ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΕΦΑΛΑΙΟ 7 Συμπεράσματα – Προτάσεις

ΠΑΡΑΡΤΗΜΑ 2

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΚΕΦΑΛΑΙΟ 3 Ανάλυση των SMART διεπαφών

ΚΕΦΑΛΑΙΟ 4 Ανάλυση του Συστήματος

Πρόλογος

Η εργασία αυτή εκπονήθηκε στα πλαίσια της διπλωματικής εργασίας μεταπτυχιακού επιπέδου του τμήματος Διδακτικής της τεχνολογίας και Ψηφιακών Συστημάτων του Πανεπιστημίου Πειραιώς. Στόχο της αποτελεί η ανάπτυξη μιας πλατφόρμας λογισμικού για την διαχείριση συσκευών που δρουν σε περιβάλλοντα Smart Spaces.

Μερικά από τα γνωρίσματα που χαρακτηρίζουν περιβάλλοντα τέτοιου τύπου, είναι η μεγάλη ποικιλία διαφορετικών συσκευών και τεχνολογιών που δρουν κάτω από αυτά, η συνεχής προσθήκη νέων συσκευών και τεχνολογιών και η διαρκής αλλαγή των μεταβλητών τους.

Η πλατφόρμα που αναπτύχθηκε έχει σαν σκοπό να ενταχθεί σαν ένα ενδιάμεσο επίπεδο μεταξύ του περιβάλλοντος – συσκευών και του χρήστη/διαχειριστή, με τέτοιο τρόπο ώστε να αποκρύπτει τις λεπτομέρειες της εκάστοτε τεχνολογίας κάθε συσκευής και να παρουσιάζει τις προς διαχείριση συσκευές με έναν ορισμένο τρόπο. Για παράδειγμα, για τον τελικό χρήστη που χρησιμοποιεί την πλατφόρμα, δεν θα πρέπει να υπάρχει διαφορά ανάμεσα σε δύο συσκευές ίδιου τύπου, π.χ. δύο "έξυπνα" φώτα, ανεξαρτήτως της τεχνολογίας υλοποίησης και επικοινωνίας που χρησιμοποιούν. Οι δύο συσκευές αυτές θα παρουσιάζονται και θα διαχειρίζονται με τον ίδιο τρόπο και θα μπορούν επιπρόσθετα να επικοινωνούν και να συνεργάζονται μεταξύ τους μέσα από καλά ορισμένες διεπαφές τις πλατφόρμας. Η αρχιτεκτονική και ο σχεδιασμός της πλατφόρμας θα πρέπει να επιτρέπει την εισαγωγή νέων τεχνολογιών με εύκολο και γρήγορο τρόπο, ώστε να μειώνεται αισθητά ο χρόνος ανάπτυξης λογισμικού.

Επιπρόσθετα, η πλατφόρμα θα διαχειρίζεται με διαφανή στον τελικό χρήστη τρόπο, το επόμενο χαρακτηριστικό των Smart Spaces, δηλαδή την εισαγωγή και απόσυρση συσκευών. Νέες συσκευές, θα πρέπει να μπορούν να εγκαθίστανται και να απεγκατασταθούν, με τρόπο ώστε να μην υπάρχουν παρενέργειες τύπου "Επανεκκίνησης" του συστήματος ή επιμέρους στοιχείων του.

Για την υλοποίηση της πλατφόρμας επιλέχθηκε η χρήση της τεχνολογίας OSGi, η οποία προσφέρει σημαντικά πλεονεκτήματα, τα οποία περιγράφονται σε επόμενα κεφάλαια. Επιπλέον, οι τύποι των συσκευών και ο τρόπος παρουσίασης και διαχείρισης, επιλέχθηκαν με τέτοιο τρόπο ώστε να εφαρμόζουν καλά σε ένα περιβάλλον Smart Space τύπου Smart Home. Δηλαδή έχουν δημιουργηθεί διεπαφές για έξυπνα φώτα, διακόπτες και κάμερες, που είναι

αρκετές για την επίδειξη της δυναμικότητας της πλατφόρμας. Στα επόμενα κεφάλαια θα γίνει λεπτομερής παρουσίαση αυτών και όλων των κρίσιμων σημείων της πλατφόρμας, μαζί με πληροφορίες εγκατάστασης και παραδειγμάτων χρήσης. Τέλος, θα παρουσιαστούν κάποια συμπεράσματα και θα γίνουν προτάσεις για περαιτέρω ανάπτυξη.

Ευχαριστίες

Στο σημείο αυτό, κρίνω απαραίτητο να ευχαριστήσω τους συνεργάτες μου στην εταιρία inAccessNetworks, για τις χρήσιμες γνώσεις που μου έχουν δώσει και την υποστήριξη που μου έχουν δείξει. Αλφαβητικά:

Καρράς Γιάννης, Λιβέρεζας Γιάννης, Οικονόμου Δημήτρης και Παπαδόπουλος Νεκτάριος.

Επιπρόσθετα, θα ήθελα να ευχαριστήσω την καθηγήτρια μου δ/ιδα Σταυρουλάκη, λέκτορα του Πανεπιστημίου Πειραιώς, για την ευκαιρία που μου έδωσε για την εκπόνηση της διπλωματικής εργασίας με το τόσο ενδιαφέρον και επίκαιρο αυτό θέμα.

Κεφάλαιο 1ο

Εισαγωγή

Στην ενότητα αυτή, δίνονται μερικές εισαγωγικές πληροφορίες σχετικά με τα περιβάλλοντα Smart Spaces και την τεχνολογία OSGi. Σκοπός είναι, ο αναγνώστης να αποκτήσει μια βασική ιδέα για το τι είναι ένα περιβάλλον Smart Space, τι είναι η τεχνολογία OSGi και πως μπορεί να χρησιμοποιηθεί στην διαχείριση ενός τέτοιου περιβάλλοντος, όπως και για ποιόν λόγω επιλέχθηκε αυτή η τεχνολογία ανάμεσα σε άλλες.

1.1 Pervasive Systems

Οι πιο χρήσιμες τεχνολογίες στον τομέα των υπολογιστών είναι αυτές που εξαφανίζονται. Εισέρχονται με τέτοιον τρόπο στην καθημερινότητα του περιβάλλοντος στο οποίο δρουν και τελικά ενοποιούνται με αυτό και δεν ξεχωρίζονται [3]. Τα Pervasive Systems έχουν σαν κύριο στόχος τους αυτή τη θεωρεία. Ας δούμε όμως λίγο πιο αναλυτικά τι αποτελεί ένα Pervasive System, ποιοι είναι οι στόχοι αυτού και ποιες οι προκλήσεις που πρέπει να αντιμετωπιστούν.

Ένα Pervasive Computer System, αποτελείται από έξυπνες συσκευές (αισθητήρες κτλ.), ένα σύνολο δικτύων (ενσύρματα, ασύρματα, add-hock), ένα λογισμικό με αρχιτεκτονική που επιτρέπει την εύρεση συσκευών και υπηρεσιών, να παρέχει τρόπους διαχείρισης των συσκευών, να παρέχει ασφάλεια, να είναι Context-aware και Energy-aware και να παρέχει τρόπους επικοινωνίας με τον χρήστη [3].

Επιγραμματικά, οι απαιτήσεις ενός τέτοιου συστήματος είναι οι ακόλουθές:

- Υποστήριξη ετερογενών συσκευών
- Υποστήριξη για περιεχόμενο (Context)
- Υποστήριξη για επιλογή συσκευών/υπηρεσιών
- Υποστήριξη για γρήγορη ανάπτυξη και εγκατάσταση εφαρμογών

Με κύριες προκλήσεις την κλιμάκωση (scalability) και την διαδραστικότητα (Interoperability) [3]

1.1.1 Smart Spaces

Τα Smart Spaces, αποτελούν μια ειδική κατηγορία των Pervasive Systems. Σκοπός τους είναι να συλλέγουν πληροφορίες από το περιβάλλον και τον χρήστη, ώστε να προσφέρουν υπηρεσίες που θα τον διευκολύνουν στην διαβίωσή του. Πρέπει να του αποκρύπτουν τα ιδιαίτερα χαρακτηριστικά του περιβάλλοντος και ιδανικά ο χρήστης δεν θα πρέπει να χρειάζεται κάποιον ιδιαίτερο εξοπλισμό για να αλληλεπιδράσει με αυτά [3][4].

Συγκεκριμένα τα Smart Spaces έχουν τα ακόλουθα χαρακτηριστικά:

- Δυνατότητες αισθήσεων: Αυτό απαιτείται για να είναι δυνατή η συλλογή πληροφοριών από τον περιβάλλοντα χώρο, γνωστό και ως Context Awareness. Ένα σύνολο αισθητήρων δηλαδή, που παρέχει πληροφορίες για τα δεδομένα του φυσικού περιβάλλοντος.
- Δυνατότητες προσαρμοστικότητας: Το περιβάλλον/σύστημα, θα πρέπει να μπορεί να προσαρμόζεται σε νέες συνθήκες.
- Δυνατότητα δράσης: Το περιβάλλον θα πρέπει να έχει την δυνατότητα να αλλάζει τις συνθήκες του με βάση τις επικρατούσες συνθήκες ή μετά από προτροπή του χρήστη.

Τα Smart Spaces χρησιμοποιούνται σε διάφορα περιβάλλοντα όπως αυτοκίνητα, νοσοκομεία, σπίτια κ.α. Μεγάλο ενδιαφέρον φαίνεται να έχει η χρήση τους σε οικιακά περιβάλλοντα, τα γνωστά Smart Homes, για την διαχείριση συσκευών ώστε να διευκολύνουν τους χρήστες σε καθημερινές εργασίες, παρέχουν ένα άλλο επίπεδο ασφάλειας, υπηρεσίες για την οικονομία ενέργειας και ένα πλήθος άλλων υπηρεσιών. Η εργασία αυτή είχε σαν επίκεντρο αυτού του είδους των Smart Spaces, λόγω της οικειότητας του συγγραφέα από τον χώρο εργασίας του.

Στο σημείο αυτό κρίνεται ενδιαφέρον να αναφερθεί πως η «μόδα» των Smart Spaces έχει αρχίσει να εξαπλώνεται με γοργούς ρυθμούς. Μεγάλες εταιρίες όπως η Echelon δραστηριοποιούνται ενεργά σε αυτό τον χώρο. Πρωτόκολλα επικοινωνίας ορίζονται ως Standards σε Ευρώπη και Αμερική (π.χ. Lontalk, Z-wave κ.α.) τα οποία και συνεχώς εμπλουτίζονται, ακολουθούμενα από εταιρίες μικροηλεκτρονικής (Echelon, SecyourIT, Continental κ.α.) που κατασκευάζουν συσκευές που υλοποιούν αυτά τα πρωτόκολλα.

Επίσης , ενδιαφέρον έχει να αναφερθεί, πως στην Ελληνική αγορά η εταιρία inAccessNetworks, δραστηριοποιείται ενεργά στο χώρο αυτό, παρέχοντας υπηρεσίες Smart Home και μάλιστα κάνοντας χρήση της τεχνολογίας OSGi, η οποία αποτελεί και θέμα των επόμενων ενοτήτων.

1.2 Η επιλογή του OSGi

Για την εργασία αυτή επιλέχθηκε η τεχνολογία OSGi, σαν την καρδιά του συστήματος διαχείρισης συσκευών σε ένα περιβάλλον Smart Space. Αρκετές είναι οι τεχνολογίες οι οποίες θα μπορούσαν να είναι υποψήφιες προς χρήση. Στην έρευνα που έχει διεξαχθεί στο [4], αναφέρεται ένας αριθμός τέτοιον τεχνολογιών και γίνεται μια βασική σύγκριση αυτών. Ακολουθεί ένας συγκριτικός πίνακας της μελέτης:

Feature	SLP	Jini	Salutation	UPnP	OSGi
Developer	IETF	Sun Microsystems	Salutation Consortium	Microsoft	OSGi Alliance
Licence	open source	open license, but a fee for commercial use	open source	open (only for members)	open source
Version	2	1.0	2.1	0.91	4
Network transport	TCP/IP	independent	independent	TCP/IP	independent
Programming language	independent	Java	independent	independent	Java
OS and platform	dependent	independent	dependent	dependent	independent
Code mobility	no	yes (Java RMI)	no	no	yes
Service attributes searchable	yes	yes	yes	no	yes
Central cache repository	yes (optional)	optional using SLP	yes (optional)	no	no
Operation without directory	yes	Lookup Table required	yes	-	-
Security	IP dependent	Java based	authentication	IP dependent	Java based

Εικόνα 1.1: Σύγκριση τεχνολογιών εύρεσης υπηρεσιών [4]

Από τον πίνακα αυτόν φαίνεται πως το OSGi είναι μια από τις καλύτερες επιλογές ανάμεσα στις υπόλοιπες τεχνολογίες. Επιπρόσθετα σε αυτά, στην επιλογή ρόλο έπαιξε και η εμπειρία του συγγραφέα στην τεχνολογία OSGi από τον χώρο εργασίας του.

Σημείωση:

Για την υλοποίηση της εργασίας χρησιμοποιήθηκε το OSGi Framework implementation με το όνομα OSCAR. Ο OSCAR, αποτελεί μια υλοποίηση του OSGi r3 Specification και για την χρήση του μπορείτε να αναφερθείτε στο [9].

Στο Internet κυκλοφορούν και άλλες δωρεάν υλοποιήσεις του OSGi r3 Specification, όπως για παράδειγμα από την Knopflerfish το ομώνυμο και από την Eclipse το Equinox. Η επιλογή του OSCAR έγινε μόνο για λόγους της εξοικείωσης του συγγραφέα σε αυτή την υλοποίηση και η πλατφόρμα θα πρέπει να λειτουργεί κανονικά σε οποιαδήποτε άλλη.

1.3 Το Πλαίσιο OSGi (Framework) [8]

Η πολυπλοκότητα του λογισμικού αυξάνεται με αλματώδεις ρυθμούς. Στις μέρες μας, ένα μεγάλο μέρος της πολυπλοκότητας αυτής απορρέει από τους μικρούς κύκλους ζωής των προϊόντων, τις απαιτήσεις για πολύ μεγαλύτερες δυνατότητες και λόγω τις μεγάλης ποικιλίας ίδιων ουσιαστικά προϊόντων (e.g. διαφορετικό Hardware και λειτουργικό σύστημα). Αυτές οι τάσεις προκαλούν κόστη στην ανάπτυξη λογισμικού που αποτελεί από τα μεγαλύτερα μέρη του συνολικού κόστους της διαδικασίας ανάπτυξης λογισμικού.

Στις μέρες μας, η ανάπτυξη λογισμικού εξαρτάται από την δυνατότητα επαναχρησιμοποίησης της λειτουργικότητας προϋπάρχοντος λογισμικού. Τα τελευταία 20 χρόνια έχουν αναπτυχθεί αρκετά επαναχρησιμοποιήσιμα και «de facto standard» κομμάτια λογισμικού που χρησιμοποιούνται για την ανάπτυξη νέου. Ένα παράδειγμα σε αυτό είναι η ανάπτυξη του λογισμικού ανοιχτού κώδικα. Στην πραγματικότητα βέβαια, η χρήση τέτοιων βιβλιοθηκών δεν έρχεται με μηδενικό κόστος. Οι βιβλιοθήκες αυτές με το πέρασμα του χρόνου έχουν γίνει τόσο πολύπλοκες που και οι ίδιες απαιτούν άλλες βιβλιοθήκες οι οποίες μπορεί να μην χρησιμέψουν ποτέ στην ανάπτυξη ενός συγκεκριμένου λογισμικού.

Μία λύση στο σημερινό πρόβλημα αυτό της ανάπτυξης λογισμικού θα ήταν η δημιουργία νέου κώδικα αντί η σπατάλη χρόνου για την προσαρμογή υπάρχοντα σε ένα νέο έργο. Στην πραγματικότητα, η διαδικασία αυτή της προσαρμογής κώδικα καταναλώνει πολύ χρόνο από τους σημερινούς προγραμματιστές. Συνεπώς, είναι σαφής η ανάγκη για εργαλεία που θα

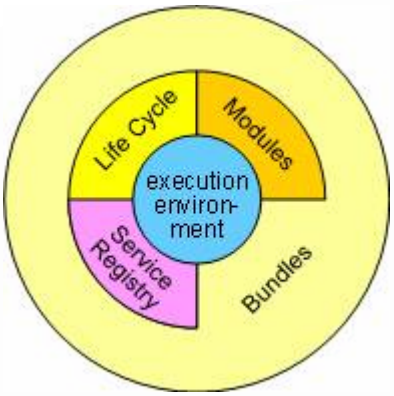
παρέχουν standards στην διαδικασία της μεταφοράς κώδικα ώστε η διαδικασία αυτή να γίνεται αξιόπιστα, γρήγορα και οικονομικά.

Η τεχνολογία OSGi αποτελεί ένα δυναμικό υπόστρωμα για την Java. Η Java παρέχει στον κώδικα μεταφερσημότητα που απαιτείται για την υποστήριξη προϊόντων σε διαφορετικές πλατφόρμες. Η τεχνολογία OSGi παρέχει την δυνατότητα υλοποίησης προγραμμάτων από την χρήση μικρότερων μονάδων οι οποίες συνεργάζονται μεταξύ τους μέσα από standard διαδικασίες για την δημιουργία μιας εφαρμογής η οποία τελικά παρατάσσεται (deployed).

Η πλατφόρμα OSGi παρέχει την δυνατότητα δυναμικής αλλαγής συγκεκριμένων ιδιοτήτων (properties) προγραμμάτων, χωρίς να απαιτείται επανεκκίνηση του συστήματος. Επιπρόσθετα, για να ελαχιστοποιηθούν τα ζευγαρώματα κώδικα (coupling) όπως και για να μπορεί ο κώδικας να είναι ευκολότερα διαχειριζόμενος, χρησιμοποιείται το SOA (Service Oriented Architecture), ώστε οι υπηρεσίες να μπορούν να εντοπίζουν η μία την άλλη κατά απαίτηση.

Το **OSGi Alliance**, έχει δημιουργήσει μια πληθώρα των πιο συχνά χρησιμοποιούμενων υπηρεσιών από τους προγραμματιστές, όπως HTTP Server, configuration, logging, security, User administration, XML και άλλες. Οι υπηρεσίες αυτές μπορούν να αποκτηθούν από διαφορετικούς προμηθευτές ή και να δημιουργηθούν από οποιονδήποτε.

Αυτοί που τελικά υιοθετούν την τεχνολογία OSGi, εποφελούντε από καλύτερες αποκρίσεις σε on-time-market απαιτήσεις και από τα μικρότερα κόστη ανάπτυξης, επειδή το OSGi παρέχει pre-built και pre-tested modules, έτοιμα να ενταχθούν στον νέο κώδικα με standard διαδικασίες. Επιπρόσθετα, η τεχνολογία OSGi, μειώνει το κόστος συντήρησης και παρέχει νέες δυνατότητες στις υπηρεσίες μεταπώλησης, μια και νέες εφαρμογές μπορούν να ενταχθούν πολύ εύκολα.

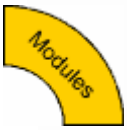


Το βασικό στοιχείο της τεχνολογίας OSGi είναι το OSGi Framework. Το Framework παρέχει ένα standard περιβάλλον στις εφαρμογές που λειτουργούν σε αυτό, τα bundles. Το Framework χωρίζεται σε έναν αριθμό από στρώματα, τα οποία ονομαστικά είναι τα παρακάτω:

- L0: Περιβάλλον Εκτέλεσης
- L1: Modules
- L2: Διαχείριση του κύκλου ζωής
- L3: Μητρώο υπηρεσιών

Επιπρόσθετα ένα επίπεδο ασφάλειας υπάρχει ανάμεσα σε όλα τα στρώματα.

Το περιβάλλον εκτέλεσης (L0) αποτελείται από την προδιαγραφή της Java. Διαφορετικά περιβάλλοντα Java όπως J2SE, CDC, CLDC, MIDP κτλ. μπορούν να χρησιμοποιηθούν.



Το επίπεδο 1, προδιαγράφει τις πολιτικές φόρτωσης των κλάσεων. Το OSGi Framework παρέχει ένα πολύ δυνατό και πολύ αυστηρά καθορισμένο μοντέλο φόρτωσης κλάσεων. Χτίζεται πάνω στο στρώμα της Java, αλλά παρέχει Modularity. Στην Java, κανονικά ορίζεται ένα classpath που περιέχει όλες τις χρησιμοποιούμενες κλάσεις και τους πόρους. Το OSGi παρέχει ένα ιδιωτικό χώρο κλάσεων για κάθε Module, όπως και ελεγχόμενο Linking μεταξύ των κλάσεων διαφορετικών modules.

Το επίπεδο 2, προσθέτει την δυναμική δυνατότητα στα bundles να εγκατασταθούν, να ξεκινήσουν, να σταματήσουν, να ενημερωθούν και να απεγκατασταθούν, on the runtime. Τα bundles εξαρτώνται από το επίπεδο 1 για την φόρτωση των κλάσεων. Χρησιμοποιούνται αυστηροί μηχανισμοί εξαρτήσεων ώστε να διασφαλιστεί ένα σωστό περιβάλλον λειτουργίας. Το επίπεδο αυτό προστατεύεται από δικλείδες ασφαλείας που το καθιστούν εικονικά απαραβίαστο σε επιθέσεις.

Το επίπεδο 3 αποτελεί το μητρώο υπηρεσιών. Το επίπεδο αυτό παραχωρεί την δυνατότητα επικοινωνίας μεταξύ των bundles. Τα bundles μπορούν να επικοινωνήσουν με τον παραδοσιακό τρόπο διαμοιρασμού κλάσεων. Αλλά αυτός ο τρόπος δεν παρέχει μεγάλη δυναμικότητα σε κώδικα που αφαιρείται και προστίθεται. Το επίπεδο αυτό παρέχει την δυνατότητα διαμοιρασμού αντικειμένων (Java Objects) μεταξύ των bundles. Ένας αριθμός από Events έχει οριστεί ώστε να γνωστοποιείται η εγκατάσταση ή η απεγκατάσταση νέων υπηρεσιών στα bundles. Οι υπηρεσίες είναι Java Objects που μπορούν να αναπαριστούν οτιδήποτε, όπως για παράδειγμα μια φυσική συσκευή π.χ. ένα UPnP Device.

1.3.1 Ασφάλεια

Το μοντέλο ασφάλειας του OSGi βασίζεται στον μοντέλο ασφάλειας της Java και Java2. Η γλώσσα Java εκ κατασκευής περιορίζει πολλές πιθανές επιθέσεις. Για παράδειγμα, οι τακτικές τύπου buffer overflow δεν έχουν αντίκτυπο. Επίσης, οι Access Modifiers της Java, δίνουν την δυνατότητα απόκρυψης κώδικα από τρίτους. Επιπρόσθετα το μοντέλο OSGi παρέχει την δυνατότητα χρήσης private classes, κάτι που δεν παρέχεται από την Java.

1.3.2 Standard Υπηρεσίες OSGi

Μια πληθώρα από standard υπηρεσίες για το OSGi έχουν δημιουργηθεί προς χρήση από τους προγραμματιστές. Αυτές αναφέρονται στο παράρτημα 1.

1.3.3 Συμπεράσματα

Οι προδιαγραφές του OSGi είναι τόσο ευρέος εφαρμόσιμες διότι η πλατφόρμα αποτελεί ένα μικρό στρώμα λογισμικού που επιτρέπει σε πολλαπλές μονάδες λογισμικού γραμμένες σε Java, να συνεργάζονται κάτω από την ίδια Java Virtual Machine (JVM). Παρουσιάζει ένα εκτεταμένο μοντέλο ασφάλειας έτσι ώστε τα επιμέρους στοιχεία να λειτουργούν κάτω από ένα ασφαλές περιβάλλον. Επίσης κάτω από τις κατάλληλες άδειες, τα bundles μπορούν να επικοινωνούν και να επαναχρησιμοποιούν κώδικα με τρόπο που δεν υπάρχει σε άλλο περιβάλλον Java.

Επιπλέον, η διαδεδομένη χρήση του OSGi από επιχειρήσεις δημιουργεί μια μεγάλη αγορά από υπηρεσίες OSGi, οι οποίες μπορούν να λειτουργήσουν σε μια μεγάλη ποικιλία συσκευών, από πολύ μικρές μέχρι πολύ μεγάλες.

Συνεπώς, η χρήση του OSGi μπορεί να μειώσει αισθητά το κόστος ανάπτυξης του λογισμικού όπως και να προσθέσει νέες ευκαιρίες για τις επιχειρήσεις στην αγορά.

Κεφάλαιο 2ο

Αρχιτεκτονική του Συστήματος

2.1 Απαιτήσεις

Όπως έχει προαναφερθεί, στόχος του συστήματος είναι η διαχείριση συσκευών που λειτουργούν σε ένα περιβάλλον Smart Space, το οποίο έχει εξειδικευτεί σε περιβάλλον Smart Home. Η διαχείριση από την πλατφόρμα πρέπει να γίνεται με τέτοιον τρόπο ώστε οι διαφορετικές τεχνολογίες που απαρτίζουν τις συσκευές, να αποκρύβονται από τον εκάστοτε διαχειριστεί-χρήστη.

Επιπρόσθετα, η πλατφόρμα θα πρέπει να παρέχει την δυνατότητα στις συσκευές να επικοινωνούν και να αλληλεπιδρούν μεταξύ τους, για την επίτευξη ενός στόχου, όπως για παράδειγμα σε ένα σύστημα συναγερμού, ένας έξυπνος αισθητήρας κίνησης που χρησιμοποιεί μια τεχνολογία Α (π.χ. ασύρματη τεχνολογία **ZWave**), να μπορεί να προκαλέσει την λειτουργία ενός έξυπνου λαμπτήρα μιας τεχνολογίας Β (π.χ. ενσύρματης τεχνολογίας **Lontalk**), κάτι που δεν θα ήταν δυνατό χωρίς την χρήση της πλατφόρμας.

Τέλος η πλατφόρμα θα έχει την δυνατότητα να χειρίζεται δυναμικά εγκατάσταση και αφαίρεση συσκευών, κατά την ώρα λειτουργίας, με το σύστημα να παραμένει σταθερό και να είναι συνεχώς έτοιμο για χρήση. Επίσης το σύστημα θα πρέπει να σχεδιαστεί με τέτοιο τρόπο ώστε να μπορεί να προσαρτήσει νέες τεχνολογίες με εύκολο τρόπο.

Αν μπορούσαμε να δημιουργήσουμε μια λίστα με τους στόχους του συστήματος, θα είχαμε κάτι σαν την λίστα που ακολουθεί:

1. Απόκρυψη τεχνολογίας συσκευών

Συσκευές ίδιου τύπου, για παράδειγμα έξυπνοι αισθητήρες, αν και διαφορετικής τεχνολογίας, πρέπει να παρουσιάζονται και να διαχειρίζονται από τον τελικό χρήστη με τον ίδιο ακριβώς τρόπο. Με άλλα λόγια ο χρήστης θα αγνοεί την τεχνολογία της κάθε συσκευής.

2. Συνεργασία συσκευών διαφορετικής τεχνολογίας

Δύο συσκευές που ανήκουν στην ίδια ομάδα, όπως για παράδειγμα έξυπνοι διακόπτες και έξυπνοι λαμπτήρες, θα πρέπει να μπορούν να συνεργάζονται μεταξύ τους ανεξαρτήτως τεχνολογίας.

3. Οι συσκευές εγκαθίστανται και αφαιρούνται on the runtime

Το πλήθος των συσκευών μπορεί να αυξομειώνεται κατά την διάρκεια εκτέλεσης του λογισμικού. Για παράδειγμα σε ένα έξυπνο σπίτι προστίθενται νέοι λαμπτήρες, κάμερες κτλ.

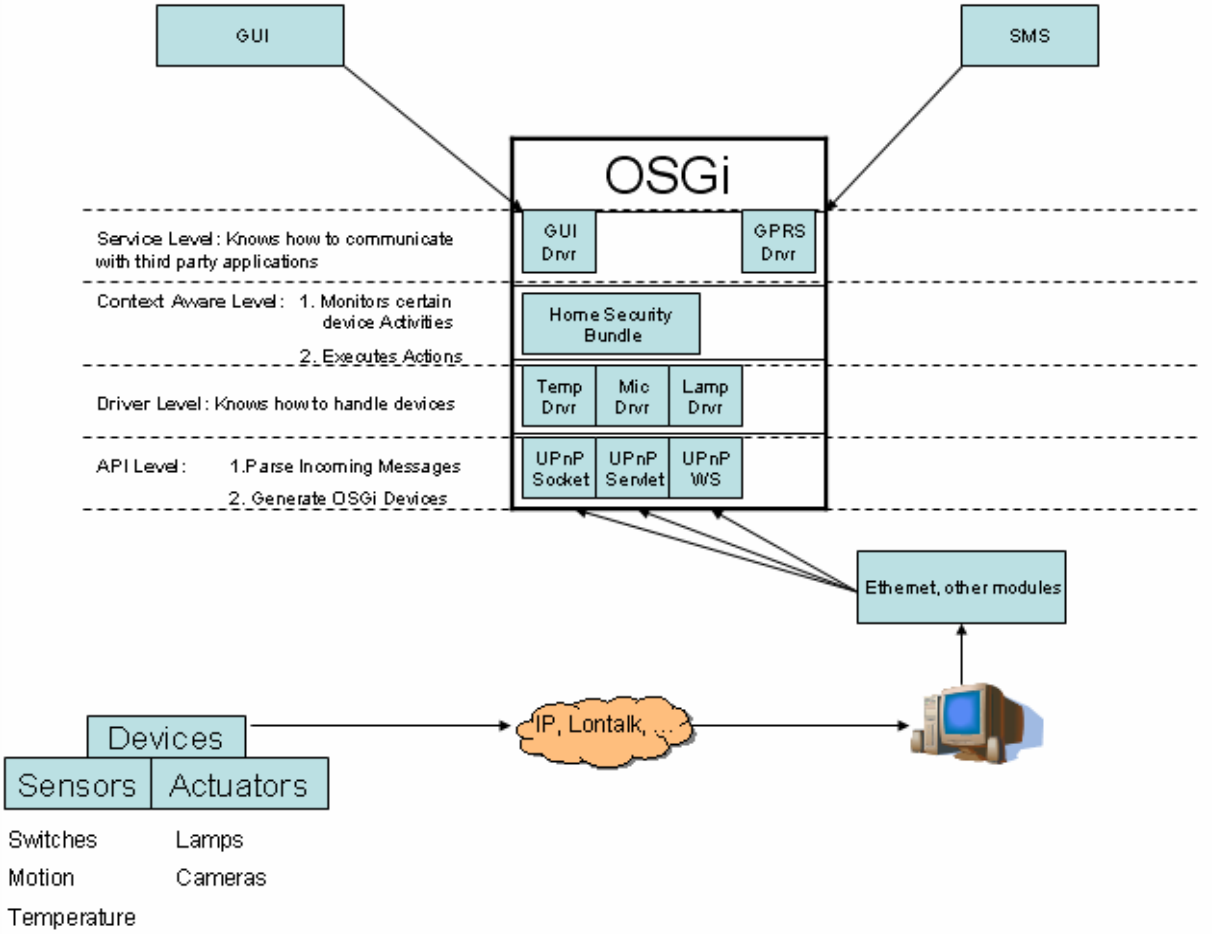
4. Νέες τεχνολογίες μπορούν εύκολα να εισαχθούν στο σύστημα

Ένα σύστημα τέτοιου τύπου για να μπορέσει να επιβιώσει στον χρόνο, θα πρέπει εκ των πραγμάτων να μπορεί να ενσωματώνει με την ελάχιστη δυνατή προσπάθεια νέες τεχνολογίες. Για παράδειγμα, η εισαγωγή έξυπνων λαμπτήρων μια τεχνολογίας Γ, θα πρέπει να μπορεί να γίνει από τους διαχειριστές του συστήματος με την ελάχιστη δυνατή προσπάθεια.

5. Συνεργασία με τον έξω κόσμο

Το σύστημα θα πρέπει να έχει αναπτυχθεί με τέτοιο τρόπο ώστε η επικοινωνία του με εφαρμογές τρίτων να μπορεί να γίνει με απλό και πλήρη τρόπο. Για παράδειγμα ένα νέο User Interface, δεν θα πρέπει να απαιτεί πολύ προσπάθεια να για να εγκατασταθεί.

Γενικά το σύστημα με μια πρώτη ματιά, φαίνεται να έχει την ακόλουθη μορφή από υψηλό επίπεδο:



Εικόνα 2.1: Η Αρχιτεκτονική του συστήματος

2.2 Επίτευξη στόχων

Σε προηγούμενο κεφάλαιο έχουν αναφερθεί μερικές από τις δυνατότητες της τεχνολογίας OSGi. Στο σημείο αυτό, παρατηρώντας τις απαιτήσεις του συστήματος, αρχίζει και γίνεται εμφανές γιατί επιλέχθηκε αυτή η τεχνολογία σαν το πλαίσιο της αρχιτεκτονικής του όλου συστήματος.

Ένα από κύρια χαρακτηριστικά της τεχνολογίας OSGi είναι η εγκατάσταση και η αφαίρεση υπηρεσιών, όπως και η δυνατότητα εύρεσης υπηρεσιών που δρουν σε αυτό. Με άλλα λόγια πολύ εύκολα μπορεί να αντιμετωπιστεί η **απαίτηση 3**, εάν υποθέσουμε ότι οι συσκευές παρουσιάζονται σαν υπηρεσίες.

Προσθέτοντας σε αυτό την δυνατότητα του OSGi να μπορεί να εισάγει bundles "on the runtime", μπορούμε εύκολα να εισάγουμε νέες τεχνολογίες (**απαίτηση 4**), δηλαδή νέα driver bundles. Επίσης με παρόμοιο τρόπο επιτυγχάνεται και η **απαίτηση 5**, εγκαθιστώντας δηλαδή νέα bundles που μπορούν να επικοινωνούν με εξωτερικές εφαρμογές.

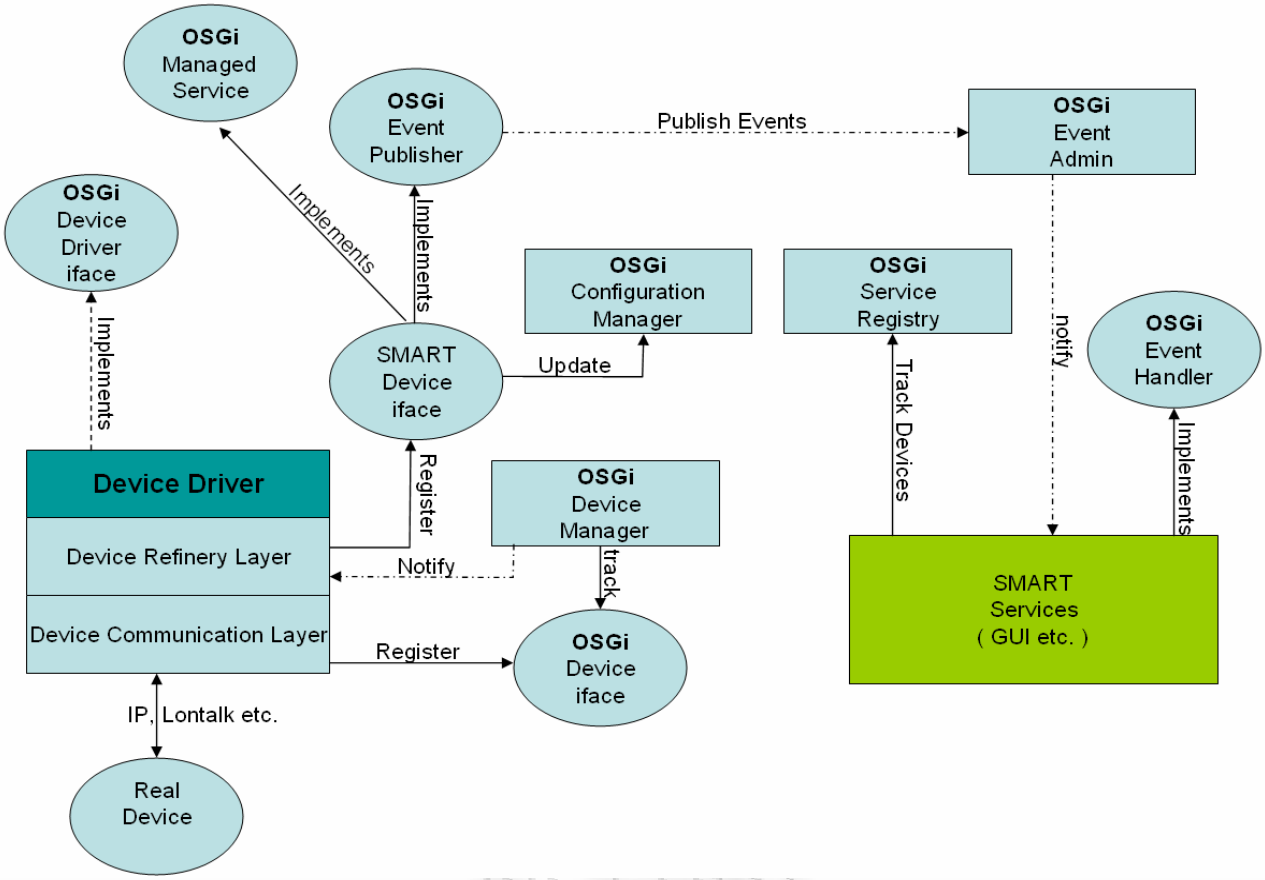
Επιπρόσθετα, το OSGi (r4 και έπειτα) παρέχει την δυνατότητα της υπηρεσίας γεγονότων, (Event Admin Service). Η υπηρεσία αυτή θα χρησιμοποιηθεί για την συνεργασία των υπηρεσιών-συσκευών που δρουν μέσα στο OSGi, ώστε να επιτευχθεί και η **απαίτηση 2**.

Τέλος, η **απαίτηση 1**, από την οποία απορρέει τελικά και η επίτευξη της απαίτησης 2, έχει να κάνει με την δημιουργία ενός κοινού Interface για συσκευές ίδιου τύπου.

Στα κεφάλαια που ακολουθούν γίνεται μια πλήρης ανάλυση των επιμέρους στοιχείων του λογισμικού που δημιουργήθηκε, ώστε τα παραπάνω να γίνουν ευκολότερα κατανοητά.

2.3 Ανάλυση της αρχιτεκτονικής

Στην ενότητα αυτή θα παρουσιαστεί αναλυτικά η αρχιτεκτονική του συστήματος. Στην εικόνα που ακολουθεί, παρουσιάζονται οι σχέσεις μεταξύ των οντοτήτων που αλληλεπιδρούν με το OSGi Framework. Έτσι θα γίνει μια αρχική παρουσίαση ώστε ο αναγνώστης να μπορεί να έχει μια αρχική άποψη για τα βασικά στοιχεία της αρχιτεκτονικής, όπως για παράδειγμα τα SMART Devices, SMART Drivers, SMART Services κτλ. και πώς αυτά συνδέονται με βασικά στοιχεία του OSGi Framework όπως το Service Registry, Configuration Manager, Device Manager και Event Admin. Στις ενότητες που θα ακολουθήσουν θα γίνει αναλυτική αναφορά σε κάθε μια από αυτές τις ενότητες που αναπτύχθηκαν.



Εικόνα 2.2: Γενική Ανάλυση Επιπέδου OSGi

Στην εικόνα παρατηρούμε σε υψηλό επίπεδο ανάλυσης τον τρόπο με τον οποίο έχει σχεδιαστεί να γίνεται οι επικοινωνία μεταξύ των επιμέρους υπηρεσιών του συστήματος, μέσα από την πλατφόρμα OSGi.

Η οντότητα **Device Driver**, αποτελεί μια υπηρεσία, η οποία έχει σαν σκοπό να επικοινωνεί με τις πραγματικές συσκευές που δρουν σε ένα Smart Space. Βρίσκεται ανάμεσα στον πραγματικό κόσμο και στον κόσμο της πλατφόρμας OSGi. Η υπηρεσία αυτή γνωρίζει τις ιδιαιτερότητες τις κάθε πραγματικής συσκευής και τον τρόπο επικοινωνίας με αυτή.

Για παράδειγμα η επικοινωνία μπορεί να γίνεται χρησιμοποιώντας το πρωτόκολλο IP με την χρήση Sockets και μιας γλώσσας XML με τις συσκευές. Όπως για παράδειγμα με μια συσκευή UPnP.

Η υπηρεσία αυτή, για κάθε νέα συσκευή η οποία εισέρχεται στο σύστημα, δημιουργεί

μια εγγραφή στο **OSGi registry** για αυτή την συσκευή, σαν τύπο **OSGi Device**. Ομοίως, διαγράφει μια έγγραφη για κάθε συσκευή που εγκαταλείπει το Smart Space.

Η υπηρεσία **Device Manager** που ορίζεται στο OSGi [1] (r3, κεφάλαιο 11, σελίδα 223), έχει σαν σκοπό να συγκεντρώνει OSGi Devices και κάθε τέτοια συσκευή να την παραδώσει στον κατάλληλο OSGi Driver. Η υπηρεσία Device Driver της πλατφόρμας μας, υλοποιεί την διεπαφή OSGi Device Driver, συνεπώς και δέχεται ειδοποιήσεις από τον OSGi Device Manager για τις νέες ή διαγραμμένες συσκευές.

Στο επίπεδο αυτό, ο Device Driver μας, δρα σαν το διυλιστικό επίπεδο της πλατφόρμας μας. Το επίπεδο δηλαδή το οποίο αποκρύπτει από τα παραπάνω στρώματα τις τεχνολογικές ιδιαιτερότητες κάθε συσκευής. Το κάνει αυτό χρησιμοποιώντας το προηγούμενο επίπεδο που καλύπτει τον τομέα της επικοινωνίας με την συσκευή, και παρουσιάζοντας την συσκευή αυτή σαν μια **SMART Device**.

2.3.1 SMART Devices

Το επίπεδο λοιπόν των Device Drivers, τελειώνει με την εγγραφή SMART Devices στο OSGi registry. Σαν SMART Devices, ορίζουμε τις υπηρεσίες της πλατφόρμας, οι οποίες υλοποιούν μια τουλάχιστον **SMART Interface**. Υπάρχει μια τέτοια διεπαφή για κάθε τύπου συσκευής που θέλουμε να χρησιμοποιεί η πλατφόρμα. Για παράδειγμα, για SMART Lights, SMART Switches, SMART Cameras κτλ. Οι διεπαφές αυτές προσφέρουν έναν κοινό τρόπο διαχείρισης και παρουσίασης των συσκευών στα ανώτερα στρώματα. Σε παρακάτω κεφάλαια θα αναλυθούν οι διεπαφές που έχουν δημιουργηθεί για την πλατφόρμα, οι οποίες φυσικά μπορούν να εμπλουτιστούν.

Συνεπώς, στο σημείο που έχει δημιουργηθεί μια SMART συσκευή, άλλες υπηρεσίες, χρησιμοποιώντας δεδομένες μεθόδους του OSGi, μπορούν να τις ανακτήσουν και να τις διαχειριστούν κατά βούληση. Επιπρόσθετα, μια SMART συσκευή, σαν υπηρεσία OSGi που είναι, μπορεί να ψάξει με την σειρά της και να διαχειριστεί, άλλες υπηρεσίες ή άλλες SMART συσκευές. Συνεπώς, έχουμε καταφέρει να αποκρύψουμε τις τεχνολογικές ιδιαιτερότητες κάθε συσκευής, ενώ οι συσκευές μπορούν να αλληλεπιδρούν μεταξύ τους.

Επιπρόσθετα, μια SMART Device, χρησιμοποιεί την υπηρεσία Event Admin του OSGi [2] (r4-mobile*, Παράγραφος 113, σελίδα 187), ώστε οι υπόλοιπες υπηρεσίες του OSGi, να μπορούν να δεχθούν γεγονότα που έχουν να κάνουν με την ενημέρωση κάποιων χαρακτηριστικών κάθε συσκευής. Για παράδειγμα, όταν ένας διακόπτης πατηθεί, ένα Event θα δημιουργηθεί και θα ενημερωθούν οι ενδιαφερόμενοι.

Τέλος, οι SMART συσκευές μπορούν προαιρετικά να χρησιμοποιούν την υπηρεσία Configuration Admin του OSGi [1] (r3, κεφάλαιο 10, σελίδα 181) ώστε να μπορούν να αλλάζουν On the runtime, την τιμή διαφόρων ιδιοτήτων τους. Για παράδειγμα, για μια υπηρεσία Sever, να μπορεί να αλλάξει δυναμικά η IP διεύθυνση ή η πόρτα κτλ.

* Η πλατφόρμα που αναπτύχθηκε χρησιμοποιεί την υλοποίηση R3 του OSGi λόγω της εξοικείωσης του συγγραφέα από τον χώρο εργασίας του. Στην έκδοση αυτή δεν υπήρχε η υπηρεσία Event Admin, κάτι που προστέθηκε στην επόμενη έκδοση R4. Παρόλα αυτά, υπάρχει και χρησιμοποιήθηκε, μια έκδοση της υπηρεσίας αυτής, που αναπτύχθηκε ειδικά για την έκδοση R3 του OSGi. Περισσότερες πληροφορίες σχετικά: http://concierge.sourceforge.net/bundles/.

2.3.2 SMART Services

Σαν SMART Services, ορίζουμε τις υπηρεσίες του OSGi, οι οποίες χρησιμοποιούν συσκευές τύπου SMART. Οι υπηρεσίες αυτές χρησιμοποιούν το OSGi registry για την εύρεση SMART Devices μέσα από δεδομένες διαδικασίες του OSGi. Επιπρόσθετα, μπορούν προαιρετικά να υλοποιούν την διεπαφή Event handler του OSGi, για να ενημερώνονται για αλλαγές σε ιδιότητες των συσκευών.

Στην πλατφόρμα που υλοποιήθηκε, το GUI χρησιμοποιεί το OSGi registry για την ανάκτηση και παρουσίαση των συσκευών. Επίσης, η υπηρεσία bridging που αναπτύχθηκε, χρησιμοποιεί την υπηρεσία Event Admin για την ενημέρωση πάνω στα χαρακτηριστικά των συσκευών. Οι υπηρεσίες αυτές θα αναλυθούν σε κεφάλαια που ακολουθούν.

Κεφάλαιο 3ο

Ανάλυση των SMART διεπαφών

Όπως έχει προαναφερθεί, όλη η απόκρυψη των τεχνολογικών ιδιαιτεροτήτων των συσκευών που δρουν στο Smart Space περιβάλλον μας, γίνεται με την χρήση κατάλληλων SMART διεπαφών. Αυτές έχουν σαν σκοπό να παρέχουν στα ανώτερα στρώματα υπηρεσίες με δεδομένες μεθόδους προσπέλασης των χαρακτηριστικών κάθε συσκευής. Για παράδειγμα, ένας λαμπτήρας, ανεξαρτήτως τεχνολογίας, έχει κάποια δεδομένα χαρακτηριστικά, π.χ. πρέπει με κάποιον τρόπο να ανάβει και να σβήνει. Τα χαρακτηριστικά αυτά αντικατοπτρίζονται στις SMART Interfaces.

Επιπρόσθετα, αναφέρθηκε ότι οι SMART διεπαφές χρησιμοποιούν την υπηρεσία Event Admin του OSGi, προς ενημέρωση αλλαγής των χαρακτηριστικών τους στις άλλες υπηρεσίες του Framework. Κάποια από τα χαρακτηριστικά που ακολουθούν Events τέτοιου τύπου, είναι κοινά για όλα τα SMART Interfaces, ενώ κάποια άλλα ορίζονται από την κάθε διεπαφή. Στο κεφάλαιο αυτό λοιπόν, θα γίνει πλήρης ανάλυση των Events αυτού του τύπου, ώστε οι ενδιαφερόμενοι να μπορούν να αναπτύξουν εύκολα και γρήγορα υπηρεσίες που θα τα χρησιμοποιούν.

3.1 unipi.smart.service.SMARTDevice

Η διεπαφή αυτή προορίζεται ώστε να δώσει κοινή υπόσταση μεταξύ όλων των SMART Devices εντός του Framework. Περιλαμβάνει τρεις βασικές μεθόδους που πρέπει να έχει κάθε συσκευή τέτοιου τύπου και παρουσιάζονται στον παρακάτω πίνακα.

Method Name	Return type	Attrs	Remarks
getID	String	-	Αυτή η μέθοδος χρησιμοποιείται για να επιστρέψει έναν κωδικό, μοναδικό μεταξύ των SMART Devices. Δεν πρέπει να είναι ποτέ null και το μέγεθός του να είναι μεγαλύτερο του μηδενός
getType	int	-	Επιστρέφει έναν ακέραιο που δηλώνει τον τύπο της

			συσκευής. Το πεδίο τιμών που επιστρέφονται εδώ ορίζεται στο interface unipi.smart.service.Constants
fireEvent	void	Dictionary	Κάνει publish ένα Event στην υπηρεσία EventAdmin του OSGi. Στην παράμετρο τύπου Dictionary, περνάνε όλα τα δεδομένα που είναι απαραίτητα σε τρίτες οντότητες να αντιληφθούν πλήρως το Event. Το topic που χρησιμοποιείται είναι το unipi.smart.service.Constants.SMART_EVENT_TOPIC

Πίνακας 3.1: Μέθοδοι του SMARTDevice Interface

3.2 unipi.smart.service.Constants

Η διεπαφή αυτή, τύπου abstract class, έχει σαν στόχο να παρέχει κοινά χαρακτηριστικά μεταξύ των SMART Devices, όπως και Utility μεθόδους σε αυτά τα χαρακτηριστικά. Στην συνέχεια παρουσιάζονται οι μέθοδοι που ορίζει όπως και τα final static πεδία.

Method Name	Return type	Attrs	Remarks
getTypeNamely	String	int	Αποτελεί μια Utility μέθοδο που έχει σαν σκοπό για τον δεδομένο ακέραιο που δέχεται σαν παράμετρο, που αποτελεί το Device type όπως συσκευής όπως ορίζεται σε αυτή την κλάση, να επιστρέψει ένα φιλικό όνομα για αυτόν τον τύπο.

Πίνακας 3.2: Μέθοδοι του Constants Interface

Field Name	Type	Field value	Remarks
PROP_UPDATE_TOPIC	String	unipi/smart/ service/smartdevice/ update	Το Event Topic κάτω από το οποίο πρέπει να γίνονται publish όλα τα Events των SMART Devices αναφορικά με updates των χαρακτηριστικών τις.
DEV_ID_ATTR	String	SMART_DEVICE_ID_ATTR	Αποτελεί το property name του χαρακτηριστικού του Event που δηλώνει ποιο SMART Device το έκανε publish.
DEV_TYPE_ATTR	String	SMART_DEVICE_TYPE	Αποτελεί το property name του χαρακτηριστικού του Event που δηλώνει τον τύπο του SMART Device που το έκανε publish.
TYPE_SENSOR	int	1	Δηλώνει ένα SMART Device τύπου Sensor, π.χ τις διακόπτης, αισθητήρας κίνησης κτλ.
TYPE_LIGHT	int	2	Δηλώνει ένα SMART Device τύπου Light, π.χ. τις λαμπτήρας.
TYPE_CAMERA	int	3	Δηλώνει ένα SMART Device τύπου Camera.
Types	String[3]	Τα προηγούμενα 3 Contants	Χρησιμοποιείται για indexed προσπέλαση των Device Types.

Πίνακας 3.3: Πεδία του Constants Interface

3.3 unipi.smart.service.SwitchIface

Η διεπαφή αυτή ορίστηκε για την παρουσίαση και διαχείριση συσκευών τύπου διακόπτη. Παρέχει τα κατάλληλα πεδία και τις μεθόδους που απαιτούνται για την διαχείριση τέτοιου τύπου συσκευών. Οι μέθοδοι και τα πεδία παρουσιάζονται αμέσως μετά.

Field Name	Type	Field value	Remarks
STATE_OFF	int	0	Η τιμή του πεδίου αυτού δηλώνει ότι ο διακόπτης είναι κλειστός
STATE_ON	int	1	Η τιμή του πεδίου αυτού δηλώνει ότι ο διακόπτης είναι ανοιχτός
CURRENT_STATE	String	CUR_STATE_ATTR	Το property name που χρησιμοποιείται στο Event για να δηλώσει την κατάσταση του διακόπτη. Παίρνει τιμές STATE_ON ή STATE_OFF

Πίνακας 3.4: Πεδία του SwitchIface Interface

Method Name	Return type	Attrs	Remarks
turnOn	void	void	Χρησιμοποιείται για να θέσει τον διακόπτη σε STATE_ON
turnOff	void	void	Χρησιμοποιείται για να θέσει τον διακόπτη σε STATE_OFF

getState	int	void	Χρησιμοποιείται για την προσπέλαση της παρούσας κατάστασης του διακόπτη

Πίνακας 3.5: Μέθοδοι του SwitchIface Interface

3.4 unipi.smart.service.LightIface extends SwitchIface

Η διεπαφή αυτή έχει δημιουργηθεί για να καλύψει της απαιτήσεις σε συσκευές τύπου λαμπτήρα. Παρατηρούμε ότι συμπεριλαμβάνει όλα τα στοιχεία της διεπαφής SwitchIface που παρουσιάστηκε νωρίτερα. Τα νέα στοιχεία που παρουσιάζονται σε αυτή είναι τα επόμενα:

Field Name	Type	Field value	Remarks
CURRENT_LEVEL	String	CUR_LEVEL_ATTR	Το property name που χρησιμοποιείται στο Event για να δηλώσει το επίπεδο φωτισμού του λαμπτήρα. Το πεδίο τιμών του πρέπει να είναι μεταξύ 0-100.

Πίνακας 3.6: Πεδία του LightIface Interface

Method Name	Return type	Attrs	Remarks
setLevel	void	int	Χρησιμοποιείται για να θέσει ένα νέο επίπεδο φωτεινότητας στον λαμπτήρα. Το πεδίο τιμών της παραμέτρου πρέπει να είναι 0-100

getLevel	int	void	Χρησιμοποιείται για να ανακτήσει το επίπεδο φωτεινότητας ενός λαμπτήρα.

Πίνακας 3.7: Μέθοδοι του LightIface Interface

3.5 unipi.smart.service.CameraIface extends SwitchIface

Η διεπαφή αυτή δημιουργήθηκε για της ανάγκες παρουσίασης συσκευών τύπου Surveillance Camera. Περιέχει όλες τις παραμέτρους και μεθόδους του SwitchIface. Τα νέα στοιχεία που προστίθενται είναι τα εξής:

Field Name	Type	Field value	Remarks
PRESENTATION_URL	String	presentation_url	Το πεδίο αυτό αποτελεί το property name του στοιχείου του Event που περιέχει την πληροφορία για το URL που παρουσιάζει το GUI της συσκευής (εάν υπάρχει)
STILL_IMG_URL	String	Still_img_url	Το πεδίο αυτό αποτελεί το property name του στοιχείου του Event που περιέχει την πληροφορία για το URL που περιέχει τα snap shots της κάμερας ανά τακτά χρονικά διαστήματα

Πίνακας 3.8: Πεδία του CameraIface Interface

Method Name	Return type	Attrs	Remarks
getPresentationUrl	URL	void	Επιστρέφει το presentation URL της κάμερας. Αν δεν υπάρχει ή αν η κάμερα είναι εκτός λειτουργίας επιστρέφεται null
getStillImageUrl	URL	void	Επιστρέφει το Still Image URL της κάμερας. Αν η κάμερα είναι εκτός λειτουργίας επιστρέφεται null

Πίνακας 3.9: Μέθοδοι του CameraIface Interface

Στο σημείο αυτό ολοκληρώθηκε η παρουσίαση των διεπαφών που ορίζουν τα SMART Devices της πλατφόρμας. Στον παρακάτω πίνακα για λόγους ευκολίας, παρουσιάζονται συγκεντρωτικά τα χαρακτηριστικά που πρέπει να υπάρχουν σε ένα SMART Event, δηλαδή με topic unipi.smart.service.Constants.PROP_UPDATE_TOPIC.

Iface/ Property	DEV_ID _ATTR	DEV_TYPE _ATTR	CURRENT _LEVEL	CURRENT _STATE	PRESENTATION _URL	STILL_IMG _URL
SwitchIface	X	X		X		
LightIface	X	X	X	X		
CameraIface	X	X		X	X	X

Πίνακας 3.10: Σύνολο χαρακτηριστικών Smart Interfaces

Κεφάλαιο 4ο

Ανάλυση του Συστήματος

Σε προηγούμενα κεφάλαια σχολιάστηκαν οι στόχοι του συστήματος και αναλύθηκαν τα βασικά σημεία της αρχιτεκτονικής του ώστε να μπορεί ο αναγνώστης να δημιουργήσει τις δικές του εφαρμογές στην πλατφόρμα χρησιμοποιώντας αυτή την αρχιτεκτονική.

Για την ολοκλήρωση αυτής της εργασίας, υλοποιήθηκαν υπηρεσίες βασισμένες στην αρχιτεκτονική που αναπτύχθηκε. Στο κεφάλαιο αυτό θα αναλυθούν, εν συντομία, τα bundles που αναπτύχθηκαν, προς παραδειγματισμό για τους ενδιαφερομένους. Σε επόμενα κεφάλαια θα δοθούν και παραδείγματα λειτουργίας του συνόλου της εφαρμογής, ώστε να γίνει αντιληπτός και με οπτικά μέσα ο τρόπος λειτουργίας του συστήματος.

Εισαγωγικά αναφέρεται πως έχουν δημιουργηθεί οι παρακάτω υπηρεσίες, που δρουν άμεσα στην αρχιτεκτονική και τον σχεδιασμό του συστήματος. Οι οντότητες αυτές είναι:

- Driver Bundle για virtual IP devices
- Refinement Bundles για virtual IP devices (Lights, switches, cameras)
- Driver Bundle για UPnP Devices
- Refinements Bundle για UPnP φώτα.
- User Interface Bundle
- Inter-Device Interaction Service Bundle

Επιπρόσθετα, εκτός των bundles που δρουν άμεσα με την αρχιτεκτονική του συστήματος, αναπτύχθηκαν και βοηθητικά bundles για την ολοκλήρωση του συστήματος. Αυτά επιγραμματικά είναι τα παρακάτω:

- CasInit Bundle. Bundle/Configuration manager
- Virtual IP Device manager

Τέλος, λόγω έλλειψης πραγματικών συσκευών UPnP, χρησιμοποιήθηκαν τα UPnP tools της Intel. Ο τρόπος χρήσης τους φαίνεται σε επόμενο κεφάλαιο, όπου γίνεται επίδειξη του τρόπου χρήσης του συστήματος. Για περισσότερες πληροφορίες απόκτησης τους κτλ. δίνονται στο παράρτημα 2.

4.1 IP Device Driver

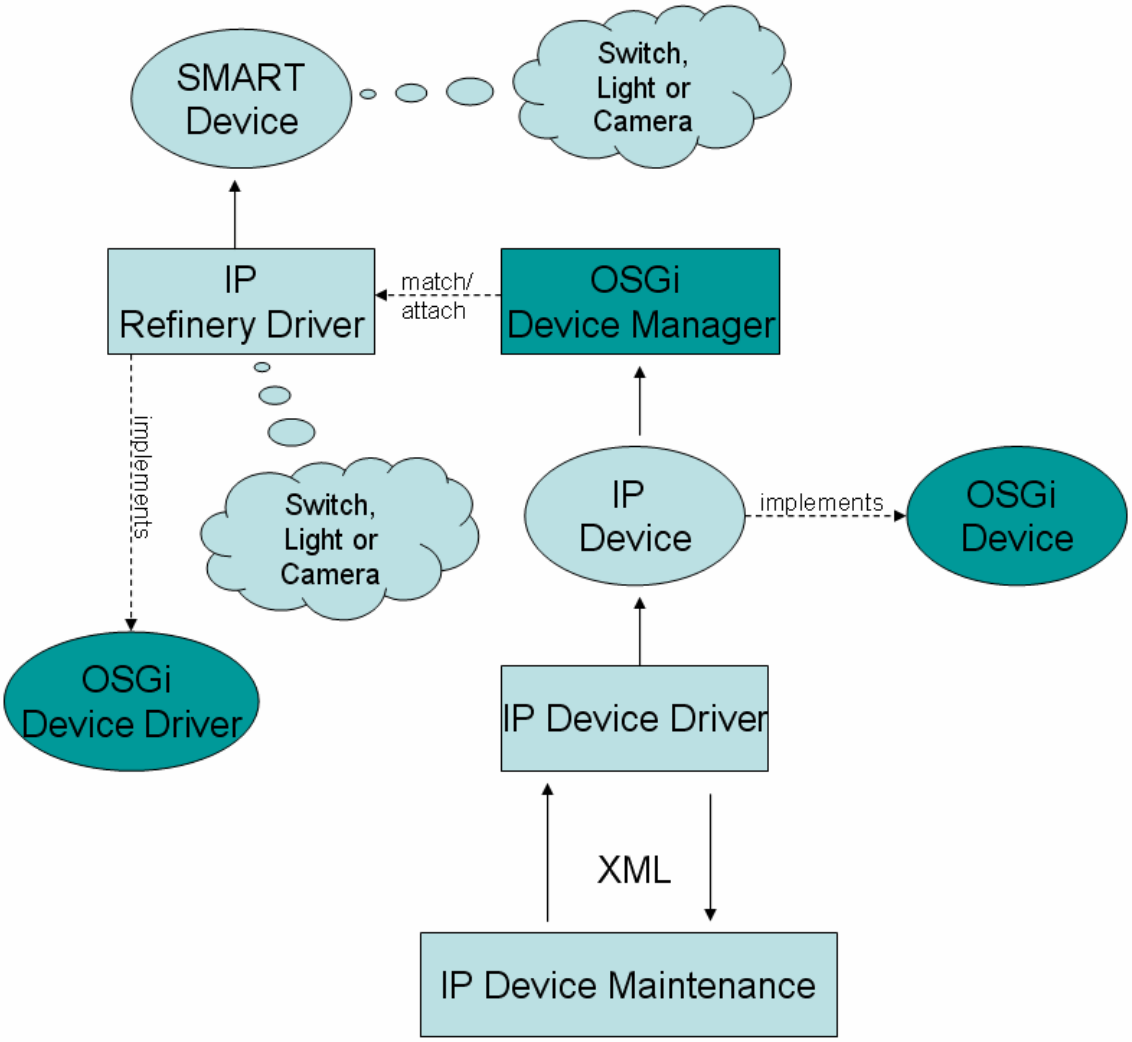
Η υπηρεσία αυτή δημιουργήθηκε για την διαδικασία της ανάπτυξης και ελέγχου της πλατφόρμας. Αποτελείται από μια σειρά bundles και μια εξωτερική του OSGi εφαρμογή.

Αναφορικά με τα Driver bundles, ο σκοπός τους είναι τελικά να παρουσιάσουν στο σύστημα, SMART Devices. Εν γένει, αυτά αποτελούνται από ένα bundle που αποτελεί τον **IP Base Driver**, σκοπός του οποίου είναι να επιμελείται τα καθήκοντα επικοινωνίας μέσω ενός καναλιού IP με της συσκευές που είναι συνδεδεμένες στο δίκτυο. Επικοινωνεί μαζί τους με ένα καλά καθορισμένο πρωτόκολλο και παρουσιάζει στο OSGi Framework, υπηρεσίες τύπου **IP Devices.**

Στην συνέχεια αυτές οι υπηρεσίες, συγκεντρώνονται από τους **IP Refinement Drivers**. Σκοπός αυτών είναι, ανάλογα με τα ιδιαίτερα δεδομένα κάθε IP Device, να παρουσιάσουν στο OSGi Framework τα ανάλογα SMART Devices. Αναπτύχθηκαν τρεις διαφορετικοί IP Refinement Drivers για την υποστήριξη φώτων, διακοπτών και καμερών αντιστοίχως. Στην συνέχεια τα τελευταία, χρησιμοποιούνται με καλά ορισμένους τρόπους από άλλες SMART Services του Framework.

Στο σημείο αυτό, αξίζει να σημειωθεί, πως η επιλογή του κατάλληλου Refinement Driver για το κάθε IP Device που δημιουργείται από τον IP Base Driver, γίνεται με την χρήση της υπηρεσίας **Device Manager** του OSGi. Σκοπός αυτής τις υπηρεσίας αυτής, είναι η εύρεση του καλύτερα εξειδικευμένου Driver στο Framework για κάθε Device, με βάση ένα πρωτόκολλο που ορίζεται στο OSGi Specification. Περισσότερες πληροφορίες μπορούν να βρεθούν στο OSGi [1] r3, κεφάλαιο 11, σελίδα 223.

Όπως αναφέρθηκε, ο IP Base Driver επικοινωνεί με τις IP Devices στο δίκτυο. Για την υλοποίηση της διπλωματικής δεν ήταν δυνατό να συγκεντρωθεί ένας μεγάλος αριθμός από τέτοιες διαφορετικού τύπου συσκευές, λόγω κόστους. Συνεπώς, μια εξωτερική οντότητα από το OSGi αναπτύχθηκε, με σκοπό την παρουσίαση εικονικών τέτοιων συσκευών στο δίκτυο. Η εφαρμογή αυτή ονομάζεται **IP Device Maintenance**, και γνωρίζει το πρωτόκολλο επικοινωνίας που χρησιμοποιεί ο IP Base Driver με τις IP Devices. Στην συνέχεια παρουσιάζεται μια εικόνα που συγκεντρώνει τα παραπάνω. Έπειτα ακολουθεί η παρουσίαση του πρωτοκόλλου του IP Base Driver, όπως και η παρουσίαση χρήσης της εξωτερικής οντότητας.



Εικόνα 4.1: Αρχιτεκτονική IP Drivers

Μερικά επιπλέον χρήσιμα στοιχεία για τον IP Base Driver, είναι ότι χρησιμοποιεί αρχιτεκτονική Client-Sever. Συγκεκριμένα, αυτό αποτελεί την πλευρά του **Sever** ακούγοντας στην πόρτα **8888** για αιτήσεις. Οι αιτήσεις πρέπει να ακολουθούν το πρωτόκολλο που περιγράφεται στην συνέχεια.

4.1.1 Ανάλυση πρωτοκόλλου επικοινωνίας

Αρχικά παρατίθεται το XML Schema των μηνυμάτων που θα αποστέλλονται μεταξύ Client – Server.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

<xs:element name="smart">
	<xs:complexType>
		<xs:element name="msg" minOccurs="1">
			<xs:attribute name="type" type="xs:string" use="required"/>
			<xs:attribute name="propName" type="xs:string" use="required"/>
			<xs:attribute name="propValue" type="xs:string" use="required"/>
		</xs:element>
	</xs:complexType>
</xs:element>
```

Ακολουθούν δύο τέτοιου τύπου μηνύματα σαν παραδείγματα:

```
<smart><msg type="hello" propName="type" propValue="LIGHT"/></smart>
<smart><msg type="update" propName="level" propValue="50"/></smart>
```

Το πρωτόκολλο που πρέπει να ακολουθείται είναι εξαιρετικά απλό. Εν συντομία, οι συσκευές που επιθυμούν να ενταχθούν στην πλατφόρμα, εγκαθιδρύουν μία σύνδεση στην πόρτα 8888 σε μία well known IP του IP Base Driver. Στην συνέχεια ανταλλάσσεται το πρώτο μήνυμα, Hello message, στο οποίο η συσκευή δηλώνει των τύπο της. Εν συνεχεία, μπορεί να στείλει

μια σειρά από Update messages, στα οποία δηλώνει την κατάσταση διαφόρων μεταβλητών της. Για παράδειγμα, ένας διακόπτης αν είναι ανοιχτός ή κλειστός. Για την απεγκατάσταση μια σύνδεσης, αρκεί η μία πλευρά να κλείσει το Socket. Επιπρόσθετα σημειώνεται ότι κάθε μήνυμα **ΠΡΕΠΕΙ** να καταλήγει με τον χαρακτήρα **"\0"**, που δηλώνει το τέλος το μηνύματος.

Ακολουθεί ένα συγκεντρωτικός πίνακας των στοιχείων κάθε τύπου μηνύματος.

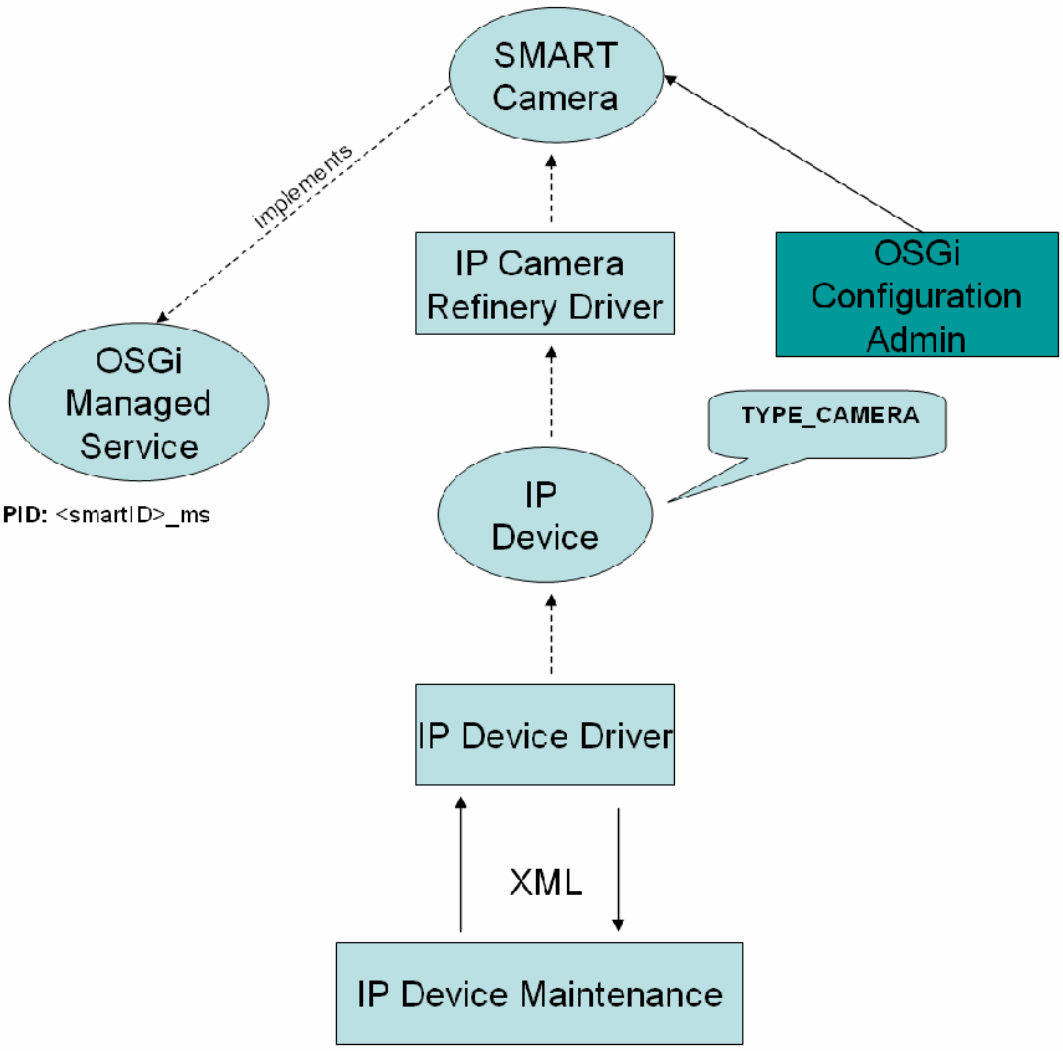
Message Type	**Message Field**	**Value Set**
Hello	type	{LIGHT, SWITCH, CAMERA}
Update (get)	state	{0,1}
Update (get)	level	Ακέραιος {0-100}

Πίνακας 4.1: Τύποι μηνυμάτων

Τέλος, ο τύπος μηνύματος **get**, μπορεί να αποστέλλεται μόνο από την πλευρά του IP Base Driver στους Clients/Devices, με σκοπό να ενημερωθεί για την συγκεκριμένη τιμή ενός property. Οι συσκευές πρέπει να απαντούν με μηνύματα τύπου update σε αυτά τα ερωτήματα, ενημερώνοντας για την τιμή του συγκεκριμένου property.

4.1.2 IP Surveillance Camera

Αυτή η παράγραφος περιγράφει συγκεκριμένα το bundle που αναπτύχθηκε για την υποστήριξη συσκευών τύπου «κάμερας παρακολούθησης» με σύνδεση IP, λόγω των ιδιαιτεροτήτων αυτής. Αμέσως μετά ακολουθεί μια εικόνα που περιγράφει την αρχιτεκτονική του bundle αυτού που στην συνέχεια θα αναλυθεί.



Εικόνα 4.2: Αρχιτεκτονική IP Camera

Όπως κάθε IP Refinement Driver που έχει αναπτυχθεί, έτσι και ο IP Camera Refinement Driver, χρησιμοποιεί IP Devices που εγγράφονται στο Framework από τον IP Device Driver. Για κάθε συσκευή που ταιριάζει σε αυτόν, με βάση τον Device Manager, θα δημιουργηθεί ένα SMART Device, που θα υλοποιεί το SMART interface, CameraIface .

Επιπρόσθετα, κάθε τέτοια υπηρεσία, όπως φαίνεται και στο σχήμα, υλοποιεί το OSGi interface Managed Service. Με άλλα λόγια, χρησιμοποιώντας την υπηρεσία Configuration Admin του OSGi, μπορούν να αλλάξουν οι τιμές σε κάποια properties του on the runtime. Το PID που χρησιμοποιείται για την εγγραφή της κάθε συσκευής στο Framework είναι της

μορφής: **<SMART_ID>_ms**.

Εν γένει, κάθε IP Surveillance Camera, έχει τα εξής δύο στοιχεία:

- Ένα URL στο οποίο παρουσιάζεται η εικόνα λαμβάνει κάθε στιγμή η Camera (still images)
- Ένα URL στο οποίο παρουσιάζονται χειριστήρια για την Camera.

Σε περιπτώσεις που η συσκευή υποστηρίζει UPnP, τα στοιχεία αυτά πρέπει να παρουσιάζονται στο UPnP announcement και να λαμβάνονται αυτόματα από τους ενδιαφερόμενους. Στην περίπτωσή μας όμως, ο IP Camera Driver δεν αναπτύχθηκε για την διαχείριση UPnP enabled Cameras. Έτσι τα δύο αυτά σημαντικά στοιχεία για την λειτουργία του πρέπει να δίνονται σαν configuration properties. Ακολουθεί ένας πίνακας με τα properties αυτά:

Property Name	Property Type	Property Value	Meaning
presentation_url	String	HTTP URL	Το URL διαχείρισης της κάμερας.
stillimg_url	String	HTTP URL	Το URL του still image της κάμερας.

Πίνακας 4.2: IP Camera Configuration Properties

Σημείωση:

Στο σημείο αυτό σημειώστε ότι, στα URL **πρέπει** να ορίζεται και ο αριθμός της πόρτας στην οποία ακούει ο server, ακόμα και αν αυτή είναι η default 80 για το http.

4.1.3 Ανάλυση λειτουργίας του IP Device Maintenance

Όπως αναφέρθηκε, σκοπός αυτής της εφαρμογής είναι η παρουσίαση εικονικών IP Devices στον IP Base Driver της πλατφόρμας που αναπτύχθηκε. Συνεπώς, η υλοποίηση της εφαρμογής αυτής γνωρίζει το πρωτόκολλο επικοινωνίας που αναφέρθηκε παραπάνω.

Η εφαρμογή αυτή, επιτρέπει στον χρήστη την δημιουργία και διαχείριση εικονικών IP Devices, μέσα από μια διεπαφή τύπου Terminal. Ο χρήστης/διαχειριστής, μπορεί να πληκτρολογήσει μερικές πολύ απλές εντολές, ώστε είτε να δημιουργήσει μια καινούργια συσκευή, να αλλάξει τις μεταβλητές του ή και να δει την κατάσταση της καθεμίας. Στην

ενότητα αυτή θα αναλυθούν όλοι οι διαφορετικοί τύποι εντολών. Παραδείγματα φαίνονται στο κεφάλαιο που ασχολείται σε βάθος με την επίδειξη του συστήματος σαν σύνολο.

Το σύνολο των εντολών που αναγνωρίζει το σύστημα, είναι οι ακόλουθες:

- **help :** Εμφανίζει ένα μενού με την σύνταξη των εντολών.
- **exit :** Έξοδος από την εφαρμογή.
- **create <type> :** Δημιουργία μια νέας συσκευής τύπου <type>. Το σύνολο των τύπων αναφέρεται στο πρωτόκολλο επικοινωνίας.
- **ls :** Εμφανίζει μια λίστα με όλες τι εικονικές συσκευές που έχουν δημιουργηθεί.
- **remove <id> :** Αφαιρεί την συσκευή με κωδικό <id>
- **get <id> <name> :** Εμφανίζει της τιμή του property <name> για την συσκευή με κωδικό <id>. Τα πιθανά property names, αναφέρονται στο πρωτόκολλο επικοινωνίας.
- **set <id> <name> <value> :** Θέτει την τιμή <value> για το property <name> της συσκευής με κωδικό <id>. Οι τιμές ακολουθούν το πρωτόκολλο επικοινωνίας.

4.2 UPnP Device Drivers

Στα πλαίσια των απαιτήσεων της εργασίας αυτής ήταν η χρήση διαφορετικών τεχνολογιών για τους τύπους συσκευών που θα έπαιρναν μέρος στην χρήση της πλατφόρμας, ώστε να είναι εμφανής ο τρόπος όμοια διαχείρισης των συσκευών ανεξαρτήτως τεχνολογίας.

Εκτός από τις IP Devices, που ακολουθούν ένα απλό πρωτόκολλο επικοινωνίας που δημιουργήθηκε στα πλαίσια τις διπλωματικής, αναπτύχθηκαν και χρησιμοποιήθηκαν οι κατάλληλες οντότητες για την υποστήριξη συσκευών UPnP. Συγκεκριμένα η πλατφόρμα ως έχει, υποστηρίζει UPnP Lights.

Η έκδοση r3 του OSGi Specification που χρησιμοποιείται για την υλοποίηση αυτής της εργασίας, ορίζει interfaces αναφορικά με την διαχείριση συσκευών UPnP [1] (OSGi r3, κεφάλαιο 25, σελίδα 503). Τα κατάλληλα bundles που υλοποιούν τα interfaces αυτά, ανακτηθήκαν από το http://oscar-osgi.sourceforge.net/ και τα υλοποιήθηκαν και διαμοιράστηκαν για λογαριασμό της DomoWare (http://domoware.isti.cnr.it/).

Επιπρόσθετα, λόγω τις έλλειψης πραγματικών UPnP enabled συσκευών κατά την διάρκεια εκπόνησης της διπλωματικής αυτής εργασίας, χρησιμοποιήθηκαν ευρέως γνωστά και χρησιμοποιούμενα εργαλεία τα οποία δημιουργούν εικονικές UPnP συσκευές. Το λογισμικό αυτό αναπτύχθηκε και παρέχεται δωρεάν για χρήση από την εταιρία **Intel** (http://www.intel.com/cd/ids/developer/asmo-na/eng/downloads/upnp/tools/index.htm). Το λογισμικό αυτό παρέχεται και στο συνοδευτικό CD της εργασίας. Στο παράτημα 3, δίνονται περισσότερες πληροφορίες για την χρήση τους.

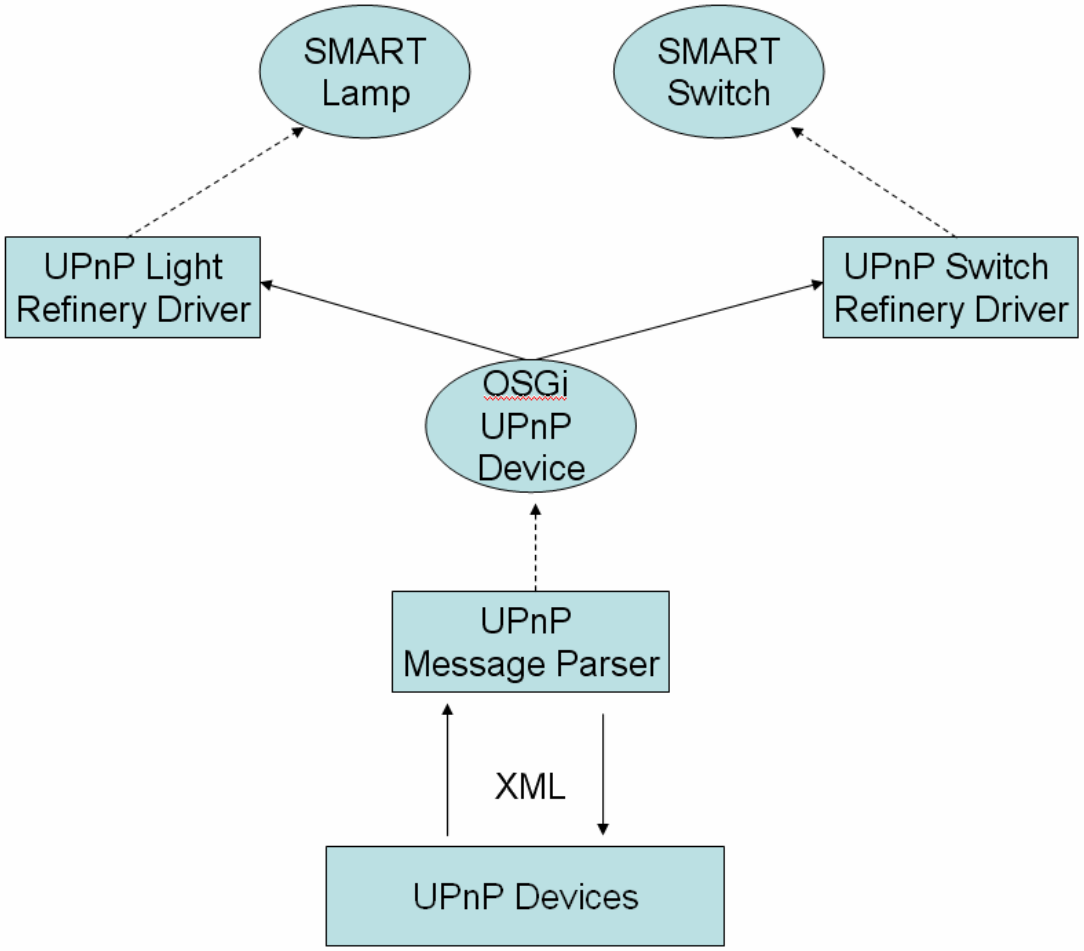
Τέλος, αναπτύχθηκε λογισμικό (Bundle) το οποίο έχει σαν σκοπό την διαχείριση UPnP Lights, με την χρήση των υπηρεσιών UPnP που παρέχονται από το OSGi Framework.

4.2.1 Αρχιτεκτονική UPnP Light Driver

Για την ανάπτυξη αυτού του λογισμικού, χρησιμοποιούνται οι εξειδικευμένες οντότητες UPnP του OSGi που αναφέρθηκαν παραπάνω.

Στην εικόνα που ακολουθεί, παρουσιάζεται η αλληλεπίδραση των υπηρεσιών αυτών με τις υπηρεσίες UPnP Refinery Drivers και συγκεκριμένα του UPnP Light Driver που αναπτύχθηκε. Η εικόνα αυτή εν συντομία παρουσιάζει τον τρόπο με τον οποίο από ένα UPnP XML μήνυμα, το οποίο αντιλαμβάνονται οι UPnP οντότητες του OSGi, καταλήγουμε στην δημιουργία μιας SMART συσκευής, την οποία μπορούν εν συνεχεία να χρησιμοποιήσουν άλλες οντότητες, όπως SMART Services, όπως θα δούμε σε επόμενα κεφάλαια.

Στην σελίδα που ακολουθεί, παρουσιάζεται η σχετική εικόνα, η οποία θα αναλυθεί αμέσως μετά.



Εικόνα 4.3: Αρχιτεκτονική UPnP Refinery Drivers

Στην εικόνα αυτή παρατηρούμε ότι τα μηνύματα που καταφθάνουν στο δίκτυο από τις συσκευές UPnP, αναλύονται από συγκεκριμένες UPnP aware οντότητες (DomoWare). Αυτές οι οντότητες έχουν σαν σκοπό την δημιουργία UPnP OSGi Devices για κάθε μια UPnP enabled συσκευή που γίνεται announce στο δίκτυο και τελικά υλοποιούν όλο το κανάλι επικοινωνίας από το OSGi Framework στις UPnP Devices.

Οι UPnP Devices εν συνεχεία, ανακτώνται από τους κατάλληλους refinery drivers και αναλόγως με τον τύπο της συσκευής, δημιουργείται στο Framework το κατάλληλο SMART Device. Για παράδειγμα SMART Lights, SMART Switches κτλ.

Στα πλαίσια αυτής της διπλωματικής αναπτύχθηκε κατάλληλος UPnP Refinery Driver, ο οποίος διαχειρίζεται UPnP Lights και τα παρουσιάζει στο OSGi Framework κάτω από το interface SMARTLight.

Στο σημείο αυτό αναφέρεται εγκυκλοπαιδικά, ότι ο τύπος και οι ιδιότητες των UPnP συσκευών και μηνυμάτων ορίζεται από το **UPnP Forum** [5](http://www.upnp.org/). Στην συνέχεια παρουσιάζεται το template ενός UPnP Light όπως αυτό είχε δοθεί από το UPnP Forum την περίοδο συγγραφής αυτής της εργασίας:

```
<?xml version="1.0"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
<specVersion>
<major>1</major>
<minor>0</minor>
</specVersion>
<URLBase>base URL for all relative URLs</URLBase>
<device>
<deviceType>urn:schemas-upnp-org:device:DimmableLight:1</deviceType>
<friendlyName>short user-friendly title</friendlyName>
<manufacturer>manufacturer name</manufacturer>
<manufacturerURL>URL to manufacturer site</manufacturerURL>
<modelDescription>long user-friendly title</modelDescription>
<modelName>model name</modelName>
<modelNumber>model number</modelNumber>
<modelURL>URL to model site</modelURL>
<serialNumber>manufacturer's serial number</serialNumber>
<UDN>uuid:UUID</UDN>
<UPC>Universal Product Code</UPC>
<iconList>
<icon>
<mimetype>image/format</mimetype>
<width>horizontal pixels</width>
```

<height>vertical pixels</height>
<depth>color depth</depth>
<url>URL to icon</url>
<icon/>
XML to declare other icons, if any, go here
</iconList>
<serviceList>
<service>
<serviceType>urn:schemas-upnp-org:service: SwitchPower:1</serviceType>
<serviceId>urn:upnp-org:serviceId: SwitchPower:1</serviceId>
<SCPDURL>URL to service description</SCPDURL>
<controlURL>URL for control</controlURL>
eventSubURL>URL for eventing</eventSubURL>
< service</>
<service>
<serviceType>urn:schemas-upnp-org:service:Dimming:1</serviceType>
<serviceId>urn:upnp-org:serviceId:Dimming:1</serviceId>
<SCPDURL>URL to service description</SCPDURL>
<controlURL>URL for control</controlURL>
<eventSubURL>URL for eventing</eventSubURL>
</service>
Declarations for other services added by UPnP vendor (if any) go here
<serviceList>
/ deviceList<>
Description of embedded devices added by UPnP vendor (if any) go here
</deviceList>
<presentationURL>URL for presentation</presentationURL>
</device>
</root>

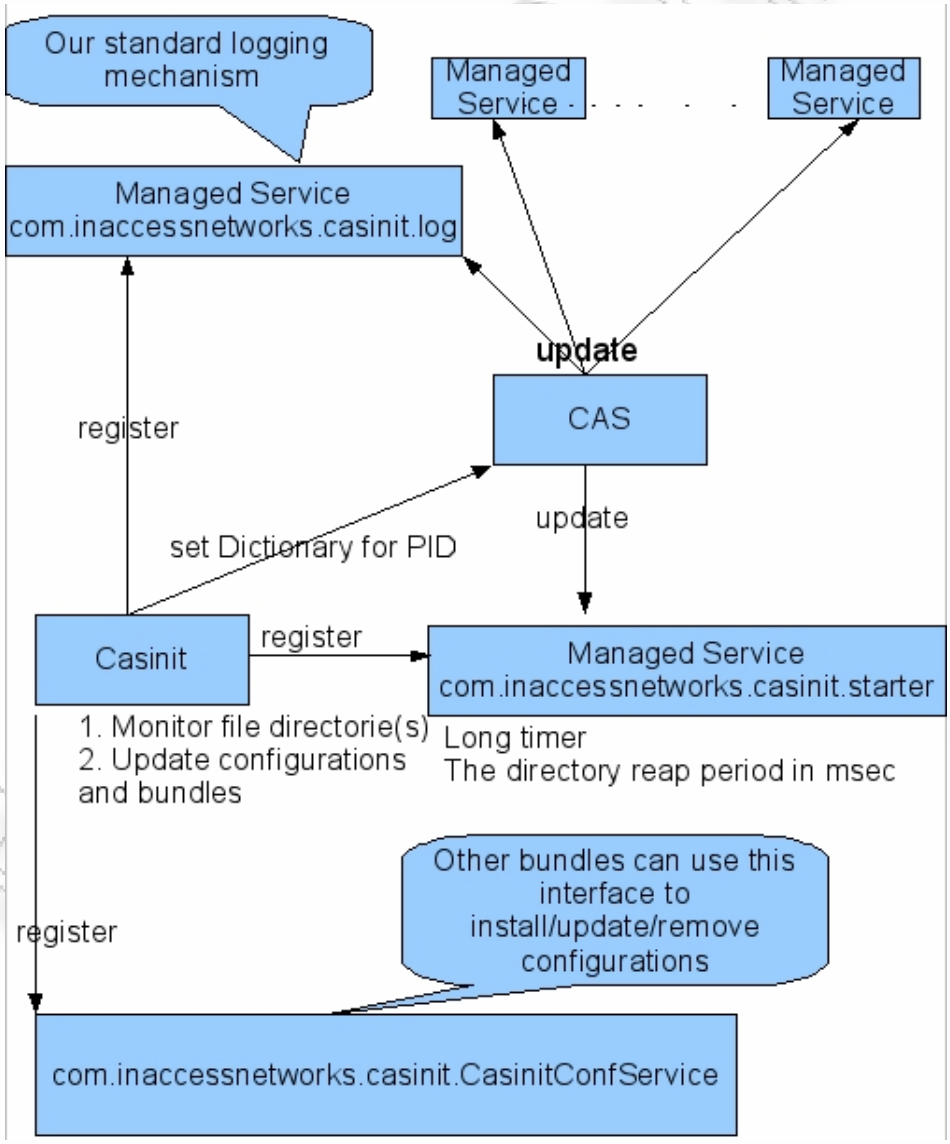
Περισσότερες πληροφορίες σχετικά στο [5]

4.3 Η Υπηρεσία CASInit

4.3.1 Σκοπός

Η υπηρεσία αυτή ανήκει στην ομάδα των βοηθητικών υπηρεσιών που χρησιμοποιήθηκαν στα πλαίσια αυτής της διπλωματικής για την διευκόλυνση της διαδικασίας ανάπτυξης, ελέγχου και χρήσης.

Σκοπός της υπηρεσίας αυτής, είναι η διαχείριση σε υψηλό επίπεδο των bundles του Framework και των Configurations που θα λαμβάνουν αυτά μέσω της υπηρεσίας Configuration Admin του OSGi. Με άλλα λόγια, η υπηρεσία αυτή διευκολύνει την διαδικασία install/remove/update των bundles και των ομοίως για τα Configurations.



Εικόνα 4.4: Αρχιτεκτονική υπηρεσίας CASinit

4.3.2 Bundle Management

Για την διαχείριση των bundles, ο CASinit παρακολουθεί έναν κατάλογο. Τα bundles που προστίθενται στον κατάλογο αυτό, γίνονται αυτόματα install και start στο default start level του Framework (όπως αυτό ορίζεται από το Framework system property "oscar.startlevel.framework") . Περισσότερες πληροφορίες για τα Framework system properties στο [9]. Επιπρόσθετα, γίνεται αυτόματα refresh το Framework αναφορικά με τις εξαρτήσεις μεταξύ βιβλιοθηκών των bundles. Αν ένα bundle αναιρεθεί από τον κατάλογο, θα γίνει αυτόματα uninstall και αν για κάποιον λόγω μεταβληθεί το timestamp του, θα γίνει αυτόματα Update. Ο κατάλογος που παρακολουθείται by default, είναι ο κατάλογος κάτω από τον οποίο εκτελείται η JVM, δηλαδή, ο τρέχων κατάλογος. Επιπρόσθετα όμως, έχει δοθεί η δυνατότητα στον χρήστη να μπορεί να δώσει αυτός την διαδρομή στον υπό παρακολούθηση κατάλογο. Αυτό μπορεί να γίνει μέσω της μεταβλητής περιβάλλοντος του OSGi με όνομα "**com.inaccessnetworks.casinit.bundle.path**" , τιμή της οποίας είναι η πλήρης διαδρομή στον κατάλογο όπου θα αποθηκεύονται τα bundles.

Επιπλέον, δίνεται η δυνατότητα στον χρήστη, να μπορεί να παρακάμψει την προκαθορισμένη συμπεριφορά του CASinit για ένα καινούργιο bundle, δηλαδή, install στο default start level του Framework. Αυτό μπορεί να γίνει με της ύπαρξη property files, στον ίδιο κατάλογο με τα bundles. Τα αρχεία αυτά, πρέπει να έχουν το όνομα του bundle για το οποία θέλουν να αλλάξουν την συμπεριφορά του CASinit, ακολουθούμενα από την σταθερά "**.framework.properties**". Δηλαδή, για ένα bundle με το όνομα my_bundle.jar, το property file που θα αντιστοιχεί σε αυτό θα πρέπει να έχει το όνομα my_bundle.jar.framework.properties. Τα περιεχόμενα του αρχείου αυτού πρέπει να είναι τα εξής:

- **STARTLEVEL={0…N}**
- **ACTION={start, install, stop}**

Η παράμετρος, startlevel, ορίζει το start level στο οποίο θα λειτουργεί το εν λόγω bundle και η τιμή του είναι ένας ακέραιος μεγαλύτερος ή ίσος του μηδενός

Η παράμετρος ACTION, ορίζει την ενέργεια που θα εκτελέσει ο CASinit σε αυτό το bundle. Η τιμή start, ορίζει ότι το bundle εγκατασταθεί και θα ξεκινήσει την εκτέλεσή του ενώ η τιμή install ότι μόνο θα εγκατασταθεί χωρίς να ξεκινήσει εκτέλεση. Επιπλέον, ο CASinit, παρακολουθεί τα αρχεία αυτά για αλλαγές, οι οποίες μπορούν να γίνουν on the runtime.

Αυτός είναι και ο λόγος ύπαρξης της τιμής stop, η οποία θα σταματήσει την εκτέλεση του bundle.

4.3.3 Configuration Management

Η δυνατότητα αυτή λειτουργεί με παρόμοιο τρόπο, όπως αυτή της διαχείρισης των bundles. Ομοίως ο default προς παρακολούθηση κατάλογος είναι ο τρέχων, κάτι που μπορεί να αλλάξει από το Framework system property " **com.inaccessnetworks.casinit.config.path**". Η διαφορά, είναι ότι τα αρχεία που ενδιαφέρουν σε αυτή την περίπτωση, είναι configuration files γραμμένα σε **XML**. Ένα νέο τέτοιο αρχείο, σημαίνει για τον CASinit ένα νέο configuration προς τον Configuration Admin. Η αφαίρεσή του, την αφαίρεση του configuration από τον Configuration Admin και η ανανέωση του (αλλαγή του timestamp του αρχείου), την ενημέρωση των αλλαγών στον Configuration Admin.

Το OSGi Framework, ορίζει δύο τύπους Configurations για υπηρεσίες. Αυτές είναι τα ManagedServices και τα ManagedServiceFactories [1] (r3, Configuration Admin Service, κεφάλαιο 10). Και οι δύο τύποι υποστηρίζονται από τον CASinit, ανάλογα με την σύνταξη του κάθε αρχείου. Το όνομα του κάθε configuration αρχείου, **πρέπει** να είναι το ίδιο με το Managed Service PID ή Managed Service Factory FPID για το οποίο προορίζεται, ακολουθούμενο από την σταθερά "**.xml**".

Στην συνέχεια παρατίθενται τα XML Schemas για κάθε περίπτωση, μαζί με παραδείγματα του τρόπου χρήσης.

4.3.1.1 Managed Services

Ακολουθεί το XML Schema για τα αρχεία που διαχειρίζονται Configurations για Managed Services.

```
<xs:simpleType name=«casinit_types_basic» type=«xs:string»>
   <xs:enumeration value="String"/>
   <xs:enumeration value="Integer"/>
   <xs:enumeration value="Long"/>
```

```
    <xs:enumeration value="Float"/>
    <xs:enumeration value="Double"/>
    <xs:enumeration value="Short"/>
    <xs:enumeration value="Character"/>
    <xs:enumeration value="Boolean"/>
    <xs:enumeration value="Byte"/>
    <xs:enumeration value="long"/>
    <xs:enumeration value="int"/>
    <xs:enumeration value="short"/>
    <xs:enumeration value="char"/>
    <xs:enumeration value="boolean"/>
    <xs:enumeration value="byte"/>
    <xs:enumeration value="double"/>
    <xs:enumeration value="float"/>
</xs:simpleType>
<xs:simpleType name=«casinit_types_full» type=«xs:string»>
    <xs:extension base="casinit_types_basic">
    <xs:enumeration value="Vector"/>
    <xs:enumeration value="String[]"/>
    <xs:enumeration value="Integer[]"/>
    <xs:enumeration value="Long[]"/>
    <xs:enumeration value="Float[]"/>
    <xs:enumeration value="Double[]"/>
    <xs:enumeration value="Short[]"/>
    <xs:enumeration value="Character[]"/>
    <xs:enumeration value="Boolean[]"/>
    <xs:enumeration value="Byte[]"/>
    <xs:enumeration value="long[]"/>
```

```
    <xs:enumeration value="int[]"/>
    <xs:enumeration value="short[]"/>
    <xs:enumeration value="char[]"/>
    <xs:enumeration value="boolean[]"/>
    <xs:enumeration value="byte[]"/>
    <xs:enumeration value="double[]"/>
    <xs:enumeration value="float[]"/>
</xs:simpleType>
<xs:complexType name=«casinit_property» maxOccurs=«unbound»>
    <xs:attribute name="name" type="xs:string" use=«required»/>
    <xs:attribute name="type" type="casinit_types_full" use=«required»/>
    <xs:attribute name="value" type="xs:string"/>
    <xs:attribute name="oftype" type="casinit_types_basic"/>
    <xs:element name=«data» minOccurs=«0» maxOccurs=«unbound»/>
        <xs:complexType>
            <xs:attribute name=«value» type=«casinit_types_full»/>
        </xs:complexType>
    </xs:element>
</xs:complexType>
<xs:element name="simple">
    <xs:complexType>
        <xs:element name=«service» minOccurs=«1» maxOccurs=«1»>
            <xs:complexType>
                <xs:element name="property" type="casinit_property"/>
            </xs:complexType>
        </xs:element>
    </xs:complexType>
</xs:element>
```

Ακολουθεί ένα απλό παράδειγμα χρήσης

```
<simple>
  <service>
    <property name="ip" type="String" value="127.0.0.1" />
    <property name="port" type="Integer" value="8081" />
  </service>
</simple>
```

Στο παράδειγμα αυτό, η εν λόγω υπηρεσία θα δεχθεί ένα property με το όνομα ip και την τιμή 127.0.0.1 που θα είναι τύπου String και ένα property με όνομα port με τιμή 8081, που θα είναι τύπου Integer. Όπως αναφέρθηκε, οι τιμές αυτές μπορούν να αλλάξουν on the runtime με την σύνταξη των αρχείων που τις περιέχουν και το Framework θα ειδοποιηθεί αυτόματα για τις αλλαγές από τον CASinit.

4.3.1.2 **Managed Service Factories**

Ακολουθεί το XML Schema για την σύνταξη των αρχείων που αναφέρονται σε Manages Services Factories. Σημειώνεται ότι ο τύπος "casinit.property", είναι ο ίδιος με τα Managed Services.

```
<xs:element name="factory">
  <xs:complexType>
    <xs:element name=”service” minOccurs=”1” maxOccurs=”unbound”>
      <xs:complexType>
        <xs:element name=”property” type="casinit_property"/>
      </xs:complexType>
    </xs:element>
  </xs:complexType>
  <xs:element name =”key”>
```

```
      <xs:complexType>
        <xs:attribute name=”name” type=”xs:string” use=”required” />
      </xs:complexType>
    </xs:element>
</xs:element>
```

Ακολουθεί ένα παράδειγμα :

```
    <factory>
      <service>
        <property name="ip" type="String" value="127.0.0.1" />
        <property name="port" type="String" value="8081" />
        <property name="alias" type="String" value="alias1" />
      </service>

      <service>
        <property name="ip" type="String" value="127.0.0.1" />
        <property name="port" type="String" value="8080" />
        <property name="alias" type="String" value="alias2" />
      </service>
      <key name="ip,port" />
    </factory>
```

Στο παράδειγμα αυτό, παρατηρούμε ότι θέλουμε να δημιουργήσουμε Configuration για δύο υπηρεσίες του Managed Service Factory αυτού. Στο σημείο αυτό, γίνεται απαραίτητο για τον CASinit, να έχει έναν τρόπο να μπορεί να ξεχωρίζει τις διαφορετικές υπηρεσίες, ώστε αν υπάρξει αλλαγή τιμών στο αρχείο, να γνωρίζει ποια υπηρεσία πρέπει να ενημερώσει στο Framework. Έτσι, εισάγεται η έννοια του κλειδιού. Στο XML Schema ορίζεται το element **key**. Σκοπός του είναι να βοηθήσει τον CASinit να ξεχωρίζει τις διαφορετικές υπηρεσίες. Σαν τιμή του κλειδιού, είναι μια λίστα, χωρισμένη με κόμματα. Τα στοιχεία τις λίστας, αποτελούν κάποια property names του Configuration και ο συνδυασμός τους πρέπει να είναι μοναδικός

ανάμεσα σε όλα τα configurations στο αρχείο.

Στο παράδειγμά μας με τις δύο υπηρεσίες, χρησιμοποιείται για κλειδί η τιμή "ip,port". Αυτό σημαίνει ότι για την πρώτη υπηρεσίας, κλειδί θα είναι ο συνδυασμός των τιμών 127.0.0.1 και 8081, ενώ για την δεύτερη 127.0.0.1 8080. Οι τιμές αυτές δεν μπορούν να μεταβληθούν on the runtime. Εναλλακτικά, προτείνεται η διαγραφή του αρχείου και η εισαγωγή νέου με τις νέες τιμές για αυτά τα properties. Σημειώνεται, ότι ο τύπος δεδομένων που απαρτίζουν ένα κλειδί, δεν μπορεί να είναι τύπου Array ή Vector.

4.3.2 Χρήση Εξειδικευμένων Δεδομένων

Εκτός των απλών τύπων δεδομένων (Strings, Integers κτλ.), στο XML Schema των Configuration αρχείων του CASinit, αναφέρεται ότι μπορούν να χρησιμοποιηθούν τύποι όπως Arrays και Vectors. Στην ενότητα αυτή παρουσιάζεται με ποιόν τρόπο μπορεί να γίνει αυτό.

4.3.2.1 Χρήση Arrays

Στην περίπτωση αυτή, σαν "type" χρησιμοποιείται μια από τις επιλογές arrays που δηλώνονται στο XML Schema. Δηλαδή, int[], Integer[], double[], Double[] κτλ. . Για την εισαγωγή των δεδομένων χρησιμοποιείται το XML element "data" , όπως αυτό ορίζεται στο XML Schema. Ακολουθεί ένα απλό παράδειγμα χρήσης.

```
<property name="ips" type="String[]" >
    <data value="192.168.1.1"/>
    <data value="192.168.2.1"/>
</property>
```

Στο παράδειγμα αυτό δημιουργείται μια Array τύπου String, μεγέθους 2, με τα στοιχεία "192.168.1.1" και "192.168.2.1".

Επιπρόσθετα όμως, μπορεί να χρησιμοποιηθεί και ο "παραδοσιακός" τρόπος εισαγωγής τιμών, δηλαδή με την παράμετρο value, όπως στο παράδειγμα που ακολουθεί:

```
<property name="ports" type="Integer[]" value="8081" >
    <data value="8888"/>
</property>
```

Στην περίπτωση αυτή, οι τιμές των πεδίων "data" παραλείπονται. Δηλαδή στο παράδειγμα μας θα είχαμε το αποτέλεσμα μιας Array τύπου Integer μεγέθους 1, με τιμή 8081.

4.3.2.2 Χρήση Vectors

Στην περίπτωση αυτού του τύπου δεδομένων, δεν χρησιμοποιείται το πεδίο "value". Στην περίπτωση αυτή, εισάγεται η έννοια του πεδίου "oftype", το οποίο έχει σαν σκοπό να δηλώσει τον τύπο δεδομένων που θα περιέχει το Vector (μπορεί να περιέχει **μόνο ένα** τύπο δεδομένων). Το πεδίο τιμών ορίζεται στο XML Schema. Ακολουθεί ένα παράδειγμα:

```
<property name="names" type="Vector" oftype="String" >
    <data value="lon1"/>
    <data value="lon2"/>
</property>
```

Στο παράδειγμα αυτό, δημιουργείται ένα Vector που περιέχει τα στοιχεία, "lon1" και "lon2" τα οποία ορίζονται να είναι τύπου String.

Επιπλέον Σημειώσεις για το στοιχείο data

Το στοιχείο data, μπορεί να χρησιμοποιηθεί για την εισαγωγή δεδομένων και για τους υπόλοιπους τύπους δεδομένων με τον ίδιο τρόπο, όπως στο παράδειγμα που ακολουθεί :

```
<property name="id" type="Integer" >
    <data value="8888"/>
    <data value="8889"/>
</property>
```

Στην περίπτωση που υπάρχει ταυτόχρονα και το πεδίο value, αυτό υπερισχύει και χρησιμοποιείται τελικά. Σε περιπτώσεις όπως στο παράδειγμα, που υπάρχουν περισσότερα του ενός στοιχεία data, χρησιμοποιείται το πρώτο που ορίζεται. Στην περίπτωσή μας δηλαδή, η τιμή 8888.

4.3.3 Χρήση της υπηρεσίας com.inaccessnetworks.casinit.CasinitConfService

Ο CASinit, εκτός από την δυνατότητα τροποποίησης των Configurations για τα Managed Services και Manages Service Factories, παρέχει την δυνατότητα και σε άλλες υπηρεσίες του Framework να κάνουν το ίδιο, μέσω της υπηρεσίας com.inaccessnetworks.casinit.CasinitConfService που προσφέρει. Αυτή η υπηρεσία έχει σαν σκοπό να προσθέσει ένα επίπεδο αφαίρεσης ανάμεσα στις υπηρεσίες και τον Configuration Admin του Framework. Η διεπαφή της υπηρεσίας αυτής παρουσιάζεται αμέσως μετά :

Method Name	Return type	Attrs	Remarks
getConfigurations	String[]	void	Επιστρέφει ένα array of Strings που περιέχει όλα τα PIDs που διαχειρίζεται ο Casinit. Αν δεν υπάρχουν, επιστρέφεται κενό Array, ποτέ null
setConfiguration	boolean	String pid, InputStream in	Δημιουργεί ή ενημερώνει ένα Configuration για το δοθέν PID. Τα δεδομένα για το Configuration διαβάζονται από το δοθέν InputStream, βασισμένα στα XML Schemas του CASinit. Επιστρέφει true μετά από επιτυχή διαδικασία.
deleteConfiguration	boolean	String pid	Διαγράφει ένα Configuration για

			το δεδομένο PID. Επιστρέφει true μετά από επιτυχή διαγραφή.

Πίνακας 4.3: Μέθοδοι του CasinitConfService Interface

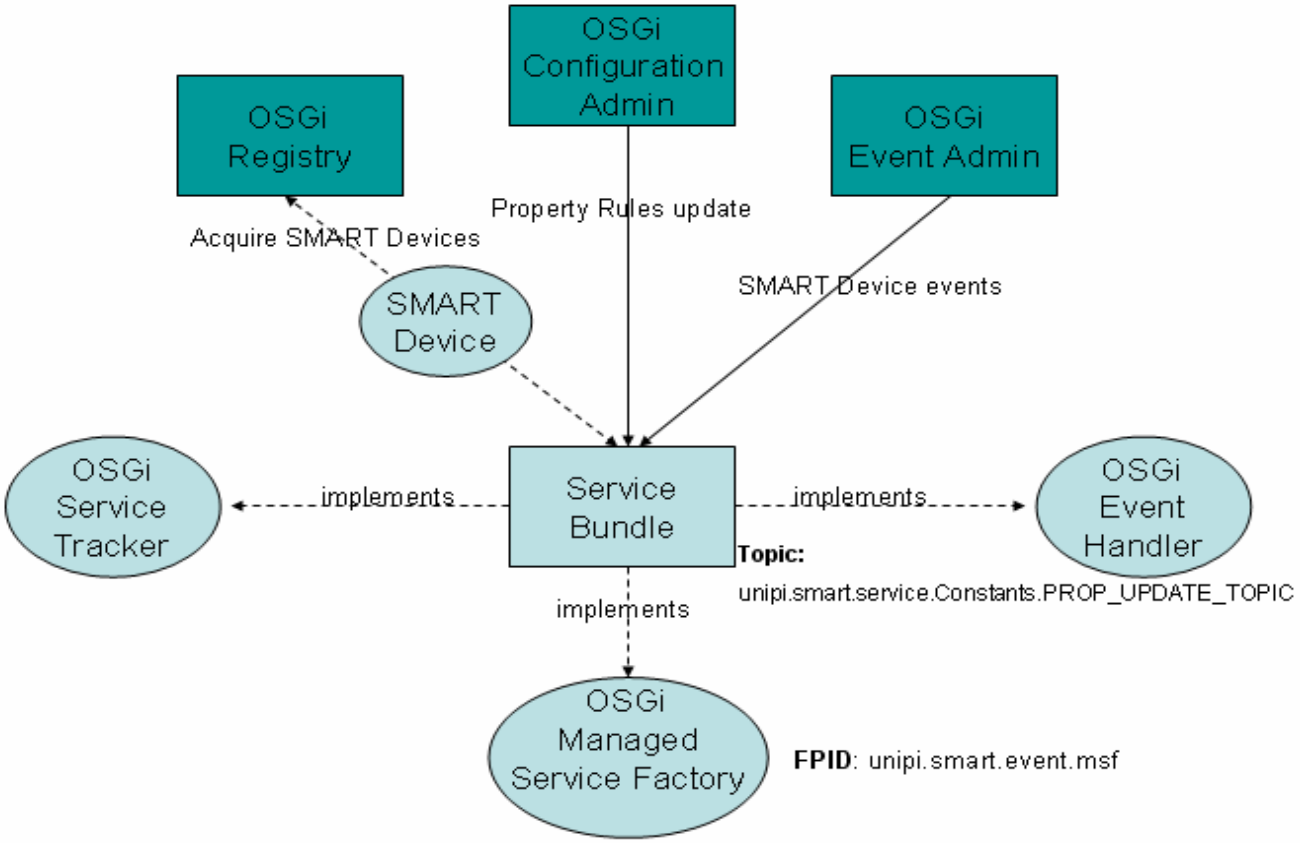
4.4 Inter Device Interaction Service

Η υπηρεσία (bundle) αυτή δημιουργήθηκε για την παρουσίαση των δυνατοτήτων της πλατφόρμας, να διαχειρίζεται σε υψηλό επίπεδο συσκευές ανεξαρτήτως τεχνολογίας. Όπως έχει προαναφερθεί, ένας από του στόχους της πλατφόρμας είναι η απόκρυψη των τεχνολογικών λεπτομερειών των εκάστοτε συσκευών, με την χρήση των SMART Interfaces.

Η υπηρεσία αυτή έχει την γνώση της διαχείρισης υπηρεσιών που υλοποιούν τις SMART Interfaces. Με άλλα λόγια ξέρει πώς να αναζητεί και να διαχειρίζεται SMART Devices. Πιο συγκεκριμένα, η υπηρεσία αυτή, με βάση ένα σύνολο κανόνων που θα δίνονται από τον χρήστη, θα μπορεί να λάβει αποφάσεις για την διαχείριση συγκεκριμένων συσκευών.

Για παράδειγμα, ο χρήστης θα μπορεί να ορίσει τον κανόνα, όταν ο SMART Switch A, έρθει στην θέση ON, να ανάψουν τα SMART Lights B σε επίπεδο 100% και το SMART Light Γ σε επίπεδο φωτεινότητας 50%. Ο SMART Switch A, μπορούσε να αντιπροσωπεύει στον πραγματικό κόσμο έναν ανιχνευτή κίνησης, ο οποίος όταν ανιχνεύσει κίνηση, θα θέλαμε το SMART Light B, που θα μπορούσε να αντιπροσωπεύει την παροχή ηλεκτρικού ρεύματος σε έναν συναγερμό, να ανάψει και τέλος το SMART Light Γ, που θα αντιπροσώπευε έναν λαμπτήρα φωτισμού να ανάψει στο επίπεδο 50%.

Με τον τρόπο αυτό ο χρήστης μπορεί εύκολα να ορίσει χρήσιμες υπηρεσίες τέτοιου τύπου, που χρησιμοποιούν συσκευές οι οποίες δεν θα μπορούσαν να αλληλεπιδράσουν μεταξύ τους χωρίς την χρήση της πλατφόρμας. Στο παράδειγμά μας, ο ανιχνευτής κίνησης για παράδειγμα θα μπορούσε να χρησιμοποιεί ένα ασύρματο πρωτόκολλο, το οποίο δεν έχει τίποτα κοινό με τα ενσύρματα πρωτόκολλα που θα χρησιμοποιούσαν ο συναγερμός και ο λαμπτήρας. Στην εικόνα που ακολουθεί, παρουσιάζεται εν συντομία η αρχιτεκτονική της υπηρεσίας αυτής, η οποία εν συνεχεία θα αναλυθεί.



Εικόνα 4.5: Αρχιτεκτονική Inter Device Interaction Service

Στην εικόνα της αρχιτεκτονικής της υπηρεσίας παρατηρούμε τον τρόπο λειτουργίας της και αλληλεπίδρασης της με άλλες υπηρεσίες του OSGi Framework.

Η υπηρεσίας μας, υλοποιεί το Managed Service Factory Interface του OSGi Framework. Η υπηρεσία Configuration Admin του Framework, ειδοποιεί για configurations τις οποίες θα θέτει ο διαχειριστής/χρήστης του συστήματος (για ευκολία μπορεί να χρησιμοποιηθεί η βοηθητική υπηρεσία CASinit, όπως παρουσιάζεται στο κεφάλαιο παρουσίασης του τρόπου χρήσης της πλατφόρμας). Τα configurations αυτά αποτελούν τους κανόνες κάτω από τους οποίους θα λειτουργεί η υπηρεσία. Το πρωτόκολλο σύνταξης των κανόνων, ακολουθεί ένα ορισμένο συντακτικό τρόπο, οποίος θα αναλυθεί εν συνεχεία.

Επιπρόσθετα, η υπηρεσία μας χρησιμοποιεί την υπηρεσίας Service Tracker του OSGi, για να λαμβάνει πληροφόρηση από το OSGi Registry για εγκατάσταση ή διαγραφή SMART Devices. Με αυτό τον τρόπο θα μπορεί να χρησιμοποιήσει τις δυνατότητες τους όταν αυτό

είναι χρήσιμο, να ανάψει δηλαδή όταν χρειαστεί έναν λαμπτήρα.

Τέλος, η υπηρεσία υλοποιεί το interface Event Handler του OSGi, ώστε η υπηρεσία Event Admin του OSGi, να ειδοποιεί την υπηρεσία μας για γεγονότα που έχουν να κάνουν με property updates των SMART Devices. Για παράδειγμα ότι ένας SMART Switch ήρθε στην θέση ON. Έτσι η υπηρεσία μας έχει τελικά όλη της πληροφορία και τους πόρους που χρειάζεται για να λάβει αποφάσεις και να διαχειριστεί SMART Devices με βάση τους κανόνες που δίνει ο χρήστης. Στην επόμενη ενότητα αναλύεται το συντακτικό των κανόνων αυτών.

4.4.1 Σύνταξη SMART Rules

Η υπηρεσία αυτή χρησιμοποιεί την υπηρεσία Configuration Admin για την ανάκτηση και ανανέωση των κανόνων διαχείρισης. Οι δυνατότητες τον κανόνων αυτής της προς επίδειξη υπηρεσίας, είναι η συσχέτιση ενός συνόλου από **trigger events,** όπως για παράδειγμα, ένας SMART Switch να έρθει στην θέση ON, με ένα σύνολο από **action results**, όπως για παράδειγμα το άναμμα ενός λαμπτήρα. Οι κανόνες λοιπόν, βασίζονται στην λογική ότι όταν συγκεκριμένα properties κάποιων SMART Devices, αποκτήσουν κάποιες συγκεκριμένες τιμές, τότε να γίνει η ανάθεση κάποιων τιμών σε κάποια properties κάποιων άλλων SMART Devices.

Ένας κανόνας για να είναι λειτουργικός απαιτεί την ύπαρξη των παρακάτω Configuration Properties, των οποίων η σημασία και αναλύεται:

Property Name	Property Type	Property Value Set
name	String	Μοναδικό όνομα ανάμεσα στους κανόνες
association_type	String	{AND | OR}
sensor_events	Vector (of Strings)	Εξαρτάται από τον SMART Device Type
actuator_actions	Vector (of Strings)	Εξαρτάται από τον SMART Device Type

Πίνακας4.4: Configuarion Properties των κανόνων

Property "name"

Σκοπός του property αυτού, είναι να θέσει ένα μοναδικό όνομα ανάμεσα στους κανόνες. Οι τιμή του μπορεί να είναι οποιοδήποτε String, αρκεί να είναι μοναδικό ανάμεσα στους υπόλοιπους κανόνες.

Property "association_type"

Σκοπός του property αυτού, είναι να θέσει τον τύπο του κανόνα αναφορικά με τα **trigger events**. Στην περίπτωση που έχει την τιμή "**OR**", αρκεί ένα από τα trigger events να είναι αληθές για την εκτέλεση των action results. Στην περίπτωση που έχει την τιμή "**AND**", πρέπει να είναι να αληθή όλα τα properties των trigger events τγια την εκτέλεση των action results.

Property "sensor_events"

Η τιμή του property αυτού, αποτελεί **το σύνολο των trigger events** ενός κανόνα. Είναι τύπου Vector of Strings και κάθε στοιχείο του αποτελεί ένα ξεχωριστό trigger event, ο τρόπος σύνταξης του οποίου είναι ο εξής:

<SMART_device_ID>$<SMART_Device_type>$<property_name>$<property_value>

Εν συντομία, ένα trigger event, αποτελείται από το μοναδικό στο Framework SMART Device ID, των συγκεκριμένο SMART type αυτού, το property name το οποίο θα παρακολουθείται και το property value στο οποίο επιθυμούμε να βρεθεί για την εκτέλεση ενός action results. Το σύνολο property names που μπορούν να χρησιμοποιηθούν εξαρτάται άμεσα από αυτά που υποστηρίζει το Device type που έχει επιλεγεί, ενώ τα property values εξαρτώνται από το σύνολο των επιτρεπόμενων για αυτό το property name, όπως αυτά ορίζονται από τα SMART Devices Interfaces. Τα παραπάνω χωρίζονται μεταξύ τους με το σύμβολο "**$**". Για παράδειγμα:

SMART_SENSOR1**$**SMART_SENSOR**$**state**$**1

Όπου ορίζεται ότι για την εκτέλεση των action results, πρέπει το property **state** του SMART Device **SMART_SENSOR1**, τύπου **SMART_SENSOR**, να πάρει την τιμή **1**.

Property "actuator_actions"

Όπως έχει αναφερθεί, σκοπός των κανόνων είναι τελικά η τροποποίηση κάποιων properties σε κάποια δεδομένω SMART Devices, δεδομένων κάποιων συμθηκών (trigger events). Το property αυτό, έχει σαν στόχο να ορίσει αυτές τις δράσεις. Το συντακτικό του είναι πανομοιότυπο με την σύνταξη του sensor_events που μόλις αναλύθηκε. Για παράδειγμα:

SMART_LIGHT2**$**SMART_LIGHT**$**level**$**50

Στο παράδειγμα αυτό, ορίζεται ότι όταν οι κανόνες των trigger events είναι αληθείς, το property **level** του SMART Device **SMART_LIGHT2** που είναι τύπου **SMART_LIGHT**, να πάρει την τιμή **50**.

4.5 User Interface Bundle

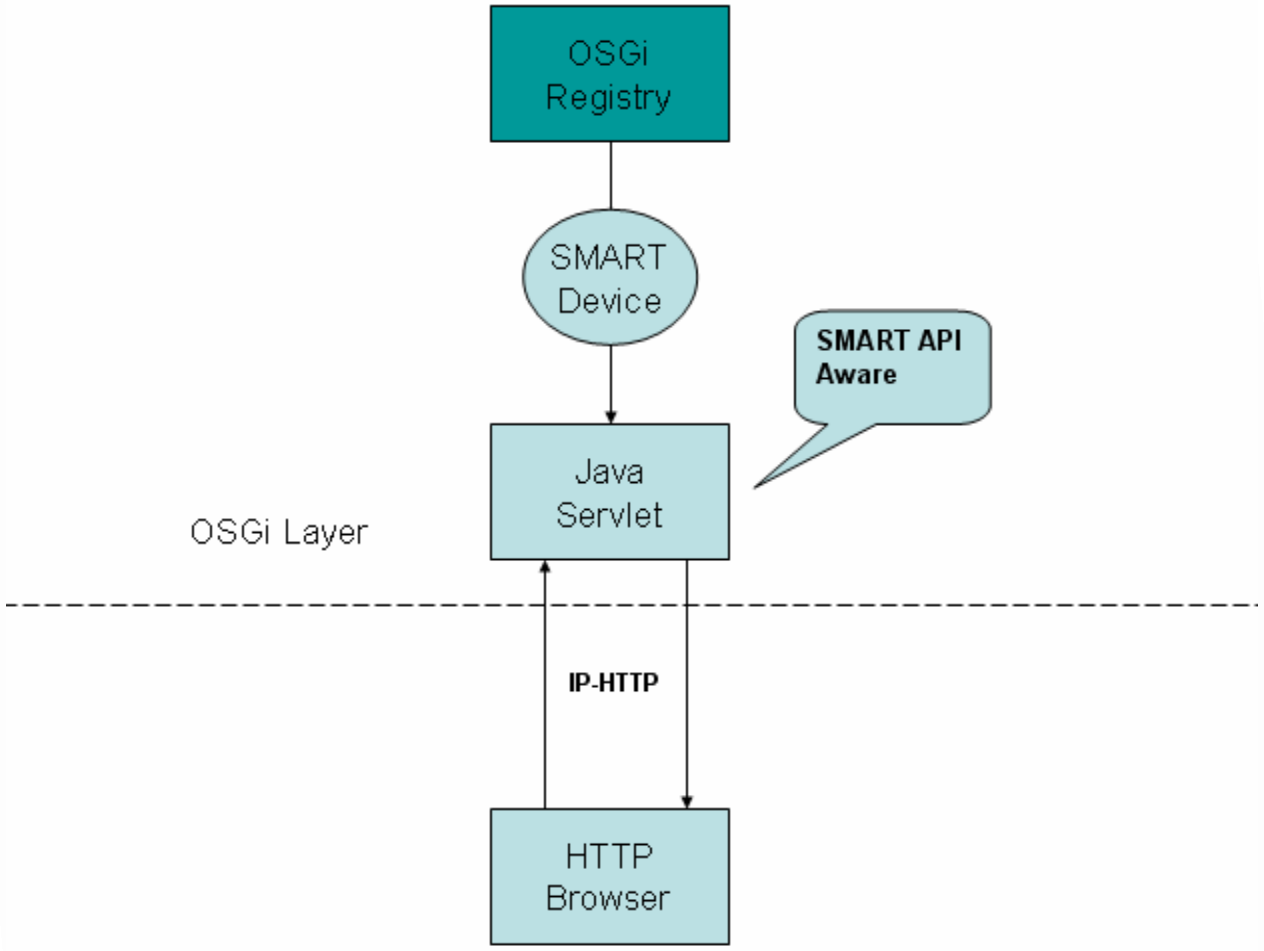
Σκοπός της διπλωματικής εργασίας αυτής είναι η διαχείριση συσκευών που δρουν σε ένα περιβάλλον SMART Space, με τρόπο τέτοιον ώστε οι τεχνολογικές του ιδιαιτερότητες να είναι διάφανες στο χρήστη. Συνεπώς, περνάμε στην περιοχή του χειρισμού των συσκευών από την πλευρά του χρήστη, δηλαδή στην δημιουργία ενός User Interface.

Η αρχιτεκτονική της πλατφόρμας είναι έτσι κατασκευασμένη, ώστε να είναι εύκολο σε εξωτερικές οντότητες να συνδεθούν και να λάβουν στοιχεία για τα SMART Devices του συστήματος. Αρκεί η δημιουργία του κατάλληλου Driver σε OSGi.

Το σύνολο των τεχνολογιών που θα μπορούσαν να χρησιμοποιηθούν λοιπόν για την παρουσίαση και διαχείριση των συσκευών από τον χρήστη είναι πολύ μεγάλο. Στην γενικότερη περίπτωση, οποιαδήποτε εξωτερική τεχνολογία επιτρέπει την χρήση Sockets, είναι υποψήφια προς χρήση.

Στα πλαίσια αυτής της διπλωματικής δύο ήταν οι υποψήφιες τεχνολογίες προς χρήση. Η πρώτη η χρήση της Adobe Flash ενώ η δεύτερη η χρήση HTTP Requests. Επιλέχθηκε τελικά η χρήση του HTTP λόγω της ευκολίας ανάπτυξης και υποστήριξης από το OSGi. Συγκεκριμένα ο OSGi driver αποτελεί ένα **Java Servlet**, που υποστηρίζεται με μεγάλη ευκολία από της

υπηρεσίες του OSGi. Μελλοντικά, αυτό θα μπορούσε να αναβαθμιστεί και στην χρήση AJAX, όταν αυτό υποστηρίζεται από το OSGi. Ακολουθεί, μια εικόνα που παρουσιάζει την αρχιτεκτονική αυτή, η οποία και θα αναλυθεί εν συνεχεία.



Εικόνα 4.6: Αρχιτεκτονική User Interface

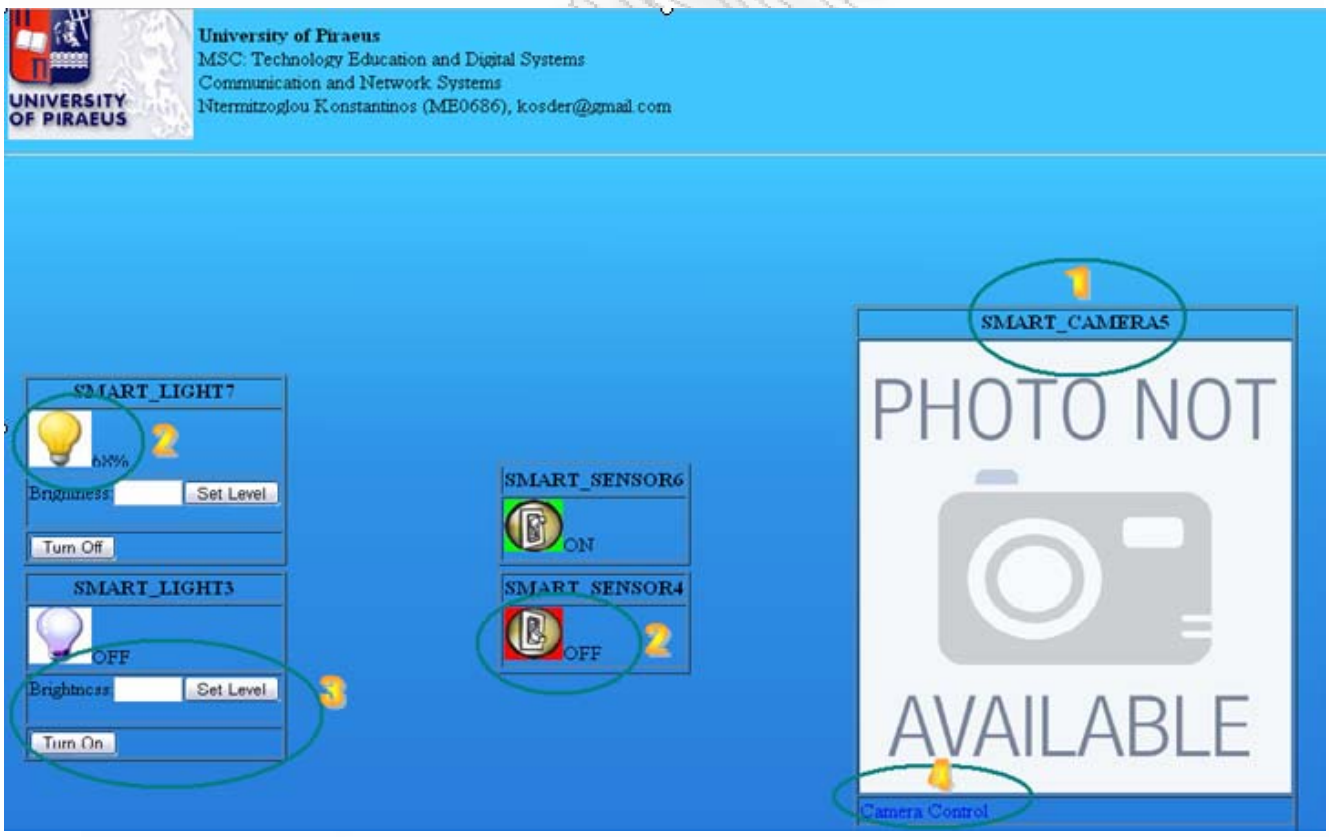
Στην εικόνα λοιπόν αυτή, παρατηρούμε ότι οι εξωτερικές οντότητες, δηλαδή στην περίπτωση μας οι HTTP Bowsers, επικοινωνούν με το OSGi με βάση το πρωτόκολλο HTTP.

Από την πλευρά του OSGi, υπάρχει υλοποιημένη η υπηρεσία **HTTP Service** [1] (OSGi r3, κεφάλαιο 14, σελίδα 287) του OSGi, στην οποία έχει εγκατασταθεί ένα Java Servlet, το οποίο και αποτελεί τον User Interface Driver που αναπτύχθηκε στα πλαίσια της διπλωματικής. Όπως θα δούμε και στο κεφάλαιο παρουσίασης, το URL στο οποίο ακούει η υπηρεσίας μας είναι το:

<IP>:8080/display

Όπου <IP> είναι η IP του υπολογιστή στον οποίο τρέχει η πλατφόρμα μας.

Τα HTTP requests που καταφθάνουν τελικά από τον εκάστοτε Browser στο HTTP Servlet μας, αναλύονται και τελικά δημιουργείται και επιστρέφεται η κατάλληλη HTML σελίδα. Εν συντομία, επιστρέφεται μια σελίδα, η οποία περιέχει κατηγοριοποιημένα όλα τα SMART Devices τα οποία είναι εγγεγραμμένα εκείνη τη στιγμή το framework, δίνοντας τα κατάλληλα controls για την διαχείρισή τους. Ακολουθείται η πολιτική **polling** για την ανανέωση των στοιχείων της σελίδας. Με άλλα λόγια η σελίδα ανανεώνεται αυτόματα κάθε μερικά δευτερόλεπτα και μπορούμε να δούμε πιθανές αλλαγές στα στοιχεία των SMART Devices, π.χ. ένας λαμπτήρας να άναψε. Ακολουθεί μια εικόνα που παρουσιάζει την μορφή της σελίδας αυτής, αναλύοντας στην συνέχεια τα στοιχεία της για την ευκολότερη κατανόηση των παραδειγμάτων που θα ακολουθήσουν στο κεφάλαιο παρουσίασης της πλατφόρμας.



Εικόνα 4.7: Η βασική επιφάνεια χρήστη

Στην HTML σελίδα αυτή παρουσιάζονται πέντε διαφορετικές SMART συσκευές. Στα αριστερά

δύο λαμπτήρες, στο κέντρο δύο αισθητήρες και δεξιά μια κάμερα. Στην συνέχεια επεξηγούνται τα σημεία που έχουν σημειωθεί πάνω στην εικόνα ψηφιακά:

1.

Πάνω από κάθε SMART συσκευή υπάρχει ένα όνομα-κωδικός, το οποίο αντιπροσωπεύει τον μοναδικό κωδικό SMART_ID που δίνεται σε κάθε SMART Device.

2.

Για κάθε μια SMART συσκευή που παρουσιάζεται σε αυτή την επιφάνεια, υπάρχει και ένα ειδικό πεδίο, που συνοδεύεται από μια χαρακτηριστική εικόνα που δηλώνει την παρούσα κατάσταση της εκάστοτε συσκευής.

3.

Αντίθετα με τις συσκευές τύπου αισθητήρα και κάμερας που είναι παθητικές, η συσκευή τύπου λαμπτήρα δίνει την δυνατότητα στον χρήστη να αλλάξει τις ιδιότητές της. Πιο συγκεκριμένα δηλαδή, να τον ανάψει ή σβήσει ή/και να θέσει το επίπεδο φωτεινότητας αυτού.

4.

Για τις συσκευές τύπου κάμερας, εκτός από την εικόνα της κάμερας, παρέχεται και το URL στο οποίο βρίσκονται τα camera controls του κατασκευαστή. Σημειώνεται ότι, όπως στην εικόνα του παραδείγματος, όταν η εικόνα της κάμερας δεν μπορεί να ανακτηθεί, π.χ. επειδή η κάμερα είναι εκτός λειτουργίας, παρουσιάζεται η εικόνα που πληροφορεί για το συμβάν.

Κεφάλαιο 5ο

Εγκατάσταση και εκκίνηση

Στο παρόν κεφάλαιο γίνεται μια επισκόπηση για την εγκατάσταση και εκτέλεση της πλατφόρμας. Σημειώνεται, ότι η αρχιτεκτονική λόγω της χρήσης Pure Java, λειτουργεί ανεξαρτήτως λειτουργικού συστήματος με συμβατό JVM. Παρόλα αυτά, στο κεφάλαιο αυτό δίνονται οδηγίες για χρήση σε περιβάλλον Linux, το οποίο και προτείνεται για χρήση. Στην πραγματικότητα όμως δεν πρέπει να υπάρχουν ουσιώδεις διαφορές για χρήση κάτω από άλλες πλατφόρμες.

5.1 Προαπαιτούμενα

Λειτουργικό Σύστημα: Οποιοδήποτε σύστημα με δυνατότητα υποστήριξης JVM όπως για παράδειγμα Linux ή Microsoft Windows. Κατά την διάρκεια υλοποίησης χρησιμοποιήθηκε το λειτουργικό σύστημα **Linux Kubuntu 7.10** το οποίο μπορεί να αποκτηθεί δωρεάν από το http://www.kubuntu.org .

JVM: Το OSGi Framework που αποτελεί την βάση της πλατφόρμας που αναπτύχθηκε, είναι γραμμένο στην γλώσσα προγραμματισμού Java, συνεπώς η εκτέλεση της πλατφόρμας απαιτεί εγκατεστημένη JVM. Συνίσταται η χρήση SUN **JDK 1.6** ή μεγαλύτερη, η οποία μπορεί να αποκτηθεί δωρεάν από το http://java.sun.com/javase/downloads/index.jsp. Εναλλακτικά, στο συνοδευτικό CD παρέχεται η έκδοση 1.6 για περιβάλλον Linux, κάτω από τον φάκελο essential/java/ .

Hardware: Γενικά, η εφαρμογή δεν έχει ιδιαίτερες απαιτήσεις σε hardware. Εμπειρικά, προτείνεται η χρήση συστήματος με **32MB RAM** και επεξεργαστή της τάξης των **300MHz**. Οι απαιτήσεις σε χώρο αποθήκευσης εξαιρώντας την JVM, είναι σχεδόν μηδενικές για τα σημερινά δεδομένα, της τάξης των **5MB**. Συνεπώς, η χρήση της πλατφόρμας θα μπορούσε να γίνει εύκολα και σε **embedded συστήματα**, όπου οι πόροι είναι δυσεύρετοι.

5.2 Ειδικές απαιτήσεις

Η πλατφόρμα κατά την εκτέλεσή της απαιτεί την χρήση σε κάποιες TCP/UDP ports από το σύστημα. Αυτές είναι οι εξής:

TCP

8080: Για χρήση HTTP

8888: Για χρήση του IP Base Driver της πλατφόρμας

UDP

8008 και **1900:** Για την χρήση UPnP

5.3 Εγκατάσταση

Στην παράγραφο αυτή περιγράφεται ο τρόπος εγκατάστασης σε ένα περιβάλλον Linux. Στο συνοδευτικό CD, εκτός από τον πηγαίο κώδικα του λογισμικού που αναπτύχθηκε, περιέχονται όλα τα απαιτούμενα binary αρχεία για την άμεση εκτέλεση της πλατφόρμας.

5.3.1 Εγκατάσταση του JVM

Σαν root, αντιγράψτε τον φάκελο /essential/java/jdk1.6.0_03 στο file system του συστήματος σας, έστω στο path /opt/jdk1.6.0_03 .

Εν συνεχεία εκτελέστε την εντολή

PATH=<JDK_DIR>/bin:$PATH

Δηλαδή αν χρησιμοποιήσατε το προτεινόμενο path /opt/jdk1.6.0_03, η εντολή πρέπει να είναι η:

PATH=/opt/jdk1.6.0_03/bin/:$PATH

Αυτή η εντολή δόθηκε για να γίνει το JVM κάναμε install, το default JVM του συστήματος. Η εντολή αυτή θα πρέπει να εκτελείται κάθε φόρα πριν την εκτέλεση της πλατφόρμας. Για να αποφευχθεί αυτό, μπορείτε προαιρετικά να βάλλεται την εντολή αυτή στο αρχείο .bashrc του χρήση που θα εκτελεί την πλατφόρμα ώστε να γίνεται αυτόματα. Αυτό μπορεί να γίνει με την εντολή:

echo PATH=<JDK_DIR>/bin/:$PATH >> /home/<user>/.bashrc

όπου <user> είναι το όνομα του εκάστοτε χρήστη, ενω <JDK_DIR> είναι το full path

εγκατάστασης του JDK που εγκαταστάθηκε μόλις.

5.3.2 Εγκατάσταση της πλατφόρμας

Η πλατφόρμα βρίσκεται έτοιμη προς εκτέλεση στο συνοδευτικό CD. Αρκεί να αντιγραφεί ο φάκελος : platform/oscar/ σε έναν κατάλογο του συστήματος με δικαιώματα ανάγνωσης, εγγραφής και εκτέλεσης. Έστω στον κατάλογο /home/<user>/oscar/ , όπου <user> είναι το όνομα χρήστη που θα εκτελεί την πλατφόρμα.

5.3.3 Εκτέλεση του συστήματος

Έστω ότι η εγκατάσταση της πλατφόρμας έγινε στον κατάλογο : /home/<user>/oscar/, τότε η εκτέλεση της πλατφόρμας μπορεί να γίνει με την εκτέλεση του "**run.sh**" script, που βρίσκεται στο /home/<user>/oscar/run.sh . Το αποτέλεσμα θα πρέπει να είναι κάτι σαν το επόμενο:

```
Welcome to Oscar.
==================

-> 18:55:26.643 EVENT  Starting Jetty/4.2.x
18:55:26.699 EVENT  Started SocketListener on 0.0.0.0:8080
18:55:26.705 EVENT  Started org.mortbay.http.HttpServer@a39137
18:55:26.884 EVENT  Started ServletHttpContext[/]
18:55:26.898 EVENT  Started HttpContext[/display/icon_LAMP_ON.jpg]
18:55:26.899 EVENT  Started HttpContext[/display/icon_LAMP_OFF.jpg]
18:55:26.900 EVENT  Started HttpContext[/display/icon_SWITCH_ON.jpg]
18:55:26.900 EVENT  Started HttpContext[/display/icon_SWITCH_OFF.jpg]
18:55:26.902 EVENT  Started HttpContext[/display/icon_IMAGE_UNAVAILABLE.jpg]
18:55:26.903 EVENT  Started HttpContext[/display/icon_main_bg.jpg]
18:55:26.904 EVENT  Started HttpContext[/display/icon_logo_unipi.jpg]
Available : org.osgi.service.cm.ConfigurationAdmin
Registered traces command
```

```
Registered set properties command
Registered get properties command
Registered get configuration command
Registered remconf configuration command
Registered set configuration command
Registered create factory pid command
Registered create pid command
Registered garbage collector command
[Device Manager] info: Passive start
IPDevMsgParser started
IPDevReceive reader Thread started
[Device Manager] info: Activating
[Device Manager] info: driver found, IP LIGHT DEVICE DRIVER/9
[Device Manager] info: driver found, UPnP LIGHT DEVICE DRIVER/22
[Device Manager] info: driver found, IP CAMERA DEVICE DRIVER/19
[Device Manager] info: driver found, IP SENSOR DEVICE DRIVER/4
->
```

Κεφαλαίο 6ο

Επίδειξη λειτουργίας

Στο κεφάλαιο αυτό γίνεται η χρήση της πλατφόρμας και των εξωτερικών οντοτήτων για λόγους επίδειξης της λειτουργικότητας της. Σκοπός του κεφαλαίου είναι να επιδείξει όλα τα χαρακτηριστικά της σε υψηλό επίπεδο ανάλυσης, δείχνοντας πως αξιοποιείται η τεχνολογία OSGi για να ξεπεραστούν τα προβλήματα και να επιτευχθούν οι στόχοι που αναγράφονται στην εισαγωγική ενότητα αυτού του εγγράφου.

Εν συντομία, το κεφάλαιο αυτό θα ξεκινήσει δημιουργώντας μια σειρά από συσκευές τύπου IP Devices, τις οποίες στη συνέχεια θα χειριστούμε με τη βοήθεια της πλατφόρμας. Όλα τα βήματα θα περιγράφονται αναλυτικά και θα παρουσιάζονται σχετικές εικόνες για την ευκολότερη κατανόηση του αναγνώστη. Στην συνέχεια θα προστεθεί μια σειρά από UPnP Devices ώστε να φανεί πως παρουσιάζονται και λειτουργούν σε αρμονία με τις υπόλοιπες συσκευές. Τέλος θα χρησιμοποιήσουμε της υπηρεσία που δημιουργήσαμε ώστε να κάνουμε τις συσκευές αυτές να αλληλεπιδράσουνε μεταξύ τους και να λειτουργήσουν σαν σύνολο ανεξαρτήτως τεχνολογίας.

6.1 Έναρξη της πλατφόρμας

Ακολουθώντας τις οδηγίες του ομώνυμου κεφαλαίου για την εκτέλεσή της πλατφόρμας, εφόσον όλα έχουν εγκατασταθεί όπως θα έπρεπε, θα έχουμε το ακόλουθο αποτέλεσμα στην οθόνη:

```
Welcome to Oscar.
==================
-> 19:12:56.718 EVENT  Starting Jetty/4.2.x
19:12:56.796 EVENT  Started SocketListener on 0.0.0.0:8080
19:12:56.796 EVENT  Started org.mortbay.http.HttpServer@7a78d3
19:12:56.968 EVENT  Started ServletHttpContext[/]
19:12:56.984 EVENT  Started HttpContext[/display/icon_LAMP_ON.jpg]
19:12:56.984 EVENT  Started HttpContext[/display/icon_LAMP_OFF.jpg]
```

```
19:12:56.984 EVENT  Started HttpContext[/display/icon_SWITCH_ON.jpg]
19:12:56.984 EVENT  Started HttpContext[/display/icon_SWITCH_OFF.jpg]
19:12:56.984 EVENT  Started HttpContext[/display/icon_IMAGE_UNAVAILABLE.jpg]
19:12:56.984 EVENT  Started HttpContext[/display/icon_main_bg.jpg]
19:12:56.984 EVENT  Started HttpContext[/display/icon_logo_unipi.jpg]
Available : org.osgi.service.cm.ConfigurationAdmin
Registered traces command
Registered set properties command
Registered get properties command
Registered get configuration command
Registered remconf configuration command
Registered set configuration command
Registered create factory pid command
Registered create pid command
Registered garbage collector command
[Device Manager] info: Passive start
IPDevMsgParser started
IPDevReceive reader Thread started
[Device Manager] info: Activating
[Device Manager] info: driver found, IP SENSOR DEVICE DRIVER/4
[Device Manager] info: driver found, IP LIGHT DEVICE DRIVER/9
[Device Manager] info: driver found, IP CAMERA DEVICE DRIVER/20
[Device Manager] info: driver found, UPnP LIGHT DEVICE DRIVER/11
```

Στο σημείο αυτό το Framework είναι έτοιμο. Για την επαλήθευση αυτού αρκεί να εκτελέσουμε την εντολή "**ps**" στον φλοιό εντολών του Framework και να δούμε ότι η κατάσταση όλων των εγκατεστημένων bundles είναι "active." Δηλαδή κάτι σαν το ακόλουθο :

```
-> ps
START LEVEL 5
  ID   State         Level  Name
   [0] [Active     ] [    0] System Bundle (1.0.5)
   [1] [Active     ] [    1] OSGi Util (1.0.0)
   [2] [Active     ] [    1] OSGi Service (1.0.2)
   [3] [Active     ] [    1] CASinit (2.4.0)
   [4] [Active     ] [    1] IP Sensor Device Driver (1.0.0)
   [5] [Active     ] [    1] EventAdmin Service (1.0.0.RC2)
   [6] [Active     ] [    1] Shell TUI (1.0.0)
```

```
  [7] [Active     ] [    1] HTTP Service (1.1.2)
  [8] [Active     ] [    1] SMART Event Handler (1.0.0)
  [9] [Active     ] [    1] IP Light Device Driver (1.0.0)
 [10] [Active     ] [    1] SMART Device Control (1.0.0)
 [11] [Active     ] [    1] UPnP Light Device Driver (1.0.0)
 [12] [Active     ] [    1] CMService (1.0.0)
 [13] [Active     ] [    1] OSGi Service (1.0.2)
 [14] [Active     ] [    1] os4os-Extended_OscarShell (1.0.0)
 [15] [Active     ] [    1] DeviceManager (1.0.0)
 [16] [Active     ] [    1] Shell Service (1.0.2)
 [17] [Active     ] [    1] Servlet (1.0.0)
 [18] [Active     ] [    1] Domoware UPnP Base Driver 3.0.2 (3.0.2)
 [19] [Active     ] [    1] IPDev Base Driver (1.0.0)
 [20] [Active     ] [    1] IP Camera Device Driver (1.0.0)
 [21] [Active     ] [    1] SMART API (1.0.0)
 [22] [Active     ] [    1] Domoware UPnP Base Driver Extra 1.0.0 (1.0.0)
->
```

6.2 Δημιουργία IP Lights

Στο σημείο αυτό, θέλουμε να δημιουργήσουμε μερικές συσκευές για να διαχειριστούμε με την πλατφόρμα. Αρχικά θα δημιουργήσουμε δύο εικονικούς IP λαμπτήρες, χρησιμοποιώντας την εξωτερική οντότητα **IP Device Maintenance**. Υπενθυμίζουμε ότι αυτή η οντότητα αντιπροσωπεύει εικονικές συσκευές στο πραγματικό κόσμο. Δηλαδή πρέπει να φανταστούμε ότι οι αλλαγές που κάνουμε χρησιμοποιώντας αυτό το εργαλείο, αντιπροσωπεύουν τις αλλαγές που θα γινόντουσαν πάνω σε πραγματικές συσκευές. Για παράδειγμα, αν χρησιμοποιήσουμε το εργαλείο για να κλείσουμε έναν λαμπτήρα, είναι το ίδιο με το να κλείναμε έναν πραγματικό λαμπτήρα που ακολουθούσε το πρωτόκολλό μας.

Εκτελώντας λοιπόν το IP Device Maintenance, απλά γράφοντας :

```
java –jar ipdev_maint.jar
```

και ενώ βρισκόμαστε στον ίδιο κατάλογο με το εν λόγω αρχείο, θα πρέπει να πάρουμε μια κονσόλα που περιμένει εντολές μας. Για την δημιουργία λοιπόν δύο εικονικών IP λαμπτήρων, αρκεί να γράψουμε δύο φορές την εντολή :

create LIGHT

Έχουμε λοιπόν δημιουργήσει 2 λαμπτήρες, και στην κονσόλα του IP Device Maintenance θα δούμε τα εξής μηνύματα:

```
Main started.
->create LIGHT
Created IPDevice with ID: 1
->IPdevice 1 Connected to OSGi
Data sent to OSGi: <smart><msg type="hello" propName="type" propValue="LIGHT" />
</smart>
DEBUG: IPDevice 1 received command from OSGi
Parsing message <smart><msg type="get" propName="state" propValue="" /></smart>
Data sent to OSGi: <smart><msg type="update" propName="state" propValue="null" /
></smart>
DEBUG: IPDevice 1 received command from OSGi
Parsing message <smart><msg type="get" propName="level" propValue="" /></smart>
Data sent to OSGi: <smart><msg type="update" propName="level" propValue="null" /
></smart>
->create LIGHT
Created IPDevice with ID: 2
->IPdevice 2 Connected to OSGi
Data sent to OSGi: <smart><msg type="hello" propName="type" propValue="LIGHT" />
</smart>
DEBUG: IPDevice 2 received command from OSGi
Parsing message <smart><msg type="get" propName="state" propValue="" /></smart>
Data sent to OSGi: <smart><msg type="update" propName="state" propValue="null" /
></smart>
DEBUG: IPDevice 2 received command from OSGi
Parsing message <smart><msg type="get" propName="level" propValue="" /></smart>
Data sent to OSGi: <smart><msg type="update" propName="level" propValue="null" /
></smart>
```

Για την επαλήθευση, πληκτρολογούμε την εντολή "**ls**" στον IP Device Maintenance για να δούμε την λίστα με τις συσκευές που έχουν δημιουργηθεί:

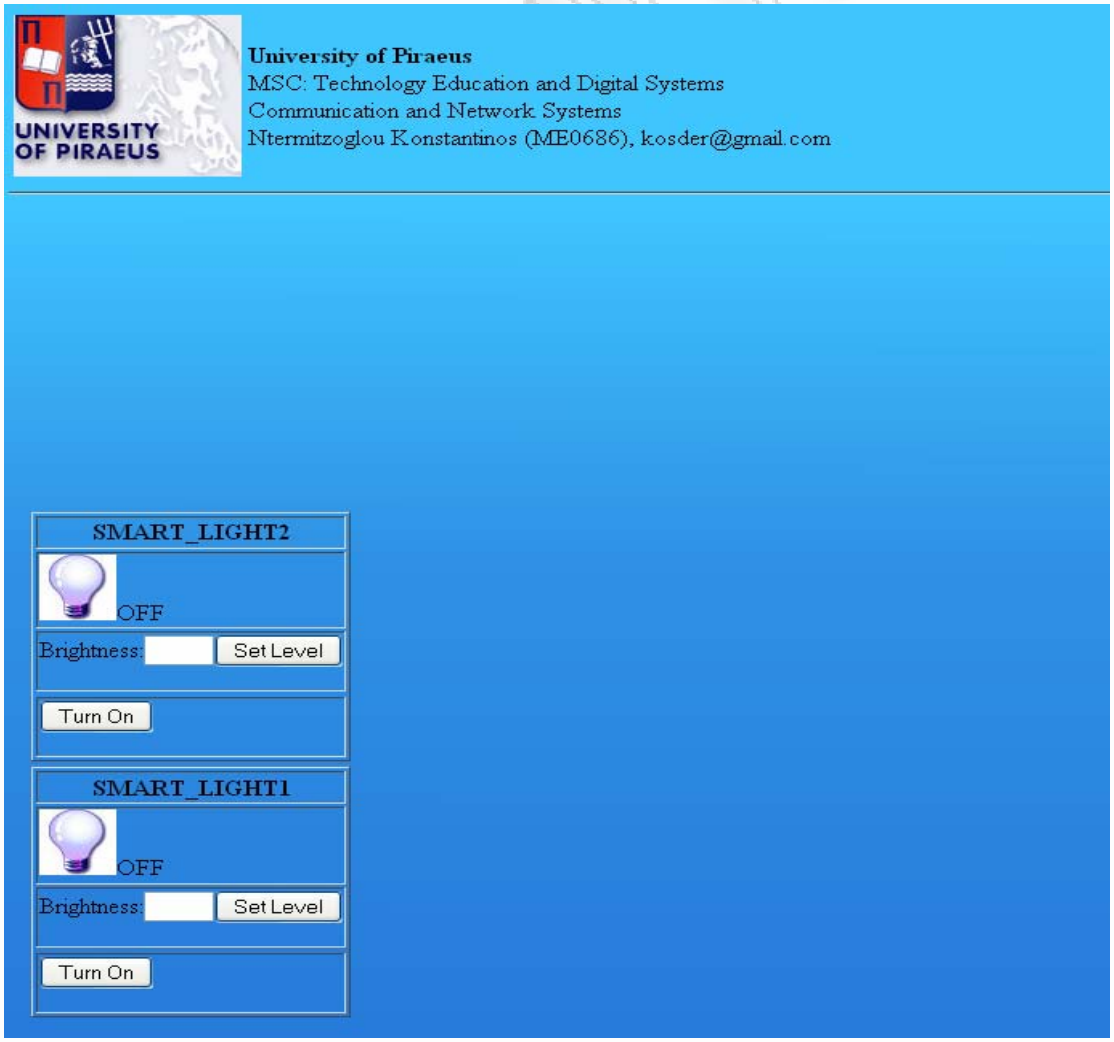
```
->ls
IP Device IDs List
----------------------
IPDevice 1 ID: 1
IPDevice 2 ID: 2
```

6.3 Εμφάνιση συσκευών στην πλατφόρμα

Στο σημείο αυτό αξίζει να δούμε τι συμβαίνει από την πλευρά της πλατφόρμας. Για να το κάνουμε αυτό, χρησιμοποιούμε κάποιον web Browser και χρησιμοποιούμε την διεύθυνση:

http://localhost:8080/display

που είναι η διεύθυνση στην οποία ακούει το Servlet GUI της πλατφόρμας μας. Θα έχουμε κάτι σαν την εικόνα που ακολουθεί:



Εικόνα 6.1: Αποτύπωση δύο IP Lights

Παρατηρούμε λοιπόν ότι πολύ σωστά έχουν δημιουργηθεί controls για δύο SMART Lights, τα οποία μπορούμε να χειριστούμε.

6.4 Ανάκαμψη από προβλήματα

Μπορεί κάποιος στο σημείο αυτό να αναρωτηθεί, τι θα συμβεί στην περίπτωση που για κάποιον λόγω η πλατφόρμα σταματήσει να λειτουργεί για κάποιο διάστημα και έπειτα ξαναρχίσει την λειτουργία της. Το πρωτόκολλο είναι δομημένο με τέτοιον τρόπο ώστε να ανακάμπτει από τέτοιες περιπτώσεις. Έστω λοιπόν ότι κλείνουμε την πλατφόρμα, και έπειτα σε λειτουργία. Τα ακόλουθα μηνύματα θα εμφανιστούν στην πλευρά του IP Device Maintenance :

Κατά την παύση της πλατφόρμας:

->
IPDevice 1 lost Connection with the OSGi.
IPDevice 2 lost Connection with the OSGi.
->

Και κατά την επανεκκίνηση της :

->
IPdevice 2 Connected to OSGi
Data sent to OSGi: <smart><msg type="hello" propName="type" propValue="LIGHT" />
</smart>
IPdevice 1 Connected to OSGi
Data sent to OSGi: <smart><msg type="hello" propName="type" propValue="LIGHT" />
</smart>
DEBUG: IPDevice 2 received command from OSGi
Parsing message <smart><msg type="get" propName="state" propValue="" /></smart>
Data sent to OSGi: <smart><msg type="update" propName="state" propValue="null" />
</smart>
DEBUG: IPDevice 2 received command from OSGi
DEBUG: IPDevice 1 received command from OSGi
Parsing message <smart><msg type="get" propName="level" propValue="" /></smart>
Parsing message <smart><msg type="get" propName="state" propValue="" /></smart>

Data sent to OSGi: <smart><msg type="update" propName="level" propValue="null" />
></smart>
Data sent to OSGi: <smart><msg type="update" propName="state" propValue="null" />
></smart>
DEBUG: IPDevice 1 received command from OSGi
Parsing message <smart><msg type="get" propName="level" propValue="" /></smart>
Data sent to OSGi: <smart><msg type="update" propName="level" propValue="null" />
></smart>
->

Και φυσικά οι συσκευές είναι ξανά λειτουργικές σαν να μην είχε συμβεί ποτέ κάποια αλλαγή.

Στο σημείο αυτό, αρχίζει να γίνεται εμφανής η δυναμικότητα της πλατφόρμας σχετικά με τον χειρισμό περιβαλλόντων που αλλάζουν με γρήγορους ρυθμούς. Αυτό θα φανεί ακόμα καλύτερα και στην συνέχεια όπου θα προστεθούν επιπλέον συσκευές και διαφορετικών τύπων, μερικές από τις οποίες και θα αφαιρέσουμε και θα ξανά προσθέσουμε.

6.5 Δημιουργία IP Sensors

Στο σημείο αυτό θα προσθέσουμε άλλες τρεις εικονικές συσκευές. Αυτή την φορά τύπου Sensors. Αυτό μπορεί να γίνει εκτελώντας στο IP Device Maintenance τρεις φορές την εντολή :

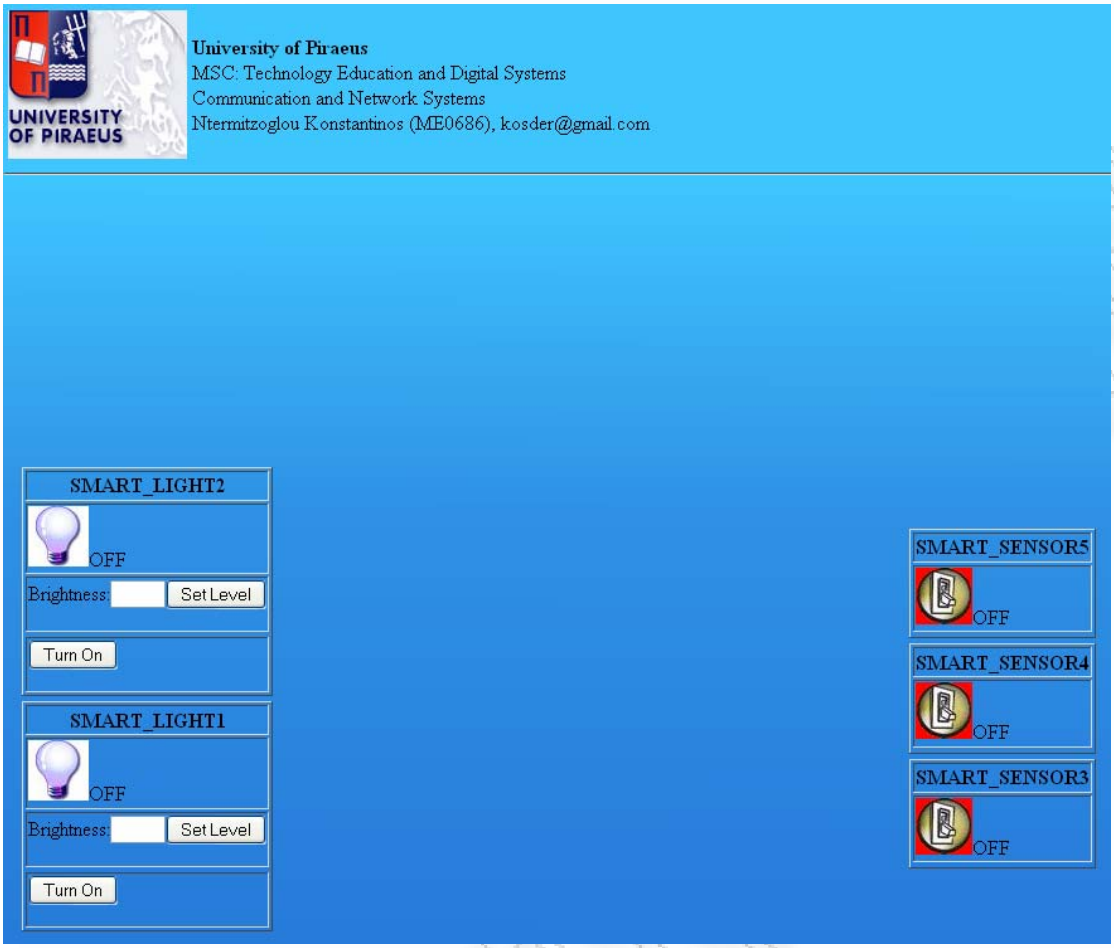
create SENSOR

Θα παρατηρήσουμε τα εξής στο IP Device Maintenance:

->**create SENSOR**
Created IPDevice with ID: 3
->IPdevice 3 Connected to OSGi
Data sent to OSGi: <smart><msg type="hello" propName="type" propValue="SENSOR" />
></smart>
DEBUG: IPDevice 3 received command from OSGi

Parsing message <smart><msg type="get" propName="state" propValue="" /></smart>
Data sent to OSGi: <smart><msg type="update" propName="state" propValue="null" />
</smart>
->**create SENSOR**
Created IPDevice with ID: 4
->IPdevice 4 Connected to OSGi
Data sent to OSGi: <smart><msg type="hello" propName="type" propValue="SENSOR" />
</smart>
DEBUG: IPDevice 4 received command from OSGi
Parsing message <smart><msg type="get" propName="state" propValue="" /></smart>
Data sent to OSGi: <smart><msg type="update" propName="state" propValue="null" />
</smart>
->**create SENSOR**
Created IPDevice with ID: 5
->IPdevice 5 Connected to OSGi
Data sent to OSGi: <smart><msg type="hello" propName="type" propValue="SENSOR" />
</smart>
DEBUG: IPDevice 5 received command from OSGi
Parsing message <smart><msg type="get" propName="state" propValue="" /></smart>
Data sent to OSGi: <smart><msg type="update" propName="state" propValue="null" />
</smart>
->

Ενώ στην πλατφόρμα θα είχαμε το αποτέλεσμα που απεικονίζεται στην εικόνα που ακολουθεί στην αμέσως επόμενη σελίδα.



Εικόνα 6.2: Πρόσθεση τριών IP Sensors

6.6 Αλλαγή ιδιοτήτων των συσκευών

Στο σημείο αυτό θα δούμε πως η πλατφόρμα διαχειρίζεται αλλαγές σε κάποια στοιχεία των συσκευών. Θα χρησιμοποιήσουμε το εργαλείο IP Device Maintenance, για να ανάψουμε έναν λαμπτήρα στο επίπεδο 55% και επίσης να θέσουμε έναν αισθητήρα στην θέση On. Αυτό το κάνουμε εκτελώντας διαδοχικά τις ακόλουθες εντολές:

set 1 state 1

set 1 level 55

set 5 state 1

Στον IP Device Maintenance θα είχαμε κάτι σαν:

```
->
->set 1 state 1
Data sent to OSGi: <smart><msg type="update" propName="state" propValue="1" /></
smart>
IPDevice 1 state = 1
->set 1 level 55
Data sent to OSGi: <smart><msg type="update" propName="level" propValue="55" /><
/smart>
IPDevice 1 level = 55
->
->
->set 5 state 1
Data sent to OSGi: <smart><msg type="update" propName="state" propValue="1" /></
smart>
IPDevice 5 state = 1
->
```

Ενώ από την πλευρά της πλατφόρμας:

```
-> ENQUEUED: <smart><msg type="update" propName="state" propValue="1" /></smart>
Parsing message <smart><msg type="update" propName="state" propValue="1" /></sma
rt> from device 2
SMART_LIGHT2 Received an update event! state 1
Setting state to 1
ENQUEUED: <smart><msg type="update" propName="level" propValue="55" /></smart>
Parsing message <smart><msg type="update" propName="level" propValue="55" /></sm
art> from device 2
SMART_LIGHT2 Received an update event! level 55
ENQUEUED: <smart><msg type="update" propName="state" propValue="1" /></smart>
Parsing message <smart><msg type="update" propName="state" propValue="1" /></sma
rt> from device 5
SMART_SENSOR5 Received an update event! state 1
Setting state to 1
```

Εικόνα 6.3: Αλλαγή μεταβλητών από το IP Device Maintenance

6.6.1 Χειρισμός συσκευών από την επιφάνεια χρήστη

Έως το σημείο αυτό είδαμε πως θα αντικατοπτριστούν οι αλλαγές στην πλατφόρμα, αν οι φυσικές (εικονικές στην περίπτωσή μας) μεταβάλλουν την κατάστασή τους.

Η πλατφόρμα προσφέρει την δυνατότητα χειρισμού για ορισμένες συσκευές. Στην περίπτωσή μας του λαμπτήρες. Σημειώνεται ότι οι Sensors για την πλατφόρμα σε αυτό το σημείο είναι παθητικές συσκευές. Θα αλλάξουμε λοιπόν την φωτεινότητα και του δεύτερου λαμπτήρα στο επίπεδο 100%, χρησιμοποιώντας τα χειριστήρια που προσφέρονται από την επιφάνεια χρήστη. Το αποτέλεσμα στον IP Device Maintenance θα ήταν τα εξής μηνύματα:

```
->
DEBUG: IPDevice 2 received command from OSGi
Parsing message <smart><msg type="set" propName="state" propValue="1" /></smart>
Data sent to OSGi: <smart><msg type="update" propName="state" propValue="1" /></smart>
```

DEBUG: IPDevice 2 received command from OSGi

Parsing message <smart><msg type="set" propName="level" propValue="100" /></smart>

Data sent to OSGi: <smart><msg type="update" propName="level" propValue="100" /></smart>

Και φυσικά οι αλλαγές αυτές θα αντικατοπτριστούν και στην επιφάνεια χρήση μετά την πραγματική αλλαγή τους στις φυσικές συσκευές, με την βοήθεια των "**update**" μηνυμάτων που αποστέλλονται από τον IP Device Maintenance. Έτσι το αποτέλεσμα στην επιφάνεια χρήστη θα ήταν αυτό που φαίνεται στην εικόνα που ακολουθεί :

Εικόνα 6.4: Χρήση GUI για αλλαγή κατάστασης

6.7 Αφαίρεση συσκευών on the run time

Η πλατφόρμα έχει σχεδιαστεί για να χειρίζεται SMART SPACES. Δηλαδή περιβάλλοντα που διαρκώς αλλάζουν κατάσταση. Μέχρι τώρα είδαμε πως διαχειρίζεται την ένταξη νέων συσκευών. Ας δούμε τι θα γίνει στην περίπτωση αφαίρεσης μερικών.

Συγκεκριμένα, ας αφαιρέσουμε έναν IP SENSOR και ένα IP LIGHT. Δίνουμε τις ακόλουθες εντολές στο IP Device Maintenance :

remove 1

remove 5

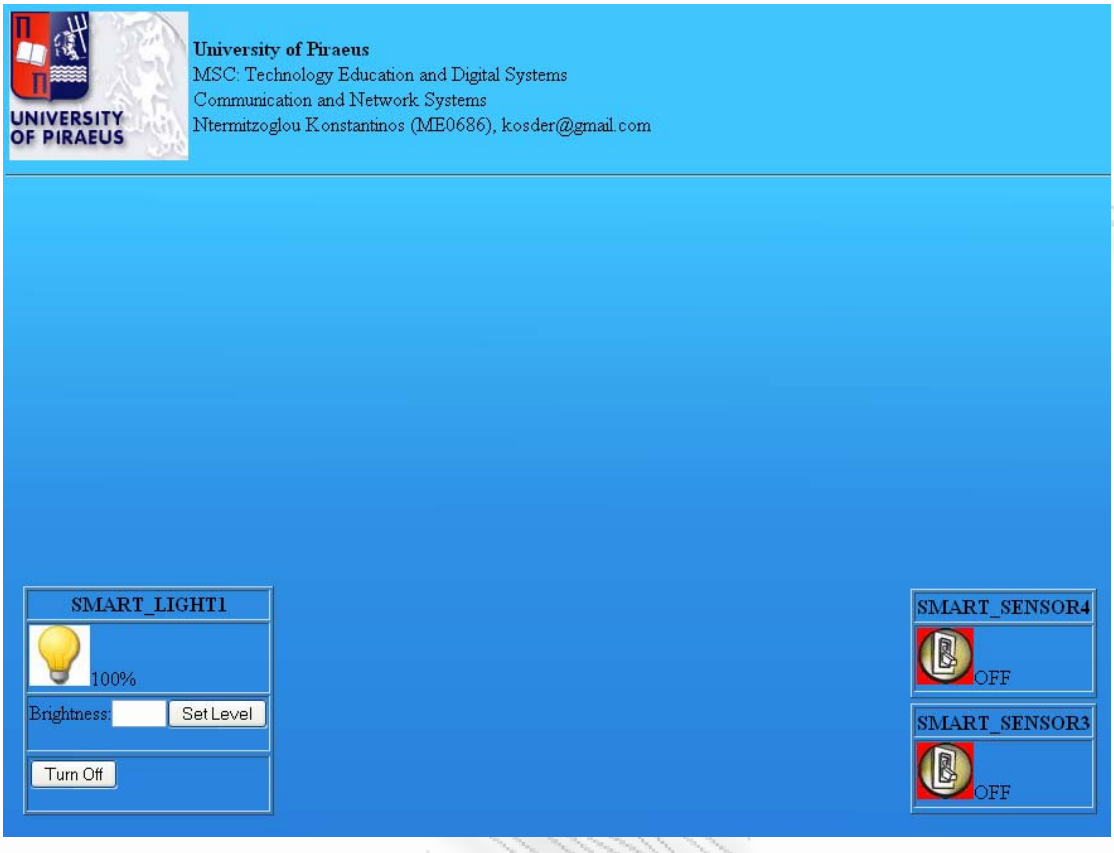
αυτό θα έχει σαν αποτέλεσμα τα ακόλουθα μηνύματα από την πλευρά του :

->**remove 1**
IPDevice 1 cleaning up
Removed IPDevice with ID: 1
-> IPDevice 1 lost Connection with the OSGi.

->**remove 5**
IPDevice 5 cleaning up
Removed IPDevice with ID: 5
-> IPDevice 5 lost Connection with the OSGi.

Ενώ από την πλευρά της πλατφόρμας:
-> [Device Manager] info: device lost, IPDEVICE48/19
-> [Device Manager] info: device lost, IPDEVICE60/19

Στο σημείο αυτό αξίζει να υπενθυμίσουμε για την ύπαρξη του OSGi Device Manager, του οποίου τα μηνύματα παρατηρούμε πρακτικά ακριβώς από επάνω.



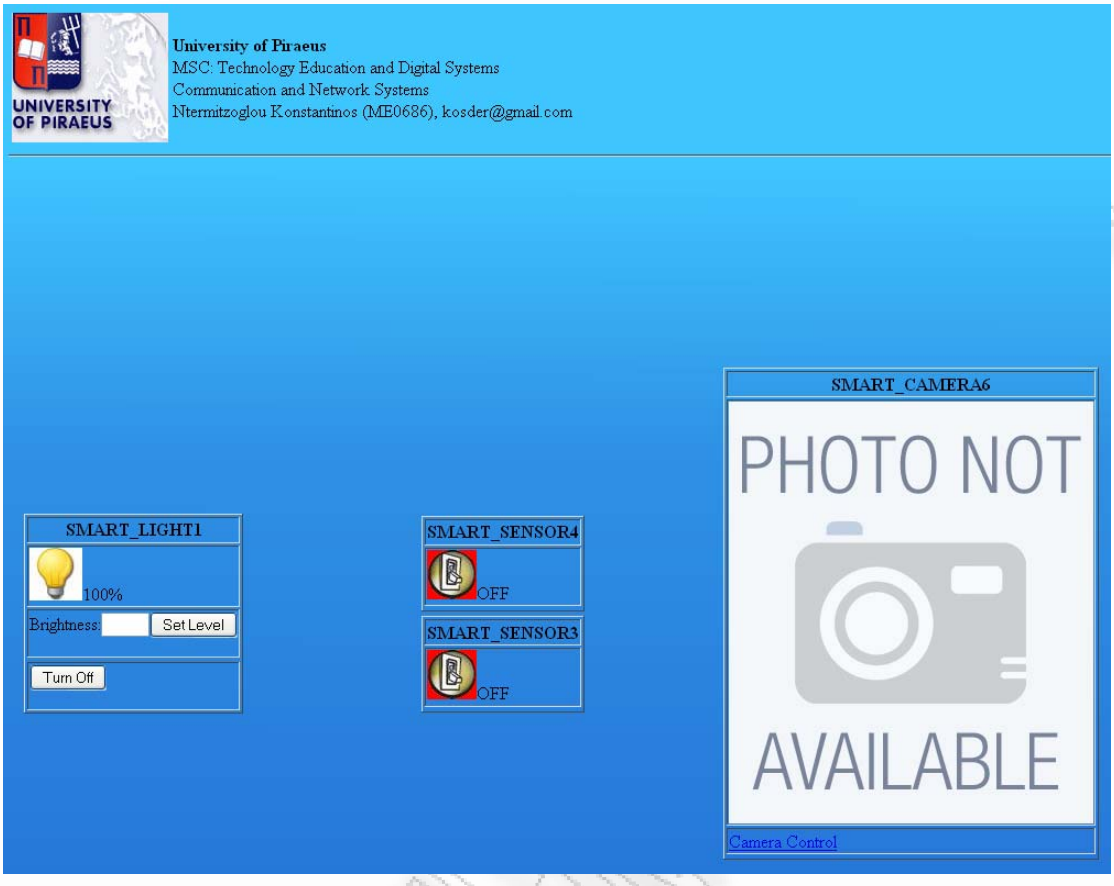
Εικόνα 6.5: Αφαίρεση συσκευών

6.8 Εισαγωγή IP Surveillance Camera

Στο σημείο αυτό θα εισάγουμε και θα χρησιμοποιήσουμε άλλη μία συσκευή τύπου IP enabled Surveillance Camera. Αρχικά, την δημιουργούμε χρησιμοποιώντας των IP Device Maintenance ως εξής :

create CAMERA

αυτή η εντολή από τη πλευρά της πλατφόρμας, μετά την ανάλυση των μηνυμάτων που θα επακολουθήσουν, θα είχε το αποτέλεσμα της απεικόνισης της εικόνας που καταγράφεται από μια IP Camera, όπως φαίνεται στην εικόνα που ακολουθεί.



Εικόνα 6.6: Εισαγωγή IP enabled Surveillance Camera

Στην εικόνα αυτή παρατηρούμε ότι η κάμερα προς το παρόν είναι ανενεργή. Πρέπει να την ενεργοποιήσουμε θέτοντας τις κατάλληλες τιμές στα properties που αναλύονται στο σχετικό κεφάλαιο, χρησιμοποιώντας την υπηρεσία OSGi Configuration Admin.

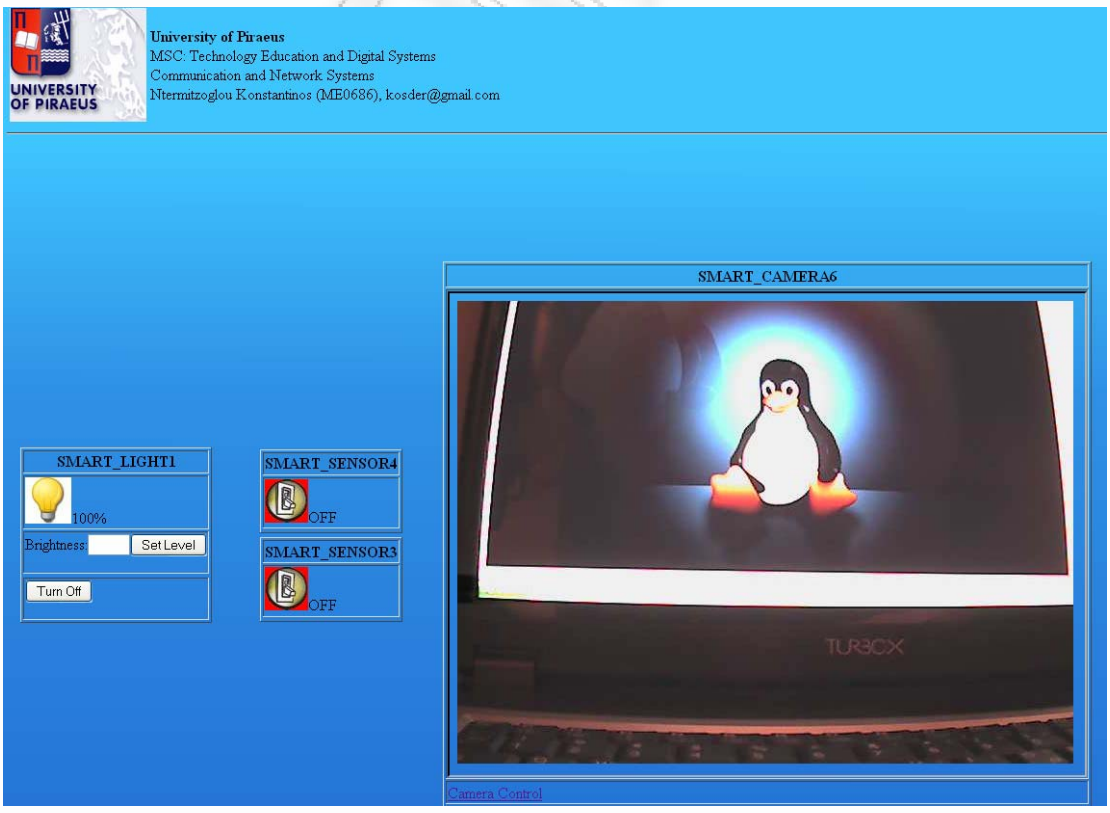
6.8.1 IP Camera set-up με την χρήση του CASinit

Στο σημείο αυτό θυμίζουμε την ύπαρξη της υπηρεσίας CASinit που έχει αναλυθεί σε προηγούμενο κεφάλαιο. Υπενθυμίζουμε ότι η υπηρεσία αυτή αναπτύχθηκε, εκτός των άλλων, για την ευκολία θέσης τιμών στον OSGi Configuration Admin. Θα χρησιμοποιήσουμε λοιπόν την υπηρεσία CASinit για την εκκίνηση λειτουργίας της κάμερας.

Στον κατάλογο που έχουμε ορίσει να παρατηρεί ο CASinit (oscar/casinit/conf/) για configuration files, προσθέτουμε ένα αρχείο με όνομα: **SMART_CAMERA6_ms.xml** με το παρακάτω περιεχόμενο:

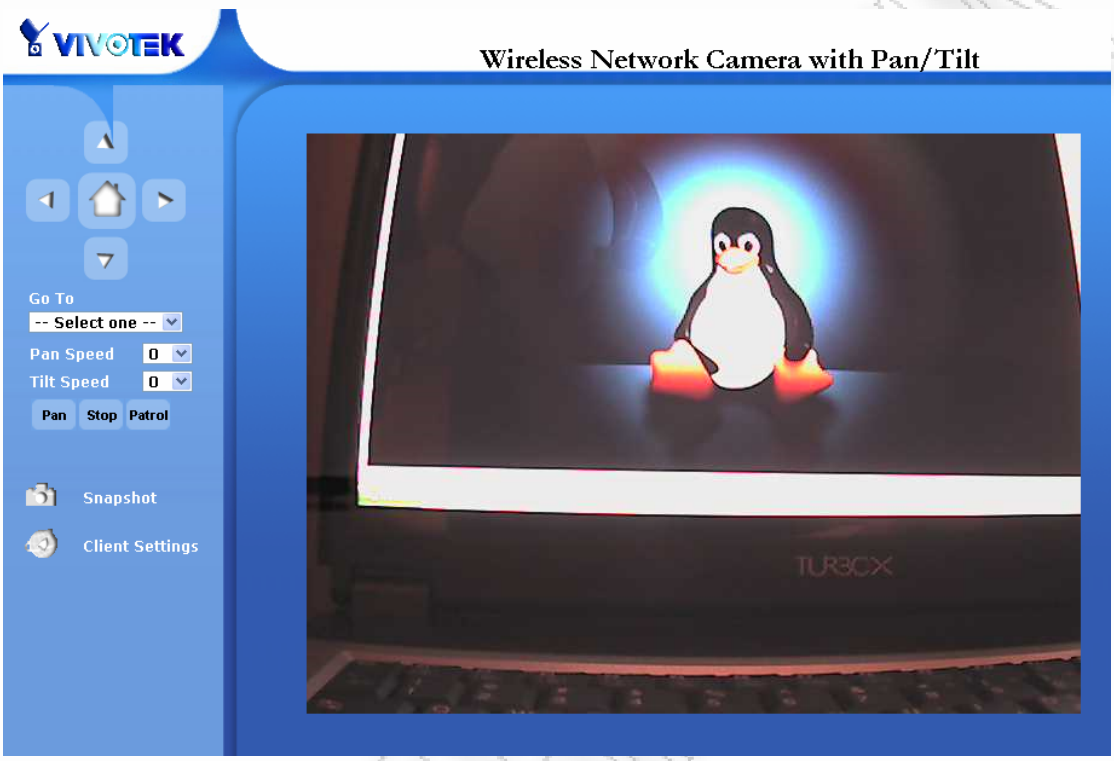
```
<simple>
        <service>
                <property name="presentation_url" type="String"
        value="http://192.168.2.86:80"/>
                <property name="stillimg_url" type="String"
        value="http://192.168.2.86:80/cgi-bin/video.jpg"/>
        </service>
</simple>
```

Εν συντομία, αυτό το αρχείο λέει στον IP Camera Driver της πλατφόρμας, ότι η συγκεκριμένη κάμερα (SMART_CAMERA6), χρησιμοποιεί σαν presentation URL το : **http://192.168.2.86:80** , ενώ την εικόνα που θα εμφανίζει θα την λαμβάνει από το URL : **http://192.168.2.86:80/cgi-bin/video.jpg**. Έτσι στην επιφάνεια χρήστη της πλατφόρμας θα εμφανιστεί το αποτέλεσμα που φαίνεται στην εικόνα που ακολουθεί :



Εικόνα 6.7: IP Camera σε λειτουργία

Επιπρόσθετα κάνοντας κλικ στο "camera controls" θα μεταφερθούμε στο presentation URL της κάμερας:



Εικόνα 6.8: Presentation URL

6.9 Πρόσθεση UPnP συσκευών

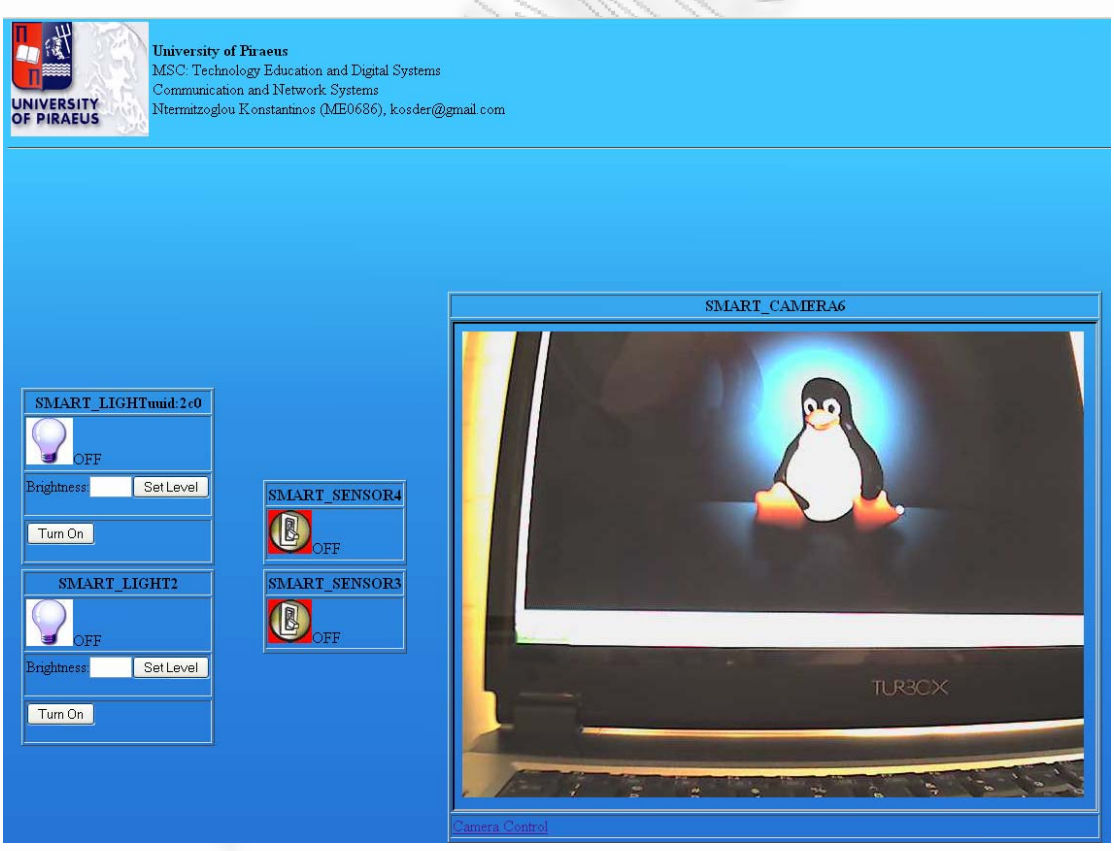
Μέχρι το σημείο αυτό έχουμε παρακολουθήσει τον τρόπο που διαχειρίζεται η πλατφόρμα την προσθαφαίρεση και λειτουργικότητα των συσκευών που ακολουθούν το IP πρωτόκολλο που έχουμε ορίσει ειδικά για αυτού του τύπου τις συσκευές.

Στις βασικές δυνατότητες τις πλατφόρμας όμως, συγκαταλέγεται και η λειτουργικότητα της απόκρυψης τεχνολογίας των φυσικών συσκευών και η κοινή τους διαχείριση από την πλευρά του χρήστη. Δηλαδή η απόκρυψη ή διαφάνεια της τεχνολογίας των συσκευών. Για την παρουσίαση λοιπόν αυτής της λειτουργικότητας, θα χρησιμοποιήσουμε το σύνολο του Software που αναπτύχθηκε, για την διαχείριση ενός UPnP enabled λαμπτήρα.

Για την παρουσίαση αυτής της περίπτωσης θα χρησιμοποιηθούν τα Intel UPnP tools, για την δημιουργία ενός εικονικού λαμπτήρα τέτοιου τύπου, που προσομοιώνει τα UPnP μηνύματα στο δίκτυο. Δημιουργούμε λοιπόν έναν λαμπτήρα που είναι αρχικά κλειστός και σε επίπεδο φωτεινότητας μηδέν. Θα πρέπει να δούμε ένα παράθυρο της ακόλουθης μορφής:

Εικόνα 6.9: UPnP Light

Κατά την δημιουργία του ο λαμπτήρας, δημιουργεί στο δίκτυο κάποια UPnP μηνύματα, γνωστά σαν "announcement" μηνύματα, που πληροφορούν τους ενδιαφερομένους για την ύπαρξή του. Τα κατάλληλα bundles του OSGi που έχουν αναπτυχθεί και χρησιμοποιούνται, μεταφράζουν αυτά τα μηνύματα σε άλλη μια συσκευή τύπου SMART Light.



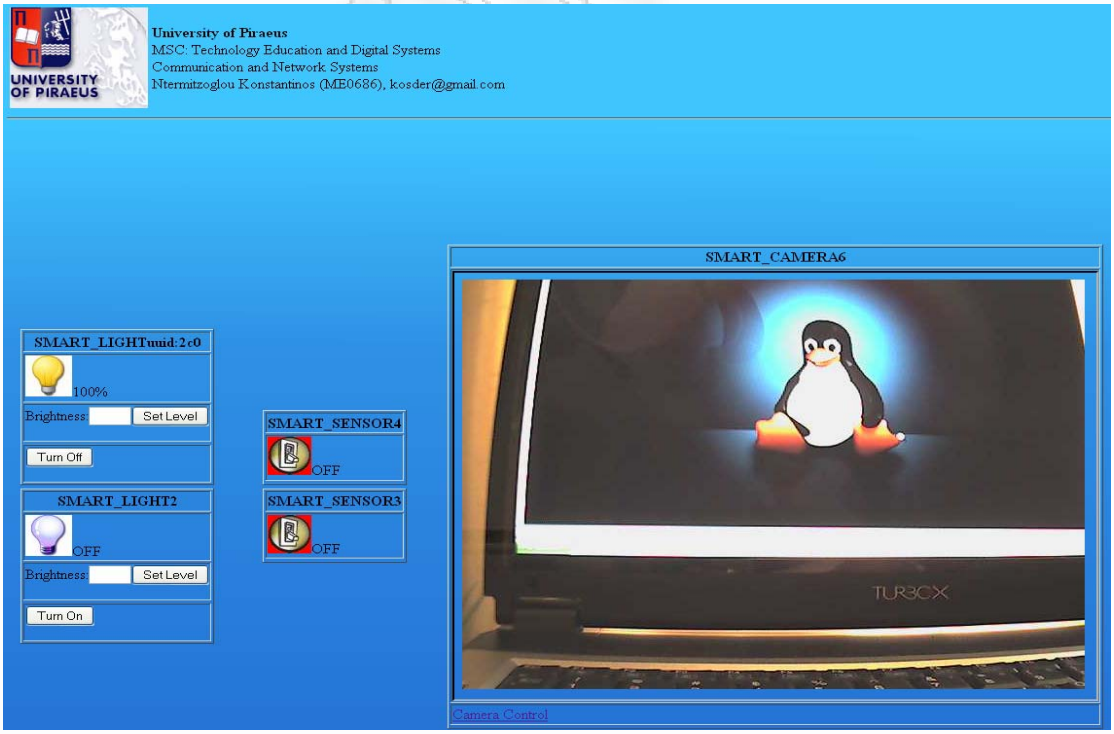
Εικόνα 6.10: Εισαγωγή ενός UPnP Light

Την συσκευή αυτή μπορούμε πλέον να την χειριστούμε όπως οποιαδήποτε άλλη

SMART συσκευή αυτού του τύπου. Για το παράδειγμά μας λοιπόν, θα ανάψουμε τον λαμπτήρα χρησιμοποιώντας το GUI της πλατφόρμας. Με την ενέργειά μας αυτή θα δημιουργηθούν τα κατάλληλα UPnP μηνύματα στο δίκτυο και ο UPnP λαμπτήρας μας θα έχει την παρακάτω αντίδραση:

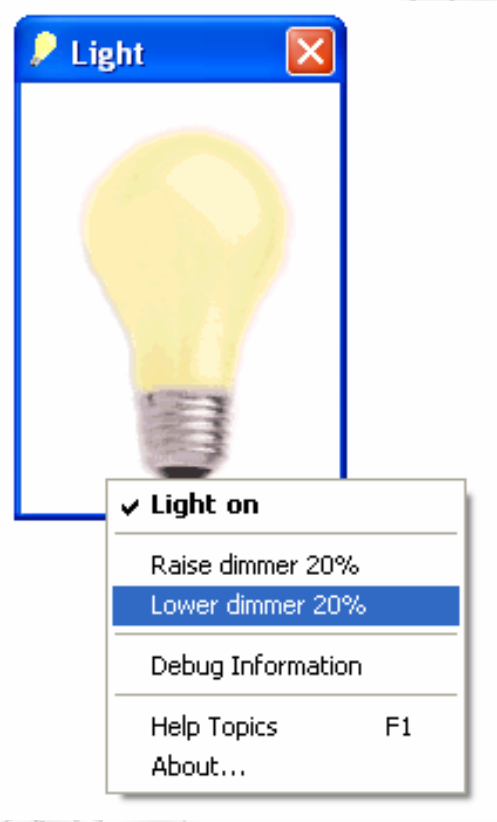
Εικόνα 6.11: UPnP λαμπτήρας σε επίπεδο 100%

Φυσικά, μετά το ανάμα του λαμπτήρα, θα δημιουργηθούν από την πλευρά του τα κατάλληλα UPnP μηνύματα, προς ενημέρωση για την παρούσα κατάστασή του, τα οποία θα ακούσει η πλατφόρμα και θα αντικατοπτρίσει της αλλαγές:



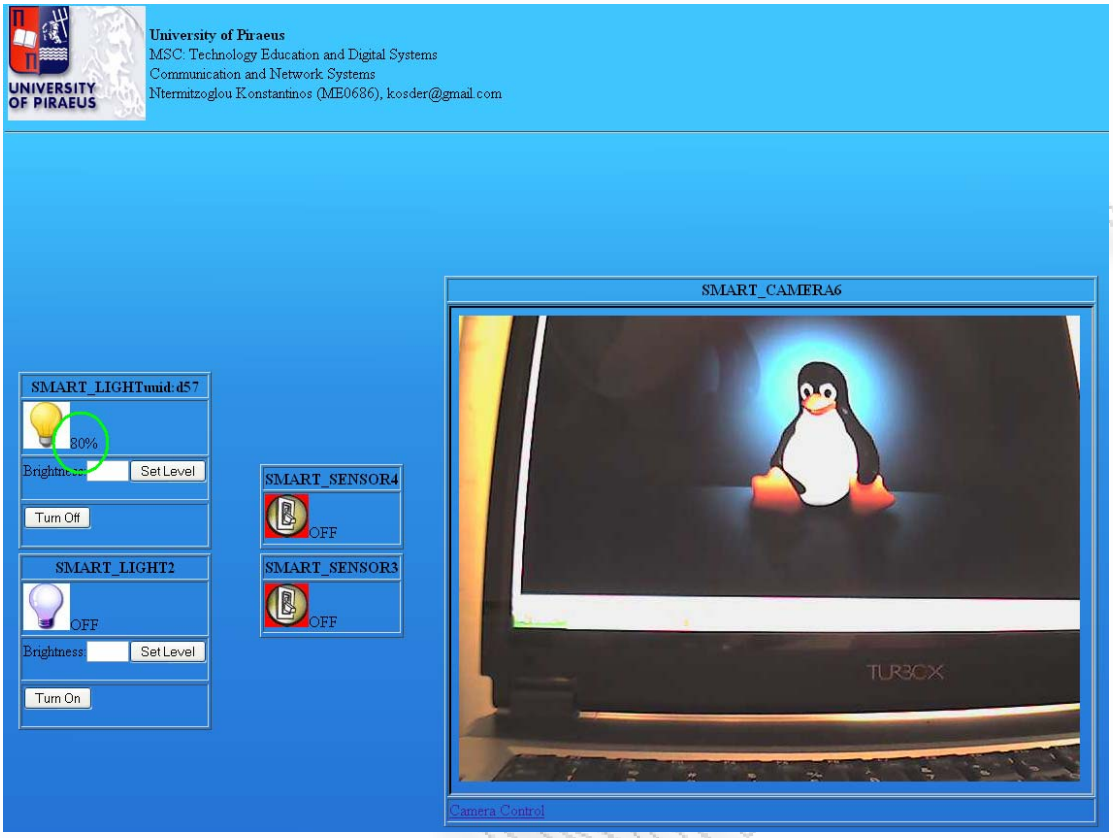
Εικόνα 6.12: Φωτεινότητα UPnP λαμπτήρα στο GUI

Στο τελευταίο παράδειγμα αναφορικά με τα UPnP μηνύματα, θα χρησιμοποιήσουμε και τον αντίθετο μονοπάτι. Θα αλλάξουμε δηλαδή την κατάσταση του UPnP λαμπτήρα με την χρήση των Intel tools και θα δούμε πως αυτό αντικατοπτρίζεται στην πλατφόρμα. Ας θέσουμε λοιπόν το επίπεδο φωτεινότητας στο 80% με αυτόν τον τρόπο.



Εικόνα 6.13: Μείωση φωτεινότητας UPnP Light στο 80%

Το OSGi Framework, με τα κατάλληλα Bundles που διαθέτει, θα αναλύσει τα UPnP μηνύματα, τα οποία εν συνεχεία θα έχουν ως αποτέλεσμα την αλλαγή των ιδιοτήτων συγκεκριμένων στοιχείων της πλατφόρμας, συγκεκριμένα της φωτεινότητας του κατάλληλου SMART λαμπτήρα, όπως φαίνεται και στην εικόνα που ακολουθεί:



Εικόνα 6.14: Επίπεδο φωτεινότητας UPnP Light στο GUI

Σημείωση:

Ο ενδιαφερόμενος αναγνώστης σχετικά με τα UPnP μηνύματα που ανταλλάσσονται, μπορεί να χρησιμοποιήσει το εργαλείο UPnP Device Sniffer του πακέτου των Intel UPnP tool, με τον τρόπο που περιγράφεται στο σχετικό παράρτημα.

6.10 Χρήση του Bundle Inter Device Interaction Service

Όπως έχει προαναφερθεί αρκετές φορές, ένας από τους λειτουργικότερους στόχους της πλατφόρμας είναι η διαφανής λειτουργία και αλληλεπίδραση των συσκευών που ανήκουν σε αυτήν, ανεξαρτήτως τεχνολογίας. Στο σημείο λοιπόν αυτό για την επίδειξη της ικανότητας αυτής, θα χρησιμοποιηθεί η υπηρεσία που αναπτύχθηκε για αυτό τον σκοπό, Inter Device Interaction Service.

Θυμίζουμε, ότι η υπηρεσία αυτή χρησιμοποιεί ένα σύνολο κανόνων, με σκοπό να αλλάξει τις τιμές συγκεκριμένων properties κάποιων συσκευών, ανάλογα με την κατάσταση κάποιων άλλων. Στο παράδειγμά μας, θα προσπαθήσουμε να μιμηθούμε ένα απλό σύστημα

συναγερμού, κάτι πολύ διαδεδομένο σε ένα περιβάλλον SMART HOME.

Πιο συγκεκριμένα, θέλουμε όταν ο υποτιθέμενος ανιχνευτής κίνησης SMART_SENSOR4 που έστω ότι διαθέτουμε, ανιχνεύσει κάποια κίνηση, να ανάψει ο UPnP λαμπτήρας στο επίπεδο 100%. Δημιουργούμε λοιπόν το σύνολο των κανόνων που μας ενδιαφέρει με το εισάγουμε στην πλατφόρμα με την χρήση του βοηθητικού bundle CASinit που έχει δημιουργηθεί. Πιο συγκεκριμένα, δημιουργούμε το αρχείο με όνομα "**unipi.smart.event.msf.xml**", κάτω από τον φάκελο παρατήρησης των configuration files του CASinit, με το ακόλουθο περιεχόμενο:

```
<factory>
        <service>

      <property name="name" value="Alarm" type="String"/>
                <property name="association_type" type="String" value="OR"/>
                <property name="sensor_events" type="Vector" oftype="String">
                        <data value="SMART_SENSOR4$SMART_SENSOR$state$1"/>
                </property>
                <property name="actuator_actions" type="Vector" oftype="String">
                        <data                              value="SMART_LIGHTuuid:20a235d1-75de-4a93-a3e1-
5cbb2c80d36b$SMART_LIGHT$level$100"/>
                </property>

        </service>
   <key name="name"/>
</factory>
```

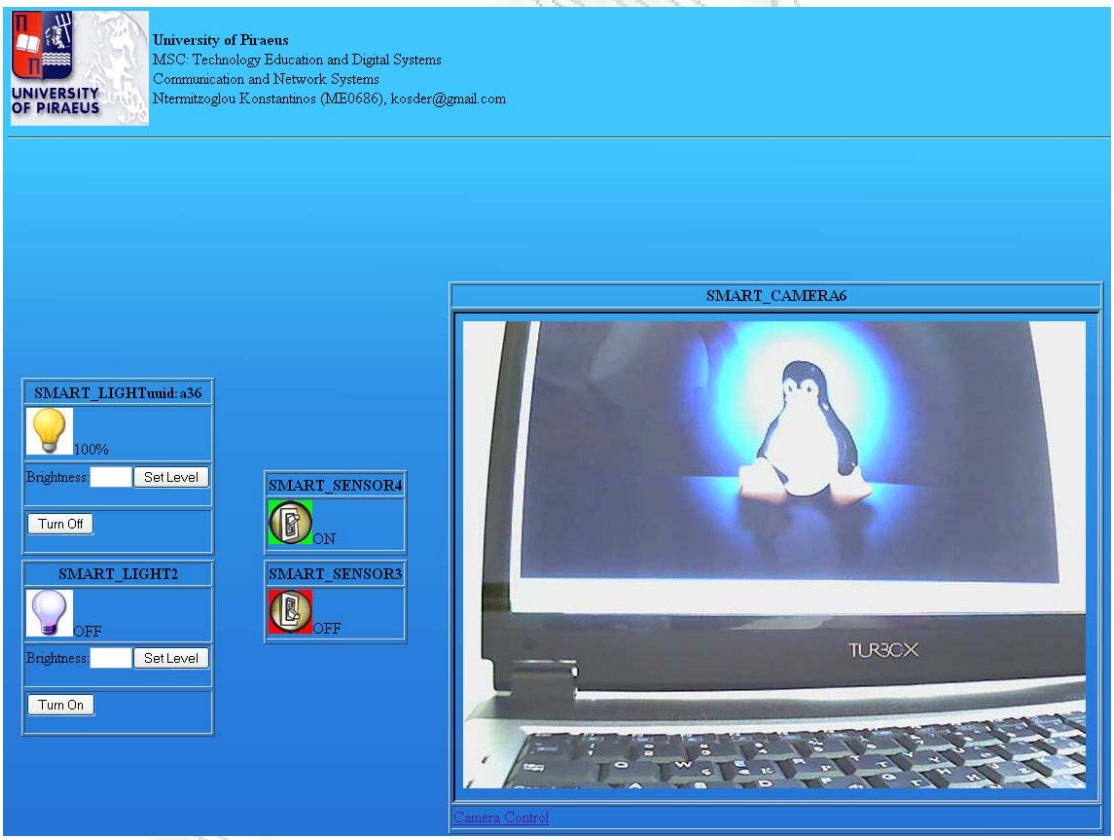
Χρησιμοποιούμε λοιπόν το λογισμικό IP Device Maintenance, για να θέσουμε τον SMART_SENSOR4 στην θέση on γράφοντας την εντολή:

set 4 state 1

Αυτή η κίνηση θα έχει σαν αποτέλεσμα την εφαρμογή των κανόνων που θέσαμε στην υπηρεσία Inter Device Interaction Service και θα παρατηρήσουμε την εικόνα του UPnP Light στα Intel UPnP να γίνεται:

Εικόνα 6.15: UPnP Light σε επίπεδο 100% με τη χρήση I.D.I.S.

Και φυσικά, οι αλλαγές στο GUI της πλατφόρμας θα είναι επίσης εμφανής μετά την ενεργοποίηση των κανόνων:



Εικόνα 6.16: Το GUI της πλατφόρμας μετά την χρήση του I.D.I.S.

Στο σημείο αυτό λοιπόν, μπορούμε να παρατηρήσουμε, πως με την βοήθεια της πλατφόρμας που έχει αναπτυχθεί, δυο συσκευές λειτουργικότητα εντελώς διαφορετική μεταξύ τους, που χρησιμοποιούν δύο τελείως διαφορετικά πρωτόκολλα επικοινωνίας, μπορούν να αλληλεπιδράσουν μεταξύ τους για την επίτευξη ενός κοινού στόχου.

Κεφάλαιο 7ο

Συμπεράσματα – Προτάσεις

7.1 Γενικά

Στα πλαίσια της διπλωματικής αυτής εργασίας αναπτύχθηκε ένα πακέτο λογισμικού, με σκοπό την διαχείριση συσκευών σε ένα περιβάλλον Smart Space, κάνοντας χρήση των υπηρεσιών του OSGi Framework.

Η αρχιτεκτονική που χρησιμοποιήθηκε, σχεδιάστηκε με τέτοιον τρόπο ώστε να γίνει η καλύτερη δυνατή χρήση των υπηρεσιών OSGi για την βέλτιστη διαχείριση των συσκευών, ενώ παράλληλα το σύστημα να είναι αρκετά δυναμικό, ώστε να μπορεί να αντέξει στον χρόνο, μέσα από την εύκολη και γρήγορη αναβάθμιση του. Ας δούμε αναλυτικά τους στόχους που επιτεύχθηκαν και τις προτάσεις για μελλοντική βελτίωση του συστήματος.

7.2 Στόχοι που επιτεύχθηκαν

Με την περάτωση της υλοποίησης του λογισμικού και μετά από μια σειρά δοκιμών, φαίνεται ότι η πλατφόρμα επιτυγχάνει σε μεγάλο βαθμό τους αρχικούς στόχους για τους οποίους σχεδιάστηκε. Συγκεκριμένα, όπως παρουσιάστηκε και στο σχετικό κεφάλαιο παρουσίασης της πλατφόρμας, φάνηκε πως είναι δυνατόν να χρησιμοποιηθεί η τεχνολογία OSGi για την διαχείριση συστημάτων Smart Spaces.

Οι συσκευές μπορούν εύκολα να εισέρχονται και να εξέρχονται από το περιβάλλον, διάφορες μεταβλητές τους μπορούν να αλλάζουν κατά βούληση on the runtime, η τεχνολογία των φυσικών συσκευών είναι διαφανής στον χρήστη και αυτές να μπορούν να αλληλεπιδρούν μεταξύ τους και φυσικά εξωτερικές οντότητες όπως για παράδειγμα το User Interface μπορούν αν αλληλεπιδράσουν εύκολα με το OSGi.

Επιπρόσθετα, λόγω της φύσης του συστήματος, σημαντική προϋπόθεση ήταν εύκολη αναβάθμισή του, δηλαδή η ευκολία ανάπτυξης κομματιών λογισμικού (bundles) ώστε να υποστηρίζονται νέες τεχνολογίες. Στο σημείο αυτό αξίζει να αναφερθεί, ότι από την προσωπική εμπειρία του συγγραφέα στα συστήματα OSGi, αυτό είναι δυνατό. Για παράδειγμα,

η εισαγωγή του κατάλληλου bundle για την διαχείριση των UPnP Lights, απαίτησε χρόνο περίπου μιας εργατοημέρας, στον οποίο συμπεριλαμβάνεται ο χρόνος ανάγνωσης και κατανόησης του κεφαλαίου του OSGi Specification για την υπηρεσία UPnP.

Επιπλέον, η χρήση της τεχνολογίας Java αν και προσθέτει κάποια επιβάρυνση στην απόδοση του συστήματος λόγω της διερμήνευσης, παρέχει μεγάλη ευκολία στην μεταφερσιμότητα της πλατφόρμας, μεταξύ ετερογενών συστημάτων. Επιπρόσθετα, το memory footprint της πλατφόρμας φαίνεται να είναι αρκετά μικρό και σε συνδυασμό με την δυνατότητα χρήσης JNI για εξειδικευμένους drivers, το σύστημα θα μπορούσε να χρησιμοποιηθεί σε embedded περιβάλλοντα.

Τέλος, αξίζει να αναφερθεί ότι η εταιρία **inAccessNetworks S.A.**, η οποία δραστηριοποιείται κυρίως στο Ελλαδικό χώρο αλλά και στο εξωτερικό, με την διαχείριση συστημάτων Smart Homes, χρησιμοποιεί το μοντέλο OSGi. Περισσότερες πληροφορίες σχετικά στο http://www/inaccessnetworks.com. Συνεπώς, η χρήση OSGi για τέτοια χρήση, πραγματικά φαίνεται να ενδείκνυται και σε πραγματικά περιβάλλοντα εκτός των ερευνητικών.

7.3 Προτάσεις Επιπλέον ανάπτυξης

Εφόσον το σύστημα αναπτύχθηκε στα πλαίσια μιας διπλωματικής εργασίας, είχε σαν κύριο στόχο την παρουσίαση των δυνατοτήτων του OSGi για την διαχείριση Smart Spaces. Συνεπώς, το λογισμικό που αναπτύχθηκε δεν αντιπροσωπεύει δεν στοχεύει στην αναπαράσταση του πραγματικού κόσμου, ούτε στην αυτόματη μεταφορά του σε αυτόν. Για παράδειγμα δεν είναι δυνατόν να χρησιμοποιηθεί ως έχει σε ένα Smart Home.

Παρόλα αυτά, παρέχονται όλες οι δυνατότητες στον ενδιαφερομένους να το εμπλουτίσουν κατά βούληση. Προτείνεται λοιπόν, ο εμπλουτισμός των SMART interfaces, που είναι αυτά που αντιπροσωπεύουν τις συσκευές στον πραγματικό κόσμο, ανάλογα με την περίσταση του ενδιαφερομένου.

Επιπρόσθετα, υπηρεσίες όπως η Inter Device Interaction Service, μπορούν να αναπτυχθούν εκ νέου, ώστε να περιλαμβάνουν σύνταξη πιο σύνθετων κανόνων και ίσως μια επιφάνεια χρήστη, μιας και στην περίπτωσή μας πρόκειται για ένα command line tool.

Επίσης, προτείνεται η αναβάθμιση του παρόντος User interface με την χρήση της

τεχνολογίας Ajax με την χρήση του Google Web Toolkit (GWT). Ή εναλλακτικά η συγγραφή νέου με την χρήση της τεχνολογίας Adobe Flash για αποτελέσματα με καλύτερη αισθητική αν πρόκειται για ένα τελικό προϊόν.

Τέλος, κατά την διάρκεια ανάπτυξης της εργασίας, δεν υπήρχε η δυνατότητα χρήσης πραγματικών Smart Devices που κυκλοφορούν στην αγορά. Για τους ενδιαφερόμενους, προτείνεται η χρήση συσκευών PowerLine, με χρήση της τεχνολογίας LonTalk.

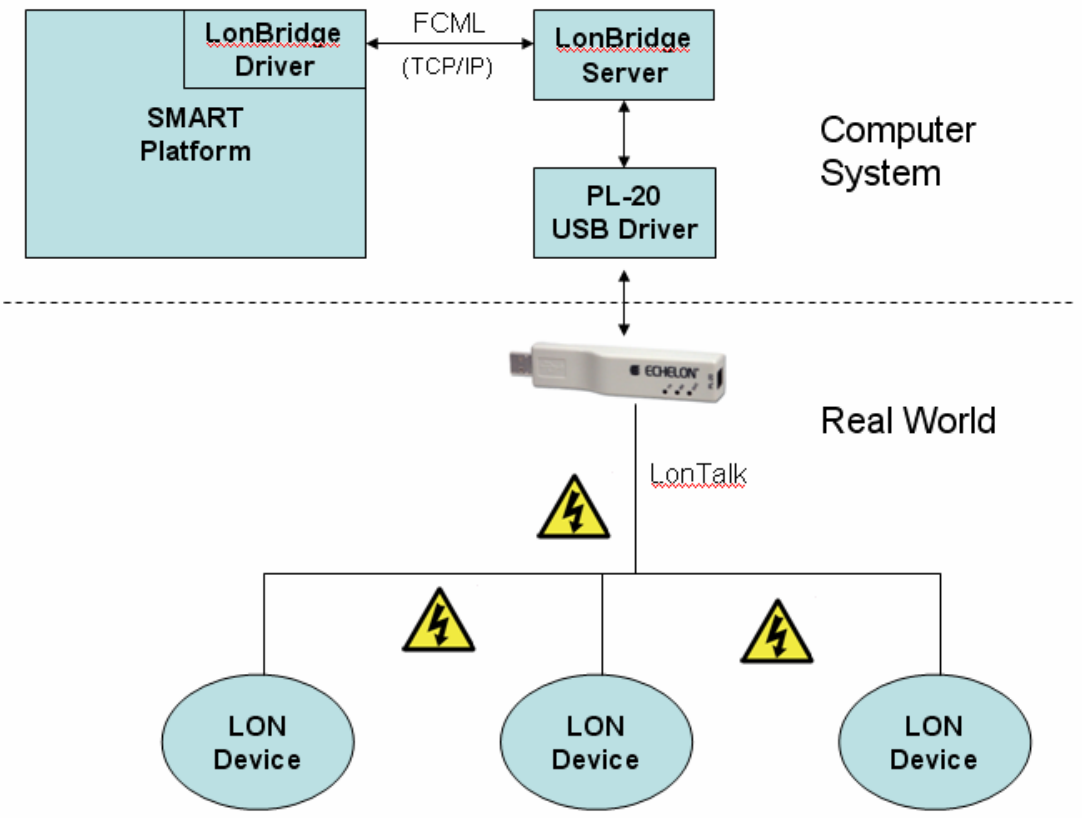
7.3.1 Τεχνολογία LON

Επειδή είναι πολύ πιθανό, ο αναγνώστης να ενδιαφερθεί για την χρήση πραγματικών συσκευών, κρίνεται σημαντικό να γίνει στο σημείο αυτό μια αναφορά στην τεχνολογία LON, ώστε να δοθεί του μια πρώτη κατεύθυνση. Πληροφορίες σχετικά με την τεχνολογία αυτή μπορούν να βρεθούν στο http://www.echelon.com .

Προτείνεται η χρήση της τεχνολογίας LON PowerLine (LON-PL). Αυτή η τεχνολογία δίνει την δυνατότητα σε συγκεκριμένες συσκευές που υλοποιούν το πρωτόκολλο LonTalk, να επικοινωνούν μεταξύ τους μέσα από την καλωδίωση του ηλεκτρικού ρεύματος.

Στην περίπτωσή μας, χρειαζόμαστε έναν τρόπο, ώστε η πλατφόρμα μας να μπορεί να επικοινωνεί με το πρωτόκολλο αυτό, με τις πραγματικές συσκευές. Προτείνεται η χρήση της USB συσκευής LON-PL20, για την οποία μπορούν να βρεθούν πληροφορίες στο [6].

Ακολουθεί μια εικόνα που παρουσιάζει σε υψηλό επίπεδο πως θα μπορούσε να χρησιμοποιηθεί αυτή η συσκευή σε συνεργασία με την πλατφόρμα που έχει αναπτυχθεί, ώστε να γίνει δυνατή η διαχείριση πραγματικών συσκευών που ακολουθούν το πρωτόκολλο LonTalk, με την χρήση της υπάρχουσας ηλεκτρικής καλωδίωσης μιας εγκατάστασης, π.χ. ενός σπιτιού.



Εικόνα 7.1: Χρήση του LON-PL20

Εν συντομία, αυτή η εικόνα περιγράφει τον τρόπο επικοινωνίας από την πλευρά της πλατφόρμας μας, με της πραγματικές συσκευές κάνοντας χρήση του εξαρτήματος LON-PL20.

Στην εικόνα αυτή βλέπουμε ότι η συσκευή αυτή συνδέεται σε μία θύρα USB του υπολογιστικού συστήματος πάνω στο οποίο εκτελείται η πλατφόρμα. Από την άλλη πλευρά συνδέεται με το δίκτυο ρεύματος της εγκατάστασης, στο οποίο είναι επίσης συνδεδεμένες οι LonTalk aware συσκευές. Συνεπώς, είναι δυνατή η επικοινωνία PL-20 με τις συσκευές.

Από την πλευρά του υπολογιστικού συστήματος, υπάρχει ένας Driver του συστήματος, ο οποίος διαχειρίζεται την συσκευή PL-20. Τον driver αυτών διαχειρίζεται ένα κομμάτι λογισμικού, γνωστό σαν LonBridge Server [7]. Το λογισμικό αυτό παρέχεται από την εταιρία 4Home σε συνεργασία με την Echelon.

Για περισσότερες πληροφορίες:

- http://www.echelon.com
- http://www.4home.com

Το λογισμικό LonBridge Server, είναι στην πραγματικότητα ένας Server στο οποίο συνδέονται πελάτες και επικοινωνούν μεταξύ τους με την χρήση του πρωτοκόλλου FCML, που έχει ορίσει η 4Home. Το πρωτόκολλο αυτό δίνει την δυνατότητα στους πελάτες του LonBridge Server, να ζητήσουν στοιχεία σχετικά με τις συσκευές του συστήματος, όπως και να αλλάξουν τις ιδιότητες τους, π.χ. την φωτεινότητα.

Το πρωτόκολλο αυτό, είναι σχετικά απλό, property based και χρησιμοποιεί το XML standard. Ομοιάζει με το πρωτόκολλο που υλοποιεί η οντότητα IP Device Maintenance της πλατφόρμας.

Φτάνοντας λοιπόν στο ζητούμενο, δηλαδή πως πρέπει να αναβαθμιστεί η πλατφόρμα μας, ώστε να χρησιμοποιήσει πραγματικές συσκευές με αυτές τις τεχνολογίες, από το σχήμα φαίνεται ότι **αρκεί η δημιουργία ενός OSGi Driver**, ο οποίος θα γνωρίζει το πρωτόκολλο FCML, θα συνδέεται σαν πελάτης στον LonBridge Server, και τελικά θα εμφανίζει στο OSGi Framework, SMART Devices ανάλογα με τα μηνύματα δέχεται. Αξίζει να αναφερθεί για ακόμα μια φορά, η ευκολία που παρέχει λοιπόν το σύστημα σχετικά με την υποστήριξη νέων τεχνολογιών.

Παράρτημα 1:

OSGi Services [8]

Standard Services

Βασισμένο στο OSGi Framework, το OSGi Alliance έχει προδιαγράψει έναν αριθμό από υπηρεσίες. Οι υπηρεσίες αυτές προδιαγράφονται με την χρήση Java Interfaces. Τα Bundles μπορούν να υλοποιούν αυτά τα Interfaces και να εγγράφουν (register) την υπηρεσία στο Service Registry του OSGi. Οι πελάτες της υπηρεσίας, μπορούν να τις ανακτήσουν από το OSGi Registry ή να αντιδρούν όταν αυτές εμφανίζονται και αποσύρονται.

Αυτή η τεχνική ομοιάζει με την Service-Oriented αρχιτεκτονική που έκανε διάσημες τις Web Services. Η ειδοποιός διαφορά μεταξύ των OSGi Services και των Web Services, είναι ότι οι τελευταίες πάντα απαιτούν ένα επίπεδο μεταφοράς (transport layer) που τις κάνουν χιλιάδες φορές πιο αργές από τις OSGi Services που χρησιμοποιούν απευθείας καλέσματα μεθόδων. Επιπλέον, τα στοιχεία του OSGi μπορούν να αντιδρούν άμεσα κατά την εμφάνιση ή απόσυρση υπηρεσιών.

Οι επόμενες ενότητες δίνουν μια σύντομη περιγραφή των OSGi release 4 υπηρεσιών. Περισσότερες πληροφορίες μπορούν να βρεθούν στο σχετικό έγγραφο. Σημειώνεται ότι η κάθε υπηρεσία περιγράφεται αφαιρετικά και υλοποιείται διαφορετικά από τον κάθε κατασκευαστή.

Framework Services

Το OSGi Framework, παρέχει τις υπηρεσίες Permission Admin Service, Package Admin Service και Start Level Service. Αυτές οι υπηρεσίες είναι ένα (προαιρετικό) κομμάτι του OSGi Framework. Ακολουθούν επιγραμματικά με μια σύντομη ανάλυση οι OSGi Frame work Services:

- **(Conditional) Permission Admin:** Οι άδειες των υπαρχόντων και μελλοντικών bundles διαχειρίζονται μέσα από αυτή την υπηρεσία. Οι άδειες ενεργοποιούνται αμέσως αφού τεθούν

- **Package Admin:** Τα Bundles μοιράζονται πακέτα που περιέχουν classes και πόρους. Η ανανέωση ενός bundles, ίσως απαιτεί την επανεκτίμηση των εξαρτήσεων μεταξύ των bundles. Αυτή η υπηρεσία παρέχει πληροφορίες σχετικά με την πραγματική κατάσταση διαμοίρασης πακέτων στο σύστημα, και μπορεί να ανανεώσει τις εξαρτήσεις.
- **Start Level:** Σαν Start Levels ορίζεται ένα σύνολο από Bundles που πρέπει να ξεκινήσουν ταυτόχρονα ή πρέπει να ξεκινήσουν πριν από κάποια άλλα. Η υπηρεσία αυτή θέτει το κατάλληλο Start Level του συστήματος, θέτει Start Levels στα bundles και εξετάζει το σύστημα.
- **URL Handler:** Το περιβάλλον της Java διαθέτει ένα μοντέλο για URL Handlers. Παρόλα αυτά όμως, είναι τύπου singleton κάτι που το κάνει αδύνατο να χρησιμοποιηθεί σε ένα συνεργατικό περιβάλλον όπως το OSGi, που ενδεχομένους να έχει πολλούς παρόχους της υπηρεσίας. Αυτή η υπηρεσία, κάνει δυνατή την παροχή μια τέτοιου τύπου υπηρεσίας από του ενδιαφερόμενους.

System Services

Οι υπηρεσίες συστήματος (system services), παρέχουν οριζόντιες λειτουργίες που είναι απαραίτητες σε σχεδόν οποιοδήποτε περιβάλλον. Οι υπηρεσίες Log Service, Configuration Admin Service, Device Access Service, User Admin Service, IO Connector Service και Preferences Service είναι παραδείγματα τέτοιων υπηρεσιών.

- Log Service: Οι πληροφορίες που αρχειοθετούνται (logged), όπως ειδοποιήσεις (warnings), σφάλματα (errors) και πληροφορίες εκσφαλμάτωσης (debug) διαχειρίζονται από αυτή την υπηρεσία. Δέχεται Log entries και τις διαθέτει σε όποια υπηρεσία έχει δηλώσει ενδιαφέρον.
- Configuration Admin Service: Η υπηρεσία αυτή παρέχει ένα ευέλικτο και δυναμικό μοντέλο για την ανάθεση και ανάκτηση δεδομένων διαμόρφωσης (configurations)
- Device Access Service: Αυτή η υπηρεσία παρέχει στο OSGi τον μηχανισμό ανάθεσης στον κατάλληλο driver μιας νέας συσκευής OSGi και αν χρειαστεί να ανακτήσει από το δίκτυο το κατάλληλο bundle που υλοποιεί αυτόν τον driver. Χρησιμοποιείται σε σενάρια τύπου Plug and Play.

- User Admin Service: Αυτή η υπηρεσία χρησιμοποιεί μια βάση δεδομένων με πληροφορίες για τους χρήστες που χρησιμοποιούνται για σκοπούς ταυτοποίησης κτλ.
- IO Connector Service: Η υπηρεσία αυτή υλοποιεί τα CDC/CLDC του javax.microedition.io πακέτου. Δίνει την δυνατότητα στα Bundles να παρέχουν νέες και εναλλακτικές υλοποιήσεις πρωτοκόλλων.
- Preferences Service: Αυτή η υπηρεσία παρέχει προσπέλαση σε μια ιεραρχική βάση δεδομένων με ιδιότητες (properties), που ομοιάζει με το Windows Registry ή την κλάση Java Preferences.
- Component Runtme: Η δυναμική φύση των υπηρεσιών – μπορούν να έρχονται και να φεύγουν συχνά – κάνει την ανάπτυξη κώδικα δυσκολότερη. Αυτή η υπηρεσία διευκολύνει αυτή την κατάσταση παρέχονται ένα μοντέλο εξαρτήσεων σε XML.
- Deployment Admin Service: Το πρωτεύων υλικό στο OSGi, είναι τα OSGi bundles που στην πραγματικότητα είναι Jar ή Zip αρχεία. Η υπηρεσία αυτή παρέχει μια δευτερεύουσα έννοια: το deployment package. Τα πακέτα αυτά παρέχουν bundles και πόρους σε ένα στοιχείο που μπορεί να εγκατασταθεί και να αφαιρεθεί.
- Evemt Admin: Πολλά Events στο OSGi έχουν ειδικούς τύπους διεπαφής, κάτι που τα κάνει δύσκολο να φιλτραριστούν σωστά και εύκολα. Η υπηρεσία αυτή παρέχει ένα generic model για Events.

Protocol Services

Το OSGi Alliance, έχουν προδιαγράψει μια σειρά από υπηρεσίες, με σκοπό να συσχετίζονται με ένα σύνολο εξωτερικών συχνά χρησιμοποιούμενων υπηρεσιών.

- Http Service: Η υπηρεσία αυτή, είναι μεταξύ άλλων, ένας διαχειριστής Servlets. Τα bundles μπορούν να παρέχουν Servlets, τα οποία θα είναι προσπελάσιμα με την χρήση του πωτοκόλλου HTTP. Έτσι για παράδειγμα, θα μπορεί να δημιουργηθεί ένας πολύ δελεαστικός HTTP Server στο οποίο θα μπορούν να εισέρχονται καινούργια Servlets on the runtime.
- UPnP Service: Το Universal Plun and Play (UPnP), είναι ένα αναδυόμενα standard, για τους υπολογιστές. Σκοπός αυτής της υπηρεσίας είναι να χαρτογραφεί συσκευές UPnP σαν UPnP Devices στο OSGi Framework και το αντίθετο.
- DMT Admin: Το Open Mobile Alliance (OMA), παρέχει μια αναλυτική προδιαγραφή για διαχείριση κινητών συσκευών κάτω από την έννοια του Device Management Tree (DMT). Η υπηρεσία αυτή προδιαγράφει πως αυτό το δέντρο μπορεί να προσπελαστεί μέσα από το OSGi.

Miscellaneous Services

- Wire Admin Service: Κανονικά, τα bundles εγκαθιδρύουν τους κανόνες για να βρουν υπηρεσίες με τις οποίες θα λειτουργήσουν. Σε πολλές όμως περιπτώσεις, αυτό μπορεί να είναι μια απόφαση την ώρα της προσάρτησης (deployment). Η υπηρεσία αυτή «ενώνει» μεταξύ τους υπηρεσίες που αναφέρονται σε ένα αρχείο. Έχει την αρχή του μοντέλου Producer-Consumer Που επικοινωνούν πάνω από ένα καλώδιο.

- XML Parser Service: Η υπηρεσία αυτή παρέχει διευκολύνσεις για το XML Parsing και είναι συμβατή με τον JAXP.
- Initial Provisioning
- Foreign Application Access

Παράρτημα 2

Intel UPnP Tools

Στο παράρτημα αυτό περιγράφεται ο τρόπος χρήσης των UPnP Tools της Intel που χρησιμοποιούνται από την πλατφόρμα που αναπτύχθηκε για την αναπαράσταση UPnP Devices.

Η πλατφόρμα παρέχει την δυνατότητα υποστήριξης UPnP Lights όπως αυτά περιγράφονται από το UPnP Forum (κατά την διάρκεια εκπόνησης της εργασίας). Περισσότερες πληροφορίες σχετικά στο [9].

Συνεπώς στο παράρτημα αυτό παρουσιάζεται ο τρόπος χρήσης των UPnP Lights των tools της Intel. Επιπρόσθετα όμως, γίνεται αναφορά και σε δύο άλλα χρήσιμα εργαλεία του πακέτου. Αυτά είναι το Device Sniffer και Device Spy. Στο συνοδευτικό CD παρέχεται το πακέτο που περιέχει τα Intel UPnP tools για περιβάλλον Microsoft Windows, /UPnP/IntelToolsForUPnPTechnology_v2.msi ή εναλλακτικά δωρεάν από το http://www.intel.com/cd/ids/developer/asmo-na/eng/downloads/upnp/tools/index.htm. Η εγκατάσταση του πακέτου απαιτεί την ύπαρξη του Microsoft .Net Framework, το οποίο μπορεί να βρεθεί στον κατάλογο του συνοδευτικού CD /UPnP/dotNet \Framework/ ή μπορείτε να το κατεβάσετε δωρεάν από το επίσημο site της Microsoft.

Intel Network Light

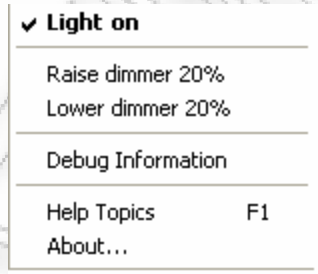
Μετά την εγκατάσταση του πακέτου των Intel UPnP tools, μπορείτε να τρέξετε την εφαρμογή αυτή, εκτελώντας το πρόγραμμα:

Έναρξη -> Προγράμματα -> Intel(R) Tools for UPnP(TM) Technology -> Network Light.

Η εφαρμογή αυτή όποτε εκτελείται δημιουργεί ένα εικονικό UPnP Light. Ο χρήστης θα δει κάτι σαν την παρακάτω εικόνα :

Εικόνα Π3.1: Network Light

Η εικόνα αυτή προσφέρει real time ενημέρωση για την κατάσταση της φωτεινότητας της συσκευής. Επιπρόσθετα, παρέχει χειριστήρια για την αλλαγή παραμέτρων της συσκευής, κάνοντας δεξί "click" επάνω στην εικόνα. Αυτό θα εμφανίσει το επόμενο μενού :



Εικόνα Π3.2: Network Light Controls

Όπου παρατηρούμε ότι μπορούμε να μεταβάλλουμε την φωτεινότητα κατά 20% κάθε φορά ή να ανάψουμε/σβήσουμε τελείως τον λαμπτήρα. Η χρήση αυτών τον χειριστηρίων θα προκαλέσει και την δημιουργία των κατάλληλων UPnP μηνυμάτων στο δίκτυο, που θα ενημερώνουν για τις αλλαγές αυτές.

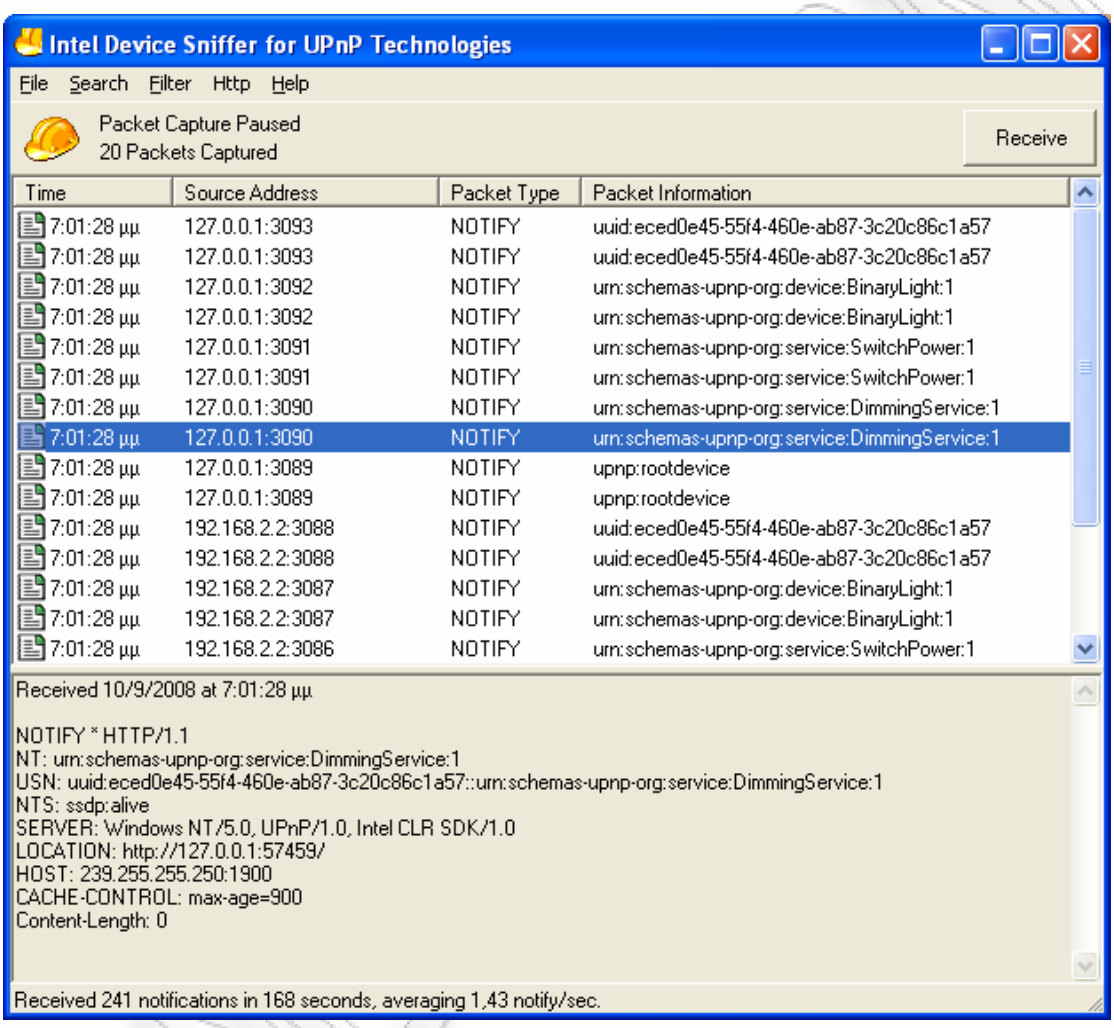
Intel Device Sniffer

Αυτή η εφαρμογή μπορεί να εκτελεστεί από το :

Έναρξη -> Προγράμματα -> Intel(R) Tools for UPnP(TM) Technology -> Device Sniffer.

Σκοπός αυτής της εφαρμογής είναι να ακούει συνεχώς το δίκτυο για UPnP μηνύματα, και να

τα εμφανίζει σε μία λίστα. Παρέχει μια σειρά βοηθητικών δυνατοτήτων, όπως για παράδειγμα φιλτράρισμα κατά IP και χρησιμοποιείται κυρίως για debugging. Στην εικόνα που ακολουθεί φαίνεται ένα παράδειγμα μετά την δημιουργία ενός UPnP λαμπτήρα:



Εικόνα Π3.3: Intel Device Sniffer

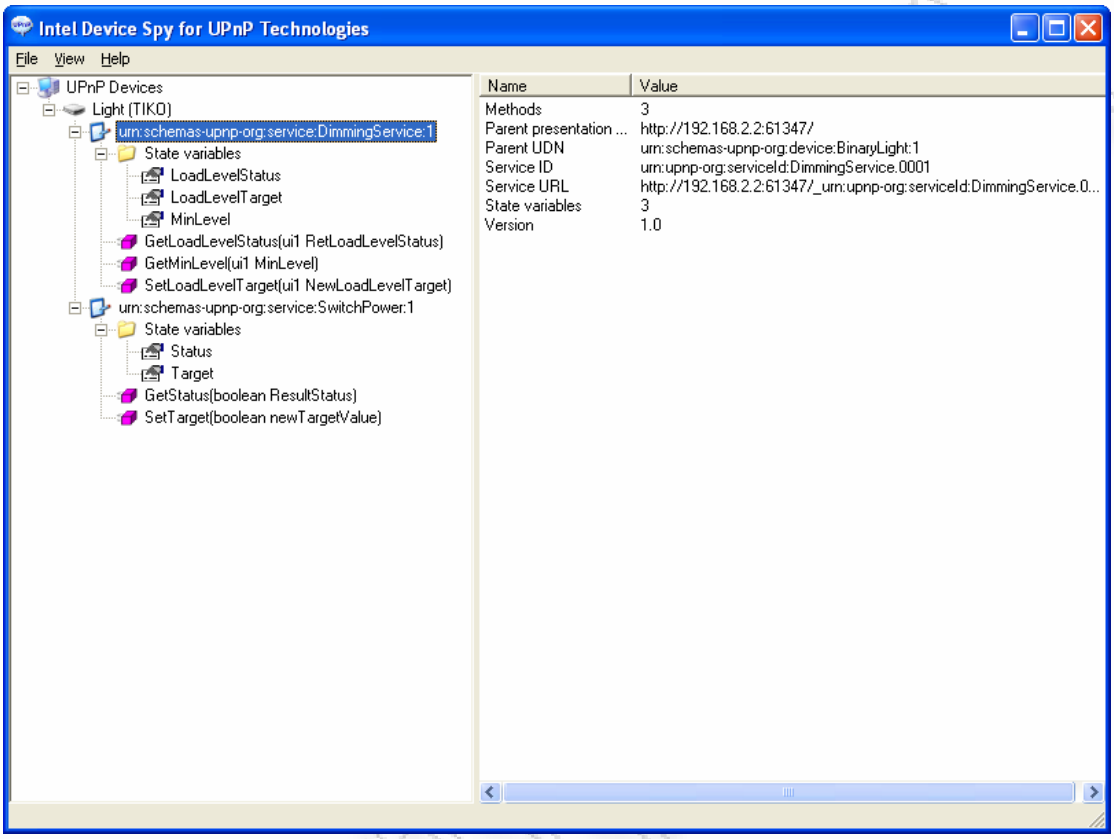
Intel Device Spy

Η εφαρμογή αυτή μπορεί να εκτελεστεί από το:

Έναρξη -> Προγράμματα -> Intel(R) Tools for UPnP(TM) Technology -> Device Spy.

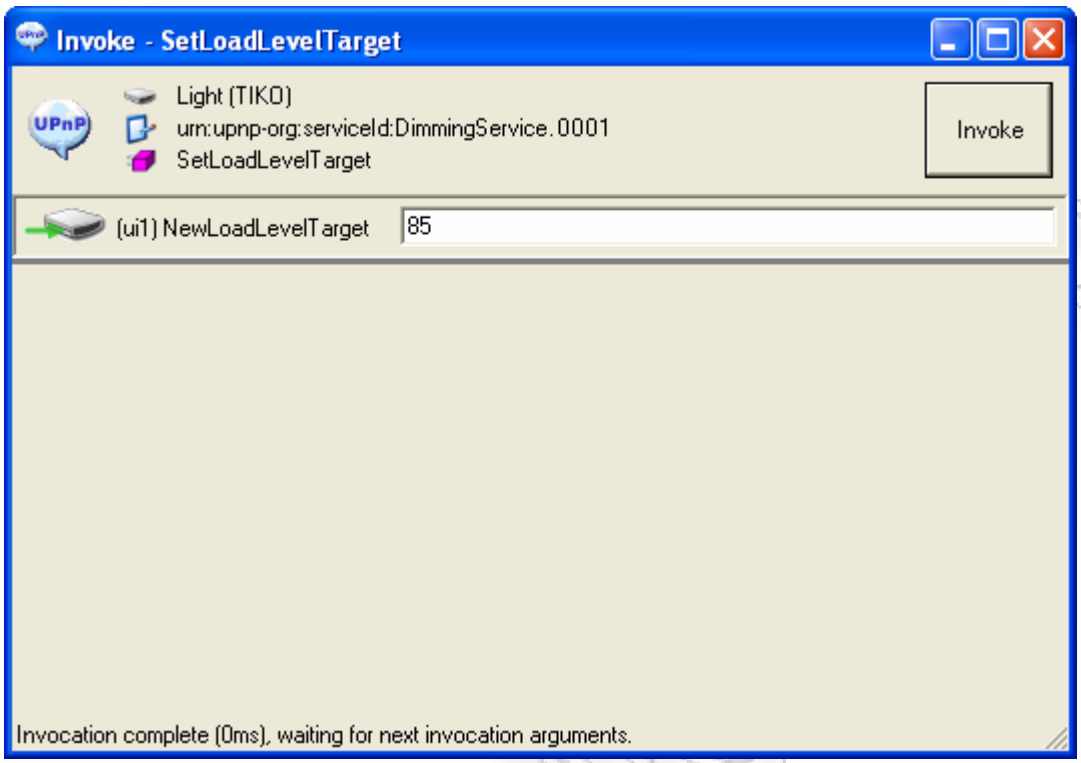
Σκοπός της εφαρμογής αυτής είναι να ακούει για UPnP announcements συσκευών και να της παρουσιάζει σε μία λίστα. Επιπρόσθετα παρουσιάζει και μία λίστα από τις υπηρεσίες και τις

μεταβλητές που παρέχει η κάθε UPnP συσκευή στο δίκτυο.

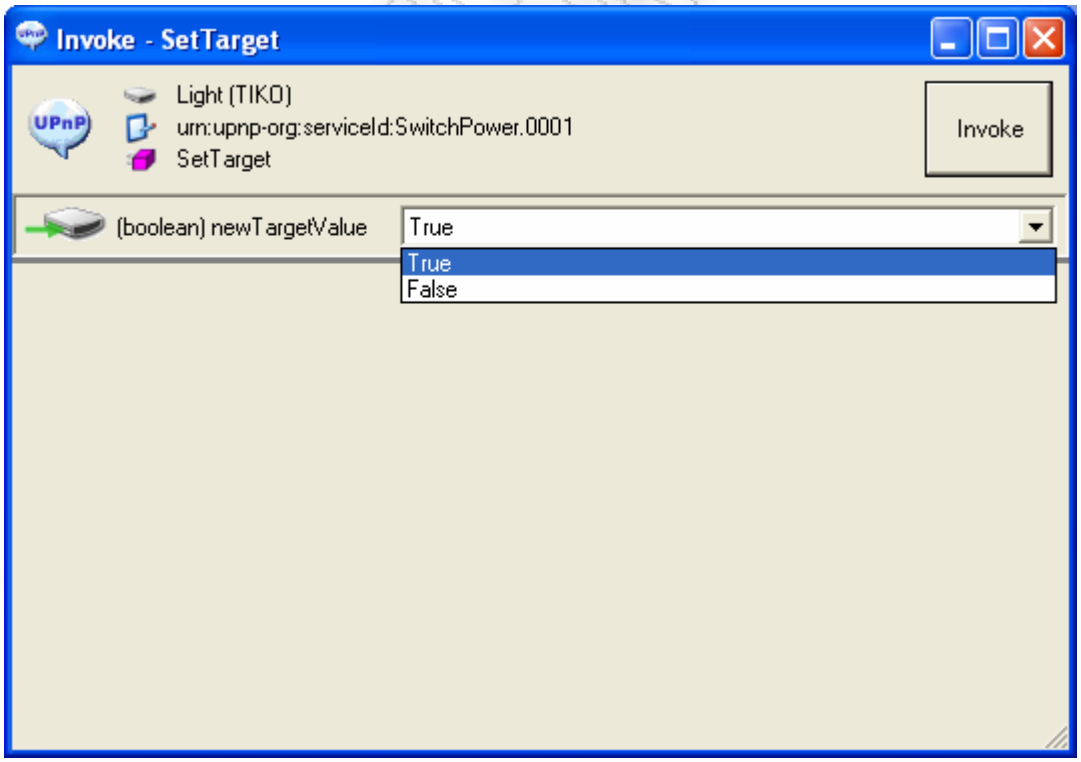


Εικόνα Π3.4: Intel Device Spy

Στην παραπάνω εικόνα παρατηρούμε πώς εμφανίζεται το Network Light που είχαμε δημιουργήσει προηγουμένως. Μεταξύ άλλων παρατηρούμε ότι παρέχει τις υπηρεσίες setLoadLevel και setTarget. Μέσα από τις υπηρεσίες αυτές μπορούμε να αλλάξουμε τις ιδιότητες ενός λαμπτήρα. Δηλαδή την κατάστασή του και το επίπεδο φωτεινότητας. Αυτό μπορεί να γίνει με δεξί "click" στην ανάλογη υπηρεσία και στην συνέχεια "invoke action". Αυτό θα είχε σαν αποτέλεσμα κάτι σαν αυτό των εικόνων που ακολουθούν.



Εικόνα Π3.5: Εκτέλεση του setLoadLevel



Εικόνα Π3.6: Εκτέλεση του setTarget

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] The OSGi Aliance, "OSGi Service Platform", Release 3, IOSPress, March 2003.

[2] The OSGi Alliance, "OSGi Service Platform Mobile Specification", Release 4, OSGi Alliance, July 2006.

[3] Heng Seng Cheng, "Smart Spaces", STIC Asie Seminar at Kuala Lumpur, 2004.

[4] Llácer Olmos, "Service and Resource Discovery in Smart Spaces", Polytechnic University of Valencia, School of Telecommunications engineering, 2007.

Internet

[5] http://www.upnp.org/standardizeddcps/documents/DimmableLight1.0cc.pdf, 2008.

[6] http://www.echelon.com/support/documentation/datashts/750xx.pdf, 2008.

[7] http://www.4home.com/home-control-services-lonmark-magazine, 2008.

[8] http://www.osgi.org/About/Technology, 2008.

[9] http://oscar.objectweb.org/usage.html, 2008.