Σχολή
Ηλεκτρολόγων Μηχανικών
και
Μηχανικών Υπολογιστών

Τεχνολογία Λογισμικού, 9ο εξάμηνο

2ο Παραδοτέο
Έγγραφο Προδιαγραφής Σχεδίασης

Έκδοση :	1.0
Ημερομηνία Έκδοσης :	16/12/2016
Κατάσταση Έκδοσης :	
Κατάσταση Έγκρισης :	
Έγκριση από :	
Προετοιμασία από :	Θεοδώσης Εμμανουήλ Μπενετόπουλος Αχιλλέας Πανταζόπουλος Γεώργιος - Μιχαήλ Φραγκάκη Μαρία – Ελένη
Έλεγχος από :	
Θέση αρχείου :	
Όνομα αρχείου :	SDD.pdf
Αριθμός εγγράφου :	

Πίνακας Περιεχομένων

1. Εισαγωγή

1.1 Επισκόπηση

Στην αρχή του παρόντος εγγράφου περιγράφουμε συνοπτικά τις κυριότερες αποφάσεις που πήραμε σχετικά με τη σχεδίαση και τη λειτουργικότητα του συστήματος. Στη συνέχεια περιγράφουμε την αρχιτεκτονική του συστήματος συνολικά (Component & Deployment diagrams), και κατόπιν εμβαθύνουμε σε αυτήν με την περιγραφή των επιμέρους μονάδων που το αποτελούν, καθώς και το πώς αυτές αλληλεπιδρούν (Class diagram & pseudocode). Τέλος, παρουσιάζουμε τη συμπεριφορά των υπομονάδων του συστήματος κατά τη διάρκεια ζωής της εφαρμογής (State diagrams).

1.2 Αναφορές

[1] Πρότυπο RFC 3261 – SIP : https://www.ietf.org/rfc/rfc3261.txt

[2] Διαφάνειες μαθήματος

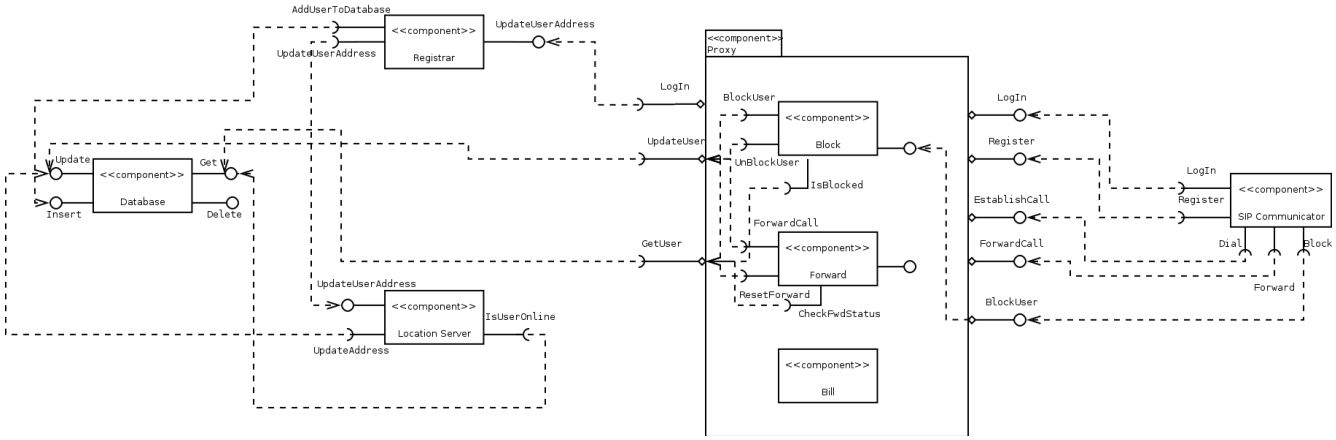
2. Μείζονες Σχεδιαστικές Αποφάσεις

Το κυριότερο κομμάτι του συστήματός μας είναι ο Proxy Server. Αυτός αποτελεί τη μοναδική διεπαφή των χρηστών με την υπηρεσία, και οποιοδήποτε αίτημα του χρήστη περνάει αναγκαστικά από τον Proxy, ο οποίος με τη σειρά του είτε το εξυπηρετεί, είτε το προωθεί στον αρμόδιο εξυπηρετητή.
Όπως θα γίνει εμφανές και παρακάτω, οι λειτουργίες του Billing, του Blocking και του Forwarding υλοποιούνται σε ανεξάρτητα modules, τα οποία ο Proxy χρησιμοποιεί για να πραγματοποιήσει τις αντίστοιχες λειτουργίες. Συνεπώς, είναι πολύ εύκολο αυτά να αλλάξουν για να καλύπτουν τις ανάγκες της εφαρμογής και να προσαρμοστούν σε οποιεσδήποτε αλλαγές στην αρχιτεκτονική του συστήματος ανα πάσα στιγμή, χωρίς ιδιαίτερο προγραμματιστικό κόπο.
Όλα τα στοιχεία χρηστών αποθηκεύονται σε ένα MySQL database, το οποίο έχουν το δικαίωμα να ενημερώνουν τόσο ο Proxy Server, όσο και οι Registrar και Location servers.

3. Αρχιτεκτονική

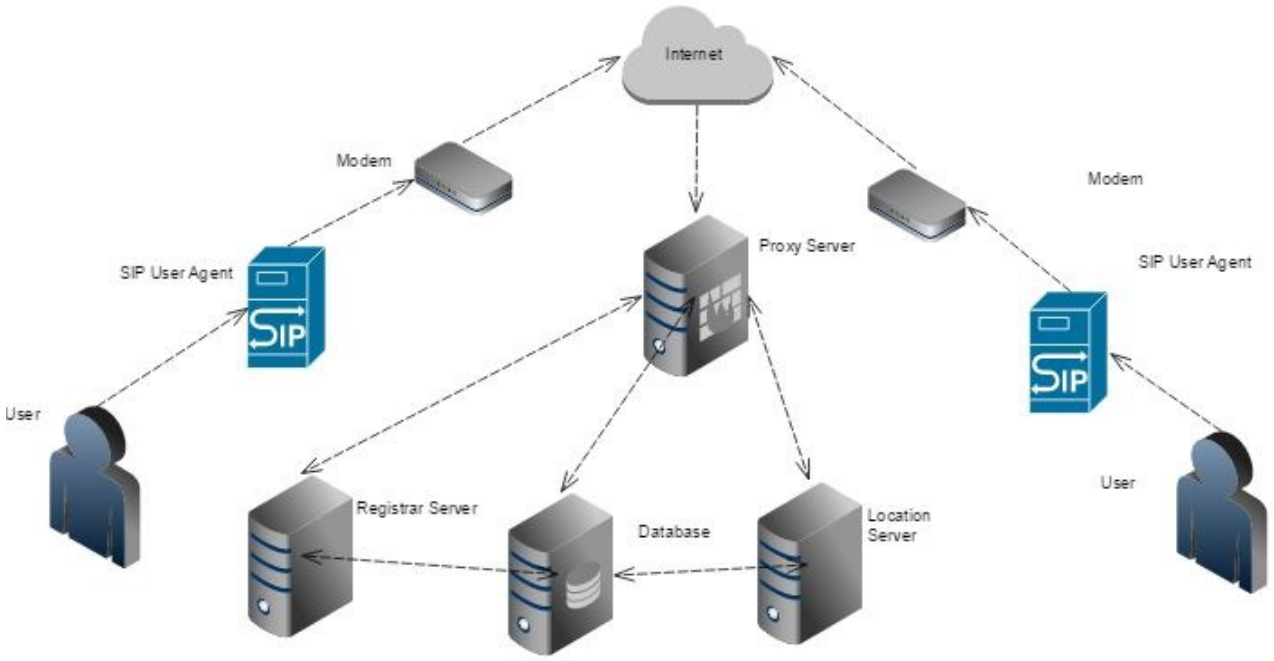
Component Diagram

Παρακάτω παρουσιάζεται το component diagram. Το διάγραμμα αυτό περιέχει πληροφορία αναφορικά με το ποιες διαπροσωπείες είναι διαθέσιμες στο σύστημα. Επιπλέον, εμπεριέχει πληροφορία αναφορικά με το ποιος είναι ο πάροχος αυτής της διαπροσωπείας και ποιος χρειάζεται αυτή τη διαπροσωπεία ώστε να λειτουργεί ορθά και να μπορεί, στη συνέχεια, να παρέχει τις δικές του διαπροσωπείες στο υπόλοιπο σύστημα.



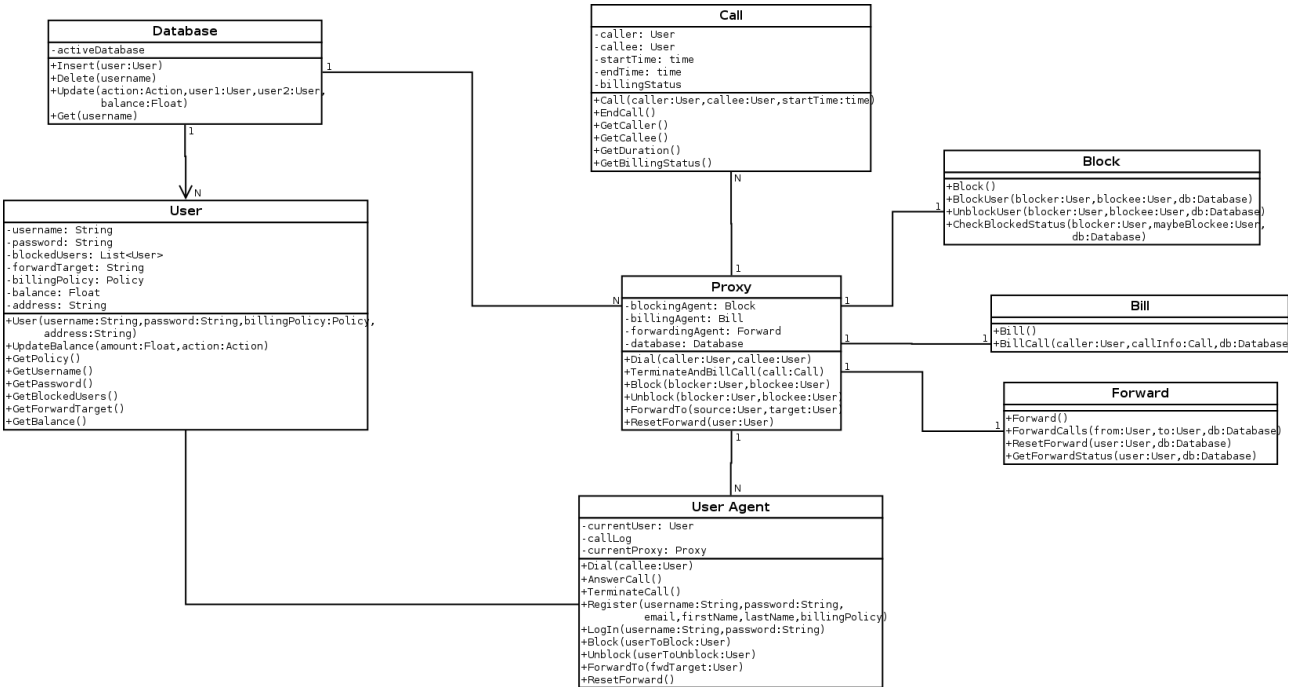
Deployment Diagram

Παρακάτω φαίνεται η μακροσκοπική επικοινωνία χρήστη - συστήματος μέσω του Deployment Diagram. Όπως έχει ήδη αναφερθεί, ο χρήστης μέσω του SIP User Agent αποκτά πρόσβαση στις λειτουργίες του συστήματος. Οι επιθυμιές του χρήστη ερμηνεύονται ως αιτήσεις προς τον Proxy Server, ο οποίος επικοινωνεί με τους υπόλοιπους Servers του συστήματος με σκοπό να εξυπηρετήσει τις ανάγκες του χρήστη.



4. Λεπτομερές Διάγραμμα Κλάσεων

4.1 Διάγραμμα Κλάσεων UML



Όπως αναφέραμε και πριν, ο Proxy εξυπηρετεί ένα μόνο μέρος των αιτημάτων που είναι πιθανόν να γίνουν από τους χρήστες. Για όλα τα υπόλοιπα αιτήματα παρέχει μεν ένα interface στο χρήστη, αλλά όταν τα λάβει τα προωθεί στο αρμόδιο component. Αυτό αποτελεί υλοποίηση του "Chain of Responsibility" pattern.

4.2 Λεπτομέρειες Μεθόδων

Ακολουθεί ο ψευδοκώδικας για όσες από τις μεθόδους που αναγράφονται στο class diagram είτε δημιουργήσαμε εμείς, είτε προϋπήρχαν και τροποποιήσαμε (στις περισσότερες κλάσεις έχει παραληφθεί ο κώδικας των accessors, καθώς δεν εμφανίζει κάποιο ενδιαφέρον).

```
================================================================================
Call class:
================================================================================

Call(caller, callee, startTime) { //constructor
    this.caller = caller;
    this.callee = callee
    this.startTime = getcurrenttime();
    this.billingStatus = UNBILLED;
}

EndCall() {
    this.endTime = getcurrenttime();
}

GetCaller() {
    return this.caller;
}
```

```
GetDuration() {
    return this.endTime - this.startTime;
}


================================================================================
Block class:
================================================================================

BlockUser(blocker, blockee, db) {
    while(db.Update(ACTION_BLOCK, blocker, blockee) != SUCCESS) {
        wait();
    }

    return SUCCESS;
}

UnblockUser(blocker, blockee, db) {
    blockeeUsername = blockee.GetUsername();
    while(db.Update(ACTION_UNBLOCK, blocker, blockeeUsername) != SUCCESS) {
        wait();
    }

    return SUCCESS;
}

CheckBlockedStatus(blocker, maybeBlockee, db) {
    blockedList = (db.Get(blocker)).GetBlockedUsers();
    mbUsername = maybeBlockee.GetUsername();

    for (entry e in blockedList)
        if e == mbUsername then return true;

    return false;
}

================================================================================
Bill class:
================================================================================

BillCall(callInfo, db) {
    caller = callInfo.GetCaller();
    plan = caller.GetBillingPlan();
    duration = callInfo.GetDuration();

    cost = switch plan {
        case PlanA:
            ...
        case PlanB:
            ...
        ...
    };

    caller.UpdateBalance(cost, ACTION_CHARGE, db);
    return SUCCESS;
}

================================================================================
Forward class:
================================================================================

ForwardCalls(source, dest, db) {
    while (db.Update(ACTION_FORWARD, source, dest, 0) != SUCCESS) {
        wait();
    }

    return SUCCESS;
}

ResetForward(source, db) {
    while (db.Update(ACTION_UNFORWARD, source, NULL, 0) != SUCCESS) {
```

```
        wait();
    }

    return SUCCESS;
}

GetForwardStatus(user, db) {
    target = (db.Get(user)).GetForwardTarget();

    if (target == null)
        return false;

    return true;
}
```

```
================================================================================================
User Class:
================================================================================================
```

```
User(username, password, billingPolicy, address) {
    this.username = username;
    this.password = password;
    this.blockedUsers = empty();
    this.forwardTarget = null;
    this.billingPolicy = billingPolicy;
    this.balance = 0.0;
    this.address = address;

    while(db.Insert(this) != SUCCESS) {
        wait();
    }
}

UpdateBalance(amount, action, db) {
    while (db.Update(action, this, null, amount) != SUCCESS) {
        wait();
    }

    return SUCCESS;
}
```

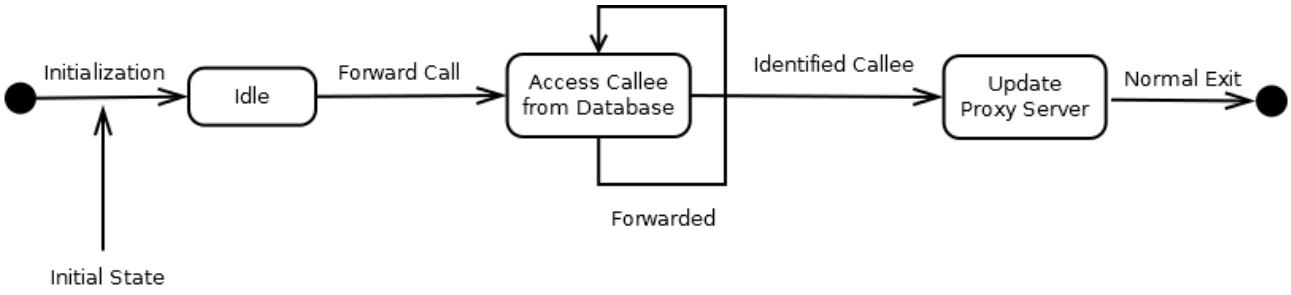
```
================================================================================================
Database Class:
================================================================================================
```

```
Insert(user) {
    query = "INSERT ...";
    openDatabase.ExecuteQuery(query);
    return SUCCESS;
}

Update(action, user1, user2, num) {
    query = "UPDATE ...";
    openDatabase.ExecuteQuery(query);
    Get(user1);
    return SUCCESS;
}

Delete() {
    query = "DELETE ...";
    openDatabase.ExecuteQuery(query);
    return SUCCESS;
}

Get(user) {
    query = "GET ...";
    res = openDatabase.ExecuteQuery(query);
    UpdateUserBasedOnQuery(user, res);
    return SUCCESS;
}
```

```
================================================================================================
Proxy Class:
================================================================================================

Dial(caller, callee) {
    calleeUser = db.Get(callee);
    isBlocked = blockingAgent.CheckBlockedStatus(calleeUser, caller, database);

    if (isBlocked) then
        //signify unavailable
    else
        forwardTarget = {};
        while ((targetUser = forwardingAgent.GetForwardStatus(calleeUser, database)) !=
null)
            if (targetUser in forwardTargets) then
                //signify loop
            else
                add targetUser to forwardTargets
            calleeUser = targetUser;

        newCall = new Call(caller, calleeUser);
        //regular call procedure
        return newCall;
    endif

    return null;
}

TerminateAndBillCall(call) {
    call.EndCall();
    billingAgent.BillCall(call, database);
}

Block(blocker, blockee) {
    blockingAgent.BlockUser(blocker, blockee, database);
    return SUCCESS;
}

UnblockUser(blocker, blockee) {
    blockingAgent.UnblockUser(blocker, blockee, database);
    return SUCCESS;
}

ForwardTo(source, target) {
    forwardingAgent.ForwardCalls(source, target, database);
    return SUCCESS;
}

ResetForward(user) {
    forwardingAgent.ResetForward(user, database);
}

================================================================================================
User Agent Class:
================================================================================================

Block(userToBlock) {
    curentProxy.Block(currentUser, userToBlock);
}

Unblock(userToUnblock) {
    curentProxy.Unblock(currentUser, userToUnblock);
}

ForwardTo(fwdTarget) {
    currentProxy.ForwardTo(currentUser, fwdTarget);
}

ResetForward() {
    currentProxy.ResetForward(currentUser);
}
```

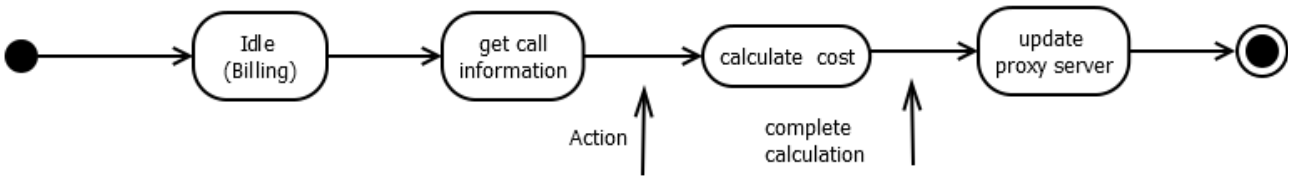
5. Διαγράμματα Κατάστασης

Forwarding



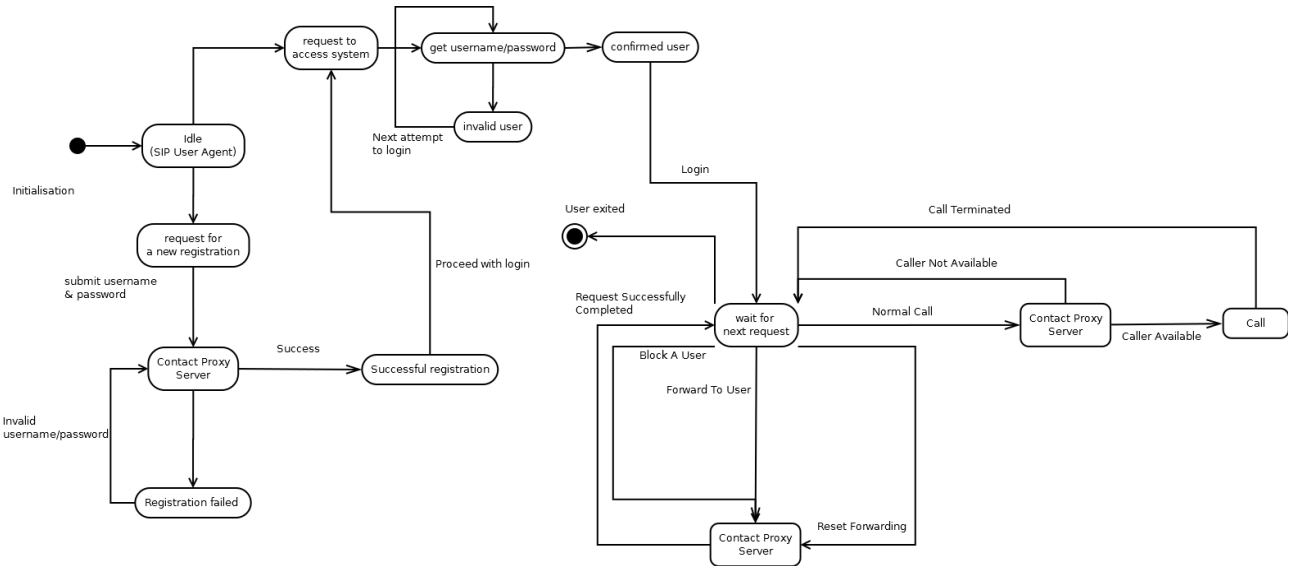
Στο παραπάνω διάγραμμα φαίνεται η ακολουθία καταστάσεων για το Forwarding Component του συστήματος. Ξεκινώντας από μια αρχική κατάσταση Idle, δίνεται το έναυσμα της προώθησης κλήσης. Η επόμενη κατάσταση περιλαμβάνει την λύση του προβλήματος προώθησης. Στην περίπτωση που ο προωθημένος καλούμενος έχει ενεργοποιήσει το σύστημα προώθησης, τότε μέσω της ανάδρασης βρίσκεται ο αμέσως επόμενος καλούμενος χρήστης. Στην περίπτωση αυτή ενημερώνεται ο Proxy Server και η διαδικασία ολοκληρώνεται ομαλά.

Billing



Παραπάνω φαίνεται η ακολουθία καταστάσεων για την χρέωση της κλήσης. Ο Proxy Server λαμβάνει από την βάση δεδομένων τα απαραίτητα στοιχεία της κλήσης που χρειάζονται για την κοστολόγησή της (στοιχεία του caller στον οποίο εναποτίθεται η χρέωση, η διάρκεια της κλήσης και όποιες άλλες παράμετροι έχουν τεθεί από την πολιτική χρέωσης της εφαρμογής μας). Μόλις τα στοιχεία διατεθούν στον server υπολογίζεται η αντίστοιχη χρέωση, ο Proxy server ανανεώνεται και η διαδικασία τερματίζει.

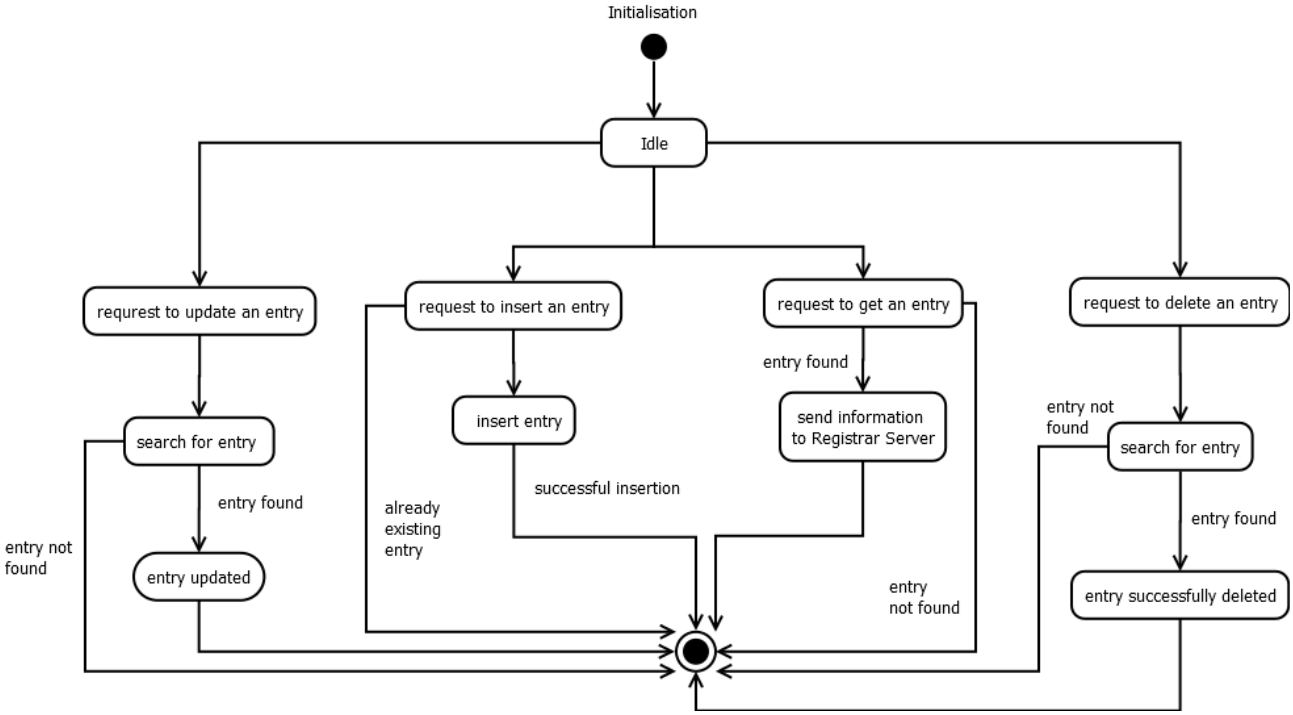
SIP User Agent



Παραπάνω φαίνεται η ακολουθία καταστάσεων για τον SIP User Agent. Αρχικά, εκκινεί από μία κατάσταση Idle μέχρι να δεχθεί ένα αίτημα προς διεκπεραίωση. Μία περίπτωση είναι να δεχθεί αίτημα για εγγραφή ενός χρήστη στο σύστημα. Ο SIP User Agent επικοινωνεί με τον Proxy Server για να ενημερωθεί αν η εγγραφή πετυχαίνει ή όχι. Εάν η εγγραφή αποτύχει τότε ο SIP User Agent αποστέλλει νέα στοιχεία εγγραφής στον Proxy Server έως ότου να γίνουν δεκτά. Όταν η εγγραφή ολοκληρωθεί ο χρήστης ζητάει άδεια για πρόσβαση στο σύστημα. Ελέγχεται αν τα στοιχεία που υποβάλλει είναι έγκυρα ή όχι και αναλόγως του επιτρέπεται πρόσβαση ή όχι. Από την στιγμή που ο user κάνει login, ο SIP User Agent αναμένει να δεχθεί αίτημα για τις εξής ενέργειες:

1. Κλήση, όπου ο SIP User Agent επικοινωνεί με τον Proxy Server. Υπάρχουν δύο σενάρια: είτε ο callee είναι διαθέσιμος, οπότε και η κλήση διεκπεραιώνεται κανονικά έως ότου τερματιστεί και ο SIP User Agent περνάει σε φάση αναμονής για νέο αίτημα, είτε ο callee δεν είναι διαθέσιμος, οπότε ο SIP User Agent αναμένει εκ νέου αίτημα.
2. Blocking/unblocking, όπου και πάλι ο SIP User Agent επικοινωνεί με τον Proxy Server ο οποίος αναλαμβάνει την πραγματοποίηση του αιτήματος και ο SIP User Agent αναμένει καινούργιο αίτημα.
3. Reset Forwarding, αίτημα το οποίο κι πάλι αποστέλλεται στον Proxy Server για περάτωση και ο SIP User Agent μπαίνει σε κατάσταση αναμονής για νέο αίτημα.
4. Αποσύνδεση του χρήστη από το σύστημα, οπότε ο SIP User Agent τερματίζει την λειτουργία του.

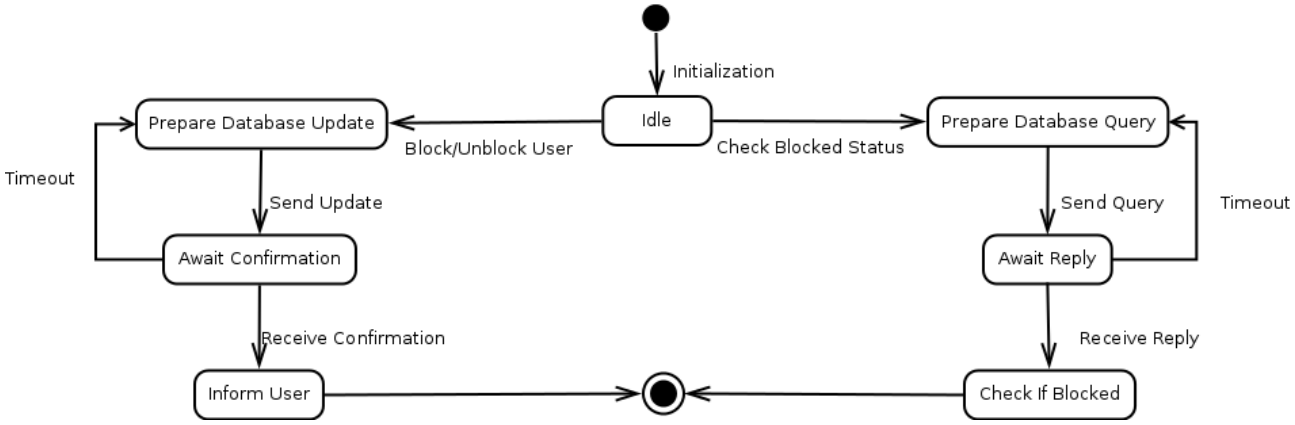
Database



Παραπάνω φαίνεται η ακολουθία καταστάσεων για την Βάση Δεδομένων. Αρχικά η ΒΔ ξεκινάει από μία αδρανή (idle) κατάσταση αναμένοντας κάποιο αίτημα. Τα πιθανά αιτήματα που μπορεί να δεχθεί η Βάση Δεδομένων είναι :

1. Εισαγωγή νέου entry (Insert): Η Βάση Δεδομένων καλείται να εισάγει μία νέα καταχώριση. Σε αυτήν την περίπτωση είτε η εγγραφή ολοκληρώνεται επιτυχώς είτε αποτυγχάνει σε περίπτωση που υπάρχει ήδη κάποια καταχώριση με τα ίδια στοιχεία.
2. Διαγραφή κάποιου entry (Delete): Η βάση δεδομένων αναζητά το entry για το οποίο έγινε το αίτημα διαγραφής. Εάν υπάρχει, η διαγραφή γίνεται επιτυχώς και η λειτουργία τερματίζει, ειδάλλως εάν το entry δεν υπάρχει στην βάση, η λειτουργία τερματίζει δίχως να κάνει κάτι περαιτέρω.
3. Αναβάθμιση κάποιου υπάρχοντος entry (Update): Εάν η εν λόγω καταχώριση υπάρχει στην βάση, η ανανέωση των στοιχείων της ολοκληρώνεται κανονικά, διαφορετικά η λειτουργία τερματίζει.
4. Get entry: Η Βάση Δεδομένων δέχεται ένα αίτημα για να αποστείλλει στοιχεία ενός συγκεκριμένου entry στον Registrar Server. Ξανά, εάν η εγγραφή υπάρχει στην βάση, τα στοιχεία στέλνονται επιτυχώς αλλιώς η λειτουργία τερματίζει με error (Entry not found).

Blocking



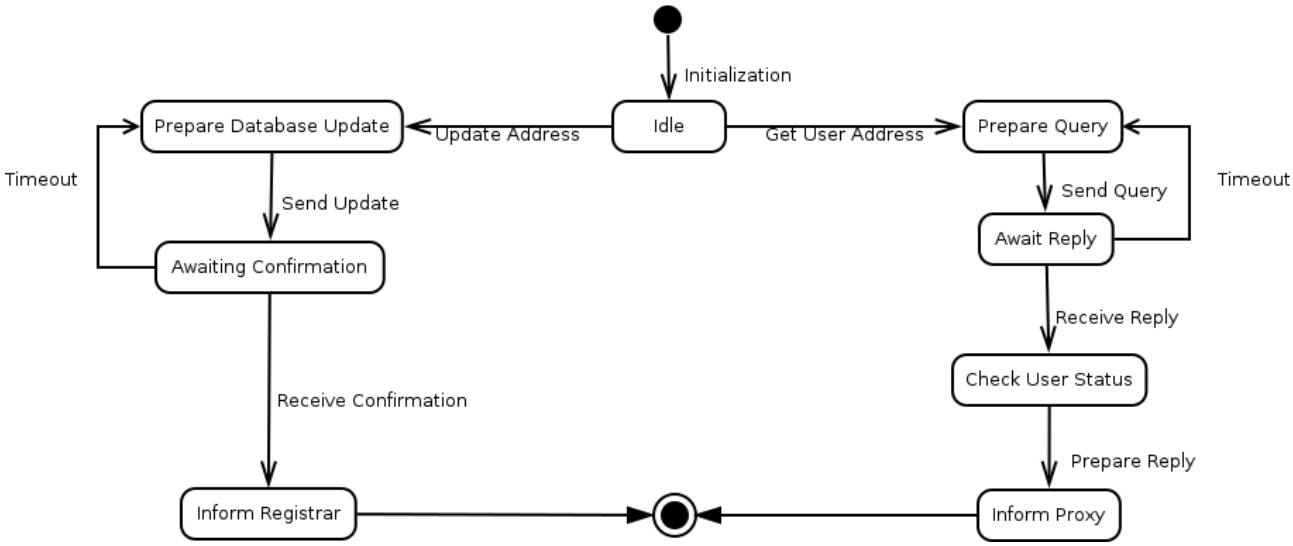
Στο παραπάνω διάγραμμα φαίνεται η ακολουθία καταστάσεων για τις ενέργειες σχετικές με τον αποκλεισμό χρηστών. Αρχικά ο Proxy Server εκκινεί από μια κατάσταση Idle, μέχρις ότου να δεχθεί σήμα εισόδου αναφορικά με την επιθυμητή ενέργεια του χρήστη.
Εδώ υπάρχουν δύο δυνατά σενάρια ενέργειας χρήστη: είτε θέλει να ελέγξει αν κάποιος χρήστης είναι αποκλεισμένος, είτε θέλει να προσθέσει/αφαιρέσει κάποιον χρήστη από την λίστα αποκλεισμού.

Στην πρώτη περίπτωση, ο Proxy Server ετοιμάζει το αίτημα για την Βάση Δεδομένων, το αποστέλλει, και αναμένει απάντηση. Μόλις λάβει απόκριση από τη Βάση Δεδομένων, ενημερώνει τον χρήστη για το αίτημα του, και οδηγείται στην τερματική κατάσταση.

Αντίστοιχα, στην άλλη περίπτωση, ο Proxy Server ετοιμάζει το αίτημα για την ανανέωση της Βάσης Δεδομένων, το αποστέλλει, και αναμένει απάντηση. Μόλις λάβει απόκριση από τη Βάση Δεδομένων, ενημερώνει το χρήστη για το αποτέλεσμα του αιτήματός του, και οδηγείται στην τερματική κατάσταση.

Σε κάθε περίπτωση, εάν εκπνεύσει ένα χρονικό παράθυρο, τότε ο Proxy Server αποστέλλει εκ νέου αίτημα στη Βάση Δεδομένων.

Location Server



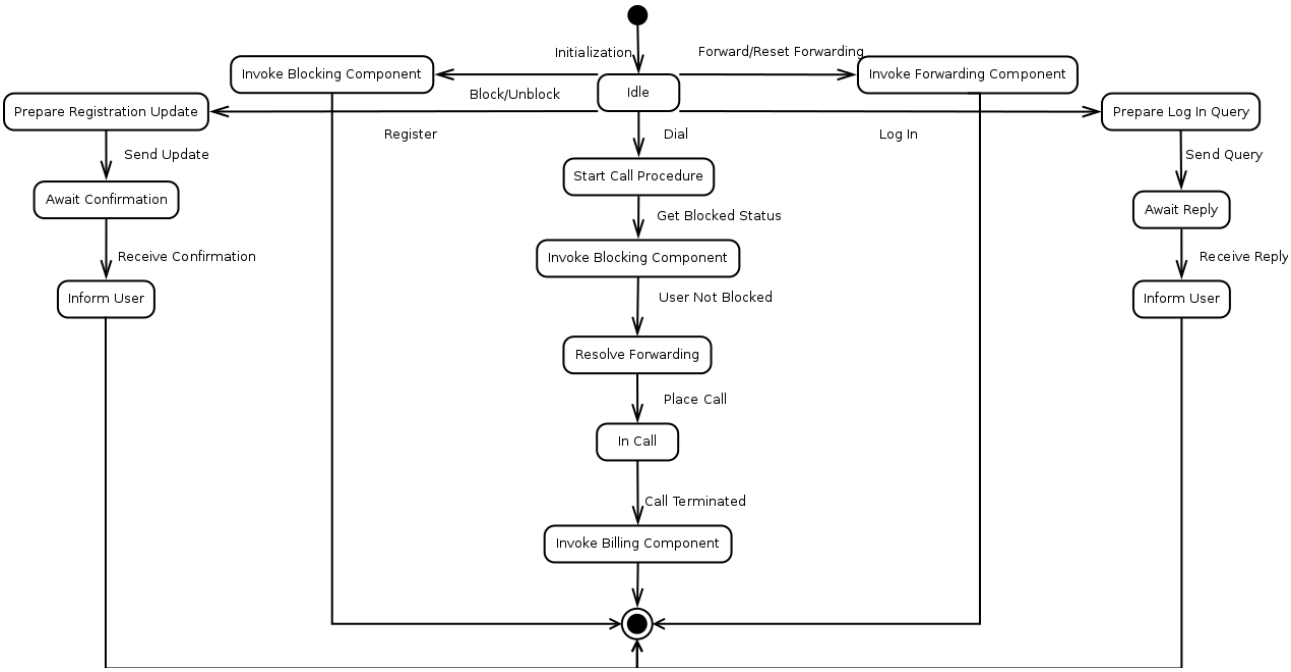
Στο παραπάνω διάγραμμα φαίνεται η ακολουθία καταστάσεων για τις ενέργειες σχετικές με την τοποθεσία χρηστών. Αρχικά ο Location Server εκκινεί από μια κατάσταση Idle, μέχρις ώτου να δεχθεί σήμα εισόδου αναφορικά με την προβλεπόμενη ενέργεια.
Εδώ υπάρχουν δύο δυνατά σενάρια ενέργειας: είτε ο Location Server πρέπει να αποκριθεί για την θέση κάποιου χρήστη, είτε πρέπει να ανανεώσει την θέση κάποιου χρήστη.

Στην πρώτη περίπτωση, ο Location Server ετοιμάζει το αίτημα για την Βάση Δεδομένων, το αποστέλλει, και αναμένει απάντηση. Μόλις λάβει απόκριση από τη Βάση Δεδομένων, ελέγχει την απόκριση, ενημερώνει τον Proxy Server για το αποτέλεσμα του αιτήματος, και οδηγείται στην τερματική κατάσταση.

Αντίστοιχα, στην άλλη περίπτωση, ο Location Server ετοιμάζει το αίτημα για την ανανέωση της Βάσης Δεδομένων, το αποστέλλει, και αναμένει απάντηση. Μόλις λάβει απόκριση από τη Βάση Δεδομένων, ενημερώνει τον Registrar Server για το αποτέλεσμα του αιτήματος, και οδηγείται στην τερματική κατάσταση.

Σε κάθε περίπτωση, εάν εκπνεύσει ένα χρονικό παράθυρο, τότε ο Location Server αποστέλει εκ νέου αίτημα στη Βάση Δεδομένων.

Proxy Server



Στο παραπάνω διάγραμμα φαίνεται η ακολουθία καταστάσεων σχετικά με τις ενέργειες του Proxy Server. Αρχικά ο Proxy Server εκκινεί από μια κατάσταση Idle, μέχρις ότου να δεχθεί σήμα εισόδου αναφορικά με την προβλεπόμενη ενέργεια.

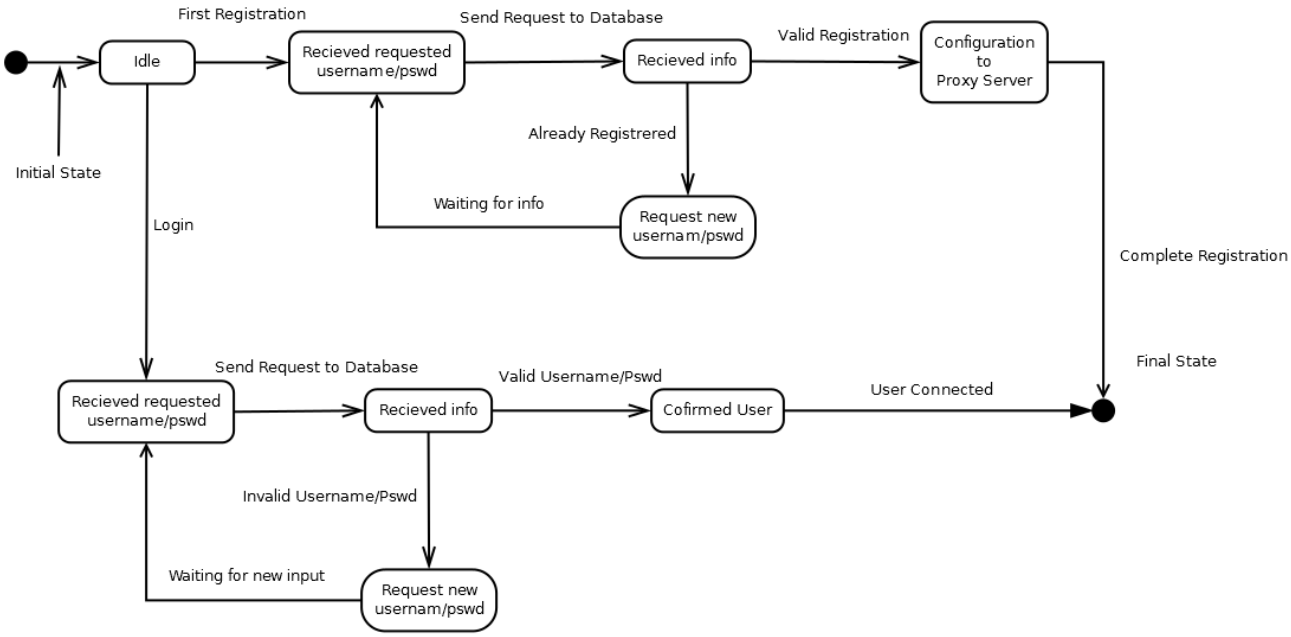
Εδώ υπάρχουν πέντε δυνατά σενάρια ενέργειας:
- ο χρήστης έκανε αίτημα αναφορικά με την προώθηση κλήσεων.
- ο χρήστης έκανε αίτημα αναφορικά με τον αποκλεισμό χρηστών.
- ο χρήστης έκανε αίτημα αναφορικά με την εγγραφή.
- ο χρήστης έκανε αίτημα αναφορικά με την σύνδεση στο σύστημα.
- ο χρήστης έκανε αίτημα κλήσης άλλου χρήστη.

Ανάλογα με την περίπτωση, ο Proxy Server ακολουθεί διαφορετικές ενέργειες. Συγκεκριμένα:
- Στην περίπτωση που ο χρήστης έκανε αίτημα αναφορικά με την προώθηση κλήσεων, τότε ο Proxy Server απλώς καλεί το component αναφορικά με την προώθηση κλήσεων, και οδηγείται στην τερματική κατάσταση.
- Στην περίπτωση που ο χρήστης έκανε αίτημα αναφορικά με τον αποκλεισμό χρηστών, τότε ο Proxy Server απλώς καλεί το component αναφορικά με τον αποκλεισμό χρηστών, και οδηγείται στην τερματική κατάσταση.
- Στην περίπτωση που ο χρήστης έκανε αίτημα αναφορικά με την εγγραφή του στο σύστημα, τότε ο Proxy Server ετοιμάζει το αίτημα για την ανανέωση της Βάσης Δεδομένων, το αποστέλνει, και αναμένει απάντηση. Μόλις λάβει απόκριση από τη Βάση Δεδομένων, ενημερώνει τον χρήστη για το αποτέλεσμα του αιτήματός του, και οδηγείται στην τερματική κατάσταση.
- Στην περίπτωση που ο χρήστης έκανε αίτημα αναφορικά με την σύνδεσή του στο σύστημα, τότε ο Proxy Server ετοιμάζει το αίτημα για την ανανέωση της Βάσης Δεδομένων, το αποστέλλει, και αναμένει απάντηση. Μόλις λάβει απόκριση από τη Βάση Δεδομένων, ενημερώνει το χρήστη για το αποτέλεσμα του αιτήματός του, και οδηγείται στην τερματική κατάσταση.
- Στην περίπτωση που ο χρήστης έκανε αίτημα κλήσης άλλου χρήστη, τότε ο Proxy Server εκκινεί τη διαδικασία κλήσης. Αρχικά, καλεί το component αναφορικά με τον αποκλεισμό χρηστών για να αποφανθεί εάν είναι επιτρεπτή η κλήση του επιθυμητού χρήστη. Στον ομαλό τερματισμό του component αποκλεισμού χρηστών, έπειτα καλείται το component αναφορικά

με την επίλυση των προωθήσεων. Μόλις αποκριθεί και αυτό, ο Proxy Server εκκινεί την κλήση μεταξύ των δύο χρηστών.

Στην περάτωση της κλήσης, ο Proxy Server καλεί το component για την χρέωση του κατάλληλου χρήστη, και οδηγείται στην τερματική κατάσταση.

Registrar Server



Στο παραπάνω διάγραμμα φαίνεται η ακολουθία καταστάσεων του Registrar Server. Ο ίδιος, ξεκινώντας από μία αρχική κατάσταση IDLE, δέχεται ένα σήμα εισόδου με το οποίο προσδιορίζει αν ο χρήστης είναι εγγεγραμμένος στο σύστημα.

Σε αυτήν την περίπτωση ακολουθεί το μονοπάτι Login. Αναμένει τα χαρακτηριστικά του χρήστη και συμβουλεύεται την βάση δεδομένων για την ορθότητά τους. Αν τα χαρακτηριστικά του χρήστη είναι μη έγκυρα, τότε ακολουθείται το μονοπάτι Invalid Username / Pswd κατά το οποίο αναμένει την είσοδο νέων χαρακτηριστικών. Εναλλακτικά το Login είναι έγκυρο, ο χρήστης ενημερώνεται και ο Registrar Server οδηγείται στην τερματική κατάσταση.

Στην περίπτωση όπου ο χρήστης εισέρχεται για πρώτη φόρα στο σύστημα, ο Registrar Server ακολουθεί το μονοπάτι καταστάσεων της περίπτωσης First Registration. Ο δεύτερος αναμένει από τον χρήστη το χαρακτηριστικό του username και password. Εν συνεχεία ο Registrar Server ενημερώνεται μέσω της βάσης δεδομένων για την ύπαρξη των στοιχείων αυτών. Στην περίπτωση οπού υπάρχει ήδη χρήστης με αυτά τα στοιχεία καταχωρημένα, τότε ο Registrar Server αναμένει την είσοδο νέων στοιχείων. Εναλλακτικά τα στοιχεία θεωρούνται έγκυρα και καταχωρούνται, ενώ ο Registrar Server οδηγείται στην τερματική κατάσταση.