Ανάπτυξη Κατανεμημένης Εφαρμογής Διασύνδεσης Ετερογενών και Απομακρυσμένων Πειραματικών Υποδομών στο Περιβάλλον *Google Web Toolkit*

Δημήτρης Μπούσης

Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής

Πανεπιστήμιο Πατρών

Επιβλέπων Καθηγητής : Χ.Ζαρολιάγκης

Συνεπιβλέποντες : Ι.Χατζηγιαννάκης , Ε.Θεοδωρίδης

Διπλωματική εργασία για τον τίτλο του

Μηχανικού Ηλεκτρονικών Υπολογιστών και Πληροφορικής

Σεπτέμβριος 2011

Ευχαριστίες

Για την βοήθεια και συνεργασία, θα ήθελα να ευχαριστήσω τους

- **Χρήστο Ζαρολιάγκη**, Καθηγητής Τμήματος Μηχανικών Η/Υ & Πληροφορικής , Πανεπιστήμιο Πατρών.
- **Ιωάννη Χατζηγιαννάκη**, Διευθυντής στην Ερευνητική Μονάδα 1 του Ε.Α.Ι.Τ.Υ & Ακαδημαϊκός Συνεργάτης του Τμήματος Μηχανικών Η/Υ & Πληροφορικής, Πανεπιστήμιο Πατρών.
- **Ιωάννη Γάκο**, Συμφοιτητής & Συνεργάτης στην Ερευνητική Μονάδα 1 του Ε.Α.Ι.Τ.Υ, Πανεπιστήμιο Πατρών.
- **Δημήρης Αμαξιλάτης**, Συμφοιτητής & Συνεργάτης στην Ερευνητική Μονάδα 1 του Ε.Α.Ι.Τ.Υ, Πανεπιστήμιο Πατρών.
- **Ευάγγελο Θεοδωρίδη**, Ερευνητής στην Ερευνητική Μονάδα 1 του Ε.Α.Ι.Τ.Υ & Ακαδημαϊκός Συνεργάτης του Τμήματος Μηχανικών Η/Υ & Πληροφορικής , Πανεπιστήμιο Πατρών.
- **Χρήστο Κονίνη**, Ερευνητής στην Ερευνητική Μονάδα 1 του Ε.Α.Ι.Τ.Υ & Ακαδημαϊκός Συνεργάτης του Τμήματος Μηχανικών Η/Υ & Πληροφορικής , Πανεπιστήμιο Πατρών.
- **Sonke Nommensen**, Ερευνητής στο Ινστιτούτο Τηλεματικής του Πανεπιστημίου του Lubeck , Γερμανία
- **Malte Legenhausen**, Ερευνητής στο Ινστιτούτο Τηλεματικής του Πανεπιστημίου του Lubeck , Γερμανία

Περίληψη

Το παρόν κείμενο αποτελεί τεχνική αναφορά της διπλωματικής εργασίας του συγγραφέα για το ακαδημαϊκό έτος 2010-2011.Παρουσιάζεται το έργο *WiseUI*, μια εφαρμογή ιστού η οποία αναπτύχθηκε για την εξυπηρέτηση χρηστών ετερογεννών δικτύων ασύρματων συσκευών οι οποίες φέρουν αισθητήρες (δίκτυο αισθητήρων) και χρησιμοποιούν ως υποδομή λογισμικό το οποίο αναπτύχθηκε για την πλατφόρμα *WISEBED*. Σε αυτή την αναφορά θα παρουσιαστούν οι στόχοι και προκλήσεις του έργου, οι τεχνολογίες και τα εργαλεία που χρησιμοποιήθηκαν με κύρια αναφορά στο *Google Web Toolkit (GWT)*, οι υπηρεσίες που αναπτύχθηκαν και οι μελλοντικές κατευθύνσεις του έργου.

Περιεχόμενα

Κεφάλαιο 1

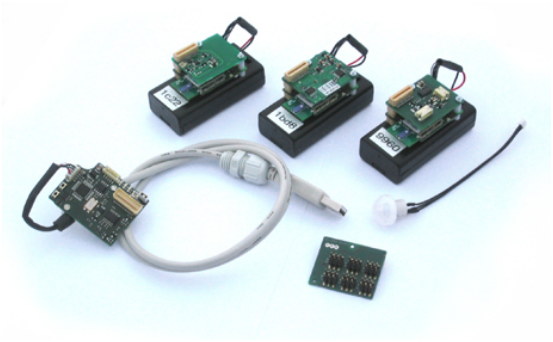
Εισαγωγή

1.1 Κίνητρο και σημασία

Τα Ασύρματα Δίκτυα Αισθητήρων [73] (ΑΔΑ - *Wireless Sensor Network , WSN*) είναι ασύρματα δίκτυα αυτόνομων συσκευών οι οποίες φέρουν αισθητήρες ώστε να παρακολοθούν περιβαλλοντικά μεγέθη. Ενώ αρχικά αναπτύχθηκαν με αμυντικό σκοπό, σύντομα βρήκαν εφαρμογές και στο βιομηχανικό χώρο.

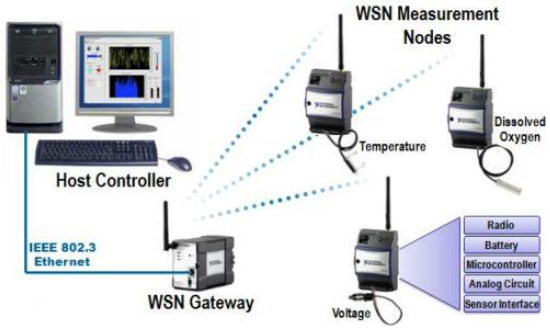
Οι συσκεύες αισθητήρων [57] (κόμβοι - *Nodes*) φέρουν ένα ή περισσότερους αισθητήρες. Μια τυπική αρχιτεκτονική ενός κόμβου αποτελείται από ένα σύστημα πομπού/δέκτη, ένα μικροελεγκτή, ένα σύστημα διεπαφής του μικροελεγκτή με τους αισθητήρες και μια μπαταρία ως πηγή ενέργειας. Το μέγεθος ενός κόμβου μπορεί να είναι τόσο μεγάλο όσο ένα σύγχρονο κινητό τηλέφωνο αλλά και αρκετά μικρό όσο το κεφάλι μιας καρφίτσας. Το κόστος στην αγορά ενός κόμβου είναι και αυτό μεταβλητό, από ένα ως και αρκετές εκατοντάδες ευρώ. Το μέγεθος και το κόστος καθορίζονται από τα αποτελέσματα που επιθυμεί ο σχεδιαστής, οι οποίες εξαρτούνται από την υπολογιστική ισχύ, ταχύτητα τηλεπικοινωνίας και κατανάλωση ενέργειας της συσκευής. Ένα άλλο απαραίτητο συστατικό στη λειτουργία ενός κόμβου είναι και το λογισμικό που εκτελείται στον μικροελεγκτή του. Το λογισμικό αυτό μπορεί να χρησιμοποιηθεί ώστε να παρέχεται λειτουργικότητα όπως δρομολόγηση, κρυπτογράφηση, μετρικές δικτύωσης, κ.α.

Τα ΑΔΑ αποτελούνται ως επι το πλείστον από ασύρματες μη-κατευθυνόμενες συνδέσεις μεταξύ των κόμβων τους. Πέρα από την συλλογή των δεδομένων από τους αισθητήρες, σκοπός τους είναι η διάδοση των δεδομένων σε ένα κύριο κόμβο-πύλη (*Gateway Node*) συνήθως με συνεργατικό τρόπο, όπως δρομολόγηση ή μέθοδος-πλημμύρα. Συχνά, ο κόμβος-πύλη είναι αυτός που επικοινωνεί με ένα υπολογιστικό σύστημα το οποίο επεξεργάζεται, διαχειρίζει η αποθηκεύει τα δεδομένα που συλλέγει ένα ΑΔΑ. Τα ΑΔΑ, όπως και τα περισσότερα ασύρματα δίκτυα, εμφανίζουν ιδιαίτερη ευελιξία όσον αφορά την τοπολογία του δικτύου. Μπορεί να είναι μια αρκετά απλή τοπολογία, π.χ τοπολογία αστέρα, αλλά και ένα πιο σύνθετο πλέγμα συνδέσεων μεταξύ των κόμβων. Ένα ΑΔΑ, ανάλογα με τον αριθμό των συμμετέχοντων κόμβων, μπορεί να αποτελείται από ένα αριθμό μικρότερων ΑΔΑ τα οποία είναι



Εικόνα 1.1: Κόμβοι - Συσκευές με αισθητήρες

ασύνδετα μεταξύ τους. Αυτή η αρχιτεκτονική του δικτύου παρουσιάζεται σε δίκτυα με ιδιαίτερα μεγάλο αριθμό κόμβων. Οι κόμβοι-πύλες σχηματίζουν ένα δίκτυο μεταξύ τους ή χρησιμοποιούν το Διαδίκτυο ώστε να συγκεντρωθούν οι μετρήσεις όλων των κόμβων του ευρήτερου ΑΔΑ.



Εικόνα 1.2: Ασύρματο Δίκτυο Αισθητήρων

Για τις επιστήμες των υπολογιστών και των τηλεπικοινωνιών, τα ΑΔΑ είναι μια δραστήρια περιοχή ερευνών με αρκετά συνέδρια ανά τον κόσμο να διεξάγωνται ετησίως. Αποτέλεσμα της έντονης ερευνητικής δραστηριότητας είναι τα πρότυπα δικτύωσης χαμηλών επιπέδων από την IEEE και των υψηλών επιπέδων από την IETF. Όσον αφορά το υλικό, κύριος στόχος της έρευνας είναι η σχεδίαση και παραγωγή χαμηλού κόστους και μεγέθους συσκευών.Τέλος, το λογισμικό των ΑΔΑ είναι ο τομέας-πρόκληση για τους κατασκευαστές καθώς ως κύριοι σχεδιαστικοί παράγωντες είναι η μεγιστοποίηση του χρόνου ζωής των συσκευών, αποτελεσματικότητα και ανοχή στα σφάλματα και αυτο-προσαρμογή σε αλλαγές του δικτύου και του περιβάλλοντος. Έξυπνοι αλγόριθμοι, πρωτόκολλα, λειτουργικά συστήματα και εργαλεία αναπτύσσονται ειδικά για αυτές τις συσκευές.

Μερικές εφαρμογές των ΑΔΑ είναι οι εξής

- **Παρακολούθηση περιοχής**. Μια περιοχή παρακολουθείται από ΑΔΑ για την εμφάνιση διαφόρων φαινομένων. Π.χ εντοπισμός εισβολέων σε μια περιοχή, χαρτογράφηση εδάφους πλούσιο σε αέρια και πετρέλαιο μέσω μετρήσεων της θερμοκρασία και της πίεσης, περιοχές στάθμευσης με την παρουσία/απουσία οχημάτων. Συνήθως στις περιοχές που αναφέρουμε και στη συνέχεια υπάρχει

χαρακτηριστική έλλειψη καλωδίωσης ώστε να υποστηριχθούν καλωδιωμένα δίκτυα αισθητήρων.

- **Μέτρηση μόλυνσης του αέρα**. Σε μεγάλες πόλεις έχουν τοποθετηθεί ΑΔΑ ώστε να γίνεται καταμέτρηση μόλυνσης σε περιοχές με κυκλοφοριακό ή βιομηχανικές περιοχές.
- **Εντοπισμός πυρκαγιών σε δασικές εκτάσεις**. ΑΔΑ μπορούν να τοποθετηθούν σε δάση ώστε να γίνεται εντοπισμός της πυρκαγίας. Αισθητήρες που καταγράφουν την θερμοκρασία, υγρασία και τα αέρια μπορούν να δώσουν το στίγμα μιας πυρκαγιάς. Η Πυροσβεστική Υπηρεσία συχνά αναφέρει το πόσο σημαντικό είναι η έγκαιρος συναγερμός πυρκαγιάς από αυτά τα συστήματα. Επιπλέον τα ΑΔΑ μπορούν να δώσουν στοιχεία για το πως αναπτύσσεται μια πυρκαγιά στην απειλούμενη έκταση.
- **Επίβλεψη θερμοκηπείων**. Συστήματα με ΑΔΑ μπορούν να τοποθετηθούν σε χώρους όπως τα θερμοκήπεια ώστε να παρατηρούν τα αέρια του χώρου. Όταν παρουσιάζονται αλλαγές μπορούν αυτόματα να ειδοποιήσουνε τον υπεύθυνο του θερμοκηπείου ή να ενεργοποιηθεί το κατάλληλο αυτόματο σύστημα συντήρησης.
- **Εντοπισμός κατολισθήσεων**. Μικροί κόμβοι που σχηματίζουν ένα ΑΔΑ μπορούν να τοποθετηθούν σε δυσπρόσιτες περιοχές και να εντοπίσουν κινητικότητα στο έδαφος και άλλες ενδείξεις κατολίσθησης αρκετά πριν συμβεί.
- **Επίβλεψη κατάστασης μηχανημάτων**. Αισθητήρες μπορούν να τοποθετηθούν σε βιομηχανικά μηχανήματα και να παρακολουθούν την κατάσταση τους, συντονίζοντας έτσι την λειτουργία της βιομηχανικής μονάδος.
- **Επίβλεψη υδάτων και λημμάτων**. Η τοποθέτηση ΑΔΑ σε υδάτινες περιοχές μπορούν να δώσουν χρήσιμα στοιχεία για αυτές.
- **Εφαρμογές στην γεωργία**. Αυτόματα συστήματα μπορούν να τοποθετηθούν σε αντλίες και δεξαμενές ώστε να γίνει άδρευση νερού αποτελεσματικά και χωρίς σπατάλες. Επιπλέον, τα ΑΔΑ απαλλάσουν τον καλλιεργητή από τη συντήρηση ενός καλωδιωμένου διαδικτύου σε γεωργικές εκτάσεις.
- **Παρακολούθηση κατασκευών**. Τα ΑΔΑ μπορούν να χρησιμοποιηθούν για την παρακολούθηση κατασκευών όπως κτίρια, γέφυρες, σήραγγες. Έτσι μπορεί να γίνει ημερήσια λεπτομερής καταγραφή της κυκλοφορίας και της χρήσης των κατασκευών χωρίς να γίνονται επι τόπου επισκέψεις από τους μηχανικούς της κατασκευής.

1.2 Στόχος

Κύρια χαρακτηριστικά των ΑΔΑ είναι τα εξής :

- **Κατανάλωση ισχύος από μπαταρία ή άλλο τρόπο συλλογής ενέργειας**. Η έλλειψη καλωδίωσης το επιβάλλει αυτό. Μια διαδεδομένη μέθοδος συλλογής ενέργειας είναι η εκμετάλευση της ηλιακής ακτινοβολίας.

- **Ανοχή στις βλάβες των κόμβων**. Υπάρχουν πολλές αιτίες που οδηγούν στην απενεργοποίηση ενός κόμβου. Ένα ΑΔΑ πρέπει να μπορεί να ανταπεξέλθει στην αποστολή του ακόμα και χωρίς τους απενεργοποιημένους κόμβους.

- **Ετερογένεια των κόμβων**. Το σύνολο των κόμβων χαρακτηρίζεται από διαφορετικότητα, όχι μόνο ως προς τον τύπο των αισθητήρων που φέρουν, αλλά και στην αρχιτεκτονική τους και στις πολιτικές επικοινωνίας.

- **Κινητικότητα των κόμβων**. Οι κόμβοι ενδέχεται να μετακινούνται κατά την λειτουργία των ΑΔΑ , π.χ αισθητήρας σε βιομηχανικό τροχό.

- **Αποτυχίες στην επικοινωνία**. Παρεμβολλές, φυσικά εμπόδια μπορούν να παρακωλύσουν την επικοινωνία μεταξύ δύο κόμβων σε ένα ΑΔΑ.

- **Δυναμική τοπολογία του δικτύου**. Από αποτυχίες κόμβων/συνδέσμων το δίκτυο συχνά χρειάζεται να επαναδιαμορφωθεί έτσι η τοπολογία του αλλάζει πολύ συχνά.

- **Κλιμακωσιμότητα**. Το δίκτυο μπορεί να λειτουργήσει το ίδιο αποδοτικά με αρκετά μικρο και αρκετά μεγάλο αριθμό κόμβων.

- **Λειτουργία σε αντίξοες συνθήκες**. Τα ΑΔΑ συχνά καλούνται να τοποθετηθούν και να λειτουργούν σε περιβάλλοντα όπου ο άνθρωπος έχει δύσκολη πρόσβαση λόγω των ιδιαίτερων συνθηκών τους. π.χ τροπικά δάση με καταρακτώδη βροχή.

- **Λειτουργία δίχως επίβλεψη**. Τα ΑΔΑ δεν χρεάζονται κάποιο ιδιαίτερα χειρισμό, καθώς σπανίως έχουν την ανάγκη ρύθμισης και ελέγχου κατά την λειτουργία τους

- **Ο χρόνος χρήσης των κόμβων είναι σημαντικός πόρος**. Ένα ΑΔΑ αποτελεί κοινόχρηστο πόρο για τους χρήστες του. Η διαμοίραση πρέπει να γίνεται βάσει δίκαιης πολτικής που να μην παραβιάζεται

Τα παραπάνω χαρακτηριστικά μας αποκαλύπτουν την περίπλοκη πλευρά των ΑΔΑ. Ενώ η ανάπτυξη και η λειτουργία του δικτύου μπορεί να γίνει σχετικά εύκολα, η χρήση ενός ΑΔΑ είναι μια πολύπλοκη διαδικασία. Η ετερογένεια και των κόμβων, η δυναμικότητα του δικτύου και τυχόν αποτυχίες συνδέσμων ή κόμβων καλούν οι χρήστες τους να έχουν εξειδικεύμενες γνώσεις δικτύωσης και να γνωρίζουν λεπτομέρειες όπως για παράδειγμα αρχιτεκτονική των κόμβων. Οι χρήστες των ΑΔΑ δεν είναι απαραίτητα επιστήμονες της πληροφορικής ή γνώστες των λεπτομερειών του δικτύου. Πολιτικοί μηχανικοί, βιολόγοι, ερευνητές και ανειδίκευτο προσωπικό είναι συνήθως οι τελικοί χρήστες των ΑΔΑ καθώς οι αισθητήρες του δικτύου δίνουν χρήσιμες πληροφορίες για να επιτελέσουν το έργο τους. Καλείται λοιπόν η ανάγκη ανάπτυξης υποδομών που βοηθούν τον χρήστη του ΑΔΑ να εκμεταλευτεί τις δυνατότητες τους.

Είναι λογικό οι σχεδιαστές ενός ΑΔΑ να επιθυμούν να ελέγχουν όσο μπορούν τις ιδιαιτερότητες του δικτύου. Μια πρόταση είναι μια πλατφόρμα με επιπλέον υπολογιστικά συστήματα και λογισμικό ώστε να λειτουργεί ως πειραματική υποδομή (ΠΥ -- *Testbed*).Το ελεγχόμενο αυτό περιβάλλον μπορεί

να αναπτυχθεί με τη χρήση μερικών υπολογιστών που λειτουργούν ως εξυπηρετητές των χρηστών και ειδικών υπηρεσιών. Οι υπολογιστές αυτοί επικοινωνούν απευθείας με τους κόμβους-πύλες ώστε έτσι οι υπεύθυνοι του δικτύου να ελέγχουν τους κόμβους του δικτύου και οι χρήστες να διαχειρίζονται τα δεδομένα που δίνουν οι αισθητήρες του δικτύου. Με το Διαδικτύου ο έλεγχος μπορεί να γίνει και απομακρυσμένα, χωρίς την φυσική παρουσία του διαχειριστή/χρήστη στο χώρο που στεγάζεται το δίκτυο. Γενικά κάποιος μπορεί να δεί την ΠΥ ως ένα αφαιρετικό επίπεδο το οποίο αποκρύπτει λεπτομέρειες της δομής και αρχιτεκτονικής του ΑΔΑ και των κόμβων του. Περισσότερα για τις ΠΥ και λεπτομέρειες για την πλατφόρμα *WISEBED* θα συζητηθούν στο δεύτερο κεφάλαιο.

Οι ΠΥ παρέχουν όπως αναφέρθηκε και ένα σύνολο λογισμικού, το οποίο απαρτίζεται από εξυπηρετητές, βάσεις δεδομένων, ελεγκτές και εργαλεία. Ένα από αυτά τα εργαλεία είναι και ένα που απευθύνεται αποκλειστικά στον τελικό χρήστη του ΑΔΑ μέσω των ΠΥ και εστιάζει στην διεξαγωγή πειραμάτων στους κόμβους ενός ΑΔΑ. Σκοπός της διπλωματικής αυτής εργασίας είναι η ανάπτυξη αυτού του εργαλείου και συγκεκριμένα μιας διαδικτυακής εφαρμογής ιστού που επικοινωνεί με μια ΠΥ ώστε ο χρήστης του να μπορεί εύκολα να εξάγει δεδομένα και πληροφορία από τους αισθητήρες των ΑΔΑ. Ονομάζουμε την εφαρμογή *WiseUI* και αναπτύχθηκε το ακαδημαϊκό έτος 2010-2011 για τις ανάγκες της πλατφόρμας *WISEBED*.Να διευκρινστεί πως το *WiseUI* αποτελεί προτίστως εργαλείο μιας ΠΥ και όχι ενός ΑΔΑ. Το *WiseUI* δεν έχει άμεση σχέση με ένα ΑΔΑ καθώς δεν έχει γνώση των λεπτομερειών υλοποίησης και το σχεδιασμού του. Οποιαδήποτε πληροφορία παρέχεται σχετικά με το ΑΔΑ στην εφαρμογή γίνεται μέσω των υπηρεσιών της ΠΥ. Το έργο *WiseUI* έχει στόχο να γίνει το εργαλείο του πειραματιστή-χρήστη των πόρων μιας ΠΥ που ανήκει στο δίκτυο του *WISEBED*. Τα προϋπάρχοντα εργαλεία είναι εφαρμογές κονσόλας τα οποία ενώ παρέχουν την κύρια λειτουργικότητα δεν είναι ούτε εύχρηστα ούτε επεκτέσιμα ενώ για να καλύψουν πιο σύνθετες ανάγκες χρειάζεται να επαναπρογραματιστούν από χρήστες με γνώσεις προγραμματισμού και λεπτομεριών πάνω στις υπηρεσίες που παρέχει η ΠΥ.

1.3 Συνεισφορά

Η εφαρμογή που αναπτύχθηκε έχει ως σκοπό να διευκολύνει και να παρέχει μερικές επιπλέον υπηρεσίες στον χρήστη. Είναι μια σύγχρονη εφαρμογή ιστού και έτσι είναι προσβάσιμη παγκοσμίως μέσω ένος περιηγητή ιστού. Αντίθετα με μια εφαρμογή γραμμής εντολών η οποία αποτελούσε αρχικό εργαλείο για τον χρήστη, στην περίπτωση του *WiseUI* υπάρχει γραφική διεπαφή χρήστη χρησιμοποιώντας τεχνολογίες όπως το υπερκείμενο και σεναριακές γλώσσες περιηγητών όπως η γνωστή γλώσσα *JavaScript*. Ο συγγραφέας της αναφοράς, ασχολήθηκε κυρίως με την παροχή του πειραματισμού του χρήστη με την ΠΥ, υλοποιώντας τις απαραίτητες λειτουργίες μαζί με την παρουσίαση τους στη γραφική διεπαφή της εφαρμογής.

Η κύρια λειτουργικότητα της εφαρμογής είναι να προωθήσει με απλό και κατανοητό τρόπο τις λειτουργίες της ΠΥ στον τελικό χρήστη. Το *WiseUI* :

- Παρέχει χρήσιμες πληροφορίες για την υπάρχουσα ΑΔΑ υποδομή.
- Δυνατότητα για τον χρήστη να κάνει κράτησεις κόμβους του δικτύου ώστε στη συνέχεια να τους χρησιμοποιήσει για πειράματα.

- Προσφέρει πληροφορίες και επιτρέπει στον χρήστη τη διαχείριση των κρατήσεων του και να συντονίζεται με τις κρατήσεις άλλων χρηστών.
- Αποθήκευση λογισμικού χαμηλού επιπέδου (προγράμματα λειτουργιας κόμβων) σε βάση δεδομένων. Το λογισμικό αυτό αποτελεί το βασικό συστατικό για την διεξαγωγή πειραμάτων.
- Ζωντανή τροφοδοσίατων δεδομένων που παρέχουν οι αισθητήρες του πειράματος στους χρήστες της εφαρμογής.
- Έλεγχος πειράματος στους κόμβους που συμμετέχουν.
- Παροχή δομημένης αναφοράς (βλπ *WiseML* στο δεύτερο κεφάλαιο της αναφοράς) της εκτέλεσης του πειράματος.

Το λογισμικό διατίθεται υπό την άδεια ελεύθερο λογισμικού *Apache License 2.0* ώστε μελλοντικά ο αριθμός των προγραμματιστών να μεγαλώσει από την κοινότητα του ελεύθερουκ αι ανοιχτού λογισμικού. Το κείμενο της συγκεκριμένης άδειας εμπεριέχεται στον πηγαίο κώδικα της εφαρμογής. Επίσης, το λογισμικό που χρησιμοποιήθηκε για την ανάπτυξη του έργου είναι εξ'ολοκλήρου ελεύθερο λογισμικό η λογισμικό ανοιχτού κώδικα.

1.4 Δομή

Το παρών κείμενο αποτελεί τεχνική αναφορά του έργου *WiseUI*. Σε αυτό το σημείο θα κλείσουμε το εισαγωγικό κεφάλαιο της αναφοράς δίνωντας μια σύντομη περιγραφή του υπόλοιπου κειμένο.

Στο επόμενο κεφάλαιο θα γίνει σύντομη περιγραφή του περιβάλλοντος *WISEBED* παρουσιάζοντας τις υποδομές που αναπτύχθηκαν από τη συνεργασία ευρωπαϊκών ακαδημαικών ιδρυμάτων.

Στο τρίτο κεφάλαιο θα συζητηθούν οι προκλησεις και ορισμένα τεχνικά θέματα σχετικά με την ανάπτυξη του λογισμικού *WiseUI* τα οποία εμφανίζονται συχνά στην ανάπτυξη περίπλοκών δικτυακών εφαρμογών.

Στο τέταρτο περιγράφονται τεχνολογίες, εργαλεία, πλαίσια/πακέτα λογισμικού και βιβλιοθήκες που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής. Θα δωθεί έμφαση στο *Google Web Toolkit* (*GWT*) η οποία χρησιμοποίηθηκε ως κύρια τεχνολογία.

Στο πέμπτο κεφάλαιο θα αναλυθεί εκτενώς ο σχεδιασμός και η υλοποίηση των υπηρεσιών του *WiseUI*. Θα συζητηθεί η αρχιτεκτονική της εφαρμογής, η χρήση των υπηρεσιών του *WISEBED* στην εφαρμογή. Επίσης θα σταθούμε σε ορισμένες τεχνικές λεπτομέριες σχετικές με την ανάπτυξη της εφαρμογής.

Τέλος στο έκτο κεφάλαιο παρουσιάζεται μια εκτέλεση της εφαρμογής με τη χρήση εικόνων,ενώ στο τελευταίο θα συζηθούν συμπεράσματα και μελλοντικές κατευθύνσεις της εφαρμογής. Στο τέλος της αναφοράς καταγράφεται η βιβλιογραφία που χρησιμοποιήθηκε κατα τη διάρκεια της εκπόνησης της διπλωματικής εργασίας.

Κεφάλαιο 2

Η πλατφόρμα WISEBED - Προγραμματιστική Διασύνδεση με Διεπαφές & Βιβλιοθηκές

2.1 Η πλατφόρμα WISEBED

Η πλατφόρμα *WISEBED* [74] [18] αποτελεί σύμπραξη εννέα ερευνητικών και ακαδημαϊκών ινστιτούτων της Ευρώπης με σκοπό την έρευνα στο χώρο των ΑΔΑ. Οι εργασίες ξεκίνησαν τον Ιούνιο του 2008 και τερματίστηκαν τον Μάιο του 2011. Η χρηματοδότηση της ανάπτυξης προήλθε από την Ευρωπαϊκή Επιτροπή στο πλαίσιο του προγράμματος *Information Communication Technologies* μέλος του 7ού πλαισίου με αριθμό έργου 224460.

Εικόνα 2.1: Λογότυπο της πλατφόρμας *WISEBED*

Στόχος της πλατφόρμας *WISEBED* είναι η παροχή μιας υποδομής πολλών επιπέδων αποτελούμενη από μια ΠΥ αυτόνομων ΑΔΑ κατανεμημένες γεωγραφικά στην Ευρώπη. Η συγκεκριμένη υποδομή παρέχει στους ερευνητές ένα συνδυασμό υλικού, λογισμικού , αλγορίθμων και δεδομένων στηριζόμενη σε ένα μεγάλο διασυνδεμένο δίκτυο αποτελούμενο από πολλά μικρότερα ΑΔΑ. Οι ερευνητές πλέον διαθέτουν μια υποδομή που μπορεί να δώσει ευελιξία στο έργο τους κρύβωντας τις λεπτομέριες κατασκευής και διατήρησης ενός μεγάλου δικτύου ΑΔΑ. Μερικά από τα συμμετέχοντα ερευνητικά ιδρύματα είναι τα Ε.Α.Ι.Τ.Υ του Πανεπιστημίου Πατρών , Ινστιτουτό Τηλεματικής του Πανεπιστημίου του Lubeck,

Πανεπιστημίου του Lancaster κ.α. Σε πρώτη φάση ο στόχος ήταν η διαμόρφωση ενός μεγάλου δικτύου αισθητήρων (περί τους 2000). Για την ανάπτυξη του έργου σχηματίστηκαν 6 πακέτα εργασίας με καθένα να αναλαμβάνει τους ευρύτερους τομείς του έργου όπως υλικό, λογισμικό, αλγόριθμοι, δεδομένα, πληροφόρηση, διαχείρηση & αξιολόγηση. Το έργο *WiseUI* ανήκει στο δεύτερο πακέτο εργασίας το οποίο είναι αυτό του λογισμικού.

Λόγω της κατανεμημένης φύσης της εκμετάλλευσης των πειραματικών υποδομών, η συνεργασία και ανταλλαγή ιδεών τόσο σε θεωρητικό όσο και σε πρακτικό επίπεδο θα είναι πολύ πιο αποδοτική, προσφέροντας έτσι ευέλικτη εφαρμογή των θεωρητικών αποτελεσμάτων σε αλγορίθμους, μηχανισμούς και πρωτόκολλα για την κατασκευή λογισμικού. Απότερος στόχος του έργου είναι η κατασκευή μιας ενοποιημένης ΠΥ (*Federated Testbed*) η οποία εκμεταλεύεται την υποδομή κάθε φορέα που προσέρχεται στο έργο. Οι ερευνητές των ιδρυμάτων μπορούν να αναπτύσουν εφαρμογές σε υψηλό επίπεδο βασιζόμενοι στα δεδομένα που προσφέρουν οι συνεργαζόμενες ΠΥ, κατασκευάσουν πρωτοποριακούς αλγορίθμους & πρωτόκολλα για τα ΑΔΑ υποσυστήματα του έργου και τέλος να προσφέρουν τον κατάλληλο εξόπλισμο τους στο έργο για συνεργασία.

Η γενική αρχιτεκτονική της πλατφόρμας *WISEBED* συνοψίζεται ως εξής :

- Τοπικά δίκτυα συσκευών αισθητήρων δομημένο έτσι ώστε να αποτελεί μιας έγκυρης ΠΥ στα πρότυπα που ορίζει το έργο.
- Υπολογιστές εξυπηρετητές και κόμβοι-πύλες που αναλαμβάνουν το έργο επικοινωνίας των χρηστών με τις συσκευές. Οι εξυπηρετητές αυτοί ελέγχουν το μεγαλύτερο μέρος μιας ΠΥ που είναι εγκατεστημένοι.
- Ένα υψηλού επιπέδου δίκτυο όπου χρήστες και εφαρμογές χρησιμοποιούν πόρους του έργου. Το δίκτυο είναι προσβάσιμο μέσω Διαδικτύου.

Το υλικό που συμμετέχει στη πλατφόρμα *WISEBED* έχει οριστεί από τις αρχικές συναντήσεις των μελών για την ανάπτυξης της. Κάθε φορέας οφείλει να το αποκτήσει και να εγκαταστίσει το υλικό που θα συνιστά την υλική υποδομή της ΠΥ του φορέα. Το δίκτυο των ΠΥ είναι γεωγραφικά κατανεμημένο στην Ευρώπη σε εγκαταστάσεις κάθε φορέα. Είναι σημαντικό το υλικό που χρησιμοποιείται στο έργο να είναι τελευταίας τεχνολογίας και να χρησιμοποιεί πρότυπα και πλατφόρμες που ήδη χρησιμοποιούνται σε άλλες υποδομές.

Το σύνολο του λογισμικού που συμμετέχει στο έργο μπορεί να χωριστει σε τρία μεγάλα, πλήρως διαχωρίσημα συστατικά. Ένα απο αυτά είναι ένα σύνολο εργαλείων ανάπτυξης λογισμικού για τα προγράμματα τα οποία εκτελούνται σε όλα τα επίπεδα του ΑΔΑ και των κόμβων του. Άλλο ένα απο αυτά τα συστατικά είναι ένα σύστημα εξομοίωσης ΑΔΑ και κόμβων έτσι ώστε να δοκιμάζεται λογισμικό σε ένα εικονικό σύστημα πριν στηθεί το πραγματικό. Τέλος, στο κομμάτι που θα σταθούμε στη συνέχεια του κεφαλαίου είναι το σύστημα διαχείρησης μιας ΠΥ. Η λειτουργικότητα του επικεντρώνεται στην χρήση της ΠΥ και των πόρων του από τους χρήστες π.χ εξουσιοδότηση χρηστών,δέσμευση κόμβων και εκτέλεση πειραμάτων στους δεσμευμένους κόμβους.

Στη συνέχεια του κεφαλαίου θα επικεντρωθούμε σε λογισμικό και υπηρεσίες που προσφέρει το *WISEBED* απαραίτητες για την ανάπτυξη της εφαρμογής *WiseUI* ενώ κατόπιν θα αναφερθούμε και

γλώσσα σήμανσης *WiseML* .Στο τέλος του κεφαλαίου γίνεται σύντομη αναφορά στη βιβλιοθήκη *Wiselib* μια βιλιοθήκη ανάπτυξης αλγορίθμων η οποία αποτελεί τον θεμέλιο λίθο για την ανάπτυξη πειραμάτων από τους ερευνητές που χρησιμοποιούν τις υποδομές του *WISEBED*.

2.2 Το σύστημα Testbed Runtime (TR)

Το *Testbed Runtime (TR)* [69] αποτελείται από ένα σύνολο πακέτων λογισμικού και συνιστά το κυριότερο συστατικό για την εγκατάσταση μιας ΠΥ ώστε να πληρεί τις προϋποθέσεις του *WISEBED*. Ο σχεδιασμός αυτού το λογισμικού έχει ως κύριους αξονες την λειτουργικότητα και την τμηματοποίση του σε διάφορα μικρότερα πακέτα λογισμικού που συνεργάζονται αρμονικά. Κατα τις πρώτες συζητήσεις στο *WISEBED* είχε σχεδιαστεί η βασική διεπαφή των υπηρεσιών που θα αναπτυχθούν και το *TR* έχει υλοποίησει αυτόν τον σχεδιασμό. Αναπτύχθηκε από τον ερευνητή Daniel Bimschas του Ινστιτούτου Τηλεαματικής του Πανεπιστημίου του Lubeck. Η γλώσσα υλοποίησης είναι η *Java* [36].

Εικόνα 2.2: Λογότυπο του έργου *Testbed Runtime*

Οι κύριες λειτουργίες του *TR* είναι :

- Συγχρονισμός των συσκευών που συμμετέχουν στην ΠΥ και στα κατώτερα ΑΔΑ.
- Συλλογή δεδομένων από τους αισθητήρες των ΑΔΑ.
- Παρέχει υπηρεσίες διαχείρησης ελέγχου των συσυκευών και των πειραματικών εκτελέσεων ως προς τους χρήστες του ΑΔΑ.
- Παρέχει ένα σύστημα ταυτοποίησης των χρηστών που ανήκουν σε μια ΠΥ.
- Σύστημα επικοινωνίας μεταξύ ΠΥ με χρήση υπηρεσιών ιστού.
- Εφόσον έχει ενεργοποιηθεί η λειτουργία των ενοποιημένων ΠΥ, το *TR* μπορεί να επιτρέψει στην χρήστη εγκατάστασης μιας εικονικής σύνδεσης μεταξύ ΠΥ, οι οποίες βρίσκονται σε διαφορετικό ίδρυμα.

Οι υλοποιήσεις των υπηρεσιών που χρησιμοποιεί η εφαρμογή *WiseUI* περιγράφονται στη συνέχεια.

2.2.1 Η διεπαφή Sensor Network Authentication and Authorization API (SNAA API)

Το *SNAA API* [67] είναι μια από τις διεπαφές των υπηρεσιών του *TR* για τους χρήστες μιας ΠΥ. Σκοπός του είναι η παροχή υπηρεσιών ταυτοποίησης και εξουσιοδότησης των χρηστών στην ΠΥ και

στο ΑΔΑ που υπηρετεί. Ο σχεδιασμός του *SNAA* επιτρέπει στους διαχειριστές μιας ΠΥ να έχουν επιλογή ως προς το κύριο σύστημα ταυτοποίησης και εξουσιοδότησης. Τα υποστηριζόμενα συστήματα είναι το *Shibboleth*[60] και *Java Authentication & Authorization Services - JAAS*[38] υλοποιήσεις του πρωτόκολου *Kerberos* [43], της διαχειρισης χρηστών των *UNIX*, *Windows*, *Solaris* και το σύστημα *PAM* [51]. Ένας από τους λόγους για τον οποίο υποστηρίζονται αρκετά τέτοια συστήματα είναι ότι η διεπαφή *SNAA API* πρέπει να υποστηρίζει πολλές επιλογές λόγω της απαραίτητης γενικότητας. Επίσης, οι πάροχοι των ΠΥ πλέον έχουν την επιλογή να τοποθετήσουν οποιοδήποτε σύστημα καλύπτει τις ανάγκες τους και την υποδομή τους. Τέλος, με τα πολλαπλά συστήματα είναι ευκολότερη να επιτευχθεί η ενοποίηση των ΠΥ σε μια ενοποιημένη εικονική ΠΥ. Επιπλέον η διεπαφή υποστηρίζει και την αϋθαίρετηϋλοποίηση (*Dummy*) η οποία δίνει πρόσβαση σε οποιοδήποτε χρήση αλλά περιορίστηκε μόνο για δοκιμαστικούς σκοποούς κατα την ανάπτυξη των έργων .

Οι χρήστες ταυτοποιούνται στέλνωντας τα προσωπικά τους στοιχεία, όνομα χρήστη και κωδικό, χρησιμοποιώντας το *SNAA API* στην ΠΥ που θα ήθελαν να συνδεθούν. Το όνομα χρήστη συνήθως συνοδεύεται και με το χαρακτηριστικό όνομα-πρόθεμα (*URN Prefix*) της ΠΥ που θέλουν να συνδεθούν και γράφεται ως εξής : `üsername@urn-prefiẍ`. Το σύστημα απαντάει με ένα ειδικά διαμορφωμένο κλειδί το οποίο ονομάζουμε Μυστικό Κλειδί Ταυτοποίησης (ΜΚΤ - *Secret Authentication Key*. Το ΜΚΤ είναι έτσι διαμορφωμένο ώστε κάθε χρήστης που έχει ταυτοποιηθεί να έχει μια αποκλειστική τιμή του κλειδιού. Επιπλέον για το *SNAA* υποστηρίζονται οι μηχανισμοί των συνόδων (*Sessions*) με τον τρόπο που είναι γνωστοί σε εφαρμογές ιστού. Δηλαδή, το ΜΚΤ ισχύει για ένα περιορισμένο μικρό διάστημα. Μετά το πέρας αυτής της χρονικής περιόδου ο χρήστης δεν είναι πλέον ταυτοποιημένος/εξουσιοδοτημένος και χρειάζεται να επεναταυτοποιηθεί ώστε να ανανεώσει το ΜΚΤ και να συνεχίσει την χρήση του συστήματος.

Παρουσιάζουμε την προγραμματιστική διεπαφή του *SNAA API* γραμμένη σε *Java* με τις παρακάτω μεθόδους :

Κώδικας 2.1: SNAA API

```
List<SecretAuthenticationKey> authenticate(List<AuthenticationTriple> authenticationData);
boolean isAuthorized(List<SecretAuthenticationKey> authenticationData, Action action);
```

Η πρώτη μέθοδος με όνομα `authenticate()` είναι αυτή που ο χρήστης καλει για να ταυτοποιηθεί. Επιστρέφει μια λίστα από έγκυρα αντικείμενα τύπου `Secret␣uthenticationKey` στην περίπτωση που ο χρήστης ταυτοποιηθεί. Η κλάση `AuthenticationTriple` περιέχει την τριάδα των στοιχειών του χρήστη με αυτά να είναι το όνομα χρήστη, ο κωδικός του και το χαρακτηριστικό όνομα-πρόθεμα της ΠΥ που προσπαθεί να ο χρήστης να ταυτοποιηθεί. Η κλάση `SecretAuthenticationKey` περιέχει δύο συμβολοσειρές με την μια να είναι το χαρακτηριστικό όνομα-πρόθεμα της ΠΥ στο οποίο ο χρήστης έχει ταυτοποιηθεί με την κλήση `authenticate()`. Σε περίπτωση σφάλματος κατά την διαδικασία ταυτοποίησης μια κατάλληλη εξαίρεση (*Java Exception*) ενεργοποιείται.

Η μέθοδος με όνομα *isAuthorised()* είναι η μια μέθοδος την οποία καλεί ο χρήστης εφόσον έχει ταυτοποιηθεί (χρησιμοποιεί το ΜΚΤ που έχει ανακτήσει από την μέθοδο `authenticate`) για να ελέγξει αν είναι εξουσιοδοτημένος να προβεί σε μια ενέργεια. Η ενέργεια προς έλεγχο εξουσιοδότησης εκφράζεται με την κλάση της *Java*,`Action`. Η μέθοδος επιστρέφει λογικές τιμές με `true` αν ο χρήστης είναι

εξουσιοδοτημένος να προβεί στην ενέργια `action` και `false` αν ο χρήστης δεν είναι εξουσιοδοτημένος.

Παρατηρούμε επίσης ότι χρησιμοποιείται έντονα η κλάσση της λίστας (`java.util.List`) στις παραπάνω κλήσεις, κάτι που θα δούμε και στη συνέχεια της αναφοράς. Ο λόγος χρήσης της λίστας αναφέρεται στην περίπτωση που ο χρήστης θέλει να ταυτοποιηθεί σε μια ενοποιημένη ΠΥ. Στην περίπτωση της ενοποιημένης ΠΥ , από τις ΠΥ που συμμετέχουνε μια είναι ορισμένη να εκτελεί τα χρέη του ταυτοποιητή. Οι χρήστες συνδέονται στο *TR* αυτής της ΠΥ και κάνουν ταυτοποίηση σε όλα τα υπόλοιπα ΠΥ τοποθετώντας τα χαρακτηριστικά τους στη λίστα. Η ΠΥ φροντίζει με τη σειρά της να κάνει την ταυτοποίηση στα ΠΥ που ορίζει ο χρήστης και να επιστρέψει ΜΚΤ από αυτές στο χρήστη. Ομοίως και στην περίπτωση που ο χρήστης θέλει να ελέγξει αν είναι εξουσιοδοτημένος για μια ενέργεια. Στην περίπτωση όμως αυτή, όλες οι ΠΥ πρέπει να συμφωνήσουνε στο αν ο χρήστης έχει την εξουσιοδότηση που ζητάει. Πιο συχνά, η διεπαφή χρησιμοποιείται για χρήστες σε μια ΠΥ, έτσι οι λίστες θα περιέχουνε μόνο ένα στοιχείο.

Επιπλέον, το *SNAA* παρέχει και μια βοηθητική κλάση με όνομα `SNAAServiceHelper` η οποία παρέχει στον προγραμματιστή ένα στιγμιότυπο της υπηρεσίας για την ΠΥ που επιθυμεί. Το στιγμιότυπο αυτό παρέχει υλοποιημένο το *SNAA API* στο χρήστη ρυθμισμένο σύμφωνα με τις προδιαγραφές του ΠΥ. Κάθε ΠΥ αναλαμβάνει να κοινοποιεί τις υπηρεσίες τις μέσω υπηρεσιών ιστού (*Web Services*)[71] σε μια συγκεκριμένη διεύθυνση ιστού (*Web Service URL*). Για παράδειγμα για την ΠΥ που λειτουργεί στο E.A.I.T.Y της Πάτρας η διεύθυνση είναι `http://hercules.cti.gr:8890/snaa/shib1` και το σύστημα που χρησιμοποιεί είναι το *Shibboleth*. Από αυτή την κλάση μας ενδιαφέρει η παρακάτω μέθοδος :

Κώδικας 2.2: SNAAServiceHelper

```
SNAA getSNAAService(String endpointUrl)
```

Η μέθοδος `getSNAAService()` παίρνει ένα αλφαριθμιτικό (`String`) ως παράμετρο μόνο και αυτή είναι η διεύθυνση στην οποία εκτίθεται η υπηρεσία *SNAA* για μια ΠΥ και επιστρέφει την υλοποίηση της διεπαφής σύμφωνα με τις προδιαγραφές του ΠΥ μέσω ενός στιγμιοτύπου της κλάσσης `SNAA`.

2.2.2 Η διεπαφή Reservation System API (RS API)

Το *RS API* [66] είναι μια διεπαφή των υπηρεσιών του *TR* και επιτρέπει στους χρήστες του να κάνουν και να διαχειρίζονται κρατήσεις των κόμβων που του ΑΔΑ συμμετέχουν στην ΠΥ. Ο σχεδιασμός του *RS* όπως και στο *SNAA* επιτρέπει στους διαχειριστές μιας ΠΥ να έχουν επιλογή ως προς το σύστημα που υλοποιεί την διεπαφή, επιτρέπωντας τους να καλύψουν τις ανάγκες τους με το σύστημα που επιθυμούν. Συνήθως τα συστήματα που καταγράφουν αυτές τις κρατήσεις είναι γνωστές στον χώρο σχεσιακές βάσεις δεδομένων όπως , *MYSQL* [45], *Oracle Database* [49] , *IBM DB2* [33]. Επίσης ως δοκιμαστικές υλοποιήσεις υπάρχει η αϋθαίρετηϋλοποίηση , όπως την περιγράψαμε και στο *SNAA*, αλλά και μια υλοποίηση που αποθηκεύει προσωρινά τις κρατήσεις στην μνήμη του συστήματος. Οι δύο αυτές υλοποιήσεις χρησιμοποιούνται κυρίως κατα την περίοδο ανάπτυξης της ΠΥ για δοκιμαστικούς λόγους και μόνο. Τέλος για την αποθήκευση αλλα και την αναπαράσταση των κρατήσεων χρησιμοποιείται και η διαδικτυακή υπηρεσία ημερολογίου της *Google* με όνομα *Google Calendar Service*[24]. Η χρήση του

Google Calendar Service αποτυπώνει με σαφή τρόπο το πρόγραμμα των κρατήσεων για μια ΠΥ με τη μορφή ενός ημερολογίου, παρέχωντας έτσι στο χρήστη τις χρονικές περιόδους όπου οι κόμβοι ενός ΑΔΑ είναι διαθέσιμοι σε αυτόν για χρήση. Ακόμα η υπηρεσία αυτή μπορεί να αποτελέσει εναλλακτικό τρόπο αποθήκευσης των κρατήσεων χρησιμοποιώντας την υποδομή της *Google* απαλλάσωντας έτσι τους διαχειριστές του ΠΥ το κόστος της διατήρησης των κρατήσεων. Όπως όλες οι υπηρεσίες του *TR*, έτσι και το *RS* αναπτύχθηκε στη γλώσσα προγραμματισμού *Java* και χρησιμοποιεί πρότυπα όπως το *Java Peristence API*[42] για την αποθήκευση δεδομένων σε συστήματα όπως αυτά που περιγράψαμε.

Παρουσιάζουμε την προγραμματιστική διεπαφή του *RS API* γραμμένη σε *Java* με τις παρακάτω μεθόδους :

Κώδικας 2.3: RS API

```
List<SecretReservationKey> makeReservation(List<SecretAuthenticationKey> authenticationData,
     PublicReservationData reservation)
List<PublicReservationData> getReservations(XMLGregorianCalendar from, XMLGregorianCalendar to)
List<ConfidentialReservationData> getReservation(List<SecretReservationKey> secretReservationKey)
void deleteReservation(List<SecretReservationKey> secretReservationKey)
```

Η μέθοδος `makeReservation()` είναι η μέθδος που χρησιμοποιείται για να γίνει η δέσμευση των κόμβων. Δέχεται δύο παραμέτρους, η πρώτη είναι η μια λίστα με αντικείμενα τύπου `Secret□uthenticationKey` ώστε να επιβεβαιωθεί από το σύστημα η εξουσιοδότηση του χρήστη, ενώ η δεύτερη παράμετρος είναι ένα αντικείμενο που περιέχει τα χαρακτηριστικά μιας κράτησης.Τα βασικά χαρακτηριστικά μιας κράτασης είναι ο χρόνος έναρξης της κράτησης, ο χρόνος λήξης της κράτησης και τα μοναδικά ονόματα των κόμβων που συμμετέχουν στην κράτηση. Τα όνομα ενός κόμβου είναι μοναδικός καθώς συνθέτεται από το χαρακτηριστικό όνομα-πρόθεμα (*URN Prefix*) της ΠΥ που ανήκει ο κόμβος καθώς και από ένα τετραψήφιο αριθμό δεκαξαδικής βάσης που συνήθως αντλείται από την δικτυακή διεύθυνση υλικού που διαθέτει (*Network MAC Address*). Επιπλέον χαρακτηριστικά είναι το όνομα του χρήστη που κάνει την κράτηση και το όνομα-πρόθεμα της ΠΥ που έχει καταγραφεί ο χρήστης στο δίκτυο του *WISEBED*. Η μέθοδος `makeReservation()` χρειάζεται αποκλειστικά τα βασικά χαρακτηριστικά της κράτησης που αναφερθήκαμε και είναι με την μορφή ενός στιγμιοτύπου κλάσσης `PublicReservationData`. Η κλήση αυτής της μεθόδου επιστρέφει ένα στιγμιότυπο της κλάσης `SecretReservationKey` το οποίο ονομάζουμε Μυστικό Κλειδί Κράτησης (ΜΚΚ - *Secret Reservation Key*, το οποίο περιέχει τα πεδία που περιγράψαμε παραπάνω για το ΜΚΤ πιθανότητα με διαφορετικές τιμές. Αντίθετα με το ΜΚΤ το ΜΚΚ έχει διάρκεια μέχρι τη λήξη της κράτησης , ενώ το ΜΚΤ έχει καθορισμένη από το σύστημα διάρκεια ζωής.Σε περίπτωση σφάλματος κατά την διαδικασία ταυτοποίησης μια κατάλληλη εξαίρεση (*Java Exception*) ενεργοποιείται. Όπως το ΜΚΤ λειτουργεί ως αναγνωριστικό ενός ταυτοποιημένου χρήστη στο σύστημα, έτσι και το ΜΚΚ λειτουργεί ως ένα αναγνωριστικό της κράτησης για τον χρήστη ώστε να μπορεί να τη διαχειρστεί ο χρήστης και το *RS*.

Στη συνέχεια οι μέθοδοι `getReservations()` και `getReservation()` επιστρέφουν της κρατήσεις που εχουνε γίνει. Η διαφορά τους έγκειται στις παραμέτρους που δέχονται. Η `getReservations()` δύο παραμέτρους της ημερολογιακής κλάσης `XMLGregorianCalendar` και σηματοδοτούν την έναρξη και τη λήξη μιας κράτησης. Οι κρατήσεις που επιστρέφονται , ως αντικείμενα της κλάσης `PublicReservationData`, είναι όσες είναι αποθηκευμένες στο σύστημα *RS* από όλους τους χρήστες δίνωντας έτσι το πρόγραμμα

των κρατήσεων που δεν έχουνε ακόμα λήξει. Η μέθεδος `getReservation()` επιστρέφει τις κρατήσεις που έχουνε γίνει χρησιμοποιώντας το ΜΚΚ που δίνεται ως παράμετρος, έτσι επιστρέφωνται οι κρατήσεις που έχουνε γίνει από ένα χρήστη. Το αντικείμενο της κλάσσης που επιστρέφεται είναι κλάσης `ConfidentialReservationData` η οποία είναι υποκλάση της `PublicReservationData`. Περιέχει επιπλέον εμπιστευτικές πληροφορίες και δεδομένα όπως το όνομα του χρήστη που έκανε την κράτηση.

Τέλος η συνάρτηση `deleteReservation` διαγράφει τις κρατήσεις τις οποίες δίνωνται ως παράμετροι τα ΜΚΚ τους. Οι διεγραμμένες κρατήσεις δεν μπορούν να ανακτηθούν. Όλες οι παραπάνω μέθοδοι μπορούν παρουσιάζουν εξαιρέσεις σε περίπτωση που δεν μπορεί να ανακτιθούν κράτησεις μέσω των παραμέτρων ή αν οι χρήστες δεν είναι εξουσιοδοτούμενοι να προβούν στη χρήση των μεθόδων.

Επιπλέον, το *RS* παρέχει και μια βοηθητική κλάση με όνομα `RSServiceHelper` η οποία παρέχει στον προγραμματιστή ένα στιγμιότυπο της υπηρεσίας για την ΠΥ που επιθυμεί. Το στιγμιότυπο αυτό παρέχει υλοποιημένο το *RS API* στο χρήστη ρυθμισμένο σύμφωνα με τις προδιαγραφές του ΠΥ. Κάθε ΠΥ αναλαμβάνει να κοινοποιεί τις υπηρεσίες τις μέσω υπηρεσιών ιστού (*Web Services*)[71] σε μια συγκεκριμένη διεύθυνση ιστού (*RS Service URL*). Για παράδειγμα για την ΠΥ που λειτουργεί στο E.A.I.T.Y της Πάτρας η διεύθυνση είναι `http://hercules.cti.gr:8888/rs`. Από αυτή την κλάση μας ενδιαφέρει η παρακάτω μέθοδος :

Κώδικας 2.4: RSServiceHelper

```
SNAA getRSService(String endpointUrl)
```

Η μέθοδος `getRSService()` παίρνει ένα αλφαριθμιτικό (`String`) ως παράμετρο μόνο και αυτή είναι η διεύθυνση στην οποία εκτίθεται η υπηρεσία *RS* για μια ΠΥ και επιστρέφει την υλοποίηση της διεπαφής σύμφωνα με τις προδιαγραφές του ΠΥ μέσω ενός στιγμιοτύπου της κλάσσης `RS`.

2.2.3 Η διεπαφή Wireless Sensor Network API (iWSN API)

Το *iWSN API* [65] είναι μια διεπαφή των υπηρεσιών του *TR* η οποία επιτρέπει στους χρήστες του να κάνουν και να διαχειρίζονται τους κόμβους και το ΑΔΑ που συμμετέχει σε μια ΠΥ. Με τη χρήση της διεπαφής ο χρήστης μπορεί να εξάγει χρήσιμα δεδομένα από το δίκτυο που είναι και η ουσία της πλατφόρμας *WISEBED*. Η διεπαφή μπορεί να χωριστεί σε τρία μέρη τα οποία ενώ διακριτά συνεργάζονται μεταξύ τους ώστε να επιτευχθεί η επικοινωνία του χρήστη με το ΠΥ και να αντλήσει χρήσιμες πληροφορίες από αυτό. Τα τρία αυτά μέρη είναι τα εξής :

- ***Session Management API***. Η διεπαφή αυτή σε συνεργασία με το σύστημα των κρατήσεων εγκαθιστά την επικοινωνία του χρήστη με την ΠΥ σε ότι έχει σχέση με τον πειραματισμό του χρήστη με το ΑΔΑ κατα τη διάρκεια μιας κράτησης δημιουργώντας ένα είδος συνόδου μεταξύ χρήστη και ΠΥ.

- ***Controller API***. Η διεπαφή αυτή εφόσον έχει εγκατασταθεί η επικοινωνία χρήστη-ΠΥ αναλαμβάνει να αποστείλει στο χρήστη δεδομένα από τους κόμβους που συμμετέχουν στο πείραμα του χρήστη και αναφορές κατάστασης μια ενέργειας στο που έχει κληθεί από το χρήστη.

- ***WSN API***. Αποτελεί την διεπαφή εντολών,αιτήσεων και ενεργειών στις οποίες μπορεί να προβεί ασύγχρονα ο χρήστης κατά τη διάρκεια μιας συνόδου.

Θα περιγράψουμε με περισσότερη λεπτομέρια τη χρήση του *iWSN API* σε ένα από τα επομενά κεφάλαια. Το *iWSN API* προϋποθέτει την ύπαρξη μηχανισμών ταυτοποίησης & εξουσιοδότησης καθώς και αυτών των κρατήσεων, παρόλλα αυτά αρκετές από τις λειτουργίες του δεν χρειάζονται απαραίτητα κάποια κράτηση ή ταυτοποίηση για να εκτελεστούν. Στη συνέχεια θα συζητήσουμε το σύνολο των μεθόδων του *iWSN API*.

Κώδικας 2.5: iWSN API

```
//SessionManagement
java.lang.String        getNetwork()
java.lang.String        getInstance(GetInstance parameters)
void    free(List<SecretReservationKey> secretReservationKey)



//Controller
void    receive(Message msg)
void    receiveStatus(RequestStatus status)

//WSN
String  areNodesAlive(List<java.lang.String> nodes)
String  describeCapabilities(String capability)
String  destroyVirtualLink(String sourceNode, String targetNode)
String  disableNode(String node)
String  disablePhysicalLink(String nodeA, String nodeB)
String  enableNode(String node)
String  enablePhysicalLink(tring nodeA, String nodeB)
String  flashPrograms(List<String> nodeIds, List<Integer> programIndices, List<Program> programs)
List<java.lang.String>  getNeighbourhood(String node)
String  getNetwork()
String  getPropertyValueOf(String node, String propertyName)
String  getVersion()
String  resetNodes(List<String> nodes)
String  send(List<String> nodeIds, Message message)
String  setVirtualLink(String sourceNode, tring targetNode, String remoteServiceInstance,
```

Αρχικά το *SessionManagement API* προσφέρει τρείς μεθόδους. Η μέθοδος `getNetwork()` επιστρέφει μια αναπαράσταση του ΑΔΑ της ΠΥ που ο χρήστης επιθυμεί. Η αναπαράσταση αυτή είναι ένα αλφαριθμιτικό με τη μορφή της γλώσσας σήμανσης *WiseML*. Για αυτή τη γλώσσα θα συζητηθούν περισσότερες λεπτομέριες στη συνέχεια του κεφαλαίου. Η μέθοδος `getInstance()` επιστρέφει ένα τοπικό στιγμιότυπο της υπηρεσίας συνόδων για το χρήστη. Παράμετροι για να παραχθεί σωστά το στιγμιότυπο είναι τα ΜΚΚ της κράτησης του χρήστη καθώς και μια διεύθυνση του τοπικού ελεγκτή υπηρεσιών ιστού του χρήστη (*Web Service Controller*) για την υπηρεσιά που υλοποιεί το *Controller API*. Η μέθοδος αυτή επιστρέφει τη διεύθυνση στην οποία το ΠΥ έχει σϊηκώσειγϊα την υπηρεσιά που υλοποιεί το *WSN API*.

Το *Controller API* θέτει τις επικοινωνιακές βάσεις μεταξύ του χρήστη και της ΠΥ. Η υπηρεσία αυτή είναι υλοποιημένη μέσω ελεγκτών υπηρεσιών ιστού και συνεργάζεται με το *WSN API* ώστε να εμφανίζονται τα αποτελέσματα των ασύγχρονων ενεργιών του. Επίσης είναι υπεύθυνο για να επιστρέφει τις μετρήσεις των κόμβων στο ΑΔΑ ως μηνύματα από το χρήστη. Οι μέθοδοι `receive()` και

`receiveStatus()` δεν επιστρέφουν κάτι αλλά οι παράμετροι τους σε συνδυασμό την τεχνολογία των ελεγκτών ιστού περιέχουν τα μηνύματα που επιστρέφει το ΠΥ στον χρήστη. Η μέθοδος `receive()` δέχεται ως παραμέτρους το μήνυμα που στέλνει το ΠΥ στο χρήστη. Το μήνυμα περιέχει μια χρονοσφραγίδα , τον τύπο του μηνύματος π.χ Δεδομένα/Πληροφορία , διαγνωστικό μήνυμα και τα δεδομένα. Η άλλη μέθοδος έχει ως σκοπό την πληροφόρηση της κατάστασης μιας ενέργειας του χρήστη στο ΠΥ , καθώς δέχεται ένα μήνυμα κατάστασης που δείχνει το ποσοστό ολοκλήρωση ή ένα απρόσμενο συμβαν που μπορεί να συνέβει κατα την εκτέλεση μιας ενέργειας.

Στη συνέχεια το *WSN API* μας δίνει μέσω των μεθόδων ένα σύνολο ενεργειών στις οποίες μπορεί να προβεί ένας χρήστης. Οι ενέργειες είανι ασύγχρονες και ένα είδος συγχρονισμού για τα κατάσταση-/αποτελέσματα των ενεργειών επιτυγχάνετε με χρήστη των άλλων δύο διεπαφών που συζητήσαμε. Δεν θα μπούμε στη διαδικασία να περιγράψουμε όλες τις μεθόδους του *WSN API* καθώς τα ονόματα τους είναι αρκετά περιγραφικά ως προς τη λειτουργία τους. Θα σταθούμε στις μέθοδους `flashPrograms()` , `resetNodes()` και `setVirtualLink()`. Η μέθοδος `flashPrograms()` δίνει την εντολή στο ΠΥ να φορτώσει ένα πρόγραμμα λειτουργίας σε ένα σύνολο από κόμβους. Τα δεδομένα του προγράμματος λειτουργίας εμπεριέχοντε στις παραμέτρους της συνάρτησης μαζί με το σύνολο των κόμβων. Η μέθοδος `resetNodes()` φροντίζει στο να επανεκκινηθεί η λειτουργία ενός συνόλου από κόμβους. Με τη συντονισμένη επανεκκίνηση συχνά οι πειραματιστές θέλουν να πετύχουν και επανεκκίνηση ενός κατανεημημένου αλγορίθμου. Τέλος η μέθοδος `setVirtualLink()` στα πλαίσια του ενοποιημένου ΠΥ επιτρέπει την δημιουργία εικονικών δεσμών μεταξύ κόμβων που δεν ανήκουν στο ίδιο ΠΥ. Με αυτό το τρόπο ενώ οι κόμβοι είναι απομακρυσμένοι μπορούν να γίνουν γειτονική και να ανταλλάζουν δεδομένα μέσω του διαδικτύου με τη χρήση *WSN API*. Ως παράμετρους η μέθοδος δέχεται τα μοναδικά ονόματα των κόμβων , καθώς και τη διεύθυνση του *WSN API* του απομακρυσμένου ΠΥ ώστε να ολοκληρωθεί η εγκατάσταση της εικονικής σύνδεσης.

Επιπλέον, το *iWSN* παρέχει και μια βοηθητική κλάση με όνομα `WSNServiceHelper` η οποία παρέχει στον προγραμματιστή ένα στιγμιότυπο της υπηρεσίας για την ΠΥ που επιθυμεί. Το στιγμιότυπο αυτό παρέχει υλοποιημένο το *WSN API* στο χρήστη ρυθμισμένο σύμφωνα με τις προδιαγραφές του ΠΥ. Κάθε ΠΥ αναλαμβάνει να κοινοποιεί τις υπηρεσίες τις μέσω υπηρεσιών ιστού (*Web Services*)[71] σε μια συγκεκριμένη διεύθυνση ιστού (*Web Service URL*). Για παράδειγμα για την ΠΥ που λειτουργεί στο E.A.I.T.Y της Πάτρας η διεύθυνση είναι `http://hercules.cti.gr:8888/sessions`. Από αυτή την κλάση μας ενδιαφέρουν οι παρακάτω μέθοδοι:

Κώδικας 2.6: WSNServiceHelper

```
SessionManagement getSessionManagementService(String endpointUrl)
Controller getControllerService(String endpointUrl)
WSN getWSNService(String endpointUrl)
```

Οι παραπάνω μέθοδοι έχουνε ένα αλφαριθμιτικό (`String`) ως παράμετρο μόνο και αυτή είναι η διεύθυνση στην οποία εκτίθεται η υπηρεσία *iWSN* για μια ΠΥ και επιστρέφει την υλοποίηση της αντίστοιχης διεπαφής σύμφωνα με τις προδιαγραφές του ΠΥ μέσω ενός στιγμιοτύπου των κλάσσεων `SessionManagement`, `Controller` και `WSN`.

2.3 Η βιβλιοθήκη WiseML

Η πλατφόρμα *WISEBED* διαθέτει πολλά και διαφορετικά συστατικά που συνθέτουν ένα πλήρες και περίπλοκο οικοσύστημα από ΠΥ,ΑΔΑ και δεδομένα. Εμφανίστηκε λοιπόν η ανάγκη αναπαράστασης της πληροφορίας και των δεδομένων του έργου με ένα δομημένο τρόπο ο οποίος μπορεί να συνταχθεί και να είναι επεξεργάσιμος από μια μηχανή. Χρησιμοποιώντας την ευρέως διαδεδομένη τεχνολογία των γλωσσών σήμανσης (*Markup Languages*) και της *XML*[77], αναπτύχθηκε η γλώσσα *WiseML*[76]. Σκοπός της *WiseML* είναι η χρήση της σε δομημένα κείμενα ώστε να περιγράφεται και να αποθηκεύεται η πληροφορία στα πλαίσια της λειτουργίας ενός ΠΥ που ανήκει στο *WISEBED*. Το σχήμα της γλώσσας αποτελείται από τρία μέρη, το *Setup*, *Scenario* και το *Trace*. Με αυτό το τρόπο μπορεί να περιγραφή η δομή ενός ΠΥ και του ΑΔΑ που εξυπηρετεί, ενέργειες κατα τη διάρκεια πειραμάτων και δεδομένα που προέρχονται από τους κόμβους που σημμετέχουν σε πειράματα. Το κείμενο είναι τύπου *XML* και αυτό σημαίνει πως μπορεί να είναι έγκυρο χωρίς να περιέχει απαραίτητα όλες τις σημάνσεις που αναφέραμε και θα αναφερθούν στη συνέχεια. Παρόλα αυτά είναι απαραίτητο να έχει σημάνσης έναρξης και λήξης για κάθε στοιχείο που χρειάζεται να περιγραφεί.

Κώδικας 2.7: Κύρια δομή ενός WiseML κειμένου

```
<wiseml version""=1.0 xmlns"=http://wisebed.eu/ns/wiseml/1.0">
    <setup>
        <!-- static information section -->
    </setup>
    <scenario>
        <!-- scenario information -->
    </scenario>
    <trace id>
        <!-- dynamic information -->
    </trace>
</wiseml>
```

Το *Setup* αποτελεί το κομμάτι της *WiseML* η οποία περιγράφει κυρίως τους κόμβους, συνδέσμους που σημμετέχουν στο ΑΔΑ ενός ΠΥ ή σε ένα πείραμα. Υπό αυτό το κομμάτι περιγράφεται μέσω συντεταγμένων οι γεωγραφικές διαστάσης μιας ΠΥ, τα σημεία στα οποία βρίσκονται οι κόμβοι και η κινητικότητα της τοπολογίας. Επιπλέον δίνονται πληροφορίες για τους κόμβους του δικτύου όπως το είδος των αισθητήρων που φέρουν,ο τύπος τους, το πρόγραμμα λειτουργίας που τρέχουν, ο τύπος δεδομένων που εκφράζουν τις μετρήσεις τους οι αισθητήρες κ.α. Στο κομμάτι του *Setup* μπορούν να περιγραφούνε και οι φυσικοί και οι εικονικοί σύνδεσμοι (ασύρματοι) μεταξύ κόμβων. Η περιγραφή αυτή έχει να κάνει με το αν οι σύνδεσμοι είναι κρυπτογραφημένοι, αν περιέχουνε θόρυβο η οποιαδήποτε άλλη μέτρηση πάνω στο σύνδεσμο μπορεί να ενδιαφέρει τον ερευνητή. Ένα παράδειγμα του κομματιού *Setup* ακολουθεί.

Κώδικας 2.8: Παράδειγμα Setup

```
<setup>
    <!-- setup properties section -->
    <origin>
        <x>1.0</x>
        <y>5.8</y>
        <z>2</z>
```

```
        <phi>5</phi>
        <theta>0</theta>
    </origin>
    <timeinfo>
        <start>2009-08-28T14:03:00Z</start> <end>2009-08-28T15:03:00Z</end> <!-- OR <duration>12</duration>
             --> <unit>seconds</unit>
    </timeinfo>
    <interpolation>cubic</interpolation>
    <description>this is an exmaple WiseML.</description>
    <!-- instantiation of nodes -->
    <node id="urn:wisebed:node:tud:M4A0P2OV">
        <position>
            <x>0</x>
            <y>0</y>
            <z>6</z>
            </position>
            <gateway>1</gateway>
            <programDetails>blinkfast.tnode</programDetails>
            <nodeType>TNode v4</nodeType>
            <description>A fast blinking TNode</description>
            <capability>
                <name>urn:wisebed:node:capability:ranger</name>
                <datatype>decimal</datatype>
                <unit>meters</unit>
                <default>0</default>
            </capability>
    </node>
    <node id="urn:wisebed:node:...:...">...</node> ...
    <node id="urn:wisebed:node:...:...">...</node> -->
    <link source="urn:wisebed:node:tud:330010" target="urn:wisebed:node:tud:330006">
        <encrypted>true</encrypted>
        <virtual>false</virtual>
        <rssi datatype="decimal" unit="dBm" default="-120" /> <capability>
        <name> urn:wisebed:node:capability:noise</name> <datatype>decimal</datatype>
        <unit>dBm</unit>
        </capability>
    </link>
    <link source="urn:wisebed:...:...: target="urn:wisbed:...:...:>...</link> ...
    <link source="urn:wisebed:...:...:" target="urn:wisbed:...:...:>...</link>
</setup>
```

Σε αντιθεση με τα άλλα δύο κομμάτια το κομμάτι του *Setup* είναι απαραίτητο σε κάθε κείμενο *WiseML*. Αν και πιθανότητα η απουσία του δεν επηρεάζει την *XML*-εγκυρότητα του, ένα κείμενο σε *WiseML* δίνει της απαραίτητες πληροφορίες των συμμετέχοντων πόρων σε ένα πείραμα ή ΠΥ.

Το κομμάτι του *Scenario* χρησιμοποιείται για να εξομοιωθούν συνθήκες οι οποίες αλλάζουνε την υπάρχουσα κατάσταση στους κόμβους και συνδέσμους μιας ΠΥ κατα τη διάρκεια ενός πειράματος , αλλάζοντας έτσι και τα παραγώμενα δεδομένα. Διάφορες ένεργιες μπορούνε να προκαλέσουνε τέτοιες αλλαγές όπως η ενεργοποίηση ή απενεργοποίηση ενός κόμβου ή συνδέσμου, τοποθέτεση εσφαλμένων ενδείξεων κ.α . Ακολουθεί ένα παράδειγμα για το κομμάτι του *Scenario*.

Κώδικας 2.9: Παράδειγμα Scenario

```
<scenario>
    <timestamp>0</timestamp>
    <enableNode id="..." />
```

```
    <disableNode id="..." />
    <enableLink source="..." target="..." />
    <disableLink source="..." target="..." />
        <node id="...">
            <position>
                <x>0</x>
                <y>1</y>
                <z>2</z>
                <phi>0</phi>
                <theta>1</theta>
            </position>
                <data key="lqi">50</data>
        </node>
</scenario>
```

Οποιασδήποτε αλλαγές επιβάλει το *Scenario* σε δεδομένα που τέθηκαν κατα το κομμάτι του *Setup* παρουσιάζονται στο δυναμικό κομμάτι του κειμένου το οποίο ονομάζεται *Trace*. Το κομμάτι αυτό είναι το δυναμικό κομμάτι του κειμένου και βασίζονται σε καινούρια δεδομένα ή αλλαγές οι οποίες παρουσιάζονται στο χρόνο. Το *Trace* είναι το κομμάτι στο οποίο καταγράφονται κατάλληλα τα δεδομένα που προέρχονται από τους αισθητήρες των κόμβων , αλλαγές στην θέση τους όταν έχουμε να κάνουμε με κινητούς κόμβούς και διάφορες μετρήσεις στους συνδέσμους μεταξύ κόμβων. Κύριο χαρακτηριστικό του κομματιού αυτού είναι η χρήση χρονοσφραγίδων ώστε να περιγραφεί η πάροδος του χρόνου στα πλαίσια ενός πειράματος. Ένα παράδειγμα του *Trace* είναι το ακόλουθο.

Κώδικας 2.10: Παράδειγμα Trace

```
<trace>
    <timestamp>2</timestamp>
    <node id="...">
        <position>
            <x>2</x>
            <y>4</y>
            <z>6</z>
        </position>
        <data key="ranger">12</data>
        <!-- <data key="...">
        <link source="..." target="...">
            <rssi>12</rssi>
            <data key="lqi">100</data>
        </link>
        <timestamp>2</timestamp>
        <!-- ... -->
</trace>
```

2.4 Η βιβλιοθήκη Wiselib

Η πλατφόρμα *WISEBED* περιλάμβανει και την ανάπτυξη της βιβλιοθήκη *Wiselib*[75] για την ανάπτυξη κατανεμημένων αλγορίθμων που εκτελούνται σε ΑΔΑ. Η βιβλιοθήκη αυτή περιέχει μια συλλογή αλγορίθμων που χρησιμοποιούνται σε διάφορους τομείς της έρευνας πάνω στα ΑΔΑ, όπως δρομολόγηση και τοπικοποίηση του δικτύου. Η βιβλιοθήκη είναι υπο συνεχή ανάπτυξη και υποστηρίζει μερι-

κές από τις ευρέως γνωστές πλατφόρμες ΑΔΑ όπως τις *iSense*[34], *Contiki*[11] η το πρόσφατο εξομοιωτή ΑΔΑ με το όνομα *Shawn*[59]. Έχει αναπτυχθεί εξ'ολοκλήρους στη γλώσσα προγραμματισμού *C++*[6] και χρησιμοποιεί τα πρότυπα της γλώσσας (*C++ templates*) όπως οι βιβλιοθήκες *Boost*[5] και *CGAL*[10]. Ο σχεδιασμός της βιβλιοθήκης επιτρέπει την ανάπτυξη κώδικα ο οποίος μπορεί να μεταφραστεί για διαφορετικές πλατφόρμες αποδοτικά.

Η χρήση της βιβλιοθήκης οδηγεί στην ανάπτυξη των προγραμμάτων λειτουργίας των κόμβων που έχουμε αναφέρει σε αυτό το κεφάλαιο με εύκολο τρόπο, παρέχωντας διεπαφές του λειτουργικού συστήματος του κόμβου. Έτσι απλοποιείται η διαδικασία ανάπτυξης των προγραμμάτων λειτουργίας και ο προγραμμάτιστης δεν χρειάζεται να ασχοληθεί με λειτουργικότητα χαμηλού επιπέδου ή με λεπτομέριες του υλικού του κόμβου. Επιπλέον οι αλγόριθμοι που αναπτύσωνται με τη χρήση της *Wiselib* μπορούν να δοκιμαστούν σε ενα περιβάλλον εξομοιώσης ώστε να ελεγχθούν και ύστερα να μεταφραστούν σε κώδικα πραγματικών συσκευών. Επίσης η υπάρχουσα υλοποίηση αλγορίθμων επίσης βοηθάει τον προγραμματιστή στο να συγκεντρωθεί στο κύριο πρόβλημα που θέλει να λύσει π.χ η *Wiselib* διαθέτει ένα αλγόριθμο δρομολόγησης ο οποίος μπορεί να χρησιμοποιηθεί στα πλαίσια ανάπτυξης ενός αλγορίθμου συλλογής τιμών από τους αισθητήρες ενός ΑΔΑ.

Κεφάλαιο 3

Προκλήσεις

Στο κεφάλαιο αυτό καταγράφονται οι προκλήσεις που εμφανίστηκαν κατά τη διάρκεια ανάπτυξης της εφαρμογής *WiseUI*. Καθώς το έργο έχει να κάνει με την ανάπτυξη μιας εφαρμογή ιστού θα αναφερθούμε για την ανάγκη της ασύγχρονης επικοινωνίας μεταξύ Πελάτη(*Client*) και εξυπηρέτη(*Server*), τον τρόπο με τον οποίο μεταφέρεται πληροφορία με την μορφή αντικείμενων, το μοντέλο αποθήκευσης δεδομένων, την ανάγκη για γρήγορη και χωρίς λάθη ανάπτυξη του έργου.

3.1 Ασύγχρονη επικοινωνία

Στις περισσότερες διαδικτυακές εφαρμογές, συναντούμε το μοντέλου Πελάτη-Εξυπηρετητής (*Client-Server Model*)[8]. Στις εφαρμογές ιστού συνήθως το ρόλο του Πελάτη τον αναλαμβάνει ένα ειδικό πακέτο λογισμικού ο περιηγητής ιστού (*Web Browser*)[70]. Ο περιηγητής ιστού αποτελεί το εργαλείο που τυπώνει τη γραφική διεπαφή της εφαρμογής και ο χρήστης διαχειρίζεται μια εφαρμογή μέσω αυτού του λογισμικού. Οι εξελίξεις στο χώρο του ιστού τις δύο τελευταίες δεκαετίες έχουν άλλαξει τα χαρακτηριστικά του μοντέλου αυτού. Πλέον ένας περιηγητής ιστού με τη χρήση τεχνολογιών όπως η *JavaScript*[41] μπορεί να προσφέρει ένα σημαντικό ποσοστό στη λειτουργικότητας της εφαρμογής ξέχωρα από το να παρέχει μια γραφική διεπαφή για τον χρήστη. Το μοντέλο πλέον της επικοινωνίας αλλάζει και αρχίζει να χάνει την διακριτότητα του σε ένα βαθμό. Σε μια πολύπλοκη διαδικτυακή εφαρμογή, έτσι και στο *WiseUI*, υπήρξε η ανάγκη για ασύγχρονη επικοινωνία μεταξύ του Πελάτη και εξυπηρέτη. Οι σύγχρονες εφαρμογές απαιτούν αλληλεπιδραστική συμπεριφορά, που σημαίνει την ανταλλαγή δεδομένων από και προς τον εξυπηρέτη χωρίς να είναι απαραίτητη η εμφάνιση συνθηκών ανταγωνισμού (*Race Conditions*)[53]. Η εμφάνιση συνθηκών ανταγωνισμού είναι ένα κλασσικό παράδειγμα ελατωματικής συμπεριφοράς μιας εφρμογής λογισμικού και οδηγεί σε προβλήματα ως προς την λειτουργικότητα και την χρηστικότητα της εφαρμογής. Λύση σε αυτό το πρόβλημα δίνει η τεχνολογία της ασύγχρονης επικοινωνίας. Δεδομένα, αιτήσεις και εντολές μεταφέρονται ή εκτελούνται στον Εξυπηρετητής, χωρίς η πλευρά του Πελάτη να να αναμένει κάποια ανταπόκριση ή επιβεβαίωση αμέσω ύστερα από την κλήση που έκανε. Αντίθετα, συνεχίζεται η εκτέλεση του κώδικα του Πελάτη ανεμπόδιστα. Εν τέλει ο Εξυ-

πηρετητής επιστρέφει μια πληροφορία/απάντηση σχετικά με την ασύγχρονη κλήση που έγινε και ο Πελάτης μπορεί πλέον εκείνη τη στιγμή να διαχειριστεί την επιστρεφόμενη πληροφορία από τον Εξυπηρετητής. Όπως αναφέρθηκε, η γλώσσα που χρησιμοποιείται για τους πελάτες-περιηγητές στον ιστό είναι η *JavaScript* και με τη χρήση των μεθολογιών και τεχνολογιών *AJAX (Asynchronous Javascript and XML)*[2] μπορεί να επιτευχθεί ασύγχρονη επικοινωνία μεταξύ του περιηγήτη και του Εξυπηρετητής της εφαρμογής. Κύριο πλεονεκτήμα που προσφέρει η ασύγχρονη επικοινωνία στο χώρο των εφαρμογών ιστού είναι η ταχύτητα στην διαδραστικότητα στη χρήση της γραφικής διεπαφής στο χρήστη. Πλέον δεν χρειάζεται να φορτωθεί και να επαναγραφτεί ολόκληρη στατική σελίδα ιστού με νέα δεδομένα, αλλά να αλλάζει ένα μικρό μόνο μέρος της σελίδας με τη νέα πληροφορία. Το έργο *WiseUI* είναι εξ'ολοκλήρου κατασκευασμένο με χρήση ασύγχρονων κλήσεων εκμεταλευόμενο τις δυνατότητες που προσφέρει η εργαλειοθήκη του *GWT*. Περισσότερες λεπτομέριες θα δωθούν στα επόμενα δύο κεφάλαια της αναφοράς.

3.2 Μεταφορά αντικειμένων

Μέχρι και σήμερα, οι περισσότερες εφαρμογές ιστού χρησιμοποιούν το μοντέλο Πελάτη-Εξυπηρετητής με τον εξής απλό τρόπο. Ο Πελάτης κάνει μια αίτηση μέσω του πρωτοκόλλου *HTTP*[32] για ένα πόρο που παρέχει ο Εξυπηρετητήςς όπως μια σελίδα γραμμένη σε γλώσσα *HTML*[31]. Ο Εξυπηρετητήςς επιστρέφει μια κατάλληλα διαμορφωμένη σελίδα με ή χωρις νέα πληροφορία για τον χρήστη που έκανε την αίτηση. Με την χρήση νέων τεχνολογίων πλέον οι εξυπηρετητές δεν χρείαζεται να επιστρέψουν μια σελίδα , η οποία είναι γραμμένη κατάλληλα ώστε να εμφανίζει τη πληροφορία με συγκεκριμένο τρόπο, αλλά μόνο την απαραίτητη πληροφορία μειώνοντας έτσι το επικοινωνιακό φόρτο της εφαρμογής. Επιπλέον όπως θα δουμε και στη συνέχεια οι τεχνολογίες που θα χρησιμοποιήσουμε ανήκουν στο χώρο της *Java*, μια γλώσσα η οποία έχει ως κύριο χαρακτηριστικό την αντικεινοστρέφεια. Το μοντέλου του αντικειμενοστραφούς προγραμματισμού θεωρείται από τα πλέον διαδομένα και από τα πιο εύχρηστα για τον προγραμματιστή , πράγμα που αποτελεί έναν από τους κυριότερους λόγους υιοθέτησης της αντικειμενόστρεφειας από σχεδόν όλες τις σύγχρονες γλώσσες προγραμματισμού. Αναπόφευκτα η πληροφορία αποκτά και αυτή πλέον την μορφή αντικειμένων ώστε να είναι εύκολη η διαχείριση και επεξεργασία της. Για παράδειγμα σε μια διαδικτυακή εφαρμογή ατζέντας επαφών, ο Εξυπηρετητήςς σχεδιάζεται έτσι ώστε οι επαφές να εκφράζονται ως αντικείμενα.

Κώδικας 3.1: Κλάσση αναπαράστασης επαφής - Contact

```
public class Contact{
    private String name;
    private String phone;
    private String address;
    public void setName(final String name){
        this.name = name;
    }
    public String getName(){
        return name;
    }
    //...
}
```

Αντίστοιχα μπορεί και ο Πελάτης να έχει μια παρόμοια έκφραση των επαφών ώστε να τις εμφανίσει και να τις διαχειριστεί. Στη περίπτωση του *WiseUI*, η κύρια τεχνολογία που χρησιμοποιείται είναι η εργαλειοθήκη *GWT* η οποία βασίζεται εξ΄ολοκλήρου στη γλώσσα *Java*. Όπως βλέπουμε η χρήση της τεχνικής του αντικειμενοστραφούς προγραμματισμού είναι διάσπαρτη στα συστατικά του κωδικά μας. Θα χρειαστεί λοιπόν να γίνεται εύκολα και αποδοτικά η ανταλλαγή στιγμιοτύπων των κλάσσεων της εφαρμογής μεταξύ Πελάτη και Εξυπηρετητής. Θα δούμε στο επόμενο κεφάλαιο την τεχνική των μεταφέρσιμων αντικειμένων με λεπτομέρια.

3.3 Αποθήκευση δεδομένων

Ένα κύριο χαρακτηριστικό των σύγχρονων εφαρμογών είναι η αποθήκευση δεδομένων (*Data Persistence*)[50]. Η αποθήκευση των δεδομένων αφορά τις λειτουργίες που χρειάζεται μια εφαρμογή ώστε τα δεδομένα-πληροφορία που δημιουργεί και διαχειρίζεται να ανακτώνται και μετά τον τερματισμό της εφαρμογής. Με την αποθήκευση των δεδομένων στην μνήμη ενός υπολογιστικού συστήματος θα οδηγούσε σε διαγραφή τους κατα τον τερματισμό της εφαρμογής ή του συστήματος.

Η διατήρηση της πληροφορίας μιας συγκεκριμένης κατάστασης, επιτυγχάνεται με αποθήκευση των δεδομένων σε μέσα που έχουν την ικανότητα να διατηρούνε την αποθηκευμένη πληροφορία ακόμα και αν είναι τελείως αποφορτισμένα, όπως ο σκληρός δίσκος ή η μνήμη flash. Για παράδειγμα, λογισμικά επεξεργασίας εικόνας ή κειμένου, επιτυγχάνουν την διατήρηση της κατάστασης αποθηκεύοντας τα έγγραφα σε αρχεία. Προφανώς τα παραπάνω δεν αποτελούνε πλέον πρόκληση στις μέρες μας αλλά αυτό που αποτελεί σοβαρή πρόκληση, είναι η αποτελεσματική αντιμετώπιση των θεμάτων που προκύπτουν με την αύξηση του όγκου των δεδομένων που προορίζονται για αποθήκευση στη βάση δεδομένων. Σε ένα πολύ μεγάλο βαθμό, τα παραπάνω προβλήματα αναλαμβάνει να αντιμετωπίσει ένα σύστημα διαχείρισης βάσης δεδομένων (*Database Management System*)[14]. Πρόκειται για ένα πακέτο λογισμικού που ελέγχει την δημιουργία, επίβλεψη και χρήση μιας βάσης δεδομένων [78]. Βοηθά τους διαχειριστές βάσεων δεδομένων να προχωρούν στην ευέλικτη ανάπτυξη των συστημάτων τους. Αναφορικά, μια βάση δεδομένων είναι μία ολοκληρωμένη συλλογή από δεδομένα, καταχωρήσεις και άλλα αντικείμενα. Ένα ΣΔΒΔ, επιτρέπει διαφορετικές εφαρμογές χρηστών να έχουν ταυτόχρονη πρόσβαση στην ίδια βάση δεδομένων. Μπορούν να χρησιμοποιούν ποικιλία από μοντέλα βάσεων δεδομένων, όπως είναι το σχεσιακό μοντέλο (*Relational Database*)[54] ή το μοντέλο αντικειμένων (*Object Database*)[46], έτσι ώστε να μπορούν με αρκετά βολικό τρόπο να περιγράψουν και να υποστηρίξουν εφαρμογές με διαφορετικές απαιτήσεις η καθεμία. Τυπικά, υποστηρίζουν γλώσσες ερωτημάτων, που είναι στην πραγματικότητα υψηλού επιπέδου προγραμματιστικές γλώσσες ή αλλιώς γλώσσες επικεντρωμένες στις βάσεις δεδομένων που απλοποιούν σε πολύ σημαντικό βαθμό την συγγραφή εφαρμογών βάσεων δεδομένων. Οι γλώσσες αυτές κάνουν επίσης πιο ευέλικτη την οργάνωση μιας βάσης όπως επίσης απλουστεύουν την ανάκτηση και παρουσίαση της πληροφορίας που προέρχεται από αυτή. Ένα ΣΔΒΔ, παρέχει διευκολύνσεις για τον έλεγχο πρόσβασης στα δεδομένα, την ενίσχυση της ακεραιότητας των δεδομένων, την διαχείριση του ελέγχου παράλληλης πρόσβασης, την ανάκτηση μιας βάσης δεδομένων ύστερα από κάποιο σφάλμα μέσω αντιγράφων ασφαλείας όπως και την διαχείριση θεμάτων ασφαλείας.

3.4 Ευέλικτη ανάπτυξη λογισμικού

Είναι πολύ κρίσιμο κατά την ανάπτυξη οποιασδήποτε εφαρμογής, είτε μιλάμε για το διαδίκτυο είτε για επιφάνεια εργασίας, η διαδικασία συγγραφής να είναι όσο τον δυνατόν λιγότερο επίπονη και χρονοβόρα γίνεται για τον μηχανικό. Ειδικά όταν έχουμε να κάνουμε με πολύπλοκες και εκτενής εφαρμογές αυτό απαιτεί την σωστή οργάνωση του κώδικα, την γρήγορη εκσφαλμάτωση της εφαρμογής, τον διαμοιρασμό των απαιτούμενων εργασιών ανά την ομάδα κτλ. Έτσι παρουσιάζεται αυτόματα η ανάγκη για την χρήση τεχνολογιών που ευνοούν την αντιμετώπιση τέτοιου είδους προκλήσεων. Γενικότερα, υπάρχουν πρωτότυπα διαδικασιών που ακουλουθούνται κατά τη διάρκεια υλοποίησης μιας υπηρεσίας και βασίζονται κυρίως σε ευέλικτες [1], επαναληπτικές και αυξητικές μεθόδους ανάπτυξης[35].

3.5 Δοκιμή λογισμικού

Η δοκιμή λογισμικού είναι μια έρευνα που εφαρμόζεται για να παρέχει στους ενδιαφερόμενους πληροφορίες που έχουν να κάνουν με την ποιότητα του λογισμικού ή της υπηρεσίας που προσφέρεται. Η διαδικασία αυτή μπορεί να παρέχει αντικειμενική και ανεξάρτητη εικόνα του λογισμικού βοηθώντας την επιχειρησιακή πλευρά να κατανοήσει και να εκτιμήσει τους κινδύνους που παρουσιάζονται κατά την ανάπτυξη του λογισμικού. Στις περισσότερες περιπτώσεις οι τεχνικές Η διαδικασία δοκιμής λογισμικού θα μπορούσε να οριστεί ως η διαδικασία επικύρωσης και επιβεβαίωσης ότι ένα προϊόν λογισμικού:

- Πληρεί όλες τις προδιαγραφές όπως αυτές ορίζονται από το σχεδιασμό και την ανάπτυξη.
- Λειτουργεί όπως αναμενόταν - και
- Μπορεί να υλοποιηθεί με τα ίδια χαρακτηριστικά.

Οι πλειονότητα των σημερινών εφαρμογών παρέχουν ένα μεγάλο σύνολο από υπηρεσίες και συνεχώς εμπλουτίζονται με νέα χαρακτηριστικά. Παράλληλα οι χρήστες γίνονται όλο και πιο απαιτητικοί και η ανάγκη για εφαρμογές με σταθερές εκδόσεις που δεν θα παρουσιάζουν προβλήματα είναι όλο και πιο επιτακτική. Έτσι, με τον όρο δοκιμή λογισμικού αναφερόμαστε στη διαδικασία της επικύρωσης και της επαλήθευσης ότι ένα προϊόν λογισμικού πληρεί όλες τις προδιαγραφές βάση των οποίων είχε σχεδιαστεί και αναπτυχθεί και λειτουργεί όπως ανεμενόταν. Στις περισσότερες περιπτώσεις, η διαδικασία των δοκιμών εφαρμόζεται όταν η συγγραφή του κώδικα έχει ολοκληρωθεί. Προφανώς αυτό δεν είναι απαραίτητο καθώς κάτι τέτοιο καθορίζεται από την εκάστοτε μεθοδολογία που ακολουθείται. Υπάρχει, για παράδειγμα, μέθοδος ανάπτυξης που υπαγορεύει πως η συγγραφή των δοκιμών θα πρέπει να γίνεται πριν την συγγραφή του κώδικα της κυρίως εφαρμογής, εξασφαλίζοντας έτσι τις συνεχείς δοκιμές της εφαρμογής καθώς και την ύπαρξη δοκιμών ξεχωριστά για κάθε χαρακτηριστικό της. Σύγχρονα μοντέλα ανάπτυξης λογισμικού όπως το Agile, υπαγορεύουν τη συγγραφή των δοκιμών παράλληλα με την ανάπτυξη του κυρίως λογισμικού, ανατίθοντας στον προγραμματιστή το μεγαλύτερο μέρος της συγγραφής δοκιμών, πριν αυτό φθάσει στα χέρια της αρμόδιας ομάδας για την εφαρμογή εκτενών δοκιμών [62][68]

Κεφάλαιο 4

Εργαλεία ανάπτυξης - Τεχνολογίες

Στη συνέχεια παρουσιάζονται οι τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξης της εφαρμογής. Τα κριτήρια που χρησιμοποιήθηκαν για την επιλογή των εργαλείων ήταν κυρίως παράγοντες όπως η ταχύτητα ανάπτυξης, η ευελεξία που παρέχουν στον προγραμματιστή όσον αφορά τις δυνατότητες παραμετροποίησης αυτών καθώς φυσικά και το κατά πόσο κατάλληλες ήταν για τις ανάγκες που παρουσιάζονταν ανά διαστήματα.

4.1 Το πακέτο λογισμικού Google Web Toolkit

Το *Google Web Toolkit* (*GWT*)[27] είναι ένα σύνολο από εργαλεία για την ανάπτυξη πολύπλοκων εφαρμογών ιστού. Η εργαλειοθήκη *GWT* αναπτύχθηκε από την πασίγνωστη εταιρία *Google* και χρησιμοποιεί τα εργαλεία αυτά για την ανάπτυξη δικών της προϊόντων, όπως το *Google Docs*[25] και το *GMail*[23], τα οποία χρησιμοποιούνται από εκατομμύρια χρήστες καθημερινά. Ο λόγος για τον οποίο αναπτύχθηκε το συγκεκριμένο σύνολο εργαλειών ήταν η ανάγκη για την γρήγορη ανάπτυξη σύνθετων διαδικτυακών εφαρμογών που δεν θα απαιτούσαν από τον προγραμματιστή να έχει εξαιρετικές γνώσεις σε τεχνολογίες όπως *JavaScript* και *AJAX*. Αυτός ήταν και ένας από τους κύριους λόγους για τον οποίο επιλέξαμε να χρησιμοποιήσουμε αυτή την τεχνολογία για την ανάπτυξη της εφαρμογής *WiseUI*. Το *GWT*

Εικόνα 4.1: Το λογότυπο του Google Web Toolkit

δεν είναι μια βιβλιοθήκη λογισμικού, τουλάχιστον όχι μόνο μια βιβλιοθήκη. Ο προγραμματιστής μπορεί

να χρησιμοποιεί ένα διαγλωσσικό μεταφραστή από *Java* σε *JavaScript* με αποτέλεσμα να προγραμματίζει στην καθιερωμένη και ώριμη γλώσσα *Java* μέσω του μεταφραστή να παρέγει λειτουργικό,συμπαγή και βελτιστοποιημένο κώδικα *JavaScript*. Ένα από τα προβλήματα ανάπτυξης εφαρμογών *JavaScript* είναι η ανύπαρκτη (χωρίς βοήθεια ειδικού λογισμικού) απασφαλμάτωση κώδικα. Με τη χρήστη αυτού το μεταφραστή μπορούν να βρεθούνε εύκολα σφάλματα στον κώδικα και να αποφευχθεί το γνωστό πρόβλημα της άσπρης οθόνης¨ το οποίο δεν μπορεί να βοηθήσει τον προγραμματιστή να εντοπίσει τα λάθη του στον κώδικα.

Ένα άλλο πλεονέκτημα που παρέχει η *Java* στο *GWT* είναι η εικονική μηχανή της *Java* (*Java Virtual Machine*)[40]. Το *GWT* μπορεί να χρησιμοποιηθεί σε δύο λειτουργίες. Η πρώτη ονομάζεται λειτουργία παραγωγής (*Production Mode*) το οποίο χρησιμοποιεί τον μεταφραστή που περιγράψαμε . Η άλλη λειτουργία ονομάζεται λειτουργία ανάπτυξης (*Development Mode*) χρησιμοποιεί την εικονική μηχανή της *Java* ώστε ο κώδικας να εκτελεστεί στη μηχανή αυτή χωρίς να μεταφράζεται σε *JavaScript*. Αυτό επιτρέπει στην εύκολη δοκιμή της υπό ανάπτυξης εφαρμογή σε ένα σύγχρονο περιηγητή ιστού με τη χρήση ενός προσθέτου (*plugin*) για πολλούς γνωστούς περιηγητές ιστού.

Οι βιβλιοθήκες που συνθέτουν το *GWT* περιλαμβάνουν γραφικών συστατικών που ονομάζωνται *Widgets*, ένα απλό ασύγχρονο μηχανισμό απομακρυσμένης κλήσης διαδικασιών (*Remote Procedure Calls - RPC*)[56], διαχειριστή ιστορικού του περιηγητή. Επίσης περιλαμβάνει μια διαφορετική έκδοση της πρότυπης κεντρικής βιλιοθήκης της *Java*, όπως τα πακέτα `java.lang` και `java.util`, υλοποιημένη σε *JavaScript*.

Τα γραφικά συστατικά διεπαφής είναι ένα ισχυρό προνόμιο που προσφέρει το *GWT*. Μπορούν να επαναχρησιμοποιηθούν και να επεκεταθούν και αποτελούν μια ιδιαίτερα αποδοτική λύση καθώς η ανάπτυξη τέτοιων συστατικών είναι ιδιαίτερα χρονοβόρα. Παρόλα αυτά, το *GWT* αδικείται αν κάποιος το περιορισεί αποκλειστικά ως εργαλείο κατασκευής γραφικών διεπαφών για εφαρμογές ιστού. Το *GWT* είναι μια εργαλειοθήκη λογισμικού ανοιχτού κώδικα για την ανάπτυξη *JavaScript* εφαρμογών υψηλής επίδοσης. Οι μηχανικοί που ανάπτυξηαν το *GWT* έχουν τονίσει σε διάφορες παρουσιάσεις πως δεν είναι απλώς μια άλλη *JavaScript & AJAX* βιβλιοθήκη , αλλά ενα εργαλείο ανοικτής φιλοσοφίας ως προς την ανάπτυξη μιας εφαρμογής ιστού [26]. Το *GWT* δεν επιβάλλει από τη φύση του κάποια συγκεκριμένη αρχιτεκτονική στην εφαρμογή, αντίθετα οι προγραμματιστές αναλαμβάνουν το σχεδιασμό της εφαρμογής χωρίς να επεμβαίνει εκεί το *GWT*.

Σημαντική έκδοση στην ιστορία του *GWT* είναι η έκδοση 2.0. Η έκδοση αυτή έφερε αρκετές αλλάγες στις βιβλιοθήκες της εργαλειοθήκης και διάφορα άλλες προσθήκες ώστε να βοηθήσει τους προγραμματιστές που τη χρησιμοποιούν. Εισάχθηκε ο διαχωρισμός κώδικα (*Code Splitting*), όπου ο προγραμματιστής μπορεί να εισάγει σημεία διαχωρισμού στον κώδικα που προγραμματίζει ώστε ο μεταφρασμένος κώδικας σε γλώσσα *JavaScript* να αποτελείται από πολλά μικρότερα αρχεία και όχι από ένα μεγαλύτερο εννιαίο αρχείο. Καθώς ο κώδικας της *JavaScript* χρειάζεται να μεταφερθεί στο περιβάλλον του περιηγητή με αυτό το τρόπο μεταφέρονται μικρότερα αρχεία κάθε φορά που ζητούνται ,βελτιώνεται έτσι την επικοινωνία Πελάτη-Εξυπηρετητής ελαχιστοποιώντας τον επικοινωνιακό φόρτο. Ένα άλλο χαρακτηριστικό που χρησιμοποιείται ιδιαίτερα στο έργο *WiseUI*, είναι η χρήση του εργαλείου *UiBinder*[16] ώστε ο σχεδιαστής της γραφική διεπαφής να προσθέτει γραφικά στοιχεία σε μια σελίδα με δηλώσεις σε ένα

ειδικό *XML* κείμενο. Με το *UiBinder*, επιτρέπεται ο εμφανής διαχωρισμός κατα την ανάπτυξη της γραφικής διεπαφής της εφαρμογής από την υπολοιπη εφαρμογή. Τέλος άλλο ένα χρήσιμο εργαλείο είναι και το *Client Bundle*[28], με το οποίο στατικούς πόρους όπως εικόνες, *CSS stylesheets*, κείμενο κ.α μπορούν να συγκεντρωθούν σε ένα ειδικό πακέτο και να σταλθούν στον Πελάτη-περιηγητή με μόνο μεταφορά. Με αυτό το τρόπο δεν χρειάζεται να γίνονται συνεχόμενες μεταφορές πόρων σε κάθε αίτηση που κάνει ο περιηγητής όταν χρησιμοποιεί την εφαρμογή.

4.2 Εισαγωγή εξαρτήσεων με Guice & GIN

Ένα από τα πιο συχνά προβλήματα που αντιμετοπίζουν οι προγραμματιστές κατά την ανάπτυξη λογισμικού είναι το φαινόμενο του υψηλού ζευγαρωματός κώδικα (*Code Coupling*)[9], δηλαδή όταν συστατικά ενός πακέτου λογισμικού εξαρτώνται το ένα από το άλλο με ισχυρό τρόπο πάνω στον κώδικα σε βαθμό το λογισμικό να είναι δύσκολο να διατηρθεί και να επεκταθεί. Στον αντικειμενοστραφή προγραμματισμό , ένα αντικείμενο διαθέτει υψηλό ζευγάρωμα κώδικα όταν όλες του οι εξαρτήσεις παρουσιάζονται στην υλοποίηση του και οποιαδήποτε αλλαγή πρέπει να έχει γραφτεί εκ των προτέρων. Είναι επιθυμητό το αντικείμενο αυτό να ανακτά αυτές τις εξαρτήσεις του τη στιγμή που τις χρειάζεται χωρίς αυτές να είναι ισχυρά δεσμευμένες στον κώδικα του. Η Εισαγωγή Εξαρτήσεων (*Dependency Injection*)[17] είναι μια τεχνική η οποία επιτρέπει την ενσωμάτωση των συστατικών-εξαρτήσεων σε ένα κομμάτι λογισμικού με μια απλή αναφορά τους , ενώ η υλοποίηση τους βρίσκεται σε άλλο σημείο του κώδικα δειχνώντας έτσι ανεξάρτητα κομμάτια λογισμικού. Μοιάζει αρκετά με το προγραμματιστικό πρότυπο του Εργοστασίου (*Factory Method Pattern*)[21] αλλά στοχεύει στη χρήση λιγότερου κώδικα για να επιτευχθεί χαμηλότερο ζευγάρωμα κώδικα, συχνά χρησιμοποιώντας επιπλέον έτοιμο λογισμικό και δηλώσεις σε μια άλλη μέταγλώσσα όπως *XML*.

Το *Guice* είναι ένα ανοιχτού κώδικα πλαίσιο λογισμικού για τη γλώσσα *Java* και έχει αναπτυχθεί από μηχανικούς της *Google*. Παρέχει υποστήριξη για την εισαγωγή εξαρτήσεων με τη χρήση υποσημειώσεων της *Java*(*Java Annotations*)[37], για τη διαχείριση Java αντικειμένων. Όλα τα αντικείμενα που δηλώνονται ως εξαρτώμενα στο *Guice* δημιουργούνται και αρχικοποιούνται κατα την εκτέλεση της εφαρμογής , σχηματίζωντας ένα γράφο εξαρτήσεων και τοποθετούνται σε μια δεξάμενη αντικειμένων για χρήση όποτε ζητηθούν. Οι εξαρτήσεις συνήθως δηλώνονται ως παράμετροι στους δημιουργούς των αντικείμενων και με τη κλήση του εισαγωγέα του *Guice* (*Guice Injector*) τα αντικείμενα δημιουργούνται μαζί με τις απαραίτητες εξαρτήσεις. Πλέον η χρήση της κλήσης `new` δεν είναι χρήσιμη καθώς μια κλήση του εισαγωγέα παρέχει τα αντικείμενα που χρειάζεται η εφαρμογή κατάλληλα διαμορφωμένα. Το *Guice* σε αντίθεση με άλλα γνωστά συστήματα εισαγωγής εξαρτήσεων χρησιμοποιεί κλάσσεις και διεπαφές της *Java* ώστε να δηλωθούν οι εξαρτήσεις στην εφαρμογή αντίθετα με κάποια μέταγλώσσα όπως *XML*.

Για να οριστεί μια βιβλιοθήκη στο *GWT* χρειάζεται να πληρεί κάποια στοιχεία που θέτουν οι περιορισμοί του *GWT*. Μια βιλιοθήκη σαν το *Guice* είναι εξαιρετικά χρήσιμη σε μεγάλα έργα *Java*, όπως εφαρμογές γραμμένες στο *GWT*. Για αυτό οι μηχανικοί της *Google* ανέπτυξαν το *GIN*, ένα κλώνο του *Guice* για περιβάλλοντα *GWT* παρέχωντας ίδια λειτουργικότητα.

4.3 Εξυπηρετητές Εφαρμογής & Java Servlets

Ως Εξυπηρετητή Εφαρμογή (*Application Server*)[4], αναφερόμαστε σε ένα πακέτο λογισμικού που παρέχει το κατάλληλο περιβάλλον ώστε να εκτελούνται εφαρμογές με τους κατάλληλους πόρους και υπηρεσίες. Συνήθως, η εφαρμογή αυτή διαθέτει ένα σύνολο χρηστών οι οποίοι έχουν απομακρυσμένη πρόσβαση στις υπηρεσίες της εφαρμογής. Ο όρος προήλθε από το μοντέλο Πελάτη-Εξυπηρετητής και διαφοροποιείται από άλλους εξυπηρετητές ως προς την λειτουργικότητα , βλπ εξυπηρετητές βάσεων δεδομένων η αρχείων. Συνήθως ο Εξυπηρετητήςς εφαρμογής είναι προσβάσιμος μέσω του Παγκόσμιου Ιστού και διεπαφή με τον χρήστη είναι ιστοσελίδες. Οι εξυπηρητητές εφαρμογών όμως μπορούν να αναλάμβουν και μεγαλύτερο έργο απο αυτό, παρέχωντας υπηρεσίες και επικοινωνία με άλλα συστήματα χρήσιμα για την εφαρμογή που εξυπηρετούν. Και σε αυτή τη περίπτωση χρησιμοποιείται το μοντέλο Πελάτη-Εξυπηρετητής, με τον Εξυπηρετητής να παρέχει στον Πελάτη τις υπηρεσίες του μέσω προγραμματιστικής διεπαφής. Στα πλαίσια της εφαρμογής *WiseUI* θα χρησιμοποιήσουμε τον γνωστό Εξυπηρετητής εφαρμογών με όνομα *Apache Tomcat*[3].

Τα *Java Servlets*[39] είναι ειδικές κλάσεις, ορισμένες από το πρότυπο *Java Servlet API 3.0*, που χρησιμοποιούνται για να υλοποιηθεί η λειτουργικότητα της εφαρμογής στα πλαίσια ενός Εξυπηρετητής εφαρμογών. Βασίζονται στο μοντέλο προγραμματισμού αίτηση-απάντηση, ένα *Servlet* δέχεται αιτήσεις και βάσει της υλοποίησης επιστρέφει κάταλληλα διαμορφωμένες απαντήσεις. Η πιο συνιθισμένη εφαρμογή τους βρίσκεται στον ιστό, όπου οι αιτήσεις είναι για πόρους μέσω του *HTTP* πρωτοκόλλου και οι απαντήσεις που εκδίδουν είναι δυναμικές ιστοσελίδες. Παρόλα αυτά ένα *Servlet* μπορεί να επεκταθεί ώστε να ανταποκρίνεται σε οποιουδήποτε τύπου αίτησης. Στην περίπτωση του *WiseUI* χρησιμοποιούμε ένα ειδικό είδους *Servlet* το οποίο ονομάζεται *RemoteServiceServlet* και παρέχεται από τις βιβλιοθήκες του *GWT*. Σκοπός αυτών των ειδικών *Servlet* δεν είναι η εξυπηρέτηση *HTTP* αίτησεων, αλλά η ασύγχρονη εκτέλεση απομακρυσμένων μεθόδων και η επιστροφή των αποτελεσμάτων τους. Έτσι πλέον διαθέτουμε ασύγχρονη επικοινωνία μεταξύ Πελάτη και Εξυπηρετητής κρύβωντας τις λεπτομέριες της ασύγχρονης επικοινωνίας. Ο Πελάτης κάνει αίτηση για την εκτέλεση μιας ρουτίνας στον Εξυπηρετητής εφαρμογής, όπου με τη σειρά του επιστρέφει τα αποτελέσματα της κλήσης στον Πελάτη.

4.4 Αποθήκευση δεδομένων με JPA & Hibernate

Στο προηγούμενο κεφάλαιο αναφερθήκαμε στο ζήτημα της αποθήκευσης δεδομένων και στη σημασία του για τις σύγχρονες εφαρμογές λογισμικού. Οι σχεσιακές βάσεις δεδομένων είναι αρκετά πιο διαδομένες λόγω του ότι εκμεταλεύονται χαρακτηριστικά και συσχετίσεις που βρίσκονται στη φύση των δεδομένων της βάσεις και ως εκ τούτου θεωρούνται ως η κλασσική επιλογή σε σύγκριση με τις αντικεμενοστραφείς βάσεις δεδομένων στις περισσότερες εφαρμογές. Στις υποχρεώσεις των προγραμματιστών είναι η ανάπτυξη λογισμικού να περιλαμβάνει την σύνδεση της εφαρμογής με ένα σύστημα διαχείρισης βάσεων δεδομένων, ορισμός των οντοτήτων και των συσχετίσεων στο περιβάλλον της βάσης, υλοποίηση και εκτέλεση ερωτημάτων προς σύστημα διαχείρισης και τέλος ανάκτηση των αποτελεσμάτων από τα ερωτήματα.

Στην κοινότητα της *Java* το πρότυπο που επικρατεί είναι το *Java Peristence API - JPA*[42]. Είναι ένα

πρότυπο ώστε οι *Java* προγραμματιστές να μπορούν να υλοποιήσουνε τις παραπάνω προδιαγραφές για τη χρήση ΣΔΒΔ για τις εφαρμογές τους. Η χρήση του *JPA* επιτρέπει τη σύνδεση με βάσεις δεδομένων με τη χρήση συγκεκριμένων κλάσσεων-συνδέσμων (*Java DataBase Connectivity - JDBC*), εκτέλεση ερωτημάτων και ανάκτηση απαντήσεων μέσω μιας ειδικής γλώσσας ερωτημάτων (*Java Persistence Query Language*) και δημιουργία μεταδεδομένων για την αντιστοίχιση συσχετίσεων με αντικειμένων.

Το θέμα της αντιστοίχησης δεδομένων και συσχετίσεων είναι ένα αρκετά σημαντικό θέμα που απασχολεί τους μηχανικούς λογισμικού την τελευταία δεκαετία και ονομάζεται *Object/Relational Mapping - ORM*[47]. Το *ORM* είναι μια προγραμματιστική τεχνική η οποία αναπτύχθηκε ώστε τα δεδομένα από το σχεσιακό μοντέλο μιας βάσης δεδομένων να μπορούν να μεταφερθούν στο μοντέλο του αντικειμενοστραφούς προγραμματισμού. Τα δύο αυτά μοντέλα δεν είναι μόνο διαφορετικά αλλα και ασυμβίβαστα. Ζητήματα όπως η διακριτότητα των δεδομένων,η κληρονομικότητα των κλάσσεων,η ταυτότητα των δεδομένων, οι συσχετίσεις και πλοήγηση στα δεδομένα είναι ιδιαιτέρως σημαντικά και παρουσιάζονται έντονα κατά την αντιστοίχιση αυτών των δυο μοντέλων [48]. Ως εκ τούτου είναι απαραίτητο ενα ειδικό λογισμικό πακέτο το οποίο να προσφέρει λύση στα παραπάνω ζητήματα, δίχως να απασχολεί τον προγραμματιστή της εφαρμογής με τις λεπτομέριες αυτών των ζητημάτων. Το πρότυπο *JPA* προσφέρει τέτοιες λύσεις αλλά για το *WiseUI* προτιμήσαμε τη συνεργασία του *JPA* με τη πολυ γνωστή βιβλιοθήκη της *Hibernate*[55].

Η *Hibernate* είναι μια βιβλιοθήκη για τη αντιστοίχηση συσχετίσεων με αντικειμένα για τη γλώσσα της *Java*. Παρέχει ένα προγραμματιστικό πλαίσιο για τη αντιστοίχηση του αντικειμενοστραφούς μοντέλου σε μια παραδοσιακή σχεσιακή βάση δεδομένων, κατασκευάζοντας έτσι μια εικονική βάση αντικείμενων. Έτσι ο προγραμματιστής πλέον μπορεί να διαχειριστεί πίνακες και δεδομένα της βάσης προγραμματίζοντας τα κατάλληλα αντικείμενα και κλάσσεις με τη χρήση μεθόδων.

Πιο συγκεκριμένα, κύριο χαρακτηριστικό της *Hibernate* είναι η αντιστοίχιση Java κλάσεων σε πίνακες βάσεων δεδομένων καθώς και από *Java* τύπους δεδομένων σε *SQL* τύπους δεδομένων.Ο τρόπος με τον οποίο η Hibernate επιτυγχάνει την αντιστοίχιση *Java* κλάσεων σε πίνακες βάσεων δεδομένων είναι είτε με την κατάλληλη ρύθμιση ενός ξεχωριστού *XML* αρχείου είτε με τη χρήση *Java Annotations* πάνω σε κλάσσεις που ορίζουν το μοντέλο δεδομένων της βάσης. Χρησιμοποιώντας ένα *XML* αρχείο, η Hibernate μπορεί να δημιουργήσει πρότυπο πηγαίο κώδικα για τις κλάσεις που προορίζονται για να αναπαριστούν το μοντέλο. Το σχήμα της εκάστοτε βάσης δεδομένων διαμορφώνεται από τη Hibernate .

Ο συνδυασμός του *JPA* μαζί με την βιβλιοθήκη *Hibernate* προσφέρει μια ουσιαστική και ασφαλή διεπαφή της βάσης δεδομένων με την υπόλοιπη εφαρμογή. Τα δεδομένα της βάσης είναι πλέον πολυ πιο εύκολα διαχειρίσιμα καθώς πλέον έχουν την αναπαράσταση αντικείμενων. Μηχανισμοί συνόδων και κρυφής μνήμης που προσφέρει αυτός ο συνδυασμός κάνει την επικοινωνία με τη βάση εύκολη και γρήγορη. Τέλος, η βάση δεδομένων πλέον γίνεται πιο μεταφέρσιμη από ποτέ καθως αντιγράφωντας τις ρυθμίσεις και τα μεταδεδομένα μια βάση δεδομένων μπορεί να χρησιμοποιηθεί από περισσότερες από μια εφαρμογές.

4.5 Συναλλαγές με Spring Transactions

Είναι αναγκαίο να γίνει μία σύντομη αναφορά στην έννοια των συναλλαγών (*Transactions*) [15] πριν γίνει η συζήτηση για την τεχνολογία με όνομα *Spring Transactions Framework*[64]. Έτσι με τον όρο συναλλαγή, αναφερόμαστε σε ένα σύνολο ενεργειών πάνω σε ένα διαμοιρασμένο πόρο, όπως μια βάση δεδομένων,αρχεία ή αδόμητα δεδομένα, οι οποίες οποία πληρούν τις εξής ιδιότητες.

- **Ατομικότητα**. Το σύνολο των ενεργειών που συνιστούν μια συναλλαγή εκτελείται στο σύνολο του. Σε περίπτωση αποτυχίας όλες οι ενέργειες που εκτελέστηκαν ακυρώνονται και οποιεσδήποτε αλλαγές στον κοινόχρηστο πόρο αναιρούνται.
- **Συνέπεια**. Υποθέτωντας ότι ο πόρος βρίσκεται σε μια συνεπή ή αλλιως αποδεκτή κατάσταση πριν την έναρξη της συναλλαγής, με την λήξη της ο πόρος βρίσεκται ξανά σε αποδεκτή κατάσταση.
- **Απομόνωση**. Μια συναλλαγή δεν μπορεί να λειτουργήσει σε πόρο που χρησιμοποιεί μια άλλη συναλλαγή.
- **Διάρκεια**. Με τη λήξη της (*commit*), η συναλλαγή έχει καταγραφεί και τα αποτελέσματα/αλλαγές έχουν εγγραφεί στον πόρο και δεν γίνεται να χαθούν από βλάβες όπως διακοπή ρεύματος, σφάλμα της εφαρμογής, τερματισμός λειτουργίας της εφαρμογής.

Οι παραπάνω ιδιότητες καθιστούν τις συναλλαγές ως ιδανικό χαρακτηριστικό για ένα σύγχρονο σύστημα βάσεων δεδομένων. Αν και οι συναλλαγές δεν εφαρμόζονται μόνο στις βάσεις δεδομένων, μας ενδιαφέρει η χρήση τους στη βάση δεδομένων που θα αναπτύξουμε στα πλαίσια του έργου *WiseUI*. Το πλαίσιο ανάπτυξης λογισμικού *Spring Framework*[63], προσφέρει πλούσια υποστήριξη για την υλοποίηση συναλλαγών μέσω του *Spring Transactions Framework*, παρέχωντας ένα συνεπή αφαιρετικό μηχανισμό για τη διαχείριση τους. Συγκεκριμένα :

- Υλοποίηση κατάλληλη για τοπικές αλλά και γενικευμένες συναλλαγές, δηλαδή σε όλο το εύρος των Πελάτη-Εξυπηρετητής .
- Υποστήριξη εμφωλευμένων συναλλαγών, δηλαδή έναρξη μιας συναλλαγής εντός κώδικας μιας άλλης συναλλαγής.
- Δημιουργία σημείων αποθήκευσης (*Savepoints*) της συναλλαγής. Σε περίπτωση που χρειάζεται να ανακληθεί μια συναλλαγή, τότε η εκτέλεση της επιστρέφει στο τελευταίο *savepoint* που έχει οριστεί.
- Παρέχει ένα συνεπές προγραμματιστικό μοντέλο μεταξύ των διάφορων γνώστων προγραματιστικών διεπαφών όπως τα *JTA, JDBC, Hibernate, JPA* και *JDO*.
- Υποστηρίζει δηλωτική και προγραμματιστική διαχείριση των συναλλαγών.
- Εύκολη υποστήριξη για συναλλαγών σε συστήματα βάσεων δεδομένων ώστε να αναπτυχθούν.
- Παρέχει μια απλούστερη διεπαφή για την προγραμματιστική διαχείριση των συναλλαγών σχετικά με άλλες πολύπλοκες υλοποιήσεις όπως το *JTA*.

4.6 Μεταφορά Δεδομένων με DTOs και Dozer

Στο προηγούμενο κεφάλαιο αναφερθήκαμε στο ότι πλεον μεταβήκαμε από την μεταφορά δεδομένων στη μεταφορά αντικειμένων για την επικοινωνία Πελάτη-Εξυπηρετητής. Οι ασύγχρονες κλήσεις του *GWT* μας επιτρέπουν να ορίσουμε εύκολα τις μεθόδους που κάνουν αυτή τη μεταφορά των αντικειμένων, τώρα θα μιλήσουμε για αυτά τα αντικείμενα.

Η γλώσσα της *Java* διαθέτει στη βιβλιοθήκη της μια ειδική διεπαφή με όνομα `java.lang.Serializable`[58]. Όποια κλάση υλοποιεί αυτή τη διεπαφή γίνεται σειριοποιήσιμη και μπορεί πλέον να είναι κατάλληλη προς αποθήκευση στο δίσκο και εκπομπής σε ένα δίκτυο καθώς οι ειδικές δομές δεδομένων που απαιτούνται στη δημιουργία ενός αντικειμένου στην εικονική μηχανή της *Java* αλλάζουν μορφή ώστε να είναι μεταφέρσιμες ή αποθηκεύσιμες. Υπάρχουν λόγοι που δεν γίνεται όλες οι κλάσεις που αναπτύσωνται να είναι σειριοποιήσιμες αλλά δεν θα συζητηθούν σε αυτή την αναφορά. Το *GWT* επιβάλει με τη προγραμματιστική διεπαφή του οι κλάσεις που μεταφέρονται μεταξύ Πελάτη και Εξυπηρετητής να είναι σειριοποιήσιμες ώστε να επιτευχθεί η επικοινωνία και αποτελεί υποχρέωση του προγραμματιστεί να τις σχεδιάσει.

Η πιο απλή τεχνική είναι αυτή των αντικείμενων μεταφοράς δεδομένων *Data Transfer Objects - DTO*[13]. Τα *DTO* είναι σειριοποιήσιμες κλάσεις αντικειμένων τα οποία περιέχει πρωτογενή πεδία δεδομένων (π.χ *int,char,boolean*) και σειριοποιήσιμων αντικειμένων. Εφόσον κατασκευαστεί ένα τέτοιο αντικείμενο και συμπληρωθεί με τα κατάλληλα δεδομένα, τότε είναι έτοιμο για μεταφορά. Το κύριο μειονέκτημα των *DTO* είναι η ύπαρξη επιπλέον κλάσεων που περιγράφουν ένα μοντέλο που πιθανότατα έχει ήδη αποτυπωθεί σε άλλες κλάσσεις. Για παράδειγμα, το αντικείμενο της κλάσης `Car` το οποίο χρησιμοποιείται από τη *Hibernate* απαιτεί την μια κλάση με όνομα `CarDTO` η οποία είναι σειριοποιήσιμη. Οι επιπλέον αυτές κλάσεις χρειάζονται μερικές γραμμές κώδικα ώστε να αντιγραφουν τα πεδία της μιας κλάσης στην άλλη. Παρόλα αυτά η χρήση των *DTOs* είναι αναγκαία για την εφαρμογή μας και για να διευκολύνουμε την ανάπτυξη θα χρησιμοποιήσουμε το εργαλείο *Dozer*.

Το *Dozer*[20] έχει τη δυνατότητα να αντιστοιχεί πεδία όμοιων κλάσσεων και να αντιγράφει τις τιμές αυτών των πεδιών από ένα αντικείμενο σε ένα άλλο. Έτσι επιλύουμε το πρόβλημα των επιπλέον γραμμών κώδικα, καθώς αντικείμενα κλάσσεων που δεν είναι μεταφέρσιμα μπορούν να αντιγραφούν στα αντίστοιχα μεταφέρσιμα αντικείμενα τους με τη κλήση της μεθόδου `map()` από το στιγμιότυπο του *Dozer* στην εφαρμογή. Το *Dozer* υποστηρίζει αντιστοίχιση απλών πεδίων, πολύπλοκων τύπων, αμφίδρομη αντιστοίχιση, υπονοούμενη,ρητή καθώς και αναδρομική αντιστοίχιση κλάσεων. Δεν περιορίζεται στην απλή αντιστοίχιση μεταξύ ονομάτων και πεδίων, αλλά αυτομάτως αναλαμβάνει και μετατροπές μεταξύ διαφορετικών τύπων αντικειμένων. Τα περισσότερα σενάρια μετατροπής υποστηρίζονται με αφαιρετικό τρόπο ενώ παράλληλα δίνεται η δυνατότητα για λεπτομερή καθορισμό ξεχωριστών σεναρίων μέσω ρυθμίσεων που περέχονται μέσο ενός κειμένου γραμμένο σε *XML*.

4.7 Δοκιμή λογισμικού με JUnit & GWTTestCase

Το *JUnit*, είναι ένα προγραμματιστικό πλαίσιο για την ανάπτυξη δοκιμαστικών ρουτίνων σε εφαρμογές λογισμικού για την γλώσσα προγραμματισμού *Java*. Το *JUnit* έχει παίξει καθοριστικό ρόλο στην

οδηγούμενη με δοκιμές ανάπτυξη λογισμικού. Μπορεί να ενσωματωθεί στη διαδικασία μεταγλώτισης και κατασκευής του εκτελέσιμου μιας εφαρμογής ώστε μέσω τον δοκιμών να αποφασιστεί αν η εφαρμογή πληρεί τις κατάλληλες προϋποθέσεις για να εκδοθεί στο ευρύτερο κενό.

Οι δοκιμές αυτές μπορούν να επικεντρώνονται σε υλοποιημένες μεθόδους της εφαρμογής, να ελέγχεται η σωστή διασύνδεση των επι μέρους συστατικών της εφαρμογής και ο έλεγχος τιμών σε δεδομένων σύμφωνα με προδιαγραφές που έχουν τεθεί.

Στην περίπτωση του *GWT* το πλαίσιο *JUnit* είναι σχεδόν εξ'ολοκλήρου γραμμένο σε *Java* και το πλαίσιο *JUnit* υποστηρίζεται ως ενδεδειγμένο πλαίσιο ανάπτυξης δοκιμών. Επιπλέον, το *GWT* εμπεριέχει μία ειδική κλάση δοκιμών , υποκλάση της κλάσης `TestCase` του *JUnit*, την κλάση `GWTTestCase`, η οποία μπορεί να εφαρμοστεί σε δοκιμές σε κώδικα που απαιτεί *JavaScript* κατά την εκτέλεσή τους ή εξομοίωση γεγονότων που παράγωνται από τον περιηγητή του χρήστη (*DOM events*).

Κεφάλαιο 5

Σχεδιασμός νέων υπηρεσιών στα πλαίσια του WiseUI

Στο κεφάλαιο αυτό θα συζητηθούν λεπτομέριες για τον σχεδιασμό και την υλοποίηση της εφαρμογής *WiseUI*. Θα παρουσιαστούν οι περιπτώσεις χρήσης της, γενική και ειδική περιγραφή της αρχιτεκτονικής και το πεδίο των δεδομένων της εφαρμογής. Σε επίπεδο υλοποίησης θα τονιστεί ο τρόπος χρήσης των τεχνολογίων που έχουμε αναφαίρει στα προηγούμενα κεφάλαια.

5.1 Προδιαγραφές & Χρήση του WiseUI

Όπως αναφέρθηκε και στο εισαγωγικό κεφάλαιο ο σκοπός του *WiseUI* είναι να αντικαταστίσει ή να συμπληρώσει υπάρχοντα δύσχρηστα εργαλεία (εφαρμογές κονσόλας), παρέχωντας λειτουργικότητα σε ένα εύχρηστο γραφικό περιβάλλον. Η βασική λειτουργικότητα είναι αυτή που επιτρέπει την δέσμευση κόμβων του ΑΔΑ σε ΠΥ και την εκτέλεση πειραματών σε αυτή. Θα αντλήσουμε αυτή τη βασική λειτουργικότητα από τα υπάρχοντα εργαλεία χρησιμοποιώντας κομμάτια λογισμικού από αυτά[22]. Άκολουθεί μια περιγραφή της διαδικασίας :

1. Προετοιμασία της διαδικασίας επιλέγωντας την ΠΥ που θα ήθελε ο χρήστης να χρησιμοποιήσει. Ορισμός των *URLs* και των ενδοιάμεσων αντικειμένων (*Java Proxy Objects*) για την επικοινωνία του χρήστη με τις υπηρεσίες της ΠΥ.
2. Ταυτοποίηση και εξουσιοδότηση χρήστη μέσω των μεθόδων της διεπαφής *SNAA API*.
3. Πραγματοποίηση κρατήσεων σε κόμβους του ΑΔΑ της ΠΥ μέσω των μεθόδων της διεπαφλης *RS API*.
4. Προετοιμσία της μονάδας του ελεγκτή για τη λήψη δεδομένων και ανάκτηση στιγμιοτύπου για τον έλεγχο του πειράματος στον Πελάτη από το *iWSN API* της ΠΥ.
5. Πειραματισμός με τη χρήση κλήσεων στις μεθόδους του *iWSN API*.

Καθώς το έργο υλοποιείται ως εφαρμογή ιστού και χρησιμοποιεί *state of the art* τεχνολογίες μπορούμε να επεκτείνουμε τη βασική λειτουργικότητα προς όφελος του χρήστη.

1. Με τη χρήση μιας βάσης δεδομένων ως αποθηκευτικό χώρο ο χρήστης μπορεί να διατηρεί εγγραφές με πληροφορίες για διάφορετικες ΠΥ που υλοποιούν τις διεπαφές της πλατφόρμας *WISEBED*.

2. Μερικές από τις υπηρεσίες μιας ΠΥ δεν χρειάζονται ταυτοποίηση. Συνήθως είναι υπηρεσίες που παρέχουν πληροφορίες προς το κοινό για την υποδομή. Τέτοιες πληροφορίες πρέπει να παρέχωνται μέσω της εφαρμογής στον χρήστη σε διάφορες μορφές.

3. Οι κρατήσεις αναπαριστούνται με διαισθητικό τρόπο χρησιμοποιώντας ένα ημερολόγιο εντός της εφαρμογής.

4. Κατά τον πειραματισμό του χρήστη με την ΠΥ, δίνεται όσο το δυνατόν περισσότερος έλεγχος του πειράματος και της ΠΥ στο χρήστη ανάλογα με την εξουσιοδότηση που τον ενδιαφέρει.

5. Παροχή της βάσης δεδομένων του Εξυπηρετητής της εφαρμογής για την αποθήκευση των πειραμάτων (προγράμματα λειτουργίας).

6. Ζωντανή παραλαβή δεδομένων από τους κόμβους στον πειραματιστή.

7. Το πρότυπο της *WiseML* έχει πλέον καθιερωθεί στην κοινότητα του *WISEBED* είναι χρήσιμο λοιπόν να δίνονται λεπτομέριες για μια ΠΥ ή ένα πείραμα σε ένα έγγραφο αυτής της μορφής.

8. Χρήσιμες ενδείξεις μέσω της γραφικής διεπαφής σε περίπτωση σφαλμάτων και εξεαιρέσεων.

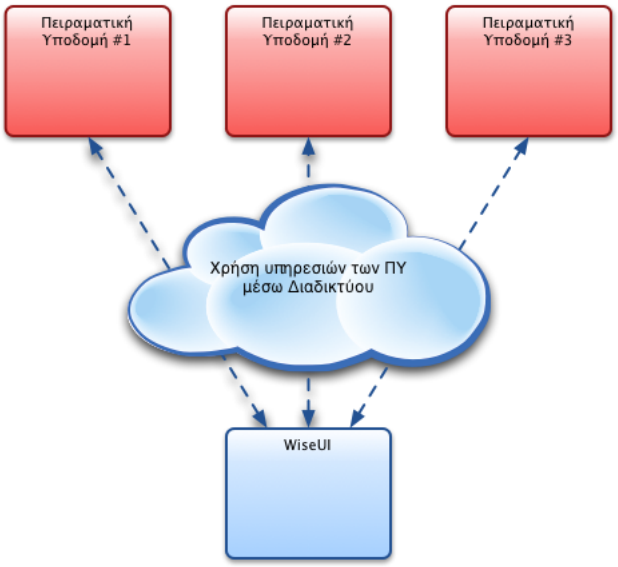
Όλες οι παραπάνω προδιαγραφές συνιστούν τον στόχο του έργου *WiseUI*, ο οποίος είναι ένα λειτουργικό και εύκολο στο χρήση εργαλείο για τον πειραματιστή. Η γραφική διεπαφή πρέπει να είναι αρκετά διαισθητική δίνωντας στον χρήστη τον έλεγχο και τις πληροφορίες του πειράματος και της ΠΥ που το φιλοξενεί.

5.2 Συνολική Αρχιτεκτονική Εφαρμογής

Ενώ η εφαρμογή έχει αναπτυχθεί με βάση το μοντέλο Πελάτη-Εξυπηρετητή, αυτές οι δύο οντότητες είναι ισχυρά εξαρτημένες μεταξύ τους σε σημείο που δεν μπορεί η εφαρμογή να είναι ολοκληρωμένη χωρίς μια απο αυτές. Ενώ ο Εξυπηρετητήςς φροντίζει να παρέχεται η βασική λειτουργικότητα, ο Πελάτης με τη γραφική διεπαφή είναι αυτός που παρέχει στο χρήστη τον έλεγχο του πειράματος και την παρουσίαση της πληροφορίας. Παράλληλα ο Πελάτης δεν θα μπορούσε να είναι από μόνος του λειτουργικός, καθώς το περιβάλλον στο οποίο είναι εγκατεστημένος, το οποίο είναι ένας περιηγητής ιστού δεν επιτρέπει την χρήση ορισμένου απαραίτητου λογισμικού. Οπότε, οι δύο αυτές οντότητες καλούνται να συνυπάρχουν και να θεωρούνται ως μια εννιαία εφαρμογή, παρόλο που εγκαθιστούνται σε διαφορετικά μηχανήματα στις περίσσότερες περιπτώσεις.

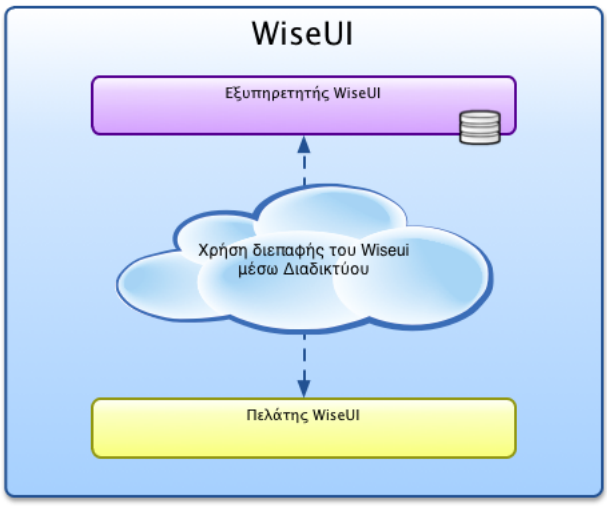
Το παρακάτω σχήμα μας δίνει μια απλουστευμένη αναπαράσταση της επικοινωνίας του *WiseUI* με ΠΥ που ακολουθούν την πλατφόρμα *WISEBED*. Οι απομακρυσμένες ΠΥ παρέχουν τις υπηρεσίες των

διεπαφών *SNAA,RS,iWSN* μέσω διαδικτύου ως υπηρεσίες ιστού. Το *WiseUI* κάνει κλήση στις μεθόδους των διεπαφών σύμφωνα με τον χειρισμό από τον χρήστη της εφαρμογής και στο επιστρέφονται τα αποτελέσματα των κλήσεων αυτών πίσω στην εφαρμογή.

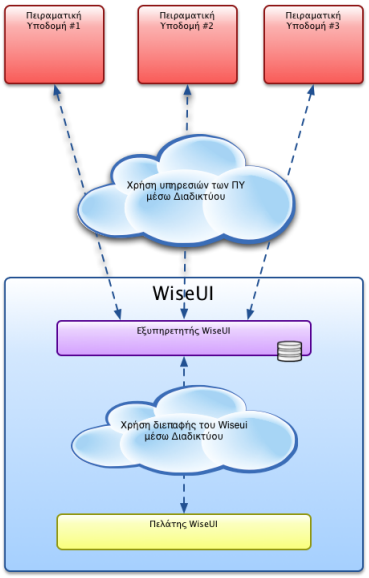


Εικόνα 5.1: WiseUI-ΠΥ επικοινωνία

Όπως έχουμε αναφέρει, το μοντέλο που ακολουθεί η εφαρμογή είναι αυτό του Πελάτη-Εξυπηρετητή με τον ρόλο του Πελάτη (Πελάτης *WiseUI*) να αναλαμβάνει το λογισμικό ανεπτυγμένο με τη χρήση του *GWT* και εγκαθιστάται ως κώδικας *JavaScript* στον περιηγητή του χρήστη. Το ρόλο του Εξυπηρετητή (Εξυπηρετητής *WiseUI*) το λογισμικό που εγκαθιστάται σε ένα Εξυπηρετητής *Java* εφαρμογών και συνοδεύεται από μια βάση δεδομένων. Επί της ουσίας η επικοινωνία μεταξύ Πελάτη και Εξυπηρετητή είναι μια παραδοσιακή επικοινωνία εφαρμογών ιστού με κύριο όχημα των δεδομένων να είναι τυπικές αιτήσεις-αποκρίσης μέσω του πρωτοκόλλου *HTTP*. Παράλλληλα, οι τεχνολογίες που χρησιμοποιούμε επιτρέπουν η επικοινωνία αυτή να είναι ασύγχρονη, να είναι γραμμένη αποκλειστικά στη γλώσσα *Java* και να να μπορεί να μοντελοποιηθεί ως μια διεπαφή που χρησιμοποιεί ο Πελάτης και υλοποιεί ο Εξυπηρετητής, σχεδόν όμοια με τη διεπαφή που υλοποιεί το *WiseUI* με τις υπηρεσιές των ΠΥ. Οι εικόνες 5.2, 5.3 δείχνουν το σχήμα της συνολικής αρχιτεκτονικής του *WiseUI* αλλά και της επικοινωνίας του με τις ΠΥ.

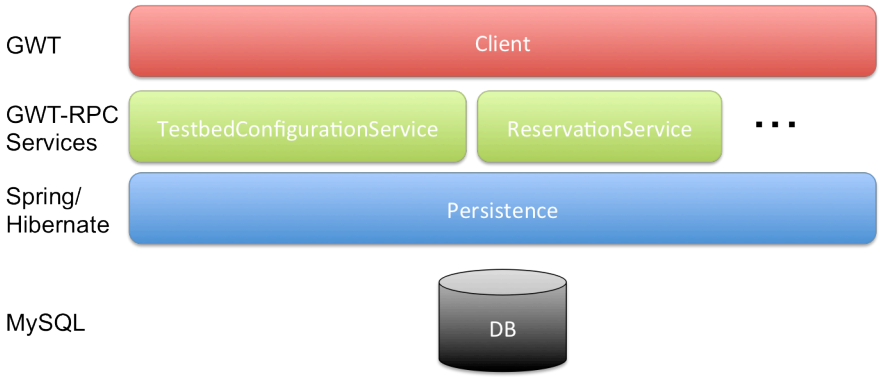


Εικόνα 5.2: Συνολική αρχιτεκτονική της εφαρμογής *WiseUI*



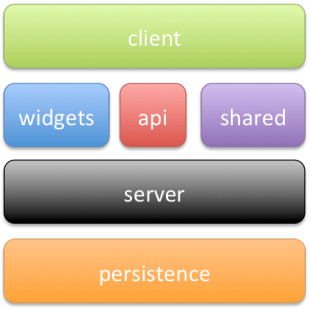
Εικόνα 5.3: WiseUI-ΠΥ επικοινωνία

Μια συνολική αναπαράσταση της αρχιτεκτονικής της εφαρμογής μπορεί να παρουσιαστεί ως ένα σύνολο επιπέδων λογισμικού. Εικόνα 5.5.



Εικόνα 5.4: Συνολική αρχιτεκτονική της εφαρμογής *WiseUI*

Ως προς την υλοποίηση του έργου ο κώδικας αποτελείται από τα παρακάτω πακέτα λογισμικού της *Java*.



Εικόνα 5.5: Τα πακέτα λογισμικού της εφαρμογής *WiseUI*

- Πακέτο *eu.wisebed.wiseui.client*. Περιέχει τον κώδικα που υλοποιεί της προδιαγραφές του Πελάτης του *WiseUI* γραμμένος σε *Java* και χρησιμοποιεί το *GWT* και αναλαμβάνει την παρουσίαση των δεδομένων που προέρχουνται από τον Εξυπηρετητή του *WiseUI*.
- Πακέτο *eu.wisebed.wiseui.server*: Το σύνολο του κώδικα που υλοποιεί τις υπηρεσίες του Εξυπηρετητή του *WiseUI*. Αναλαμβάνει να διαχειριστεί τα δεδομένα και τις υπηρεσίεςμιας ΠΥ και της βάσης δεδομένων της εφαρμογής.
- Πακέτο *eu.wisebed.wiseui.api*: Η δήλωση της διεπαφής μεταξύ Πελάτη και Εξυπηρετητής της εφαρμογής *WiseUI*.
- Πακέτο *eu.wisebed.wiseui.shared*: Κώδικας ο οποίος είναι διαμοιράσιμος μεταξύ Πελάτη και Εξυπηρετητής της εφαρμογής *WiseUI*.

- Πακέτο *eu.wisebed.wiseui.widgets*: Πακέτο με υλοποίηση γραφικών συστατικών (*widges*) για τη γραφική διεπαφή του πελάτη.

- Πακέτο *eu.wisebed.wiseui.persistence*: Το πακέτο αυτό περιέχει την υλοποίηση της βάσης δεδομένων και των υπηρεσιών της. Ο κώδικας μας είναι συμβατός με το πρότυπο *JPA*.

Σε αυτό το σημείο θα παρουσιάσουμε τη προγραμματιστική διεπαφή της εφαρμογής του *WiseUI*. Καθώς ο Εξυπηρετητής έχει αναλάβει την υλοποίηση της βασικής λειτουργικότητας της εφαρμογής, ο Πελάτης μέσω της γραφικής διεπαφής χρειάζεται να κάνει κλησεις μέσω της διεπαφής για να χρησιμοποιήσει αυτή τη λειτουργικότητα. Η διεπαφή αυτή είναι χωρισμένη σε διάφορα αρχεία ανάλογα με την ομάδα λειτουργικότητας που ανήκει η κάθε μέθοδος. Είναι δηλωμένη στο πακέτο `eu.wisebed.wiseui.api` το οποίο αποτελείται από 6 *Java* διεπαφές (*Java Interfaces*). Όλες αυτές ο διεπαφές κληρονομούν την κλάση `RemoteService` της βιβλιοθήκες του *GWT* ώστε να επιτευχθεί η ασύγχρονη επικοινωνία. Οι διεπαφές είναι οι εξής:

- `CalendarService`. Συγχρονισμός των αντικειμένων ημερομηνιας μεταξύ Πελάτη και Εξυπηρετητή.

Κώδικας 5.1: Διεπαφή CalendarService

```
public interface CalendarService extends RemoteService {
    Date addDays(Date date, int days);
    Date subtractDays(Date date, int days);
    Date addMonth(Date date);
    Date subtractMonth(Date date);
}
```

Η μέθοδος `addDays()` προσθέτει σε ένα στιγμιότυπο της κλάσης `Date` ένα ακέραιο αριθμό ημερών ώστε να αλλάξει η ημερομηνία, ενώ η `subtractDays()` αφαιρεί ένα αριθμό ημερών. Η `addMonth()` προσθέτει ένα μήνα στο στιγμιότυπο της ημερομηνίας και η `substractMonth` αφαιρεί ένα μήνα.

- `SessionManagmentService`. Ανάκτηση της εγκατάστασης του ΠΥ (*Testbed Setup*) από τον Πελάτη μέσω του Εξυπηρετητή σε γλώσσα *WiseML-XML*.

Κώδικας 5.2: Διεπαφή SessionManagementService

```
public interface SessionManagementService extends RemoteService {
    String getWisemlAsXml(String urn) throws WisemlException;
    Wiseml getWiseml(String urn) throws WisemlException;
}
```

Και οι δύο οι μέθοδοι έχουν σκοπό την επιστροφή μια περιγραφής της εγκατάστασης των κόμβων της ΠΥ σε μορφή *WiseML*. Η διαφορά τους έγκειται στον τύπο της επιστροφής. Στην πρώτη μέθοδο επιστρέφεται η περιγραφή ως στιγμιότυπο της κλάσης `String` αντίθετα η δεύτερη επιστρέ-

φει την εγκατάσταση ως στιγμίοτυπο της κλάσης `Wiseml` μια κλάση που είναι αντίγραφο (όχι απαραίτητα ακριβές) της κλάσης `eu.wisebed.ns.1.Wiseml` που παρέχεται από το *SessionManagement API* που ανήκει στο *iWSN API*.

- `TestbedConfigurationService`. Ανάκτηση και επεξεργασία εγγραφών-πληροφορίων των ΠΥ στη βάση δεδομένων στον Εξυπηρητητή.

Κώδικας 5.3: Διεπαφή TestbedConfigurationService

```
public interface TestbedConfigurationService extends RemoteService {
    List<TestbedConfiguration> getConfigurations();
    void storeConfiguration(TestbedConfiguration testbedConfiguration);
    void removeConfiguration(Integer id);
}
```

Αυτή η διεπαφή ευθύνεται για την διαχείρηση των εγγραφών για τις ΠΥ στη βάση δεδομένων στον Εξυπηρετητής της εφαρμογής *WiseUI*. Η μέθοδος `getConfigurations()` επιστρέφει μια λίστα από τις διαθέσιμες ΠΥ αποθηκευμένες στη βάση δεδομένων. Οι μέθοδοι `storeConfiguration()` και `removeConfiguration()` αποθηκεύουν και αφαιρούν αντίστοιχα εγγραφές-ΠΥ από τη βάση δεδομένων.

- `SNAAService`. Ταυτοποίηση και εξουσιοδότηση του χρήστη σε μια επιλεγμένη ΠΥ.

Κώδικας 5.4: Διεπαφή SNAAService

```
public interface SNAAService extends RemoteService {
    SecretAuthenticationKey authenticate(
            String endpointUrl,
            String urn,
            String username,
            String password)
            throws
            AuthenticationException;
}
```

Η μοναδική μέθοδος της διεπαφής , η `authenticate()`, ταυτοποιεί και εξουσιοδοτηθεί το χρήστη σε μια ΠΥ και επιστρέφει το μυστικό κλειδί ταυτοποίησης στον Πελάτη της εφαρμογής ως ένα στιγμιότυπο της κλάσης `SecretAuthenticationKey`.

- `ReservationService`. Δημιουργία και διαχείριση κρατήσεων των κόμβων σε μια επιλεγμένη ΠΥ, καθώς και επίβλεψη κρατήσεων άλλων χρηστών της ΠΥ.

Κώδικας 5.5: Διεπαφή ReservationService

```
public interface ReservationService extends RemoteService {
    List<SecretReservationKey> makeReservation(
            String rsEndpointUrl,
            List<SecretAuthenticationKey> secretAuthenticationKeys,
            ConfidentialReservationData confidentialReservationData)
            throws
            AuthenticationException, ReservationException;

    List<PublicReservationData> getPublicReservations(
            String rsEndpointUrl,
            Date pivotDate,
            Range dateRange) throws ReservationException;

    List<ConfidentialReservationData> getConfidentialReservations(
            String rsEndpointUrl,
            List<SecretAuthenticationKey> secretAuthenticationKeys,
            Date pivotDate,
            Range dateRange)
            throws
            AuthenticationException, ReservationException;

    void deleteReservation(
            String rsEndpointUrl,
            List<SecretAuthenticationKey> secretAuthenticationKeys,
            List<SecretReservationKey> secretReservationKeys)
            throws
            ReservationException;

    enum Range {
        ONE_DAY,
        WEEK,
        MONTH
    }
}
```

Οι παραπάνω μέθοδοι επιτρέπουν την διαχείριση και την παρακολούθηση των κρατήσεων που συμβαίνουν σε ένα χρονικό διάστημα στην ΠΥ. Η μέθοδος `makeReservation()` κάνει μια κράτηση για δεδομένη χρονική διάστημα σε ένα υποσύνολο των κόμβων της ΠΥ που έχει ταυτοποιηθεί ο χρήστης. Όπως φαίνεται και από τα ορίσματα αυτής της μεθόδου οι πληροφορίες παρέχονται από το στιγμιότυπο της κλάσης `ConfidentialReservationData` και η ένδειξη ταυτοποίησης του χρήστη παρέχεται από μια λίστα με αντικείμενα της κλάσης `SecretAuthenticationKey` (συνήθως αποτελείται από ένα μόνο αντικείμενο). Οι μέθοδοι `getPublicReservations()` και `getConfidentialReservations()` επιστρέφουνε για μια χρονική περίοδο κράτησείς που έχουνε γίνει στους κόμβους ενός ΑΔΑ. Η διαφορά τους έγκειται στο ο,τι η πρώτη μέθοδος επιστρέφει όλες τις κρατήσεις που έχουνε γίνει στην ΠΥ από όλους τους χρήστες. Αντίθετα η μέθοδος `getConfidentialReservations()` επιστρέφει τις κρατήσεις που έχουνε γίνει από το χρήστη

του *WiseUI*. Τέλος η `deleteReservation` για ένα ταυτοποιημένο χρήστη διαγράφει τις κρατήσεις που έχει κάνει ο ίδιος παρέχωντας και τα μυστικά κλειδία των κράτησεων.

- `ExperimentationService`. Μέθοδοι για τον πειραματισμό του χρήστη κατα τη διάρκεια μιας κράτησης. Διάφορες μέθοδοι για το φόρτωμα εικόνων την εκτέλεση πειραμάτων και τη λήψη αποτελεσμάτων αυτών.

Κώδικας 5.6: Διεπαφή ExperimentationService

```
public interface ExperimentationService extends RemoteService {
    void startExperimentController(
            String sessionManagementUrl,
            List<SecretReservationKey> secretReservationKeys,
            List<String> nodeUrns)
            throws
            ExperimentationException;

    void flashExperimentImage(
            List<SecretReservationKey> secretReservationKeys,
            Integer imageId,
            List<String> nodeUrns)
            throws
            ExperimentationException;

    void resetExperimentNodes(
            List<SecretReservationKey> secretReservationKeys,
            List<String> nodeUrns)
            throws
            ExperimentationException;

    Message returnExperimentMessage(
            List<SecretReservationKey> secretReservationKeys)
            throws
            ExperimentationException;

    String returnExperimentWiseMLReport(
            List<SecretReservationKey> secretReservationKeys)
            throws
            ExperimentationException;

    void stopExperimentController(
            List<SecretReservationKey> secretReservationKeys)
            throws
            ExperimentationException;

    List<BinaryImage> returnUploadedExperimentImages()
            throws
            ExperimentationException;
}
```

Η μέθοδος `startExperimentController()` σηματοδοτεί την έναρξη του πειράματος. Καλείται από τον χρήστη όταν έχει φτάσει η χρόνος έναρξης της κράτησης και φροντίζει την εγκαθίδρυση ενός ελεγκτή επικοινωνίας του Εξυπηρετητή του *WiseUI* με τις υπηρεσίες που παρέχει η ΠΥ και τηβ παραλαβή ενός στιγμιοτύπου `WSN` για τον έλεγχο του πειράματος. Η μέθοδος

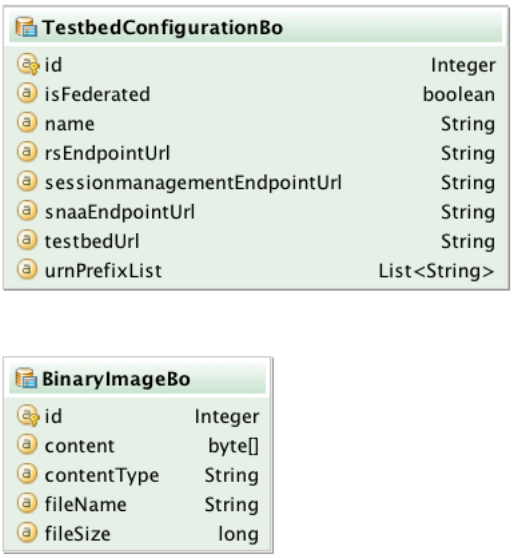
`flashExperimentImage()` δίνει εντολή να φορτωθεί στους κόμβους της κράτησης ένα πρόγραμμα λειτουργίας (*Experiment Image*) το οποίο είναι αποθηκευμένο στη βάση δεδομένων της εφαρμογής στον Εξυπηρετητή. Η μέθοδος `resetExperimentNodes()` δίνει την εντολή της επανεκκίνησης σε ένα σύνολο από κόμβους ώστε να επανεκτελεστεί το πρόγραμμα λειτουργίας που είναι ήδη φορτωμένο σε αυτούς. Για τη λήψη των δεδομένων από τους αισθητήρες των κόμβων καλείται ανα τακτά χρονικά διαστήματα η μέθοδος `returnExperimentMessage`, η οποία επιστρέφει τα δεδομένα των αισθητήρων με τη μορφή μηνυμάτων. Κατα τη διάρκεια της λήψης αυτών των μηνυμάτων , ο Εξυπηρετητής κατασκευάζει ένα έγγραφο περιγραφής του πειράματος στη γλώσσα *WiseML*. Με την κλήση της μεθόδου `returnExperimentWiseMLReport` επιστρέφεται και εμφανίζεται στο Πελάτη αυτό το κείμενο ως αναφορά της εκτέλεσης του πειράματος. Η μέθοδος `stopExperimentController()` τερματίζει τον ελεγκτή λήψης μηνυμάτων και απελευθερώνει το στιγμιότυπο ελέγχου `WSN`, όμως αυτό δεν σημαίνει απάραίτητη λήξη της διαδικασίας, καθώς ο χρήστης μπορεί να επανακκινήσει το πείραμα του όποτε αυτός το επιθυμεί. Τέλος, μια καθαρά βοηθητική μέθοδος άλλα ιδιαίτερως χρήσιμη είναι αυτή με το όνομα `returnUploadedExperimentImages()`. Αυτή η μέθοδος επιστρέφει μια λίστα με τις εγγραφές των προγραμμάτων λειτουργίας που είναι ανεβασμένες στη βάση δεδομένων της εφαρμογής. Τα προγράμματα λειτουργίας μπορεί να ανήκουν και σε άλλους χρήστες και για λόγους απλότητας της εφαρμογής δεν υλοποιήσαμε κάποια πολιτική ιδιοκτησίας των προγραμμάτων λειτουργίας στη βάση δεδομένων.

5.3 Αποθήκευση Δεδομένων & Πληροφορίων

Κατά την ανάπτυξη της εφαρμογής αποφασιστήκε η χρήση της βάσης δεδομένων να όσο πιο μινιμαλιστική γίνεται. Αυτό γίνεται καθώς η εφαρμογή μπορεί να εκμεταλευτεί και τις αποθηκευτικές ικανότητες των υπηρεσιών του ΠΥ για αποθήκευση πληροφοριών όπως τα μυστικά κλειδιά κρατήσεων ή ταυτοποίησης. Η προσθήκη τέτοιων δεδομένων στη βάση θα απαιτούσε ένα πιο σύνθετο μοντέλο από αυτό που θα παρουσιαστεί στη συνέχεια. Σε περίπτωση πρόσθηκής τέτοιων πληροφοριών θα ήταν αναπόφευκτος ο συγχρονισμός της βάσης δεδομένων του *WiseUI* με αυτές των ΠΥ που επικοινωνεί. Συνεπώς, αποφασίστηκε η βάση δεδομένων του να υποστηρίζει ένα μικρό και απλο μοντέλο δεδομένων και πληροφοριών που τροφοδοτεί ο ίδιος ο χρήστης στο σύστημα. Αντίθετα πληροφορίες και δεδομένα που παράγει η ΠΥ όπως τα μυστικά κλειδιά ταυτοποίησης και κράτησης επιλέξαμε να μην τα αποθηκεύουμε στη μνήμη αλλά να ανακτούνται εκ της ίδιας της ΠΥ.

Το μοντέλο του πεδίου δεδομένων (*Domain Model*)[19], το οποίο παρουσιάζεται στην εικόνα 5.6 αποτελείται από δύο οντότητες, οι οποίες αποθηκεύονται στη βάση δεδομένων της εφαρμογής. ΄Οπως βλέπουμε και οι δύο οντότητες καταλήγουν το όνομα τους σε `Bo` το οποίο σηματοδοτεί ότι αποτελούν για την εφαρμογή σημαντικά αντικείμενα του μοντέλου δεδομένων (*Business Objects - BOs*) . Επιπλέον και οι δύο οι οντότητες διαθέτουν ένα πεδίο `id` ώστε να μπορούν να ταυτοποιηθούν στα πλαίσια της βάσης δεδομένων.

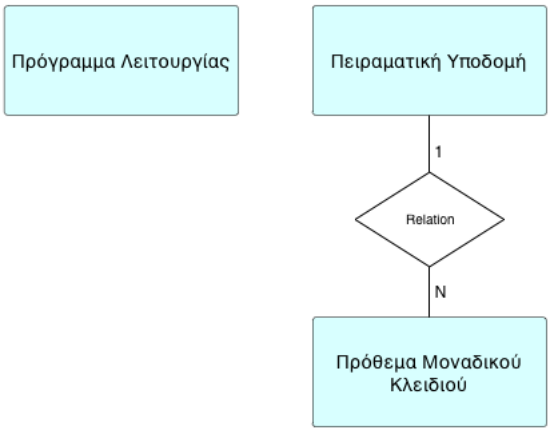
Η οντότητα με όνομα `TestbedConfigurationBo` είναι μια *ORM* αναπαράσταση μιας ΠΥ με τις



Εικόνα 5.6: Οι οντότητες της βάσης δεδομένων

πληροφορίες που τη συνοδεύουν όπως το όνομα, μια διεύθυνση στον ιστό και οι διευθύνσεις που αντιστοιχούν στις υπηρεσίες *SNAA, RS, SessionManagment* και μια ένδειξη στο αν η ΠΥ αυτή αποτελεί ενοποίηση (*Federation*) άλλων ΠΥ. Επίσης διαθέτει και μια λίστα από απο προθέματα μονάδικων ονομάτων (*urnPrefix*). Το πρόθεμα μοναδικού ονόματος είναι ένα μοναδικό αλφαριθμητικό που χρησιμοποιείται ως αναγνωριστικό της ΠΥ στο δίκτυο των ΠΥ του *WISEBED*. Η δομή της λίστας χρησιμοποιείται στην περίπτωση που έχουμε να κάνουμε με μια ενοποίηση από ΠΥ. Καθώς δεν χρησιμοποιοείται ιδιαίτερη η μορφή των ενοποιημένων ΠΥ η λίστα αυτή συχνά περιέχει μόνο ένα στοιχείο, παρόλα αυτά επιλέγουμε να χρησιμοποιούμε τη δομή ώστε σε μελλοντική ανάπτυξη της εφαρμογής να μην αντιμετοπίσουμε προβλήματα συμβατότητας. Οι λίστες και οι συλλογές στο κόσμο των αντικείμενο μεταφράζονται ως σχέσεις πληθικότητας (*Cardinality*)[7] στο σχεσιακό κόσμο, δηλαδή ως σχέσεις 1-προς-N,N-προς-1,N-προς-M. Στην περίπτωση μας η σχέση είναι 1-προς-N καθώς για την ενοποιημένη ΠΥ αντιστοιχούν N προθέματα μοναδικών ονομάτων, ενό για ένα πρόθεμα αντιστοιχεί μόνο μια ΠΥ.

Η δεύτερη οντότητα ονομάζεται `BinaryImageBo` και είναι *ORM* αναπαράσταση ενός προγράμματος λειτουργίας. Θέλουμε να αποθηκεύσουμε τα προγράμματα λειτουργίας στον Εξυπηρετητή *WiseUI* ώστε σε περίπτωση επανάληψης ενός πειράματος με ένα προηγούμενο πρόγραμμα λειτουργίας να μην ξαναχρειαστεί να το ανεβάσει ο χρήστης από τον Πελάτη. Πλέον, μπορεί να ανατρέξει στις εγγραφές της βάσης δεδομένων και να ανακτήση το παλιό αυτό πρόγραμμα λειτουργίας και να το φορτώσει απ'ευθείας στους κόμβους που έχει δεσμεύσει. Η οντότητα αυτή αποτελείται από το όνομα του αρχείου του προγράμματος λειτουργίας , το μέγεθος του τα δεδομένα του ως πίνακας από *byte* και μια περιγραφή του περιεχομένου του προγράμματος λειτουργίας. Η παρακάτω εικόνα 5.7 δίνει το απλό διάγραμμα οντοτήτων συσχετίσεων του σχήματος της βάσης δεδομένων της εφαρμογής.



Εικόνα 5.7: Διάγραμμα Οντοτήτων-Συσχετήσεων

Αναφέρθηκα παραπάνω ότι δεδομένα και πληροφορίες που παράγονται από την ΠΥ δεν αποθηκεύονται στη βάση δεδομένων. Επιλέξαμε αυτό το πολύ μικρό μοντέλο για τη βάση μας καθώς στόχος πρώτος κατά την ανάπτυξη της εφαρμογής ήταν η ένσωμάτωση των υπηρεσιών της ΠΥ. Μια προσθήκη όπως, τα αποτελέσματα των πειραμάτων από ένα χρήστη θα χρειάζοταν οντότητες όπως το πείραμα και ο χρήστης. Αντίθετα επιλέξαμε τα δεδομένα όπως τα μυστικά κλειδιά των κρατήσεων να μην αποθηκεύονται καθόλου στην εφαρμογή παρα μόνο κρατιούνται στη μνήμη κατά την εκτέλεση ενός πειράματος. Τα μυστικά κλειδιά ταυτοποίησης τα χρησιμοποιούμε ως αναγνωριστικό σύνδεσης και τα αποθηκεύουμε ως *cookies*[30] στον περιηγητή του χρήστη.

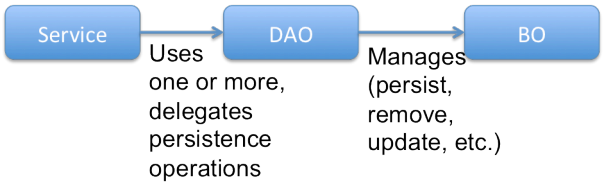
Στα πλαίσια της εφαρμογής αναπτύχθηκε μια διεπαφή για την επικοινωνία μεταξύ Εξυπηρετητή *WiseUI* και την επεξεργασία των δεδομένων που είναι αποθηκευμένα στη βάση δεδομένων. Η διεπαφή αυτή ονομάζεται `PersistenceService` και βρίσκεται στο πακέτο `eu.wisebed.wiseui.api`. Η διεπαφή παρουσιάζεται ως εξής :

Κώδικας 5.7: Διεπαφή PersistenceService

```
public interface PersistenceService {

    TestbedConfiguration storeTestbedConfiguration(TestbedConfiguration testbedConfiguration);
    TestbedConfiguration loadTestbedConfiguration(Integer id);
    void removeTestbedConfiguration(Integer id);
    List<TestbedConfiguration> loadAllTestbedConfigurations();
    BinaryImage storeBinaryImage(BinaryImage binaryImage);
    BinaryImage loadBinaryImage(Integer id);
    void removeBinaryImage(Integer id);
    List<BinaryImage> loadAllBinaryImages();

}
```

Η παραπάνω διεπαφή υλοποιείται στο πακέτο `eu.wisebed.wiseui.persistence`. Οι παραπάνω μέθοδοι αποθηκεύουν, προβάλουν, αφαιρούν το σύνολο ή ένα υποσύνολο των αντικείμενων που αποτελούν τις οντότητες που περιγράψαμε παραπάνω. Οι κλάσσεις των αντικείμενοων που προέρχονται από

τις μεθόδους αυτές δεν είναι *Business Objects* αλλά εκφράζουν τις ίδιες οντότητες τις ΠΥ και του προγράμματος λειτουργίας και είναι μεταφέρσιμα (*DTOs*). Η υλοποίηση βασίζεται στο ευρέως διαδεδομένο σχεδιαστικό πρότυπο των αντικειμένων πρόσβασης δεδομένων (*Data Access Objects - DAOs*)[12].



Εικόνα 5.8: Τα DAOs και τα BOs

Η πρόσβαση στη βάση δεδομένων γίνεται μέσω του διαχειριστή συναλαγών του πλαισίου *Spring* (*Spring Transaction Managements*). Με αυτές τις τεχνολογίες ο προγραμματιστής δεν χρειάζεται να υλοποιήσει μια μονάδα διαχείριση συναλλαγών. Μπορεί να χρησιμοποιήσει έτοιμες λύσεις δηλώνοντας αυτές στον κώδικα και στις ρυθμίσεις τις εφαρμογής.

5.4 Εξυπηρετητής της εφαρμογής

Ο Εξυπηρετητής είναι το κέντρο της εφαρμογής καθώς συνεργάζεται με όλα τα μέρη της, δηλαδή τον Πελάτης της, τις ΠΥ και την βάση δεδομένων της. Για να επιτευχθεί αυτό πρέπει να υλοποιησεί όλες τις διεπαφές που έχουν οριστεί στο πακέτο *eu.wisebed.wiseui.api*. Ένας παραδοσιακός εξυπηρετητής δέχεται αιτήσεις από τους πελάτες για πόρους όπως ιστοσελίδες και ο εξυπηρετητής απαντάει επιστρέφωντας τον αντίστοιχο πόρο. Στην περίπτωση του *WiseUI* τα πράγματα είναι διαφορετικά. Επιθυμούμε ο πελάτης να κάνει αιτήσεις για την ενεργοποίηση/κλήση μιας υπηρεσίας. Ο εξυπηρετητής εκτελεί την υλοποίηση της υπηρεσίας και επιστρέφει το αποτέλεσμα της κλήσης στον πελάτη. Οι κλήσεις του πελάτη πρέπει να είναι ασύγχρονες, δηλαδή ο πελάτης δεν θα πρέπει να περιμένει την απάντηση μιας αίτησης απο τον εξυπηρετητή, αλλά να συνεχίζει την εκτέλεση του.Η υλοποίηση του εξυπηρετητή περιλαμβάνεται στο πακέτο *eu.wisebed.wiseui.server*, η κύρια τεχνολογία που θα χρησιμοποιηθεί είναι η τεχνολογία των *Java Servlets* και συγκεκριμένα την υλοποίηση που προσφέρει το *GWT* με όνομα `RemoteServiceServlet`. Επίσης, στόχος του Εξυπηρετητή είναι και η αντιγραφή των δεδομένων που προέρχονται από την βάση δεδομένω και από τις ΠΥ σε μεταφέρσιμα αντικείμενα (*Data Transefer Objects - DTOs*). Αυτό επιτυγχάνεται με τη χρήση της κλήσης της συνάρτησης του `mapper.map()` του εργαλείου *Dozer*. Ακολουθεί η υλοποίηση των `TestbedConfigurationService` και `SNAAService` ως παράδειγμα :

Κώδικας 5.8: Υλοποίηση του SNAAService

```
@Singleton
public class SNAAServiceImpl extends RemoteServiceServlet implements SNAAService {
    ...
    @Inject
    public SNAAServiceImpl(final DozerBeanMapper mapper) {
        this.mapper = mapper;
```

```
    }

    @Override
    public SecretAuthenticationKey authenticate(final String endpointUrl,
                                                final String urn,
                                                final String userName,
                                                final String password) throws AuthenticationException {
        ifNullOrEmptyArgument(endpointUrl, "SNAA Endpoint URL not set!");
        ifNullOrEmptyArgument(urn, "URN not set!");
        ifNullOrEmptyArgument(userName, "User name not set!");
        ifNullOrEmptyArgument(password, "Password not set!");

        final SNAA snaaService = SNAAServiceHelper.getSNAAService(endpointUrl);
        final AuthenticationTriple triple = createTriple(urn, userName, password);
        eu.wisebed.api.snaa.SecretAuthenticationKey key;

        try {
            key = snaaService.authenticate(Arrays.asList(triple)).get(0);
        } catch (final AuthenticationExceptionException e) {
            LOGGER.error(AUTHENTICATION_FAILED, e);
            throw new AuthenticationException(e.getMessage(), e);
        } catch (final SNAAExceptionException e) {
            LOGGER.error(AUTHENTICATION_FAILED, e);
            throw new AuthenticationException(e.getMessage(), e);
        }

        return mapper.map(key, SecretAuthenticationKey.class);
    }
    ...
}
```

Ο παραπάνω κώδικας δείχνει την διαδικασία ταυτοποίησης του χρήστη. Αρχικά ελέγχει τις παραμέτρους της μεθόδου `authenticate()` και στη συνέχεια δημιουργεί ένα στιγμιότυπο της υπηρεσίας `SNAA` με τη χρήση της μεθόδου `SNAAServiceHelper.getSNAAAservice()` ώστε να κάνει κλήση της συνάρτησης `authenticate` για την ταυτοποίηση του χρήστη με την επιλεγμένη ΠΥ. Η επιτυχής ταυτοποίηση επιστρέφει ένα μυστικό κλειδί ταυτοποίησης στον χρήστη, ενώ σε περίπτωση κάποιου σφάλματος μια εξαίρεση της *Java* (*Java Exception*) καλείται και διακόπτεται η εκτέλεση της μεθόδου. Η εξαίρεση αυτή βασιζόμενη στους μηχανισμούς του `RemoteServiceServlet` φτάνει και στον πελάτη ώστε να ειδοποιήσει το χρήστη για το σφάλμα κάτα την ταυτοποίηση.

Κώδικας 5.9: Υλοποίηση του TestbedConfigurationService

```
public class TestbedConfigurationServiceImpl extends RemoteServiceServlet implements
     TestbedConfigurationService {
    ...
    @Inject
    public TestbedConfigurationServiceImpl(final PersistenceService persistenceService) {
        this.persistenceService = persistenceService;
    }

    @Override
    public List<TestbedConfiguration> getConfigurations() {
        return persistenceService.loadAllTestbedConfigurations();
    }
```

```
    @Override
    public void storeConfiguration(final TestbedConfiguration testbedConfiguration) {
        persistenceService.storeTestbedConfiguration(testbedConfiguration);
    }

    @Override
    public void removeConfiguration(final Integer id) {
        persistenceService.removeTestbedConfiguration(id);
    }
}
```

Στον παραπάνω κώδικα βλέπουμε πως ο Εξυπηρετητής χρησιμοποιεί την διεπαφή *Persistence Service* για την επικοινωνία του με τη βάση δεδομένων της εφαρμογής. Παρατηρούμε πως η ίδια η υπηρεσία επιστρέφει αντικείμενα τα οποία μπορούν απευθείας να μεταφερθούν στον πελάτη χωρίς ο εξυπηρετητής να χρειάζεται να τα αντιγράψει σε μεταφέρσιμα αντικείμενα χρησιμοποιώντας το εργαλείο *Dozer*.

Στη συνέχια θα σταθούμε στο κομμάτι του πειραματισμού στην πλευρά του Εξυπηρετητή. Όπως έχει αναφερθεί για την υλοποίηση των υπηρεσίων του Εξυπηρετητή χρησιμοποιείται η τεχνολογία των *Servlets* στα πλαίσια ενός εξυπηρετητή εφαρμογών για *Java* εφαρμογές π.χ *Tomcat, Resin, Websphere,* κ.α. Γενικά τα *Servlets* χρησιμοποιούνται ως παροχοί υπηρεσιών και είναι μοναδικά αντικείμενα με χρόνο ζωής ίσο με αυτό του εξυπηρετητή της εφαρμογής. Δηλαδή, όσο εκτελείται ο εξυπηρετητής, θα παρέχονται και τα *Servlets*. Αναφέρθηκε ότι τα *Servlets* είναι μοναδικά αντικείμενα(*Singleton*)[61], δηλαδή για κάθε μια υπηρεσία δημιουργείται ένα και μοναδικό στιγμιότυπο της υπηρεσίας μέχρις ώτου φτάσει ο τερματισμός του εξυπηρετητή ή ένα σφάλμα τέτοιο ώστε να καταρεύσει ολόκληρο το σύστημα του Εξυπηρετητή.

Εφόσον τα στιγμιότυπα των *Servlets* είναι μοναδικά, άρα και κοινόχρηστα από τους χρήστες της εφαρμογής και καθώς επιθυμούμε να χρησιμοποιήσουμε την βάση δεδομένων όσο το λιγότερο γίνεται, χρειάζεται να αναπτυχθεί ένας μηχανισμός που υποστηρίζει πολλαπλά ταυτόχρονα πειράματα. Για αυτό το λόγο κατασκευάστηκε η κλάση με όνομα `ExperimentController` η οποία υλοποιεί την διεπαφή `Controller` η οποία έχει οριστεί από τις υπηρεσίες *iWSN* των ΠΥ του *WISEBED*. Ακολουθεί ένα μέρος της υλοποίησης του ελεγκτή `ExperimentController`.

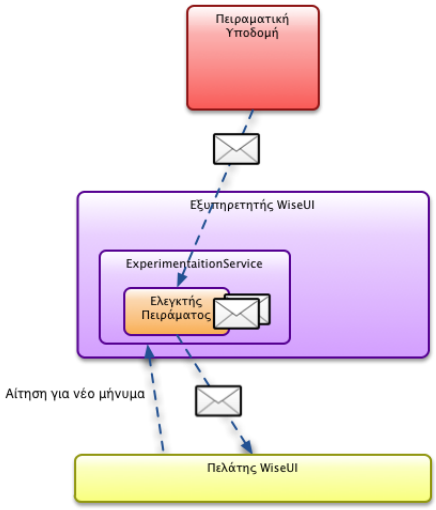
Κώδικας 5.10: Υλοποίηση του ExperimentController

```
@WebService(serviceName = "ControllerService",
        targetNamespace = Constants.NAMESPACE_CONTROLLER_SERVICE,
        portName = "ControllerPort",
        endpointInterface = Constants.ENDPOINT_INTERFACE_CONTROLLER_SERVICE)
public class ExperimentController implements Controller {

    @Inject
    public ExperimentController(
            final Mapper mapper,
            final WiseML wiseml,
            final Setup wisemlSetup,
            final Trace wisemlTrace,
            final WiseMLController wisemlController,
            final Queue<RequestStatus> requestStatusQueue,
            final Queue<Message> messageQueue,
            final Queue<String> notificationQueue) {
        this.setMapper(mapper);
        ..
        this.setNodeUrns(new ArrayList<String>());
    }

    public final void startSessionManagement() throws ExperimentationException{
        try{
            ifNull(sessionManagement,"Session management service is null");
        }catch(RuntimeException cause){
            throw new ExperimentationException(cause.getMessage());
        }

        String wsnEndpointURL=null;
        try {
            wsnEndpointURL = sessionManagement.getInstance(
                copyRsToWsn(secretReservationKeys),localEndpointUrl);
        } catch (ExperimentNotRunningException_Exception cause) {
            throw new ExperimentationException("Experiment not running on "
                    + localEndpointUrl);
        } catch (UnknownReservationIdException_Exception cause) {
            throw new ExperimentationException("Unknown reservation Id");
        }

        wsn = WSNServiceHelper.getWSNService(wsnEndpointURL);
    }

    public void freeSessionManagement() throws ExperimentationException{

        try{
            ifNull(sessionManagement,"Session management service is null");
        }catch(RuntimeException cause){
            throw new ExperimentationException(cause.getMessage());
        }

        try {
            sessionManagement.free(copyRsToWsn(secretReservationKeys));
        } catch (ExperimentNotRunningException_Exception cause) {
            throw new ExperimentationException("Experiment is not running");
        } catch (UnknownReservationIdException_Exception cause) {
            throw new ExperimentationException("Unknown reservation ID provided");
        }
```

```
    }

    public void publish() throws MalformedURLException {

        try{
            ifNullOrEmpty(localEndpointUrl,"Empty or null URL string");
        }catch(RuntimeException cause){
            throw new MalformedURLException(cause.getMessage());
        }

        String bindAllInterfacesUrl =
            convertHostToZeros(localEndpointUrl);
        String key = secretReservationKeys.get(0).getSecretReservationKey()

        Endpoint endpoint = Endpoint.publish(bindAllInterfacesUrl, this);
        endpoint.setExecutor(Executors.newCachedThreadPool());
    }


    public void receive(List<eu.wisebed.api.common.Message> msgs) {
        for(eu.wisebed.api.common.Message msg : msgs) {

            // set experiment message
            final String source = msg.getSourceNodeId();
            final String timeStamp = msg.getTimestamp().toXMLFormat();
            final String data = StringUtils.toHexString(msg.getBinaryData());
            final String level = msg.getBinaryData()[1] == 0x00 ? "DEBUG" : "FATAL";

            // map message and put it in queue
            Message message = mapper.map(msg, Message.class);
            message.setTimestamp(
                    msg.getTimestamp().toGregorianCalendar().getTime());
            messageQueue.add(message);

            // set WiseML message
            eu.wisebed.wiseml.model.trace.Message traceMessage = new
                eu.wisebed.wiseml.model.trace.Message();
            traceMessage.setTimestamp((long)msg.getTimestamp().getMillisecond());
            traceMessage.setData("[" + level +"][" + data +"]");
            traceMessage.setId(source);

            // add WiseML message to trace
             if(wisemlTrace.getChildren() == null) {
                wisemlTrace.setChildren(Trace.listFactory());
            }
            wisemlTrace.getChildren().add(traceMessage);
        }
    }
    ...
}
```

Η κλάση `ExperimentController` όχι μόνο υλοποιεί τις βασικές λειτουργίες του ελεγκτή μηνυμάτων από τις ΠΥ αλλά διατηρεί ως γνωρίσματα το μυστικό κλειδί της κράτησης, το οποίο χρησιμοποιείται κα ως αναγνωριστικό του ελεγκτή στο περιβάλλον του Εξυπηρετητή, ένα στιγμιότυπο της υπηρεσίας `WSN`, δομές ουρών ώστε να αποθηκεύονται τα μηνύματα που φθάνουν με την ΠΥ στη σωστή σειρά και στιγμιότυπα κλάσεων για την κατασκευή του μιας αναφοράς του πειράματος γραμ-

μένη στη γλώσσα *WiseML*. Επιπλέον έχουν υλοποιηθεί μέθοδοι όπως οι *startSessionManagement()* και *freeSessionManagement()* για την λήψη και αποδεύμεση του στιγμιοτύπου `WSN`, η μέθοδος `publish()` η οποία εκδίδει σε ένα νέο νήμα εκτέλσης τον ελεγκτή ώστε να επιτευχθεί η επικοινωνία με την ΠΥ, χωρίς να παρεμποδίζει την λειτουργία του Εξυπηρετητή. Απαραίτητη είναι η υλοποίηση της μεθόδου `receive()` η οποία συλλέγει τα μηνύματα που φτάνουν από τος κόμβους της ΠΥ και τα αποθηκεύει στη δομή μιας ουράς δεδομένων ώστε να διατηρηθεί η σειρά παραλαβής. Παράλληλα, τα μηνύματα αυτά διαμορφώνονται ώστε να συνθέσουν την αναφορά σε *WiseML* η οποία επιστρέφει στο χρήστη όποτε αυτός την ζητήσει κατά τη διάρκεια του πειράματος. Χρησιμοποιούμε αυτό τον τρόπο τροφοδότησης μηνυμάτων καθώς η αρχιτεκτονική της εφαρμογής απαιτεί από τον Πελάτη του *WiseUI* να καλεί για ένα νέο μήνυμα ανά 500 *milliseconds*. Με αυτό τον τρόπο, όσο η ουρά των μηνυμάτων περιέχει κάποιο μήνυμα το επιστρέφει στο χρήστη με τη σωστή σειρά όποτε αυτός το ζητήσει. Τα μηνύματα αυτά περιέχουν την ταυτότητα του κόμβου που τα εκπέμπει, τον τύπο του μηνύματος και τη χρονοσφραγίδα της εκπομπής.



Εικόνα 5.9: Λήψη μηνυμάτων από τους κόμβους στον Εξυπηρετητή και μετά στον Πελάτη

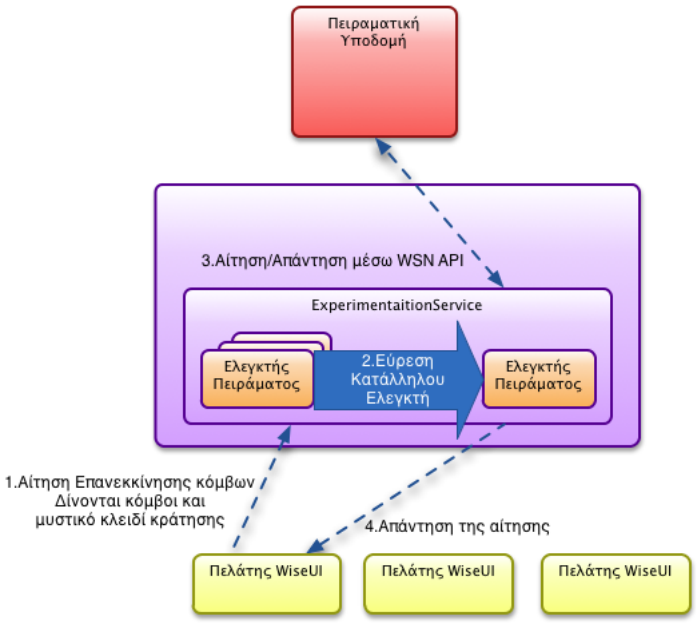
Με λίγα λόγια η κλάση `ExperimentController` αποτελεί την έκφραση ενός εν ενεργεία πειράματος για τον Εξυπηρετητή. Για να διαχειριστεί ο Εξυπηρετητής ταυτόχρονα ζωντανά πειράματα, χωρίς να χρησιμοποιείσει την βάση δεδομένων, πρέπει να τα αποθηκεύει στη μνήμη του. Με την έναρξη ενός πειράματος ένα ενεργό πείραμα τοποθετείται στη λίστα των εν ενεργεία πειραμάτων και και Με την λήξη του πειράματος αφαιρείται το πείραμα από τη λίστα.˝Έτσι με αυτό τον τρόπο το κάθε ενεργό πείραμα, από οποιονδήποτε χρήστη, διατηρείται στη μνήμη του Εξυπηρετητή. Η κλάση `ExperimentationServiceImpl` , υλοποιεί την διεπαφή `ExperimentationService` και διατηρεί μια λίστα αντικειμένων `ExperimentController` , ως λίστα πειρμάτων. Αναφέραμε παραπάνω ότι τα μυστικά κλειδιά κράτησης μπορούν να λειτουργήσουν ως αναγνωριστικά του πειράματος, έτσι ένα αντικείμενο `ExperimentController` μπορεί να ανακτηθεί από τη λίστα μέσω του μυστικού κλειδιού κράτησης. Απο αυτό το σημείο μπορεί να συνε-

χιστεί ο πειραματισμός καθώς ανακτήθηκε ο ελεγκτής με το κατάλληλο κλειδί, ο οποίος φέρει και το κατάλληλο στιγμιότυπο WSN. Ακολουθεί παράδειγμα από την υλοποίηση ExperimentationServiceImpl η μέθοδος resetExperimentNodes().

Κώδικας 5.11: Υλοποίηση της μεθόδου resetExperimentNodes()

```
public void resetExperimentNodes(
        List<SecretReservationKey> secretReservationKeys,
        List<String> nodeUrns) throws ExperimentationException {


    // get experiment controller
    ExperimentController controller = findExperimentControllerBySecretReservationKey(secretReservationKeys);

    // if controller not found or if it has not queue
    try {
        ifNull(controller, "Unexpected. Controller not " +
            "properly set on the server");
        ifNull(controller.getMessageQueue(), "Unexpected. Message queue not " +
            "properly set on the controller.");
        ifNull(controller.getRequestStatusQueue(), "Unexpected. Message queue" +
            " not properly set on the controller." );
        ifNull(controller.getNotificationQueue(), "Unexpected. Notification" +
            " queue not properly set on the controller.");
    } catch (RuntimeException cause) {
        throw new ExperimentationException(cause.getMessage());
    }

    jobs.submit(new Job(
            "(reset nodes)(" + secretReservationKeys.get(0).getSecretReservationKey() + ")",
            controller.getWsn().resetNodes(nodeUrns),
            nodeUrns,Job.JobType.resetNodes
            ));
    jobs.join();
}
```

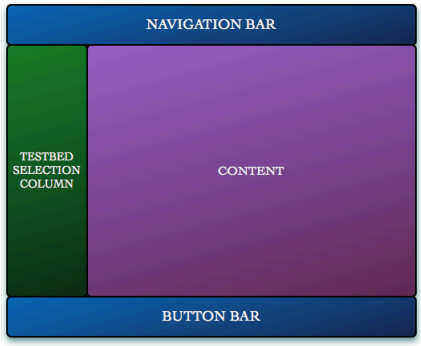
Όπως βλέπουμε με τη χρήση της μεθόδου findExperimentControllerBySecretReservationKey() γίνεται η αναζήτηση στη λίστα των ελεγκτών , για αυτόν που φέρει το μυστικό κλειδί κράτησης που διαθέτει ο χρήστης. Όπως αναφέρθηκε παραπάνω , υπό κανονικές συνθήκες ο ενεργός ελεγκτής, δηλαδή ένα τρέχων πείραμα, θα πρέπει να βρίσκεται στη λίστα.Σε περίπτωση που δεν βρεθεί αυτός ο Εξυπηρετητής επιστρέφει το λάθος στον Πελάτη με τη χρήση μιας εξαίρεσης. Εαν βρεθεί συνεχίζεται η διαδικασία με την κατασκευή και η καταχώρηση μιας ασύγχρονης αίτησης επανεκκίνης των κόμβων μέσω της μεθόδου jobs.submit() της κλάσης AsyncJobObserver. Στα ορίσματα της συνάρτησης jobs.submit() εκτός των κόμβων που προορίζονται για επανεκκίνηση, τίθεται και η κλήση controller.getWsn().resetNodes(nodeUrns) η οποία προήλθε από το στιγμιότυπο WSN που ανακτήθηκε αρχικά. Με την ολοκλήρωση της αίτησης, έχει γίνει επανεκκίνηση της λειτουργίας όλων των επιλεγμένων κόμβων. Ακολουθεί εικόνα της παραπάνω περιγραφής.



Εικόνα 5.10: Παράδειγμα επανεκκίνησης κόμβων στα πλαίσια της υπηρεσίας ExperimentationService

5.5 Πελάτης της Εφαρμογής

Το *WiseUI*, είναι μια πλούσια εφαρμογή ιστού με το κομμάτι του Πελάτη να προσφέρει πέρα από τη γραφική διεπαφή και στη λειτουργικότητα της. Ακολουθεί το περίγραμμα της γραφικής διεπαφής της εφαρμογής.

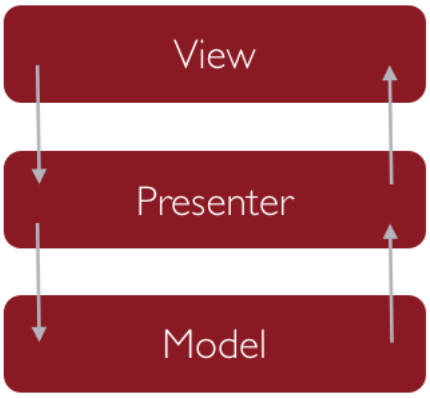


Εικόνα 5.11: Περίγραμμα της γραφικής διεπαφής του WiseUI

Το βόρειο τμήμα θα είναι η μπάρα περιήγησης (*Navigation Bar*) ενώ στα αριστερά βρίσκεται η στήλη με τις διαθέσιμες προς επιλογή ΠΥ (*Testbed Selection Column*). Ο συνδυασμός αυτών των δύο προκαλούν αλλαγές στο κύριο πλαίσιο της γραφικής διεπαφής το οποίο ονομάζεται περιεχόμενο (*Content*) καθώς και σε ένα μέρος της μπάρας που περιέχει τα κουμπιά για τη λειτουργικότητα της εφαρμογής

(*Button Bar*). Χρήση αυτών των κουμπιών μπορεί και αυτή με τη σειρά της να προκαλέσει αλλαγές στο περιεχόμενο της εφαρμογής.

Στις 8 Δεκεμβρίου του 2009 κυκλοφορεί η έκδοσης 2.0 του *GWT*. Τον ίδιο χρόνο στα πλαίσια του παγκόσμιο συνεδρίου προγραμματιστών τεχνολογιών της *Google*, *Google I/O 2009*, ο *Ray Ryan* μηχανικός της *Google* στο έργο *GWT* έδωσε μια ομιλία με τίτλο *Best Practices for Architecting GWT App* η οποία παρουσίαζε τις καλύτερες πρακτικές για ανάπτυξη εφαρμογών μεγάλης κλίμακας με χρήση του *GWT*. Ένα από τα θέματα που έθιξε είναι η ενθάρυνση της χρήσης του σχεδιαστικού πρότυπου με όνομα Μοντέλο-Όψη-Παρουσίαση (*Model-View-Presenter - MVP*)[44]. Το ΜΟΠ αποτελεί ένα από τα πιο διαδεδομένα σχεδιαστικά πρότυπα για ανάπτυξη διεπαφών χρηστών σε εφαρμογές και προέρχεται από το ευρήτερα γνωστό Μοντέλο-Όψη-Ελεγκτής (*Model-View-Controller - MVC)*. Στόχος του ΜΟΠ είναι ο διαχωρισμός των υλοποιήσεων της λογικής της εφαρμογής και της παρουσίασης του στη τη γραφική διεπαφή. Το Μοντέλο αναφέρεται στο σύνολο των δεδομένων τα οποία επρόκειτο να τοποθετηθούν στη γραφική διεπαφή. Η Όψη αναφέρεται στο σύνολο των στατικών γραφικών συστατικών όπως τίτλοι,παράθυρα,πάνελ αλλά και διαδραστικών συστατικών όπως κουμπιά, σκρόλ, πεδία κειμένων. Η Παρουσίαση αποτελεί το μεσαίο επίπεδο στο πρότυπο και συνεργάζεται με την Όψη και το Μοντέλο. Σκοπός της Παρουσίασης είναι η τροποποίηση και η πιθανή επεξεργασία των δεδομένων του μοντέλου ώστε να παρουσιαστούν από την Όψη στον χρήστη της εφαρμογής. Επιπλέον, τα διαδραστικά συστατικά της όψης ενεργοποιούν γεγονότα (*UI Events*) τα οποία διαχειρίζεται καταλλήλως η Παρουσίαση.



Εικόνα 5.12: Σχεδιαστικό πρότυπο ΜΟΠ

Ας εξετάσουμε ένα παράδειγμα χρήσης του προτύπου ΜΟΠ. Θα δούμε μια μέθοδο η οποία υλοποιείται στα πλαίσια της κλάσης `ExperimentPresenter` η οποία αναλαμβάνει το ρόλο της Παρουσίασης των δεδομένων και τη κλάση `ExperimentViewImpl` μαζί με το αρχείο `ExperimentViewImpl.ui.xml` τα οποία αποτελούν στην περίπτωση μας Όψης. Προς χάριν συντομίας θα σταθούμε σε μια μέθοδο με όνομα `startExperiment()` η οποία εκτελείται όταν ο χρήστης επιλέξει την έναρξη ενός πειράματος. Για τον κώδικα της όψης ο αναγνώστης μπορεί να παραπεμφθεί στον πηγαίο κώδικα καθώς, είναι αρκετά μεγάλος για να ενσωματωθεί στην αναφορά

Κώδικας 5.12: Παράδειγμα Παρουσίασης - ExperimentPresenter - Μέθοδος startExperiment()

```
...
@Override
public void startExperiment() {

    final ExperimentPresenter currentExperiment = this;
    AsyncCallback<Void> callback = new AsyncCallback<Void>() {

        @Override
        public void onFailure(Throwable caught) {
            MessageBox.error("Experimentation Service", caught.getMessage(), caught, null);
        }

        @Override
        public void onSuccess(Void result) {

            injector.getExperimentationManager().addExperimentToActiveList(currentExperiment);
            status = ExperimentStatus.RUNNING;

            view.setStatus(status.getStatusText());
            view.deactivateStartExperimentButton();
            view.activateFlashExperimentButton();
            view.activateResetNodesButton();
            view.activateStopExperimentButton();
            view.activateDownloadWiseMLButton();

            startExperimentMessageCollectorTimer();
        }

    };
    List<SecretReservationKey> secretReservationKeys = new ArrayList<SecretReservationKey>();
    for(int i=0;i<userData.size();i++) {
        SecretReservationKey key = new SecretReservationKey();
        key.setSecretReservationKey(userData.get(i).getSecretReservationKey());
        key.setUrnPrefix(userData.get(i).getUrnPrefix());
        secretReservationKeys.add(key);
    }
    service.startExperimentController(
            sessionManagementUrl,
            secretReservationKeys,
            nodeUrns,callback);
}
...
```

Με την έναρξη μιας κράτησης, ο χρήστης έχει την δυνατότητα να ξεκινήσει το πείραμα πατώντας το κουμπί *Start Experiment* στον πάνελ του πειράματος. Με το πάτημα του κουμπίου καλείται η μέθοδος `startExperiment()`, η οποία αρχικά επικοινωνεί με τον εξυπηρετητή για την έναρξη του πειράματος. Σε περίπτωση σφάλματος , εμφανίζεται στην οθόνη ένα μήνυμα λάθος που εξηγεί τη πήγε στραβά στη διαδικασία. Σε περίπτωση επιτυχημένης έναρξης του πειράματος διαμορφώνεται το πάνελ του πειράματος, αλλάζωντας τον τίτλο της κατάστασης, και ενεργοποιεί τις υπόλοιπες επιλογές. Παραάλληλα, ενεργοποιείται ο χρονιστής λήψης μηνυμάτων τον οποίο θα συζητήσουμε στη συνέχεια. Το Μοντέλο στο παραπάνω παράδειγμα είναι η αλλαγές που προκύπτουν στην όψη , όπως η αλλαγή κατάστασης στο πείραμα (από κατάσταση *READY* σε κατάσταση *RUNNING*).

Σε αυτό το σημείο θα αναφερθούμε σε δύο σημαντικές κλάσης που προσφέρει η βιβλιοθήκη του *GWT* η οποίες ονομάζονται `Place` και `Activity`.Η έννοια του `Place` παίζει σημαντικό ρόλο για τον Πελάτη της εφαρμογής. Στο *GWT 2.1* ως `Place` ονομάζουμε ένα *Java* αντικείμενο το οποίο παρουσιάζει την κατάσταση της εφαρμογής (π.χ ποιά σελίδα βλέπει ο χρήστης). Κάθε `Place` μπορεί να διαμορφωθεί και να κωδικοποιηθεί ώστε να καταγραφέι ως *URL* στη μπάρα και την ιστορία του περιηγητή. Έτσι αν κάποιος βρίσκεται στη σελίδα που παρουσιάζει τις λεπτομέριες μιας ΠΥ μπορεί να μεταβεί στη σελίδα των κρατήσεων με ένα σύνδεσμο, παράλληλα αλλάζει και η διεύθυνση *URL* στον περιηγήτη. Έτσι η εφαρμογή μας αποκτά ένα ιστορικό της κατάστασης που έχει μεταβεί ο χρήστης. Η αλλαγή ενός `Place` γίνεται μέσω της κλήσης της μεθόδου `PlaceController.goTo()` την οποία παρέχει η βιβλιοθήκη του *GWT*.

Οι αλλαγές μεταξύ των `Place` σε μια εφαρμογή δεν είναι αρκετές από μόνες τους για την αλλαγή του περιεχομένου μια εφαρμογής. Σε κάθε αλλαγή από το ένα `Place` στο άλλο, ενεργοποιείται και μια δραστηριότητα `Activity`. Τα `Activities` είναι οι οποίες αντιστοιχίζονται με αυτές των `Place`, μέσω των κλάσσεων που υλοποιούν την `ActivityMapper`, και υλοποιούν την μεθόδο `start()`. Η υλοποίησης της μεθόδου αυτή αρχικοποιεί τα δεδομένα προς παρουσίαση στην όψη (*View*) της εφαρμογής.

Στη συνέχεια θα δούμε ένα παράδειγμα από τον κώδικα του *WiseUI*.

Κώδικας 5.13: Κύρια κλάση που υλοποιεί το EntryPoint και την onModuleLoad()

```
public class WiseUi implements EntryPoint {
    ...
    private final WiseUiGinjector injector = GWT.create(WiseUiGinjector.class);

    /**
     * This is the entry point method.
     */
    public void onModuleLoad() {
        final WiseUiView appWidget = injector.getAppWidget();

        injector.getNavigationActivityManager().setDisplay(appWidget.getNavigationPanel());
        injector.getTestbedListActivityManager().setDisplay(appWidget.getTestbedListPanel());

        // Start ActivityManager for the main widget with our ActivityMapper
        injector.getContentActivityManager().setDisplay(appWidget.getContentPanel());

        RootPanel.get().add(appWidget.asWidget());

        // Goes to place represented on URL or default place
        injector.getPlaceHistoryHandler().handleCurrentHistory();

        // Init authentication & reservation manager.
        injector.getAuthenticationManager().init();
        injector.getReservationManager().init();
        injector.getExperimentationManager().init();

        hideLoadingIndicator();
    }
    ...
}
```

Εδώ παρουσίαζεται η κύρια κλάση, η οποία κληρονομεί την κλάση `EntryPoint` της εφαρμογής στον

Πελάτη, η οποία αποτελεί τον πυρήνα της εφαρμογής και αρχικοποιεί της *manager* κλάσης για την αρχικοποίηση των *Activities* και την τοποθέτηση όλων των όψεων στο παράθυρο της εφαρμογής.

Κώδικας 5.14: Έναρξη μιας δραστηριότητας (Activity) ανάλογα με την αλλαγή του Place στο Περιεχόμενο της εφαρμογής

```
public class ContentActivityMapper implements ActivityMapper {

    private WiseUiGinjector injector;

    @Inject
    public ContentActivityMapper(final WiseUiGinjector injector) {
        super();
        this.injector = injector;
    }

    public Activity getActivity(final Place place) {
        final WiseUiPlace wiseUiPlace = (WiseUiPlace) place;
        final NavigationPlace navigationPlace = (NavigationPlace) wiseUiPlace.get(NavigationPlace.class);
        final Integer index = navigationPlace.getIndex();
        Activity mappedActivity = null;
        if (index.equals(0)) {
            final TestbedSelectionActivity activity = injector.getTestbedSelectionActivity();
            activity.setPlace(wiseUiPlace);
            mappedActivity = activity;
        } else if (index.equals(1)) {
            final ReservationActivity activity = injector.getReservationActivity();
            activity.setPlace(wiseUiPlace);
            mappedActivity = activity;
        } else if (index.equals(2)) {
            mappedActivity = injector.getExperimentationActivity();
        }
        return mappedActivity;
    }
}
```

Αυτό το κομμάτι κώδικα δείχνει οτί ανάλογα με τον δείκτη που φέρει το `Place` επιστρέφεται το κατάλληλο `Actvity`. Έτσι αν ο χρήστης επιθυμεί για να μεταβεί ο χρήστης από την οθόνη των κρατήσεων σε αυτή του πειραματισμού χρειάζεται να γίνει η κλήση της παραπάνω μεθόδου ώστε να ενεργοποιηθεί η κατάλληλ δραστηριότητα.

Στη συνέχεια θα σταθούμε στη λειτουργικότητα που προσδίδει το κομμάτι του Πελάτη στην εφαρμογή *WiseUI*. Ένα σημαντικό κομμάτι της λειτουργικότητας είναι η λήψη των δεδομένων του πειράματος από τον Εξυπηρετητή στον Πελάτη. Θα χρησιμοποιηθεί η τεχνική του *Polling*[52], για την λήψη των δεδομένων. Ο Πελάτης ανά χρονικά διαστήματα των 0,5 δευτερολέπτων ζητά από τον Εξυπηρετητή της εφαρμογής να παραδώσει το τελευταίο διαθέσιμο μήνυμα. Στον παραπάνω κώδικα της μεθόδου `startExperiment()`, με την έναρξη του πειράματος , ξεκινά και η διαδικασία λήψης μηνυμάτων με την κλήση της μεθόδου `startMessageCollectorTimer()`.

Κώδικας 5.15: Παράδειγμα Παρουσίασης - ExperimentPresenter - Μέθοδος startMessageCollectorTimer()

```
...
private void startExperimentMessageCollectorTimer() {

    // source is this presenter
    final ExperimentPresenter source = this;

    //experiment stop timer counts till the reservation ends
    if(experimentMessageCollectionTimer == null) {
        experimentMessageCollectionTimer = new Timer() {

            @Override
            public void run() {
                // setup callback
                AsyncCallback<Message> callback = new AsyncCallback<Message>() {

                    @Override
                    public void onFailure(Throwable caught) {
                        MessageBox.error("Experimentation Service", caught.getMessage(), caught, null);
                    }

                    @Override
                    public void onSuccess(final Message result) {
                        if(result == null) return;

                        outputMap.get(result.getSourceNodeId()).getView().addOutput(result.prettyToString());
                    }

                };

                // make the rpc call
                List<SecretReservationKey> secretReservationKeys = new ArrayList<SecretReservationKey>();
                for (Data anUserData : userData) {
                    SecretReservationKey key = new SecretReservationKey();
                    key.setSecretReservationKey(anUserData.getSecretReservationKey());
                    key.setUrnPrefix(anUserData.getUrnPrefix());
                    secretReservationKeys.add(key);
                }
                service.returnExperimentMessage(secretReservationKeys, callback);
            }
        };
    }

    // start experiment collection timer
    experimentMessageCollectionTimer.scheduleRepeating(500);
}
...
```

Η μέθοδος `startMessageCollectorTimer()` χρησιμοποιεί την κλάση `Timer` της βιβλιοθήκης του *GWT* και έχει προγραμματιστεί να εκτελεί μια αίτηση για ένα μήνυμα δεδομένων προς τον Εξυπηρετητή ανά 500 *milliseconds*. Σε περίπτωση σφάλματος παρουσιάζεται στον χρήστη το αντίστοιχο μήνυμα λάθους, αντίθετα υπο κανονική λειτουργία είτε εκτυπώνεται το μήνυμα που παραδώθηκε στο πάνελ του κόμβου ο οποίος το έφτιαξε, είτε δεν εκτυπώνεται τίποτα εφόσον δεν παραδώθηκε κάποιο μήνυμα.

Στην παραπάνω ενότητα της αποθήκευσης των δεδομένων αναφερθήκαμε ότι και ο Πελάτης αποθη-

κεύει, έστω και προσωρινά, κάποια χρήσιμα δεδομένα. Ένα τέτοιο παράδειγμα είναι αυτό του μυστικού κλειδιού ταυτοποίησης. Το κλειδί αυτό είναι μια τιμή που παράγεται από τις υπηρεσίες της ΠΥ και δείχνει ότι ο χρήστης που το κατέχει είναι ταυτοποιημένος και εξουσιοδοτημένος για χρήση των πόρων της ΠΥ. Καθώς αυτό το κλειδί μπορεί να χρησιμοποιηθεί για να ανακτηθούν οι κρατήσεις που έχει κάνει ο χρήστης θα αποθηκεύσουμε το κλειδί αυτό ως *Cookie* στον περιηγητή του χρήστη και έχει χρονική διάρκεια περίπου μιας ώρας. Κάθε φορά που ο χρήστης χρειάζεται να δείξει ότι είναι ταυτοποιημένος μπορεί να ανακτήσει το κλειδί αυτό μέσω των μηχανισμό διαχείριση *Cookies* του *GWT*. Αν και η συγκεκριμένη προσέγγιση είναι βολική, η απενεργοποίηση των *Cookies* από τον χρήστη στον περιηγητή του, καθιστά την εφαρμογή άχρηστη καθώς η ανάκτηση του κλειδιού αυτού γίνεται προς το παρόν μόνο με αυτό τον τρόπο. Όποτε είναι απαραίτητο ο χρήστης να έχει ενεργοποιημένη την επιλογή των *Cookies* στον περιηγητή του.

Σε αυτό το σημείο κλείνουμε το κεφάλαιο της περιγραφής του σχεδιασμού και της αρχιτεκτονικής της εφαρμογής. Ο σχεδιασμός αυτός επικεντρώνεται στην υλοποίηση της βασικής λειτουργικότητας για ένα εργαλείο πειραματισμού με ΠΥ στα πλαίσια του *WISEBED*. Πολλές επιλογές που έχουνε γίνει είναι αρκετά δεσμευτικές για την επέκταση της εφαρμογής, όπως το περιορισμένο μοντέλο δεδομένων στη βάση δεδομένων, παρόλα αυτά με αυτή την έκδοση θέτονται ισχυρές βάσεις για την ανάπτυξη ενός εύχρηστου και διαδραστικού εργαλείου πειραματισμού το οποίο θα αντικαταστήσει τα ήδη υπάρχοντα.

Κεφάλαιο 6

Εκτέλεση διαδικτυακής εφαρμογής

Έχοντας παρουσιάσει όλες τις τεχνικές λεπτομέρειες από τις οποίες απαρτίζεται το *WiseUI*, στο κεφάλαιο αυτό θα παρουσιάσουμε την τελική εφαρμογή μετά το πέρας της υλοποίησής της.Θα παρουσιαστεί ένα βήμα προς βήμα ένα κλασικό σενάριο εκτέλεσης ώστε να δείξουμε τις προδιαγραφές που αναφέρθηκαν στην αρχή του προηγούμενου κεφαλαίου. Το σενάριο περιλαμβάνει την ταυτοποίηση και την εξουσιοδότηση ενός χρήστη, τη δημιουργία/διαχείριση κρατήσεων σε κόμβους που ανήκουν σε μια επιλεγμένη ΠΥ και τέλος την εκτέλεση ενός πειράματος στους επιλεγμένους κόμβους και έκδοση αποτελεσμάτων.

6.1 Σενάριο εκτέλεσης

Οι εικόνες από την εκτέλεση της εφαρμογής βρίσκονται στο τέλος του κεφαλαίου.

Η εφαρμογή αποτελείται από τρία κύρια μέρη, τα οποία εμφανίζονται ως τρεις καρτέλες στο παράθυρο της εφαρμογής. Το πρώτο με τον τίτλο *"Testbeds"* κρατάει πληροφορίες για τις υπάρχουσες ΠΥ που γνωρίζει η εφαρμογή. Αριστερά, βρίσκεται η λίστα με τις υπάρχουσες ΠΥ, μέσω της οποίας ο χρήστης μπορεί να επιλέξει μία από αυτές και να δει λεπτομέρειες όσον αφορά τους κόμβους και την εγκατάσταση τους στο χώρο που φιλοξενεί την ΠΥ. Τα κουμπιά που βρίσκονται ακριβώς κάτω από τη λίστα, προσφέρουν λειτουργίες όπως η προσθήκη μιας νέας πειραματικής υποδομής, την πλήρη διαγραφή ή την αλλαγή των παραμέτρων μιας ήδη υπάρχουσας. Επιπλέον ένα από τα κουμπιά (εικονίδιο με το κλειδί) ανοίγει ένα παράθυρο διαλόγου με τον χρήστη ώστε να ταυτοποιηθει και πιθανώς να εξουσιοδοτηθεί και απόκτηση πρόσβασης για εκμετάλλευση σε επόμενο στάδιο των πειραματικών πόρων καθώς και την ανανέωση της λίστας αυτής. Το μεγαλύτερο πλαίσιο της οθόνης, εναλλάσει την λειτουργία του ανάλογα με το ποιός από τους τρεις διακόπτες είναι επελεγμένος στο ακριβώς κάτω μέρος. Σε αυτή την εικόνα, φαίνεται προεπιλεγμένος ο πρώτος διακόπτης, με τον οποίο παρουσιάζεται στο χάρτη με διαισθητικό τρόπο ένα πλαίσιο που διαμορφώνεται από τις συντεταγμένες τοποθέτησης του κάθε αισθητήρα ξεχωριστά.

Επιλέγοντας το δεύτερο διακόπτη, στο κυρίως πλαίσιο εμφανίζονται λεπτομερώς όλοι οι εγκατεστημένοι αισθητήρες σε μία επεκτάσιμη λίστα. Εκεί φαίνεται συνολικά ο αριθμός τους καθώς και ο τύπος του αισθητήρα. Όπως φαίνεται στην παρακάτω εικόνα, επιλέγοντας κάποιον από τους κόμβους, βλέπουμε στο δεξί μέρος της εφαρμογής λεπτομέρειες σχετικά με τις ιδιότητες του συγκεκριμένου αισθητήρα. Εκεί απεικονίζονται πληροφορίες όπως το αναγνωριστικό του κόμβου, ο τύπος, οι ακριβείς τιμές των συντεταγμένων τοποθέτησης, μια σύντομη περιγραφή αλλά και αναλυτικά οι διαθέσιμες λειτουργίες που παρέχει με τη μορφή δεδομένων που επιστρέφει τις τιμές.

Με την επιλογή του τρίτου διακόπτη εναλλαγής απεικόνισης, παρουσιάζεται σε μορφή *WiseML* όλη η απαραίτητη πληροφορία για μια πειραματική υποδομή. Εικόνες 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7.

Για τη χρήση των πειραματικών πόρων μιας υποδομής, θα πρέπει να γίνει ταυτοποίηση των στοιχείων του χρήστη, πριν ο χρήστης προχωρήσει στη δημιουργία μιας νέας κράτησης. Ο χρήστης επιλέγει την ΠΥ στην οποία επιθυμεί να δεσμεύσεις κόμβους. Εικόνα 6.8. Σε περίπτωση που ο χρήστης επιχειρήσει να κάνει μία νέα κράτηση χωρίς να έχει προηγηθεί η διαδικασία της ταυτοποίησης, το σύστημα θα του απαγορέψει την ενεργεια και θα τον ενημερώσει κατάλληλα. Στην εικόνα 6.9 φαίνεται ο διάλογος ταυτοποίησης, στον οποίο ο χρήστης καλείται να δώσει το όνομα και το συνθηματικό που του έχουν ανατεθεί από τον διαχειριστή της εκάστοτε υποδομής.

Με την επιτυχή ταυτοποίηση των στοιχείων του χρήστη, γίνεται και εξουσιοδότηση για την εκμετάλλευση της πειραματικής υποδομής. Έτσι, για να προχωρήσουμε στη δημιουργία μιας νέας κράτησης μεταβαίνουμε στη δεύτερη καρτέλα της εφαρμογής. Στο κυρίως πλαίσιο, και ενώ η λίστα με τις υποδομές έχει παραμείνει σταθερή, φαίνεται μια λίστα με τους διαθέσιμους αισθητήρες σε επεκτάσιμη μορφή καθώς και ένα ημερολόγιο όπου απεικονίζονται όλες οι προυπάρχουσες κρατήσεις για ένα συγκεκριμένο χρονικό διάστημα. Το πρώτο πράγμα που πρέπει να γίνει για τη δημιουργία μιας κράτησης, είναι να ορίσουμε ποιούς αισθητήρες θέλουμε να χρησιμοποιήσουμε για την εκτέλεση των πειραμάτων μας. Έτσι για παράδειγμα, όπως φαίνεται και στην εικόνα 6.10, επιλέγουμε ένα υποσύνολο από αισθητήρες για να προχωρήσουμε στο επόμενο βήμα για τη δημιουργία της κράτησης.

Εφόσον έχουμε επιλέξει τους αισθητήρες που θα χρησιμοποιήσουμε, μεταβαίνουμε στο πλαίσιο του ημερολογίου. Εκεί, έχοντας μετακινηθεί στο επιθυμητό διάστημα χρόνου κάνουμε διπλό κλικ και εμφανίζεται ο διάλογος δημιουργίας νέας κράτησης, όπως φαίνεται στην εικόνα 6.11. Στο διάλογο αυτό καθορίζουμε το διάστημα χρόνου κατά το οποίο το σύστημα θα δεσμεύσει τους αισθητήρες για λογαριασμό μας. Έχοντας λοιπόν καθορίσει όλες τις παραμέτρους, μπορούμε πλέον να προχωρήσουμε με τη δημιουργία της κράτησης. Ως επιπλέον λειτουργικότητα, μέσω του ίδιου διαλόγου ο χρήστης μπορεί να αλλάξει τις παραμέτρους μιας ήδη υπάρχουσσας κράτησης.

Μετά την επιτυχή δημιουργία της νέας κράτησης, το σύστημα επιστρέφει στον χρήστη ένα αποκλειστικό μυστικό κλειδί, το οποίο και θα χρησιμοποιήσει σε επόμενο βήμα για την εκτέλεση των πειραμάτων. Το κλειδί αυτό, προσαρτάται στο εκάστοτε αντικείμενο της κάθε κράτησης όπως αυτά αναπαρίστανται στο ημερολόγιο και έτσι ο χρήστης δεν χρειάζεται να απομνημονεύει το κάθε κλειδί που του επιστρέφεται για τις κρατήσεις του. Παράλληλα, ο χρήστης βλέπει τη νέα του κράτηση να προσαρτάται στο ημερολόγιο. Για να πετύχουμε μια πιο διαισθητική απεικόνιση των κρατήσεων που ανήκουν στον χρήστη, επιλέξαμε οι κρατήσεις που δεν του ανήκουν να έχουν χρώμα μπλε ενώ αυτές που έχει

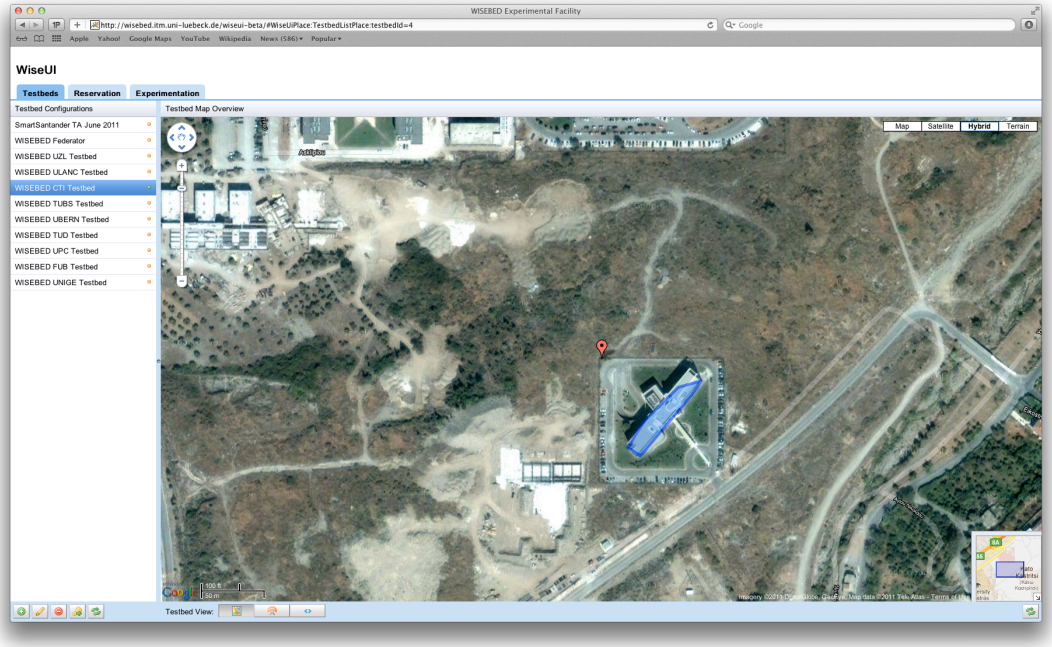
δημιουργήσει ο ίδιος να έχουνε χρώμα κόκκινο. Τα παραπάνω διακρίνονται ξεκάθαρα στην εικόνα 6.12.

Μέσω του ημερολογίου, κάνοντας διπλό κλικ σε κάποια κράτηση, ο Πελάτης μπορεί να δει όλες τις σχετικές λεπτομέρειες, όπως σε ποιόν χρήστη ανήκει, το διάστημα χρόνου κατά το οποίο τα πειράματα πρόκειται να εκτελεστούν καθώς και όλους τους αισθητήρες που είναι δεσμευμένοι. Σε περίπτωση που η κράτηση αυτή ανήκει στον χρήστη που είναι συνδεδεμένος στο σύστημα εκείνη τη στιγμή, τότε ο ίδιος μπορεί αλλάζοντας τις παραμέτρους του χρόνου να μεταθέσει την κράτηση προς εκτέλεση σε κάποια άλλη χρονική στιγμή, να τη διαγράψει πλήρως ή τέλος να κάνει απλή επισκόπηση των στοιχείων της.

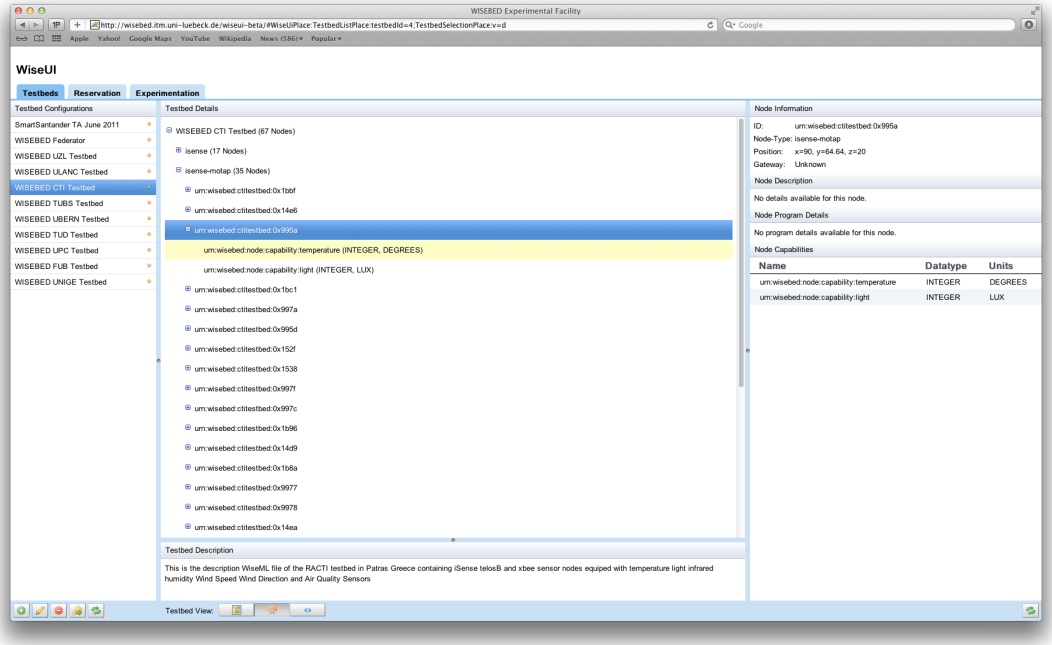
Μεταβαίνοντας στην τρίτη καρτέλα της εφαρμογής, μπορούμε πλέον να δούμε ότι το πείραμά μας είναι έτοιμο προς εκτέλεση. Εικόνα 6.13. Εκεί απεικονίζεται το σύνολο των κρατήσεων και του χρήστη καθώς και οι κρατήσεις του. Η έναρξη της κράτησης δεν σηματοδοτεί και την ένερξη του πειράματος. Ο χρήστης πρέπει να δώσει ρητά την εντολή έναρξης του πειράματος με τη χρήση του κουμπιού έναρξης(*Start Experiment*). Με την εκκίνηση του πειράματος περισσότερες λειτουργίες γίνονται διαθέσιμες . Εικόνες 6.14 6.15

Πλέον ο χρήστης μπορεί να πειραματιστεί απευθείας με τους κόμβους που συμμετέχουνε στο πείραμα. Μπορεί να φορτώσει ένα πρόγραμμα λειτουργίας (*Flash Experiment Image*) στους κόμβους που συμμετέχουνε είτε να τους επανακκινήσει ώστε να εκτελέσουνε το πρόγραμμα που είναι ήδη φορτωμένο σε αυτούς (*Reset Experiment Nodes*). Για να φορτώσει ένα προγράμμα λειτουργίας, ο χρήστης καλείται να είτε επιλέξει από τη βάση δεδομένων είτε να ανεβάσειένα πρόγραμμα λειτουργίας . Εικόνες 6.17 και 6.16. Με την ολοκλήρωση αυτών των ενεργιών ο χρήστης πλέον μπορεί να λάβει δεδομένα από μέσω του πειράματος. Επιλέγωντας έναν από τους κόμβους που συμμετέχει στο πείραμα μπορεί να δεί μια ζωντανή ροή δεδομένων σε ένα ειδικό παράθυρο. Η εικόνα 6.18 , δείχνει την έξοδο του πειράματος .

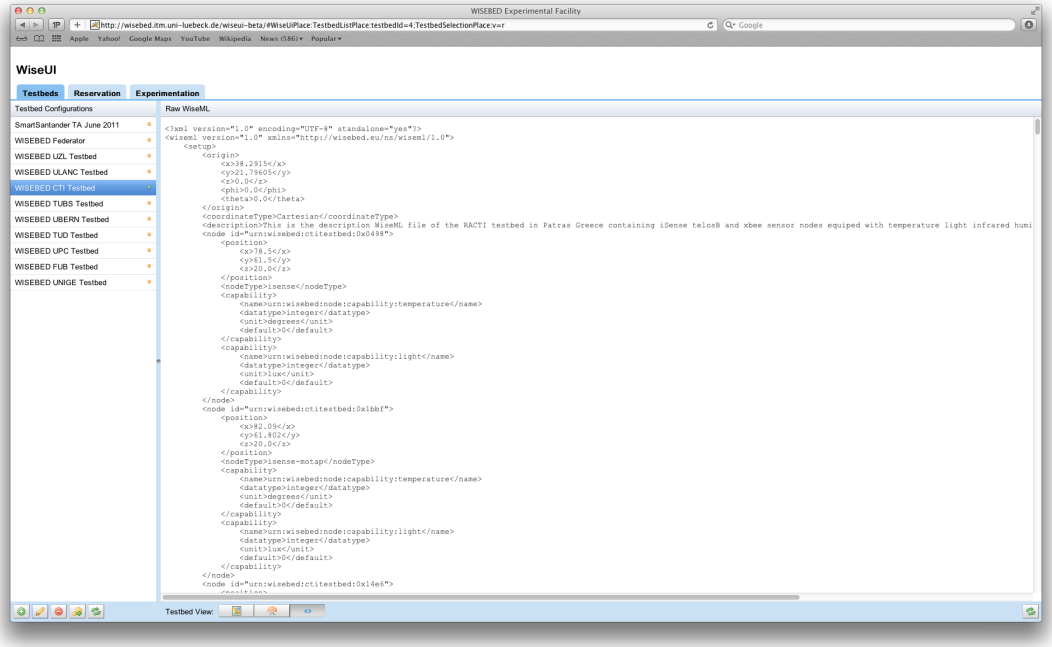
Ο χρήστης μπορεί να διακόψει την ροή δεδομένων πατώντας το κουμπί που σταματάει το πείραμα (*Stop Experiment*). Εφόσον δεν έχει εξαντληθεί ο χρόνος κράτησης, ο χρήστης μπορεί να επαννακινήσει το πείραμα και να αλλάξει πρόγραμμα λειτουργίας ή να διαγράψει την κράτηση στην καρτέλα της διαχείρησης κρατήσεων. Τέλος άλλη μια δυνατότητα είναι η εκτύπωση του πειράματος σε κείμενο γραμμένο σε *WiseML format* (*Download WiseML*) , το οποίο εμφανίζεται σε ένα άλλο ειδικό παράθυρο της εφαρμογής.



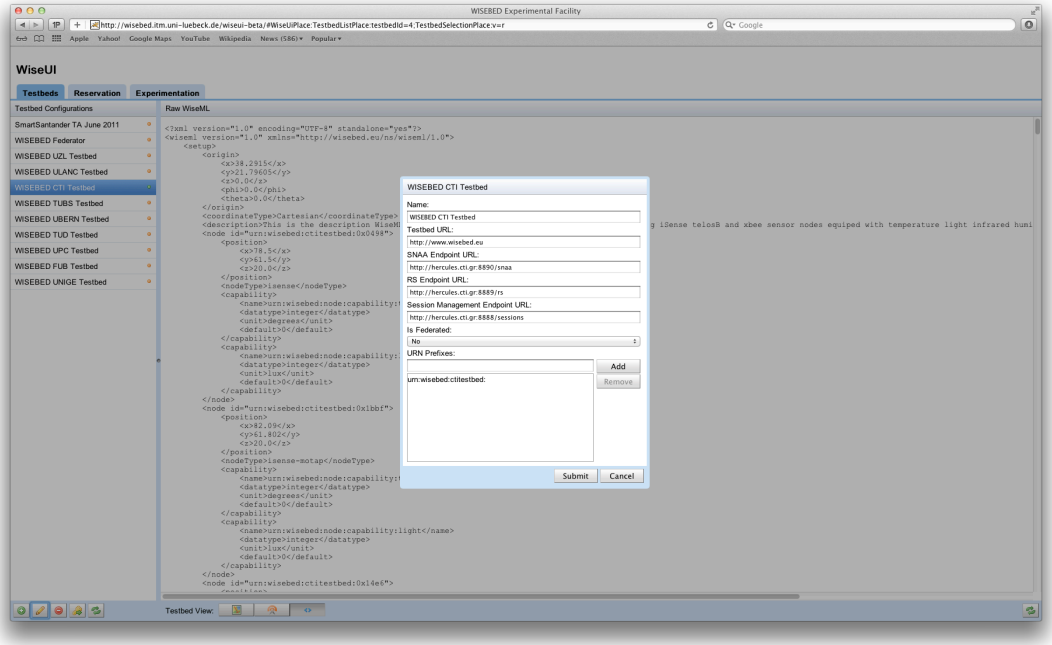
Εικόνα 6.1: Το χαρτογραφημένο ΑΔΑ στην της επιλεγμένης ΠΥ



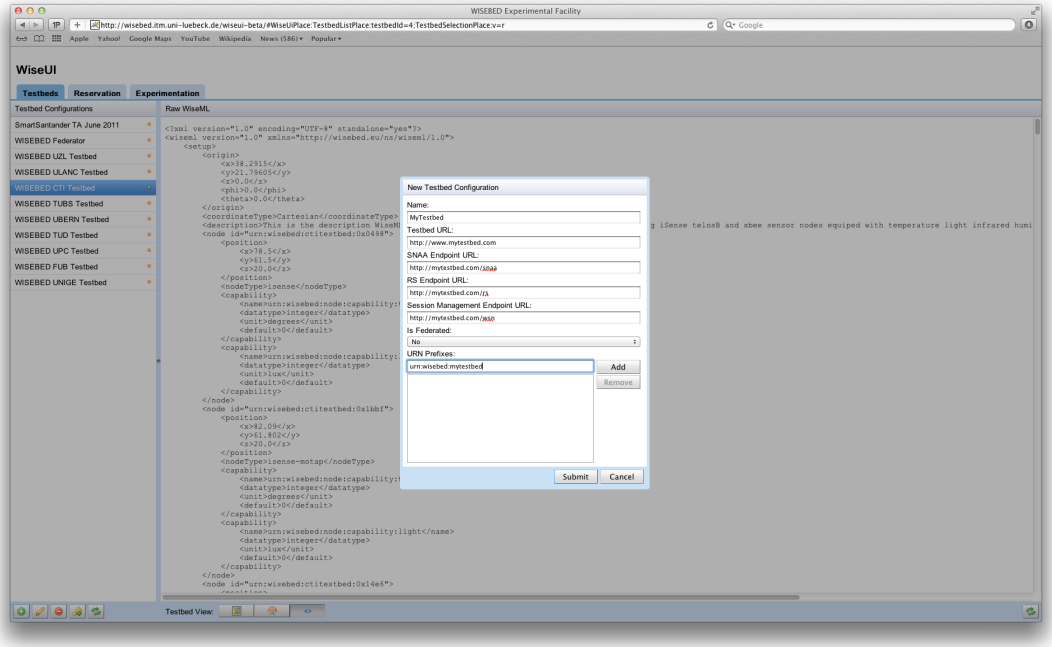
Εικόνα 6.2: Λεπτομέριες της επιλεγμένης ΠΥ



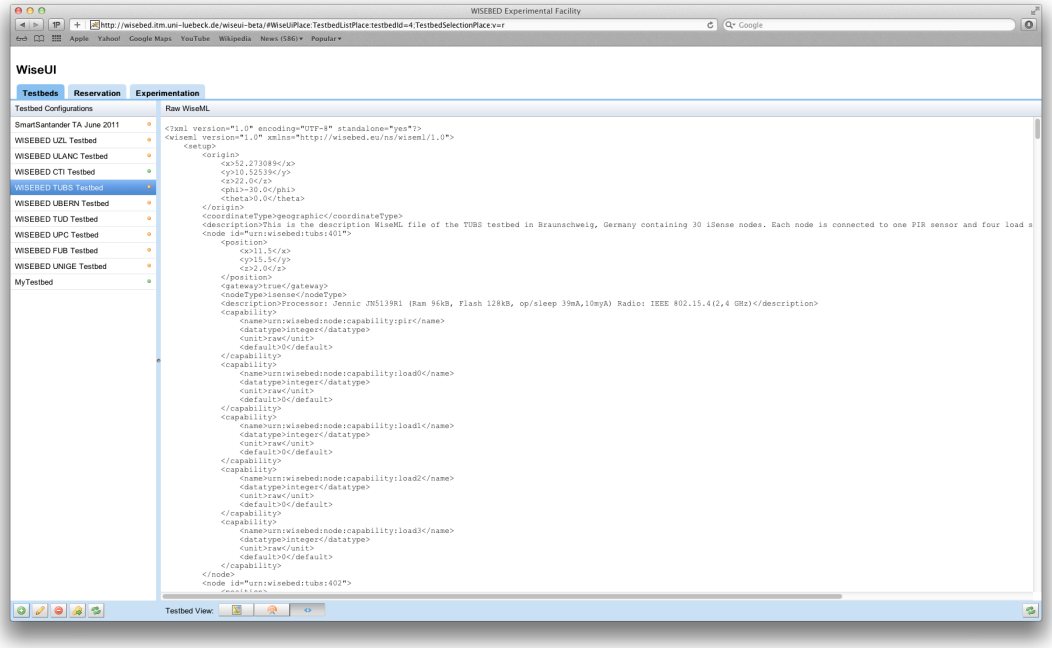
Εικόνα 6.3: Λεπτομέρειες πειραματικής υποδομής σε WiseML



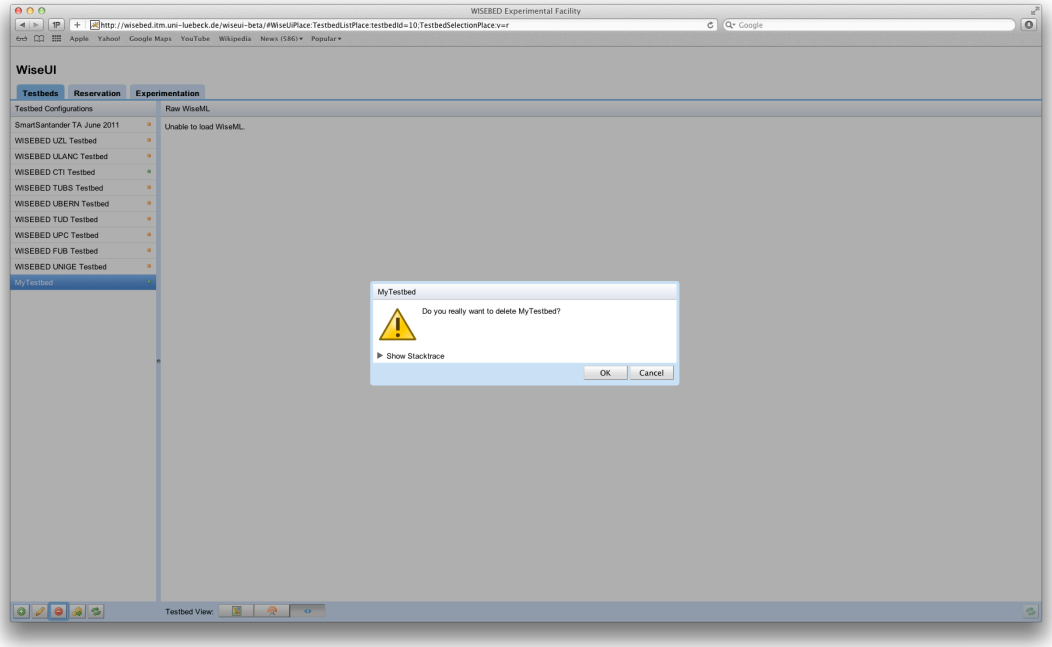
Εικόνα 6.4: Επεξεργασία εγγραφής ΠΥ



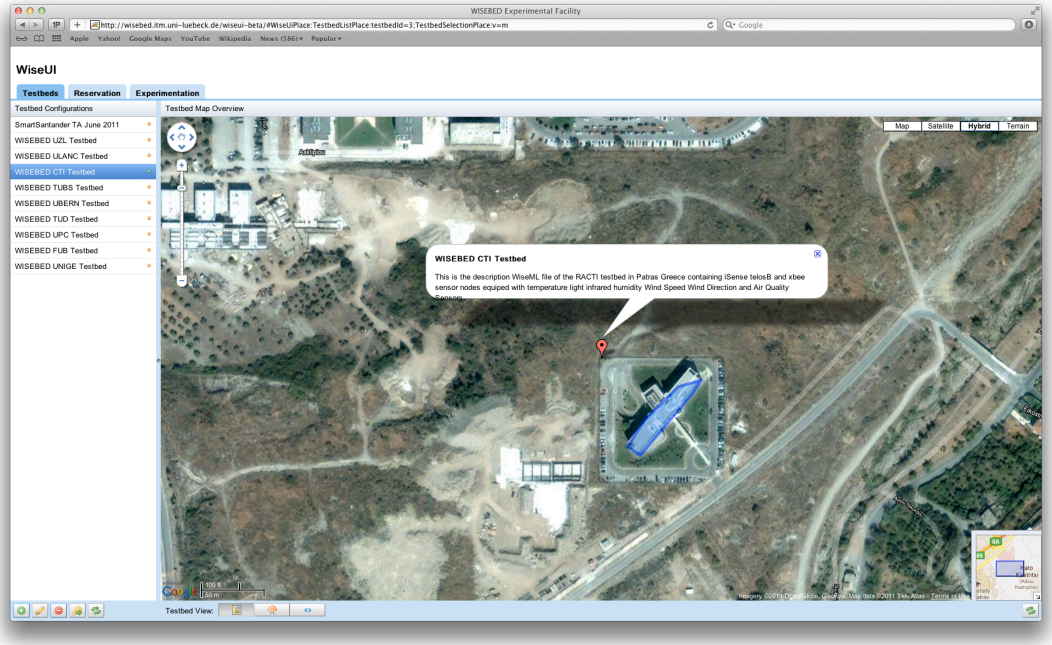
Εικόνα 6.5: Διαδικασίας πρόσθεσης νέας ΠΥ



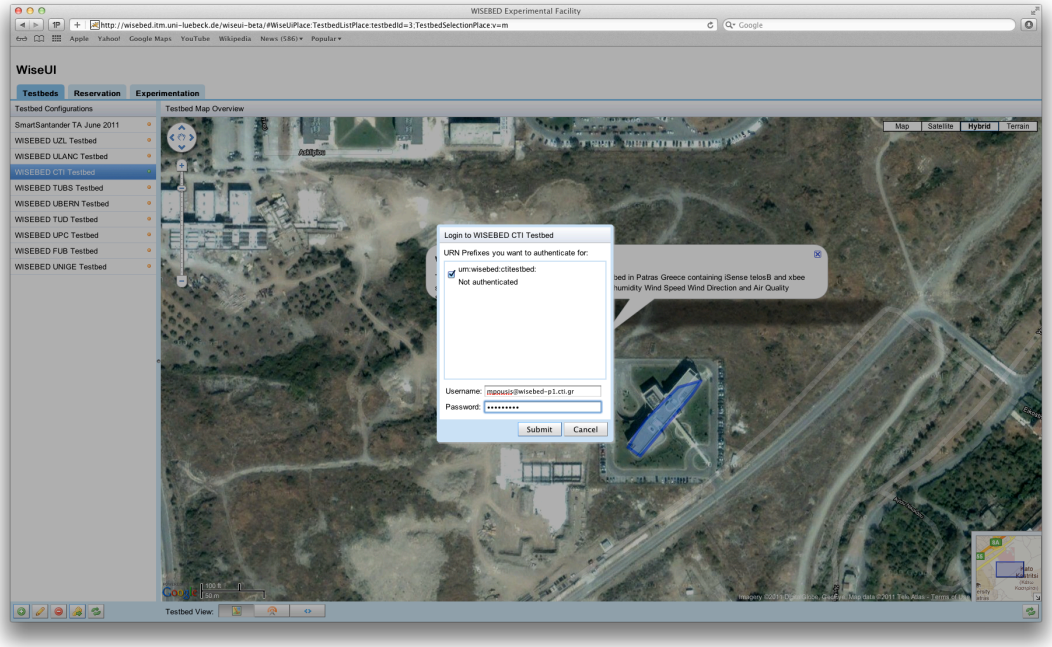
Εικόνα 6.6: Η νέα προσθήκη



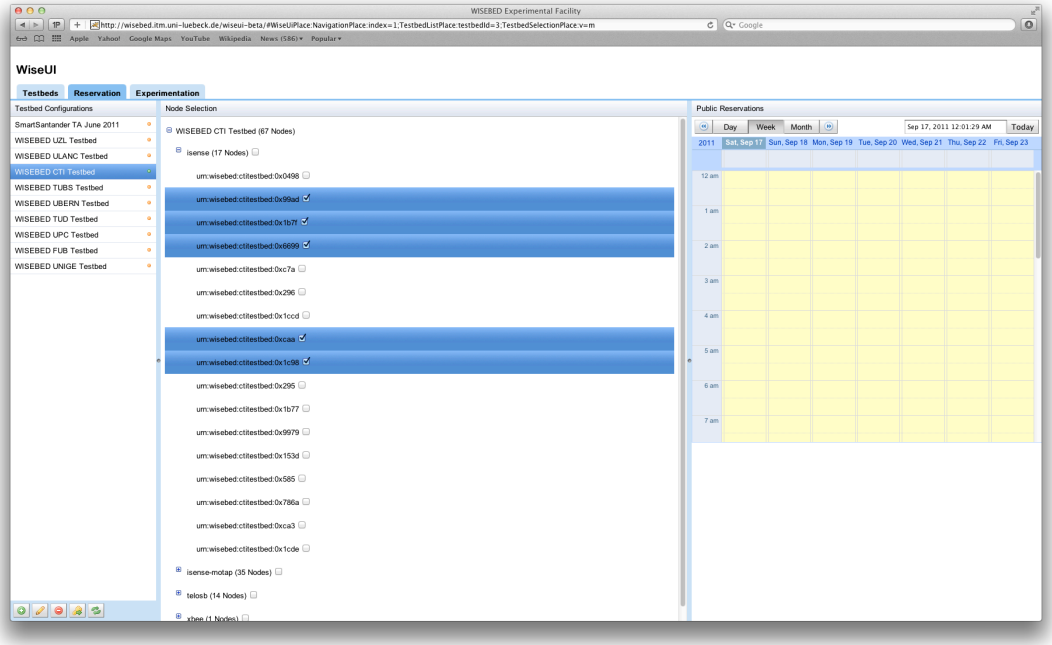
Εικόνα 6.7: Διαδικασίας αφαίρεσης νέας ΠΥ



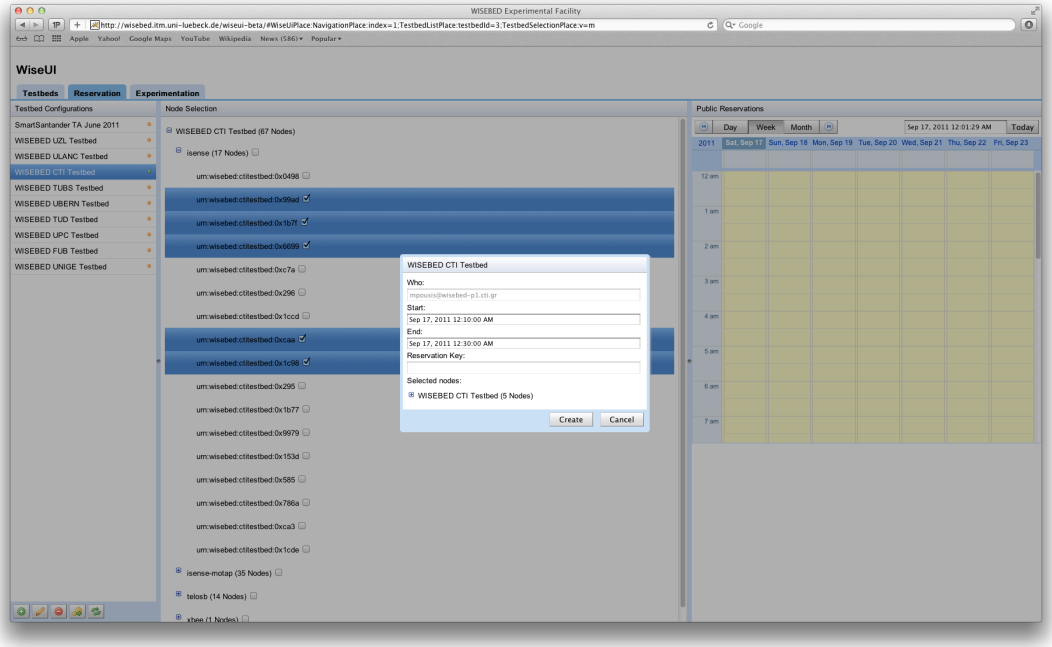
Εικόνα 6.8: Επιλογή της ΠΥ του E.A.I.T.Y



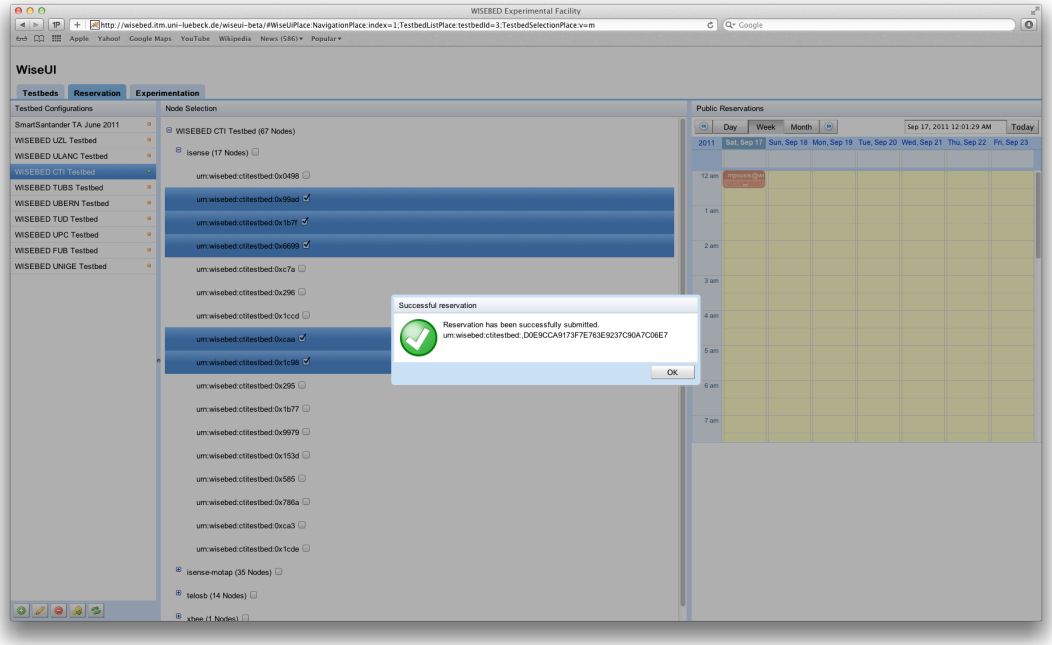
Εικόνα 6.9: Διάλογος ταυτοποίησης χρήστη στην επιλεγμένη ΠΥ



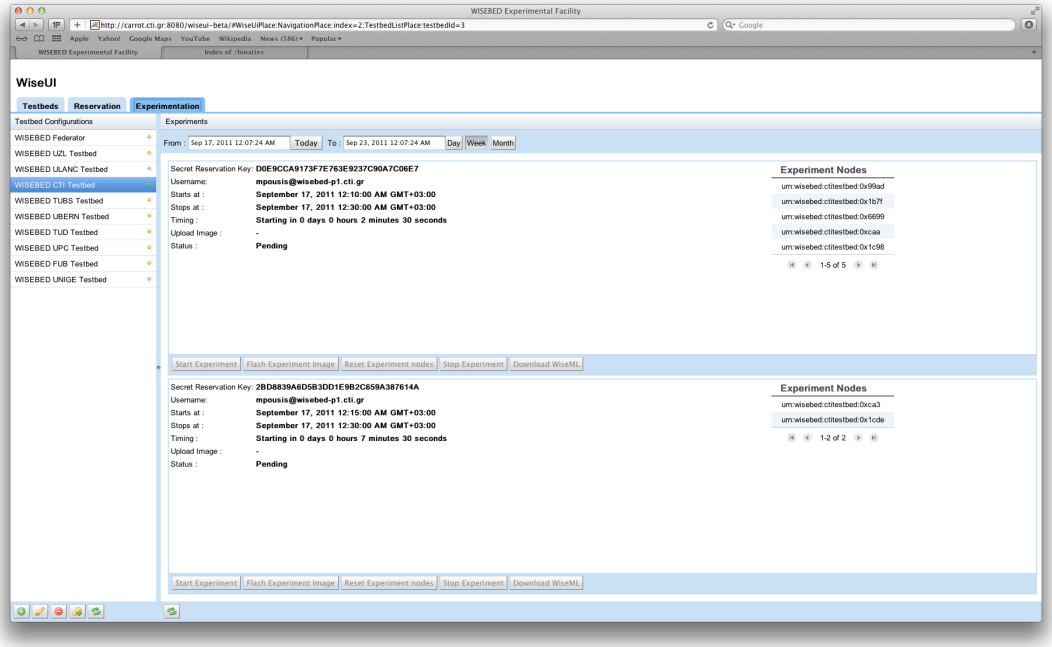
Εικόνα 6.10: Επιλογή Κόμβων για συμμετοχή σε μια κράτηση



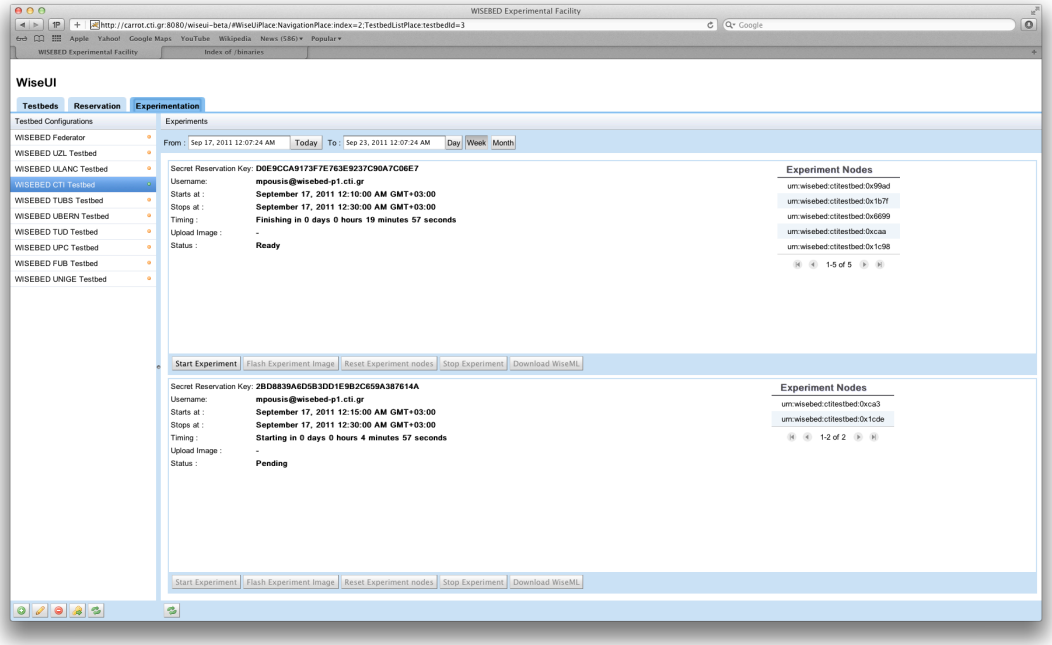
Εικόνα 6.11: Ρύθμιση τελικών παραμέτρων νέας κράτησης



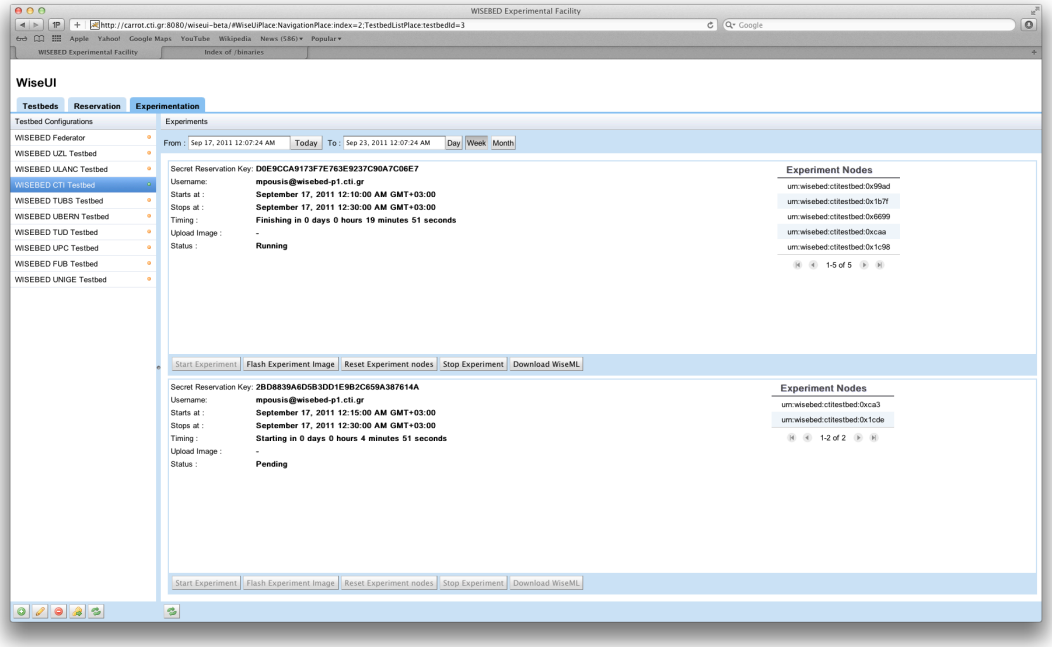
Εικόνα 6.12: Επιτυχημένη δημιουργία κράτησης



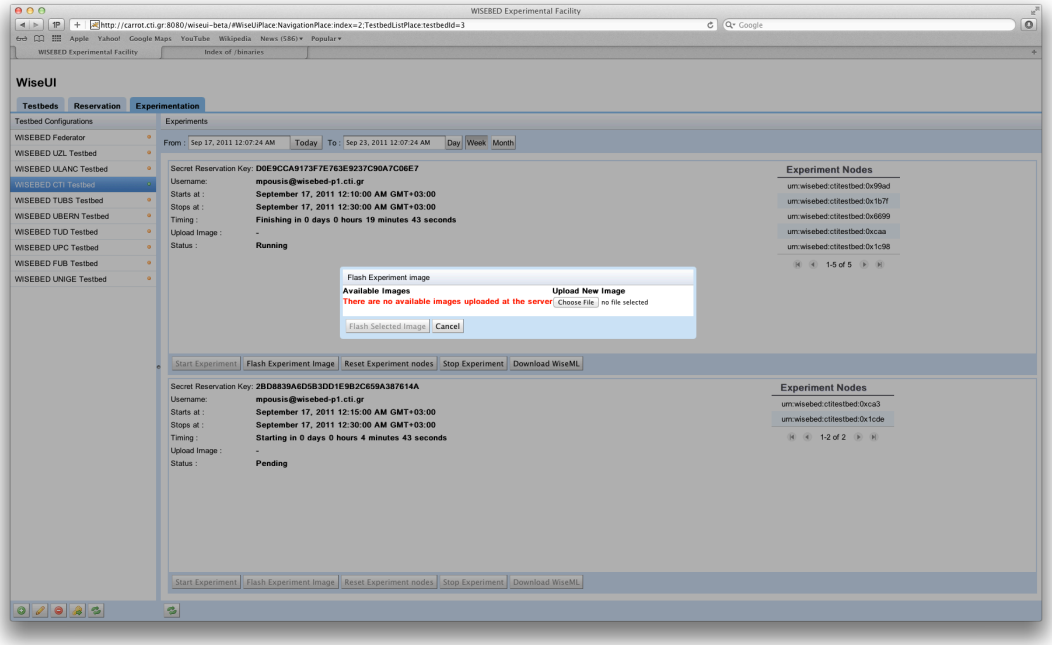
Εικόνα 6.13: Τα πανελ πειραματισμού



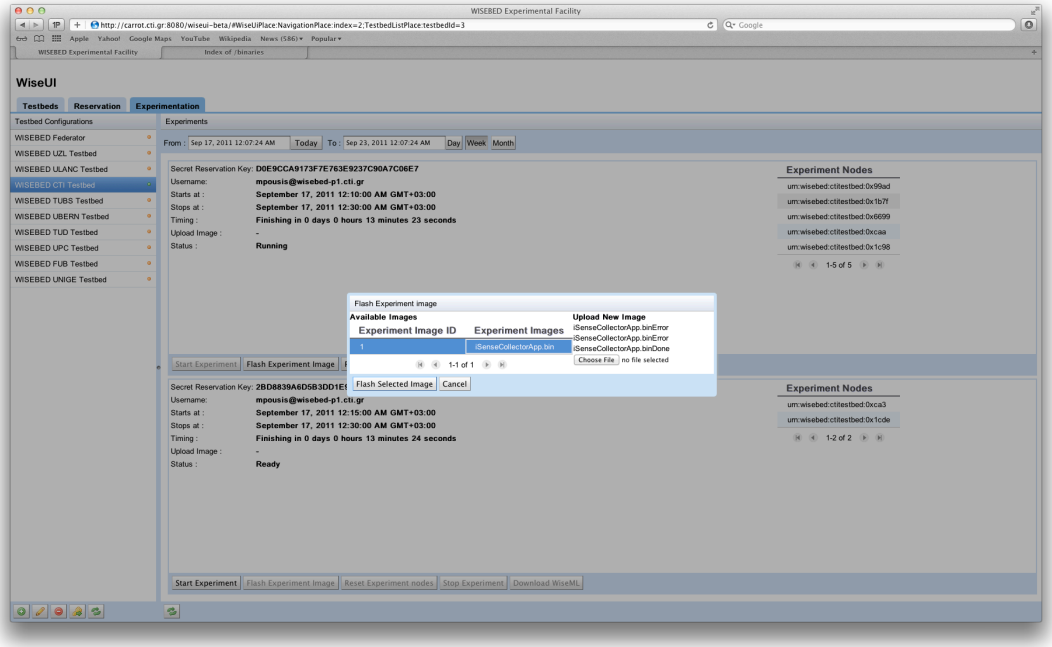
Εικόνα 6.14: Η άφιξη του χρόνου κράτησης



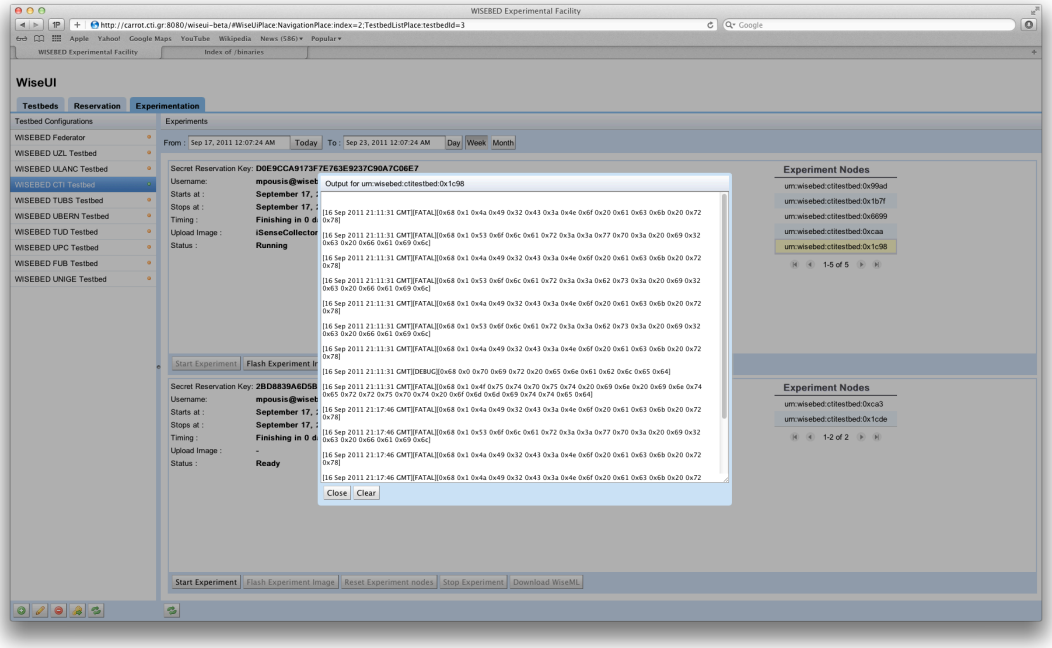
Εικόνα 6.15: Έναρξη πειραματικής διαδικασίας - Ενεργοποίηση επιλογών για πειραματισμό



Εικόνα 6.16: Ανέβασμα προγράμματος λειτουργίας



Εικόνα 6.17: Φόρτωση προγράμματος λειτουργίας στους κόμβους της κράτησης



Εικόνα 6.18: Δεδομένα από τους αισθητήρες μέσω πειραματισμού

Κεφάλαιο 7

Συμπεράσματα και μελλοντικές κατεθύνσεις

7.1 Συμπεράσματα και μελλοντικές κατεθύνσεις

Το *WiseUI*, πληρεί τις προδιαγραφές που έχουμε θέσει, οι οποίες συνοψίζονται σε μια εφαρμογή-εργαλείο για τον χρήστη ενός ετερογενούς ΑΔΑ. Η εφαρμογή παρέχει ένα διασθητικό γραφικό περιβάλλον, που επιτρέπει στο χρήστη με μεγάλη ευελιξία να δεσμεύσει κόμβους και να λάβει μια αναλυτική αναφορά της εκτέλεσης των πειραμάτων του σε δεσμευμένους κόμβους. Εκτός αυτού, πρόκειται για μια εφαρμογή ιστού άρα πλήρως ανεξαρτητη από παραμέτρους όπως το λειτουργικό σύστημα και ο περιηγητής στον οποίο εκτελείται η εφαρμογή. Παράλληλα η υπηρεσία είναι προσβάσιμη ανεξαρτήτου τοποθεσίας και για την εκτέλεσή της απαιτείται αποκλειστικά ένας περιηγητής. Η πρόσβαση σε ΑΔΑ γίνεται πιο εύκολη από κάθε άλλη φορά και ο διαδικτυακός χαρακτήρας της εφαρμογής και η άδεια ανοικτού λογισμικού με την οποία διατίεται δίνουν πολλές προοπτικές για μελλοντικές κατευθύνσεις.

Μέχρις στιγμής, το *WiseUI*, χρησιμοποιεί έναν περιορισμένο αριθμό υπηρεσιών από αυτές που παρέχονται από το περιβάλλον *Testbed Runtime* για πειραματικές πλατφόρμες. Αυτόματα λοιπόν, η λειτουργικότητα της εφαρμογής αποκτά νέες κατευθύνσεις, ενσωματώνοντας κλήσεις σε μεθόδους που δεν εμπεριέχονται στην τρέχουσα έκδοση ενώ είναι διαθέσιμες από το *TR*. Μία από τις πιθανές κατευθύνσεις είναι η δημιουργία εικονικών συνδέσεων μεταξύ αισθητήρων διαφορετικών πειραματικών υποδομών. Οι μέθοδοι `setVirtualLink` και `destroyVirtualLink` του *TR*, βρίσκονται σε στάδιο ανάπτυξης και θα μπορούσαν να ενσωματωθούν ως νέες υπηρεσίες, σε επόμενη κύκλο ανάπτυξης του *WiseUI*. Με τη δημιουργία εικονικών συνδέσεων, οι χρήστες θα αποκτήσουν πρόσβαση σε μεγαλύτερες ΠΥ με πλουσιότερη ποικιλία σε αισθητήρες, μέσω εικονικής ένωσης μικρότερων ΠΥ, ενώ θα τους δωθεί η δυνατότητα να εκτελέσουν πιο πολύπλοκα πειράματα. Ένα άλλο επιπλέον χαρακτηριστικό του *WiseUI*, θα μπορούσε να είναι η δυνατότητα ενεργοποίησης/απενεργοποίησης των κόμβων μιας πειραματικής υποδομής. Ο διαχειριστής μιας ΠΥ, αποκτά με αυτόν τον τρόπο μεγαλύτερο έλεγχο στην υποδομή. Αυτή τη στιγμή, σε περίπτωση βλάβης κάποιας συσκευής, ο διαχειριστής θα πρέπει να έχει

πρόσβαση στον Εξυπηρετητής του *TR* ώστε να αποσυνδέσει τις ελατωματικούς κόμβους μέσω ρυθμίσεων στον Εξυπηρετητής. Σε περίπτωση ενσωμάτωσης των μεθόδων `enableNode` και `disableNode` του *TR*, ο διαχειριστής θα είναι σε θέση να ενεργοποιήσει και να απενεργοποιήσει συσκευές μέσω του περιηγητή παρακολουθώντας με διαδραστικό τρόπο τις αλλαγές του στο ΑΔΑ. Εκτός από τα παραπάνω, οι υπηρεσίες που προσφέρονται από την εφαρμογή συνδέονται άμεσα με την εξέλιξη του *TR*. Έτσι, όσο προχωρά η ανάπτυξη του *TR*, τόσο πιο πλούσιο μπορεί να γίνετα και το *WiseUI* σε λειτουργικότητα.

Μία πιθανή μελλοντική κατεύθυνση, που προς το παρόν δεν συνδέεται με καμία προγραμματιστική διεπαφή, είναι να προσδωθεί στο *WiseUI* ένας κοινωνικός χαρακτήρας, κάτι που αντιπροσωπεύει την πλειονότητα των σύγχρονων διαδικτυακών εφαρμογών ιστού. Στα πλαίσια μιας τέτοιας προσθήκης, μια πολύ χρήσιμη λειτουργία θα ήταν η δυνατότητα διαμοιρασμού πειρμάτων και πειραματικών δεδομένων, τόσο σε επίπεδο πόρων (προγράμματα λειτουργίας, γραμμένα σε *Wiselib*) , αλλα ακόμα και σε επίπεδο αναφοράς (αναφορές γραμμένες σε *WiseML*). Επίσης, οι χρήστες θα μπορούν να μοιράζονται και τις κρατήσεις/πειράματα με διαμοιρασμό των ειδικών κλειδιών που αναφέραμε στο 2ο κεφάλαιο.

Η προσθήκη επιπέδων εξουσιοδότησης όσον αφορά τις άδειες χρήσης μιας πειραματικής υποδομής, θα μπορούσε να είναι μια χρήσιμη λειτουργία. Σε περίπτωση που ο διαχειριστής μπορούσε να εξουσιοδοτήσει συγκεκριμένους χρήστες για την εκτέλεση πειραμάτων σε συγκεκριμένους πειραματικούς πόρους, τότε ο έλεγχος πρόσβασης σε μια πειραματική πλατφόρμα θα ήταν πιο αυξημένος και η χρήση της θα γινόταν με μεγαλύτερη ασφάλεια.

Επιπλέον με χρήση σύγχρονων τεχνολογίων, μπορεί να επιτευχθεί μια καλύτερη αναπαράσταση της υποδομής με τη χρήση τριδίαστατων μοντέλων του δικτύου αλλά και του χώρου στον οποίο βρίσεκεται το ΑΔΑ. Μέχρις στιγμής η απεικόνιση που χρησιμοποιούμε είναι σε δύο διαστάσεις.

Πάντως, τις μελλοντικές κατευθύνσεις του *WiseUI*, θα καθορίσει σε πολύ μεγάλο βαθμό και το *GWT* καθώς είναι το περιβάλλον που χρησιμοποιήθηκε για την ανάπτυξη του λογισμικού στην πλευρά του Πελάτη. Με την είσοδο της *HTML5*[29] στις τεχνολογίες διαδικτύου, η *Google* δεν καθυστέρησε να υποστηρίξει και να αναβαθμίσει ορισμένα από τα συστατικά του *GWT*. Είναι σημαντικό να σημειωθεί, πως η δυνατότητα τοπικής αποθήκευσης στον περιηγητή (*Web Storage*)[72], μπορεί να οδηγήσει στον επανασχεδιασμό πολλών λειτουργιών που υλοποιούνται στο *WiseUI*. Η τοπική ανάκτηση δεδομένων μπορεί να αφαιρέση από τον εξυπηρέτη λειτουργικότα και να κάνει την γραφική διεπαφή πολύ πιο γρήγορη και λειτουργική καθώς έτσι μειώνεται ο επικοινωνιακός φόρτος μεταξύ περιηγητή-Εξυπηρετητής για το *WiseUI*.

Αδιαμφισβήτητα, η εφαρμογή αποτελεί απόδειξη για το ότι η χρήση της μπορεί να βρει εφαρμογή σε πραγματικές συνθήκες. Τα ΑΔΑ τείνουν να κάνουν την εμφάνισή τους όλο και πιο συχνή στην καθημερινότητά μας και ο συνδυασμός αυτών με τις δυνατότητες του ιστού, θα καταφέρει να λύσει πολλά προβλήματα και να βελτιώσει αρκετά την καθημερινότητά μας.

Εν κατακλείδη, ένα από τα πιο ασφαλή συμπεράσματα που μπορούμε να βγάλουμε είναι ότι για την ανάπτυξη του *WiseUI* έχουν χρησιμοποιηθεί οι πιο σύγχρονες τεχνολογίες και μέθοδοι για ευέλικτη ανάπτυξη λογισμικού. Η χρήση όλων των εργαλείων που παρουσιάστηκαν παραπάνω έχει ως αποτέλεσμα η εφαρμογή να είναι εύκολα επεκτάσιμη και φαίνεται ξεκάθαρα ότι υπάρχουν θετικές προοπτικές για τη υλοποίηση όλων των πιθανών μελλοντικών κατευθύνσεων που αναφέρθηκαν.

Παράρτημα : Εικόνες

Παράρτημα : Κώδικες

Αναφορές - Βιβλιογραφία

[1] Agile software development. http://en.wikipedia.org/wiki/Agile_software_development.

[2] Ajax (programming). http://en.wikipedia.org/wiki/Ajax_(programming).

[3] Apache tomcat. http://tomcat.apache.org/.

[4] Application servers. http://en.wikipedia.org/wiki/Application_server.

[5] Boost c++ library. "http://www.boost.org/".

[6] The c++ programming language. http://www.cplusplus.com/.

[7] Cardinality. http://en.wikipedia.org/wiki/Cardinality_(data_modeling).

[8] The client-server model. http://en.wikipedia.org/wiki/Client–server_model.

[9] Code coupling. http://en.wikipedia.org/wiki/Coupling_(computer_programming).

[10] Computational geometry algorithms library. http://www.cgal.org/.

[11] The contiki os. the operating system for the internet of things. http://contiki-os.blogspot.com/.

[12] Data access objects. http://en.wikipedia.org/wiki/Data_access_object.

[13] Data transfer objects. http://en.wikipedia.org/wiki/Data_transfer_object.

[14] Database management system. http://en.wikipedia.org/wiki/Database_management_system.

[15] Database transactions. http://en.wikipedia.org/wiki/Database_transaction.

[16] Declerative layout with uibinder. http://code.google.com/webtoolkit/doc/latest/DevGuideUiBinder.html.

[17] Dependency injection. http://en.wikipedia.org/wiki/Dependency_injection.

[18] Design of the hardware infrastructure, architecture of the software infrastructure design of library of algorithms. http:/www.wisebed.eu/images/stories/deliverables/d1.1-d3.1.pdf.

[19] Domain model. http://en.wikipedia.org/wiki/Domain_model.

[20] Dozer. http://dozer.sourceforge.net/.

[21] The factory method design pattern.

[22] Getting started with the beanshell client. https://github.com/itm/testbed-runtime/tree/master/clients/scripting-client.

[23] Gmail - google mail. https://mail.google.com/.

[24] Google calendar service. http://code.google.com/apis/calendar/.

[25] Google docs. https://docs.google.com/.

[26] The google web toolkit manifest. http://code.google.com/webtoolkit/makinggwtbetter.html#introduction.

[27] Google web toolkit overview. http://code.google.com/webtoolkit/overview.html.

[28] Gwt's client bundle. http://code.google.com/webtoolkit/doc/latest/DevGuideClientBundle.html.

[29] Html 5. http://el.wikipedia.org/wiki/HTML5.

[30] Http cookie. http://en.wikipedia.org/wiki/HTTP_cookie.

[31] Hypertext markup language. http://en.wikipedia.org/wiki/HTML.

[32] Hypertext transfer protocol. http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol.

[33] Ibm db2. http://www-01.ibm.com/software/data/db2.

[34] The isense platform. http://www.isensesystems.com/.

[35] Iterative and incremental development. http://en.wikipedia.org/wiki/Iterative_and_incremental_development.

[36] Java. http://www.java.com.

[37] Java annotations. .

[38] Java authentication authorization services. http://download.oracle.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html.

[39] Java servlets. http://en.wikipedia.org/wiki/Java_Servlet.

[40] Java virtual machine. http://en.wikipedia.org/wiki/Java_Virtual_Machine.

[41] Javascript. http://el.wikipedia.org/wiki/JavaScript.

[42] Jpa. http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html.

[43] Kerberos. http://web.mit.edu/kerberos/.

[44] Model-view-presenter. http://en.wikipedia.org/wiki/Model-view-presenter.

[45] Mysql. http://www.mysql.com/.

[46] Object oriented database. http://en.wikipedia.org/wiki/Object_database.

[47] Object/relational mapping. http://en.wikipedia.org/wiki/Object-relational_mapping.

[48] Object/relational paradigm mismatch. http://www.hibernate.org/about/orm.

[49] Oracle database. http://www.oracle.com/us/products/database/index.html.

[50] Persistence. http://en.wikipedia.org/wiki/Persistence_(computer_science).

[51] Pluggable authentication module. http://en.wikipedia.org/wiki/Pluggable_authentication_module.

[52] Polling. http://en.wikipedia.org/wiki/Polling_(computer_science).

[53] Race conditions. http://en.wikipedia.org/wiki/Race_condition.

[54] Relational database. http://en.wikipedia.org/wiki/Relational_database.

[55] Relational persistence for java and .net. http://www.hibernate.org/.

[56] Remote proceudre call. http://en.wikipedia.org/wiki/Remote_procedure_call.

[57] Sensor node. http://en.wikipedia.org/wiki/Sensor_node.

[58] Serialization. http://en.wikipedia.org/wiki/Serialization#Java.

[59] Shawn: A customizable sensor network simulator. https://www.itm.uni-luebeck.de/ShawnWiki/index.php/Shawn_Introduction.

[60] Shibboleth. http://shibboleth.internet2.edu/.

[61] Singleton objects. http://en.wikipedia.org/wiki/Singleton_pattern.

[62] Software testing. http://en.wikipedia.org/wiki/Software_testing.

[63] Spring framework. http://www.springsource.com/.

[64] Spring framework's transaction management. http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/transaction.html.

[65] Technical report tr-iwsn-api-v2.3: iwsn api 2.3. http://www.wisebed.eu/images/stories/deliverables/TR/TR-iWSN-API-V2.3.pdf.

[66] Technical report tr-rs-api-v1: Rs api 1.0. http://www.wisebed.eu/images/stories/deliverables/TR/TR-RS-API-V1.pdf.

[67] Technical report tr-snaa-api-v1: Snaa api 1.0. http://www.wisebed.eu/images/stories/deliverables/TR/TR-SNAA-API-V1.pdf.

[68] Test driven development. http://en.wikipedia.org/wiki/Test-driven_development.

[69] Testbed runtime. https://github.com/itm/testbed-runtime/wiki.

[70] Web browser. http://en.wikipedia.org/wiki/Web_browser.

[71] Web services. http://en.wikipedia.org/wiki/Web_service.

[72] Web storage. http://en.wikipedia.org/wiki/Web_Storage.

[73] Wireless sensor networks. http://en.wikipedia.org/wiki/Wireless_sensor_network.

[74] Wisebed - wireless sensor network testbeds - eu project. http://www.wisebed.eu/.

[75] The wiselib library. https://github.com/ibr-alg/wiselib/wiki.

[76] Wiseml schema. http://dutigw.st.ewi.tudelft.nl/wiseml/wiseml_schema.pdf.

[77] Xml. http://el.wikipedia.org/wiki/XML.

[78] Βάση Δεδομένων. http://el.wikipedia.org/wiki/□□□□_□□□□□□□□□.