



ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ  
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ  
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών  
Τομέας Ηλεκτρονικής και Υπολογιστών  
Εργαστήριο Επεξεργασίας Πληροφορίας και Υπολογισμών (ΕΠΥ)

### Διπλωματική Εργασία

---

*Διεπαφές γεφύρωσης κόμβων ROS με  
IoT πλατφόρμες για τη δημιουργία  
υβριδικών εφαρμογών απομακρυσμένης  
διαχείρισης πολλαπλών συστημάτων*

---

Εκπόνηση:  
Φιλίππου Αρίστιππος  
ΑΕΜ: 7795

Επιβλέποντες: Αναπληρωτής Καθηγητής κ. Ανδρέας Λ. Συμεωνίδης  
Υποψήφιος Διδάκτορας Κωνσταντίνος Παναγιώτου

**Θεσσαλονίκη 2019**

## Ευχαριστίες

Ξεκινώντας το συγγραφικό κομμάτι της διπλωματικής θα ήθελα να ευχαριστήσω όλους εκείνους που με τον τρόπο τους συνέβαλαν στην ολοκλήρωση της παρούσας εργασίας. Πιο συγκεκριμένα, θα ήθελα να ευχαριστήσω τον Αναπληρωτή Καθηγητή κ. **Ανδρέα Συμεωνίδη** για την εμπιστοσύνη που μου έδειξε αναθέτοντας μου την εκπόνηση του συγκεκριμένου θέματος. Επίσης θα ήθελα να ευχαριστήσω τον Υπ. Δρ. **Κωνσταντίνο Παναγιώτου** και τον μεταδιδακτορικό ερευνητή Δρ. **Εμμανουήλ Τσαρδούλια** για την καθοδήγηση, τις διορθώσεις και τις ιδέες που μου πρότειναν καθ' όλη τη διάρκεια της διπλωματικής μου εργασίας, τόσο κατά την ανάπτυξη του τεχνικού όσο και του συγγραφικού μέρους.

Τέλος, θα ήθελα να ευχαριστήσω θερμά την οικογένεια μου η οποία με στήριξε και ήταν δίπλα μου σε όλη την διάρκεια των φοιτητικών μου χρόνων καθώς και τους φίλους μου οι οποίοι στέκονται αρωγοί σε οποιοδήποτε εμπόδιο εμφανιστεί.

## Περίληψη

Η ρομποτική αν και μικρή σε ηλικία επιστήμη, τις τελευταίες δεκαετίες έχει πραγματοποιήσει άλματα προόδου, γεγονός που την καθιστά πλέον ώριμη τεχνολογία για εφαρμογή σε καθημερινές εργασίες. Η έννοια του ρομπότ εξελίχθηκε με την πάροδο του χρόνου και από τις απλές μηχανές φθάσαμε στην δημιουργία συσκευών οι οποίες καθορίζουν σε μεγάλο βαθμό την καθημερινότητα των ανθρώπων επιδρώντας σε πολλούς τομείς όπως η βιομηχανία, η εκπαίδευση, η εξερεύνηση του διαστήματος κ.α. Δεν θα ήταν υπερβολή να ισχυριζόμασταν ότι στόχος της επιστημονικής κοινότητας είναι η συνεχόμενη επέκταση του εύρους των ρομποτικών συστημάτων. Σε αυτή την κατεύθυνση, έχουν δημιουργηθεί τις τελευταίες δεκαετίες πληθώρα ρομποτικών πλαισίων (framework) και μεσολογισμικών (middlewares) για την εύκολη και γρήγορη ανάπτυξη ρομποτικών συστημάτων και εφαρμογών.

Από την άλλη πλευρά, η συνεχόμενη ανάπτυξη των διαδικτυακών υπηρεσιών και κυρίως του Internet of Things έχει δημιουργήσει το κατάλληλο πλαίσιο για την απομακρυσμένη παρακολούθηση και συλλογή παραμέτρων από τα “Things”, τα οποία αντιστοιχούν στις συσκευές που συμμετέχουν σε έναν κόμβο IoT. Το επόμενο βήμα αφορά την ενσωμάτωση των ρομποτικών συσκευών σ’ ένα δίκτυο IoT, γεγονός που θα δημιουργήσει το πλαίσιο για απομακρυσμένη επικοινωνία και αλληλεπίδραση μεταξύ ρομποτικών και μη συσκευών. Άλλωστε, η ραγδαία εξέλιξη στον κλάδο του IoT και ο μεγάλος αριθμός έξυπνων συσκευών χαμηλού κόστους δημιούργησε την ανάγκη ανάπτυξης πρωτοκόλλων και εργαλείων για την επικοινωνία μεταξύ αυτών και τη διασύνδεση τους στον παγκόσμιο ιστό.

Στα πλαίσια της παρούσας διπλωματικής εργασίας παρουσιάζεται ένα σύστημα το οποίο δίνει την δυνατότητα για δημιουργία εφαρμογών οι που επικοινωνούν και αλληλεπιδρούν απευθείας με τις ρομποτικές συσκευές, φιλοσοφία η οποία εναρμονίζεται με την ιδέα του IoT. Μέσω αυτού του συστήματος δίνεται η δυνατότητα για απευθείας αλληλεπίδραση διαφόρων αισθητήρων και ρομποτικών συσκευών γεγονός που διευκολύνει την ανάπτυξη εφαρμογών που διαχειρίζονται πολλαπλές συσκευές εισάγοντας παράλληλα τα ρομπότ στους IoT κόμβους.

## Abstract

### **Bridging interfaces of ROS nodes with IoT platforms for building hybrid, multi-system remote applications.**

Robotics is an interdisciplinary branch of engineering and science that has grown rapidly in recent years. The concept of robot has evolved over time, from simple machines to devices that greatly affect people's daily lives in areas such as industry, education and space exploration. It would not be an exaggeration to argue that the goal of the scientific community is to continually expand the range of robotic systems. In this direction, numerous of robotic frameworks and middleware have been developed in order to facilitate the rapid development of robotic systems and applications.

On the other hand, the continuous development of services, including Internet of Things, has established structures for remote monitoring and configuration of the "Things", which correspond to the devices participating in an IoT hub. The next challenge is to incorporate robotic devices into an IoT network, which will create the framework for remote communication and interaction between robotic and non-robotic devices.

The main goal of this thesis is to build a system that enables the development of applications that communicate and interact directly with robotic devices, a philosophy that is in line with the idea of IoT. This system enables the direct interaction of various sensors and robotic devices, which facilitates the development of applications that control multiple devices while introducing robots to IoT nodes.

Filippou Aristippos, [aristip1908@gmail.com](mailto:aristip1908@gmail.com)  
Electrical and computer Engineering Department  
Aristotle University of Thessaloniki, Greece 2019

---

## Πίνακας Περιεχομένων

Ευχαριστίες.....	2
Περίληψη.....	3
Abstract.....	4
Πίνακας Περιεχομένων .....	5
Λίστα Εικόνων.....	7
1 ΕΙΣΑΓΩΓΗ.....	9
1.1 Κίνητρο .....	9
1.2 Περιγραφή του Προβλήματος .....	10
1.3 Στόχοι της Διπλωματικής .....	11
1.4 Διάρθρωση Διπλωματικής .....	11
2 ΕΠΙΣΚΟΠΗΣΗ ΕΡΕΥΝΗΤΙΚΗΣ ΠΕΡΙΟΧΗΣ .....	13
2.1 Rosbridge.....	13
2.2 Centralized ROS Master .....	14
2.3 RosLink Bridge .....	15
2.4 AWS IoT Bridge.....	17
2.5 Open HAB και IoT Bridge.....	18
2.6 ROSTful .....	18
3 ΥΠΟΒΑΘΡΟ .....	20
3.1 Θεωρητικό Υπόβαθρο .....	20
3.1.1 Internet of things (IoT).....	20
3.1.2 Διαδίκτυο και APIs .....	26
3.1.3 Network Robot System .....	29
3.2 Τεχνολογικό Υπόβαθρο και Εργαλεία.....	32
3.2.1 Robot Operating System (ROS).....	32
3.2.2 RabbitMQ.....	34
3.2.3 Swagger Api .....	39
3.2.4 Python Flask.....	40
3.2.5 Mongo DB και Robo3T.....	41
3.2.6 Jinja2 .....	41

---

3.2.7	Node-RED .....	42
4	ΥΛΟΠΟΙΗΣΗ ΣΥΣΤΗΜΑΤΟΣ .....	44
4.1	Αρχιτεκτονική .....	44
4.2	Βάση Δεδομένων.....	45
4.3	REST API.....	47
4.4	Σύνδεση με το RabbitMQ.....	48
4.5	Πιστοποίηση Χρηστών και Συσκευών .....	48
4.6	Απομακρυσμένος Έλεγχος Συσκευών .....	50
4.7	ROS Environment .....	51
4.8	Γεφύρωση Περιβάλλοντος ROS με την Απομακρυσμένη Πλατφόρμα 52	
4.8.1	Μετατροπή ROS Μηνυμάτων και Υπηρεσιών.....	53
4.8.2	Bridging templates.....	54
4.8.3	Δημιουργία Γεφυρών .....	58
4.9	Διαδικτυακή Εφαρμογή .....	60
4.10	Γραφικό περιβάλλον.....	62
5	Σενάρια Χρήσης του Συστήματος.....	67
5.1	Εγγραφή Συσκευής και Χρήστη.....	67
5.2	Έλεγχος Περιβάλλοντος Συσκευής.....	68
5.3	Δημιουργία και Τερματισμός Γεφύρωσης.....	69
5.4	Δημιουργία Node-RED Εφαρμογής.....	72
6	Συμπεράσματα και ανοιχτά θέματα .....	74
6.1	Συμπεράσματα .....	74
6.2	Ανοιχτά Θέματα .....	75
	Βιβλιογραφία.....	76
	Παράρτημα Α.....	78

## Λίστα Εικόνων

Εικόνα 1: Τυπική σύνδεση ROS μεταξύ χρηστών και συσκευών .....	14
Εικόνα 2: Centralized προσέγγιση του ROS master .....	15
Εικόνα 3: Η προσέγγιση της τεχνολογίας RosLink.....	15
Εικόνα 4: Η αρχιτεκτονική του RosLink .....	17
Εικόνα 5: Η αρχιτεκτονική του IoT .....	23
Εικόνα 6: Μέθοδος επικοινωνίας Publish-Subscribe μέσω MQTT .....	24
Εικόνα 7 Η επικοινωνία Publish-Subscribe μέσω AMQP .....	25
Εικόνα 8: Η RESTful αρχιτεκτονική του CoAP .....	25
Εικόνα 9:Τα βασικά στοιχεία του ROS .....	33
Εικόνα 10:Η επικοινωνία τύπου Publish-Subscribe μέσω του RabbitMQ.....	35
Εικόνα 11:Ροή μηνύματος από τον Publisher στον Subscriber .....	36
Εικόνα 12:Διαδικασία υλοποίησης της επικοινωνίας Publish-Subscribe .....	37
Εικόνα 13: Υλοποίηση της RPC μεθόδου επικοινωνίας .....	38
Εικόνα 14:Λειτουργία της μηχανής δημιουργίας template.....	42
Εικόνα 15:Αρχιτεκτονική συστήματος .....	44
Εικόνα 16: Μοντέλο συσκευής της βάσης δεδομένων .....	46
Εικόνα 17: Πίνακας χρηστών της βάσης δεδομένων .....	47
Εικόνα 18:Αρχείο του RabbitMQ server το οποίο τον τρόπο αυθεντικοποίησης .....	49
Εικόνα 19:Διαδικασία η οποία ακολουθείται για την γεφύρωση ενός ROS Publisher .....	56
Εικόνα 20: Διαδικασία η οποία ακολουθείται για την γεφύρωση ενός Broker Publisher .....	57
Εικόνα 21: Διαδικασία η οποία ακολουθείται για την γεφύρωση ενός ROS service .....	58
Εικόνα 22: Η διαδικασία γεφύρωσης ενός ROS publisher.....	59
Εικόνα 23: Η διαδικασία τερματισμού μιας γέφυρας επικοινωνίας.....	60
Εικόνα 24: Σελίδα για εγγραφή μιας συσκευής στην βάση δεδομένων .....	63
Εικόνα 25: Σελίδα για εγγραφή ενός χρήστη στην βάση δεδομένων .....	64
Εικόνα 26: Σελίδα στην οποία ο χρήστης εισάγει τα στοιχεία εγγραφής του για να του επιτραπεί η είσοδος.....	64
Εικόνα 27: Δυνατότητες εφαρμογής στην κατηγορία Device.....	65
Εικόνα 28: Δυνατότητες εφαρμογής στην κατηγορία Monitor .....	65
Εικόνα 29: Δυνατότητες εφαρμογής στην κατηγορία Bridge .....	66
Εικόνα 30: Δυνατότητες εφαρμογής στην κατηγορία Terminate.....	66
Εικόνα 31: Συμπλήρωση των στοιχείων εγγραφής μιας συσκευής.....	67
Εικόνα 32: Σύνδεση του χρήστη με συσκευές ώστε να έχει την δυνατότητα μετέπειτα να τις χρησιμοποιεί .....	68
Εικόνα 33:Σελίδα η οποία παρουσιάζει τα στατιστικά στοιχεία της συσκευής .....	69

Εικόνα 34: Σελίδα η οποία επιστρέφει το ROS περιβάλλον της κάθε συσκευής .....	69
Εικόνα 35: Φόρμα στην οποία ο χρήστης εισάγει τα στοιχεία και δημιουργείται η γεφύρωση ενός ROS publisher .....	70
Εικόνα 36: Το ROS περιβάλλον της συσκευής μετά την γεφύρωση ενός ROS Publisher .....	71
Εικόνα 37: Φόρμα για την διακοπή της γεφύρωσης ενός ROS topic .....	71
Εικόνα 38: Διάγραμμα εφαρμογής ελέγχου θερμοκρασίας σε περιβάλλον Node-RED .....	73



---

## 1 ΕΙΣΑΓΩΓΗ

### 1.1 Κίνητρο

Η εξέλιξη της τεχνολογίας υπήρξε πάντοτε ένας από τους καθοριστικότερους παράγοντες που σηματοδοτούσε αλλαγές στην πορεία της ανθρωπότητας, έχοντας πολλαπλές επιδράσεις στην ζωή των ανθρώπων. Είναι άλλωστε κοινώς αποδεκτό ότι διανύουμε μια χρονική περίοδο η οποία μπορεί να χαρακτηριστεί και ως τεχνολογική επανάσταση και θα οδηγήσει στην μετάβαση σε μια διαφορετική, κοινωνική αλλά και οικονομική, πραγματικότητα. Αυτή η τεχνολογική εξέλιξη έχει γίνει αισθητή σε διαφόρους τομείς όπως η ιατρική, η βιομηχανία, η γενετική και η τεχνολογία τροφίμων.

Μία από τις πολλές επιστήμες η οποία γνωρίζει άνθηση τα τελευταία χρόνια είναι η επιστήμη της ρομποτικής. Βέβαια, η ρομποτική δεν είναι μια επιστήμη η οποία ανακαλύφθηκε πρόσφατα αλλά έχει τις ρίζες της από την αρχαιότητα, όπου ο άνθρωπος ονειρευόταν μηχανές οι οποίες θα μιμούνται ή θα ξεπερνούν τις ανθρώπινες δεξιότητες. Ο κλάδος της ρομποτικής ασχολείται με την σύνθεση διαφόρων ανθρώπινων λειτουργιών χρησιμοποιώντας ποικίλους μηχανισμούς, αισθητήρες, ενεργοποιητές και αλγορίθμους τεχνητής νοημοσύνης, έχοντας κάνει αισθητή την επίδρασή του σε διαφόρους τομείς της ανθρώπινης καθημερινότητας, όπως η εξερεύνηση του διαστήματος, οι αγροτικές εφαρμογές, η έρευνα και διάσωση, η προσωπική βοήθεια, ακόμα και η διασκέδαση.

Στην σημερινή εποχή η ταυτόχρονη ανάπτυξη των τεχνολογιών που σχετίζονται με τον τομέα της ρομποτικής αλλά και του διαδικτύου έχει δημιουργήσει μια νέα ερευνητική περιοχή που χαρακτηρίζεται ως ρομποτική νέφους (cloud robotics) [9]. Η ρομποτική νέφους είναι ένα αναδυόμενο πεδίο ρομποτικής που χρησιμοποιεί τεχνολογίες του Διαδικτύου με επίκεντρο τα πλεονεκτήματα της σύγκλισης των υποδομών και των κοινών υπηρεσιών. Επιτρέπει στα ρομπότ να επωφελούνται από τις ισχυρές υπολογιστικές μονάδες, τους αποθηκευτικούς αλλά και επικοινωνιακούς πόρους των σύγχρονων κέντρων δεδομένων. Επιπλέον, αφαιρεί τα γενικά έξοδα για συντήρηση και ενημερώσεις και μειώνει την εξάρτηση από τα διάφορα λογισμικά (middlewares).

Από την άλλη πλευρά το διαδίκτυο των πραγμάτων (IoT) αποτελεί ένα καινούργιο και βασικό κεφάλαιο στη διασύνδεση συσκευών στον παγκόσμιο ιστό και είναι σχεδόν βέβαιο ότι τα επόμενα χρόνια θα αποτελεί κυρίαρχη τεχνολογία όπου όλο και περισσότερες συσκευές θα συνδέονται σε αυτό. Η ενσωμάτωση λοιπόν των ρομποτικών συστημάτων με το IoT αποτελεί ένα πολλά υποσχόμενο τομέα της ρομποτικής νέφους στον οποίο ο έλεγχος και η διαχείριση των ρομπότ θα γίνεται μέσω του διαδικτύου.

Το διαδίκτυο των ρομποτικών πραγμάτων (IoRT) αποτελεί μια νέα επαναστατική ιδέα η οποία αναπτύχθηκε πρώτη φορά από την ABI research και στοχεύει στη συγχώνευση των αισθητήρων και διαδικτυακών αντικειμένων με ρομποτικά και αυτόνομα συστήματα. Επίκεντρο αυτής της τεχνολογίας είναι η διεύρυνση των δυνατοτήτων του τρέχοντος διαδικτύου των πραγμάτων και των σημερινών ρομποτικών συστημάτων επιτρέποντας έτσι την δημιουργία νέων υπηρεσιών [10] [11].

## 1.2 Περιγραφή του Προβλήματος

Η συνεχόμενη τεχνολογική ανάπτυξη γύρω από το IoT σε συνδυασμό με την καθημερινή εξέλιξη των ρομποτικών συστημάτων έχουν καταστήσει κυρίαρχη την ανάγκη της μεταξύ τους επικοινωνίας, όπου τα ρομπότ ή ακόμα και τα διάφορα αισθητήρια τα οποία διαθέτουν θα αντιμετωπίζονται ως ανεξάρτητες συσκευές. Άλλωστε, τα σύγχρονα ρομπότ διαθέτουν αισθητήρια και έχουν υπολογιστικές αλλά και επικοινωνιακές δυνατότητες, γεγονός που τα καθιστά ιδανικά για την σύνδεση τους σε κόμβους IoT. Παραδείγματα αυτής της διασύνδεσης μπορεί να είναι η αυτόματη παρακολούθηση των ασθενών, η αυτόματη διαχείριση και συνεργασία των παραγωγικών δραστηριοτήτων, ο έλεγχος και η διαχείριση ηλεκτρικών και ενεργειακών μονάδων και η παροχή βοήθειας σε καταστάσεις έκτακτης ανάγκης ή ακόμα και καταστάσεις πανικού και κινδύνου.

Το γενικότερο πρόβλημα λοιπόν το οποίο τίθεται προς επίλυση είναι η δημιουργία ενός μηχανισμού με την χρήση του οποίου τα διάφορα ρομποτικά και αυτόνομα συστήματα θα μπορούν εύκολα, γρήγορα και με δυναμικό τρόπο να συνδέονται σε κόμβους και πλατφόρμες IoT. Κατά αυτό τον τρόπο δίνεται η δυνατότητα τόσο για απομακρυσμένη παρακολούθηση, έλεγχο και αλληλεπίδραση μεταξύ τους όσο και για απομακρυσμένη εκτέλεση εφαρμογών. Δίνοντας λοιπόν την δυνατότητα στις ρομποτικές συσκευές να συμμετέχουν στο IoT, δημιουργείται η συνθήκη για απευθείας επικοινωνία και διασύνδεση μεταξύ των ρομπότ αλλά και αλληλεπίδραση τους με τις διάφορες συσκευές και πράκτορες που συμμετέχουν, επιτρέποντας την ταυτόχρονη εκτέλεση πολλαπλών συνεργαζόμενων προγραμμάτων, δημιουργώντας έτσι ένα κατακευματισμένο σύστημα. Σε αυτό μπορεί να προστεθεί και η χρησιμοποίηση της cloud τεχνολογίας μέσω της οποίας δίνεται η δυνατότητα για απευθείας πρόσβαση και εκτέλεση έτοιμων αλγορίθμων ρομποτικής, γεγονός που αποβλέπει στην εξοικονόμηση χρόνου και πόρων. Απόρροια όλων των παραπάνω είναι οι υπολογισμοί να γίνονται σε επίπεδο διαδικτύου σε αντίθεση με την μέχρι τώρα προσέγγιση που ήθελε όλες τις διεργασίες να εκτελούνται στην τοπική υπολογιστική μονάδα. Τέλος, μέσω αυτού του μηχανισμού δίνεται η δυνατότητα για εύκολη διασύνδεση των ρομποτικών συστημάτων με έτοιμες πλατφόρμες IoT γεγονός που δημιουργεί ένα νέο πλαίσιο δυνατοτήτων στην ανάπτυξη εφαρμογών που διαχειρίζονται πολλαπλά ρομπότ. Παρ' όλ' αυτά οι εφαρμογές αυτές είναι αναγκαίο να μην διαθέτουν ισχυρες χρονικές απαιτήσεις (real

time constraints), καθώς οι καθυστερήσεις που οφείλονται στην διαδικτυακή επικοινωνία μπορεί να οδηγήσουν σε ανεπιτυχή εκτέλεση της εφαρμογής, όπως στην περίπτωση εκτέλεσης ενός αλγορίθμου εξερεύνησης.

### 1.3 Στόχοι της Διπλωματικής

Στόχος της παρούσας διπλωματικής εργασίας είναι η δημιουργία ενός εργαλείου το οποίο θα επιτρέπει την εύκολη και γρήγορη διασύνδεση ρομποτικών συστημάτων σε μια διαδικτυακή πλατφόρμα, μέσω της οποίας πλέον θα επιτυγχάνεται η επικοινωνία μεταξύ τους. Μ' αυτό τον τρόπο διευκολύνεται η αλληλεπίδραση μεταξύ τους καθιστώντας πολύ πιο γρήγορη και εύκολη την υλοποίηση multi-robot εφαρμογών. Έχοντας ως βάση αυτό τον μηχανισμό εισάγουμε την φιλοσοφία του IoT και των things στον τομέα της ρομποτικής. Συνοπτικά λοιπόν θα μπορούσαμε να πούμε ότι η δημιουργία του μηχανισμού «γεφύρωσης» των ρομποτικών συστημάτων με μία διαδικτυακή πλατφόρμα εισάγει έναν νέο, γρήγορο και εύχρηστο τρόπο επικοινωνίας μεταξύ των ρομπότ και διευρύνει τα όρια ανάπτυξης ρομποτικών εφαρμογών. Τέλος, ο έλεγχος και η λειτουργία αυτού του μηχανισμού «γεφύρωσης» επιτυγχάνεται με την βοήθεια μιας εύχρηστης και φιλικής διαδικτυακής εφαρμογής.

### 1.4 Διάρθρωση Διπλωματικής

Η διάρθρωση της διπλωματικής εργασίας αποτελείται από έξι κεφάλαια τα οποία έχουν ως εξής:

- **Κεφάλαιο 1:** Παρουσιάζεται η περιγραφή του προβλήματος που καλείται να λύσει η διπλωματική εργασία, το κίνητρο υλοποίησής της καθώς και ο τελικός της στόχος.
- **Κεφάλαιο 2:** Το δεύτερο κεφάλαιο αφορά την επισκόπηση της ερευνητικής περιοχής. Γίνεται μία αναφορά σε παρόμοια συστήματα, που ήδη υπάρχουν και επιτελούν επιμέρους τμήματα του συνολικού συστήματος που θα υλοποιηθεί, καθώς και ολοκληρωμένα συστήματα τα οποία για διαφορετικούς λόγους δε λύνουν τα προβλήματα που παρουσιάστηκαν στο πρώτο κεφάλαιο.
- **Κεφάλαιο 3:** Το τρίτο κεφάλαιο πραγματεύεται το θεωρητικό και τεχνολογικό υπόβαθρο στο οποίο βασίζεται η διπλωματική εργασία διευκολύνοντας την κατανόηση του εγγράφου. Εισάγονται δηλαδή οι απαραίτητες έννοιες και τεχνολογίες που σχετίζονται με την εκπόνηση της διπλωματικής εργασίας όπως επίσης και τα εργαλεία τα οποία χρησιμοποιήθηκαν και συνδυάστηκαν για την υλοποίηση του συνολικού συστήματος.
- **Κεφάλαιο 4:** Αποτελεί τον πυρήνα της διπλωματικής εργασία καθώς παρουσιάζονται αναλυτικά οι επιμέρους υλοποιήσεις που απαρτίζουν το συνολικό σύστημα το οποίο δημιουργήθηκε.

- **Κεφάλαιο 5:** Στο πέμπτο κεφάλαιο γίνεται η περιγραφή των σεναρίων χρήσης του συνολικού συστήματος, μέσω των οποίων ο αναγνώστης κατανοεί βαθύτερα την χρησιμότητα του συνολικού συστήματος το οποίο υλοποιήθηκε.
- **Κεφάλαιο 6:** Στο έκτο και τελευταίο κεφάλαιο παρατίθενται τα συμπεράσματα από τη δημιουργία του συστήματος αλλά και μελλοντικές αλλαγές που θα μπορούσαν να γίνουν δίνοντας ένα πιο ολοκληρωμένο τελικό αποτέλεσμα.

## 2 ΕΠΙΣΚΟΠΗΣΗ ΕΡΕΥΝΗΤΙΚΗΣ ΠΕΡΙΟΧΗΣ

Στο παρόν κεφάλαιο γίνεται μια παρουσίαση διαφόρων τεχνολογιών που αποβλέπουν στην απομακρυσμένη επικοινωνία μεταξύ των ρομπότ. Πιο συγκεκριμένα περιγράφονται διάφοροι τρόποι με τους οποίους επιτυγχάνεται η απομακρυσμένη σύνδεση των ρομπότ, χρησιμοποιώντας το διαδίκτυο ή υιοθετώντας σε συγκεκριμένες περιπτώσεις την φιλοσοφία του IoT.

### 2.1 Rosbridge

Μια τεχνολογία η οποία έχει αναπτυχθεί για την διασύνδεση και επικοινωνία των ρομπότ μέσω διαδικτύου είναι η εγκατάσταση web servers στην πλευρά των ρομπότ με χαρακτηριστικότερο παράδειγμα αυτό του πρωτοκόλλου επικοινωνίας Rosbridge<sup>1</sup>, μια διεπαφή προγραμματισμού εφαρμογών η οποία επιτρέπει την γεφύρωση των διεπαφών JSON<sup>2</sup> με αυτές του ROS. Μ' αυτό τον τρόπο δίνεται η δυνατότητα στους χρήστες να μπορούν χωρίς την γνώση του ROS αλλά και από οποιαδήποτε πλατφόρμα επιθυμούν να έχουν πρόσβαση στο ROS ώστε να χειρίζονται και να ελέγχουν ρομπότ. Φυσικά αυτή η διαδικασία επιτρέπει τους προγραμματιστές να χρησιμοποιούν το διαδίκτυο ώστε να χειρίζονται ρομπότ γεγονός που αποτελεί την πρώτη ενσωμάτωση του ROS και ευρύτερα των ρομποτικών εφαρμογών στο χώρο του διαδικτύου.

Πιο συγκεκριμένα, το Rosbridge είναι ένα πακέτο λογισμικού στο ROS που προσφέρει πρόσβαση στα ROS topics και services, μέσω είτε των TCP sockets<sup>3</sup> είτε των Web-sockets<sup>4</sup> ως μηνύματα JSON. Και οι δύο τρόποι λειτουργίας χρησιμοποιούν την ίδια μορφή JSON τόσο για τις αιτήσεις όσο και για τις απαντήσεις. Το πακέτο Rosbridge χωρίζεται σε δύο μέρη, το πρωτόκολλο και την υλοποίηση.

Το πρωτόκολλο Rosbridge είναι μια προδιαγραφή για την αποστολή μηνυμάτων από και προς το ROS, βασισμένο σε JSON. Η ιδέα είναι ότι, λόγω της μορφής των JSON μηνυμάτων (strings), οποιαδήποτε γλώσσα μπορεί να υλοποιήσει το πρωτόκολλο Rosbridge και να αλληλοεπιδράσει με το ROS. Το πρωτόκολλο καλύπτει θέματα subscribe και publish, κλήσεις services, λήψη παραμέτρων και παραμετροποίηση, συμπίεση μηνυμάτων και άλλα.

Η χρήση του πραγματοποιείται μέσω του πακέτου rosbridge\_suite<sup>5</sup>, το οποίο είναι μια συλλογή πακέτων που παρέχουν ένα επίπεδο επικοινωνίας τύπου WebSocket. Τα πακέτα περιλαμβάνουν:

---

<sup>1</sup> <https://wiki.ros.org/rosbridge>

<sup>2</sup> <https://en.wikipedia.org/wiki/JSON>

<sup>3</sup> [https://en.wikipedia.org/wiki/Network\\_socket](https://en.wikipedia.org/wiki/Network_socket)

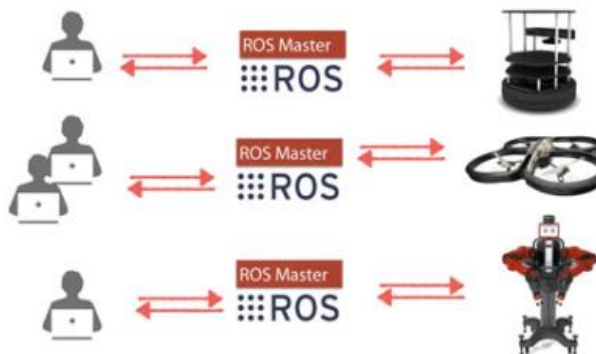
<sup>4</sup> <https://en.wikipedia.org/wiki/WebSocket>

<sup>5</sup> [http://wiki.ros.org/rosbridge\\_suite](http://wiki.ros.org/rosbridge_suite)

- **rosbridge\_library**: Το rosbridge library είναι υπεύθυνο για τη λήψη του μηνύματος JSON και την αποστολή των εντολών στο ROS και αντίστροφα.
- **Rosapi**: Κάνει συγκεκριμένες ενέργειες ROS, διαθέσιμες μέσω κλήσεων υπηρεσιών. Αυτό περιλαμβάνει την παραλαβή και ρύθμιση παραμέτρων, τη λήψη λιστών θεμάτων και άλλα.
- **rosbridge\_server**: Το Rosbridge\_server παρέχει απομακρυσμένη πρόσβαση μέσω websockets.

## 2.2 Centralized ROS Master

Για να επιτευχθεί η επικοινωνία των ρομπότ το ROS επιτρέπει τον έλεγχο μεταξύ των διαφόρων ρομπότ σε ένα κοινό σταθμό εργασίας (workstation) κάτω από το ίδιο κόμβο (ROS master) λύση η οποία περιορίζει τις εφαρμογές μόνο σε τοπικά δίκτυα χωρίς να υπάρχει δυνατότητα ελέγχου και χειρισμού των ρομπότ μέσω διαδικτύου, όπως φαίνεται στην εικόνα 1. Διάφορες λύσεις όπως η δημιουργία μιας διεύθυνσης δικτύου τύπου NAT(network address translation) και η προώθηση των θυρών του δεν αποτελούν ευρεία λύση και περιορίζονται μόνο σε συγκεκριμένες εφαρμογές [13].



Εικόνα 1: Τυπική σύνδεση ROS μεταξύ χρηστών και συσκευών

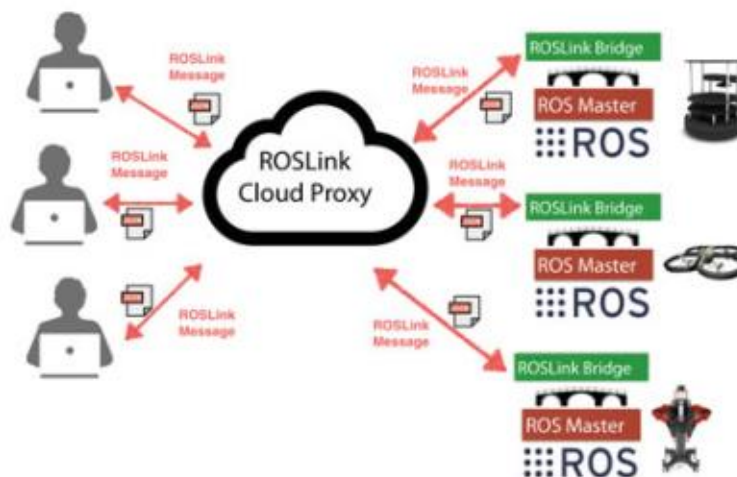
Μια centralized προσέγγιση η οποία θα μπορούσε να λύσει το πρόβλημα της επικοινωνίας των ρομπότ διαμέσου του διαδικτύου είναι η χρήση ενός κεντρικού κόμβου (ROS master) για όλα τα ρομπότ ο οποίος θα τρέχει σε έναν κεντρικό υπολογιστή και θα έχει συγκεκριμένη διεύθυνση IP (εικόνα 2). Με αυτόν τον τρόπο ο κάθε χρήστης θα μπορούσε να συνδεθεί στον ίδιο κοινό κόμβο και να επικοινωνήσει με τα ρομπότ κάνοντας εγγραφή και δημοσίευση στα topics τα οποία τον ενδιαφέρουν. Ωστόσο αυτή η λύση χρειάζεται ιδιαίτερη προσοχή καθώς το κάθε ROS topic πρέπει να έχει διαφορετικό και συγκεκριμένο όνομα πράγμα ιδιαίτερα δύσκολο όταν ο αριθμός των ρομπότ είναι πολύ μεγάλος. Επίσης ένα άλλο πρόβλημα το οποίο θα μπορούσε να προκύψει είναι η υπερφόρτωση του κοινού κόμβου αλλά και η αδυναμία δημιουργίας συγκεκριμένου τρόπου κοινής χρήσης όπου ο κάθε χρήστης θα μπορεί να χρησιμοποιεί συγκεκριμένα ρομπότ και όχι όλα αυτά τα οποία είναι συνδεδεμένα στο κοινό κόμβο.



Εικόνα 2: Centralized προσέγγιση του ROS master

### 2.3 RosLink Bridge

Στόχος του Roslink bridge είναι ο έλεγχος και η επικοινωνία ενός ρομπότ το οποίο λειτουργεί σε ROS μέσω διαδικτύου. Ουσιαστικά πρόκειται για ένα ασύγχρονο πρωτόκολλο επικοινωνίας μεταξύ του ROS και του διαδικτύου προσπαθώντας να δημιουργήσει μια γέφυρα επικοινωνίας μεταξύ του ROS και ενός cloud server (εικόνα 3) [13].



Εικόνα 3: Η προσέγγιση της τεχνολογίας RosLink

Στη βιβλιογραφία, τα περισσότερα από τα σχετικά έργα έχουν εστιάσει στη χρήση ενός μοντέλου client server<sup>6</sup> δύο επιπέδων όπου ο server υλοποιείται στο

<sup>6</sup> [https://en.wikipedia.org/wiki/Client%E2%80%93server\\_model](https://en.wikipedia.org/wiki/Client%E2%80%93server_model)

ρομπότ και ο client υλοποιείται στην εφαρμογή του χρήστη. Παρόλα αυτά η αρχιτεκτονική του RosLink στηρίζεται σε ένα μοντέλο server client τριών επιπέδων όπου ο client υλοποιείται στο ρομπότ και τον χρήστη, ενώ ο server βρίσκεται σε δημόσιο τομέα (public domain) και λειτουργεί ως διακομιστής, όντας υπεύθυνος για τη σύνδεση των ρομπότ με τους χρήστες τους.

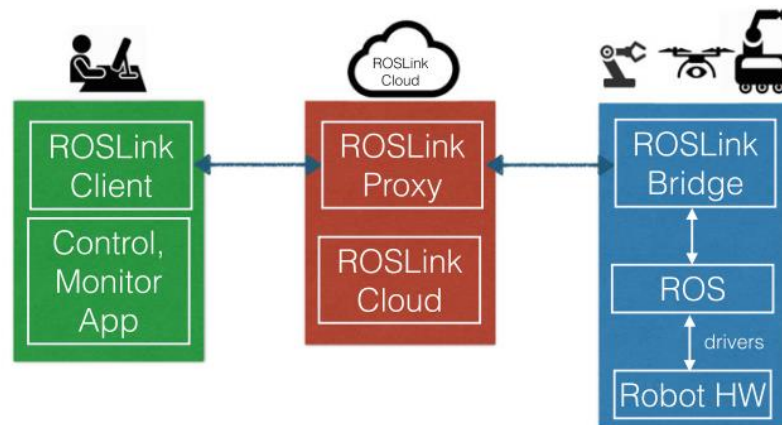
Πιο συγκεκριμένα η αρχιτεκτονική του Roslink αποτελείται από τρία μέρη (εικόνα 4).

- **Roslink Bridge:** Αποτελεί τον πυρήνα του συστήματος, όντας η διεπαφή μεταξύ του ROS και του πρωτοκόλλου RosLink, έχοντας δύο κύριες λειτουργίες. Από την μια πλευρά διαβάζει δεδομένα από μηνύματα των ROS topics και services, τα δεδομένα μετατρέπονται σε μορφή JSON σύμφωνα με τις προδιαγραφές του πρωτοκόλλου RosLink και στέλνονται στο διακομιστή ή ακόμη και στην ίδια την εφαρμογή του χρήστη. Κατά την αντίθετη διαδικασία η γέφυρα RosLink λαμβάνει τα δεδομένα από τον διακομιστή ή την εφαρμογή σε μορφή JSON και τα αποκωδικοποιεί ώστε να είναι συμβατά με το ROS όπου και εκτελούνται.
- **The RosLink Proxy and Cloud:** Ο RosLink proxy<sup>7</sup> ενεργεί ως διακομιστής μεταξύ του RosLink bridge (ενσωματωμένο στο ρομπότ) και την εφαρμογή του χρήστη. Ο ρόλος του είναι να συνδέσει την εφαρμογή με ένα ρομπότ το οποίο χρησιμοποιεί ROS. Ο Roslink proxy λοιπόν είναι υπεύθυνος για την αποστολή μηνυμάτων μεταξύ ρομπότ και χρήστη. Επιτρέπει στον χρήστη να ενημερώνεται για την κατάσταση του ρομπότ και να προωθεί εντολές ελέγχου προς αυτό. Από την άλλη ο RosLink proxy επικοινωνεί με το RosLink cloud ώστε να επιτυγχάνεται η συντήρηση και η διαχείριση λίστας ρομπότ και χρηστών, και να δημιουργείται η κατάλληλη σύνδεση μεταξύ τους. Τέλος είναι υπεύθυνος ώστε να εκτελούνται όλες τις λειτουργίες διαχείρισης, συμπεριλαμβανομένης της ασφάλειας, της παρακολούθησης και της ποιότητας της υπηρεσίας.
- **The RosLink Client Application:** Αντιπροσωπεύει μια εφαρμογή ελέγχου και παρακολούθησης του ρομπότ. Αυτή η εφαρμογή χρησιμοποιείται για την παρακολούθηση της κατάστασης του ρομπότ που λαμβάνεται μέσω των μηνυμάτων RosLink από το RosLink proxy. Τέλος στέλνει εντολές μέσω μηνυμάτων RosLink για τον έλεγχο της δραστηριότητας του ρομπότ.

---

<sup>7</sup> [https://en.wikipedia.org/wiki/Proxy\\_server](https://en.wikipedia.org/wiki/Proxy_server)





Εικόνα 4: Η αρχιτεκτονική του RosLink

Οι διάφορες τεχνολογίες του RosLink bridge επικοινωνούν μεταξύ τους με το RosLink πρωτόκολλο επικοινωνίας. Το πρωτόκολλο αυτό είναι χωρισμένο σε δύο βασικά μέρη, στο πρωτόκολλο που χρησιμοποιείται για επικοινωνία μεταξύ χρηστών, διαδικτύου και ρομπότ (Transport Protocol) και στην μετατροπή των μηνυμάτων τύπου ROS από και σε μορφή JSON (RosLink Message Types).

## 2.4 AWS IoT Bridge

Η IoT πλατφόρμα AWS (Amazon Web Services)<sup>8</sup> παρέχει ασφαλή και αμφίδρομη επικοινωνία μεταξύ συσκευών συνδεδεμένων στο διαδίκτυο όπως αισθητήρες, ενεργοποιητές, ενσωματωμένοι μικροελεγκτές ή ακόμα και έξυπνες συσκευές. Χρησιμοποιεί κυρίως το πρωτόκολλο επικοινωνίας MQTT(βλέπε υποενότητα 3.2.6.1) και με αυτό τον τρόπο επιτυγχάνεται η συλλογή δεδομένων τηλεμετρίας από πολλαπλές συσκευές και η αποθήκευση τους ώστε να υπάρχει η δυνατότητα ανάλυσης. Τέλος η πλατφόρμα δίνει της δυνατότητα στους χρήστες για δημιουργία εφαρμογών που επιτρέπουν τον έλεγχο αυτών των συσκευών από τα κινητά τηλέφωνα ή ακόμα και από τα tablet τους.

Η γέφυρα `aws_iot_mqtt`<sup>9</sup> αποτελεί ένα πακέτο από εκτελέσιμα αρχεία τα οποία είναι υπεύθυνα για την επικοινωνία ενός ρομπότ το οποίο χρησιμοποιεί ROS με την IoT πλατφόρμα AWS. Η επικοινωνία αυτή επιτυγχάνεται με την βοήθεια της γέφυρας `mqtt_bridge` η οποία χρησιμοποιεί τα ROS μηνύματα ως πρωτόκολλο επικοινωνίας. Πιο συγκεκριμένα τα μηνύματα τύπου ROS μετατρέπονται σε μορφή JSON ώστε να είναι συμβατά με το πρωτόκολλο επικοινωνίας MQTT το οποίο χρησιμοποιείται από την IoT πλατφόρμα, και αντίστροφα, τα μηνύματα αποκωδικοποιούνται από την μορφή JSON σε μορφή ROS ώστε να μπορούν να εκτελεστούν από την πλευρά του ρομπότ.

<sup>8</sup> <https://aws.amazon.com/choosing-a-cloud-platform/>

<sup>9</sup> [https://github.com/groove-x/mqtt\\_bridge](https://github.com/groove-x/mqtt_bridge)

## 2.5 Open HAB και IoT Bridge

Η πλατφόρμα open Home Automation Bus (HAB)<sup>10</sup> είναι, όπως εμφανίζει και το όνομα της, μια πλατφόρμα ανοιχτού κώδικα η οποία υλοποιεί την τεχνολογία έξυπνού σπιτιού. Είναι υπεύθυνη για την σύνδεση και επικοινωνία διαφόρων έξυπνων συσκευών σε επίπεδο σπιτιού και διαχείρισης τους από τον χρήστη μέσω διαδικτυακών και android εφαρμογών.

Η γέφυρα *iot bridge*<sup>11</sup> είναι μια τεχνολογία η οποία επιτρέπει την αμφίδρομη επικοινωνία μεταξύ του ROS και της IoT πλατφόρμας Open HAB. Μ αυτό τον τρόπο δίνεται η δυνατότητα σε ένα ρομπότ το οποίο χρησιμοποιεί ROS να επικοινωνεί και να ανταλλάσσει δεδομένα με οποιαδήποτε έξυπνη συσκευή είναι συνδεδεμένη στην Open HAB πλατφόρμα.

Η παραπάνω διαδικασία υλοποιείται στηριζόμενη σε τρεις βασικές διαδικασίες:

- Η γέφυρα *iot* κάνει *subscribe* στο ROS topic *iot\_command* (*diagnostic\_msgs/KeyValue*), και όταν λάβει ένα μήνυμα το κάνει *publish* στην πλατφόρμα IoT η οποία με την σειρά της επικοινωνεί με την συσκευή για την εκτέλεση του μηνύματος.
- Η *iot bridge* κάνει *subscribe* στο ROS topic *iot\_set* (*diagnostic\_msgs/KeyValue*), και όταν λάβει ένα μήνυμα το κάνει *publish* στην Open HAB και μ αυτό τον τρόπο αναβαθμίζεται η κατάσταση της συσκευής.
- Η γέφυρα *IoT* κάνει *publish* στο ROS topic *iot\_updates* (*diagnostic\_msgs/KeyValue*), δηλαδή όταν λάβει ένα μήνυμα από μια συσκευή που είναι συνδεδεμένη στο IoT, το μήνυμα γίνεται *publish* στο ROS topic και αναβαθμίζεται η κατάσταση της συσκευής σε επίπεδο ROS.

## 2.6 ROSTful

Το ROSTful<sup>12</sup>, είναι ένας web server η αρχιτεκτονική του οποίου είναι τύπου RESTful. Μέσω του ROSTful γίνονται διαθέσιμα τα ROS topics, services και actions τα οποία «τρέχουν» σε επίπεδο συσκευής. Η φιλοσοφία του ROSTful είναι η διασύνδεση του ROS περιβάλλοντος μέσω του διαδικτύου και συγκεκριμένα του πρωτοκόλλου HTTP, διαδικασία η οποία επιτυγχάνεται με την χρησιμοποίηση ενός RESTful API. Ως εκ τούτου, είναι δυνατόν να χρησιμοποιείται από οποιοδήποτε πρόγραμμα είναι γραμμένο σε γλώσσα προγραμματισμού “python”.

Το ROSTful υποστηρίζει τόσο την ανταλλαγή μηνυμάτων σε μορφή JSON με την χρησιμοποίηση του πακέτου *Rosbridge* όσο και την αλληλεπίδραση μηνυμάτων

---

<sup>10</sup> <https://www.openhab.org/docs/>

<sup>11</sup> [http://wiki.ros.org/iot\\_bridge](http://wiki.ros.org/iot_bridge)

<sup>12</sup> <https://github.com/pyros-dev/rostful>

σε μορφή ROS γεγονός που αυξάνει την αποδοτικότητά του. Ωστόσο, η φιλοσοφία του ROSTful είναι διαφορετική με αυτή του Rosbridge. Το Rosbridge παρέχει ένα API για το ROS μέσω JSON, χρησιμοποιώντας web sockets. Από την άλλη, το ROSTful επιτρέπει την αλληλεπίδραση, μέσω web services και συγκεκριμένα post και get requests, των ROS topics, services και actions. Ο ROSTful client είναι ένας ROS κόμβος ο οποίος επικοινωνεί με ένα ROSTful web service επιτρέποντας την αλληλεπίδραση με τα ROS services, topics, action. Τέλος, ο ROSTful server είναι WSGI<sup>13</sup> συμβατός και συνεπώς μπορεί να χρησιμοποιηθεί με τους περισσότερους web servers, όπως ο Apache<sup>14</sup> και ο IIS<sup>15</sup>.

---

<sup>13</sup> [https://en.wikipedia.org/wiki/Web\\_Server\\_Gateway\\_Interface](https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface)

<sup>14</sup> [https://en.wikipedia.org/wiki/Apache\\_HTTP\\_Server](https://en.wikipedia.org/wiki/Apache_HTTP_Server)

<sup>15</sup> [https://en.wikipedia.org/wiki/Internet\\_Information\\_Services](https://en.wikipedia.org/wiki/Internet_Information_Services)

### 3 ΥΠΟΒΑΘΡΟ

Το παρόν κεφάλαιο πραγματεύεται το θεωρητικό και τεχνολογικό υπόβαθρο στο οποίο βασίζεται η διπλωματική εργασία εισάγοντας τις απαραίτητες έννοιες και τεχνολογίες για την κατανόηση του κειμένου. Πραγματοποιείται λοιπόν μια πρώτου βαθμού ανάλυση των χρήσιμων εννοιών οι οποίες σχετίζονται με το θεωρητικό πλαίσιο της εργασίας καθώς και μια σύντομη περιγραφή των βασικότερων εργαλείων λογισμικού που χρησιμοποιήθηκαν. Ο αναγνώστης χρειάζεται να κατανοήσει τον βασικό τρόπο λειτουργίας και τις δυνατότητες τους, για να είναι σε θέση εν συνεχεία να εμβαθύνει στον λόγο που επιλέχθηκε το κάθε ένα από αυτά αλλά και τον τρόπο με τον οποία εκείνα συνδυάστηκαν για την διάρθρωση της συγκεκριμένης εργασίας.

#### 3.1 Θεωρητικό Υπόβαθρο

##### 3.1.1 Internet of things (IoT)

Η ιδέα του IoT εμφανίστηκε πρώτη φορά στο πανεπιστήμιο Carnegie Mellon το 1982, όπου και συζητήθηκε η δημιουργία ενός δικτύου από έξυπνα αντικείμενα, σε μια προσπάθεια να συνδέσουν στο δίκτυο έναν αυτόματο πωλητή αναψυκτικών. Ο όρος Internet of things επινοήθηκε για πρώτη φορά από τον Kevin Ashton το 1999 στο πλαίσιο της διαχείρισης της αλυσίδας προμηθειών του, όπου και άρχισαν να αναπτύσσονται διάφορες τεχνολογίες όπως το RFID.

Παρ' όλ' αυτά τα τελευταία χρόνια ο όρος έχει εξελιχθεί με την τεχνολογία του internet of things να εμφανίζεται σε όλο και περισσότερους τομείς όπως στον τομέα της υγείας των μεταφορών σε μεγάλες βιομηχανίες ακόμα και σε έξυπνα σπίτια. Σύμφωνα με έρευνες, υπάρχουν βάσιμες ενδείξεις ότι ο αριθμός των έξυπνων συσκευών οι οποίες θα συνδέονται στο IoT θα φτάσει τα 25 δισεκατομμύρια έως το 2020 [1].

##### 3.1.1.1 Τι είναι το IoT;

Το διαδίκτυο των πραγμάτων (Internet of Things-IoT) είναι ένα δίκτυο έξυπνων αντικειμένων, που επικοινωνούν μεταξύ τους με σκοπό τη συλλογή και την ανταλλαγή δεδομένων. Την τελευταία δεκαετία, λόγω της ανάπτυξης των υποδομών και τεχνολογιών του διαδικτύου, έχουν διατυπωθεί αρκετοί ορισμοί σχετικά με το IoT. Σύμφωνα με το [4] οι τρεις βασικοί άξονες του IoT είναι τα φυσικά αντικείμενα, ο συνδυασμός των διαφόρων ελεγκτών και τέλος το διαδίκτυο.

##### 3.1.1.2 Τα πράγματα (things) στο IoT

Ο όρος πράγματα, συναντάται στη βιβλιογραφία συχνά και ως “έξυπνα αντικείμενα”, ή πιο απλά ως αντικείμενα. Ως “thing” ορίζεται ένα φυσικό ή εικονικό αντικείμενο, που έχει την ικανότητα ενσωμάτωσης σε δίκτυα επικοινωνιών, καθώς

και την ικανότητα της αναγνώρισής του από το σύστημα [5]. Τα φυσικά αντικείμενα, φέρουν ενσωματωμένα ηλεκτρονικά, λογισμικό, αισθητήρες (παθητικές συσκευές που προσφέρουν ή παράγουν μετρήσεις), ενεργοποιητές (ενεργητικές συσκευές που αλληλεπιδρούν με το περιβάλλον -actuators) και μπορούν να συνδεθούν διαδίκτυο.

#### 3.1.1.3 Βασικά Χαρακτηριστικά και Απαιτήσεις

Το IoT μπορεί να δημιουργήσει νέες ευκαιρίες για ανάπτυξη καινοτόμων εφαρμογών, γύρω από διάφορους τομείς στηριζόμενο όμως σε μερικά κοινά χαρακτηριστικά και αρχές [2], όπως:

- **Επεκτασιμότητα** : Βάση της φιλοσοφίας του Internet of things είναι ο τεράστιος αριθμός των συσκευών που επικοινωνούν και ανταλλάσσουν πληροφορίες μεταξύ τους. Η αποτελεσματική διαχείριση του ανταλλασσόμενου όγκου δεδομένων αποτελεί μια από τις προτεραιότητες του IoT.
- **Ανομοιογένεια** : Το IoT χαρακτηρίζεται από μεγάλη ανομοιογένεια, καθώς ένας μεγάλος αριθμός διαφορετικών συσκευών συνδέονται σ αυτό. Η επικοινωνία , η υποστήριξη και γενικότερα η διαχείριση αυτών των συσκευών αποτελεί πρωταρχικό στόχο του Internet of Things.
- **Ευελιξία** : Καθώς του περιβάλλον του IoT χαρακτηρίζεται ευμετάβλητο υπάρχει η ανάγκη για δυναμική και συνεχής διαχείριση και επαναπρογραμματισμός των συσκευών
- **Μείωση του κόστους** : Η ελαχιστοποίηση του κόστους ανάπτυξης και συντήρησης όπως και η μείωση της απαιτούμενης ενέργειας αποτελούν βασικό μέλημα της τεχνολογίας του IoT.
- **Ασφάλεια** : Ένα από τα βασικότερα γνωρίσματα όλων των τεχνολογιών είναι η αξιοπιστία και η ασφάλεια. Σ αυτό το μήκος κύματος η τεχνολογία του IoT είναι απαραίτητο να εξασφαλίζει ασφάλεια και ακεραιότητα μεταξύ των επικοινωνιών προστατεύοντας τις συσκευές αλλά και τα προσωπικά δεδομένα.

#### 3.1.1.4 Αρχιτεκτονική IoT

Η αρχιτεκτονική του IoT παρουσιάζει διάφορες μορφές με την επικρατέστερη να είναι αυτή των πέντε επιπέδων [5] όπως παρουσιάζεται στην εικόνα 5. Τα πέντε αυτά επίπεδα περιγράφονται παρακάτω.

- **Επίπεδο αντίληψης (Perception Layer)**  
Σκοπός αυτού του επιπέδου είναι η αναγνώριση των αντικειμένων και η συλλογή πληροφοριών από αυτά. Στη βιβλιογραφία συναντάται συχνά και ο όρος ‘επίπεδο έξυπνων αντικειμένων’ ή ‘επίπεδο αισθητήρων’. Το επίπεδο

αντίληψης μπορεί να περιλαμβάνει κάμερες, αισθητήρες, ενεργοποιητές, ετικέτες RFID, GPS, τερματικά και δίκτυα αισθητήρων.

- **Επίπεδο δικτύου (Network Layer)**

Βασική λειτουργία του επιπέδου δικτύου αποτελεί η ορθή και γρήγορη δρομολόγηση και μετάδοση των πακέτων σε ένα δίκτυο. Στο επίπεδο αυτό περιλαμβάνονται τεχνολογίες δικτύων, όπως ασύρματα ή ενσύρματα δίκτυα και τοπικά δίκτυα (LAN). Τα κυριότερα μέσα, για τη μετάδοση των πληροφοριών, είναι τα FTTx, 3G/4G, WiFi, ETHERNET, Bluetooth, ZigBee και η τεχνολογία υπερύθρων (infrared). Επιπλέον, στο επίπεδο αυτό μπορεί να πραγματοποιηθεί η αποθήκευση και η επεξεργασία ενός μεγάλου αριθμού δεδομένων.

- **Επίπεδο Εφαρμογής (Application Layer)**

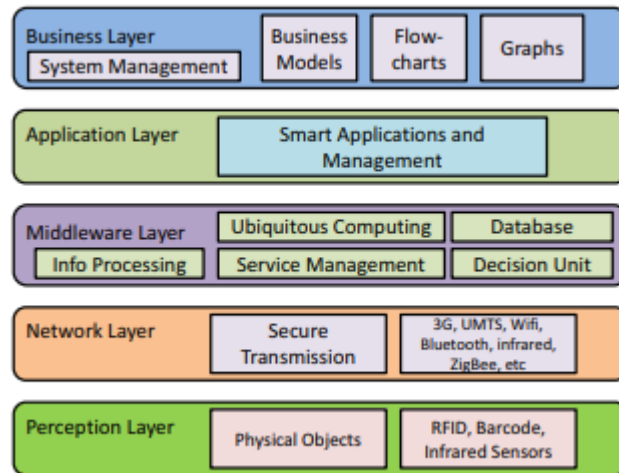
Το επίπεδο εφαρμογής είναι υπεύθυνο για την επεξεργασία των ληφθέντων δεδομένων από το επίπεδο δικτύου. Μέσω διαφόρων εφαρμογών, ο χρήστης μπορεί να έρθει σε επαφή με τα έξυπνα αντικείμενα που παρέχει το επίπεδο αντίληψης. Οι εφαρμογές αυτές έχουν επεκταθεί σε πολλούς τομείς της ανθρώπινης καθημερινότητας όπως η υγεία η βιομηχανία τα έξυπνα σπίτια και άλλες.

- **Επίπεδο Ενδιάμεσου Λογισμικού (Middleware Layer)**

Το επίπεδο αυτό συνδέει το επίπεδο δικτύου με το επίπεδο εφαρμογής. Στην ουσία, αποτελεί το λογισμικό, το οποίο επιτρέπει στις εφαρμογές την πρόσβαση στα δεδομένα που παρέχουν τα έξυπνα αντικείμενα. Στο επίπεδο αυτό πραγματοποιούνται λειτουργίες επεξεργασίας και αποθήκευσης δεδομένων. Οι πλατφόρμες λογισμικού, αποτελούν ένα τμήμα αυτού του επιπέδου.

- **Επίπεδο Επιχείρησης (Business Layer)**

Το κέρδος πάντα αποτελούσε βασικό στόχο των εταιριών. Το επίπεδο αυτό ασχολείται με την οικονομική διαχείριση των παρεχόμενων υπηρεσιών. Οι επιχειρήσεις, μέσω των εφαρμογών τους, παρέχουν πληροφορίες και δεδομένα, ζητώντας ως αντάλλαγμα ένα χρηματικό αντίτιμο.



Εικόνα 5: Η αρχιτεκτονική του IoT

#### 3.1.1.5 Πρωτόκολλα Επικοινωνίας Δεδομένων

Σε αυτή την ενότητα, εξετάζουμε τα βασικότερα πρωτόκολλα επικοινωνίας δεδομένων, τα οποία έχουν τυποποιηθεί από διαφορετικούς οργανισμούς, και ενσωματώνονται στην τεχνολογία του IoT για τη μετάδοση μηνυμάτων [3]

##### 3.1.1.5.1 Message Queue Telemetry Transport (MQTT)

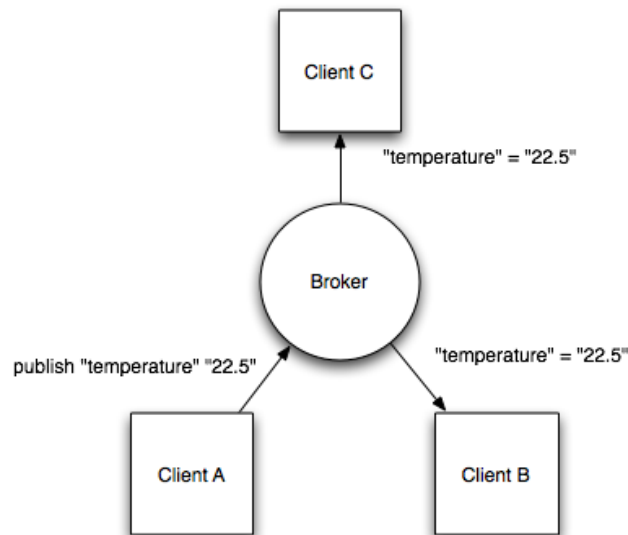
Το MQTT<sup>16</sup> είναι ένα πρωτόκολλο ανταλλαγής μηνυμάτων που αναπτύχθηκε με τη βοήθεια του Andy Stanford-Clark της IBM και της Arlen Nirperer της Arcom το 1999 και έχει σχεδιαστεί για απευθείας επικοινωνία μηχανής με μηχανή (M2M) . Παρέχει συνδεσιμότητα μεταξύ εφαρμογών και χρηστών στη μια πλευρά (one end) και δικτύου και επικοινωνιών στην άλλη (other end).

Ανήκει στην κατηγορία πρωτοκόλλων που χρησιμοποιούν την αρχιτεκτονική publish-subscribe όπως φαίνεται στην εικόνα 6 , όπου το σύστημα αποτελείται από τρία βασικά στοιχεία, τους publishers τους subscribers και του διαμεσολαβητή μηνυμάτων (message broker)<sup>17</sup>. Στο πεδίο εφαρμογής του IoT, οι publishers αποτελούνται από αισθητήρες που συνδέονται με τους subscribers ώστε να τους στείλουν δεδομένα. Από την πλευρά οι subscribers είναι αισθητήρες ή ακόμα και εφαρμογές που ενημερώνονται κάθε φορά που λαμβάνουν νέα δεδομένα. Οι διαμεσολαβητές μηνυμάτων είναι υπεύθυνοι να ταξινομήσουν τα δεδομένα σε topics ώστε η επικοινωνία μεταξύ publisher και subscriber να γίνεται μεταξύ των topics ενδιαφέροντος τους.

<sup>16</sup> <http://mqtt.org>

<sup>17</sup> [https://en.wikipedia.org/wiki/Message\\_broker](https://en.wikipedia.org/wiki/Message_broker)

Στα πλεονεκτήματα του MQTT συγκαταλέγονται η απλότητα του, μόλις πέντε μέθοδοι API, και το εξαιρετικά χαμηλό αποτύπωμα δεδομένων του που το καθιστά ιδανικό για τις σημερινές κινητές και αναπτυσσόμενες εφαρμογές του IoT.



Εικόνα 6: Μέθοδος επικοινωνίας Publish-Subscribe μέσω MQTT

#### 3.1.1.5.2 Advanced Message Queuing Protocol (AMQP)

Το AMQP<sup>18</sup>, σχεδιάστηκε από τον John O'Hara της JP Morgan Chas κυρίως για τον κλάδο της χρηματοπιστωτικής βιομηχανίας έχοντας παρόμοια αρχιτεκτονική με το πρωτόκολλο MQTT.

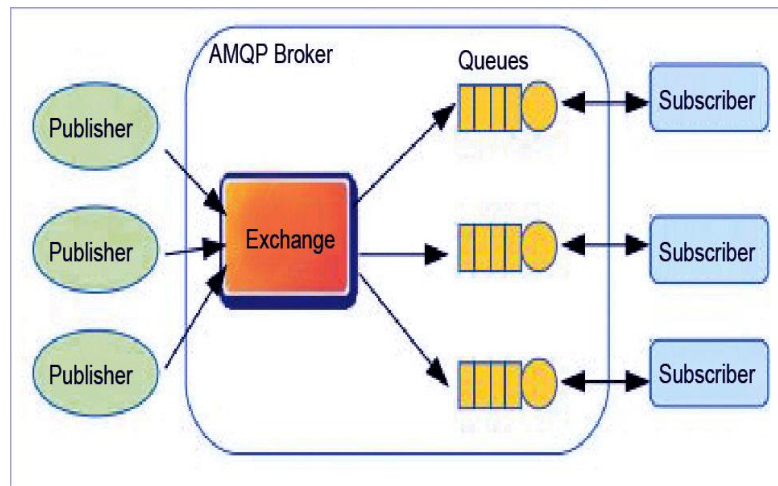
Η κύρια διαφορά τους είναι ότι ο διαμεσολαβητής (message broker) στην περίπτωση του AMQP είναι διαιρεμένος σε δύο βασικά στοιχεία, queues και exchanges, όπως φαίνεται στην εικόνα 7. Τα exchanges είναι υπεύθυνα για τη λήψη των μηνυμάτων από τους publishers, και την δρομολόγηση τους σε queues. Οι subscribers συνδέονται με το κομμάτι των queues, οι οποίες ουσιαστικά αντιπροσωπεύουν τα topics και συλλέγουν τα δεδομένα από τους αισθητήρες κάθε φορά που είναι διαθέσιμα.

Είναι ένα δυαδικό πρωτόκολλο (binary wire protocol)<sup>19</sup> που σχεδιάστηκε για διαλειτουργικότητα μεταξύ διαφορετικών προμηθευτών. Εταιρείες όπως η JP Morgan το χρησιμοποιούν για να επεξεργάζονται 1 δισεκατομμύριο μηνύματα την ημέρα. Η NASA το χρησιμοποιεί για την πλατφόρμα της Nebula και η Google το χρησιμοποιεί για περίπλοκη επεξεργασία συμβάντων.

<sup>18</sup> <https://www.amqp.org/>

<sup>19</sup> [https://en.wikipedia.org/wiki/Binary\\_protocol](https://en.wikipedia.org/wiki/Binary_protocol)

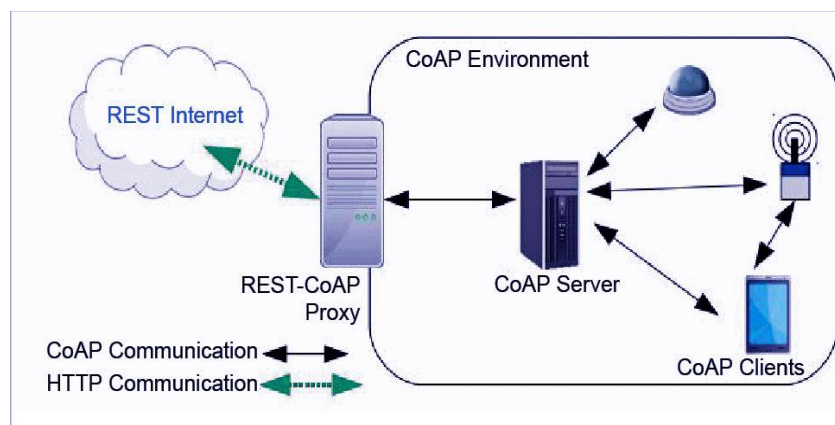




Εικόνα 7 Η επικοινωνία Publish-Subscribe μέσω AMQP

#### 3.1.1.5.3 CoAP (Constrained Application Protocol)

Το CoAP<sup>20</sup> είναι ένα πρωτόκολλο διαδικτύου (internet application protocol) για συσκευές περιορισμένων πόρων. Έχει σχεδιαστεί για να επιτρέπει σε απλές συσκευές να ενταχθούν στο IoT μέσω μη ιδανικών δικτύων που διαθέτουν χαμηλή διαθεσιμότητα εύρους ζώνης. Χρησιμοποιεί RESTful αρχιτεκτονική (εικόνα 8), για επικοινωνία μηχανής με μηχανή (M2M) μέσω request/response και είναι ειδικά σχεδιασμένο για IoT εφαρμογές που βασίζονται σε πρωτόκολλα HTTP.



Εικόνα 8: Η RESTful αρχιτεκτονική του CoAP

#### 3.1.1.5.4 STOMP (Simple Text Oriented Messaging Protocol)

Το STOMP<sup>21</sup> είναι ένα απλό, διαλειτουργικό πρωτόκολλο, το οποίο σχεδιάστηκε για ασύγχρονη ανταλλαγή μηνυμάτων μεταξύ clients με την

<sup>20</sup> [https://en.wikipedia.org/wiki/Constrained\\_Application\\_Protocol](https://en.wikipedia.org/wiki/Constrained_Application_Protocol)

<sup>21</sup> <https://stomp.github.io>

μεσολάβηση κάποιων servers, ώστε να παρέχει εύκολη και διαδεδομένη ανταλλαγή μηνυμάτων.

Ως προς την δομή του, είναι ένα πρωτόκολλο βασισμένο σε πλαίσια (frame based protocol)<sup>22</sup>, που έχουν την ίδια μοντελοποίηση με αυτά του HTTP. Το πλαίσιο αποτελείται από μια εντολή (command), μια σειρά από προαιρετικές επικεφαλίδες (headers) και ένα προαιρετικό σώμα (body).

Η επικοινωνία μεταξύ client και server υλοποιείται μέσω ενός "frame" που αποτελείται από έναν αριθμό γραμμών. Η πρώτη γραμμή περιέχει την εντολή, ακολουθούμενη από κεφαλίδες με τη μορφή (<key>: <value> (μία ανά γραμμή), ακολουθούμενη από μια κενή γραμμή και έπειτα από το περιεχόμενο του σώματος, που τελειώνει με έναν χαρακτήρα NULL.

### 3.1.2 Διαδίκτυο και APIs

Η συνεχής εξέλιξη του διαδικτύου έχει δημιουργήσει επιτακτική την ανάγκη επικοινωνίας μεταξύ των εφαρμογών. Ως αποτέλεσμα, εκτός από την εξέλιξη του IoT την τελευταία δεκαετία, επήλθε μεγάλη πρόοδος στον τομέα των διαδικτυακών υπηρεσιών (Web Services)<sup>23</sup>. Παρακάτω παρουσιάζονται συνοπτικά βασικές τεχνολογίες που διέπουν την επικοινωνία στο διαδίκτυο [6].

#### 3.1.2.1 TCP, UDP και IP

Το επίπεδο TCP<sup>24</sup> (Transmission Control Protocol) αποτελεί ένα από τα δυο βασικά πρωτόκολλα του επιπέδου μεταφοράς στο μοντέλο OSI<sup>25</sup>. Παρέχει αξιόπιστη αμφίδρομη επικοινωνία, ενώ περιγράφεται από τα ακόλουθα χαρακτηριστικά.

- Μεταφορά δεδομένων
- Σύνδεση δύο τερματικών(endpoints)
- Τα δεδομένα να παραλαμβάνονται όπως στάλθηκαν
- Αποστολή δεδομένων σε ακολουθία
- Αξιόπιστη εκκίνηση και τερματισμό

Το UDP<sup>26</sup> (User Datagram Protocol) είναι το άλλο βασικό πρωτόκολλο στο επίπεδο μεταφοράς που χρησιμοποιείται στο διαδίκτυο. Συγκρινόμενο με το TCP, το UDP είναι λιγότερο πολύπλοκο αλλά ταυτόχρονα και λιγότερο αξιόπιστο όσον αφορά την λήψη των πακέτων που αποστέλλονται στο άλλο τερματικό. Το πρωτόκολλο αυτό

---

<sup>22</sup> [https://en.wikipedia.org/wiki/Frame\\_\(networking\)](https://en.wikipedia.org/wiki/Frame_(networking))

<sup>23</sup> [https://en.wikipedia.org/wiki/Web\\_service](https://en.wikipedia.org/wiki/Web_service)

<sup>24</sup> [https://el.wikipedia.org/wiki/Transmission\\_Control\\_Protocol](https://el.wikipedia.org/wiki/Transmission_Control_Protocol)

<sup>25</sup> [https://en.wikipedia.org/wiki/OSI\\_model](https://en.wikipedia.org/wiki/OSI_model)

<sup>26</sup> <https://el.wikipedia.org/wiki/UDP>

χρησιμοποιείται κυρίως σε περιπτώσεις όπου απαιτείται μεγάλη ταχύτητα στην αποστολή των δεδομένων, όπως η ζωντανή μετάδοση ομιλίας ή βίντεο.

Το IP<sup>27</sup> (Internet Protocol) παρέχει έναν μηχανισμό επικοινωνίας για υπολογιστές και συσκευές. Το πρωτόκολλο αυτό διαχωρίζει τον κάθε συνδεδεμένο στο διαδίκτυο υπολογιστή ή συσκευή μέσω ενός μοναδικού αριθμού γνωστού ως IP address (διεύθυνση IP). Τα TCP ή UDP πακέτα περικλείονται συνήθως σε πακέτα IP[6].

#### 3.1.2.2 HTTP

Το πρωτόκολλο HTTP<sup>28</sup> είναι το βασικό πρωτόκολλο μοντέλου Client-Server που εμφανίζεται στο Web και το πιο συμβατό με την υπάρχουσα υποδομή δικτύου, που χρησιμοποιείται καθημερινά από τους Web προγραμματιστές. Η επικοινωνία μεταξύ client και server γίνεται μέσω μηνύματος request/response με τον client να στέλνει ένα HTTP request και τον server να επιστρέφει ένα response που περιέχει τον πόρο που ζητήθηκε, σε περίπτωση αποδοχής του αιτήματος. Υποστηρίζει επίσης τη μεταφορά και TCP και UDP πακέτων.

- Το πρωτόκολλο HTTP ορίζει ένα σύνολο μεθόδων, οι οποίες με τη σειρά τους ορίζουν την επιθυμητή προς εκτέλεση ενέργεια για τη διαχείριση ενός πόρου. Οι βασικοί μέθοδοι που χρησιμοποιούνται ευρέως είναι: POST: Δημιουργία νέου πόρου
- GET: Ανάκτηση πληροφορίας
- PUT: Ενημέρωση του περιεχομένου του πόρου
- DELETE: Διαγραφή ενός πόρου

#### 3.1.2.3 URI

Στην επιστήμη των υπολογιστών, πόρος (resource) ενός συστήματος είναι οποιοδήποτε φυσικό ή λογικό συστατικό, η διαθεσιμότητα του οποίου είναι περιορισμένη σε ένα υπολογιστικό σύστημα. Ως πόρος μπορεί να χαρακτηριστεί ένα ηλεκτρονικό έγγραφο, μια εικόνα, μια πηγή πληροφορίας, μια υπηρεσία ή ακόμα και ένα σύνολο άλλων πόρων. Για να καταστεί εφικτή η χρήση ενός πόρου, θα πρέπει να προσδιορίζεται από τουλάχιστον ένα URI (Universal Resource Identifier–Ομοιόμορφο Αναγνωριστικό Πόρου)<sup>29</sup>. Το URI είναι το όνομα και η διεύθυνση του πόρου. Η πιο συνηθισμένη μορφή URI είναι το URL (Uniform Resource Locator)<sup>30</sup>, το οποίο καθορίζει την διεύθυνση του πόρου στο διαδίκτυο και επιτρέπει την πρόσβαση σε αυτό.

---

<sup>27</sup> [https://en.wikipedia.org/wiki/Internet\\_Protocol](https://en.wikipedia.org/wiki/Internet_Protocol)

<sup>28</sup> [https://en.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol)

<sup>29</sup> <https://tools.ietf.org/html/rfc3986>

<sup>30</sup> [https://el.wikipedia.org/wiki/Uniform\\_Resource\\_Locator](https://el.wikipedia.org/wiki/Uniform_Resource_Locator)

#### 3.1.2.4 XML

Η XML<sup>31</sup> είναι μια γλώσσα σήμανσης, η οποία καθορίζει ένα σύνολο κανόνων για την κωδικοποίηση εγγράφων και τη μετατροπή τους σε αναγνώσιμη μορφή και από τον άνθρωπο και από τη μηχανή. Αποτελεί μια μορφοποίηση δεδομένων κειμένου, με υποστήριξη Unicode για όλες τις γλώσσες του κόσμου. Η σχεδίαση της XML εστιάζει σε έγγραφα, ωστόσο χρησιμοποιείται ευρέως για την αναπαράσταση αυθαίρετων δομών δεδομένων, όπως αυτές που χρησιμοποιούνται στα web services.

#### 3.1.2.5 Υπηρεσίες Διαδικτύου (Web Services) και Τεχνολογίες

Οι διαδικτυακές υπηρεσίες είναι μια τεχνολογία που επιτρέπει την επικοινωνία μεταξύ δυο εφαρμογών, ανεξαρτήτως πλατφόρμας και γλώσσας προγραμματισμού. Σύμφωνα με την IBM ως web service ορίζεται η διεπαφή λογισμικού (software interface) που περιγράφει μια συλλογή από λειτουργίες, οι οποίες είναι προσβάσιμες από το δίκτυο μέσω πρότυπων μηνυμάτων XML. Χρησιμοποιεί πρωτόκολλα βασισμένα στη γλώσσα XML για την περιγραφή μιας λειτουργίας προς εκτέλεση ή δεδομένων προς ανταλλαγή.

##### 3.1.2.5.1 Αρχιτεκτονική REST

Ο όρος REST (REpresentational State Transfer)<sup>32</sup> εισήχθη και ορίστηκε το 2000 από τον Roy Fielding στη διδακτορική του διατριβή στην οποία εμβαθύνει στις διάφορες αρχιτεκτονικές αρχές λογισμικού οι οποίες χρησιμοποιούνται στο διαδίκτυο.

Η αρχιτεκτονική REST ορίζει ένα σύνολο κανόνων για τη σχεδίαση υπηρεσιών διαδικτύου, χρησιμοποιώντας τους πόρους ενός συστήματος. Καθορίζει επίσης και τον τρόπο διευθυνσιοδότησης και μεταφοράς των καταστάσεων του πόρου, μέσω του πρωτοκόλλου HTTP, από ένα ευρύ φάσμα πελατών, ενώ είναι γραμμένες σε διαφορετικές γλώσσες.

Τα τελευταία χρόνια, αποτελεί το κυρίαρχο μοντέλο σχεδίασης υπηρεσιών διαδικτύου. Μια REST διαδικτυακή υπηρεσία ακολουθεί τέσσερις βασικές αρχές σχεδίασης:

- Χρήση βασικών HTTP μεθόδων
- Έλλειψη κατάστασης
- Διευθυνσιοδότηση κάθε πόρου από ένα μοναδικό URI
- Χρήση XML ή/και JSON

---

<sup>31</sup> <https://en.wikipedia.org/wiki/XML>

<sup>32</sup> [https://en.wikipedia.org/wiki/Representational\\_state\\_transfer](https://en.wikipedia.org/wiki/Representational_state_transfer)

### 3.1.2.6 REST API

Ως REST API χαρακτηρίζεται μια διεπαφή προγραμματισμού εφαρμογών (API)<sup>33</sup>, βασισμένη στην αρχιτεκτονική REST, χρησιμοποιώντας τις HTTP μεθόδους. Ένα REST API περιγράφει ένα σύνολο πόρων και ένα σύνολο λειτουργιών που μπορούν να καλέσουν αυτούς τους πόρους. Οι λειτουργίες ενός REST API μπορούν να κληθούν από οποιονδήποτε πελάτη HTTP (HTTP client) [7].

### 3.1.3 Network Robot System

Ως δικτυωμένα Ρομποτικά Συστήματα (Network Robot Systems - NRS) χαρακτηρίζονται τα ρομποτικά συστήματα τα οποία αποτελούνται από δικτυωμένα οχήματα, αισθητήρες, ενεργοποιητές και συσκευές επικοινωνίας και τα οποία είναι ικανά να αλληλεπιδρούν με συνεργατικό τρόπο τόσο με τον άνθρωπο όσο και με άλλα ρομπότ [8].

#### 3.1.3.1 Networked Robot

Ο όρος “δικτυωμένο ρομπότ” (networked robot<sup>34</sup>) αναφέρεται σε ρομποτικές συσκευές που συνδέονται σε ενσύρματα ή/και ασύρματα δίκτυα επικοινωνίας, όπως το Internet ή το LAN, χρησιμοποιώντας πληθώρα πρωτοκόλλων επικοινωνίας.

Υπάρχουν δύο υποκατηγορίες δικτυωμένων ρομπότ:

- **Tele-operated robots (Ρομπότ διαχειριζόμενα από απόσταση)**  
Σ αυτή την κατηγορία ανήκουν τα ρομπότ στα οποία οι άνθρωποι στέλνουν εντολές και λαμβάνουν αποτελέσματα (feedback) μέσω του δικτύου. Τέτοια συστήματα υποστηρίζουν την έρευνα την εκπαίδευση και την ευαισθητοποίηση του κοινού, καθιστώντας πολύτιμους πόρους προσβάσιμους από το ευρύ κοινό.
- **Autonomous robots (Αυτόνομα ρομπότ)**  
Εδώ τα ρομπότ και οι αισθητήρες ανταλλάσσουν δεδομένα μέσω του δικτύου με την ελάχιστη ανθρώπινη αλληλεπίδραση. Σε τέτοια συστήματα, το δίκτυο των αισθητήρων επεκτείνεται πέρα από την αποτελεσματική εμβέλεια ανίχνευσης των ρομπότ, επιτρέποντας την μεταξύ τους επικοινωνία για τον συντονισμό των ενεργειών τους μέσω μεγάλων αποστάσεων. Τα ρομπότ μπορούν να αναπτύξουν, να επιδιορθώσουν και να διατηρήσουν το δίκτυο των αισθητήρων για να αυξήσουν τη μακροζωία και τη χρησιμότητά τους. Cloud Robotics

---

<sup>33</sup> [https://en.wikipedia.org/wiki/Application\\_programming\\_interface](https://en.wikipedia.org/wiki/Application_programming_interface)

<sup>34</sup> <http://www.ieee-ras.org/technicalcommittees/117-technical-committees/networked-robots/146-networked-robots>.

Cloud Robotics (ρομποτική νέφους) ονομάζεται το πεδίο της ρομποτικής που αξιοποιεί το διαδίκτυο ως έναν πόρο για παράλληλους υπολογισμούς και ανταλλαγής τεράστιων ποσοτήτων δεδομένων σε πραγματικό χρόνο. Εκμεταλλεύεται τις υπηρεσίες και τις υποδομές του cloud και άλλες υπάρχουσες τεχνολογίες του διαδικτύου για την ανάπτυξη, την κατασκευή και τη βελτίωση των ρομποτικών συστημάτων [9].

Τα κύρια πλεονεκτήματα από την χρήση του cloud είναι:

- **Big-Data:** πρόσβαση σε απομακρυσμένες βιβλιοθήκες με εικόνες, χάρτες, τροχιές και άλλα σύνολα δεδομένων.
- **Cloud Computing:** πρόσβαση σε υπολογιστικούς πόρους για στατιστικές αναλύσεις, μάθηση και βέλτιστο σχεδιασμό κίνησης.
- **Collective Robot Learning:** επαναχρησιμοποίηση και διαμοιρασμός της γνώσης μεταξύ ρομπότ διαφορετικών ειδών.
- **Open-source and Open-Access:** ανταλλαγή ανοιχτού κώδικα, δεδομένων και σχεδίων για την ανάπτυξη software και hardware για ρομπότ.
- **Crowd-sourcing and Call Centers:** απομακρυσμένη ανθρώπινη καθοδήγηση για τον χειρισμό εξαιρέσεων και την αποκατάσταση σφαλμάτων.

#### 3.1.3.2 Internet of Robotics Things (IoRT)

Το Internet of Robotic Things (IoRT) χαρακτηρίζεται ως μια νέα τεχνολογία η οποία επιτρέπει την ανάπτυξη προηγμένων ρομποτικών υπηρεσιών και τη διασύνδεση των ρομπότ με άλλες ετερογενείς ευφυείς συσκευές σε μια κατακεντρωμένη αρχιτεκτονική από διάφορες πλατφόρμες που λειτουργούν τόσο στην άκρη του δικτύου (edge) όσο και στο cloud, αξιοποιώντας τις υποδομές και υπηρεσίες του cloud και τις υπάρχουσες τεχνολογίες του διαδικτύου [13] [14].

Στην ουσία το IoRT ξεπερνά τις έννοιες των networked robots και του cloud robotics, εκμεταλλευόμενο ορισμένες πτυχές του cloud computing και τις δυνατότητες του IoT, ώστε να υπάρχει ευελιξία στον σχεδιασμό και στην υλοποίηση νέων εφαρμογών για δικτυωμένα ρομποτικά συστήματα[14].

Η αρχιτεκτονική του IoRT είναι ίδια με αυτή του IoT (βλ. υποενότητα 3.2.5). Βαρύτητα, όμως, πρέπει να δοθεί στο επίπεδο του Ενδιάμεσου Λογισμικού (Middleware/Processing/Service and Application Support), το οποίο εμπεριέχει κάποια σημαντικά χαρακτηριστικά για την ανάπτυξη ρομποτικών συστημάτων [14].

- **Robotic Cloud Platform:** υπηρεσίες και τεχνολογίες εξειδικευμένες για ρομπότ, όπως RT (Robot Technology) middleware, Robot Operating System (ROS), Robot Service Network Protocol (RSNP), Open Robot/Resource

interface for the Network (ORiN), CANOpen, open source ubiquitous network robot platform (UNR-PF) κ.ά.

- **M2M2A (Machine-to-Machine-to-Actuator) solutions:** συνδυασμός ρομποτικών τεχνολογιών και πληροφοριών από διάφορους αισθητήρες μέσω δικτύου για την αυτόματη λήψη πρακτικών αποφάσεων από προηγμένα ρομπότ ώστε να επιφέρουν αλλαγές στον φυσικό κόσμο.
- **IoT Cloud Robotics Infrastructure:** αξιοποίηση υπηρεσιών του IoT cloud (όπως επεξεργασία εικόνας και video, αναγνώριση θέσης, έλεγχος επικοινωνίας κ.ά.) από τα ρομποτικά συστήματα.

Το IoRT έχει διάφορα χαρακτηριστικά που διαφέρουν από τις παραδοσιακές υπηρεσίες της ρομποτικής (networked robots, cloud robotics) και τα οποία συνοψίζονται παρακάτω[14]:

- **Απλότητα:** Απλοποιείται η δημιουργία πολύπλοκων εφαρμογών με τη χρήση βασικών στοιχείων του web.
- **Ευαισθητοποίηση περιβάλλοντος(context awareness):** Οι αποφάσεις των ρομποτικών συστημάτων είναι context-aware, καθώς λαμβάνουν πληροφορίες σχετικά με τον περιβάλλον χώρο από διάφορους αισθητήρες.
- **Διαφοροποίηση:** : Ο τελικός χρήστης μπορεί να λαμβάνει μόνο τις υπηρεσίες που επιθυμεί, χωρίς να χρειάζεται να λάβει υπόψιν του τα φυσικά χαρακτηριστικά των ρομπότ που τις εκτελούν.
- **Επεκτασιμότητα:** Υποστηρίζεται η επέκταση των ήδη υπαρχόντων ρομποτικών υπηρεσιών, με την προσθήκη νέων ρομπότ ή την αναβάθμιση των υπηρεσιών που παρέχοντα
- **Διαλειτουργικότητα:** Υποστηρίζεται η επικοινωνία με την υποδομή ή με συσκευές διαφορετικού είδους μέσω διαφόρων διαλειτουργικών πρωτοκόλλων επικοινωνίας.
- **Προσαρμοστικότητα:** Υπάρχει η δυνατότητα λήψης αποφάσεων και δυναμικής προσαρμογής με βάση τις μεταβαλλόμενες συνθήκες του περιβάλλοντος και τις συνθήκες λειτουργίας τους.
- **Απομακρυσμένη διαχείριση:** Πολλά ρομποτικά συστήματα αποτελούνται από κέντρα δεδομένων που βρίσκονται σε απομακρυσμένες γεωγραφικές περιοχές ανά την υφήλιο, ενώ ο έλεγχος τους πραγματοποιείται απομακρυσμένα μέσω cloud υπηρεσιών από οποιαδήποτε συσκευή με πρόσβαση στο Internet.



## 3.2 Τεχνολογικό Υπόβαθρο και Εργαλεία

### 3.2.1 Robot Operating System (ROS)

Το ROS<sup>35</sup> είναι ένα μεταλειτουργικό σύστημα (meta-operating system) για την ανάπτυξη λογισμικού για ρομπότ. Εκτελείται πάνω σε ένα λειτουργικό σύστημα (κυρίως Linux) και παρέχει μια συλλογή από εργαλεία και βιβλιοθήκες, μαζί με ένα δίκτυο μέσω του οποίου μπορούν όλα τα προγράμματα να επικοινωνούν μεταξύ τους. Όλες αυτές οι δυνατότητες στοχεύουν στην απλοποίηση του έργου δημιουργίας περίπλοκων και ισχυρών συμπεριφορών ρομπότ, υποστηρίζοντας παράλληλα ένα μεγάλο αριθμό ρομποτικών συστημάτων [15] [30].

Τα βασικά στοιχεία από τα οποία συγκροτείται το ROS είναι:

- **Nodes**<sup>36</sup>: Ένας κόμβος (node) είναι πρακτικά ένα εκτελέσιμο αρχείο που υλοποιεί μία λειτουργικότητα.
- **Messages**<sup>37</sup>: Ο τύπος δεδομένων ROS που χρησιμοποιείται κατά την ανάγνωση ή δημοσίευση σε ένα Topic, Service ή Action.
- **Topics**<sup>38</sup>: Οι κόμβοι μπορούν να δημοσιεύουν (Publish) μηνύματα σε ένα topic καθώς και να εγγραφούν (Subscribe) σε ένα topic για τη λήψη μηνυμάτων.
- **Services**<sup>39</sup>: Πρόκειται για έναν άλλον τρόπο με τον οποίο οι κόμβοι μπορούν να επικοινωνούν μεταξύ τους, γνωστό και ως RPC (remote procedure call). Η βασική του διαφορά με παραπάνω είναι ότι ένας κόμβος μπορεί να καλέσει μια υπηρεσία για να πάρει την απάντηση σε ένα ερώτημα που υλοποιείται σε έναν άλλο κόμβο.
- **Actions**<sup>40</sup>: Όπως και στα services, τα actions χρησιμοποιούνται για την εκτέλεση ενός αιτήματος από τον server και την αποστολή του αποτελέσματος στον client, με τη διαφορά ότι προσφέρει τη δυνατότητα ακύρωσης της διαδικασίας πριν την ολοκλήρωση της, αλλά και την παρακολούθηση του σταδίου στο οποίο βρίσκεται η διαδικασία.
- **Parameter Server**<sup>41</sup>: Ο διακομιστής παραμέτρων είναι ένα κοινόχρηστο λεξικό πολλαπλών επιλογών, το οποίο είναι προσβάσιμο μέσω API. Οι κόμβοι χρησιμοποιούν αυτόν τον διακομιστή για την αποθήκευση και ανάκτηση παραμέτρων κατά το χρόνο εκτέλεσης

---

<sup>35</sup> <https://www.ros.org/>

<sup>36</sup> <http://wiki.ros.org/Nodes>

<sup>37</sup> <http://wiki.ros.org/msg>

<sup>38</sup> <http://wiki.ros.org/Topics>

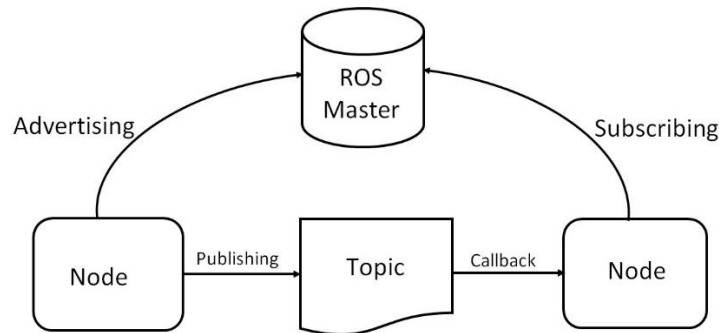
<sup>39</sup> <http://wiki.ros.org/Services>

<sup>40</sup> <http://wiki.ros.org/actionlib>

<sup>41</sup> <http://wiki.ros.org/Parameter%20Server>



- **Master<sup>42</sup>**: Βασική λειτουργία του είναι να οργανώνει τους nodes και τα διάφορα topics και services με αποτέλεσμα να επιτρέπει σε κάθε node να μπορεί να επικοινωνήσει με τα υπόλοιπα, όπως παρουσιάζεται στην εικόνα 9.



Εικόνα 9: Τα βασικά στοιχεία του ROS

Η ισχυρή υπηρεσία ονοματοδοσίας του ROS οργανώνει ιεραρχικά τους πόρους του δικτύου (nodes, topics, services). Διαθέτει επίσης ένα χαρακτηριστικό το οποίο ονομάζεται “pushing” για να οργανώνει και να ταυτοποιεί αυτούς τους πόρους. Με αυτό το τρόπο γίνεται πιο εύκολη και σωστή η αναφορά σε αυτούς τους πόρους, με σχετικό ή απόλυτο τρόπο, προωθώντας έτσι τη δυνατότητα επαναχρησιμοποίησης και την αποφυγή προβλημάτων συγκρούσεων ονόματος. Ο master παρέχει ένα σύστημα αναζήτησης υπηρεσιών (lookup service) για την αναζήτηση οποιασδήποτε καταχωρημένης υπηρεσίας που πληροί τις απαιτήσεις για ορισμένα χαρακτηριστικά.

Οι κύριες γλώσσες που χρησιμοποιούνται στο ROS είναι η C++ και η Python. Άλλες δευτερεύουσες γλώσσες περιλαμβάνουν Java, LISP, Octave και LabView. Όσον αφορά τη διαλειτουργικότητα μεταξύ των γλωσσών, τα σημεία σύνδεσης είναι οι διεπαφές των μηνυμάτων και υπηρεσιών. Το βασικό χαρακτηριστικό που κάνει το ROS διαθέσιμο σε διαφορετικές γλώσσες είναι το σύνολο εργαλείων για την παραγωγή μηνυμάτων (genmsg, gensrv). Αυτά τα εργαλεία λαμβάνουν μια περιγραφή γλώσσας για τα μηνύματα ή τις υπηρεσίες και παράγουν συγκεκριμένο κώδικα στην αναγκαία γλώσσα.

Το ROS εστιάζεται σε συστήματα που βασίζονται σε Unix (Linux, OSX, κ.λπ.). Έχει επίσης υλοποιηθεί μια μικρή βιβλιοθήκη ROS ώστε να μπορεί να χρησιμοποιείται σε μικροελεγκτές AVR και υποστηρίζεται η ανάπτυξη κόμβων που χρησιμοποιούν δίκτυο EtherCAT<sup>43</sup>. Ωστόσο, το ROS είναι ανεπαρκές για εφαρμογές

<sup>42</sup> <http://wiki.ros.org/Master>

<sup>43</sup> <https://en.wikipedia.org/wiki/EtherCAT>

σε πραγματικό χρόνο, δεδομένου ότι ούτε το υποκείμενο λειτουργικό σύστημα, ούτε ο κύριος μηχανισμός μεταφοράς (TCP / IP) είναι κατάλληλα.

Ένα από τα ισχυρότερα σημεία του ROS είναι ο υψηλός αριθμός ρομποτικών πακέτων λογισμικού με drivers και αλγόριθμους ρομποτικής που παρέχει. Πακέτα χαμηλού επιπέδου όπως κινηματικά μοντέλα, σχεδιασμός κίνησης, Bayesian φιλτράρισμα, μηχανική μάθηση, επεξεργασία εικόνας αντίληψη τριών διαστάσεων και πακέτα υψηλού επιπέδου όπως πλοήγηση, αναγνώριση και χειρισμός αντικειμένων κ.α. Όλη αυτή η ποικιλία καθίσταται δυνατή από τη φιλοσοφία ενσωμάτωσης με άλλες βιβλιοθήκες ρομποτικής (OpenCV, OpenSLAM, κ.α.). Το ROS προσφέρει συμβατότητα με άλλα ρομποτικά πλαίσια, όπως το YARP, το OpenRTM, το OROCOS, και OpenRave. Παρέχει επίσης μια καλή υποστήριξη από προσομοιωτές, όπως Stage, Gazebo και Webots.

Τέλος, σε αντίθεση με άλλα ρομποτικά πλαίσια, το ROS είναι περισσότερο από ένα σύνολο βιβλιοθηκών που παρέχουν έναν μηχανισμό επικοινωνίας και ένα πρωτόκολλο. Αντίθετα, οι κόμβοι αναπτύσσονται μέσα σε ένα build system που παρέχεται από το ROS. Η πρόθεση είναι ότι ένα σύστημα που εκτελεί ROS θα πρέπει να αποτελείται από πολλές ανεξάρτητες μονάδες. Το build system είναι χτισμένο πάνω από το CMake, το οποίο εκτελεί αρθρωτές κατασκευές και των δύο κόμβων καθώς και τα μηνύματα που μεταδίδονται μεταξύ τους [14].

### 3.2.2 RabbitMQ

Το RabbitMQ<sup>44</sup> είναι ένα λογισμικό ανοιχτού τύπου, το οποίο ανήκει στην κατηγορία των διαμεσολαβητών μηνυμάτων (message brokers) που χρησιμοποιούν τεχνολογία ουράς στην δομή τους (message-queueing). Με απλά λόγια είναι μια τεχνολογία όπου οι ουρές μπορούν να καθοριστούν και οι εφαρμογές μπορούν να συνδεθούν στην ουρά και να μεταφέρουν ένα μήνυμα σε αυτήν. Οι ουρές μηνυμάτων υλοποιούν την ασύγχρονη επικοινωνία μεταξύ των εφαρμογών κατά την οποία εφαρμογές που παράγουν (publish) μηνύματα και άλλες που καταναλώνουν (subscribe) δεν επικοινωνούν άμεσα μεταξύ τους αλλά μέσω του διαμεσολαβητή μηνυμάτων [12].

Ο message broker αποθηκεύει τα μηνύματα μέχρι να συνδεθεί μια εφαρμογή λήψης και λάβει ένα μήνυμα από την ουρά. Η εφαρμογή η οποία έλαβε το μήνυμα μπορεί στην συνέχεια να το επεξεργαστεί με οποιοδήποτε τρόπο επιθυμεί. Ο publisher μπορεί να προσθέτει μηνύματα σε μια ουρά χωρίς να χρειάζεται να περιμένει τότε αυτά θα καταναλωθούν, διαδικασία η οποία απεικονίζεται στην εικόνα 10.

---

<sup>44</sup> <https://www.rabbitmq.com/documentation.html>



Εικόνα 10: Η επικοινωνία τύπου Publish-Subscribe μέσω του RabbitMQ

### 3.2.2.1 RabbitMQ Server Concepts

Τα επιμέρους κομμάτια τα οποία είναι απαραίτητο να καθοριστούν ώστε να επιτευχθεί η επικοινωνία μέσω του RabbitMQ παρουσιάζονται παρακάτω:

- **Producer:** Εφαρμογή η οποία στέλνει το μήνυμα σε ένα συγκεκριμένο topic.
- **Consumer:** Εφαρμογή η οποία λαμβάνει το μήνυμα και αναλόγως ορίζεται η διαδικασία που θα ακολουθηθεί.
- **Queue:** Buffer ο οποίος αποθηκεύει τα μηνύματα.
- **Message:** Τα δεδομένα τα οποία στέλνονται από τον publisher στον subscriber μέσω του πρωτοκόλλου επικοινωνίας AMQP.
- **Channel:** Το κανάλι ουσιαστικά αποτελεί μια εικονική σύνδεση μέσα στην σύνδεση. Η επικοινωνία μεταξύ publisher και subscriber υλοποιείται μέσω ενός καναλιού.
- **Exchange:** Έχει τέσσερις διαφορετικούς τύπους γεγονόσ που καθορίζει τον τρόπο με οποίο λαμβάνονται τα μηνύματα από τους publishers και προωθούνται στις ουρές, διαδικασία η οποία αναλύεται εκτενέστερα στην επόμενη υποενότητα.
- **Binding:** Είναι ο συνδετικός κρίκος μεταξύ queue και exchange.
- **Routing key:** Αποτελεί το κλειδί βάσει του οποίου ορίζεται ο τρόπος με τον οποίο θα δρομολογηθούν τα μηνύματα στις ουρές.

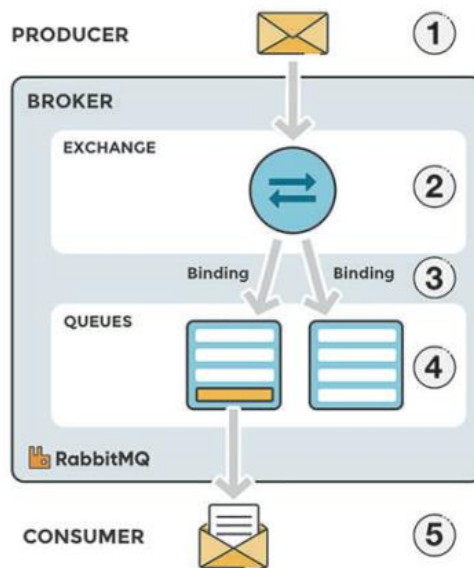
### 3.2.2.2 Exchanges

Τα μηνύματα δεν στέλνονται κατευθείαν στις ουρές, αλλά περνάνε πρώτα από το στάδιο των exchanges. Ένα exchange είναι υπεύθυνο για την κατάλληλη προώθηση των μηνυμάτων σε διαφορετικές ουρές. Ο ρόλος λοιπόν των exchanges είναι να δέχονται μηνύματα από τους producers και αναλόγως να τα μεταφέρουν στις σωστές ουρές, διαδικασία η οποία επιτυγχάνεται με την βοήθεια των bindings και routing keys τα οποία αναλύονται εκτενέστερα παρακάτω.

Η ροή την οποία ακολουθεί ένα μήνυμα για να μεταδοθεί από τον producer στον consumer φαίνεται στην εικόνα 11. Τα διακριτά βήματα μετάδοσης των μηνυμάτων αναγράφονται πιο κάτω:

1. Ο producer δημοσιεύει ένα μήνυμα σε ένα exchange, το είδος του οποίου πρέπει να καθοριστεί την στιγμή της δημιουργίας του. Τα διάφορα είδη των exchanges θα αναλυθούν παρακάτω.

2. Ο exchange λαμβάνει το μήνυμα και είναι υπεύθυνος να το προωθήσει στις ουρές διαδικασία η οποία καθορίζεται από το είδος του exchange κάθε φορά.
3. Στην συγκεκριμένη περίπτωση έχουμε δύο διαφορετικά είδη σύνδεσης μεταξύ των exchanges και queues τα οποία καθορίζονται από την τιμή της μεταβλητής binding.
4. Το μήνυμα παραμένει στην ουρά μέχρι να το διαχειριστεί ο consumer.
5. Όταν ο consumer λάβει το μήνυμα, αυτό αφαιρείται από την ουρά.



Εικόνα 11: Ροή μηνύματος από τον Publisher στον Subscriber

Το RabbitMQ υποστηρίζει τέσσερα διαφορετικά είδη exchanges τα οποία παρουσιάζονται παρακάτω:

- **Direct:** Μεταβιβάζει τα μηνύματα στις ουρές ανάλογα με το routing key του κάθε μηνύματος. Στην περίπτωση αυτού του είδους exchange, το μήνυμα προωθείται στην ουρά της οποίας το binding key ταυτίζεται με αυτό του μηνύματος.
- **Topic:** Σ' αυτή την περίπτωση, το μήνυμα προωθείται σε μια ή περισσότερες ουρές ανάλογα με την αντιστοίχιση μεταξύ του routing key του μηνύματος και του pattern που χρησιμοποιήθηκε κατά την δημιουργία της ουράς.
- **Fanout:** Σ' αυτό το είδος, τα μηνύματα προωθούνται από τα exchanges στις ουρές οι οποίες έχουν συνδεθεί με αυτά, αδιαφορώντας για τα binding keys.
- **Header:** Χρησιμοποιούν την επικεφαλίδα του μηνύματος και ανάλογα πραγματοποιούν την προώθηση των μηνυμάτων από τα exchanges στις ουρές.

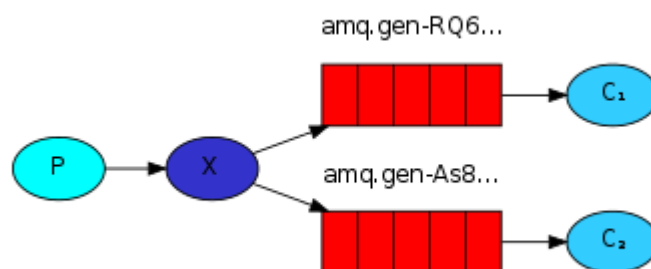
### 3.2.2.3 Μοντέλα Επικοινωνίας

Ο message broker RabbitMQ υποστηρίζει διαφορετικά μοντέλα μέσω των οποίων πραγματοποιείται η επικοινωνία μεταξύ των συσκευών που συνδέονται σε αυτόν. Στις παρακάτω υποενότητες περιγράφονται αναλυτικά τα δύο θεμελιώδη μοντέλα επικοινωνίας και ο τρόπος με τον οποίο μπορούν να υλοποιηθούν.

#### 3.2.2.3.1 Μοντέλο επικοινωνίας publish-subscribe

Η πρώτη και βασικότερη μέθοδος την οποία υποστηρίζει το RabbitMQ για την υλοποίηση της επικοινωνίας είναι η μέθοδος publish-subscribe (pub/sub). Η φιλοσοφία αυτής της μεθόδου είναι η ανταλλαγή μηνυμάτων μεταξύ των συσκευών «κάτω» από ένα συγκεκριμένο topic. Έτσι παραδείγματος χάρη μια συσκευή μπορεί να κάνει publish σ ένα topic και μία άλλη κάνοντας subscribe στο topic αυτό να λαμβάνει τις τιμές του μηνύματος το οποίο στάλθηκε από τον publisher. Στη συνέχεια ο subscriber μόλις λάβει το μήνυμα μπορεί να το επεξεργαστεί με τρόπο ο οποίος ορίζεται εξ αρχής. Κατ' αυτό τον τρόπο επιτυγχάνεται η σύνδεση των δυο συσκευών γεγονός που μπορεί να οδηγήσει σε διάφορες δράσεις.

Μια αναπαράσταση της διαδικασίας που ακολουθείται ώστε να πραγματοποιηθεί η επικοινωνία τύπου pub/sub, παρουσιάζεται στην εικόνα 12. Ο Publisher δεν μπορεί να επικοινωνήσει κατευθείαν με την ουρά η οποία αποθηκεύει το μήνυμα, σε αντίθεση, στέλνει το μήνυμα στο exchange, όπου ανάλογα με το είδος του ορίζεται ο τρόπος με τον οποίο θα προωθηθεί στις ουρές. Η σύνδεση της κάθε ουράς με το exchange, επιτυγχάνεται μέσω των bindings. Μέσω αυτής της διαδικασίας το μήνυμα, φτάνει στον consumer για την τελική επεξεργασία.



Εικόνα 12: Διαδικασία υλοποίησης της επικοινωνίας Publish-Subscribe

Ένα παράδειγμα στο οποίο μπορεί να εφαρμοστεί αυτό το πλαίσιο επικοινωνίας είναι μεταξύ μιας συσκευής ανίχνευσης καπνού και μιας συσκευής πυρόσβεσης. Έπειτα, για να μπορέσει να κάνει publish την τιμή του καπνού απαραίτητη προϋπόθεση αποτελεί ο καθορισμός του topic, στο οποίο θα δημοσιεύονται τα δεδομένα και ο ρυθμός δημοσίευσής τους. Από την άλλη η συσκευή πυρόσβεσης για να λάβει τα δεδομένα πρέπει με τη σειρά της να ακολουθήσει τον ίδιο τρόπο σύνδεσης με το RabbitMQ, να ορίσει το topic από το

οποίο θα παίρνει τα δεδομένα, το όνομα του οποίου πρέπει να είναι κοινό με τον publisher. Τέλος για την περίπτωση του subscriber ορίζεται και μια συνάρτηση callback η οποία ενεργοποιείται μόλις γίνει η λήψη των δεδομένων, και καθορίζει τον τρόπο με τον οποίο αυτά θα επεξεργαστούν.

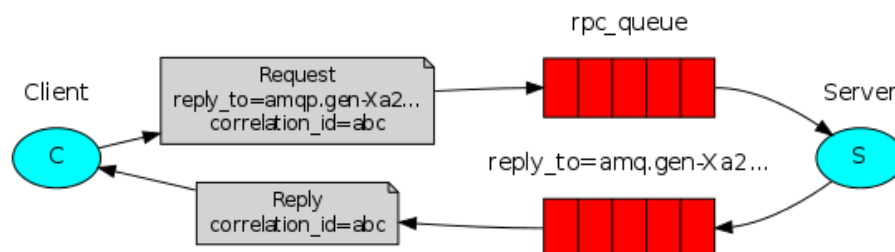
### 3.2.2.3.2 Μοντέλο επικοινωνίας RPC

Με τον όρο RPC (remote procedure call) ορίζουμε μια διαδικασία η οποία εκτελείται σε ένα απομακρυσμένο σύστημα, είτε αυτός είναι υπολογιστής, είτε ένα κοινό δίκτυο και είναι γραμμένη ακριβώς με τον ίδιο τρόπο που θα αναπτυσσόταν αν εκτελούνταν τοπικά. Αποτελεί ουσιαστικά μια μορφή επικοινωνίας client-server η οποία επιτυγχάνεται μέσω request-response μηνυμάτων. Αυτή η μέθοδος αποτελεί τον δεύτερο τρόπο επικοινωνίας των συσκευών που συνδέονται στο RabbitMQ μετά την μέθοδο pub-sub η οποία αναπτύχθηκε παραπάνω. Αποτελείται από δύο μέρη, τον rpc server ο οποίος εκτελεί μια συγκεκριμένη λειτουργία και τον rpc client ο οποίος στέλνει τα δεδομένα (request) στον server για να λάβει τα αποτελέσματα της διαδικασίας η οποία εκτελέστηκε (response).

Πιο αναλυτικά, η διαδικασία και ο τρόπος με τον οποίο επιτυγχάνεται αυτή η επικοινωνία αναλύετε στα επόμενα βήματα (εικόνα 13).

- Όταν ο client ξεκινά να λειτουργεί δημιουργείται μια callback queue η οποία είναι μοναδική.
- Στην περίπτωση ενός rpc request προς τον server, ο client στέλνει ένα μήνυμα το οποίο έχει δύο παραμέτρους. Την παράμετρο **reply\_to** η οποία «δείχνει» στην callback queue που έχει υλοποιηθεί στο πρώτο βήμα, και την παράμετρο **correlation\_id**, η οποία είναι μοναδική για κάθε client request.
- Όταν ο server λάβει το rpc request, εκτελεί την διαδικασία και στέλνει τα αποτελέσματα πίσω στον client χρησιμοποιώντας την reply\_to queue.

Όταν ο client λάβει το μήνυμα επιστροφής ελέγχει το correlation\_id και αν αυτό ταυτίζεται με την τιμή την οποία είχε στείλει στον server συνεχίζει στην εκτέλεση της εφαρμογής.



Εικόνα 13: Υλοποίηση της RPC μεθόδου επικοινωνίας

#### 3.2.2.4 Πρωτόκολλα Επικοινωνίας

Ο message broker RabbitMQ αναπτύχθηκε για να χρησιμοποιεί κυρίως το πρωτόκολλο επικοινωνίας AMQP, το οποίο αποτελεί και τον πυρήνα του. Ωστόσο υποστηρίζει και άλλα πρωτοκόλλα επικοινωνίας, όπως το MQTT, το STOMP αλλά και το πρωτόκολλο HTTP, τα οποία μπορούν να συνδυαστούν, δίνοντας με αυτόν τον τρόπο νέες δυνατότητες στην ανάπτυξη εφαρμογών. Πρόκειται για μια πολύ σπουδαία ιδιότητα του message broker, καθώς δίνεται η δυνατότητα αλληλεπίδρασης μεταξύ εφαρμογών που χρησιμοποιούν διαφορετικά πρωτόκολλα επικοινωνίας. Άλλωστε το κάθε πρωτόκολλο επικοινωνίας έχει τα δικά του χαρακτηριστικά και χρησιμοποιείται ανάλογα με την εφαρμογή που αναπτύσσεται κάθε φορά.

Αρχικά το RabbitMQ υποστηρίζει το πρωτόκολλο επικοινωνίας AMQP 0-9-1, όπως παρουσιάστηκε αναλυτικά στην υποενότητα 2.2.6.2, το οποίο είναι ένα δυαδικό πρωτόκολλο που ορίζει αρκετά ισχυρή σημασιολογία μηνυμάτων (messaging semantics). Είναι ένα εύκολο πρωτόκολλο στην ανάπτυξη του και ως εκ τούτου υπάρχει ένας μεγάλος αριθμός βιβλιοθηκών διαθέσιμος για πολλές διαφορετικές γλώσσες και περιβάλλοντα προγραμματισμού. Στην παρούσα διπλωματική εργασία χρησιμοποιήθηκε η βιβλιοθήκη “rika” της γλώσσας προγραμματισμού python.

Ο message broker RabbitMQ υποστηρίζει και το πρωτόκολλο επικοινωνίας STOMP, η αναλυτική περιγραφή του οποίου παρουσιάστηκε στο κεφάλαιο 2.2.6.4. Το STOMP λοιπόν, είναι ένα πρωτόκολλο μηνυμάτων που βασίζεται σε κείμενο και δίνει έμφαση στην απλότητα. Ορίζει ελάχιστα την σημασιολογία μηνυμάτων, παρ’ όλ’ αυτά είναι εύκολο να εφαρμοστεί και ακόμα πιο εύκολο να εφαρμοστεί μερικώς. Άλλωστε, είναι το μοναδικό πρωτόκολλο που μπορεί να χρησιμοποιηθεί χειροκίνητα μέσω telnet<sup>45</sup>.

Το πρωτόκολλο επικοινωνίας MQTT, το οποίο υποστηρίζεται από το RabbitMQ και αναλύθηκε στην ενότητα 2.2.6.1, αποτελεί ένα δυαδικό πρωτόκολλο που δίνει έμφαση στην «ελαφριά» υλοποίηση του μοντέλου επικοινωνίας publish-subscribe, με στόχο συγκεκριμένες συσκευές. Έχει σαφώς καθορισμένη σημασιολογία μηνυμάτων σε επίπεδο επικοινωνίας publish-subscribe, αλλά δεν ενδείκνυται σε άλλες περιπτώσεις.

#### 3.2.3 Swagger Api

Η προδιαγραφή OpenAPI<sup>46</sup> (πρώην Swagger Specification) είναι ένα μοντέλο περιγραφής API μέσω προδιαγραφών για REST API (βλέπε υποενότητα 2.3.6). Ένα

---

<sup>45</sup> <https://el.wikipedia.org/wiki/Telnet>

<sup>46</sup> <https://swagger.io/specification/>



---

αρχείο OpenAPI επιτρέπει την πλήρη περιγραφή των προδιαγραφών ενός Web API, συμπεριλαμβανομένων:

- Διαθέσιμων τερματικών υπηρεσιών και των λειτουργιών τους (GET, POST, κ.α.)
- Παραμέτρων και πληροφοριών που αποστέλλονται καθώς και των απαντήσεων που περιμένει να λάβει από τον server.
- Μεθόδων αναγνώρισης και ταυτοποίησης.
- Στοιχείων επικοινωνίας, αδειών χρήσης, όρων χρήσης και άλλων πληροφοριών.

Το Swagger είναι ένα σύνολο εργαλείων ανοικτού κώδικα που βασίζονται στην προδιαγραφή OpenAPI που μπορεί να βοηθήσει και να επιταχύνει τη σχεδίαση, τη δημιουργία, την τεκμηρίωση και τέλος την επαναχρησιμοποίηση των Web APIs[17].

Τα πιο ευρέως διαδεδομένα εργαλεία Swagger περιλαμβάνουν:

- **Swagger Editor:** Εφαρμογή που χρησιμοποιείται για τη συγγραφή προδιαγραφών τύπου OpenAPI, η οποία ενσωματώνει εργαλεία αποσφαλμάτωσης.
- **Swagger UI:** Επιτρέπει τη γραφική απεικόνιση των προδιαγραφών και την αλληλεπίδραση με αυτές.
- **Swagger Codegen:** Δημιουργεί αρχεία διακομιστή και βιβλιοθήκες πελάτη από ένα αρχείο περιγραφής προδιαγραφών OpenAPI

### 3.2.4 Python Flask

Το Flask<sup>47</sup> είναι ένα micro web framework υλοποιημένο σε Python, το οποίο παρέχει εργαλεία, βιβλιοθήκες και τεχνολογίες που επιτρέπουν τη δημιουργία μιας διαδικτυακής εφαρμογής. Αυτή η εφαρμογή μπορεί να είναι μερικές ιστοσελίδες, ένα blog, ένα wiki ή να είναι τόσο μεγάλη όσο μια εφαρμογή ημερολογίου ή ένας εμπορικός ιστότοπος. Ονομάζεται micro framework, διότι δεν απαιτεί ειδικά εργαλεία ή βιβλιοθήκες γεγονός που συνεπάγεται πλεονεκτήματα αλλά και μειονεκτήματα. Στα πλεονεκτήματα μπορούμε να συγκαταλέγουμε τους ελάχιστους υπολογιστικούς πόρους που απαιτεί, καθώς και την μικρή εξάρτηση για τυχόν αναβαθμίσεις και κενά ασφαλείας. Από την άλλη πλευρά, χρειάζεται «χειροκίνητη» εγκατάσταση επεκτάσεων από τον χρήστη για την προσθήκη επιπλέον λειτουργιών.

---

<sup>47</sup> <https://www.fullstackpython.com/flask.html>



### 3.2.5 Mongo DB και Robo3T

Η MongoDB<sup>48</sup> είναι μία ανοιχτού κώδικα NoSQL βάση δεδομένων η οποία χρησιμοποιεί έγγραφα τύπου JSON. Τα βασικά πλεονεκτήματα που παρουσιάζει η MongoDB σε αντίθεση με τις σχεσιακές βάσεις δεδομένων είναι :

- **Ευέλικτο μοντέλο δεδομένων:** Οι βάσεις δεδομένων NoSQL προέκυψαν για να αντιμετωπίσουν τις απαιτήσεις για τα δεδομένα που κυριαρχούν στις σύγχρονες εφαρμογές. Η μορφή τους προσφέρει ένα ευέλικτο μοντέλο δεδομένων, διευκολύνει την αποθήκευση και τον συνδυασμό δεδομένων οποιασδήποτε δομής και επιτρέπει τη δυναμική τροποποίηση της μορφής τους χωρίς να δημιουργείται κάποιας μορφής επίπτωση στην επίδοση.
- **Επεκτασιμότητα και απόδοση:** Οι βάσεις δεδομένων NoSQL χτίστηκαν με έμφαση στην επεκτασιμότητα. Αυτό επιτρέπει στη βάση δεδομένων την εκτέλεση στο ακόμα και στο νέφος, επιτρέποντας σχεδόν απεριόριστη ανάπτυξη με υψηλότερη απόδοση και χαμηλότερη καθυστέρηση από τις σχεσιακές βάσεις δεδομένων

Το Robo3T (πρώην Robomongo)<sup>49</sup> είναι ένα εργαλείο το οποίο τρέχει σε διάφορα λειτουργικά συστήματα (cross platform) και επιτρέπει την διαχείριση της βάσης δεδομένων. Προσφέρει ένα γραφικό περιβάλλον διεπαφής χρήστη όπου δίνεται η δυνατότητα για εύκολο έλεγχο και αλληλεπίδραση με τα περιεχόμενα της βάσης δεδομένων[20].

### 3.2.6 Jinja2

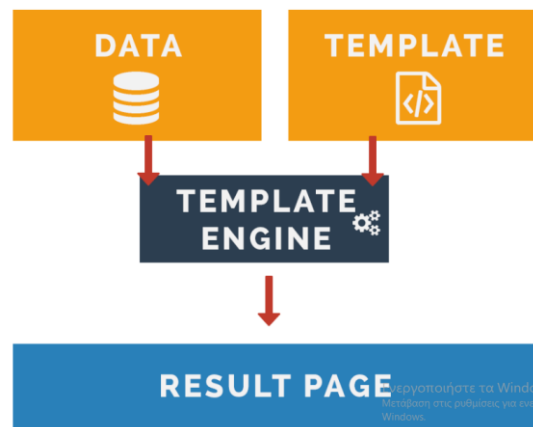
Μια μηχανή δημιουργίας template (template engine)<sup>50</sup>, είναι ένα λογισμικό το οποίο συνδυάζει ένα template με ένα μοντέλο δεδομένων (data model) με σκοπό την δημιουργία τελικών εγγράφων όπως παρουσιάζεται στην εικόνα 14. Πρακτικά επιτρέπει την δημιουργία εφαρμογών οι οποίες όταν εκτελεστούν αντικαθιστούν τις μεταβλητές των στατικών templates που έχουν υλοποιηθεί, με πραγματικές τιμές, μετατρέποντας έτσι το template σε ένα τελικό αρχείο. Χρησιμοποιείται ευρέως στον σχεδιασμό ιστοτόπων σε γλώσσα HTML καθώς και σε ανάπτυξη εφαρμογών που βασίζονται σε μεταβλητές πραγματικού χρόνου.

---

<sup>48</sup> <https://www.mongodb.com/what-is-mongodb>

<sup>49</sup> <https://robomongo.org/>

<sup>50</sup> [https://en.wikipedia.org/wiki/Template\\_engine](https://en.wikipedia.org/wiki/Template_engine)



Εικόνα 14:Λειτουργία της μηχανής δημιουργίας template

Η Jinja2<sup>51</sup> αποτελεί γλώσσα μια δημιουργίας template (text-based template language), γραμμένη με και για τη γλώσσα Python. Δημιουργήθηκε από τον Ronacher Armin και ξεκίνησε ως ένα όμοιο σύστημα template με του Django<sup>52</sup> αλλά με τον καιρό επεκτάθηκε και πλέον αποτελεί ένα ισχυρότερο εργαλείο παρέχοντας περισσότερες δυνατότητες στους χρήστες της.

Διαθέτει πλήρη υποστήριξη, είναι γρήγορη αλλά και ασφαλής και παρέχει ένα ολοκληρωμένο περιβάλλον εκτέλεσης κώδικα που χρησιμοποιείται ευρέως με την άδεια της BSD ( Berkeley Software Distribution).

### 3.2.7 Node-RED

Το Node-RED<sup>53</sup> είναι ένα flow-based, αναπτυξιακό εργαλείο ανοικτού κώδικα, για ανάπτυξη εφαρμογών Web/IoT με χρήση διαγραμμάτων ροής δεδομένων, το οποίο αναπτύχθηκε αρχικά από την IBM για τη διασύνδεση συσκευών υλικού, APIs και διαδικτυακών υπηρεσιών ως μέρος του IoT.

Το Node-RED παρέχει έναν web browser-based editor, ο οποίος περιλαμβάνει μια μεγάλη ποικιλία κόμβων που παρέχουν διάφορες λειτουργίες. Πέρα από τους έτοιμους κόμβους που παρέχονται από το εργαλείο, είναι δυνατή και η δημιουργία custom κόμβων που τρέχουν Javascript κώδικα. Στοιχεία εφαρμογών μπορούν να αποθηκευτούν ή να μοιραστούν για επαναχρησιμοποίηση. Ο χρόνος εκτέλεσης (runtime) είναι χτισμένος στο Node.js, επιτρέποντας του να τρέχει σε συσκευές αλλά και στο Cloud. Οι ροές που δημιουργούνται στο Node-RED αποθηκεύονται χρησιμοποιώντας JSON αρχεία.

<sup>51</sup> <https://jinja.palletsprojects.com/en/2.10.x/>

<sup>52</sup> <https://docs.djangoproject.com/en/2.2/topics/templates/>

<sup>53</sup> <https://nodered.org/docs/>

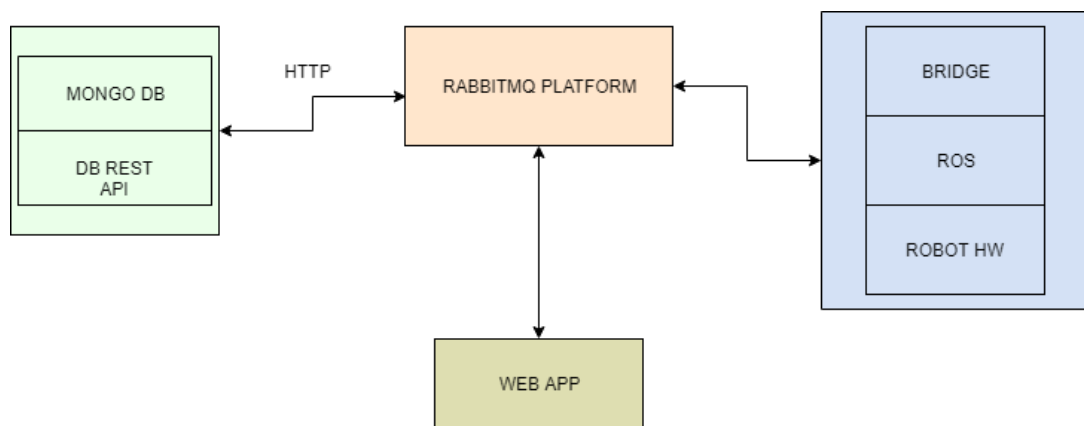
Τέλος, περιλαμβάνει μια μεγάλη ποικιλία εργαλείων για την αποσφαλμάτωση και την οπτικοποίηση των λειτουργιών και τη ροή των δεδομένων όπως το Node-RED Dashboard και το Node-RED Line Tool.

## 4 ΥΛΟΠΟΙΗΣΗ ΣΥΣΤΗΜΑΤΟΣ

Στο κεφάλαιο αυτό παρουσιάζονται τα επιμέρους στοιχεία του συστήματος που υλοποιήθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας. Αρχικά περιγράφεται η αρχιτεκτονική στην οποία βασίζεται η υλοποίηση και στην συνέχεια αναλύονται οι επιμέρους τεχνολογίες και τα εργαλεία τα οποία συνδυάστηκαν ώστε να επιτευχθεί η τελική έκδοχή της εφαρμογής.

### 4.1 Αρχιτεκτονική

Η αρχιτεκτονική του συστήματος χωρίζεται σε τέσσερα επιμέρους κομμάτια (εικόνα 15), απαραίτητα για την ορθή λειτουργία του.



Εικόνα 15: Αρχιτεκτονική συστήματος

Στο πρώτο κομμάτι έχουμε μία βάση δεδομένων η οποία είναι τύπου NoSQL (mongo) και είναι υπεύθυνη για την διατήρηση εγγραφών που έχουν να κάνουν τόσο με τους χρήστες της εφαρμογής όσο και με τα ρομπότ τα οποία συνδέονται, για λόγους ταυτοποίησης (authentication) και εξουσιοδότησης (authorization). Για την ορθή λειτουργία του συστήματος απαραίτητη προϋπόθεση αποτελεί η διατήρηση αρχείου τόσο με τα στοιχεία των χρηστών όσο και με τα στοιχεία των συσκευών που συνδέονται σε αυτή. Η αλληλεπίδραση της βάσης δεδομένων με τις υπόλοιπες λειτουργίες του συστήματος υλοποιείται μέσω ενός REST API (βλέπε υποενότητα 2.6.6) με την χρήση του οποίου ανανεώνονται οι τιμές των εγγραφών της βάσης.

Ένα πολύ σημαντικό κομμάτι του συστήματος αποτελείται από τον message broker RabbitMQ, ο οποίος είναι υπεύθυνος για την επικοινωνία και αλληλεπίδραση των συσκευών που είναι συνδεδεμένες με την εφαρμογή. Ως message broker το RabbitMQ επιτρέπει την ανταλλαγή μηνυμάτων μεταξύ εφαρμογών που χρησιμοποιούν διαφορετικά πρωτόκολλα επικοινωνίας και ως εκ τούτου αν συνδυαστεί με το μηχανισμό γεφύρωσης με το ROS δημιουργείται ένα εργαλείο το

οποίο επιτρέπει την ανάπτυξη εφαρμογών οι οποίες θα επικοινωνούν απευθείας, ανεξαρτήτως πρωτοκόλλου, με το robot.

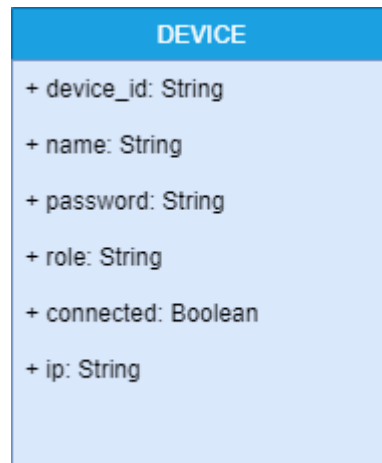
Ο πυρήνας του συστήματος αποτελείται από τις συσκευές οι οποίες συνδέονται στον message broker RabbitMQ ώστε να υλοποιηθεί η επικοινωνία, η ανταλλαγή πληροφοριών και μηνυμάτων αλλά και οι ταυτόχρονες δράσεις μεταξύ τους. Το ROS αποτελεί το πιο ευρέως διαδεδομένο μεσολογισμικό για την ανάπτυξη λογισμικού ρομποτικών συστημάτων. Ωστόσο όπως έχει αναλυθεί σε προηγούμενα κεφάλαια η χρήση του ROS περιορίζεται σημαντικά όταν η επικοινωνία και ο έλεγχος των ρομπότ γίνεται μέσω του διαδικτύου. Η λύση σε αυτό τον περιορισμό, η οποία προτείνεται στην παρούσα διπλωματική είναι η δημιουργία γέφυρας επικοινωνίας μεταξύ των διεπαφών του ROS και του RabbitMQ. Κατ' αυτό τον τρόπο διευκολύνεται η δημιουργία ρομποτικών εφαρμογών οι οποίες μέσω του RabbitMQ μπορούν να επικοινωνούν απευθείας με τις ρομποτικές συσκευές.

Τέλος η αλληλεπίδραση του χρήστη με το υπόλοιπο σύστημα υλοποιείται μέσω της διαδικτυακής εφαρμογής η οποία είναι υπεύθυνη για την εκτέλεση όλων επιμέρους λειτουργιών του συστήματος, διαθέτοντας ένα εύχρηστο και φιλικό γραφικό προς το χρήστη.

## 4.2 Βάση Δεδομένων

Η βάση δεδομένων που χρησιμοποιήθηκε είναι η MongoDB η οποία είναι τύπου NoSQL και επιτρέπει τη δυναμική δημιουργία αρχείων με μορφή τύπου JSON τα οποία αποτελούν τα στοιχεία της. Για την λειτουργία του συστήματος απαραίτητή είναι η δημιουργία δύο πινάκων εγγραφών, ένας ο οποίος να διαχειρίζεται τους χρήστες οι οποίοι αλληλεπιδρούν με την εφαρμογή και ένας ο οποίος διατηρεί τα χαρακτηριστικά των διαφόρων συσκευών οι οποίες χρησιμοποιούνται κατά την εκτέλεση του συστήματος.

Το μοντέλο των συσκευών παρουσιάζεται στην εικόνα 16 όπου φαίνονται οι διάφορες μεταβλητές.



Εικόνα 16: Μοντέλο συσκευής της βάσης δεδομένων

Πιο συγκεκριμένα οι ιδιότητες (πεδία ορισμού) του μοντέλου των συσκευών (DEVICE) είναι:

- **device\_id**: Αποτελεί τον μοναδικό αριθμό για την κάθε συσκευή και γενικά το κλειδί του πίνακα των συσκευών, γεγονός που ξεχωρίζει την κάθε συσκευή από την άλλη.
- **name**: Περιέχει το όνομα της κάθε συσκευής και είναι και αυτό μοναδικό.
- **Password**: Περιέχει τον κωδικό χρήσης της εκάστοτε συσκευής.
- **role**: Περιέχει τον ρόλο που έχει η κάθε συσκευή στην εφαρμογή.
- **ip**: Η διεύθυνση ip στην οποία τρέχει η κάθε συσκευή.
- **connected**: Μεταβλητή η οποία δείχνει αν μια συσκευή είναι συνδεδεμένη η όχι.

Από την άλλη πλευρά το μοντέλο του χρήστη, περιέχει τις παρακάτω μεταβλητές (εικόνα 17):

- **user\_id**: Περιέχει τον μοναδικό αριθμό κάθε χρήστη.
- **username**: Περιέχει το όνομα σύνδεσης του κάθε χρήστη και είναι μοναδικό για κάθε εγγραφή
- **password**: Περιέχει τον κωδικό σύνδεσης του κάθε χρήστη.
- **fullname**: Το όνομα και επίθετο του εκάστοτε χρήστη.
- **devices**: Οι συσκευές τις οποίες διαθέτει και μπορεί να διαχειριστεί ο κάθε χρήστης. Τα στοιχεία αυτής της μεταβλητής κάνουν αναφορά σε εγγραφές του πίνακα Device.

USER
+ user_id: String
+ username: String
+ password: String
+ fullname: String
+ devices: Device

Εικόνα 17: Πίνακας χρηστών της βάσης δεδομένων

### 4.3 REST API

Η αλληλεπίδραση της εφαρμογής με την βάση δεδομένων η οποία διαχειρίζεται τους χρήστες αλλά και τις εγγραφές για τα ρομπότ τα οποία συμμετέχουν στην εφαρμογή επιτυγχάνεται με την υλοποίηση ενός REST API. Για να δημιουργηθεί αυτή η υπηρεσία το βασικό σχέδιο της υλοποιήθηκε με εργαλεία Swagger τα οποία δίνουν τη δυνατότητα στον χρήστη, με απλές περιγραφές σε γλώσσα JSON/yaml, να δημιουργήσουν τις απαραίτητες υπηρεσίες του API, χρησιμοποιώντας προδιαγραφές Open-API για τη σχεδίαση REST(ful) αρχιτεκτονικών.

Παρακάτω παρουσιάζονται λεπτομερώς οι διάφορες υπηρεσίες του API όπως προκύπτουν από τις προδιαγραφές του συστήματος.

- **/register/user: [POST]** Εγγράφει έναν χρήστη στην βάση δεδομένων.
- **/register/device: [POST]** Εγγράφει μια συσκευή στην βάση δεδομένων.
- **/devices/connected: [GET]** Επιστρέφει μια λίστα με τις συσκευές οι οποίες είναι συνδεδεμένες.
- **/devices: [GET]** Επιστρέφει μια λίστα με όλες τις συσκευές που έχουν εγγραφεί
- **/devices/details/{device\_name}: [GET]** Επιστρέφει τις λεπτομέρειες κάθε συσκευής, εκτός από τον κωδικό πρόσβασης.
- **/devices/connect/{device\_name}: [POST]** Θέτει μια συσκευή ως συνδεδεμένη.
- **/user/register\_device: [POST]** Προσθέτει στη λίστα συσκευών κάθε χρήστη μια νέα συσκευή.
- **/user/devices: [GET]** Επιστρέφει λίστα με τις συσκευές που έχουν αντιστοιχηθεί σε κάθε χρήστη.

Οι επόμενες υπηρεσίες συνδέονται με τις μεθόδους αυθεντικοποίησης που χρησιμοποιεί ο message broker RabbitMQ, διαδικασία η οποία αναλύεται λεπτομερώς στην ενότητα 5.6 .

- **/auth/user:[GET]** Χρησιμεύει στην αυθεντικοποίηση των χρηστών και συσκευών.
- **/auth/vhost: [GET]:** Χρησιμεύει για την αυθεντικοποίηση των virtual host του RabbitMQ.
- **/auth/topic: [GET]:** είναι υπεύθυνη για την αυθεντικοποίηση των topic του RabbitMQ.
- **/auth/resource: [GET]:** Χρησιμεύει για την αυθεντικοποίηση των resources του RabbitMQ.

#### 4.4 Σύνδεση με το RabbitMQ

Το RabbitMQ αποτελεί τον διαμεσολαβητή μηνυμάτων, ο οποίος υποστηρίζει ένα πλήθος πρωτοκόλλων, δίνοντας την δυνατότητα σύνδεσης σε απομακρυσμένες συσκευές προκειμένου να επιτευχθεί η επικοινωνία και η ανταλλαγή πληροφοριών μεταξύ τους. Για να συνδεθεί μια συσκευή στον message broker είναι απαραίτητος ο καθορισμός των παραμέτρων σύνδεσης οι οποίες παρουσιάζονται αναλυτικά παρακάτω.

Συγκεκριμένα κάθε ρομπότ το οποίο επιθυμεί να αλληλεπιδράσει με το RabbitMQ πρέπει να ορίσει τις παρακάτω παραμέτρους σύνδεσης:

- **host:** Καθορίζεται η διεύθυνση Ip και το hostname στο οποίο τρέχει το RabbitMQ. Για την εκτέλεση τοπικής προσομοίωσης μπορούμε να δώσουμε την τιμή «localhost» στην μεταβλητή.
- **port:** Ο αριθμός της θύρας στον οποίο «ακούει» ο message broker, όπου για την περίπτωση της τοπικής σύνδεσης είναι προκαθορισμένος στην τιμή «5672».
- **vhost (virtual host):** Αποτελεί έναν τρόπο διαχωρισμού των διαφόρων συσκευών που χρησιμοποιούν τον ίδιο message broker. Ουσιαστικά διαφορετικές συσκευές μπορούν να συνδεθούν στον ίδιο broker έχοντας όμως πρόσβαση σε συγκεκριμένα κομμάτια αυτής.
- **username:** Το όνομα της συσκευής που συνδέεται στο RabbitMQ.
- **password:** Ο κωδικός της συσκευής που συνδέεται στον broker ο οποίος σε συνδυασμό με το username δημιουργεί την συνθήκη αυθεντικοποίησης της συσκευής για σύνδεση στο RabbitMQ, διαδικασία η οποία θα αναλυθεί λεπτομερώς σε επόμενο κεφάλαιο.

#### 4.5 Πιστοποίηση Χρηστών και Συσκευών

Για την ασφαλή λειτουργία της εφαρμογής είναι απαραίτητη η πιστοποίηση τόσο των συσκευών όσο και των χρηστών που συνδέονται σε αυτή. Κατά την διαδικασία εγγραφής τους στην βάση δεδομένων τύπου mongo, που αναλύθηκε σε προηγούμενη ενότητα, εισάγεται ένα όνομα και ένας κωδικός χρήσης τόσο στην περίπτωση των συσκευών όσο και των χρηστών. Ο κωδικός αυτός αποτελεί το κλειδί



κρυπτογράφησης για την δημιουργία ενός token, το οποίο θα χρησιμοποιηθεί έπειτα για την αυθεντικοποίηση.

Το RabbitMQ υποστηρίζει αρκετούς μηχανισμούς με τους οποίους επιτυγχάνεται η πιστοποίηση. Στην παρούσα διπλωματική εργασία χρησιμοποιείται ο μηχανισμός PLAIN SASL (Simple authentication and security layer). Το SASL είναι ένα framework το οποίο χρησιμοποιείται για ασφάλεια δεδομένων στα διάφορα διαδικτυακά πρωτόκολλα. Το PLAIN SASL είναι ένας μηχανισμός του SASL, ο οποίος επιτρέπει την πιστοποίηση με την χρήση username/password.

Για την επίτευξη της πιστοποίησης το RabbitMQ διαθέτει μια σειρά από διάφορα backends τα οποία μπορούν να ενεργοποιηθούν μέσω plugins και εκτελούνται ώστε να εξεταστεί αν θα επιτραπεί η σύνδεση ή όχι στον broker. Για την ανάπτυξη του μηχανισμού πιστοποίησης χρησιμοποιήθηκε το HTTP backend του RabbitMQ, το οποίο δηλώνεται στο αρχείο διαχείρισης του RabbitMQ server με τρόπο ο οποίος φαίνεται στην εικόνα 18. Αυτό επιτρέπει στο RabbitMQ να πιστοποιεί τους χρήστες και τις συσκευές που θέλουν να συνδεθούν σ αυτό κάνοντας HTTP requests σ έναν server. Τέλος, υπάρχει και η δυνατότητα χρησιμοποίησης παραπάνω του ενός backend για να υλοποιηθεί η πιστοποίηση.

```
auth_backends.1= http
auth_http.http_method    = get
auth_http.user_path       = http://127.0.0.1:8080/auth/user
auth_http.vhost_path      = http://127.0.0.1:8080/auth/vhost
auth_http.resource_path   = http://127.0.0.1:8080/auth/resource
auth_http.topic_path      = http://127.0.0.1:8080/auth/topic
```

Εικόνα 18:Αρχείο του RabbitMQ server το οποίο τον τρόπο αυθεντικοποίησης

Για την πιστοποίηση των χρηστών και συσκευών το backend του RabbitMQ στέλνει requests στο endpoint «/auth/user» του API το οποίο έχει αναπτυχθεί για την αλληλεπίδραση της βάσης δεδομένων με την υπόλοιπη εφαρμογή. Το plugin ορίζει το προκαθορισμένο πλαίσιο με το οποίο το API πρέπει να απαντάει στα requests του RabbitMQ. Για να συνδεθεί λοιπόν ένας χρήστης ή μια συσκευή στο RabbitMQ, παρέχει το username και το password το οποίο χρησιμοποιείται από τον message broker ώστε να στείλει requests στο url «/auth/user» του API. Στη συνέχεια ο κώδικας του url αναγνωρίζει αν το username και το password αφορούν συσκευή ή χρήστη, και διασταυρώνει το token το οποίο δημιουργείται με κλειδί το password, με αυτό που υπάρχει στην βάση δεδομένων και απαντά στο request. Η απάντηση είναι αναγκαίο να έχει συγκεκριμένη μορφή, το API δηλαδή είναι απαραίτητο να επιστρέφει ένα HTTP response με status code 200 OK, και σώμα μηνύματος «allow» σε περίπτωση που πρέπει να επιτραπεί η είσοδος στην χρήστη και «deny» σε αντίθετη περίπτωση.

Τέλος τα URIs των vhost, resources και topic\_path επιστρέφουν ένα μήνυμα «allow» με status code 200 OK σε κάθε περίπτωση, καθώς αποτελούν προαπαιτούμενο για την πιστοποίηση μέσω HTTP backend του RabbitMQ και δεν χρησιμοποιούνται από την εφαρμογή για κάποιου είδους περαιτέρω αυθεντικοποίησης.

Συμπερασματικά, αναπτύχθηκε ένας μηχανισμός πιστοποίησης των συσκευών ο οποίος αν συνδυαστεί με τον message broker δημιουργείται μια ολοκληρωμένη πλατφόρμα μέσω της οποίας οι χρήστες έχουν την δυνατότητα να εγγράφουν τις συσκευές και έπειτα να αλληλεπιδρούν με τους πόρους (resources) της κάθε συσκευής.

#### 4.6 Απομακρυσμένος Έλεγχος Συσκευών

Μια από τις απαραίτητες και πλέον καθοριστικές λειτουργίες του συστήματος είναι η δυνατότητα που δίνεται στον χρήστη να μπορεί να βλέπει τις συσκευές που έχουν συνδεθεί στο RabbitMQ γεγονός απαραίτητο για την συνολικά ορθή της λειτουργία. Διευρύνοντας αυτή την δυνατότητα κρίθηκε αναγκαίο ο χρήστης να μπορεί να βλέπει τις διεργασίες που τρέχουν σε κάθε συσκευή καθώς και διάφορα στατιστικά στοιχεία που έχουν να κάνουν με την χρησιμοποίηση του συστήματος και των διαφόρων αισθητήρων (Device monitoring Agent). Έτσι δίνεται η δυνατότητα στον χρήστη να επιλέγει οποιαδήποτε στιγμή ανάλογα με τις δυνατότητες του συστήματος την παύση της σύνδεσης της συσκευής με την πλατφόρμα γεγονός που στοχεύει στην βέλτιστη χρήση της εφαρμογής.

Για την υλοποίηση του Device Monitoring Agent χρησιμοποιήθηκε η βιβλιοθήκη της psutil<sup>54</sup> (process and system utilities). Είναι μια βιβλιοθήκη η οποία υποστηρίζεται από πολλά λειτουργικά συστήματα και είναι χρήσιμη κυρίως για τον έλεγχο του συστήματος, τους πόρους που καταναλώνονται από τις διάφορες διεργασίες που τρέχουν στις συσκευές, τις απαιτήσεις σε μνήμη και υπολογιστική ισχύ καθώς και στα χαρακτηριστικά ορισμένων αισθητήρων.

Στην περίπτωση του Device Monitoring Agent τα στοιχεία τα οποία συλλέγονται για κάθε συσκευή ώστε να είναι διαθέσιμα στην κρίση του κάθε χρήστη είναι:

- **Χρήση CPU:** Μπορούν να αντληθούν πληροφορίες που αφορούν τον αριθμό των πυρήνων, την συχνότητα (frequency) στην οποία λειτουργεί ο επεξεργαστής καθώς και το ποσοστό χρήσης του επεξεργαστή.
- **Χρήση μνήμης:** Δίνει την δυνατότητα στο χρήστη να γνωρίζει το ποσοστό της μνήμης το οποίο χρησιμοποιείται από την συσκευή καθώς και διάφορες άλλες παραμέτρους.

---

<sup>54</sup> <https://pypi.org/project/psutil>

- **Χρήση δίσκου:** Εμπεριέχει πληροφορίες σχετικά με την χρήση των διαφόρων δίσκων της κάθε συσκευής.
- **Συνδέσεις δικτύου:** Αφορά τις συνδέσεις δικτύου κάθε συσκευής, tcp, udp αλλά και unix.
- **Θερμοκρασία υλικού:** Επιστρέφει την hardware θερμοκρασία κάθε συσκευής η οποία μπορεί να αναφέρεται στην cpu, στον σκληρό δίσκο, η οτιδήποτε άλλο ορίζεται από το εκάστοτε λειτουργικό σύστημα.
- **Μπαταρία συσκευής:** Περιέχει πληροφορίες που αφορούν την κατάσταση μπαταρίας της συσκευής η οποία χρησιμοποιείται.
- **Εκτελούμενες διεργασίες:** Επιστρέφει πληροφορίες που αφορούν τις διεργασίες που εκτελούνται σε κάθε συσκευή καθώς και διαφόρους τρόπους κατηγοριοποίησής τους όπως αυτές που απαιτούν περισσότερη υπολογιστική ισχύ ή περισσότερη μνήμη.

Για να επιτευχθεί αυτή λειτουργία και να γίνουν διαθέσιμα στον χρήστη στοιχεία τα οποία αφορούν την χρήση και την λειτουργία της κάθε συσκευής επιλέχθηκε η μέθοδος επικοινωνίας pub/sub μεταξύ της συσκευής και του message broker RabbitMQ. Κατ' αυτή την διαδικασία δημιουργείται ένας publisher ο οποίος δημοσιεύει όλες τις πληροφορίες της εκάστοτε συσκευής σε ένα topic, το όνομα του οποίου καθορίζεται βάσει της ονομασίας της συσκευής. Έτσι από την αντίθετη πλευρά σε επίπεδο RabbitMQ δημιουργείται ένας subscriber ο οποίος κάνοντας εγγραφή στο topic το οποίο αφορά την συσκευή από την οποία θέλει να αντλήσει πληροφορίες έχει πλέον διαθέσιμες όλες τις πληροφορίες που απαιτούνται για την σωστή διαχείριση της κάθε συσκευής. Αυτή η διαδικασία αποτελεί μια εφαρμογή η οποία επιτρέπει ουσιαστικά την απομακρυσμένη διαχείριση συσκευών, λειτουργία η οποία είναι απαραίτητη σε επίπεδο Internet of Things και βασικός πυλώνας του συνολικού προγράμματος το οποίο αναπτύχθηκε.

#### 4.7 ROS Environment

Όπως έχει αναλυθεί σε προηγούμενα κεφάλαια ένας από τους στόχους της παρούσας διπλωματικής εργασίας είναι η απομακρυσμένη διασύνδεση και επικοινωνία μεταξύ των ρομπότ. Το Robot Operating System αποτελεί τον πιο ευρέως διαδεδομένο τρόπο ανάπτυξης ρομποτικών συστημάτων εξασφαλίζοντας ένα πλαίσιο λειτουργίας ιδιαίτερα χρηστικό. Ωστόσο μια ιδιαιτερότητα του ROS, η οποία περιορίζει αισθητά την εφαρμογή του είναι ο τοπικός του χαρακτήρας και οι περιορισμοί που εμφανίζονται όταν η διαχείριση των ρομπότ γίνεται απομακρυσμένα μέσω του διαδικτύου. Το κενό αυτό έρχεται να καλυφθεί από την παρούσα εφαρμογή η οποία αποβλέπει στην γεφύρωση του ROS με τον RabbitMQ broker.

Το πρώτο βήμα για να επιτευχθεί η γεφύρωση του ROS με το RabbitMQ είναι ο έλεγχος του ROS περιβάλλοντος που τρέχει στην εκάστοτε συσκευή. Όπως έχει

αναλυθεί σε προηγούμενο κεφάλαιο τα κύρια δομικά μέρη του ROS βάσει των οποίων γίνεται η επικοινωνία μεταξύ των ρομπότ είναι τα topics, messages, services και nodes. Σύμφωνα με αυτό ένας κόμβος (node) μπορεί να κάνει publish ένα message σ ένα topic ένας άλλος κόμβος μπορεί να κάνει subscribe στο topic αυτό λαμβάνοντας τα δεδομένα του μηνύματος η οι δύο κόμβοι να επικοινωνούν μεταξύ τους μέσω services τα οποία προσομοιώνουν την διαδικασία remote procedure call (RPC).

Για να μπορέσει να πραγματοποιηθεί η γεφύρωση των διεπαφών του ROS με το RabbitMQ ο χρήσης είναι απαραίτητο να γνωρίζει τα topics, services και nodes που «τρέχουν» στο ρομπότ ώστε να επιλέξει ποια από αυτά θέλει να γεφυρώσει με τον message broker. Η παραπάνω διαδικασία επιτυγχάνεται με την χρήση ενός script σε επίπεδο συσκευής, το οποίο χρησιμοποιεί τις βιβλιοθήκες της Python rosservice, rostopic, rosnodes και την μέθοδο Client της rospy, ώστε να επιστρέφει τα ROS topics, services, nodes τα οποία είναι κάθε στιγμή ενεργά και «τρέχουν» στο εκάστοτε ρομπότ.

Το επόμενο βήμα, το οποίο είναι απαραίτητο ώστε ο χρήστης συνδέοντας το ρομπότ στη πλατφόρμα να έχει την δυνατότητα να αντλήσει πληροφορίες σχετικά με το ROS πλαίσιο το οποίο τρέχει στην συσκευή, είναι η αποστολή των πληροφοριών στο RabbitMQ. Τα ROS δεδομένα λοιπόν συλλέγονται από το ρομπότ και γίνονται publish σε τρία διαφορετικά topics το όνομα των οποίων καθορίζεται από το όνομα της συσκευής και το είδος των δεδομένων που αυτά αφορούν, topics, services η nodes. Κατά την αντίστροφη διαδικασία λοιπόν ο χρήστης συνδέεται στον message broker, επιλέγει για ποια συσκευή θέλει να δει τα ROS δεδομένα, παρέχοντας το όνομα της, δημιουργείται ένας subscriber στο αντίστοιχο topic, ο οποίος εμφανίζει τις πληροφορίες στον χρήστη. Μέσω αυτής της διαδικασίας δίνεται η δυνατότητα στον χρήστη απομακρυσμένα να έχει επίγνωση για το ROS περιβάλλον στο οποίο τρέχει η εκάστοτε συσκευή, έτσι ώστε σε επόμενο βήμα της εφαρμογής να μπορέσει να επιλέξει ποια από τα ROS κομμάτια θα γεφυρώσει με το RabbitMQ.

#### 4.8 Γεφύρωση Περιβάλλοντος ROS με την Απομακρυσμένη Πλατφόρμα

Η ενότητα αυτή αποτελεί τον πυρήνα της διπλωματικής εργασίας καθώς παρουσιάζει τον τρόπο με τον οποίο γίνεται ουσιαστικά η γεφύρωση του Robot Operating System με τον message broker RabbitMQ. Πλέον τα ρομπότ ανταλλάσσουν πληροφορίες και αλληλεπιδρούν μέσω του διαδικτύου και συγκεκριμένα μέσω του RabbitMQ γεγονός που προσομοιώνει την λειτουργία των έξυπνων συσκευών σε επίπεδο Internet of Things. Με την χρήση της εφαρμογής δημιουργείται η ευκαιρία της απομακρυσμένης διαχείρισης των συσκευών από τον χρήστη, δυνατότητα που αποτελεί ένα πρώτο βήμα για την ενσωμάτωση των ρομποτικών εφαρμογών σε δίκτυο IoT. Η χρήση του ROS, όπως έχουμε περιγράψει αναλυτικά, περιορίζεται όταν η διαχείριση των συσκευών δεν γίνεται τοπικά αλλά μέσω του διαδικτύου βάζοντας

αρκετούς περιορισμούς στην ανάπτυξη ρομποτικών εφαρμογών όπου βάση τους αποτελεί το διαδίκτυο. Για να επιτευχθεί η γεφύρωση των διεπαφών του ROS με το IoT αναπτύχθηκε ένα σύνολο διεργασιών οι οποίες παρουσιάζονται αναλυτικά στις επόμενες ενότητες.

#### 4.8.1 Μετατροπή ROS Μηνυμάτων και Υπηρεσιών

Όπως έχει αναλυθεί η επικοινωνία των συσκευών με την χρήση του ROS επιτυγχάνεται με δύο βασικούς τρόπους, μέσω της διαδικασίας pub/sub και μέσω των ROS services. Στην περίπτωση publish subscribe ένας κόμβος κάνει publish ένα μήνυμα σε ένα συγκεκριμένο topic και ένας άλλος κόμβος κάνει subscribe στο topic αυτό προκειμένου να λάβει τα δεδομένα του μηνύματος ώστε να προχωρήσει στην εξεργασία τους. Η δεύτερη μέθοδος επικοινωνίας πραγματοποιείται μέσω services, όπου ένας κόμβος κάνει request σε έναν άλλο κόμβο και περιμένει την απάντηση (response), τρόπος ο οποίος ουσιαστικά προσομοιώνει την φιλοσοφία remote procedure call (RPC).

Στην περίπτωση Pub-Sub η επικοινωνία μεταξύ των κόμβων γίνεται με την ανταλλαγή ROS μηνυμάτων. Το ROS χρησιμοποιεί απλοποιημένη γλώσσα περιγραφής μηνυμάτων για την περιγραφή των τιμών που δημοσιεύουν οι κόμβοι του. Αυτή η περιγραφή διευκολύνει τα εργαλεία ROS να δημιουργούν αυτόματα τον πηγαίο κώδικα για τον τύπο του μηνύματος σε πολλές γλώσσες προορισμού. Έτσι λοιπόν για την επικοινωνία μεταξύ κόμβων απαραίτητος είναι ο προσδιορισμός του είδους του μηνύματος το οποίο ανταλλάσσεται μεταξύ publisher και subscriber. Από την άλλη πλευρά και στην περίπτωση της επικοινωνίας με την μορφή ROS services, το ROS χρησιμοποιεί απλοποιημένη γλώσσα για να περιγράψει ένα είδος service, φιλοσοφία η οποία πρόκειται στο πλαίσιο επικοινωνίας μεταξύ μηνυμάτων. Με απλά λόγια λοιπόν το ROS έχει αναπτύξει τον δικό του τρόπο επικοινωνίας με την χρήση .msg αρχείων, στην περίπτωση του pub/sub και .srv αρχείων, στην περίπτωση των services, για την αλληλεπίδραση των κόμβων.

Σ αυτό το σημείο λοιπόν εμφανίζεται ένα πρόβλημα το οποίο αφορά της γεφύρωση της επικοινωνίας του ROS με το RabbitMQ. Το ROS, όπως αναλύθηκε έχει αναπτύξει τον δικό του τρόπο με το οποίο επιτυγχάνεται η ανταλλαγή δεδομένων, ο οποίος είναι αναγνωρίσιμος μόνο από αυτό. Επομένως τα δεδομένα τα οποία ανταλλάσσονται μεταξύ των συσκευών στο επίπεδο του ROS, δεν μπορούν να αναγνωριστούν και κατ' επέκταση να χρησιμοποιηθούν από τον message broker RabbitMQ. Για αυτό τον λόγο χρησιμοποιήθηκε η βιβλιοθήκη `ros_conversions_py`<sup>55</sup> η οποία μετατρέπει τόσο τα ROS μηνύματα όσο και τα ROS responses και requests, στην περίπτωση των services, από την μορφή που είναι αποδεκτή μόνο από το ROS, σε

---

<sup>55</sup> [https://github.com/robotics-4-all/rosconversions\\_py](https://github.com/robotics-4-all/rosconversions_py)

python λεξικά (dictionaries) έτσι ώστε να μπορεί να επιτευχθεί η ανταλλαγή δεδομένων και σε επίπεδο RabbitMQ.

Πιο συγκεκριμένα, ο μηχανισμός μετατροπής των ROS μηνυμάτων και service response, request σε λεξικά περιλαμβάνει τις παρακάτω μεθόδους:

- **ros\_msg\_to\_dict (message):** Μετατρέπει το ROS μήνυμα σε λεξικό ώστε να είναι διαχειρίσιμα τα δεδομένα του από τις μεθόδους επικοινωνίας του message broker.
- **dict\_to\_ros\_msg (dict, message\_type):** Μετατρέπει το λεξικό σε ROS μήνυμα, έχοντας και ως παράμετρο το είδος του μηνύματος για να μπορεί να επιτευχθεί η γεφύρωση του RabbitMQ με το ROS.
- **ros\_srv\_req\_to\_dict (srv\_req):** Μετατρέπει τα ROS service requests σε λεξικά τα οποία μπορούν να συμμετέχουν στην επικοινωνία του message broker.
- **ros\_srv\_resp\_to\_dict(srv\_req):** Μετατρέπει τα ROS service responses σε λεξικά διαχειρίσιμα από το RabbitMQ.
- **dict\_to\_ros\_srv\_request (service\_type, dictionary):** Μετατρέπει τα λεξικά τα οποία αφορούν ROS service requests σε μορφή αναγνωρίσιμη από το ROS.
- **dict\_to\_ros\_srv\_response (service\_type, dictionary):** Μετατρέπει τα λεξικά τα οποία αφορούν ROS service responses σε μορφή αναγνωρίσιμη από το ROS.

#### 4.8.2 Bridging templates

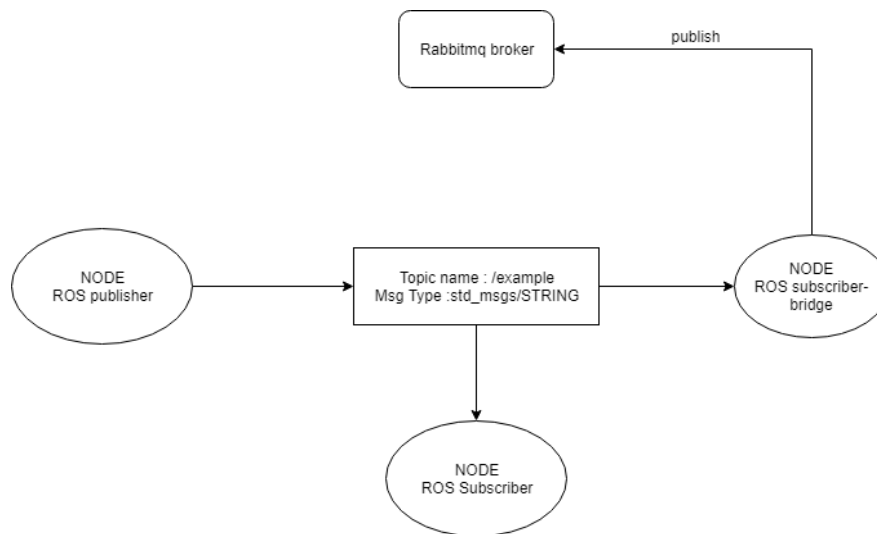
Στην ενότητα αυτή παρουσιάζεται ο τρόπος με την οποίο επιτυγχάνεται η γεφύρωση της λειτουργίας και των διεπαφών του ROS με τον message broker RabbitMQ έτσι ώστε η διαχείριση και η επικοινωνία μεταξύ των ρομποτικών συσκευών να γίνεται πλέον απομακρυσμένα, μέσω του διαδικτύου. Αυτή η φιλοσοφία επικοινωνίας αποτελεί και το πλαίσιο λειτουργίας του Internet of Things, όπου το ρόλο των έξυπνων συσκευών θα έχουν πλέον τα ρομπότ. Για να πραγματοποιηθεί η δημιουργία γεφυρών επικοινωνίας δημιουργήθηκαν templates οι οποίες διαμορφώνονται ανάλογα με το μοντέλο επικοινωνίας το οποίο επιθυμούμε να συνδέσουμε, είτε αυτό αφορά το μοντέλο publish subscribe, είτε το μοντέλο επικοινωνίας μέσω services. Ουσιαστικά η «γέφυρα» είναι ένα template το οποίο διαμορφώνεται ανάλογα με την περίπτωση και εκτελείται ώστε να επιτευχθεί η σύνδεση του RabbitMQ και του ROS. Στη συνέχεια αναλύονται λεπτομερώς τα είδη των διαφόρων templates καθώς και ο τρόπος με τον οποίο συνδυάζονται. Τέλος για την ανάπτυξη των templates χρησιμοποιήθηκε η βιβλιοθήκη της python jinja2, η οποία είναι και η πιο ευρέως διαδιδόμενη μηχανή παραγωγής template.



#### 4.8.2.1 Μοντέλο Publish/Subscribe

Η πρώτη περίπτωση γεφύρωσης του ROS με το RabbitMQ αφορά την δημιουργία μηχανισμού σύνδεσης του μοντέλου επικοινωνίας publish subscribe. Όπως έχουμε αναλύσει σε προηγούμενες ενότητες ο κυριότερος τρόπος επικοινωνίας των διαφόρων κόμβων του ROS επιτυγχάνεται μέσω της διαδικασίας publish και subscribe, διαδικασία η οποία ορίζεται από το topic το οποίο καθορίζει και το πλαίσιο κάτω από το οποίο θα επιτευχθεί η επικοινωνία. Από την άλλη πλευρά και στην περίπτωση του message broker, αναπτύχθηκε το μοντέλο επικοινωνίας pub/sub το οποίο χρησιμοποιεί το πρωτόκολλο επικοινωνίας AMQP, για την επίτευξη της επικοινωνίας μεταξύ των συσκευών. Απομένει λοιπόν να εξηγηθεί ο τρόπος με τον οποίο τα δύο μοντέλα επικοινωνίας γεφυρώνονται έτσι ώστε οι ρομποτικές συσκευές να επικοινωνούν μέσω του RabbitMQ broker.

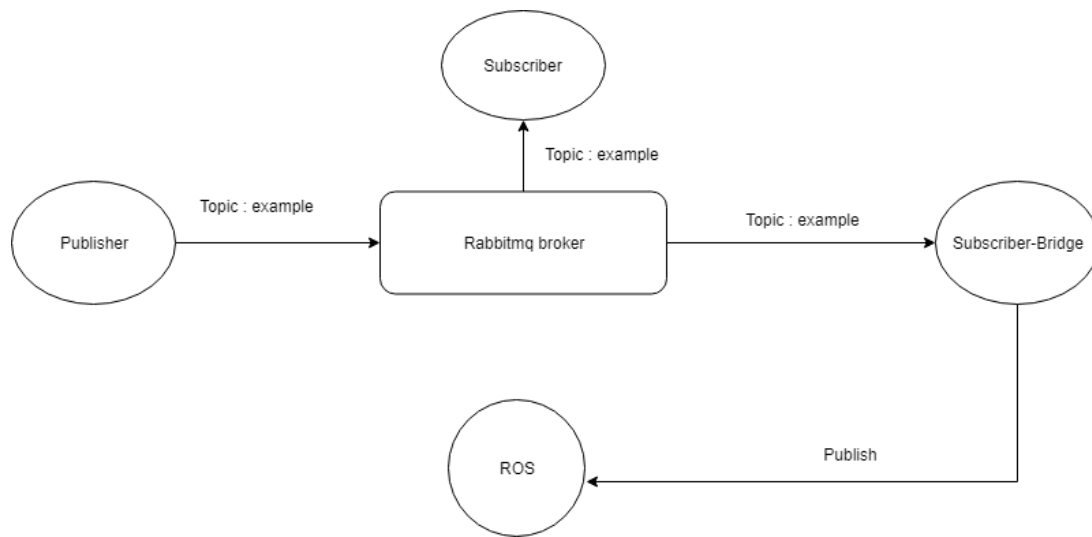
Αρχικά, η πρώτη κατηγορία γεφύρωσης είναι η περίπτωση ενός ROS κόμβου ο οποίος κάνει publish δεδομένα σε ένα ROS topic. Στόχος είναι η δημιουργία ενός μηχανισμού ο οποίος θα κάνει publish τα ίδια δεδομένα σε ένα αντίστοιχο topic στην πλευρά του message broker. Αυτή η διαδικασία μπορεί να επιτευχθεί με την δημιουργία ενός είδους subscriber template το οποίο είναι υπεύθυνο για την γεφύρωση του ROS publisher με το RabbitMQ. Σύμφωνα με αυτό για κάθε ROS publisher που θέλουμε να συνδέσουμε με τον message broker δημιουργείται ένας ROS subscriber ο οποίος είναι υπεύθυνος να λαμβάνει τα δεδομένα, να μετατρέπει τα ROS μηνύματα σε λεξικά τα οποία είναι αναγνωρίσιμα από την πλατφόρμα και να τα κάνει publish σε ένα αντίστοιχο topic του message broker (εικόνα 19). Η διαδικασία λοιπόν η οποία ακολουθείται για την γεφύρωση ενός ROS publisher είναι, η κατάλληλη διαμόρφωση του template με τα δεδομένα του συγκεκριμένου publisher που θέλουμε να γεφυρώσουμε και η δημιουργία ενός εκτελέσιμου αρχείου το οποίο κάθε φορά που ο ROS publisher στέλνει δεδομένα, τα ίδια δεδομένα θα δημοσιεύονται και στον message broker σε αντίστοιχο topic. Με αυτόν τον τρόπο επιτυγχάνεται η γεφύρωση της επικοινωνίας μεταξύ ROS και RabbitMQ στο κομμάτι που αφορά τους ROS publishers.



Εικόνα 19: Διαδικασία η οποία ακολουθείται για την γεφύρωση ενός ROS Publisher

Κατά την αντίθετη διαδικασία είναι αναγκαίο να υπάρχει γεφύρωση και των broker publishers, δηλαδή των συσκευών που δημοσιεύουν δεδομένα στον broker RabbitMQ προς το ROS. Ομοίως με την παραπάνω διαδικασία, για κάθε topic το οποίο υφίσταται σε επίπεδο RabbitMQ και θέλουμε να το γεφυρώσουμε με το ROS χρησιμοποιείται μια νέα μορφή template η οποία κάνει subscribe στο topic το οποίο επιθυμεί ο χρήστης να συνδέσει με το ROS, λαμβάνει τα δεδομένα που ανταλλάσσονται μέσω του πρωτοκόλλου AMPQ, τα μετατρέπει από την μορφή των λεξικών σε μορφή ROS μηνυμάτων τα οποία είναι αναγνωρίσιμα από το ROS και τα κάνει publish στο αντίστοιχο ROS topic (εικόνα 20). Μέσω της εισαγωγής στη γενική μορφή του broker template, των παραμέτρων που αφορούν το συγκεκριμένο topic δημιουργείται ένα εκτελέσιμο αρχείο το οποίο κάθε φορά που ο broker publisher στέλνει δεδομένα σε ένα topic, τα ίδια δεδομένα δημοσιεύονται και σε επίπεδο ROS. Με την δημιουργία αυτού του μηχανισμού ουσιαστικά έχει δημιουργηθεί όλο το πλαίσιο το οποίο επιτρέπει την γεφύρωση της επικοινωνίας του RabbitMQ και του ROS, για το μοντέλο publish subscribe.





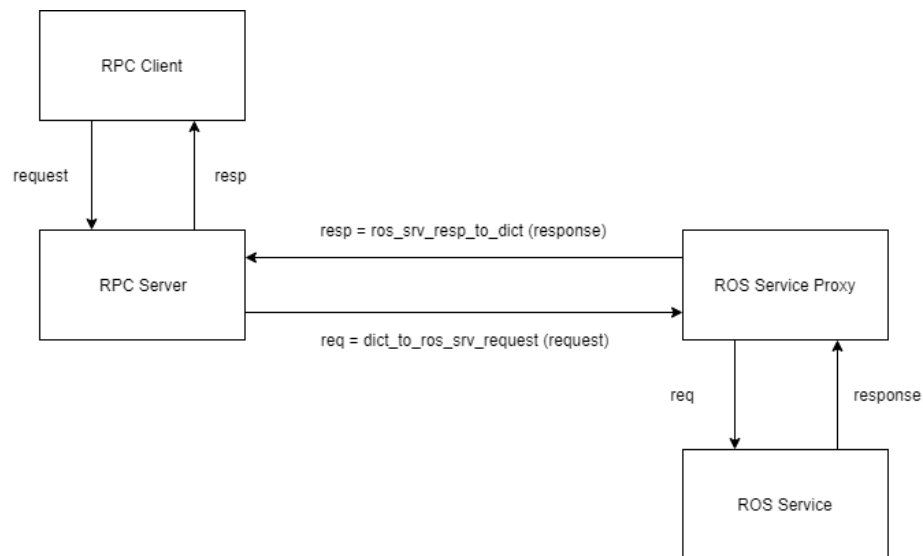
Εικόνα 20: Διαδικασία η οποία ακολουθείται για την γεφύρωση ενός Broker Publisher

#### 4.8.2.2 Μοντέλο RPC

Το μοντέλο publish/subscribe αποτελεί έναν πολύ ευέλικτο τρόπο επικοινωνίας τον οποίο χρησιμοποιεί το ROS ώστε να επιτυγχάνεται η ανταλλαγή μηνυμάτων μεταξύ των διάφορων κόμβων. Ωστόσο, πρόκειται για ένα μοντέλο μονόδρομης επικοινωνίας χωρίς να υποστηρίζει την μέθοδο αλληλεπίδρασης RPC η οποία χρησιμοποιείται πλέον ευρύτατα. Για να διευθετηθεί αυτό το ROS υποστηρίζει την επικοινωνία μεταξύ κόμβων μέσω Services, όπου ο node-client δημιουργεί ένα request προς τον server και ο server αντίστοιχα απαντάει μέσω ενός response μηνύματος. Η διαδικασία αυτή προσομοιώνει το μοντέλο επικοινωνίας RPC που έχει αναπτυχθεί για να επικοινωνούν οι διάφορες συσκευές σε επίπεδο RabbitMQ. Επομένως κρίθηκε αναγκαίο να υλοποιηθεί ένας μηχανισμός ο οποίος να γεφυρώνει το μοντέλο επικοινωνίας μέσω ROS services με το μοντέλο επικοινωνίας RPC που υποστηρίζει ο message broker, ώστε να ολοκληρωθεί η σύνδεση ROS και RabbitMQ με αποτέλεσμα οι διάφορες συσκευές οι οποίες χρησιμοποιούν το ROS για την μεταξύ τους αλληλεπίδραση, να επικοινωνούν πλέον μέσω του RabbitMQ.

Για να επιτευχθεί η γεφύρωση των ROS services με τον message broker, δημιουργήθηκε ένα είδος template, όπως και στην περίπτωση της μεθόδου αλληλεπίδρασης publish/subscribe, το οποίο ουσιαστικά συνδέει έναν RPC server με έναν ROS service proxy. Πιο αναλυτικά, η βάση της υλοποίησης των templates για την γεφύρωση της επικοινωνίας μεταξύ ROS services και RabbitMQ, είναι η δημιουργία ενός RPC server, ο οποίος συνδέεται με έναν ROS service proxy. Όταν ο RPC server σε επίπεδο message broker λάβει ένα request, τα δεδομένα του μηνύματος μετατρέπονται σε μορφή αναγνωρίσιμη από το ROS, όπως έχει παρουσιαστεί στην υποενότητα 5.10.1 και η νέα πλέον ROS μορφή του request στέλνεται στον ROS service proxy ο οποίος στέλνει πίσω την απάντηση του αιτήματος. Το response

λοιπόν, μεταφράζεται από την ROS μορφή του σε λεξικό το οποίο είναι συμβατό με τον message broker, και ο RabbitMQ server επιστρέφει την απάντηση (εικόνα 21). Συμπερασματικά, εισάγοντας τις κατάλληλες παραμέτρους στο template, δημιουργείται ένα εκτελέσιμο αρχείο το οποίο γεφυρώνει αν πάσα στιγμή την επικοινωνία μεταξύ ενός RPC server και ενός ROS service. Υλοποιώντας λοιπόν και αυτή την σύνδεση μεταξύ του μοντέλου επικοινωνίας RPC σε επίπεδο message broker και του μοντέλου αλληλεπίδρασης μέσω services σε επίπεδο ROS, έχει δημιουργηθεί το απαραίτητο πλαίσιο για την επίτευξη της γεφύρωσης των διεπαφών μεταξύ ROS και RabbitMQ.



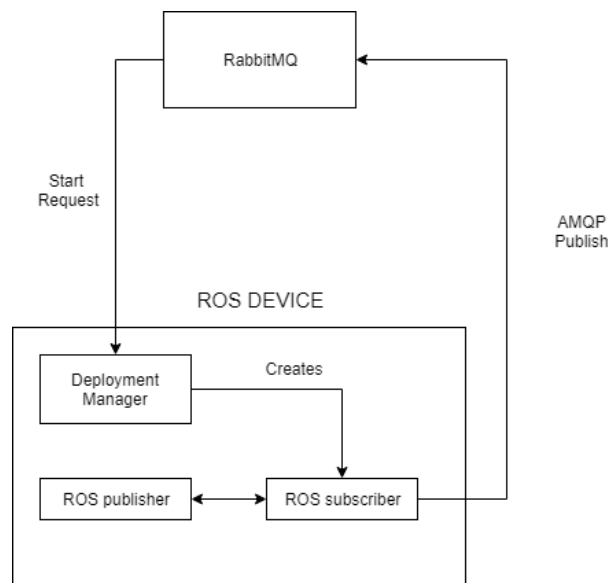
Εικόνα 21: Διαδικασία η οποία ακολουθείται για την γεφύρωση ενός ROS service

#### 4.8.3 Δημιουργία Γεφυρών

Στις προηγούμενες ενότητες έγινε μια περιγραφή του τρόπου με τον οποίο υλοποιείται το πλαίσιο για την γεφύρωση της επικοινωνίας μεταξύ του ROS και του RabbitMQ. Παρουσιάστηκε ο μηχανισμός μέσω του οποίου επιτυγχάνεται η αλληλεπίδραση μεταξύ συσκευών από το επίπεδο του ROS στο επίπεδο του RabbitMQ broker, τόσο στην περίπτωση του μοντέλου επικοινωνίας publish/subscribe όσο και στην περίπτωση της ανταλλαγής πληροφοριών μέσω ROS services. Συνοπτικά θα μπορούσαμε να πούμε ότι για κάθε είδος επικοινωνίας που γεφυρώνεται υπάρχει ένα συγκεκριμένο είδος template, το οποίο διαμορφώνεται κατάλληλα σε κάθε περίπτωση βάσει των παραμέτρων του κάθε μοντέλου, με αποτέλεσμα να δημιουργείται ένα εκτελέσιμο αρχείο το οποίο γεφυρώνει την επικοινωνία. Σ αυτή την ενότητα περιγράφεται ο τρόπος με τον οποίο ο message broker επικοινωνεί με την εκάστοτε συσκευή, ώστε να ενεργοποιήσει τον μηχανισμό γεφύρωσης.

Βαδίζοντας στο τελικό βήμα της διαδικασίας η οποία απαιτείται για την γεφύρωση των διεπαφών του ROS με το RabbitMQ, αναπτύχθηκε μια διαδικασία η

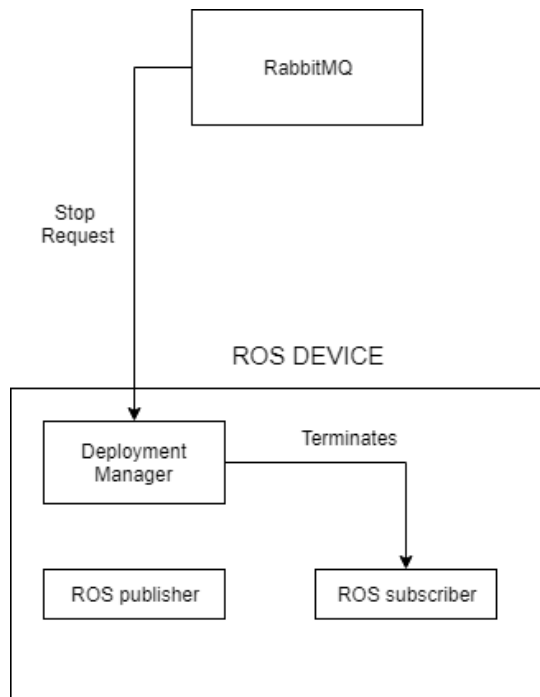
οποία επιτρέπει την αλληλεπίδραση της συσκευής με τον message broker με σκοπό την δημιουργία αλλά και την εκτέλεση του γεφύρωσης της επικοινωνίας μεταξύ ROS και RabbitMQ. Για κάθε μοντέλο επικοινωνίας που επιθυμούμε να γεφυρώσουμε, είτε αυτό αφορά έναν ROS publisher, είτε έναν Broker publisher, είτε ένα ROS service, έχει δημιουργηθεί σε επίπεδο συσκευής ο “deployment manager” ένας RPC server, ο οποίος διαχειρίζεται την δημιουργία της γεφύρωσης. Ο deployment manager δέχεται ένα request μήνυμα το οποίο περιέχει όλες τις απαραίτητες παραμέτρους που απαιτούνται για να δημιουργηθούν, μέσω των templates, τα εκτελέσιμα αρχεία τα οποία υλοποιούν τελικά την σύνδεση της επικοινωνίας. Για παράδειγμα, σε περίπτωση που ο χρήστης επιθυμεί την σύνδεση ενός ROS publisher με τον message broker, στέλνει ένα request μήνυμα προς τον deployment manager, ο οποίος τρέχει στην εκάστοτε συσκευή, το μήνυμα περιέχει το όνομα και το είδους του ROS topic που επιθυμεί να γεφυρώσει, καθώς και την παράμετρο «start» και έτσι δημιουργείται το εκτελέσιμο αρχείο το οποίο υλοποιεί την γέφυρα επικοινωνίας μεταξύ ROS publisher και RabbitMQ, διαδικασία η οποία φαίνεται στην εικόνα 22. Αντίστοιχη διαδικασία ακολουθείται και στην περίπτωση των ROS services αλλά και της γεφύρωσης ενός Broker publisher με το ROS.



Εικόνα 22: Η διαδικασία γεφύρωσης ενός ROS publisher

Κατά την αντίθετη διαδικασία, ο χρήστης είναι δυνατόν να επιθυμεί κάποια στιγμή να διακόψει την γεφύρωση ROS-RabbitMQ. Γι' αυτό τον σκοπό, ο εκάστοτε deployment manager ο οποίος λειτουργεί σε επίπεδο συσκευής ενσωματώνει και μια επιπλέον λειτουργία εκτός της δημιουργίας σύνδεσης μεταξύ ROS και message broker, αυτή της παύσης της επικοινωνίας μεταξύ τους. Αρκεί λοιπόν ο χρήστης να επιλέξει την σύνδεση η οποία έχει δημιουργηθεί και επιθυμεί να διακόψει δίνοντας τις απαραίτητες μεταβλητές, και με την μορφή ενός request μηνύματος προς τον RPC server του ρομπότ, το οποίο έχει και την παράμετρο «stop», η γέφυρα επικοινωνίας διακόπτεται (εικόνα 23). Η παραπάνω διαδικασία επιτυγχάνεται μέσω της παύσης

εκτέλεσης του εκτελέσιμου αρχείου το οποίο έχει δημιουργηθεί μέσω των templates για την γεφύρωση της επικοινωνίας.



Εικόνα 23: Η διαδικασία τερματισμού μιας γέφυρας επικοινωνίας

Συμπερασματικά, για την ολοκλήρωση της διαδικασίας γεφύρωσης των διεπαφών του ROS με το RabbitMQ δημιουργήθηκε ένας μηχανισμός μέσω RPC ο οποίος επιτρέπει την αλληλεπίδραση μεταξύ του message broker και του ROS, με σκοπό την ενεργοποίηση ή παύση του μηχανισμού γεφύρωσης των διαφόρων μοντέλων επικοινωνίας. Μ αυτό τον τρόπο έχει ολοκληρωθεί όλο το πλαίσιο επικοινωνίας το οποίο απαιτείται προκειμένου να επιτευχθεί η γεφύρωση της επικοινωνίας.

#### 4.9 Διαδικτυακή Εφαρμογή

Το τελευταίο κομμάτι του συστήματος αποτελείται από την διαδικτυακή (web) εφαρμογή μέσω την οποίας επιτυγχάνεται η αλληλεπίδραση του χρήστη με το υπόλοιπο σύστημα. Για την πραγματοποίηση της διαδικτυακής εφαρμογής χρησιμοποιήθηκε το framework flask διότι είναι αρκετά απλό στη χρήση, προσφέροντας παράλληλα τεράστια ευελιξία και σχετικά απλή αποσφαλμάτωση. Στη συνέχεια πραγματοποιήθηκε και ο προγραμματισμός σε python για τη συγκεκριμένη λειτουργία που θα επιτελούσε η κάθε υπηρεσία. Στο παρόν κεφάλαιο περιγράφεται το back-end κομμάτι της web εφαρμογής οι βασικές λειτουργίες του οποίου φαίνονται συνοπτικά παρακάτω:

- Εγγραφή συσκευών και χρηστών στο σύστημα.

- Σύνδεση συσκευών με χρήστες και παρουσίαση των συσκευών που αντιστοιχούν σε κάθε χρήστη.
- Παρουσίαση του ROS περιβάλλοντος κάθε συσκευής.
- Απεικόνιση διαφόρων στατιστικών χρήσης της κάθε συσκευής (Device monitor).
- Δημιουργία γέφυρας επικοινωνίας μεταξύ των ROS-RabbitMQ topics και services.
- Τερματισμός της γεφύρωσης επικοινωνίας μεταξύ ROS-RabbitMQ topics και services.

Πιο συγκεκριμένα, η διαδικτυακή εφαρμογή επιτρέπει την εγγραφή των συσκευών αλλά και των χρηστών στο σύστημα μέσω του REST API που έχει δημιουργηθεί. Έπειτα, κάθε εγγεγραμμένος χρήστης μπορεί να συνδεθεί στην εφαρμογή έχοντας πρόσβαση σε όλες τις λειτουργίες του συστήματος. Μέσω του API της εφαρμογής ο χρήστης έχει την δυνατότητα να εγγράψει ένα ρομπότ στην κατοχή του, παρέχοντας το όνομα και τον κωδικό της συσκευής, και έπειτα να το χρησιμοποιήσει. Επίσης, μια πολύ σημαντική δυνατότητα η οποία δίνεται στον χρήστη είναι εισάγοντας το όνομα της συσκευής, να έχει την δυνατότητα να βλέπει τα ROS topics, services, nodes που είναι ενεργά κάθε στιγμή. Με αυτό τον τρόπο του δίνεται η ευχέρεια να επιλέξει στη συνέχεια για ποια από αυτά θα δημιουργήσει γέφυρες επικοινωνίας με τον message broker. Βασικό συστατικό της υλοποίησης αποτελεί και η απεικόνιση των στατιστικών στοιχείων κάθε συσκευής, όπως οι πόροι που χρειάζεται σε μνήμη και υπολογιστική ισχύ, οι διάφορες μετρήσεις από αισθητήρες καθώς και οι διεργασίες και οι συνδέσεις κάθε συσκευής. Ο χρήστης λοιπόν, δίνοντας το όνομα της συσκευής που τον ενδιαφέρει, έχει πρόσβαση σε πληθώρα στοιχείων και μπορεί να επιλέξει ανά πάσα στιγμή, ανάλογα με την κρίση του, την αποσύνδεση της συσκευής από την εφαρμογή. Η πιο σημαντική λειτουργία της διαδικτυακής εφαρμογής είναι η δυνατότητα που δίνεται στο χρήστη για δημιουργία αλλά και διακοπή γεφυρών επικοινωνίας μεταξύ των ROS-RabbitMQ topics και services. Ακόμα ο χρήστης μέσω του API της εφαρμογής επιλέγει την συσκευή με την οποία θέλει να δημιουργήσει την αλληλεπίδραση μεταξύ ROS-RabbitMQ παρέχοντας το όνομα της. Έπειτα επιλέγει το είδος της επικοινωνίας που επιθυμεί να συνδέσει είτε αυτό αφορά ένα ROS service, είτε ROS topics είτε topics τα οποία «τρέχουν» στον message broker. Κατ' αυτό τον τρόπο υλοποιείται η γεφύρωση των διεπαφών ROS και message broker, η οποία μπορεί να τερματιστεί ανά πάσα στιγμή μετά από εντολή του χρήστη. Τέλος με την έξοδο του χρήστη από την διαδικτυακή εφαρμογή διακόπτονται όλες οι γέφυρες επικοινωνίας.

Παρακάτω παρουσιάζονται όλες οι υπηρεσίες του flask API όπως προκύπτουν από τις προδιαγραφές του συστήματος.

- **/user/signup** : Δέχεται ως είσοδο τα στοιχεία εγγραφής του χρήστη, τα οποία αποθηκεύονται στην βάση δεδομένων.

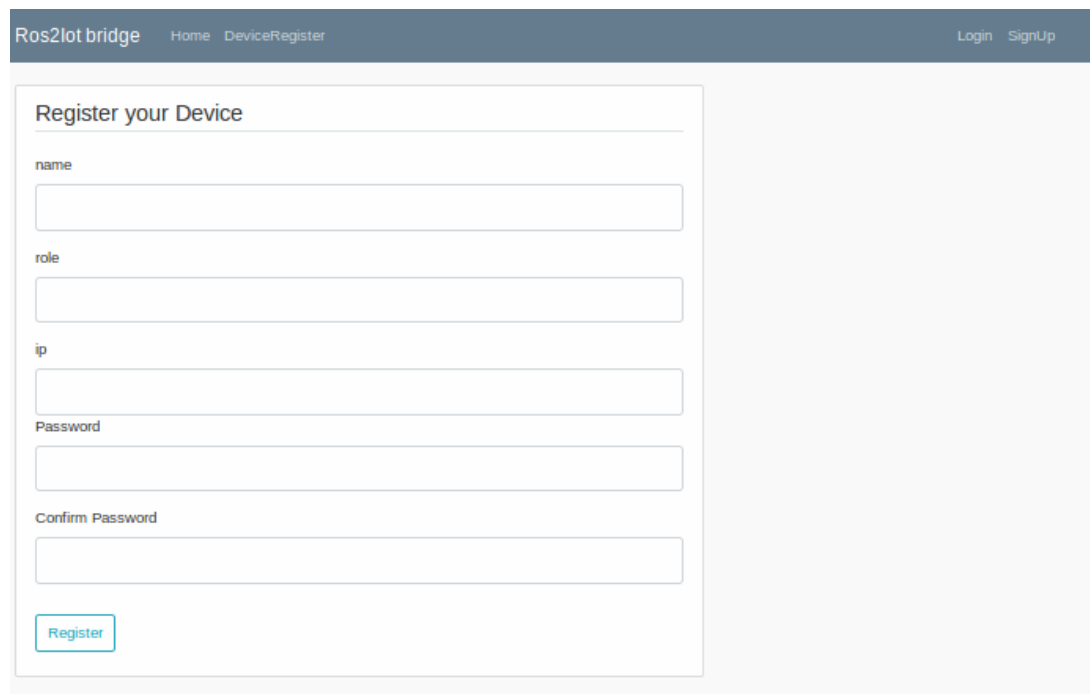
- **/user/login:** Δέχεται το username και password του χρήστη κα αναλόγως του επιτρέπεται ή όχι η είσοδος.
- **/register/device:** Δέχεται ως είσοδο τα στοιχεία της εκάστοτε συσκευής τα οποία εγγράφονται στην βάση δεδομένων.
- **/user/{username}/logout:** Αποσυνδέει τον χρήστη από το σύστημα.
- **/user/{username}/connectdevice/{devicename}:** Δέχεται το όνομα και τον κωδικό της συσκευής και αν είναι αληθή καταχωρεί μια νέα συσκευή στην κατοχή του χρήστη.
- **/user/{username}/connecteddevices:** Επιστρέφει λίστα με τις συσκευές τις οποίες κατέχει ο κάθε χρήστης.
- **/user/{username}/monitor/devices:** Δέχεται ως είσοδο το όνομα της συσκευής και επιστρέφει τα στατιστικά της στοιχεία.
- **/user/{username}/monitor/ros/{devicename}:** Δέχεται ως είσοδο το όνομα της συσκευής και επιστρέφει το ROS περιβάλλον της (nodes, services και topics).
- **/user/{username}/bridge/rostopic:** Έχει ως είσοδο το όνομα της συσκευής, το όνομα και το είδος του ROS topic και δημιουργεί την γεφύρωση της επικοινωνίας με τον message broker.
- **/user/{username}/terminate/rostopic:** Δέχεται το όνομα της συσκευής, το όνομα και το είδος του ROS topic και τερματίζεται, εφόσον είναι ενεργή, η γεφύρωση του topic μεταξύ ROS και RabbitMQ.
- **/user/{username}/bridge/broktopic:** Δέχεται σαν είσοδο το όνομα της συσκευής, το όνομα και το είδος του topic το οποίο «τρέχει» σε επίπεδο message broker και πραγματοποιείται η γεφύρωση του με το ROS.
- **/user/{username}/terminate/broktopic:** Δέχεται σαν είσοδο το όνομα της συσκευής, το όνομα και το είδος του topic το οποίο «τρέχει» σε επίπεδο message broker και τερματίζεται, εφόσον είναι ενεργή, η γεφύρωση του με το ROS.
- **/user/{username}/bridge/service:** Δέχεται σαν είσοδο το όνομα της συσκευής, το όνομα και το είδος του ROS service και υλοποιείται η σύνδεση του με το RabbitMQ.
- **/user/{username}/terminate/broktopic:** Δίνεται η δυνατότητα στο χρήστη μέσω της εισαγωγής του ονόματος της συσκευής καθώς και του ονόματος και του είδους του ROS service να τερματιστεί, εφόσον είναι ενεργή, η γεφύρωση της επικοινωνίας.

#### 4.10 Γραφικό περιβάλλον

Για την διαχείριση του συνολικού συστήματος ο χρήστης είναι απαραίτητο να συνδεθεί στην διαδικτυακή εφαρμογή που έχει υλοποιηθεί και μέσω του γραφικού περιβάλλοντος να είναι σε θέση να χρησιμοποιήσει όλες τις λειτουργίες του

συστήματος. Για την υλοποίηση του front-end στοιχείου της εφαρμογής δημιουργήθηκαν templates χρησιμοποιώντας την γλώσσα σήμανσης HTML(Hyper Text Markup Language), καθώς και τη γλώσσα CSS(Cascade Style Sheets) η οποία συνδυάζεται με την HTML για τον καλύτερο έλεγχο της εμφάνισης της εφαρμογής. Στην παρόν κεφάλαιο γίνεται μια συνοπτική περιγραφή του γραφικού περιβάλλοντος της εφαρμογής, το οποίο παρουσιάζεται λεπτομερώς, μέσω ενός σεναρίου χρήσης, στο κεφάλαιο 6.

Στην αρχική σελίδα της εφαρμογής ο χρήστης μπορεί να επιλέξει να εγγραφεί μια συσκευή, να κάνει εγγραφή ο ίδιος ή να συνδεθεί στην εφαρμογή εάν έχει δημιουργήσει ήδη λογαριασμό. Ακολουθώντας, ανάλογα με την επιλογή του εμφανίζονται οι φόρμες συμπλήρωσης για κάθε περίπτωση αντίστοιχα όπως φαίνονται στις εικόνες 24, 25, 26.



The screenshot shows a web application interface for 'Ros2lot bridge'. The top navigation bar includes 'Home' and 'DeviceRegister' links, along with 'Login' and 'SignUp' buttons. The main content area features a form titled 'Register your Device'. This form contains five input fields: 'name', 'role', 'ip', 'Password', and 'Confirm Password'. A 'Register' button is positioned at the bottom left of the form.

Εικόνα 24: Σελίδα για εγγραφή μιας συσκευής στην βάση δεδομένων

The screenshot shows the 'Join Today' registration page of the 'Ros2lot bridge' application. The page has a dark blue header with the application name and navigation links 'Home' and 'DeviceRegister'. On the right side of the header are links for 'Login' and 'SignUp'. The main content area is a light gray box containing the registration form. The form is titled 'Join Today' and includes four input fields: 'Username', 'FullName', 'Password', and 'Confirm Password'. Below these fields is a blue 'Sign Up' button. At the bottom of the form, there is a link 'Already Have An Account? Sign In'.

Εικόνα 25: Σελίδα για εγγραφή ενός χρήστη στην βάση δεδομένων

The screenshot shows the 'Log In' page of the 'Ros2lot bridge' application. The page has a dark blue header with the application name and navigation links 'Home' and 'DeviceRegister'. On the right side of the header are links for 'Login' and 'SignUp'. The main content area is a light gray box containing the login form. The form is titled 'Log In' and includes two input fields: 'Username' and 'Password'. Below these fields is a blue 'Login' button. At the bottom of the form, there is a link 'Need An Account? Sign Up Now'.

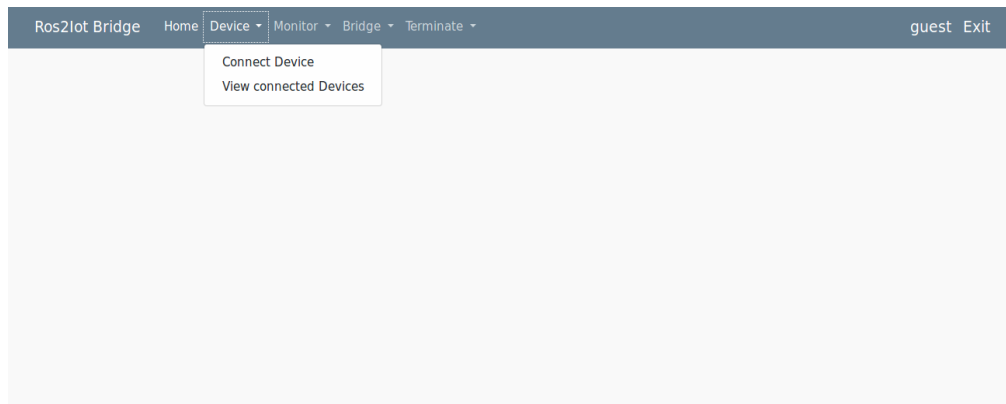
Εικόνα 26: Σελίδα στην οποία ο χρήστης εισάγει τα στοιχεία εγγραφής του για να του επιτραπεί η είσοδος

Εφόσον επιτευχθεί η ταυτοποίηση του χρήστη παρέχοντας τον σωστό συνδυασμό username και password, ο χρήστης εισέρχεται στην αρχική σελίδα η οποία αναφέρεται στους εγγεγραμμένους χρήστες. Έπειτα κάθε χρήστης έχει πρόσβαση στις διάφορες λειτουργίες του συστήματος, μέσω του navigation bar που βρίσκεται στο πάνω μέρος της εφαρμογής. Αυτές οι λειτουργίες μπορούν να κατηγοριοποιηθούν σε τέσσερις βασικές ενότητες κάθε μια εκ των οποίων επιτελεί διάφορες υλοποιήσεις.

Αρχικά, στην κατηγορία των Device, ο χρήστης μπορεί να επιλέξει να συνδεθεί με μια συσκευή, προκειμένου να μπορεί να την διαχειριστεί, ή να εμφανίσει την λίστα με τις συσκευές που έχει στην κατοχή του (εικόνα 27). Προκειμένου ο χρήστης να

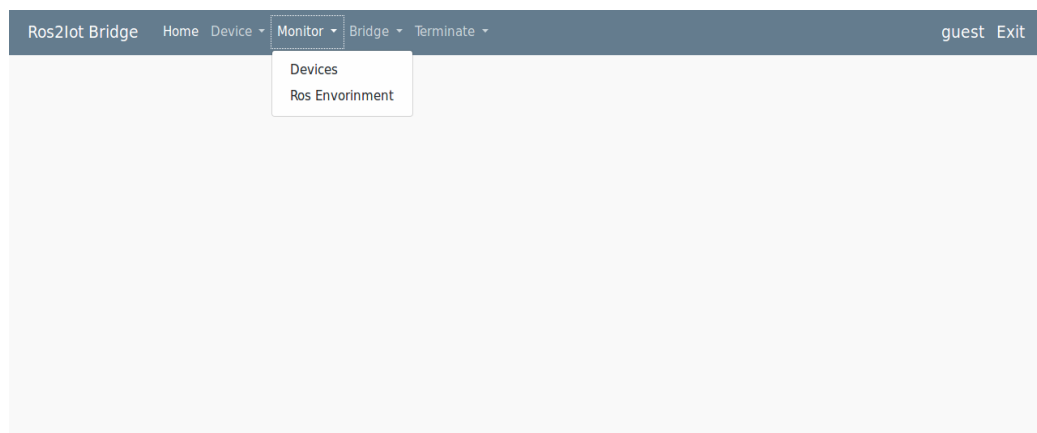


μπορέσει να συνδεθεί με μια συσκευή, απαραίτητη είναι η χορήγηση του ονόματος και του κωδικού της συσκευής.



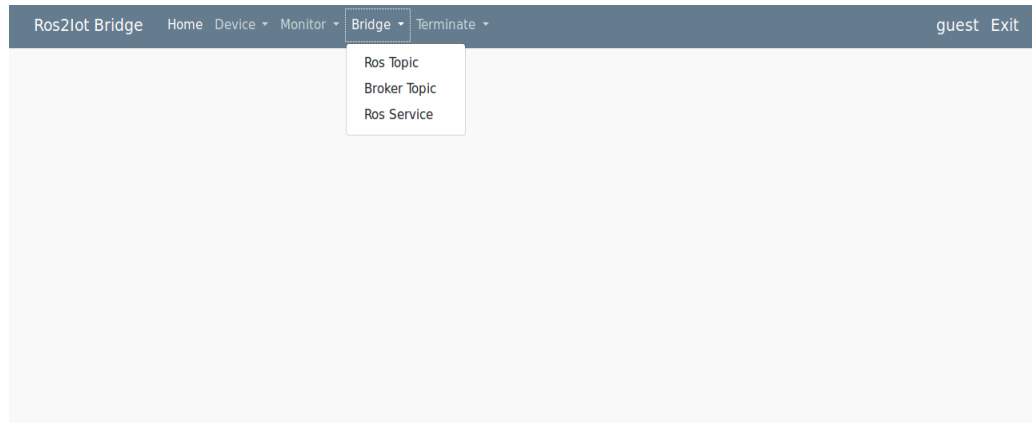
Εικόνα 27: Δυνατότητες εφαρμογής στην κατηγορία Device

Στην κατηγορία monitor, ο χρήστης έχει την δυνατότητα να επιλέξει αν επιθυμεί να δει τα στατιστικά στοιχεία ή το ROS περιβάλλον το οποίο «τρέχει» στην κάθε συσκευή (εικόνα 28). Στην περίπτωση του Device monitor, ο χρήστης εισάγει στην φόρμα το όνομα της συσκευής και ανάλογα του εμφανίζεται μια διεπαφή η οποία αναφέρεται στα λειτουργικά στοιχεία της συσκευής. Αντίστοιχη διαδικασία ακολουθείται και στην περίπτωση monitor του ROS Environment.



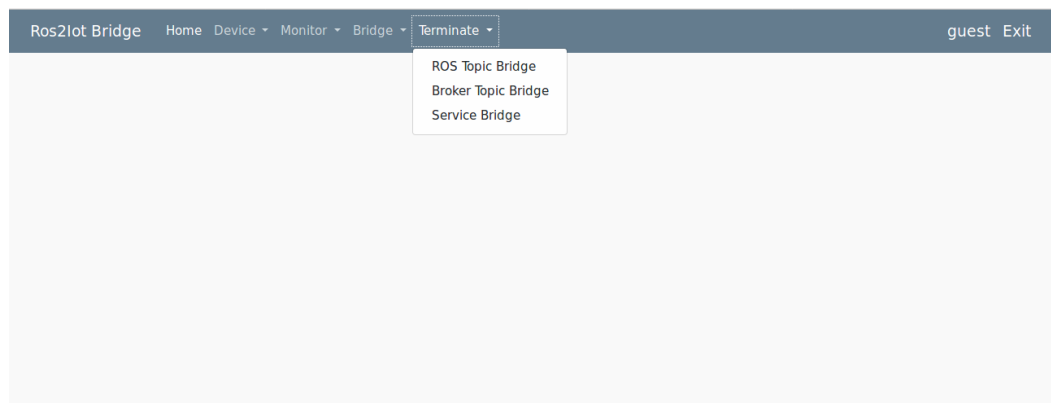
Εικόνα 28: Δυνατότητες εφαρμογής στην κατηγορία Monitor

Μια άλλη δυνατότητα της εφαρμογής είναι η λειτουργία Bridge βάσει της οποίας δίνεται η δυνατότητα στον χρήστη να επιλέξει το είδος της επικοινωνίας το οποίο επιθυμεί να γεφυρώσει μεταξύ RabbitMQ-ROS. Ο χρήστης μπορεί να επιλέξει να συνδέσει ένα ROS topic με το RabbitMQ, ένα message broker topic με το ROS, καθώς και ένα ROS service με το RabbitMQ όπως φαίνεται στην εικόνα 29. Στην περίπτωση του ROS topic, ο χρήστης εισάγει το όνομα της συσκευής, το όνομα και το είδος του ROS topic και δημιουργεί την γέφυρα επικοινωνίας. Αντίστοιχα η ίδια διαδικασία ακολουθείται και στην περίπτωση των message broker topics και ROS services.



Εικόνα 29: Δυνατότητες εφαρμογής στην κατηγορία Bridge

Τέλος βασική δυνατότητα η οποία παρέχεται στον χρήστη είναι η διακοπή της επικοινωνίας των topics, services μεταξύ ROS και message broker. Στη βάση αυτής της δυνατότητας ο χρήστης στην κατηγορία terminate, μπορεί να διακόψει το είδος της επικοινωνίας το οποίο επιθυμεί επιλέγοντας μια από τις τρεις κατηγορίες (εικόνα 30).



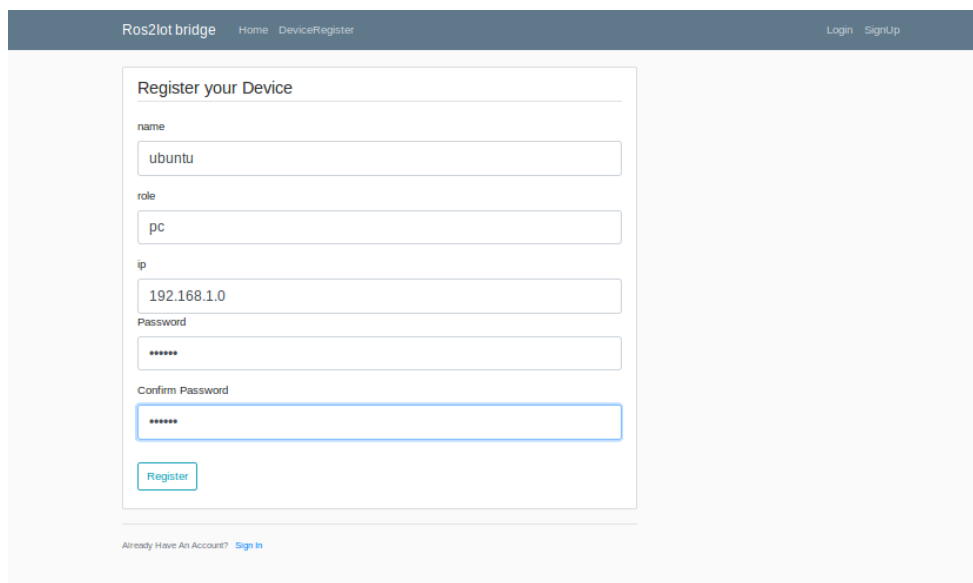
Εικόνα 30: Δυνατότητες εφαρμογής στην κατηγορία Terminate

## 5 Σενάρια Χρήσης του Συστήματος

Στο παρόν κεφάλαιο παρουσιάζονται δύο σενάρια χρήσης του συνολικού συστήματος. Αρχικά περιγράφεται ένα σενάριο χρήσης όπου αναλύεται λεπτομερώς η διαδικασία η οποία είναι απαραίτητο να ακολουθηθεί από τον χρήστη ώστε να δημιουργηθεί μια γέφυρα σύνδεσης μεταξύ ενός ROS publisher με τον message broker RabbitMQ. Το δεύτερο σενάριο χρήσης αφορά την δημιουργία μιας εφαρμογής, με τη βοήθεια του εργαλείου Node-RED, η οποία επικοινωνεί με το ρομπότ μετά την γεφύρωση των απαραίτητων topics και services.

### 5.1 Εγγραφή Συσκευής και Χρήστη

Αρχικά είναι απαραίτητο να εγγράψουμε την συσκευή στην βάση δεδομένων, προκειμένου να μπορεί να συνδεθεί με το RabbitMQ. Επιλέγοντας λοιπόν το «DeviceRegister» και συμπληρώνοντας την φόρμα με τα στοιχεία της συσκευής, όπως παρουσιάζεται στην εικόνα 31, έχουμε καταχωρήσει την συσκευή στην βάση δεδομένων. Έπειτα μπορούμε να εκτελέσουμε, από την πλευρά της συσκευής, όλα εκείνα τα εκτελέσιμα αρχεία που επιτυγχάνουν την σύνδεση των διαφόρων υπηρεσιών της συσκευής με τον message broker. Αντίστοιχη διαδικασία εγγραφής είναι απαραίτητο να ακολουθηθεί και για την περίπτωση των χρηστών, όπου κάθε χρήστης επιλέγοντας την ενότητα «Sign Up» είναι αναγκαίο να δημιουργήσει έναν λογαριασμό, τα στοιχεία του οποίου αποθηκεύονται στην βάση δεδομένων και χρησιμοποιούνται αργότερα προκειμένου να επιτευχθεί η ταυτοποίηση του. Τέλος επιλέγοντας «Login» εισάγοντας το username και το password και έχοντας γίνει επιτυχής ταυτοποίηση έχει πλέον πραγματοποιηθεί η είσοδος του χρήστη στο σύστημα.



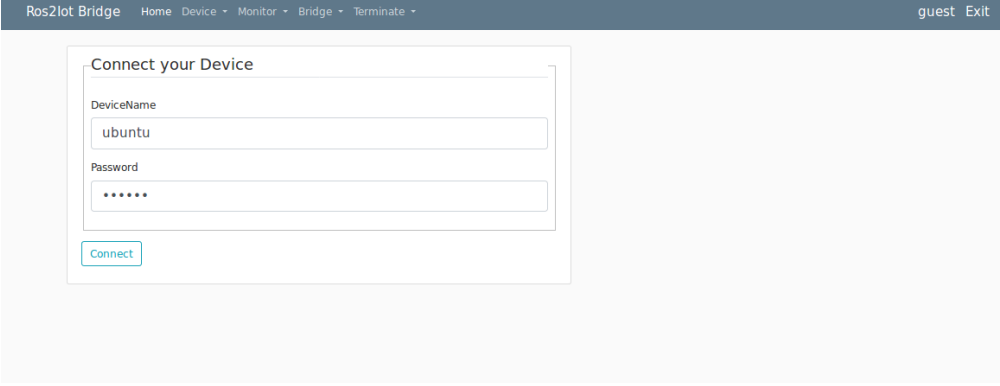
The screenshot shows a web interface for 'Ros2IoT bridge' with a 'DeviceRegister' tab. The 'Register your Device' form contains the following fields and values:

- name: ubuntu
- role: pc
- ip: 192.168.1.0
- Password: (masked with asterisks)
- Confirm Password: (masked with asterisks)

A 'Register' button is located at the bottom of the form. Below the form, there is a link: 'Already Have An Account? [Sign In](#)'.

Εικόνα 31: Συμπλήρωση των στοιχείων εγγραφής μιας συσκευής

Για να έχει την δυνατότητα ο χρήστης να χρησιμοποιήσει την συσκευή είναι αναγκαίο να την προσθέσει στις συσκευές τις οποίες κατέχει, διαδικασία η οποία επιτυγχάνεται μέσω της λειτουργίας «Device/Connect Device». Εισάγοντας λοιπόν το σωστό συνδυασμό `device_name` και `password` ο χρήστης μπορεί πλέον να χρησιμοποιήσει όλες τις λειτουργίες που αφορούν την συγκεκριμένη συσκευή (εικόνα 32). Τέλος μέσω της λειτουργίας «Device/View Connected Devices», του δίνεται η δυνατότητα να γνωρίζει ανά πάσα στιγμή τις συσκευές τις οποίες κατέχει.

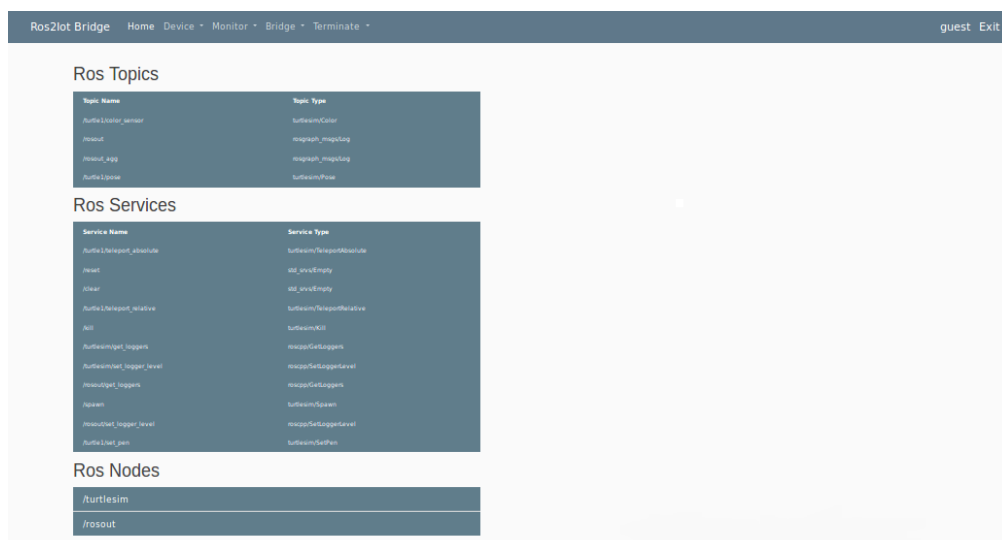
The image shows a web browser window with the title 'Ros2lot Bridge'. The navigation bar includes 'Home', 'Device', 'Monitor', 'Bridge', and 'Terminate'. The user is logged in as 'guest' with an 'Exit' link. The main content area is titled 'Connect your Device' and contains a form with two input fields: 'DeviceName' with the value 'ubuntu' and 'Password' with masked characters '\*\*\*\*\*'. A 'Connect' button is located below the form.

Εικόνα 32: Σύνδεση του χρήστη με συσκευές ώστε να έχει την δυνατότητα μετέπειτα να τις χρησιμοποιεί

## 5.2 Έλεγχος Περιβάλλοντος Συσκευής

Για την ορθή λειτουργία της εφαρμογής είναι αναγκαίο ο χρήστης να είναι σε θέση οποιαδήποτε στιγμή να ελέγχει στοιχεία τα οποία αφορούν τις απαιτήσεις σε πόρους μνήμης και υπολογιστικής ισχύς κάθε συσκευής καθώς και διάφορες τιμές από αισθητήρες όπως η μπαταρία και η θερμοκρασία της. Στην επιλογή λοιπόν «Monitor/Device» εμφανίζεται μια φόρμα στην οποία ο χρήστης εισάγει το όνομα της συσκευής την οποία επιθυμεί να ελέγξει. Ακολουθώντας εμφανίζεται η σελίδα με τα στοιχεία της συσκευής. Στο παρόν σενάριο χρήσης έχουμε επιλέξει ορισμένες τιμές στοιχείων που αφορούν υπολογιστική ισχύ, χρήση μνήμης και χρήση δίσκου, καθώς για την συγκεκριμένη συσκευή δεν είναι ενεργοποιημένοι οι αισθητήρες που αφορούν τιμές μπαταρίας και θερμοκρασίας. Επίσης τιμές οι οποίες αφορούν τις διεργασίες που εκτελούνται στην συσκευή καθώς και τις διάφορες συνδέσεις θεωρήθηκε ότι θα έκαναν πολύπλοκη την παρουσίαση της λειτουργίας της εφαρμογής. Όπως παρουσιάζεται στην εικόνα 33, βλέπουμε στην κατηγορία «Device Stats» τις τιμές που αφορούν την συνολική αλλά και επί τοις εκατό χρήση της CPU καθώς και την επί τοις εκατό χρήση της μνήμης. Επίσης στην ενότητα «Memory Used», ο χρήστης είναι σε θέση να δει στοιχεία που αφορούν την συνολική αλλά και επιμέρους χρήση της μνήμης. Μ αυτή την λειτουργία του δίνεται η δυνατότητα να επιλέγει ανά πάσα στιγμή αν επιθυμεί την συνέχιση ή την παύση χρήση της συσκευής.

Επεκτείνοντας την λειτουργία ελέγχου και παρακολούθησης της συσκευής, μια άλλη πολύ σημαντική δυνατότητα της εφαρμογής είναι η απεικόνιση του ROS περιβάλλοντος το οποίο «τρέχει» στην πλευρά της συσκευής. Ο χρήστης επιλέγοντας την κατηγορία «Monitor/ROS Environment», μπορεί οποιαδήποτε στιγμή να δει τα ROS topics, services, nodes που είναι ενεργά στην συσκευή καθώς και το είδος αυτών, όπως φαίνεται στην εικόνα 34. Κατ' αυτό τον τρόπο μπορεί στην συνέχεια να επιλέξει τις γέφυρες επικοινωνίας με τον message broker όπου θα υλοποιήσει.

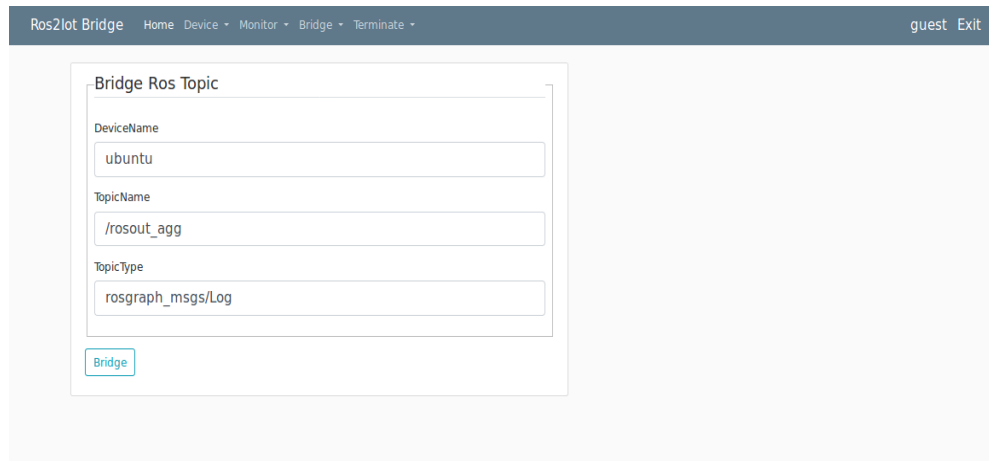


### 5.3 Δημιουργία και Τερματισμός Γεφύρωσης

---

Φιλίππου Αρίστιπος - ΑΕΜ 7795

για το ROS topic «/rosout\_agg». Επιλέγοντας λοιπόν «bridge/Ros topic» μας εμφανίζεται η φόρμα συμπλήρωσης στοιχείων (εικόνα 35), όπου συμπληρώνουμε το όνομα της συσκευής καθώς και το όνομα και το είδος του ROS topic το οποίο θέλουμε να γεφυρώσουμε. Στην περίπτωση που τα στοιχεία που εισάγουμε είναι σωστά η εφαρμογή μας ενημερώνει ότι η επιλογή μας πραγματοποιήθηκε.



The screenshot shows the 'Ros2lot Bridge' web application. At the top, there is a navigation bar with links: Home, Device, Monitor, Bridge, and Terminate. On the right of the bar, it says 'guest Exit'. The main content area is titled 'Bridge Ros Topic'. It contains three input fields: 'DeviceName' with the value 'ubuntu', 'TopicName' with the value '/rosout\_agg', and 'TopicType' with the value 'rosgraph\_msgs/Log'. Below these fields is a blue button labeled 'Bridge'.

Εικόνα 35: Φόρμα στην οποία ο χρήστης εισάγει τα στοιχεία και δημιουργείται η γεφύρωση ενός ROS publisher

Αν μεταβούμε τώρα στην ενότητα «Monitor/Ros Environment» θα παρατηρήσουμε, όπως φαίνεται στην εικόνα 36, ότι έχει δημιουργηθεί ένας νέος ROS κόμβος σε επίπεδο συσκευής. Αυτός ο κόμβος υλοποιεί και την γέφυρα επικοινωνίας καθώς οποιαδήποτε μήνυμα γίνει publish στο ROS topic «/rosout\_agg» γίνεται publish και σε ένα αντίστοιχο topic σε επίπεδο message broker. Με αυτό τον τρόπο μια άλλη συσκευή θα μπορούσε να λάβει τα δεδομένα ROS, όπου ο χρήστης επιλέγοντας το «Bridge/Broker topic» και συμπληρώνοντας το όνομα της συσκευής, καθώς και το όνομα και είδος του topic θα ήταν σε θέση να συνδέσει απομακρυσμένα τις δύο συσκευές οι οποίες χρησιμοποιούν ROS, μέσω του RabbitMQ. Αντίστοιχη διαδικασία ακολουθείται και στην περίπτωση γεφύρωσης ενός ROS service με το RabbitMQ.

Ros2lot Bridge

Home

Device

Monitor

Bridge

Terminate

guest

Exit

Ros Topics

Topic Name	Topic Type
turtle3/color_sensor	turtle3/Color
/rosout	roscpp/roscpplog
/rosout_agg	roscpp/roscpplog
turtle3/pose	turtle3/Pose

Ros Services

Service Name	Service Type
/ros_topic_rosout_agg_bridge_14461_1568195190292/set_logger_level	turtle3/SetLoggerLevel
turtle3/set_report_absolute	std_msgs/Empty
/clear	std_msgs/Empty
/clear	turtle3/SetReportRelative
turtle3/set_report_relative	turtle3/Kill
/kill	roscpp/GetLoggers
turtle3/get_loggers	roscpp/GetLoggerLevel
turtle3/set_logger_level	roscpp/GetLoggers
/rosout/get_loggers	turtle3/Pose
/pose	roscpp/GetLoggerLevel
/ros_topic_rosout_agg_bridge_14461_1568195190292/get_loggers	turtle3/SetPen
/rosout/set_logger_level	turtle3/SetReportAbsolute
turtle3/set_pen	std_msgs/Empty

Ros Nodes

/turtlesim
/rosout
/ros_topic_rosout_agg_bridge_14461_1568195190292

Εικόνα 36: Το ROS περιβάλλον της συσκευής μετά την γεφύρωση ενός ROS Publisher

Τέλος, ο χρήστης μέσω της εφαρμογής είναι δυνατό ανά πάσα στιγμή να σταματάει την εκτέλεση οποιασδήποτε γέφυρας έχει δημιουργήσει. Μεταβαίνοντας λοιπόν στην ενότητα «Terminate/ROS Topic», και συμπληρώνοντας τα στοιχεία της συσκευής και της επικοινωνίας την οποία θέλει να διακόψει (εικόνα 37), η γέφυρα επικοινωνίας διακόπτεται γεγονός το οποίο μπορεί να γίνει αντιληπτό και από την παύση λειτουργίας του ROS κόμβου που ήταν υπεύθυνος για την δημιουργία της γέφυρας επικοινωνίας.

Ros2lot Bridge		Home	Device	Monitor	Bridge	Terminate	guest	Exit
Terminate Ros Topic								
DeviceName								
ubuntu								
TopicName								
/rosout_agg								
Terminate								

Εικόνα 37: Φόρμα για την διακοπή της γεφύρωσης ενός ROS topic

## 5.4 Δημιουργία Node-RED Εφαρμογής

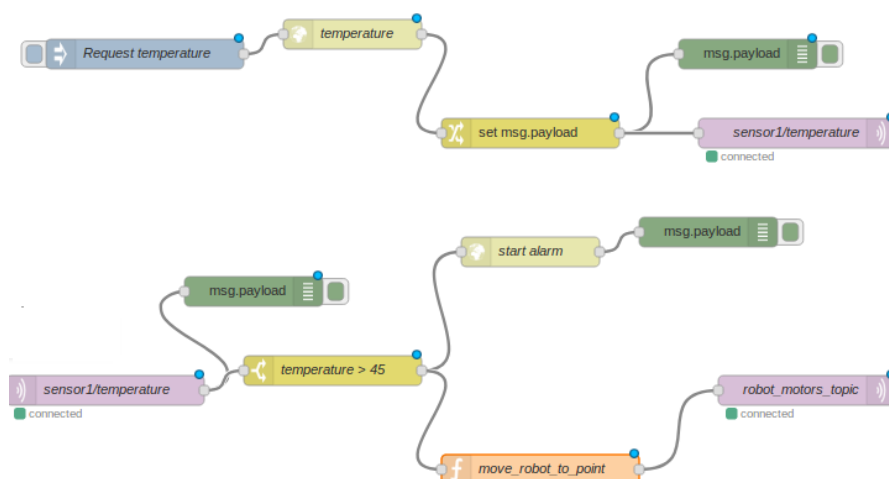
Σ' αυτό το σενάριο χρήσης αναλύεται μια εφαρμογή η οποία επικοινωνεί απομακρυσμένα, μέσω του συστήματος του οποίου υλοποιήθηκε στην παρούσα διπλωματική εργασία, με την ρομποτική συσκευή. Για να επιτευχθεί αυτό, το πρώτο βήμα είναι η γεφύρωση των απαραίτητων topics, services του ROS με τον message broker, διαδικασία η οποία αναλύθηκε λεπτομερώς στο προηγούμενο σενάριο χρήσης. Στόχος του συγκεκριμένου σεναρίου χρήσης είναι να τονίσει την ευκολία με την οποία μπορούν να αναπτυχθούν εφαρμογές οι οποίες επικοινωνούν απευθείας με το ρομπότ, μέσω της χρήσης του συνολικού συστήματος, γεγονός που διευρύνει τα όρια στην ανάπτυξη ρομποτικών εφαρμογών.

Η εφαρμογή η οποία υλοποιήθηκε φαίνεται στο παρακάτω Node-RED διάγραμμα (εικόνα 38). Πρόκειται για μία εφαρμογή η οποία ελέγχει, μέσω ενός αισθητήρα, την θερμοκρασία μιας συστοιχίας μπαταριών. Η τιμή της θερμοκρασίας λαμβάνεται μέσω ενός HTTP request, και γίνεται publish στον message broker στο topic "sensor1/temperature". Το πρωτόκολλο επικοινωνίας που χρησιμοποιείται στην συγκεκριμένη περίπτωση είναι το MQTT το οποίο υποστηρίζεται από το RabbitMQ μέσω της ενεργοποίησης του MQTT plugin<sup>56</sup>. Το επόμενο κομμάτι της εφαρμογής αφορά έναν subscriber ο οποίος εγγράφεται στο topic "sensor1/temperature" και λαμβάνει κάθε στιγμή την θερμοκρασία από τον αισθητήρα. Αν η θερμοκρασία ξεπεράσει την προβλεπόμενη τιμή, στην συγκεκριμένη περίπτωση τους 45 βαθμούς, ενεργοποιείται μέσω ενός HTTP request ο συναγερμός. Επίσης ένα ρομπότ πυρόσβεσης είναι δυνατόν να μετακινηθεί στο σημείο, διαδικασία η οποία υλοποιείται μέσω του συστήματος γεφύρωσης του ROS με το RabbitMQ. Μια συνάρτηση κίνησης κάνει publish, μέσω MQTT, τα δεδομένα κίνησης στο "robot\_motors\_topic" του message broker και με αυτόν τον τρόπο η εφαρμογή επικοινωνεί απευθείας με το ρομπότ, δεδομένου ότι έχει προηγηθεί η γεφύρωση του συγκεκριμένου topic.

---

<sup>56</sup> <https://www.rabbitmq.com/mqtt.html>





Εικόνα 38: Διάγραμμα εφαρμογής ελέγχου θερμοκρασίας σε περιβάλλον Node-RED

Συμπερασματικά, μέσω της παραπάνω εφαρμογής, παρουσιάζεται ένα σενάριο χρήσης το οποίο υλοποιεί την επικοινωνία και αλληλεπίδραση μιας συσκευής με ένα ρομπότ. Μέσω του συστήματος που αναπτύχθηκε στο πλαίσιο της παρούσας διπλωματικής, δίνεται η δυνατότητα για εύκολη και γρήγορη ανάπτυξη εφαρμογών που υλοποιούν την απευθείας επικοινωνία μεταξύ συσκευών, είτε αυτές αφορούν διαφόρων τύπου αισθητήρες είτε ρομπότ. Αν σε αυτό προστεθεί και το πλήθος διαφορετικών πρωτοκόλλων τα οποία μπορούν να συνδυαστούν σε κάθε περίπτωση γίνεται σαφές ότι διευρύνονται αρκετά τα όρια ανάπτυξης ρομποτικών εφαρμογών.

## 6 Συμπεράσματα και ανοιχτά θέματα

Στο παρόν κεφάλαιο παρουσιάζεται μια περιγραφή των συμπερασμάτων ως απόρροια της ενασχόλησης και εμβάθυνσης γύρω από το επιστημονικό πεδίο της διπλωματικής. Επίσης αναλύονται τα ανοιχτά θέματα τα οποία είναι δυνατόν να ενσωματωθούν στην παρούσα εφαρμογή, δίνοντας ένα ακόμα πιο ολοκληρωμένο αποτέλεσμα.

### 6.1 Συμπεράσματα

Το IoT (Internet of Things) εξελίσσεται με ραγδαίο ρυθμό και αναμένεται να αποτελέσει ένα από τα βασικότερα θέματα έρευνας τουλάχιστον για μια δεκαετία. Πολλές συσκευές έχουν αρχίσει να μετατρέπονται ήδη σε έξυπνα αντικείμενα καλύπτοντας πλέον ένα ευρύ φάσμα δραστηριοτήτων της καθημερινότητας. Η απομακρυσμένη διαχείριση και συλλογή δεδομένων, από διάφορα αντικείμενα αποτελεί μια από τις πιο υποσχόμενες τεχνολογικές καινοτομίες γύρω από τον τομέα της ανάπτυξης διαδικτυακών εφαρμογών.

Έξυπνες συσκευές θεωρούνται και τα ρομπότ, των οποίων η ευφυΐα και η μεταξύ τους αλληλεπίδραση τα κάνει ιδανικά αντικείμενα για το IoT. Εξάλλου, η ανάπτυξη εφαρμογών επικοινωνίας και διαχείρισης ρομποτικών συσκευών αποτελεί πλέον μια πραγματικότητα και ένα αναπόσπαστο κομμάτι της καθημερινότητας των ανθρώπων. Σ αυτή την κατεύθυνση το ROS αποτελεί το πλέον διαδεδομένο εργαλείο ανάπτυξης ρομποτικών εφαρμογών οι οποίες επιτρέπουν την αλληλεπίδραση και επικοινωνία των συσκευών. Ωστόσο, ένας περιορισμός ο οποίος εμφανίζεται στην δημιουργία εφαρμογών με βάση το ROS, είναι η απομακρυσμένη διαχείριση και επικοινωνία των συσκευών διαμέσου του διαδικτύου.

Το κενό λοιπόν που υπάρχει στην χρήση του ROS από συσκευές οι οποίες λειτουργούν απομακρυσμένα, έρχεται να διερευνήσει η παρούσα διπλωματική εργασία. Σ' αυτή την κατεύθυνση δημιουργήθηκε ένα σύστημα το οποίο επιτρέπει την γεφύρωση των διεπαφών του ROS με τον message broker RabbitMQ, με σκοπό την απομακρυσμένη αλληλεπίδραση μεταξύ πολλαπλών ρομπότ. Αναπτύχθηκε δηλαδή, ένας μηχανισμός γεφύρωσης ο οποίος τρέχει σε επίπεδο ρομποτικής συσκευής και ουσιαστικά επιτρέπει την επικοινωνία και ανταλλαγή δεδομένων των ρομπότ μέσω του RabbitMQ, προσομοιώνοντας την φιλοσοφία του IoT. Απόρροια όλων των παραπάνω είναι η ανάπτυξη μιας πλατφόρμας όπου διάφορες ρομποτικές συσκευές μπορούν να εγγραφούν και μέσω αυτής οι εφαρμογές να επικοινωνούν απευθείας με το ρομπότ, χρησιμοποιώντας διαφορετικά πρωτοκόλλα επικοινωνίας.

Συμπερασματικά, οποιοσδήποτε χρήστης με την χρησιμοποίηση της διαδικτυακής εφαρμογής, έχει την δυνατότητα να συνδέσει απομακρυσμένα τις συσκευές, είτε αυτές αφορούν ρομπότ είτε διάφορους αισθητήρες, συλλέγοντας

δεδομένα διαδικτυακά, εναρμονίζοντας έτσι την φιλοσοφία του IoT. Δίνεται λοιπόν η δυνατότητα για γεφύρωση της επικοινωνίας των ρομπότ, η οποία μπορεί να αφορά ROS topics ή services, και για συνεχή έλεγχο των διαφόρων πόρων που καταναλώνει η κάθε συσκευή, δυνατότητα η οποία αποτελεί αναπόσπαστο κομμάτι της διαχείρισης πολλαπλών συσκευών.

## 6.2 Ανοιχτά Θέματα

Παρακάτω, γίνεται μια παρουσίαση των ανοιχτών θεμάτων καθώς και των μελλοντικών επεκτάσεων, τα οποία ενσωματώνοντας τα στη παρούσα εφαρμογή είναι δυνατόν να οδηγήσουν σε ένα ακόμα πιο διευρυμένο αποτέλεσμα.

### **Γεφύρωση των ROS actions**

Στην παρούσα διπλωματική εργασία δημιουργήθηκε ο μηχανισμός γεφύρωσης των topics και services μεταξύ του ROS και του message broker RabbitMQ. Μια μελλοντική επέκταση αφορά την δημιουργία του μηχανισμού γεφύρωσης των ROS actions όπως αυτά αναλύθηκαν στο κεφάλαιο 4.2.

### **Ενσωμάτωση πολιτικών QoS (Quality of Service)**

Το QoS αφορά την μέτρηση της συνολικής απόδοσης μιας υπηρεσίας, όπως είναι η τηλεφωνία, το δίκτυο υπολογιστών ή μια υπηρεσία τύπου Cloud. Εστιάζει κυρίως στην απόδοση που βλέπουν οι χρήστες κατά την διάρκεια εκτέλεσης της υπηρεσίας. Μια επέκταση η οποία είναι δυνατόν να προστεθεί στην παρούσα εργασία είναι ο έλεγχος της ποιότητας των μηνυμάτων που γεφυρώνονται μεταξύ του ROS και του message broker αλλά και η χρονική τους καθυστέρηση. Για παράδειγμα ένα video το οποίο συλλέγεται από το ρομπότ και γίνεται publish στο RabbitMQ, είναι δυνατόν να καθυστερεί ή ακόμα η ποιότητά του να είναι χαμηλή.

### **Εξουσιοδότηση χρηστών ROS περιβάλλοντος**

Στην εκπόνηση της διπλωματικής αναπτύχθηκε τόσο ο μηχανισμός πιστοποίησης (authentication) των χρηστών και συσκευών όσο και ο μηχανισμός εξουσιοδότησης των συσκευών όπου οι χρήστες έχουν πρόσβαση σε συγκεκριμένες συσκευές. Μια επιπλέον προσθήκη θα μπορούσε να είναι και ο μηχανισμός εξουσιοδότησης του ROS περιβάλλοντος κάθε συσκευής, μέσω του οποίου ο χρήστης θα έχει την δυνατότητα πρόσβασης και μετέπειτα γεφύρωσης συγκεκριμένων topics, services ή actions.

## Βιβλιογραφία

[1] Luigi Atzori, Antonio Iera, Giacomo Morabito. "The Internet of Things: A survey", 2010.

[2] Borgia E." The Internet of Things vision: Key features, applications and open issues", Computer Communications, 2014.

[3] Tara Salman, Raj Jain." A Survey of Protocols and Standards for Internet of Things", Department of Computer Science and Engineering Washington University in St. Louis.

[4] Adrian McEwen, Hakim Cassimally. "Designing the Internet of things", Wiley and Sons Ltd, 2014.

[5] Rafiullah Khan, Sarmad Ullah Khan, Rifaqat Zaheer and Shahid Khan. "Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges", 2012.

[6] F. HUANG. "Web Technologies for the Internet of Things", Aalto University, 2013.

[7] V. Surwase. "REST API Modeling Languages –A Developer ' s Perspective", IJSTE -International Journal of Science Technology & Engineering, τόμ. 2, αρ. 10, 2016.

[8] A. Sanfeliu, N. Hagita, and A. Saffiotti, "Network Robot Systems, Robotics and Autonomous Systems", vol. 56, pp. 793–797, Oct. 2008. ISSN: 0921-8890, doi: 10.1016/j.robot.2008.06.007.

[9] A. R. Chowdhury. "IoT and Robotics: a synergy", PeerJ PrePrints, vol. 5, p. e2760, 2017.

[10] O. Vermesan, A. Bröring, E. Tragos, M. Serrano, D. Bacciu, S. Chessa, C. Gallicchio, A. Micheli, M. Dragone, A. Saffiotti, P. Simoens, F. Cavallo, and R. Bahr. "Internet of Robotic Things – Converging Sensing/Actuating, Hyperconnectivity, Artificial Intelligence and IoT Platforms ", June 2017. ISBN 9788793609105.

[11] P. P. Ray. "Internet of Robotic Things: Concept, Technologies, and Challenges", IEEE Access, vol. 4, pp. 9489–9500, 2016. doi: 10.1109/ACCESS.2017.2647747.

[12] Iovissa Johansson. "THE OPTIMAL RABBITMQ GUIDE from beginner to advanced", 2017.

[13] Anis Koubaa, Maram Alajlan and Basit Qureshi. "ROSLink: Bridging ROS with the

Internet-of-Things for Cloud Robotics", 2017.

[14] K. Martin and B. Hoffman. latency”Mastering CMake: “A cross-platform build system”, KitwareInc, 2008.

[15] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully B. Foote, Jeremy Leibs Rob Wheeler, Andrew Y. Ng. “ROS: an open-source robot operating system”, ICRA workshop on Open Source Software, 2019.

## Παράρτημα Α

Template το οποίο υλοποιεί γεφύρωση της επικοινωνίας ενός ROS publisher με τον message broker RabbitMQ:

```

1. import rospy
2. from amqp_protocol import PublisherSync
3. from amqp_protocol import amqpbase
4. from src.Conversions import ros_msg_to_dict
5. from amqp_protocol.publisher import PublisherSync
6. {% set msgPkg = ros_topic_type.split('/')[0] %}
7. {% set msgType = ros_topic_type.split('/')[1] %}
8. from {{ msgPkg }}.msg import {{ msgType }}
9.
10. class {{bridge_name}}(object):
11.
12.     def __init__(self):
13.         self.ros_topic = "{{ros_topic_name}}"
14.         self.amqp_topic = self.ros_topic.replace("/", ".")
15.         self.amqp_broker_host = "{{amqp_broker[\"host\"]}}"
16.         self.amqp_broker_port = "{{amqp_broker[\"port\"]}}"
17.         self.amqp_broker_username = "{{amqp_broker[\"username\"]}}"
18.         self.amqp_broker_password = "{{amqp_broker[\"password\"]}}"
19.         self.amqp_broker_vhost = "{{amqp_broker[\"vhost\"]}}"
20.         self.ros_message_type = {{msgType}}
21.         self.ros_node_name = "ros_topic{}_bridge".format(self.ros_topic.replace("/", "_"))
22.         self._debug = False
23.
24.         self.broker_conn_params = amqpbase.ConnectionParameters(host=self.amqp_broker_host, port=self.amqp_broker_port, vhost=self.amqp_broker_vhost)
25.
26.         self.broker_conn_params.credentials = amqpbase.Credentials(self.amqp_broker_username, self.amqp_broker_password)
27.
28.         self.broker_conn = amqpbase.SharedConnection(self.broker_conn_params)
29.
30.     @property
31.     def debug(self):
32.         return self._debug
33.
34.     @debug.setter
35.     def debug(self, val):
36.         self._debug = val
37.
38.     def run(self):
39.         self._init_ros_subscriber()
40.         self._init_rabbitmq_publisher()
41.         rospy.loginfo("bridge ros topic <{}> to amqp topic {}".format(self.ros_topic, self.amqp_topic))
42.         while not rospy.is_shutdown():
43.             pass
44.             # self.broker_conn.sleep(0.01)
45.
46.
47.
48.     def _init_ros_subscriber(self):
49.         if self._debug:

```

```

50.         log_level = rospy.DEBUG
51.     else:
52.         log_level = rospy.INFO
53.         rospy.init_node(self.ros_node_name,anonymous=True,log_level=log_level)
54.         rospy.Subscriber(self.ros_topic,self.ros_message_type,self._ros_callback)
55.         rospy.loginfo("ros subscriber for topic <{}> ready".format(self.ros_topic))
56.
57.
58.     def _init_rabbitmq_publisher(self):
59.         self.broker_pub = PublisherSync(self.amqp_topic, connection=self.broker_conn,debug=True)
60.         rospy.loginfo("rabbitmq publisher for topic <{}> ready".format(self.amqp_topic))
61.
62.
63.     def _ros_callback(self,msg):
64.         try:
65.             data = ros_msg_to_dict(msg)
66.         except Exception as e:
67.             rospy.logerr("ros msg conversion error <{}>".format(e))
68.             self._publish(data)
69.
70.     def _publish(self,data):
71.         if not isinstance(data,dict):
72.             raise ValueError("broker publish data should be type of dict")
73.         try:
74.             self.broker_pub.publish(data,thread_safe=True)
75.             rospy.loginfo("broker publish message")
76.         except Exception as e:
77.             rospy.logerr("exception <{}> while trying to publish message".format(e))
78.
79. if __name__ == '__main__':
80.     bridge = {{bridge_name}}()
81.     bridge.run()

```

Template το οποίο υλοποιεί γεφύρωση της επικοινωνίας ενός ROS publisher με τον message broker RabbitMQ:

```

1. import rospy
2. from amqp_protocol import Credentials,ConnectionParameters,SharedConnection,SubscriberSync
3. from src.Conversions import dict_to_ros_msg
4. {% set msgPkg = ros_topic_type.split('/')[0] %}
5. {% set msgType = ros_topic_type.split('/')[1] %}
6. from {{ msgPkg }}.msg import {{ msgType }}
7.
8. class {{bridge_name}}(object):
9.
10.     def __init__(self):
11.         self.ros_topic = "{{ros_topic_name}}"
12.         self.amqp_topic = "{{amqp_topic}}"
13.         self.amqp_broker_host = "{{amqp_broker[\"host\"]}}"
14.         self.amqp_broker_port = "{{amqp_broker[\"port\"]}}"
15.         self.amqp_broker_username = "{{amqp_broker[\"username\"]}}"
16.         self.amqp_broker_password = "{{amqp_broker[\"password\"]}}"

```

```
17.         self.amqp_broker_vhost = "{amqp_broker[vhost]}"
18.         self.ros_message_type = {msgType}
19.         self.ros_node_name = "broker_topic{}_bridge".format(self.amqp_topic.
replace(".", "_"))
20.         self._debug = False
21.
22.
23.         self.broker_conn_params = ConnectionParameters(host=self.amqp_broker
_host,port = self.amqp_broker_port, vhost= self.amqp_broker_vhost)
24.
25.         self.broker_conn_params.credentials = Credentials(username=self.amqp
_broker_username,password = self.amqp_broker_password)
26.
27.         self.broker_conn = SharedConnection(self.broker_conn_params)
28.
29.         @property
30.         def debug(self):
31.             return self._debug
32.
33.         @debug.setter
34.         def debug(self,val):
35.             self._debug = val
36.
37.         def run(self):
38.             self._init_rabbitmq_subscriber()
39.             self._init_ros_publisher()
40.             rospy.loginfo("bridge rabbitmq topic <{}> to ros topic {}".format(se
lf.amqp_topic,self.ros_topic))
41.             while not rospy.is_shutdown():
42.                 rospy.sleep(0.01)
43.
44.         def _init_rabbitmq_subscriber(self):
45.             self.sub = SubscriberSync(self.amqp_topic,on_message=self._rabbitmq
callback,
46.                                     connection=self.broker_conn,queue_size=20,
debug=self.debug)
47.
48.             rospy.loginfo("rabbitmq subscriber to topic {} ready".format(self.am
qp_topic))
49.             self.sub.run_threaded()
50.
51.
52.         def _init_ros_publisher(self):
53.             rospy.init_node(self.ros_node_name)
54.             self.ros_pub = rospy.Publisher(self.ros_topic,self.ros_message_type,
queue_size=20)
55.             rospy.loginfo("ros publisher to topic {} ready".format(self.ros_top
ic))
56.
57.         def _rabbitmq_callback(self,msg,meta):
58.             try:
59.                 ros_msg = dict_to_ros_msg("{ros_topic_type}",msg)
60.                 self.ros_pub.publish(ros_msg)
61.             except Exception as e:
62.                 rospy.logerr("could not convert dict to ros message".format(str(
e)))
63.
64.         if __name__ == "__main__":
65.             bridge = {bridge_name}()
66.             bridge.run()
```



Template το οποίο υλοποιεί γεφύρωση της επικοινωνίας ενός ROS service με τον message broker RabbitMQ:

```

1. import rospy
2. from amqp_protocol import RpcServer, RpcClient, Credentials, ConnectionParameters, SharedConnection
3. from src.Conversions import ros_srv_req_to_dict, ros_srv_resp_to_dict, dict_to_ros_srv_request
4. {% set srvPkg = ros_srv_type.split('/')[0] %}
5. {% set srvType = ros_srv_type.split('/')[1] %}
6. from {{ srvPkg }}.srv import {{ srvType }}, {{ srvType }}Request, {{ srvType }}Response
7.
8. class {{ bridge_name }}(object):
9.
10.     def __init__(self, connection=None):
11.         self.ros_service_uri = '{{ ros_srv_name }}'
12.         self.amqp_rpc_name = '{{ amqp_rpc_name }}'
13.         self.amqp_broker_ip = "{{amqp_broker[\"host\"]}}"
14.         self.amqp_broker_port = "{{amqp_broker[\"port\"]}}"
15.         self.amqp_broker_vhost = "{{amqp_broker[\"vhost\"]}}"
16.         self.username = "{{amqp_broker[\"username\"]}}"
17.         self.password = "{{amqp_broker[\"password\"]}}"
18.         self.rpc_name = self.amqp_rpc_name
19.         self.ros_service_type = {{ srvType }}
20.         self.ros_srv_type_str = '{{ ros_srv_type }}'
21.         self.ros_node_name = "ros_service{}_bridge".format(self.ros_service_uri.replace("/", "_"))
22.         if connection:
23.             self.broker_conn = connection
24.             return
25.         self.conn_params = ConnectionParameters(
26.             vhost=self.amqp_broker_vhost,
27.             host=self.amqp_broker_ip,
28.             port=self.amqp_broker_port)
29.         self.conn_params.credentials = Credentials(
30.             self.username, self.password)
31.
32.         self.broker_conn = SharedConnection(self.conn_params)
33.
34.     def run(self):
35.         self._init_rpc_server()
36.         self._init_ros_service()
37.         self.rpc_server.run_threaded()
38.
39.     def _init_rpc_server(self):
40.         self.rpc_server = RpcServer(self.amqp_rpc_name, on_request=self._rpc_server_callback, connection=self.broker_conn)
41.
42.     def _init_ros_service(self):
43.
44.         rospy.init_node(self.ros_node_name, anonymous=True)
45.         rospy.loginfo("waiting for ros service {} response".format(self.ros_service_uri))
46.         rospy.wait_for_service(self.ros_service_uri)
47.         self.ros_srv = rospy.ServiceProxy(self.ros_service_uri, self.ros_service_type)
48.         rospy.loginfo("ros service client ready")
49.
50.     def _rpc_server_callback(self, msg, meta):
51.         try:
52.             srv_req = dict_to_ros_srv_request(self.ros_srv_type_str, msg)

```

```
53.         resp = self.ros_srv(srv_req)
54.     except rospy.ServiceException as exc:
55.         rospy.logerr('ROS Service call failed: {}'.format(exc))
56.         resp = {{srvType}}Response()
57.         data = ros_srv_resp_to_dict(resp)
58.         return data
59.
60. if __name__ == "__main__":
61.     bridge = {{bridge_name}}()
62.     bridge.run()
```

