Ανάπτυξη Κατανεμημένων Εφαρμογών στο Περιβάλλον Google Web Toolkit

Γάκος Ιωάννης

Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής

Πανεπιστήμιο Πατρών

Επιβλέπων Καθηγητής, Ζαρολιάγκης Χρήστος

Συνεπιβλέποντες, Χατζηγιαννάκης Ιωάννης, Θεοδωρίδης Ευάγγελος

30 Σεπτεμβρίου 2011

Ευχαριστίες

Για την βοήθεια και συνεργασία, θα ήθελα να ευχαριστήσω τους

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

Περίληψη

Στα πλαίσια της διπλωματικής εργασίας του συγγραφέα, υλοποιήθηκε η διαδικτυακή εφαρμογή WiseUI, στα πλαίσια του περιβάλλοντος WISEBED. Σκοπός της εφαρμογής είναι να παρέχει στους χρήστες ένα περιβάλλον για την ευέλικτη και γρήγορη χρήση, όλων των υπηρεσιών που προσφέρονται από ετερογενείς πειραματικές υποδομές, εξοπλισμένες με ασύρματους αισθητήρες. Περιληπτικά, στο κείμενο αυτό παρουσιάζονται όλες οι τεχνικές λεπτομέρειες που απαρτίζουν το WiseUI, των οποίων η χρήση τους έπαιξε καταλητικό ρόλο στην επίτευξη των αρχικών στόχων. Στο κείμενο αυτό, γίνεται μια εισαγωγή σχετικά με τους στόχους και το κίνητρο του WiseUI. Στη συνέχεια παρουσιάζονται οι προγραμματιστικές διεπαφές που χρησιμοποιήθηκαν, οι προκλήσεις που αντιμετωπίστηκαν, το Google Web Toolkit ως το κυρίως εργαλείο ανάπτυξης για την πλευρά του πελάτη, οι υπόλοιπες τεχνολογίες και η αρχιτεκτονική της, καθώς και οι πιθανές μελλοντικές κατευθύνσεις της εφαρμογής.

Περιεχόμενα

Περιεχόμενα

Κεφάλαιο 1

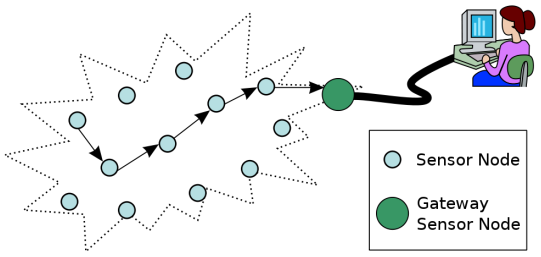
Εισαγωγή

1.1 Κίνητρο και σημασία

Βασικό κίνητρο για την παρούσα διπλωματική εργασία, στάθηκαν τα ασύρματα δίκτυα αισθητήρων (ΑΔΑ). Αν και έχουν κάνει την εμφάνισή τους από πολύ νωρίς, την τελευταία πενταετία με την αλματώδη πρόοδο της τεχνολογίας, τα ΑΔΑ τείνουν να εισβάλλουν όλο και πιο γρήγορα στην καθημερινότητά μας. Αποτελεί λοιπόν πρόκληση, να καταφέρουμε να εκμεταλλευτούμε τις δυνατότητές τους και να παρέχουμε πρόσβαση στους χρήστες, μέσω ενός αφαιρετικού μηχανισμού που δεν θα απαιτεί την γνώση τεχνολογιών χαμηλού επιπέδου. Με τη χρήση πρώιμων τεχνολογιών, η εκτέλεση πειραμάτων σε μια πειραματική υποδομή γινόταν μέσα από μια εφαρμογή γραμμής εντολών, χωρίς δηλαδή να παρέχεται κάποιο γραφικό περιβάλλον. Αυτό έκανε την διαδικασία αρκετά πολύπλοκη και κουραστική, γεγονός που δημιούργησε την ανάγκη ανάπτυξης μιας πλούσιας διαδικτυακής εφαρμογής, που θα ήταν προσβάσιμη μέσω οποιουδήποτε περιηγητή. Το WiseUI έρχεται να καλύψει τις παραπάνω ανάγκες, εκμεταλλευόμενο όλες τις δυνατότητες των ΑΔΑ μέσω ενός γραφικού περιβάλλοντος, αποκρύπτοντας παράλληλα από τον χρήστη όλες τις λεπτομέρειες που υλοποιούνται σε χαμηλότερα επίπεδα.

Ένα ασύρματο δίκτυο αισθητήρων, αποτελείται από χωρικά κατανεμημένους αυτόνομους αισθητήρες για την παρακολούθηση φυσικών ή περιβαλλοντικών συνθηκών όπως είναι η θερμοκρασία, ο ήχος, η δόνηση, η πίεση, η κίνηση και τα επίπεδα ρύπων, δεδομένα τα οποία μπορούν να συλλεχθούν και να αποσταλούν με συνεργατικό τρόπο σε μια κεντρική τοποθεσία (Εικόνα 1.1). Στην πλειονότητα των ΑΔΑ, η επικοινωνία μπορεί να γίνει αμφίδρομα, επιτρέποντας με αυτόν τον τρόπο τον έλεγχο της λειτουργίας των αισθητήρων. Παρά το γεγονός ότι τα ΑΔΑ έκαναν την πρώτη εμφάνισή τους στον στρατό, όπου χρησιμοποιούνταν στα στρατόπεδα για αμυντικούς σκοπούς, σήμερα χρησιμοποιούνται σε βιομηχανικές αλλά και καθημε-

ρινές εφαρμογές, όπως είναι η διαδικασία βιομηχανικής παρακολούθησης.



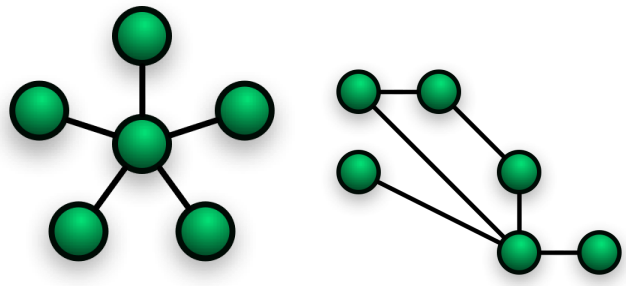
Εικόνα 1.1: Τυπική αρχιτεκτονική ενός ΑΔΑ.

Αποτελείται από κόμβους, των οποίων ο αριθμός μπορεί να κυμαίνεται από μερικές εκατοντάδες αλλά και χιλιάδες, όπου κάθε κόμβος είναι συνδεδεμένος με έναν ή περισσότερους αισθητήρες. Κάθε ΑΔΑ αποτελείται από ένα σύνολο επιμέρους κομματιών. Αυτά είναι ένας πομποδέκτης με μια εσωτερική κεραία ή κάποια σύνδεση σε εξωτερική κεραία, ένας μικροελεγκτής, δηλαδή ένα ηλεκτρονικό κύκλωμα ως περιβάλλον επικοινωνίας με τους αισθητήρες και μία πηγή ενέργειας, συχνά μία μπαταρία ή κάποια άλλη ενσωματωμένη λύση. Ένας κόμβος μπορεί να ποικίλει σε μέγεθος από ένα κουτί παπουτσιών μέχρι και έναν κόκο σκόνης, αν και για μικροσκοπικά μεγέθη δεν έχουν κατασκευαστεί ακόμη πλήρως λειτουργικοί αισθητήρες. Το κόστος των αισθητήρων μπορεί και αυτό να κυμαίνεται από μερικές εκατοντάδες ευρώ μέχρι και ένα μονοψήφιο αριθμό, ανάλογα με την πολυπλοκότητα του καθενός. Οι σταθερές μεγέθους και κόστως των συσκευών έχουν αντίστοιχη συμπεριφορά όπως με αυτές των μνημών, της ενέργειας, της υπολογιστικής ταχύτητας και του εύρους ζώνης επικοινωνίας. Η τοπολογία των ΑΔΑ μπορεί να ποικίλει από ένα απλό δίκτυο με αρχιτεκτονική αστέρα μέχρι και ένα προχωρημένο multi-hop πλέγμα (Εικόνα 1.2) [19] [13]. Οι τεχνικές διάδοσης μεταξύ των ενδιάμεσων κόμβων ενός δικτύου μπορεί να βασίζεται είτε σε απλή δρομολόγηση είτε στη μέθοδο της πλημμύρας [27].

Με την πάροδο του χρόνου, τα ΑΔΑ έχουν βρει εφαρμογή σε πολλές περιοχές, αρκετές φορές βελτιώνοντας την καθημερινότητά μας αλλά και την ποιότητα ζωής, ενώ παρέχουν δυνατότητες που στο παρελθόν ήταν τεχνολογικά ανέφικτες. Με τα δίκτυα αυτά, γίνεται πιο αποτελεσματική η επίβλεψη περιοχών, όπου ένα ΑΔΑ τοποθετείται για την παρακολούθηση ενός φαινομένου όπως η κίνηση. Για παράδειγμα, οι αισθητήρες χρησιμοποιούνται για την παρακολουθηση αγωγών πετρελαίου ή και για τον έλεγχο εκπομπής αερίων. Τα ΑΔΑ χρησιμοποιούνται επίσης για την παρακολούθηση των επιπέδων ρύπανσης σε αρκετές πόλεις όπως το Λονδίνο, προστατεύοντας με αυτόν τον τρόπο τους πολίτες από συγκεντρώσεις επικίνδυνων αερίων. Μπορούν επίσης να εγκατασταθούν σε ένα δάσος για τον εντοπισμό

εστίας πυρκαγιών, όπου οι κόμβοι είναι εξοπλισμένοι με κατάλληλους αισθητήρες για τον έλεγχο της θερμοκρασίας, της υγρασίας και των αερίων που παράγονται από την πυρκαγιά των φυτών. Με αυτόν τον τρόπο, γίνεται ευκολότερη η πρόληψη της καταστροφής των δασών, καθώς η εκάστοτε πυροσβεστική υπηρεσία ειδοποιείται άμεσα για πιθανές εστίες. Η διαχείριση των θερμοκηπίων γίνεται πιο ευέλικτη και αποτελεσματική, με τους αισθητήρες να ενημερώνουν τον επαγγελματία για τα επίπεδα θερμοκρασίας και υγρασίας στον εσωτερικό χώρο. Σε περίπτωση που κάποιο από τα φαινόμενα που παρακολουθούνται, πέσουν κάτω από τα επιτρεπτά επίπεδα, υπάρχει η δυνατότητα να ειδοποιηθεί ο διαχειριστής του θερμοκηπίου είτε μέσω ηλεκτρονικού ταχυδρομείου, είτε μέσω γραπτού μηνύματος στο κινητό τηλέφωνο. Παράλληλα μπορούν να ενεργοποιηθούν αυτόματα ανεμιστήρες, μηχανισμοί υγρασίας και διάφορα άλλα συστήματα υποστήριξης, χωρίς να είναι απαραίτητη η φυσική παρουσία του διαχειριστή στο χώρο του θερμοκηπίου. Με την τοποθέτηση κόμβων στο έδαφος, μπορούν να γίνουν αντιληπτά φαινόμενα κατολίσθησης, με τη βοήθεια αισθητήρων που είναι σε θέση να εντοπίζουν την κίνηση του εδάφους.

Στο χώρο της βιομηχανίας, τα ΑΔΑ μπορούν να χρησιμοποιηθούν για τη μηχανική επιτήρηση υγείας ανάλογα με την κατάσταση του κάθε ασθενούς, καθώς προσφέρουν σημαντικά οικονομικά οφέλη και παρέχουν νέες διευκολύνσεις. Επίσης, στη γεωργία η χρήση ασύρματων αισθητήρων μπορεί να διευκολύνει την εργασία του επαγγελματία, καθώς μπορεί απομακρυσμένα να ελέγξει τις στάθμες του νερού, τη ροή των αγωγών και πολλά άλλα. Τέλος, τα ΑΔΑ βρίσκουν εφαρμογή και στο χώρο των κατασκευών, όπου γίνεται δυνατός ο έλεγχος της κίνησης σε κτίρια και υποδομές, όπως γέφυρες και σύραγγες.



Εικόνα 1.2: Τοπολογία αστέρα και τοπολογία πλέγματος.

1.2 Στόχος

Έχοντας παρουσιάσει όλα τα κίνητρα και τη σημασία των ΑΔΑ, ως στόχος τίθεται η δημιουργία ενός σύγχρονου εργαλείου για την εκμετάλλευση των δυνατοτήτων που παρέχονται από τις πειραματικές πλατφόρμες. Όμως τα ΑΔΑ, πρέπει να διατηρούν κάποια βασικά χαρακτηριστικά, γεγονός που κάνει τόσο την κατασκευή όσο και την χρήση τους μια πολύπλοκη διαδικασία. Αναφορικά, ένα χαρακτηριστικό είναι οι σταθερές κατανάλωσης ενέργειας για κάθε κόμβο που τροφοδοτείται με μπαταρίες. Η αντιμετώπιση πιθανής αποτυχίας κάποιων κόμβων όπως και η κινητικότητά τους, αποτελούν βασικές παραμέτρους των ΑΔΑ. Πιθανές αποτυχίες κατά την επικοινωνία καθώς και η ετερογένεια των κόμβων θα πρέπει να ληφθούν υπόψιν. Η επεκτασιμότητα, η ικανότητα να μπορούν να ανταπεξέλθουν σε δύσκολες καιρικές συνθήκες, η ευκολία χρήσης και η απρόβλεπτη συμπεριφορά των κόμβων αποτελούν επιπλέον χαρακτηριστικά.

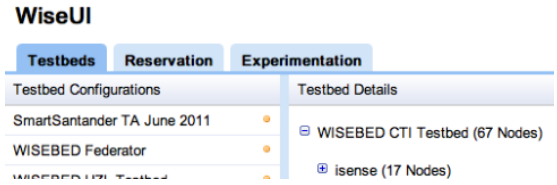
Όπως αναφέρθηκε και παραπάνω, όλα αυτά τα χαρακτηριστικά κάνουν την χρήση των ΑΔΑ όλο και πιο πολύπλοκη. Τίθεται λοιπόν, ως βασικός στόχος, η χρήση των πειραματικών υποδομών να γίνει με όσο το δυνατόν πιο εύκολο και ανώδυνο τρόπο. Η μέχρι τώρα απαίτηση από τους χρήστες να είναι εξειδικευμένοι σε συγκεκριμένες τεχνολογίες, προκειμένου να καταφέρουν να χρησιμοποιήσουν την πειραματική υποδομή, αποτελεί ένα πολύ σημαντικό μειονέκτημα. Θα πρέπει λοιπόν να ληφθεί υπόψιν, ότι οι χρήστες πρέπει να μπορούν να εκτελέσουν τα πειράματά τους αντιμετωπίζοντας όλες τις λεπτομέρειες μιας πειραματικής πλατφόρμας ως ένα μαύρο κουτί.

1.3 Συνεισφορά

Οι υπηρεσίες που προσφέρει το WiseUI, αναφέρονται αποκλειστικά στη λειτουργικότητα των πειραματικών υποδομών. Μέσω της εφαρμογής αυτής, ο χρήστης μπορεί να δει όλες τις λεπτομέρειες που παρέχονται σχετικά με μια πειραματική υποδομή. Μπορεί να δημιουργήσει κρατήσεις σε κάποιο υποσύνολο των αισθητήρων που βρίσκονται εγκατεστημένοι σε μια πειραματική πλατφόρμα, καθώς επίσης να προχωρήσει είτε στην διαμόρφωση των παραμέτρων μιας ήδη υπάρχουσας κράτησης, είτε την πλήρη διαγραφή της. Για την εκτέλεση των πειραμάτων ο χρήστης μπορεί να αποθηκεύσει σε σχεσιακή βάση δεδομένων, κώδικα χαμηλού επιπέδου ο οποίος και θα εκτελεστεί από τους αισθητήρες την στιγμή που ο ίδιος ορίσει. Παράλληλα, μπορεί να λαμβάνει τα μηνύματα που παράγουν οι αισθητήρες ζωντανά κατά την διάρκεια εκτέλεσης του πειράματος αλλά και να ελέγχει τους αισθητήρες που συμμετέχουν σε αυτό. Τέλος, μετά το πέρας της εκτέλεσης του πειράματος παρέχεται δομημένη αναφορά, την οποία ο χρήστης μπορεί να αποθη-

κεύσει τοπικά για περαιτέρω μελέτη.

Η εφαρμογή WiseUI, συγκεντρώνει όλες τις παραπάνω υπηρεσίες που προσφέρονται από τις πειραματικές υποδομές σε ένα αρκετά διαισθητικό γραφικό περιβάλλον, ενώ διευκολύνει και επιταχύνει εκπληκτικά όλες τις ενέργειες που απαιτούνται για την εκμετάλλευση των πόρων μιας υποδομής. Στα πλαίσια της παρούσας διπλωματικής εργασίας, έγινε κύρια συνεισφορά στην υλοποίηση του μηχανισμού κρατήσεων. Ξεκινώντας από το πιο χαμηλό επίπεδο, υλοποιήθηκε η ασύγχρονη επικοινωνία του πελάτη με τον διακομιστή, για την αποστολή όλων των απαραίτητων δεδομένων που απαιτούνται για τη δημιουργία μιας νέας κράτησης. Παράλληλα, χτίσαμε την αντίστοιχη γραφική διεπαφή, έχοντας ως κύριο άξονα τη διευκόλυνση του χρήστη κατά την αλλιλεπίδρασή του με την εφαρμογή. Στο σύνολο της εφαρμογής WiseUI, υπήρξε επίσης σημαντική συνεισφορά σε θεμελιώδη τμήματα που την απαρτίζουν, όπως είναι ο σχεδιασμός και ο μηχανισμός διαχείρισης της βάσης δεδομένων. Τέλος, καθ' όλη τη διάρκεια της ανάπτυξης, προχωρήσαμε σε προσθήκες, διορθώσεις και βελτιστοποιήσεις που αφορούσαν την εκσφαλμάτωση και τις αποδόσεις της εφαρμογής.



Το λογισμικό διατίθεται υπό την άδεια ελεύθερου λογισμικού Apache License 2.0 και το κείμενό της εμπεριέχεται στον πηγαίο κώδικα της εφαρμογής. Όλες οι τεχνολογίες και τα εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής, είναι ελεύθερου λογισμικού και λογισμικού ανοιχτού κώδικα.

1.4 Δομή

Στο κείμενο αυτό παρουσιάζονται όλες οι προγραμματιστικές διεπαφές που χρησιμοποιήθηκαν, για την διασύνδεση των υπηρεσιών του WiseUI με τις πειραματικές υποδομές, οι πιο σημαντικές προκλήσεις που αντιμετωπίσαμε, τα εργαλεία που χρησιμοποιήθηκαν όχι μόνο για την ανάπτυξη του WiseUI αλλά και για την αντιμετώπιση πολλών από τις προκλήσεις, η αρχιτεκτονική της εφαρμογής, οι νέες υπηρεσίες που προσφέρονται καθώς και ένα σενάριο εκτέλεσης για την πλήρη κατανόηση του αναγνώστη ως προς τη χρήση της εφαρμογής.

Όσον αφορά την προγραμματιστική διασύνδεση, εκεί θα αναφερθούμε αρχικά στην

πλατφόρμα Wisebed, σε μια προσπάθεια να δώσουμε μια αφαιρετική εικόνα σχετικά με τις υπηρεσίες που καλείται να υλοποιήσει το WiseUI. Στο ίδιο κεφάλαιο θα παρουσιάσουμε το Testbed Runtime (TR), ένα πακέτο από βιβλιοθήκες για την προγραμματιστική διασύνδεση της εφαρμογής με υπηρεσίες ταυτοποίησης, δημιουργίας κρατήσεων καθώς και επικοινωνίας με τους πειραματικούς πόρους μιας υποδομής.

Στο κεφάλαιο 3, παρουσιάζονται οι πιο κρίσιμες από τις προκλήσεις που κληθήκαμε να αντιμετωπίσουμε κατά την διάρκεια ανάπτυξης της εφαρμογής. Θα παρουσιάσουμε θέματα που αφορούν την ασύγχρονη επικοινωνία μεταξύ πελάτη και εξυπηρετητή, τη μεταφορά αντικειμένων από τη μία πλευρά στην άλλη, τα θέματα που παρουσιάζονται εν όψει της ανάγκης για μόνιμη αποθήκευση δεδομένων σε κατάλληλη σχεσιακή βάση καθώς και το σύστημα διαχείρισης αυτής. Επιπρόσθετα, θα αναφερθούμε σε μεθόδους σχετικά με την ευέλικτη ανάπτυξη λογισμικού, που βρίσκουν εφαρμογή στο σύνολο της βιομαχανίας λογισμικού και όχι αποκλειστικά στο WiseUI, ενώ τέλος θα παρουσιαστούν τεχνικές για την εξασφάλιση ποιότητας με τη χρήση δοκιμών.

Κατά την προσπάθεια αντιμετώπισης όλων των παραπάνω προκλήσεων, καταλήξαμε στη χρήση κάποιων εργαλείων που προσέφεραν τις πιο ικανοποιητικές λύσεις, σε σχέση πάντα με τις ανάγκες της εφαρμογής μας. Εκεί παρουσιάζουμε στο σύνολό του το Google Web Toolkit, το κυρίως πακέτο λογισμικού που χρησιμοποιήθηκε για την ανάπτυξη της εφαρμογής. Θα αναφερθούμε σε κάποιες βιβλιοθήκες που προσφέρουν ευελιξία όσον αφορά την έγχυση εξαρτήσεων ανά το λογισμικό, την μεταφορά αντικειμένων, τον τρόπο αντιμετώπισης εκτέλεσης παράλληλων λειτουργιών με χρήση Spring Transactions, το σύστημα διαχείρισης που χρησιμοποιήσαμε για τη βάση δεδομένων μας και τέλος το πλαίσιο για την ανάπτυξη των δοκιμών της εφαρμογή μας.

Στο κεφάλαιο 5, παρουσιάζεται η αρχιτεκτονική του WiseUI σε επίπεδο λογισμικού, όπου φαίνονται όλα τα πρότυπα σχεδιασμού καθώς και οι τεχνικές που χρησιμοποιήθηκαν τόσο στην πλευρά του πελάτη όσο και στο κομμάτι του εξυπηρετητή. Έπειτα παρουσιάζεται αναλυτικά ένα σενάριο εκτέλεσης και τέλος κάνουμε μια επισκόπηση στο σύνολο της εφαρμογής μιλώντας για τα συμπεράσματά μας μετά το πέρας της ανάπτυξης, καθώς και όλα τα σχέδια που υπάρχουν ως μελλοντικές κατευθύνσεις του WiseUI.

Κεφάλαιο 2

Προγραμματιστική διασύνδεση – Βιβλιοθήκες (API)

2.1 Το περιβάλλον Wisebed

Στόχος του έργου WISEBED είναι να παρέχει στην ερευνητική κοινότητα μια πολυεπίπεδη υποδομή από διασυνδεδεμένες πειραματικές πλατφόρμες, εξοπλισμένες με δίκτυα ασύρματων αισθητήρων, επιδιώκοντας τον διεπιστημονικό συνδυασμό υλικού, λογισμικού, αλγορίθμων και δεδομένων. Με αυτόν τον τρόπο, θα κατασκευαστεί, όχι μόνο ένα μεγάλο δίκτυο, αλλά μία υψηλά επεκτάσιμη δομή από πολλαπλά μικρού βελενικούς δίκτυα συσκευών και πειραματικών υποδομών. Έτσι, η έρευνα θα μπορεί πιο εύκολα να επικεντρωθεί σε εφαρμογές πάνω σε ποιοτικότερες υποδομές, εξαιτίας της ανομοιογένειας των πειραματικών υποδομών, την τοποθεσία αλλά και τις ιδιότητες των μελών της.

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 ευέλικτη εφαρμογή των θεωρητικών αποτελεσμάτων σε αλγορίθμους, μηχανισμούς και πρωτόκολλα για την κατασκευή λογισμικού.

Νέα δεδομένα και συμπεράσματα θα μπορέσουν να εξαχθούν, καθορίζοντας έτσι

μια νέα πορεία για την ερευνητική δραστηριότητα. Στόχος είναι, η ενοποιημένη αυτή αρχιτεκτονική υποδομών να είναι διαθέσιμη προς εκμετάλλευση και χρήση από την ευρύτερη ακαδημαϊκή κοινότητα κατευθύνοντας με αυτόν τον τρόπο τα κατανεμημένα πρότυπα σε ένα διαφορετικό επίπεδο [28].

2.2 Το πακέτο λογισμικού Testbed Runtime(TR)

Το Testbed Runtime αναφέρεται συνολικά στην υλοποίηση του λογισμικού της πλατφόρμας WISEBED. Στα πλάισια του TR, υλοποιούνται όλες οι προγραμματιστικές διεπαφές για τη διαχείρηση των δικτύων ασύρματων αισθητήρων, που βρίσκονται εγκατεστημένα στις διάφορες πειραματικές υποδομές. Ονομαστικά, οι διεπαφές αυτές είναι οι iWSN, RS και SNAA, οι οποίες περιγράφονται με λεπτομέρεια παρακάτω.

Διαδοχικά στις τρεις αυτές παραγράφους, θα παρουσιαστούν οι βασικές κλήσεις που χρησιμοποιούνται κατά την εκτέλεση ενός κλασσικού σεναρίου χρήσης μιας πειραματικής υποδομής. Οι προγραμματιστικές διεπαφές, περιέχουν προφανώς πολλές επιπλέον μεθόδους αλλά στο κείμενο αυτό, θα περιοριστούμε στις πιο βασικές και χρήσιμες για τον προγραμματιστή. Η λειτουργικότητα που παρέχουν αυτές οι μέθοδοι, είναι η ταυτοποίηση και εξουσιοδότηση ενός χρήστη για πρόσβαση και χρήση μιας πειραματικής υποδομής, η δημιουργία μιας κράτησης σε συγκεκριμένους πειραματικούς πόρους, η τροφοδότηση των πόρων που ο χρήστης έχει δεσμεύσει με ένα πείραμα και τέλος η άντληση της εξόδου που παράγουν οι αισθητήρες για παρουσίαση στον τελικό χρήστη [21].

2.2.1 Η διεπαφή SNAA

Ομοίως με το RS, έτσι και το SNAA είναι μια προγραμματιστική διεπαφή. Η συγκεκριμένη είναι υπεύθυνη για την πιστοποίηση των στοιχείων του χρήστη καθώς και την εξουσιοδότηση αυτού για δέσμευση, πρόσβαση και χρήση των διαφόρων πειραματικών υποδομών. Αυτές είναι οι δύο βασικές λειτουργίες που παρέχει, ενώ η διαδικασία που ακολουθείται κάθε φορά διαμορφώνεται ως εξής. Αρχικά γίνεται η πιστοποίηση του χρήστη, εξασφαλίζοντας έτσι το γεγονός ότι ο χρήστης έχει πρόσβαση στην συγκεκριμένη υποδομή. Σε περίπτωση που η ταυτοποίηση των στοιχείων του χρήστη είναι πετυχημένη, τότε δίνεται στον ίδιο ένα μυστικό κλειδί

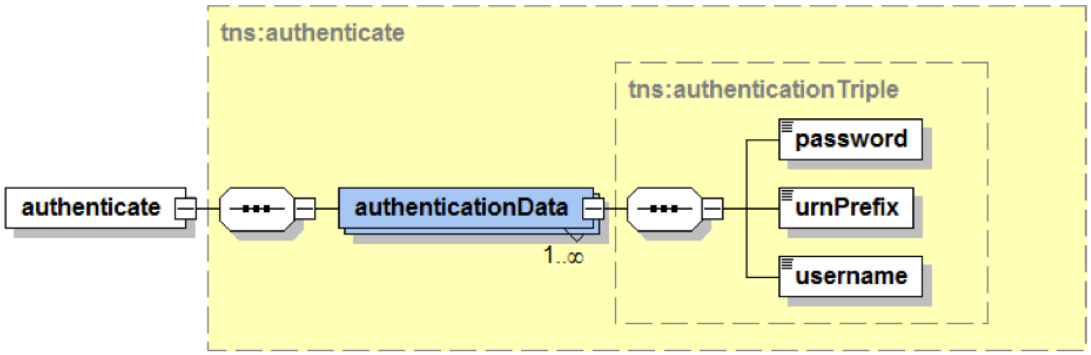
πιστοποίησης γνωστό και ως secretAuthenticationKey. Ο χρήστης παρουσιάζει αυτό το κλειδί στο RS, κάθε φορά που θέλει να δεσμεύσει κάποιους κόμβους και το RS με τη σειρά του ελέγχει αν ο συγκεκριμένος χρήστης είναι εξουσιοδοτημένος για πρόσβαση στους πειραματικούς πόρους.

Η πιστοποίηση και εξουσιοδότηση του χρήστη γίνεται μέσω της μεθόδου authenticate, που παρέχεται από το πακέτο του SNAA. Παρακάτω παρουσιάζεται αναλυτικά το περιβάλλον εκτέλεσης της συγκεκριμένης μεθόδου.

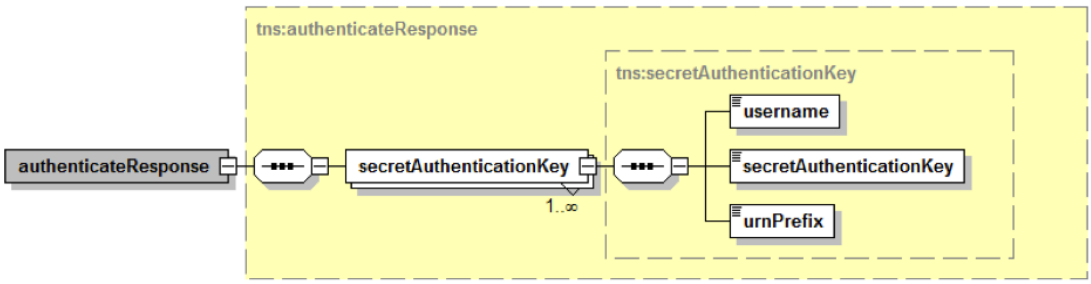
Σημασιολογία	Επιστρέφει μία λίστα από κλειδιά ταυτοποίησης για ένα σύνολο αναγνωριστικών αλλιώς σηματοδοτεί κάποιο σφάλμα
Υπογραφή	List<SecretAuthenticationKey> authenticate(List<AuthenticationTriple > authenticationData) throws AuthenticationExceptionException , SNAAExceptionException;
Επιστρέφει	Μια πλειάδα (urnprefix, secretauthenticationkey) για την αντίστοιχη πλειάδα που έχει δεχθεί ως είσοδο αλλιώς σηματοδοτεί κάποιο σφάλμα

Όπως αναφέρθηκε και παραπάνω, η μέθοδος authenticate χρησιμοποιείται για την ταυτοποίηση των χρηστών. Δέχεται ως όρισμα μία ακολουθία από πλειάδες (urnprefix, secretauthenticationkey) για την αντίστοιχη πλειάδα που έχει δεχθεί ως είσοδο, αλλιώς σηματοδοτεί κάποιο σφάλμα. Στην εικόνα 2.1 απεικονίζεται η είσοδος, ενώ στην εικόνα 2.2 φαίνεται η δομή της απάντησης, όπως αυτή διαμορφώνεται μετά το πέρας της διαδικασίας ταυτοποίησης. Η μέθοδος αυτή, ολοκληρώνει την εκτέλεσή της χωρίς τη σηματοδότηση κάποιου σφάλματος, μόνο σε περίπτωση που η ταυτοποίηση για όλες τις πλειάδες αναγνωριστικών που τις έχουν δοθεί ώς είσοδοι είναι επιτυχής. Σε αντίθετη περίπτωση θα σημειωθεί η εξαίρεση AuthenticationExceptionException. Εάν προκύψει κάποιο άλλο σφάλμα, τότε η συνάρτηση θα σηματοδοτήσει την εξαίρεση γενικότερου ενδιαφέροντος SNAAExceptionException. Οι δύο αυτές εξαιρέσεις ορίζονται από το ίδιο το πακέτο. Όπως η συνάρτηση εκτελεστεί επιτυχώς, επιστρέφει μια πλειάδα (urnprefix, secretauthenticationkey) για κάθε urnprefix που έχει δεχθεί ως είσοδο.

Ανάλογα με την πειρατική υποδομή στην οποία εφαρμόζεται, η διεπαφή μπορεί είτε να υποστηρίζει την αποδοχή μιας λίστας με μία μόνο καταχώρηση (π.χ. λόγω του ότι η συγκεκριμένη υποδομή μπορεί να εξυπηρετεί μόνο ένα πρόθεμα πόρων), είτε να μπορεί να δεχτεί αναγνωριστικά για περισσότερα από ένα προθέματα πόρων [16].



Εικόνα 2.1: Αίτηση ταυτοποίησης



Εικόνα 2.2: Απάντηση ταυτοποίησης

2.2.2 Η διεπαφή RS

Η προγραμματιστική διεπαφή του RS, είναι κατάλληλη για την δέσμευση και γενικότερη διαχείρηση των διάφορων πειραματικών υποδομών. Οι προδιαγραφές και η λειτουργικότητα της συγκεκριμένης βιβλιοθήκης, καθορίζονται από τις κοινότητες που είναι ενεργές στα πλαίσια του έργου WISEBED.Ο προγραμματιστής μπορεί μέσω του RS να δεσμεύσει μέρος των πειραματικών πόρων που παρέχονται από το σύνολο μιας υποδομής. Έτσι, χρησιμοποιώντας τα στοιχεία ταυτοποίησης που του έχουν ανατεθεί, μπορεί να δημιουργεί πολλαπλές κρατήσεις για συγκεκριμένα χρονικά διαστήματα. Η βιβλιοθήκη παρέχει μια πλειάδα λειτουργιών στον προγραμματιστή, που του δίνει τη δυνατότητα να ανακτήσει μία λίστα με όλες τις κρατήσεις που του ανήκουν αλλά και για συγκεκριμένα χρονικά διαστήματα. Για τη δημιουργία μίας νέας κράτησης σε προκαθορισμένους από το χρήστη πειραματικούς πόρους, χρησιμοποιείται η μέθοδος makeReservation που παρέχεται από το πακέτο RS. Παρακάτω παρουσιάζεται αναλυτικά το περιβάλλον εκτέλεσης της συγκεκριμένης μεθόδου.

Σημασιολογία	Δεσμεύει ένα σύνολο κόμβων για ένα συγκεκριμένο διάστημα χρόνου
Υπογραφή	List <SecretReservationKey> makeReservation(List<SecretAuthenticationKey > authenticationData , ConfidentialReservationData reservation) throws AuthorizationExceptionException, RSExceptionException, ReservervationConflictExceptionException ;
Παράμετροι	authenticationData Μια ακολουθία από πλειάδες (urnprefix, secretauthenticationkey). Αυτές οι πλειαδες έχουν ανακτηθεί μετά την κλήση της authorize() που βρίσκεται στο πακέτο SNAA. reservation Πληροφορίες για το ποιοί κόμβοι πρόκειται να δεσμευθούν και για ποιό διάστημα χρόνου.
Επιστρέφει	Μια πλειάδα (urnprefix, secretauthenticationkey) για την αντίστοιχη πλειάδα που έχει δεχθεί ως είσοδο αλλιώς σηματοδοτεί κάποιο σφάλμα.

Η μέθοδος αυτή, χρησιμοποιείται για τη δέσμευση ενός συνόλου από κόμβους για ένα συγκεκριμένο χρονικό διάστημα.

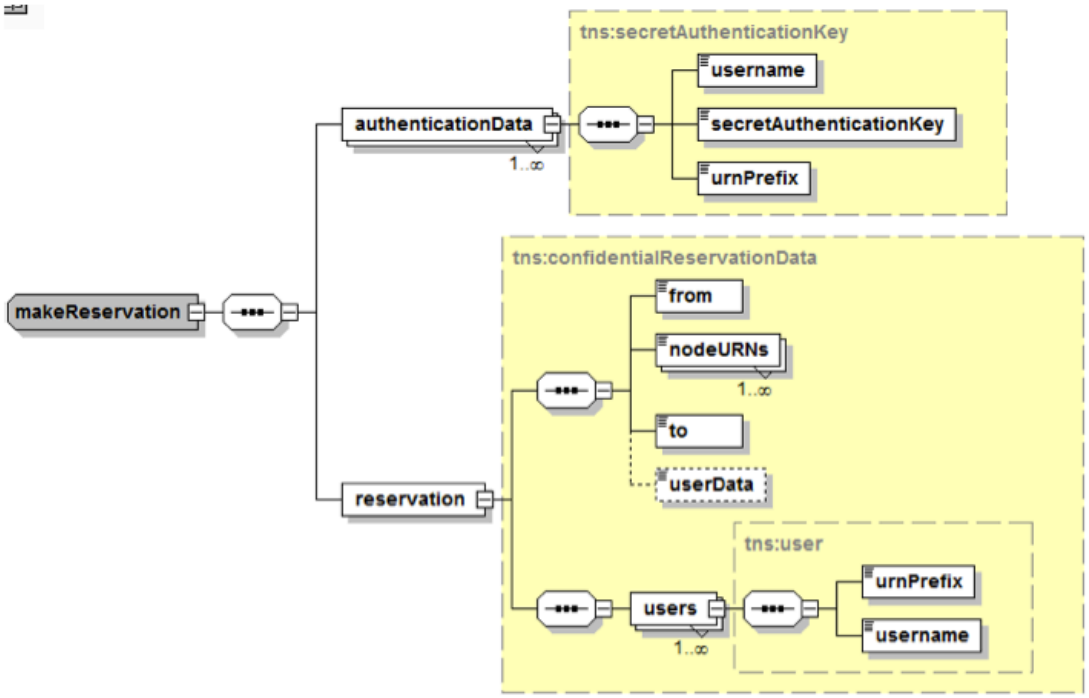
Η πρώτη παράμετρος, περιέχει τα μυστικά κλειδιά ταυτοποίησης που έχουν επιστραφεί από την κλήση της μεθόδου authorize() που βρίσκεται στο SNAA. Το σύστημα κρατήσεων αναλαμβάνει να διασταυρώσει την εξουσιοδότηση του χρήστη με κλήση της μεθόδου isAuthorized(), παρέχοντας σε αυτή τα παραπάνω μυστικά κλειδιά.

Η δεύτερη παράμετρος, περιέχει μια λίστα με τα αναγνωριστικά όλων των κόμβων που προορίζονται για δέσμευση, για το χρονικό διάστημα που ορίζεται μεταξύ των συμπεριλαμβανόμενων παραμέτρων from και to. Η παράμετρος userData μπορεί να περιέχει αυθαίρετη (δημόσιας προβολής) πληροφορία σχετικά με την κράτηση. Η συγκεκριμένη πληροφορία μπορεί να φανεί χρήσιμη στον πελάτη για την αποθήκευση επιπρόσθετης πληροφορίας στο σύστημα κρατήσεων.

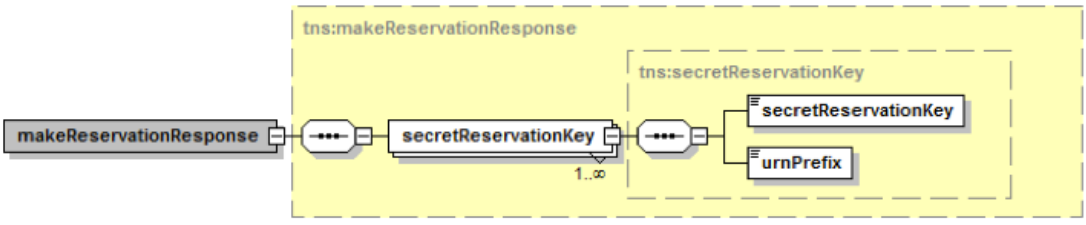
Επιπρόσθετα, το σύστημα κρατήσεων ελέγχει για το αν η συγκεκριμένη κράτηση μπορεί να πραγματοποιηθεί, εκτελώντας εσωτερικά ελέγχους, όπως για παράδειγμα αν το σύνολο των κόμβων που προορίζονται για κράτηση δεν είναι δεσμευμένοι από κάποιο άλλο χρήστη για το ορισμένο χρονικό διάστημα.

Σε περίπτωση που επιβεβαιωθεί η εξουσιοδότηση και η διαθεσιμότητα όλων των κόμβων, η συνάρτηση επιστρέφει ένα σύνολο από πλειάδες (urnprefix, secretreser-

vationkey) για κάθε πλειάδα που εχει δεχθεί ως είσοδο, αλλιώς σηματοδοτεί κάποιο σφάλμα. Σε περίπτωση που η κράτηση δεν είναι δυνατόν να πραγματοποιηθεί λόγω του ότι οι παράμετροί της συμπιπτουν με μία άλλη ήδη υπάρχουσα κράτηση, τότε σηματοδοτείται μία εξαίρεση ReservationConflict που εμπεριέχεται στο πακέτο RS. Για σφάλματα που προκύπτουν εσωτερικά στο σύστημα κρατήσεων, επιστρέφεται ένα σφάλμα RS. Στις εικόνες 2.3 και 2.4 απεικονίζονται αντίστοιχα τα μηνύματα αίτησης για τη δημιουργία μιας νέας κράτησης και της απάντησης της συνάρτησης [15].



Εικόνα 2.3: Αίτηση δημιουργίας νέας κράτησης



Εικόνα 2.4: Απάντηση δημιουργίας νέας κράτησης

2.2.3 Το σύνολο διεπαφών iWSN

Το iWSN είναι μια προγραμματιστική διεπαφή που διαχωρίζεται σε τρία διαφορετικά πακέτα, ανάλογα με τη λειτουργικότητα που παρέχει το καθένα. Αυτά είναι το Session Management API, το Controller API και τέλος το WSN API.

2.2.3.1 Η διεπαφή Session Management API

Πιο αναλυτικά, το Session Management API, προσφέρει λειτουργίες για την παροχή πρόσβασης ανά πείραμα στο WSN API. Για παράδειγμα περιέχει μεθόδους που επιστρέφουν στον χρήστη πληροφορίες σχετικά με την εγκατάσταση της εκάστοτε πειραματικής υποδομής, την κατάσταση στην οποία βρίσκονται συγκεκριμένοι κόμβοι, καθώς επίσης είναι χρήσιμη για την έναρξη μιας νέας συνόδου χρήσης των πειραματικών πόρων.

Σε μερικές περιπτώσεις η διεπαφή WSN θα είναι μοναδική οντότητα, οπότε ο εξυπηρετητής της πειραματικής πλατφόρμας θα επιστρέφει πάντα την ίδια διεύθυνση (αυτή είναι και η προκαθορισμένη συμπεριφορά σε ιδιωτικές πειραματικές υποδομές). Για τους υπόλοιπους χρήστες θα επιστραφεί μια νέα οντότητα της διεπαφής η οποία όμως έχει δημιουργηθεί σε διαφορετική πόρτα σύνδεσης. Αυτή η συμπεριφορά είναι απαραίτητη σε περίπτωση που διαφορετικοί χρήστες δεσμεύουν διαφορετικούς πόρους από την ίδια πειραματική πλατφόρμα. Μια πιθανή εναλλακτική υλοποίηση θα ήταν να χρησιμοποιηθούν αναγνωριστικά κλειδιά συνόδου για κάθε νέα κλήση που γίνεται στη διεπαφή του Wisebed. Παρ' όλα αυτά μία τέτοια λύση παρουσιάζεται προβληματική σε πολλές περιπτώσεις, κάτι που κάνει την υλοποίησή της αρκετά πιο πολύπλοκη.

Για την εξυπηρέτηση μιας νέας συνόδου, είναι απαραίτητο να είναι γνωστές κάποιες πληροφορίες καθώς και οι προδιαγραφές της πειραματικής υποδομής, όπως για παράδειγμα τον τύπο των αισθητήρων που υπάρχουν εγκατεστημένοι στην πλατφόρμα. Το Session Management API παρέχει ένα σύνολο από μεθόδους που εξυπηρετούν αυτές τις ανάγκες.

2.2.3.2 Η διεπαφή Controller API

Το Controller API είναι η προγραμματιστική διεπαφή που στην ουσία επιτυγχάνει την επικοινωνία με τους κόμβους της εκάστοτε πειραματικής υποδομής. Η επικοινωνία με τους αισθητήρες γίνεται με ασύγχρονο τρόπο και οι ίδιοι τροφοδοτούνε τις μεθόδους του Controller API κάθε φορά που παράγουν κάποιο μήνυμα, ανάλογα με τις δυνατότητές τους. Με τις μεθόδους που παρέχονται από αυτό, δίνεται η δυνατότητα για ενημέρωση σχετικά με την κατάσταση στην οποία βρίσκεται ένας αισθητήρας, όσον αφορά μια λειτουργία που έχει κληθεί να εκτελέσει, την

λήψη με ασύγχρονο τρόπο της εξόδου ενός κόμβου, καθώς και την ειδοποίηση για τον τερματισμό εκτέλεσης ενός πειράματος. Όπως είπαμε, η επικοινωνία με τους πειραματικούς πόρους βασίζεται στην ασύγχρονη επικοινωνία και αυτό γιατί ένας από τους πιο βασικούς στόχους του έργου WISEBED, είναι η κλιμακωσιμότητα. Έτσι όταν γίνεται ανάθεση εκτέλεσης μιας λειτουργίας σε έναν μεγάλο αριθμό πειραματικών πόρων, η οντότητα που έχει πραγματοποιήσει την κλήση δεν περιμένει τον τερματισμό αυτής αλλά συνεχίζει κανονικά την εκτέλεσή της.

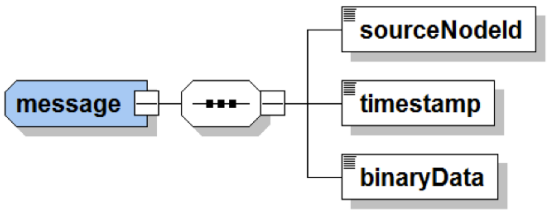
Ασύγχρονες λειτουργίες (όπως η επαναφορά κάποιων κόμβων στην αρχική τους κατάσταση ή η αποστολή κάποιου εκτελέσιμου προγράμματος), επιστρέφουν ένα αναγνωριστικό (τύπου RequestId), έτσι ώστε ο ελεγκτής πειράματος να μπορεί να αντιστοιχίσει τις ασύγχρονες απαντήσεις του με τις αρχικές αιτήσεις που είχε δεχθεί. Η εικόνα 2.5 αναπαριστά αυτό τον τύπο δεδομένων που σε ιδανική περίπτωση είναι μια μοναδική ακολουθία χαρακτήρων που λειτουργεί ως αναγνωριστικό [12].

Εικόνα 2.5: Τύπος μηνύματος αίτησης

Η αναφορά κατάστασης στην οποία βρίσκεται η εκτέλεση μιας λειτουργίας γίνεται στον ελεγκτή, μέσω μηνυμάτων των οποίων ο τύπος αναπαρίσταται στην εικόνα 2.6. Σε ένα μήνυμα ανανεωμένης κατάστασης, περιέχονται το αρχικό αναγνωριστικό της αίτησης και ένας αριθμός από τιμές που κάθε μία τους αντιστοιχεί σε έναν κόμβο. Για παράδειγμα, όταν προγραμματίσουμε έναν αριθμό από κόμβους, ένα μήνυμα κατάστασης περιέχει ολοκληρωμένη ή επιμέρους πληροφορία σχετικά με την πρόοδο της συνολικής λειτουργίας, όπως λόγου χάριν η πληροφορία για το εκάστοτε υποσύνολο επαναπρογραμματισμένων κόμβων. Ο ελεγκτής πρέπει από εκεί και έπειτα να παρακολουθεί την πρόοδο μιας λειτουργίας μέσω των πιθανών μηνυμάτων που εισρέουν, μέχρι τη στιγμή που όλοι οι κόμβοι αναφέρουν επιτυχία ή αποτυχία. Το μήνυμα ανανεωμένης κατάστασης περιέχει το αναγνωριστικό του κόμβου από τον οποίο προέρχεται το παραχθέν μήνυμα, μία αριθμητική τιμή και ένα μήνυμα που είναι αναγνώσιμο από άνθρωπο. Η ερμηνεία της τιμής εξαρτάται από το είδος της εργασίας που εκτελείται. Όπως αναφέρθηκε και παραπάνω,

Εικόνα 2.6: Τύπος μηνύματος κατάστασης αίτησης

τα μηνύματα που παράγει ο εκάστοτε κόμβος βασίζονται σε χαρακτήρες, παρ' όλα αυτά μπορούν πλέον να μεταφέρουν και δυαδική πληροφορία. Στην εικόνα 2.7, φαίνεται ότι το μήνυμα μπορεί να περιέχει και μετα-πληροφορία, όπως για παράδειγμα η χρονική στιγμή παραγωγής του μηνύματος ή ο τύπος του μηνύματος, αν δηλαδή το μήνυμα περιέχει πληροφορία εκσφαλμάτωσης ή σφάλματος [12].



Εικόνα 2.7: Τύπος δεδομένων μηνύματος

Κεφάλαιο 3

Προκλήσεις

Στο κεφάλαιο αυτό, καταγράφονται όλες οι προκλήσεις που κληθήκαμε να αντιμετωπίσουμε κατά τη διάρκεια ανάπτυξης της εφαρμογής. Θα αναφερθούμε στην ανάγκη της ασύγχρονης επικοινωνίας μεταξύ πελάτη και διακομιστή, στον τρόπο με τον οποίο τα διάφορα αντικείμενα μεταφέρονται 'πάνω από το σύρμα', στο μοντέλο αποθήκευσης δεδομένων, στην ανάγκη για γρήγορη και χωρίς λάθη ανάπτυξη.

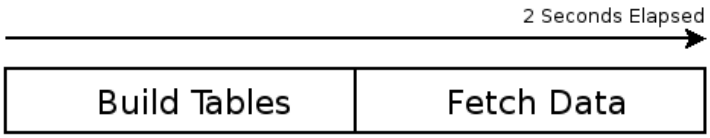
3.1 Ασύγχρονη επικοινωνία

Όπως σε κάθε πολύπλοκη διαδικτυακή εφαρμογή, έτσι και στη δική μας υπήρξε η ανάγκη για ασύγχρονη επικοινωνία μεταξύ πελάτη και διακομιστή. Οι σύγχρονες εφαρμογές απαιτούν αλληλεπιδραστική συμπεριφορά, που σημαίνει την ανταλλαγή δεδομένων από και προς τον διακομιστή, χωρίς να είναι απαραίτητη η εναλλαγή της εικόνας που βλέπει ο τελικός χρήστης στην οθόνη του. Τα δεδομένα μεταφέρονται χωρίς η μία πλευρά να αναμένει κάποια ανταπόκριση και έτσι ο τελικός χρήστης μπορεί να συνεχίζει την εκτέλεση της εφαρμογής, αδιαφορώντας για το αν έχει λάβει κάποια απάντηση η όχι από τον διακομιστή. Το μοντέλο αντικειμένων, ανεξάρτητα από τον περιηγητή που χρησιμοποιείται, μπορούν να προσπελαστούν με τη χρήση JavaScript για τη δυναμική τους παρουσίαση και έτσι ο τελικός χρήστης να είναι σε θέση να αλληλεπιδράσει με την πληροφορία που έρχεται στην οθόνη του [2].

Για να γινει πιο κατανοητό, θα φέρουμε ένα παράδειγμα όπου στη δική μας εφαρμογή παρουσιάζεται η ανάγκη για ασύγχρονη επικοινωνία με τον διακομιστή. Όπως θα δούμε και παρακάτω που θα δώσουμε και περισσότερες λεπτομέρειες σχετικά με το WiseUI, ένα κομμάτι της υπηρεσίας λειτουργεί έτσι ώστε να λαμβάνει μηνύματα από τον εκάστοτε κόμβο χωρίς ο πελάτης να ζητά συνεχώς για νέα έξοδο.

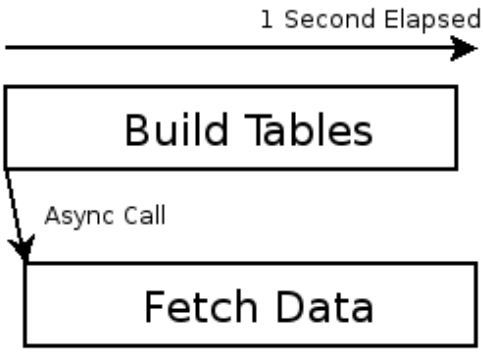
Ο εκάστοτε αισθητήρας παράγει τη δική του πληροφορία και αυτή φθάνει στον τελικό χρήστη, χωρίς αυτός να κάνει κάποια ενέργεια.

Ακόμα πιο διασθητικά, για να δούμε με νούμερα πόσο αποτελεσματική μπορεί να είναι αυτή η μέθοδος επικοινωνίας, σκεφτείτε ότι η κατασκευή ενός πίνακα απαιτεί ένα δευτερόλεπτο και για τη λήψη των δεδομένων από τον διακομιστή για την τοποθέτηση στον πίνακα απαιτείται άλλο ένα. Σε περίπτωση που η επικοινωνία πραγματοποιηθεί με σύγχρονο τρόπο, η όλη διαδικασία απαιτεί τουλάχιστον δύο δευτερόλεπτα για να ολοκληρωθεί.



Εικόνα 3.1: Σύγχρονη επικοινωνία

Εάν όμως η λήψη των δεδομένων γίνει με ασύγχρονο τρόπο, ο χρόνος που απαιτείται για την κατασκευή και συμπλήρωση του πίνακα είναι ένα δευτερόλεπτο, παρά το γεγονός ότι στην πραγματικότητα γίνεται δουλειά δύο δευτερολέπτων [7].



Εικόνα 3.2: Ασύγχρονη επικοινωνία

3.2 Μεταφορά αντικειμένων

Ο τρόπος με τον οποίο θα γίνεται η ανταλλαγή της πληροφορίας μεταξύ πελάτη και διακομιστή, είναι μια πολύ σημαντική πρόκληση, καθώς από αυτήν εξαρτώνται

σημαντικοί παράγοντες της εφαρμογής όπως είναι η ταχύτητα και η σταθερότητά της. Στην επιστήμη της πληροφορικής, στην περιοχή της αποθήκευσης και μετάδοσης δεδομένων, η μεταφορά αντικειμένων είναι η διαδικασία κατά την οποία μια δομή δεδομένων ή πιο απλά ένα αντικείμενο, μετατρέπεται σε μια συγκεκριμένη μορφή, η οποία με τη σειρά της μπορεί να μεταφερθεί πάνω από έναν σύνδεσμο. Στη συνέχεια, μόλις αυτή φθάσει στον προορισμό της μπορεί να επανακατασκευασθεί και να επανέλθει στην αρχική της μορφή, χωρίς παραλλαγές και εξαρτήσεις από ετερογενή υπολογιστικά περιβάλλοντα [17].

Το ερώτημα εδώ, το οποίο στην ουσία αποτελεί και την πρόκληση, είναι γιατί τα αντικείμενα που ανακτώνται από τη βάση δεδομένων, δεν μπορούν να γίνουν 'κατανοητά' τη στιγμή που φθάσουν στον περιηγητή για παρουσίαση. Αυτό συμβαίνει διότι, όταν ένα αντικείμενο μετατραπεί σε αντικείμενο αντιστοίχησης για την αποθήκευσή του σε μία βάση δεδομένων, αυτό το αντικείμενο εμπλουτίζεται με την αρμόζουσα πληροφορία που δείχνουν για το που προορίζεται. Αυτός ο εμπλουτισμός δεν μπορεί να γίνει χωρίς την ενσωμάτωση αυτής της πληροφορίας στην ήδη υπάρχουσα ακουλουθία πληροφορίας του ίδιου του αντικειμένου. Στη περίπτωση της Hibernate, για την οποία θα αναφερθούμε αναλυτικά σε επόμενο κεφάλαιο, η βιβλιοθήκη Javassist αντικαθιστά και επανεγγράφει το bytecode αυτών των αντικειμένων, με οντότητες που προορίζονται προς αποθήκευση στη βάση δεδομένων, έτσι ώστε η Hibernate να μπορέσει να συνεχίσει τις λειτουργίες της. Για το μηχανισμό του GWT για την απομακρυσμένη κλήση διεργασιών, αυτό σημαίνει ότι τη στιγμή που το αντικείμενο είναι έτοιμο να μεταφερθεί 'πάνω από το σύρμα', στην πραγματικότητα πρόκειται για διαφορετικό αντικείμενο από αυτό που ο μεταγλωττιστής είχε αρχικά καταλάβει ότι θα μεταφερθεί. Έτσι λοιπόν, αυτό που συμβαίνει κατά την προσπάθεια του deserialize, ο μηχανισμός του GWT RPC, δεν γνωρίζει πλέον τον νέο τύπο του αντικειμένου και αρνείται να το κάνει deserialize.

Μία ενδεχόμενη λύση στο παραπάνω πρόβλημα, θα ήταν η αντικατάσταση των τύπων των αντικειμένων για ακόμη μια φορά στην αντίθετη κατεύθυνση, πριν γίνει επιστροφή από την απομακρυσμένη κλήση της διεργασίας στην πλευρά του διακομιστή. Κάτι τέτοιο είναι εφικτό και θα μπορούσε να προσφέρει μία λύση στο πρόβλημα της μεταφοράς αντικειμένων, παρ' όλα αυτά ακόμα δεν έχουμε αποφύγει κάποια σημαντικά μειονεκτήματα. Ένα άλλο πολύ σημαντικό πλεονέκτημα της χρήσης Hibernate, είναι το γεγονός ότι μπορούμε να φορτώσουμε όσα συνδεδεμένα αντικείμενα χρειαζόμαστε και όχι όλες τις συσχετίσεις. Για παράδειγμα, στην πλευρά του διακομιστή, θα μπορούσαμε να φορτώσουμε από τη λίστα επαφών του τηλεφώνου μας μία καταχώρηση, να την αλλάξουμε, και να φορτώσουμε τις συνδεδεμένες εγγραφές με μία κλήση της account.getRecords() και μερικές ενέργειες που έγιναν πάνω σε αυτές τις εγγραφές. Οι ξεχωριστές λειτουργίες της Hibernate, θα φροντίσουν στην πραγματικότητα να φέρουν τις εγγραφές όταν γίνει

η κλήση της μεθόδου, κάνοντας€ αυτές διαθέσιμες μόνο όταν υπάρξει ανάγκη για να τις χρησιμοποιήσουμε.

Σύμφωνα με όλα τα παραπάνω, είναι αναμενόμενο να φανταστεί κανείς, ότι αυτό θα προκαλέσει μια περίεργη συμπεριφορά του μηχανισμού κλήσης απομακρυσμένων διεργασιών του GWT, τη στιγμή που τα Hibernate αντικείμενα προσπαθήσουν να ταξιδέψουν από την Java πλευρά του διακομιστή στην περιοχή του περιηγητή. Σε περίπτωση που μία GWT RPC υπηρεσία επιχειρήσει να αποκτήσει πρόσβαση στις συσχετίσεις με τον τρόπο που περιγράφηκε, είναι πολύ πιθανό να δούμε την άρση μιας εξαίρεσης όπως η LazyInitializaionException [26].

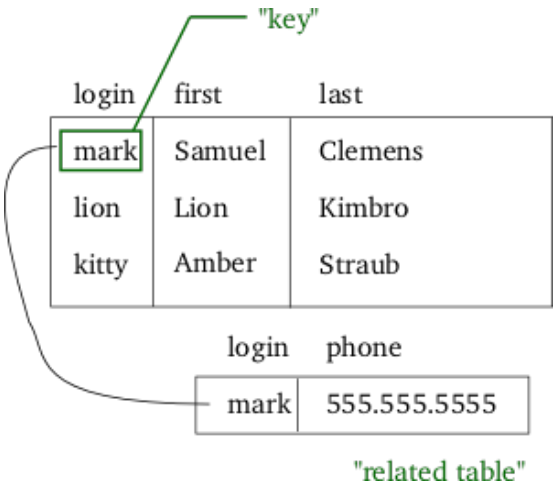
3.3 Αποθήκευση δεδομένων & Σύστημα Διαχείρισης Βάσης Δεδομένων

Στην επιστήμη της πληροφορικής, με τον όρο persistence αναφερόμαστε στο χαρακτηριστικό που έχει μια κατάσταση να επιβιώνει μετά τον τερματισμό της διεργασίας που τη δημιούργησε αρχικά. Χωρίς αυτή τη δυνατότητα, η πληροφορία για μια οποιαδήποτε κατάσταση, θα μπορούσε να μείνει αποκλειστικά στην μνήμη τυχαίας προσπέλασης και θα χανόταν τη στιγμή που η μνήμη θα αποφορτιζόταν, όπως ακριβώς συμβαίνει κατά τη διακοπή λειτουργίας ενός υπολογιστή.

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 [14]. Προφανώς τα παραπάνω δεν αποτελούνε κάποια πρόκληση στις μέρες μας αλλά γίνεται μια αναφορά για λόγους καλύτερης ροής της λογικής του κειμένου.

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. Πρόκειται για ένα πακέτο λογισμικού, που ελέγχει την δημιουργία, επίβλεψη και χρήση μιας βάσης δεδομένων. Βοηθά τους διαχειριστές των ΒΔ, 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. Μπορούν να χρησιμοποιούν ποικιλία

από μοντέλα ΒΔ, όπως είναι το μοντέλο συσχετίσεων (εικόνα 3.3) ή το μοντέλο αντικειμένων, 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 ΒΔ που απλοποιούν σε πολύ σημαντικό βαθμό την συγγραφή των εφαρμογών. 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 ΒΔ, ύστερα από κάποιο σφάλμα μέσω αντιγράφων ασφαλείας όπως και την διαχείριση θεμάτων ασφαλείας [4].



Εικόνα 3.3: Κλειδί συσχετίσεως

3.4 Ευέλικτη ανάπτυξη λογισμικού

Είναι πολύ κρίσιμο κατά την ανάπτυξη οποιασδήποτε εφαρμογής, είτε μιλάμε για το διαδίκτυο είτε για την 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 επαναληπτικές και αυξητικές μεθόδους ανάπτυξης [1][11].

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 δοκιμών, συμπεριλαμβάνουν τη διαδικασία εκτέλεσης ενός προγράμματος ή μιας εφαρμογής με την πρόθεση να βρεθούν λάθη και προβλήματα στο λογισμικό.

Η διαδικασία δοκιμής λογισμικού θα μπορούσε να οριστεί ως, η διαδικασία επικύρωσης και επιβεβαίωσης ότι ένα προϊόν λογισμικού:

1. Πληρεί όλες τις προδιαγραφές όπως αυτές ορίζονται από το σχεδιασμό και την ανάπτυξη.
2. Λειτουργεί όπως αναμενόταν – και
3. 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 [18][20].

Κεφάλαιο 4

Εργαλεία ανάπτυξης - Τεχνολογίες

Στο κεφάλαιο αυτό, παρουσιάζονται οι τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη 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 εργαλείο προγραμματισμού Google Web Toolkit

Το Google Web Toolkit (GWT) είναι ένα σύνολο από εργαλεία, για την υλοποίηση και βελτιστοποίηση πολύπλοκων διαδικτυακών εφαρμογών. Η ίδια η Google χρησιμοποιεί τα εργαλεία αυτά για την ανάπτυξη δικών της προϊόντων, που εκατομμύρια χρήστες έχουν στη διάθεσή τους 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 και βιβλιοθήκες όπως η Javascript και η XMLHttpRequest. Αυτός ήταν και ένας από τους κύριους λόγους για τον οποίο επιλέξαμε να χρησιμοποιήσουμε αυτή την τεχνολογία, για την δημιουργία της δικής μας διαδικτυακής εφαρμογής.

Ένα πολύ ισχυρό χαρακτηριστικό του GWT, είναι ότι η υλοποίηση των εφαρμογών γίνεται μέσω της προγραμματιστικής γλώσσας JAVA, μια γλώσσα ευρέως διαδεδομένη και ίσως η πιο ώριμη που υπάρχει μέχρι σήμερα στη βιομηχανία της τεχνολογίας λογισμικού. Το GWT παρέχει ένα σύνολο από προγραμματιστικές διεπαφές

γραμμένες σε JAVA, καθώς και αρκετά γραφικά που μπορούν να ενσωματωθούν πολύ εύκολα σε μια νέα εφαρμογή και να υποστούν πολλαπλές προσαρμογές από τον εκάστοτε προγραμματιστή. Αυτό επιτρέπει στον μηχανικό να γράψει εύκολα εφαρμογές που επικοινωνούν με τον διακομιστή με ασύγχρονο τρόπο (AJAX), καθώς μπορεί αυτόματα να τις συντάξει έτσι ώστε να είναι συμβατές με την πλειονότητα των προγραμμάτων περιήγησης, είτε για σταθερούς υπολογιστές είτε ακόμα και για κινητές συσκευές.

Εικόνα 4.1: Εικονίδιο Google Web Toolkit

Παράλληλα, το GWT παρέχει στον προγραμματιστή μια πλειάδα βοηθητικών λειτουργιών, που διαδραματίζουν πολύ σημαντικό ρόλο στην γρήγορη και ορθή ανάπτυξη της εφαρμογής. Η εκσφαλμάτωση γίνεται με το ίδιο ακριβώς τρόπο όπως θα γινόταν κατά την ανάπτυξη μιας οποιαδήποτε διαδικτυακής εφαρμογής σε Javascript. Στον μηχανικό, παρέχεται η δυνατότητα να προβεί εύκολα σε αλλαγές στον κώδικά του και αμέσως να δει τις επιπτώσεις αυτών στο πρόγραμμα περιήγησης. Την ίδια στιγμή, μπορεί να επιβλέπει τις εναλλαγές στις τιμές των μεταβλητών, να θέτει σημεία διακοπής στην εκτέλεση του κώδικα καθώς επίσης και να χρησιμοποιήσει οποιοδήποτε άλλο πρόσθετο εργαλείο εκσφαλμάτωσης, συμβατό με Java εφαρμογές.

Ένα ιδιαίτερο χαρακτηριστικό του GWT, είναι τα εργαλεία που διαθέτει για τη βελτιστοποίηση των εφαρμογών. Το πρόγραμμα που παρέχεται για την ανάγνωση του πηγαίου κώδικα και τη μετατροπή του σε εκτελέσιμο, κάνει περιεκτικές βελτιστοποιήσεις σε όλη τη βάση του κώδικα που έχει συντάξει ο προγραμματιστής. Για παράδειγμα γίνεται αυτόματη αφαίρεση κομματιών κώδικα που στην ουσία δεν εκτελούνται ποτέ καθώς και η τμηματοποίηση και ελαχιστοποίηση του κώδικα που πρέπει να φθάσει τον τελικό χρήστη προς εκτέλεση, επιταχύνοντας έτσι την εκκίνηση της εφαρμογής [9].

4.2 Το λογισμικό Guice & GIN

Το Guice είναι ένα ανοιχτού κώδικα πλαίσιο λογισμικού για τη γλώσσα Java και είναι γραμμένο από τους μηχανικούς της Google. Παρέχει υποστήριξη για την

έγχυση εξαρτήσεων με τη χρήση σχολίων, με σκοπό τη διαχείριση Java αντικειμένων.

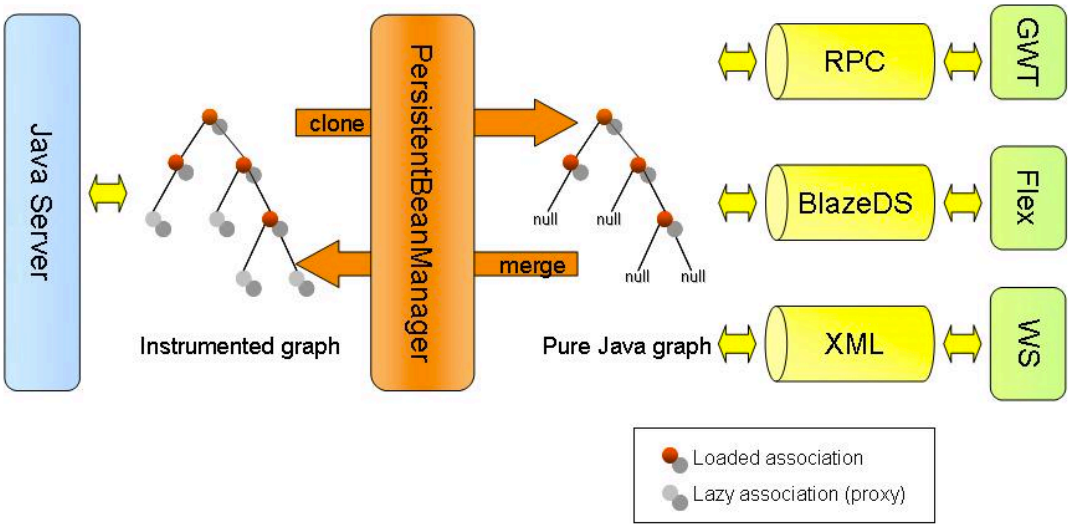
Η έννοια της έγχυσης εξαρτήσεων στον αντικειμενοστραφή προγραμματισμό, είναι ένα σχεδιαστικό πρότυπο σύμφωνα με το οποίο σε ένα κομμάτι λογισμικού παρέχονται όλα τα επιμέρους συστατικά λογισμικού που χρειάζεται για να εκτελεστεί. Οι αλληλοεξαρτήσεις που δημιουργούνται μεταξύ των διαφόρων μερών ενός λογισμικού, γίνονται όλο και πιο πολύπλοκες με την αύξηση της έκτασης του κώδικα και η ανάγκη για υιοθέτηση της παραπάνω μεθόδου είναι επιτακτική. Κατά την έγχυση εξαρτήσεων ένα μέρος του λογισμικού τροφοδοτείται με όλα τα απαραίτητα στοιχεία, όχι κατά την αρχικοποίηση αλλά κατά την εκτέλεση της εφαρμογής.

Σε πιο τεχνικό επίπεδο, το Guice απαλλάσσει τον προγραμματιστή από την ανάγκη να χρησιμοποιεί συνεχώς factories και το new στον κώδικά του. Πολύ συνοπτικά, στον αντικειμενοστραφή προγραμματισμό ένα factory είναι ένα αντικείμενο που χρησιμοποιείται για τη δημιουργία άλλων αντικειμένων. Θα μπορούσε κανείς να πει, ότι είναι μια αφαίρεση της έννοιας του constructor. Κάθε factory αντικείμενο, διαθέτει μία μέθοδο για κάθε αντικείμενο που μπορεί να δημιουργήσει. Αυτές οι μέθοδοι δέχονται παραμέτρους που περιγράφουν το πως δημιουργείται ένα αντικείμενο και επιστρέφουν το δημιουργηθέν αντικείμενο. Τα factory αντικείμενα, χρησιμοποιούνται σε καταστάσεις όπου η διαχείριση ενός αντικειμένου απαιτεί μια πιο πολύπλοκη διαδικασία από την απλή δημιουργία του. Είναι σε θέση να αποφασίσουν τη δημιουργία μιας κλάσης ενός αντικειμένου, να ζητήσουν αντικείμενα από μία ολόκληρη συλλογή ή ακόμα και να κάνουν πολύπλοκες ρυθμίσεις σε κάποιο αντικείμενο. Αυτά τα είδη αντικειμένων, έχουν αποδειχθεί πολύ χρήσιμα, ενώ έχουν αναπτυχθεί αρκετά πρότυπα σχεδιασμού που υποστηρίζουν τη χρήση τους σε πολλές γλώσσες προγραμματισμού. Χαρακτηριστικό είναι ότι με τη τη χρήση τους, η συγγραφή δοκιμών και η επανάχρησή τους για την εξασφάλιση της ορθής λειτουργίας του λογισμικού γίνεται απλούστερη [6].

4.3 Η βιβλιοθήκη Gilead

Έχοντας κάνει την εμφάνισή του αρχικά με το όνομα hibernate4gwt, το Gilead είναι μια βιβλιοθήκη που απλοποιεί τη διαδικασία αποστολής αντικειμένων που έχουν ανακτηθεί από μία βάση δεδομένων παρέχοντας λύσεις σε κοινά προβλήματα που αφορούν το serialization των δεδομέμων. Η βασική ιδέα πίσω από την υλοποίηση του Gilead είναι η μετατροπή των οντοτήτων που έρχονται από μία βάση δεδομένων σε Plain Old Java Objects, αφαιρώντας την επιπρόσθετη πληροφορία και αντικαθιστώντας αυτές με τις αντίστοιχες παραδοσιακές μορφές. Αυτό που κάνει το Gilead ξεχωριστό στο χώρο του serialization, είναι ότι απαλάσσει τον μηχανικό από την επίπονη διαδικασία της μετατροπής των δεδομένων για την μεταφορά τους εκτός

του JVM, από το οποίο και προέρχονται. Βέβαια ένα μειονέκτημά του είναι ότι οι εξαρτήσεις που φέρνει είναι υπερβολικά πολλές σε όγκο, κάτι το οποίο δεν ενδείκνυται για μικρές εφαρμογές όπου υπάρχει ανάγκη για χρήση όσο το δυνατόν λιγότερων εξωτερικών πακέτων [8].



Εικόνα 4.2: Φόρτωση συσχετίσεων με Gilead

4.4 Το σύστημα αντιστοίχισης Dozer

Το Dozer έχει τη δυνατότητα να αντιστοιχεί Java Beans σε Java Beans, αντιγράφοντας αναδρομικά δεδομένα από ένα αντικείμενο σε κάποιο άλλο. Αναφορικά ένα Java Bean είναι ένα αντικείμενο Java, το οποίο είναι εξαρχής σειριοποιήσιμο. Το Dozer υποστηρίζει αντιστοίχιση απλών ιδιοτήτων, πολύπλοκων τύπων, αμφίδρομη αντιστοίχιση, υπονοούμενη-ρητή καθώς και αναδρομική αντιστοίχιση. Δεν περιορίζεται στην αντιστοίχιση μεταξύ ονομάτων ιδιοτήτων, αλλά αυτομάτως αναλαμβάνει και μετατροπές μεταξύ διαφορετικών τύπων αντικειμένων. Τα περισσότερα σενάρια μετατροπής υποστηρίζονται με αφαιρετικό τρόπο, ενώ παράλληλα δίνεται η δυνατότητα για χειρονακτικό καθορισμό ξεχωριστών σεναρίων μέσω XML εγγράφων.

Το σύστημα αντιστοίχησης, μπορεί να χρησιμοποιηθεί οποιαδήποτε στιγμή παρουσιάζεται η ανάγκη για την αντιστοίχιση ενός Java Bean σε ένα άλλο Java Bean διαφορετικού τύπου. Το μεγαλύτερο κομμάτι της αντιστοίχησης πεδίων, μπορεί να γίνει αυτόματα από το Dozer με τη χρήση αντανάκλασης, παρ' όλα αυτά, όπως

αναφέρθηκε και παραπάνω, η περιγραφή καθορισμένων από το χρήστη μετατροπών, μπορεί να γίνει μέσω XML εγγράφων. Λόγω του ότι η αντιστοίχηση γίνεται αμφίδρομα, χρειάζεται να καθοριστεί η σχέση μόνο προς μία κατεύθυνση. Σε περίπτωση που κάποιες ιδιότητες μεταξύ των αντικειμένων συμπίπτουν, ο προγραμματιστής απαλάσσεται και πάλι από την ανάγκη να προβεί σε συγκεκριμένες ρυθμίσεις [5].

4.5 Μεταφορά αντικειμένων μέσω Data Transfer Objects (DTOs)

Ένας από τους πιο εύκολους τρόπους για να αντιμετωπίσουμε τα προβλήματα μεταφοράς αντικειμένων από την πλευρά του διακομιστή στην πλευρά του πέλάτη, είναι να εισάγουμε την έννοια ενός ελαφρού αντικειμένου που να κινείται μεταξύ ενός βαρύ Hibernate αντικείμενου και της αναπαράστασης της πληροφορίας που χρειαζόμαστε να μεταφέρουμε στην πλευρά του πελάτη. Αυτό το αντικείμενο μεταφοράς, θα το ονομάζουμε DTO.

Το DTO είναι ένα συνθησισμένο αντικείμενο της Java, το οποίο περιέχει μόνο απλά πεδία δεδομένων τα οποία μπορούμε να προσπελάσουμε και να έχουμε πρόσβαση στην πλευρά του πελάτη για την απεικόνιση αυτών στην σελίδα της εφαρμογής. Τα Hibernate αντικείμενα, μπορούν έπειτα να κατασκευαστούν σε αντικείμενα μεταφοράς δεδομένων. Τα DTOs θα περιέχουν μόνο τα δεδομένα που προορίζονται για αποθήκευση στη βάση δεδομένων και καμία επιπλέον πληροφορία έχει προσαρτηθεί σε αυτά κατά την ανάκτησή τους μέσω της Hibernate [24].

4.6 Διαχείριση δοσοληψιών μέσω Spring Transactions

Είναι αναγκαίο να γίνει μία σύντομη αναφορά στην έννοια των transactions, πριν ξεκινήσουμε να μιλάμε για Spring Transactions και αυτό διότι θα μας διευκολύνει στην καλύτερη κατανόηση των μηχανισμών που προσφέρει το συγκεκριμένο σύστημα. Έτσι με τον όρο transaction processing, αναφερόμαστε στην διαδικασία επεξεργασίας πληροφορίας που χωρίζεται σε ατομικές, αδιαίρετες λειτουργίες, που ονομάζονται transactions. Κάθε transaction, πρέπει να επιτυγχάνει είτε να αποτυγχάνει ως μονάδα, ενώ είναι μία λειτουργία που δεν μπορεί να παραμείνει σε ενδιάμεση κατάσταση κατά την εκτέλεσή της.

Η διαδικασία του transaction processing είναι σχεδιασμένη με τέτοιο τρόπο ώστε ένα υπολογιστικό σύστημα να παραμένει σε μια γνωστή και συνεπή κατάσταση, μέσω της επιβεβαίωσης ότι οποιεσδήποτε ανεξάρτητες λειτουργίες έχει κληθεί να

φέρει εις πέρας, θα ολοκληρωθεί ή θα ακυρωθεί η εκτέλεσή τους επιτυχώς.

Για παράδειγμα, υποθέστε μια τυπική τραπεζική συναλλαγή κατά την οποία πρέπει να μεταφερθούν 700 δολλάρια από έναν λογαριασμό καταθέσεων σε έναν λογαριασμό ταμιευτηρίου. Από τραπεζική σκοπιά, η συγκεκριμένη συναλλαγή αν και είναι μία αυτοτελής λειτουργία, στην πραγματικότητα συμπεριλαμβάνει τουλάχιστον δύο ξεχωριστές λειτουργίες που πρέπει να εκτελεστούν από το υπολογιστικό σύστημα. Αυτές είναι από τη μία η δέσμευση από τον λογαριασμό καταθέσεων των 700 δολλαρίων και από την άλλη η πίστωση του ποσού αυτού στον λογαριασμό που προορίζεται να γίνει η μεταφορά. Σε περίπτωση που η λειτουργία της δέσμευσης εκτελεστεί επιτυχώς, αλλά κατά τη διαδικασία της πίστωσης προκύψει κάποιο σφάλμα (ή αντιστρόφως), στο τέλος της ημέρας θα παρουσιαστούν λάθος υπόλοιπα στα βιβλία της τράπεζας. Θα πρέπει λοιπόν, να υπάρξει ένας μηχανισμός κατά τον οποίο θα εξασφαλίζεται η επιτυχία ή αποτυχία και των δύο λειτουργιών, έτσι ώστε να μην υπάρξει ποτέ ασυνέπεια στο σύνολο της βάσης δεδομένων μιας τράπεζας. Η διαδικασία του transaction processing, είναι αυτή που παρέχει όλους τους απαραίτητους μηχανισμούς για την αποφυγή τέτοιου είδους προβλημάτων.

Ο μηχανισμός του transaction processing, επιτρέπει σε πολλές ατομικές λειτουργίες να ενοποιηθούν με αυτόματο τρόπο σε ένα μοναδιαίο, ατομικό transaction. Το σύστημα αυτό εξασφαλίζει ότι θα ολοκληρώσουν την εκτέλεσή τους χωρίς την παρουσίαση κάποιου σφάλματος όλες οι λειτουργίες ή καμία. Εάν κάποιες από αυτές ολοκληρώσουν την εκτέλεσή τους επιτυχώς ενώ καποιες άλλες παρουσιάσουν σφάλματα, ο μηχανισμός του transaction processing επαναφέρει όλες τις λειτουργίες (ακόμα και αυτές που έχουν εκτελεστεί επιτυχώς), διαγράφωντας έτσι όλα τα ίχνη του transaction και επαναφέροντας το σύστημα στη συνεπή και γνωστή κατάσταση που βρισκόταν, τη στιγμή ακριβώς πριν ξεκινήσει η επεξεργασία του transaction. Σε περίπτωση που όλες οι λειτουργίες του transaction ολοκληρωθούν επιτυχώς, το transaction υποβάλλεται από το σύστημα, και μονιμοποιούνται όλες οι αλλαγές στη βάση δεδομένων, με το transaction να μην μπορεί έπειτα από αυτό να επανέλθει στην αρχική του κατάσταση.

Το transaction processing προστατεύει το σύστημα από σφάλματα σε επίπεδο υλικού και λογισμικού, που μπορεί να προκαλέσουν ένα transaction να μην ολοκληρωθεί πλήρως και να αφήσουν το σύστημα σε μία άγνωστη και ασυνεπή κατάσταση. Εάν το υπολογιστικό σύστημα σπάσειστα μέσα εκτέλεσης ενός transaction, το σύστημα transaction processing εγγυάται ότι θα ακυρωθούν όλες οι λειτουργίες των μη υποβληθέντων (π.χ. όχι πλήρως επεξεργασμένων) transactions.

Τα transactions επεξεργάζονται υπό αυστηρή χρονολογική σειρά. Σε περίπτωση που το transaction ν+1 τείνει να επεμβαίνει στο ίδιο κομμάτι μιας βάσης δεδομένων με ένα transaction ν, τότε το transaction ν+1 δεν πρόκειται να ξεκινήσει μέχρι να

υποβληθεί το transaction ν. Πριν υποβληθεί οποιοδήποτε transaction, πρέπει επίσης να υποβληθούν όλα τα υπόλοιπα transactions που επηρεάζουν το ίδιο κομμάτι του συστήματος, έτσι ώστε να μην υπάρξουν 'τρύπες' στην ακολουθία των transactions που προηγούνται [23].

Το πλαίσιο ανάπτυξης λογισμικού Spring, προσφέρει πλήρη υποστήριξη για τα transactions, γεγονός που το κάνει έναν από τους πιο βασικούς λόγους χρήσης του. Το πλαίσιο Spring παρέχει ένα συνεπές αφαιρετικό μηχανισμό για τη διαχείριση των abstractions και παρουσιάζει τα ακόλουθα πλεονεκτήματα [3].

1. Παρέχει ένα συνεπές προγραμματιστικό μοντέλο μεταξύ των διάφορων transaction APIs όπως το JTA, JDBC, Hibernate, JPA και JDO.
2. Υποστηρίζει δηλωτική διαχείριση των transactions.
3. Παρέχει μια απλούστερη διεπαφή για την προγραμματιστική διαχείριση των transactions σχετικά με άλλες πολύπλοκες υλοποιήσεις όπως το JTA.

4.7 Η βιβλιοθήκη αντιστοίχισης Hibernate

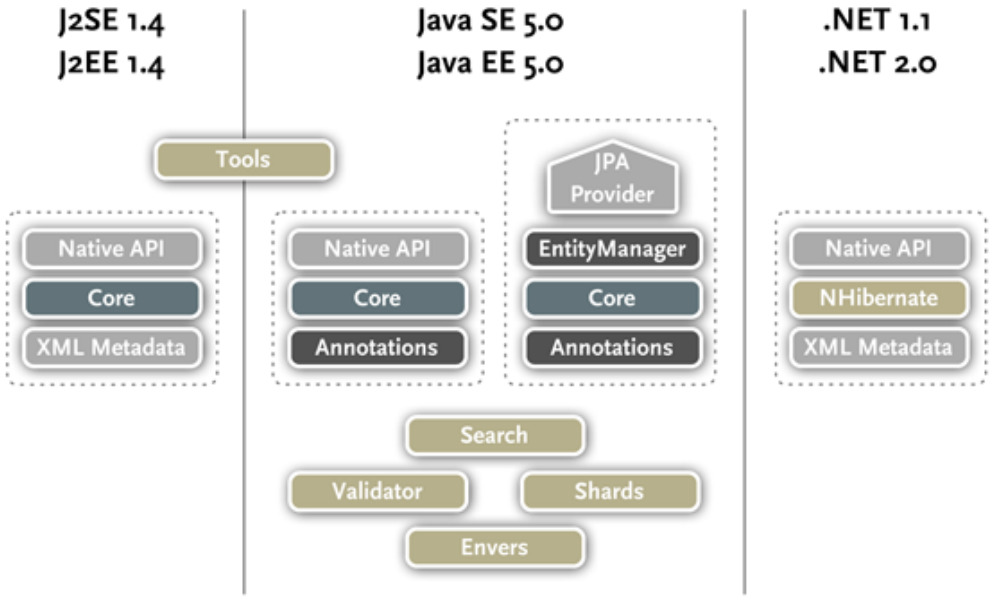
Η Hibernate είναι μια βιβλιοθήκη για την αντιστοίχη συσχετίσεων μεταξύ αντικειμένων για τις γλώσσες JAVA και .NET. Στην πραγματικότητα παρέχει ένα προγραμματιστικό πλαίσιο για την αντιστοίχηση αντικειμενοστραφών μοντέλων σε μια παραδοσιακή σχεσιακή βάση δεδομένων. Παρέχει στον προγραμματιστή μια υψηλού επιπέδου διεπαφή που του επιτρέπει να αποκτά πρόσβαση στη βάση με μεθόδους διαχείρισης κλασσικών αντικειμένων.

Πιο συγκεκριμένα, κύριο χαρακτηριστικό της Hibernate είναι η αντιστοίχηση Java κλάσεων σε πίνακες βάσεων δεδομένων καθώς και από Java τύπους δεδομένων σε SQL τύπους δεδομένων. Παράλληλα παρέχει διεπαφές για ερωτήματα και ανάκτηση πάνω σε δεδομένα με τη Hibernate να αναλαμβάνει μέσω των διεπαφών αυτών τη δημιουργία των κατάλληλων SQL κλήσεων, απαλάσσοντας έτσι τον μηχανικό από την χειρονακτική διαχείριση των αποτελεσμάτων καθώς και την μετατροπή της μορφής των αντικειμένων. Με αυτόν τον τρόπο η Hibernate καταφέρνει να διατηρεί την εκάστοτε εφαρμογή άμεσα μεταφέρσιμη σε όλες τις υποστηριζόμενες SQL βάσεις δεδομένων με ελάχιστο κόστος στις επιδόσεις της.

Ο τρόπος με τον οποίο η Hibernate επιτυγχάνει την αντιστοίχηση Java κλάσεων σε πίνακες βάσεων δεδομένων είναι είτε με την κατάλληλη ρύθμιση ενός XML αρχείου είτε με τη χρήση Java Annotations. Χρησιμοποιώντας ένα XML αρχείο, η Hibernate μπορεί να δημιουργήσει πρότυπο πηγαίο κώδικα για τις κλάσεις που προορίζονται για αντιστοίχηση. Το σχήμα της εκάστοτε βάσης δεδομένων διαμορφώνεται από

τη Hibernate είτε μέσω της δομής του XML αρχείου είτε μέσω των Java Annotations.

Για την διευθέτηση σχέσεων ένα-προς-πολλά και πολλά-προς-πολλά μεταξύ των κλάσεων παρέχεται αντίστοιχη υποστήριξη. Παράλληλα με την διαχείρηση συσχετίσεων μεταξύ αντικειμένων, η Hibernate μπορεί επίσης να διαχειριστεί αντανακλαστικές συσχετίσεις κατά τις οποίες ένα αντικείμενο διατηρεί μία ένα-προς-πολλά συσχέτιση με άλλα στιγμιότυπα του ίδιου τύπου [10].



Εικόνα 4.3: Hibernate περιβάλλοντα και υποστήριξη

4.8 Δοκιμές - JUnit, GWTTestCase

Το JUnit, είναι ένα πλαίσιο εφαρμογής δοκιμών σε μονάδες ενός λογισμικού για την γλώσσα προγραμματισμού Java. Το JUnit έχει παίξει καθοριστικό ρόλο στην οδηγούμενη με δοκιμές ανάπτυξη λογισμικού και διασυνδέεται ως ένα JAR κατά τη διάρκεια της μεταγλώττισης.

Παρ' όλο που το GWT είναι σχεδόν εξολοκλήρου γραμμένο σε Java και το πλαίσιο δοκιμών JUnit φαίνεται ιδανικό για χρήση, το GWT εμπεριέχει μία ειδική TestCase υποκλάση, την κλάση GWTTestCase, η οποία μπορεί να εφαρμόσει δοκιμές σε κώδικα που απαιτεί Javascript κατά την εκτέλεσή της [22].

Η μέθοδος που δίνεται στην εικόνα 4.4, δέχεται τα αλφαριθμητικά 'foo' και 'bar' σε οποιαδήποτε μορφή των γραμμάτων καθώς και το 'Baz' αλλά αυτή τη φορά

συγκρίνοντας πεζά και κεφαλαία. Στην εικόνα 4.5 φαίνεται η υλοποίηση της

```
/**
 * A sample routine that runs in the module that you want to run a
 * unit test on.  Accepts the strings "foo" and "bar" in any case,
 * and the string "Baz" as an exact match.  All other strings fail.
 *
 * @param input A string to validate
 * @return true if the string passes validation.
 */
public boolean exampleValidator (String input) {
  if (input.toLowerCase().equals("foo")){
    return true;
  } else if (input.toLowerCase().equals("bar")) {
    return true;
  } else if (input.equals("Baz")) {
    return true;
  }
  return false;
}
```

Εικόνα 4.4: Μέθοδος exampleValidator

μεθόδου testExampleValidator, όπου εσωτερικά της καλείται η μέθοδος example-Validator δεχόμενη ως ορίσματα διάφορα αλφαριθμητικά για να επιβεβαιωθεί η ορθή λειτουργία της [?].

```
import com.google.gwt.junit.client.GWTTestCase;
import com.example.foo.client.Foo;

public class FooTest extends GWTTestCase {

  /**
   * Specifies a module to use when running this test case. The returned
   * module must cause the source for this class to be included.
   *
   * @see com.google.gwt.junit.client.GWTTestCase#getModuleName()
   */
  public String getModuleName() {
    return "com.example.foo.Foo";
  }

  /**
   * Test the method 'exampleValidator()' in module 'Foo'
   */
  public void testExampleValidator() {
    Foo myModule = new Foo();

    assertTrue(myModule.exampleValidator("foo"));
    assertTrue(myModule.exampleValidator("Foo"));
    assertTrue(myModule.exampleValidator("FOO"));
    assertTrue(myModule.exampleValidator("fOo"));
    assertTrue(myModule.exampleValidator("bar"));
    assertTrue(myModule.exampleValidator("Baz"));

    assertFalse(myModule.exampleValidator("fooo"));
    assertFalse(myModule.exampleValidator("oo"));
    assertFalse(myModule.exampleValidator("baz"));
  }
```

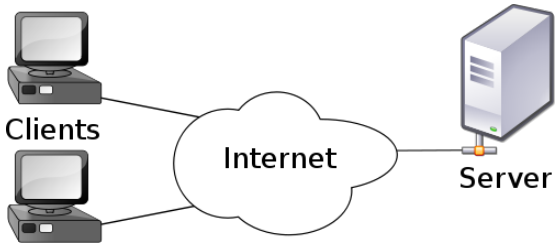
Εικόνα 4.5: Υλοποίηση δοκιμής testExampleValidator

Κεφάλαιο 5

Σχεδιασμός νέων υπηρεσιών στα πλαίσια του WiseUI

5.1 Αρχιτεκτονική

Η αρχιτεκτονική πολύπλοκων διαδικτυακών εφαρμογών όπως το WiseUI, τείνει να γίνεται όλο και πιο πολύπλοκη παράλληλα με την συνεχή αύξηση της λειτουργικότητας που προσφέρουν. Μια πολύπλοκη όμως αρχιτεκτονική δεν είναι ιδανική για τη συνεχή εξέλιξη μιας εφαρμογής, καθώς με αυτόν τον τρόπο η προσθήκη νέων χαρακτηριστικών θα γινεται όλο και πιο επίπονη και χρονοβόρα. Έτσι απαιτείται προσεκτικός σχεδιασμός, με τον οποίο όλα τα επιμέρους στοιχεία που απαρτίζουν την εφαρμογή θα συνδυάζουν την λειτουργικότητά τους με μία όχι ισχυρά συνδεδεμένη συνοχή μεταξύ τους.



Εικόνα 5.1: Αρχιτεκτονική μοντέλου πελάτη-διακομιστή

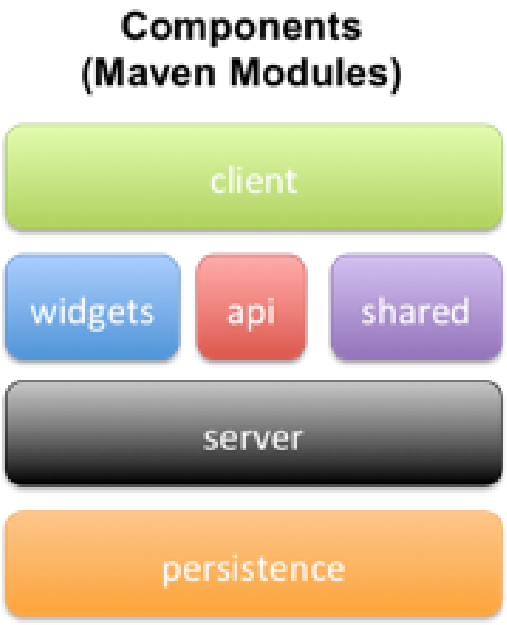
Όπως και το σύνολο των διαδικτυακών εφαρμογών, η δομή του WiseUI μπορεί να διαχωριστεί με εντελώς αφαιρετικό και απλοϊκό τρόπο σε δύο λογικά κομμάτια που ονομάζονται επίπεδα. Θα μπορούσε λοιπόν, κάποιος να δει την εφαρμογή ως μια αρχιτεκτονική δύο επιπέδων, όπου στην περίπτωση του WiseUI από τη μία

πλευρά βρίσκεται ένας 'έξυπνος' πελάτης και από την άλλη ένας 'αδαής' εξυπηρετητής. Αν και παρακάτω, θα αναφερθούμε πολύ πιο αναλυτικά στην αρχιτεκτονική της εφαρμογής που αναπτύξαμε, σε γενικές γραμμές ο πελάτης είναι αυτός που διαχειρίζεται το παρουσιαστικό της εφαρμογής μαζί με την επιχειρισιακή λογική, ενώ ο διακομιστής είναι υπεύθυνος για την επικοινωνία με τις πειραματικες πλατφόρμες και την διαχείριση της βάσης δεδομένων. Αυτή η πλήρως αποσυνδεδεμένη αρχιτεκτονική του WiseUI, όπου η απεικόνιση και η βάση δεδομένων έχουν έμμεση αλληλοεξάρτηση, παίζει καταλυτικό ρόλο στην δυνατότητα επεκταστιμότητας της εφαρμογής [25].

Το WiseUI, είναι σχεδιασμένο από την αρχή ακολουθώντας τρεις βασικούς άξονες κατά την ανάπτυξη. Αρχικά, είναι η προσπάθεια να διατηρούμε μια πλήρως αποσυνδεμένη λογική μεταξύ των κομματιών που απαρτίζουν την εφαρμογή. Δεύτερον, κάναμε μεγάλες προσπάθειες να διατηρήσουμε έναν διακομιστή ο οποίος δεν θα είναι γνώστης της κατάστασης της εφαρμογής στην πλευρά του πελάτη. Στα πλαίσια αυτής της προσπάθειας παρουσιάστηκε το πρόβλημα αντιμετώπισης των ενεργών συνόδων ενός χρήστη και το πως ο διακομιστής θα καταλαβαίνει για το αν ο χρήστης έχει εξουσιοδότηση για αποστολή νέων αιτήσεων. Τέλος, σύμφωνα με τον τρίτο σχεδιαστικό άξονα, προσπαθήσαμε να διατηρήσουμε τον ασύγχρονο χαρακτήρα σε κάθε επικοινωνία που υπάρχει μεταξύ πελάτη και εξυπηρετητή. Όλα τα παραπάνω, είναι τρεις βασικοί πυλώνες σχεδιασμού για τους οποίους, οι ίδιοι οι μηχανικοί της Google, κάνουν ιδιαίτερη αναφορά στα βοηθητικά κείμενά τους σχετικά με την αρχιτεκτονική των εφαρμογών που αναπτύσσονται με το GWT.

Στην εικόνα 5.2 παρουσιάζεται η συνολική δομή όλων των στοιχείων που απαρτίζουν το WiseUI. Στο πιο χαμηλό επίπεδο, βρίσκεται το στοιχείο όπου αποθηκεύονται μόνιμα όλα τα δεδομένα που είναι απαραίτητα για μελλοντική χρήση. Αμέσως πιο πάνω, βρίσκεται ο διακομιστής, ο οποίος δέχεται συνεχώς αιτήσεις από τα πιο υψηλά επίπεδα, τις επεξεργάζεται και απαντά αναλόγως στον πελάτη. Στο τρίτο επίπεδο βρίσκονται τρία διαφορετικά στοιχεία που δεν έχουνε άμεση εξάρτηση μεταξύ τους και αυτά είναι τα γραφικά που απαρτίζουν την εφαρμογή και τροφοδοτούνται συνεχώς δεδομένα από τον διακομιστή, το περιβάλλον διεπαφής με τον διακομιστή καθώς και τα μοντέλα των διαμοιραζόμενων αντικειμένων μεταξύ τους. Στο πιο υψηλό και τελευταίο επίπεδο, βρίσκεται το στοιχείο του πελάτη το οποίο είναι και το πιο πλούσιο από πλευράς λειτουργικότητας και επιμέρους αρχιτεκτονικής στο εσωτερικό του.

Έχοντας παρουσιάσει συνολικά την αρχιτεκτονική του WiseUI, μπορούμε πλέον να προχωρήσουμε στην λεπτομερή περιγραφή των αρχιτεκτονικών που ακουλουθούν ξεχωριστά πελάτης και διακομιστής.



Εικόνα 5.2: Αρχιτεκτονική WiseUI

5.1.1 Πελάτης

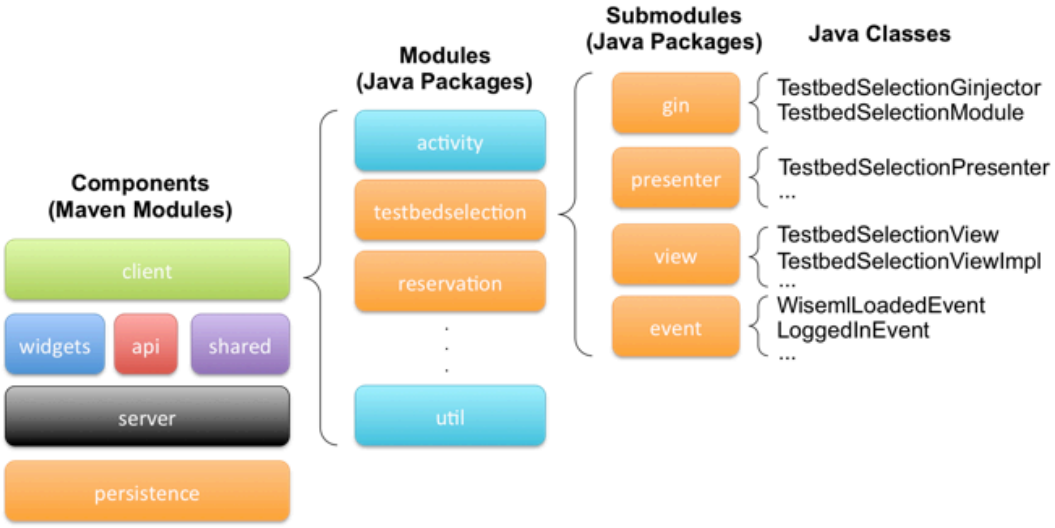
Το WiseUI, είναι μια πλούσια διαδικτυακή εφαρμογή με το στοιχείο του πελάτη να προσφέρει την μεγαλύτερη λειτουργικότητα, κάτι που δημιούργησε την ανάγκη επιλογής σχεδιαστικών προτύπων που θα ευνοούν την ευέλικτη επέκταση της εφαρμογής. Ο πελάτης χωρίζεται σε δύο κομμάτια. Το ένα είναι το κομμάτι της πλοήγησης και το άλλο του περιεχομένου. Σε αυτό του περιεχομένου, είναι τοποθετημένη λειτουργικότητα όπως η επιλογή των πειραματικών υποδομών και η διαχείριση κρατήσεων. Κάθε ένα από τα κομμάτια έχει το δικό της στοιχείο διαχείρισης, το οποίο ονομάζεται δραστηριότητα. Κάθε δρασηριότητα διαχειρίζεται τη δική της περιοχή στο γραφικό περιβάλλον της εφαρμογής και στην εικόνα 5.3 φαίνεται το σημείο όπου γινεται η αρχικοποίηση μιας τέτοιας δραστηριότητας.

Για την καλύτερη κατανόηση της δομής του πελάτη, κάθε κομμάτι του αποτελείται από τέσσερα επιμέρους στοιχεία. Αυτά είναι οι δραστηριότητες, οι παρουσιαστές, οι όψεις και τα στοιχεία έγχυσης εξαρτήσεων όπως παρουσιάζεται στην εικόνα 5.4 [29].

Ακολουθώντας λοιπόν οδηγίες σχετικά με τις καλύτερες σχεδιαστικές πρακτικές,

```
public void onModuleLoad() {
    final WiseUiView appWidget = injector.getAppWidget();

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

    // 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();
}
```

Εικόνα 5.3: onModuleLoad



Εικόνα 5.4: Αρχιτεκτονική των επιμέρους στοιχείων του πελάτη

το πρότυπο Μοντέλο-Όψη-Παρουσιαστής (ΜΟΠ) αποτελεί τον βασικό άξονα σχεδιασμού και οργάνωσης του κώδικα στην πλευρά του πελάτη (Εικόνα 5.8). Το πρότυπο αυτό είναι παράγωγο του Μοντέλου-Όψη-Ελεγκτή, το οποίο επίσης χρησιμοποιείται για την κατασκευή περιβαλλόντων αλληλεπίδρασης. Στο ΜΟΠ, ο παρουσιαστής παίζει το ρόλο του μεσάζοντα μεταξύ όψης και μοντέλου, η όψη είναι υπεύθυνη για την διαχείριση όλων των γεγονότων που λαμβάνουν χώρα στο γραφικό περιβάλλον (π.χ. το πάτημα ενός πλήκτρου από το πληκτρολόγιο του χρήστη) και το μοντέλο αποτελεί την αυστηρή παρουσίαση των οντοτήτων που παίρνουν μέρος στην εφαρμογή. Είναι ένα πρότυπο που διευκολύνει την δημιουργία αυτοματοποιημένων δοκιμών και διαμοιράζει με ευελιξία τη λειτουργι-

κότητα σε διάφορα κομμάτια του λογισμικού. Σε πιο τεχνικό επίπεδο, το μοντέλο είναι έναν περιβάλλον όπου ορίζονται όλα τα δεδομένα τα οποία πρόκειται να απεικονισθούν ή σε οποιαδήποτε περίπτωση να επιδράσουν στο περιβάλλον αλληλεπίδρασης του χρήστη (Εικόνα 5.5). Η όψη είναι επίσης ένα περιβάλλον που

```
public class Node implements Serializable {
    private String id;

    public Node() {}

    public String getId() {
        return id;
    }

    public void setId(final String id) {
       this.id = id;
    }
}
```

Εικόνα 5.5: Παράδειγμα Μοντέλου

αναλαμβάνει την απεικόνιση των δεδομένων ή αλλιώς των μοντέλων και δρομολογεί όλες τις εντολές του χρήστη (γεγονότα), προς τον παρουσιαστεί για να επεξεργαστεί με τη σειρά του τα δεδομένα (Εικόνα 5.6). Ο παρουσιαστής πραγματοποιεί ενέργειες όχι μόνο πάνω στα μοντέλα αλλά και στην ίδια την όψη. Ο ίδιος παίρνει τα δεδομένα από τα μοντέλα, τα αποθηκεύει και τα μορφοποιεί για απεικόνιση στην όψη (Εικόνα 5.7).

```
package eu.wisebed.wiseui.client.reservation.view;

import com.google.gwt.user.client.ui.IsWidget;
import com.google.inject.ImplementedBy;

@ImplementedBy(NewReservationViewImpl.class)
public interface NewReservationView extends IsWidget {

    void setPresenter(Presenter presenter);


    public interface Presenter {}
}
```

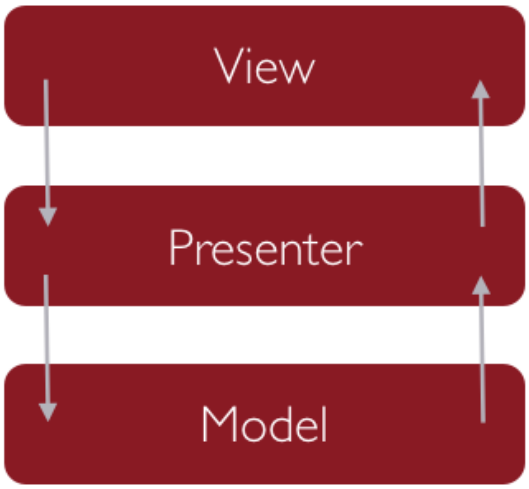
Εικόνα 5.6: Παράδειγμα Όψης

5.1.2 Διακομιστής

Κατά την ανάπτυξη του WiseUI προσπαθήσαμε να διατηρήσουμε έναν διακομιστή του οποίου η πολυπλοκότητα λειτουργίας, θα είναι όσο το δυνατόν πιο μικρή γίνεται. Ο διακομιστής δεν έχει επίγνωση της κατάστασης στην οποία βρίσκεται

```java
package eu.wisebed.wiseui.client.reservation.presenter;

import com.google.inject.Inject;

import eu.wisebed.wiseui.client.reservation.view.NewReservationView;
import eu.wisebed.wiseui.client.reservation.view.NewReservationView.Presenter;

public class NewReservationPresenter implements Presenter{

    private final NewReservationView view;

    @Inject
    public NewReservationPresenter(final NewReservationView view){
        this.view = view;
    }
}
```

Εικόνα 5.7: Παράδειγμα Παρουσιαστή

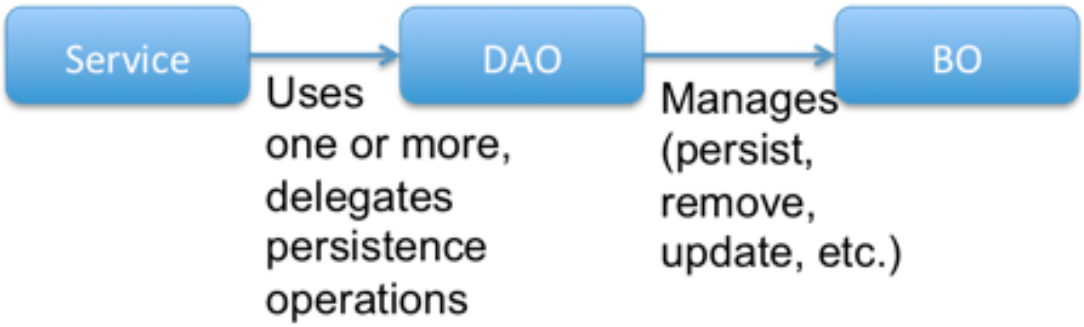


Εικόνα 5.8: Μοντέλο-Όψη-Παρουσιαστής

ο πελάτης και το μόνο για το οποίο είναι υπεύθυνος είναι η λήψη αιτήσεων σε συνδυασμό με δεδομένα και η προώθηση αυτών στις διεπαφές των πειραματικών υποδομών. Το πακέτο λογισμικού του διακομιστή, αποτελείται από ένα σύνολο κλάσεων οι οποίες διαχωρίζονται λογικά ανάλογα με τον τύπο των αιτήσεων που εξυπηρετούν. Υπάρχει για παράδειγμα, η κλάση SNAAServiceImpl που είναι υπεύθυνη για την ταυτοποίηση των στοιχείων ενός χρήστη και η ReservationServiceImpl που δέχεται αιτήσεις σχετικές με κρατήσεις στις πειραματικές πλατφόρμες.

Αυτές οι κλάσεις, μετά την επιτυχή επικοινωνία με τις πειραματικές πλατφόρμες, υπάρχει περίπτωση να χρειαστεί να αποθηκεύσουν κάποια πληροφορία στη βάση δεδομένων της εφαρμογής. Έτσι στον διακομιστή, υλοποιείται ένα ακόμα πακέτο, το οποίο εξυπηρετεί όλες τις αιτήσεις που έχουν να κάνουν με αποθήκευση δεδομέ-

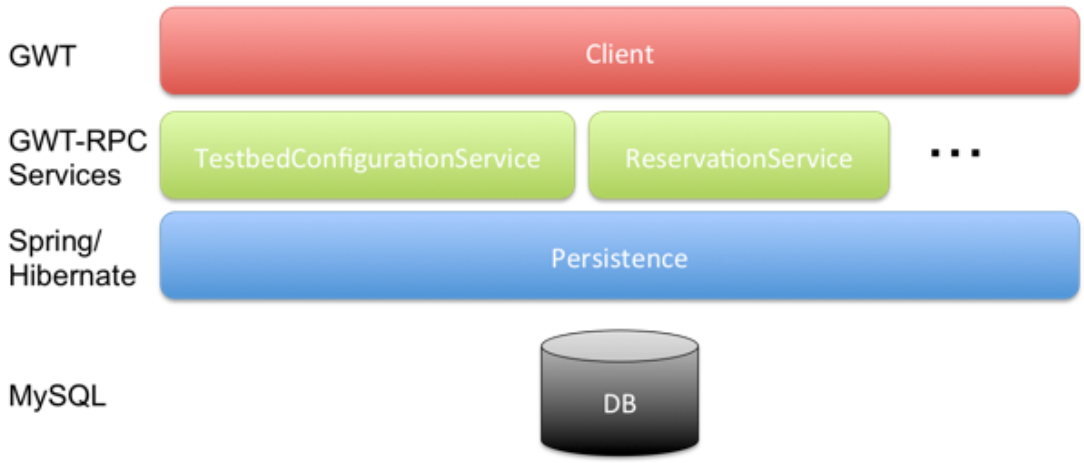
νων. Το πακέτο αυτό αποτελεί ένα στρώμα της αρχιτεκτονικής του WiseUI, το οποίο αλλιλεπιδρά άμεσα με τις υπηρεσίες απομακρυσμένης κλήσης και το μηχανισμό διαχείρισης της βάσης δεδομένων. Πιο αναλυτικά, αυτό το πακέτο λειτουργεί με τρεις διαφορετικούς τύπους κλάσεων. Ο πρώτος τύπος ειναι τα επιχειρησιακά αντικείμενα (Business Objects ή BOs), τα οποία είναι ατόφια αντικείμενα Java και ορίζουν τη δομή που η Hibernate καλείται να αποθηκεύσει στη βάση δεδομένων. Ο δεύτερος τύπος είναι αντικείμενα δεδομένων πρόσβασης (Data Access Objects ή DAOs) και ως δομές διαχειρίζονται τα επιχειρησιακά αντικείμενα, όπως για παράδειγμα η αποθήκευση και εντοπισμός αυτών. Ο τρίτος και τελευταίος τύπος είναι οι υπηρεσίες, που είναι στην πραγματικότητα υπηρεσίες για τη διαχείριση των transactions και χρησιμοποιείται από το πακέτο του πελάτη (Εικόνα 5.9).



Εικόνα 5.9: Τύπος κλάσεων του persistence layer

Τα αντκείμενα δεδομένων πρόσβασης διαμοιράζονται μεταξύ πελάτη και εξυπηρετητή και βρίσκονται στο αντίστοιχο σε ονομασία πακέτο (shared) ενώ μπορούν να χρησιμοποιηθούν στην πλευρά του πελάτη. Τα επιχειρησιακά αντικείμενα είναι αυτά που εμπεριέχουν τους σχολιασμούς της Hibernate. Στο εσωτερικό μηχανισμό της, η Hibernate λειτουργεί αποκλειστικά με τέτοια αντικείμενα ενώ στο περιβάλλον προγραμματιστικής διεπαφής, δέχεται και επιστρέφει DTOs. Με αυτόν τον τρόπο, τα διάφορα πακέτα που απαρτίζουν το WiseUI είναι πλήρως αποσυνδεδεμένα μεταξύ τους (Εικόνα 5.10).

Όπως αναφέρθηκε και παραπάνω, στο WiseUI επιλέξαμε την χρήση Spring Transactions. Σε πιο πρακτικό επίπεδο, χωρίς αυτό το μηχανισμό, ο μηχανικός θα έπρεπε για κάθε transaction να χρησιμοποιεί τη HibernateUtil για να ανοίγει και να κλείνει τις συνόδους αλλιλεπίδρασης με τη βάση 'χειρονακτικά'. Αυτό σημαίνει ότι για κάθε transaction, θα έπρεπε να εξασφαλίσουμε το κλείσιμο της συνόδου, χρησιμοποιώντας finally-blocks στον κώδικά μας, μια τακτική που συνήθως εμφανίζεται αρκετά επιρρεπής σε σφάλματα. Το Spring, παρέχει αυτό το μηχανισμό του



Εικόνα 5.10: Συνολική αρχιτεκτονική του WiseUI

οποίου η χρήση του βασίζεται σε σχολιασμούς με τη λέξη κλειδί @Transactional. Εσωτερικά και με πλήρη αδιαφάνεια προς τον μηχανικό, αυτό πραγματοποιείται με aspect oriented programming. Στην εικόνα 5.11 φαίνεται ένα παράδειγμα μιας μόνο για ανάγνωση transaction, που υλοποιείται σαν μέθοδος στην κλάση PersistenceService. Αυτό που κάνει ο σχολιασμός @Transactional, είναι ότι ανοίγει μία νέα σύνοδο όταν καλείται η μέθοδος και την κλείνει όταν ολοκληρωθεί η εκτέλεσή της. Αυτό απαλάσσει τον προγραμματιστή από τη χειρονακτική διαχείριση των transactions ενώ παράλληλα μειώνεται ο κίνδυνος επικαλυπτόμενες συνόδους και γενικότερα σφάλματα [30].

```
@Override
@Transactional(readOnly = true)
public TestbedConfiguration loadTestbedConfiguration(final Integer id) {
    Preconditions.notNullArgument(id, "Argument 'id' is null!");

    LOGGER.info("loadTestbedConfiguration( " + id + " )");

    final TestbedConfigurationBo bo = testbedConfigurationDao.findById(id);
    Preconditions.notNull(bo, "TestbedConfiguration with id '"
            + id
            + "' does not exist!");
    return dozerBeanMapper.map(bo, TestbedConfiguration.class);
}
```

Εικόνα 5.11: Παράδειγμα μηχανισμού Spring Transaction

Κεφάλαιο 6

Εκτέλεση διαδικτυακής εφαρμογής

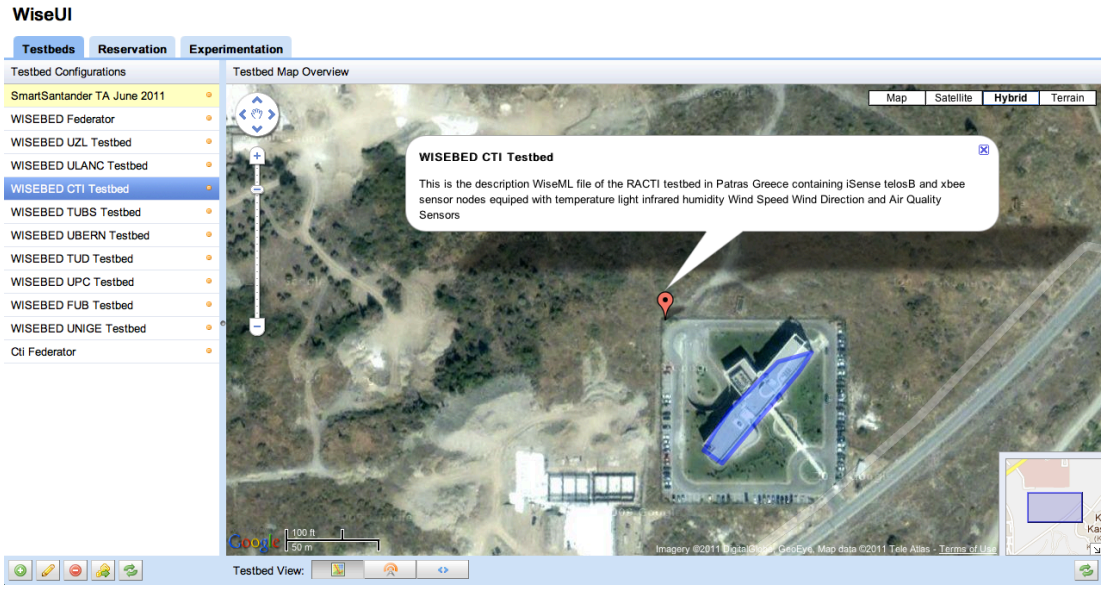
Έχοντας παρουσιάσει όλες τις τεχνικές λεπτομέρειες από τις οποίες απαρτίζεται το WiseUI, στο κεφάλαιο αυτό θα παρουσιάσουμε την τελική εφαρμογή μετά το πέρας της υλοποίησής της.

6.1 Σενάριο εκτέλεσης

Ο πιο ιδανικός τρόπος για να δείξουμε τις υπηρεσίες που προσφέρει η εφαρμογή, είναι να παρουσιάσουμε βήμα προς βήμα ένα κλασικό σενάριο εκτέλεσης. Κατά τη διάρκεια εκτέλεσης του σεναρίου, θα πραγματοποιήσουμε αρχικά την ταυτοποίηση των στοιχείων ενός χρήστη μαζί με την εξουσιοδότησή του για δέσμευση και χρήση των πόρων μιας πειραματικής υποδομής. Έπειτα, αφού επιλέξουμε ένα υποσύνολο των διαθέσιμων πόρων, θα δημιουργήσουμε μία νέα κράτηση για ένα χρονικό διάστημα που έχει προκαθοριστεί από το χρήστη. Φθάνοντας στο τελευταίο στάδιο εκτέλεσης του πειράματος, θα μεταβούμε στο κομμάτι της εκτέλεσης του πειράματος στους κόμβους. Εκεί θα παρουσιάσουμε την εκκίνηση, την 'ζωντανή' λήψη της εξόδου που παράγουν οι αισθητήρες κατά την εκτέλεση και τέλος την λήψη σε μορφή WiseML της συνολικής εξόδου του πειράματος.

Πριν προχωρήσουμε στην ανάλυση του σεναρίου, θα περιγράψουμε αρχικά την δομή του περιβάλλοντος της εφαρμογής καθώς και το ρόλο του κάθε πλαισίου ξεχωριστά. Όπως φαίνεται και στην εικόνα 6.1, το WiseUI απαρτίζεται από τρία βασικά μέρη καθένα από τα οποία αντιπροσωπεύεται από μία καρτέλα στην κορυφή της εφαρμογής, παρουσιάζοντας εδώ το πρώτο μέρος στο οποίο γίνεται η επισκόπηση των διαθέσιμων πειραματικών υποδομών. Αριστερά, βρίσκεται η

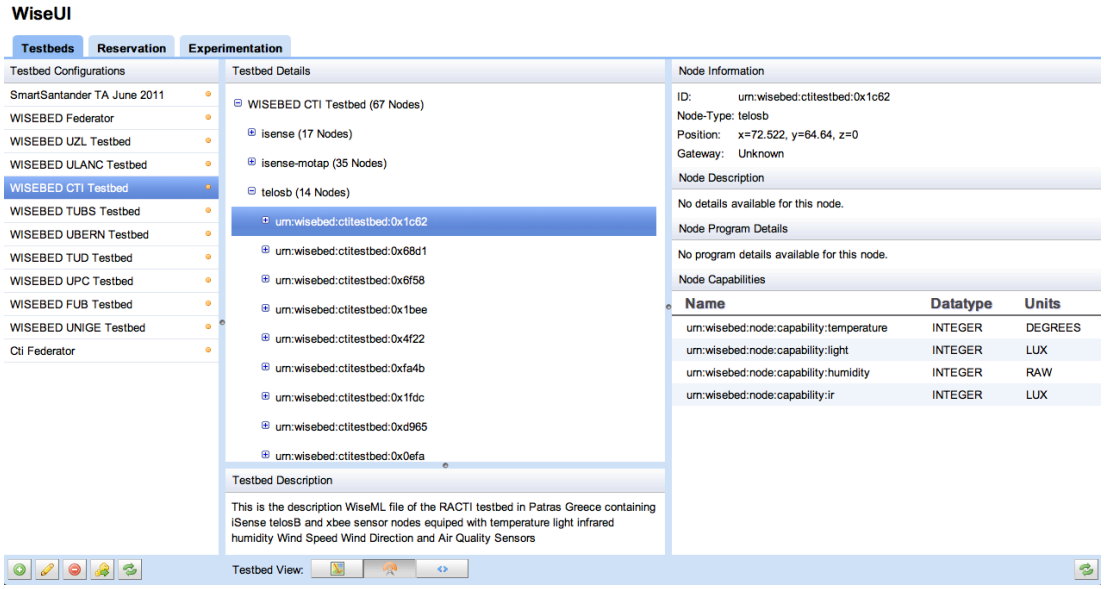
λίστα με τις υπάρχουσες πειραματικές υποδομές, μέσω της οποίας ο χρήστης μπορεί να επιλέξει μία από αυτές και να δει λεπτομέρειες όσον αφορά τους αισθητήρες που βρίσκονται εγκατεστημένοι στην συγκεκριμένη υποδομή. Τα κουμπιά που βρίσκονται ακριβώς κάτω από τη λίστα, προσφέρουν λειτουργίες όπως η προσθήκη μιας νέας πειραματικής υποδομής, την πλήρη διαγραφή ή την αλλαγή των παραμέτρων μιας ήδη υπάρχουσας, την ταυτοποίηση και απόκτηση πρόσβασης για εκμετάλλευση σε επόμενο στάδιο των πειραματικών πόρων καθώς και την ανανέωση της λίστας αυτής. Το κυρίως και μεγαλύτερο πλαίσιο της εφαρμογής, 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.



Εικόνα 6.1: Χάρτης πειραματικής υποδομής CTI

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 εικόνα 6.2, 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.

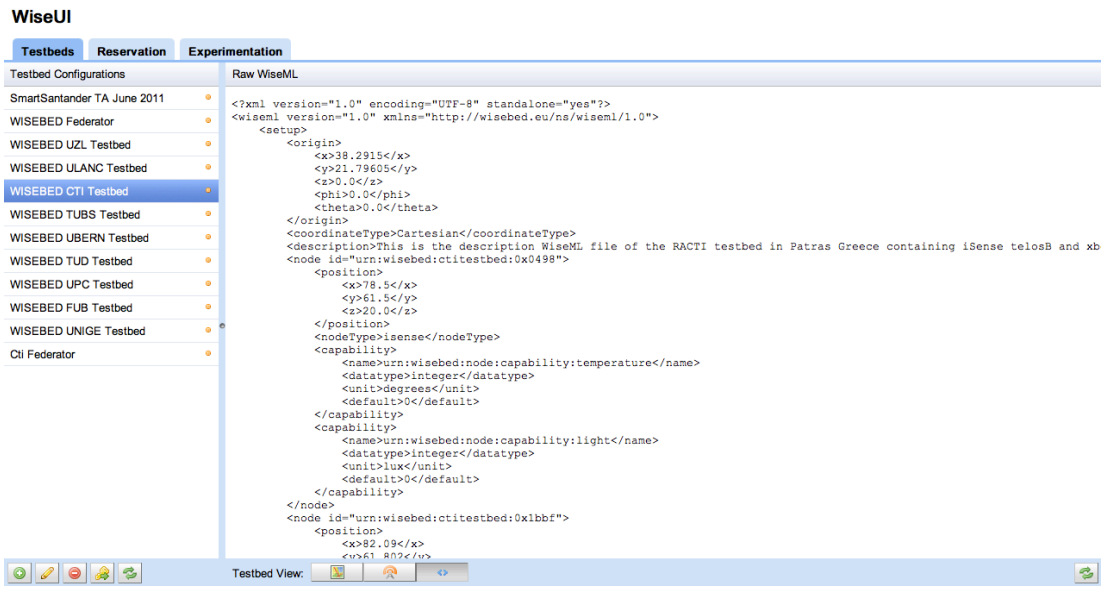


Εικόνα 6.2: Λεπτομέρειες πειραματικής υποδομής

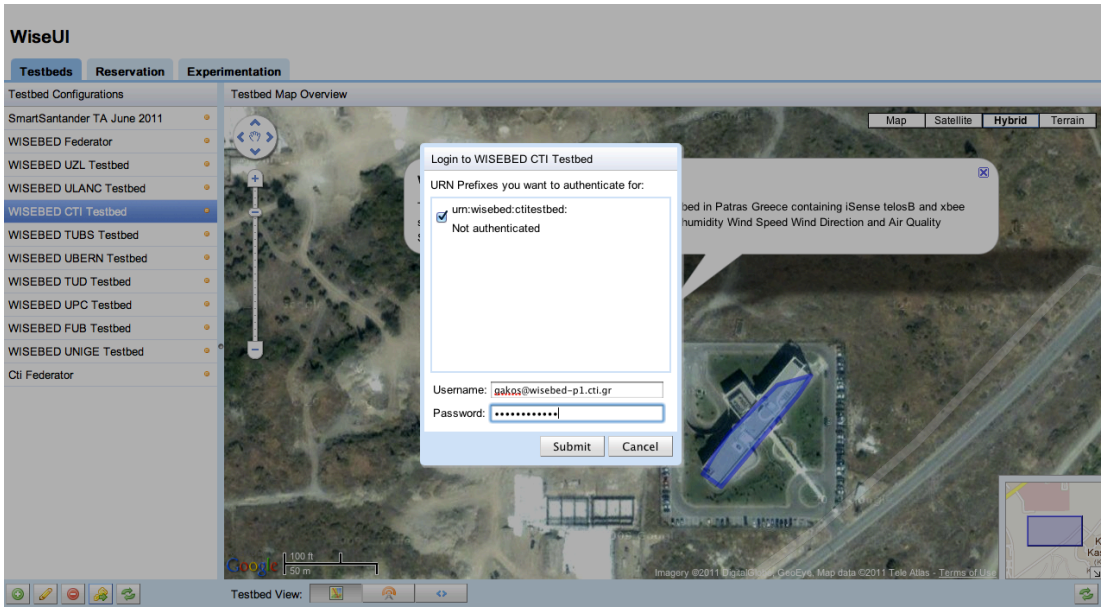
Στην εικόνα 6.3, βλέπουμε ότι με την επιλογή του τρίτου διακόπτη εναλλαγής απεικόνισης, παρουσιάζεται σε μορφή WiseML όλη η απαραίτητη πληροφορία για μια πειραματική υποδομή. Αυτή η πληροφορία έρχεται απευθείας από τον διακομιστή και παρουσιάζεται στον πελάτη χωρίς να έχει υποστεί καμία απολύτως επεξεργασία. Αυτό θα είναι στην ουσία και το κείμενο που θα 'αναγνωστεί' από τα ενδότερα της εφαρμογής και θα παρουσιάσει, όπως είδαμε και παραπάνω, με πιο διαισθητικό τρόπο την εκάστοτε πειραματική υποδομή.

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. 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 6.4 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



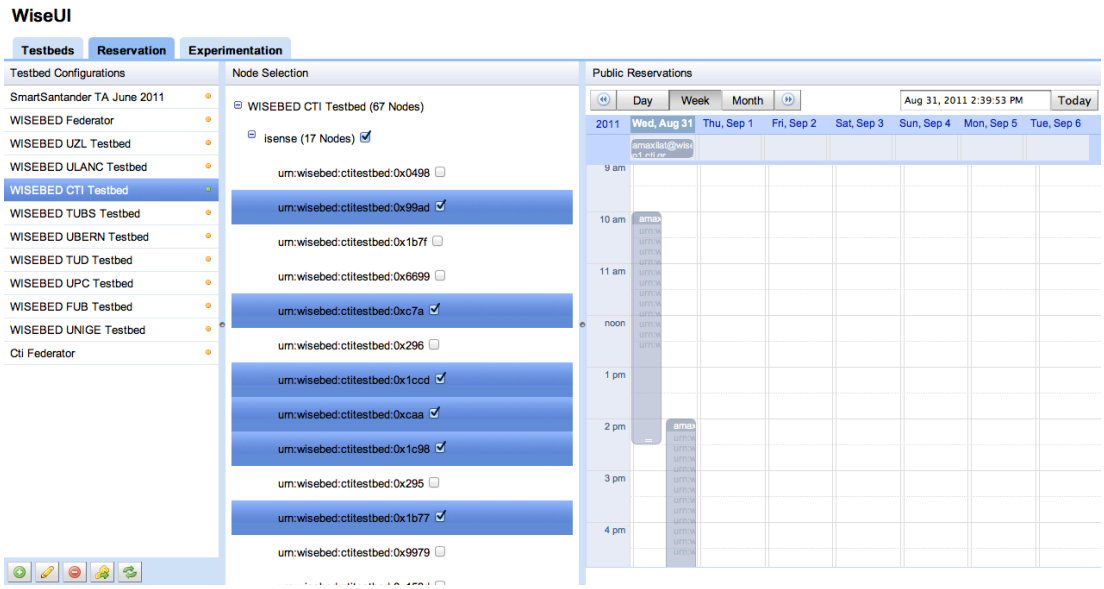
Εικόνα 6.3: Λεπτομέρειες πειραματικής υποδομής σε WiseML



Εικόνα 6.4: Είσοδος σε πειραματική υποδομή

στη δημιουργία μιας νέας κράτησης μεταβαίνουμε στη δεύτερη καρτέλα της εφαρμογής. Στο κυρίως πλαίσιο, και ενώ η λίστα με τις υποδομές έχει παραμείνει

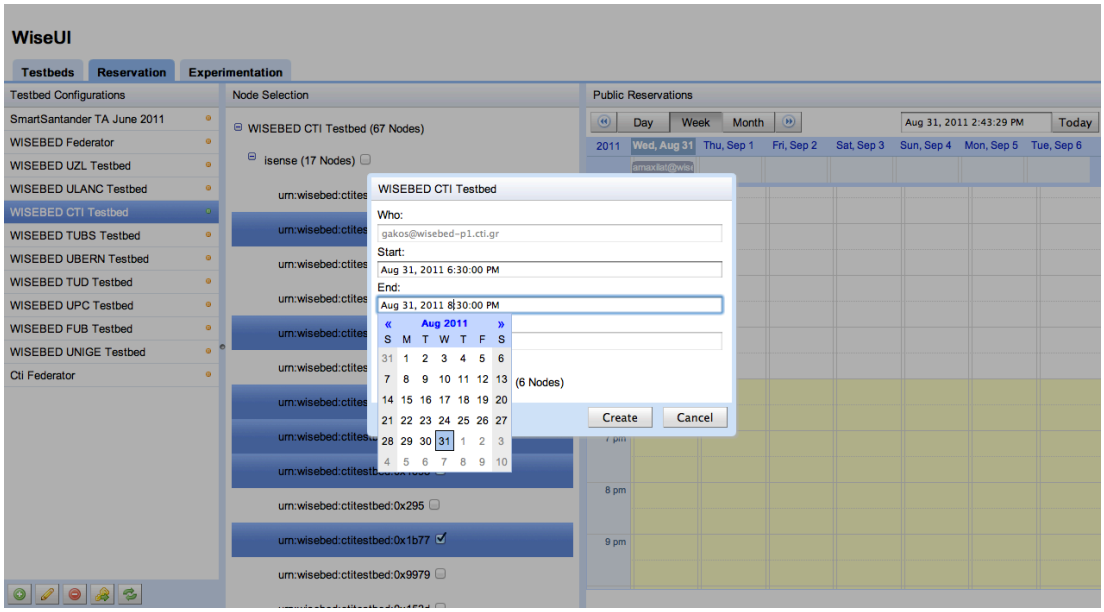
σταθερή, φαίνεται μια λίστα με τους διαθέσιμους αισθητήρες σε επεκτάσιμη μορφή καθώς και ένα ημερολόγιο όπου απεικονίζονται όλες οι προυπάρχουσες κρατήσεις για ένα συγκεκριμένο χρονικό διάστημα. Το πρώτο πράγμα που πρέπει να γίνει για τη δημιουργία μιας κράτησης, είναι να ορίσουμε ποιούς αισθητήρες θέλουμε να χρησιμοποιήσουμε για την εκτέλεση των πειραμάτων μας. Έτσι για παράδειγμα, όπως φαίνεται και στην εικόνα 6.5, επιλέγουμε ένα υποσύνολο από αισθητήρες για να προχωρήσουμε στο επόμενο βήμα για τη δημιουργία της κράτησης.



Εικόνα 6.5: Επιλογή υποσυνόλου αισθητήρων για δέσμευση

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 6.6. 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 υπάρχουσας 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 ημερο-

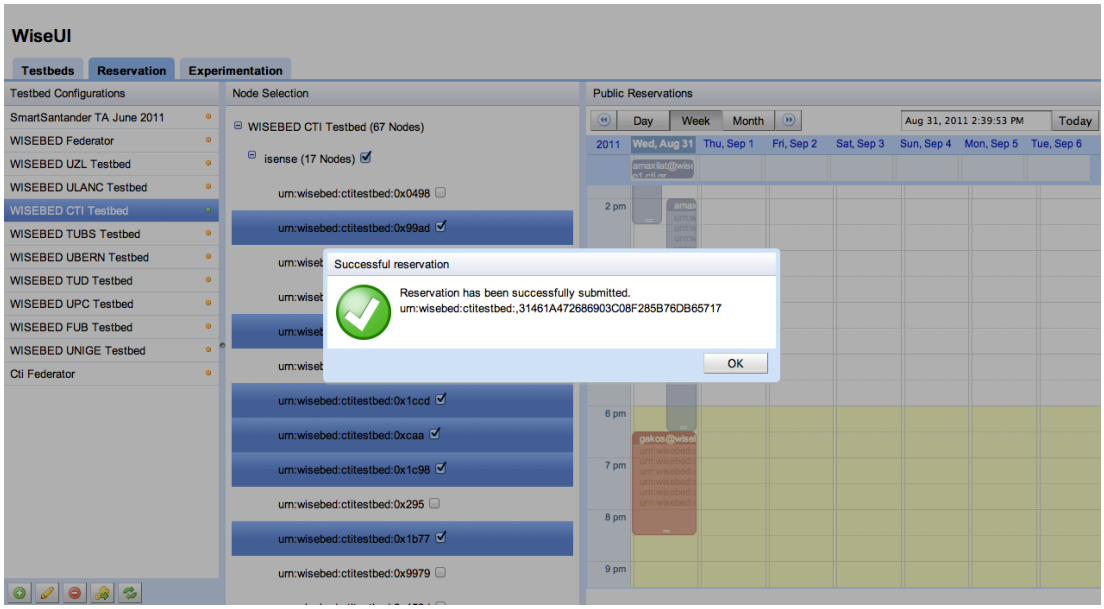


Εικόνα 6.6: Ρύθμιση τελικών παραμέτρων νέας κράτησης

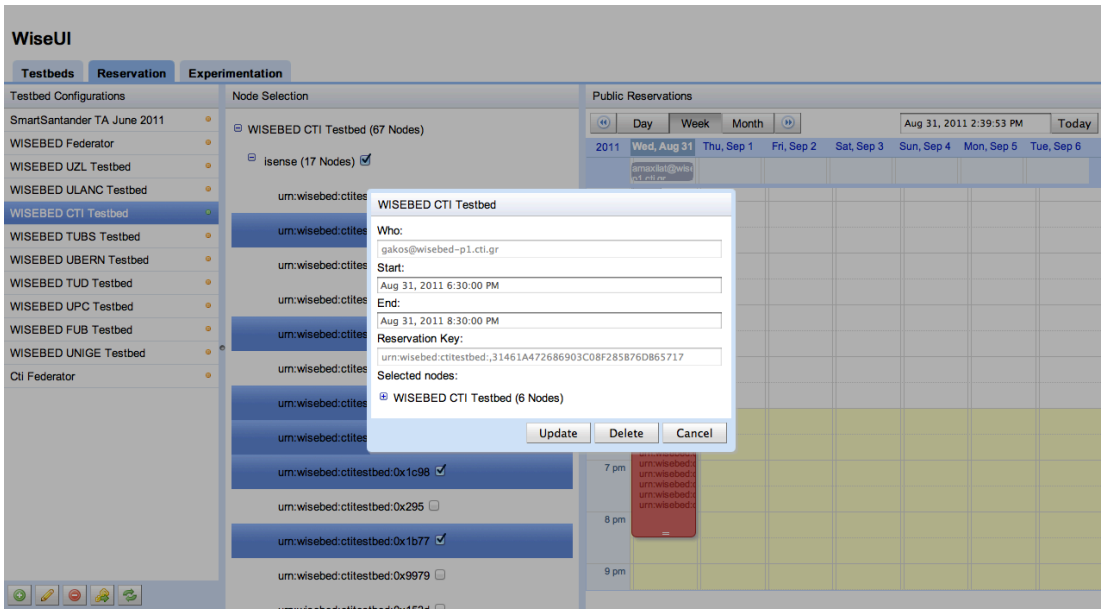
λόγιο και έτσι ο χρήστης δεν χρειάζεται να απομνημονεύει το κάθε κλειδί που του επιστρέφεται για τις κρατήσεις του. Παράλληλα, ο χρήστης βλέπει τη νέα του κράτηση να προσαρτάται στο ημερολόγιο. Για να πετύχουμε μια πιο διαισθητική απεικόνιση των κρατήσεων που ανήκουν στον χρήστη, επιλέξαμε οι κρατήσεις 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 6.7.

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. Ο διάλογος αυτός φαίνεται στην εικόνα 6.8.

Μεταβαίνοντας στην τρίτη καρτέλα της εφαρμογής, μπορούμε πλέον να δούμε ότι το πείραμά μας είναι έτοιμο προς εκτέλεση. Εκεί απεικονίζεται το σύνολο των κρατήσεων και ο χρήστης μπορεί να δει πόσος χρόνος του απομένει, μέχρι τη στιγμή που θα μπορεί πλέον να ανεβάσει το εκτελέσιμο για την έναρξη των πειραμάτων του (Εικόνα 6.9).

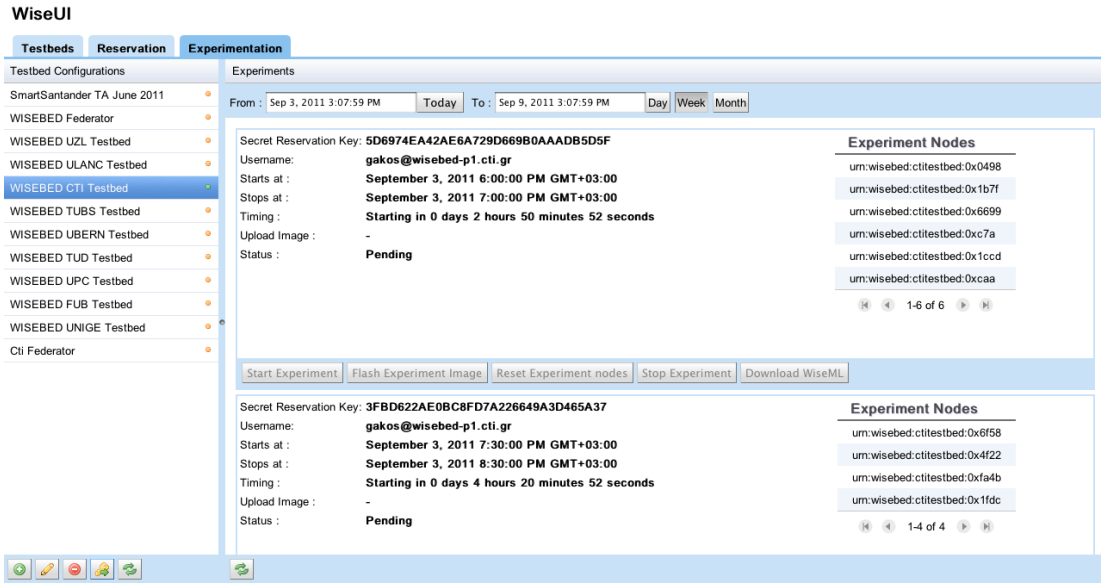


Εικόνα 6.7: Επιτυχής δημιουργία νέας κράτησης



Εικόνα 6.8: Επισκόπηση στοιχείων νέας κράτησης

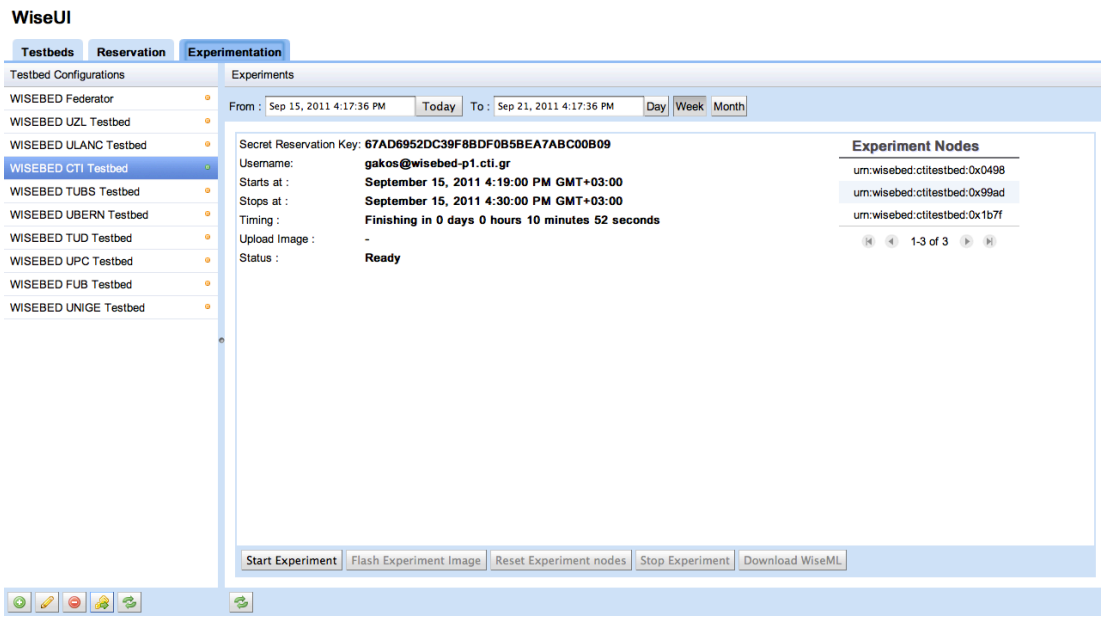
Με την άφιξη του χρόνου έναρξης της κράτησης, ενεργοποιείται, όπως φαίνεται και στην εικόνα 6.10, το κουμπί για την έναρξη του πειράματος. Πατώντας



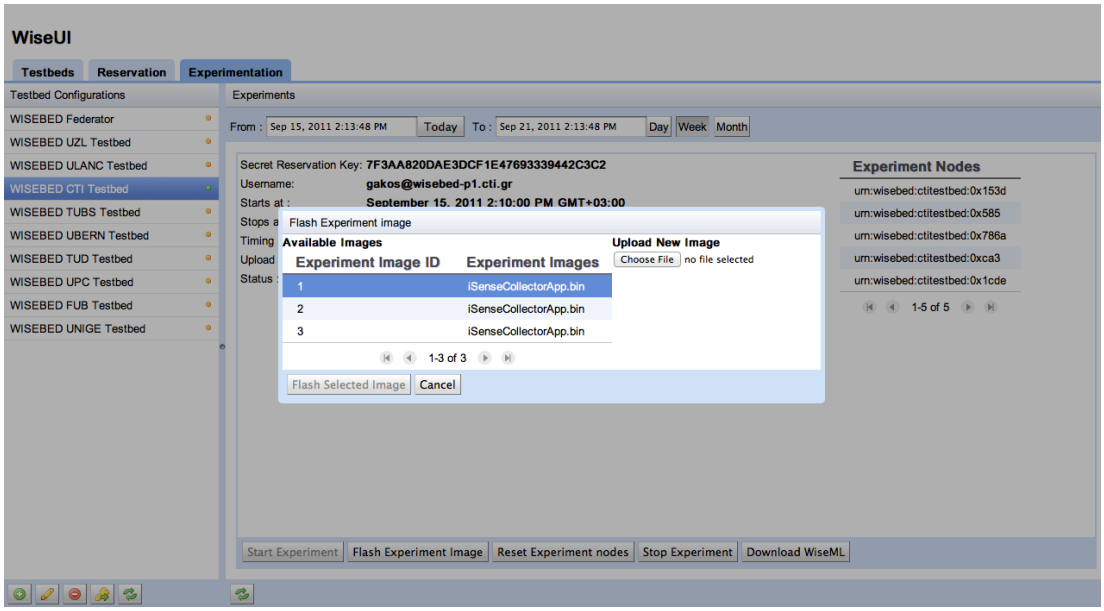
Εικόνα 6.9: Επισκόπηση πειράματος που πρόκειται να εκτελεστεί

αυτό το κουμπί, γίνονται διαθέσιμα όλα τα υπόλοιπα που χρησιμεύουν για τη φόρτωση του εκτελέσιμου κώδικα στους αισθητήρες, την επαναφορά των κόμβων στην αρχική τους κατάσταση, τη διακοπή εκτέλεσης του πειράματος και τέλος το κατέβασμα στο τοπικό μηχάνημα, της συνολικής εξόδου που έχουν φέρει οι αισθητήρες. Επόμενο βήμα για την έναρξη του πειράματος, είναι να παρέχουμε στους αισθητήρες το κώδικα χαμηλού επιπέδου που περιλαμβάνει όλες τις οδηγίες για το τι θα εκτελεστεί σε κάθε κόμβο. Στην εικόνα 6.11 φαίνεται ο διάλογος που προσφέρει τη συγκεκριμένη υπηρεσία. Εκτός όμως από το παραπάνω, το WiseUI διατηρεί στη βάση δεδομένων, τα πιο πρόσφατα πειράματα που ο χρήστης έχει εκτελέσει σε προηγούμενες κρατήσεις. Με αυτόν τον τρόπο, επιταχύνεται η διαδικασία φόρτωσης πειραμάτων στους κόμβους καθώς παρακάμπτεται η χρονοβόρα αποστολή του κώδικα από τον πελάτη στον διακομιστή. Επίσης, ο χρήστης μπορεί να τρέξει ένα παλιότερο πείραμα μέσω ενός άλλου μηχανήματος, χωρίς να είναι απαραίτητο να μεταφέρει μαζί του κάποιο μέσο αποθήκευσης που περιέχει τον κώδικα αυτό.

Μετά το πέρας της διαδικασίας φόρτωσης του κώδικα στους αισθητήρες, μπορούμε να αρχίσουμε να παρατηρούμε την έξοδο που παράγουν οι αισθητήρες κατά την εκτέλεση των οδηγιών πειράματος. Στο δεξιό κομμάτι του πλαισίου επισκόπησης του πειράματος, απεικονίζονται σε μία λίστα με δυνατότητα επιλογής, όλοι οι



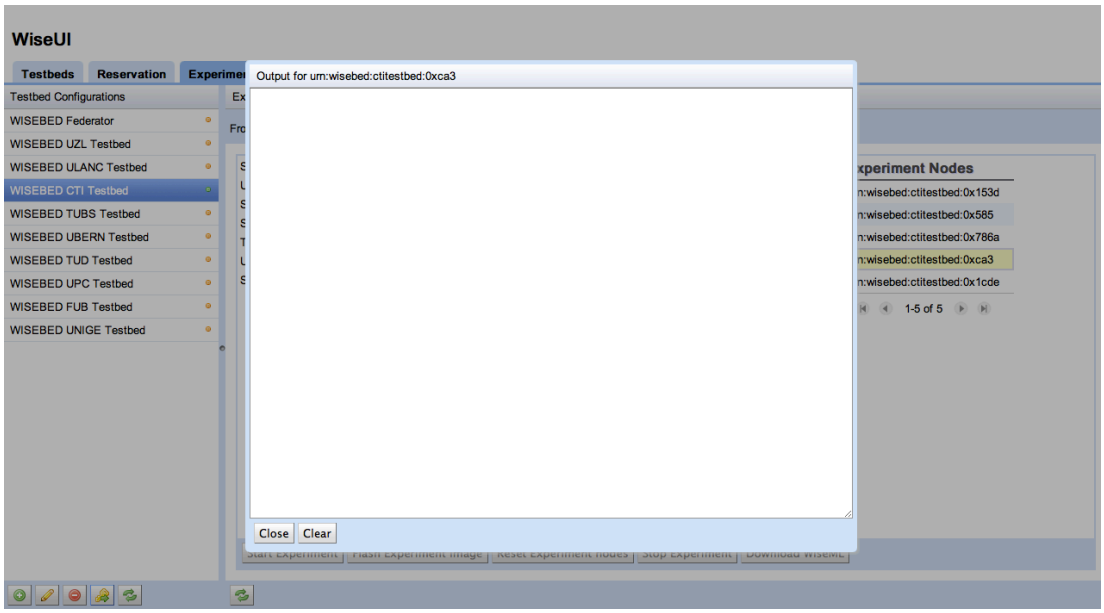
Εικόνα 6.10: Πείραμα έτοιμο προς εκκίνηση



Εικόνα 6.11: Διάλογος φόρτωσης κώδικα χαμηλού επιπέδου στους κόμβους

αισθητήρες που συμμετέχουν στο πείραμα. Κάνοντας διπλό κλικ σε κάποιον από

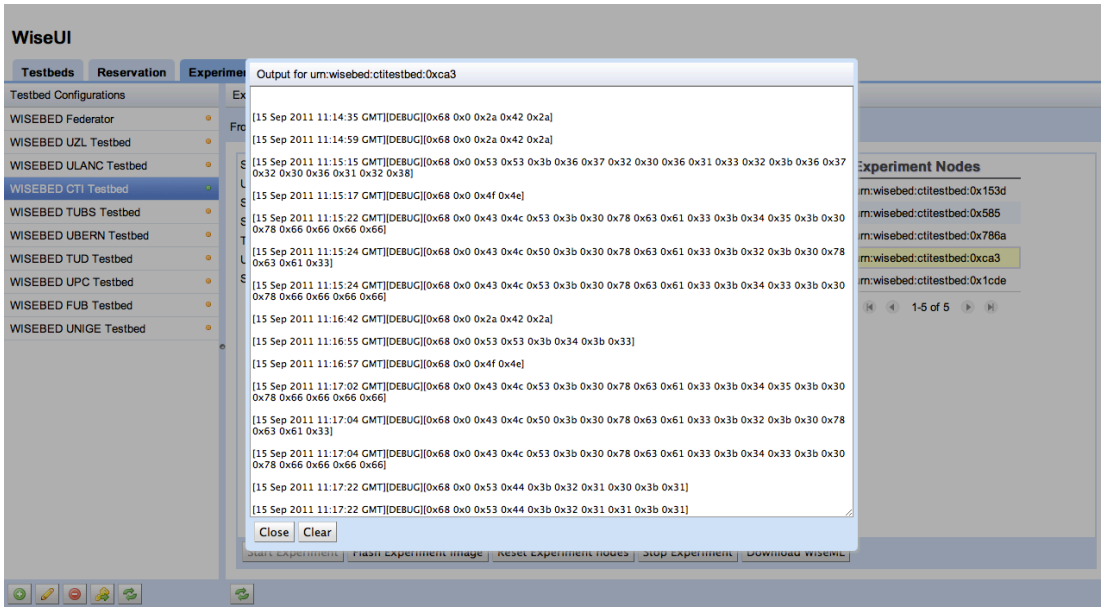
τους κόμβους αυτούς, παρουσιάζεται ένα παράθυρο όπου απεικονίζεται η έξοδος του εκάστοτε αισθητήρα. Στην εικόνα 6.12, ο διάλογος αυτός φαίνεται άδειος καθώς ο συγκεκριμένος κόμβος δεν έχει αρχίσει ακόμη να παράγει κάποια έξοδο.



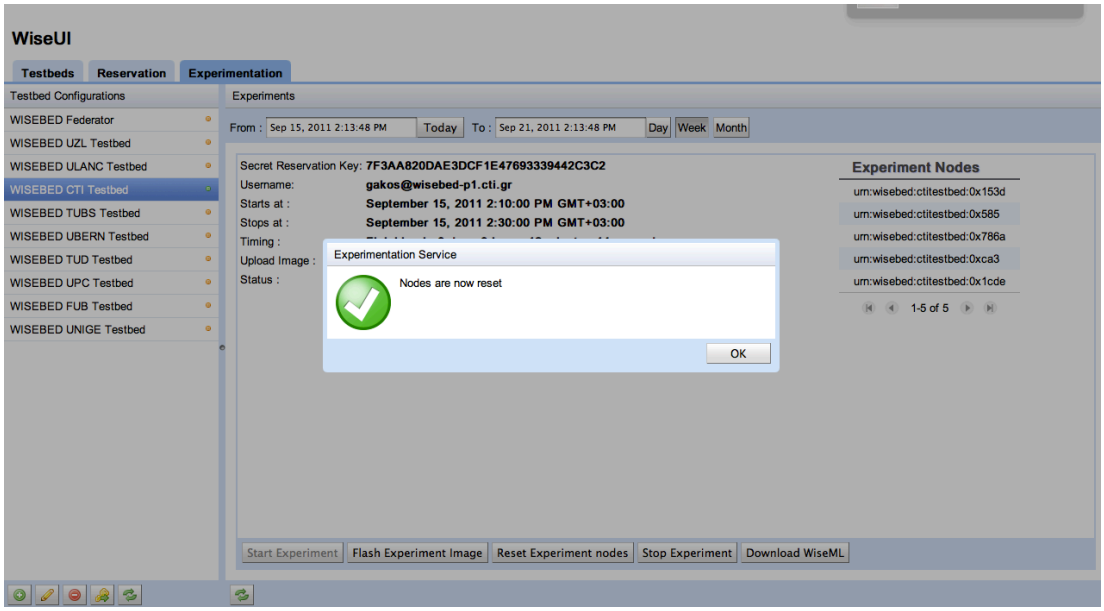
Εικόνα 6.12: Ο κόμβος δεν έχει φέρει ακόμα κάποια έξοδο

Έχοντας αφήσει να περάσουν μερικά δευτερόλευτα, κάνουμε πάλι διπλό κλίκ στον αισθητήρα και βλέπουμε στο παράθυρο να εμφανίζονται δυναμικά τα μηνύματα που παράγει ο κόμβος. Τα μηνύματα έχουν μια συγκεκριμένη δομή, όπου στην αρχή παρουσιάζεται ο χρόνος παραγωγής του μηνύματος, αμέσως μετά ο τύπος του μηνύματος και τέλος οι τιμές που παράγει ο αισθητήρας ανάλογα με τις οδηγίες που εκτελεί σε δεκαεξαδική μορφή (Εικόνα 6.13).

Κατά τη διάρκεια μιας κράτησης, ο χρήστης μπορεί να τρέξει παραπάνω από ένα πειράματα, φορτώνοντας απλά έναν νέο κώδικα χαμηλού επιπέδου στους κόμβους. Για να γίνει αυτό, θα πρέπει με την ολοκλήρωση του κάθε πειράματος να απαιτεί από το WiseUI την επαναφορά των κόμβων στην αρχική τους κατάσταση, 'καθαρίζοντας' τους περιορισμένους πόρους αποθήκευσης του κάθε αισθητήρα. Αφού γίνει αυτό, τότε οι κόμβοι είναι πλεόνδιακομιστή σε θέση να αποθηκεύσουν στη μνήμη τους το νέο κώδικα προς εκτέλεση (6.14).



Εικόνα 6.13: Ο κόμβος παράγει έξοδο και την παρουσιάζει στον πελάτη



Εικόνα 6.14: Επαναφορά κόμβων στην αρχική κατάσταση

Κεφάλαιο 7

Συμπεράσματα και μελλοντικές κατεθύνσεις

Μετα το πέρας της υλοποίησης του WiseUI, μπορούμε με ασφάλεια να πούμε ότι η διαδικασία εκτέλεσης πειραμάτων σε πειραματικές υποδομές έχει επιταχυνθεί πλήρως. Η υπηρεσία παρέχει ένα αρκετά διασθητικό γραφικό περιβάλλον, που επιτρέπει στο χρήστη με μεγάλη ευελιξία να δεσμεύσει πειραματικούς πόρους και να λάβει μια αναλυτική αναφορά της εκτέλεσης των πειραμάτων του. Εκτός αυτού, πρόκειται για μια πλούσια διαδικτυακή εφαρμογή, κάτι που απορρέει πολλά οφέλη. Ένα από αυτά είναι η πλήρης ανεξαρτητοποίηση 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 ασύρματα δίκτυα αισθητήρων, γίνεται πιο εύκολη από κάθε άλλη φορά και ο διαδικτυακός χαρακτήρας της εφαρμογής δίνει πολλές προοπτικές για μελλοντικές κατευθύνσεις.

Το WiseUI, χρησιμοποιεί έναν περιορισμένο αριθμό υπηρεσιών από αυτές που παρέχονται από το Testbed Runtime. Αυτόματα λοιπόν, η λειτουργικότητα της εφαρμογής αποκτά νέες κατευθύνσεις, ενσωματώνοντας μεθόδους που δεν εμπεριέχονται στην τρέχουσα έκδοση ενώ είναι υλοποιημένες στο 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 πειραματικές πλατφόρμες με πλουσιότερη ποικιλία σε αισθητήρες, ενώ θα τους δωθεί η δυνατότητα να εκτελέσουν πιο απαιτητικά και πολύπλοκα πειράματα. Ένα άλλο επιπλέον χαρακτηριστικό του WiseUI, θα μπορούσε να είναι η δυνατότητα ενεργοποίησης και απενεργοποίησης των κόμβων μιας πειραματικής υποδομής. Ο

διαχειριστής μια πειραματικής πλατφόρμας, αποκτά με αυτόν τον τρόπο μεγαλύτερο έλεγχο στην υποδομή και πιο συγκεκριμένα στους αισθητήρες της. Αυτή τη στιγμή, σε περίπτωση πιθανής βλάβης σε κάποια συσκευή, ο διαχειριστής θα πρέπει να έχει φυσική πρόσβαση στο χώρο όπου βρίσκονται τοποθετημένοι οι αισθητήρες και να την αποσυνδέσει από το δίκτυο. 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 χωρίς να είναι απαραίτητη η παρουσία του στο χώρο με τις συσκευές. Εκτός από τα παραπάνω, οι υπηρεσίες που προσφέρονται από την εφαρμογή, συνδέονται άμεσα με την εξέλιξη του TR. Έτσι, όσο προχωρά η ανάπτυξη 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 σύγχρονωνδιακομιστή διαδικτυακών εφαρμογών. Στα πλαίσια μιας τέτοιας προσθήκης, μια πολύ χρήσιμη λειτουργία θα ήταν η δυνατότητα διαμοιρασμού πειραμάτων, τόσο σε επίπεδο αναφοράς αλλά ακόμα και σε επίπεδο εκτέλεσης. Χρήσιμη θα παρουσιαζόταν η δυνατότητα των χρηστών να μοιράζονται τον κώδικα χαμηλού επιπέδου με τον οποίο προγραμματίζουν τους αισθητήρες, έτσι ώστε να μπορούν να εκτελέσουν τα ίδια πειράματα σε κάποια δική τους πειραματική πλατφόρμα, συγκρίνοντας στο τέλος τα αποτελέσματα και βγάζοντας χρήσιμα συμπεράσματα.

Η προσθήκη διαβαθμίσεων 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.

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 στις τεχνολογίες διαδικτύου, η Google δεν καθυστέρησε να υποστηρίξει σχετικές διεπαφές στο GWT. Είναι σημαντικό να σημειωθεί, πως η δυνατότητα τοπικής αποθήκευσης στα μηχανήματα των πελατών, μπορεί να οδηγήσει στον επανασχεδιασμό πολλών λειτουργιών που υλοποιούνται στο WiseUI. Η τοπική ανάκτηση δεδομένων μπορεί να κάνει τον διακομιστή πολύ πιο 'ελαφρύ' και την διεπαφή πολύ πιο γρήγορη και λειτουργική καθώς θα δίνονται μεγαλύτερα περιθώρια για χρήση του WiseUI χωρίς να είναι απαραίτητη η σύνδεση στο διαδίκτυο.

Αδιαμφισβήτητα, η εφαρμογή αποτελεί απόδειξη για το ότι η χρήση της μπορεί

να βρει εφαρμογή σε πραγματικές συνθήκες. Τα ασύρματα δίκτυα αισθητήρων τείνουν να κάνουν την εμφάνισή τους όλο και πιο συχνή στην καθημερινότητά μας και ο συνδυασμός αυτών με τις δυνατότητες του διαδικτύου, θα καταφέρει να λύσει πολλά προβλήματα και να βελτιώσει αρκετά την καθημερινότητά μας.

Εν κατακλείδει, 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.

Αναφορές

[1] Agile software development. http://en.wikipedia.org/wiki/Agile_software_development. 21

[2] Ajax (programming). http://en.wikipedia.org/wiki/Ajax_(programming). 16

[3] Chapter 9. transaction management. http://static.springsource.org/spring/docs/2.0.x/reference/transaction.html. 28

[4] Database management system. http://en.wikipedia.org/wiki/Database_management_system. 20

[5] Dozer. http://dozer.sourceforge.net/documentation/about.html. 26

[6] Factory method pattern - concept. http://en.wikipedia.org/wiki/Factory_method_pattern#Concept. 24

[7] Getting used to asynchronous calls. http://code.google.com/p/google-web-toolkit-doc-1-5/wiki/DevGuideGettingUsedToAsyncCalls. 17

[8] Gilead. http://noon.gilead.free.fr/gilead/. 25

[9] Google web toolkit overview. http://code.google.com/webtoolkit/overview.html. 23

[10] Hibernate (java). http://en.wikipedia.org/wiki/Hibernate_(Java). 29

[11] Iterative and incremental development. http://en.wikipedia.org/wiki/Iterative_and_incremental_development. 21

[12] iwsn api 2.3. http://www.wisebed.eu/images/stories/deliverables/TR/TR-iWSN-API-V2.3.pdf. 14, 15

[13] Mesh networking. http://en.wikipedia.org/wiki/Mesh_networking. 2

[14] Persistence (computer science). http://en.wikipedia.org/wiki/Persistence_(computer_science). 19

[15] Reservation system api (rs api). http://www.wisebed.eu/images/stories/deliverables/TR/TR-RS-API-V1.pdf. 12

[16] Sensor network authentication and authorization api (snaa api). http://www.wisebed.eu/images/stories/deliverables/TR/TR-SNAA-API-V1.pdf. 9

[17] Serialization. http://en.wikipedia.org/wiki/Serialization. 18

[18] Software testing. http://en.wikipedia.org/wiki/Software_testing. 21

[19] Star network. http://en.wikipedia.org/wiki/Star_network. 2

[20] Test driven development. http://en.wikipedia.org/wiki/Test-driven_development. 21

[21] Testbed runtime. https://github.com/itm/testbed-runtime/wiki. 8

[22] Testing methodologies using the google web toolkit. http://code.google.com/webtoolkit/articles/testing_methodologies_using_gwt.html. 29

[23] Transaction processing. http://en.wikipedia.org/wiki/Transaction_processing. 28

[24] Using data transfer objects. http://code.google.com/webtoolkit/articles/using_gwt_with_hibernate.html#Integration_Strategies. 26

[25] Web application - structure. http://en.wikipedia.org/wiki/Web_application#Structure. 32

[26] Why hibernate objects can't be understood when they reach the browser world. http://code.google.com/webtoolkit/articles/using_gwt_with_hibernate.html. 19

[27] Wireless sensor networks. http://en.wikipedia.org/wiki/Wireless_sensor_network. 2

[28] Wisebed - wireless sensor network testbeds - eu project. http://www.wisebed.eu/. 8

[29] Wiseui design note - application architecture. https://github.com/itm/wiseui/wiki/ApplicationArchitectureDesignNote. 33

[30] Wiseui design note - persistence. https://github.com/itm/wiseui/wiki/DesignNotePersistence. 38