ΤΕΧΝΟΛΟΓΙΚΟ
ΕΚΠΑΙΔΕΥΤΙΚΟ
ΙΔΡΥΜΑ
ΜΕΣΟΛΟΓΓΙΟΥ

ΤΕΣΥΑ, Περιφέρεια Νοτιάδων

ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΜΕΣΟΛΟΓΓΙΟΥ
ΤΜΗΜΑ ΤΗΛΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΘΕΜΑ

«Πράκτορας σε περιβάλλον Google Wave για την γεφύρωση με δίκτυα VOIP με χρήση τεχνολογιών σύνθεσης φωνής»

ΣΑΠΡΑΝΙΔΗΣ ΜΑΞΙΜΟΣ

ΑΛΕΦΡΑΓΚΗΣ ΠΑΝΑΓΙΩΤΗΣ

ΝΑΥΠΑΚΤΟΣ 2011

Περιεχόμενα.

1. Εισαγωγή.

Η παρούσα εργασία έχει ως στόχο τον σχεδιασμό και την ανάπτυξη μιας εφαρμογής η οποία θα επιτρέψει την χρήση του Google Wave Federation protocol σε συνδυασμό με τεχνολογίες οι οποίες είναι ευρέως διαδεδομένες στον χώρο του VoIP.
Η εργασία αυτή απευθύνεται σε όσους ενδιαφέρονται για τις τεχνολογίες που χρησιμοποιούνται στα δίκτυα επόμενης γενιάς και στον τρόπο με τον οποίο μπορούμε να πραγματοποιήσουμε κλήσεις μέσω internet, με την βοήθεια κινητών συσκευών.
Ακόμα είναι μία πολύ καλή μελέτη για τον τρόπο με τον οποίο μπορούμε να πραγματοποιήσουμε σύγκληση μεταξύ δύο ανόμοιων δικτύων ώστε να αξιοποιήσουμε τις δυνατότητες τους. Πιο συγκεκριμένα θα παρουσιάσουμε και θα μελετήσουμε το Google Wave Federation protocol και τον τρόπο με τον οποίο μπορούμε να επιτρέψουμε την χρήση του για την πραγματοποίηση κλήσεων φωνής.
Κατά την διάρκεια τις εργασίας θα παρουσιάσουμε τεχνολογίες οι οποίες έχουν να κάνουν με κινητές συσκευές και τις ιδιαιτερότητες τις οποίες παρουσιάζει το περιβάλλον ανάπτυξης. Επίσης θα μιλήσουμε για το Jslee το οποίο αποτελεί ένα Service Delivery Platform (SDP) και χρησιμοποιείται από πολλές εταιρίες τηλεπικοινωνιών με στόχο την παροχή υπηρεσιών σε δίκτυα επόμενης γενιάς.
Τέλος θα παρουσιάσουμε ένα καινούργιο πρωτόκολλο το οποίο παρότι δεν έχει την δυνατότητα για μετάδοση σηματοδοσίας κλήσεων, θα προσπαθήσουμε να το χρησιμοποιήσουμε ώστε να εκμεταλλευτούμε όλα τα χαρακτηριστικά τα οποία προσφέρει.
Το τελικό σύστημα θα μπορεί να χρησιμοποιηθεί τόσο στο περιβάλλον του Google Wave (του προιόντος) όσο και με τρόπο τέτοιο που να επιτρέπει κλήσεις προς άλλους παρόχους VoIP.

1.1 Ιστορική αναδρομή.

Η ανάγκη για επικοινωνία υπάρχει από την αρχή της ιστορίας του ανθρώπου. Αρχικά η ανθρώπινη επικοινωνία περιοριζόταν από την απόσταση, κάτι που αργότερα ξεπεράστηκε μέσω διαφόρων άλλων τρόπων επικοινωνίας όπως ήταν οι φρυκτορίες, το ακουστικό κέρας και ο τηλέγραφος που είναι ο πρόγονος του σημερινού τηλεφώνου.
Μετά την ανακάλυψη του τηλεφώνου οι χρήστες μπορούσαν να χρησιμοποιήσουν το τηλέφωνο για να κάνουν μόνο κλήσεις φωνής. Αργότερα προστέθηκαν και άλλες υπηρεσίες που σήμερα είναι πολύ δημοφιλής μεταξύ των χρηστών, όπως είναι τα γραπτά μηνύματα τύπου SMS, ο αυτόματος τηλεφωνητής και άλλα.
Σήμερα οι περισσότεροι άνθρωποι χρησιμοποιούν την τηλεφωνία ως το κύριο μέσο επικοινωνίας καθώς όμως περνούν τα χρόνια νέες μέθοδοι έχουν αρχίσει και εμφανίζονται με σκοπό την καλύτερη και πιο αποτελεσματική εξυπηρέτηση των χρηστών. Αυτό γίνεται προσπαθώντας να αναβαθμιστεί το επίπεδο των παρεχόμενων υπηρεσιών και να προσαρμοστούν οι νέες τεχνολογίες στις συνήθειες των χρηστών.

Καθώς η παραδοσιακή τηλεφωνία κέρδιζε την εμπιστοσύνη των ανθρώπων, μία νέα τεχνολογική ανακάλυψη έγινε, το διαδίκτυο.
Η ραγδαία εξέλιξη των ηλεκτρονικών υπολογιστών και κυρίως των δικτύων τον περασμένο αιώνα, έκανε πρόσφορο το έδαφος για την εισαγωγή σε πιο εξελιγμένες μορφές επικοινωνίας, όπως το ηλεκτρονικό ταχυδρομείο (e-mail), αλλά και διαφόρων εφαρμογών άμεσων μηνυμάτων (instant messaging) και συνδιαλέξεων μέσω video(video conferencing).Επιπλέον η εξάπλωση των δικτύων που βασίζονται στο πρωτόκολλο IP άρχισε να γίνεται ολοένα και μεγαλύτερη και τα δίκτυα που εξυπηρετούσαν την κίνηση, αποδοτικότερα και πιο αξιόπιστα. Αυτό διευκόλυνε την εξάπλωση των τεχνολογικών VoIP, το οποίο είναι ένας συνδυασμός των παραδοσιακών δικτύων κυκλώματος με τα δίκτυα μετάδοσης πακέτων και διαφέρει κυρίως στον τρόπο με τον οποίο μεταδίδεται τόσο η σηματοδοσία μιας κλήσης καθώς και η κλήση αυτή καθ'αυτή.

1.2 Ασύγχρονη επικοινωνία.

Στο χώρο τον τηλεπικοινωνιών, ένα ασύγχρονο σύστημα είναι αυτό το οποίο επιτρέπει την επικοινωνία μεταξύ δύο συμβαλλόμενων μερών χωρίς να υπάρχει κάποιος χρονικός περιορισμός, δηλαδή είναι η επικοινωνία που μπορεί να πραγματοποιηθεί σε οποιοδήποτε χρονικό διάστημα. Αυτό πρακτικά σημαίνει ότι οι υπολογιστές δεν χρειάζονται κάποιο εξωτερικό ρολόι για να συγχρονίσουν τον τρόπο που επικοινωνούν .Για τους χρήστες των συστημάτων, αυτό μεταφράζεται στο ότι δε χρειάζεται να συγχρονιστούν ώστε να επικοινωνήσουν, αλλά μπορούν να "αφήσουν" ο ένας στον άλλο ένα μήνυμα ώστε να το επεξεργαστεί και να απαντήσει όταν αυτός μπορεί. Στην πραγματικότητα, αυτού του τύπου η επικοινωνία είναι ευρέως διαδεδομένη μεταξύ ανθρώπων, αφού λόγο των χρονικών περιορισμών που μας επιβάλλονται από τον σύγχρονο τρόπο ζωής γίνεται ολοένα και δυσκολότερο να συγχρονιστούμε. Αυτού του είδους η επικοινωνία χρησιμοποιείται από πολλές εφαρμογές όπως το e-mail τα SMS και τα προγράμματα άμεσου μηνύματος.
Τα κυριότερα μειονεκτήματα της μορφής αυτής είναι ότι μπορεί να είναι αργή και αναξιόπιστη κυρίως όταν επιβάλλεται να γνωρίζουμε ότι το μήνυμα μας έχει παραδοθεί, αλλά επίσης δεν βοηθάει σε συζητήσεις που ανταλλάσσονται πολλά μηνύματα.

σαν ένα ανεξάρτητο επίπεδο.
Με αυτήν την λογική μπορούμε να πούμε ότι το RTP είναι ένα πλαίσιο (framework) το οποίο έχει αφεθεί ημιτελές επίτηδες ώστε να μπορεί να προσαρμοστεί στης εκάστοτε ανάγκες. Όταν χρησιμοποιούνται μαζί το RTP και το RTCP τότε για το καθένα λαμβάνουμε μία πόρτα σε κάθε συμμετέχοντα. Για λόγους ευκολίας συνήθως το RTCP θα πάρει τον αμέσως μεγαλύτερο αριθμό πόρτας από ότι θα έχει πάρει το RTP.

Το RTP μπορεί να χρησιμοποιηθεί σε συνδυασμό με διάφορα επίπεδα μεταφοράς αλλά ο πιο συνηθισμένος συνδυασμός χρησιμοποιεί το UDP. Αυτό γίνεται ώστε να εκμεταλλευτούμε τις ικανότητες του για γρήγορη μεταφορά, αλλά και να χρησιμοποιήσουν τις ικανότητες που προσφέρει για πολυπλεξία.
Στο RTP έχουμε τους εξής ορισμούς:

- Payload: είναι τα δεδομένα που μεταδίδονται μέσα σε ένα πακέτο RTP.
- Packet: ένα πακέτο RTP αποτελείτε από μία επικεφαλίδα(όπως ορίζεται από το RTP), μία λίστα που περιέχει όλα τα συμβαλλόμενα μέρη, και τα περιεχόμενα του Payload. Αναλόγως το πρωτόκολλο που χρησιμοποιούμε για την μεταφορά, μπορεί να χρειαστεί να κάνουμε χρήση του μηχανισμού του encapsulation.
- Port: είναι η αφαιρετική έννοια που χρησιμοποιείται από το επίπεδο μεταφοράς για να ξεχωρίσει τις υπηρεσίες μιας διεύθυνσης.
- Transport address: είναι ο συνδυασμός μιας διεύθυνσης δικτύου και του αριθμού Port, με τον οποίο μπορούμε να αναγνωρίσουμε την υπηρεσία στην μεριά του δέκτη.
- Media type: είναι ένα σύνολο από είδη δεδομένων τα οποία μπορούν να μεταφερθούν στα πλαίσια μίας μόνο συνόδου RTP.
- Multimedia session: ένα σύνολο από παράλληλες συνόδους RTP, που υπάρχουν ανάμεσα σε μία ομάδα συμμετεχόντων.
- Session: είναι μία συσχέτιση ανάμεσα στα συμβαλλόμενα μέρη που επικοινωνούν κάνοντας χρήση του RTP.
- Synchronization source (SSRC): Η πηγή από την οποία ξεκινάει η ροή πακέτων RTP, αναγνωρίζεται από έναν αριθμό 32 bit ο οποίος βρίσκεται αποθηκευμένος μέσα στην επικεφαλίδα των πακέτων RTP. Αυτό γίνεται ώστε να μην είναι συνδεδεμένος με τις διευθύνσεις δικτύου και έτσι να μπορούμε πάντα να αναγνωρίζουμε τον κωδικό αυτό. Όλα τα πακέτα που έχουν αποσταλεί από τον ίδιο αποστολέα σε μία δεδομένη χρονική στιγμή, και με τον ίδιο αύξοντα αριθμό, συγκεντρώνονται στον παραλήπτη ώστε να γίνει η αναπαραγωγή τους. Αυτό γίνεται γιατί μπορεί ο αποστολέας να μεταβάλλει τα πακέτα του σύμφωνα με κάποιες παραμέτρους που λαμβάνει υπόψη του. Έτσι άμα αλλάξει ο codec θα πρέπει να αναπαραχθούν με διαφορετικό τρόπο τα πακέτα.
- End System: ένα πρόγραμμα το οποίο παράγει τα δεδομένα τα οποία θέλουμε να σταλούν μέσω του RTP, ή ένα πρόγραμμα το οποίο καταναλώνει τέτοια πακέτα.

Για να μπορέσουν οι δύο χρήστες να επικοινωνήσουν θα πρέπει να στέλνουν τα δεδομένα του ήχου σε μικρά κομμάτια τα οποία θεωρούμε ότι δεν ξεπερνάνε τα 20ms σε διάρκεια. Κάθε τέτοιο κομμάτι ήχου έχει μία αντίστοιχη επικεφαλίδα RTP, τα οποία μπαίνουν μέσα σε ένα πακέτο UDP. Η επικεφαλίδα του RTP χρησιμεύει για να μπορούμε να ξέρουμε τον τύπο της κωδικοποίησης του ήχου(PCM,ADPCM) έτσι ώστε αυτό να μπορεί να αλλάξει αν χρειαστεί όπως στην περίπτωση που κάποιος που έχει κακή σύνδεση προσπαθήσει να συνδεθεί στην συζήτηση. Ακόμα επειδή η συνομιλία αυτή θα πραγματοποιηθεί μέσω του διαδικτύου και ένα χαρακτηριστικό αυτού είναι ότι συχνά τα πακέτα χάνονται και ότι κανείς δεν εγγυάται ότι τα πακέτα θα φτάσουν με την σειρά που στάλθηκαν. Για αυτόν τον λόγο στην επικεφαλίδα του RTP περιέχονται πληροφορίες που σχετίζονται με τον χρόνο, όπως επίσης και αριθμός ακολουθίας πακέτου έτσι ώστε να είναι δυνατό για τον παραλήπτη να βάλουν στην σωστή σειρά τα πακέτα αλλά και να μπορεί να εξάγει στατιστικά στοιχεία για την ποιότητα της επικοινωνίας. Αυτό είναι αρκετά χρήσιμο για τον λόγο ότι κατά την διάρκεια μια συνομιλίας διάφοροι χρήστες μπορούν να μπούνε και να φύγουν από την συνομιλία, αυτό σημαίνει ότι ο καθένας μπορεί να έχει διαφορετικό τύπο γραμμής και διαφορετική υποστήριξη στους

διάφορους τύπους κωδικοποίησης κάτι που δείχνει και την ελαστικότητα του πρωτοκόλλου που αναφέραμε παραπάνω. Όλα τα στατιστικά σχετικά με την ποιότητα της γραμμής αλλά και για να ξέρουν όλοι ποιος είναι παρών σε μια συζήτηση, τα προγράμματα που χρησιμοποιούν οι χρήστες, μπορούν να στέλνουν περιοδικά μία ενημέρωση που θα περιέχει το όνομα του χρήστη και μία αναφορά σχετικά με την ποιότητα του ήχου.
Για να μπορέσει κάποιος να φύγει από μία συζήτηση που είναι μέλος τότε στέλνει το RTCP πακέτο BYE.

2.3 Το πρωτόκολλο SDP(Session Description Protocol)

Το πρωτόκολλο SDP χρησιμοποιείται για να περιγράψουμε συνόδους μεταξύ χρηστών οι οποίες περιέχουν πολυμέσα, ή καλύτερα για να είμαστε σε θέση να περιγράψουμε τα χαρακτηριστικά με βάση τα οποία επικοινωνούν. Πιο αναλυτικά να περιγράψουμε τα πακέτα που προορίζονται για πρόσκληση έναρξης συνόδου, αναγγελίας συνόδου και γενικά όλων των μορφών αρχικοποίησης μιας συνόδου.
Το SDP είναι μόνο ένας τρόπος για να περιγράψουμε μία σύνοδο. Δεν μπορεί να χρησιμοποιηθεί για να μεταφέρει τα πραγματικά αρχεία που χρειάζονται για μία σύνοδο, γι'αυτόν τον λόγο και χρησιμοποιείται μαζί με άλλα πρωτόκολλα τα οποία κάνουν την μεταφορά των δεδομένων. Κάποια από αυτά τα πρωτόκολλα είναι το SIP και το RTP που περιγράφηκαν παραπάνω.
Έχει σχεδιαστεί ετσι ώστε να είναι αρκετά γενικού σκοπού με στόχο να μπορεί να χρησιμοποιείται στην ευρύτερη περιοχή των δικτύων. Παρολ'αυτά δεν έχει δημιουργηθεί για να μπορεί να διαπραγματευτεί το περιεχόμενο των δεδομένων ούτε και την κωδικοποίηση που θα έχουν, έτσι μπορούμε να το θεωρήσουμε σαν έναν τρόπο ώστε να "θυμούνται" τα συμβαλλόμενα μέρη τα χαρακτηριστικά μιας συνόδου όπως είναι τα Codec με τα οποία θα γίνει η κωδικοποίηση και αποκωδικοποίηση του ήχου.

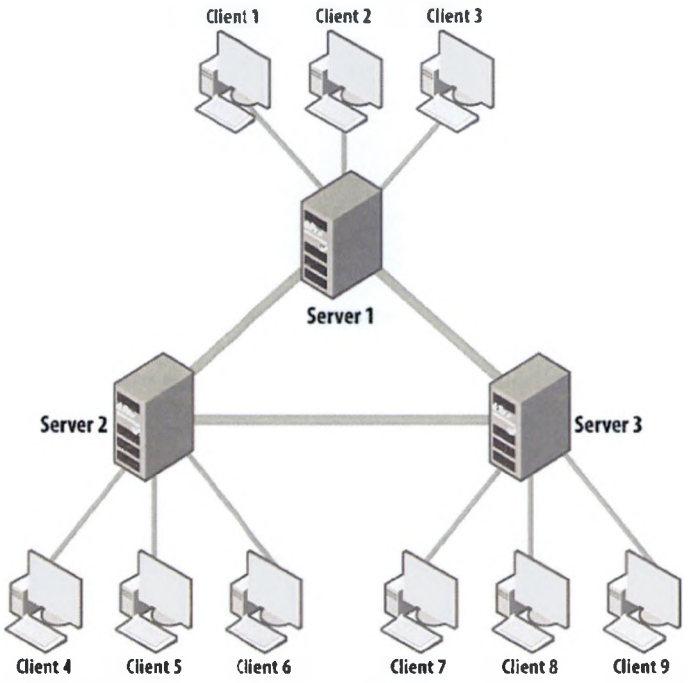
2.4 Το VOIP

Από τότε που άρχισε να χρησιμοποιείται το τηλέφωνο μέχρι τις αρχές της δεκαετίας του 1990, τα τηλεφωνικά δίκτυα δεν άλλαξαν και πολύ σε φιλοσοφία. Φυσικά υπήρχαν πολλές βελτιώσεις στον τομέα τον παρεχόμενων υπηρεσιών όπως είναι η αναγνώριση κλήσης, ανακατεύθυνση κλήσης καθώς και κάποιες άλλες υπηρεσίες. Στις αρχές της δεκαετίας του 1990 άρχισε μία προσπάθεια με σκοπό την μεταφορά τόσο φωνής όσο και video μέσω IP δικτύων γνωστό σήμερα ως VoIP(Voice Over IP). Η λογική που ακολουθήθηκε ήταν απλή: παίρνουμε τα κομμάτια φωνής ή video, τα χωρίζουμε σε μικρά κομμάτια και τα μεταδίδουμε μέσω του δικτύου. Στο άλλο άκρο τα κομμάτια ανασυντίθενται και έτσι οι δύο χρήστες μπορούν να επικοινωνήσουν. Το VOIP είναι πολύ σημαντικό γιατί αντίθετα με την άποψη που έχει επικρατήσει δεν περιορίζεται μόνο στο να μεταφέρει φωνή και άρα να περιορίζεται μόνο για τηλεπικοινωνιακές κλήσεις, άλλα για να μεταφέρει σε πραγματικό χρόνο φωνή video και γενικότερα δεδομένα όπως είναι για παράδειγμα το κείμενο. Αυτό μας επιτρέπει να χρησιμοποιήσουμε μία μόνο γραμμή για να μεταφέρουμε όλες τις τηλεπικοινωνιακές ανάγκες μας και είναι γνωστό σαν σύγκλιση, κάτι πολύ σημαντικό γιατί βοηθάει στην μείωση του κόστους, με ταυτόχρονη αύξηση των υπηρεσιών. Πρέπει να προσέξουμε ότι το VoIP μπορεί να συνεργαστεί με όλα τα πρωτόκολλα που αναφέρθηκαν ως τώρα. Στις μέρες μας πολλές εταιρίες προωθούν την χρήση υπηρεσιών που βασίζονται στο δίκτυο, όπως είναι οι εφαρμογές γεωγραφικής θέσης (Location Based), και έτσι η ευελιξία η οποία προσφέρεται από το VoIP στην ταχύτητα και ευκολία ανάπτυξης τέτοιων υπηρεσιών είναι ένας πολύ σημαντικός παράγοντας διάδοσης του. Σε αυτήν την ενότητα θα περιγράψουμε τον γενικό τρόπο λειτουργίας του VOIP, δεν θα αναλύσουμε την λειτουργία απλά θα αναφέρουμε με απλό και θεωρητικό τρόπο την λειτουργία του. Η γενική ιδέα είναι να μπορεί οποιοσδήποτε έχει πρόσβαση σε μία συσκευή με σύνδεση στο διαδίκτυο και τις απαραίτητες δυνατότητες να μιλήσει με άλλους χρήστες.

Όταν ένας χρήστης μιλάει, η φωνή του μετατρέπεται από το μικρόφωνο της συσκευής που χρησιμοποιεί και στέλνεται μέσω του διαδικτύου σε κάποιον άλλο χρήστη. Εκεί ακολουθείται η αντίστροφη διαδικασία ώστε ο λήπτης να ακούσει τελικά την φωνή του αποστολέα. Αυτό είναι το πιο απλό παράδειγμα που θα μπορούσαμε να παρουσιάσουμε αφού στην πραγματικότητα πριν στείλουμε ένα πακέτο θα πρέπει να έχουν ακολουθηθεί πολλά στάδια ώστε να μπορέσουμε να βρούμε την διαθεσιμότητα του παραλήπτη, τις ικανότητες για αναπαραγωγή που αυτός έχει, καθώς επίσης θα πρέπει να μετατρέψουμε την αναλογική φωνή σε ψηφιακή, και τέλος να επιτρέψουμε την επικοινωνία. Ο παραλήπτης χρησιμοποιεί τον ίδιο αλγόριθμο ώστε να μπορέσει να αποσυμπιέσει τα δεδομένα της φωνής μας και να αναπαράγει το περιεχόμενο. Ο αλγόριθμος συμπίεσης και αποσυμπίεσης είναι γνωστός σαν compressor/de-compressor ή για λόγους συντομίας Codec. Υπάρχουν πολλοί τύποι codec και για ένα πολύ μεγάλο φάσμα χρήσεων όπως είναι οι codecs που χρησιμοποιούνται για την συμπίεση ταινιών και αρχείων ήχου, ενώ για χρήση σε τηλεπικοινωνιακά συστήματα χρησιμοποιούνται κυρίως codecs τα οποία προσφέρουν ελαστικότητα στις αλλοιώσεις. Όταν τελικά θα καταφέρουμε να συμπιέσουμε την φωνή έρχεται η ώρα της αποστολής των δεδομένων μέσω του δικτύου. Επειδή όμως το δίκτυο μέσω του οποίου θα στείλουμε τα δεδομένα βασίζεται στο πρωτόκολλο IP, δεν μπορούμε να ξέρουμε αν όλα τα πακέτα που θα σταλούν θα φτάσουν στον προορισμό τους ή αν θα φτάσουν με την ίδια σειρά που στάλθηκαν. Έτσι κάθε δίκτυο που χρησιμοποιεί το VOIP να μπορεί να αντεπεξέλθει σε αυτά τα προβλήματα με τρόπο που είναι αποδεκτός και κατανοητός για τον παραλήπτη. Η διαδικασία
που αναλαμβάνει να γεμίσει τα κενά και να κάνει την φωνή αποδεκτή είναι γνωστή και σαν PLC(Packet Loss Concealment). Παρόλο που το VoIP χρησιμοποιείται με δίκτυα IP μπορεί κάποιος να χρησιμοποιήσει ένα ειδικό συστατικό δικτύου που ονομάζεται Media gateway έτσι ώστε να μπορέσει να "γεφυρώσει" τους δύο τύπους τηλεφωνίας. Με αυτόν τον τρόπο λειτουργούν και πολύ VoIP πάροχοι οι οποίοι προσφέρουν την δυνατότητα επικοινωνίας με όλους τους χρήστες ακόμα και με αυτούς που βρίσκονται εντός του παραδοσιακού τηλεφωνικού δικτύου.

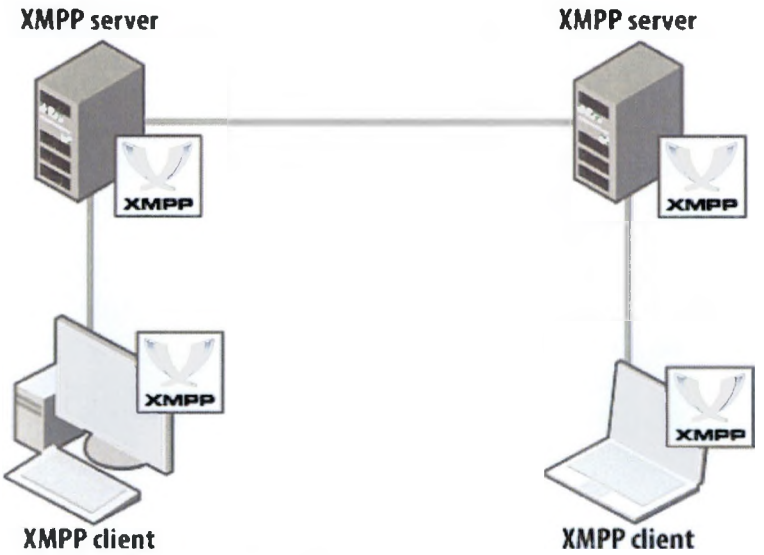
2.5 Το πρωτόκολλο XMPP

Το XMPP(extensible messaging and presence protocol) είναι ένα πρωτόκολλο που βοηθάει την επικοινωνία σε πραγματικό χρόνο χρησιμοποιώντας το XML σαν τρόπο αναπαράστασης των μηνυμάτων. Με άλλα λόγια μας επιτρέπει να στέλνουμε μικρά αρχεία XML άμεσα μεταξύ δύο ή και περισσότερων χρηστών. Επιγραμματικά μπορούμε να αναφέρουμε ότι το XMPP χρησιμοποιείται σε εφαρμογές instant messaging, σε δικτυακά παιχνίδια απομακρυσμένης διαχείρισης συστημάτων. Ακόμα σε πολλές περιπτώσεις χρησιμοποιείται σαν ενδιάμεσο λογισμικό σε εφαρμογές cloud computing άλλα και σε εφαρμογές VOIP.
Το XMPP δεν είναι ένα πρόγραμμα ανοιχτού κώδικα άλλα ένα standard, με αποτέλεσμα να μπορεί να χρησιμοποιηθεί από οποιονδήποτε. Επίσης ορίζει άλλα πρωτόκολλα και τύπους δεδομένων που πρέπει να χρησιμοποιηθούν ώστε να καταφέρουμε να έχουμε άμεση επικοινωνία.
Η αρχιτεκτονική του XMPP είναι όμοια σε πολλά σημεία με αυτήν που συναντάμε σε άλλα πολύ γνωστά πρωτόκολλα όπως είναι το HTTP . Αυτή είναι γνωστή σαν Client server αρχιτεκτονική. Αυτό έχει το προτέρημα ότι χωρίζει τα σημεία για τα οποία πρέπει να ανησυχούν οι προγραμματιστές, δηλαδή δίνει την ελευθερία στους προγραμματιστές που έχουν αναλάβει το κομμάτι του πελάτη να επικεντρωθούν στο γραφικό περιβάλλον και σε αυτούς που έχουν αναλάβει το κομμάτι του εξυπηρετητή στην ταχύτητα και την επεκτασιμότητα. Με αυτόν τον τρόπο επιτρέπεται στους εξυπηρετητές να εφαρμόσουν κάποιες υπηρεσίες που είναι διαθέσιμες από το πρωτόκολλο όπως αυτές της ασφάλειας. Τέλος, οι δημιουργοί του πρωτοκόλλου προσπάθησαν και έχουν καταφέρει να κρατήσουν το κομμάτι τον client απλό και να ωθήσουν όλη την επεξεργασία και την υπηρεσιακή λογική να γίνεται στον εξυπηρετητή, επιτρέποντας έτσι την παραπάνω αποδοχή του αφού ακόμα περισσότερες συσκευές μπορούν να το υποστηρίξουν.



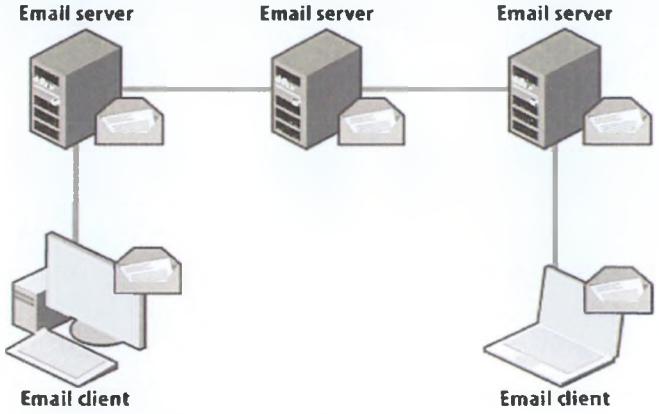
Εικόνα 2.1 XMPP example

Μια βασική ιδιότητα είναι ότι δίνεται στους εξυπηρετητές η δυνατότητα να επικοινωνήσουν με άλλους XMPP εξυπηρετητές ώστε να παρέχουν υπηρεσίες. Αυτό είναι γνωστό και σαν federation. Η κύρια διαφορά μεταξύ του fedaration που παρουσιάζει το XMPP (direct federation) και αυτού που παρουσιάζεται σε άλλα πρωτόκολλα όπως για παράδειγμα το e-mail (indirect federation), είναι ότι όταν πρέπει να σταλεί ένα μήνυμα μέσω ενός άλλου εξυπηρετητή δεν μετριέται κανένα hop στο δίκτυο δηλαδή γίνεται άμεσα. Ακόμα, εξίσου σημαντικό στοιχείο του πρωτοκόλλου είναι ότι επιτρέπει πέραν των γνωστών οντοτήτων που συναντάμε σε ένα δίκτυο που βασίζεται στην αρχιτεκτονική client-server (δηλαδή πέραν των γνωστών client και server), να ορίσουν και άλλες οντότητες που είναι γνωστές σαν bots.



Εικόνα 2.2: XMPP federation

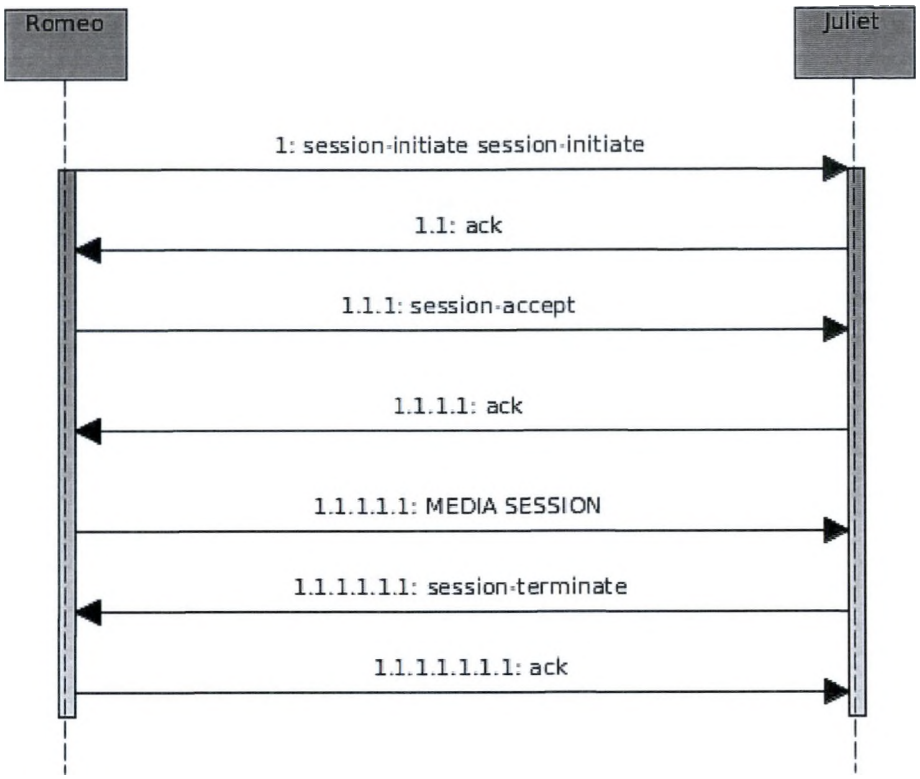
Αυτές οι οντότητες είναι αυτοματοποιημένα προγράμματα που παρέχουν κάποιου είδους υπηρεσία συνεισφέροντας με αυτόν τον τρόπο σε μία πιο ολοκληρωμένη εμπειρία για τους χρήστες.
για να μπορούν οι χρήστες να επικοινωνούν μεταξύ τους πρέπει να μπορεί ό ένας να αναγνωρίσει τον άλλον και καθένας να έχει μια μοναδική διεύθυνση με την οποία να μπορεί να χαρακτηριστεί. Αυτή η διεύθυνση είναι γνωστή σαν Jabber ID(JID) και όπως σε άλλα πρωτόκολλα (SIP,e-mail) είναι της μορφής max.sapranidis@xmpp-server.com όπου το max.sapranidis είναι το όνομα κάθε χρήστη και το xmpp-server.com το domain name. Όπως είναι εύκολα κατανοητό, οι χρήστες δεν υποχρεούνται να θυμούνται διευθύνσεις IP που είναι δυσνόητες διότι χρησιμοποιείται το σύστημα DNS ώστε να γίνει αντιστοίχιση των ονομάτων και των διευθύνσεων IP.
Μια διαφορά του xmpp είναι ότι για κάθε πακέτο που ανταλλάσσεται δεν υπάρχει κάποια επιβεβαίωση ότι το πακέτο έφτασε. Γενικά θεωρείται ότι κάθε πακέτο έχει φτάσει χωρίς να υπάρχει πρόβλημα όταν δεν στέλνεται πακέτο λάθους.



Εικόνα 2.3: SMTP Federation

2.6 Το πρωτόκολλο Jingle

Σκοπός του Jingle(XEP-0166), είναι να επιτρέψει σε οντότητες (Entities) του XMPP να εγκαθιδρύσουν μια ομότιμη επικοινωνία ώστε να ανταλλάξουν πολυμέσα.
Για το κομμάτι της ανταλλαγής μηνυμάτων(σηματοδοσία) χρησιμοποιείται το XMPP ενώ για την μεταφορά των πολυμέσων κάποιο άλλο πρωτόκολλο. Ο σχεδιασμός του είναι τέτοιος που επιτρέπει στους προγραμματιστές να το χρησιμοποιήσουν για μια μεγάλη γκάμα από εφαρμογές. Αυτό επιτυγχάνεται με την χρήση διαφόρων επεκτάσεων που είναι διαθέσιμες και αφορούν εφαρμογές ήχου, βίντεο, ανταλλαγής αρχείων, αλλά και υποστηρίξεις συνεργατικών λειτουργιών (Collaborative editing).
Για την μεταφορά των δεδομένων μπορούμε να χρησιμοποιήσουμε τόσο το UDP όσο και το TCP.
Αυτός ο τρόπος σχεδιασμού έχει επίπτωση και στην παρεχόμενη ασφάλεια αφού είναι δυνατόν να χρησιμοποιηθεί τόσο το TLS όσο και το DTLS, για την κρυπτογράφηση.
Στους αρχικούς στόχους του Jingle, ήταν να παρέχει τον τρόπο ώστε να σχεδιάσουμε προγράμματα για απλή μεταφορά φωνής και video και όχι να αποτελέσει ένα πλήρες τηλεφωνικό πρωτόκολλο που να παρέχει εξελιγμένες υπηρεσίες. Έτσι το πρωτόκολλο είναι απλό στην εκμάθηση του. Η αλληλουχία των μηνυμάτων για τον ορισμό μιας κλήσης φαίνεται στο παρακάτω σχήμα.



Εικόνα 2.4: Jingle Call flow

κάθε μήνυμα αναπαρίσταται με την βοήθεια ενός αρχείου XML το οποίο περιέχει τον τύπο του μηνύματος και διάφορες πληροφορίες σχετικά με τον αποστολέα και τον παραλήπτη. Παράδειγμα ενός τέτοιου μηνύματος, θα μπορούσε να είναι:

```
<iq from='romeo@montague.lit/orchard'
    id='zid615d9'
    to='juliet@capulet.lit/balcony'
    type='set'>
  <jingle xmlns='urn:xmpp:jingle:1'>
         action='session-initiate'
         initiator='romeo@montague.lit/orchard'
         sid='a73sjjvkla37jfea'>
     <content creator='initiator' name='this-is-a-stub'>
       <description xmlns='urn:xmpp:jingle:apps:stub:0'/>
       <transport xmlns='urn:xmpp:jingle:transports:stub:0'/>
     </content>
  </jingle>
</iq>
```

Από το παραπάνω μήνυμα, μπορούμε να παρατηρήσουμε τις ομοιότητες με το XMPP αλλά και τις διάφορες πληροφορίες που περιέχονται στο μήνυμα και έχουν να κάνουμε με τον σκοπό του μηνύματος, την ταυτότητα του , τον αποστολέα, τον παραλήπτη και το επίπεδο μεταφοράς που χρησιμοποιείται.

2.7 Wave Federation Protocol

Όπως αναφέραμε συνοπτικά στην εισαγωγή, το wave federation protocol, είναι μία επέκταση πάνω από το XMPP, και συγκεκριμένα πάνω από το XEP-116, με σκοπό την σχεδόν άμεση διάδοση των μηνυμάτων σε όλους τους παραλήπτες.
Αρχικά πρέπει να τονίσουμε ότι το federation protocol, είναι ένα πρωτόκολλο και όχι ένα προϊόν, ενώ το Google wave είναι ένα προϊόν το οποίο κάνει χρήση του πρωτοκόλλου.
Η αρχική ιδέα του πρωτοκόλλου ήταν να μπορεί ο καθένας να χρησιμοποιεί έναν πάροχο της επιλογής του, χωρίς όμως περιορισμούς στις προσφερόμενες υπηρεσίες. Έτσι επιτρέπεται η μετάδοση μηνυμάτων από χρήστες οι οποίοι βρίσκονται σε διαφορετικό πάροχο.
Αυτό οφείλεται στο μοντέλο το οποίο ακολουθεί το federation protocol και έχει να κάνει με τον τρόπο με τον οποίο πρέπει να λειτουργούν οι πάροχοι αυτοί.
Για να κατανοήσουμε τον μοντέλο αυτό θα πρέπει πρώτα να καταλάβουμε τον τρόπο με τον οποίο λειτουργούν οι υπηρεσίες αυτές.
Κάθε τέτοια υπηρεσία, έχει ένα Wave Store, το οποίο χρησιμοποιείται για αποθήκευση των διαθέσιμων διαδικασιών (Wave Operations), και ένα εξυπηρετητή ο οποίος χρησιμοποιείται για να εφαρμόζει τους επιχειρηματικούς μετασχηματισμούς (Operational Transformations), καθώς και να γράφει και να διαβάζει τους μετασχηματισμούς του Wavelet στο Store.
Επίσης η υπηρεσία του Wave μοιράζεται τα μηνύματα με άλλες ομότιμες υπηρεσίες όταν οι χρήστες τους βρεθούν μέσα σε ένα κοινό Wave.
Με αυτόν τον τρόπο, κάθε υπηρεσία Wave έχει ένα σύνολο από αποθηκευμένα Waves , κάποια από τα οποία είναι τοπικά (δηλαδή φιλοξενούνται από τον διακομιστή) και άλλα όχι. Το σημαντικό στοιχείο σε αυτό είναι ότι η υπηρεσία του Wave κρατάει τοπικά μια προσωρινή μνήμη με όλα τα αρχεία (cached), τα οποία γνωρίζει και απλά εφαρμόζει σε αυτά ότι αλλαγές που χρειάζονται, σύμφωνα με τις απαιτούμενες ενέργειες κάθε φορά (operations).
Για την αποτελεσματικότερη εφαρμογή των αλλαγών που χρειάζονται, ορίζονται δύο ξεχωριστές οντότητες, μία για να διαχειρίζεται τις αλλαγές τοπικά και μία για να ενημερώνει τις άλλες υπηρεσίες. Αυτές οι οντότητες ονομάζονται federation host και federation remote αντίστοιχα.
Για να γίνει η μεταφορά των αλλαγών, ο Wave πάροχος που επιθυμεί να ενημερώσει για μία αλλαγή, συνδέεται με τον αντίστοιχο πάροχο που πρέπει να λάβει την αλλαγή. Αφού η σύνδεση ολοκληρωθεί, η αλλαγή δρομολογείται προς τον ενδιαφερόμενο πάροχο. Για να διασφαλιστεί η αξιόπιστη μεταφορά των μηνυμάτων το πρωτόκολλο εφαρμόζει τον εξής μηχανισμό.
Κάθε Federation Host πρέπει να διατηρεί μια προσωρινή ουρά από ενέργειες που πρέπει να γίνουν(operations) για κάθε Domain στο οποίο γνωρίζει. Αυτή η ουρά αποθηκεύεται σε κάποια μνήμη για παραπάνω ασφάλεια. Οι ενέργειες κρατούνται στην μνήμη μέχρι ο αποδέκτης τους να ενημερώσει για την επιτυχή λήψη τους. Η διαδικασία ολοκληρώνεται αφαιρώντας κάθε ενέργεια που έχει επιβεβαιωθεί από την λίστα. Στην περίπτωση που δεν γίνει αυτό, ή στην περίπτωση που κατά την αποστολή αντιμετωπίσει οποιοδήποτε πρόβλημα, τότε η αποστολή θα επαναληφθεί, χρησιμοποιώντας exponential backoff αλγόριθμο, ώστε να βελτιστοποιηθεί η πιθανότητα επιτυχούς αποστολής, και να ελαχιστοποιηθεί η πιθανότητα πλημμύρας του αποδέκτη με αιτήματα.
Με τον παραπάνω τρόπο μπορούμε να ελαχιστοποιήσουμε σε μεγάλο βαθμό τον όγκο των μηνυμάτων τα οποία μεταφέρονται από το δίκτυο, αφού πρακτικά μεταφέρουμε μόνο εντολές από μία υπηρεσία Wave σε κάποια άλλη, με μοναδικό μειονέκτημα του περιορισμού που θέτει το XMPP για την ύπαρξη μιας αμφίδρομης επικοινωνίας καθ'όλη την διάρκεια της επικοινωνίας.

Αυτός είναι ο μηχανισμός των Delta. Τα Delta περιγράφονται σαν αλλαγές οι οποίες αν εφαρμοστούν σε έναν πόρο, τότε θα φτάσουμε στο ίδιο αποτέλεσμα. Έτσι στην περίπτωση μας, ένα Delta, περιγράφει τις αλλαγές οι οποίες πρέπει να γίνουν σε ένα μήνυμα XML ώστε να έχουμε την νέα εκδοχή του. Με αυτόν τον τρόπο αποφεύγουμε την ανταλλαγή ολόκληρου του μηνύματος, και περιοριζόμαστε στο να μεταφέρουμε μόνο το αντίστοιχο Delta. Έτσι κάθε εξυπηρετητής οφείλει να διατηρεί μια ενεργή σύνδεση για κάθε Wavelet στο οποίο έχει πρόσβαση, καθώς και να διαχειρίζεται μία συνεχή ροή από Delta.

2.8 JAIN-SLEE

Το Jain-Slee ή πιο απλά JSLEE είναι ένα standard το οποίο παράγεται απο το Java Community Process δηλαδή τον μηχανισμό που είναι υπεύθυνος για την ανάπτυξη των τεχνικών προδιαγραφών για τεχνολογίες που βασίζονται στην γλώσσα java.
Οι τεχνικές περιγραφές του είναι διαθέσιμες σε όλους τους ενδιαφερόμενους, και ορίζουν τον τρόπο με τον οποίο πρέπει να λειτουργεί τόσο το περιβάλλον στο οποίο τρέχουν τα προγράμματα (Container) όσο και τις ευθύνες και υποχρεώσεις τις οποίες έχει ο προγραμματιστής. Όπως επίσης και τους διαθέσιμους ρόλους και τις υποχρεώσεις αυτών (προγραμματιστές , διαχειριστές δικτύου κλπ.).
Η αρχιτεκτονική του JSLEE ορίζει ένα περιβάλλον το οποίο είναι στοχευμένο στο να εξυπηρετεί υπηρεσίες που είναι φτιαγμένες με στόχο τα επικοινωνιακά περιβάλλοντα, δηλαδή υπηρεσίες voip, ανταλλαγής άμεσων μηνυμάτων και άλλα.
Έτσι ορίζεται ένα μοντέλο βασισμένο σε συστατικά-οντότητες(component model), με τα οποία μπορούμε να συνθέσουμε ανάλογα με τις επιμέρους ανάγκες την λογική ενός προγράμματος. Τα συστατικά αυτά έχουν την δυνατότητα να μπορούν να ξαναχρησιμοποιηθούν, επιτρέποντας μας να έχουμε καλύτερο έλεγχο κατά την διάρκεια του σχεδιασμού πάνω στην λογική με την οποία θα συνθέσουμε το πρόγραμμα μας, αφού έτσι μπορούμε να χωρίσουμε και να συνθέσουμε αυτά τα συστατικά αναλόγως τις ανάγκες ενός προβλήματος, σε άλλες οντότητες, που είναι υψηλότερου επιπέδου και πιο περίπλοκες. Επίσης η αρχιτεκτονική SLEE ορίζει τις σχέσεις μεταξύ τον συστατικών-υπηρεσιών και του ίδιου του περιβάλλοντος πάνω στο οποίο τρέχουν (container).
Ακόμα οι προδιαγραφές υποστηρίζουν την ανάπτυξη application server οι οποίοι είναι σε θέση να παρέχουν υψηλή διαθεσιμότητα και κλιμακούμενη κατανομή (scalable distribution). Με αυτόν τον τρόπο δεν δεσμεύουν τους σχεδιαστές ενός εξυπερετητή σε κάποιο συγκεκριμένο τρόπο σχεδιασμού κατά την διάρκεια της ανάπτυξης, επιτρέποντας τους να επιλέξουν οι ίδιοι τον σχεδιασμό που ταιριάζει στις δικές τους ανάγκες.
Ένα ακόμα πολύ σημαντικό χαρακτηριστικό το οποίο παρέχει το standard, είναι ότι επιτρέπει σε κάθε πρόγραμμα που έχει αναπτυχτεί για να τρέχει σε έναν τέτοιου είδους εξυπηρετητή, να μπορεί να εγκατασταθεί (deployed) σε οποιοδήποτε άλλον συμβατό εξυπηρετητή και να λειτουργήσει με ακριβώς τον ίδιο τρόπο παρέχοντας τις ίδιες ακριβώς λειτουργίες χωρίς καμιά αλλαγή.
Πέρα από το μοντέλου των συστατικών-οντοτήτων που παρέχεται, δίνεται και ο τρόπος διαχείρισης των διασυνδέσεων που χρησιμοποιούνται για την διαχείριση του περιβάλλοντος στο οποίο τρέχουν τα προγράμματα (container), όπως επίσης και ένα σύνολο από άλλα εργαλεία που βοηθούν κατά την διάρκεια της ανάπτυξης, όπως είναι οι κλάσεις Timer Facility,Trace Facility και άλλες.
Για να μπορέσουμε να καταλάβουμε καλύτερα όλα τα παραπάνω και να είμαστε σε θέση να κατανοήσουμε καλύτερα τον τρόπο με τον οποίο είναι δομημένο ένα πρόγραμμα το οποίο έχει αναπτυχθεί για έναν Jslee application server, θα πρέπει να δούμε όλα τα συστατικά τα οποία προσφέρονται από το standard και τον ρόλο που παίζει το καθένα σε ένα συνολικό πρόγραμμα.

Event driven application.

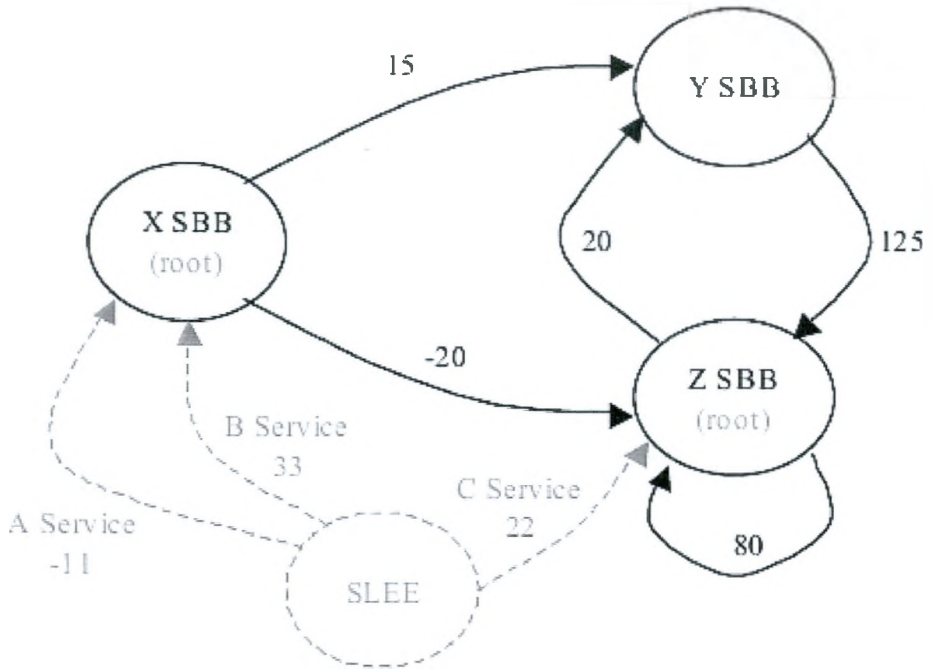
Ως event driven applications χαρακτηρίζουμε αυτά τα προγράμματα τα οποία δεν έχουν κάποιο ενεργό νήμα εκτέλεσης (active thread of execution). Τα προγράμματα αυτά παρέχουν μεθόδους οι οποίες καλούνται όταν συμβάντα φτάνουν στο πρόγραμμα αυτό. Οι μέθοδοι αυτοί περιέχουν κώδικα ειδικά σχεδιασμένο και διαμορφωμένο ώστε να βρίσκει το είδος του συμβάντος και να το επεξεργάζεται καταλλήλως, ώστε να αποφασίσει άμα πρέπει να προχωρήσει σε περεταίρω επεξεργασία. Π.χ. Την περίπτωση ενός εξυπηρετητή που δέχεται μηνύματα τύπου Sip, άμα το πρόγραμμα είναι σχεδιασμένο στο να δέχεται και να επεξεργάζεται μηνύματα τύπου Invite τότε μπορεί να ορίσει ότι θέλει να δέχεται όλα τα συμβάντα τα οποία σχετίζονται με αυτόν τον τύπο μηνυμάτων. Το πιο γνωστό παράδειγμα τέτοιου είδους σχεδιασμού είναι αυτό ενός state machine όπου δέχεται συμβάντα τα επεξεργάζεται και τροποποιεί την κατάσταση του καταλλήλως.

Συμβάντα και είδος συμβάντος (Event and event types).

Ένα event αντιπροσωπεύει ένα συμβάν το οποίο πρέπει να επεξεργαστούμε. Περιέχει πληροφορίες σχετικές με το συμβάν και μπορεί να προέρχεται τόσο από το ίδιο μας το περιβάλλον (Slee) όσο και από κάποιο εξωτερικό πόρο, όπως είναι η στοίβα επικοινωνίας. Όλη η επικοινωνία μεταξύ του Slee και όλων των υπηρεσιών που μπορεί να ενδιαφέρονται γίνονται μέσα από events (συμβάντα), π.χ. Όταν ενδιαφερόμαστε να ειδοποιηθούμε όταν το πεδίο του χρόνου ενός μηνύματος SIP έχει λήξει τότε μπορούμε να θέσουμε ένα Time Facility resource από το Slee και αυτό θα μας ειδοποιήσει μέσω του αντίστοιχου συμβάντος ότι ο χρόνος αναμονής έχει λήξει. Ακόμα συμβάντα μπορούν να στέλνονται και μεταξύ των προγραμμάτων που τρέχουν μέσα στο Slee ώστε να επικοινωνούν.

Συστατικά προγραμμάτων (Application components -Service building blocks).

Όπως αναφέραμε και στην εισαγωγή της παραγράφου αυτής, το standard ορίζει το πώς ένα πρόγραμμα μπορεί να συνθεθεί από διάφορα συστατικά. Αυτά τα συστατικά είναι γνωστά σαν Service Building Blocks ή προς συντομία Sbbs. Κάθε συστατικό τύπου Sbb ορίζει τον τύπο των συμβάντων (events) τα οποία ενδιαφέρεται να επεξεργαστεί, και ορίζει κατάλληλες μεθόδους οι οποίες καλούνται όταν ένα τέτοιο συμβάν συμβεί. Επίσης ένα Sbb μπορεί να παρέχει κάποια διεπαφή με την οποία μπορούμε να χρησιμοποιήσουμε για να το καλούμε με σύγχρονο τρόπο. Αυτά τα συστατικά δημιουργούνται από το Slee κατά την διάρκεια εκτέλεσης όταν απαιτείται η επεξεργασία συμβάντων και διαγράφονται από αυτό όταν δεν χρειάζονται πια. Ο προγραμματιστής των sbb μπορεί να συνθέσει sbbs ορίζοντας μια σχέση γονέα παιδιού ανάμεσα στα διάφορα sbbs. Όπως είναι προφανές με αυτόν τον τρόπο, δημιουργούνται γράφοι όπου κάθε κόμβος μέσα στον γράφο είναι ένα Sbb και κάθε ακμή αποτελεί την σχέση μεταξύ αυτών. Κάθε ακμή μπορεί να περιγραφεί από ένα βάρος το οποίο περιγράφει την προτεραιότητα με την οποία διανέμονται τα συμβάντα ανάμεσα στα παιδιά ενός γονέα. Ακόμα πρέπει να πούμε ότι κάθε γονέας μπορεί να είναι το παιδί άλλων γονέων, όπως επίσης και του ίδιου του του εαυτού. Με τον τρόπο αυτό δίνεται η δυνατότητα τόσο στον προγραμματιστή όσο και στους διαχειριστές δικτύου να ελέγξουν τον τρόπο με τον οποίο διανέμεται η πληροφορία μεταξύ τον sbbs.
Επίσης αυτό έχει αντίκτυπο και στον συνολικό σχεδιασμό ενός συστήματος αφού ο προγραμματιστής μπορεί να δημιουργήσει μια υπηρεσία (βλέπε παρακάτω παράγραφο) η οποία να αποτελείται από πολλά sbb το καθένα από τα οποία υλοποιεί με πολύ αποτελεσματικό τρόπο μία συγκεκριμένη λογική.
Η εικόνα 2.5 παρουσιάζει έναν τέτοιο γράφο, ο οποίος θα μπορούσε να είναι ο γράφος από οποιαδήποτε υπηρεσία.Το SLEE είναι ένας application server που προσφέρει ένα περιβάλλον εκτέλεσης (container) μέσα στο οποίο μπορούν να εκτελεστούν συστατικά λογισμικού (software components). Το περιβάλλον αυτό είναι σχεδιασμένο και βελτιστοποιημένο για προγράμματα που είναι βασισμένα στην αρχιτεκτονική event driven.



Εικόνα 2.5: jain-slee sbb graph

Εξωτερικοί πόροι και είδη πόρων (Resources and Resource adaptors)

Ως Resource (εξωτερικός πόρος) ορίζεται ένα σύστημα το οποίο δεν εντάσσεται στο Slee, δηλαδή που είναι εξωτερικό ως προς αυτό. Για παράδειγμα μία συσκευή δικτύου, μια στοίβα πρωτοκόλλου, καθώς επίσης και μία βάση δεδομένων, μπορούν να χαρακτηριστούν σαν εξωτερικοί πόροι. Για να είναι σε θέση τα προγράμματα τα οποία εκτελούνται μέσα στο περιβάλλον του Slee να δέχονται συμβάντα που παράγοντα από εξωτερικούς πόρους, το JSLEE έχει ορίσει την έννοια του Resource adaptor. Ένας resource adaptor έχει την ευθύνη να προσαρμόζει τους πόρους στις απαιτήσεις που θέτει το Slee. Υπάρχουν πέντε έννοιες για τα resource adaptors οι οποίες είναι:

- Είδος resource adaptor: Ορίζει τα κοινά χαρακτηριστικά για ένα είδος resource adaptor. Ορίζει τις διεπαφές της γλώσσας Java τις οποίες υλοποιεί. Επίσης ορίζει το είδος των συμβάντων τα οποία προκαλούν.
- Resource adaptor: Είναι η υλοποίηση ενός η περισσότερων ειδών resource adaptor, καθώς μπορεί να υπάρχουν πολλές υλοποιήσεις για το ίδιο είδος resource adaptor. Αποτελείται από ένα σύνολο από κλάσεις java και περιγραφών εγκατάστασης (deployment descriptor) . Πρέπει να περιέχει μία τουλάχιστον κλάση java που να υλοποιεί την διεπαφή που ορίζεται από το είδος του resource adaptor.
- Resource adaptor οντότητα (entity): Είναι μια οντότητα τύπου resource adaptor, αυτό γίνεται γιατί πολλαπλές οντότητες μπορούν να αρχικοποιηθούν από ένα και μόνο resource adaptor, με στόχο την διευκόλυνση στον τρόπο που παρέχεται και διακινείται η πληροφορία από και προς το Slee. Ο βασικότερος ρόλος του είναι να προωθεί τα συμβάντα που προέρχονται από έναν πόρο τον οποίο αντιπροσωπεύει αυτό το resource adaptor, προς το περιβάλλον του Slee.
- Resource adaptor αντικείμενο (object): Είναι τα αντικείμενα τα οποία χρησιμοποιεί το Slee ώστε να αλληλεπιδρά με τις οντότητες των resource adaptors.
- Slee endpoint: Είναι η διεπαφή ανάμεσα στο resource adaptor και στο ίδιο το Slee. Υλοποιείτο από το Slee, και είναι το σημείο εισόδου των resource adaptors ώστε να μπορούν να πυροδοτούν

συμβάντα.
Με τον τρόπο αυτό μπορούμε να επιτύχουμε μεγάλο επίπεδο αφαιρετικότητας σε σχέση με το υποκείμενο επίπεδο δικτύου και υποδομών, και να βασιστούμε στον μηχανισμό αυτό ώστε να μας παρέχει όλα όσα είναι απαραίτητα για την καλή λειτουργία της υπηρεσίας μας, ώστε να μας επιτρέψει να επικεντρωθούμε στον σχεδιασμό της υπηρεσίας που θέλουμε να προσφέρουμε.

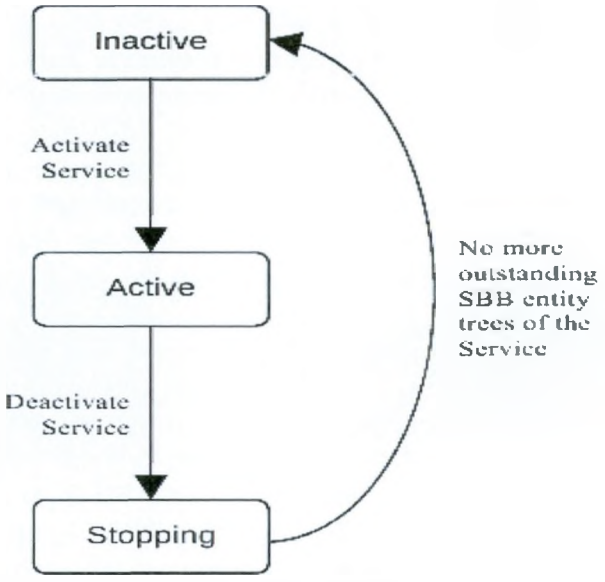
Service

Ένα Service (υπηρεσία) είναι ένα στοιχείο-μονάδα το οποίο εγκαθίσταται σε έναν εξυπηρετητή από τον διαχειριστή ενός συστήματος (System administrator) και παρέχει διεπαφές για τον έλεγχο της υπηρεσίας.
Μία υπηρεσία παρέχει τις πληροφορίες με τις οποίες μπορούμε να την περιγράψουμε.
Αυτές είναι:

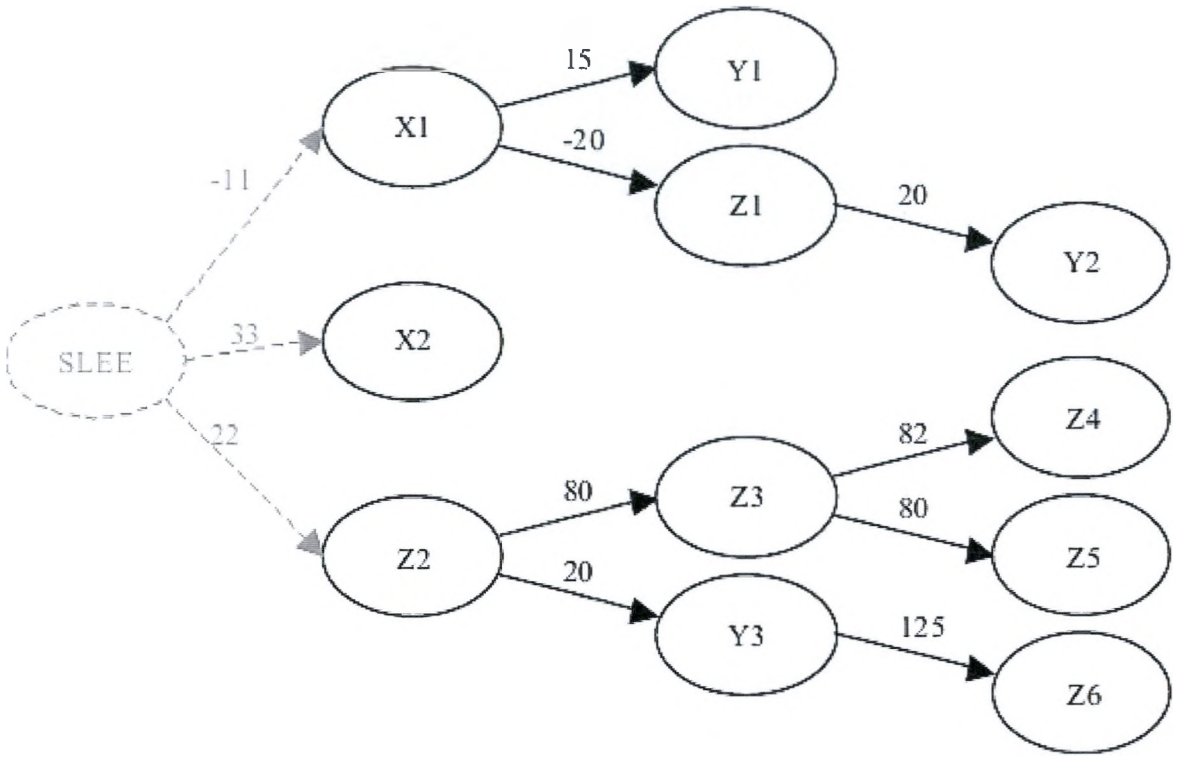
- το όνομα της.
- το όνομα του προμηθευτή της και την έκδοση της.

Βέβαια η βασικότερη λειτουργία για μια υπηρεσία είναι ότι παρέχει πληροφορίες για όλες τις οντότητες οι οποίες ανήκουν και συνθέτουν αυτή την υπηρεσία. Στις πληροφορίες αυτές μπορεί να περιέχονται οι κλάσεις java, τα συστατικά Profile και τα Service building blocks,
καθώς επίσης ορίζει μια λογική ιεραρχία από το περιβάλλον Slee προς το αρχικό Service Building Block. Δηλαδή προσδιορίζει το αρχικό Service Building Block για μία υπηρεσία, που είναι και αυτό στο οποίο εξ ορισμού δρομολογούνται τα events καθώς επίσης προσδιορίζει και την γενικότερη προτεραιότητα με την οποία δρομολογείται ένα event. Επίσης παρέχει στο Slee τις πληροφορίες που χρειάζεται για να αρχικοποιήσει το αρχικό SBB αυτής της υπηρεσίας. Κάθε υπηρεσία μπορεί να είναι σε μία από τις τρείς παρακάτω καταστάσεις:

- Ανενεργή. Η υπηρεσία βρίσκεται σε αυτήν την κατάσταση όταν εγκαθίσταται αρχικά στον εξυπηρετητή.
- Ενεργή. Η υπηρεσία βρίσκεται σε αυτήν την κατάσταση από την ανενεργή κατάσταση.
- Κατάσταση διακοπής λειτουργίας. Η υπηρεσία βρίσκεται σε αυτήν την κατάσταση από την ενεργή κατάσταση.



Εικόνα 2.6: Jain-Slee State Machine

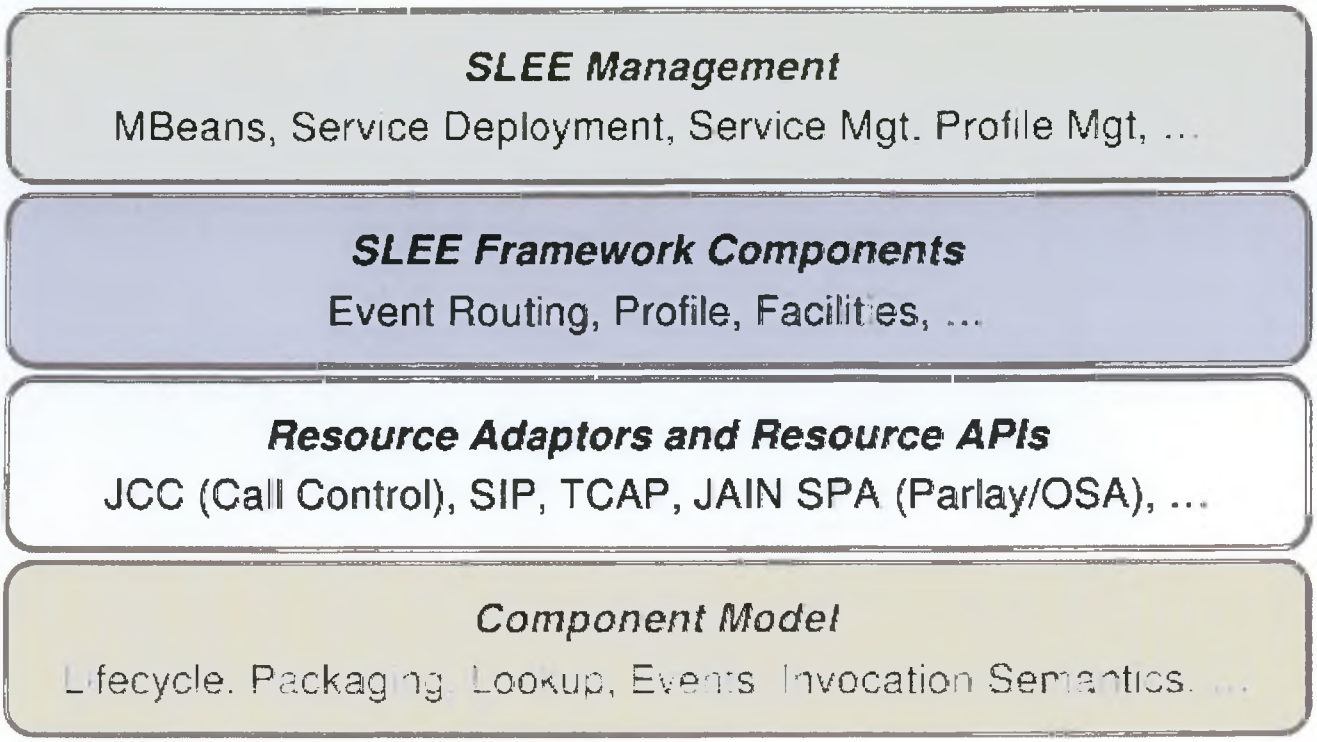


Εικόνα 2.7: Jain-Slee, sbb execution

Στο παραπανω σχημα παρουσιαζεται ο τροπος με τον οποιο μια υπηρεσια μπορει να κληθει απο το Slee, κάθε κύκλος στο σχήμα μας είναι και ένα Sbb με τα βάρη των γραμμών να αναπαριστούν την προτεραιότητα την οποία έχει κάθε sbb κατά την διάρκεια της μετάδοσης ενός συμβάντος. Αυτός ο σχεδιασμός είναι που επιτρέπει στο Slee να επιτυγχάνει χαμηλούς χρόνους αποκρίσεις με υψηλή ρυθμαπόδοση.

Activity Context

είναι μία αληλουχία από events τα οποία ανήκουν στην ίδια κλήση. Μπορούμε να το φανταστούμε σαν την λογική οντότητα μιας κλήσης (ή ενός Activity μιας και το Jslee είναι φτιαγμένο για να είναι ανεξάρτητο δικτύου.). Ακόμα μπορεί να χρησιμοποιηθεί και σαν ένα μέσω ώστε να επικοινωνήσουν μεταξύ τους τα διάφορα Sbb μιας υπηρεσίας, αποθηκεύοντας και ανακτώντας πληροφορίες, μέσα από τον μηχανισμό του ActivityContextInterface.



Εικόνα 2.8: Τομή της αρχιτεκτονικής του Jain-Slee.

2.9 Περιγραφή του Android

Το Android, είναι μία στοίβα λογισμικού (software stack) ανοιχτού κώδικα που απευθύνεται σε συσκευές κινητών τηλεφώνων. Επίσης προσφέρει το λειτουργικό σύστημα, υλοποιημένα προγράμματα, καθώς παίζει και τον ρόλο του Midleware.

Η ανάπτυξη του λειτουργικού, βασίζεται σε μία ομάδα από εταιρίες γνωστή με το όνομα Openhandshet alliance. Παρολ'αυτά η βασικότερη εταιρία που αναπτύσσει και καθορίζει την ανάπτυξη του λειτουργικού είναι η Google, η οποία το εξαγόρασε το 2005 από τους αρχικούς ιδιοκτήτες.

Το Android βασίζεται στον πυρήνα του Linux, πάνω στον οποίο έχουν προσαρμόσει την εικονική μηχανή(virtual machine) Dalvik που έχει αναπτυχθεί από την Google και είναι ειδικά σχεδιασμένη για κινητές συσκευές. Η μεγαλύτερη ιδιαιτερότητα της εικονικής αυτής μηχανής, είναι ότι βασίζεται σε καταχωρητές (Register based) και όχι σε κάποια στοίβα(Stack Based) για αποθήκευση εντολών κατά την εκτέλεση.

Επίσης ένα ακόμα ιδιαίτερο χαρακτηριστικό είναι η συμβατότητα που επιτρέπει να έχουν τα προγράμματα του android με αυτά της γλώσσας Java Standard Edition, έτσι δίνει την δυνατότητα στους προγραμματιστές να προσαρμόσουν με πολύ εύκολο τρόπο τα υπάρχοντα προγράμματα τους πολύ γρήγορα.

Επίσης μπορούμε, αν αυτό κριθεί αναγκαίο, να χρησιμοποιήσουμε και άλλες γλώσσες προγραμματισμού, όπως είναι η C/C++.

Όπως προείπαμε, ο πυρήνας του λειτουργικού συστήματος βασίζεται στο Linux, κάτι το οποίο είναι πολύ σημαντικό, αφού επιτρέπει στο λειτουργικό να εκτελεί πολλές διεργασίες ταυτόχρονα. Έτσι το Android προσφέρει κάποια δομικά συστατικά (components) τα οποία μπορούμε να χρησιμοποιήσουμε ώστε να αναπτύξουμε τα προγράμματα μας.

Έτσι έχουμε τα εξής:

- Activity: Κάθε πρόγραμμα το οποίο θέλει να κάνει χρήση ενός γραφικού ενδιάμεσου, πρέπει να έχει ένα τουλάχιστον Activity.
- Service: Είναι μία κλάση που επιτρέπει να εκτελούμε κώδικα στο παρασκήνιο χωρίς να χρειαστεί να παρουσιάσουμε κάποιο γραφικό ενδιάμεσο.
- BroadcastReceiver: Είναι μία κλάση που επεκτείνουμε ώστε να μπορούμε να δεχθούμε συγκεκριμένα μηνύματα, για τα οποία έχουμε δηλώσει το ενδιαφέρον μας όταν κάποιο άλλο συστατικό τα στείλει.

Εικόνα 2.9: Τομή της αρχιτεκτονικής του Android.

3. Σχεδιασμός συστήματος.

3.1 Specification requirements.

Αρχικά αυτό που θέλαμε να πετύχουμε ήταν να φτιάξουμε ένα σύστημα το οποίο θα επιτρέπει στους χρήστες του να επικοινωνούν πραγματοποιώντας κλήσεις φωνής, καθώς και να μπορούν να ανταλλάσουν μηνύματα, επιτρέποντας την συνεργατικότητα μεταξύ των χρηστών και την διάδοση των μηνυμάτων με σχεδόν άμεσο τρόπο.
Η κύρια ιδέα ήταν να μπορέσουμε να αξιοποιήσουμε όλα τα θετικά στοιχεία που εισήγαγε το Wave, και να τα συνδυάσουμε με αυτά που προσφέρουν οι υπάρχουσες εφαρμογές VoIP, προσφέροντας έτσι ένα σύστημα το οποίο θα συνδυάζει έναν μοναδικό τρόπο επικοινωνίας.
Επιπρόσθετα θέλαμε το σύστημα μας να είναι σε θέση να χρησιμοποιεί τόσο τεχνολογίες σύνθεσης φωνής, όσο και αναγνώρισης, με στόχο την πιο ευχάριστη αλλά και εύκολη χρήση από τους χρήστες.
Πρέπει να τονίσουμε εδώ ότι αποφασίσαμε να χρησιμοποιήσουμε το Android γιατί είναι ένα νέο λειτουργικό σύστημα με μεγάλο μερίδιο αγοράς, αλλά κυρίως γιατί είναι λειτουργικό ανοιχτού κώδικα στο οποίο μπορεί κάποιος να αναπτύξει εφαρμογές χρησιμοποιώντας την γλώσσα προγραμματισμού Java. Αυτό σημαίνει ότι μπορούμε να χρησιμοποιήσουμε γνωστά εργαλεία ανοιχτού κώδικα τα οποία προσφέρουν πολλές επιλογές, διευκολύνοντας έτσι σημαντικά την διαδικασία. Για τους δυο τελευταίους λόγους αποφασίσαμε επίσης να χρησιμοποιήσουμε και τον Mobicents, ο οποίος βασίζεται στον γνωστό Jboss application server και η άδεια χρήσης του είναι LGPL.

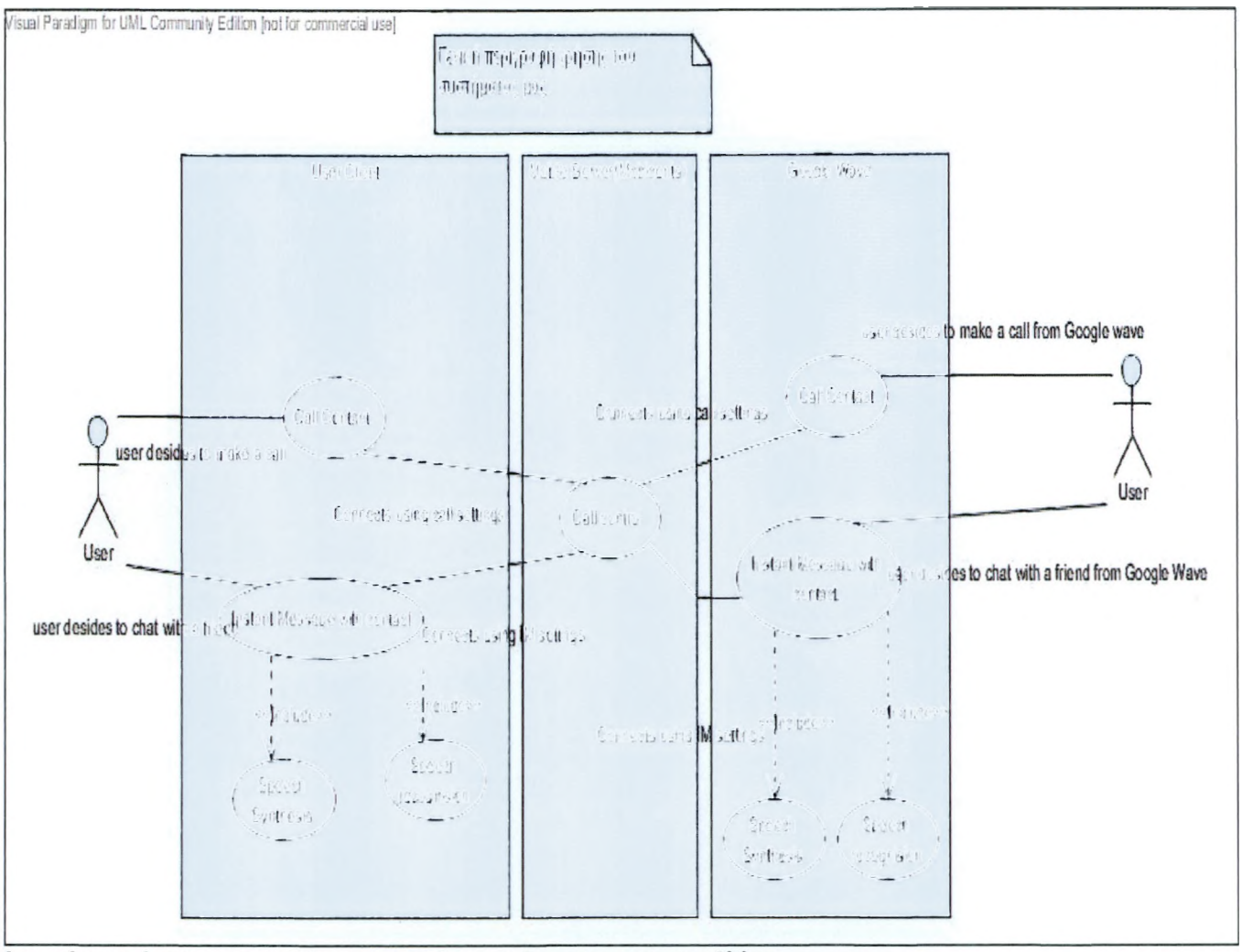
Ο αρχικός σχεδιασμός αποτελείται από τρία βασικά μέρη τα οποία είναι:

- το λογισμικό του χρήστη, το οποίο περιλαμβάνει το λογισμικό που θα έπρεπε να φτιάξουμε για το Android.
- ο εξυπηρετητής, που θα ενώνει το Wave με τις κινητές συσκευές και γι'αυτόν τον λόγο θα πρέπει να είναι σε θέση να καλύπτει τις ανάγκες για κλήσεις VoIP και για την εξυπηρέτηση των άμεσων μηνυμάτων, καθώς να είναι σε θέση να το κάνει με όσο το δυνατόν πιο αποδοτικό τρόπο.
- Τέλος ένα extension του Wave(του πρωτοκόλλου), το οποίο θα είναι και η πύλη πρόσβασης που θα επιτρέπει στον εξυπηρετητή μας να επικοινωνεί με το περιβάλλον του Wave.

Σε αυτό το κεφάλαιο θα αναλύσουμε την κάθε οντότητα από τις παραπάνω ξεχωριστά, αναλύοντας τις αρχικές σκέψεις αλλά και τους λόγους που τελικά αποφασίσαμε να προχωρήσουμε στον τελικό σχεδιασμό, παρουσιάζοντας λογικά διαγράμματα τύπου UML ώστε να γίνει πιο εύκολη η ανάγνωση και η κατανόηση του συστήματος.
Στο πρώτο υποκεφάλαιο θα επικεντρωθούμε στην ανάπτυξη του λογισμικού που σχεδιάσαμε για το Android, έπειτα στον εξυπηρετητή Mobicents και τέλος στο Wave. Σε όλα τα κεφάλαια θα παρουσιαστούν και αντίστοιχα διαγράμματα, της γλώσσας UML, με σκοπό την καλύτερη κατανόηση της διαδοχής τον μηνυμάτων, καθώς και του συνολικού συστήματος. Για τον ίδιο λόγο, μερικά από τα διαγράμματα που παρουσιάζουμε επιλέξαμε να είναι πολύ αναλυτικά και να παρουσιάζουν όλα τα μηνύματα τα οποία στέλνονται, παρότι αυτό θεωρείται λάθος στην αναπαράσταση των διαγραμμάτων.

3.2 Android VoIP client

Μια γενική εικόνα του συστήματος φαίνεται στο διάγραμμα χρήσης UML (Use Case) που ακολουθεί. Όπως αναφέραμε, ο αρχικός στόχος μας ήταν να επιτρέψουμε στους χρήστες να μπορούν να μιλούν τόσο σε πραγματικό χρόνο όσο και μέσω μηνυμάτων κειμένου. Έτσι θα πρέπει το σύστημα μας να υποστηρίζει εξίσου καλά και τις δύο αυτές λειτουργίες. Ακόμα δε θέλαμε να φτιάξουμε ένα λογισμικό που θα επέτρεπε την πρόσβαση στο Wave, αλλά να συνδυάσουμε ένα πρόγραμμα VoIP (Softphone) με το Google Wave, με



τρόπο τέτοιο ώστε να μην είναι εμφανής η χρήση του πρωτοκόλλου.

Για τον λόγο αυτό έπρεπε να επιλέξουμε ένα πρωτόκολλο επικοινωνίας το οποίο χρησιμοποιείται σε τέτοιες εφαρμογές. Τα πρωτόκολλα που χρησιμοποιούνται ευρέως σε προγράμματα softphone είναι το SIP καθώς και το H.323.

Έτσι αποφασίσαμε να διαλέξουμε ένα από τα δύο για λόγους συμβατότητας με άλλες εφαρμογές αλλά και παρόχους VoIP.

Επιπρόσθετα, μιλάμε για ένα πρόγραμμα που θα έχει σαν στόχο κινητές συσκευές, έτσι θα έπρεπε η τελική λύση να είναι τέτοια που να μην επιβαρύνει τον χρήστη με διάφορους περιορισμούς ενέργειας και κινητικότητας, αφού εξ'ορισμού τα κινητά τηλέφωνα είναι φτιαγμένα για να λειτουργούν με συγκεκριμένη ενέργεια και με τρόπο τέτοιο που να επιτρέπει στους χρήστες να κινούνται στον χώρο.

Αυτός ο περιορισμός έχει αντίκτυπο στο πρωτόκολλο επικοινωνίας που πρέπει να χρησιμοποιηθεί, αφού θα

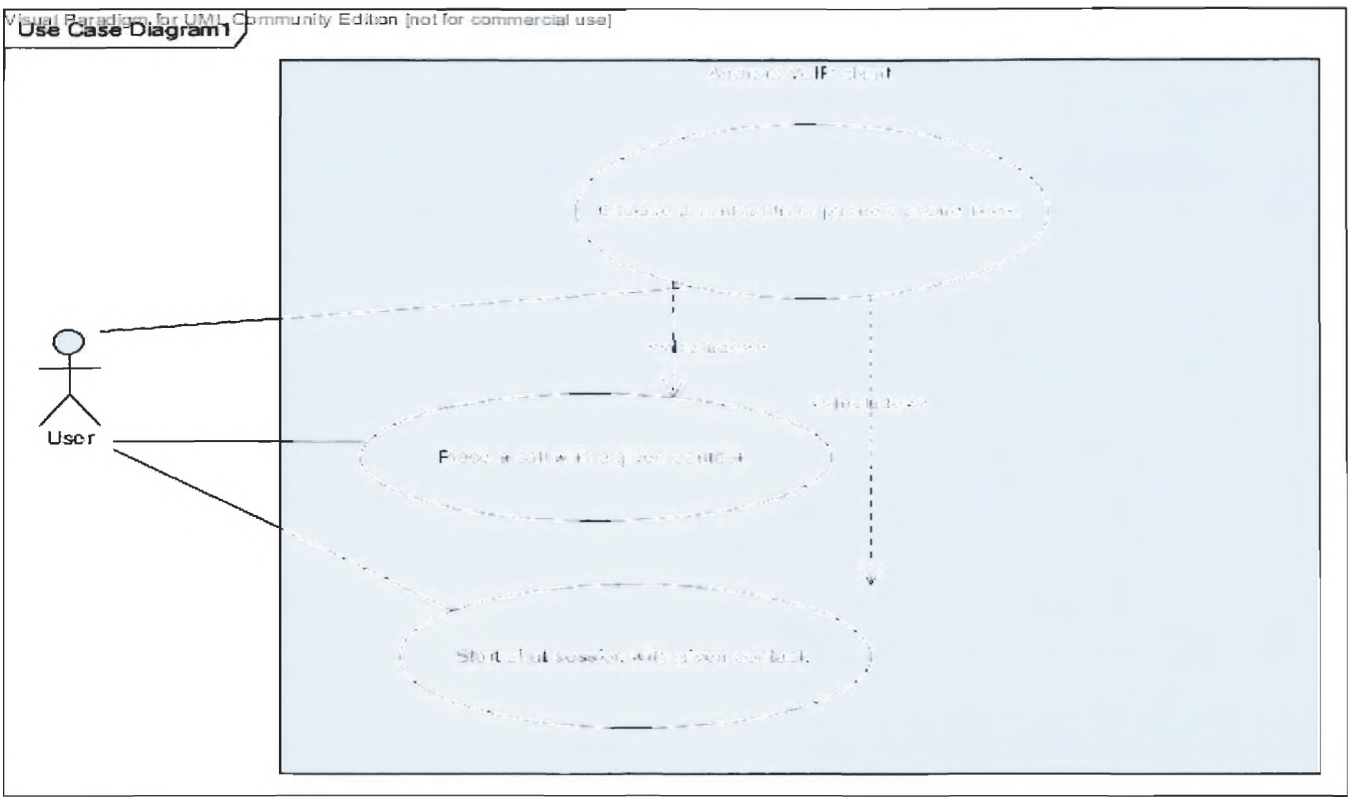
πρέπει να είναι ικανό να εξυπηρετήσει τις ανάγκες που αναφέρθηκαν προηγουμένως με την ελάχιστη δυνατή κατανάλωση ενέργειας και επιτρέποντας πλήρη κινητικότητα στον χώρο.

Το SIP και το H.323 είναι ανταγωνιστικά πρωτόκολλα και το καθένα προσφέρει κάποια θετικά και κάποια αρνητικά στοιχεία.
Το H.323 είναι σχεδιασμένο για να μπορεί να προσφέρει κλήσεις πολυμέσων. Έχει γίνει πολύ δουλειά για το πώς πρέπει να είναι δομημένα όλα τα επίπεδα σε μία εφαρμογή VoIP που βασίζεται σε αυτό, κάτι το οποίο βοηθάει πολύ κατά την διάρκεια του σχεδιασμού, αφού επιτρέπει καλύτερη κατανόηση και διάκριση των διαφορετικών επιπέδων.
Παρόλα αυτά δε έχει καταφέρει να κερδίσει μεγάλο μέρος τις αγοράς, κάτι που επηρέασε σημαντικά την τελική απόφαση για το ποιο πρωτόκολλο θα χρησιμοποιηθεί.
Ακόμα η έλλειψη για μια καλή υλοποίηση στοίβας (stack) για το πρωτόκολλο αυτό στην Java αλλά και η έλλειψη ενός αντίστοιχου resource adaptor από την μεριά του εξυπηρετητή (επόμενη ενότητα), θα μας οδηγήσουν σε πολύ μεγάλη πολυπλοκότητα κατά την υλοποίηση μας.
Ο λόγος είναι ότι όλα τα παραπάνω θα έπρεπε να φτιαχτούν ώστε να ικανοποιούν τους κανόνες και τις οδηγίες που ορίζονται από το πρωτόκολλο ώστε να μπορέσουμε να το χρησιμοποιήσουμε αργότερα.
Σε αντίθεση, για το SIP υπάρχουν δύο πολλοί αξιόπιστες και με πολύ καλή υποστήριξη στοίβες λογισμικού. Το αρνητικό του SIP είναι ότι παρ'ότι αρχικά δείχνει να είναι εύκολο στην υλοποίηση και έχει σχεδιαστεί με βάση αυτή την λογική, τελικά δε είναι. Το μεγαλύτερο πρόβλημα είναι ότι οι περισσότερες εταιρείες προσφέρουν δικές τους υλοποιήσεις για το ίδιο πρωτόκολλο οι οποίες μπορεί να μην είναι εκατό τις εκατό συμβατές η μία με την άλλη, οδηγώντας σε προβλήματα κατά την διάρκεια των κλήσεων. Αυτό οφείλεται κυρίως στο γεγονός ότι δε υπάρχουν αυστηροί κανόνες τους οποίους πρέπει κάποιος να ακολουθήσει, κάτι που οδηγεί τελικά στο να αυξηθεί ακόμα περισσότερο η πολυπλοκότητα μιας υλοποίησης.
Τέλος, το μεγαλύτερο πρόβλημα που αντιμετωπίσαμε ήταν ότι δε υπήρχε επίσημη υποστήριξη για καμία από τις παραπάνω στοίβες επικοινωνίας από το ίδιο το λειτουργικό σύστημα.
Λόγω της ιδιαιτερότητας του Android που αναφέραμε στο δεύτερο κεφάλαιο, ένα μεγάλος μέρος από τις ήδη υπάρχουσες εφαρμογές που είναι γραμμένες για την γλώσσα Java Standard Edition, υποστηρίζεται από το Android με λίγες ή και καθόλου αλλαγές. Έτσι οι ήδη υπάρχουσες στοίβες που είναι γραμμένες για την Java υποστηρίζονται και από το Android, αυτός ήταν ο λόγος που τελικά επιλέξαμε το SIP.
Οι πιο γνωστές στοίβες που υπάρχουν είναι το MJSIP και το JAIN-SIP που μπορούν να χρησιμοποιηθούν και για ανάπτυξη στο Android. Επιπλέον η πληθώρα των RFC που υπάρχει για το SIP επιτρέπει την χρήση του τόσο για σηματοδοσία κλήσεων όσο και για μεταφορά άμεσων μηνυμάτων.
Η διαδικασίες που απαιτούνται για την ανάπτυξη εφαρμογών, περιγράφονται μέσα από διαφορετικά RFC τα οποία όμως βοηθούν πολύ αναλύοντας πολλά από τα προβλήματα τα οποία μπορούν να παρουσιαστούν.
Τέλος αξίζει να αναφέρουμε ένα ακόμα πρωτόκολλο που θα μπορούσαμε να χρησιμοποιήσουμε, αυτό είναι το XMPP. Το XMPP υποστηρίζεται από το λειτουργικό σύστημα και είναι σε θέση να χρησιμοποιηθεί για αποστολή άμεσων μηνυμάτων και μεταφορά πολυμέσων, μέσω του Jingle.
Οι λόγοι που δε χρησιμοποιήθηκε είναι δύο:

- Ο πρώτος λόγος είναι ότι έχει μικρή ανταπόκριση ανάμεσα στους παρόχους VoIP.
- Και ο δεύτερος είναι ότι, δε θα μπορούσαμε να αναπτύξουμε το κατάλληλο πρόγραμμα στον server (Mobicents) μας ώστε να προσφέρουμε τις υπηρεσίες που θέλουμε (αναλύεται στο κεφάλαιο 4).

Όλες οι προαναφερθείσες λύσεις είναι αποδεκτές από άποψη κατανάλωσης ενέργειας αφού το μόνο που χρειάζεται είναι να μπορούμε να διατηρήσουμε μία σύνδεση με τον Server μας ώστε να πραγματοποιούμε και να δεχόμαστε κλήσεις και μηνύματα. Αυτό, παρ'ότι δε είναι η βέλτιστη λύση από μεριάς διαχείρισης ενέργειας, είναι αποδεκτή αφού η επιβάρυνση δε είναι μεγάλη, επιτρέποντας στον χρήστη να μετακινείται

χωρίς πρόβλημα, αφού κάθε φορά που είναι αναγκαίο να αλλάξει διεύθυνση IP τότε απλά ενημερώνει τον αντίστοιχο Registar εξυπηρετητή.
Στην συνέχεια θα παρουσιάσουμε σε UML Use Case διάγραμμα ένα γενικό σχήμα για το πως και τι ενέργειες θα είναι το πρόγραμμα ικανό να πραγματοποιήσει.

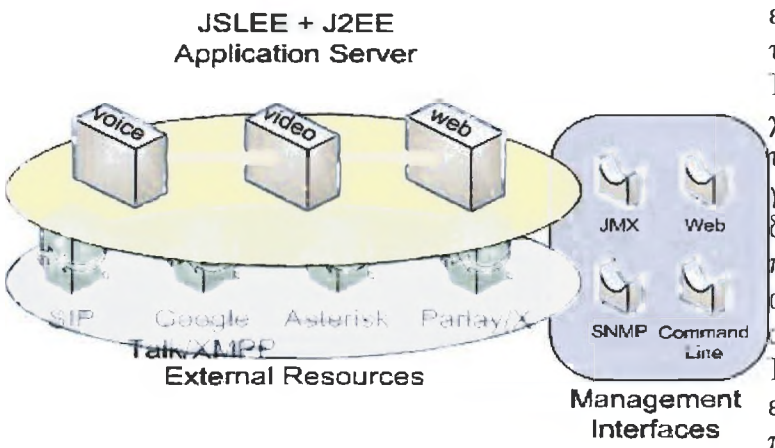


Κάθε χρήστης θα πρέπει να επιλέξει μία από τις διαθέσιμες επαφές του από τον τηλεφωνικό κατάλογο του κινητού τηλεφώνου και να διαλέξει ανάλογα τον τρόπο επικοινωνίας που θέλει(κλήση ή μηνύματα), αξιοποιώντας και τεχνολογίες σύνθεσης και αναγνώρισης φωνής για να το κάνει αυτό.
Η ιδέα του να επιλέγει ο χρήστης έναν συνομιλητή από τον τηλεφωνικό κατάλογο, είναι επέκταση του τρόπου με τον οποίο το λειτουργικό διαχειρίζεται τις επαφές, επιτρέποντας στον χρήστη να βάλει πολλαπλά πεδία, και ανανεώνοντας ΄τες αυτόματα. Έτσι όταν ένας χρήστης συνδέεται με το Wave και προσθέτει φίλους, αυτοί συγχρονίζονται αυτόματα από το λειτουργικό με τον τηλεφωνικό κατάλογο του κινητού τηλεφώνου.
Στην περίπτωση που ο χρήστης μας επιθυμεί να χρησιμοποιήσει την υπηρεσία άμεσου μηνύματος, τότε πρέπει το σύστημα να τον διευκολύνει παρέχοντας του την δυνατότητα κάθε φορά που δέχεται ένα νέο μήνυμα να του το διαβάζει, καθώς και να επιτρέπει την σύνθεση φωνής, ώστε να είναι σε θέση ο χρήστης απλά να λέει αυτό που θέλει και το πρόγραμμα να το γράφει και να το στέλνει αυτόματα.
Για την μεταφορά των μηνυμάτων χρησιμοποιείται το SIMPLE, και για τις υπηρεσίες ανάγνωσης και σύνθεσης τις ενσωματωμένες υπηρεσίες του Android. Γι'αυτό και η συσκευή η οποία μπορεί να χρησιμοποιήσει το πρόγραμμα καλό θα ήταν να τρέχει έκδοση μεγαλύτερη από την 1.6, και να έχει εγκατεστημένο το Android Market.
Μέσα στα πλαίσια της συνεργατικότητας, εντάσσεται επίσης και η δυνατότητα να μπορούν δύο ή και

παραπάνω άτομα να επεξεργάζονται το ίδιο μήνυμα. Έτσι θα πρέπει να κάθε φορά που γίνεται αυτό να ξέρουμε ποιο ήταν το μήνυμα αυτό το οποίο τροποποιήθηκε, και να το μετατρέπουμε αναλόγως.

3.3 Mobicents Telecommunication Server.

Στο προηγούμενο κεφάλαιο παρουσιάσαμε περιληπτικά τον Mobicents. Αυτό που δε αναλύσαμε ήταν οι λόγοι που μας οδήγησαν στην επιλογή του συγκεκριμένου server καθώς και της πλατφόρμας Jain-Slee. Όπως αναφέραμε και στην εισαγωγική ενότητα του κεφαλαίου αυτού, έχουμε μεριμνήσει ώστε όλες οι τεχνολογίες που χρησιμοποιούμε να είναι ανοιχτού κώδικα Αυτό βοήθησε πολύ στην τελική επιλογή μας αφού ο Mobicents είναι ο μόνος πιστοποιημένος server που υποστηρίζει το Jain-Slee.
Η σημασία του να είναι ανοιχτού κώδικα και να βασίζεται σε ανοιχτά standards, γίνεται αντιληπτή μόλις δούμε την υπάρχουσα κατάσταση στον χώρο των τηλεπικοινωνιών, όπου υπάρχουν πολλές καινούργιες τεχνολογίες οι οποίες διαρκώς εξελίσσονται, ενσωματώνοντας καινούργιες υπηρεσίες ή τροποποιώντας παλαιότερες.



Εικόνα 3.1: δομή του Jain-slee

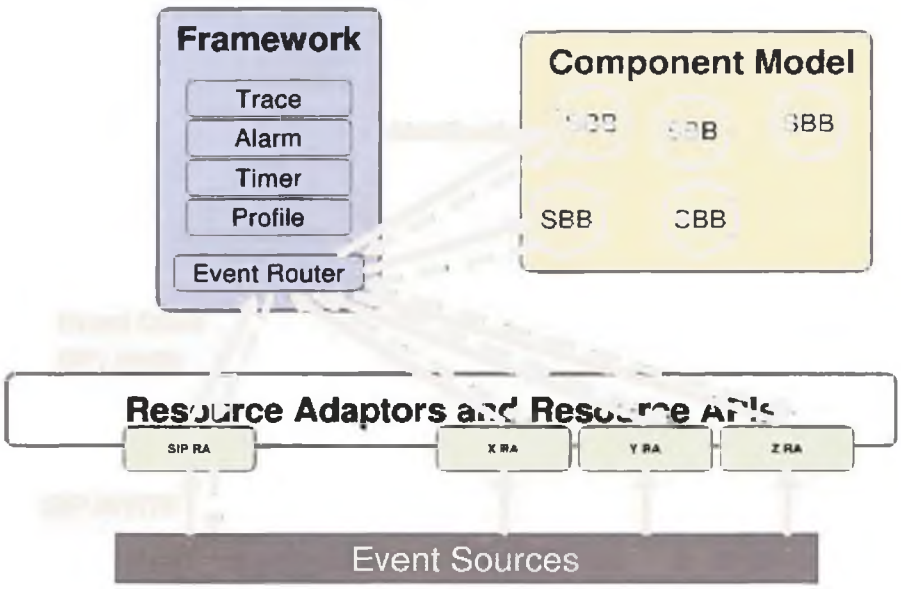
Έτσι είναι πολύ δύσκολο για τις εταιρίες του χώρου αυτού, να εξελίσσουν ή να τροποποιούν τις υποδομές τους ώστε να καλύψουν την ζήτηση για αυτές τις υπηρεσίες. Την κατάσταση αυτή δυσχεραίνει πολύ το γεγονός ότι τα συστήματα που προσφέρουν τις υπηρεσίες χαρακτηρίζονται από κάθετη ολοκλήρωση(vertical integrated) και ανομοιογένεια.
Έτσι δεν υπάρχει εύκολος τρόπος για να επεκταθούν ή να συνδυαστούν οι υπηρεσίες που προσφέρουν, αφού δεν υπάρχει κοινή διεπαφή που να ορίζεται από ανοιχτά standards.
Το Jslee έχει σχεδιαστεί από την αρχή ώστε να είναι επεκτάσιμο, και να μπορεί να συνδυαστεί με ένα πλήθος από άλλες υπηρεσίες. Επίσης μπορεί να συνδυαστεί με το περιβάλλον J2EE και να χρησιμοποιήσει όλες τις παρεχόμενες δυνατότητες του περιβάλλοντος τις Java όπως είναι το RMI το JNDI, το JMX και άλλα όπως φαίνεται στην παραπάνω εικόνα.
Αυτό σημαίνει ότι μπορούμε να συνδυάσουμε όλα τα καλά που προσφέρει το περιβάλλον Java Enterprise Edition για ανάπτυξη εφαρμογών, με γνωστά πρωτόκολλα και τεχνολογίες που χρησιμοποιούνται για ανάπτυξη τηλεπικοινωνιακών εφαρμογών.

Το γεγονός αυτό καθιστά τον συγκεκριμένο server ιδανικό για ανάπτυξη εφαρμογών κατά τα πρότυπα των δικτύων επόμενης γενιάς (NGN), και κατ'επέκταση μια πολύ καλή επιλογή και για τις ανάγκες της δικής μας εφαρμογής, αφού μπορεί να μας προσφέρει τον τρόπο να χρησιμοποιούμε δύο διαφορετικά πρωτόκολλα με ένα πρόγραμμα.

Εικόνα3.2: προσφερόμενες τεχνολογίες πάνω στο περιβάλλον του JBoss

Στο επίπεδο της ανάπτυξης, οι επιλογές μας περιορίζονταν ανάμεσα στο περιβαλλόντων Jain-Slee και το Sip-Servlets.
Τα Sip-Servlets είναι ιδανικά για εφαρμογές που συνδυάζουν και επεκτείνουν τις παραδοσιακές διαδικτυακές εφαρμογές με τις δυνατότητες που προσφέρει το SIP. Βέβαια γίνεται άμεσα κατανοητό ότι κάτι τέτοιο είναι πολύ περιοριστικό στην δική

μας περίπτωση αφού στόχος δεν είναι να μπορέσουμε να στέλνουμε μηνύματα από και προς το Wave αλλά να αξιοποιήσουμε το πρωτόκολλο Wave federation ώστε να το ενώσουμε με τις VoIP εγαρμογές.
Για τον λόγο αυτό τελικά αποφασίσαμε η υλοποίηση να γίνει με χρήση του Jain-Slee και αυτό γιατί η συγκεκριμένη τεχνολογία είναι επεκτάσιμη μέσω του μηχανισμού των Resource Adaptors(Βλέπε κεφάλαιο 2.4).

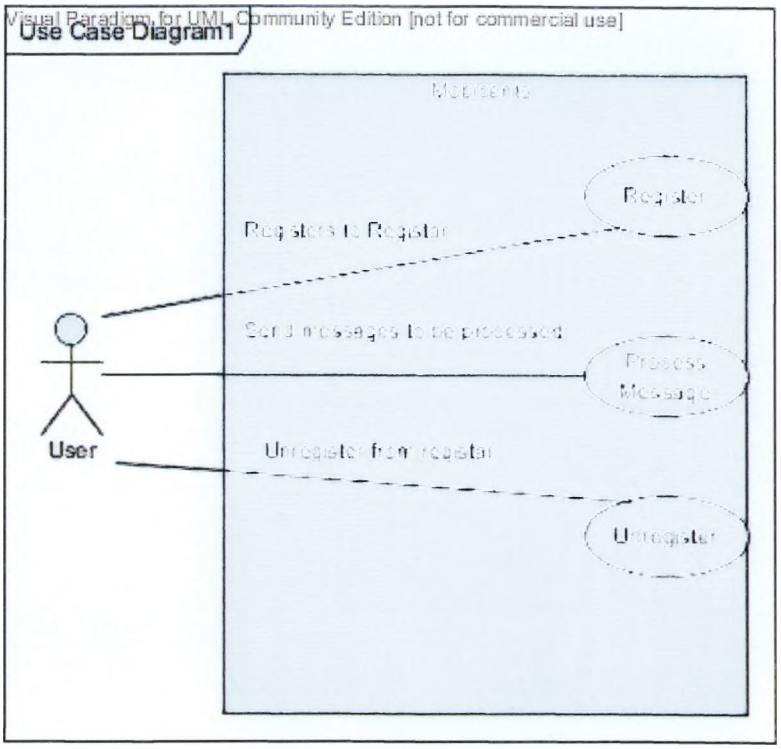


Εικόνα 3.2: Ο τρόπος με τον οποίο δρομολογούνται τα events από τα RA, στα Sbbs

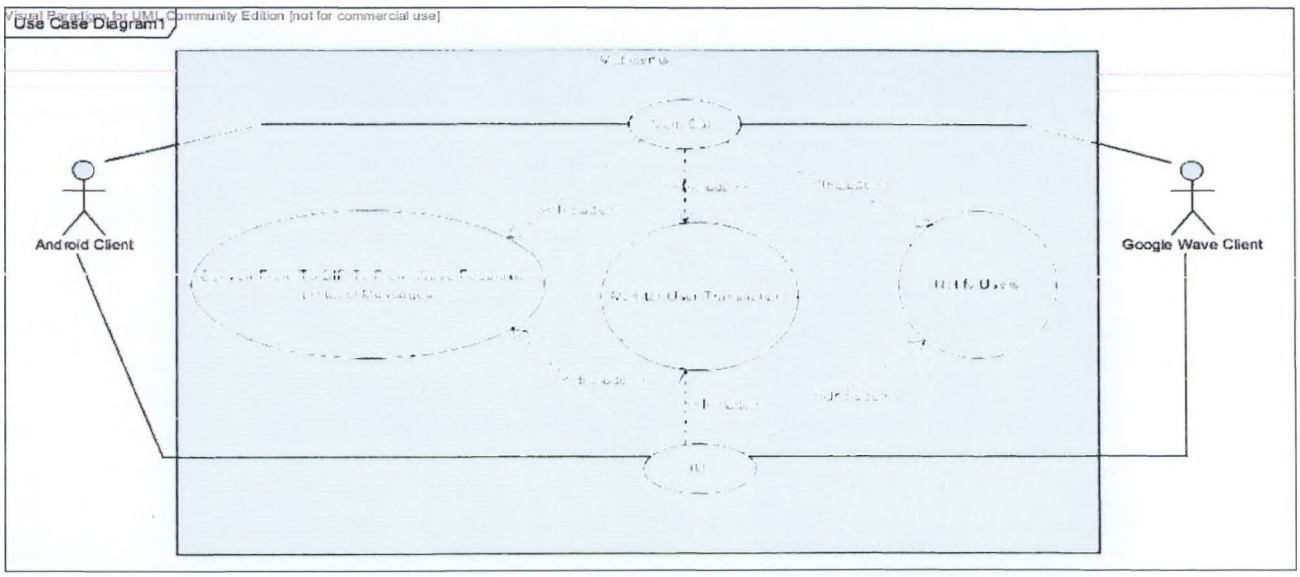
Τα αρνητικά της λύσης αυτής είναι ότι το Jain-Slee είναι πολύ πιο δύσκολο στην εκμάθηση από ότι τα Sip-Servlets. Λόγω τον επιλογών που είναι διαθέσιμες από το Jain-Slee, όπως είναι τα Sbb's και τα RA's είναι πιο εύκολο για κάποιον άπειρο να κάνει λάθος στον τρόπο που εγκαθιστά τον Server.

Μια χαρακτηριστική εικόνα με τις επιλογές που προσφέρονται από το περιβάλλον του Jain-Slee παρουσιάζεται στην εικόνα 3.1.

Όπως προείπαμε, ο Mobicents μπορεί να εγκατασταθεί τόσο πάνω από τον Tomcat όσο και από τον Jboss, καθένας από τους οποίους έχει ξεχωριστές δυνατότητες και υποστηρίζει διαφορετικές τεχνολογίες υλοποίησης. Στην περίπτωση μας επιλέξαμε να χρησιμοποιήσουμε το περιβάλλον του Jboss, αυτό γιατί μόνο με αυτόν μας δίνεται η δυνατότητα να χρησιμοποιήσουμε το Jain-Slee, κάτι το οποίο είναι αναγκαίο για να μπορέσουμε να γεφυρώσουμε το λογισμικό του Android με το Google Wave. Ο Mobicents θα χρησιμοποιηθεί σαν ένας ενδιάμεσος μεταξύ τον δύο περιβαλλόντων. Από αυτό μπορούμε να καταλάβουμε επίσης ότι ο Server μας πρέπει να μπορεί να παρακολουθεί αλλά και να ενημερώνει όλους τους χρήστες για την διαθεσιμότητα και τις προθέσεις των συνομιλητών τους. Δηλαδή μπορούμε να πούμε ότι πρέπει να λειτουργεί και σαν Sip Registar Server (κεφάλαιο 2), στο παρακάτω σχήμα παρουσιάζουμε τις δυνατότητες του χρήστη.



Έτσι θα πρέπει ο server να είναι σε θέση να καταλαβαίνει αλλά και να συνδυάζει τα δύο πρωτόκολλα. Αυτό φαίνεται και στο διάγραμμα που ακολουθεί.



Κάθε φορά που ένας χρήστης θα επιλέγει να έρθει σε επαφή με κάποιον άλλον, τότε ο Server μας θα πρέπει πρώτα να τον ψάξει και αν τον βρει, να τον ενημερώσει ότι κάποιος προσπαθεί να επικοινωνήσει μαζί του. Ειδάλλως να ενημερώνει τον καλούντα ότι δεν μπόρεσε να τον βρει, και με αυτόν τον τρόπο να τερματίζεται η συναλλαγή μεταξύ τον δύο.
Ανεξάρτητα από τον τρόπο που επιθυμούν οι χρήστες να επικοινωνήσουν ,οι βασικές διαδικασίες που θα ακολουθούνται θα είναι ίδιες πάντα, από την μεριά του Mobicents.
Δηλαδή θα πρέπει να μπορούμε να εναλλάσσουμε τα μηνύματα ανάλογα με το πρωτόκολλο ,όπου αυτό

γίνεται με την βοήθεια των SBBs (περισσότερα στο επόμενο κεφάλαιο).
Να μπορούμε να αναγνωρίζουμε την συναλλαγή ώστε να δρομολογούνται τα μηνύματα με κατάλληλο τρόπο στον σωστό παραλήπτη και να ενημερώνουμε τους χρήστες για το αν κάποιος είναι διαθέσιμος ή όχι, κάτι που γίνεται μέσω του μηχανισμού του Registar όπως αναφέραμε και προηγουμένως.
Όπως είδη γνωρίζουμε από το κεφάλαιο δύο, το Sip προσφέρει δύο διαφορετικές οντότητες για τους εξυπηρετητές. Αυτές είναι η Back to Back User Agent(B2BUA) και η Proxy. Έτσι έπρεπε να γίνει επιλογή σύμφωνα με τις απαιτήσεις του συστήματος για τον τρόπο με τον οποίο θα πρέπει να εγκαταστήσουμε τον εξυπηρετητή μας. Αν και μία μικτή αρχιτεκτονική δεν αποτρέπεται από τις προδιαγραφές του πρωτοκόλλου(specification), κάτι τέτοιο δεν κρίθηκε αναγκαίο.
Για να πάρουμε την κατάλληλη απόφαση σύμφωνα με τον τρόπο που θα χρησιμοποιούσαμε τον εξυπηρετητή μας, έπρεπε να έχουμε κατά νου τον τρόπο λειτουργίας του Jingle του federation protocol και του Sip. Έτσι θέλαμε να αναπτύξουμε ένα σύστημα το οποίο θα λειτουργεί ταυτόχρονα σαν Proxy και σαν Gateway ανάμεσα στο Jingle και το Sip. Για τον σχεδιασμό του Proxy και περισσότερα για τον τρόπο λειτουργίας του θα δούμε σε επόμενο κεφάλαιο. Σε αυτό το κεφάλαιο θα αναπτύξουμε περισσότερο την ιδέα πίσω από το Gateway και πώς μπορούμε να ενώσουμε το Jingle με το sip ώστε να πετύχουμε ένα σύστημα που χρησιμοποιεί και τα δύο.
Το Jingle, αναπτύχθηκε μεταγενέστερα του Sip, και σχεδιάστηκε με τρόπο τέτοιο, που επιτρέπει την αντιστοίχιση των μηνυμάτων του στα αντίστοιχα του Sip.
Οι βασικότερες διαφορές των δύο είναι

1. Τα μηνύματα του Sip, αλλά και το περιεχόμενο των πεδίων της επικεφαλίδας, χρησιμοποιούν δομή απλού κειμένου παρόμοια με αυτή που συναντάμε στο HTTP.
2. Το Sip κάνει χρήση του SDP, για να συμφωνήσει τις παραμέτρους μιας κλήσης. Σε αντίθεση το Jingle ορίζει τα στοιχεία αυτά σαν πεδία μέσα στο <content> πεδίο του xml μηνύματος. Η αντιστοίχιση βέβαια μεταξύ των δύο είναι εφικτή.
3. Το κανάλι που χρησιμοποιεί το Sip για να μεταδώσει την σηματοδοσία είναι το UDP. Αντίθετα το Jingle χρησιμοποιεί το TCP και το UDP.
4.

Παραθέτουμε έναν πίνακα με τις διαθέσιμες αντιστοιχίες των δύο.

Jingle	***Sip***
<jingle/> 'action'	Πολλαπλές αντιστοιχίες, βλέπε επόμενο πίνακα
<jingle/> 'initiator'	Χωρίς αντιστοιχία
<jingle/> 'responder'	Χωρίς αντιστοιχία
<jingle/> 'sid'	Local-part από το Call-ID
local-part of 'initiator	Το κομμάτι του <username> που βρίσκεται στο o= του SDP
<content/> 'creator'	Χωρίς αντιστοιχία
<content/> 'name'	Χωρίς αντιστοιχία
<content/> 'profile'	Το κομμάτι του <proto> που βρίσκεται στο m= του SDP
<content/> 'senders' η τιμή τόσο για τον αποστολέα όσο και τον παραλήπτη.	Το a= του sendrecv,recvonly, ή του sendonly

Ο παραπάνω πίνακας δεν ορίζει αντιστοιχία για το 'action', αυτό γιατί μπορεί να έχει πολλαπλές αντιστοιχίες. Έτσι έχουμε:

Jingle Action	Sip Method
content-accept	transport-info (1XX)
content-add	INVITE request
content-modify	INVITE request
content-remove	INVITE request
session-accept	INVITE response (1XX ή 2XX)
session-info	Εξαρτάται από την κλήση
session-initiate	INVITE request
session-terminate	BYE
transport-info	Εξαρτάται απο την κλήση

Τα παραπάνω ισχύουν μόνο για το κομμάτι που αφορά το Sip και το Jingle. Όμως το Sip χρησιμοποιεί και το SDP για τις κλήσεις, στην περίπτωση αυτή έχουμε:

1. Και στα δύο, το είδος τον δεδομένων που θα μεταφέρουμε καθορίζεται από την λέξη κλειδί 'audio'
2. Το <bandwidth/> στοιχείο πρέπει να αντιστοιχιστεί στο αντίστοιχο 'b=' του SDP.
3. Αν το payload-type είναι στατικό, τότε πρέπει να αντιστοιχιστεί στο 'm=' (RFC 4566). Έτσι αν για παράδειγμα έχουμε το μήνυμα:

```
<description xmlns='urn:xmpp:jingle:apps:rtp:1' media='audio'>
 <payload-type id="13" name="CN"/>
</description>
```

τότε αυτό θα μετατραπεί σε :

m=audio 9999 RTP/AVP 13

Ενώ άμα το payload-type, είναι δυναμικό τότε θα έχουμε:

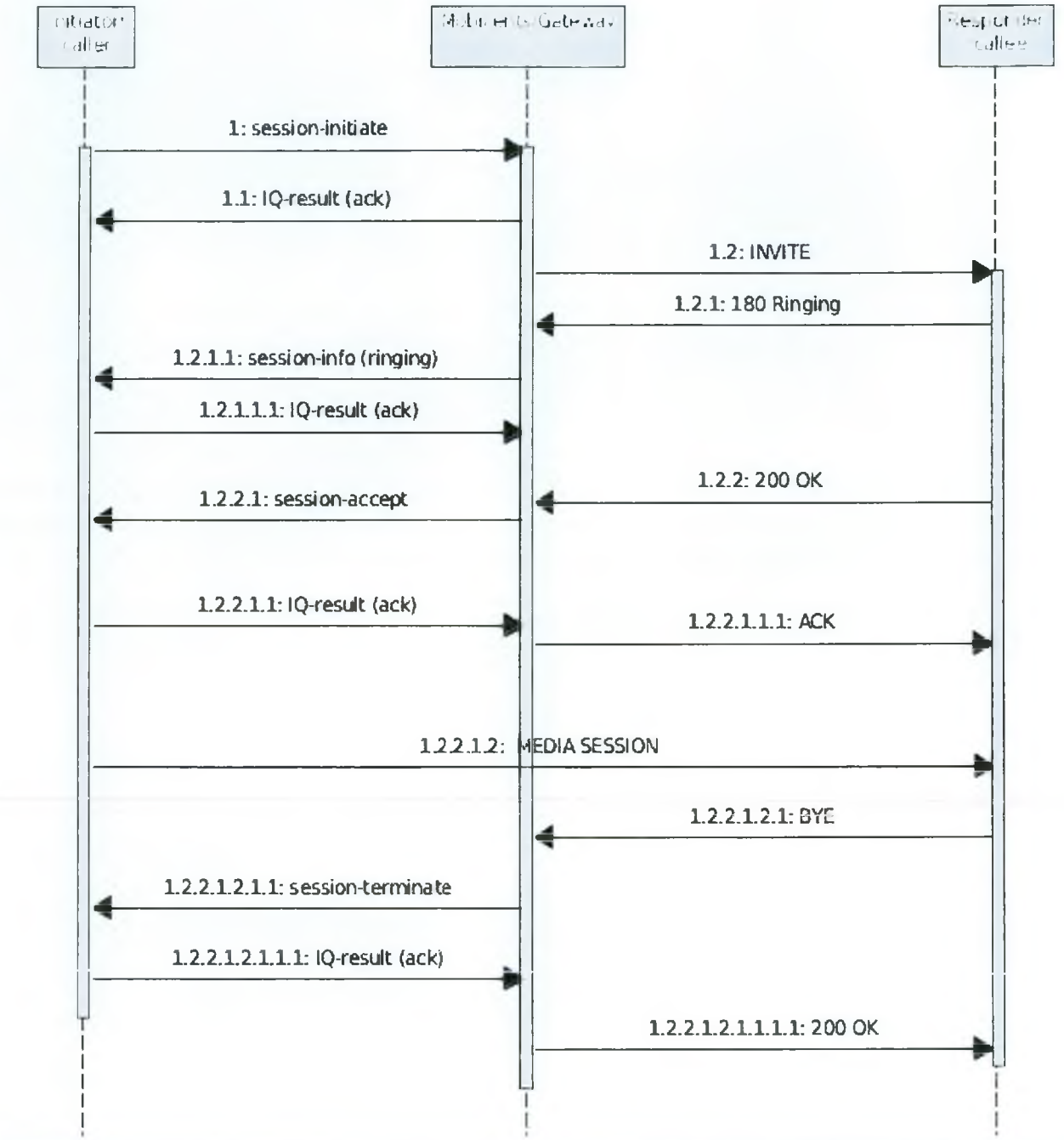
```
<description xmlns='urn:xmpp:jingle:apps:rtp:1' media='audio'>
 <payload-type id='96' name='speex' clockrate='16000'/>
</description>
```

Με το αντίστοιχο SDP πεδίο να είναι :

```
m=audio 9999 RTP/AVP 96
a=rtpmap:96 speex/16000
```

Το θετικό τις προσέγγισης αυτής είναι ότι επιτρέπει να βάλουμε επιπρόσθετες παραμέτρους, αλλάζοντας απλά και μόνο τον αριθμό τον <parameter> στοιχείων μέσα στο μήνυμα του Jingle. Έτσι μπορούμε να έχουμε μια πλήρη αντιστοίχιση των παραμέτρων του ενός με το άλλο.

Τέλος θα πρέπει να δείξουμε πώς τα διαγράμματα κλήσεων του κάθε πρωτοκόλλου που παρουσιάστηκαν στο κεφάλαιο δύο, μπορούν να ενωθούν ώστε να πετύχουμε την υπηρεσία που θέλουμε.



Εικόνα 3.3: Jingle to Sip call flow.

Οι μετατροπές των μηνυμάτων είναι οι εξής:

Περιγραφή	Jingle	SIP
Αρχικά στέλνεται το μήνυμα για να αρχίσει η κλήση από τον καλούντα προς τον εξυπηρετητή.	**<iq from='juliet@example.com/t3hr0zny' id='hu2s61f4' t/v3rsch1kk3l1jk' type='set'> <jingle xmlns='urn:xmpp:jingle:1'**	

	action='session-initiate' initiator='juliet@example.com/ t3hr0zny' sid='a73sjjvkla37jfea'> <content creator='initiator' media='audio' name='this-is-the-audio-content'> <description xmlns='urn:xmpp:jingle:app:rtp:1'> <payload-type id='96' name='speex' clockrate='16000'/> <payload-type id='97' name='speex' clockrate='8000'/> <payload-type id='18' name='G729'/> </description> <transport xmlns='urn:xmpp:jingle:transport:raw-udp'> <candidate ip='192.0.2.101' port='49172' generation='0'/> </transport> </content> </jingle> </iq>	
Ο εξυπηρετητής απαντάει εκ μέρους του παραλήπτη με ένα IQ-result.	**<iq from='juliet@example.com/t3hr0zny' id='hu2s61f4' to='romeo@example.net/v3rsch1kk3l1jk' type='result'/>**	
Σε αυτό το σημείο ο εξυπηρετητής κάνει την μετατροπή του μηνύματος σε SIP και στέλνει το αίτημα.		**INVITE sip:romeo@example.net SIP/2.0 Via: SIP/2.0/TCP client.example.com:5060;branch=z9hG4bK74bf9 Max-Forwards: 70 From: Juliet Capulet <sip:juliet@example.com>;tag=t3h To: Romeo Montague <sip:romeo@example.net> Call-ID: 3848276298220188511@example.com CSeq: 1 INVITE Contact: <sip:juliet@client.example.com;transport=tcp> Content-Type: application/sdp Content-Length: 184 v=0 o=alice 2890844526 2890844526 IN IP4 client.example.co s=- c=IN IP4 192.0.2.101 t=0 0 m=audio 49172 RTP/AVP 0 a=rtpmap:96 SPEEX/16000 a=rtpmap:97 SPEEX/8000 a=rtpmap:18 G729**
Ο παραλήπτης απαντάει με μήνυμα 180 Ringing, δείχνοντας την επιθυμία του να επικοινωνήσει.		**SIP/2.0 180 Ringing Via: SIP/2.0/TCP client.example.com:5060;branch=z9hG4bK74bf9 ;received=192.0.2.101 From: Juliet Capulet <sip:juliet@example.com>;tag=t3h To: Romeo Montague <sip:romeo@example.net>;tag=v3rsch1kk3l1jk Call-ID: 3848276298220188511@example.com**

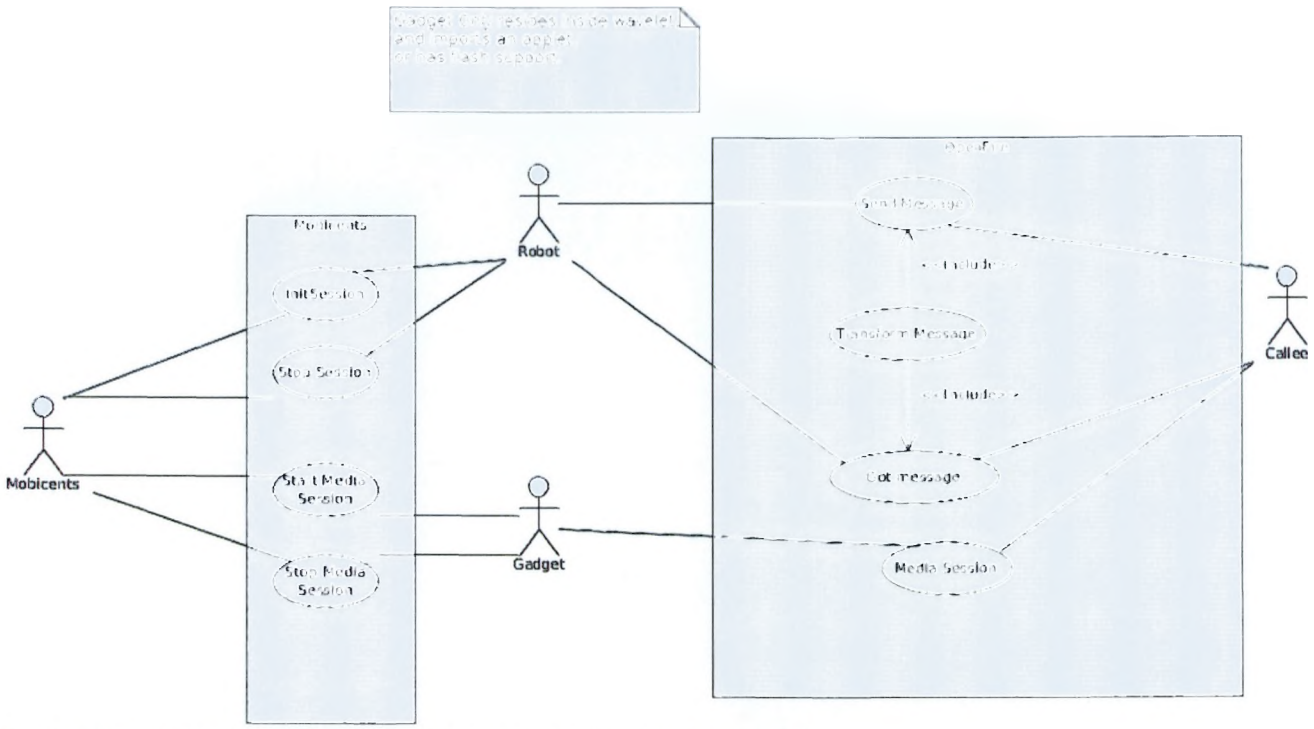
		CSeq: 1 INVITE **Contact: <sip:romeo@client.example.net;transport=tc** **Content-Length: 0**
Ο εξυπηρετητής κάνει τις απαραίτητες αντιστοιχίσεις μεταξύ Jingle και SIP και προωθεί το μήνυμα.	**<iq from='romeo@montague.net/v3rsch1kk3l1jk'** **id='ol3ba71g'** **to='juliet@example.com/t3hr0zny'** **type='set'>** **<jingle xmlns='urn:xmpp:jingle:1'** **action='session-info'** **initiator='juliet@example.com/t3hr0zny'** **sid='a73sjjvkla37jfea'>** **<ringing xmlns='urn:xmpp:jingle:app:rtp:1-info'/>** **</jingle>** **</iq>**	
Το μήνυμα 180 που μόλις πήρε επιβεβαιώνεται από τον καλούντα	**<iq from='juliet@example.com/t3hr0zny'** **id='ol3ba71g'** **to='romeo@example.net/v3rsch1kk3l1jk'** **type='result'/>**	
Ο παραλήπτης επιβεβαιώνει την συνεδρία συμφωνώντας για τις παραμέτρους της κλήσης με ένα μήνυμα 200 OK		**SIP/2.0 200 OK** **Via: SIP/2.0/TCP client.example.com:5060;branch=z9hG4bK74bf9 ;received=192.0.2.101** **From: Juliet Capulet <sip:juliet@example.com>;tag=t** **To: Romeo Montague <sip:romeo@example.net>;tag=v3rsch1kk3l1jk** **Call-ID: 38482762982201885l1@example.com** **CSeq: 1 INVITE** **Contact: <sip:romeo@client.example.net;transport=tc** **Content-Type: application/sdp** **Content-Length: 147** **v=0** **o=romeo 2890844527 2890844527 IN IP4 client.exampl** **s=-** **c=IN IP4 192.0.2.201** **t=0 0** **m=audio 3456 RTP/AVP 0** **a=rtpmap:97 SPEEX/8000** **a=rtpmap:18 G729/8000**
Το παραπάνω μήνυμα μετατρέπεται στο αντίστοιχο μήνυμα session-accept	**<iq from='romeo@example.net/v3rsch1kk3l1jk'** **id='pd1bf839'** **to='juliet@example.com/t3hr0zny'** **type='set'>** **<jingle xmlns='urn:xmpp:jingle:1'** **action='session-accept'** **initiator='juliet@example.com/t3hr0zny'** **responder='romeo@example.net/v3rsch1kk3l1jk'** **sid='a73sjjvkla37jfea'>** **<content creator='initiator'** **media='audio'** **name='this-is-the-audio-content'>** **<description xmlns='urn:xmpp:jingle:app:rtp:1'>**	

	<payload-type id='97' name='speex' clockrate='8000'/> <payload-type id='18' name='G729'/> <payload-type id='0' name='PCMU' clockrate='8000'/> </description> <transport xmlns='urn:xmpp:jingle:transport:raw-udp'> <candidate ip='192.0.2.101' port='49172' generation='0'/> </transport> </content> </jingle> </iq>	
Αν οι παράμετροι της κλήσης επιβεβαιώνονται και από τον καλούντα τότε στέλνει ένα μήνυμα acknowledge	**<iq from='romeo@example.net/v3rsch1kk3l1jk' id='pd1bf839' to='juliet@example.com/t3hr0zny' type='result'/>**	
Σε αυτό το σημείο η συνεδρία έχει οριστεί και τα συμβαλλόμενα μέρη μπορούν να αρχίσουν να ανταλλάσσουν μηνύματα RTP. Χρησιμοποιώντας το codec Speex με συχνότητα ρολογιού 8000		
Θεωρώντας ότι ο παραλήπτης λήγει την σύνοδο έχουμε		**BYE sip:juliet@client.example.com SIP/2.0 Via: SIP/2.0/TCP client.example.net:5060;branch=z9hG4bKnashds7 Max-Forwards: 70 From: Romeo Montague <sip:romeo@example.net>;tag=8321234356 To: Juliet Capulet <sip:juliet@example.com>;tag=9fxced Call-ID: 3848276298220188511@example.com CSeq: 1 BYE Content-Length: 0**
Που θα αντιστοιχιστεί στο μήνυμα	**<iq from='romeo@example.net/v3rsch1kk3l1jk' id='rv301b47' to='juliet@example.com/t3hr0zny' type='set'> <jingle xmlns='urn:xmpp:jingle:1' action='session-terminate' initiator='juliet@example.com/t3hr0zny' reasoncode='no-error' sid='a73sjjvkla37jfea'/> </iq>**	
Και τέλος έχουμε επιβεβαίωση του τερματισμού της	**<iq from='romeo@example.net/v3rsch1kk3l1jk' id='rv301b47' to='juliet@example.com/t3hr0zny'**	

κλήσης, χωρίς να προωθήσουμε το μήνυμα προς τον παραλήπτη	type='result'/>	

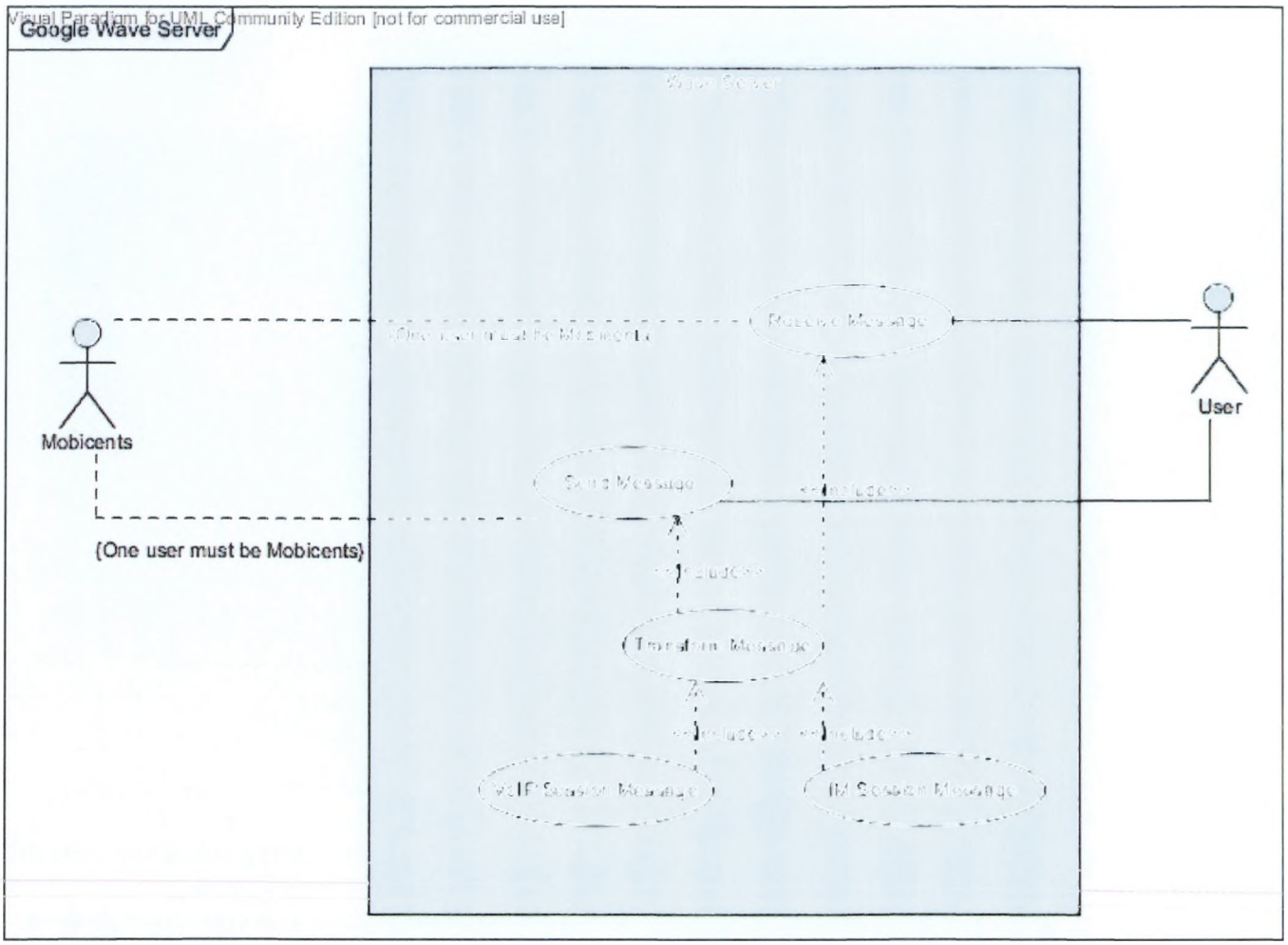
3.4 Google Wave Client.

Μέχρι τώρα έχουμε παρουσιάσει τις βασικές δυνατότητες που πρέπει να έχει το σύστημα. Αρχικά αυτό που θέλαμε ήταν να επιτρέψουμε οι κλήσεις να γίνονται μέσα από το περιβάλλον που προσέφερε το Google Wave μέσω του AppEngine, χρησιμοποιώντας το Robot API για την μεταφορά της σηματοδοσίας και το Gadget API για να παίρνουμε είσοδο από το μικρόφωνο, καθώς και να αναπαράγουμε τον ήχο. Οι δύο αυτές οντότητες θα πρέπει να είναι σε συνεχή επαφή και να ενημερώνει η μία την άλλη, τόσο για τα υποστηριζόμενα codec, όσο και για το πότε μια σύνοδος είναι ενεργή ή όχι.
Αυτό τελικά δεν κατέστη δυνατόν, αφού το περιβάλλον του AppEngine είναι πολύ περιοριστικό, και δεν επιτρέπει να αναπτυχθούν προγράμματα τα οποία να έχουν αμφίδρομη επικοινωνία με το Google Wave, όπως επίσης δεν επιτρέπει να τροποποιήσουμε το ίδιο το πρωτόκολλο.
Έτσι η μόνη επιλογή θα ήταν να αναπτύξουμε κάποιο διαδικτυακό πρόγραμμα γραμμένο σε flash ή κάποια άλλη γλώσσα προγραμματισμού που να μας επιτρέπει να πραγματοποιήσουμε σηματοδοσία. Φυσικά αυτό δεν ήταν μέσα στους στόχους μας.
Για να ξεπεράσουμε τους παραπάνω περιορισμούς αποφασίσαμε να εγκαταστήσουμε έναν δικό μας εξυπηρετητή, ώστε να μπορέσουμε να τροποποιήσουμε το Wave federation protocol, με τρόπο τέτοιο που να επιτρέπει να ενσωματώσουμε το Jingle με το Wave federation protocol στο ένα άκρο της σύνδεσης και το sip στο άλλο.
Στους αρχικούς μας στόχους θέλαμε να προσθέσουμε τα επιπρόσθετα μηνύματα τα οποία καθορίζει το Jingle, καθώς και να τροποποιήσουμε τον μηχανισμό του Wave federation protocol,ώστε να εκμεταλλευτούμε τις δυνατότητες που προσέφερε δημιουργώντας τον τρόπο με τον οποίο τα Wave federation protocol μπορεί να μεταφέρει εκτός από μηνύματα XMPP και αυτά του Jingle.
Λόγω όμως της πολύ κακής τεκμηρίωσης (documentation) που υπήρχε γύρω από τον μέχρι τότε διαθέσιμο κώδικα, καθώς και του γεγονότος ότι δεν ήταν διαθέσιμα όλα τα κομμάτια κώδικα αλλά και της δυσκολίας της ανάγνωσης και κατανόησης ενός τόσο μεγάλου προγράμματος,οδηγηθήκαμε στο να χρησιμοποιήσουμε τα διαθέσιμα API, έτσι ώστε να αναπτύξουμε ένα Robot που θα ήταν μέλος της συζήτησης και υπεύθυνο για την σηματοδοσία.
Επίσης το Robot αυτό θα πρόσεθετε σε κάθε wavelet και ένα Gadget το οποίο θα αναλάμβανε το κομμάτι του RTP και θα επικοινωνούσε με το Robot, με σκοπό να το ενημερώσει για τα διαθέσιμα codec που υποστηρίζονται, αλλά και για να μπορεί να Robot να ελέγχει το Gadget όταν μια σύνοδος είναι ενεργή.
Αυτός είναι μόνο ένα από τους διαθέσιμους τρόπους για να παρέχουμε τις υπηρεσίες για τις οποίες ενδιαφερόμασταν. Ακόμα είχαμε σκεφτεί να χρησιμοποιήσουμε τα Message Stanza του XMPP πάνω στα οποία βασίζεται και το Wave federation protocol και να τα τροποποιήσουμε ώστε να μεταφέρουμε τις απαραίτητες πληροφορίες για μία σύνδεση. Τέλος αξίζει να αναφέρουμε ότι ο πιο σωστός τρόπος για να μεταφέρουμε μηνύματα Jingle θα ήταν να επεκτείνουμε τα διαθέσιμα μηνύματα τα οποία υποστηρίζονται από το πρωτόκολλο, τροποποιώντας κατάλληλα και όλα τα συμβαλλόμενα μέρη τα οποία χρησιμοποιούνται ώστε να διαδώσουν τα Delta, και να κάνουν τα απαραίτητα operation transformations.
Για τον λόγο αυτό εγκαταστήσαμε τον OpenFire, ο οποίος υποστηρίζει το Wave in a box που είναι η ανοιχτού κώδικα εκδοχή του wave federation protocol και προσφέρει το Agent API για να επεκτείνουμε την λειτουργικότητα του.



Εικόνα 3.4: Γενική δομή του συστήματος μας (server to server)

Το πρόβλημα με αυτήν την προσέγγιση είναι ότι το Agent API δεν είναι το ίδιο με τα αντίστοιχα API που παρέχονται από το κλειστού κώδικα federation protocol, και είναι πολύ περιοριστικά ως προς τις δυνατότητες που προσφέρει. Καθώς επίσης στα μελλοντικά σχέδια που έχουν στόχο να αντικατασταθεί σύντομα από άλλα API, όσο ο κώδικας του πρωτοκόλλου σταδιακά θα ανοίγει σε όλους.
Ο server μας θα έπρεπε να είναι σε θέση να χρησιμοποιεί το Wave Federation Protocol σαν το κύριο πρωτόκολλο επικοινωνίας και να δρομολογεί τα μηνύματα στους αποδεκτές τους, με τον τρόπο που έχουμε ήδη περιγράψει στο κεφάλαιο δύο.
Η λογική που ακολουθούμε είναι ίδια με αυτή του Mobicents, δηλαδή ο εξυπηρετητής δρα ως ενδιάμεσος μεταξύ δυο συνομιλητών, δρομολογώντας μηνύματα από άκρο σε άκρο. Με την διαφορά ότι μέσα σε κάθε blip όταν πρόκειται για μήνυμα VoIP περνάμε το περιεχόμενο του Jingle.
Αφού όλα τα μηνύματα ξεκινάνε με τον <iq το μόνο που θα έπρεπε να κάνουμε είναι το Robot να βλέπει αυτά τα μηνύματα, και να τα επεξεργάζεται απαντώντας κατάλληλα για κάθε ένα.
Ακόμα το Gadget θα πρέπει να είναι ρυθμισμένο με τέτοιο τρόπο, που να μην είναι κοινό για όλους, κάτι το οποίο επιτρέπεται στο wave, και έτσι οι ρυθμίσεις του θα είναι διαφορετικές για καθέναν από τους συμμετέχοντες σε ένα Wavelet.
Κάτι πολύ σημαντικό που φαίνεται στο διάγραμμα χρήσης είναι ότι ο server αφού λάβει ένα μήνυμα, τότε το επεξεργάζεται και ανάλογα με τον τύπο του, αποφασίζει τι είδους είναι αυτό και πως πρέπει να το διαχειριστεί. Άμα το μήνυμα αυτό έχει σκοπό να πραγματοποιήσει μια κλήση τότε επεξεργάζεται το μήνυμα που δέχεται, εξάγει την απαραίτητη πληροφορία για να πραγματοποιηθεί μια κλήση και φέρνει σε επαφή τα συμβαλλόμενα μέρη, χρησιμοποιώντας την δυνατότητα που προσφέρεται από το πρωτόκολλο ώστε να αφαιρέσει το μήνυμα από την συζήτηση σηματοδοσίας, χωρίς οι χρήστες να το ξέρουν. Το ίδιο ισχύει και για την περίπτωση που η επικοινωνία είναι τύπου άμεσου μηνύματος, με την διαφορά ότι αντί να αφαιρέσουμε όλη την πληροφορία, αφαιρούμε μόνο το κομμάτι που αφορά το Jingle, αφήνοντας το μήνυμα μέσα στο blip. Έτσι πρακτικά κάθε μήνυμα που στέλνεται στον Server μας "φιλτράρεται" και ανάλογα ακολουθείται η κατάλληλη λογική.



Τέλος θα πρέπει να αναφέρουμε και την εναλλακτική λύση που ήταν να χρησιμοποιήσουμε το πρωτόκολλο πελάτη εξυπηρετητή(Client-server protocol) το οποίο ορίζεται από το Wave federation, και βασίζεται στην διάδοση μηνυμάτων τύπου Json. Αυτό όμως πρακτικά δεν ήταν συμβατό με τις απαιτήσεις που είχαμε θέσει εξ'αρχής, αφού θα ήταν πολύ περιοριστικό στον τρόπο με τον οποίο συμφωνούνται οι σύνοδοι, καθώς επίσης και στον τρόπο με τον οποίο αυτές διατηρούνται.

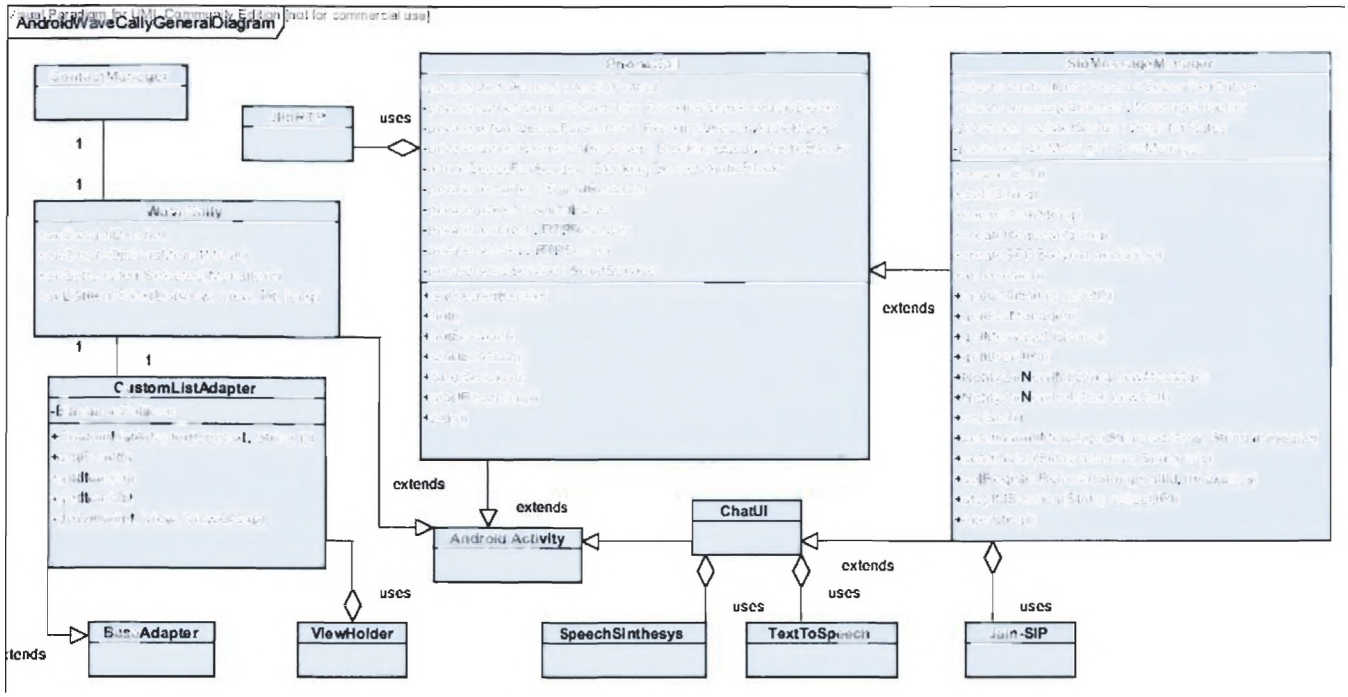
Κεφάλαιο 4

4.1 Android

Στο κεφάλαιο 3 αναφέραμε γενικά στοιχεία και διαγράμματα σχετικά με το τι υπηρεσίες θέλουμε να προσφέρουμε. Στο κεφάλαιο αυτό θα επικεντρωθούμε στην υλοποίηση και στις δυσκολίες αυτής. Με την βοήθεια των διαγραμμάτων της γλώσσας UML θα παρουσιάσουμε σταδιακά το τελικό μας σχέδιο καθώς και τους λόγους που μας οδήγησαν στο να αποκλίνουμε από αυτόν.

Όπως περιγράψαμε και στο κεφάλαιο 3.1, μέσα στους στόχους αυτής της εργασίας είναι και η ανάπτυξη

ενός προγράμματος για το Android. Το διάγραμμα κλάσεων αυτού παρουσιάζεται στο διάγραμμα 4.1.



Το σύστημα μας αποτελείται από την βασική διεπαφή χρήστη που είναι η κλάση WaveCally. Η κλάση αυτή μπορεί να διαχειριστεί τις επαφές του χρήστη με την βοήθεια του ContactManager, που είναι μια κλάση του λειτουργικού συστήματος που μας επιτρέπει να έχουμε πρόσβαση στις επαφές του κινητού τηλεφώνου. Αυτό γίνεται γιατί είναι προτιμότερο να χρησιμοποιήσουμε τον μηχανισμό που παρέχει το λειτουργικό σύστημα ώστε να διαχειριστούμε τις επαφές του χρήστη και έτσι να επιτρέψουμε και σε άλλες εφαρμογές να έχουν πρόσβαση σε αυτές.
Για να παρουσιάσουμε τα αποτελέσματα της παραπάνω διαδικασίας χρησιμοποιούμε μία λίστα στην οποία εκτός από το όνομα και τα στοιχεία του χρήστη βάζουμε και ένα Bitmap (εικόνα).
Επίσης στην λίστα αυτή διαχειριζόμαστε και τις επιλογές που μπορεί να κάνει ο χρήστης ώστε να καλέσουμε ή να στείλουμε κάποιο άμεσο μήνυμα στην επιλεγμένη επαφή.
Το πρώτο πράγμα που κάνει το πρόγραμμα μας όταν ξεκινάει είναι να συνδεθεί με το εξυπηρετητή και να δηλώσει τις ικανότητες του. Αυτή η διαδικασία είναι γνωστή σαν Registration δηλαδή εγγραφή.
Τον ρόλο του Registar server στο δικό μας πρόγραμμα, τον αναλαμβάνει ο Mobicents.
Το παραπάνω σχήμα παρουσιάζει μια γενική εικόνα του συστήματος μας από την μεριά του πελάτη (κινητή συσκευή).
Όπως εύκολα μπορεί να γίνει αντιληπτό, η βασικότερη κλάση μας είναι η SipMessageManager η οποία προσφέρει μεθόδους για εύκολη διαχείριση των μηνυμάτων που ανταλλάσσονται, όπως επίσης και για την αρχικοποίηση του επιπέδου που είναι υπεύθυνο για την σηματοδοσία.
Ακόμα περιέχει στατικές μεθόδους και πεδία, έτσι ώστε να μπορούμε να έχουμε πρόσβαση στις υπηρεσίες που προσφέρει από διαφορετικές κλάσεις του προγράμματος. Οδηγηθήκαμε σε αυτήν την απόφαση καθώς με αυτόν τον τρόπο μπορούμε να μοιράσουμε τις υπηρεσίες αυτές χωρίς να πρέπει να μετατρέπουμε τα αντικείμενα των κλάσεων με την χρήση του μηχανισμού Parcel που είναι ένας τρόπος που προσφέρεται από το android για επικοινωνία μεταξύ δύο διαφορετικών διεργασιών (Inter Process Communication).
Ξεκινώντας το πρόγραμμα θα εμφανιστεί το γραφικό ενδιάμεσο που περιγράψαμε παραπάνω, δίνοντας μας την δυνατότητα είτε να επιλέξουμε την επαφή και να της στείλουμε κάποιο μήνυμα, είτε να επιλέξουμε το

εικονίδιο που βρίσκεται δίπλα από το όνομα της επαφής ώστε να κληθεί ο χρήστης.
Στο σημείο αυτό θα αναλύσουμε τον τρόπο λειτουργίας για την αποστολή και λήψη μηνυμάτων.

Όσον αφορά την διαδικασία αυτή , όταν ο χρήστης επιλέξει να πατήσει πάνω στο όνομα μίας επαφής, εμφανίζεται ένα γραφικό ενδιάμεσο το οποίο ονομάζεται ChatUI όπως μπορούμε να διακρίνουμε και στο σχήμα 4.0.1

Εικόνα 4.0.1: ChatUI

Αυτό το ενδιάμεσο (ChatUI) μας παρέχει την δυνατότητα να βλέπουμε τα μηνύματα που μας στέλνουν οι χρήστες, καθώς και να γράψουμε και να στείλουμε εμείς μηνύματα. Επιπρόσθετα προσφέρει δύο υπηρεσίες, από την μια πλευρά να μπορεί να διαβάζει ένα κείμενο και να το μετατρέπει σε ομιλία με την βοήθεια της τεχνολογίας TextToSpeech που προσφέρει το λειτουργικό σύστημα, κάτι που γίνεται αυτόματα μόνο όταν ο χρήστης έχει επιλέξει την κατάλληλη επιλογή που δίνεται μέσα από το menu της εφαρμογής. Σε αντίθετη περίπτωση το μήνυμα αυτό απλά θα προστεθεί στην λίστα της συνομιλίας. Για να γίνει αυτό θα πρέπει η κλάση η οποία είναι υπεύθυνη για τον χειρισμό των μηνυμάτων να επεκτείνει (extends) την TextToSpeech.OnInitListener και να υλοποιεί την μέθοδο onInit().
Το αντίστροφο γίνεται (από ομιλία σε κείμενο) χρησιμοποιώντας την υπηρεσία του λειτουργικού, ACTIVITY_RECOGNIZE_SPEECH όταν κάνουμε την αντίστοιχη χειρονομία (gesture) την οποία μπορεί να ορίσει ο χρήστης. Ακόμα θα πρέπει να ρυθμίσουμε το Speech Synthesys να αναγνωρίζει προτάσεις και όχι μόνο λέξεις.
Οι δύο αυτές υπηρεσίες μπορούν να χρησιμοποιηθούν μόνο σε κινητά τα οποία έχουν εγκατεστημένο το Android Market αφού προστατεύονται από άδειες χρήσης.
Εδώ οφείλουμε να πούμε ότι το ChatUI συγκριτικά με την κλάση PhoneCall είναι πιο απλή στην υλοποίηση της. Αυτό γιατί μπορούμε να την υλοποιήσουμε χρησιμοποιώντας λιγότερες μεθόδους αλλά και γιατί δεν χρειάζεται να χρησιμοποιήσουμε το πρωτόκολλο SDP και RTP, από την στιγμή που το μόνο που μας ενδιαφέρει είναι να μεταφέρουμε μηνύματα κειμένου, κάτι που μπορεί να γίνει κάλλιστα χρησιμοποιώντας το payload του πακέτου.
Οι μέθοδοι που χρησιμοποιούνται για την μεταφορά του μηνύματος είναι:

- Η sendInvite που χρησιμοποιείται για να συνομιλήσουμε με έναν χρήστη.
- Η sendInstantMessage για να στείλουμε τα μηνύματα που θέλουμε.
- Η createRequest για να δημιουργήσουμε ένα αίτημα τύπου Sip.
- Η stopIMSession όταν το πρόγραμμα μας τερματίζεται και θέλουμε να ελευθερώσουμε την σύνοδο.

- Και τέλος η setRegisterRefresh για να στέλνουμε μηνύματα ώστε να κρατάμε την σύνοδο ανοιχτή.

Όλες οι παραπάνω μέθοδοι βρίσκονται στην κλάση SipMessageManager.

Αυτές οι λειτουργίες γίνονται αντιληπτές από οποιονδήποτε χρήστη του προγράμματος. Βέβαια αυτό που είναι πραγματικά ενδιαφέρον είναι το τί πρέπει να γίνει ώστε να μπορέσουμε να παρέχουμε αυτές τις υπηρεσίες.

Αρχικά όπως αναφέραμε πρέπει να συνδεθούμε με τον Registar Server για να μπορούμε να δεχτούμε και να λάβουμε κλήσεις. Αυτό το βήμα περιγράφεται στο RFC 3261 ως registrations(κεφάλαιο 10.1).

Αυτή η σύνδεση θα πρέπει να παραμείνει ενεργή για όλη την διάρκεια που το πρόγραμμα μας παραμένει ανοιχτό. Γι΄ αυτό χρησιμοποιούμε ένα component που παρέχει το Android, το Service, και μπορεί να διατηρεί την σύνδεση ενεργή χωρίς να χρειάζεται το πρόγραμμα να είναι εμφανές όλη την ώρα. Για να το πετύχει αυτό πρέπει ανά τακτά χρονικά διαστήματα τα οποία δεν πρέπει ξεπερνούν σε χρόνο αυτόν που ορίζεται στο πεδίο λήξης χρόνου του μηνύματος SIP, να στέλνουμε μηνύματα τύπου keep alive. Αυτό είναι ένας τρόπος να δηλώνουμε πάντα στον εξυπηρετητή ότι επιθυμούμε να διατηρήσουμε την σύνδεση ανοιχτή, ώστε όταν κάποιος θελήσει να επικοινωνήσει μαζί μας να μας ειδοποιήσει. Πρέπει να έχουμε κατά νου ότι η συχνότητα με την οποία στέλνουμε μηνύματα τύπου keep alive μπορεί να έχει σημαντικές επιπτώσεις στην κατανάλωση πόρων, και έτσι θα πρέπει να είμαστε πολύ προσεχτικοί με τους χρόνους που θα επιλέξουμε τόσο για το κάθε πότε θα πρέπει να λήγει ένα registration μήνυμα όσο και με το κάθε πότε θα το ανανεώνουμε. Για να έχουμε καλύτερο αποτέλεσμα θα πρέπει να εξομοιώσουμε όλες τις πιθανές περιπτώσεις, κάτι που είναι εκτός των ορίων της εργασίας.

Για να έχουμε μία καλύτερη εικόνα σχετικά με την δομή του προγράμματος παραθέτουμε την εικόνα 4.1.1 καθώς και την εικόνα 4.1.2 στην οποία μέσα από ένα sequence diagram παρουσιάζουμε την διαδικασία του Registration, την οποία ακολουθούμε πάντα ανεξάρτητα του είδους της επικοινωνίας.

Η διαδικασία άμεσου μηνύματος μπορεί να διαιρεθεί σε δύο μέρη, την αποστολή και την λήψη.

Κατά την αποστολή θα πρέπει να διαβάζουμε από το γραφικό ενδιάμεσο (ChatUI) το μήνυμα που έχει πει ή γράψει ο χρήστης, και να τροφοδοτεί την κλάση που αναλαμβάνει το επίπεδο της σηματοδοσίας με αυτό. Κατά την αποστολή το μόνο που πρέπει να κάνουμε είναι να δημιουργήσουμε την διεύθυνση του αποστολέα και του παραλήπτη με την βοήθεια της κλάσης SipURI. Αυτό γιατί βάσει του RFC κάθε μήνυμα πρέπει να έχει έναν αποστολέα και έναν παραλήπτη επόμενο είναι να βρούμε ένα Dialog για το μήνυμα αυτό έτσι είτε θα χρησιμοποιήσουμε έναν ήδη υπάρχον, αν υπάρχει κάποια κλήση, είτε θα δημιουργήσουμε ένα χρησιμοποιώντας την μέθοδο createRequest την κλάσης Dialog. Αυτό γίνεται για να μπορέσουμε να κατασκευάσουμε το μήνυμα τύπου SIP. Αυτή είναι μια διαδικασία που ακολουθούμε κάθε φορά που θέλουμε να δημιουργήσουμε ένα τέτοιο μήνυμα ανεξάρτητα από την χρήση του και αφορά την κατασκευή μηνύματος με την χρήση του Jain-Sip stack. Σύμφωνα με το RFC αυτό που θα αλλάζουμε στα μηνύματα είναι το είδος του μηνύματος. Έτσι για να δείξουμε ότι το μήνυμα αυτό είναι τύπου άμεσου μηνύματος θα πρέπει στο τύπο περιεχομένου του μηνύματος (Content Type), να ορίσουμε ότι πρόκειται για plain/text μήνυμα. Το τελευταίο πράγμα που χρειάζεται για να σταλεί το μήνυμα είναι να δημιουργήσουμε ένα ClientTransaction μέσω του οποίου θα στείλουμε το μήνυμα. Για να γίνει αυτό χρησιμοποιούμε την μέθοδο getNewClientTransaction.

Κατά την λήψη του μηνύματος το ChatUI θα ειδοποιηθεί από την κλάση MessageListener που επεκτείνει (implements) την SipListener απο το Jain-Sip.

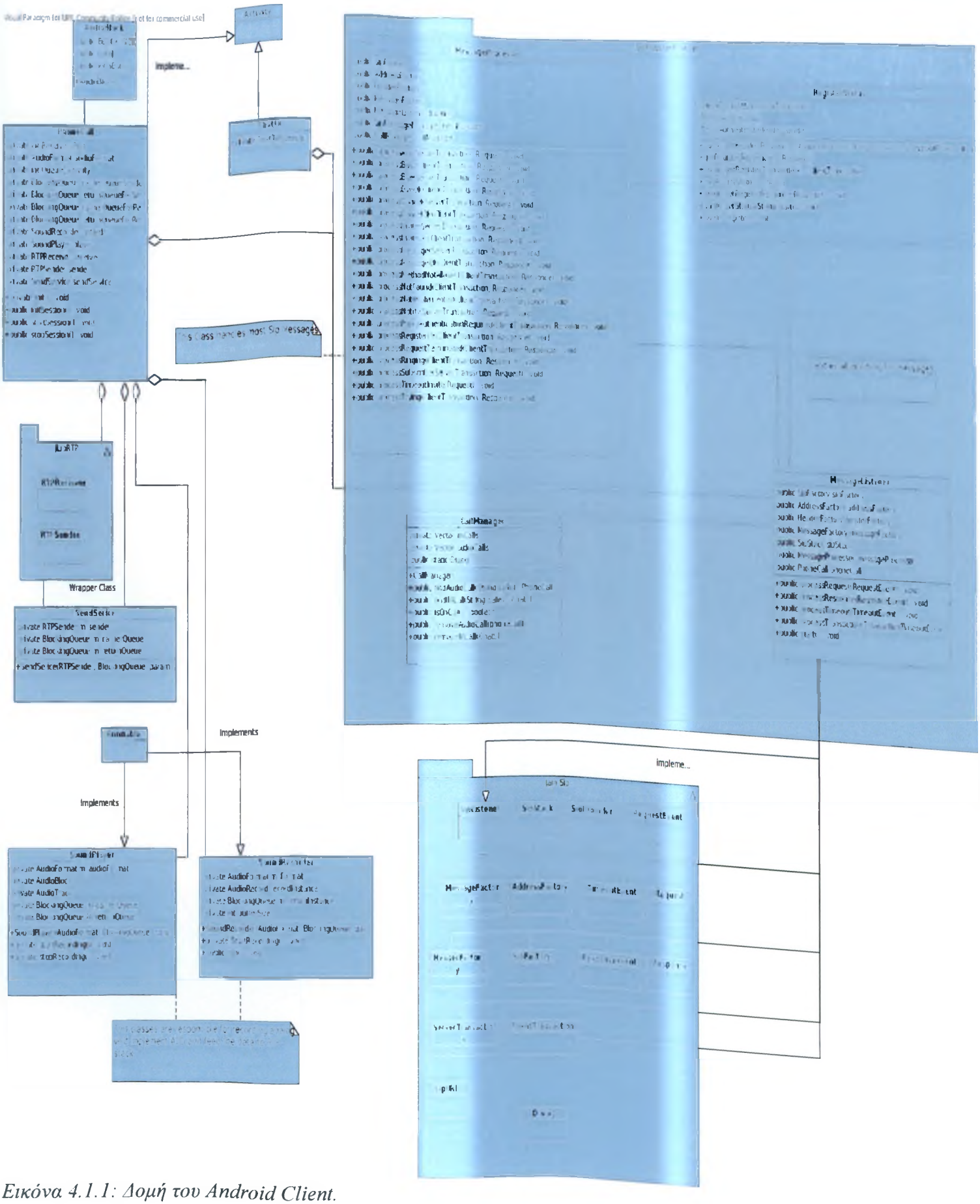
Σε αυτήν την κλάση θα χρησιμοποιήσουμε την μέθοδο processRequest, η οποία εκτελείται όταν έρθει ένα νέο μήνυμα. Σε αυτήν την μέθοδο φτιάχνουμε ένα ServerTransaction αφού σύμφωνα με το RFC όταν λαμβάνουμε ένα νέο μήνυμα πρέπει να ενεργήσουμε σαν server entity ώστε να ολοκληρωθεί η συναλλαγή. Η συναλλαγή αυτή θα χαρακτηριστεί από το αντίστοιχο transaction-id που βρίσκουμε στο ληφθέν μήνυμα.

Τέλος θα διαχωρίσουμε τα μηνύματα βάσει του τύπου τους, αυτό γιατί η μέθοδος processRequest, έχει σχεδιαστεί ώστε να μπορεί να χρησιμοποιηθεί για όλους τους τύπους μηνυμάτων.

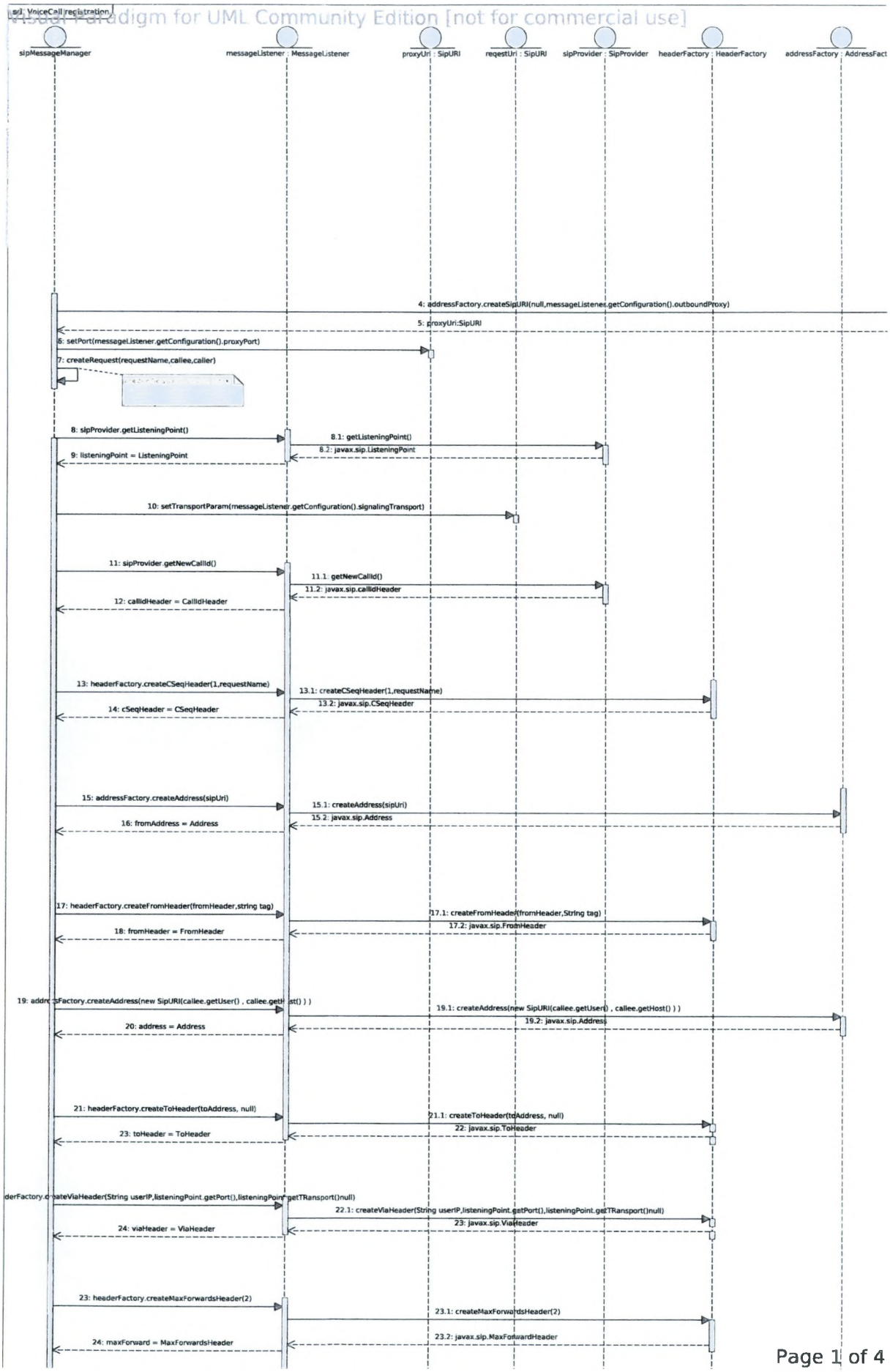
Αφού επεξεργαστούμε το μήνυμα αν αυτό πρόκειται για ένα μήνυμα τύπου MESSAGE τότε θα καλέσουμε την μέθοδο processMessage απο την κλάση MessageProcessor που είναι υπεύθυνη για να επεξεργαστεί το μήνυμα ώστε να εξάγουμε το περιεχόμενο του και να ενεργοποιήσουμε το ChatUI για να μπορέσει ο χρήστης να συνομιλήσει.

Για να είναι σε θέση η εφαρμογή να προσφέρει υπηρεσίες μεταφοράς φωνής, η διαδικασία που θα πρέπει να ακολουθήσουμε είναι διαφορετική και πολύ πιο περίπλοκη. Αυτό ισχύει λόγω των διαφορετικών δυνατοτήτων που υπάρχουν τόσο στο επίπεδο δικτύου όσο και στις ίδιες τις συσκευές.

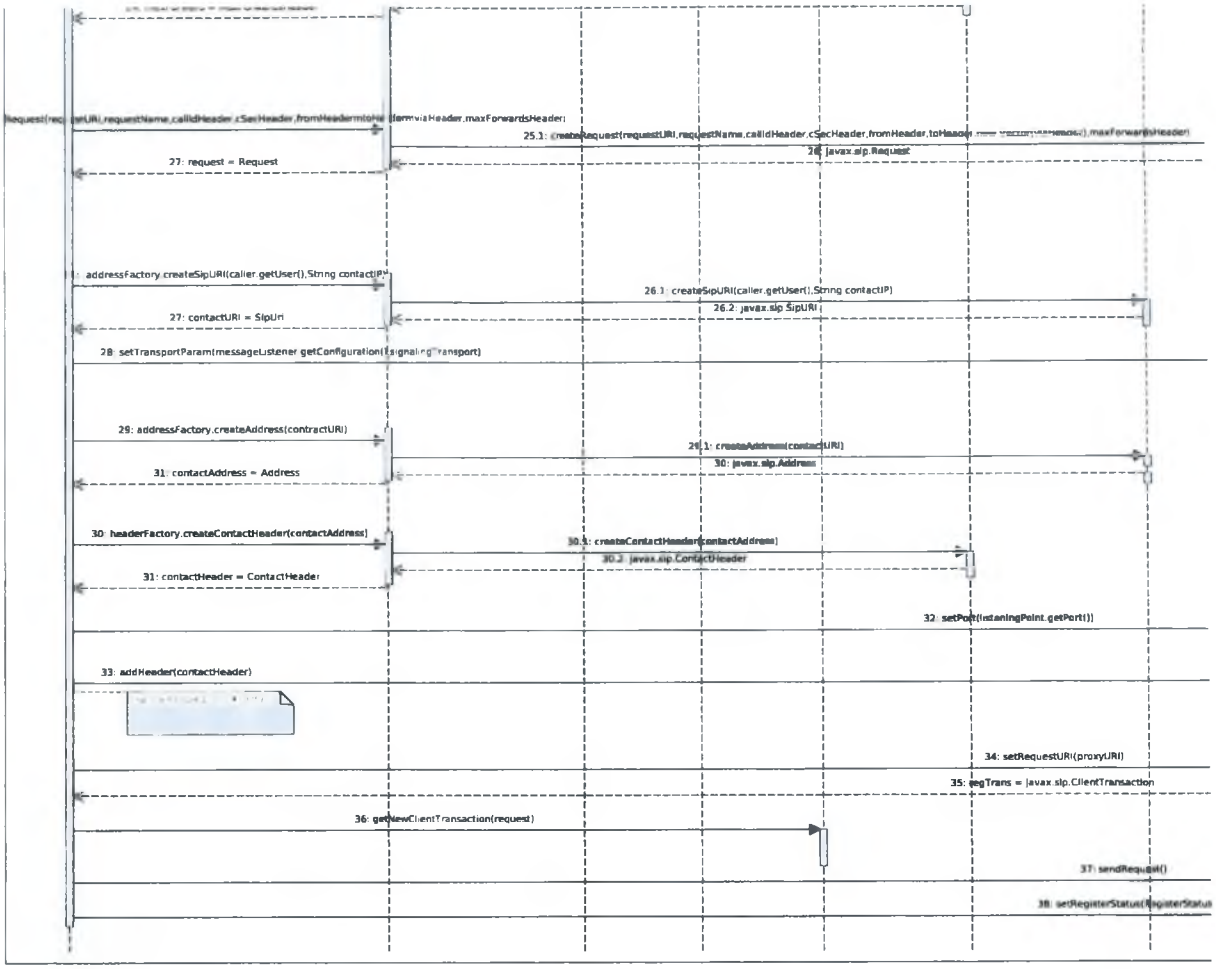
Έτσι στο επίπεδο του δικτύου έχουμε να αντιμετωπίσουμε προβλήματα που αφορούν την αξιοπιστία και την απόδοση του ίδιου του δικτύου. Οι επιπτώσεις αυτές δεν γίνονται τόσο πολύ αισθητές όταν η συσκευή βρίσκεται εντός ενός ασύρματου τοπικού δικτύου (wlan) αλλά όταν η συσκευή είναι συνδεδεμένη στο δίκτυο κινητής τηλεφωνίας, όπου η διαφορά στην ποιότητα σήματος μπορεί να ποικίλει. Για τον λόγο αυτό πρέπει πριν την πραγματοποίηση μιας κλήσης να ελέγχουμε την ποιότητα της σύνδεσης και αν αυτή κριθεί ανεπαρκής να μην επιτρέψουμε στον χρήστη να πραγματοποιήσει την κλήση. Για να το πετύχουμε αυτό χρησιμοποιούμε τις δυνατότητες τις οποίες παρέχει το Android και επιτρέπουμε την κλήση να πραγματοποιηθεί μόνο σε περίπτωση που το δίκτυο είναι τύπου 3G ή η συσκευή βρίσκεται συνδεδεμένη σε κάποιο ασύρματο τοπικό δίκτυο. Με τον τρόπο

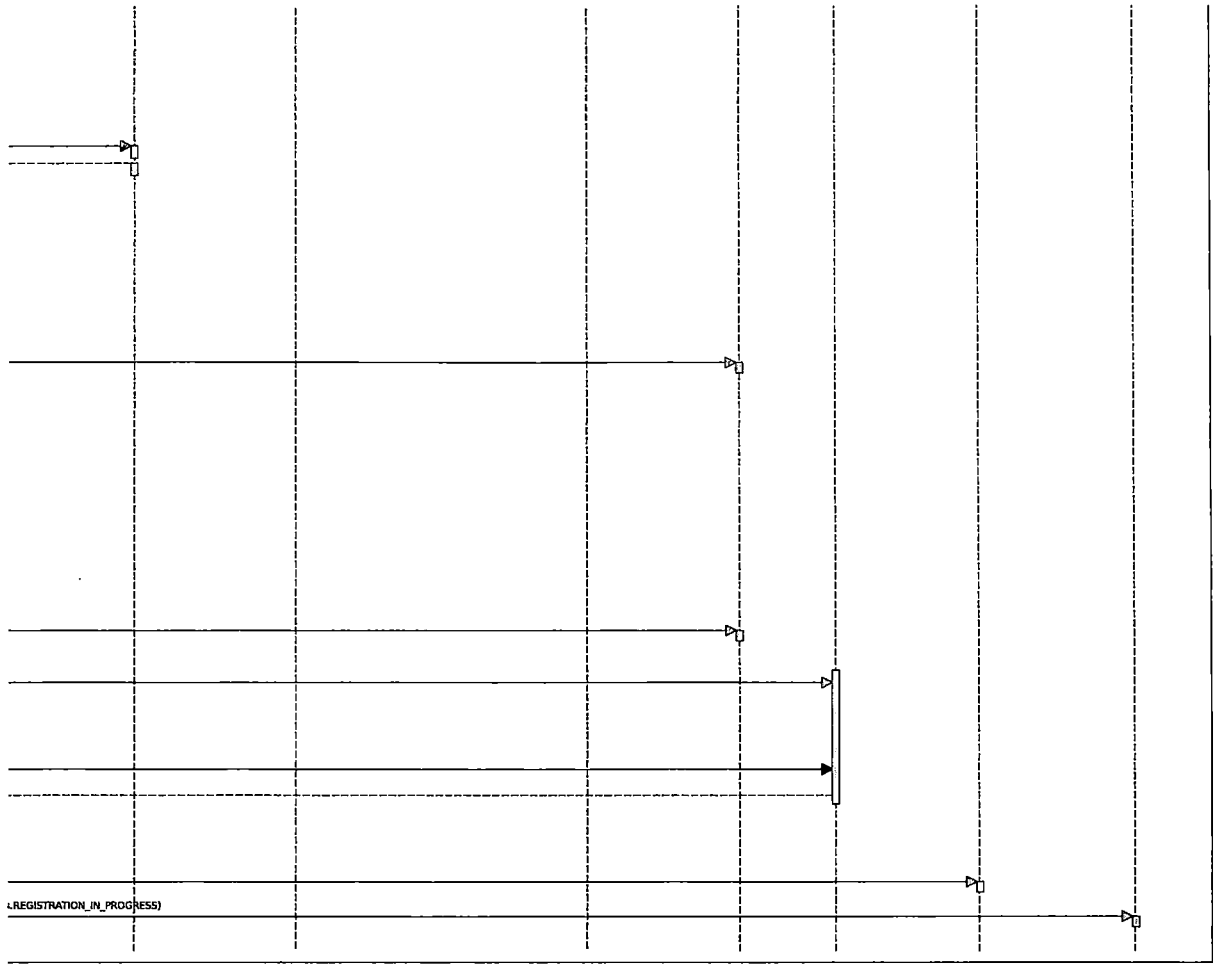


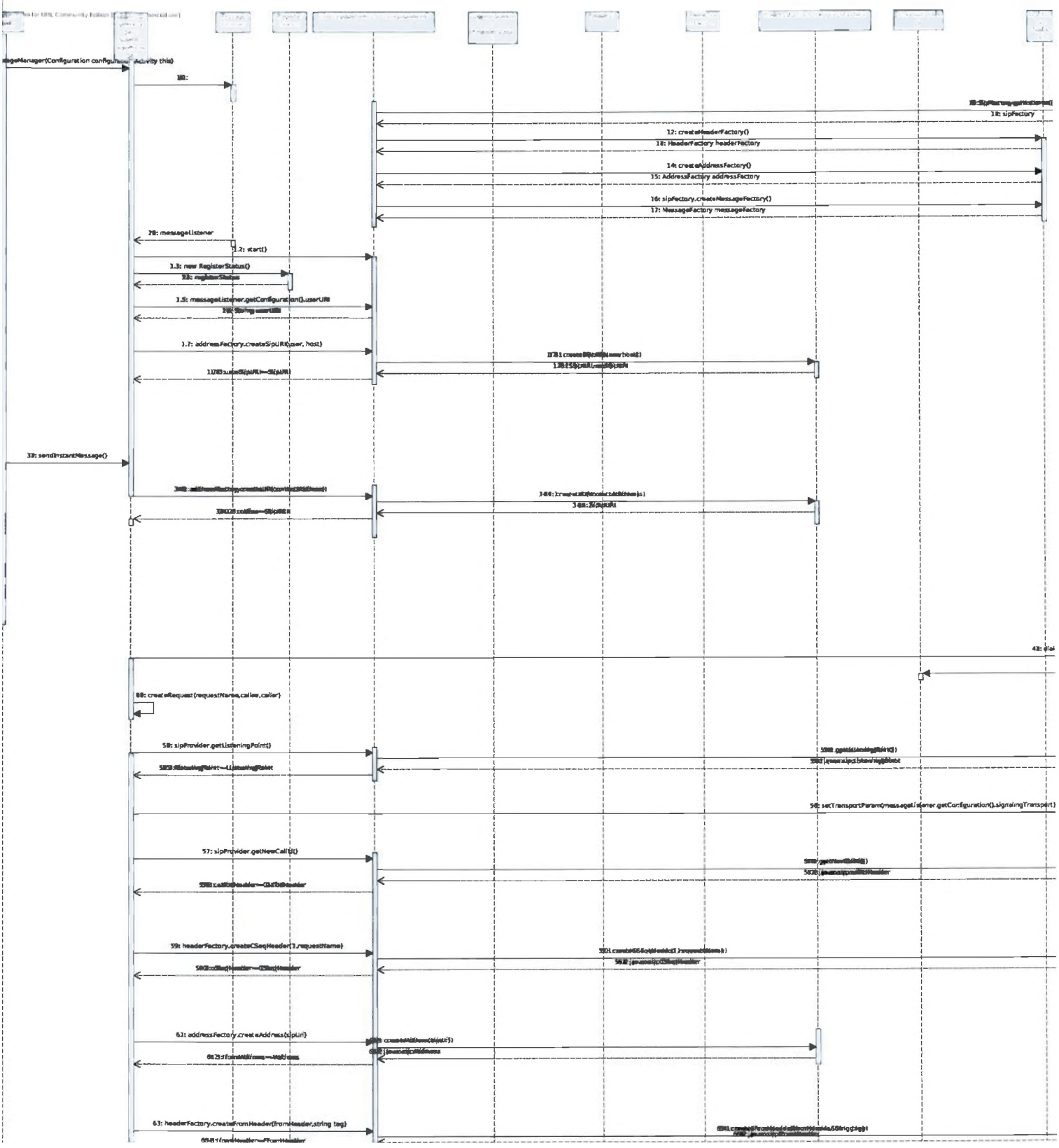
Εικόνα 4.1.1: Δομή του Android Client.



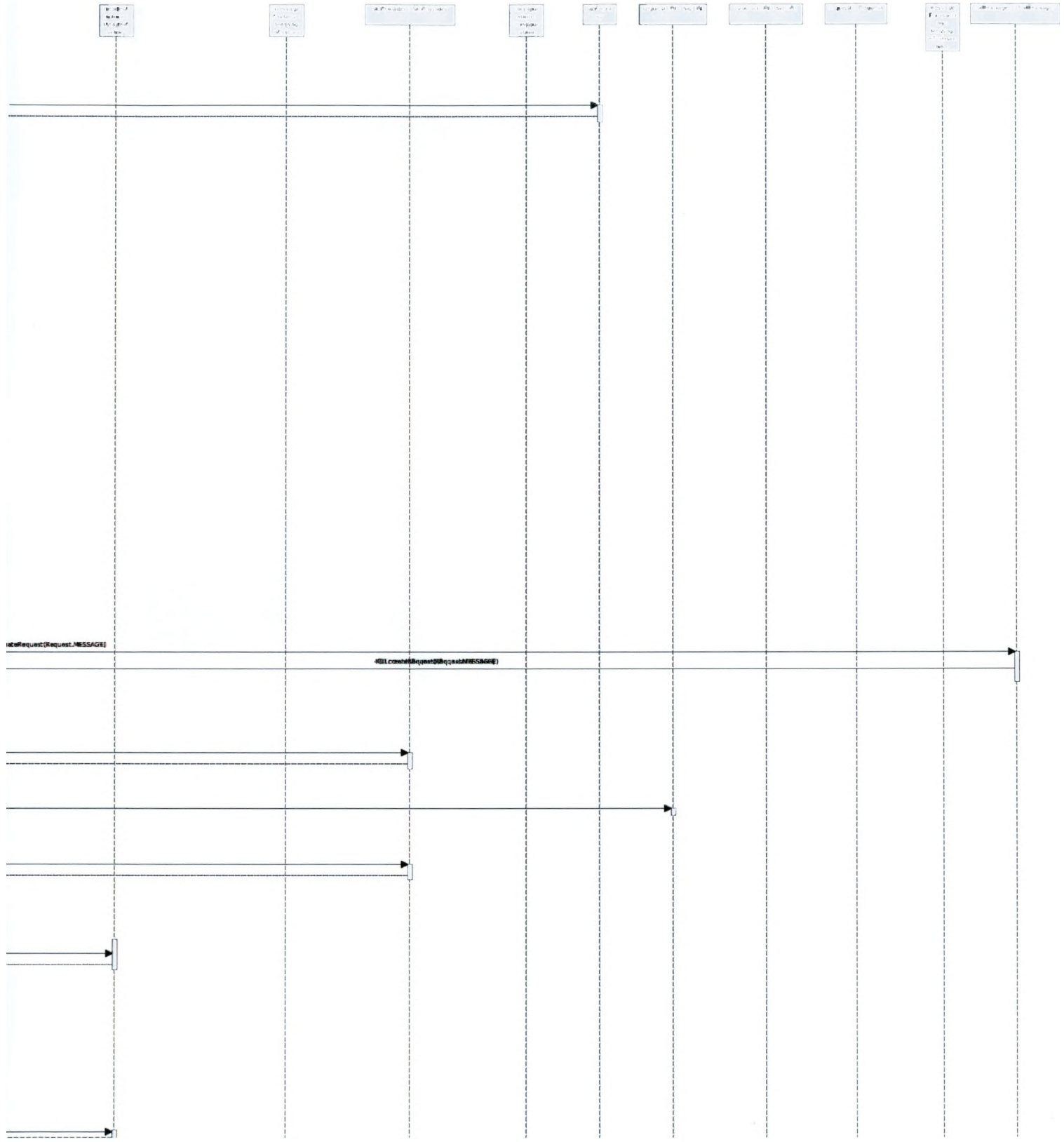

messageFactory : MessageFactory
contactURI
request : Request
regTrans : ClientTransaction
RegistrationStatus
1: new MessageListener(activity,configuration)
2: messageListener:MessageListener

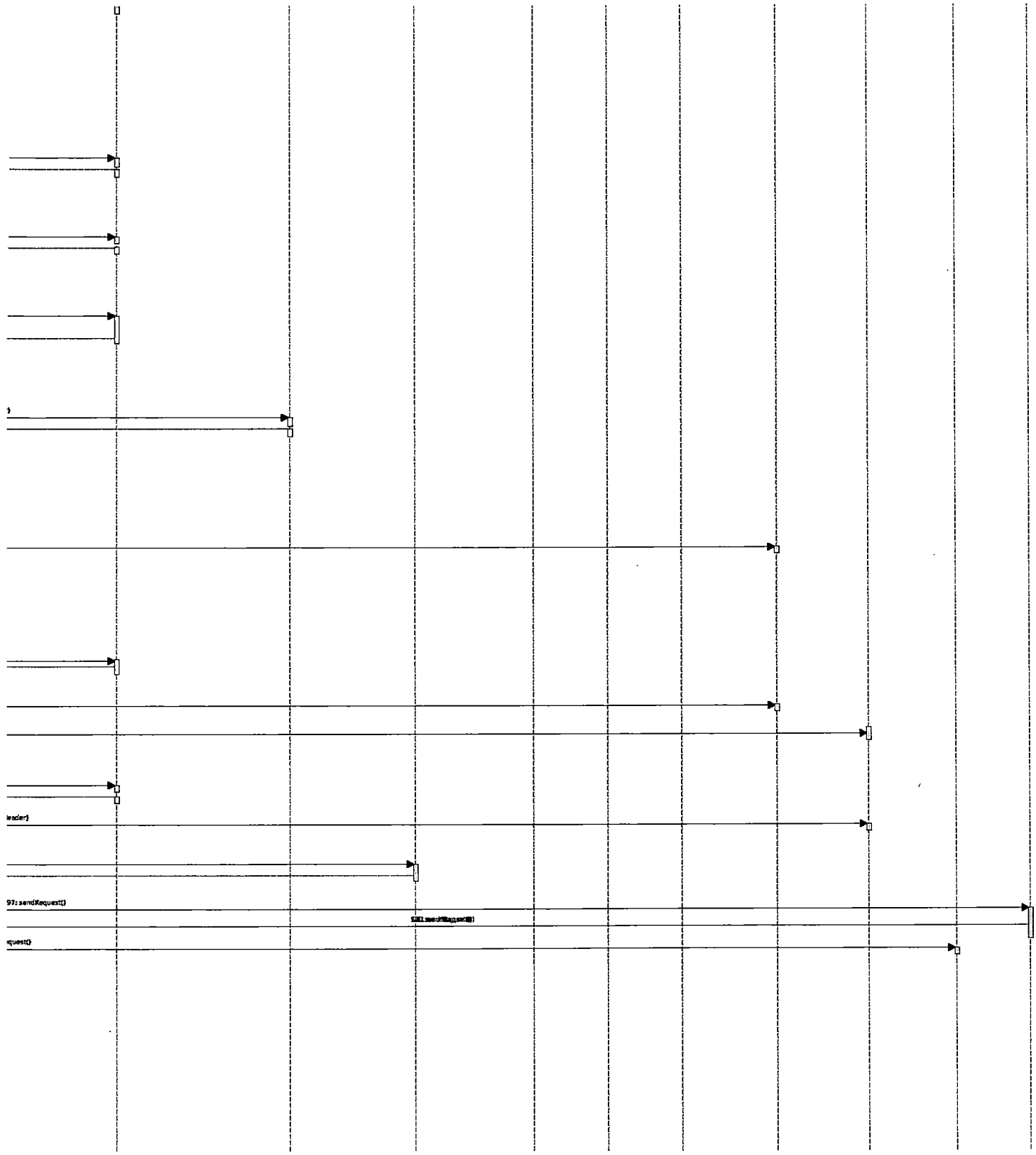

26: javax.sip.Request
27: request = Request
26.1: createSipURI(caller.getUser(),String contactIP)
26.2: javax.sip.SipURI
27: contactURI = SipUri
28: setTransportParam(messageListener.getConfiguration(signalingTransport)
29: addressFactory.createAddress(contractURI)
30: javax.sip.Address
31: contactAddress = Address
30: headerFactory.createContactHeader(contactAddress)
30.2: javax.sip.ContactHeader
31: contactHeader = ContactHeader
32: setPort(listaningPoint.getPort())
33: addHeader(contactHeader)
34: setRequestURI(proxyURI)
35: reqTrans = javax.sip.ClientTransaction
36: getNewClientTransaction(request)
37: sendRequest()

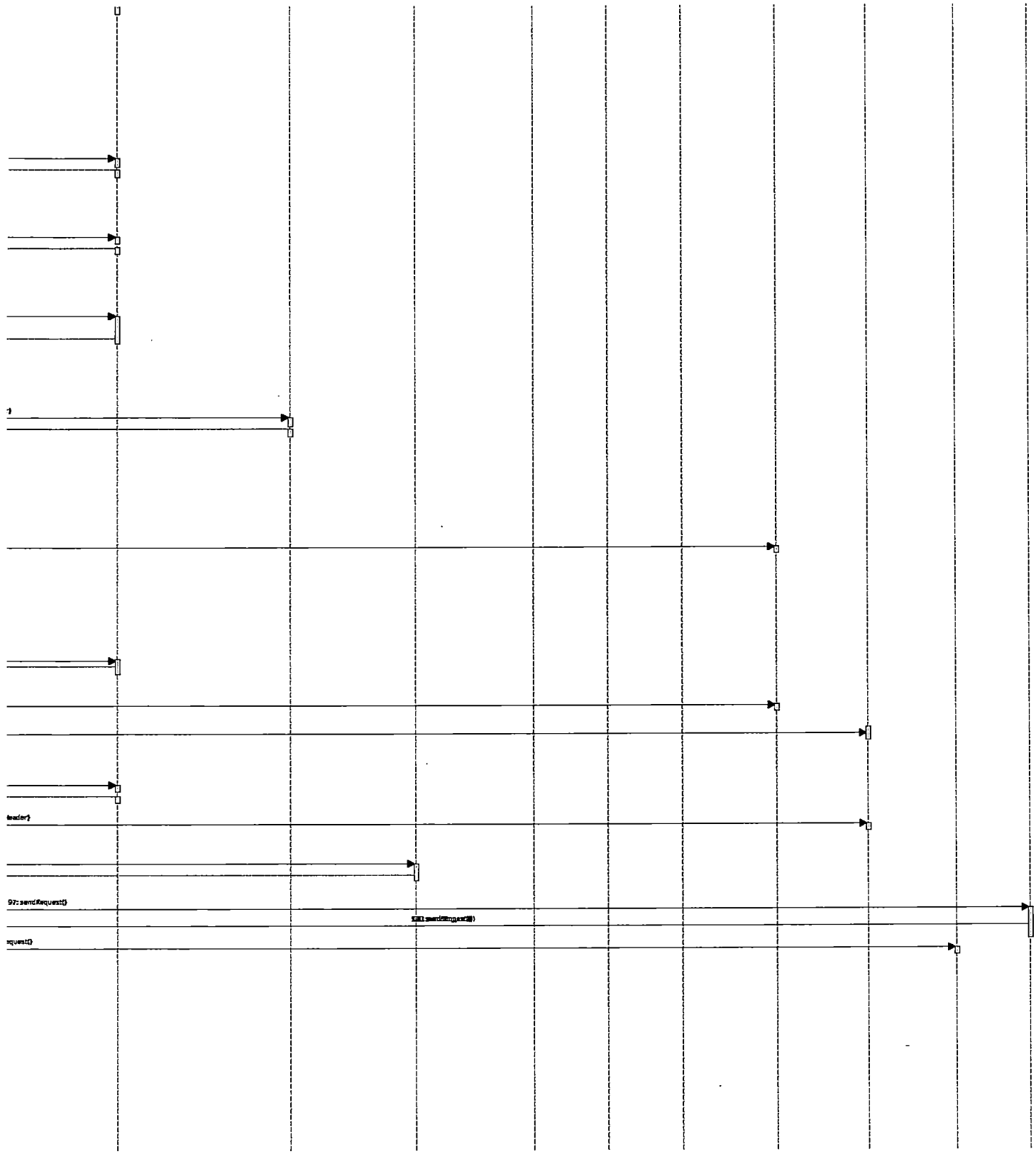

:REGISTRATION_IN_PROGRESS)


12: createHeaderFactory()
13: HeaderFactory headerFactory
14: createAddressFactory()
15: AddressFactory addressFactory
16: sipFactory.createMessageFactory()
17: MessageFactory messageFactory
18: sipFactory
20: messageListener
1.2: start()
1.3: new RegisterStatus()
1.5: messageListener.getConfiguration().userURI
1.7: addressFactory.createSipURI(user, host)
33: sendInstantMessage()
48: dial
49: createRequest(requestName,callee,caller)
58: sipProvider.getListeningPoint()
56: setTransportParam(messageListener.getConfiguration().signalingTransport)
57: sipProvider.getNewCallId()
59: headerFactory.createCSeqHeader(1,requestName)
61: addressFactory.createAddress(sipUri)
63: headerFactory.createFromHeader(fromHeader,string tag)

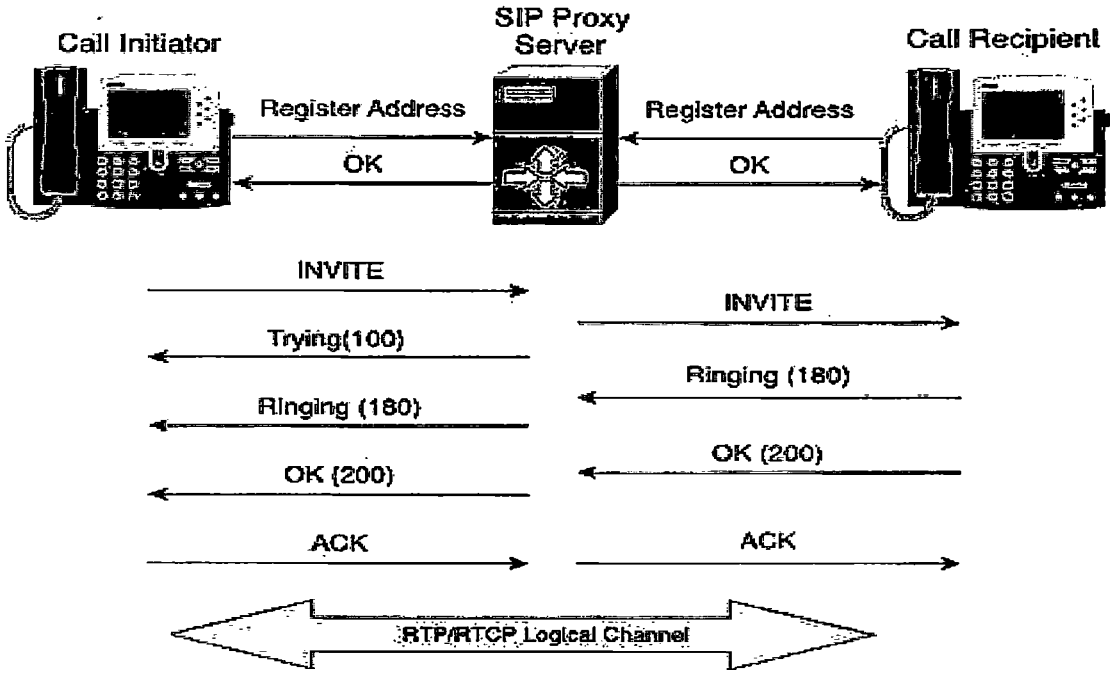

66: addressFactory.createAddress(new SipURI(callee.getUser() , callee.getHost()))
67: headerFactory.createToHeader(toAddress, null)
67: createToHeader(toAddress, null)
68: javax.sip.ToHeader
69: toHeader = ToHeader
68: headerFactory.createViaHeader(String userIP,listeningPoint.getPort(),listeningPoint.getTransport(),null)
70: javax.sip.ViaHeader
71: viaHeader = ViaHeader
69: headerFactory.createMaxForwardsHeader(2)
72: javax.sip.MaxForwardsHeader
73: createRequest(requestURI,requestName,callIdHeader,cSeqHeader,fromHeader,toHeader,new Vector(viaHeader),maxForwards,
74: javax.sip.Request
75: request = Request
70: addressFactory.createSipURI(caller.getUser(),String contactIP)
76: javax.sip.SipURI
77: setTransportParam(messageListener.getConfiguration().signalingTransport)
73: addressFactory.createAddress(contactURI)
73.2: contactAddress = Address
78: javax.sip.Address
74: headerFactory.createContactHeader(contactAddress)
79: javax.sip.ContactHeader
75: setPort(listeningPoint.getPort())
77: addHeader(contactHeader)
80: headerFactory.createContentTypeHeader
"text",
"plain;charset=UTF-8")
81: contentTypeHeader
82: contentTypeHeader=ContentTypeHeader
98: setContent(message, conten
98: sipProvider.getNewClientTransaction(messageRequest)
98.1: getNewClientTransaction(messageRequest))
98.2: ClientTransaction
98.3: messageTransaction= ClientTransaction
98: s







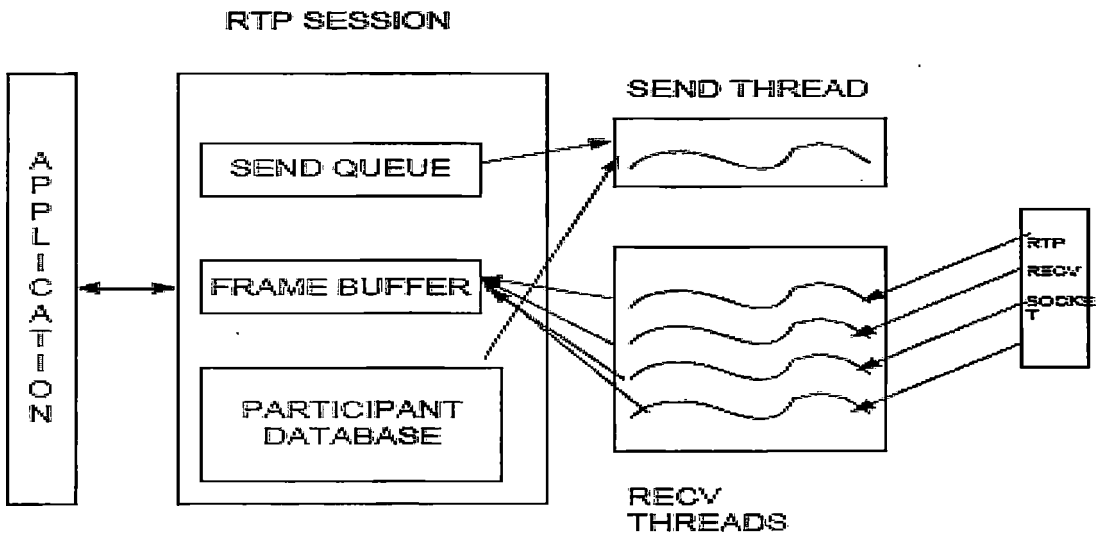
αυτό εκτός του ότι περιορίζουμε τις δυνατότητες επικοινωνίας κάποιου χρήστη, (αφού πρακτικά η κάλυψη που προσφέρεται από το δίκτυο δεν είναι πάντα καλής ποιότητας), δεν μπορεί να εξασφαλίσει σίγουρα ότι η κλάση παρότι το δίκτυο είναι τύπου 3G θα μεταφερθεί χωρίς προβλήματα και με καλή ποιότητα ήχου.
Μία απλή διαδικασία κλήσης όπως περιγράφεται από το RFC παρουσιάζεται στο παρακάτω σχήμα



Για να μπορέσουμε να έχουμε καλύτερο έλεγχο πάνω στον τρόπο με τον οποίο θα γίνουν οι κλήσεις, χρησιμοποιούμε μια κλάση η οποία έχει γενικό χαρακτήρα (Global scope). Αυτή η κλάση θα είναι υπεύθυνη για να τροποποιεί αλλά και να ενημερώνει όλες τις ενδιαφερόμενες κλάσεις για την παρούσα κατάσταση του χρήστη την δεδομένη στιγμή. Έτσι θα μπορούμε όταν παραδείγματος χάριν δεν πληρούνται οι περιορισμοί που αναφέραμε παραπάνω, να εμφανίζουμε ένα μήνυμα λάθους στον χρήστη. Ή όταν ο χρήστης βρίσκεται ήδη σε τηλεφωνική κλήση να μην του επιτρέπουμε να πραγματοποιήσει και άλλη κλήση. Αυτή η κλάση είναι η CallManager. Έτσι πριν ξεκινήσουμε μία καινούργια κλήση θα πρέπει πρώτα να ελέγξουμε άμα ο χρήστης μας, βρίσκεται ήδη σε κλήση. Αυτό γίνεται μέσα από τις μεθόδους που παρέχει η κλάση αυτή και συγκεκριμένα από την isAlreadyInAudioCall() η οποία επιστρέφει true ή false ανάλογα. Για την περίπτωση που περιγράφουμε θεωρούμε ότι ο χρήστης ξεκινάει για πρώτη φορά μία κλήση, οπότε η κατάσταση του θα είναι IDLE δηλαδή αδρανής, που είναι και η αρχική κατάσταση στην οποία βρίσκεται εξ ορισμού. Άλλες καταστάσεις είναι η onIM, όταν υπάρχει τουλάχιστον μία σύνοδος άμεσου μηνύματος, inCall, στην περίπτωση που βρίσκεται σε κλήση,
NOT_IN_A_CALL,TRYING,RINGING
,INCOMING_CALL,BUSY,TEMPORARY_UNAVAILABLE,CANCEL και τέλος unregistered στην περίπτωση λάθους ή στην περίπτωση που απλά ο χρήστης δεν έχει ξεκινήσει ακόμα την διαδικασία Registration. Αυτή είναι και η αρχική κατάσταση με την οποία ξεκινάει το πρόγραμμα. Αφού ελέγξουμε την κατάσταση του χρήστη, τότε αρχικοποιούμε την κλάση Call, η οποία είναι υπεύθυνη για να διατηρεί πληροφορίες που έχουν να κάνουν με την τρέχουσα σύνοδο, όπως την διεύθυνση του συνομιλητή μας, πληροφορίες σχετικά με το Sip Dialog που σχετίζεται με αυτήν την κλήση, την κατάσταση(status) αυτής, καθώς και την διεύθυνση SIP(Sip Uri) του συνομιλητή.
Όπως αναφέραμε προηγουμένως η πραγματοποίηση μιας κλήσης φωνής, είναι πολύ πιο περίπλοκη διαδικασία, σε σχέση με την απλή αποστολή γραπτών μηνυμάτων. Αυτό έγκειται σε μεγάλο ποσοστό στους

περιορισμούς που μπορεί να παρουσιαστούν από την συσκευή στην οποία τρέχει το πρόγραμμα μας. Αφού λόγω της πληθώρας συσκευών που υποστηρίζουν το Android, υπάρχει διαφορά στις επεξεργαστικές δυνατότητες που προσφέρει κάθε μια. Για να ξεπεραστεί αυτό, αλλά και για να μπορέσει το πρόγραμμα μας να λειτουργήσει με άλλα Softphone της αγοράς, πρέπει να είμαστε όσο το δυνατόν πιο κοντά σε αυτά που ορίζονται από το specification. Έτσι θα πρέπει να πληροφορούμε τους συνομιλητές μας σε επίπεδο εφαρμογής για τις δυνατότητες που έχει η συσκευή μας. Για τον λόγο αυτό θα πρέπει να χρησιμοποιήσουμε το πρωτόκολλο SDP. Αυτό παρέχει έναν εύκολο, πολύ καλά ορισμένο τρόπο με τον οποίο μπορούμε να παρέχουμε επαρκείς πληροφορίες για να εγκαθιδρύσουμε ή να συμμετάσχουμε σε μία σύνοδο πολυμέσων (multimedia session). Ακόμα για να μπορέσουμε να μεταφέρουμε την φωνή από τον ένα συνομιλητή στον άλλο, θα πρέπει να χρησιμοποιηθεί άλλο πρωτόκολλο, αφού το SIP και το SDP χρησιμοποιούνται μόνο για να δημιουργήσουμε συνόδους και όχι για την μεταφορά αυτή καθ'αυτή της φωνής. Για τον λόγο αυτό χρησιμοποιείται το πρωτόκολλο RTP, όπως φαίνεται και στην παραπάνω εικόνα.

Αφού δημιουργήσουμε μία σύνοδο με το SIP και με την βοήθεια του SDP, είμαστε σε θέση να αρχίσουμε να μεταφέρουμε από το ένα άκρο στο άλλο την φωνή μέσω του πρωτοκόλλου RTP. Για να γίνει αυτό θα πρέπει να μετατρέπουμε με κάποιον τρόπο την φωνή των χρηστών, αφού η ανθρώπινη φωνή είναι ένα αναλογικό σήμα. Αυτό μπορεί να γίνει με την βοήθεια της διαδικασίας που είναι γνωστή σαν A2D Convertion, δηλαδή μετατροπή αναλογικού σήματος σε ψηφιακό. Όλες οι απαραίτητες κλάσεις για την μετατροπή αυτή παρέχονται από το λειτουργικό σύστημα. Για να μετατρέψουμε τον ήχο από αναλογικό σε ψηφιακό χρησιμοποιούμε την κλήση AudioTrack και για την αντίστροφη διαδικασία την AudioRecord. Τα δείγματα αυτά μετά τροφοδοτούνται στο επίπεδο του RTP για να μπορέσουν να μεταφερθούν.



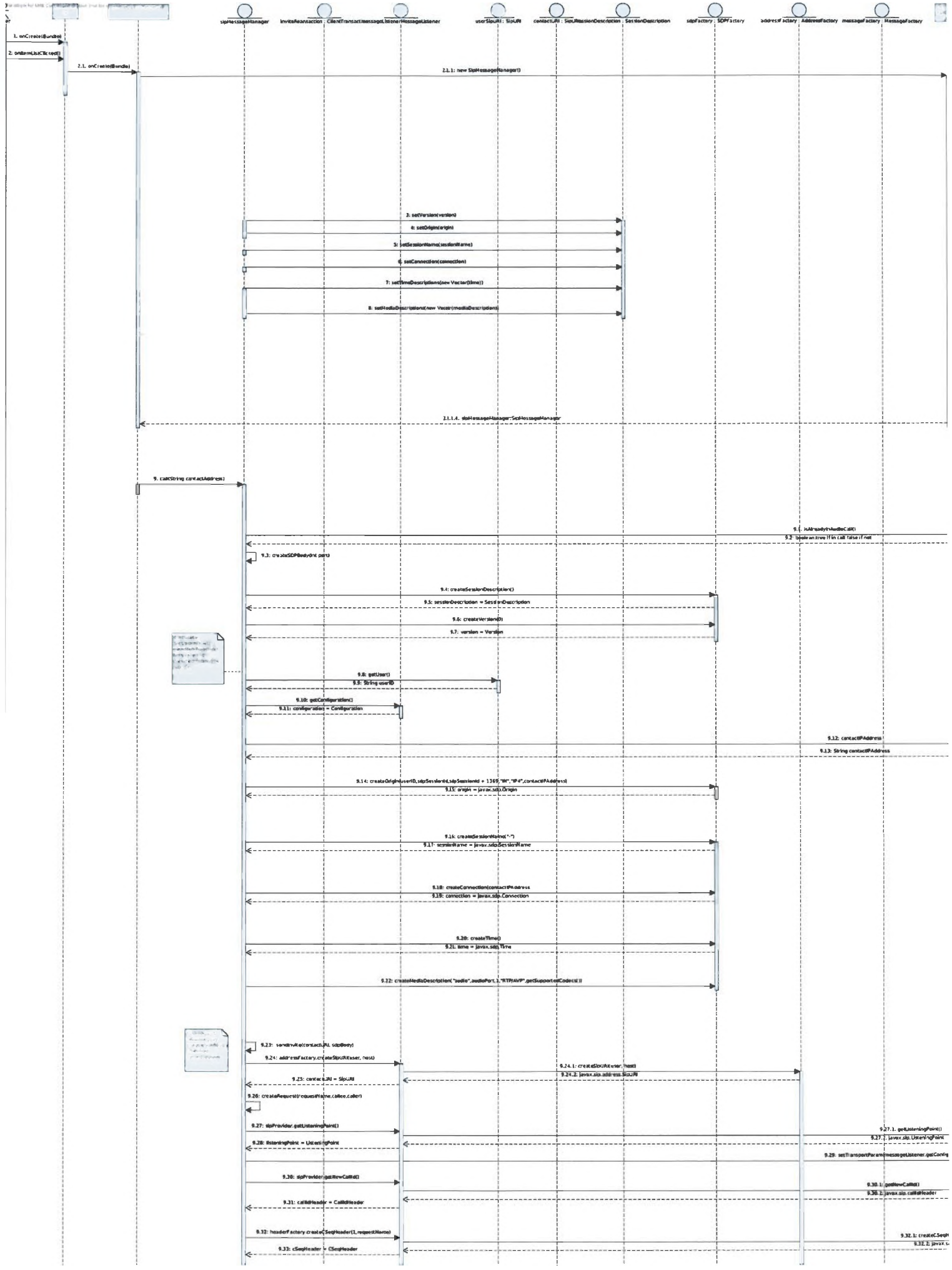
Εικόνα 4.3: Γέμισμα του buffer από το νήμα του RTP για αποστολή και απο το νήμα λήψης για αναπαραγωγή.

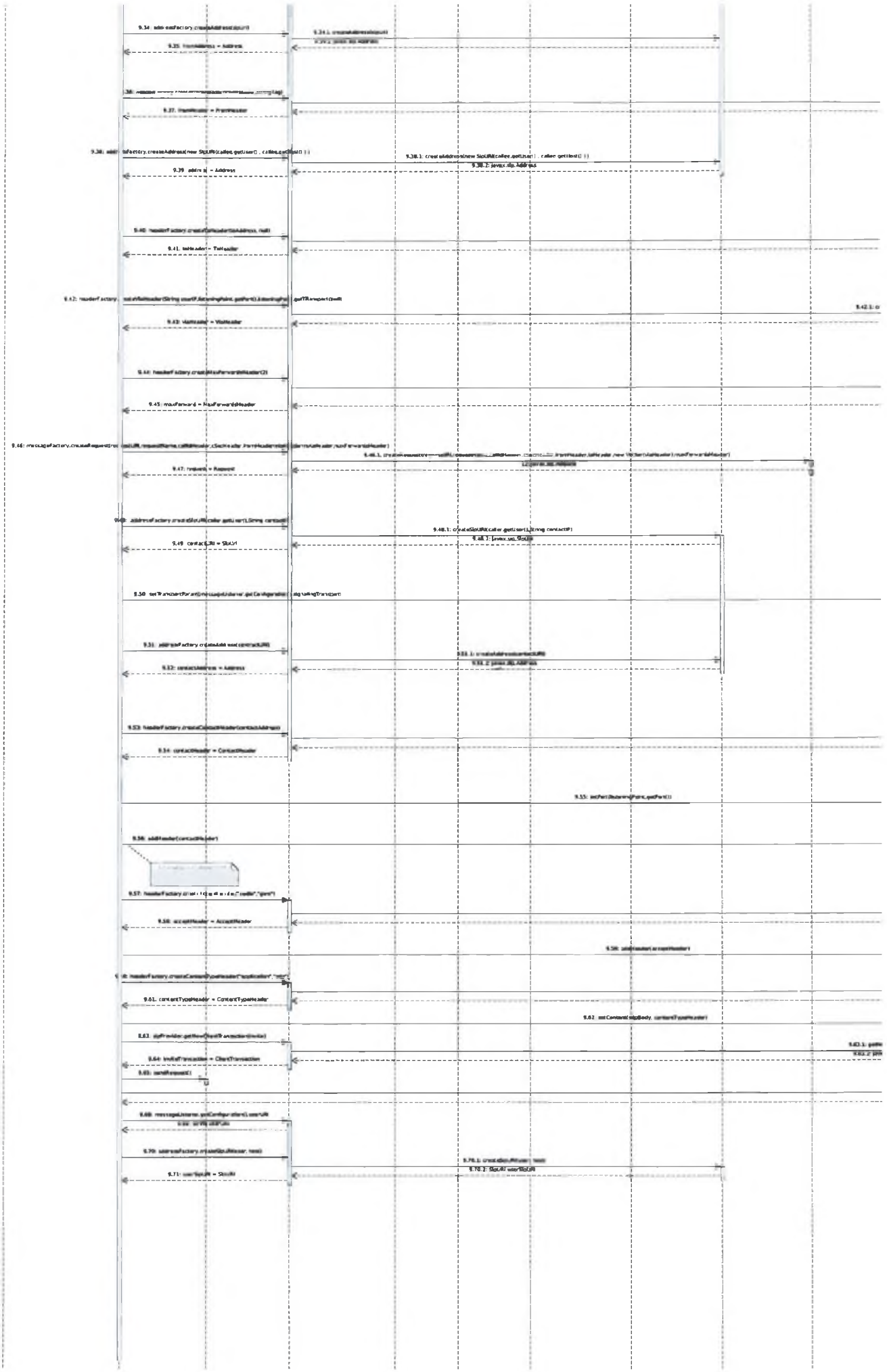
Σε αυτό το σημείο αντιμετωπίσαμε το πρώτο σοβαρό πρόβλημα, το οποίο είχε να κάνει με την μεταφορά των δειγμάτων αυτών μέσω του RTP.
Έτσι τα προβλήματα που αντιμετωπίσαμε ήταν:

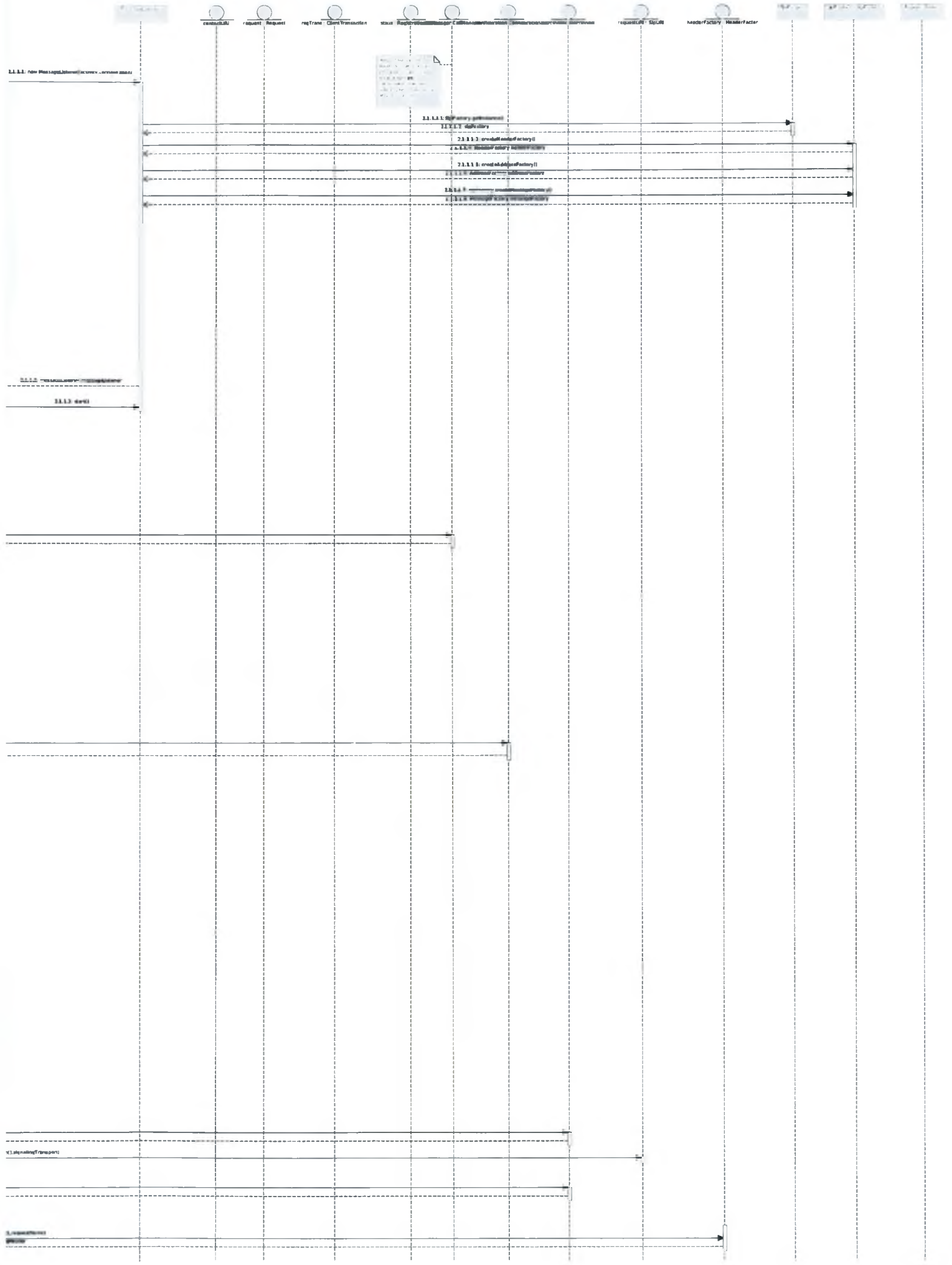
- Έλλειψη μιας υλοποιημένης στοίβας για το πρωτόκολλο αυτό, από το λειτουργικό σύστημα.
- Το μέγεθος των δειγμάτων που παράγονται από την μετατροπή της φωνής είναι 16 bit.

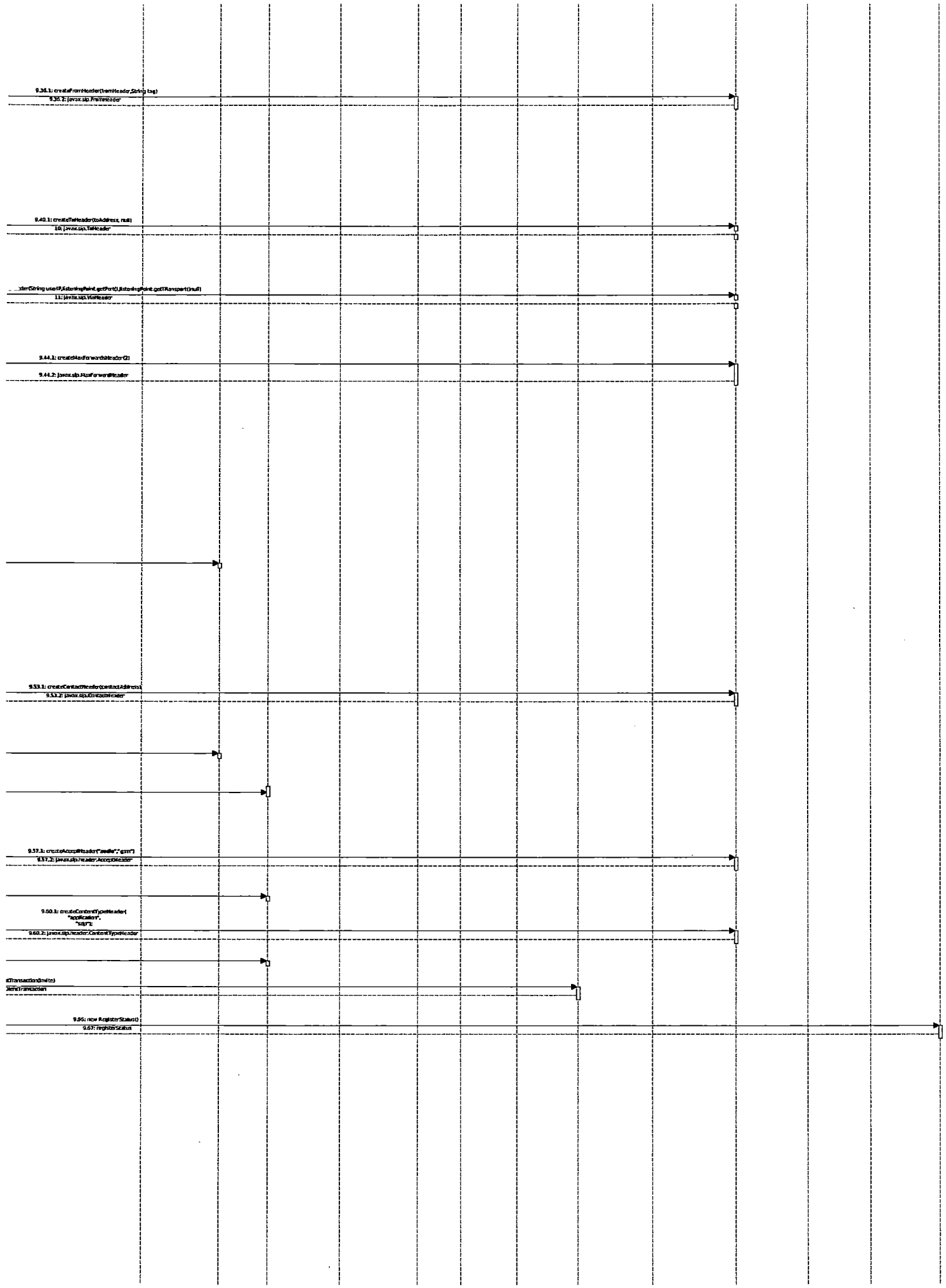
Η λύση στο πρώτο από τα παραπάνω προβλήματα δόθηκε με την χρήση του JlibRTP που προσφέρει μια υλοποίηση για την στοίβα του επιπέδου RTP.

Το δεύτερο πρόβλημα αποδείχθηκε πιο περίπλοκο από ότι μπορεί να φαίνεται. Αρχικά πρέπει να αναφερθεί ότι το JlibRTP, μπορεί να μεταφέρει πακέτα μεγέθους 8 bit και όχι 16 bit. Έτσι η προφανής λύση ήταν να σπάσουμε τα πακέτα και για κάθε πακέτο 16 bit να δημιουργήσουμε 2 των 8. Αυτό όμως δεν πέτυχε λόγω δυσκολιών που εμφανίστηκαν στην υλοποίηση, πιο συγκεκριμένα ο χωρισμός αυτός δεν γινόταν με επιτυχία και τα δεδομένα δεν ήταν αξιόπιστα ώστε να μπορέσουμε να αναπαράγουμε το δειγματοληφθέν ήχο.
Όπως και όταν στέλνουμε άμεσα μηνύματα έτσι και όταν θέλουμε να πραγματοποιήσουμε κλήσεις φωνής, το πρώτο πράγμα που πρέπει να κάνουμε είναι να ακολουθήσουμε την διαδικασία του registration. Έπειτα, με την βοήθεια της μεθόδου createSDPBody του MessageManager, φτιάχνουμε το κατάλληλο μήνυμα SDP για να περιγράψουμε την πόρτα, τις διευθύνσεις και τα codec που θα συμμετάσχουν στην κλήση.
Το επόμενο βήμα είναι να δημιουργήσουμε το Sip μήνυμα invite και να το στείλουμε στον ενδιαφερόμενο χρήστη. Για να γίνει αυτό θα πρέπει να δημιουργηθεί ένα ClientTransaction και να καλέσουμε την μέθοδο του ,sendRequest. Ακόμα από το Transaction αυτό θα πρέπει να πάρουμε και να αποθηκεύσουμε το Dialog ώστε να μπορέσουμε να αναγνωρίσουμε την σύνοδο αυτή και να επεξεργαστούμε τα επόμενα μηνύματα που θα έρθουν.

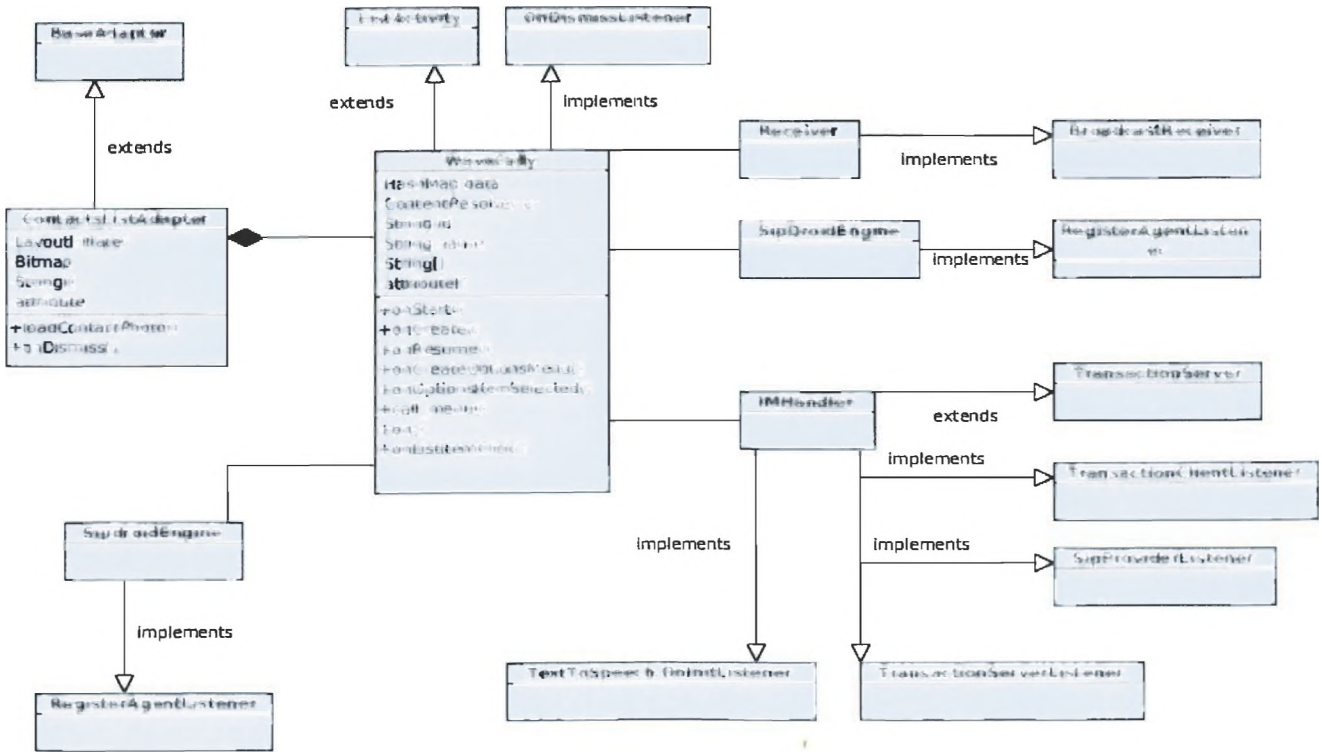

sipMessageManager
InviteRoomAction
ClientTransaction|messageListener:MessageListener
userSipURI : SipURI
contactURI : SipURI
sessionDescription : SessionDescription
sdpFactory : SDPFactory
addressFactory : AddressFactory
messageFactory : MessageFactory
1: onCreate(Bundle)
2: onItemClick()
2.1: onCreate(Bundle)
2.1.1: new SipMessageManager()
3: setVersion(version)
4: setOrigin(origin)
5: setSessionName(sessionName)
6: setConnection(connection)
7: setTimeDescriptions(new Vector(time))
8: setMediaDescriptions(new Vector(mediaDescription))
2.1.1.4: sipMessageManager=SetMessageManager
9: callString contactAddress)
9.1: isAlreadyInAudioCall()
9.2: boolean true if in call false if not
9.3: createSDPBodyInt part)
9.4: createSessionDescription()
9.5: sessionDescription = SessionDescription
9.6: createVersion(0)
9.7: version = Version
9.8: getUser()
9.9: String userID
9.10: getConfiguration()
9.11: configuration = Configuration
9.12: contactIPAddress
9.13: String contactIPAddress
9.14: createOrigin(userID,sdpSessionId,sdpSessionId + 1369,"IN","IP4",contactIPAddress)
9.15: origin = javax.sdp.Origin
9.16: createSessionName("-")
9.17: sessionName = javax.sdp.SessionName
9.18: createConnection(contactIPAddress
9.19: connection = javax.sdp.Connection
9.20: createTime()
9.21: time = javax.sdp.Time
9.22: createMediaDescription("audio",audioPort,1,"RTP/AVP",getSupportedCodecs())
9.23: sendInvite(contactURI, sdpBody)
9.24: addressFactory.createSipURI(user, host)
9.24.1: createSipURI(user, host)
9.24.2: javax.sip.address.SipURI
9.25: contactURI = SipURI
9.26: createRequest(requestName,callee,caller)
9.27: sipProvider.getListeningPoint()
9.27.1: getListeningPoint()
9.27.2: javax.sip.ListeningPoint
9.28: listeningPoint = ListeningPoint
9.29: setTransportParam(messageListener.getConfig
9.30: sipProvider.getNewCallId()
9.30.1: getNewCallId()
9.30.2: javax.sip.CallIdHeader
9.31: callIdHeader = CallIdHeader
9.32: headerFactory.createCSeqHeader(1,requestName)
9.32.1: createCSeqH
9.32.2: javax.s
9.33: cSeqHeader = CSeqHeader



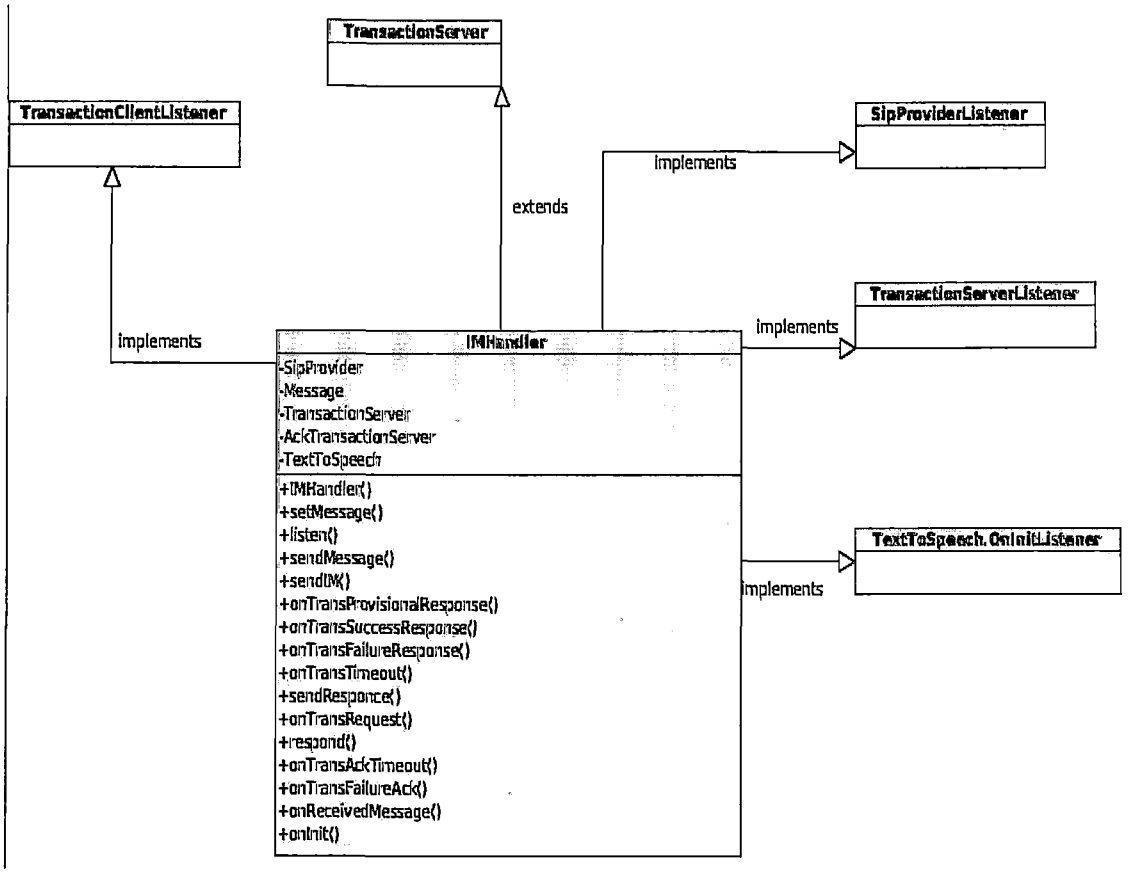



9.36.1: createFromHeader(fromHeader,String tag)
9.36.2: javax.sip.FromHeader
9.40.1: createToHeader(toAddress, null)
10: javax.sip.ToHeader
11: javax.sip.ViaHeader
9.44.1: createMaxForwardsHeader(2)
9.44.2: javax.sip.MaxForwardsHeader
9.53.1: createContactHeader(contactAddress)
9.53.2: javax.sip.ContactHeader
9.57.1: createAcceptHeader("audio","gsm")
9.57.2: javax.sip.header.AcceptHeader
9.60.1: createContentTypeHeader(
"application",
"sdp");
9.60.2: javax.sip.header.ContentTypeHeader
9.65: new RegisterStatus()
9.67: registerStatus

Επειδή δεν καταφέραμε να επιλύσουμε το πρόβλημα αυτό, αποφασίσαμε να χρησιμοποιήσουμε το SipDroid, ένα ανοιχτού κώδικα project, που βασίζεται στην στοίβα MJSIP για το επίπεδο της σηματοδοσίας και προσφέρει δική του υλοποίηση για το επίπεδο RTP.
Το SipDroid πέραν από την πραγματοποίηση των κλήσεων φωνής δεν προσέφερε κάποια από τις υπηρεσίες που προαναφέραμε, αφού έχει σχεδιαστεί για να προσφέρει καθαρά υπηρεσίες VoIP μέσω τρίτων παρόχων. Έτσι έπρεπε να σχεδιάσουμε τόσο το κομμάτι του που είναι υπεύθυνο για την υπηρεσία άμεσου μηνύματος όσο και την αρχική οθόνη με την οποία επιλέγει ο χρήστης τους συνομιλητές του. Ο σχεδιασμός του SipDroid δεν διέφερε σε μεγάλο ποσοστό από το υπάρχον πρόγραμμα και έτσι ήταν αρκετά εύκολο να το τροποποιήσουμε. Πιο αναλυτικά το SipDroid, έχει μία βασική κλάση η οποία επεκτείνει το BroadcastReceiver. Αυτό είναι ένα συστατικό (Component) του λειτουργικού συστήματος, το οποίο ενημερώνεται από το λειτουργικό κάθε φορά που ένα συγκεκριμένο μήνυμα στέλνεται από κάποια άλλη υπηρεσία. Επιπλέον αυτό μας παρέχει στατικές μεθόδους με τις οποίες μπορούμε να έχουμε πρόσβαση τόσο στο υποκείμενο επίπεδο του Sip, αλλά και στις υπηρεσίες του που έχουμε για κάθε κλήση. Γνωρίζοντας τα παραπάνω μπορούμε να σκεφτούμε την κλάση αυτή σαν ένα κεντρικό σημείο ελέγχου με το οποίο ελέγχουμε όλο το πρόγραμμα μας. Ακόμα υπάρχει η κλάση Call η οποία είναι υπεύθυνη για να διατηρεί



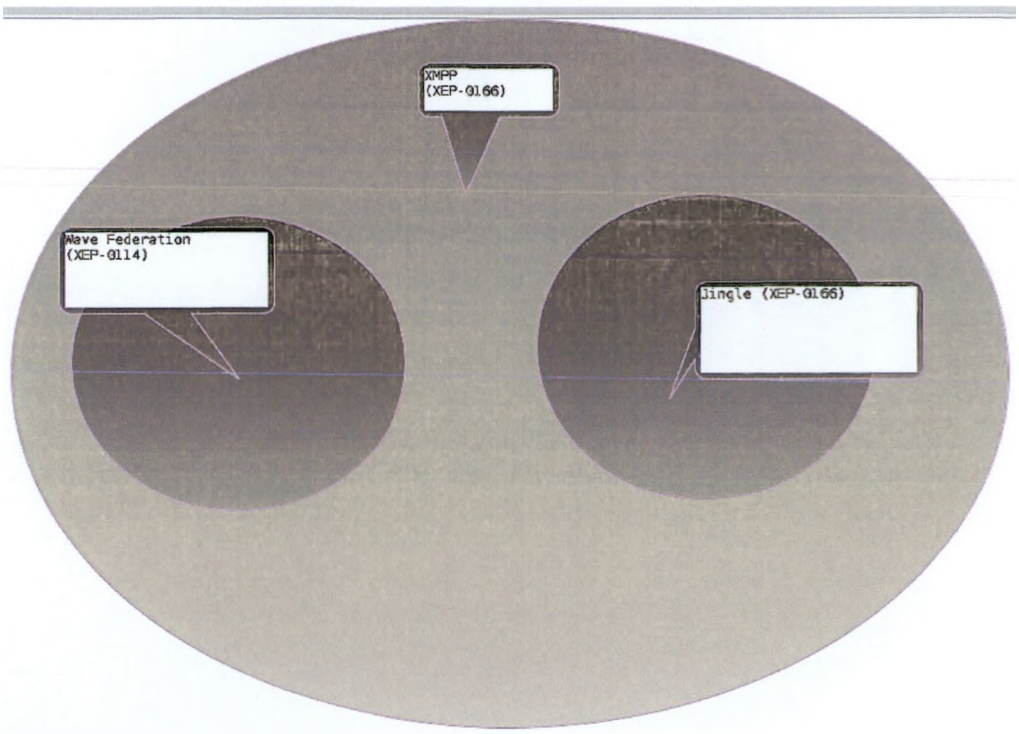
πληροφορίες σχετικές με κάθε κλήση που πραγματοποιούμε αλλά και να αλλάζει την διαθεσιμότητα μας. Έτσι μπορούμε να παρατηρήσουμε ότι η λογική του SipDroid είναι αρκετά κοντά με τον αρχικό μας σχεδιασμό, και για αυτόν τον λόγο αυτό που τελικά χρειάστηκε να κάνουμε είναι να προσαρμόσουμε την ChatUI ώστε να λειτουργεί βάσει του MJSIP. Αυτό μας οδήγησε στην δημιουργία του IMHandler, που είναι υπεύθυνο για την αποστολή-λήψη μηνυμάτων καθώς και για να ενημερώνει το γραφικό ενδιάμεσο. Το τελικό διάγραμμα κλάσεων φαίνεται στο παρακάτω σχήμα.


TransactionServer
TransactionClientListener
SipProviderListener
implements
extends
TransactionServerListener
implements
implements
IMHandler
-SipProvider
-Message
-TransactionServer
-AckTransactionServer
-TextToSpeech
+IMHandler()
+setMessage()
+listen()
+sendMessage()
+sendIM()
+onTransProvisionalResponse()
+onTransSuccessResponse()
+onTransFailureResponse()
+onTransTimeout()
+sendResponce()
+onTransRequest()
+respond()
+onTransAckTimeout()
+onTransFailureAck()
+onReceivedMessage()
+onInit()
TextToSpeech.OnInitListener
implements

4.2 Ο εξυπηρετητής Mobicents.

Το πιο σημαντικό κομμάτι της υλοποίησης μας, είναι το κομμάτι του εξυπηρετητή. Το κομμάτι αυτό είναι που αναλαμβάνει να γεφυρώσει το federation protocol με το SIP, καθώς επίσης και να προσφέρει την επιχειρησιακή λογική του προγράμματος. Κατά τον σχεδιασμό του προγράμματος θέλαμε να απελευθερώσουμε όσο το δυνατόν περισσότερες από τις δυνατότητες που προσφέρονται από το περιβάλλον του εξυπηρετητή. Έτσι προσπαθήσαμε να εκμεταλλευτούμε στο έπακρο την δυνατότητα που προσφέρεται για παράλληλη επεξεργασία, με σκοπό η διαδικασία της σηματοδοσίας και της μετατροπής από το ένα πρωτόκολλο στο άλλο, να γίνει όσο το δυνατόν γρηγορότερα.

Το τελικό πρόγραμμα δεν περιλαμβάνει το κομμάτι της μετατροπής από το SIP στο Jingle, γιατί δεν υπήρχε τρόπος να χρησιμοποιηθεί το πρωτόκολλο αυτό, σε συνεργασία με το JSLEE. Αυτό οφείλεται στην έλλειψη Resource Adaptor που να υλοποιεί τις απαραίτητες διαδικασίες ώστε να στέλνει events στην υπηρεσία που θα αναλάμβανε την μετατροπή. Για να μπορέσουμε να αναπτύξουμε ένα δικό μας Resource Adaptor και να προσφέρουμε τέτοιου είδους υπηρεσίες αποδείχθηκε πολύ πιο δύσκολο από ότι αρχικά φαίνεται, αφού κάτι τέτοιο, όχι μόνο απαιτεί καλή γνώση του τρόπου με τον οποίο αναπτύσσουμε ένα Resource Adaptor, αλλά και καλή γνώση των Jingle, αλλά και Wave federation protocol. Στο παρακάτω σχήμα παρουσιάζεται το πώς αυτά τα πρωτόκολλα αλληλεπιδρούν.



Εικόνα4 1: Wave-XMPP-Jingle relation

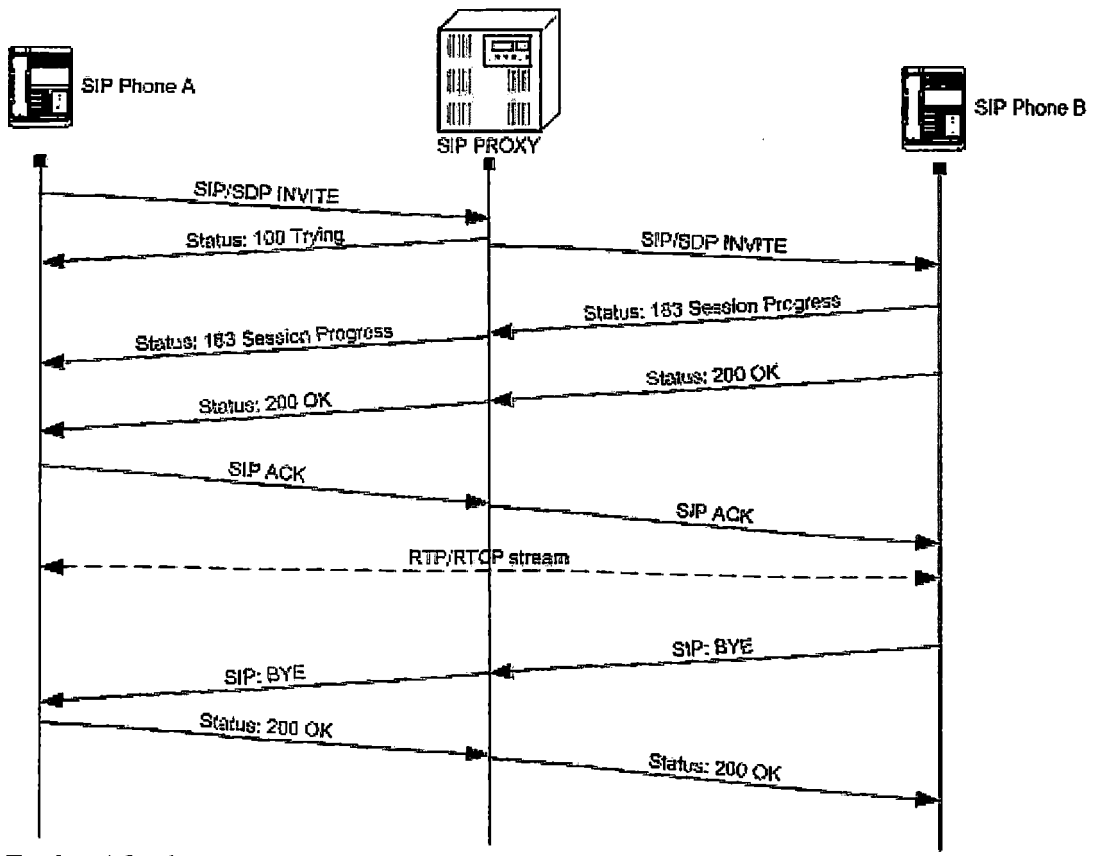
Είναι πολύ σημαντικό να παρατηρήσουμε ότι το wave protocol, δεν είναι τίποτε άλλο από μια επέκταση (extension) πάνω στο αρχικό XMPP, και βασίζεται στο XEP-0166 που είναι και το βασικό πρωτόκολλο για το XMPP. Το Jingle, είναι και αυτό με την σειρά του επέκταση πάνω στο XMPP, γι'αυτό τον λόγο τα παρουσιάζουμε σαν ομόκεντρους κύκλους.

Από την στιγμή που θα μπορούσαμε να φτιάξουμε έναν Resource Adaptor ο οποίος να μπορεί να

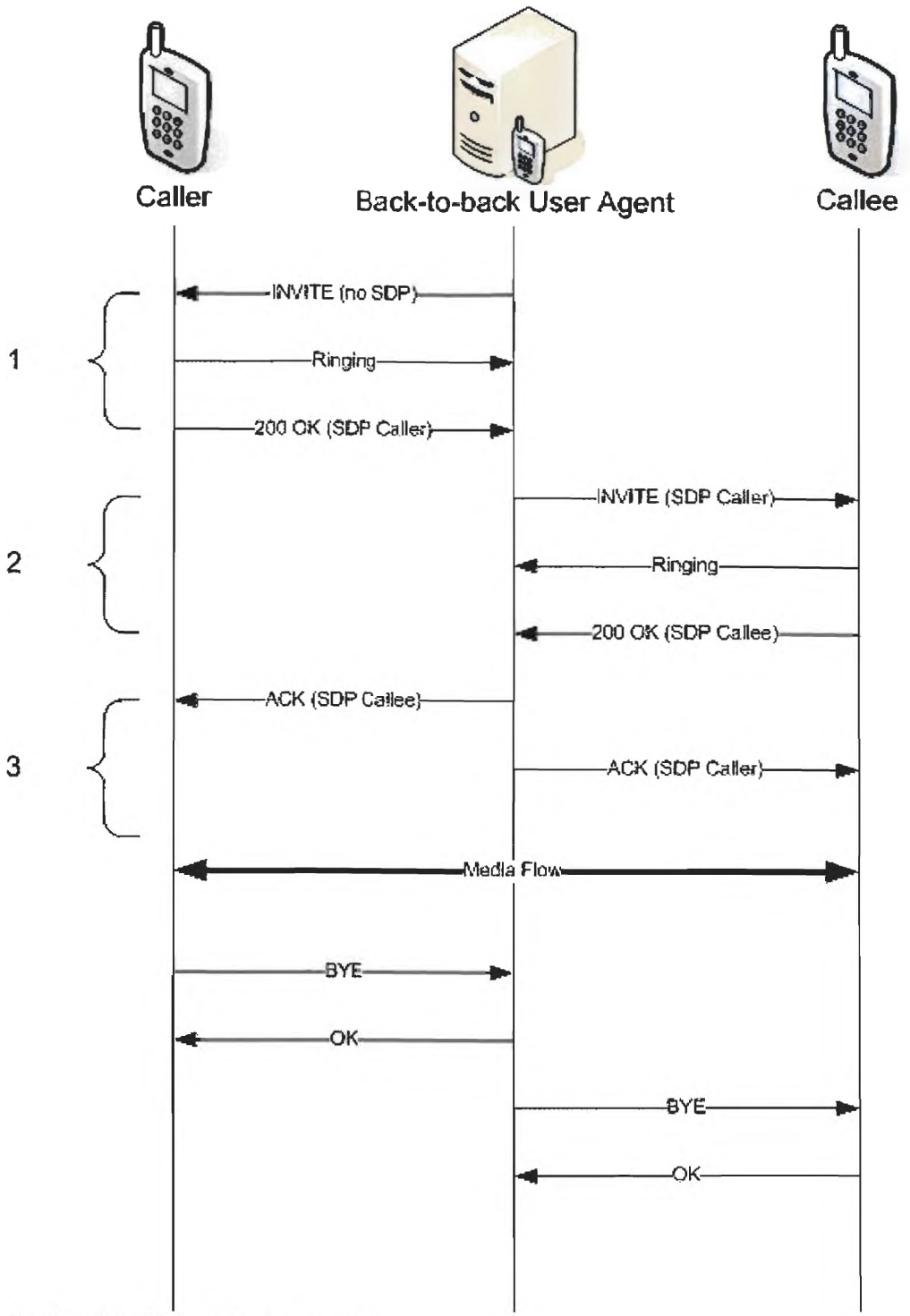
επικοινωνεί με το XMPP, θα μπορούσαμε να τον επεκτείνουμε ώστε να ανταπεξέρχεται στης ανάγκες του federation protocol, καθώς επίσης και του Jingle. Αυτά όμως είναι στην θεωρία, γιατί στην πράξη, δεν υπάρχει κάποιο έτοιμο stack στο οποίο θα μπορούσαμε να βασιστούμε όπως συμβαίνει με το Sip. Το μόνο που υπάρχει είναι το Smack API το οποίο όμως δεν προσφέρεται για δημιουργία προγραμμάτων για εξυπηρετητές αφού παρέχει μόνο την απαραίτητη λειτουργικότητα για εφαρμογές πελάτη. Καθώς και το Tinder API το οποίο όμως προσφέρει μια βάση πάνω στην οποία μπορούμε να βασιστούμε ώστε να δημιουργήσουμε την δική μας υλοποίηση. Έτσι θα έπρεπε να δημιουργήσουμε δική μας υλοποίηση για το XMPP.
Επίσης θα μπορούσαμε να χρησιμοποιήσουμε τον OpenFire, ο οποίος είναι ένας εξυπηρετητής που προσφέρει υλοποίηση για το XMPP, καθώς επίσης μπορεί και να χρησιμοποιηθεί τόσο με το Jingle όσο και με το federation protocol μέσο του Wave In A Box. Κάτι τέτοιο όμως δεν θα είχε διαφορά αφού πάλι λόγω της έλλειψης ενός Resource Adaptor ικανού να ενεργοποιεί τις υπηρεσίες του Slee, δεν μας βόλευε παρά μόνο για την ανταλλαγή άμεσων μηνυμάτων.
Έτσι από τα παραπάνω μπορούμε εύκολα να παρατηρήσουμε ότι για κάτι που αρχικά μοιάζει απλό, αποδεικνύεται ότι τελικά δεν είναι.
Αφού πέραν του χαμηλού επιπέδου κώδικα που πρέπει να γράψουμε ώστε να μπορούμε να χρησιμοποιήσουμε το XMPP, θα πρέπει να υλοποιεί και τα specification που υπάρχουν για το federation protocol ή το jingle.
Ένα ακόμα εμπόδιο στην προσέγγιση αυτή, ήταν και οι κακές προδιαγραφές που προσφέρονταν για το Federation protocol, αφού την περίοδο που σχεδιάζαμε το σύστημα, η Google άρχιζε να ανοίγει των κώδικα του πρωτοκόλλου.
Επίσης ο κώδικας που ήταν διαθέσιμος χρησιμοποιούσε το Guice, ένα Injection Dependency Framework, που έχει αναπτυχθεί από την Google. Έτσι πριν αρχίσουμε να κοιτάμε όσα κομμάτια ήταν διαθέσιμα ώστε να μπορέσουμε να κατανοήσουμε των κώδικα, θα έπρεπε να εξοικειωθούμε με αυτό, που παρότι δεν είναι δύσκολο ούτε περίπλοκο στην χρήση του, σε τόσο μεγάλα project, αποδείχθηκε πολύ δυσκολότερο από ότι αρχικά πιστεύαμε. Για να αντιληφθούμε την δυσκολία, παραθέτουμε μία εικόνα η οποία έχει παραχθεί με το Grapher. Ένα εργαλείο το οποίο επιτρέπει την ανάλυση σε project βασισμένα στο Guice.

WaveClientRegistry
ClientFrontend
ClientFrontendImpl
WaveletProvider
WaveServer
WaveServerImpl
CertificateManager
LocalWaveletContainerFactory
RemoteWaveletContainerFactory
XmppManager
CertificateManagerImpl
OutgoingPacketTransport
XmppFederationHost
XmppFederationRemote
XmppDisco
WaveSignatureVerifier
WaveStore
Injector
ComponentPacketTransport
WaveCertPathValidator
CertPathStore
WaveSignatureProvider
IncomingPacketHandler
DefaultCertPathStore
WaveSignatureFactory
VerifiedCertChainCache
TrustRootsProvider
DefaultCacheImpl
DefaultTrustRootsProvider
TimeService
DefaultTimeSource

Λόγω των παραπάνω, τελικώς η λύση που περιγράψαμε στο κεφάλαιο 3.3 δεν είναι εφικτή, αφού δεν θα μπορούσαμε να περάσουμε τα μηνύματα τύπου Jingle απο και προς τον Mobicents.
Τελικά αποφασίσαμε να φτιάξουμε μία υλοποίηση που βασίζεται αποκλειστικά στο Sip. Αυτό για λόγους απλότητας στην υλοποίηση. Επίσης όντας σε θέση να
προσφέρουμε αυτήν την υλοποίηση, μπορούμε μελλοντικά να την επεκτείνουμε απλά μετατρέποντας τα μηνύματα και μεταφράζοντας το ένα πρωτόκολλο στο άλλο (Sip-XMPP), με τον τρόπο που αναφέραμε στο κεφάλαιο 3.
Ο εξυπηρετητής σχεδιάστηκε να λειτουργεί σαν Proxy, προσφέροντας ταυτόχρονα και υπηρεσίες Registration στους χρήστες. Αυτός ο σχεδιασμός προτιμήθηκε έναντι άλλων όπως π.χ. Του Back to back user agent, καθώς το μόνο που θέλαμε είναι ο εξυπηρετητής μας να παρεμβαίνει όσο το δυνατόν λιγότερο ανάμεσα στους χρήστες, με στόχο την γρηγορότερη εξυπηρέτηση των μηνυμάτων (εξάλλου η αρχική ιδέα του Federation protocol είναι ο σχεδόν μηδενικός χρόνος διάδοσης τον μηνυμάτων, κάτι που θέλαμε να κρατήσουμε ώστε να μπορούμε να επεκτείνουμε το σύστημα μελλοντικά).Αλλά επιτρέποντας ταυτόχρονα στον εξυπηρετητή να επεξεργαστεί την κίνηση των μηνυμάτων με σκοπό να μετατρέψει ,όπου αυτό απαιτείται, την κίνηση μεταφράζοντας τα μηνύματα από το ένα πρωτόκολλο στο άλλο. Επίσης στο σχεδιασμό B2BUA, η επικοινωνία γίνεται σε επίπεδο Dialog όσον αφορά το Sip, κάτι που σημαίνει ότι θα έπρεπε να κρατάμε καθ'όλη την διάρκεια τις πληροφορίες για αυτό. Ακόμα η αρχιτεκτονική αυτή προτιμήθηκε αφού επιτρέπει να υλοποιήσουμε την επιχειρησιακή λογική που είναι αναγκαία με όσο το δυνατόν λιγότερα μηνύματα. Αυτό μπορεί να φανεί και από τα σχήματα που ακολουθούν όπου φαίνεται η λογική πίσω από τις αρχιτεκτονικές σε επίπεδο σηματοδοσίας (ανταλλαγής μηνυμάτων).



Εικόνα4 2 : back to back user agent

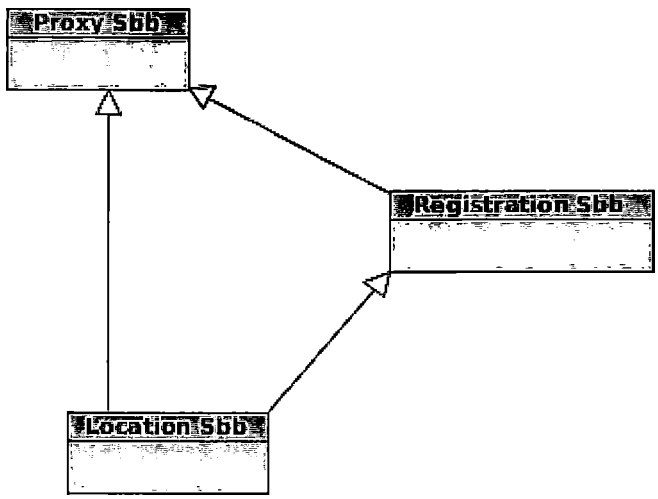


Εικόνα4 3: Σχήμα Proxy server

Για να εκμεταλλευτούμε όλες τις δυνατότητες του περιβάλλοντος ανάπτυξης, σχεδιάσαμε την υπηρεσία μας (Service)ώστε να αποτελείται από τρία διαφορετικά sbbs. Κάθε ένα από τα διαφορετικά αυτά sbb καλύπτει διαφορετικές πτυχές της υπηρεσίας.

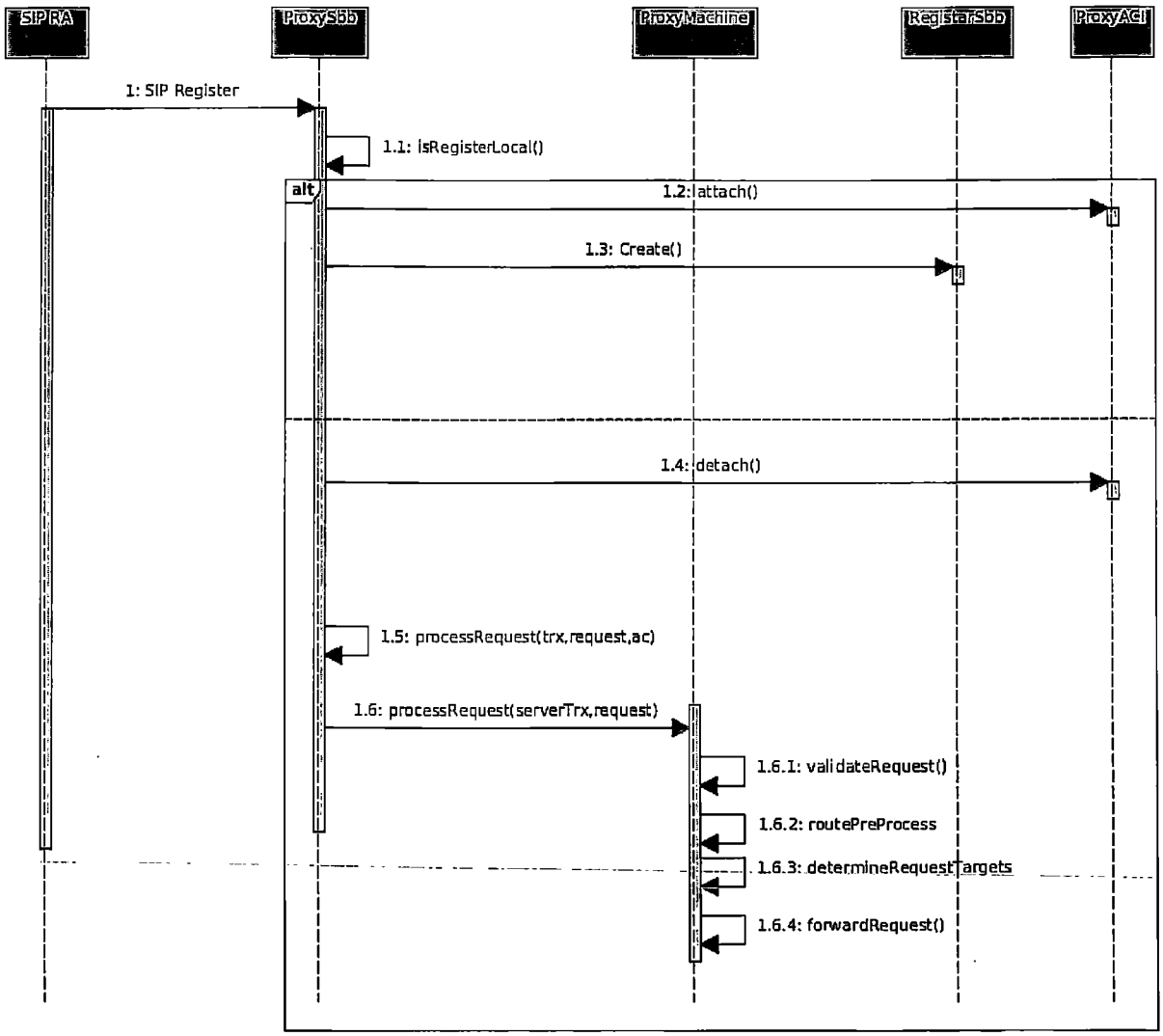
Έτσι επιτρέπεται να χωρίσουμε την επιχειρησιακή λογική και να προσφέρουμε καλύτερη απόδοση εκμεταλλευόμενοι τις δυνατότητες του Jslee, καθώς επίσης και μεγαλύτερο έλεγχο στην λειτουργία των επιμέρους υπηρεσιών επιτρέποντας μελλοντικές μεταβολές, αν αυτό κριθεί αναγκαίο.
Για να μπορέσουμε να προσφέρουμε καλύτερο έλεγχο στον τρόπο λειτουργίας των Proxy και Registrar sbb's, τα σχεδιάσαμε με τέτοιο τρόπο ώστε να μπορούν να ελέγχονται μέσα από τον μηχανισμό JMX που προσφέρει η java παρέχοντας αντίστοιχα δύο Mbeans, ένα για κάθε ένα Sbb.
Έτσι για το Registar Sbb έχουμε το RegistarConfiguratorMBean που επιτρέπει να ελέγχουμε παραμέτρους που έχουν να κάνουν με τον χρόνο που κρατάμε πληροφορίες για κάθε εγγεγραμμένο χρήστη. Και για το Proxy Sbb προσφέρουμε το ProxyConfiguratorMBean, που προσφέρει έλεγχο πάνω στο Sip, όπως είναι το σχήμα Uri, και το port number στο οποίο είναι συνδεδεμένη η υποκείμενη στοίβα για το Sip και άλλα.

Ο τρόπος και ο σχεδιασμός λειτουργίας της υπηρεσίας, έχει βασιστεί πάνω σε ένα από τα πολλά παραδείγματα τα οποία προσφέρονται από το Mobicents. Η υπηρεσία μας, έχει ένα sbb πατέρα, το οποίο ονομάζεται Proxy, και είναι αυτό που καλείται πρώτο για κάθε κλήση. Αυτό ορίζει δύο παιδιά το ένα, είναι το Registration sbb και το άλλο είναι το Location sbb. Το Registar sbb ορίζει σαν παιδί του το Location sbb, το οποίο είναι και το τελικό sbb τις υπηρεσίας μας.



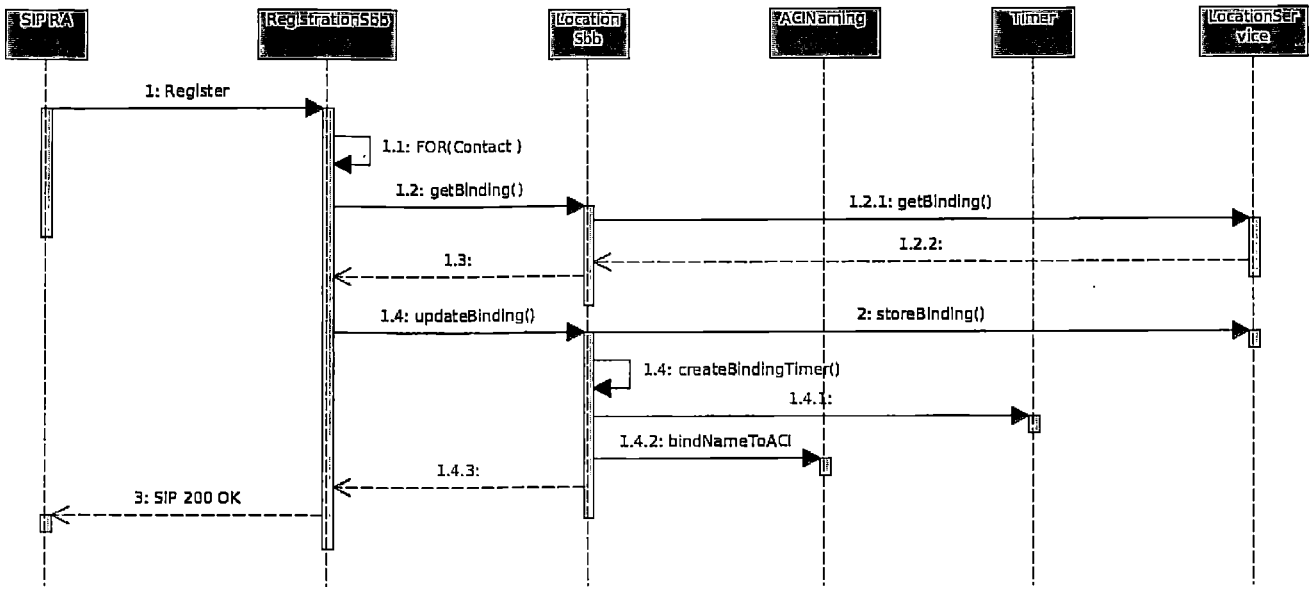
Εικόνα 4 4 : Ιεραρχία των Sbb του συστήματος μας.

Το Proxy sbb είναι υπεύθυνο για να δρομολογεί τα μηνύματα βάσει του πεδίου του παραλήπτη, του μηνύματος Sip, αλλά και του είδους του μηνύματος.
Όταν ένα Registration μήνυμα σταλεί στον εξυπηρετητή, τότε το Proxy sbb καλεί το Registar sbb, βάζοντας το στον γράφο εκτέλεσης, μέσα από την σχέση που έχουμε δηλώσει στο sbb-jar.xml αρχείο. Η συνολική διαδικασία φαίνεται στο διάγραμμα UML



Εικόνα4 5: Proxy server Sequence Diagram

Το Registar sbb είναι υπεύθυνο για να προσφέρει υπηρεσίες Registrar Server, Όπως φαίνεται στο παραπάνω σχήμα, το Registar sbb, είναι πατέρας του Location sbb. Έτσι όταν το sbb αυτό ενεργοποιείται, ενεργοποιεί ταυτόχρονα το αντίστοιχο location sbb χωρίς όμως να του περάσει τον έλεγχο. Όταν καλέσουμε το Registrar sbb, τότε αρχικά πρέπει να το αφαιρέσουμε από το Activity Context του Jslee. Αυτό πρέπει να γίνει ώστε να αποτρέψουμε το sbb αυτό να επεξεργαστεί άλλα μηνύματα κατά την διάρκεια της κλήσης. Έπειτα καλούμε την μέθοδο addBinding του Location sbb.
Έτσι το Location sbb συνδέεται με την βάση δεδομένων όπου αποθηκεύουμε τα στοιχεία κάθε εγγεγραμμένου χρήστη (AddressOfRecord-AOR). Η πρόσβαση στην βάση γίνεται χρησιμοποιώντας το Java Persistance API (JPA) για λόγους επεκτασιμότητας, αφού μπορεί να χρησιμοποιηθεί σε συνδυασμό με πολλές βάσεις δεδομένων, επηρεάζοντας με αυτόν τον τρόπο την συνολική απόδοση του συστήματος αφού η κάθε βάση δεδομένων προσφέρει διαφορετικές δυνατότητες.
Ακόμα το Location sbb αναλαμβάνει να αρχικοποιήσει το Timer Facility του εξυπηρετητή ώστε να μπορεί το Jslee να μας ειδοποιεί για το πότε κάθε registration μήνυμα λήγει και να είμαστε σε θέση να αφαιρέσουμε από την βάση την εγγραφή αυτή. Για τον ίδιο λόγο δημιουργούμε και ένα Activity Context το οποίο συνδέουμε στο Activity αυτό.



Εικόνα4 6: Registar Server Sequence Diagram

Σε κάθε άλλη περίπτωση, ο εξυπηρετητής λειτουργεί σύμφωνα με το RFC 3261 που στην παράγραφο 16 ορίζει την συμπεριφορά που πρέπει να έχει ένας τέτοιος εξυπηρετητής. Έτσι δέχεται μηνύματα και ανάλογα το πεδίο του αποστολέα το οποίο βρίσκει μέσα από το μήνυμα που στέλνεται στον εξυπηρετητή αναλαμβάνει να δρομολογήσει τα μηνύματα του ενός χρήστη προς τον άλλον. Πρέπει να τονίσουμε ότι ο σχεδιασμός του συστήματος μας ορίζει ότι κάθε οντότητα που επιθυμεί να χρησιμοποιήσει τις υπηρεσίες του εξυπηρετητή, θα πρέπει πρώτα να έχει εγγραφεί. Σε αντίθετη περίπτωση, ένα μήνυμα λάθους θα επιστρέφεται και η σύνδεση θα τερματίζεται. Αυτό γιατί όλα τα μηνύματα που δεν είναι τύπου INVITE, θεωρούμε ότι έχουν να κάνουν με εγγεγραμμένους χρήστες.
Ο λόγος που δόθηκε αυτή η ιεραρχία στον γράφο των sbb, είναι γιατί θέλαμε να εκμεταλλευτούμε το ιεραρχικό μοντέλο του Jslee, που επιτρέπει γρήγορες συναλλαγές, χρησιμοποιώντας ασύγχρονα γεγονότα (events) ώστε να μπορέσουμε να πετύχουμε όσο το δυνατόν μεγαλύτερη επεκτασιμότητα και ευελιξία στο σύστημα μας.
Έτσι μπορούμε να ξεχωρίσουμε σε ένα service το οποίο έχουμε εγκαταστήσει στον εξυπηρετητή όλα τα συμβαλλόμενα μέρη από τα οποία αποτελείται. Αυτό μπορεί να βοηθήσει πολύ στον τρόπο που λειτουργεί το service, αφού ανά πάσα στιγμή μπορούμε να αλλάξουμε ή να τροποποιήσουμε την λογική οποιουδήποτε από αυτά.
Επίσης επιτρέπει στο slee να εκμεταλλευτεί πλήρως τις δυνατότητες που του προσφέρουμε από μεριάς εγκατάστασης. Αυτό σημαίνει ότι το διευκολύνουμε ώστε να μπορέσει να μοιράσει σωστά τους πόρους του συστήματος ώστε το σύστημα να μπορεί να ανταποκριθεί ακόμα και κάτω από υψηλό φόρτο.
Ακόμα λόγω των ιδιαιτεροτήτων των δικτύων αυτών που επιτάσσουν υψηλή απόδοση σε συνδυασμό με υψηλή αξιοπιστία, χρησιμοποιήσαμε στην ουσία μία βασική υπηρεσία, αυτή του proxy, και προσπαθήσαμε να την επιβαρύνουμε μόνο όταν αυτό ήταν αναγκαίο.(όταν ο χρήστης ξεγραφόταν από την υπηρεσία, ή όταν έληγε ο χρόνος τις εγγραφής του.) Έτσι με αυτόν τον τρόπο, θα μπορούσαμε να παρέχουμε στα πλαίσια μιας κλήσης, την απαιτούμενη λογική για να μετατρέψουμε, το ένα πρωτόκολλο με το άλλο, όντας σε θέση να ανταποκριθούμε κάτω από υψηλό φόρτο, κρατώντας τους χρόνους διάδοσης των μηνυμάτων σε σχεδόν άμεσους χρόνους, κάτι δηλαδή που θα διευκόλυνε το σύστημα μας άμα τα χρησιμοποιούσαμε σε συνδυασμό με το Wave federation protocol.

Ένα ακόμα σημαντικό χαρακτηριστικό που μας προσφέρει ο σχεδιασμός αυτός και έχει να κάνει με την διάκριση των υπηρεσιών, είναι ότι θα μπορούσαμε να προσθέτουμε Sbbs μέσα σε αυτήν την υπηρεσία, ώστε να την επεκτείνουμε όσο χρειάζεται. Ένα παράδειγμα θα ήταν να προσθέσουμε ένα Sbb το οποίο να προσφέρει την υπηρεσία της μετατροπής των μηνυμάτων. Με αυτόν τον τρόπο, θα μπορούσαμε από το ήδη υπάρχον Proxy sbb, να ειδοποιούμε το sbb αυτό μόνο όταν ο παραλήπτης του μηνύματος ήταν χρήστης Jingle, και έτσι θα επιτρέπαμε στον εξυπηρετητή να λειτουργεί τόσο σαν Gateway προς το Jingle, όσο και σαν παραδοσιακός Proxy, ανάλογα με το είδος του παραλήπτη.

Κεφάλαιο 5 Μέτρηση απόδοσης συστήματος/Συμπεράσματα.

5.1 Μέτρηση της επίδοσης της εφαρμογής του .Islee.

Για να μετρήσουμε την απόδοση του εξυπηρετητή μας, χρησιμοποιήσαμε ένα εργαλείο το οποίο έχει αναπτυχθεί από την εταιρία Hewlet packard με σκοπό να εξομοιώνει κίνηση VoIP, βάσει σεναρίων τα οποία που δίνουμε σαν είσοδο. Ακόμα καθ'όλη την διάρκεια των εξομοιώσεων καταγράφαμε την κίνηση με το Wireshark, ώστε να μπορούμε να επαληθεύουμε αλλά και να αναπαραστήσουμε γραφικά τα στοιχεία της κίνησης.
Στην περίπτωση μας, θέλαμε να μετρήσουμε την απόδοση του συστήματος όχι μόνο από την μεριά του Proxy, αλλά και του Registar server.
Έτσι σχεδιάσαμε δύο διαφορετικά test τα οποία τρέξαμε για να μετρήσουμε την απόδοση του εξυπηρετητή. Επειδή η μέτρηση της απόδοσης ενός συστήματος είναι κάτι το οποίο ποικίλει και έγκειται καθαρά στον τρόπο με τον οποίο γίνονται οι μετρήσεις, αλλά και τον στόχο τον οποίο έχουν, πρέπει να τονίσουμε ότι ο σκοπός των εξομοιώσεων αυτόν ήταν να δούμε την απόδοση του εξυπηρετητή από την μεριά των πόσων κλήσεων μπορεί να εξυπηρετήσει στην μονάδα του χρόνο, και όχι το πόσο καλά μπορεί να ανταποκριθεί σε περίπτωση κάποιας βλάβης(όπως παραδείγματος χάριν την βλάβη κάποιου κόμβου στο δίκτυο).

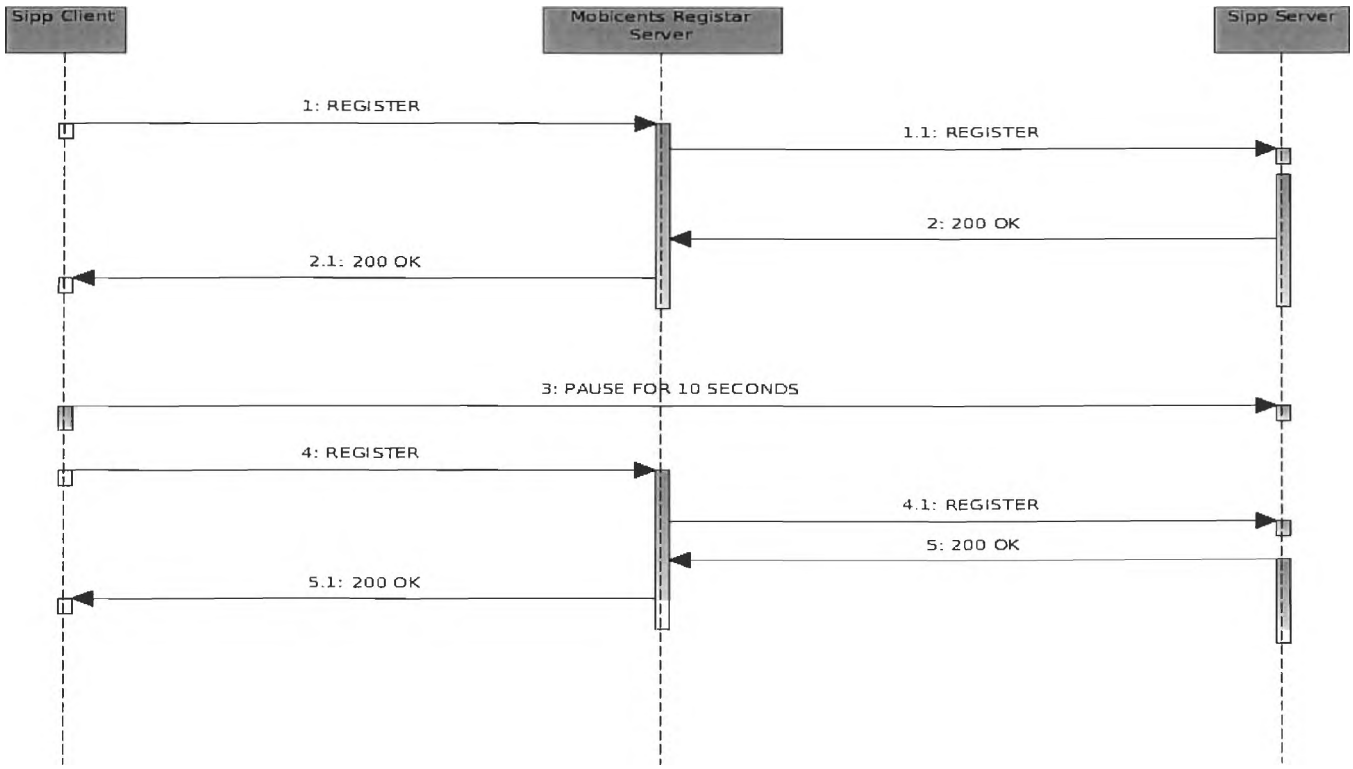
Το περιβάλλον στο οποίο έγιναν οι εξομοιώσεις είναι,

Λειτουργικό σύστημα:	**Ubuntu Linux 11.04 i586**
Έκδοση Java:	Java HotSpot(TM) Client VM (build 19.1-b02, mixed mode, sharing) έκδοση1.6.0_24
Έκδοση Mobicents:	2.3.1 GA Jboss Final
Επεξεραστής:	Intel pentium 4 Extreme edition
Μνήμη RAM:	2 Gigabyte DDR
Βάση δεδομένων:	HSQL

Το σενάριο της εξομοίωσης του Registar περιλαμβάνει δύο μηνύματα: Ένα για εγγραφή του χρήστη, και ένα που τον διαγράφει.
Μεταξύ των δύο μηνυμάτων υπάρχει μία παύση της τάξης των δέκα δευτερολέπτων. Έτσι μετά το πρώτο Register περιμένουμε ένα μήνυμα επιβεβαίωσης (200 OK), δίνοντας χρόνο πέντε δευτερολέπτων στον εξυπηρετητή για να απάντησει. Αυτό γίνεται ώστε να αποφύγουμε πιθανές ανεπιθύμητες αναμεταδόσεις λόγω αργοπορημένης παράδοσης της απάντησης. Σε αντίθετη περίπτωση ξαναστέλνουμε το μήνυμα.
Άμα η παραπάνω διαδικασία ολοκληρωθεί χωρίς πρόβλημα, τότε στέλνουν ένα μήνυμα για να ξεγραφτούμε από τον εξυπηρετητή.
Με αυτόν τον τρόπο θα μπορέσουμε να δοκιμάσουμε την δυνατότητα του εξυπηρετητή για να δέχεται ένα

μήνυμα και να το αποθηκεύουμε στην βάση δεδομένων και μετά να το σβήνουμε. Ακόμα, ρυθμίζουμε το Sipp, ώστε να πραγματοποιήσει αρχικά 10 κλήσεις μέσα σε ένα δευτερόλεπτο, με μέγιστο αριθμό κλήσεων τις πέντε εκατομμύρια κλήσεις. Ο αριθμός τον κλήσεων ανά δευτερόλεπτο θα αυξάνετε σταδιακά, μέχρι να αρχίσουμε να βλέπουμε τις πρώτες αναμεταδόσεις που είναι και ένα αρχικό δείγμα ότι η υπηρεσία μας δεν μπορεί να ανταποκριθεί στους προβλεπόμενους χρόνους.
Το αποτέλεσμα της παραπάνω διαδικασίας ήταν:

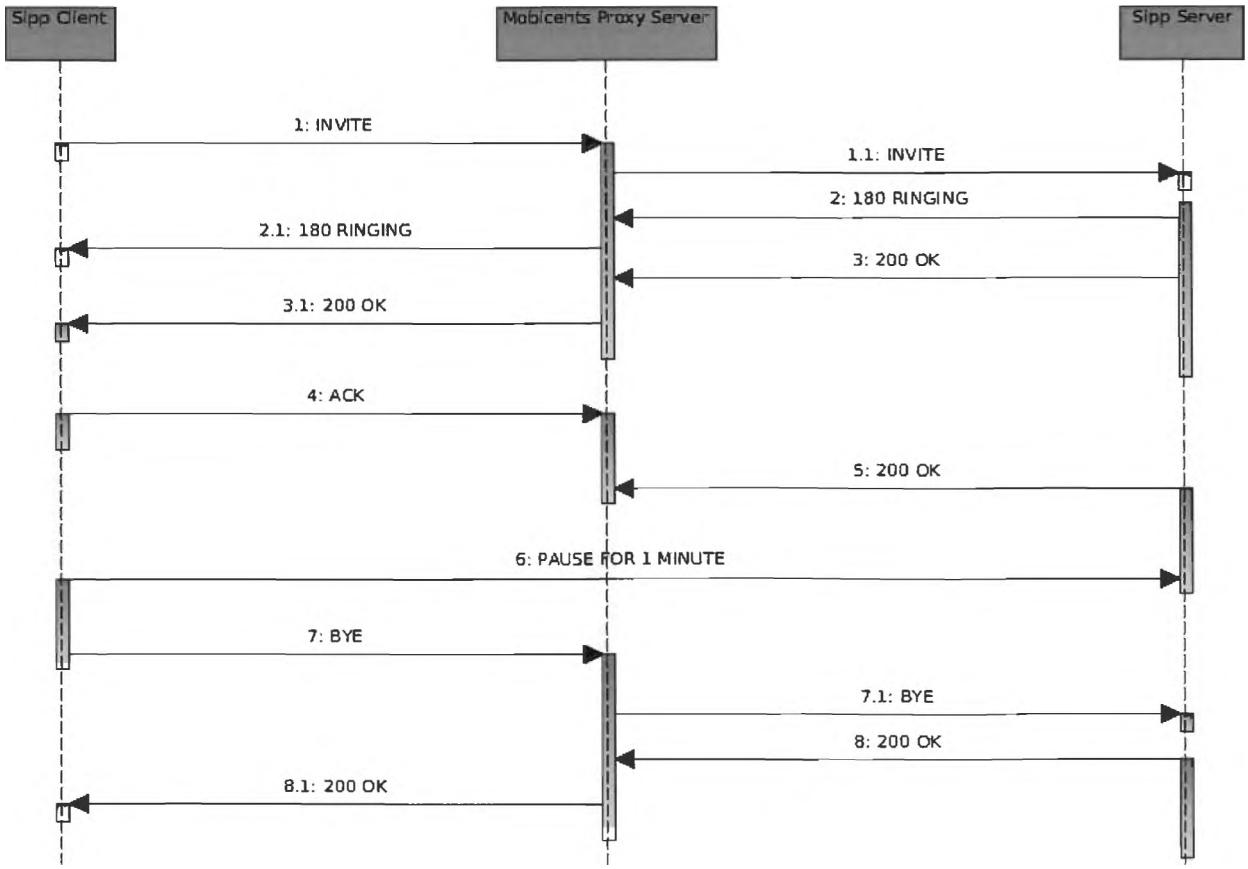
Σύνολο κλήσεων:	**2796**
Call-rate	80 κλήσεις/ ανά δευτερόλεπτο
Μέγιστος αριθμός παράλληλων κλήσεων	18.7



Ακολουθεί το διάγραμμα τον κλήσεων πάνω στο οποίο βλέπουμε τον αριθμό τον κλήσεων που πραγματοποιήθηκαν σε συνάρτηση με τον χρόνο που χρειάστηκε ώστε να αρχίσουμε να χάνουμε κλήσεις.

Τα παραπάνω στατιστικά θα μπορούσαν να αλλάξουν σημαντικά αλλάζοντας έστω και μία από τις παραμέτρους που αναφέραμε παραπάνω καθώς η εξομοίωση αυτή επικεντρώθηκε στο να αναγκάσουμε το σύστημα μας να προσπελαύνει την βάση δεδομένων διαρκώς, αφαιρώντας και προσθέτοντας εγγραφές. Από το διάγραμμα μπορούμε να παρατηρήσουμε ότι χρειάστηκε να αυξήσουμε σχεδόν κατακόρυφα τον αριθμό τον κλήσεων που έφταναν στον εξυπηρετητή, ώστε να αρχίσει να απορρίπτει κλήσεις, φτάνοντας τελικά να γίνουν 2796 κλήσεις.

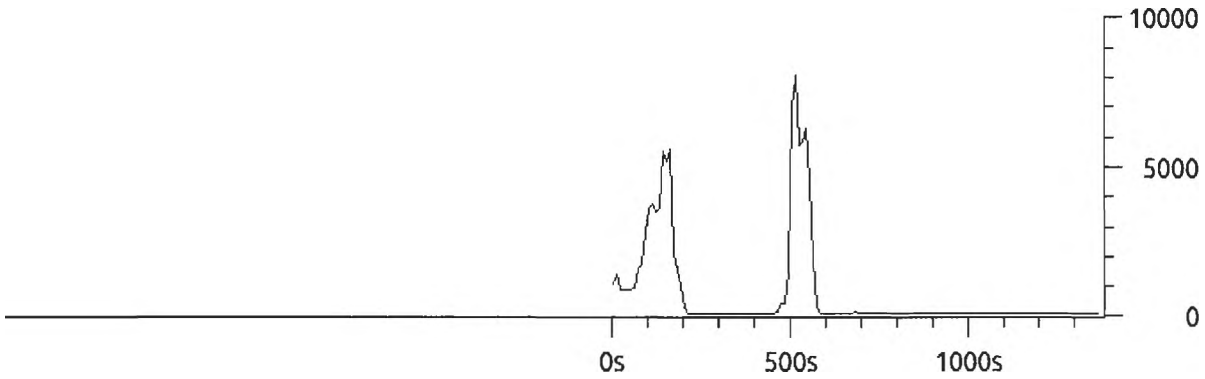
Η δεύτερη εξομοίωση, έχει να κάνει με την δοκιμή του Proxy server. Η ροή των κλήσεων παρουσιάζεται στο διάγραμμα



Σε αυτήν την περίπτωση έχουμε αλλάξει τον τρόπο με τον οποίο συμπεριφέρεται το Sipp, αφού του

επιτρέπουμε εξαρχής να φτάσει σε μεγάλο αριθμό παράλληλων κλήσεων, τον οποίο πάλι μεταβάλουμε κατά την διάρκεια τις εκτέλεσης. Μεταξύ των μηνυμάτων, παρεμβάλουμε και μία παύση του ενός λεπτού, η οποία αντιπροσωπεύει τον χρόνο τον οποίο διαρκεί κάθε κλήση. Έτσι το αντίστοιχο Transaction σε επίπεδο Sip είναι ενεργό στον εξυπηρετητή μας.

Σύνολο κλήσεων:	**5462**
Call-rate	70.0 κλήσεις/ ανά δευτερόλεπτο
Μέγιστος αριθμός παράλληλων κλήσεων	32.54



Όπως μπορούμε να δούμε και από το παραπάνω σχήμα, η υπηρεσία μας μπόρεσε να διαχειριστεί πάνω από 5000 κλήσεις συνολικά σε διάστημα 3 λεπτών, πριν αρχίσει να απορρίπτει κλήσεις. Και σε αυτό το διάγραμμα μπορούμε να παρατηρήσουμε το πόσο καλά μπορεί να ανταποκριθεί η υπηρεσία μας, σε περίπτωση απρόσμενης κίνησης. Αφού μετά τα 500s μπορούμε να δούμε ότι ο αριθμός τον αφίξεων αυξήθηκε σχεδόν κάθετα χωρίς να υπάρξει επίπτωση στον τρόπο με τον οποίο ο εξυπηρετητής δρομολογούσε τα πακέτα.

5.2 Συμπεράσματα-περαιτέρω μελέτη

Παρότι το τελικό αποτέλεσμα δεν ήταν το αναμενόμενο, και δεν καταφέραμε να φτιάξουμε το σύστημα που αρχικά σχεδιάζαμε, στα πλαίσια της πτυχιακής, είχα την ευκαιρία να μελετήσω πολλές τεχνολογίες τις οποίες μέχρι πρόσφατα δεν γνώριζα. Έμαθα για τον τρόπο και τις ανάγκες λειτουργίας των υπηρεσιών VoIP, τα πρωτόκολλα τα οποία χρησιμοποιούνται και τις υπηρεσίες που μπορούν να προσφέρουν. Εξοικειώθηκα με τον τρόπο που μελετάμε τα διαθέσιμα αρχεία όπως είναι τα RFC του Sip και τα XEP του XMPP, αλλά και με την δομή των κειμένων αυτών. Ακόμα εξοικειώθηκα με τον τρόπο που αναπτύσσουμε εφαρμογές για κινητά τηλέφωνα τις ιδιαιτερότητες και τις προκλήσεις του περιβάλλοντος αυτού. Έτσι είχα την δυνατότητα να μάθω περισσότερα για το λειτουργικό σύστημα Android, για τα εργαλεία που προσφέρει και τις δυνατότητες του.

Ήρθα σε επαφή με μεγάλα project, κάτι με το οποίο δεν είχα καθόλου εμπειρία, και τον τρόπο με τον οποίο πρέπει να αξιοποιείς όσο το δυνατόν περισσότερες πληροφορίες από αυτά, μελετώντας κομμάτια από πηγαίο κώδικα, αλλά και το πώς πρέπει να δουλεύεις με τα προσφερόμενα API και να συνδυάσεις πληροφορίες που παρέχονται από την κοινότητα και τα Forum υποστήριξης.

Επίσης έμαθα για διάφορα Design patterns (τρόπους σχεδιασμού), τα οποία χρησιμοποιούνται κατά την

διάρκεια του σχεδιασμού, ώστε να διευκολύνουν την διαδικασία της ανάπτυξης. Έτσι ήρθα σε επαφή με το Dependency Ingection pattern, Observer Observable pattern, αλλά και άλλα όπως είναι το μοντέλο σχεδιασμού που χρησιμοποιείται στο ίδιο το Jslee.

Με αυτόν τον τρόπο είχα την ευκαιρία να βελτιωθώ και να αποκτήσω εμπειρία στο πώς πρέπει να σχεδιάζεται ένα σύστημα. Σε αυτό βοήθησε και η ανάπτυξη του συστήματος μας, αφού έπρεπε ο σχεδιασμός να γίνει από την αρχή. Έτσι έμαθα το πώς προσαρμόζουμε τον σχεδιασμό του προγράμματος βάσει των αναγκών μας, αλλά και των διαθέσιμων τεχνολογιών. Πώς αποφασίζουμε για τις τεχνολογίες τις οποίες πρέπει να χρησιμοποιήσουμε, και πώς πρέπει να χωρίζουμε και να διαχειριζόμαστε κάθε κομμάτι ενός μεγαλύτερου προγράμματος ξεχωριστά ώστε να προσφέρουμε μία περίπλοκη υπηρεσία. Ήρθα σε επαφή με επαγγελματικά εργαλεία τα οποία χρησιμοποιούνται σε συνδυασμό με την γλώσσα java, όπως είναι το Ant, το Maven,και το Eclipse. Ακόμα στα πλαίσια της πτυχιακής είχα την ευκαιρία να βελτιώσω συνολικά τις γνώσεις μου στην Java, και ιδιαίτερα στην Java EE, αφού ο σχεδιασμός του Jslee είναι πολύ κοντά σε αυτόν και πολλά εργαλεία είναι κοινά. Αξιοποίησα τις υπάρχουσες γνώσεις μου για τον σχεδιασμό τόσο του εξυπηρετητή όσο και του πελάτη. Έμαθα το πώς πρέπει να ελέγχουμε ένα υλοποιημένο σύστημα, και το τι πρέπει να προσέχουμε κατά την διάρκεια των ελέγχων, καθώς και πώς να γράφουμε ρουτίνες έλεγχου της ποιότητας τους, και μέτρησης της απόδοσης.

Ακόμα εξοικειώθηκα με τις ανάγκες που έχουν οι εφαρμογές πραγματικού χρόνου και πώς μπορούμε να πετύχουμε καλύτερη απόδοση μέσα από προσεκτική σχεδίαση.

Η συγκεκριμένη εργασία, λόγο της ιδιαιτερότητας ότι αποτελείται από πολλά ξεχωριστά κομμάτια, επιτρέπει την περαιτέρω βελτίωση και μελέτη σε καθένα από αυτά ξεχωριστά. Έτσι για τον Android θα μπορούσαμε να την επεκτείνουμε και να μελετήσουμε αλγόριθμους που θα επέτρεπαν την καλύτερη διαχείριση ενέργειας ώστε να πετύχουμε μεγαλύτερη διάρκεια ζωής της μπαταρίας. Επίσης ενδιαφέρουσα θα ήταν μία μελέτη η οποία θα ένωνε το πρωτόκολλο presence με διάφορα profile τα οποία θα επέλεγε ο χρήστης και έτσι ο πελάτης του κινητού ανάλογα με την γεωγραφική θέση του κατόχου και το profile του, θα μπορούσε να ανανεώνει αυτόματα την διαθεσιμότητα του χρήστη, ή θα μπορούσε να πάρει κάποια μέτρα ανάλογα (π.χ. Να μην δέχεται κλήσεις εν ώρα κοινής ησυχίας, ή όταν ο κάτοχος της συσκευής είναι στον εργασιακό του χώρο.).

Στο κομμάτι του Jslee θα μπορούσαμε να φτιάξουμε δική μας υλοποίηση για το XMPP η οποία να υποστηρίζει το Jingle, ώστε να μπορέσουμε να χρησιμοποιήσουμε το mobicents σαν Gateway server. Ακόμα λόγω του πολύ καλού media server (μέσω MGCP) θα μπορούσαμε να επεκτείνουμε την υπάρχουσα υλοποίηση ώστε να υποστηρίζει εφαρμογές πολυμέσων (π.χ. Video On Demand) κάνοντας χρήση πιθανόν και της κάρτας γραφικών του συστήματος για καλύτερη απόδοση.

Στο κομμάτι του Wave Federation protocol θα μπορούσαμε να δούμε εφαρμογές πραγματικού χρόνου όπως είναι push notification εφαρμογές.

6 Παραπομπές/βιβλιογραφία

Mobicents/JSlee

- http://www.mobicents.org/community.html
- http://www.mobicents.org/
- http://www.ietf.org/rfc/rfc3261.txt
- http://www.ietf.org/rfc/rfc3550.txt
- http://xmpp.org/extensions/xep-0166.html
- https://developer.opencloud.com/devportal/display/OCDEV/Hints+and+Tips+for+Writing+Well+Performing+SLEE+Applications

- http://www.scribd.com/doc/17681458/JSLEE-11-Overview-and-Concepts
- http://www.jainslee.org/
- Jain slee specification 1.1

Google Wave

- http://code.google.com/intl/el/apis/wave/
- http://www.waveprotocol.org/code
- http://wave-protocol.googlecode.com/hg/spec/federation/wavespec.html
- http://wave-protocol.googlecode.com/hg/spec/conversation/convspec.html
- http://wave-protocol.googlecode.com/hg/spec/waveid/waveidspec.html
- http://www.waveprotocol.org/protocol/design-proposals
- http://www.waveprotocol.org/protocol/design-proposals/authentication
- http://www.waveprotocol.org/protocol/design-proposals/clientserver-protocol
- http://www.waveprotocol.org/protocol/design-proposals/new-wave-panel-undercurrent
- http://wave-protocol.googlecode.com/hg/whitepapers/google-wave-architecture/google-wave-architecture.html
- http://wave-protocol.googlecode.com/hg/whitepapers/client-server-protocol/client-server-protocol.html
- http://wave-protocol.googlecode.com/hg/whitepapers/operational-transform/operational-transform.html
- http://www.waveprotocol.org/whitepapers/wave-protocol-verification
- http://wave-protocol.googlecode.com/hg/whitepapers/access-control/access-control.html
- http://wave-protocol.googlecode.com/hg/whitepapers/attachments/attachments.html

Android/SIP/Jingle/XMPP

- http://developer.android.com/guide/index.html
- http://sipdroid.org/
- http://java.sun.com/products/jain/SIP-and-Java.html
- http://jcp.org/aboutJava/communityprocess/final/jsr032/
- http://www.hsc.com/
- http://www.mjsip.org/
- http://xmpp.org/internet-drafts/draft-saintandre-sip-xmpp-media-01.html#sip

Java voice recognition/synthesys

- http://cmusphinx.sourceforge.net/
- http://freetts.sourceforge.net/docs/index.php
- http://developer.android.com/guide/index.html
- http://code.google.com/p/eyes-free/