P-Grid και οι δυο peer που αντιστοιχούν σε αυτά μπορούν να λύσουν την αποτυχία. Αντίθετα στη γραμμή 14 καταλήγουμε αν το συζυγές μονοπάτι αντιστοιχεί σε ένα υπόδεντρο που περιέχει άνω του ενός peer. Η κατάσταση απεικονίζεται στο σχήμα 5.1β′, δεν έχουμε φτάσει σε φύλλα. Όμως, οι δυο peer που θα δώσουν την λύση βρίσκονται στο υπόδεντρο αυτό. Συνεπώς, ο αλγόριθμος τερματίζει επιστρέφοντας έναν peer που βρίσκεται στον πίνακα δρομολόγησης του *p*. Αυτός ανήκει στο συζυγές υπόδεντρο του *route1*, έχοντας δηλαδή πρόθεμα το *route2*. Στο σχήμα 5.1β′ είναι στο 'x00'.

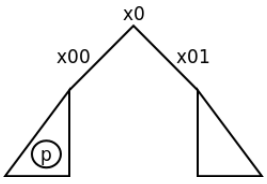
Η τελική περίπτωση που έχουμε για την διαδρομή *route1* είναι στη γραμμή 17 όπου η διαδρομή είναι πρόθεμα ενός υποδέντρου. Στο συζυγές υπόδεντρο δε μας ενδιαφέρει τι γίνεται. Ανεξάρτητα του αριθμού των peer που βρίσκονται σε αυτό, οι peer που θα δώσουν την λύση βρίσκονται στο υπόδεντρο με πρόθεμα *route1*. Η επιλογή της διαδρομής στην περίπτωση αυτή γίνεται κατά σύμβαση. Ο αλγόριθμος συνεχίζει αναδρομικά εξειδικεύοντας σε αυτό το υπόδεντρο μέχρι να φτάσει σε διαδρομή που αντιστοιχεί σε έναν peer ακριβώς.



(α′) Peer p στο μονοπάτι route2 και peer στο συζυγές μονοπάτι



(β′) Peer p στο μονοπάτι route2 και υπόδεντρο στο συζυγές μονοπάτι



(γ′) Το μονοπάτι route2 είναι πρόθεμα σε υπόδεντρο που περιέχει τον peer p

Σχήμα 5.2: Περιπτώσεις κατά την επιλογή της κατεύθυνσης *route2* ≡ *'x00'*

Αν ο peer *p* δεν έχει πρόθεμα το μονοπάτι *route1* τότε βρίσκεται στο *route2*.

Ο έλεγχος φαίνεται στην γραμμή 20. Αντίστοιχα και εδώ έχουμε παρόμοιες περιπτώσεις απλά αλλάζει η θέση του peer *p*.

Συγκεκριμένα, στην γραμμή 21 κρίνεται αν ο *p* έχει μονοπάτι όμοιο με το *route2*. Η περίπτωση που απεικονίζεται στο σχήμα 5.2α', όπου *route2* είναι ίσο με 'χ00', αντιστοιχεί στην επαλήθευση του ελέγχου στην γραμμή 12. Ο αλγόριθμος έχει οδηγηθεί σε φύλλα και άρα έχουν βρεθεί οι peer που θα λύσουν το πρόβλημα. Επιστρέφεται επομένως ο peer *p*. Αντίθετα, αν έχουμε συζυγές υπόδεντρο με πρόθεμα *route1*, τότε το ακολουθούμε επιστρέφοντας έναν peer που ανήκει σε αυτό. Στο σχήμα 5.2β' φαίνεται η περίπτωση αυτή.

Τέλος, στην γραμμή 27 είναι η περίπτωση όπου η διαδρομή *route2* είναι πρόθεμα στο υπόδεντρο που ανήκει ο *p*. Ο αλγόριθμος συνεχίζει αναδρομικά προς αυτή την διαδρομή ώστε να εξειδικευτεί περισσότερο και να καταλήξουμε σε διαδρομή που αντιστοιχεί σε peer.

Αλγόριθμος Replace

Ο αλγόριθμος επιδιόρθωσης θα καταλήξει σε έναν peer. Αυτός μαζί με τον συζυγή του πρόκειται να δώσουν την λύση στην αποτυχία. Ο αλγόριθμος 5.4 (Replace) δείχνει τον τρόπο. Συγκεκριμένα, ο ένας από τους δυο peer θα λάβει το μονοπάτι που αντιστοιχεί στην αποτυχία. Ο δεύτερος θα μειώσει το μονοπάτι του κατά ένα, δηλαδή γενικεύεται. Υπενθυμίζουμε πως ως αποτυχία θεωρείται είτε ένας peer, είτε ένα το σύνολο των peer ενός υποδέντρου. Από σύμβαση, ο peer του οποίου το μονοπάτι καταλήγει σε '1', είναι αυτός που θα αντικαταστήσει. Τέλος, ανανεώνουν τον πίνακα δρομολόγησης. Στην απλή περίπτωση που η αποτυχία είναι ο συζυγής peer, τότε απλά μειώνεται το μονοπάτι κατά ένα. Ο αλγόριθμος 5.4 εκτελείται ταυτόχρονα από τους δύο peer που είναι υπεύθυνοι για λύση. Όταν τερματίσει και από τους δυο τότε, οι δυο peer έχουν διορθώσει το πρόβλημα. Το επόμενο βήμα είναι η αναμετάδοση της λύσης μέσα στο δίκτυο.

Algorithm 5.4 Αλγόριθμος Replace

```
 1: procedure p.Replace(failed)
 2:     if failed.isConjugateTo(p) then
 3:         path(p).reduceBy(1)
 4:     else
 5:         if path(p).endsWith('1') then
 6:             path(p) ← path(failed)
 7:         else
 8:             path(p).reduceBy(1)
 9:         end if
10:     end if
11:     p.refreshRoutingTable()
12: end procedure
```

Αλγόριθμος Broadcast

Ο αλγόριθμος 5.5 (Broadcast) είναι υπεύθυνος για την διάδοση της λύσης στο δίκτυο. Ο peer στον οποίο θα καταλήξει ο αλγόριθμος επιδιόρθωσης για να διορθώσει το πρόβλημα είναι αυτός που θα καλέσει την μέθοδο *InitBroadcast*. Τα ορίσματα που απαιτούνται είναι *(α')* το μονοπάτι που αντιστοιχεί στην αποτυχία είτε αυτή είναι μεμονωμένος peer είτε ολόκληρο υπόδεντρο, *(β')* μια λίστα με όλους τους peer που εξαιτίας της επίλυσης άλλαξαν την κατάστασή τους και *(γ')* μια λίστα με όλους τους αποτυχημένους peer που έχουν πρόθεμα το προαναφερθέν μονοπάτι. Η λίστα με τους ανανεωμένους peer περιέχει τον peer που εκκινεί τον αλγόριθμο και τον συζυγή του. Σε αυτούς κατέληξε ο αλγόριθμος επιδιόρθωσης και συνεπώς αυτοί θα αλλάξουν το μονοπάτι τους ώστε να λύσουν την αποτυχία. Αρχικά ανανεώνεται ο πίνακας δρομολόγησης αφαιρώντας όλους τους peer που έχουν αποτύχει. Στη συνέχεια για κάθε επίπεδο που έχει αποθηκευμένο στον πίνακα δρομολόγησης του, επιλέγει τυχαία έναν peer από αυτό και του προωθεί την λύση.

Algorithm 5.5 Αλγόριθμος Broadcast

```
 1: procedure p.InitBroadcast(failedPath, updatedHosts, failedHosts)
 2:     p.updateRoutingTable(failedHosts)
 3:     for each level in RoutingTable do
 4:         p' ← level.selectRandomHost()
 5:         subtree ← path(level)
 6:         p'.broadcast(failedPath, updatedHosts, failedHosts, subtree)
 7:     end for
 8: end procedure

 9: procedure p.Broadcast(failedPath, updatedHosts, failedHosts, subtree)
10:     p.updateRoutingTable(failedHosts, updatedHosts)
11:     if path(p) ≡ subtree then
12:         return
13:     end if
14:     for each level in RoutingTable.levelsIn(subtree) do
15:         p' ← level.selectRandomHost()
16:         subtree ← path(level)
17:         p'.broadcast(failedPath, updatedHosts, failedHosts, subtree)
18:     end for
19: end procedure
```

Στη συνέχεια, οι υπόλοιποι peer εκτελούνε την μέθοδο *Broadcast*. Η μέθοδος αυτή έχει τα ίδια ορίσματα με την *InitBroadcast* και επιπλέον ένα, το *subtree*. Αυτό υποδηλώνει στον peer πιο είναι το υπόδεντρο για το οποίο είναι υπεύθυνος να προωθήσει την λύση. Όπως μπορούμε να δούμε από τον ψευδοκώδικα, κάθε peer είναι υπεύθυνος για το υπόδεντρο στο οποίο ανήκει. Ο λόγος για τον οποίο

γίνεται αυτό είναι ο περιορισμός του αριθμού των μηνυμάτων που πρέπει να στείλουμε. Σε αντίθετη περίπτωση πολλοί peer θα δεχόντουσαν πολλαπλά μηνύματα για την ίδια λύση. Ο peer αφαιρεί από τον πίνακα δρομολόγησης του τους αποτυχημένους peer. Αν έχει αναφορές σε αυτούς που έχουν αλλάξει κατάσταση, τις ενημερώνει, αλλιώς τους προσθέτει στον πίνακα. Καταλήγει προωθώντας την λύση επιλέγοντας τυχαία έναν peer για κάθε επίπεδο του πίνακα που έχει πρόθεμα το μονοπάτι του υποδέντρου για το οποίο είναι υπεύθυνος. Ένας peer δεν εκτελεί τον αλγόριθμο όταν το μονοπάτι ευθύνης είναι όμοιο με το μονοπάτι του. Αυτό σημαίνει πως δεν υπάρχουν peer περισσότερο εξειδικευμένοι από αυτόν.

5.3.3 Ανάλυση Πρωτοκόλλου

Το πρωτόκολλο μπορεί να αντιμετωπίσει είτε μεμονωμένες αποτυχίες peer είτε την αποτυχία ενός πλήρους υποδέντρου του δικτύου. Πριν εξηγηθούν τα βήματα που εκτελούνται σε κάθε περίπτωση, θα εξηγηθεί μια δομή που χρησιμοποιείται κατά την επίλυση προβλημάτων.

Χρησιμοποιείται τοπικά σε κάθε peer ένας πίνακας καταχώρησης (registry) των αποτυχιών που προκύπτουν κατά τη διάρκεια ζωής του συστήματος. Οι καταχωρήσεις αυτές παραμένουν μέχρι τη στιγμή που αυτές επιλυθούν. Ουσιαστικά αυτές αφορούν έναν peer που αντιστοιχεί σε ένα μονοπάτι ο οποίος έχει αποτύχει. Προϋπόθεση για την διαγραφή της καταχώρησης είναι η λήψη μηνύματος μέσω του αλγορίθμου Broadcast που περιέχει την λύση που αφορά τον συγκεκριμένο peer .

Όταν ανακαλυφθεί ένας peer που έχει αποτύχει τότε το σύστημα προσπαθεί να τον διορθώσει βρίσκοντας το σωστό ζευγάρι για την αντικατάσταση του. Συνεπώς, το πρωτόκολλο ξεκινά πάντα με επίλυση αποτυχίας μεμονωμένου peer. Είναι αρκετά πιθανό η αποτυχία να ανακαλυφθεί ταυτόχρονα από πολλούς peer και να εκκινηθεί παράλληλα από αυτούς. Επομένως, το πρώτο που γίνεται είναι ο έλεγχος της registry και η ανανέωση του πίνακα δρομολόγησης όπου αφαιρείται οποιαδήποτε αναφορά προς τον προβληματικό peer. Αν μέσα στην registry βρεθεί το ID του αποτυχημένου peer τότε σημαίνει πως το πρωτόκολλο έχει τρέξει και αναμένει για την λύση.

Επίλυση αποτυχίας μεμονωμένου peer

Αν στην registry δεν βρεθεί ο αποτυχημένος peer τότε εκτελείται εκείνο το κομμάτι του πρωτοκόλλου που επιλύει αποτυχία ενός μεμονωμένου peer.

Αρχικά εκτελείται ο αλγόριθμος 5.3 FindContinuation. Αυτός θα επιλέξει έναν peer για την συνέχιση του πρωτοκόλλου όπως αναλύθηκε προηγουμένως. Αν ο peer είναι ένας από τον πίνακα δρομολόγησης και δεν είναι ο τοπικός, αυτός που εκτελεί τον αλγόριθμο δηλαδή, τότε σημαίνει πως ακόμα δεν έχουμε φτάσει σε φύλλα. Μπορούμε να θυμηθούμε τα σχήματα 5.1β′, 5.2β′, 5.1γ′ και 5.2γ′ για να καταλάβουμε γιατί.

Αν αυτός ο peer είναι ο τοπικός τότε σημαίνει πως αυτός είναι σε θέση να επιλύσει την αποτυχία. Για υπενθύμιση μπορούμε να δούμε τις εικόνες 5.1α′ και

5.2α′ για να καταλάβουμε τον λόγο. Ο αλγόριθμος έχει φτάσει σε φύλλα. Για την συνέχεια έχουμε δυο περιπτώσεις που έχουν να κάνουν με τον συζυγή του τοπικού. Ο συζυγής του μπορεί είτε να είναι ο αποτυχημένος peer που προσπαθεί το δίκτυο να επιλύσει. Τότε μέσω της εκτέλεσης του αλγορίθμου 5.4 Replace, ο τοπικός ελαττώνει το μονοπάτι του κατά ένα. Στην περίπτωση που ο συζυγής είναι διαθέσιμος τότε ο τοπικός του στέλνει αίτηση εκτέλεσης του αλγορίθμου Replace για τον αποτυχημένο peer. Όπως αναλύθηκε παραπάνω, ο ένας peer θα πάρει το μονοπάτι του αποτυχημένου και ο άλλος θα ελαττώσει κατά ένα το μονοπάτι του.

Το τελικό βήμα του πρωτοκόλλου εκτελείται από τον peer που θα εκτελέσει τον αλγόριθμο Replace. Εκτελεί τον αλγόριθμο Broadcast (5.5) ώστε η λύση να διαδοθεί σε όλο το δίκτυο. Αφού ανανεωθεί ο πίνακας δρομολόγησης ενός peer που έχει λάβει την λύση, μόνο τότε διαγράφει την καταχώρηση του αποτυχημένου peer από την registry. Αυτό αφορά εκείνους τους peer που συμμετείχαν ενεργά στην εκτέλεση του πρωτοκόλλου. Όσοι δεν πήραν μέρος πιθανώς να μη γνωρίζουν για την αποτυχία. Δεν έχουμε πρόβλημα όμως αφού ο πίνακας δρομολόγησης τους ανανεώνεται με τους νέους peer και επομένως μαθαίνουν την καινούρια κατάσταση του δικτύου.

Ο τερματισμός του πρωτοκόλλου όταν επιλύεται ένας μεμονωμένος peer του δικτύου εγγυάται την λύση του. Να σημειωθεί πως μπορεί να έχουν αποτύχει πολλοί peer. Όταν οι δυο peer που θα προκύψουν από το πρωτόκολλο για την λύση ενός peer δεν συμμετέχουν στην λύση ενός άλλου τότε η ανάλυση που έγινε για αποτυχία μεμονωμένου peer στέκει. Ως συμμετοχή εννοείται η εκτέλεση του αλγορίθμου Replace. Μπορούν να συνεισφέρουν όμως μέσω του αλγορίθμου FindContinuation χωρίς να υπάρχει πρόβλημα. Αυτό ισχύει ακόμα και αν έχουν αλλάξει μονοπάτι χωρίς να έχει διαδοθεί η λύση στο δίκτυο. Τέλος, όσον αφορά την παράλληλη εκκίνηση του πρωτοκόλλου, είναι φανερό πως μέσω του αλγορίθμου FindContinuation το πρωτόκολλο κάποια στιγμή θα οδηγηθεί σε έναν συγκεκριμένο peer. Αυτός είναι ο συζυγής του αποτυχημένου. Επομένως, δεν επηρεάζει από ποιον και από πόσους peer θα ξεκινήσει το πρωτόκολλο. Αυτο ισχύει και στην επίλυση αποτυχίας ενός πλήρους υποδέντρου που θα αναλυθεί παρακάτω.

Επίλυση αποτυχίας ενός πλήρους υποδέντρου

Παραπάνω εξηγήθηκε πώς επιλύονται αποτυχίες μεμονωμένων peer. Όμως, σε ένα peer-to-peer σύστημα είναι σύνηθες φαινόμενο να έχουμε πολλαπλές αποτυχίες. Ειδικά στο P-Grid αυτές οι αποτυχίες μπορεί να είναι και ένα ολόκληρο υπόδεντρο με όλους τους peer που περιέχει αυτό. Αυτό που μας ενδιαφέρει είναι οι peer που συμμετέχουν στην επίλυση τους. Είναι πιθανό ένας peer να είναι υπεύθυνος για την διόρθωση πολλών αποτυχιών.

Η λογική επίλυσης είναι ίδια με αυτή της αποτυχίας ενός peer. Το σύστημα και εδώ το ενδιαφέρει να λύσει ένα μονοπάτι. Στην αποτυχία ενός peer αυτό το μονοπάτι αντιστοιχούσε σε ακριβώς έναν peer. Στην αποτυχία ενός πλήρους υποδέντρου το μονοπάτι αυτό είναι πρόθεμα σε μια ομάδα peer. Με την έννοια

πλήρες υπόδεντρο εννοούμε όλα εκείνα τα μονοπάτια στα οποία αυτό έχει εξειδικευτεί. Άρα πλήρης αποτυχία υποδέντρου ισοδυναμεί με αποτυχία όλων των peer που αντιστοιχούν στα εξειδικευμένα μονοπάτια που περιέχει. Το πρόβλημα που προκύπτει είναι η απόφαση πως ένα πλήρες υπόδεντρο έχει αποτύχει. Εδώ βοηθάει η registry που περιγράφηκε στην αρχή. Στην αντιμετώπιση της αποτυχίας πλήρους υποδέντρου μπορούμε να βρεθούμε με δυο τρόπους είτε μέσω της απευθείας κλήσης για διόρθωση ενός γνωστού προβληματικού υποδέντρου είτε κατά την διόρθωση μεμονωμένου peer. Η πρώτη περίπτωση προϋποθέτει γνώση όλων των αποτυχημένων peer του υποδέντρου. Παρακάτω ξεκινάει η ανάλυση της δεύτερης περίπτωσης αφού μέσω αυτής εξηγείται και η πρώτη.

Έστω ότι ο τοπικός peer ανακαλύπτει τον αποτυχημένο peer f με μονοπάτι xit. Ξεκινάει το πρωτόκολλο για επίλυσης μεμονωμένου peer. Τα βήματα που γίνονται αρχικά δεν αλλάζουν. Κανονικά ελέγχεται αν υπάρχει στην registry. Αν όχι τότε προστίθεται σε αυτήν και αφαιρείται η αναφορά προς αυτόν, αν υπήρχε, από τον πίνακα δρομολόγησης. Εκτελείται ο αλγόριθμος 5.3 FindContinuation ώστε να βρεθεί ένας peer που μπορεί να προχωρήσει το πρωτόκολλο. Εδώ μπορούν να προκύψουν δυο καταστάσεις, η πρώτη είναι ο αλγόριθμος να επιστρέψει κανονικά τον συνεχιστή ενώ η δεύτερη να μην βρει κάποιον peer.

Κατάσταση 1η. Έστω πως έχει βρεθεί ο peer q που θα συνεχίσει το πρωτόκολλο. Όπως περιγράφηκε στην μεμονωμένη αποτυχία, αν ο peer αυτός είναι ο τοπικός peer τότε γίνεται η αντικατάσταση του f και το δίκτυο έχει διορθώσει το πρόβλημα. Σε αντίθετη περίπτωση ο τοπικός peer πρόκειται να επικοινωνήσει με αυτόν ώστε να συνεχιστεί το πρωτόκολλο. Η κατάσταση που μας ενδιαφέρει είναι ο peer q να έχει αποτύχει. Επειδή αυτός βρέθηκε να είναι σημαντικός για την επίλυση του f, δίνεται προτεραιότητα από το πρωτόκολλο στην επίλυση του q. Πρέπει πρώτα όμως να δούμε πως ο peer q σχετίζεται με τον f.

1. Ο q έχει μονοπάτι $xi\bar{t}$ και άρα είναι συζυγής του f. Ο τοπικός peer ανακαλύπτει πως το υπόδεντρο με μονοπάτι xi έχει αποτύχει πλήρως. Συνεπώς ξεκινά το πρωτόκολλο για την επίλυσή του. Η λύση θα αναζητηθεί στο υπόδεντρο με μονοπάτι $x\bar{i}$.

2. Ο q έχει μονοπάτι yz και άρα δεν έχει σχέση με τον f. Εδώ σταματά η επίλυση του f και ξεκινά η διαδικασία επίλυσης μεμονωμένου peer για τον q. Η αναζήτηση της λύσης θα γίνει στο υπόδεντρο $y\bar{z}$.

Κατάσταση 2η. Ο αλγόριθμος FindContinuation δεν βρήκε κάποιον peer που να μπορεί να συνεχίσει το πρωτόκολλο. Εδώ ο τοπικός peer πρέπει να συμβουλευτεί την registry. Υπενθυμίζουμε πως στην registry είναι καταχωρημένος ο peer f με μονοπάτι xit. Προκύπτουν δυο καταστάσεις για την πορεία του πρωτοκόλλου.

1. Αν ο συζυγής του f που βρίσκεται στο μονοπάτι $xi\bar{t}$ έχει αποτύχει και ο τοπικός το γνωρίζει με κάποιον τρόπο τότε θα είναι καταχωρημένος στην registry. Άρα ανακαλύπτουμε πως το υπόδεντρο με μονοπάτι xi έχει

αποτύχει πλήρως. Συνεπώς ξεκινά το πρωτόκολλο για την επίλυση του. Η αναζήτηση της λύσης θα γίνει στο υπόδεντρο $x\bar{i}$.

2. Αν ο συζυγής του f δεν υπάρχει στην registry τότε ο τοπικός διακόπτει την εξέλιξη του πρωτοκόλλου τοπικά αφού δεν γνωρίζει κανέναν για να την επίλυση του f. Υποθέτουμε πως κάποιος άλλος peer του δικτύου έχει περισσότερη γνώση για το τι συμβαίνει στο υπόδεντρο όπου ανήκει ο f. Στον πίνακα δρομολόγησης για κάθε επίπεδο αποθηκεύονται πολλαπλές αναφορές προς άλλους peer. Η πιθανότητα να υπάρχει ένας peer που γνωρίζει για το υπόδεντρο του f μεγαλώνει όσο αυξάνονται οι αναφορές κάθε επιπέδου του πίνακα και όσο αυξάνονται οι peer συνολικά του δικτύου. Μια βελτιστοποίηση είναι η αναμονή του τοπικού peer ενός συγκεκριμένου χρονικού διαστήματος για την λήψη ενός μηνύματος που περιέχει την λύση είτε του f είτε του συζυγούς του είτε του υποδέντρου στο οποίο ανήκουν. Αν δε έρθει τέτοιο μήνυμα τότε λύνεται το υπόδεντρο που περιέχει τον f και τον συζυγή του.

Ακριβώς οι ίδιες επιλογές γίνονται κατά την εκτέλεση του πρωτοκόλλου απευθείας για την επίλυση ενός πλήρους υποδέντρου. Πάλι μπορούν να προκύψουν peer που έχουν αποτύχει. Αυτό που ελέγχει ο τοπικός peer είναι τα μονοπάτια των peer που είναι αποθηκευμένοι στην registry για την ανακάλυψη ενός πιο γενικού αποτυχημένου υποδέντρου. Μπορούμε να φανταστούμε το πρωτόκολλο κατά την εκτέλεση επίλυσης του υποδέντρου xt όπου προκύπτει ένας αποτυχημένος peer q με μονοπάτι $x\bar{t}*$. Κατά την πορεία επίλυσης του q πλέον καταλήγει πως όλο το υπόδεντρο με μονοπάτι $x\bar{t}$ έχει αποτύχει. Συνεπώς, μέσω της registry προκύπτει πως το γενικότερο υπόδεντρο με μονοπάτι x έχει αποτύχει. Συνεπώς επιλύεται αυτό.

Η λύση θα δοθεί όπως και στην μεμονωμένη περίπτωση. Θα βρεθούν δηλαδή δυο peer οι οποίοι θα εκτελέσουν τον αλγόριθμο 5.4 Replace. Ο ένας από αυτούς θα λάβει το μονοπάτι του αποτυχημένου υποδέντρου. Στο προηγούμενο παράδειγμα δηλαδή μετά το τέλος του Replace θα έχει μονοπάτι x.

Τέλος η λύση διαδίδεται με τον ίδιο τρόπο. Η λίστα με όλους τους peer που έχουν αποτύχει γνωστοποιείται σε όλο το δίκτυο. Οι πίνακες δρομολόγησης ανανεώνονται και το δίκτυο έχει επανέλθει σε σωστή κατάσταση.

Κεφάλαιο 6

Υλοποίηση Αρχιτεκτονικής

6.1 Εισαγωγή

Η υλοποίηση της προτεινόμενης αρχιτεκτονικής έγινε με τη χρήση της γλώσσας προγραμματισμού Java. Όπως έχει αναφερθεί η Guice χρησιμοποιήθηκε για την υποστήριξη του προτύπου dependency injection. Η υλοποίηση της CORBA που προσφέρεται από την Java καλύπτει τις ανάγκες απομακρυσμένων κλήσεων συναρτήσεων. Η ανάγκη καταγραφής μηνυμάτων καλύφθηκε από την βιβλιοθήκη log4j και της slf4j. Τέλος, για την διενέργεια ελέγχων χρησιμοποιήθηκε η βιβλιοθήκη JUnit.

Έγινε χρήση διάφορων εργαλείων όπως το Ant το οποίο καθοδηγεί μεταγλώττιση, την εκτέλεση των unit test, την εξαγωγή της τεκμηρίωσης μέσω javadoc και την δημιουργία της βιβλιοθήκης ως jar αρχείου. Αναγκαία ήταν διάφορα εργαλεία στατικής ανάλυσης του κώδικα που προσφέρονταν μέσω του IDE IntelijIDEA. Για την διαχείριση του κώδικα και των εκδόσεων του χρησιμοποιήθηκε το git. Ο κώδικας της βιβλιοθήκης υπάρχει στο αποθετήριο `https://github.com/Archimidis/pgrid`

6.2 Common Object Request Broker Architecture (CORBA)

Η υλοποίηση βασίστηκε πάνω στην CORBA [35--38] για την επικοινωνία των peer μέσω κλήσεων απομακρυσμένων συναρτήσεων (rpc). Η δομή της αρχιτεκτονικής της δε μας αφορά. Είναι σημαντικές, όμως, κάποιες λεπτομέρειες οι οποίες αφορούν την περαιτέρω παραμετροποίηση του συστήματος.

Για την υλοποίηση των υπηρεσιών πρέπει να γραφεί σε idl η περιγραφή αυτής. Μέσω της idl παράγονται ο κώδικας που είναι χρήσιμος για την κλήση μιας συνάρτησης client (stub code) και ο κώδικας για την εξυπηρέτηση αυτών των κλήσεων (skeleton code). Το δεύτερο κομμάτι είναι εκεί όπου υλοποιείται η λειτουργικότητα της διεπαφής. Η προώθηση των κλήσεων σε αυτό γίνεται μέσω του servant.

Ένα σημαντικό κομμάτι της αρχιτεκτονικής που ενδιαφέρει και εμάς είναι ο

Portable Object Adapter (POA). Ο ρόλος του είναι να ενώσει τους servant με το ORB και η διαχείριση του περιβάλλοντος το αντικείμενο που είναι η υλοποίηση της διεπαφής κατά τον χρόνο εκτέλεσης. Μέσω αυτού γίνεται η αντιστοίχιση servant με υλοποιήσεις. Επίσης, ο POA μπορεί να ενεργοποιήσει το αντικείμενο και η προσαρμογή του σε συγκεκριμένες πολιτικές. Το τελευταίο είναι που προσθέτει επιπλέον δυνατότητες παραμετροποίησης του συστήματος.

Οι πολιτικές που χρησιμοποιούνται από τον POA είναι οι παρακάτω [39]:

1. **Μοναδικότητα ID.** Καθορίζει αν περισσότερα από ένα ID αντικειμένου θα αναφέρεται στον ίδιο servant.

2. **Ανάθεση ID.** Καθορίζει αν την ανάθεση των ID την κάνει ο POA ή ο προγραμματιστής.

3. **Διάρκεια ζωής.** Καθορίζει αν τα αντικείμενα είναι παροδικά ή μόνιμα. Αν το αντικείμενο, δηλαδή, είναι προσιτό μετά την καταστροφή του POA ή όχι.

4. **Διατήρηση servant.** Καθορίζει αν ο POA κρατά τις συσχετίσεις μεταξύ ID αντικειμένου και servant σε μια δομή που ονομάζεται Active Object Map. Σε αντίθετη περίπτωση στηρίζεται στον προεπιλεγμένο servant ή σε servant locator για τον εντοπισμό του κατάλληλου servant με στόχο την εξυπηρέτηση της αίτησης.

5. **Επεξεργασία αίτησης.** Καθορίζει αν ο POA χρησιμοποιεί είτε μόνο το Active Object Map, είτε μόνο τον προεπιλεγμένο servant είτε την χρήση του servant manager. Η επιλογή που θα γίνει για την διατήρηση του servant επηρεάζει και τον τρόπο με τον οποίο θα γίνει η επεξεργασία. Συγκεκριμένα για τον servant manager υπάρχουν δυο κατηγορίες

   - **Servant activator.** βάσει αυτού δημιουργείται ο servant και όταν τελειώσει ο κύκλος ζωής του τότε καταστρέφεται.
   - **Servant locator.** Εύρεση του κατάλληλου servant ο οποίος δεν είναι καταχωρημένος στο Active Object Map.

6. **Πολιτική νημάτων.** Καθορίζει αν θα ακολουθηθεί μονονηματική πολιτική ή όχι.

7. **Σιωπηρή ενεργοποίηση servant.** Καθορίζει αν ο POA μπορεί να ενεργοποιήσει έναν servant όταν μια αναφορά προς ένα ID δημιουργηθεί.

Μπορούν συνδυασμοί μπορούν να προκύψουν από τα παραπάνω. Η χρήση μόνο του Active Object Map για την επεξεργασία των αιτήσεων συνοδεύεται με την επιλογή της αποθήκευσης αντιστοιχίσεων ID και servant σε αυτόν. Πρέπει να γνωρίζουμε πριν την εκκίνηση του συστήματος τον αριθμό των servant αφού αυτοί θα δημιουργηθούν και θα εγκατασταθούν ρητά.

Στην περίπτωση που ο αριθμός των αντικειμένων είναι μεγάλος ή δεν τον γνωρίζουμε εξ αρχής τότε είναι χρήσιμος ο servant manager. Αν επιλέξουμε την

ύπαρξη αντιστοιχίσεων στο Active Object Map, τότε χρησιμοποιείται ο servant activator. Αν δεν βρεθεί ένα αντικείμενο μέσω του ID του τότε ενεργοποιείται ο κατάλληλος servant. Σε αντίθετη περίπτωση όπου δεν έχουμε Active Object Map τότε χρησιμοποιείται ο servant locator. Ο servant επιτρέπει την παροχή διαφορετικών servant που αντιστοιχούν σε διαφορετικές υλοποιήσεις της διεπαφής που καλείται.

Συνεπώς, παρέχοντας υλοποιήσεις των servant manager μπορούμε να καθορίσουμε περαιτέρω τον κύκλο ζωής των υπηρεσιών και των αντικειμένων που χρησιμοποιούν ανάλογα με τις απαιτήσεις του περιβάλλοντος στο οποίο εκτελείται ο peer.

6.3 Επίπεδο Οντοτήτων - Entity Layer

Όπως έχει αναφερθεί στο επίπεδο οντοτήτων υπάρχει οτιδήποτε αποθηκεύει κατάσταση ή δεδομένα και παρέχει πρόσβαση σε υπηρεσίες που προσφέρει το λειτουργικό σύστημα πάνω στο οποίο εκτελείται η εφαρμογή.

Ένας peer για να περιγραφεί ολοκληρωμένα χρειάζεται η διεύθυνσή του (ip και port). Επίσης, μετά την εισαγωγή του στο δίκτυο είναι υπεύθυνος για ένα κομμάτι του χώρου κλειδιών το οποίο καθορίζεται από το μονοπάτι που του έχει δοθεί. Χρειάζεται και αυτή η πληροφορία. Τέλος, κάθε peer αναγνωρίζεται μοναδικά από ένα UUID που του δίνεται κατά την εκκίνησή του. Τα παραπάνω περιγράφονται από την διεπαφή Host.

Δίνονται διεπαφές που αναπαριστούν ένα κλειδί, Key, καθώς και ενός πεδίου κλειδιών, KeyRange, του συστήματος. Τα κλειδιά όπως έχει αναφερθεί συσχετίζονται είτε με peer είτε με πόρους που προσφέρονται από τους peer στο δίκτυο. Επίσης, προσφέρεται διεπαφή που αφορά τα μονοπάτια του P-Grid δέντρου, PGridPath. Διαφορετική σημασιολογία ή αναπαράσταση σε όλα τα παραπάνω μπορεί να δοθεί επεκτείνοντας αυτές τις διεπαφές. Η παραγωγή αντικειμένων αυτών των τύπων για χρήση από τον χρήστη γίνεται μέσω του σχεδιαστικού προτύπου factory [30].

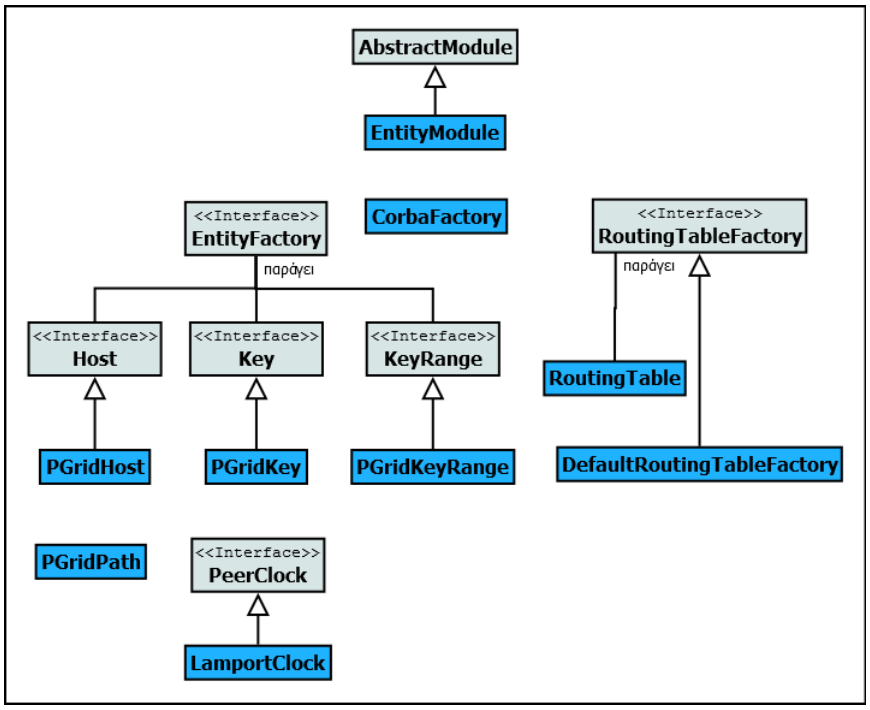
Η έννοια του χρόνου σε ένα κατανεμημένο σύστημα είναι σημαντική για την σειριοποίηση των γεγονότων που συμβαίνουν σε αυτό. Υπάρχει η έννοια του ρολογιού στην διεπαφή PeerClock και δίνεται επίσης η κλασική υλοποίηση των Lamport ρολογιών [40]. Η υλοποίηση είναι ασφαλής για σε πολυνηματικό κώδικα.

Κάθε peer που μετέχει στο δίκτυο αποθηκεύει αναφορές προς άλλους peer. Ο τρόπος και η δομή με την οποία αποθηκεύονται αυτές εκφράζονται μέσω του πίνακα δρομολόγησης, RoutingTable. Η παραγωγή αντικειμένων αυτού του τύπου επιτυγχάνεται με τη χρήση του σχεδιαστικού προτύπου abstract factory [30]. Η υλοποίηση που παρέχεται για τον πίνακα δρομολόγησης είναι ασφαλής για χρήση σε πολυνηματικό κώδικα, δεδομένης της ταυτόχρονης χρήσης από πολλές υπηρεσίες.

Η επικοινωνία μεταξύ των peer του δικτύου επιτυγχάνεται μέσω απομακρυσμένων κλήσεων συναρτήσεων (rpc). Αυτή η λειτουργικότητα προσφέρεται από

την CORBA. Ως οντότητα θεωρείται το ORB και επιπλέον για την ρύθμισή του και την παραγωγή του κατά την εκκίνηση του peer χρησιμοποιείται abstract factory. Κάθε ORB είναι μοναδικό για κάθε peer και πρέπει να υπάρχει ένα αντικείμενο αυτού του τύπου ανά εφαρμογή.

Στην 6.1 φαίνεται το UML διάγραμμα κλάσεων που αφορά το επίπεδο οντοτήτων. Η κλάση EntityModyle περιέχει την παραμετροποίηση που απαιτείται από την Guice.



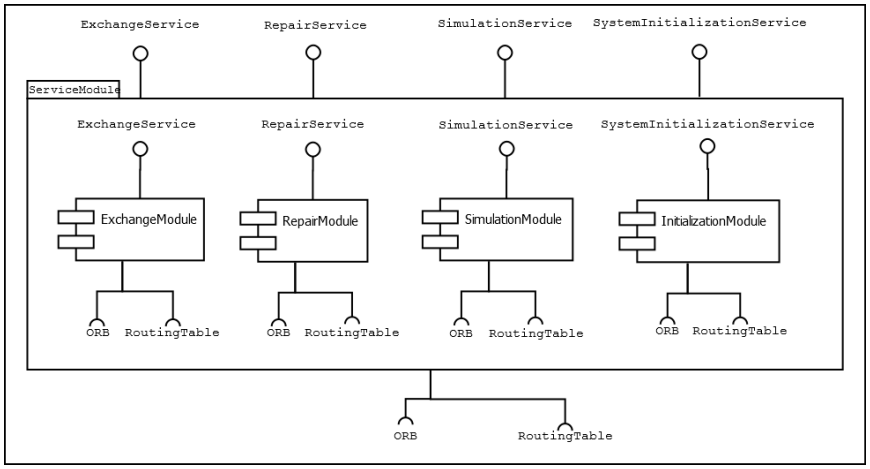
Σχήμα 6.1: Διάγραμμα κλάσεων επιπέδου οντοτήτων

6.4 Επίπεδο Υπηρεσιών - Service Layer

Στο επίπεδο των υπηρεσιών υπάρχουν όλα τα πρωτόκολλα που προσφέρει το σύστημα εκφρασμένα ως υπηρεσίες. Στην παρούσα φάση της υλοποίησης έχουμε το πρωτόκολλο Exchange και αυτό της ανοχής λαθών Repair, όπως αυτά έχουν περιγραφεί. Επίσης, έχει υλοποιηθεί μια υπηρεσία που αφορά την προσομοίωση του συστήματος και βοηθά στον έλεγχό του. Στις επόμενες ενότητες θα αναλυθούν οι υλοποιήσεις των προαναφερθέντων. Παρέχεται, επίσης και μια υπηρεσία για την αρχικοποίηση του συστήματος.

Στο διάγραμμα συστατικών (component) 6.2 φαίνονται οι μονάδες του επιπέδου. Το επίπεδο περιλαμβάνει τα module των υπηρεσιών Exchange, Repair, Simulation και αρχικοποίησης. Επίσης, φαίνονται οι τρεις διεπαφές που εκτίθενται

στα ανώτερα επίπεδα καθώς και οι εξαρτήσεις του παρόντος από το επίπεδο οντοτήτων. Επίσης, κάθε υπηρεσία έχει ανάγκη το ORB για επικοινωνία και τον πίνακα δρομολόγησης του τοπικού peer. Αυτά αποθηκεύονται στην κλάση LocalPeerContext. Κάθε κατά την διάρκεια εκτέλεσης του συστήματος, υπάρχει μόνο αντικείμενο αυτής της κλάσης το οποίο επιστρέφεται από την Guice.



Σχήμα 6.2: Διάγραμμα συστατικών επιπέδου υπηρεσιών

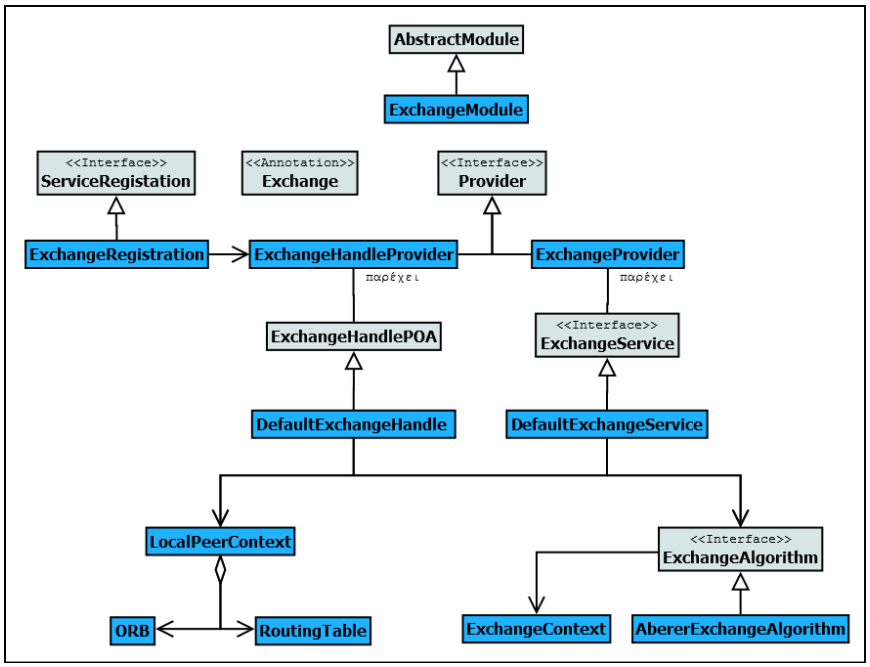
6.4.1 Υλοποίηση Πρωτοκόλλου Exchange ως Υπηρεσία

Στην εικόνα 6.3 φαίνεται το διάγραμμα κλάσεων της υπηρεσίας Exchange. Η δομή της ακολουθεί την γενική δομή της υπηρεσίας που περιγράφηκε σε προηγούμενη ενότητα.

Η διεπαφή που εκθέτει την λειτουργικότητα της υπηρεσίας στο επίπεδο των διεργασιών και της εφαρμογή είναι η ExchangeService. Με την κλήση της μεθόδου που προσφέρει, ο τοπικός peer επικοινωνεί με τον απομακρυσμένο peer που έχει δοθεί ως όρισμα και εκτελούν τον αλγόριθμο exchange. Στην περίπτωση όπου υπάρχει πρόβλημα κατά την επικοινωνία, αν παράδειγμα ο απομακρυσμένος peer έχει αποτύχει, η υπηρεσία τερματίζει χωρίς να εκτελέσει τον αλγόριθμο με ένα CommunicationException.

Αντίστοιχα, για την πλευρά του εξυπηρετητή του peer που αφορά την CORBA έχουμε την κλάση ExchangeHandlePOA που έχει παραχθεί μέσω της idl. Παρέχει την υλοποίηση του CORBA αντικειμένου η οποία θα εγκατασταθεί στην CORBA και θα μπορεί να εξυπηρετεί απομακρυσμένες αιτήσεις εκτέλεσης του αλγορίθμου.

Οι παραπάνω διεπαφές έχουν μια προεπιλεγμένη υλοποίησης. Αυτές σχετίζονται με την διεπαφή ExchangeAlgorithm και της υλοποίησης της, την κλάση AbererExchangeAlgorithm. Η μέθοδος που προσφέρει δέχεται ένα αντικείμενο τύπου ExchangeContext. Το ExchangeContext περιέχει όλη την πληροφορία που χρειάζεται, όπως ο πίνακας δρομολόγησης, για να εκτελεστεί ο αλγόριθμος.



Σχήμα 6.3: Διάγραμμα κλάσεων υπηρεσίας Exchange

Για την κατασκευή αντικειμένων των τύπων που δηλώνονται από τις διεπαφές ExchangeService και ExchangeHandlePOA υλοποιείται η διεπαφή Provider. Ο ExchangeProvider παρέχει *νέα* αντικείμενα της ExchangeService σε κάθε αίτηση παροχής.

Αρχικοποιεί την υπηρεσία με όλα τα απαραίτητα αντικείμενα και σταθερές όπως ο μέγιστος αριθμός αναδρομών που θα εκτελεστεί ο αλγόριθμος. Αντίστοιχα για την ExchangeHandlePOA έχουμε τον ExchangeHandleProvider, μόνο που εδώ παρέχεται πάντα το ίδιο αντικείμενο σε κάθε αίτηση.

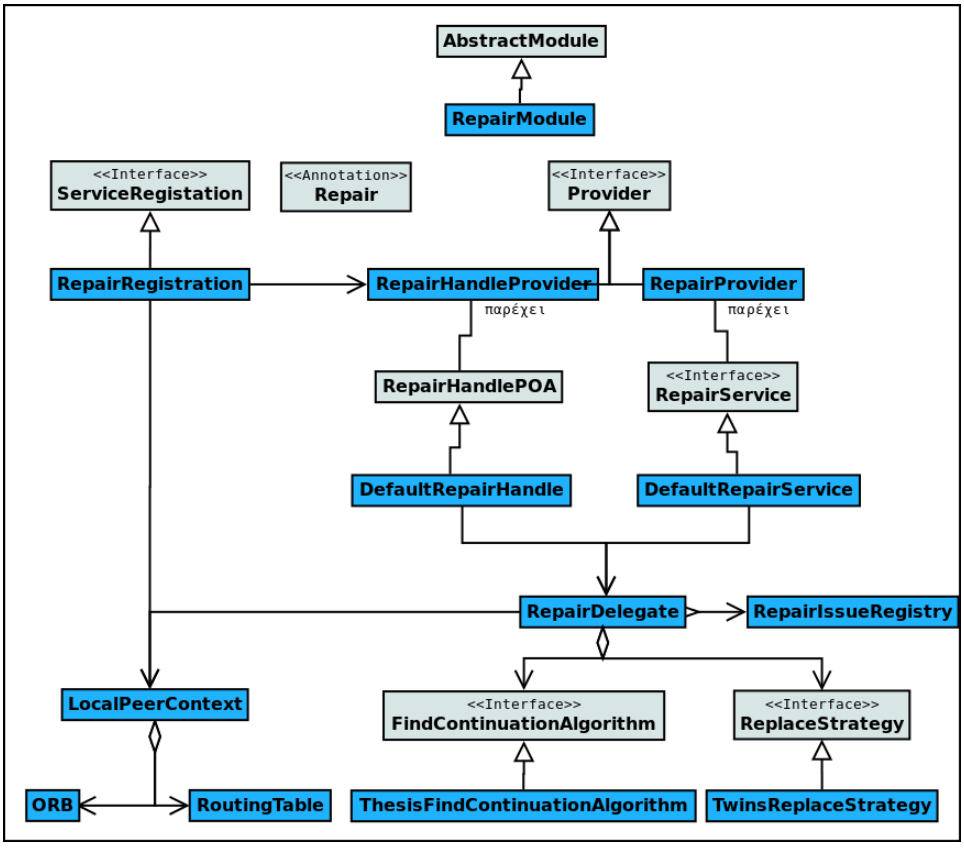
Για την εγκατάσταση της υπηρεσίας στην CORBA έχουμε την υλοποίηση της διεπαφής ServiceRegistration, την ExchangeRegistration. Για να διευκρινιστεί στην Guice ποια υλοποίηση της ServiceRegistration θέλουμε, έχουμε το Annotation Simulation.

Τέλος, η όλη σύνδεση διεπαφών και υλοποιήσεων καθώς και τον καθορισμό του κύκλου ζωής των αντικειμένων βρίσκεται στην κλάση ExchangeModule.

6.4.2 Υλοποίηση Πρωτοκόλλου Repair ως Υπηρεσία

Στην εικόνα 6.4 φαίνεται το διάγραμμα κλάσεων της υπηρεσίας Repair. Αντίστοιχα με το γενικό μοντέλο της υπηρεσίας έχουμε τις διεπαφές RepairService και RepairHandlePOA.

Η διεπαφή RepairService αφορά τις διεργασίες και την εφαρμογή. Προσφέρονται δυο μέθοδοι επιδιόρθωσης στην περίπτωση ενός αποτυχημένου peer ή την περί-



Σχήμα 6.4: Διάγραμμα κλάσεων υπηρεσίας Repair

πτωση ενός αποτυχημένου υποδέντρου του δικτύου. Μετά το πέρας εκτέλεσης της υπηρεσίας θα έχει διορθωθεί το λάθος ή οποιοδήποτε αποτυχία ανακαλυφθεί στην πορεία.

Η διεπαφή RepairHandlePOA αφορά την CORBA και παράγεται μέσω της idl. Η υλοποίηση της φροντίζει για την αποδοχή απομακρυσμένων αιτήσεων και συνέχισης του αλγορίθμου.

Εδώ ακολουθούμε διαφορετική τακτική από αυτή της υπηρεσίας Exchange. Για την αποφυγή διπλοποίησης λειτουργικότητας και της επανάληψης κώδικα σε πολλά σημεία (αρχή DRY [41]), έχουμε την κλάση RepairDelegate. Σε αυτήν αναφέρονται οι υλοποιήσεις των διεπαφών RepairService και RepairHandlePOA για την εξυπηρέτηση των αιτήσεων που δέχονται.

Παρέχονται δυο διεπαφές που αφορούν η μια τον αλγόριθμο 5.3 FindContinuation και η άλλη την τελική διόρθωσης του προβλήματος από τον αλγόριθμο 5.4 Replace όπως έχει προκύψει από την ανάλυση του πρωτοκόλλου ανοχής λαθών στην ενότητα 5.3. Και οι δυο ακολουθούν το σχεδιαστικό πρότυπο Strategy [30].

Συνολικά, η διεπαφή FindContinuationAlgorithm καθορίζει την κατεύθυνση που πρέπει να πάρει αλγόριθμος διασχίζοντας την τοπολογία του δικτύου, επιστρέφοντας τον κατάλληλο peer. Η διεπαφή ReplaceStrategy, χρησιμοποιείται όταν ο αλγόριθμος έχει καταλήξει στους δυο peer που θα λύσουν την αποτυχία. Η κλάση RepairDelegate περιέχει όλη την λογική που αφορά την επικοινωνία μεταξύ των peer για την λύση του προβλήματος. Επίσης, κατά την εκτέλεση της υπηρεσίας μπορεί να ανακαλυφθούν αποτυχημένοι peer. Κάθε ανακάλυψη αποτυχίας καταχωρείται στην RepairRegistry. Η RepairDelegate ελέγχει αν μπορούν να γενικευθούν κάποια προβλήματα από μεμονωμένους αποτυχημένους peer σε ολόκληρα αποτυχημένα υπόδεντρα και να καθοδηγήσει την εκτέλεση σε λύση υποδέντρου.

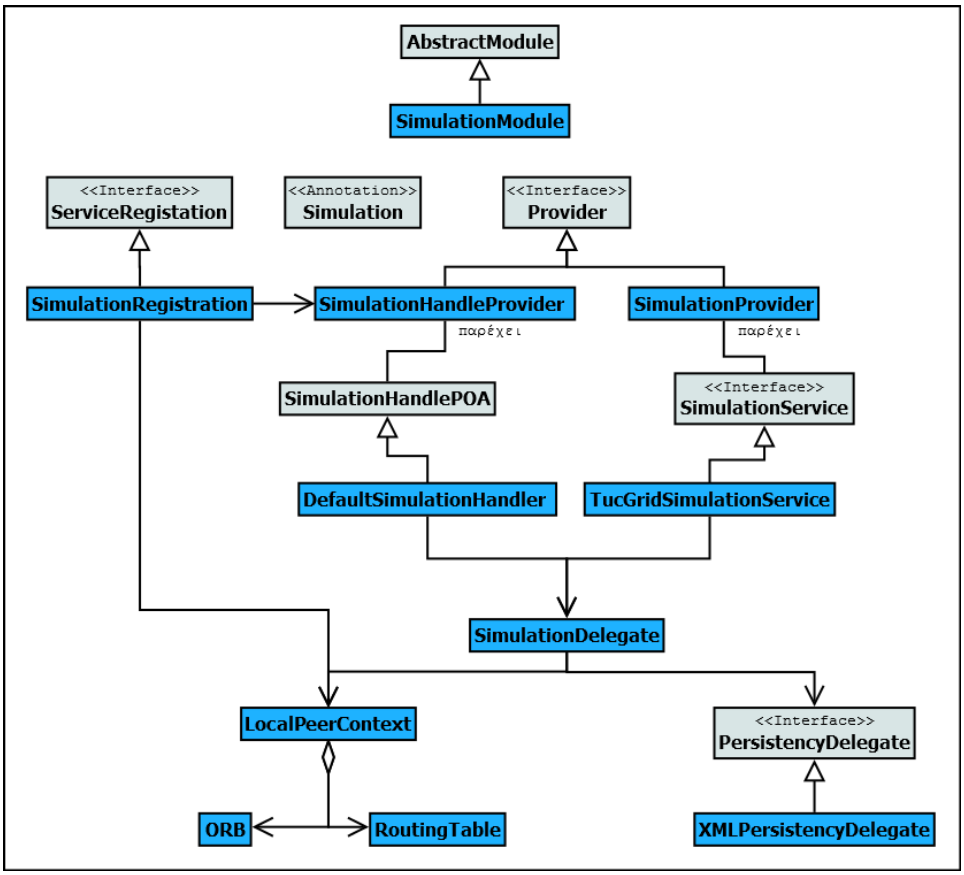
Για την κατασκευή αντικειμένων των κλάσεων που υλοποιούν τις διεπαφές RepairService και RepairHandlePOA υλοποιείται ο Provider. Ο RepairProvider προσφέρει νέα αντικείμενα τύπου RepairService σε κάθε αίτηση παροχής. Αντίστοιχα ο RepairHandleProvider προσφέρει το ίδιο αντικείμενο τύπου RepairHandlePOA. Κατά την κατασκευή και των δυο παραπάνω τύπων χρησιμοποιείται το ίδιο αντικείμενο τύπου RepairRegistry.

Έχουμε την υλοποίηση της διεπαφής ServiceRegistration, RepairRegistration, για την εγκατάσταση της υπηρεσίας την CORBA καθώς και το Annotation Repair χρήσιμο στην Guice.

Στην κλάση RepairModule έχουμε την παραμετροποίηση της υπηρεσίας, την συσχέτιση διεπαφών και υλοποιήσεων όπως φαίνονται στο διάγραμμα κλάσεων και την δήλωση του κύκλου ζωής των αντικειμένων.

6.4.3 Υλοποίηση Υπηρεσίας Προσομοίωσης

Για τον πειραματισμό και τον έλεγχο καλής λειτουργίας του συστήματος έχουμε την ανάγκη μιας υπηρεσίας που θα προσφέρει τις απαραίτητες λειτουργίες. Αυτό τον σκοπό εξυπηρετεί η υπηρεσία προσομοίωσης.



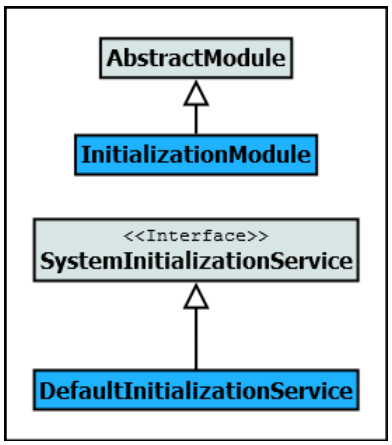
Σχήμα 6.5: Διάγραμμα κλάσεων υπηρεσίας Simulation

Η υπηρεσία προσφέρει την διεπαφή SimulationProvider στις διεργασίες και στην εφαρμογή. Οι μέθοδοι που προσφέρει είναι:

1. Ο τερματισμός ενός συγκεκριμένου peer. Ο peer τερματίζει χωρίς να ειδοποιήσει το δίκτυο και έτσι προσομοιώνεται μια αποτυχία μέσα στο δίκτυο.
2. Η ερώτηση για πληροφορίες ενός peer. Οι πληροφορίες περιλαμβάνουν τον πίνακα δρομολόγησης και το μονοπάτι στο οποίο βρίσκεται ο ερωτηθέν peer.
3. Ο τερματισμός της προσομοίωσης. Στην ουσία στέλνεται σήμα στο δίκτυο και κάθε peer τερματίζει.

Στην εικόνα 6.5 φαίνεται το διάγραμμα κλάσεων της υπηρεσίας. Ακολουθείται η ίδια λογική με αυτή της υπηρεσίας Repair.

6.4.4 Υλοποίηση Υπηρεσίας Αρχικοποίησης Συστήματος

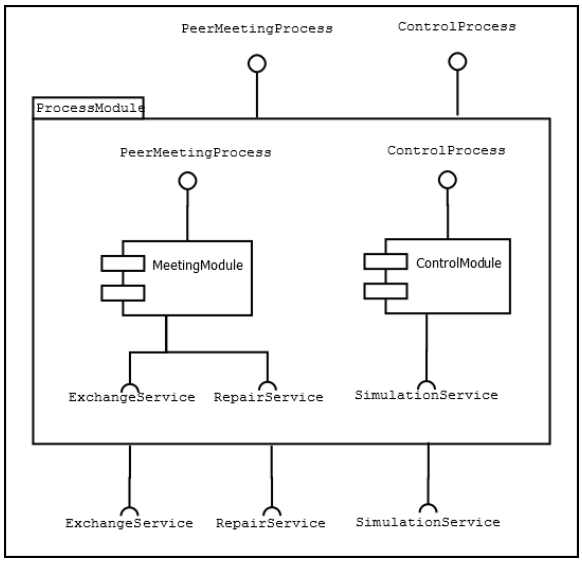


Σχήμα 6.6: Διάγραμμα κλάσεων υπηρεσίας αρχικοποίησης

Η Guice κάνει μια μερική αρχικοποίηση του συστήματος μέσω των συσχετισμών που υπάρχουν μεταξύ διεπαφών και υλοποιήσεων αλλά δεν είναι αρκετή. Η υπηρεσία SystemInitializationService είναι υπεύθυνη για την πλήρη αρχικοποίηση του συστήματος. Μέσω της διεπαφής της υπηρεσίας μπορούν να ενεργοποιηθούν εκείνες οι υπηρεσίες που είναι απαραίτητες στην εφαρμογή. Επίσης, ο τοπικός peer μπορεί να αρχικοποιηθεί σε μια προηγούμενη κατάσταση εκτέλεσης της εφαρμογής. Αποθηκεύεται ο πίνακας δρομολόγησης σε μορφή XML και κατά συνέπεια μπορεί να φορτωθεί. Τέλος, από αυτήν την υπηρεσία ελέγχεται η εκκίνηση και ο τερματισμός της CORBA. Στην εικόνα 6.6 φαίνεται το διάγραμμα κλάσεων της υπηρεσίας. Παρατηρούμε πως ξεφεύγει από την γενική δομή και ο λόγος είναι η απλότητα της υπηρεσίας.

6.5 Επίπεδο Διαδικασιών - Process Layer

Όπως αναλύθηκε, μια σημαντική αρχή των υπηρεσιών είναι αυτή της σύνθεσης. Μέσω αυτής της σύνθεσης προκύπτουν οι διεργασίες οι οποίες ανήκουν στο παρόν επίπεδο. Στην εικόνα 6.7 φαίνονται οι διαδικασίες που περιλαμβάνονται με τις διεπαφές που εκθέτουν καθώς και τις εξαρτήσεις από το επίπεδο των υπηρεσιών. Οι υπάρχουσες διαδικασίες είναι δυο αυτή της συνάντησης δυο peer και άλλη μια που χρησιμοποιείται στην προσομοίωση του συστήματος.



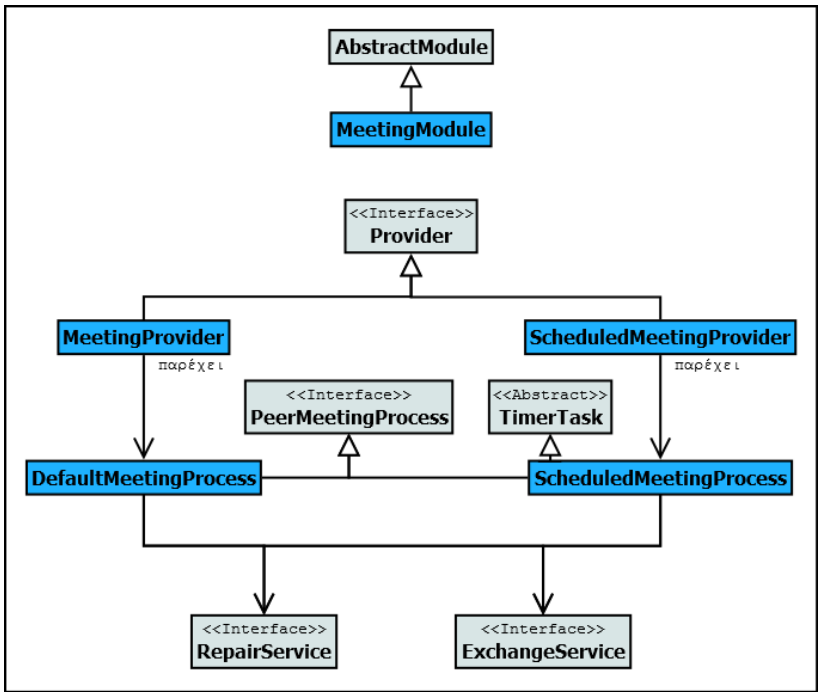
Σχήμα 6.7: Διάγραμμα συστατικών επιπέδου διαδικασιών

6.5.1 Διαδικασία Συνάντησης Peer

Η διαδικασία συνάντησης είναι τα βήματα που ακολουθούνται όταν ένας peer θέλει να εκτελέσει τον αλγόριθμο exchange. Κατά τον Aberer [4] σε κάθε επικοινωνία που γίνεται μεταξύ δυο peer εκτελείται ο αλγόριθμος. Τα βήματα που εκτελεί η διαδικασία είναι η εκτέλεση της υπηρεσίας ExchangeService και στην περίπτωση που έχουμε CommunicationException, δείγμα ότι ο απομακρυσμένος peer έχει αποτύχει, εκτελείται η υπηρεσία RepairService για την λύση του προβλήματος.

Η διαδικασία όπως φαίνεται στο διάγραμμα κλάσεων της στην εικόνα 6.8,ακολουθεί την γενική δομή που αναλύθηκε. Συγκεκριμένα, ορίζεται και παράλληλα εκτίθεται στο επίπεδο εφαρμογής η διεπαφή PeerMeetingProcess. Προσφέρει μια μέθοδο που θέτει σε κίνηση τα βήματα όπως περιγράφηκαν στην προηγούμενη παράγραφο.

Έχουμε δυο υλοποιήσεις της διεπαφής. Η πρώτη είναι μια κλασική υλοποίηση η οποία προορίζεται να χρησιμοποιηθεί από την εφαρμογή καλώντας η ίδια την διαδικασία όποτε θέλει. Η δεύτερη αφορά τον χρονικό προγραμματισμό της διεργασίες, ώστε να εκτελείται μετά το πέρας συγκεκριμένων χρονικών διαστημάτων. Αυτή επιλέγει τυχαία από τον πίνακα δρομολόγησης έναν peer και εκτελεί τη διαδικασία συνάντησης. Για να επιτευχθεί αυτό η διαδικασία επεκτείνει την κλάση TimerTask που προσφέρει η Java. Στη συνέχεια η εφαρμογή θα πρέπει να δημιουργήσει έναν Timer και να εγκαταστήσει την διαδικασία για εκτέλεση.



Σχήμα 6.8: Διάγραμμα κλάσεων διαδικασίας συνάντησης

6.5.2 Διαδικασία Ελέγχου Δικτύου

Η διαδικασία αυτή έχει δημιουργηθεί για τον έλεγχο ενός δικτύου P-Grid από έναν συγκεκριμένο peer. Αποτελείται από μια κλάση την ControlProcess και η λειτουργικότητα που προσφέρει είναι ο εξαναγκασμός αποτυχίας ενός peer και η προβολή του μονοπατιού που έχουν όλοι οι peer του δικτύου. Να σημειωθεί πως ο peer που είναι υπεύθυνος για τον έλεγχο γνωρίζει όλους τους peer του δικτύου.

Κεφάλαιο 7

Παράδειγμα Χρήσης Βιβλιοθήκης

Στην παρούσα ενότητα θα χρησιμοποιηθεί η βιβλιοθήκη για την προσθήκη επιπλέον υπηρεσιών και διαδικασιών. Στόχος είναι να παρουσιαστεί ο τρόπος και η ευκολία με την οποία μπορούν να κατασκευαστούν πολύπλοκες διαδικασίες με την αρχιτεκτονική που προτείνουμε. Επίσης, φαίνονται πώς τα χαρακτηριστικά της υπηρεσίας βοηθούν στο να έχουμε ανεξάρτητες και επαναχρησιμοποιήσιμες λειτουργικές μονάδες.

Στο σύστημα Θα προστεθεί λειτουργικότητα που αφορά τον διαμοιρασμό αρχείων μεταξύ των peer του δικτύου. Στις ακόλουθες υποενότητες θα αναλυθεί το πρόβλημα και τι προσθήκες απαιτούνται. Επίσης αναλύεται ο τρόπος με τον οποίο θα γίνουν αυτές οι προσθήκες.

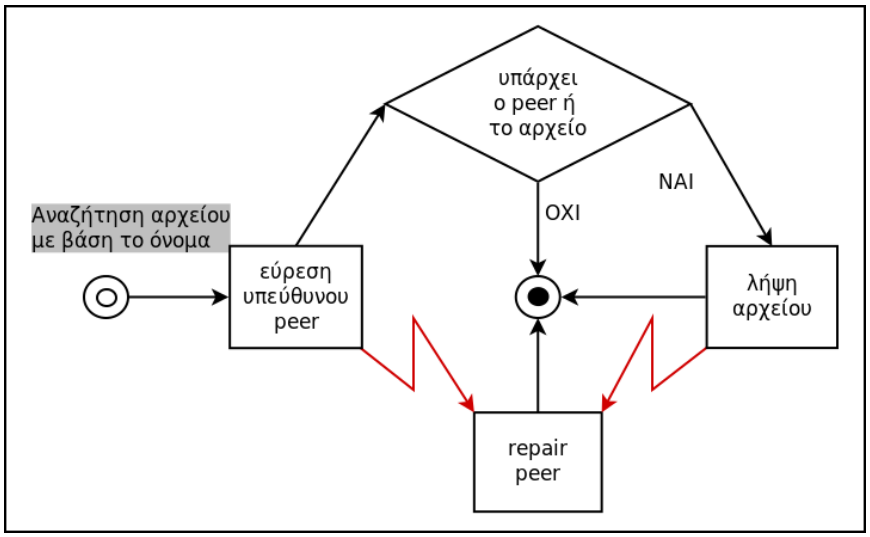
7.1 Ανάλυση Προβλήματος

Ο στόχος όπως αναφέρθηκε είναι η προσθήκη λειτουργικότητας διαμοιρασμού αρχείων. Από αυτό συνεπάγεται πως ένας peer πρέπει να έχει μια δομή όπου θα κρατά στοιχεία για τα αρχεία τα οποία είναι αποθηκευμένα τοπικά. Κάθε αρχείο αντιστοιχεί σε ένα συγκεκριμένο κλειδί του δικτύου. Βάσει αυτού μπορεί να αναζητηθεί και να βρεθεί ο peer που κατέχει το αρχείο.

Πρέπει να εισαχθούν οι απαραίτητες αφαιρέσεις ώστε να παρουσιαστεί το δίκτυο ως ένας ενιαίος αποθηκευτικός χώρος. Βασικές λειτουργίες είναι η αποθήκευση ενός αρχείου στο δίκτυο και η αναζήτηση. Η αναζήτηση αρχείου είτε θα επιστρέψει τον κάτοχο peer σήμα ότι υπάρχει στο δίκτυο είτε τίποτα.

Μια βασική λειτουργία που πρέπει να υπάρχει σε ένα σύστημα διαμοιρασμού αρχείων είναι η δυνατότητα μεταφοράς αρχείων από μεταξύ των peer. Πέρα από την λήψη του αρχείου είναι απαραίτητη η αναφορά προόδου λήψης καθώς και η κατάσταση της (ανενεργό, αναμονή, λήψη).

Τέλος η διαδικασία που είναι άμεσα χρήσιμη στο επίπεδο εφαρμογής είναι η αναζήτηση ενός αρχείου και η λήψη του σε περίπτωση που υπάρχει. Στην εικόνα 7.1 φαίνεται το διάγραμμα ροής αυτής της διεργασίας. Σε περίπτωση που ανακαλυφθεί μια αποτυχία κατά την εκτέλεση της τότε αυτή πρέπει να επιλυθεί.



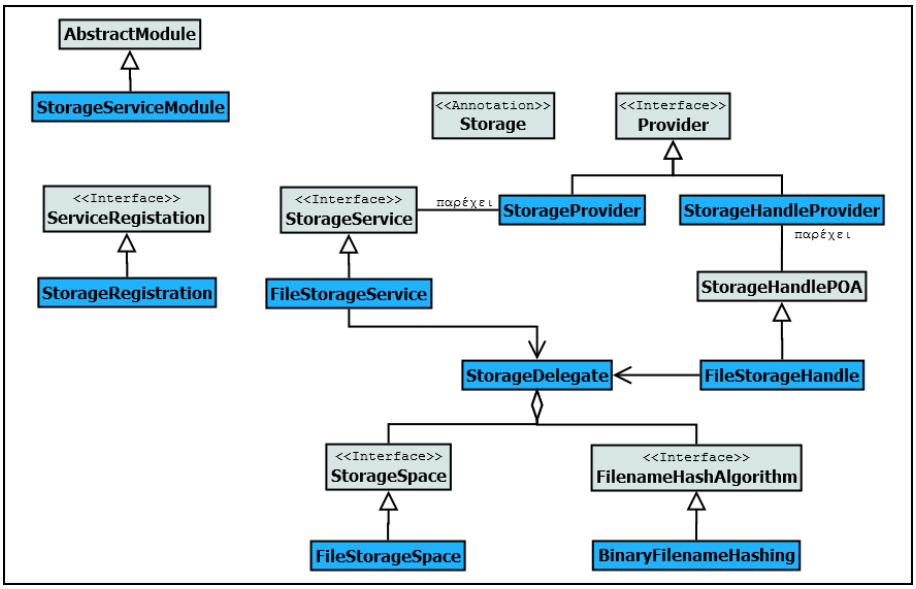
Σχήμα 7.1: Διάγραμμα ροής αναζήτησης και λήψης αρχείου

7.2 Υλοποίηση

Από την προηγούμενη ανάλυση διακρίνουμε δύο υπηρεσίες που αφορούν τον αποθηκευτικό χώρο του δικτύου (Storage Service) και την μεταφορά αρχείων μεταξύ των peer (File Transfer Service). Η σύνθεση αυτών των δυο συνιστούν την διαδικασία αναζήτησης και λήψης (Download File Process). Για απλότητα σε κάθε peer υπάρχει ένας φάκελος όπου χρησιμοποιείται ως αποθηκευτικός χώρος για τα διαμοιραζόμενα αρχεία και άλλος ένας φάκελος όπου αποθηκεύονται τα αρχεία που από την υπηρεσία λήψης.

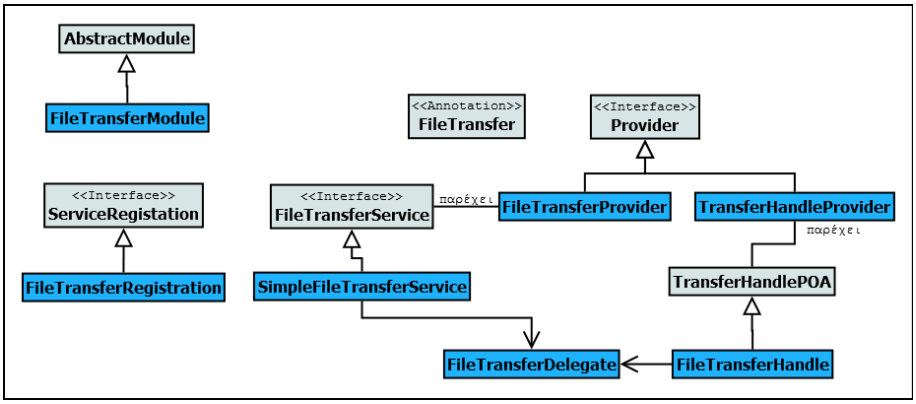
Σύμφωνα με την ανάλυση της αρχιτεκτονικής για να προστεθούν οι υπηρεσίες στο σύστημα αρκεί να υλοποιηθούν τρεις συγκεκριμένες διεπαφές/κλάσεις. Η κλάση AbstractModule περιγράφει την παραμετροποίηση της υπηρεσίας, ποιες υλοποιήσεις αντιστοιχούν σε ποιες διεπαφές. Η διεπαφή Provider είναι απαραίτητη για την παροχή αντικειμένων της υπηρεσίας. Μάλιστα, μέσω των Provider καθορίζεται και ο κύκλος ζωής της. Ο τρόπος αρχικοποίησης και εγκατάστασης της υπηρεσίας στο σύστημα γίνεται δια μέσου της διεπαφής ServiceRegistration. Τέλος, είναι αναγκαίο ο ορισμός ενός annotation ο οποίος χρησιμοποιείται για τον καθορισμό ποιας υλοποίησης της ServiceRegistration θα χρησιμοποιηθεί δεδομένου ότι υπάρχουν και άλλες υπηρεσίες.

Από εκεί και πέρα χρειάζεται μια διεπαφή που θα περιγράφει και θα εκθέτει την λειτουργικότητα της υπηρεσίας. Απαιτείται η υλοποίηση του κομματιού που αφορά την CORBA. Τέλος, εσωτερικά σε κάθε module υπηρεσίας χρειάζεται να ακολουθηθεί μια δομή όπου θα διευκολύνει την επέκταση και την τροποποίηση της συμπεριφοράς αυτής.



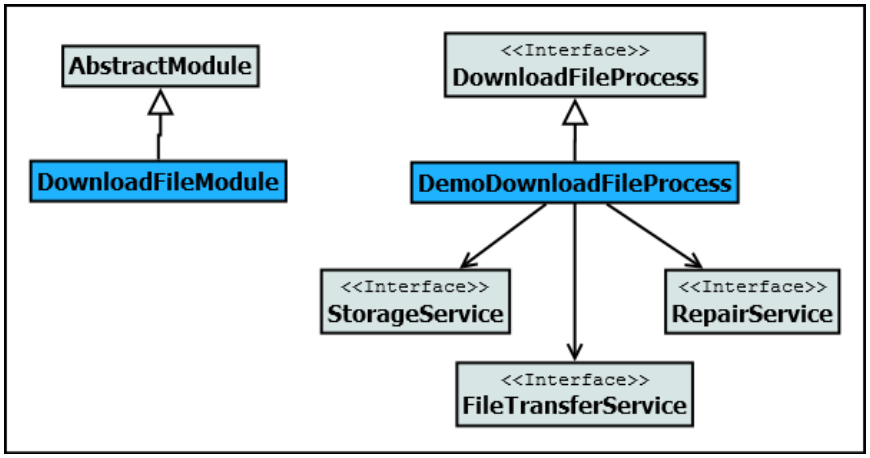
Σχήμα 7.2: Διάγραμμα κλάσεων υπηρεσίας StorageService

Υλοποίηση StorageService και FileTransferService. Στην εικόνα 7.2 φαίνεται το διάγραμμα κλάσεων της υπηρεσίας StorageService. Μπορούμε να δούμε τα αντικείμενα που υλοποιούν ή επεκτείνουν τις απαραίτητες διεπαφές/κλάσεις όπως έχει περιγραφεί παραπάνω. Οι κλάσεις StorageHandlePOA έχει παραχθεί από την idl και αφορά την CORBA. Παρέχεται Provider και για αυτή την διεπαφή. Στην εικόνα 7.3 φαίνεται το διάγραμμα κλάσεων της υπηρεσίας FileTransferService. Ακολουθείται αντίστοιχη λογική.



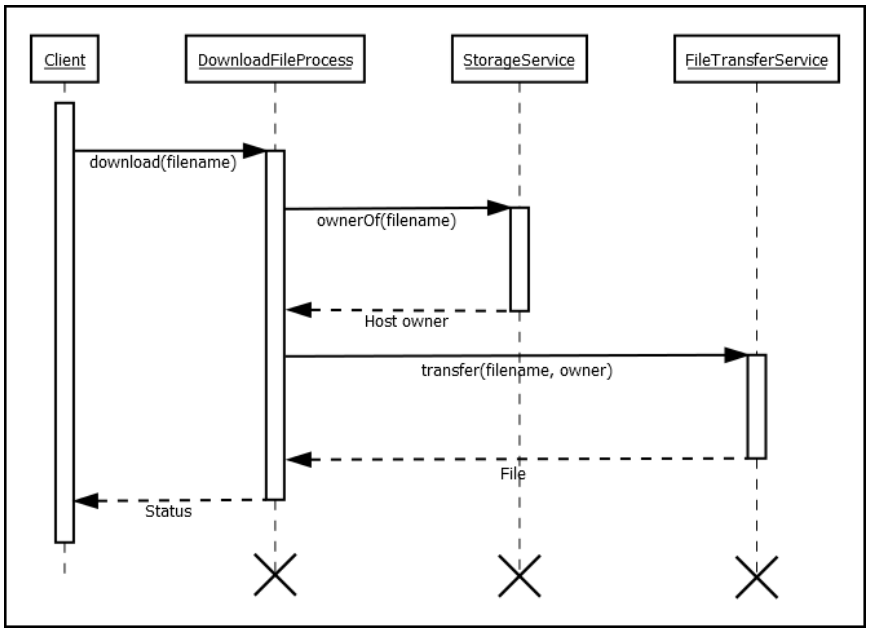
Σχήμα 7.3: Διάγραμμα κλάσεων υπηρεσίας FileTransferService

Υλοποίηση DownloadFileProcess. Αυτό που μένει είναι η υλοποίηση της διαδικασίας DownloadFileProcess. Το διάγραμμα κλάσεων φαίνεται στην εικόνα 7.4. Φαίνεται και η χρήση της υπηρεσίας επιδιόρθωσης RepairService που καταστά την διαδικασία fault-tolerant.



Σχήμα 7.4: Διάγραμμα κλάσεων διαδικασίας DownloadFileProcess

Στην εικόνα 7.5 φαίνεται το διάγραμμα ακολουθίας της διαδικασία όταν καλείται από τον χρήστη. Φαίνεται ξεκάθαρα πώς τα αντικείμενα σε επίπεδο διαδικασίας αλληλεπιδρούν. Συγκεκριμένα φαίνεται η περίπτωση όπου το αρχείο υπάρχει σε κάποιον peer το δικτύου ο οποίος και επιστρέφεται όταν τερματίζει η εκτέλεση της υπηρεσίας StorageService. Η διαδικασία επιστρέφει στον χρήστη την κατάσταση στην οποία τερμάτισε. Οι καταστάσεις που έχουν οριστεί στην διεπαφή είναι FILE_NOT_FOUND, NETWORK_ERROR και FILE_DOWNLOADED. Επίσης, φαίνεται και ο κύκλος ζωής των υπηρεσιών αυτών. Σε νέα αίτηση θα κατασκευαστούν εκ νέου τα αντικείμενα. Κάτι αντίστοιχο συμβαίνει όταν ο peer δέχεται αιτήσεις για εξυπηρέτηση υπηρεσιών. Η διαφορά είναι ότι εκεί τα αντικείμενα που αφορούν την CORBA είναι ίδια μέχρι τον τερματισμό του peer.



Σχήμα 7.5: Διάγραμμα ακολουθίας της διαδικασίας DownloadFileProcess

Κεφάλαιο 8

Επίλογος

8.1 Συμπεράσματα

Κύριος στόχος της διπλωματικής εργασίας ήταν η σχεδίαση αρχιτεκτονικής ενδιάμεσου λογισμικού και η υλοποίηση της. Απαίτηση ήταν η δημιουργία πολύπλοκων πρωτοκόλλων. Αυτό το πετύχαμε μέσω μιας υπηρεσιοκεντρικής αρχιτεκτονικής όπου κάθε λειτουργία που προσφέρει το σύστημα αντιμετωπίζεται ως υπηρεσία. Η ευκολία σύνθεσης των υπηρεσιών με ελάχιστο κώδικα οδηγεί στην δημιουργία πολύπλοκων πρωτοκόλλων και διαδικασιών. Σε σύγκριση με την υλοποίηση του P-Grid αλλά και αντίστοιχες υλοποιήσεις άλλων συστημάτων που ακολουθούν παρόμοιο αρχιτεκτονικό μοντέλο, κάτι τέτοιο απαιτεί αρκετές αλλαγές και προσθήκες από πλευράς κώδικα. Επιπλέον η υπηρεσία κρίνουμε πως είναι η σωστή αφαιρετική έννοια που χωρίζει το σύστημα σε διακριτές μονάδες. Η κατανόηση του συστήματος είναι πολύ πιο εύκολη με αυτόν τον τρόπο.

Ο δεύτερο στόχος της διπλωματικής που ήταν ταυτόχρονα και απαίτηση κατά τη φάση σχεδίασης της αρχιτεκτονική ήταν η παροχή ενός μηχανισμού συντήρησης και ανοχής λαθών. Το πρωτόκολλο που αναπτύχθηκε επιλύει αυτό το πρόβλημα χωρίς την τεχνική της αντιγραφής (replication) διατηρώντας την τοπολογία του δικτύου. Η ανθεκτικότητα του δικτύου μπορεί να αυξηθεί με την συνεργασία του πρωτοκόλλου μαζί με replication χωρίς αλλαγές. Μπορεί εύκολα να δημιουργηθεί μια υπηρεσία replication και να συνθέσουμε μια διαδικασία μαζί με την υπηρεσία διόρθωσης όπως έχει περιγραφεί.

8.2 Μελλοντική Εργασία

Η υλοποίηση του συστήματος με βάση την αρχιτεκτονική που αναλύθηκε έχει αρκετές ελλείψεις όσον αφορά την λειτουργικότητα που προσφέρει. Η προσθήκη λειτουργικότητας ενός κατανεμημένου συστήματος αποθήκευσης είναι σημαντική. Πέρα από την αποθήκευση δεδομένων προσφέρεται και η δυνατότητα επερωτήσεων πάνω σε αυτά. Φυσικά έχουμε διάφορες προκλήσεις όπως η αναζήτηση δεδομένων σε ένα εύρος κλειδιών όπως έχει περιγραφεί στη δημοσίευση

[42].Επίσης, η υλοποίηση πρωτοκόλλου εξισορρόπησης φορτίου (load-balancing) είναι απαραίτητο για την παραπάνω λειτουργικότητα.

Η σύνθεση των υπηρεσιών σε διαδικασίες μπορεί αναπαρασταθεί με ροές εργασίας (workflow) κατά τα πρότυπα των Web Service. έχουμε μια καλύτερη περιγραφή της διαδικασίας Οπότε η δημιουργία μιας μηχανής εκτέλεσης ροών εργασίας διευκολύνει αρκετά.

Όπως αναλύσαμε στην υλοποίηση της αρχιτεκτονικής, πολλά πράγματα επαναλαμβάνονται. Η αυτοματοποίηση δημιουργίας μιας υπηρεσίας, μέσω ενός εργαλείου που θα παράγει βασικές κλάσεις και διεπαφές, είναι χρήσιμη.

Βιβλιογραφία

[1] S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4):335--371, December 2004.

[2] S. Q. Zhuang, D. Geels, I. Stoica and R. H. Katz. On failure detection algorithms in overlay networks. In *IEEE INFOCOM*, volume 3, pages 2112--2123, 2005.

[3] R. Rodrigues, B. Liskov and L. Shrira. The design of a robust peer-to-peer system. In *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, EW 10, pages 117--124, New York, NY, USA, 2002. ACM.

[4] K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. *Proceedings of the 9th International Conference on Cooperative InformationSystems*, pages 179--194, 2001.

[5] K. Aberer, L.O. Alima, A. Ghodsi, S. Girdzijauskas, S. Haridi, and M. Hauswirth. The Essence of P2P: A Reference Architecture for Overlay Networks. *Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05)*, pages 11--20, 2005.

[6] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-Grid : A Self-Organizing Structured P2P System.

[7] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, R. Schmidt, and J. Wu. Advanced Peer-to-Peer Networking : The P-Grid System and its Applications. *PIK Praxis der Informationsverarbeitung und Kommunikation*, 26(2):86--89, 2003.

[8] K. Aberer, A. Datta, and M. Hauswirth. Efficient, Self-Contained Handling of Identity in Peer-to-Peer Systems. *IEEE Transactions on Knowledge and Data Engineering*, 16(07):858--869, July 2004. ISSN 1041-4347.

[9] K. Aberer, A. Datta, and M. Hauswirth. Route Maintenance Overheads in DHT Overlays. *The 6th Workshop on Distributed Data and Structures*, 2003.

[10] K. Aberer and M. Hauswirth. An Overview on Peer-to-Peer Information Systems. *Workshop on Distributed Data and Structures*, pages 1--14.

[11] K. Aberer, P. Magdalena, M. Hauswirth, and R. Schmidt. Access in P2P Systems. (February), 2002.

[12] A. Tanenbaum, S. Andrew and M. van Steen. *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006. ISBN 0132392275.

[13] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz and I. Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. In *Peer-to-Peer Systems II*, Lecture Notes in Computer Science, pages 33--44--44. Springer Berlin / Heidelberg, 2003.

[14] K. Aberer, L. O. Alima, A. Ghodsi, S. Girdzijauskas, S. Haridi and M. Hauswirth. The essence of p2p: A reference architecture for overlay networks. In *In P2P2005, The 5th IEEE International Conference on Peer-to-Peer Computing*, pages 11--20, 2005.

[15] R. Schmidt. The P-Grid System - Overview. Technical report, School of Computer and Communication Sciences, École Polytechnique Fédéral de Lausanne (EPFL), 2007.

[16] D. Grolimund. A Pattern Language for Overlay Networks. Design Patterns in Peer-to-Peer Systems. Technical report, ETH, Department of Computer Science, 2005.

[17] D. Grolimund and P. Müller. A pattern language for overlay networks in peer-to-peer systems. In U. Zdun and L. B. Hvatum, editors, *EuroPLoP 2006, Eleventh European Conference on Pattern Languages of Programs, Irsee, Germany, July 5-9, 2006*, pages 95--140. UVK - Universitaetsverlag Konstanz, 2006.

[18] M. Amoretti, M. Reggiani, F. Zanichelli and G. Conte. Peer: an Architectural Pattern Enabling Resource Sharing in Virtual Organizations. In *PLoP '05: Proceedings of the 2005 conference on Pattern languages of programs*, 2005.

[19] M. Amoretti, F. Zanichelli, and G. Conte. Sp2a: a service-oriented framework for p2p-based grids. In *Proceedings of the 3rd international workshop on Middleware for grid computing*, MGC '05, pages 1--6, New York, NY, USA, 2005. ACM.

[20] JXTA v2.0 Protocols Specification, 2007.

[21] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown and R. Metz. Reference Model for Service Oriented Architecture 1.0, oct 2006.

[22] P. Bianco, G. A. Lewis, P. Merson and S. Simanta. Architecting Service-Oriented Systems. Technical report, Carnegie Mellon, August 2011.

[23] Reference Architecture Foundation for Service Oriented Architecture Version 1.0, July 2011.

[24] D. Duggan. *Enterprise Software Architecture and Design. Entities, Services, and Resources*. John Wiley & Sons, 2012.

[25] B. Meyer. *Object-Oriented Software Construction*. Second edition, December 1994. doi: 10.1016/0168-9002(94)91548-2.

[26] F. Buschmann, K. Henney and D. C. Schmidt. *Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing*. John Wiley & Sons, 2007.

[27] M. Fowler. Inversion of Control Containers and the Dependency Injection Pattern, 2004. URL `http://martinfowler.com/articles/injection.html`.

[28] J.C.D.A. Dan Malks, J. Crupi, and D. Malks. *Core J2EE Patterns: Best Practicies and Design*. Core Design Series. Prentice Hall Ptr, 2003.

[29] S. Freeman, N. Pryce, T. Mackinnon and J. Walnes. Mock roles, not objects. In *In OOPSLA '04: Ccompanio To The 19TH Annual ACM SIGPLAN Conference On Object-Oriented Programming Systems, Languages, Aan Applications*, pages 236--246. ACM Press, 2004.

[30] E. Gamma, R. Helm, R. Johnson and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[31] J. F. Buford, H. Yu and E. K. Lua. *P2P: Networking and Applications*. Morgan Kaufmann Publishers Inc., 2009. ISBN 9780123742148.

[32] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Middleware '01, pages 329--350, London, UK, UK, 2001. Springer-Verlag. ISBN 3-540-42800-3.

[33] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22:41--53, 2004.

[34] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 149--160, New York, NY, USA, 2001. ACM. ISBN 1-58113-411-8.

[35] OMG. *CORBA Component Model Specification, Version 4.0*. Object Management Group, April 2006.

[36] OMG. *Common Object Request Broker Architecture (CORBA) Specification, Version 3.1, Part 1: CORBA Interfaces* . Object Management Group, January 2008.

[37] OMG. *Common Object Request Broker Architecture (CORBA) Specification, Version 3.1, Part 2: CORBA Interoperability* . Object Management Group, January 2008.

[38] OMG. *Common Object Request Broker Architecture (CORBA) Specification, Version 3.1, Part 3: CORBA Component Model* . Object Management Group, January 2008.

[39] Gerald Brose, Andreas Vogel, and Keith Duddy. *Java Programming with CORBA, Third Edition*. John Wiley & Sons, Inc., New York, NY, USA, 3rd edition, 2001. ISBN 0471376817.

[40] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558--565, July 1978. ISSN 0001-0782.

[41] A. Hunt and D. Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional, 1999.

[42] V. Samoladas M. Argyriou and S. Blanas. Grasp: generalized range search in peer-to-peer networks. In *Proceedings of the 3rd international conference on Scalable information systems*, InfoScale '08, pages 23:1--23:10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).