



Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών &
Μηχανικών Υπολογιστών
Τομέας Ηλεκτρονικής και Υπολογιστών
Εργαστήριο Ευφών Συστημάτων και Τεχνολογίας
Λογισμικού (ISSEL)

Διπλωματική Εργασία

Μοντελοστρεφής ανάπτυξη λογισμικού για IoT
συσκευές πραγματικού χρόνου και χαμηλής
κατανάλωσης

Εκπόνηση:
Αθανάσιος Μανώλης
ΑΕΜ: 8856

Επίβλεψη:
Αναπληρωτής Καθηγητής
Ανδρέας Λ. Συμεωνίδης
Υποψήφιος Διδάκτορας
Κωνσταντίνος Παναγιώτου

Θεσσαλονίκη, Σεπτέμβριος 2021

Quote 1
— *Quoter 1*

Quote 2
— *Quoter 2*

ΕΥΧΑΡΙΣΤΙΕΣ

Πρώτο από όλους θα ήθελα να ευχαριστήσω τον Κωνσταντίνο Παναγιώτου, ο οποίος με βοήθησε καθόλη τη διάρκεια της διπλωματικής, με κατεύθυνε όπως έπρεπε και ήταν πάντα διαθέσιμος για να με βοηθάει σε προβλήματα που αντιμετώπιζα. Επίσης θα ήθελα να ευχαριστήσω τον κ. Ανδρέα Συμεωνίδη που μου έδωσε τη δυνατότητα να αναλάβω αυτή τη διπλωματική και να ασχοληθώ με το αντικείμενο που πάντα με ενδιέφερε, αλλά και τους Μάνο Τσαρδούλια και Αλέξανδρο Φιλοθέου που μου παρείχαν τις γνώσεις τους στον τομέα της ρομποτικής, και μπόρεσα μέσα από τα πλαίσια της ομάδας R4A να γνωρίσω και να ασχοληθώ με ένα εξίσου ενδιαφέρον αντικείμενο.

Πέραν των επιστημονικών συνεργατών, θα ήθελα να ευχαριστήσω τους φίλους και τις φίλες μου για τη συμπαράστασή τους και για όλες τις όμορφες στιγμές που ζήσαμε στην περίοδο των φοιτητικών μας χρόνων, αλλά και τους γονείς μου, Βαγγέλη και Ζωή, και την αδερφή μου, Ευαγγελία, για την στήριξη και τα εφόδια που μου έδωσαν και που ήταν στο πλευρό μου όποτε τους χρειαζόμουν.

Περίληψη

Το διαδίκτυο των πραγμάτων (Internet of Things ή IoT) είναι ένας κλάδος που εξελίσσεται ραγδαία ειδικά τα τελευταία χρόνια. Άρα υπάρχει η δυνατότητα ανάπτυξης όλο και περισσότερων εφαρμογών, χρήσιμες για πολλούς ανθρώπους, είτε έχουν να κάνουν με απλές λειτουργίες σε συστήματα αυτοματισμού, είτε με μεγαλύτερης κλίμακας εφαρμογές στη βιομηχανία. Επομένως, όλο και περισσότερος κόσμος επιθυμεί να ασχοληθεί με τον τομέα αυτό, και μια μεγάλη μερίδα του είναι άτομα ακατάρτιστα. Αυτό έχει ως αποτέλεσμα να χάνουν της δυνατότητες που το IoT μπορεί να προσφέρει.

Η μοντελοστρεφής μηχανική (Model Driven Engineering ή MDE), έρχεται να δώσει λύσει σε αυτό το πρόβλημα, καθώς μπορεί να παρέχει σε αυτά τα άτομα την ανάπτυξη IoT συστημάτων σε ένα πιο αφαιρετικό επίπεδο, το οποίο είναι πιο φιλικό προς τον απλό χρήστη.

Μέσω της παρούσας διπλωματικής εργασίας, δίνεται η δυνατότητα σε κάποιον χρήστη να μοντελοποιήσει το σύστημα που επιθυμεί να υλοποιήσει, μέσω ενός εργαλείου κειμένου περιγραφής συσκευών και των μεταξύ τους συνδέσεων. Ταυτόχρονα, παρέχει την αυτόματη παραγωγή κώδικα για μια πληθώρα IoT συσκευών, προσαρμοσμένη στα χαρακτηριστικά που επιθυμεί ο χρήστης να έχει το σύστημά του, και έτσι του δίνεται έτοιμη μια βασική υλοποίηση κάποιων λειτουργιών, χωρίς να χρειάζεται να γράφει καθόλου κώδικα. Μάλιστα, ο κώδικας που παράγεται είναι χαμηλού επιπέδου, καθώς έχει σχεδιαστεί σύμφωνα με τις απαιτήσεις ενός λειτουργικού συστήματος πραγματικού χρόνου (Real Time Operating System ή RTOS), το RIOT.

Title

Model-driven development for low-consumption real-time IOT devices

Abstract

Athanasios Manolis
Intelligent Systems and Software Engineering Labgroup (ISSEL)
Electrical & Computer Engineering Department,
Aristotle University of Thessaloniki, Greece
September 2021

Internet of Things (IoT) is a field that is evolving rapidly, especially in recent years. Therefore there is a possibility of developing even more applications which prove to be useful for many people, whether they have to do with simple functions in automation systems, or with larger scale applications in the industry. For that reason, more and more people want to work in this field, and a large portion of them do not have the technical knowledge to do so. As a result, they end up losing all the potential that IoT can offer.

Model Driven Engineering (MDE), is here to solve this problem, as it can provide these individuals with the development of IoT systems at a more abstract level, which is way more user-friendly.

Through this diploma thesis, a user is given the opportunity to model the system they want to implement, using a text tool to describe the devices and their connections. At the same time, it offers automatic source code generation for a variety of IoT devices, tailored to the features the user prefers their system to have, and thus provides a basic implementation of some functions, without having to write any code whatsoever. In fact, the generated code is low-level, as it is designed according to the requirements of a Real Time Operating System (RTOS), named RIOT.

Περιεχόμενα

Ευχαριστίες	iii
Περίληψη	v
Abstract	vii
Ακρωνύμια	xiv
1 Εισαγωγή	1
1.1 Περιγραφή του Προβλήματος	1
1.2 Σκοπός - Συνεισφορά της Διπλωματικής Εργασίας	2
1.3 Διάρθρωση της Αναφοράς	2
2 Θεωρητικό Υπόβαθρο	4
2.1 Διαδίκτυο των Πραγμάτων	4
2.1.1 Δομή του IoT	4
2.1.2 Πρωτόκολλο Επικοινωνίας Δεδομένων MQTT	6
2.2 Model Driven Engineering	7
2.2.1 Μοντέλα και μετα-μοντέλα	8
2.2.2 Γλώσσες μοντελοποίησης	9
2.2.3 Μετασχηματισμοί	10
2.3 Λειτουργικά Συστήματα Πραγματικού χρόνου	10
2.4 Πρωτόκολλα επικοινωνίας	11
2.4.1 I2C	11
2.4.2 SPI	12
2.4.3 UART	12
3 Επισκόπηση της Ερευνητικής Περιοχής	14
4 Εργαλεία	16
4.1 RIOT	16
4.2 textX	17
4.3 Jinja	17
4.4 PlantUML	18
5 Μεθοδολογία και Υλοποίηση	20
5.1 Ορισμός μετα-μοντέλου Συσσκευής	21
5.1.1 Device	22
5.1.2 Board	22
5.1.3 Peripheral	23
5.1.4 CPU	24

5.1.5	NETWORK	25
5.1.6	WIFI	26
5.1.7	ETHERNET	27
5.1.8	MEMORY	27
5.1.9	BLEUTOOTH	28
5.1.10	PIN	29
5.1.11	POWER_PIN	29
5.1.12	INPUT_PIN	30
5.1.13	OUTPUT_PIN	31
5.1.14	IO_PIN	32
5.1.15	PIN_FUNC	33
5.1.16	GPIO	34
5.1.17	I2C	34
5.1.18	SPI	35
5.1.19	UART	36
5.1.20	PWM	37
5.1.21	ADC	38
5.1.22	DAC	39
5.2	Ορισμός μετα-μοντέλου Συνδέσεων	39
5.2.1	SYSTEM	40
5.2.2	INCLUDE	41
5.2.3	CONNECTION	41
5.2.4	PERIPHERAL	42
5.2.5	BOARD	43
5.2.6	POWER_CONNECTION	44
5.2.7	COM_ENDPOINT	44
5.2.8	MSG_ENTRIES	46
5.2.9	HW_CONNECTION	47
5.2.10	GPIO	47
5.2.11	I2C	48
5.2.12	SPI	48
5.2.13	UART	49
5.2.14	FREQUENCY	50
5.3	Γραμματική των DSL	51
5.3.1	Γενικό Συντακτικό	51
5.3.2	Συντακτικό Συσκευής	52
5.3.3	Συντακτικό Συνδέσεων	56
5.4	Μετασχηματισμός M2M	59
5.5	Παραγωγή κώδικα	60
5.5.1	Αρχείο κώδικα C	60
5.5.2	Αρχείο Makefile	62
5.6	Υποστηριζόμενες συσκευές	62

6	Παραδείγματα	64
6.1	Παραδείγματα χρήσης	64
6.2	Εφαρμογή με 2 σένσορες και έναν ενεργοποιητή	64
6.3	Νέο περιφερειακό	67
6.4	Νέος μικροελεγκτής	69
7	Συμπεράσματα και Μελλοντικές επεκτάσεις	72
7.1	Συμπεράσματα	72
7.2	Μελλοντικές επεκτάσεις	73
	Βιβλιογραφία	74
	Παραρτήματα	76
	Παράρτημα Α' Πρότυπα αρχεία παραγωγής κώδικα	77
	Παράρτημα Β' Διαγράμματα	92

Κατάλογος Σχημάτων

2.1	Επίπεδα αρχιτεκτονικής του IoT.	5
2.2	Ορισμός του συστήματος.	8
2.3	Ορισμός του μοντέλου.	9
2.4	Ορισμός του μετα-μοντέλου.	9
2.5	I2C	12
2.6	SPI	13
2.7	UART	13
4.1	Λειτουργικό σύστημα RIOT.	17
4.2	textX	17
4.3	Jinja	18
5.1	Μετα-μοντέλο συσκευής	21
5.2	Μετα-μοντέλο συνδέσεων	40
6.1	Συνδεσμολογία μεταξύ των συσκευών.	66
6.2	Μήνυμα για συσκευή που δεν υποστηρίζεται	67
6.3	Μήνυμα για περιφερειακό για το οποίο δεν υπάρχει το πρότυπο αρχείο.	68
6.4	Μήνυμα για συσκευή που δεν υποστηρίζεται	69
B'.1	Συνδεσμολογία μεταξύ των συσκευών	92
B'.2	Χαρακτηριστικά των συνδέσεων	93

Κατάλογος πινάκων

5.1	Ιδιότητες και Συσχετίσεις του <i>Board</i> .	23
5.2	Ιδιότητες και Συσχετίσεις του <i>Peripheral</i> .	24
5.3	Ιδιότητες του <i>CPU</i> .	25
5.4	Ιδιότητες και Συσχετίσεις του <i>WIFI</i> .	26
5.5	Ιδιότητες και Συσχετίσεις του <i>ETHERNET</i> .	27
5.6	Ιδιότητες του <i>MEMORY</i> .	28
5.7	Ιδιότητες του <i>BLEUTOOTH</i> .	28
5.8	Ιδιότητες και Συσχετίσεις του <i>POWER_PIN</i> .	30
5.9	Ιδιότητες και Συσχετίσεις του <i>INPUT_PIN</i> .	31
5.10	Ιδιότητες και Συσχετίσεις του <i>OUTPUT_PIN</i> .	32
5.11	Ιδιότητες και Συσχετίσεις του <i>IO_PIN</i> .	33
5.12	Ιδιότητες και Συσχετίσεις του <i>GPIO</i> .	34
5.13	Ιδιότητες και Συσχετίσεις του <i>I2C</i> .	35
5.14	Ιδιότητες και Συσχετίσεις του <i>SPI</i> .	36
5.15	Ιδιότητες και Συσχετίσεις του <i>UART</i> .	37
5.16	Ιδιότητες και Συσχετίσεις του <i>PWM</i> .	38
5.17	Ιδιότητες και Συσχετίσεις του <i>ADC</i> .	38
5.18	Ιδιότητες και Συσχετίσεις του <i>DAC</i> .	39
5.19	Συσχετίσεις του <i>SYSTEM</i> .	40
5.20	Ιδιότητες του <i>INCLUDE</i> .	41
5.21	Ιδιότητες και Συσχετίσεις του <i>CONNECTION</i> .	42
5.22	Ιδιότητες του <i>PERIPHERAL</i> .	43
5.23	Ιδιότητες του <i>BOARD</i> .	43
5.24	Ιδιότητες του <i>POWER_CONNECTION</i> .	44
5.25	Ιδιότητες και Συσχετίσεις του <i>COM_ENDPOINT</i> .	45
5.26	Ιδιότητες του <i>MSG_ENTRIES</i> .	46
5.27	Ιδιότητες και Συσχετίσεις του <i>GPIO</i> .	47
5.28	Ιδιότητες και Συσχετίσεις του <i>I2C</i> .	48
5.29	Ιδιότητες και Συσχετίσεις του <i>SPI</i> .	49
5.30	Ιδιότητες και Συσχετίσεις του <i>UART</i> .	50
5.31	Ιδιότητες του <i>FREQUENCY</i> .	51

Ακρωνύμια Εγγράφου

Παρακάτω παρατίθενται ορισμένα από τα πιο συχνά χρησιμοποιούμενα ακρωνύμια της παρούσας διπλωματικής εργασίας:

MDE	→	Model-Driven Engineering
DSL	→	Domain Specific Language
M2M	→	Model to Model transformation
M2T	→	Model to Text transformation
IoT	→	Internet of Things
RTOS	→	Real Time Operating System
MQTT	→	Message Queuing Telemetry Transport

1

Εισαγωγή

Αν και υπήρχε ως ιδέα εδώ και περίπου 50 χρόνια, το Διαδίκτυο των Πραγμάτων (Internet of Things - IoT), και ως όρος αλλά και ως προς τη χρήση του, ήρθε στο επίκεντρο του ενδιαφέροντος τα τελευταία 10 χρόνια, και καθημερινά γίνεται ολοένα και πιο διαδεδομένο. Πλέον μάλιστα ο συνολικός αριθμός συνδεδεμένων συσκευών στο διαδίκτυο είναι μεγαλύτερος από αυτόν του πληθυσμού της Γης.

Μπορεί κανείς να δει την εφαρμογή του σε πολλούς τομείς. Από την δημιουργία ενός "Εξυπνου Σπιτιού" μέχρι και σε κάτι τόσο ουσιώδες όπως την καλύτερη παρακολούθηση ασθενών σε νοσοκομεία και άρα την πιο σωστή περίθαλψή τους. Επίσης χρησιμοποιείται και για αυτοματισμούς στη γεωργία, και γενικότερα στη βιομηχανία.

Η ολοένα και μεγαλύτερη διάδοση του IoT, συνεπάγεται και την αξιοποίησή του από κοινό λιγότερο τεχνολογικά καταρτισμένο. Κρίνεται σκόπιμη λοιπόν η ανάπτυξη μεθόδων που θα μετατρέπουν την δημιουργία ενός συστήματος IoT σε διαδικασία πιο φιλική προς τα άτομα αυτά. Εδώ έρχεται να δώσει τη λύση η Μοντελοστρεφής Μηχανική (Model Driven Engineering - MDE), η οποία προσφέρει γρήγορη και πιο αυτοματοποιημένη ανάπτυξη λογισμικού.

1.1 ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ

Υπάρχει πληθώρα IoT συσκευών στην αγορά, εύκολα διαχειρίσιμες από όλον τον κόσμο, ανεξαρτήτως των γνώσεών του στον τομέα. Μάλιστα οι εμπορικές αυτές συσκευές προσφέρουν πολλές δυνατότητες και λειτουργίες, και άρα μπορεί ο/η καθένας/καθεμία να τις χρησιμοποιήσει και να καλυφθεί από αυτό.

Στην περίπτωση όμως που κάποιος/α επιθυμεί να αναπτύξει ένα σύστημα με IoT συσκευές από την αρχή, επειδή π.χ. θέλει να πειραματιστεί ή να υλοποιήσει κάποιες λειτουργίες πιο εξειδικευμένες, τότε απαιτείται μια μεγάλη διαδικασία για την κατασκευή και άρα πολλές γνώσεις. Το πρώτο βήμα είναι η επιλογή των κατάλληλων μικροελεγκτών, αισθητήρων, ενεργοποιητών για την υλοποίηση της ιδέας. Απαιτείται λοιπόν γνώση πάνω στον τρόπο λειτουργίας των συσκευών αυτών, καθώς και στον τρόπο διασύνδεσης και επικοινωνίας τους. Ακολουθεί η ανάπτυξη λογισμικού για την υλοποίηση των επιθυμητών λειτουργιών, κάτι το οποίο από μόνο του σημαίνει πως πρέπει να υπάρχει εμπειρία με χαμηλού επιπέδου προγραμματισμό και πρωτόκολλα επικοινωνίας.

1.2 ΣΚΟΠΟΣ - ΣΥΝΕΙΣΦΟΡΑ ΤΗΣ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Η παρούσα διπλωματική έχει ως στόχο την ανάπτυξη μιας μηχανής λογισμικού μοντελοστρεφούς λογικής, με την οποία οι χρήστες θα μπορούν να μοντελοποιούν συσκευές καθώς και την σύνδεσή τους. Στη συνέχεια το σύστημα θα υποδεικνύει τη σωστή συνδεσμολογία, και θα παράγονται αυτόματα κάποια προγράμματα που θα υλοποιούν κάποιες βασικές λειτουργίες. Ο παραγόμενος κώδικας θα αφορά συσκευές που τρέχουν το λειτουργικό σύστημα πραγματικού χρόνου RIOT.

1.3 ΔΙΑΡΘΡΩΣΗ ΤΗΣ ΑΝΑΦΟΡΑΣ

Η διάρθρωση της παρούσας διπλωματικής εργασίας είναι η εξής:

- **Κεφάλαιο 2:** Αναλύεται το θεωρητικό υπόβαθρο.
- **Κεφάλαιο 3:** Γίνεται ανασκόπηση της ερευνητικής δραστηριότητας στον τομέα μέχρι σήμερα.
- **Κεφάλαιο 4:** Παρουσιάζονται οι διάφορες τεχνικές και τα εργαλεία που χρησιμοποιήθηκαν στις υλοποιήσεις.
- **Κεφάλαιο 5:** Παρουσιάζονται τα βήματα της μεθοδολογίας που υλοποιήθηκε.
- **Κεφάλαιο 6:** Περιγράφονται 3 παραδείγματα εφαρμογής των εργαλείων που αναπτύχθηκαν.
- **Κεφάλαιο 7:** Παρουσιάζονται τα τελικά συμπεράσματα και προτείνονται θέματα για μελλοντική μελέτη, αλλαγές και επεκτάσεις.

2

Θεωρητικό Υπόβαθρο

2.1 ΔΙΑΔΙΚΤΥΟ ΤΩΝ ΠΡΑΓΜΑΤΩΝ

Το Διαδίκτυο των Πραγμάτων (*Internet of Things* ή *IoT*) περιγράφει το δίκτυο φυσικών αντικειμένων - "πραγμάτων" - που είναι ενσωματωμένα με αισθητήρες, λογισμικό και άλλες τεχνολογίες με σκοπό τη σύνδεση και την ανταλλαγή δεδομένων με άλλες συσκευές και συστήματα μέσω του διαδικτύου. Αυτές οι συσκευές μπορεί να είναι συνηθισμένα οικιακά αντικείμενα, ή και εξελιγμένα βιομηχανική εργαλεία. Με περισσότερες από 7 δισεκατομμύρια συνδεδεμένες συσκευές IoT σήμερα, οι ειδικοί αναμένουν ότι ο αριθμός αυτός θα αυξηθεί σε 10 δισεκατομμύρια έως το 2020 και 22 δισεκατομμύρια έως το 2025.

Μιας και δεν υπάρχει κάποιος συγκεκριμένος ορισμός για το IoT, αξίζει να παραθέσουμε τον ορισμό που έχει δοθεί και από το Ινστιτούτο Ηλεκτρολόγων και Ηλεκτρονικών Μηχανικών (IEEE) :

“Ένα δίκτυο αντικειμένων -το καθένα ενσωματωμένο με αισθητήρες- τα οποία είναι συνδεδεμένα στο διαδίκτυο” [1]

2.1.1 Δομή του IoT

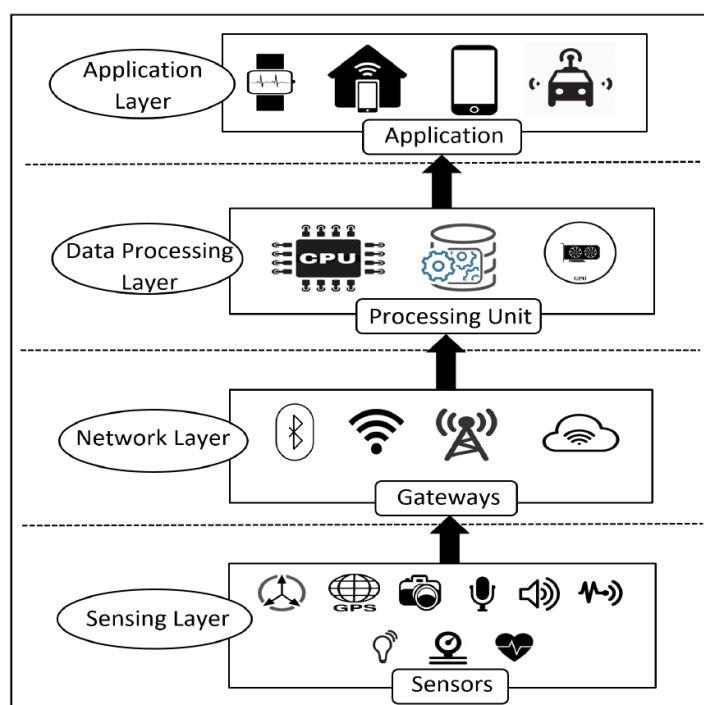
Η μονάδα υλικού σε ένα σύστημα IoT αποτελείται από τις ακόλουθες κατηγορίες:

- Αισθητήρες/ενεργοποιητές

- Μονάδες επεξεργασίας
- Μονάδες αποθήκευσης
- Μονάδες επικοινωνίας

Έχοντας προσδιορίσει της κατηγορίες του υλικού, πρέπει να προσθέσουμε το λογισμικό, το ενδιάμεσο λογισμικό (middleware) και τα απαραίτητα πρωτόκολλα τα οποία παρέχουν τη δυνατότητα για να ενώσουμε όλα αυτά τα στοιχεία μεταξύ τους με σκοπό να συγκροτούν ένα πλήρως λειτουργικό σύστημα [4].

Σε αυτήν την ενότητα, παρουσιάζεται η αρχιτεκτονική του IoT, το οποίο αποτελείται από τέσσερα επίπεδα (επίπεδο αίσθησης, επίπεδο δικτύου, επίπεδο επεξεργασίας δεδομένων, επίπεδο εφαρμογών) όπως φαίνεται και στο [σχήμα 2.1](#) [2].



Σχήμα 2.1: Επίπεδα αρχιτεκτονικής του IoT.

Επίπεδο Αίσθησης

Ο κύριος σκοπός του επιπέδου αίσθησης είναι να ανιχνεύσει οποιαδήποτε φαινόμενα στο περιβάλλον των συσκευών και να λάβει δεδομένα από τον πραγματικό κόσμο. Το επίπεδο αυτό αποτελείται από διάφορους αισθητήρες, οι οποίοι ανήκουν στις ακόλουθες κατηγορίες:

1. **Αισθητήρες κίνησης:** Μετρούν την αλλαγή στην κίνηση καθώς και τον προσανατολισμό των συσκευών.
2. **Αισθητήρες περιβάλλοντος:** Αισθητήρες φωτός, πίεσης, θερμοκρασίας κ.α. αντιλαμβάνονται την αλλαγή σε περιβαλλοντικές παραμέτρους.

3. *Αισθητήρες θέσης*: Είναι υπεύθυνοι για την εύρεση της φυσικής θέσης και τοποθεσίας της συσκευής (π.χ. μαγνητικοί σένσορες, GPS κ.α.).

Επίπεδο Δικτύου

Το επίπεδο αυτό δρα ως κανάλι επικοινωνίας για τη μεταφορά δεδομένων, τα οποία συλλέχθηκαν στο επίπεδο αίσθησης, σε άλλες συνδεδεμένες συσκευές. Στις IoT συσκευές το επίπεδο δικτύου υλοποιείται με τη χρήση ποικίλων τεχνολογιών επικοινωνίας, όπως Wi-Fi, Bluetooth, Zegbee, Z-Wave, LoRa κ.α.

Επίπεδο Επεξεργασίας Δεδομένων

Το επίπεδο επεξεργασίας δεδομένων αποτελείται από την κεντρική μονάδα επεξεργασίας των IoT συσκευών. Λαμβάνει τα δεδομένα από το επίπεδο αίσθησης, και τα αναλύει με σκοπό να πάρει αποφάσεις βάση του αποτελέσματος. Σε κάποιες IoT συσκευές, το επίπεδο αυτό αποθηκεύει το αποτέλεσμα προηγούμενων αναλύσεων με σκοπό τη βελτίωση της εμπειρίας χρήστη. Τέλος, μπορεί να μοιραστεί τα αποτελέσματα των αναλύσεων με άλλες συνδεδεμένες συσκευές μέσω του επιπέδου δικτύου.

Επίπεδο Εφαρμογών

Το τέταρτο και τελευταίο επίπεδο, υλοποιεί και παρουσιάζει τα αποτελέσματα του επιπέδου επεξεργασίας δεδομένων ώστε να εκτελέσει διάφορες λειτουργίες των IoT συσκευών.

2.1.2 Πρωτόκολλο Επικοινωνίας Δεδομένων MQTT

Υπάρχουν διάφορα πρωτόκολλα επικοινωνίας δεδομένων τα οποία χρησιμοποιούνται στο IoT, ωστόσο στην παρούσα διπλωματική εργασία χρησιμοποιήθηκε το MQTT-SN (*Message Queuing Telemetry Transport for Sensor Networks*) το οποίο είναι μια διαφορετική έκδοση του MQTT¹.

Ένα πολύ γνωστό παράδειγμα για επικοινωνία δεδομένων είναι το σύστημα "Publish/Subscribe", το οποίο θα μπορούσε να μεταφραστεί ως "Κοινοποίηση/Εγγραφή". Κατά τη λειτουργία ενός τέτοιου συστήματος, ο αποστολέας (publisher) δεν επικοινωνεί απευθείας με τον παραλήπτη (subscriber), αλλά κοινοποιεί τα μηνύματά του σε συγκεκριμένα τερματικά (topics), στα οποία ο παραλήπτης μπορεί

¹<https://mqtt.org/>

να εγγραφεί και να λάβει τα μηνύματα που τους κοινοποιούνται. Υπεύθυνος για τη λήψη και διαμοιρασμό των μηνυμάτων, είναι ένας μεσολαβητής (*broker*).

Το πρότυπο αυτό μπορεί να υλοποιηθεί μέσω του πρωτοκόλλου MQTT. Το MQTT-SN που χρησιμοποιήθηκε στην εργασία, σχεδιάστηκε ώστε να είναι όσο το δυνατότερο όμοιο με το MQTT, αλλά είναι προσαρμοσμένο στις ιδιαιτερότητες ενός ασύρματου περιβάλλοντος επικοινωνίας, όπως είναι το χαμηλό εύρος ζώνης (*bandwidth*), οι υψηλές αποτυχίες συνδέσεων, το μικρό μήκος μηνύματος κ.α.

2.2 MODEL DRIVEN ENGINEERING

Η Μοντελοστρεφής Μηχανική (*Model Driven Engineering* ή *MDE*) είναι μια διαδικασία ανάπτυξης λογισμικού. Είναι η πρακτική της ανάπτυξης μοντέλων τα οποία περιγράφουν, αναλύουν τα χαρακτηριστικά ενός συστήματος, και μέσω αυτών, χρησιμοποιώντας το κατάλληλο λογισμικό, αυτοματοποιείται η διαδικασία παραγωγής της εκάστοτε υλοποίησης, ή έστω ένα κομμάτι της [3]. Τα μοντέλα αυτά είναι κατασκευασμένα πάντα με βάση τη γραμματική κάποιου μετα-μοντέλου που τα περιγράφει.

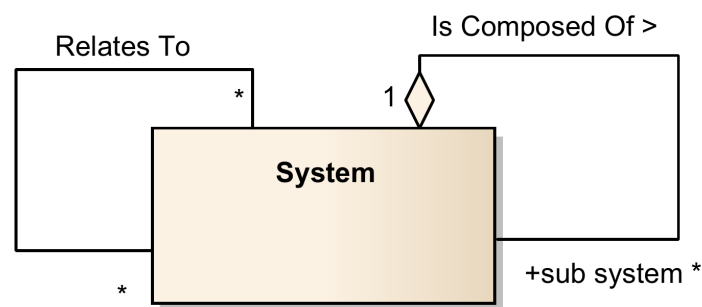
Τα μοντέλα είχαν και έχουν κεντρική σημασία σε πολλά επιστημονικά πεδία. Συγκεκριμένα για τον τομέα της τεχνολογίας λογισμικού, αποτελούν ένα σημαντικό εργαλείο για την καλύτερη αντίληψη και την ανταλλαγή γνώσεων για πολύπλοκα συστήματα. Η MDE έχει σχεδιαστεί ως ένα εργαλείο που αξιοποιεί τον παραπάνω ισχυρισμό. Μέσω αυτής, δίνεται η δυνατότητα στον/στην σχεδιαστή/ρια να πειραματιστεί σε κάποιον συγκεκριμένο κλάδο χωρίς να χρειάζεται γνώσεις πέρα από αυτό. Για παράδειγμα, για τον σχεδιασμό ενός συστήματος *IoT*, μπορεί κάποιος να προγραμματίσει τις συσκευές που έχει στην κατοχή του χωρίς να χρειάζεται να γνωρίζει π.χ. το λειτουργικό σύστημα στο οποίο αυτό θα λειτουργεί, παρά μόνο σχεδιάζοντας το μοντέλο του συστήματος που επιθυμεί, δηλώνοντας τις δυνατότητες και τα χαρακτηριστικά του. Αφού γίνει αυτό, ένα ενδιάμεσο λογισμικό θα μετατρέψει το μοντέλο σε εκτελέσιμο κώδικα, ο οποίος θα υλοποιεί συγκεκριμένες λειτουργίες.

Για να γίνουν η χρησιμότητα και ο σκοπός της MDE πλήρως αντιληπτά από τον/την αναγνώστη/ρια, σε αυτήν την ενότητα παρουσιάζονται οι βασικές έννοιες και ορισμοί της.

2.2.1 Μοντέλα και μετα-μοντέλα

Σύστημα

Στο πλαίσιο της MDE το Σύστημα ορίζεται ως μια γενική έννοια για τον προσδιορισμό μιας εφαρμογής λογισμικού, μιας πλατφόρμας λογισμικού ή οποιουδήποτε άλλου τεχνουργήματος λογισμικού. Επιπλέον, όπως φαίνεται στο [σχήμα 2.2](#), ένα σύστημα μπορεί να αποτελείται από άλλα υποσυστήματα και μπορεί να έχει σχέσεις με άλλα συστήματα (π.χ. να επικοινωνούν) [4].



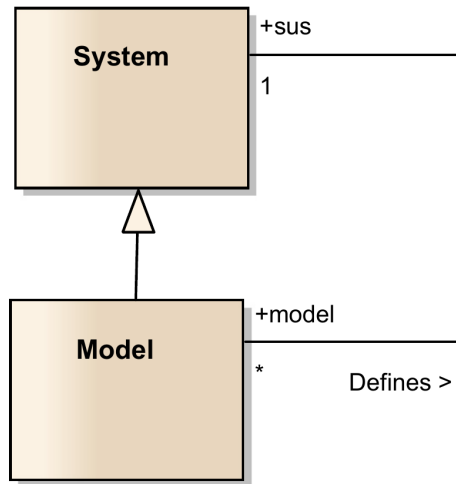
Σχήμα 2.2: Ορισμός του συστήματος.

Μοντέλο

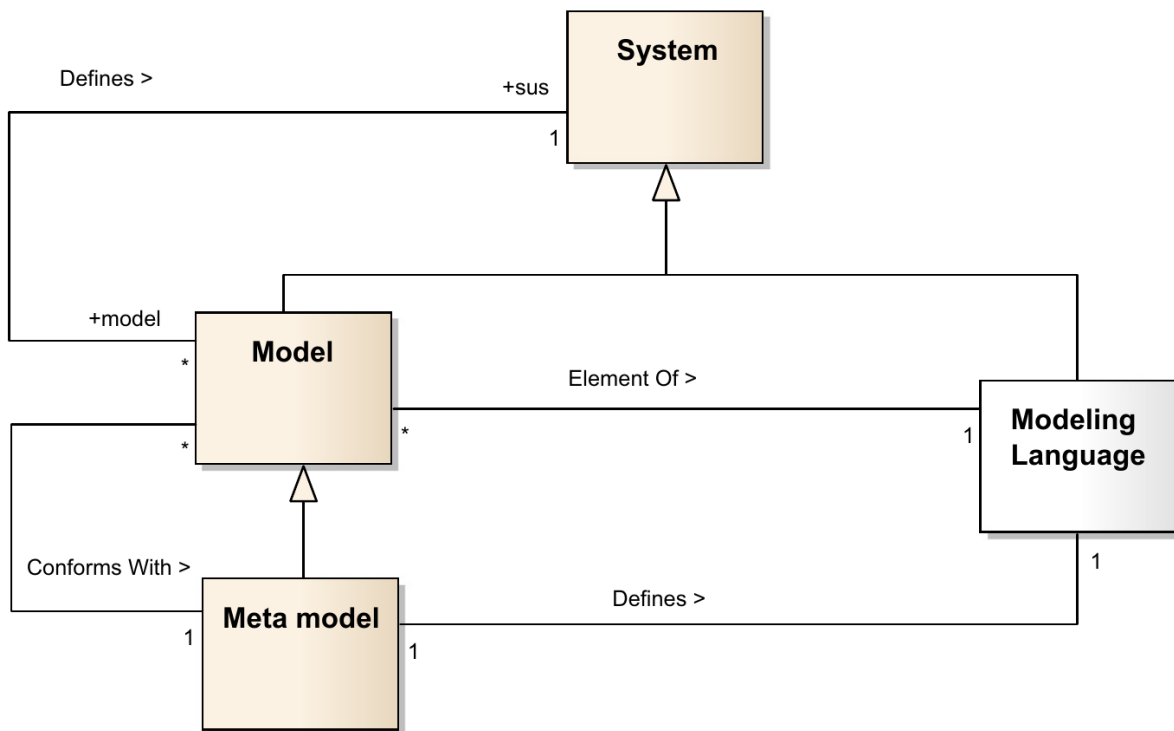
Ένα Μοντέλο είναι η απλουστευμένη αναπαράσταση ενός συστήματος, το οποίο μπορεί είτε να υφίσταται ήδη, είτε να δύναται να υφίσταται στο μέλλον. Το μοντέλο ορίζει το σύστημα και αντίστροφα. Ωστόσο, το μοντέλο από μόνο του είναι ένα σύστημα με δική του ταυτότητα, πολυπλοκότητα, χαρακτηριστικά, σχέσεις κ.τ.λ. Συνοψίζοντας, σύμφωνα και με το [2.3](#) μπορούμε να πούμε πως το μοντέλο είναι ένα σύστημα του οποίου ο σκοπός είναι ο καθορισμός ενός άλλου συστήματος, χωρίς να χρειάζεται να εξετάσουμε άμεσα το τελευταίο [4].

Μετα-Μοντέλο

Όπως και για το μοντέλο, έτσι και για το Μετα-Μοντέλο, δεν υπάρχει κάποιος κοινά αποδεκτός ορισμός. Θα μπορούσαμε ωστόσο να το χαρακτηρίσουμε ως μια απλουστευμένη αναπαράσταση ενός μοντέλου. Στην ουσία τα μετα-μοντέλα αποτελούν τον ορισμό μιας γλώσσας μοντελοποίησης, τον ορισμό της οποίας θα αναλύσουμε στη συνέχεια, καθώς παρέχουν έναν τρόπο περιγραφής όλων των μοντέλων που μπορούν να αναπαρασταθούν από τη συγκεκριμένη γλώσσα [3]. Μια αναπαράσταση του ορισμού του μετα-μοντέλου φαίνεται στο [σχήμα 2.4](#).



Σχήμα 2.3: Ορισμός του μοντέλου.



Σχήμα 2.4: Ορισμός του μετα-μοντέλου.

2.2.2 Γλώσσες μοντελοποίησης

Οι Γλώσσες Μοντελοποίησης (*Modeling Languages*) είναι ένα από τα βασικότερα κομμάτια της MDE. Μέσω αυτών γίνεται δυνατή η παρουσίαση ενός μοντέλου, και μπορεί να αποτελούνται είτε από γραφικές αναπαραστάσεις, είτε από προδιαγραφές μέσω κειμένου, είτε και από τα δύο. Οι γλώσσες αυτές χωρίζονται σε 2 μεγάλες κατηγορίες, τις Γλώσσες Συγκεκριμένου Πεδίου (*Domain Specific Languages* ή *DSL*)

και τις Γλώσσες Γενικού Σκοπού (*General Purpose Languages* ή *GPL*). Μια DSL, είναι μία γλώσσα ειδικά σχεδιασμένη για έναν συγκεκριμένο τομέα, σε αντίθεση με μία GPL, η οποία μπορεί να βρει εφαρμογή σε πληθώρα τομέων. Στο πλαίσιο της παρούσας εργασίας, αναπτύχθηκε μια DSL.

2.2.3 Μετασχηματισμοί

Πέρα από τα μοντέλα, οι *Μετασχηματισμοί* που μπορούν να γίνουν στα μοντέλα είναι ένα ακόμα σημαντικό στοιχείο της MDE. Ένας μετασχηματισμός αποτελείται από ένα πλήθος κανόνων σύμφωνα με τους οποίους ένα μοντέλο εισόδου θα μετατραπεί σε ένα μοντέλο εξόδου. Υπάρχουν δύο είδη μεσασχηματισμών, οι οποίοι παρουσιάζονται παρακάτω.

Μετασχηματισμός από μοντέλο σε μοντέλο (*Model to Model transformation* ή *M2M*): Οι μετασχηματισμοί M2M γίνονται σύμφωνα με ένα σύνολο κανόνων, βάση των οποίων τα χαρακτηριστικά ενός μοντέλου μετατρέπονται σε χαρακτηριστικά ενός άλλου μοντέλου. Γίνεται δηλαδή μία αντιστοίχιση των στοιχείων τους (*model mapping*).

Μετασχηματισμός από μοντέλο σε κώδικα (*Model to Text transformation* ή *M2T*): Κατά τη διαδικασία αυτή, σύμφωνα πάλι με ένα σύνολο κανόνων, από τα στοιχεία του μοντέλου παράγεται κώδικας προς εκτέλεση, με βάση κάποια συγκεκριμένα πρότυπα.

2.3 ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ ΠΡΑΓΜΑΤΙΚΟΥ ΧΡΟΝΟΥ

Στην παρούσα εργασία, αναπτύχθηκε μία DSL, για ένα συγκεκριμένο *Λειτουργικό Σύστημα Πραγματικού Χρόνου* (*Real Time Operating System* ή *RTOS*), το RIOT².

Real-time σύστημα είναι εκείνο το οποίο εκτελεί τις δραστηριότητες που του ανατίθενται μέσα σε συγκεκριμένα χρονικά περιθώρια. Βασικά χαρακτηριστικά των RTOS είναι τα ακόλουθα:

- Χρονοπρογραμματισμός
- Διαχείριση πόρων
- Συγχρονισμός
- Επικοινωνία
- Ακριβής χρονισμός

Παλαιότερα ένα RTOS αποσκοπούσε στην υλοποίηση μιας πολύ συγκεκριμένης λειτουργίας. Πλέον έχουν εξελιχθεί σε επίπεδο τέτοιο, ώστε να υποστηρίζουν

²<https://www.riot-os.org/>

λειτουργικά συστήματα πιο γενικού σκοπού, μέχρι και soft συστήματα πραγματικού χρόνου (δηλαδή συστήματα που λειτουργούν με υποβαθμισμένη απόδοση ανταποτελέσματα δεν παράγονται σύμφωνα με τις καθορισμένες χρονικές απαιτήσεις, σε αντίθεση με τα hard, τα οποία σε αυτήν την περίπτωση λειτουργούν λανθασμένα).

Τα RTOS δίνουν έμφαση στην προβλεψιμότητα, την αποτελεσματικότητα και περιλαμβάνουν δυνατότητες ώστε να υποστηρίζουν χρονικούς περιορισμούς. Υπάρχουν αρκετές γενικές κατηγορίες RTOS, όπως είναι οι πυρήνες (είτε είναι εμπορικοί είτε όχι), επεκτάσεις σε ήδη υπάρχοντα εμπορικά RTOS (π.χ. Unix, Linux), πυρήνες βασισμένους σε εξαρτήματα κ.α. [5]

2.4 ΠΡΩΤΟΚΟΛΛΑ ΕΠΙΚΟΙΝΩΝΙΑΣ

Τα Πρωτόκολλα επικοινωνίας είναι ένα σύνολο κανόνων που επιτρέπουν σε δύο ή περισσότερα συστήματα επικοινωνίας να ανταλλάσσουν δεδομένα μέσω οποιουδήποτε φυσικού μέσου. Οι κανόνες και ο συγχρονισμός μεταξύ των συστημάτων, η σύνταξη που πρέπει να ακολουθηθεί και η σημασιολογία ορίζονται όλα με τον όρο πρωτόκολλο. Τα πρωτόκολλα μπορούν να υλοποιηθούν τόσο από υλικό όσο και από λογισμικό ή από συνδυασμό και των δύο. Τα αναλογικά και ψηφιακά συστήματα επικοινωνίας χρησιμοποιούν ευρέως διάφορα πρωτόκολλα επικοινωνίας. Επιπλέον κάθε πρωτόκολλο έχει τη δική του περιοχή εφαρμογής.

Στην περίπτωση του IoT, τα μέρη ενός συστήματος πρέπει να επικοινωνούν μεταξύ τους για να παρέχουν την αναμενόμενη έξοδο. Κάθε οντότητα θα πρέπει να συμφωνεί με κάποιο πρωτόκολλο για την ανταλλαγή πληροφοριών. Πολλά διαφορετικά πρωτόκολλα είναι διαθέσιμα για IoT συσκευές και αναπτύσσονται ανάλογα με την περιοχή εφαρμογής.

Στην παρούσα εργασία υποστηρίζονται τρία διαφορετικά πρωτόκολλα επικοινωνίας, τα οποία παρουσιάζονται παρακάτω.

2.4.1 I2C

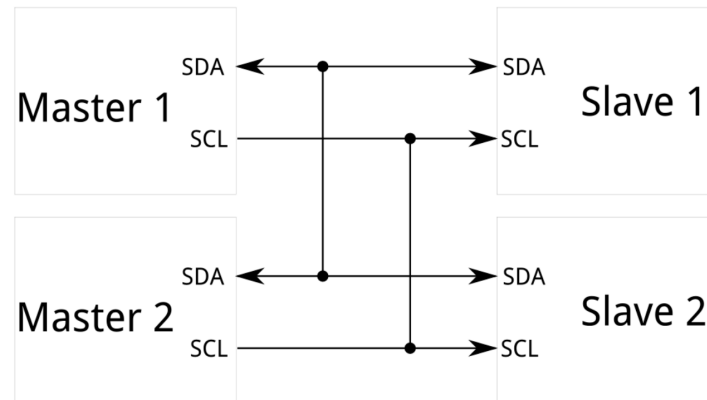
Το *Inter Integrated Circuit (I2C)* είναι ένα σειριακό πρωτόκολλο επικοινωνίας το οποίο αναπτύχθηκε από την Philips Semiconductors³. Ο κύριος σκοπός του είναι να παρέχει ευκολία στη σύνδεση περιφερειακών τσιπ με μικροελεγκτή.

Είναι master-slave (αφέντης-σκλάβος) πρωτόκολλο επικοινωνίας. Κάθε slave έχει μια μοναδική διεύθυνση. Για να επιτευχθεί η επικοινωνία, η master συσκευή αρχικά αποστέλλει την διεύθυνση του επιθυμητού slave μαζί με τη σημαία R/W

³<https://www.nxp.com/>

(read/write ή ανάγνωση/εγγραφή). Η αντίστοιχη slave συσκευή θα μεταβεί σε ενεργή λειτουργία.

Εφόσον η slave συσκευή είναι έτοιμη, ξεκινάει η επικοινωνία μεταξύ master και slave. Για να λειτουργήσει χρειάζεται δύο διαύλους, το SDA και το SCL, όπως φαίνεται και στο [σχήμα 2.5](#).



Σχήμα 2.5: I2C

2.4.2 SPI

Το *Serial Peripheral Interface (SPI)* αναπτύχθηκε από τη Motorola⁴ και αποτελείται από τους ακόλουθους διαύλους:

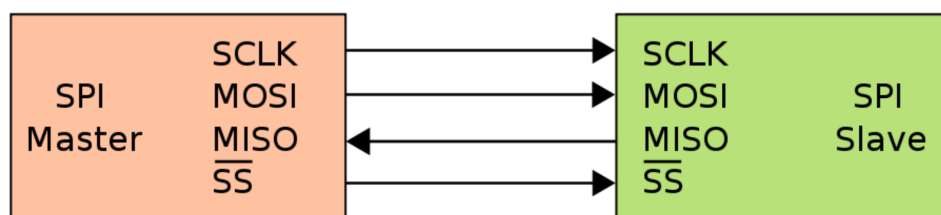
- MOSI: Master Out Slave in
- MISO: Master In Slave Out
- SS: Slave Select
- SCLK: Serial Clock
- Ακριβής χρονισμός

Μια αναπαράσταση των διαύλων μπορούμε να δούμε στο [σχήμα 2.6](#). Όπως το I2C, έτσι και το SPI είναι πρωτόκολλο επικοινωνίας master-slave. Στο SPI, πρώτα η master συσκευή διαμορφώνει το ρολόι σε μια συγκεκριμένη συχνότητα. Επιπλέον ο δίαυλος SS χρησιμοποιείται για την επιλογή του κατάλληλου slave. Αφού επιλεγεί η slave συσκευή, αρχίζει η επικοινωνία.

2.4.3 UART

Το *Universal Asynchronous Receiver/Transmitter (UART)* δεν είναι ακριβώς πρωτόκολλο αλλά ένα φυσικό κομμάτι υλικού που μετατρέπει παράλληλα δεδομένα σε

⁴<https://www.motorola.com/us/>

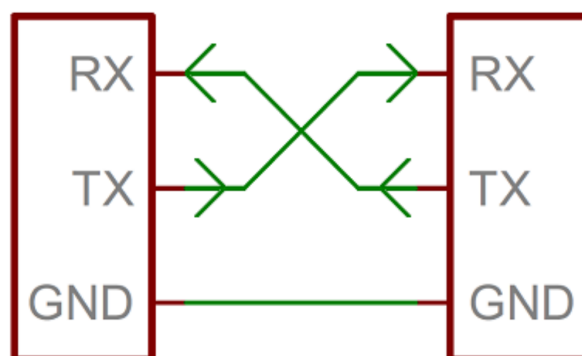


Σχήμα 2.6: SPI

σειριακά. Ο κύριος σκοπός του είναι η μετάδοση και λήψη δεδομένων σειριακά.

Αποτελείται από δύο διαύλους έναν για την αποστολή των δεδομένων και έναν για τη λήψη. Επομένως χρειάζεται δύο ακροδέκτες, ο Rx (δέκτης) και ο Tx (πομπός), το οποίο φαίνεται και στο [σχήμα 2.7](#)).

Το UART μεταδίδει δεδομένα ασύγχρονα, άρα κανένα σήμα ρολογιού δεν σχετίζεται με τη μετάδοση και τη λήψη δεδομένων. Αντί για σήμα ρολογιού, χρησιμοποιεί bit έναρξης και διακοπής μέσω πραγματικών bit δεδομένων, τα οποία καθορίζουν την έναρξη και το τέλος του πακέτου δεδομένων.



Σχήμα 2.7: UART

3

Επισκόπηση της Ερευνητικής Περιοχής

Στο κεφάλαιο αυτό γίνεται μία σύντομη αναφορά σε ήδη υπάρχουσες μοντελοκεντρικές υλοποιήσεις στον τομέα του IoT. Οι υλοποιήσεις αυτές παρέχουν είτε γλώσσες μοντελοποίησης γενικού σκοπού, είτε γλώσσες συγκεκριμένου τομέα. Αν και υπάρχουν αρκετές σχετικές δημοσιεύσεις τα τελευταία 7 περίπου χρόνια, η πρώτη απόπειρα μοντελοποίησης του RIOT έγινε στα μέσα του 2020 [6].

Οι Salihbegovic κ.α. [7] δημιούργησαν μια Οπτική γλώσσα μοντελοποίησης συγκεκριμένου τομέα (VDSML), βασισμένη σε JavaScript⁵, η οποία παρέχει μια διεπαφή χρήστη για την σχεδίαση ενός IoT συστήματος. Η εκτέλεση των παραχθέντων αρχείων γίνεται με τη χρήση του OpenHAB⁶.

Η πρώτη ολοκληρωμένη υλοποίηση έγινε από τους Harrand κ.α. [8] δημιουργώντας την ThingML, μια γλώσσα συγκεκριμένου τομέα (DSL) για τη μοντελοποίηση IoT συστημάτων χρησιμοποιώντας πεπερασμένα αυτόματα (state machines). Η ThingML φαίνεται πως είναι ένα καλό και χρήσιμο εργαλείο, το οποίο χρησιμοποιείται και συντηρείται μέχρι και σήμερα.

Οι Berrouyne κ.α. [9] δημιούργησαν ένα εργαλείο, το CypriIoT, για τη μοντελοποίηση και τον έλεγχο διαδικτυακών IoT εφαρμογών. Παρέχουν λοιπόν δύο γλώσσες. Η πρώτη στοχεύει στον σχεδιασμό ενός δικτύου με έναν πιο ευανάγνωστο και υψηλότερου επιπέδου τρόπο, χρησιμοποιώντας μάλιστα την ThingML για την μοντελοποίηση των διασυνδέσεων. Η δεύτερη, είναι μια γλώσσα πολιτικής (Policy Language), στοχεύει δηλαδή στη δημιουργία κανόνων για τον έλεγχο των διαδικτυακών συνδέσεων των συσκευών. Επίσης, παρέχουν και ένα εργαλείο παραγωγής

⁵JavaScript, <https://www.javascript.com/>

⁶OpenHAB, <https://www.openhab.org/>

κώδικα.

Σε αυτό το σημείο θα αναφερθούν και τρεις ακόμα υλοποιήσεις, μέσω των οποίων γίνεται μοντελοποίηση συγκεκριμένης πλατφόρμας (Platform-specific modeling), και άρα είναι άρρηκτα συνδεδεμένες με το αντικείμενο της παρούσας διπλωματικής. Οι υλοποιήσεις αυτές αποσκοπούν στην μετέπειτα ύπαρξη ενός πιο γενικού, ανεξάρτητου της πλατφόρμας (Platform independent) εργαλείου, για τη μοντελοποίηση της επικοινωνίας ασύρματου δικτύου αισθητήρων (WSN) σε ένα IoT σύστημα.

Αρχικά, οι Marah κ.α. [10] δημιούργησαν μια γλώσσα μοντελοποίησης συγκεκριμένου τομέα, για το λειτουργικό TinyOS⁷. Μέσω αυτής, γίνεται η μοντελοποίηση των διασυνδέσεων των συσκευών ενός IoT συστήματος, και στη συνέχεια η παραγωγή κώδικα nesC [11] για την εκτέλεση συγκεκριμένων λειτουργιών στο λειτουργικό TinyOS.

Επόμενη υλοποίηση ήταν η επέκταση μιας προηγούμενης [12] πιο περιορισμένης, η οποία παρείχε τη μοντελοποίηση εφαρμογών σε ContikiOS⁸. Στη νέα υλοποίηση από τους Asici κ.α. [13], επεκτείνεται το μετα-μοντέλο ώστε να υποστηρίζει επιπλέον εξαρτήματα που χρησιμοποιούνται σε ένα IoT σύστημα, δημιουργείται ένας γραφικός συντάκτης, και παρέχεται και ένα εργαλείο κανόνων για την παραγωγή κώδικα.

Η πιο πρόσφατη υλοποίηση [6], ήταν μια μοντελοκεντρική προσέγγιση για την ανάπτυξη IoT συστημάτων για το λειτουργικό RIOT⁹, με το οποίο θα ασχοληθούμε και στην παρούσα εργασία. Δημιουργήθηκε ένα μετα-μοντέλο για το λειτουργικό RIOT, και με βάση αυτό παράγεται μία γλώσσα μοντελοποίησης συγκεκριμένου τομέα (DSML). Τέλος, μέσω συγκεκριμένων κανόνων, παράγεται κώδικας για την εκτέλεση συγκεκριμένων λειτουργιών στο λειτουργικό RIOT. Η υλοποίηση αυτή, μοντελοποιεί ένα μεγάλο κομμάτι του RIOT, και μάλιστα φαίνεται να έχει ποιοτικά αποτελέσματα, όσον αφορά τον παραγόμενο κώδικα. Ωστόσο, επικεντρώνεται κυρίως στον τρόπο επικοινωνίας μεταξύ των συσκευών στο δίκτυο. Στην παρούσα εργασία, πέρα από τη μοντελοποίηση των βασικών στοιχείων του RIOT, και την παραγωγή κώδικα, θα παρέχουμε στον χρήστη πληροφορία σχετικά με τον τρόπο διασύνδεσης των εξαρτημάτων (μικροελεγκτών, αισθητήρων, ενεργοποιητών).

⁷TinyOS, <http://www.tinyos.net/>

⁸ContikiOS, <https://www.contiki-ng.org/>

⁹RIOT OS <https://www.riot-os.org/>

4

Εργαλεία

4.1 RIOT

Το *RIOT* [14] είναι ένα λειτουργικό σύστημα βασισμένο σε μικροπυρήνα ανοιχτού κώδικα, σχεδιασμένο για να ανταποκρίνεται στις απαιτήσεις των συσκευών IoT και άλλων ενσωματωμένων συσκευών. Αυτές οι απαιτήσεις περιλαμβάνουν ένα πολύ χαμηλό αποτύπωμα μνήμης (της τάξης των λίγων kilobytes), υψηλή ενεργειακή απόδοση, δυνατότητες σε πραγματικό χρόνο, υποστήριξη για ένα ευρύ φάσμα υλικού χαμηλής κατανάλωσης ενέργειας, στοίβες επικοινωνίας για ασύρματα και ενσύρματα δίκτυα.

Το RIOT παρέχει έναν μικροπυρήνα, πολλαπλές στοίβες δικτύου και βοηθητικά προγράμματα που περιλαμβάνουν κρυπτογραφικές βιβλιοθήκες, δομές δεδομένων, ένα τερματικό και άλλα. Το RIOT υποστηρίζει ένα ευρύ φάσμα αρχιτεκτονικών μικροελεγκτών, αισθητήρων και διαμορφώσεων για ολόκληρες πλατφόρμες, π.χ. Atmel SAM R21 Xplained Pro, Zolertia Z1, STM32 Discovery Boards κ.λ.π. σε όλο το υποστηριζόμενο υλικό (πλατφόρμες 32-bit, 16-bit και 8-bit). Το RIOT παρέχει τη δυνατότητα για προγραμματισμό εφαρμογών σε ANSI C και C++, με multithreading, χρονοδιακόπτες, κ.α.

Στην παρούσα εργασία χρησιμοποιήθηκε καθώς όλη η μοντελοποίηση έγινε με στόχο την παραγωγή κώδικα ο οποίος θα εκτελείται σε RIOT.

¹⁰<https://www.riot-os.org/>



Σχήμα 4.1: Λειτουργικό σύστημα RIOT.¹⁰

4.2 TEXTX

Η textX [15] είναι μία μετα-γλώσσα (γλώσσα για τη) για τη δημιουργία γλωσσών συγκεκριμένου τομέα (DSL) σε Python. Δηλαδή, αυτή η μετα-γλώσσα παρέχει τη γραμματική για των ορισμό νέων γλωσσών. Για κάθε γραμματική η textX δημιουργεί έναν αναλυτή και ένα μέτα-μοντέλο. Το μετά μοντέλο εμπεριέχει όλες τις πληροφορίες σχετικά με την γλώσσα και από την γραμματική παράγεται ένα σύνολο από κλάσεις στην γλώσσα Python. Ο αναλυτής θα προσπελάσει τα μοντέλα που έχουν γραφεί σε αυτήν την νέα γλώσσα και θα δημιουργήσει έναν γράφο που θα συμμορφώνεται στο μετα-μοντέλο. Στην παρούσα εργασία χρησιμοποιήθηκε για τον ορισμό της DSL που περιγράφει τα μοντέλα μέσω ενός εργαλείου κειμένου.



Σχήμα 4.2: textX¹¹

4.3 JINJA

Το Jinja¹² είναι ένα εργαλείο για παραγωγή αρχείων κειμένων μέσω προτύπων. Μέσω της χρήσης παραμέτρων, υπάρχει η δυνατότητα δυναμικής τροποποίησης των αρχείων αυτών. Αρχικά δημιουργήθηκε για την παραγωγή αρχείων HTML, αλλά μπορεί να χρησιμοποιηθεί για την παραγωγή αρχείων κειμένου σε οποιαδήποτε γλώσσα προγραμματισμού. Το Jinja είναι βιβλιοθήκη της Python, και άρα η διαδικασία παραγωγής αρχείων από τα πρότυπα, θα πρέπει να οριστεί σε ένα αρχείο κειμένου γραμμένο σε Python. Στην παρούσα εργασία χρησιμοποιήθηκε για την παραγωγή Makefile και κώδικα σε γλώσσα C ώστε να δημιουργηθεί ένα πρόγραμμα που θα τρέχει στο λειτουργικό RIOT.

¹¹<http://textx.github.io/textX/stable/>

¹²<https://jinja.palletsprojects.com/en/3.0.x/>



Σχήμα 4.3: Jinja

4.4 PLANTUML

Το *PlantUML*¹³ είναι ένα εργαλείο ανοιχτού κώδικα για τη σχεδίαση διαγραμμάτων UML, χρησιμοποιώντας μια απλή και αναγνώσιμη περιγραφή κειμένου. Στην παρούσα εργασία χρησιμοποιήθηκε για την παραγωγή διαγραμμάτων απεικόνισης των μετα-μοντέλων, αλλά και τις συνδεσμολογίας των συσκευών.

¹³<https://plantuml.com/>

5

Μεθοδολογία και Υλοποίηση

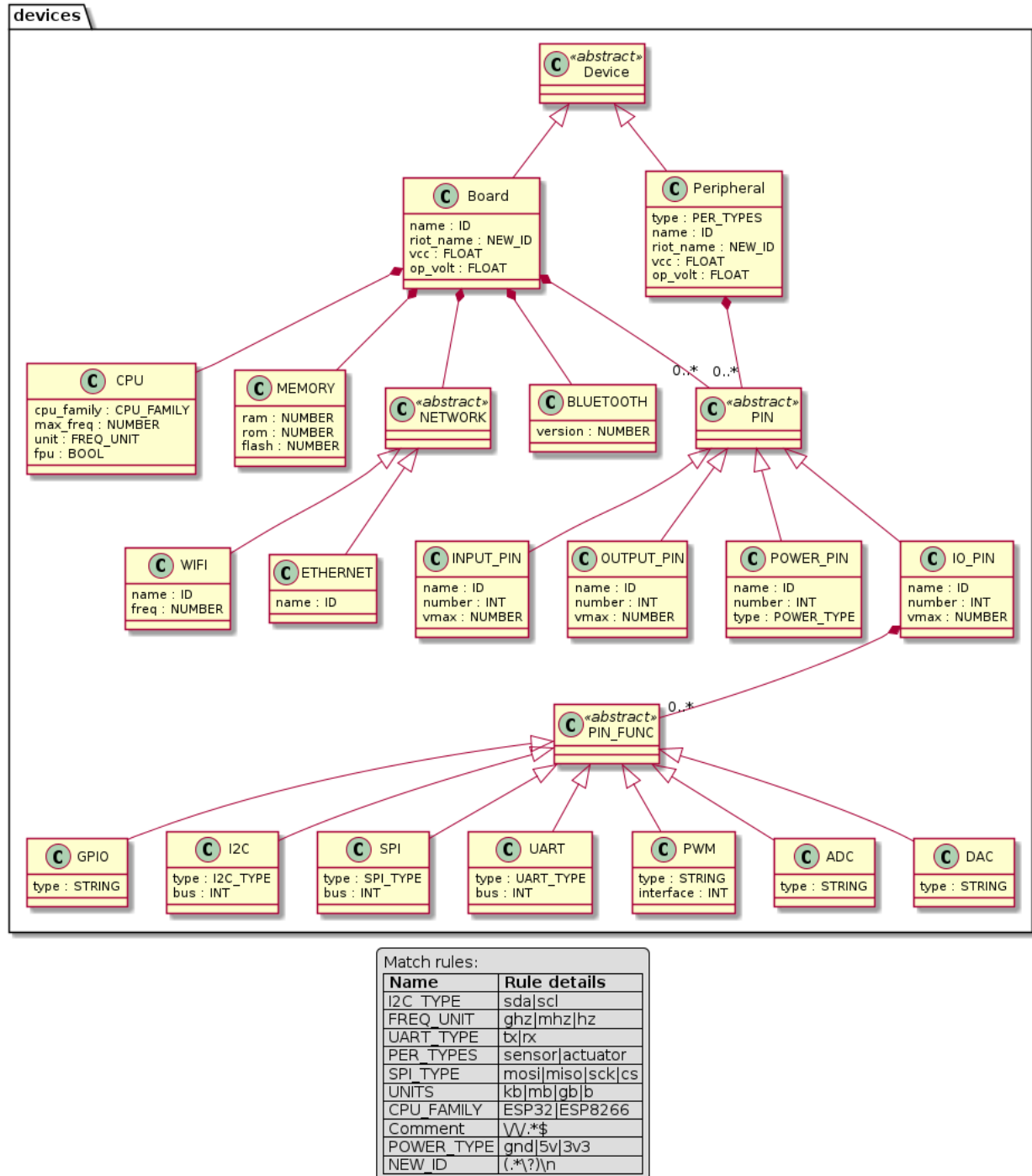
Στόχος της παρούσας εργασίας είναι να παρέχει στον χρήστη μια απλοποιημένη διαδικασία ώστε να παράγεται κώδικας βασισμένος στις συσκευές στις οποίες διαθέτει, ο οποίος θα εκτελεστεί μετέπειτα στο λειτουργικό RIOT. Κατά αυτόν τον τρόπο, ο χρήστης μπορεί να κατασκευάσει μια εφαρμογή με τις IoT συσκευές που διαθέτει, η οποία θα εκτελεί κάποιες αρκετά βασικές λειτουργίες, χωρίς να χρειάζεται να αναλωθεί στο να βρει τον τρόπο με τον οποίο υποστηρίζονται οι συσκευές αυτές από το RIOT.

Για να επιτευχθεί αυτός ο στόχος, αρχικά σχεδιάστηκαν κάποια μετα-μοντέλα που περιέχουν χαρακτηριστικά συσκευών και των συνδέσεων μεταξύ τους. Μέσω ενός εργαλείου κειμένου, ο χρήστης πρέπει να ορίσει τα χαρακτηριστικά που επιθυμεί να έχει το δικό του σύστημα (σύμφωνα πάντα με τους κανόνες των μετα-μοντέλων). Αφού γίνει αυτό, πραγματοποιούνται μετασχηματισμοί M2M από μοντέλα κειμένου σε διαγράμματα, μέσω των οποίων ο χρήστης μπορεί πιο εύκολα να αντιληφθεί πως ακριβώς θα πρέπει να είναι η συνδεσμολογία μεταξύ των συσκευών του. Τέλος, από τα μοντέλα γίνεται και πάλι μετασχηματισμός (αυτή τη φορά M2T) σε αρχεία κώδικα, τα οποία θα είναι έτοιμα ώστε να γίνει η εγγραφή τους στις συσκευές και άρα να εκτελεστούν.

Όλα όσα υλοποιήθηκαν, περιγράφονται αναλυτικά σε αυτήν την ενότητα.

5.1 ΟΡΙΣΜΟΣ ΜΕΤΑ-MΟΝΤΕΛΟΥ ΣΥΣΚΕΥΗΣ

Το μετα-μοντέλο αυτό περιέχει χαρακτηριστικά που μπορεί να έχει μια συσκευή (μικροελεγκτής ή περιφερειακό) και μας χρησιμεύουν στη διαδικασία παραγωγής κώδικα. Στο [σχήμα 5.1](#) μπορούμε να δούμε μία απεικόνισή του.



Σχήμα 5.1: Μετα-μοντέλο συσκευής

5.1.1 Device

Σύνοψη

Το στοιχείο αυτό είναι η abstract κλάση για το αν μια συσκευή είναι μικροελεγκτής ή περιφερειακό.

Ιδιότητες και Συσχετίσεις

Δεν περιλαμβάνει περαιτέρω ιδιότητες και συσχετίσεις.

Περιορισμοί

Δεν υπάρχουν περιορισμοί.

5.1.2 Board

Σύνοψη

Το στοιχείο αυτό αναπαριστά τις υπολογιστικές συσκευές (μικροελεγκτές).

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
name	ID	1..1	Το όνομα της συσκευής
riot_name	NEW_ID	1..1	Το όνομα της συσκευής όπως αναγνωρίζεται από το RIOT
vcc	FLOAT	1..1	Η τάση τροφοδοσίας της συσκευής
op_volt	FLOAT	1..1	Η τάση λειτουργίας της συσκευής
Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
Device	SuperType-Επέκταση	-	Το στοιχείο Board επεκτείνει το στοιχείο Device
CPU	Composition-Σύνθεση	1..1	Η κεντρική μονάδα επεξεργασίας της συσκευής
MEMORY	Composition-Σύνθεση	1..1	Η μνήμη της συσκευής
NETWORK	Composition-Σύνθεση	0..1	Πρωτόκολλα δικτύου που υποστηρίζει η συσκευή
PIN	Composition-Σύνθεση	0..*	Οι ακροδέκτες της συσκευής
BLUETOOTH	Composition-Σύνθεση	0..1	Υποστήριξη bluetooth

Πίνακας 5.1: Ιδιότητες και Συσχετίσεις του *Board*.

Περιορισμοί

Το riot_name μπορεί να πάρει τιμές σαν ID, με επιπλέον επιλογή να περιλαμβάνει και παύλες. Αυτό επιτυγχάνεται σύμφωνα με το NEW_ID που είναι μια κανονική έκφραση (regex).

5.1.3 Peripheral

Σύνοψη

Το στοιχείο αυτό αναπαριστά τα περιφερειακά.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
type	PER_TYPES (Enum)	1..1	Ο τύπος του περιφερειακού
name	ID	1..1	Το όνομα της συσκευής
riot_name	NEW_ID	1..1	Το όνομα της συσκευής όπως αναγνωρίζεται από το RIOT
vcc	ID	1..1	Η τάση τροφοδοσίας της συσκευής
op_volt	FLOAT	1..1	Η τάση λειτουργίας της συσκευής
Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
Device	SuperType-Επέκταση	-	Το στοιχείο Peripheral επεκτείνει το στοιχείο Device
PIN	Composition-Σύνθεση	1..1	Οι ακροδέκτες της συσκευής

Πίνακας 5.2: Ιδιότητες και Συσχετίσεις του *Peripheral*.

Περιορισμοί

- Το riot_name μπορεί να πάρει τιμές σαν ID, με επιπλέον επιλογή να περιλαμβάνει και παύλες. Αυτό επιτυγχάνεται σύμφωνα με το NEW_ID που είναι μια κανονική έκφραση (regex).

- Επιλογές των υποστηριζόμενων τύπων περιφερειακών για το type:

- sensor
- actuator

5.1.4 CPU

Σύνοψη

Το στοιχείο αυτό περιγράφει την κεντρική μονάδα επεξεργασίας της συσκευής.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
cpu_family	CPU_FAMILY (Enum)	1..1	Η αρχιτεκτονική της κεντρικής μονάδας επεξεργασίας
max_freq	NUMBER	1..1	Η μέγιστη συχνότητα της κεντρικής μονάδας επεξεργασίας
unit	FREQ_UNIT (Enum)	1..1	Μονάδα μέτρησης της μέγιστης συχνότητας
fpu	BOOL	1..1	Υποστήριξη πράξεων κινητής υποδιαστολής της κεντρικής μονάδας επεξεργασίας

Πίνακας 5.3: Ιδιότητες του *CPU*.

Δεν περιλαμβάνει περαιτέρω συσχετίσεις.

Περιορισμοί

- Επιλογές των υποστηριζόμενων αρχιτεκτονικών για το `cpu_family`:
 - ESP32
 - ESP8266
- Επιλογές των υποστηριζόμενων μονάδων μέτρησης:
 - hz
 - ghz
 - mhz

5.1.5 NETWORK

Σύνοψη

Το στοιχείο αυτό είναι η abstract κλάση για το αν μια συσκευή υποστηρίζει wifi ή ethernet (ή και τα δύο) ως τρόπο δικτύωσης.

Ιδιότητες και Συσχετίσεις

Δεν περιλαμβάνει περαιτέρω ιδιότητες και συσχετίσεις.

Περιορισμοί

Δεν υπάρχουν περιορισμοί.

5.1.6 WIFI

Σύνοψη

Το στοιχείο αυτό περιγράφει τον τρόπο δικτύωσης μέσω wifi.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
name	ID	1..1	Το όνομα της δικτύωσης μέσω wifi
freq	NUMBER	1..1	Η συχνότητα λειτουργίας
Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
NETWORK	SuperType-Επέκταση	-	Το στοιχείο WIFI επεκτείνει το στοιχείο NETWORK

Πίνακας 5.4: Ιδιότητες και Συσχετίσεις του *WIFI*.

Περιορισμοί

Επιλογές των υποστηριζόμενων μονάδων μέτρησης για το freq:

- hz
- ghz
- mhz

5.1.7 ETHERNET

Σύνοψη

Το στοιχείο αυτό περιγράφει τον τρόπο δικτύωσης μέσω ethernet.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
name	ID	1..1	Το όνομα της δικτύωσης μέσω ethernet
Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
NETWORK	SuperType-Επέκταση	-	Το στοιχείο ETHERNET επεκτείνει το στοιχείο NETWORK

Πίνακας 5.5: Ιδιότητες και Συσχετίσεις του *ETHERNET*.

Περιορισμοί

Δεν υπάρχουν περιορισμοί.

5.1.8 MEMORY

Σύνοψη

Το στοιχείο αυτό περιγράφει τη μνήμη της συσκευής.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
ram	FLOAT	1..1	Το μέγεθος της μνήμης RAM
rom	FLOAT	1..1	Το μέγεθος της μνήμης ROM
flash	FLOAT	1..1	Το μέγεθος της μνήμης FLASH

Πίνακας 5.6: Ιδιότητες του *MEMORY*.

Δεν περιλαμβάνει περαιτέρω συσχετίσεις.

Περιορισμοί

Επιλογές των υποστηριζόμενων μονάδων μέτρησης για τα ram, rom και flash:

- b
- kb
- mb
- gb

5.1.9 BLEUTOOTH

Σύνοψη

Το στοιχείο αυτό περιγράφει την υποστήριξη bluetooth από τη συσκευή.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
version	NUMBER	1..1	Η έκδοση του πρωτοκόλλου bluetooth

Πίνακας 5.7: Ιδιότητες του *BLEUTOOTH*.

Δεν περιλαμβάνει περαιτέρω συσχετίσεις.

Περιορισμοί

Δεν υπάρχουν περιορισμοί.

5.1.10 PIN

Σύνοψη

Το στοιχείο αυτό είναι η abstract κλάση που περιγράφει τα χαρακτηριστικά ενός ακροδέκτη της συσκευής.

Ιδιότητες και Συσχετίσεις

Δεν περιλαμβάνει περαιτέρω ιδιότητες και συσχετίσεις.

Περιορισμοί

Δεν υπάρχουν περιορισμοί.

5.1.11 POWER_PIN

Σύνοψη

Το στοιχείο αυτό περιγράφει τους ακροδέκτες τροφοδοσίας.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
name	ID	1..1	Το όνομα που θα δοθεί στον ακροδέκτη για πιθανή αναφορά σε αυτόν
number	INT	1..1	Ο αριθμός του ακροδέκτη (όσον αφορά την θέση του στη συσκευή)
type	POWER_TYPE (Enum)	1..1	Το είδος της τροφοδοσίας
Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
PIN	SuperType-Επέκταση	-	Το στοιχείο POWER_PIN επεκτείνει το στοιχείο PIN

Πίνακας 5.8: Ιδιότητες και Συσχετίσεις του *POWER_PIN*.

Περιορισμοί

Επιλογές των υποστηριζόμενων ειδών τροφοδοσίας για το type:

- gnd
- 5v
- 3v3

5.1.12 INPUT_PIN

Σύνοψη

Το στοιχείο αυτό περιγράφει τους ακροδέκτες που είναι αποκλειστικά εισόδου.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
name	ID	1..1	Το όνομα που θα δοθεί στον ακροδέκτη για πιθανή αναφορά σε αυτόν
number	INT	1..1	Ο αριθμός του ακροδέκτη (όσον αφορά την θέση του στη συσκευή)
vmax	NUMBER	0..1	Η μέγιστη τιμή τάσης που μπορεί να δεχτεί ο ακροδέκτης
Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
PIN	SuperType-Επέκταση	-	Το στοιχείο INPUT_PIN επεκτείνει το στοιχείο PIN

Πίνακας 5.9: Ιδιότητες και Συσχετίσεις του *INPUT_PIN*.

Περιορισμοί

Δεν υπάρχουν περιορισμοί.

5.1.13 OUTPUT_PIN

Σύνοψη

Το στοιχείο αυτό περιγράφει τους ακροδέκτες που είναι αποκλειστικά εξόδου.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
name	ID	1..1	Το όνομα που θα δοθεί στον ακροδέκτη για πιθανή αναφορά σε αυτόν
number	INT	1..1	Ο αριθμός του ακροδέκτη (όσον αφορά την θέση του στη συσκευή)
vmax	NUMBER	0..1	Η μέγιστη τιμή τάσης που μπορεί να δεχτεί ο ακροδέκτης
Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
PIN	SuperType-Επέκταση	-	Το στοιχείο OUTPUT_PIN επεκτείνει το στοιχείο PIN

Πίνακας 5.10: Ιδιότητες και Συσχετίσεις του *OUTPUT_PIN*.

Περιορισμοί

Δεν υπάρχουν περιορισμοί.

5.1.14 IO_PIN

Σύνοψη

Το στοιχείο αυτό περιγράφει τους ακροδέκτες εισόδου-εξόδου.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
name	ID	1..1	Το όνομα που θα δοθεί στον ακροδέκτη για πιθανή αναφορά σε αυτόν
number	INT	1..1	Ο αριθμός του ακροδέκτη (όσον αφορά την θέση του στη συσκευή)
vmax	POWER_TYPE (Enum)	0..1	Η μέγιστη τιμή τάσης που μπορεί να δεχτεί ο ακροδέκτης
Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
PIN	SuperType-Επέκταση	-	Το στοιχείο IO_PIN επεκτείνει το στοιχείο PIN
PIN_FUNC	Composition-Σύνθεση	0..*	Το IO_PIN μπορεί να έχει πολλές λειτουργίες

Πίνακας 5.11: Ιδιότητες και Συσχετίσεις του *IO_PIN*.**Περιορισμοί**

Δεν υπάρχουν περιορισμοί.

5.1.15 PIN_FUNC**Σύνοψη**

Το στοιχείο αυτό είναι η abstract κλάση που περιγράφει τις λειτουργίες που μπορεί να έχει ένα IO_PIN.

Ιδιότητες και Συσχετίσεις

Δεν περιλαμβάνει περαιτέρω ιδιότητες και συσχετίσεις.

Περιορισμοί

Δεν υπάρχουν περιορισμοί.

5.1.16 GPIO

Σύνοψη

Το στοιχείο αυτό περιγράφει μια GPIO διεπαφή.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
type	STRING	1..1	Το όνομα της διεπαφής GPIO
Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
PIN_FUNC	SuperType-Επέκταση	-	Το στοιχείο GPIO επεκτείνει το στοιχείο PIN_FUNC

Πίνακας 5.12: Ιδιότητες και Συσχετίσεις του *GPIO*.

Περιορισμοί

Το type θα πρέπει να έχει την τιμή "gpio", αλλιώς θα εμφανιστεί σφάλμα.

5.1.17 I2C

Σύνοψη

Το στοιχείο αυτό περιγράφει μια I2C διεπαφή.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
type	I2C_TYPE (Enum)	1..1	Το είδος της διεπαφής I2C
bus	INT	1..1	Ο αριθμός περιγραφής του διαύλου
Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
PIN_FUNC	SuperType-Επέκταση	-	Το στοιχείο I2C επεκτείνει το στοιχείο PIN_FUNC

Πίνακας 5.13: Ιδιότητες και Συσχετίσεις του I2C.

Περιορισμοί

Επιλογές των υποστηριζόμενων ειδών τροφοδοσίας για το type:

- sda
- scl

5.1.18 SPI

Σύνοψη

Το στοιχείο αυτό περιγράφει μια SPI διεπαφή.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
type	SPI_TYPE (Enum)	1..1	Το είδος της διεπαφής SPI
bus	INT	1..1	Ο αριθμός περιγραφής του διαύλου
Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
PIN_FUNC	SuperType-Επέκταση	-	Το στοιχείο SPI επεκτείνει το στοιχείο PIN_FUNC

Πίνακας 5.14: Ιδιότητες και Συσχετίσεις του *SPI*.

Περιορισμοί

Επιλογές των υποστηριζόμενων ειδών τροφοδοσίας για το type:

- mosi
- miso
- sck
- cs

5.1.19 UART

Σύνοψη

Το στοιχείο αυτό περιγράφει μια UART διεπαφή.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
type	UART_TYPE (Enum)	1..1	Το είδος της διεπαφής UART
bus	INT	1..1	Ο αριθμός περιγραφής του διαύλου
Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
PIN_FUNC	SuperType-Επέκταση	-	Το στοιχείο UART επεκτείνει το στοιχείο PIN_FUNC

Πίνακας 5.15: Ιδιότητες και Συσχετίσεις του *UART*.

Περιορισμοί

Επιλογές των υποστηριζόμενων ειδών τροφοδοσίας για το type:

- tx
- rx

5.1.20 PWM

Σύνοψη

Το στοιχείο αυτό περιγράφει μια PWM διεπαφή.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
type	STRING	1..1	Το όνομα της διεπαφής PWM
interface	INT	1..1	Ο αριθμός της διεπαφής PWM
Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
PIN_FUNC	SuperType-Επέκταση	-	Το στοιχείο PWM επεκτείνει το στοιχείο PIN_FUNC

Πίνακας 5.16: Ιδιότητες και Συσχετίσεις του *PWM*.

Περιορισμοί

Το type θα πρέπει να έχει την τιμή "rwm", αλλιώς θα εμφανιστεί σφάλμα.

5.1.21 ADC

Σύνοψη

Το στοιχείο αυτό περιγράφει μια ADC διεπαφή.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
type	STRING	1..1	Το όνομα της διεπαφής ADC
Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
PIN_FUNC	SuperType-Επέκταση	-	Το στοιχείο ADC επεκτείνει το στοιχείο PIN_FUNC

Πίνακας 5.17: Ιδιότητες και Συσχετίσεις του *ADC*.

Περιορισμοί

Το type θα πρέπει να έχει την τιμή "adc", αλλιώς θα εμφανιστεί σφάλμα.

5.1.22 DAC

Σύνοψη

Το στοιχείο αυτό περιγράφει μια DAC διεπαφή.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
type	STRING	1..1	Το όνομα της διεπαφής DAC
Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
PIN_FUNC	SuperType-Επέκταση	-	Το στοιχείο DAC επεκτείνει το στοιχείο PIN_FUNC

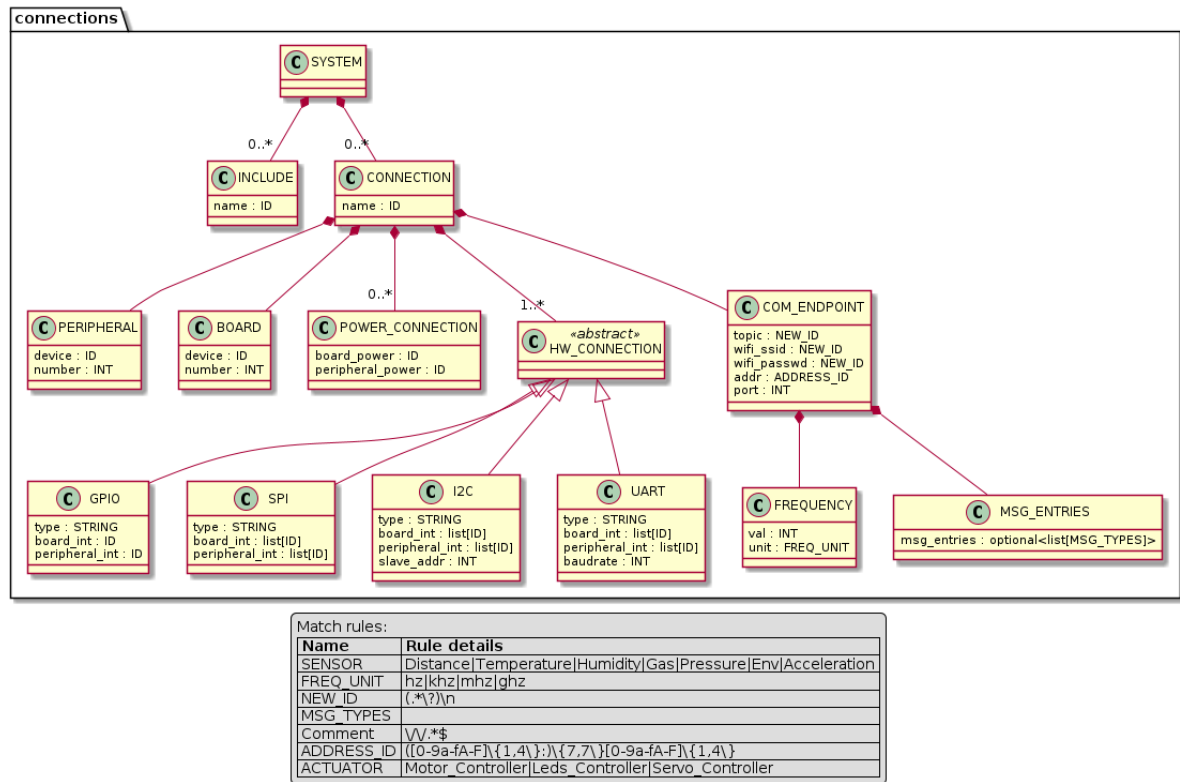
Πίνακας 5.18: Ιδιότητες και Συσχετίσεις του DAC.

Περιορισμοί

Το type θα πρέπει να έχει την τιμή "dac", αλλιώς θα εμφανιστεί σφάλμα.

5.2 ΟΡΙΣΜΟΣ ΜΕΤΑ-MONTEΛΟΥ ΣΥΝΔΕΣΕΩΝ

Το μετα-μοντέλο αυτό περιέχει χαρακτηριστικά που μπορεί να έχει το σύστημα που επιθυμεί ο χρήστης να κατασκευάσει όσον αφορά στη συνδεσμολογία μεταξύ των συσκευών αλλά και στη σύνδεσή τους σε έναν broker. Στο [σχήμα 5.2](#) μπορούμε να δούμε μία απεικόνισή του.



Σχήμα 5.2: Μετα-μοντέλο συνδέσεων

5.2.1 SYSTEM

Σύνοψη

Το στοιχείο αυτό αναπαριστά ένα σύστημα, το οποίο αποτελείται από συνδέσεις μεταξύ συσκευών (μικροελεγκτές και περιφερειακά).

Ιδιότητες και Συσχετίσεις

Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
INCLUDE	Composition-Σύνθεση	0..*	Τα ονόματα των συσκευών που απαρτίζουν το σύστημα
CONNECTION	Composition-Σύνθεση	0..*	Οι συνδέσεις μεταξύ των συσκευών

Πίνακας 5.19: Συσχετίσεις του SYSTEM.

Δεν περιλαμβάνει περαιτέρω ιδιότητες.

Περιορισμοί

Δεν υπάρχουν περιορισμοί.

5.2.2 INCLUDE

Σύνοψη

Το στοιχείο αυτό είναι το όνομα μιας συσκευής η οποία είναι μέρος του συστήματος, ώστε να ξέρουμε την ύπαρξή της.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
name	ID	1..1	Το όνομα της συσκευής

Πίνακας 5.20: Ιδιότητες του *INCLUDE*.

Δεν περιλαμβάνει περαιτέρω συσχετίσεις.

Περιορισμοί

Το ID πρέπει να είναι το όνομα συσκευής για την οποία υπάρχει configuration file (.hwd), και άρα να υποστηρίζεται από την παρούσα εργασία.

5.2.3 CONNECTION

Σύνοψη

Το στοιχείο αυτό αναπαριστά τη σύνδεση ενός μικροελεγκτή με ένα περιφερειακό.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
name	ID	1..1	Το όνομα που θα δοθεί στη σύνδεση
Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
PERIPHERAL	Composition-Σύνθεση	1..1	Το περιφερειακό της συγκεκριμένης σύνδεσης
BOARD	Composition-Σύνθεση	1..1	Ο μικροελεγκτής της συγκεκριμένης σύνδεσης
POWER_CONNECTION	Composition-Σύνθεση	0..*	Οι ακροδέκτες που χρησιμοποιούνται για την τροφοδοσία
HW_CONNECTION	Composition-Σύνθεση	1..*	Οι ακροδέκτες που χρησιμοποιούνται για τη σύνδεση μεταξύ των διεπαφών υλικού
COM_ENDPOINT	Composition-Σύνθεση	0..1	Τα χαρακτηριστικά σύνδεσης σε κάποιον broker

Πίνακας 5.21: Ιδιότητες και Συσχετίσεις του *CONNECTION*.

Περιορισμοί

Δεν υπάρχουν περιορισμοί.

5.2.4 PERIPHERAL

Σύνοψη

Το στοιχείο αυτό περιγράφει το περιφερειακό που χρησιμοποιείται σε μία σύνδεση.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
device	ID	1..1	Το όνομα του περιφερειακού
number	INT	0..1	Ο αριθμός του περιφερειακού σε περίπτωση που χρησιμοποιείται το ίδιο μοντέλο πολλαπλές φορές στο συγκεκριμένο σύστημα

Πίνακας 5.22: Ιδιότητες του *PERIPHERAL*.

Δεν περιλαμβάνει περαιτέρω συσχετίσεις.

Περιορισμοί

Δεν υπάρχουν περιορισμοί.

5.2.5 BOARD

Σύνοψη

Το στοιχείο αυτό περιγράφει τον μικροελεγκτή που χρησιμοποιείται σε μία σύνδεση.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
device	ID	1..1	Το όνομα του μικροελεγκτή
number	INT	0..1	Ο αριθμός του μικροελεγκτή σε περίπτωση που χρησιμοποιείται το ίδιο μοντέλο πολλαπλές φορές στο συγκεκριμένο σύστημα

Πίνακας 5.23: Ιδιότητες του *BOARD*.

Δεν περιλαμβάνει περαιτέρω συσχετίσεις.

Περιορισμοί

Δεν υπάρχουν περιορισμοί.

5.2.6 POWER_CONNECTION

Σύνοψη

Το στοιχείο αυτό περιγράφει τη σύνδεση (ακροδέκτες) τροφοδοσίας των περιφερειακών από τους μικροελεγκτές.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
board_power	ID	1..1	Ο ακροδέκτης τροφοδοσίας του μικροελεγκτή
peripheral_power	ID	1..1	Ο ακροδέκτης τροφοδοσίας του περιφερειακού

Πίνακας 5.24: Ιδιότητες του *POWER_CONNECTION*.

Δεν περιλαμβάνει περαιτέρω συσχετίσεις.

Περιορισμοί

Δεν υπάρχουν περιορισμοί.

5.2.7 COM_ENDPOINT

Σύνοψη

Το στοιχείο αυτό περιγράφει τα χαρακτηριστικά της σύνδεσης μιας συσκευής σε έναν broker.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
topic	NEW_ID	1..1	Το όνομα του topic στο οποίο θα στο οποίο θα γίνει publish ή subscribe (ανάλογα αν χρησιμοποιείται αισθητήρας ή ενεργοποιητής αντίστοιχα)
wifi_ssid	NEW_ID	1..1	Το όνομα του wifi δικτύου στο οποίο θα συνδεθεί ο μικροελεγκτής
wifi_password	NEW_ID	1..1	Ο κωδικός του wifi δικτύου στο οποίο θα συνδεθεί ο μικροελεγκτής
addr	ADDRESS_ID	1..1	Η IPv6 διεύθυνση του broker με τον οποίο θα επικοινωνήσει ο μικροελεγκτής
port	INT	1..1	Η πύλη σύνδεσης του broker
Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
MSG_ENTRIES	Composition-Σύνθεση	1..1	Τα είδη μηνυμάτων που θα διαμοιραστούν κατά τη συγκεκριμένη σύνδεση
FREQUENCY	Composition-Σύνθεση	0..1	Η συχνότητα με την οποία θα γίνονται publish τα μηνύματα στον broker

Πίνακας 5.25: Ιδιότητες και Συσχετίσεις του *COM_ENDPOINT*.

Περιορισμοί

Τα topic, wifi_ssid και wifi_password μπορούν να πάρουν τιμές σαν ID, με επιπλέον επιλογή να περιλαμβάνουν και παύλες. Αυτό επιτυγχάνεται σύμφωνα με το NEW_ID που είναι μια κανονική έκφραση (regex).

Το addr μπορεί να πάρει τιμές σαν μια διεύθυνση IPv6 (Internet Protocol version 6). Αυτό επιτυγχάνεται σύμφωνα με το ADDRESS_ID που είναι μια κανονική έκφραση (regex).

5.2.8 MSG_ENTRIES

Σύνοψη

Το στοιχείο αυτό περιγράφει τα είδη μηνυμάτων που θα διαμοιραστούν σε μια συγκεκριμένη σύνδεση.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
msg_entries	MSG_TYPES (Enum)	1..*	Τα είδη μηνυμάτων

Πίνακας 5.26: Ιδιότητες του *MSG_ENTRIES*.

Δεν περιλαμβάνει περαιτέρω συσχετίσεις.

Περιορισμοί

Επιλογές των ειδών μηνυμάτων που μπορούν να δηλωθούν:

- Distance
- Temperature
- Humidity
- Gas
- Pressure
- Env
- Acceleration
- Motor_Controller
- Leds_Controller
- Servo_Controller

5.2.9 HW_CONNECTION

Σύνοψη

Το στοιχείο αυτό είναι η abstract κλάση για την περιγραφή των συνδέσεων των διεπαφών υλικών των συσκευών.

Ιδιότητες και Συσχετίσεις

Δεν περιλαμβάνει περαιτέρω ιδιότητες και συσχετίσεις.

Περιορισμοί

Δεν υπάρχουν περιορισμοί.

5.2.10 GPIO

Σύνοψη

Το στοιχείο αυτό περιγράφει τη σύνδεση δύο GPIO διεπαφών.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
type	STRING	1..1	Το είδος διεπαφής (στην προκειμένη περίπτωση gpio)
board_int	ID	1..1	Η διεπαφή του μικροελεγκτή
peripheral_int	ID	1..1	Η διεπαφή του περιφερειακού
Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
HW_CONNECTION	SuperType-Επέκταση	-	Το στοιχείο GPIO επεκτείνει το στοιχείο HW_CONNECTION

Πίνακας 5.27: Ιδιότητες και Συσχετίσεις του GPIO.

Περιορισμοί

Το type θα πρέπει να έχει την τιμή "gpio", αλλιώς θα εμφανιστεί σφάλμα.

5.2.11 I2C

Σύνοψη

Το στοιχείο αυτό περιγράφει τη σύνδεση μέσω πρωτοκόλλου I2C.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
type	STRING	1..1	Το είδος διεπαφής (στην προκειμένη περίπτωση i2c)
board_int	list[ID]	1..1	Οι διεπαφές του μικροελεγκτή
peripheral_int	list[ID]	1..1	Οι διεπαφές του περιφερειακού
slave_addr	ID	1..1	Η διεύθυνση της διεπαφής που λειτουργεί ως slave
Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
HW_CONNECTION	SuperType-Επέκταση	-	Το στοιχείο I2C επεκτείνει το στοιχείο HW_CONNECTION

Πίνακας 5.28: Ιδιότητες και Συσχετίσεις του I2C.

Περιορισμοί

Το type θα πρέπει να έχει την τιμή "i2c", αλλιώς θα εμφανιστεί σφάλμα.

5.2.12 SPI

Σύνοψη

Το στοιχείο αυτό περιγράφει τη σύνδεση μέσω πρωτοκόλλου SPI.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
type	STRING	1..1	Το είδος διεπαφής (στην προκειμένη περίπτωση spi)
board_int	list[ID]	1..1	Οι διεπαφές του μικροελεγκτή
peripheral_int	list[ID]	1..1	Οι διεπαφές του περιφερειακού
Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
HW_CONNECTION	SuperType-Επέκταση	-	Το στοιχείο SPI επεκτείνει το στοιχείο HW_CONNECTION

Πίνακας 5.29: Ιδιότητες και Συσχετίσεις του SPI.

Περιορισμοί

Το type θα πρέπει να έχει την τιμή "spi", αλλιώς θα εμφανιστεί σφάλμα.

5.2.13 UART

Σύνοψη

Το στοιχείο αυτό περιγράφει τη σύνδεση μέσω πρωτοκόλλου UART.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
type	STRING	1..1	Το είδος διεπαφής (στην προκειμένη περίπτωση uart)
board_int	list[ID]	1..1	Οι διεπαφές του μικροελεγκτή
peripheral_int	list[ID]	1..1	Οι διεπαφές του περιφερειακού
baudrate	ID	1..1	Το baudrate που χρησιμοποιείται
Συσχετίσεις			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
HW_CONNECTION	SuperType-Επέκταση	-	Το στοιχείο UART επεκτείνει το στοιχείο HW_CONNECTION

Πίνακας 5.30: Ιδιότητες και Συσχετίσεις του *UART*.

Περιορισμοί

Το type θα πρέπει να έχει την τιμή "uart", αλλιώς θα εμφανιστεί σφάλμα.

5.2.14 FREQUENCY

Σύνοψη

Το στοιχείο αυτό περιγράφει τη συχνότητα με την οποία θα γίνονται publish τα μηνύματα στον broker.

Ιδιότητες και Συσχετίσεις

Ιδιότητες			
Όνομα	Τύπος	Πολλαπλότητα	Περιγραφή
val	INT	1..1	Η τιμή της συχνότητας
val	FREQ_UNIT	1..1	Η μονάδα μέτρησης της συχνότητας

Πίνακας 5.31: Ιδιότητες του *FREQUENCY*.

Δεν περιλαμβάνει περαιτέρω συσχετίσεις.

Περιορισμοί

Επιλογές των υποστηριζόμενων μονάδων μέτρησης για το `max_freq`:

- hz
- khz
- ghz
- mhz

5.3 ΓΡΑΜΜΑΤΙΚΗ ΤΩΝ DSL

Τα μοντέλα που απαρτίζουν την παρούσα εργασία, υπακούουν στους κανόνες που θέτουν τα δύο μετα-μοντέλα που αναλύθηκαν στο [υποκεφάλαιο 5.1](#) και στο [υποκεφάλαιο 5.2](#). Για τη δημιουργία τους, αναπτύχθηκαν δύο γλώσσες κειμένου μέσω των οποίων ορίζονται και περιγράφονται οι συσκευές και οι μεταξύ τους συνδέσεις.

Οι γλώσσες αυτές σχεδιάστηκαν με το εργαλείο `textX` και σε αυτήν την ενότητα περιγράφεται αναλυτικά το συντακτικό τους.

5.3.1 Γενικό Συντακτικό

Και στις δύο γλώσσες που αναπτύχθηκαν, υπάρχουν κάποιοι κοινοί κανόνες συγγραφής.

Αρχικά, κάθε ιδιότητα των μοντέλων δηλώνεται γράφοντας το όνομά της, στη συνέχεια τον χαρακτήρα ":" και τέλος την τιμή της. Για παράδειγμα:

```
name: value
```

Παράδειγμα ανάθεσης τιμής σε ιδιότητα

Δεύτερος γενικός κανόνας είναι αυτός του τρόπου συγγραφής λίστας. Στην περίπτωση δηλαδή, όπου η τιμή της ιδιότητας είναι μια λίστα από τιμές. Τότε, για κάθε στοιχείο της λίστας γράφουμε αρχικά τον χαρακτήρα "-" και στη συνέχεια τον τύπο του. Για παράδειγμα, τα pins μιας συσκευής μπορεί να είναι power, io_pin, input_pin ή output_pin. Οπότε κάποια από τα pins ενός αισθητήρα θα μπορούσαν να είναι τα εξής.

```
pins:
  - power:
    name: vcc
    number: 1
    type: 5v
  - input_pin:
    name: data_in
    number: 3
  - output_pin:
    name: data_out
    number: 4
```

Παράδειγμα ορισμού pins αισθητήρα

5.3.2 Συντακτικό Συσκευής

Μια συσκευή μπορεί να είναι είτε μικροελεγκτής είτε περιφερειακό (αισθητήρας/ενεργοποιητής). Στην αρχή του αρχείου το πρώτο που δηλώνουμε είναι αυτό, γράφοντας "Board:" για τους μικροελεγκτές και "Peripheral:" για τα περιφερειακά. Τα περισσότερα χαρακτηριστικά είναι κοινά και για τα δύο, επομένως ότι εξηγηθεί παρακάτω εφαρμόζεται με τον ίδιο τρόπο είτε έχουμε μικροελεγκτή είτε περιφερειακό. Στο τέλος θα αναλυθούν ξεχωριστά τα χαρακτηριστικά που δηλώνονται μόνο για μικροελεγκτή και αυτά που δηλώνονται μόνο για περιφερειακό.

Αξίζει να σημειωθεί πως τα χαρακτηριστικά για την κάθε συσκευή, δεν έχει σημασία με ποια σειρά θα δηλωθούν.

Βασικές Ιδιότητες

Κάποιες ιδιότητες είναι αρκετά εύκολο να συγγραφούν, καθώς αποτελούν μια απλή ανάθεση τιμής. Ο τρόπος συγγραφής τους είναι ο ακόλουθος:

```
name: value
riot_name: value
vcc: value
operating_voltage: value
```

Το `name` είναι το όνομα που δίνουμε στη συσκευή και δέχεται ως τιμή αλφαριθμητικό. Το `riot_name` είναι το όνομα της συσκευής όπως αναγνωρίζεται από το RIOT, και δέχεται ως τιμή αλφαριθμητικό με επιπλέον επιλογή να περιλαμβάνει και παύλες. Το `vcc` και το `operating_voltage` είναι η τάση τροφοδοσίας και η τάση λειτουργίας της συσκευής αντίστοιχα, και παίρνουν ως τιμή αριθμό κινητής υποδιαστολής.

Pins

Για τα pins της εκάστοτε συσκευής, δηλώνουμε το είδος του pin, και ανάλογα το είδος μπορεί να έχει συγκεκριμένα χαρακτηριστικά. Ο γενικός τρόπος σύνταξης είναι ο ακόλουθος (όπου μπορούμε να έχουμε πολλά pins και το καθένα να έχει πολλά χαρακτηριστικά):

```
pins:
  - pin_type:
      attribute: value
```

Το `pin_type` μπορεί να είναι ένα εκ των `"power"`, `"io_pin"`, `"input_pin"` και `"output_pin"`. Σε όλες τις περιπτώσεις, όπου `name` είναι είναι ένα αλφαριθμητικό που δηλώνει το όνομα του ακροδέκτη, και `number` ένας ακέραιος αριθμός που υποδηλώνει την θέση του ακροδέκτη στη συσκευή. Επίσης, όπου υπάρχει το `vmax` υποδηλώνει την μέγιστη τιμή τάσης που μπορεί να δεχτεί ο ακροδέκτης (η συγκεκριμένη ιδιότητα δεν είναι υποχρεωτική).

power

```
power:
  name: value
  number: value
  type: value
```

Όπου `type` πρέπει να είναι ένα εκ των `"gnd"`, `"5v"` και `"3v3"`.

input_pin

```
input_pin:
  name: value
  number: value
  vmax: value
```

output_pin

```
output_pin:
  name: value
  number: value
  vmax: value
```

io_pin

```
io_pin: -> function1 , function2 , function3
    name: value
    number: value
    vmax: value
```

Το `io_pin` πέρα από τα `name`, `number` και `vmax`, έχει μια επιπλέον ιδιότητα, η οποία είναι οι λειτουργίες που μπορεί να έχει. Για να τις ορίσουμε, αρχικά γράφουμε τους χαρακτήρες `"->"` και στη συνέχεια γράφουμε τις λειτουργίες, χωρίζοντάς τις με το χαρακτήρα `","` (εφόσον είναι περισσότερες από μία). Οι λειτουργίες που μπορεί να έχει ένας ακροδέκτης είναι οι ακόλουθες:

- `gpio`
- `sda`: SDA σε I2C διεπαφή
- `scl`: SCL σε I2C διεπαφή
- `mosi`: MOSI σε SPI διεπαφή
- `miso`: MISO σε SPI διεπαφή
- `sck`: SCK σε SPI διεπαφή
- `cs`: CS σε SPI διεπαφή
- `tx`: TX σε UART διεπαφή
- `rx`: RX σε UART διεπαφή
- `pwm`
- `adc`
- `dac`

Στην περίπτωση των `pin` τύπου I2C, SPI, UART και PWM ακολουθεί και ο χαρακτήρας `"-"` ώστε μετά από αυτόν να μπει ένας αριθμός, ο οποίος συμβολίζει τον αριθμό της διεπαφής στην οποία ανήκει το `pin`.

Ένα παράδειγμα με μερικά από τα `pins` ενός μικροελεγκτή θα μπορούσε να είναι το ακόλουθο:

```
pins:
    ...
    ...
    - power:
        name: gnd_1
        number: 14
        type: gnd
    - io_pin: -> gpio , mosi-1 , adc
        name: p_13
        number: 15
    - io_pin: -> gpio , rx-1
        name: p_9
        number: 16
    - io_pin: -> gpio , tx-1
        name: p_10
        number: 17
```

```

- io_pin: -> gpio
  name: p_11
  number: 18
- power:
  name: power_5v
  number: 19
  type: 5v
...
...

```

Επιπλέον χαρακτηριστικά ενός Board

Memory

```

memory:
  ram: value unit
  rom: value unit
  flash: value unit

```

Όπου value είναι ακέραιος αριθμός που δηλώνει το μέγεθος της μνήμης και unit μπορεί να είναι ένα εκ των "b", "mb", "kb" και "gb". Δεν χρειάζεται να περιλαμβάνονται και τα τρία είδη μνήμης (ram, rom, flash), αλλά τουλάχιστον ένα από αυτά.

Cpu

```

cpu:
  cpu_family: value
  max_freq: value unit
  fpu: value

```

Όπου cpu_family ένα εκ των "ESP32" και "ESP8266". Το max_freq παίρνει ως τιμή έναν αριθμό, και το unit παίρνει για τιμή ένα εκ των "hz", "ghz" και "mhz". Τέλος το fpu είναι είναι τύπου boolean. 2990 Network

```

network:
  - network_type:
    attribute: value

```

Όπου network_type είναι ένα εκ των "wifi" και "ethernet". Όπως βλέπουμε και παρακάτω, και στις δύο περιπτώσεις υπάρχει το όνομα (name) του δικτύου το οποίο δέχεται ως τιμή ένα αλφαριθμητικό. Στην περίπτωση του wifi έχουμε επιπλέον την επιλογή freq η οποία δέχεται ως value έναν αριθμό, και ως unit ένα εκ των "hz", "mhz" και "ghz". Το network δεν είναι υποχρεωτικό στοιχείο.

- wifi

```

wifi:
  name: value
  freq: value unit

```


- ethernet

```
ethernet:  
  name: value
```

Bluetooth

```
bluetooth:  
  version: value
```

Η τιμή της έκδοσης (version) είναι ένας αριθμός. Το bluetooth δεν είναι υποχρεωτικό στοιχείο.

Επιπλέον χαρακτηριστικά ενός Peripheral

Type

```
type: value
```

Το μόνο επιπλέον χαρακτηριστικό που έχει ένα περιφερειακό πέρα από τις βασικές ιδιότητες, είναι ο τύπος, που μπορεί να έχει ως τιμή ένα εκ των "sensor" και "actuator".

5.3.3 Συντακτικό Συνδέσεων

Κάθε μοντέλο βασισμένο στο μετα-μοντέλο συνδέσεων αποτελείται από ένα σύστημα. Το σύστημα αυτό αποτελείται από μία ή περισσότερες συνδέσεις μεταξύ μικροελεγκτών και περιφερειακών. Κάθε μία από τις συσκευές που χρησιμοποιείται στο σύστημα, αρχικά πρέπει να δηλωθεί με την ακόλουθη σύνταξη:

```
include value
```

Όπου value το όνομα της εκάστοτε συσκευής, όπως έχει δοθεί στο configuration file της (αρχείο κατασκευασμένο σύμφωνα με τους κανόνες του συντακτικού που αναλύεται στην [ενότητα 5.3.2](#))).

Στη συνέχεια, για κάθε μία από τις συνδέσεις που υπάρχουν στο σύστημα, γράφεται η λέξη "connection" και ο χαρακτήρας ":" και ακολουθούν οι ιδιότητές της.

Ιδιότητες

Αρχικά βλέπουμε τρεις ιδιότητες (name, board, peripheral) οι οποίες έχουν αρκετά απλή σύνταξη, απλής ανάθεσης τιμής.

```
connection :
  name: value
  board: value (number)
  peripheral: value (number)
```

Και οι τρεις ιδιότητες παίρνουν ως value κάποιο αλφαριθμητικό. Το board και το peripheral έχουν και την επιλογή (όχι υποχρεωτικό) να έχουν και έναν αναγνωριστικό αριθμό (το number), τοποθετημένο ανάμεσα στους χαρακτήρες "(" και ")". Η δυνατότητα αυτή παρέχεται για την περίπτωση όπου σε πολλαπλές συνδέσεις χρησιμοποιείται συσκευή με το ίδιο όνομα.

power_connections

```
power_connection :
  - board_pin1 -- peripheral_pin1
  - board_pin2 -- peripheral_pin2
  ...
  ...
```

Η συνδέσεις τροφοδοσίας συντάσσονται ως μια λίστα από αντιστοιχίσεις των pins του μικροελεγκτή, με αυτά του περιφερειακού. Άρα για κάθε σύνδεση pin, αρχικά αναγράφεται το όνομα του pin του μικροελεγκτή, και στη συνέχεια το όνομα του pin του περιφερειακού. Τα δύο ονόματα διαχωρίζονται από τους χαρακτήρες "--". Κάθε στοιχείο της λίστας ξεκινάει με τον χαρακτήρα "-".

hw_connections

Η συνδέσεις διεπαφών υλικού, έχουν διαφορετικό τρόπο σύνταξης ανάλογα με τον τύπο της σύνδεσης. Οι τύποι σύνδεσης που υποστηρίζονται είναι οι ακόλουθοι:

- gpio
- i2c
- spi
- uart

Ωστόσο, σε όλες τις περιπτώσεις η σύνταξη ξεκινάει με τον χαρακτήρα "-", τον τύπο σύνδεσης και τέλος τον χαρακτήρα ":", όπως φαίνεται παρακάτω (όπου type ο τύπος):

```
- type :
```

→ gpio

```
hw_connections :
  ...
  ...
  - gpio: board_pin1 -- peripheral_pin1
  - gpio: board_pin2 -- peripheral_pin2
  ...
  ...
```

Οι συνδέσεις gpio συντάσσονται ακριβώς όπως οι συνδέσεις τροφοδοσίας, με τη μόνη προσθήκη του "gpio:" το οποίο είναι ο τύπος σύνδεσης όπως αναφέρθηκε προηγουμένως.

→ i2c

```
hw_connections:
    ...
    ...
    - i2c:
        sda: board_pin -- peripheral_pin
        scl: board_pin -- peripheral_pin
        slave_address: value
    ...
    ...
```

Οι συνδέσεις i2c συντάσσονται αρχικά με τον τύπο της σύνδεσης ("- i2c:") και στη συνέχεια μια λίστα με τα στοιχεία της σύνδεσης αυτής. Τα sda και scl συντάσσονται όπως και το gpio, ενώ το slave_address παίρνει τιμή έναν δεκαεξαδικό αριθμό της μορφής "0x00" (πρώτα οι χαρακτήρες "0x" και στη συνέχεια ο δεκαεξαδικός).

→ spi

```
hw_connections:
    ...
    ...
    - spi:
        mosi: board_pin -- peripheral_pin
        miso: board_pin -- peripheral_pin
        sck: board_pin -- peripheral_pin
        cs: board_pin -- peripheral_pin
    ...
    ...
```

Οι συνδέσεις spi συντάσσονται αρχικά με τον τύπο της σύνδεσης ("- spi:") και στη συνέχεια όπως και οι συνδέσεις i2c, με τη διαφορά ότι δεν υπάρχει slave_address, και τα είδη των pin αντί για sda και scl είναι mosi, miso, sck και cs.

→ uart

```
hw_connections:
    ...
    ...
    - uart:
        tx: board_pin -- peripheral_pin
        rx: board_pin -- peripheral_pin
        baudrate: value
    ...
    ...
```

Οι συνδέσεις uart συντάσσονται αρχικά με τον τύπο της σύνδεσης ("- uart:") και στη συνέχεια όπως και οι συνδέσεις i2c, με τη διαφορά ότι αντί για slave_address

υπάρχει το στοιχείο baudrate, που παίρνει ως τιμή κάποιον ακέραιο αριθμό, και τα είδη των pin αντί για sda και scl είναι tx και rx.

communication_endpoint

```
communication_endpoint:
  topic: value
  wifi_ssid: value
  wifi_passwd: value
  address: value
  port: value
  msg: value
  frequency: value unit
```

Τελευταία ιδιότητα μιας σύνδεσης είναι τα χαρακτηριστικά για τη σύνδεση σε κάποιον broker. Τα topic, wifi_ssid και wifi_password παίρνουν ως τιμή κάποια αλληλουχία γραμμάτων, αριθμών και συμβόλων χωρίς κενά. Το στοιχείο address παίρνει ως τιμή μια διεύθυνση τύπου IPv6 και το port κάποιον ακέραιο αριθμό. Το frequency (το οποίο δεν είναι υποχρεωτικό) παίρνει ως τιμή κάποιον ακέραιο και ως unit ένα εκ των "hz", "khz", "mhz" και "ghz".

Τέλος το msg είναι το είδος του μηνύματος που θα γίνει publish ή θα ληφθεί από κάποιον subscriber και μπορεί να είναι ένα εκ των "Distance", "Temperature", "Humidity", "Gas", "Pressure", "Env", "Acceleration", "Motor_Controller", "Leds_Controller", "Servo_Controller".

5.4 ΜΕΤΑΣΧΗΜΑΤΙΣΜΟΣ M2M

Αφού δημιουργηθούν τα κατάλληλα αρχεία περιγραφής συσκευών και συνδέσεων, σύμφωνα με τους κανόνες των συντακτικών που αναλύθηκαν στο [υποκεφάλαιο 5.3](#) ξεκινάει η κύρια διαδικασία της παρούσας εργασίας.

Πρώτο βήμα, είναι η δημιουργία μοντέλων για κάθε μια από της συσκευές που χρησιμοποιούνται στην εκάστοτε υλοποίηση, αλλά και για τις μεταξύ τους συνδέσεις. Πριν όμως γίνει η παραγωγή του κατάλληλου κώδικα, που θα αναλυθεί στην επόμενη ενότητα, δημιουργούνται δύο διαγράμματα για το μοντέλο των συνδέσεων, τα οποία βοηθούν στην οπτικοποίηση και άρα καλύτερη αντίληψη από τον χρήστη για τη συνδεσμολογία και ενδοεπικοινωνία του συστήματός του.

Το πρώτο διάγραμμα είναι η παρουσίαση όλων των χαρακτηριστικών των συνδέσεων με τη μορφή οντοτήτων, ενώ το δεύτερο είναι μια οπτικοποίηση των αντιστοιχίσεων των ακροδεκτών με τους οποίους συνδέονται οι συσκευές μεταξύ τους, αλλά και των topic στα οποία επικοινωνεί η εκάστοτε συσκευή.

Για την παραγωγή των διαγραμμάτων αυτών, γίνεται χρήση του εργαλείου PlantUML. Για την παραγωγή των PlantUML αρχείων (βασισμένα στην DSL που

το εργαλείο αυτό υποστηρίζει), γίνεται αρχικά ένας M2M μετασχηματισμός του μοντέλου συνδέσεων στο μοντέλο της DSL του PlantUML. Ο μετασχηματισμός αυτός πραγματοποιείται μέσω δύο *python modules*¹⁴ που δημιουργήθηκαν στα πλαίσια της παρούσας διπλωματικής εργασίας. Ένα παράδειγμα από το κάθε διάγραμμα παρουσιάζεται στο [παράρτημα Β'](#).

5.5 ΠΑΡΑΓΩΓΗ ΚΩΔΙΚΑ

Σε αυτήν την ενότητα θα εξεταστεί η διαδικασία παραγωγής αρχείων κώδικα από την παρούσα εργασία. Τα αρχεία που παράγονται είναι δύο, ένα αρχείο σε γλώσσα C και ένα αρχείο Makefile, από τα οποία θα παραχθεί ένα εκτελέσιμο για να φορτωθεί σε κάποιον μικροελεγκτή. Τα αρχεία αυτά παράγονται μέσω κάποιων πρότυπων αρχείων (*templates*) που δημιουργήθηκαν στα πλαίσια της διπλωματικής. Τα πρότυπα αρχεία παρουσιάζονται στο [παράρτημα Α'](#).

Όπως αναφέρθηκε στο [υποκεφάλαιο 5.4](#) αρχικά παράγονται τα μοντέλα για κάθε μια από της συσκευές που χρησιμοποιούνται στην εκάστοτε υλοποίηση, αλλά και για τις μεταξύ τους συνδέσεις. Η πληροφορία που αντλείται από τα μοντέλα, δίνεται με τη μορφή παραμέτρων στα πρότυπα αρχεία. Με αυτόν τον τρόπο, από τα πρότυπα αρχεία παράγονται τα τελικά αρχεία προς εκτέλεση. Παρακάτω αναλύονται κάποιες βασικές συναρτήσεις και λειτουργίες που υλοποιούνται από τα πρότυπα αρχεία.

5.5.1 Αρχείο κώδικα C

Επικοινωνία με broker

Για την επικοινωνία του μικροελεγκτή με κάποιον broker υλοποιήθηκαν οι παρακάτω συναρτήσεις.

- `con(addr , port)`

Περιγραφή	
Σύνδεση στον broker	
Ορίσματα	
Όρισμα	Επεξήγηση
<code>addr</code>	IPv6 διεύθυνση του broker
<code>port</code>	Πύλη επικοινωνίας MQTT-SN

¹⁴<https://docs.python.org/3/tutorial/modules.html>

- `discon()`

Περιγραφή
Αποσύνδεση από τον broker
Ορίσματα
-

- `pub(topic, data, qos)`

Περιγραφή	
Publish κάποιου μηνύματος σε συγκεκριμένο topic	
Ορίσματα	
Όρισμα	Επεξήγηση
topic	Topic στο οποίο θα γίνει το publish
data	Το μήνυμα που πρόκειται να γίνει publish
qos	Ποιότητα υπηρεσιών (Quality of Service)

- `sub(topic, qos, func)`

Περιγραφή	
Publish κάποιου μηνύματος σε συγκεκριμένο topic	
Ορίσματα	
Όρισμα	Επεξήγηση
topic	Topic στο οποίο θα "ακούει"
qos	Ποιότητα υπηρεσιών (Quality of Service)
func	Η συνάρτηση που θα είναι υπεύθυνη για την εκτέλεση λειτουργιών σε περίπτωση κάποιου publish

Sensor

- `send_<sensor_name>()`

Στην περίπτωση που χρησιμοποιείται αισθητήρας, η συνάρτησή αρχικά κάνει εκκίνηση του αισθητήρα, πραγματοποιεί μία μέτρηση, και τέλος την κάνει publish στο αντίστοιχο topic στον broker. Όπου `sensor_name` θα εμφανιστεί το όνομα του αισθητήρα.

Actuator

- `receive_<actuator_name>()`

Η συνάρτηση αυτή είναι υπεύθυνη ώστε κάθε φορά που γίνεται publish στο αντίστοιχο topic, να καλεί την επόμενη συνάρτηση (`<actuator_name>_on_pub()`) ώστε

να εκτελείται η διαδικασία για τους ενεργοποιητές. Όπου `actuator_name` θα εμφανιστεί το όνομα του ενεργοποιητή.

- `<actuator_name>_on_pub()`

Στην περίπτωση όπου γίνει κάποιο `publish` στο αντίστοιχο `topic`, η συνάρτηση αυτή είναι υπεύθυνη ώστε να αποθηκεύσει το μήνυμα, να κάνει εκκίνηση του ενεργοποιητή και τέλος να δράσει ανάλογα με το είδος του ενεργοποιητή. Όπου `actuator_name` θα εμφανιστεί το όνομα του ενεργοποιητή.

5.5.2 Αρχείο Makefile

Το Makefile είναι αρχείο απαραίτητο για να γίνει `compile` του εκτελέσιμου κώδικα C. Στο RIOT υπάρχουν κάποια υλοποιημένα `modules` τα οποία μπορούν να χρησιμοποιηθούν στις εφαρμογές που αναπτύσσονται με το λειτουργικό αυτό. Στο Makefile γίνονται τα `includes` των απαραίτητων `modules`. Στην παρούσα εργασία χρησιμοποιούνται `modules` που είναι υπεύθυνα για λειτουργίες δικτύωσης, χρονοδιακοπών, καθώς και τα `modules` όλων των περιφερειακών που χρησιμοποιούνται στην εκάστοτε υλοποίηση.

Το όνομα της υλοποίησης, τα ονόματα των περιφερειακών και του μικροελεγκτή καθώς και τα στοιχεία του `wifi` στο οποίο θα συνδεθεί, δίνονται ως παράμετροι στο πρότυπο αρχείο Makefile.

5.6 ΥΠΟΣΤΗΡΙΖΟΜΕΝΕΣ ΣΥΣΚΕΥΕΣ

Για να υποστηρίζεται κάποια συσκευή από τη διαδικασία, είναι απαραίτητο να έχουν πρώτα συγγραφεί κάποια συγκεκριμένα αρχεία.

Στην περίπτωση κάποιου μικροελεγκτή, χρειάζεται να υπάρχει το αντίστοιχο `configuration` αρχείο (`.hwd`) σύμφωνα με το συντακτικό που αναλύεται στην [ενότητα 5.3.2](#).

Στην περίπτωση κάποιου περιφερειακού, χρειάζεται να υπάρχει το αντίστοιχο `configuration` αρχείο (`.hwd`) σύμφωνα με το συντακτικό που αναλύεται στην [ενότητα 5.3.2](#), αλλά και ένα πρότυπο αρχείο κώδικα C, στο οποίο υλοποιούνται κάποιες βασικές λειτουργίες οι οποίες αναλύονται στην [ενότητα 5.5.1](#).

Στα πλαίσια της παρούσας διπλωματικής εργασίας, γράφτηκαν πρότυπα αρχεία, και `configuration` αρχεία (`.hwd`) για 2 μικροελεγκτές, 3 σένσορες και 1 ενεργοποιητή. Οι συσκευές αυτές είναι οι ακόλουθες:

- NODEMCU ESP-32S¹⁵
- WEMOS D1 miniPro¹⁶
- Αισθητήρας αποστασης HC-SR04¹⁷
- Αισθητήρας περιβάλλοντος BME680¹⁸
- Αισθητήρας περιβάλλοντος MPL3115A2¹⁹
- Ενεργοποιητής LED NeoPixel Ring²⁰

¹⁵https://docs.ai-thinker.com/_media/esp32/docs/nodemcu-32s_product_specification.pdf

¹⁶https://www.wemos.cc/en/latest/d1/d1_mini.html

¹⁷<https://www.sparkfun.com/products/15569>

¹⁸<https://shop.pimoroni.com/products/bme680-breakout>

¹⁹<https://www.adafruit.com/product/1893>

²⁰<https://www.adafruit.com/product/1643>

6

Παραδείγματα

6.1 ΠΑΡΑΔΕΙΓΜΑΤΑ ΧΡΗΣΗΣ

Παρακάτω ακολουθούν τρία παραδείγματα που φανερώνουν την χρησιμότητα της παρούσας διπλωματικής. Μέσω των παραδειγμάτων αυτών είναι εμφανές το επίπεδο αφαίρεσης που έχει δημιουργηθεί και η διευκόλυνση που προσφέρει στην ανάπτυξη εφαρμογών IoT.

Στο πρώτο παράδειγμα αναλύεται η ανάπτυξη μιας εφαρμογής για τη λήψη διαφόρων μετρήσεων από περιφερειακά. Στο δεύτερο παράδειγμα περιγράφεται ο τρόπος προσθήκης ενός περιφερειακού το οποίο δεν υποστηρίζεται από την παρούσα εργασία, ενώ στο τρίτο παράδειγμα η προσθήκη ενός μικροελεγκτή.

6.2 ΕΦΑΡΜΟΓΗ ΜΕ 2 ΣΕΝΣΟΡΕΣ ΚΑΙ ΕΝΑΝ ΕΝΕΡΓΟΠΟΙΗΤΗ

Σε αυτό το παράδειγμα αναλύεται η χρήση των παραπάνω εργαλείων για την ανάπτυξη μιας εφαρμογής για τη λήψη διαφόρων μετρήσεων από περιφερειακά. Το σύστημα αποτελείται από έναν μικροελεγκτή NodeMCU ESP32 που αποτελεί την κύρια υπολογιστική μονάδα, ένα σόναρ, έναν αισθητήρα περιβάλλοντος και έναν ενεργοποιητή με LED. Επίσης, χρησιμοποιείται και ένα raspberry pi στο οποίο τρέχει ένας broker. Οι συνδέσεις μεταξύ των συσκευών ορίστηκαν σε ένα αρχείο, χρησιμοποιώντας τη γλώσσα που αναλύεται στην [ενότητα 5.3.3](#). Το αρχείο για την περιγραφή των συνδέσεων που χρησιμοποιήθηκε είναι το ακόλουθο.

```

include srf04
include bme680
include ws281x
include esp32_wroom_32

connection:
  name: sonar_esp32
  board: esp32_wroom_32
  peripheral: srf04
  power_connections:
    - gnd_1 -- gnd
    - power_5v -- vcc
  hw_connections:
    - gpio: p_14 -- echo
    - gpio: p_13 -- trigger
  communication_endpoint:
    topic: srf04.data
    wifi_ssid: Wifi_2.4GHz
    wifi_passwd: okodikos?
    address: 2a02:587:541f:44b6:a15:779:2a6e:f525
    port: 1885
    msg: Distance
    frequency: 5 hz

connection:
  name: bme680_esp32
  board: esp32_wroom_32
  peripheral: bme680
  power_connections:
    - gnd_1 -- gnd
    - power_5v -- vcc
  hw_connections:
    - i2c:
        sda: p_21 -- sda
        scl: p_22 -- scl
        slave_address: 0x76
  communication_endpoint:
    topic: bme680.data
    wifi_ssid: Wifi_2.4GHz
    wifi_passwd: okodikos?
    address: 2a02:587:541f:44b6:a15:779:2a6e:f525
    port: 1885
    msg: Env
    frequency: 2 hz

connection:
  name: ws281x_esp32
  board: esp32_wroom_32
  peripheral: ws281x
  power_connections:
    - gnd_1 -- gnd
    - power_5v -- vcc
  hw_connections:
    - gpio: p_0 -- data_in
  communication_endpoint:

```

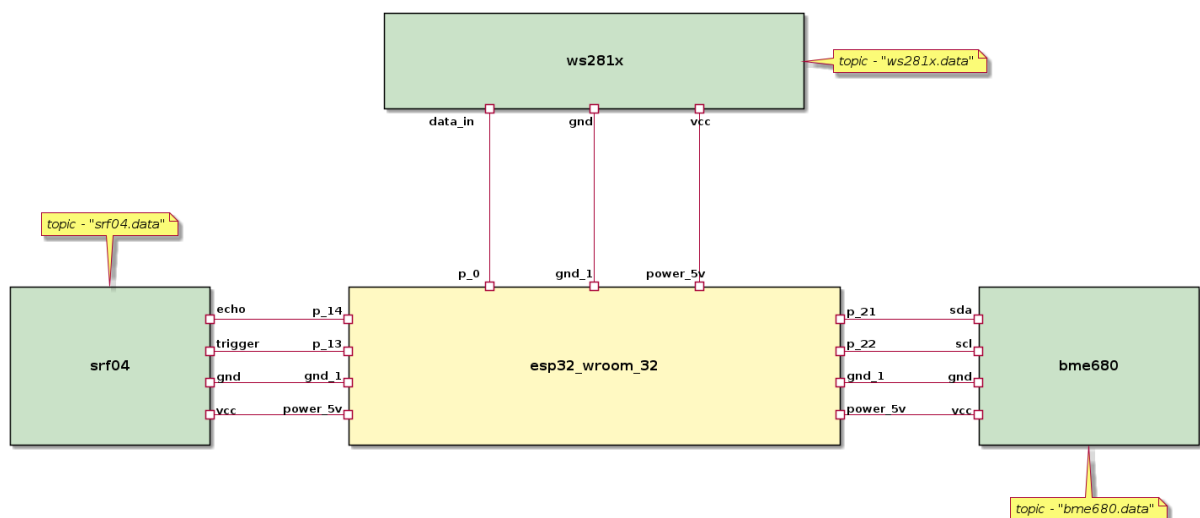
```
topic: ws281x.data
wifi_ssid: Wifi_2.4GHz
wifi_passwd: okodikos?
address: 2a02:587:541f:44b6:a15:779:2a6e:f525
port: 1885
msg: Leds_Controller
```

Για την παραγωγή του κώδικα εκτελέστηκε η παρακάτω εντολή.

```
$ ./parser.py example1
```

Όπου example1 είναι το όνομα του παραπάνω αρχείου. Με την εκτέλεση της εντολής αυτής, παράγεται ένα αρχείο κώδικα C, ένα Makefile, μία εικόνα όπου φαίνεται η συνδεσμολογία των συσκευών και μια εικόνα όπου φαίνονται όλα τα χαρακτηριστικά του μοντέλου των συνδέσεων.. Τα αρχεία δημιουργήθηκαν σύμφωνα με τα πρότυπα αρχεία που παρουσιάζονται στο [παράρτημα Α'](#).

Στο [σχήμα 6.1](#) απεικονίζεται η συνδεσμολογία. Η δεύτερη εικόνα που παράχθηκε παρουσιάζεται στο [παράρτημα Β'](#).



Σχήμα 6.1: Συνδεσμολογία μεταξύ των συσκευών.

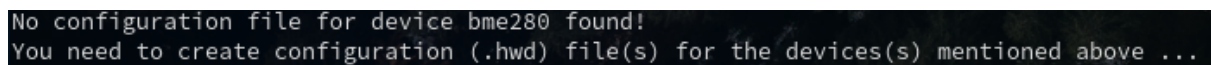
Με την εκτέλεση των παραχθέντων αρχείων ξεκινάει ο έλεγχος των περιφερειακών. Υπάρχουν τα ακόλουθα τερματικά:

- srf04.data: Εδώ κοινοποιούνται οι μετρήσεις από το sonar (απόσταση)
- bme680.data: Εδώ κοινοποιούνται οι μετρήσεις από τον αισθητήρα περιβάλλοντος (θερμοκρασία, υγρασία, πίεση)
- ws281x.data: Εδώ μπορούν να κοινοποιηθούν τιμές RGB, τις οποίες λαμβάνει ο ενεργοποιητής με τα LED και άρα τα ανάβει σύμφωνα με το δοσμένο χρώμα

6.3 ΝΕΟ ΠΕΡΙΦΕΡΕΙΑΚΟ

Για να παραχθεί ο επιθυμητός κώδικας που υλοποιεί βασικές λειτουργίες για κάποιο περιφερειακό, πρέπει πρώτα να έχουν φτιαχτεί 2 συγκεκριμένα αρχεία. Το πρώτο, είναι αυτό που γράφεται στην γλώσσα που αναλύεται στην [ενότητα 5.3.2](#), και είναι αυτό το οποίο ουσιαστικά θα περιγράφει την εκάστοτε συσκευή. Το δεύτερο, είναι ένα πρότυπο αρχείο κώδικα C, το οποίο υλοποιεί τις επιθυμητές λειτουργίες, παίρνοντας ως ορίσματα συγκεκριμένες παραμέτρους. Στην παρούσα εργασία, τα αρχεία αυτά έχουν παραχθεί για τα 4 περιφερειακά που παρουσιάζονται στο [υποκεφάλαιο 5.6](#) και άρα αν ο χρήστης επιθυμεί την προσθήκη κάποιου επιπλέον θα πρέπει να πραγματοποιήσει την ακόλουθη διαδικασία. Εδώ είναι σημαντικό να αναφερθεί, πως για να είναι σχετικά εύκολη η διαδικασία προσθήκης του περιφερειακού, ο χρήστης θα πρέπει να ελέγξει πως υπάρχει ο αντίστοιχος driver, και άρα ήδη υποστηρίζεται από το RIOT. Σε αντίθετη περίπτωση, τότε θα πρέπει ο χρήστης να γράψει από την αρχή τον driver, κάτι το οποίο είναι αρκετά περίπλοκο, και ξεφεύγει και από τα πλαίσια της παρούσας διπλωματικής.

Έστω λοιπόν ότι ο χρήστης θέλει να χρησιμοποιήσει τον αισθητήρα περιβάλλοντος BME280. Αν στο αρχείο (.con) όπου δηλώνονται οι συνδέσεις του κάθε περιφερειακού με τον μικροελεγκτή, συμπεριλάβει τον αισθητήρα αυτόν, τότε η εντολή για την παραγωγή κώδικα θα επιστρέψει το μήνυμα που φαίνεται στο [σχήμα 6.2](#), καθώς δεν υπάρχει έτοιμη υλοποίηση.



```
No configuration file for device bme280 found!
You need to create configuration (.hwd) file(s) for the devices(s) mentioned above ...
```

Σχήμα 6.2: Μήνυμα για συσκευή που δεν υποστηρίζεται

Το πρώτο βήμα που πρέπει να κάνει λοιπόν ο χρήστης, είναι να δημιουργήσει ένα .hwd αρχείο, όπου θα περιγράφει τα χαρακτηριστικά του αισθητήρα (σύμφωνα με τη γλώσσα που υλοποιήθηκε στην παρούσα εργασία).

Το περιεχόμενο του αρχείου θα μπορούσε να είναι το ακόλουθο.

```
peripheral:
  name: bme280
  type: sensor
  operating_voltage: 3.3
  vcc: 3.3
  pins:
    - power:
      name: vcc
      number: 1
      type: 5v
    - power:
      name: gnd
      number: 2
      type: gnd
    - io_pin: -> sck-0
      name: sck
```

```

    number: 3
- io_pin: -> miso-0
    name: miso
    number: 4
- io_pin: -> mosi-0
    name: mosi
    number: 5
- io_pin: -> cs-0
    name: cs
    number: 6

```

Έστω ότι ο χρήστης εκτελεί ξανά την εντολή για την παραγωγή του κώδικα. Αυτή τη φορά θα επιστρέψει το μήνυμα που φαίνεται στο [σχήμα 6.3](#), καθώς δεν υπάρχει το πρότυπο αρχείο C. Η διαδικασία θα σταματήσει, ωστόσο θα δημιουργηθεί το πρότυπο αρχείο (με το όνομα του περιφερειακού) και θα έχει το ακόλουθο περιεχόμενο.

```

void send_{{ peripheral_name[loop.index0] }}(void *arg)
{
    (void) arg;

    /* Name of the topic */
    char topic[32];
    sprintf(topic, "{{topic[loop.index0]}}");

    /* Allocate memory for the message to be published */
    char *msg = malloc(128);

    /*
     * You need to fill the rest of this function. This function
     * should first initialize the sensor, get a measurement,
     * and then publish it to the broker.
     */

    return NULL;
}

```

```

No template for peripheral bme280 found!

A template for each one of the peripheral(s) mentioned above
was created. You need to fill it with appropriate code ...

```

Σχήμα 6.3: Μήνυμα για περιφερειακό για το οποίο δεν υπάρχει το πρότυπο αρχείο.

Στο σημείο των σχολίων, ο χρήστης πρέπει να προσθέσει τον κώδικα με τον οποίο θα υλοποιηθούν οι επιθυμητές λειτουργίες. Στην περίπτωση αυτή, όπου πρόκειται για έναν αισθητήρα, θα πρέπει να γίνει πρώτα η αρχικοποίησή του, στη συνέχεια να πραγματοποιήσει μία μέτρηση, και να την κοινοποιήσει στον broker. Στην περίπτωση ενός ενεργοποιητή, θα πρέπει να γραφτεί κώδικας ώστε αρχικά να αποθηκεύεται το κοινοποιημένο μήνυμα, να γίνεται η αρχικοποίηση του ενεργοποιητή, και στη συνέχεια να πραγματοποιείται η κατάλληλη λειτουργία σύμφωνα με το μήνυμα που κοινοποιήθηκε.

Αφού συμπληρωθεί και αυτό το αρχείο, τότε πλέον το περιφερειακό αυτό υποστηρίζεται πλήρως, και άρα μπορεί να χρησιμοποιηθεί σαν όρισμα στη διαδικασία.

6.4 ΝΕΟΣ ΜΙΚΡΟΕΛΕΓΚΤΗΣ

Όπως και στο προηγούμενο παράδειγμα, έτσι και σε αυτό, σε περίπτωση που ο χρήστης θέλει να χρησιμοποιήσει κάποιον μικροελεγκτή για τον οποίο δεν υπάρχει υλοποίηση στην παρούσα εργασία, θα πρέπει να κάνει κάποια έξτρα βήματα.

Αυτή η περίπτωση επέκτασης, είναι λιγότερο χρονοβόρα από την προηγούμενη, καθώς το μόνο που χρειάζεται να γίνει από πλευράς του χρήστη, είναι να δημιουργήσει το αρχείο που γράφεται στην γλώσσα στην οποία αναπτύχθηκε στα πλαίσια της εργασίας, και περιγράφει την εκάστοτε συσκευή. Από κει και πέρα, εφόσον θέλει να χρησιμοποιήσει περιφερειακά που ήδη υποστηρίζονται, δε χρειάζεται τίποτα επιπλέον, καθώς ο κώδικας σε C που είναι απαραίτητος ώστε να λειτουργήσουν, είναι ήδη υλοποιημένος.

Έστω λοιπόν ότι ο χρήστης θέλει να χρησιμοποιήσει τον μικροελεγκτή WeMos D1 mini Pro ESP8266. Αν στο αρχείο (.con) όπου δηλώνονται οι συνδέσεις του κάθε περιφερειακού με τον μικροελεγκτή, συμπεριλάβει τον ελεγκτή αυτόν, τότε η εντολή για την παραγωγή κώδικα θα επιστρέψει το μήνυμα που φαίνεται στο [σχήμα 6.4](#), καθώς δεν υπάρχει έτοιμη υλοποίηση.

```
No configuration file for device wemos_d1_mini found!
You need to create configuration (.hwd) file(s) for the device(s) mentioned above ...
```

Σχήμα 6.4: Μήνυμα για συσκευή που δεν υποστηρίζεται

Άρα λοιπόν και πάλι ο χρήστης πρέπει να δημιουργήσει ένα .hwd αρχείο, όπου θα περιγράφει τα χαρακτηριστικά του μικροελεγκτή (σύμφωνα με τη γλώσσα που υλοποιήθηκε στην παρούσα εργασία).

Το περιεχόμενο του αρχείου θα μπορούσε να είναι το ακόλουθο.

```
board:
  name: wemos_d1_mini
  vcc: 3.3
  operating_voltage: 3.3
  memory:
    flash: 16 mb
  cpu:
    cpu_family: ESP8266
    max_freq: 160 mhz
    fpu: false
  network:
    - wifi:
        name: wifi_1
        freq: 2.5 ghz
  pins:
```

```
- io_pin: ->
  name: rst
  number: 1
- io_pin: -> adc
  name: a0
  number: 2
  vmax: 3.2
- io_pin: -> gpio
  name: d0
  number: 3
- io_pin: -> gpio, sck-0
  name: d5
  number: 4
- io_pin: -> gpio, miso-0
  name: d6
  number: 5
- io_pin: -> gpio, mosi-0
  name: d7
  number: 6
- io_pin: -> gpio, cs-0
  name: d8
  number: 7
- power:
  name: power_3v3
  number: 8
  type: 3v3
- io_pin: -> tx-0
  name: tx
  number: 9
- io_pin: -> rx-0
  name: rx
  number: 10
- io_pin: -> gpio, scl-0
  name: d1
  number: 11
- io_pin: -> gpio, sda-0
  name: d2
  number: 12
- io_pin: -> gpio
  name: d3
  number: 13
- io_pin: -> gpio
  name: d4
  number: 14
- power:
  name: gnd
  number: 15
  type: gnd
- power:
  name: power_5v
  number: 16
  type: 5v
```

Αφού συμπληρωθεί αυτό το αρχείο, τότε πλέον ο μικροελεγκτής αυτός υποστηρίζεται πλήρως, και άρα μπορεί να χρησιμοποιηθεί σαν όρισμα στη διαδικασία.

Συμπεράσματα και Μελλοντικές επεκτάσεις

7.1 ΣΥΜΠΕΡΑΣΜΑΤΑ

Η παρούσα διπλωματική εργασία παρουσιάζει μια DSL που παρέχει τη μοντελοποίηση συσκευών και της μεταξύ τους επικοινωνίας σε IoT συστήματα, που χρησιμοποιούν το λειτουργικό RIOT. Επίσης, παρέχει στον χρήστη τη δυνατότητα να παράξει αυτόματα κώδικα προς εκτέλεση, για συγκεκριμένους μικροελεγκτές και περιφερειακά, προσαρμοσμένο στις παραμέτρους που δίνει στα μοντέλα του. Ο εκτελέσιμος κώδικας, εκτελεί κάποιες βασικές λειτουργίες που μπορεί να χρειαστούν σε ένα IoT σύστημα.

Κατ' αυτόν τον τρόπο, ο χρήστης μπορεί να κατασκευάσει σε ένα πιο αφαιρετικό επίπεδο το IoT σύστημα που επιθυμεί, χωρίς να χρειάζεται να είναι πλήρως τεχνολογικά καταρτισμένος στον χαμηλού επιπέδου κώδικα που απαιτείται για εφαρμογές σε λειτουργικά συστήματα πραγματικού χρόνου, όπως και αυτό που χρησιμοποιείται στην παρούσα εργασία, το RIOT.

Τα διαγράμματα που επίσης παράγονται, μπορούν να δώσουν στον χρήστη μια καλύτερη αντίληψη του συστήματος που θέλει να δημιουργήσει, όσον αφορά τη συνδεσμολογία και τον τρόπο επικοινωνίας μεταξύ των συσκευών.

Τέλος, βάσει των κανόνων της DSL, ο χρήστης μπορεί εύκολα και έγκαιρα να αντιληφθεί πιθανά λάθη που έχει στο σύστημά του, και άρα να γλιτώσει χρόνο στη δημιουργία του.

7.2 ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

Οι δυνατότητες που παρέχονται από την παρούσα εργασία, είναι σε αρκετά αρχικό στάδιο ενός IoT συστήματος. Επομένως, μια πολύ ενδιαφέρουσα επέκταση θα ήταν να συγγραφούν πρότυπα αρχεία κώδικα για περισσότερες λειτουργίες του εκάστοτε περιφερειακού, και άρα χρήστης να έχει τη δυνατότητα επιλογής.

Επίσης, όπως ήδη αναφέρθηκε, τα εργαλεία που αναπτύχθηκαν υποστηρίζουν μόνο ένα συγκεκριμένο αριθμό μικροελεγκτών και περιφερειακών και ένα λειτουργικό σύστημα, το RIOT. Στον κόσμο του IoT υπάρχει πληθώρα συσκευών και λειτουργικών συστημάτων που το καθένα προσφέρει μοναδικές λειτουργίες οι οποίες θα μπορούσα να είναι χρήσιμες στους χρήστης. Μια πολλή καλή προσθήκη λοιπόν, θα ήταν το εργαλείο αυτό να δίνει τη δυνατότητα στον χρήστη να χρησιμοποιήσει περισσότερες συσκευές στο σύστημά του, ή ακόμα και να παράγεται κώδικας για πληθώρα λειτουργικών συστημάτων.

Τέλος, ο τρόπος με τον οποίο στήνεται ένα σύστημα από τον χρήστη για το συγκεκριμένο εργαλείο, είναι μέσω ενός εργαλείου κειμένου, κάτι το οποίο θα μπορούσε να είναι αποθαρρυντικό για κάποιους χρήστες. Μια πολύ σημαντική επέκταση θα ήταν η ανάπτυξη ενός εργαλείου όπου η δήλωση των χαρακτηριστικών του εκάστοτε συστήματος, όπως η συνδεσμολογία των συσκευών, θα γίνεται μέσα από ένα γραφικό περιβάλλον.

Βιβλιογραφία

- [1] Abiy Biru Chebudie, Roberto Minerva, and Domenico Rotondi. “*Towards a definition of the Internet of Things (IoT)*“. PhD thesis, 08 2014.
- [2] Amit Kumar Sikder, Giuseppe Petracca, Hidayet Aksu, Trent Jaeger, and A. Selcuk Uluagac. “*A Survey on Sensor-based Threats to Internet-of-Things (IoT) Devices and Applications*“. CoRR, abs/1802.02041, 2018.
- [3] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. “*Model-Driven Software Engineering in Practice*“, volume 1. 09 2012.
- [4] Alberto Rodrigues da Silva. “*Model-driven engineering: A survey supported by the unified conceptual model*“. Computer Languages, Systems & Structures, 43: 139–155, 2015.
- [5] John Stankovic and R. Rajkumar. “*Real-Time Operating Systems: Special Anniversary Issue (Guest Editors: John A. Stankovic, Wolfgang Halang, Kim-Fung Man)*“. Real-Time Systems, 28, 11 2004.
- [6] Burak Karaduman, Moharram Challenger, Raheleh Eslampanah, Joachim Denil, and Hans Vangheluwe. “*Platform-specific Modeling for RIOT based IoT Systems*“. 10 2020.
- [7] A. Salihbegovic, T. Eterovic, E. Kaljic, and S. Ribic. “*Design of a domain specific language and IDE for Internet of things applications*“. In “*2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*“, pages 996–1001, 2015.
- [8] Nicolas Harrand, Franck Fleurey, Brice Morin, and Knut Eilif Husa. “*ThingML: A Language and Code Generation Framework for Heterogeneous Targets*“. In “*Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*“, MODELS ’16, page 125–135, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450343213.
- [9] Imad Berrouyine, Mehdi Adda, Jean-Marie Mottu, Jean-Claude Royer, and Massimo Tisi. “*CyprIoT: framework for modelling and controlling network-based IoT applications*“. In “*the 34th ACM/SIGAPP Symposium*“, Limassol, Cyprus, April 2019. ACM Press.

- [10] Hussein Marah, Raheleh Eslampanah, and Moharram Challenger. “*DSML4TinyOS: Code Generation for Wireless Devices*“. 10 2018.
- [11] David Gay, Philip Levis, Rob von Behren, Matt Welsh, Eric Brewer, and David Culler. “*The nesC Language: A Holistic Approach to Networked Embedded Systems*“. volume 38, page 1, 06 2003. ISBN 1581136625.
- [12] Caglar Durmaz, Moharram Challenger, Orhan Dagdeviren, and Geylani Kardas. “*Modelling Contiki-Based IoT Systems **“. 06 2017.
- [13] Tansu Zafer Asici, Burak Karaduman, Raheleh Eslampanah, Moharram Challenger, Joachim Denil, and Hans Vangheluwe. “*Applying Model Driven Engineering Techniques to the Development of Contiki-Based IoT Systems*“. In “*2019 IEEE/ACM 1st International Workshop on Software Engineering Research Practices for the Internet of Things (SERP4IoT)*“, pages 25–32, 2019.
- [14] Emmanuel Baccelli, Oliver Hahm, Mesut Günes, Matthias Wählich, and Thomas Schmidt. “*RIOT OS: Towards an OS for the internet of things*“. Proceedings - IEEE INFOCOM, 04 2013.
- [15] I. Dejanović, R. Vadera, G. Milosavljević, and Ž. Vuković. “*TextX: A Python tool for Domain-Specific Languages implementation*“. Knowledge-Based Systems, 115:1–4, 2017. ISSN 0950-7051.

Παραρτήματα



Πρότυπα αρχεία παραγωγής κώδικα

Παρακάτω παρουσιάζονται τα διαγράμματα που παράχθηκαν κατά τη διαδικασία του M2M μετασχηματισμού, στα πλαίσια του παραδείγματος στο [υποκεφάλαιο 6.2](#).

```
1  #include <time.h>
2  #include "shell.h"
3  #include "msg.h"
4  #include "fmt.h"
5  #include "xtimer.h"
6  #include "string.h"
7
8  /* Peripheral includes */
9  {%- for name in peripheral_name.values() %}
10 #include "{{ name }}.h"
11 #include "{{ name }}_params.h"
12 {%- endfor %}
13
14 /* MQTT-S includes */
15 #include "net/emcute.h"
16 #include "net/ipv6/addr.h"
17
18 #ifndef EMCUTE_ID
19 #define EMCUTE_ID ("gertrud")
20 #endif
21 #define EMCUTE_PORT ({{ port }})
22 #define EMCUTE_ADDRESS ("{{ address }}")
23
24 #define NUMOFSUBS (16U)
25 #define TOPIC_MAXLEN (64U)
26
27 msg_t queue[8];
```

```

28
29 static emcute_sub_t subscriptions[NUMOFSUBS];
30 static char topics[NUMOFSUBS][TOPIC_MAXLEN];
31
32 {% for i in range(num_of_peripherals) -%}
33 char stack{{ id[loop.index0] }}[THREAD_STACKSIZE_DEFAULT];
34 {% endfor -%}
35 char stack_mqtt[THREAD_STACKSIZE_DEFAULT];
36
37 {% include 'templates/mqtt_con.c.templ' %}
38
39 {% for name in peripheral_name.values() %}
40 {% if peripheral_type[name] == "sensor" %}
41 /*
42  * [ {{name}} {{peripheral_type[name}}} ]
43  * This function gets a sensor measurement with frequency {{ frequency[
44    loop.index0] }} Hz
45  * and publishes it to topic '{{ topic[loop.index0] }}'.
46  */
47 {% elif peripheral_type[name] == "actuator" %}
48 /*
49  * [ {{name}} {{peripheral_type[name}}} ]
50  * This function implements the action that the actuator will do in the
51    event of a
52    * published message in the topic that it listens to ({{topic[loop.index0]
53      }})).
54  */
55 {% endif %}
56 {% include "templates/" + name + ".c.templ" %}
57 {% endfor %}
58
59 int main(void)
60 {
61     printf("This application runs on %s\n", RIOT_BOARD);
62
63     /* Initialize our subscription buffers */
64     memset(subscriptions, 0, (NUMOFSUBS * sizeof(emcute_sub_t)));
65
66     /* Start the emcute thread */
67     thread_create(stack_mqtt, sizeof(stack_mqtt),
68                   THREAD_PRIORITY_MAIN - 1,
69                   0,
70                   emcute_thread,
71                   NULL, "emcute");
72
73     /* Try to connect to the gateway */
74     if (con(EMCUTE_ADDRESS, EMCUTE_PORT))
75         printf("Couldn't connect to broker. The measurements will just be
76           printed instead.\n");
77
78     {% for name in peripheral_name.values() -%}
79     /* Start the {{ name }} thread*/
80     thread_create(stack{{ id[loop.index0] }}, sizeof(stack{{ id[loop.index0]
81       }})),
82                   THREAD_PRIORITY_MAIN - 1,

```

```
78     THREAD_CREATE_STACKTEST,  
79     {% if peripheral_type[name] == "sensor" %}  
80     send_{{ name }},  
81     {% else %}  
82     receive_{{ name }},  
83     {% endif %}  
84     NULL, "{{ name }}" );  
85 {% endfor -%}  
86  
87     return 0;  
88 }
```

Κύριο πρότυπο που καλεί και όλα τα υπόλοιπα


```
1 void *emcute_thread(void *arg)
2 {
3     (void)arg;
4     emcute_run(EMCUTE_PORT, EMCUTE_ID);
5     return NULL; // should never be reached
6 }
7
8 /*
9  * Function to get qos level
10  */
11 unsigned get_qos(int qos)
12 {
13     switch (qos)
14     {
15         case 1:
16             return EMCUTE_QOS_1;
17         case 2:
18             return EMCUTE_QOS_2;
19         default:
20             return EMCUTE_QOS_0;
21     }
22 }
23
24 /*
25  * Function that connects to the MQTT-SN gateway
26  *
27  * - param addr    MQTT-SN Gateway IP address
28  * - param port    MQTT-SN Gateway Port
29  */
30 int con(char *addr, int port)
31 {
32     sock_udp_ep_t gw = {.family = AF_INET6, .port = EMCUTE_PORT};
33     gw.port = port;
34
35     /* parse address */
36     if (ipv6_addr_from_str((ipv6_addr_t *)&gw.addr.ipv6, addr) == NULL)
37     {
38         printf("error parsing IPv6 address\n");
39         return 1;
40     }
41
42     if (emcute_con(&gw, true, NULL, NULL, 0, 0) != EMCUTE_OK)
43     {
44         printf("error: unable to connect to [%s]:%i\n", addr, port);
45         return 1;
46     }
47
48     printf("Successfully connected to gateway at [%s]:%i\n", addr, port);
49     return 0;
50 }
51
52 /*
53  * Function that disconnects from the MQTT-SN gateway
54  */
55 int discon(void)
```

```

56 {
57     int res = emcute_discon();
58
59     if (res == EMCUTE_NOGW)
60     {
61         puts("error: not connected to any broker");
62         return 1;
63     }
64     else if (res != EMCUTE_OK)
65     {
66         puts("error: unable to disconnect");
67         return 1;
68     }
69
70     puts("Disconnect successful");
71     return 0;
72 }
73
74 /*
75  * Function that publishes a message to a topic
76  *
77  * - param topic    Topic in which to publish
78  * - param data     Message to be published
79  * - param qos      Quality of service
80  */
81 int pub(char *topic, char *data, int qos)
82 {
83     emcute_topic_t t;
84     unsigned flags = EMCUTE_QOS_0;
85
86     /* parse QoS level */
87     flags |= get_qos(qos);
88
89     /* Get topic id */
90     t.name = topic;
91     if (emcute_reg(&t) != EMCUTE_OK)
92     {
93         puts("error: unable to obtain topic ID");
94         return 1;
95     }
96
97     /* Publish data */
98     if (emcute_pub(&t, data, strlen(data), flags) != EMCUTE_OK)
99     {
100         printf("error: unable to publish data to topic '%s [%i]'\n",
101             t.name, (int)t.id);
102         return 1;
103     }
104
105     printf("published %s on topic %s\n", data, topic);
106
107     return 0;
108 }
109
110 int sub(char *topic, int qos, void *func)

```

```
111 {
112     unsigned flags = EMCUTE_QOS_0;
113
114     /* parse QoS level */
115     flags |= get_qos(qos);
116
117     /* find empty subscription slot */
118     unsigned i = 0;
119     for (; (i < NUMOFSUBS) && (subscriptions[i].topic.id != 0); i++) {}
120     if (i == NUMOFSUBS)
121     {
122         puts("error: no memory to store new subscriptions");
123         return 1;
124     }
125
126     subscriptions[i].cb = func;
127     strcpy(topics[i], topic);
128     subscriptions[i].topic.name = topics[i];
129     if (emcute_sub(&subscriptions[i], flags) != EMCUTE_OK) {
130         printf("error: unable to subscribe to %s\n", topic);
131         return 1;
132     }
133
134     printf("Now subscribed to %s\n", topic);
135     return 0;
136 }
```

Πρότυπο παραγωγής κώδικα για επικοινωνία MQTT

```

1 void *send_srf04(void *arg)
2 {
3     (void) arg;
4
5     /* Name of the topic */
6     char topic[32];
7     sprintf(topic, "{{topic[loop.index0]}}");
8
9     /* Allocate memory for the message to be published */
10    char *msg = malloc(128);
11
12    /* Fix port parameter for digital sensor */
13    srf04_params_t my_params;
14
15    /* PINS */
16    my_params.trigger = GPIO_PIN( 0, {{args[loop.index0]["trigger"]}} );
17    my_params.echo = GPIO_PIN( 0, {{args[loop.index0]["echo"]}} );
18
19    /* Initialize gpio and interrupt */
20    srf04_t dev;
21    if (srf04_init(&dev, &my_params) == SRF04_OK)
22        printf("SRF04 sensor connected\n");
23    else
24        printf("Failed to connect to SRF04 sensor\n");
25
26    /* Print sensor output with frequency {{ frequency[loop.index0] }} Hz
27    */
28    while (true)
29    {
30        /* Get a sensor measurement */
31        int dist = srf04_get_distance(&dev);
32
33        if (dist == SRF04_ERR_INVALID)
34        {
35            printf("Error: No valid measurement is available\n");
36        }
37        else if (dist == SRF04_ERR_MEASURING)
38        {
39            printf("Error: measurement is in progress\n");
40        }
41        else
42        {
43            /* Create a message to be published */
44            sprintf(msg, "{id: {{ id[loop.index0] }}, SRF04 "
45                "Output: [Distance]: %.2f cm}\n",
46                (float)dist / 10);
47
48            printf("%s", msg);
49
50            /* Publish to the topic */
51            pub(topic, msg, 0);
52        }
53
54        /* Sleep for {{ 1/frequency[loop.index0] }} seconds */
55        xtimer_msleep( 1000 / {{ frequency[loop.index0] }} );

```

```
55     }  
56  
57     return NULL;  
58 }
```

Πρότυπο παραγωγής κώδικα για αισθητήρα BME680

```

1 void *send_bme680(void *arg)
2 {
3     (void) arg;
4
5     /* Name of the topic */
6     char topic[32];
7     sprintf(topic, "{{topic[loop.index0]}}");
8
9     /* Allocate memory for the message to be published */
10    char *msg = malloc(128);
11
12    bme680_t dev[BME680_NUMOF];
13    bme680_params_t myparams[BME680_NUMOF];
14    memcpy(&myparams, &bme680_params, sizeof(bme680_params_t));
15
16    for (unsigned i = 0; i < BME680_NUMOF; i++)
17    {
18        BME680_SENSOR(&dev[i]).amb_temp = 25;
19
20        myparams[i].intf.i2c.addr = 0x{{ args[loop.index0]["slave_address"]
21        }};
22
23        printf("Initialize BME680 sensor %u ... ", i);
24        if (bme680_init(&dev[i], &myparams[i]) != BME680_OK)
25            puts("failed");
26        else
27            puts("OK");
28    }
29
30    /* Print sensor output with frequency {{ frequency[loop.index0] }} Hz
31    */
32    while (true)
33    {
34        struct bme680_field_data data;
35
36        for (unsigned i = 0; i < BME680_NUMOF; i++)
37        {
38            /* trigger one measuerment */
39            bme680_force_measurement(&dev[i]);
40            /* get the duration for the measurement */
41            int duration = bme680_get_duration(&dev[i]);
42            /* wait for the duration */
43            xtimer_msleep(duration);
44            /* read the data */
45            int res = bme680_get_data(&dev[i], &data);
46
47            if (res == 0 && dev[i].sensor.new_fields)
48            {
49                /* Create a message to be published */
50                sprintf(msg, "{id: {{ id[loop.index0] }}, BME680 Output: ");
51                #ifndef MODULE_BME680_FP
52                sprintf(msg + strlen(msg),
53                    "[Temp] = %02d.%02d C, "
54                    "[Pressure] = %" PRIu32 " Pa, "
55                    "[Humidity] = %02" PRIu32 ".%03" PRIu32 " %%" ,

```

```

54     data.temperature / 100, data.temperature % 100,
55     data.pressure, data.humidity / 1000, data.humidity % 1000);
56
57     /* Avoid using measurements from an unstable heating setup */
58     if (data.status & BME680_GASM_VALID_MSK)
59         sprintf(msg + strlen(msg),
60             ", [Gas] = %" PRIu32 " ohms", data.gas_resistance);
61     #else
62         sprintf(msg + strlen(msg),
63             "[Temp] = %.2f C, "
64             "[Pressure] = %.2f Pa, "
65             "[Humidity] %.3f %%",
66             data.temperature, data.pressure, data.humidity);
67
68     /* Avoid using measurements from an unstable heating setup */
69     if (data.status & BME680_GASM_VALID_MSK)
70         sprintf(msg + strlen(msg),
71             ", [Gas] = %.0f ohms", data.gas_resistance);
72     #endif
73     sprintf(msg + strlen(msg), "}\n");
74
75     printf("%s", msg);
76
77     /* Publish to the topic */
78     pub(topic, msg, 0);
79 }
80 else if (res == 0)
81     printf("[bme680]: no new data\n");
82 else
83     printf("[bme680]: read data failed with reason %d\n", res);
84 }
85
86 /* Sleep for {{ 1/frequency[loop.index0] }} seconds */
87 xtimer_msleep( 1000 / {{ frequency[loop.index0] }} );
88 }
89
90 return NULL;
91 }

```

Πρότυπο παραγωγής κώδικα για αισθητήρα BME680

```

1 void *send_mpl3115a2(void *arg)
2 {
3     (void) arg;
4
5     /* Name of the topic */
6     char topic[32];
7     sprintf(topic, "{{topic[loop.index0]}}");
8
9     /* Allocate memory for the message to be published */
10    char *msg = malloc(128);
11
12    /* Initialize the sensor*/
13    mpl3115a2_t dev;
14    int init = mpl3115a2_init(&dev, &mpl3115a2_params[0]);
15    if ( init != MPL3115A2_OK )
16        puts("Error: Failed to initialize device!");
17
18    /* Activate measurement*/
19    if (mpl3115a2_set_active(&dev) == MPL3115A2_OK)
20        printf("MPL3115A2 sensor connected\n");
21    else
22        printf("Failed to activate measurement\n");
23
24    /* Print sensor output with frequency {{ frequency[loop.index0] }} Hz
25    */
26    while (true)
27    {
28        uint32_t pressure;
29        int16_t temp;
30        uint8_t status;
31
32        if ((mpl3115a2_read_pressure(&dev, &pressure, &status) |
33            mpl3115a2_read_temp(&dev, &temp)) != MPL3115A2_OK)
34        {
35            puts("Error: Failed to read values!");
36        }
37        else
38        {
39            /* Create a message to be published */
40            sprintf(msg, "{id: {{ id[loop.index0] }}, MPL3115A2 Output: [
41            Pressure]: %u Pa, "
42            "[Temperature]: %3d.%d C, [State]: %#02x}\n",
43            (unsigned int)pressure, temp / 10, abs(temp % 10), status);
44
45            printf("%s", msg);
46
47            /* Publish to the topic */
48            pub(topic, msg, 0);
49
50            /* Sleep for {{ 1/frequency[loop.index0] }} seconds */
51            xtimer_msleep( 1000 / {{ frequency[loop.index0] }} );
52        }
53
54    return NULL;

```


54 }

Πρότυπο παραγωγής κώδικα για αισθητήρα MPL3115A2

```

1 void ws281x_on_pub(const emcute_topic_t *topic, void *data, size_t len)
2 {
3     char *msg = (char *)data;
4
5     printf("### got publication for topic '%s' [%i] ###\n",
6           topic->name, (int)topic->id);
7     for (size_t i = 0; i < len; i++) {
8         printf("%c", msg[i]);
9     }
10    puts("");
11
12    /* Array to store splitted RGB values (as strings) */
13    char **rgb_str = malloc(3 * sizeof(char*));
14    for (int i = 0; i < 3; i++)
15        rgb_str[i] = malloc(2 * sizeof(char));
16
17    /* Array to store splitted RGB values (as integers) */
18    int *rgb = malloc(3 * sizeof(int));
19
20    ws281x_t dev;
21    ws281x_params_t my_params;
22    memcpy(&my_params, ws281x_params, sizeof(ws281x_params_t));
23    int init;
24
25    /* Connected input pin */
26    my_params.pin = GPIO_PIN( 0, 0 );
27
28    /* Number of LEDs to light up */
29    my_params.numof = 12U;
30
31    uint8_t buf[my_params.numof * WS281X_BYTES_PER_DEVICE];
32    my_params.buf = buf;
33
34    if ( (init = ws281x_init(&dev, &my_params)) != 0 )
35        printf("WS281X initialization failed with error code %d\n", init);
36    else
37        printf("WS281X actuator initialized\n");
38
39    /* Split message to RGB */
40    for (int i = 0; i < 3; i++)
41    {
42        memcpy( rgb_str[i], &msg[2*i], 2 );
43        rgb[i] = (int)strtol(rgb_str[i], NULL, 16);
44    }
45
46    /* Store RGB values in a data structure */
47    color_rgb_t colors = { .r = rgb[0], .g = rgb[1], .b = rgb[2] };
48
49    for (uint16_t i = 0; i < dev.params.numof; i++)
50        ws281x_set(&dev, i, colors);
51
52    ws281x_write(&dev);

```

```
53 }  
54  
55 {% include 'templates/subscriber.c.templ' %}
```

Πρότυπο παραγωγής κώδικα για ενεργοποιητή WS281x

```
1 void *receive_({{ peripheral_name[loop.index0] }})(void *arg)  
2 {  
3     (void) arg;  
4  
5     /* Name of the topic */  
6     char topic[32];  
7     sprintf(topic, "{{ topic[loop.index0] }}");  
8  
9     sub(topic, 0, {{ peripheral_name[loop.index0] }}_on_pub);  
10  
11     return NULL;  
12 }
```

Πρότυπο παραγωγής κώδικα για την subscriber συνάρτηση ενός ενεργοποιητή

```
# name of your application
APPLICATION = {{ connection_conf }}

# If no BOARD is found in the environment, use this default:
BOARD ?= {{ board_name }}

# This has to be the absolute path to the RIOT base directory:
RIOTBASE ?= $(HOME)/RIOT

# Include Peripheral and FMT module
{% for module in module.values() %}
USEMODULE += {{module}}
{% endfor %}

USEMODULE += fmt
USEMODULE += xtimer

# Comment this out to disable code in RIOT that does safety checking
# which is not needed in a production environment but helps in the
# development process:
DEVELHELP ?= 1

# Change this to 0 show compiler invocation lines by default:
QUIET ?= 1

# Include packages that pull up and auto-init the link layer.
# NOTE: 6LoWPAN will be included if IEEE802.15.4 devices are present
USEMODULE += gnrc_netdev_default
USEMODULE += auto_init_gnrc_netif
# Specify the mandatory networking modules for IPv6
USEMODULE += gnrc_ipv6_default
# Include MQTT-SN
USEMODULE += emcute
# Add also the shell, some shell commands
USEMODULE += shell
USEMODULE += shell_commands
USEMODULE += ps
# For testing we also include the ping6 command and some stats
USEMODULE += gnrc_icmpv6_echo
# Optimize network stack to for use with a single network interface
USEMODULE += gnrc_netif_single

# Connect board to wifi
USEMODULE += esp_wifi
CFLAGS += -DESP_WIFI_SSID="{{wifi_ssid}}"
CFLAGS += -DESP_WIFI_PASS="{{wifi_passwd}}"

# Give specific ID to this MQTT node
CFLAGS += -DEMCUTE_ID="{{ connection_conf }}"

include $(RIOTBASE)/Makefile.include
```

Πρότυπο παραγωγής αρχείου Makefile

```

1 void send_{{ peripheral_name[loop.index0] }}(void *arg)
2 {
3     (void) arg;
4
5     /* Name of the topic */
6     char topic[32];
7     sprintf(topic, "{{topic[loop.index0]}}");
8
9     /* Allocate memory for the message to be published */
10    char *msg = malloc(128);
11
12    /*
13     * You need to fill the rest of this function. This function should
14     * first initialize
15     * the sensor, get a measurement, and then publish it to the broker.
16     */
17    return NULL;
18 }

```

Πρότυπο παραγωγής κώδικα για μη υποστηριζόμενο αισθητήρα

```

1 void {{ peripheral_name[loop.index0] }}_on_pub(const emcute_topic_t *
2 topic, void *data, size_t len)
3 {
4     char *msg = (char *)data;
5
6     printf("### got publication for topic '%s' [%i] ###\n",
7     topic->name, (int)topic->id);
8     for (size_t i = 0; i < len; i++) {
9         printf("%c", msg[i]);
10    }
11    puts("");
12
13    /*
14     * You need to fill the rest of this function. This function should
15     * store
16     * the published message, initialize the actuator and then act
17     * accordingly.
18     */
19 }
20
21 {% include 'templates/subscriber.c.tmpl' %}

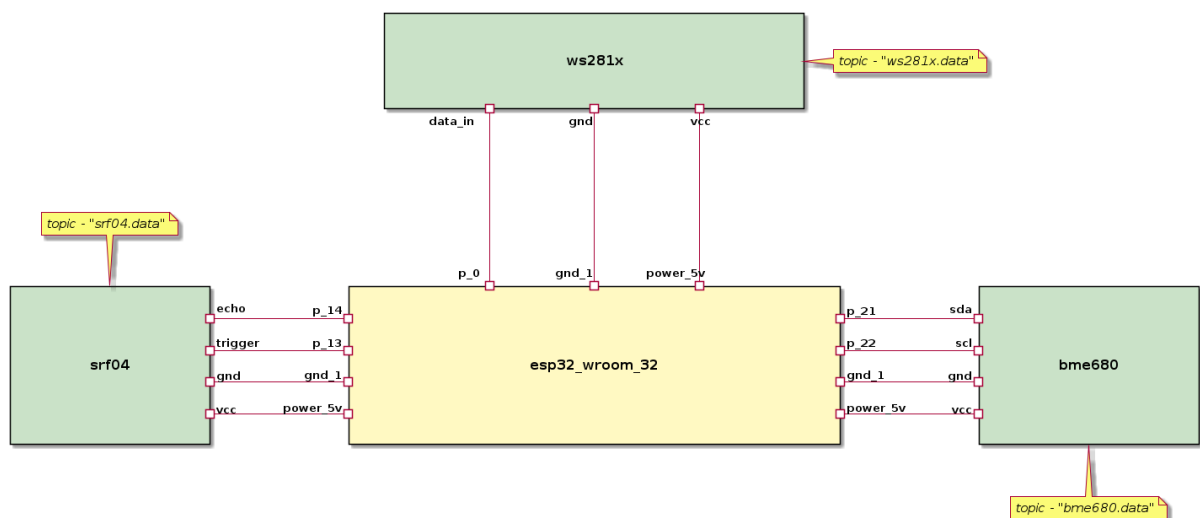
```

Πρότυπο παραγωγής κώδικα για μη υποστηριζόμενο ενεργοποιητή

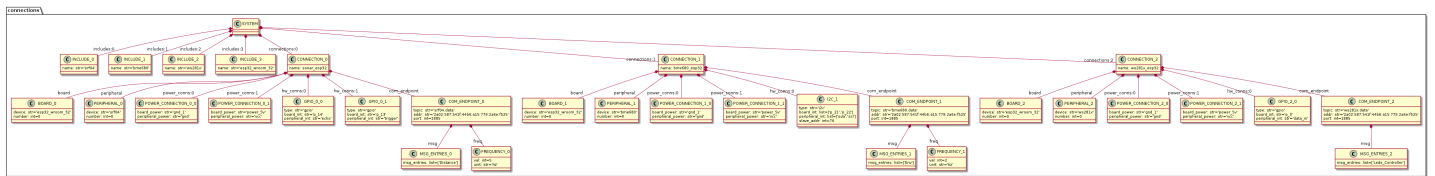
B

Διαγράμματα

Παρακάτω παρουσιάζονται τα διαγράμματα που παράχθηκαν κατά τη διαδικασία του M2M μετασχηματισμού, στα πλαίσια του παραδείγματος της ενότητας [υποκεφάλαιο 6.2](#).



Σχήμα Β.1: Συνδεσμολογία μεταξύ των συσκευών



Σχήμα Β'.2: Χαρακτηριστικά των συνδέσεων