



ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΕΥΦΥΩΝ
ΣΥΣΤΗΜΑΤΩΝ

ΣΥΝΘΕΣΗ ΥΠΗΡΕΣΙΩΝ ΓΙΑ ΤΗΝ ΔΗΜΙΟΥΡΓΙΑ ΕΦΑΡΜΟΓΩΝ ΣΕ
ΠΕΡΙΒΑΛΛΟΝ ΣΗΜΑΣΙΟΛΟΓΙΚΟΥ ΔΙΑΔΙΚΤΥΟΥ ΤΩΝ ΠΡΑΓΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
ΜΠΟΤΩΝΑΚΗΣ ΣΤΥΛΙΑΝΟΣ

Επιτροπή:

Πετράκης Ευριπίδης, Καθηγητής (Επιβλέπων)

Σαμολαδάς Βασίλης, Αναπληρωτής Καθηγητής

Δελγιαννάκης Αντώνιος, Αναπληρωτής Καθηγητής

Χανιά, Οκτώβριος 2019

Περίληψη

Ο «**Σημασιολογικός Ιστός των Πραγμάτων**» (**Semantic Web of Things**) φιλοδοξεί να ανώσει όλες τις συσκευές που ανήκουν στο «Διαδίκτυο των Πραγμάτων» (Internet of Things), αναθέτοντας μία σαφώς ορισμένη έννοια σε όλες τις οντότητες ενός συστήματος, καταφέροντας έτσι την κατανόησή τους σε επίπεδο μηχανής. Έχει σκοπό να εκμηδενίσει την ετερογένεια που προκύπτει από πολλές, διαφορετικές συσκευές που ανήκουν και αλληλεπιδρούν στο Διαδίκτυο των πραγμάτων, με τα διαφορετικά πρωτόκολλα επικοινωνίας και τον διαφορετικό τρόπο χρήσης τους σε εφαρμογές. Αυτό επιτυγχάνεται μέσω προσθήκης μιας σαφώς καθορισμένης και παγκόσμια αναγνωρίσιμης σημασιολογίας σε όλες τις συσκευές του Διαδικτύου, ώστε να υπάρχει κοινή προσέγγιση στην επικοινωνία μεταξύ τους, στην ανταλλαγή δεδομένων και στην αλληλεπίδρασή τους με τον έξω κόσμο, δηλαδή τους χρήστες του Διαδικτύου. Αυτή η εκφραστικότητα γίνεται εφικτή με τη χρήση **Οντολογίας**, η οποία παρέχει τους ορισμούς των εννοιών και των ιδιοτήτων, δηλαδή την εκφραστικότητα που χρειάζονται όλες τις συσκευές που ανήκουν στον Ιστό. Συγκεκριμένα, οι οντολογίες περιλαμβάνουν ορισμούς των εννοιών του Διαδικτύου των Πραγμάτων (π.χ. αισθητήρες, υπηρεσίες) και των ιδιοτήτων τους (π.χ. μετρήσεις αισθητήρων) μέσω δυαδικών σχέσεων. Η γλώσσα ερωτήσεων SPARQL μπορεί να χρησιμοποιηθεί για την αναζήτηση πληροφοριών σε οντολογίες και οι μηχανισμοί συλλογισμού όπως ο Pellet προσφέρουν τον πολύτιμο μηχανισμό που χρειάζεται για την εύρεση ασυνεπειών και για την εξαγωγή νέων πληροφοριών από τις πληροφορίες που υπάρχουν στο Διαδίκτυο των Πραγμάτων και αναπαριστώνται στις οντολογίες. Στη παρούσα εργασία παρουσιάζεται μια αρχιτεκτονική Σημασιολογικού Ιστού των Πραγμάτων στο Υπολογιστικό Νέφος, υιοθετώντας μια υπηρεσιοκεντρική αρχιτεκτονική, που επιτρέπει την ένταξη συσκευών του Διαδικτύου στον Σημασιολογικό Ιστό και την χρήση τους σε εφαρμογές. Υποστηρίζεται επίσης, η αυτοματοποιημένη δημιουργία εφαρμογών Σημασιολογικού Ιστού, που θα εξυπηρετούν τον έλεγχο της λειτουργίας ενός οικο-συστήματος 2 επιπέδων: επίπεδο πόλης και επίπεδο σπιτιού. Οι εφαρμογές ελέγχουν και ερμηνεύουν σημασιολογικά τις συνθήκες που επικρατούν σε μια πόλη ή σπίτι και ρυθμίζουν την ομαλή λειτουργία του οικοσυστήματος των συσκευών που ανήκουν στο σύστημα. Παρουσιάζεται επίσης, μία οντολογία, κατασκευασμένη έτσι ώστε να περιγράψει επαρκώς ένα οικοσύστημα αισθητήρων οι οποίοι παρατηρούν καιρικές συνθήκες σε πόλη και σε σπίτι.

Περίληψη στα αγγλικά

The **Semantic Web Of Things** aims to unify all the devices that belong to the Internet of Things, assigning a clearly defined meaning to all entities of a system, accomplishing their understanding at machine level. It 's purpose is to nullify the heterogeneity that results from many, different devices that belong and interact in the Internet of Things, with their different communication protocols and the different usage in applications. That is being achieved by adding a well-defined and globally recognizable semantic to all the devices if the Internet, so that there can be a common approach in the communication between them, their data exchange and their interaction with the outside world, meaning the Internet users. This expressivity is made possible by the use of **Ontology**, which provides definitions of concepts and properties, that is, the expressiveness needed by all devices that belong on the Web. Specifically, ontologies contain definitions of concepts of the Internet of Things (eg sensors, services) and their properties (eg sensor measurements) through binary relationships. The query language SPARQL can be used for the discovery of information in ontologies and reasoning mechanics such as Pellet provide the valuable mechanism that is necessary to find inconsistencies and extract new information from the information available on the Internet of Things and represented by the ontologies. This paper presents a Semantic Web of Things architecture on the Cloud, adopting a service oriented architecture, that enables the integration of devices to the Semantic Web and their usage in applications. The automated application creation of the Semantic Web is also supported. These applications serve to control the functioning of a 2 level ecosystem: city level and home level. Applications control and interpret semantically the conditions in a city or home and regulate the smooth functioning of the ecosystem of devices belonging to the system. An ontology created so that it adequately describe an ecosystem of sensors that observe weather in the city and at home, is also presented.

Ευχαριστίες (Acknowledgements): Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου, κύριο Ευριπίδη Πετράκη, για την πολύτιμη βοήθειά του, την στήριξη και την καθοδήγηση που μου προσέφερε με τις συμβουλές του. Επίσης, θα ήθελα να ευχαριστήσω όλα τα μέλη του εργαστηρίου για την άριστη επικοινωνία και συνεργασία που είχαμε, και ιδιαίτερα το συμφοιτητή μου, Αιμίλιο Τζαβάρα, με τον οποίο συνεργαστήκαμε με επιτυχία κατά την παράλληλη εκπόνηση των διπλωματικών μας εργασιών.

Περιεχόμενα

Περιεχόμενα	6
Κεφάλαιο 1	9
1.1 Σκοπός της εργασίας	10
1.1.1 Διαδίκτυο των πραγμάτων (IoT) και Υπολογιστικό Νέφος (Cloud)	10
1.1.2 Ο ορισμός του προβλήματος (Problem Definition)	11
1.2 Μεθοδολογία και Συνεισφορά της Εργασίας	12
1.3 Δομή της εργασίας	15
Κεφάλαιο 2	15
2.1 Web Thing Model	16
2.2 Semantic Web Of Things	17
2.3 Σχετικές Εργασίες	17
2.4 Σχετικές Τεχνολογίες	18
2.4.1 Fiware	18
2.4.2 Υπηρεσίες στο Fiware	18
2.4.2.1 Publish/Subscribe Context Broker - Orion Context Broker	19
2.4.2.2 Identity Manager (IdM) - Keyrock	20
2.4.2.3 Authorization PDP – AuthZForce	20
2.4.2.4 PEP Proxy – Wilma	20
2.4.2.5 Short Time HISTORIC (STH) COMET	21
2.4.2.6 Cygnus	21
2.4.2.7 Backend Device Manager - IDAS GE	21
2.4.3 RESTful Υπηρεσίες Ιστού	22
2.4.4 Βάση Δεδομένων MongoDB	23

2.4.5	Μοντέλο πληροφορίας NGSI-2	23
2.4.6	OAuth2.0	25
2.4.7	Οντολογίες	26
2.4.8	Οι οντολογίες SSN και SOSA	26
2.4.9	RDF - RDFS	28
2.4.10	OWL	29
2.4.11	SWRL	29
2.4.12	Σημασιολογικός Γράφος	30
2.4.13	SPARQL	31
2.4.14	Virtuoso	31
2.4.15	Java EE (Enterprise Edition)	32
2.4.16	Spring Boot	33
2.4.17	Apache Jena	33
2.4.18	Μηχανισμοί συλλογισμού	34
2.4.19	Protege	34
2.4.20	PHP	35
2.4.21	Google Charts	36
2.4.22	Υπηρεσία Mashup - Node RED	36
Κεφάλαιο 3		37
3.1	Διάγραμμα Αρχιτεκτονικής	37
3.2	Διαχείριση Δεδομένων	39
	Υπηρεσία Αποθήκευσης των Ιστορικών Δεδομένων (History Service)	40
3.3	Υπηρεσία Διακομιστή μεσολάβησης του Ιστού των Πραγμάτων (Web of Things Proxy)	41
3.4	Υπηρεσία Διαχείρισης Συμβάντων και Συνδρομών (Publish - Subscribe Service)	42
3.5	Υπηρεσία Οντολογίας (Ontology Service)	46
3.6	Υπηρεσία Σύνθεσης Εφαρμογών (Mashup Service)	59
3.7	Υπηρεσία Ταυτοποίησης και Εξουσιοδότησης Χρηστών (User Authentication - Authorization)	65
3.8	Υπηρεσία Λήψης Απόφασης Εξουσιοδότησης (Authorization Policy Decision Point).	65
3.9	Διακομιστής Μεσολάβησης Επιβολής Πολιτικής (Policy Enforcement Point Proxy)	66
3.10	Υπηρεσία Λογική Εφαρμογής – Σύστημα Διεπαφής Χρηστών (Application Logic- Web Application)	68
Κεφάλαιο 4		69

4.1	Υλοποίηση Υπηρεσιών	69
4.1.1	Υπηρεσία Ταυτοποίησης και Εξουσιοδότησης Χρηστών - Keyrock Identity Management (Keyrock IdM)	69
4.1.2	Υπηρεσία Λήψης Απόφασης Εξουσιοδότησης (Authorization PDP) – AuthZForce	71
4.1.3	Διακομιστής Μεσολάβησης Επιβολή Πολιτικής - Pep Proxy Wilma	73
4.1.3	Υπηρεσία Διαχείρισης Συμβάντων και Συνδρομών	74
4.1.4	Υπηρεσία Οντολογίας	88
4.1.5	Υπηρεσία Αποθήκευσης των Ιστορικών Δεδομένων	92
	FIWARE Cygnus	92
	FIWARE - COMET	95
4.1.6	Υπηρεσία Mashup	96
4.2	Σύστημα διεπαφής χρηστών	108
Κεφάλαιο 5		118
5.1	Πείραμα 1	123
5.2	Πείραμα 2	126
5.3	Πείραμα 3	130
Κεφάλαιο 6		133
6.1	Συμπεράσματα	133
6.2	Μελλοντικές Επεκτάσεις	134
Βιβλιογραφία - Αναφορές		136

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός της εργασίας

1.1.1 Διαδίκτυο των πραγμάτων (IoT) και Υπολογιστικό Νέφος (Cloud)

Το «Διαδίκτυο των Πραγμάτων» (Internet of Things) είναι ένα δίκτυο από συσκευές που είναι συνδεδεμένες μέσω του Διαδικτύου (Internet), οι οποίες μπορούν να συλλέξουν δεδομένα σε ένα πεδίο εφαρμογής. Το Διαδίκτυο των Πραγμάτων προσφέρει ατελείωτες δυνατότητες για την κοινωνία και τις επιχειρήσεις. Προσφέρει βελτιωμένη και συνεχή συλλογή πληροφορίας, εξοικονομεί χρόνο και χρήμα για τον άνθρωπο καθώς ελαχιστοποιεί την παρέμβασή του, καθώς επίσης προσφέρει απίστευτη ευκολία στην παρακολούθηση του καιρού, της υγείας και γενικά πεδίων της καθημερινής ζωής μας που είναι σημαντικά.

Το Υπολογιστικό Νέφος συνδυάζεται άψογα με το Διαδίκτυο των Πραγμάτων (Internet of Things ή IoT) και δημιουργούν νέες δυνατότητες στο πεδίο της συλλογής και ανάλυσης δεδομένων. Ουσιαστικά πλεονεκτήματα του Υπολογιστικού Νέφους είναι η

απλότητά του, η επεκτασιμότητα και η προσιπή τιμή του (δηλαδή, χωρίς αρχικές επενδύσεις, χαμηλό κόστος λειτουργίας), καθιστώντας το έτσι την βασική υποδομή για την αποθήκευση, επεξεργασία και ανάλυση δεδομένων IoT και το ιδανικό περιβάλλον ανάπτυξης εφαρμογών Διαδικτύου των Πραγμάτων.

Στην τεχνολογία του Υπολογιστικού Νέφους, οι πόροι προβλέπονται και διατίθενται κατά παραγγελία (on-demand), επομένως δίνεται η δυνατότητα στους χρήστες να χρησιμοποιήσουν τους πόρους του Νέφους με βάση τις πραγματικές ανάγκες τους και να χρεωθούν αποκλειστικά για αυτή τη χρήση τους. Επιπλέον, σημαντική είναι η ιδιότητα της επεκτασιμότητας μιας υποδομής Νέφους, η οποία καθιστά πιο εύκολη την εξυπηρέτηση των απαιτήσεων του συνεχώς αυξανόμενου πλήθους χρηστών και εφαρμογών.

Η τεχνολογία του Διαδικτύου των πραγμάτων (IoT) έχει αναπτυχθεί ραγδαία και θα συνεχίσει να αναπτύσσεται. Χαρακτηριστικά παραδείγματα της εφαρμογής του Διαδικτύου των Πραγμάτων είναι οι «φορετές» συσκευές (wearables devices), οι οποίες προσφέρουν ατελείωτες δυνατότητες στο πεδίο της απομακρυσμένης παρακολούθησης της υγείας ασθενών (smart healthcare), η παρακολούθηση οικοσυστημάτων όπως σύστημα αισθητήρων σε σπίτι το οποίο ελέγχει και την αυτόματη λειτουργία εγκατεστημένων συσκευών σύμφωνα με τα επίπεδα θερμοκρασίας ή υγρασίας (smart home - home automation), αντίστοιχο σύστημα σε πόλη (smart city), όπου μπορούμε να λάβουμε μετρήσεις για επίπεδα θερμοκρασίας, υγρασίας, βροχής, ταχύτητας αέρα, καθώς και πολλές άλλες εφαρμογές. Όλα αυτά καθιστούν αναγκαία την εύκολη δημιουργία και παροχή εφαρμογών από ένα σύστημα Διαδικτύου των Πραγμάτων (IoT), οι οποίες να ικανοποιούν τους χρήστες και να τους παρέχουν τα δεδομένα με τρόπο φιλικό και εύκολα κατανοητό προς το χρήστη.

1.1.2 Ο ορισμός του προβλήματος (Problem Definition)

Το Διαδίκτυο των Πραγμάτων προσφέρει την δυνατότητα να συνδεθούν αμέτρητες συσκευές μεταξύ τους και να ανταλλάξουν πληροφορία. Αυτό όμως απαιτεί την ομαλή ενσωμάτωση αυτών των πολυάριθμων και διαφορετικών συσκευών, με τα διαφορετικά πρωτόκολλα επικοινωνίας τους στο Διαδίκτυο των Πραγμάτων. Όσο το Διαδίκτυο μεγαλώνει σε όγκο, τόσο αυξάνεται και η ετερογένεια των δεδομένων και συσκευών. Αυτή η ετερογένεια δημιουργεί πρόβλημα στην ομοιόμορφη κατανόησή όλων των διαφορετικών οντοτήτων που αλληλεπιδρούν μεταξύ τους. Ένα σύστημα πρέπει να είναι σε θέση να καταλάβει τι συσκευές διαθέτει και ποιος είναι ο ρόλος τους, ώστε να επιτευχθεί ομαλή αλληλεπίδραση, επικοινωνία και ανταλλαγή πληροφοριών.

Αυτό γίνεται ιδιαίτερα αισθητό μόλις κάποιος θέλει να ενσωματώσει συσκευές από διάφορους κατασκευαστές σε μια ενιαία εφαρμογή ή σύστημα.

Συμπερασματικά, δημιουργείται η ανάγκη για τις συσκευές του Διαδικτύου και τα δεδομένα που παράγουν να εκπροσωπούνται και να διαχειρίζονται με συνέπεια και επίσημο τρόπο, καθώς επίσης και να υπάρχουν μηχανισμοί και τεχνολογίες που επιτρέπουν την έρευνα και την ανακάλυψη αυτών των συσκευών. Πρέπει να υπάρχει η δυνατότητα οποιαδήποτε συσκευή να μπορεί εύκολα να ενσωματωθεί και να χρησιμοποιηθεί σε οποιαδήποτε εφαρμογή, ανεξάρτητα από τα πρωτόκολλα δικτύωσης που χρησιμοποιούνται. Αυτό επιτυγχάνεται με τον «Ιστό των Πραγμάτων» (Web Of Things), το οποίο στοχεύει να εντάξει όλες τις οντότητες ενός IoT συστήματος στον Ιστό, αξιοποιώντας τα ως υποδομή, καθιστώντας τα προσβάσιμα μέσω διεπαφών.

Ενώ ο Ιστός των Πραγμάτων (Web of Things) εξυπηρετεί πολλά ζητήματα, υπάρχει δυνατότητα βελτίωσης όσον αφορά την ανακάλυψη και γενικά την διαλειτουργικότητα, καθώς και τη δυνατότητα των πρακτόρων να ερμηνεύσουν σωστά πληροφορίες από περιγραφές αντικειμένων. Η παγκόσμια επικοινωνία μηχανών μέσω αυτόνομης πληροφορίας, ανακάλυψης, ανάλυσης, ώστε να ομαδοποιηθούν και να δομηθούν σωστά τα ανταλλασσόμενα δεδομένα, είναι βασικό ζήτημα στον κόσμο του Διαδικτύου των Πραγμάτων (IoT).

Ο Σημασιολογικός Ιστός των Πραγμάτων (Semantic Web of Things) αποτελεί μία επέκταση του Διαδικτύου των Πραγμάτων και του Ιστού των Πραγμάτων, προσφέροντας επιπλέον λειτουργικότητα μέσω της αξιοποίησης των εργαλείων του Σημασιολογικού Ιστού. Ο Σημασιολογικός Ιστός έχει σαν σκοπό τα εργαλεία του να κατανοήσουν την σημασία των διαδικτυακών εφαρμογών και να αιτιολογούν το περιεχόμενό τους, με την χρήση μηχανισμών όπως του σημασιολογικού συλλογισμού (semantic reasoning). Μέσω των εργαλείων αυτών, πηγαίνουμε από το κλασικό μοντέλο αναπαράστασης της πληροφορίας ενός συστήματος Διαδικτύου των Πραγμάτων, σε ένα σημασιολογικό μοντέλο αναπαράστασης της πληροφορίας και έτσι αποκτά μία σαφώς προσδιορισμένη σημασία, με αποτέλεσμα οι μηχανές να μπορούν να “κατανοούν” την πληροφορία και έτσι να την διαχειρίζονται καλύτερα τα δεδομένα, τα οποία έως τώρα απλά τα παρουσίαζαν. Για να επιτευχθεί αυτό συνδυάζονται υπάρχουσες τεχνολογίες όπως οι οντολογίες, οι οποίες αποτελούν τις οργανωμένες συλλογές πληροφορίας, όπου φιλοξενείται η απαραίτητη γνώση (π.χ. ορισμοί εννοιών του Διαδικτύου των Πραγμάτων όπως οι αισθητήρες, τα χαρακτηριστικά τους, κλπ.) και το μοντέλο JSON-LD το οποίο σχετίζεται με την οντολογία και αποτελεί την μορφή στην οποία τα δεδομένα αναπαριστώνται και ανταλλάσσονται. Ουσιαστικά, οι οντολογίες αποτελούν το εργαλείο παράστασης της γνώσης που αφορά το Διαδίκτυο των Πραγμάτων και οτιδήποτε αυτό περικλείει (συσκευές, πρωτόκολλα επικοινωνίας τους, εφαρμογές) και την παρουσιάζει με επίσημο και σαφή τρόπο. Συμπερασματικά, μέσω

του Σημασιολογικού Ιστού και της οντολογίας επιτυγχάνεται περαιτέρω παγκόσμια διαλειτουργικότητα μεταξύ διαφορετικών συστημάτων και οντοτήτων μέσα σε αυτά τα συστήματα, καθώς και χειρισμός της γνώσης μέσω μιας αποδεκτής, κοινής, τυποποιημένης περιγραφής των Things. Έτσι, το πρόβλημα της ασάφειας των συσκευών και των δεδομένων τους βρίσκει μια λύση.

1.2 Μεθοδολογία και Συνεισφορά της Εργασίας

Σκοπός της παρούσας εργασίας είναι ο σχεδιασμός μιας αρχιτεκτονικής **Σημασιολογικού Ιστού των Πραγμάτων (Semantic Web Of Things)** που επεκτείνει τη λειτουργικότητα υπάρχοντων αρχιτεκτονικών, ενσωματώνοντας την σημασιολογία όλων των οντοτήτων που ανήκουν σε αυτές, κάνοντας το σύστημα ικανό να “κατανοήσει” τις συσκευές και τα δεδομένα τους. Προτείνουμε δηλαδή μία αρχιτεκτονική για την αυτοματοποιημένη σύνδεση συσκευών στον Ιστό των Πραγμάτων (Web Of Things), και την επεξεργασία των δεδομένων σε ένα σημασιολογικό πλαίσιο. Ενσωματώνει τεχνολογίες Διαδικτύου των Πραγμάτων, Ιστού των Πραγμάτων και εφαρμόζει τεχνολογίες Σημασιολογικού Ιστού επιτρέποντας την εύκολη παραγωγή χρήσιμων εφαρμογών, προσιτές και εμπλουτισμένες με σημασιολογική πληροφορία. Έτσι, αυτές οι εφαρμογές μπορούν να κατανοήσουν τα δεδομένα που διαχειρίζονται και να παράγουν ένα συλλογιστικά ορθό αποτέλεσμα φιλικό και χρήσιμο προς τον άνθρωπο. Συνεπώς, έπρεπε να σχεδιαστεί και να υλοποιηθεί ένα σύστημα το οποίο υποστηρίζει τη λειτουργικότητα που προβλέπει το Web Thing Model της W3C¹, καθώς επίσης και την επέκταση του συστήματος στο Σημασιολογικό Ιστό. Αυτό επιτυγχάνεται μέσω της δημιουργίας υπηρεσιών που προσθέτουν σημασιολογία σε όλες τις συσκευές του συστήματος, που μπορούν να καταλαβαίνουν, να ερμηνεύουν και να δρουν ανάλογα με τις περιστάσεις.

Για να γίνει αυτό, έπρεπε να χρησιμοποιηθούν και να ενταχθούν οι τεχνολογίες του Σημασιολογικού Ιστού. Τέτοιες τεχνολογίες είναι το **Resource Description Framework (RDF)**, η γλώσσα που εκφράζει τις διάφορες σχέσεις μεταξύ οντοτήτων και συνολικά δημιουργεί μια οντολογία, **Web Ontology Language (OWL)**, με την οποία προστίθεται ο συλλογισμός πάνω την οντολογία προσθέτοντας νόημα πάνω στην RDF, **Semantic Web Rule Language (SWRL)**, για να ορίσουμε κανόνες οι οποίοι εκτελούνται όταν το επιτρέπουν ορισμένες συνθήκες. Τα παραπάνω αποτελούν αναπόσπαστο μέρος της οντολογίας. Η ανάκτηση της χρήσιμης πληροφορίας της

¹ <https://www.w3.org/Submission/wot-model/>

οντολογίας γίνεται μέσω της τεχνολογίας **SPARQL**. Το τελικό αποτέλεσμα είναι η σύνθεση υπηρεσιών που επιτρέπουν την αυτόματη δημιουργία εφαρμογών, οι οποίες κάνουν χρήση των παραπάνω τεχνολογιών, καθιστώντας τες έτσι ικανές να ερμηνεύουν σωστά δεδομένα και να τα χρησιμοποιούν με χρήσιμο στον πραγματικό κόσμο τρόπο.

Η συνεισφορά της παρούσα εργασίας έγκειται στην πρόταση ενός περιβάλλοντος Σημασιολογικού Ιστού του Διαδικτύου των Πραγμάτων (Semantic Web of Things), δηλαδή ενός συστήματος που συνδυάζει τις γνωστές τεχνολογίες του Υπολογιστικού Νέφους με τις νέες τεχνολογίες που προτείνονται από το Σημασιολογικό Ιστό - για παράδειγμα, τα Διασυνδεδεμένα Δεδομένα (Linked Data), το πρότυπο OWL, τις οντολογίες, τη γλώσσα ερωτήσεων SPARQL, κ.ά. Συγκεκριμένα, προτείνεται η χρήση μιας Υπηρεσιοκεντρικής Αρχιτεκτονικής στο Υπολογιστικό Νέφος, η οποία ενσωματώνει τις τεχνολογίες του Σημασιολογικού Ιστού, προσδίδοντας στα δεδομένα μια σαφώς προσδιορισμένη σημασία, αξιοποιώντας τα μεταδεδομένα που αφορούν τις συσκευές / αισθητήρες, και προσφέροντας σε χρήστες σημαντικές λειτουργίες όπως η αποθήκευση, η ανάκτηση, η ενημέρωση και η διαγραφή των παραπάνω πληροφοριών. Ο σχεδιασμός αυτών των λειτουργιών βασίζεται στο μοντέλο “Web Thing Model” που αναπτύχθηκε από την Κοινοπραξία Παγκόσμια Ιστού (W3C) και αναλύεται παραπάνω αλλά και στη συνέχεια της εργασίας. Οι πειραματικές μετρήσεις αποδεικνύουν ότι πρόκειται και για μια αποδοτική αρχιτεκτονική, καθώς ο χρόνος απόκρισης του συστήματος και η κατανάλωση πόρων (CPU και μνήμης RAM) βρίσκονται σε χαμηλά επίπεδα. Επομένως, μέσω της πειραματικής διαδικασίας, καταγράφεται και αναδεικνύεται η αποτελεσματικότητα της προτεινόμενης αρχιτεκτονικής, η οποία συμπεραίνουμε ότι μπορεί να επιδείξει μια σειρά από σημαντικά πλεονεκτήματα απέναντι σε άλλες σχετικές προτεινόμενες λύσεις και αρχιτεκτονικές.

Το σύστημα που υλοποιήθηκε βασίζεται στην εργασία [1]. Προστίθενται όμως υπηρεσίες που ανήκουν στο Web Thing Model της W3C. Πρόκειται για ένα κοινό μοντέλο που έχει ως στόχο την περιγραφή των φυσικών αντικειμένων με χρήση εικονικών οντοτήτων στον Ιστό των Πραγμάτων. Το μοντέλο αυτό έχει καθοριστικό ρόλο. Υιοθετήθηκε η λογική του σχεδιασμού (π.χ. βασικοί κανόνες, υποθέσεις και παραδοχές που διέπουν το σύστημα) και κατ’ επέκταση η υλοποίηση των λειτουργιών που προτείνει το μοντέλο. Η λειτουργικότητα του Web Thing Model υλοποιήθηκε από ξεχωριστή εργασία που διεκπεραιώθηκε παράλληλα [2].

Στην παρούσα εργασία, η αρχιτεκτονική που προκύπτει από τις 2 προηγούμενες εργασίες ([1], [2]) επεκτείνεται περαιτέρω, προσφέροντας υπηρεσίες δημιουργίας εφαρμογών μέσω της σύνθεσης υπάρχουσας γνώσης για το σύστημα και άλλες εφαρμογές. Για να επιτευχθεί αυτό, χρησιμοποιήθηκαν εργαλεία σύνθεσης mashup εφαρμογών όπως το Node-RED², καθώς επίσης και εργαλεία Σημασιολογικού Ιστού

² <https://nodered.org/>

που διευκολύνουν την σύνθεση εφαρμογών με βάση το σκοπό που επιδιώκει να εξυπηρετήσει μια εφαρμογή.

Συνοψίζοντας, στην παρούσα εργασία εργασία προτείνουμε ένα νέο σχεδιασμό συστήματος που ενσωματώνει:

1. Την υπάρχουσα αρχιτεκτονική υπηρεσιών [1].
2. Τις υπηρεσίες του Web of Things Model [2].
3. Υπηρεσίες Σημασιολογικού Ιστού.

Έτσι, δημιουργείται μια εντελώς νέα υπηρεσιακή αρχιτεκτονική για το Διαδίκτυο των πραγμάτων που παρέχει την λειτουργικότητα για:

- Σημασιολογική αναπαράσταση συσκευών και υπηρεσιών.
- Διασύνδεση νέων συσκευών και εύρεση συσκευών και υπηρεσιών.
- Σύνθεση νέων υπηρεσιών χρησιμοποιώντας υπάρχουσα γνώση για συσκευές και υπηρεσίες που παρέχουν.

Στην σχεδίαση του συστήματός μας ακολουθήθηκε μία Υπηρεσιοκεντρική Αρχιτεκτονική (Service Oriented Architecture), όπου κάθε υπηρεσία του συστήματος είναι αυτόνομη και επιτελεί μια ξεχωριστή λειτουργία. Οι υπηρεσίες επικοινωνούν μεταξύ τους μέσω RESTful διεπαφών. Μέσω της Υπηρεσιοκεντρικής Αρχιτεκτονικής αυτής, υπάρχει η δυνατότητα κάθε ξεχωριστή υπηρεσία να αντικατασταθεί με κάποια άλλη, να αναβαθμιστεί, να αφαιρεθεί και συνολικά το σύστημα μπορεί να επεκταθεί πολύ εύκολα, προσθέτοντας επιπλέον λειτουργικότητα.

Για να αποδείξουμε την λειτουργικότητα του συστήματος, υλοποιήσαμε την αρχιτεκτονική με όλες τις υπηρεσίες, δημιουργήσαμε ένα περιβάλλον Διαδικτύου των Πραγμάτων με χιλιάδες εικονικές (simulated) συσκευές που παρέχουν μεγάλο όγκο δεδομένων.

Βασισμένοι στην παραπάνω αρχιτεκτονική εκτιμήσαμε την απόδοση το συστήματος. Αναλύσαμε την απόδοση σε αιτήματα που αφορούν πληροφορία για συσκευές και καταλήξαμε ότι είναι δυνατόν ένα σύστημα παροχής σημασιολογικού διαδικτύου των πραγμάτων να ανταποκρίνεται σε πραγματικό χρόνο ακόμα και σε συνθήκες μεγάλου φόρτου εργασίας (χιλιάδες αιτήματα πολλά από τα οποία εκτελούνται ταυτόχρονα). Αναλύσαμε τους περιορισμούς του συστήματος και συνθήκες που προκαλούν μείωση της απόδοσης και προτείνουμε λύσεις για την αντιμετώπισή τους.

1.3 Δομή της εργασίας

- Στο Κεφάλαιο 2 γίνεται μια σύντομη περιγραφή σχετικών τεχνολογιών καθώς και τεχνολογιών που χρησιμοποιήθηκαν στην παρούσα εργασία.
- Στο Κεφάλαιο 3 περιγράφεται η αρχιτεκτονική του συστήματος και καταγράφονται οι υπηρεσίες που το απαρτίζουν.
- Στο Κεφάλαιο 4 περιγράφεται η τεχνική λύση που αναπτύχθηκε με βάση τις προδιαγραφές του Κεφαλαίου 3.
- Στο Κεφάλαιο 5 εξετάζεται η απόδοση του συστήματος, η οποία μελετήθηκε με τη χρήση συνθετικών - αλλά ρεαλιστικών - δεδομένων που αφορούν αισθητήρες και άλλες συσκευές (τα δεδομένα αυτά δημιουργήθηκαν με σκοπό να μελετήσουμε την απόδοση του συστήματος σε συνθήκες μεγάλου όγκου δεδομένων και υπολογιστικού φορτίου).
- Στο Κεφάλαιο 6 παραθέτουμε κάποια συμπεράσματα και προτάσεις για μελλοντική διερεύνηση.

Κεφάλαιο 2

Υπόβαθρο και Υπάρχουσες Τεχνολογίες

2.1 Web Thing Model

Το Web Thing Model αποτελεί ένα μοντέλο που ορίζεται από την Κοινοπραξία του Παγκόσμιου Ιστού (World Wide Web Consortium ή W3C³) σε 2 συνδυασμό με μια διεπαφή προγραμματισμού εφαρμογών διαδικτύου (Web API) για «Πράγματα» (Things), με σκοπό αυτά να ακολουθούνται από οποιονδήποτε επιθυμεί να δημιουργήσει ένα προϊόν, μια συσκευή, μια υπηρεσία, ή μια εφαρμογή για το Web of Things. Το μοντέλο που προτείνεται από την W3C είναι μια προσπάθεια για να γίνει η αλληλεπίδραση μεταξύ των Πραγμάτων στο Διαδίκτυο των Πραγμάτων προσβάσιμη από τα πρότυπα του Ιστού (Web standards). Με αυτόν τον τρόπο, θα διευκολυνθεί η υλοποίηση των εφαρμογών του Ιστού που χρησιμοποιούν ή ανακτούν δεδομένα από αντικείμενα του πραγματικού κόσμου.

Με τον όρο “Πράγμα” (Web Thing ή απλά Thing) αναφερόμαστε στην εικονική αναπαράσταση ενός φυσικού αντικειμένου. Αυτή η αναπαράσταση αποτελεί τη βάση

³ https://el.wikipedia.org/wiki/WWW_Consortium

των πόρων (resources) του Web Things Model. Σύμφωνα με τις προδιαγραφές και τις απαιτήσεις του μοντέλου της W3C, προκειμένου ένα Πράγμα να εκθέσει τα βασικά χαρακτηριστικά του, πρέπει να υποστηρίζει το μορφότυπο JSON ως προκαθορισμένο τρόπο αναπαράστασης. Συγκεκριμένα, προτείνεται η χρήση μιας σύντομης περιγραφής της μορφής JSON για κάθε Πράγμα (βλ. Ανάκτηση ενός Web Thing).

Με τον όρο **Model** (Μοντέλο ενός Πράγματος) αναφερόμαστε σε μια αναπαράσταση που βασίζεται στο μορφότυπο JSON-LD και η οποία περιγράφει τόσο τις ιδιότητες (properties) όσο και τις ενέργειες (actions) που χαρακτηρίζουν ένα Πράγμα. Αυτή η αναπαράσταση περιέχει υπερσυνδέσμους (hyperlinks) που συνδυάζονται κατάλληλα για τη δημιουργία μιας σημασιολογικής περιγραφής της αντίστοιχης οντότητας. Επομένως, το μοντέλο ενός Πράγματος εκμεταλλεύεται τις τεχνολογίες του Σημασιολογικού Ιστού, με σκοπό την αποσαφήνιση των ιδιοτήτων που έχει το εκάστοτε Πράγμα (π.χ. τι μετράει, πού βρίσκεται, κλπ.).

Οι λειτουργίες - υπηρεσίες που προτείνονται από το Web Thing Model αφορούν κυρίως την ανάκτηση, την προσθήκη, την ενημέρωση - ή ακόμα και τη διαγραφή - δεδομένων σχετικών με τα Πράγματα (για παράδειγμα, την ανάκτηση ή την ενημέρωση μιας περιγραφής JSON ή της περιγραφής JSON-LD ενός Πράγματος). Οι λειτουργίες αυτές και η υλοποίησή τους, όπως αυτή έγινε κατά την εκπόνηση της παρούσας εργασίας, υλοποιήθηκαν από τον συμφοιτητή Αιμίλιο Τζαβάρα στην διπλωματική του εργασία[2], παράλληλα με την παρούσα εργασία.

2.2 Semantic Web Of Things

Μέσω των τεχνολογιών του Σημασιολογικού Ιστού των Πραγμάτων οι υπάρχουσες εφαρμογές, οντότητες και υπηρεσίες του Διαδικτύου των Πραγμάτων που θα αποκτήσουν σημασία και θα αρχίσουν να γίνονται σε ένα βαθμό κατανοητές και από τις μηχανές. Αυτό θα επιτευχθεί χάρη σε μεθόδους και τεχνολογίες όπως αυτή των Διασυνδεδεμένων Δεδομένων (**Linked - Data**)⁴. Τα Διασυνδεδεμένα Δεδομένα χρησιμοποιούνται για τη δημοσιοποίηση δομημένων δεδομένων στο Διαδίκτυο, αξιοποιώντας γνωστές τεχνολογίες του Ιστού (όπως HTTP και URIs) για την μορφοποίηση, με σκοπό, όχι μόνο την εξυπηρέτηση ιστοσελίδων για τους αναγνώστες, αλλά και την επέκταση των τεχνολογιών αυτών προκειμένου να πραγματοποιούν ανταλλαγή πληροφοριών. Αυτή η ανταλλαγή γίνεται με τέτοιο τρόπο ώστε οι υπολογιστές να μπορούν να διαβάζουν τις πληροφορίες αυτόματα. Έτσι, δεδομένα που παρέχουν συνδέσμους σε άλλα δεδομένα, δημιουργούν μια αλληλένδετη σχέση μεταξύ τους, κάτι το οποίο αυξάνει την χρησιμότητά τους και την σύνδεση διαφορετικών πηγών

⁴ https://en.wikipedia.org/wiki/Linked_data

δεδομένων. Τα διασυνδεδεμένα αυτά δεδομένα επιτρέπουν την ενσωμάτωση σε μεγάλη κλίμακα δεδομένων στο Διαδίκτυο και τον συλλογισμό πάνω σε αυτά. Την σύνδεση αυτή εκμεταλλεύονται οι τεχνολογίες του Σημασιολογικού Ιστού όπως τα σημασιολογικά ερωτήματα (semantic queries), για να επιστρέφουν πληροφορίες. Η SPARQL αποτελεί μια γλώσσα ερωτήσεων η οποία ανακτά και διαχειρίζεται Διασυνδεδεμένα Δεδομένα τα οποία είναι αποθηκευμένα σε RDF μορφή.

2.3 Σχετικές Εργασίες

Είναι γνωστό ότι υπάρχουν ήδη πολλές προσεγγίσεις για το σχεδιασμό συστημάτων Διαδικτύου των Πραγμάτων και μεγάλο μέρος από αυτές έχουν εφαρμοστεί σε εμπορικές πλατφόρμες IoT (διευκολύνοντας έτσι την ανάπτυξη εφαρμογών από προγραμματιστές) ή σε προσαρμοσμένες λύσεις εξυπηρετώντας τις ανάγκες ενός συγκεκριμένου πεδίου εφαρμογής. Μερικά παραδείγματα είναι οι εφαρμογές για την ηλεκτρονική υγεία (e-Health), την υποβοηθούμενη διαβίωση (ambient assisted living), την παρακολούθηση της δραστηριότητας των χρηστών, κλπ. Επιπλέον, έχουν δημοσιευθεί προσεγγίσεις που προτείνουν και χρησιμοποιούν τις τεχνολογίες του Σημασιολογικού Ιστού. Αρκετές από αυτές αναφέρονται στην χρήση των τεχνολογιών του Σημασιολογικών Τεχνολογιών για την αξιοποίηση της πληροφορίας που αφορά τους αισθητήρες, τις μετρήσεις τους και τις ενέργειές τους (actuations), για παράδειγμα χρησιμοποιώντας τις οντολογίες SSN και SOSA (αναλύονται στη συνέχεια του Κεφαλαίου). Είναι γεγονός το ότι έχουν δημοσιευθεί ενδιαφέρουσες έρευνες που καλύπτουν σημαντικές πτυχές του σχεδιασμού και της υλοποίησης συστημάτων που αξιοποιούν τεχνολογίες του Σημασιολογικού Ιστού, παραδείγματος χάριν τα άρθρα [5], [6] και [7].

Ωστόσο, αναζητείται μια ολοκληρωμένη και αποδοτική αρχιτεκτονική, η οποία θα βασίζεται στην ιδέα του Διαδικτύου των Πραγμάτων και θα συνδυάζει τις κλασικές τεχνολογίες του Υπολογιστικού Νέφους με τις νέες τεχνολογίες του Σημασιολογικού Ιστού. Ο σχεδιασμός ενός συστήματος που θα συνδυάζει όλα τα παραπάνω και θα επιτρέπει τη συλλογή και την ανάλυση δεδομένων σε πραγματικό χρόνο, αποτελεί πρόκληση που πρέπει να αντιμετωπίσει η μελλοντική δουλειά στα συστήματα IoT.

Στη συνέχεια, θα αναφερθούμε σε σχετικές τεχνολογίες που μπορούν να αποτελέσουν σημαντικά εργαλεία μιας αρχιτεκτονικής Σημασιολογικού Ιστού του Διαδικτύου των Πραγμάτων και αξιοποιούνται κατάλληλα στην παρούσα εργασία.

2.4 Σχετικές Τεχνολογίες

2.4.1 Fiware

Το FIWARE αποτελεί μια πλατφόρμα middleware, η οποία βασίζεται στο λογισμικό Openstack και παρέχει ένα αρκετά απλό αλλά ισχυρό σύνολο διεπαφών προγραμματισμού εφαρμογών (APIs) που διευκολύνουν την ανάπτυξη εφαρμογών. Πρόκειται για ένα πλαίσιο πλατφόρμας ανοικτού κώδικα (open - source), του οποίου οι υπηρεσίες μπορούν να συναρμολογηθούν μεταξύ τους, καθώς και με άλλες υπηρεσίες “τρίτων” με σκοπό την ανάπτυξη της επιτάχυνσης της ανάπτυξης έξυπνων λύσεων. Συγκεκριμένα, το FIWARE προσφέρει υπηρεσίες γενικού σκοπού (GEs), οι οποίες περιλαμβάνουν απλές διεπαφές προγραμματισμού εφαρμογών. Χάρη στις υπηρεσίες αυτές, το FIWARE επιτρέπει την ανάπτυξη και τη διανομή εφαρμογών υπηρεσιοκεντρικής αρχιτεκτονικής στο Υπολογιστικό Νέφος.

2.4.2 Υπηρεσίες στο Fiware

Χάρη στους γενικούς ενεργοποιητές (Generic Enablers) του FIWARE, μπορούμε να πραγματοποιήσουμε μια σειρά λειτουργιών γενικού σκοπού, οι οποίες παρέχονται από σαφώς καθορισμένες διεπαφές προγραμματισμού εφαρμογών (REST APIs). Με αυτόν τον τρόπο, η ανάπτυξη έξυπνων εφαρμογών σε διάφορους τομείς καθίσταται σαφώς πιο εύκολη. Μια υπηρεσία γενικού σκοπού χαρακτηρίζεται από την απλότητα στη χρήση της αλλά και από τη δυνατότητα επαναχρησιμοποίησης, επομένως μπορεί να αποτελεί συστατικό για να δημιουργούνται σύνθετα συστήματα. Ο κατάλογος υπηρεσιών του FIWARE περιλαμβάνει μια πλούσια συλλογή από γενικούς ενεργοποιητές (GE) που παρέχουν στους προγραμματιστές τη δυνατότητα εύκολης και γρήγορης υλοποίησης σύνθετων λειτουργιών, με αποτέλεσμα να διευκολύνεται η διαδικασία ανάπτυξης εφαρμογών. Οι υπηρεσίες του FIWARE είναι όλες ανοικτού κώδικα και διατίθενται στο κοινό δωρεάν. Οι υπηρεσίες του καταλόγου του οικοσυστήματος FIWARE που χρησιμοποιήθηκαν στην παρούσα εργασία είναι οι παρακάτω:

2.4.2.1 Publish/Subscribe Context Broker - Orion Context Broker

Ο Publish/Subscribe Context Broker (Orion Context Broker) του FIWARE συνιστά την υλοποίηση μιας Υπηρεσίας Διαχείρισης Συμβάντων και Συνδρομών και παρέχει για τη διαχείρισή τους RESTful διεπαφές NGSI (μοντέλο αναπαράστασης - ανταλλαγής δεδομένων στο FIWARE, βλ. 2.3.5). Χάρη σε αυτές τις διεπαφές, οι χρήστες έχουν τη δυνατότητα να εκτελέσουν διάφορες λειτουργίες, όπως:

- Εγγραφή οντοτήτων για παρακολούθηση των συμβάντων αυτών.
- Ενημέρωση των πληροφοριών των καταχωρημένων οντοτήτων.
- Ειδοποίηση όταν πραγματοποιούνται αλλαγές στις πληροφορίες των οντοτήτων.
- Ανάκτηση πληροφοριών των εγγεγραμμένων οντοτήτων.
- Διαγραφή οντοτήτων. Η Εικόνα 2.3 δείχνει την επικοινωνία που αναπτύσσει η υπηρεσία Publish/Subscribe Context Broker μέσω των RESTful διεπαφών της, τόσο με χρήστες/άλλες υπηρεσίες (μέσω του port 1026) όσο και με τη βάση δεδομένων MongoDB, όπου αποθηκεύονται οι οντότητες που διαχειρίζεται η υπηρεσία.

2.4.2.2 Identity Manager (IdM) - Keyrock

Η υπηρεσία Keyrock IdM του οικοσυστήματος FIWARE καλύπτει ένα εύρος λειτουργιών - δυνατοτήτων που αφορούν την ασφάλεια του συστήματος, για παράδειγμα η πρόσβαση των χρηστών σε υπηρεσίες, δίκτυα και εφαρμογές, η ασφάλεια και η πιστοποίηση από χρήστες σε συσκευές, δίκτυα και υπηρεσίες και η διαχείριση της εξουσιοδότησης και των προφίλ των χρηστών. Ακόμα, έχει τη δυνατότητα να διατηρεί προσωπικά δεδομένα και να παρέχει άδειες προς εφαρμογές. Όπως θα δούμε και στη συνέχεια, η υπηρεσία βασίζεται σε μεγάλο βαθμό στο πρωτόκολλο εξουσιοδότησης OAuth2.0.

2.4.2.3 Authorization PDP – AuthZForce

Το AuthZForce, αποτελεί ένα μηχανισμό απόφασης, για την έγκριση ή απόρριψη αιτημάτων πρόσβασης από χρήστες ή υπηρεσίες, σε πόρους του συστήματος. Το

AuthZForce εξακριβώνει με βάση τις εφαρμοστέες πολιτικές πρόσβασης του συστήματος (Καθορίζονται από την υπηρεσία Keyrock IDM όπως είδαμε νωρίτερα), εάν κάποιος χρήστης ή υπηρεσία έχει το δικαίωμα πρόσβασης σε ένα συγκεκριμένο πόρο του συστήματος. Η υπηρεσία παρέχει ένα REST API μέσω του οποίου δέχεται αιτήματα πρόσβασης -χρηστών ή υπηρεσιών- προς αξιολόγηση (έγκριση ή απόρριψη αιτήματος πρόσβασης). Απαντάει στην έξοδο της, θετικά εάν το αίτημα πρόσβασης είναι εξουσιοδοτημένο ή αρνητικά διαφορετικά.

2.4.2.4 PEP Proxy – Wilma

Το PEP Proxy-Wilma έχει τον ρόλο του μεσολαβητή (REST) αιτημάτων μεταξύ υπηρεσιών του συστήματος ή μεταξύ χρηστών και υπηρεσιών του συστήματος. Πιο συγκεκριμένα το PEP-Proxy δέχεται από μία υπηρεσία ή χρήστη, ένα αίτημα πρόσβασης σε πόρο του συστήματος. Το PEP-Proxy θα «ρωτήσει» την υπηρεσία AuthZForce (μέσω της REST διεπαφής του AuthZForce) εάν το αίτημα πρόσβασης στον συγκεκριμένο πόρο εγκρίνεται ή απορρίπτεται. Εφόσον το αίτημα εγκριθεί, το PEP-Proxy θα το προωθήσει στην υπηρεσία -του συστήματος- η οποία διαθέτει το συγκεκριμένο πόρο και θα επιστρέψει την απάντηση της στον αιτούντα. Διαφορετικά το αίτημα αγνοείται. Με αυτόν τον τρόπο, μόνο οι εγγεγραμμένοι-εξουσιοδοτημένοι χρήστες και οι εξουσιοδοτημένες υπηρεσίες θα μπορούν να έχουν πρόσβαση σε πόρους του συστήματος.

2.4.2.5 Short Time HISTORIC (STH) COMET

Το STH COMET αποτελεί μια υπηρεσία του FIWARE που πραγματοποιεί τη διαχείριση ιστορικών πληροφοριών χρονικής σειράς. Συγκεκριμένα, έχει τη δυνατότητα ανάκτησης της ιστορικών πληροφοριών από βάσεις δεδομένων (π.χ. από βάση δεδομένων MongoDB). Οι πληροφορίες αυτές προκύπτουν από τις αλλαγές στην τιμή των οντοτήτων που διατηρούνται στην υπηρεσία Orion Context Broker. Κάθε επικοινωνία μεταξύ του FIWARE-COMET και του Orion Context Broker, καθώς και μεταξύ του FIWARE-COMET και οποιουδήποτε “τρίτου” (π.χ. για ανάκτηση δεδομένων), επιτυγχάνεται μέσω τυποποιημένων διεπαφών NGSI.

2.4.2.6 Cygnus

Πρόκειται για μια υπηρεσία του οικοσυστήματος FIWARE που έχει το ρόλο του συνδέσμου μεταξύ του Orion Context Broker (που συνιστά πηγή δεδομένων NGSI2) και πολλών τύπων βάσεων δεδομένων - αποθετηρίων όπως MongoDB, MySQL, CKAN, DynamoDB. Το Cygnus δέχεται ροές δεδομένων NGSI2 και τις αποθηκεύει στην προκαθορισμένη βάση δεδομένων που του έχει ανατεθεί. Έχει επιπλέον τη δυνατότητα να αποθηκεύει ακατέργαστα (Raw) και συγκεντρωτικά (Aggregated) δεδομένα, ενώ δεν δεσμεύει το σύστημα να χρησιμοποιήσει συγκεκριμένου τύπου βάση δεδομένων.

2.4.2.7 Backend Device Manager - IDAS GE

Η υπηρεσία IDAS είναι μια υλοποίηση της υπηρεσίας Διαχείρισης Συσκευών Backend του FIWARE και πραγματοποιεί τη διασύνδεση των αισθητήρων - συσκευών με το Υπολογιστικό Νέφος. Αυτή η υπηρεσία, χρησιμοποιώντας εξειδικευμένους IoT Πράκτορες (Agents), μπορεί να λαμβάνει δεδομένα και να μεταφράζει ειδικά πρωτόκολλα IoT (HTTP, MQTT, Coap, LoRaWAN0.1) στο πρωτόκολλο NGSI2, που είναι το πρότυπο αναπαράστασης - ανταλλαγής δεδομένων του FIWARE. Κάθε IoT Πράκτορας είναι ένα “συστατικό” της εν λόγω υπηρεσίας και εξειδικεύεται σε συγκεκριμένο πρωτόκολλο IoT, το οποίο μεταφράζει σε NGSI2. Παράλληλα, η υπηρεσία IDAS διαθέτει το δικό της μηχανισμό προστασίας, καθώς λαμβάνει δεδομένα μόνο από τις εγγεγραμμένες φυσικές συσκευές οι οποίες έχουν προστεθεί από κάποιο χρήστη στον πίνακα συσκευών της υπηρεσίας. Έτσι, απορρίπτονται αιτήματα από κάθε μη εγγεγραμμένη συσκευή.

2.4.3 RESTful Υπηρεσίες Ιστού

Οι RESTful Υπηρεσίες Ιστού έχουν κατασκευαστεί με σκοπό να λειτουργούν γρήγορα και αποτελεσματικά στον παγκόσμιο ιστό. Το αρχιτεκτονικό πρότυπο

Representational State Transfer (REST)⁵, αξιοποιώντας τις πολύ σημαντικές ιδιότητές του (απόδοση, επεκτασιμότητα και τροποποιησιμότητα), προσφέρει στις Υπηρεσίες Ιστού τη δυνατότητα να λειτουργούν βέλτιστα στον παγκόσμιο ιστό. Σύμφωνα με το πρότυπο REST, τα δεδομένα και η λειτουργικότητα θεωρούνται πόροι (resources) και η πρόσβαση σε αυτούς επιτυγχάνεται από μια αυστηρά καθορισμένη κοινή διεπαφή που βασίζεται στη χρήση των πρότυπων μεθόδων HTTP. Συγκεκριμένα, κάθε πόρος είναι προσβάσιμος μέσω ενός Uniform Resource Identifier (URI), δηλαδή από έναν υπερσύνδεσμο του διαδικτύου. Επιπλέον, κάθε πόρος αποτελείται από μια ταυτότητα και έναν τύπο δεδομένων, ενώ υποστηρίζει και ένα σύνολο ενεργειών. Η ταυτότητα του πόρου είναι το προαναφερθέν URI και οι ενέργειες πραγματοποιούνται από τις τυπικές μεθόδους HTTP: την ανάκτηση / ανάγνωση (GET), την δημιουργία (POST), την ενημέρωση (PUT), και την διαγραφή (DELETE). Με τη λειτουργία GET είναι δυνατή η ανάκτηση της τρέχουσας κατάστασης ενός πόρου (με χρήση κάποιου είδους αναπαράστασης), με τη λειτουργία POST δημιουργείται ένας νέος πόρος, ενώ με τη λειτουργία PUT ενημερώνεται η τρέχουσα κατάσταση του πόρου. Τέλος, με τη λειτουργία DELETE διαγράφεται ένας συγκεκριμένος πόρος.

2.4.4 Βάση Δεδομένων MongoDB

Το MongoDB⁶ είναι ένα σύστημα διαχείρισης βάσεων δεδομένων (DBMS) ανοιχτού κώδικα. Χρησιμοποιεί ένα μοντέλο βάσης δεδομένων που προορίζεται για έγγραφα και υποστηρίζει διάφορες μορφές δεδομένων όπως το μορφότυπο JSON. Συνιστά μια από τις πολυάριθμες μη σχεσιακές τεχνολογίες βάσεων δεδομένων που δημιουργήθηκαν με το banner “NoSQL” προκειμένου να χρησιμοποιηθούν σε μεγάλες εφαρμογές δεδομένων και άλλες εργασίες επεξεργασίας δεδομένων που δεν ταιριάζουν καλά σε ένα άκαμπτο σχεσιακό μοντέλο. Σε αντίθεση με τις σχεσιακές βάσεις δεδομένων που χρησιμοποιούν πίνακες και σειρές, η αρχιτεκτονική του MongoDB εμπεριέχει συλλογές και έγγραφα.

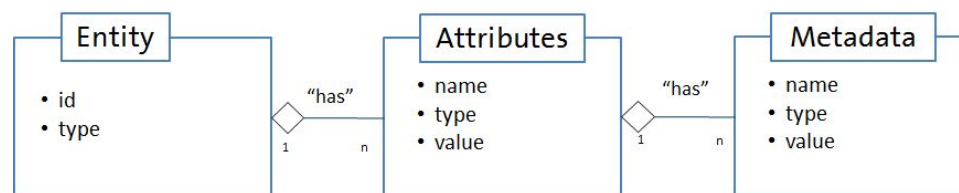
⁵ https://en.wikipedia.org/wiki/Representational_state_transfer

⁶ <https://www.mongodb.com/>

2.4.5 Μοντέλο πληροφορίας NGSI-2

Το API του FIWARE NGSI-2 (Next Generation Service Interface)⁷ ορίζει:

- Ένα μοντέλο δεδομένων για πληροφορίες πλαισίου (context information), που βασίζεται σε ένα απλό πληροφοριακό μοντέλο και χρησιμοποιεί την έννοια των οντοτήτων πλαισίου (βλ. παρακάτω).
- Μια RESTful διεπαφή δεδομένων πλαισίου για την ανταλλαγή πληροφοριών μέσω επερωτήσεων, συνδρομών και ενημερώσεων.
- Μια RESTful διεπαφή διαθεσιμότητας πλαισίου για την ανταλλαγή πληροφοριών σχετικών με το πώς να αποκτήσεις δεδομένα πλαισίου (context data). Το μοντέλο πληροφορίας NGSI βασίζεται σε ένα συγκεκριμένο πρότυπο αναπαράστασης, το οποίο φαίνεται στην εικόνα 2.1. Ακολουθεί μια εκτενής περιγραφή των βασικών συστατικών της NGSI πληροφορίας.



Εικόνα 2.1: Μοντέλο πληροφορίας NGSI.

Entity: Οι οντότητες αποτελούν το κέντρο βάρους στο μοντέλο πληροφοριών FIWARE NGSI. Μια οντότητα αντιπροσωπεύει ένα πράγμα, δηλαδή οποιοδήποτε φυσικό ή λογικό αντικείμενο (π.χ. έναν αισθητήρα, ένα άτομο, ένα δωμάτιο κ.λπ.). Κάθε οντότητα έχει ένα αναγνωριστικό οντότητας (id) και ένα τύπο (type) πχ “Sensor”. Μία οντότητα μπορεί να έχει από ένα έως η χαρακτηριστικά (Attributes). Ένα χαρακτηριστικό είναι “κομμάτι” (Part of) μίας οντότητας.

Attributes: Τα χαρακτηριστικά είναι ιδιότητες των οντοτήτων. Για παράδειγμα, η τρέχουσα ταχύτητα ενός αυτοκινήτου θα μπορούσε να διαμορφωθεί ως χαρακτηριστικό “current_speed” της οντότητας οχήματος “car_104”. Στο μοντέλο δεδομένων NGSI, τα χαρακτηριστικά έχουν ένα όνομα (name), έναν τύπο (type), και μια τιμή (value).

Το όνομα του χαρακτηριστικού περιγράφει το είδος της ιδιότητας που αντιπροσωπεύει η τιμή (value) του χαρακτηριστικού της οντότητας, για παράδειγμα η

⁷ <https://fiware.github.io/specifications/ngsiv2/stable/>

τιμή του `current_speed` αφορά την ταχύτητα (σε δεδομένα, πχ. 120) του αυτοκινήτου. Ο τύπος (type) του χαρακτηριστικού υποδηλώνει τον τύπο δεδομένων που ακολουθεί η τιμή του χαρακτηριστικού (πχ. Float, Int, String κτλ).

Ένα χαρακτηριστικό μπορεί να έχει από ένα έως η μεταδεδομένα (Metadata). Τα μεταδεδομένα είναι “κομμάτι” (Part of) ενός χαρακτηριστικού.

Metadata (Μεταδεδομένα): Τα μεταδεδομένα χρησιμοποιούνται για την περιγραφή της ιδιότητας της τιμής (value) του χαρακτηριστικού στο οποίο αναφέρονται, για παράδειγμα προσδιορισμός της μονάδας μέτρησής του (θα μπορούσε η τιμή των μεταδεδομένων να είναι “Celsius” για το χαρακτηριστικό “temperature”). Οι μεταβλητές όνομα (name), τύπος (type) και τιμή (value) των μεταδεδομένων βασίζονται στους ίδιους κανόνες που ακολουθούν και οι αντίστοιχες μεταβλητές του χαρακτηριστικού (Attribute).

Συνεπώς, μια οντότητα αποτελείται συνήθως από όλα τα παραπάνω στοιχεία (μεταδεδομένα δεν είναι απαραίτητο να υπάρχουν) και αντιπροσωπεύεται από ένα αντικείμενο JSON, του οποίου η σύνταξη βασίζεται στους κανόνες του προτύπου NGSI. Ένα απλό παράδειγμα οντότητας φαίνεται παρακάτω:

```
{
  "id": "Home215",
  "type": "Home",
  "pressure": {
    "type": "Integer",
    "value": 700,
    "metadata": {}
  },
  "temperature": {
    "type": "Float",
    "value": 15,
    "metadata": {}
  }
}
```

Εικόνα 2.2: Παράδειγμα οντότητας σπιτιού σε NGSI αναπαράσταση.

Στην εικόνα 2.2 φαίνεται η NGSI αναπαράσταση ενός σπιτιού (Home) με μοναδικό αναγνωριστικό “Home215”. Το σπίτι έχει το χαρακτηριστικό της πίεσης (pressure), καθώς και αυτό της θερμοκρασίας (temperature). Η τιμή της πίεσης είναι 700 και πρόκειται για έναν ακέραιο αριθμό (Integer), ενώ αντίστοιχα η τιμή της θερμοκρασίας είναι 15 που είναι ένας “float” αριθμός.

2.4.6 OAuth2.0

Το πρωτόκολλο ταυτοποίησης και εξουσιοδότησης OAuth2.0 είναι ένα 7 ανοιχτό πρότυπο για την ανάθεση της πρόσβασης και χρησιμοποιείται συνήθως από τους χρήστες του διαδικτύου ως ένας τρόπος για να επιτρέπουν σε ιστότοπους ή σε εφαρμογές να έχουν πρόσβαση σε άλλους ιστότοπους, χωρίς όμως να τους δίνουν τους κωδικούς πρόσβασης. Γενικά, μέσω του πρωτοκόλλου OAuth2.0, παρέχεται στους πελάτες μια “ασφαλής - εξουσιοδοτημένη πρόσβαση” σε πόρους διακομιστών, καθώς ορίζεται μια διαδικασία για τους κατόχους πόρων που επιτρέπουν την πρόσβαση τρίτων (δηλαδή πελατών) στους πόρους του διακομιστή της υπηρεσίας τους, χωρίς όμως να χρειάζεται να μοιραστούν τα διαπιστευτήριά τους.

Ο μηχανισμός OAuth2.0 έχει σχεδιαστεί έτσι ώστε να λειτουργεί σύμφωνα με το πρωτόκολλο HTTP (Hypertext Transfer Protocol) και στην ουσία επιτρέπει την έκδοση ειδικών αναγνωριστικών πρόσβασης, που ονομάζονται OAuth2 tokens, σε τρίτους πελάτες. Η έκδοση αυτή επιτυγχάνεται μέσω μιας υπηρεσίας εξουσιοδότησης (π.χ. Keyrock IdM), αφού πρώτα δοθεί η έγκριση του ιδιοκτήτη των πόρων. Επομένως, η πρόσβαση στους προστατευόμενους πόρους (που φιλοξενούνται από το διακομιστή πόρων) πραγματοποιείται με τη χρήση του διακριτικού OAuth2 token.

2.4.7 Οντολογίες

Οι οντολογίες⁸ είναι οργανωμένες συλλογές πληροφοριών που 8 περιλαμβάνουν αναπαραστάσεις, επίσημες ονομασίες και ορισμούς κατηγοριών, ιδιοτήτων και σχέσεων μεταξύ των εννοιών, των δεδομένων και των οντοτήτων που απαρτίζουν ένα ή παραπάνω εννοιολογικά πεδία. Οι οντολογίες αξιοποιούνται από γνωστές τεχνολογίες του Σημασιολογικού Ιστού όπως η γλώσσα ερωτήσεων SPARQL.

⁸ [https://en.wikipedia.org/wiki/Ontology_\(information_science\)](https://en.wikipedia.org/wiki/Ontology_(information_science))

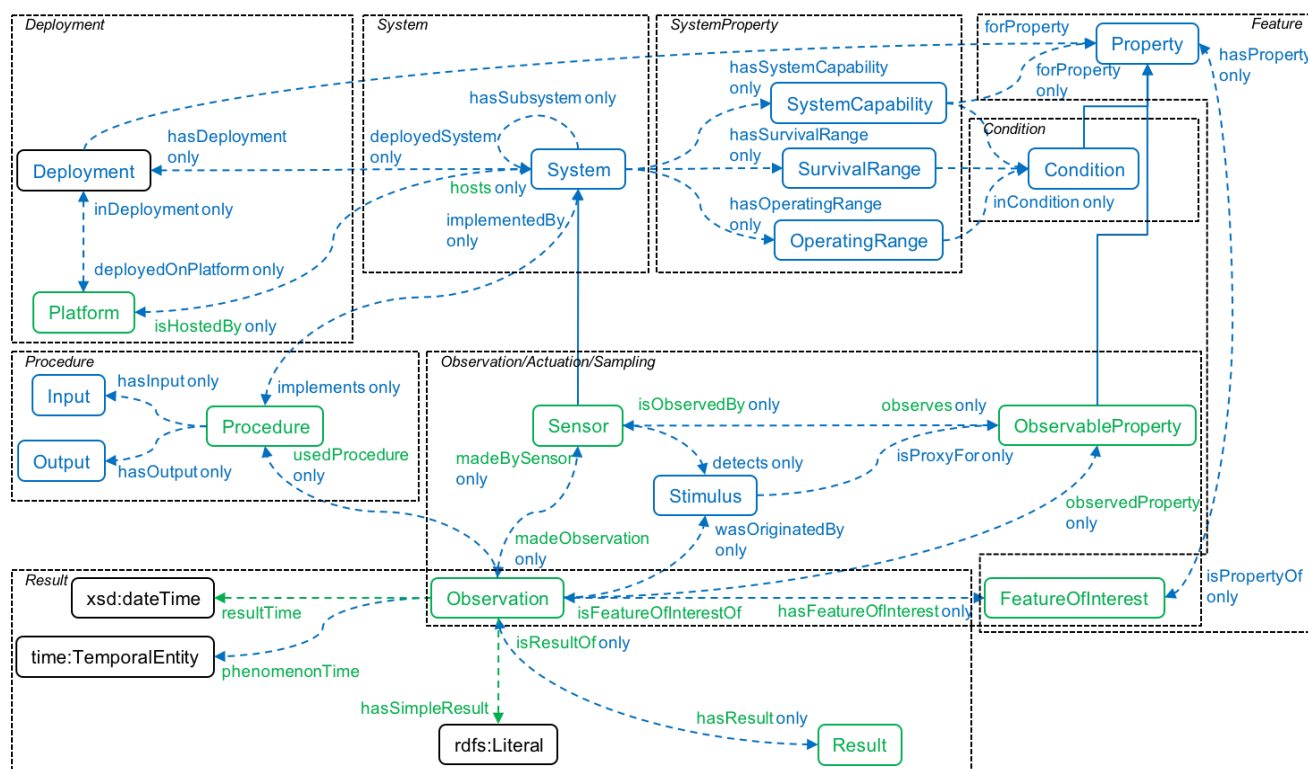
2.4.8 Οι οντολογίες SSN και SOSA

Σύμφωνα με την W3C, η οντολογία Semantic Sensor Network (SSN)⁹ είναι μια οντολογία για την περιγραφή αισθητήρων και των μετρήσεών τους (observations), των διαδικασιών που σχετίζονται με αυτές, τα μελετώμενα χαρακτηριστικά ενδιαφέροντος (features of interest), τα δείγματα (samples) που αντιπροσωπεύουν όλα τα παραπάνω, και τις ιδιότητες που έχουν μετρηθεί, όπως και τους ενεργοποιητές (actuators). Με τον όρο “ενεργοποιητές” αναφερόμαστε στα συστατικά μιας μηχανής (π.χ. συσκευής) που είναι υπεύθυνα για την κίνηση και τον έλεγχο ενός μηχανισμού ή συστήματος, π.χ. για το άνοιγμα μιας βαλβίδας. Δηλαδή, πρόκειται για οντότητες που “υποκινούν” συγκεκριμένες ενέργειες. Η οντολογία SSN ενσωματώνει στον πυρήνα της μια ελαφριά αλλά αυτοδύναμη οντολογία που ονομάζεται SOSA (Sensor, Observation, Sample, and Actuator), την οποία χρησιμοποιεί για τις βασικές κλάσεις και ιδιότητές της. Χάρη στο διαφορετικό πεδίο εφαρμογής τους αλλά και τους διαφορετικούς βαθμούς αξιοποίησης, η SSN και η SOSA είναι ικανές να υποστηρίξουν ένα μεγάλο εύρος εφαρμογών και σεναρίων χρήσης που συμπεριλαμβάνουν δορυφορικές απεικονίσεις, επιστημονική παρακολούθηση μεγάλης κλίμακας, βιομηχανικές και οικιακές υποδομές, πολιτικές επιστήμες, τεχνολογίες κατασκευής οντολογιών και τον Ιστό των Πραγμάτων. Στο ευρετήριο (namespace)¹⁰ για τους όρους της οντολογίας SSN που υπάρχει στο Διαδίκτυο είναι ορατές οι κλάσεις της οντολογίας, καθώς και οι ιδιότητες και οι συσχετίσεις τους.

Το βασικό πλεονέκτημα της οντολογίας SSN σε σχέση με την SOSA είναι ότι έχει προσθέσει εκφραστικότητα στην SOSA, λόγω των παραπάνω κλάσεων και ιδιοτήτων που περιλαμβάνει, κι έτσι μπορεί να περιγράψει ένα μεγαλύτερο πλήθος οντοτήτων και με μεγαλύτερη ακρίβεια. Γι’ αυτό το λόγο, έχει χρησιμοποιηθεί στις περισσότερες υλοποιήσεις που αφορούν το Διαδίκτυο των Πραγμάτων (IoT projects). Στην Εικόνα 2.3 αποτυπώνεται η κύρια δομή της οντολογίας SSN, στην αρχική μορφή της, η οποία περικλείει στον πυρήνα της και επεκτείνει την οντολογία SOSA. Η κλάσεις και σχέσεις που είναι ορισμένες στην οντολογία SSN φαίνονται με πράσινο και μπλε χρώμα, ενώ το υποσύνολο που είναι η οντολογία SOSA φαίνεται με πράσινο χρώμα.

⁹ <https://www.w3.org/TR/vocab-ssn/>

¹⁰ <https://www.w3.org/ns/ssn/>



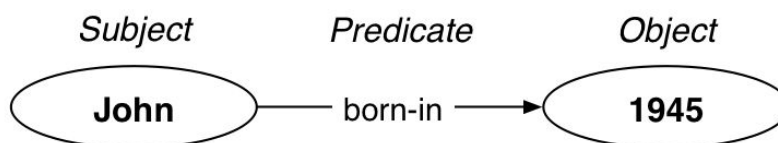
Εικόνα 2.3: Μια σφαιρική εικόνα από την οντολογία SSN (μπλε και πράσινο) και τα εννοιολογικά μοντέλα της μαζί με το υποσύνολο της SOSA (πράσινο μόνο).

2.4.9 RDF - RDFS

Resource Description Framework (εν συντομία: RDF)¹¹ ονομάζεται μια οικογένεια προδιαγραφών (specifications) που αναπτύχθηκε από την Κοινοπραξία του Παγκόσμιου Ιστού και σχεδιάστηκε αρχικά ως ένα μοντέλο μεταδεδομένων (metadata data model). Πλέον χρησιμοποιείται ως μια γενική μέθοδος για την εννοιολογική περιγραφή ή για τη μοντελοποίηση των πληροφοριών που πραγματοποιείται στους πόρους του Διαδικτύου. Για την επίτευξη των παραπάνω στόχων, αξιοποιείται μια ποικιλία συντακτικών σημειογραφιών (syntax notations) και μορφών σειριοποίησης δεδομένων. Το πρότυπο RDF χρησιμοποιείται επίσης σε εφαρμογές διαχείρισης γνώσης (knowledge management applications). Ως μοντέλο, έχει ομοιότητες με

¹¹ https://en.wikipedia.org/wiki/Resource_Description_Framework

κλασικές προσεγγίσεις εννοιολογικής μοντελοποίησης (όπως το μοντέλο ER¹² ή τα διαγράμματα κλάσεων¹³). Βασίζεται στην ιδέα της δημιουργίας δηλώσεων (statements) που αφορούν πόρους - συγκεκριμένα πόρους του Διαδικτύου - με χρήση εκφράσεων της μορφής *υποκείμενο - κατηγορημα - αντικείμενο* (*subject - predicate - object*), που είναι γνωστές ως τριπλέτες (triples). Το υποκείμενο υποδηλώνει τον πόρο, το κατηγορημα υποδηλώνει χαρακτηριστικά ή πτυχές του πόρου, και εκφράζει μια σχέση μεταξύ του υποκειμένου και του αντικειμένου.



Εικόνα 2.4: Το RDF σχήμα

Resource Description Framework Schema (εν συντομία: RDFS¹⁴) ονομάζεται ένα σύνολο κλάσεων με συγκεκριμένες ιδιότητες, οι οποίες χρησιμοποιούν το επεκτάσιμο RDF μοντέλο δεδομένων (data model) αναπαράστασης γνώσης, που παρέχει βασικά στοιχεία για την περιγραφή των οντολογιών, οι οποίες συναντώνται και με τον όρο “λεξιλόγια RDF” (RDF vocabularies) και προορίζονται για τη δόμηση πόρων RDF. Οι πόροι αυτοί μπορούν να αποθηκευτούν σε μια δομή αποθήκευσης τριπλετών (triplestore ή RDF store), από όπου μπορούν να ανακτηθούν με τη γλώσσα ερωτήσεων SPARQL. Πολλά συστατικά (components) της RDFS συμπεριλαμβάνονται στην Web Ontology Language (OWL), η οποία είναι περισσότερο εκφραστική.

2.4.10 OWL

Web Ontology Language (εν συντομία: OWL¹⁵) ονομάζεται μια οικογένεια γλωσσών αναπαράστασης γνώσης που χρησιμοποιούνται στο χώρο του

¹² https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model

¹³ https://en.wikipedia.org/wiki/Class_diagram

¹⁴ https://en.wikipedia.org/wiki/RDF_Schema

¹⁵ https://en.wikipedia.org/wiki/Web_Ontology_Language

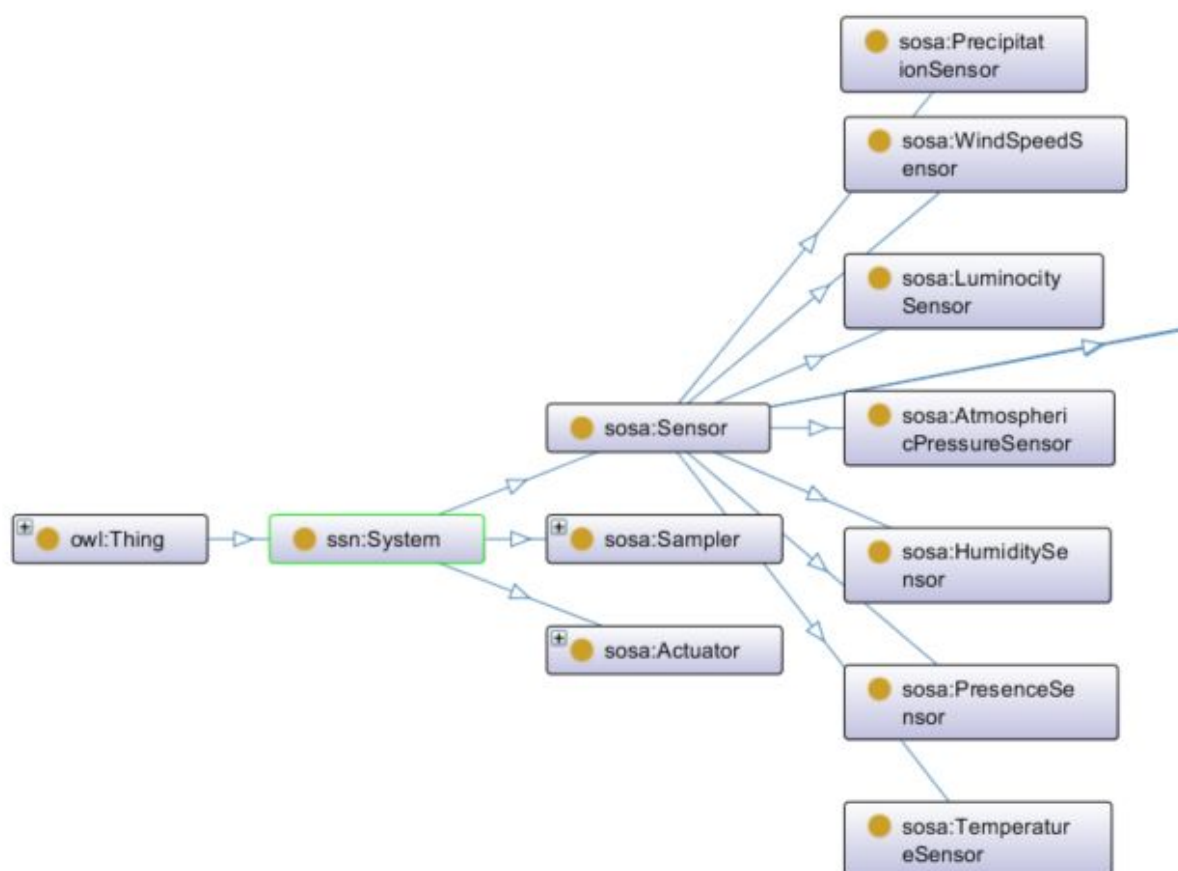
Σημασιολογικού Ιστού για τη συγγραφή οντολογιών. Οι γλώσσες αυτές χτίζονται με βάση το XML πρότυπο για αντικείμενα, που ονομάζεται RDF (Resource Description Framework) και αναπτύχθηκε από την Κοινοπραξία του Παγκόσμιου Ιστού (World Wide Web Consortium). Συγκεκριμένα, η γλώσσα OWL επιτυγχάνει την επέκταση της ικανότητας των XML, RDF και RDF Schema (RDFS) να αναπαριστούν οντολογίες του παγκόσμιου ιστού. Η OWL και το πρότυπο RDF έχουν προσελκύσει σε μεγάλο βαθμό το ενδιαφέρον του ακαδημαϊκού χώρου, του χώρου της Ιατρικής καθώς και αυτού της διαφήμισης. Η οικογένεια OWL περιέχει πολλά είδη, σειριοποιήσεις (serializations), συντακτικά και προδιαγραφές (specifications) με παρόμοια ονόματα. Η OWL και η OWL2 χρησιμοποιούνται για την αναφορά στις προδιαγραφές OWL του 2004 και του 2009, αντίστοιχα. Για την αναφορά σε συγκεκριμένη έκδοση προδιαγραφών χρησιμοποιούνται τα πλήρη ονόματα της έκδοσης, τα οποία περιλαμβάνουν και την έκδοση της εκάστοτε προδιαγραφής (π.χ. OWL2 EL). Ωστόσο, για πιο γενικές αναφορές, χρησιμοποιείται ο όρος Οικογένεια OWL.

2.4.11 SWRL

Semantic Web Rule Language (εν συντομία: SWRL) ονομάζεται μια γλώσσα που προτείνεται για το Σημασιολογικό Ιστό, η οποία χρησιμοποιείται για να εκφράσει κανόνες και λογική σε μια οντολογία. Οι κανόνες SWRL αποτελούνται από δύο κομμάτια, το προηγούμενο (body) και το επακόλουθο (head). Η ουσιαστική λειτουργία ενός κανόνα είναι ότι μόλις ικανοποιηθούν οι συνθήκες στο σώμα, τότε ικανοποιούνται και οι συνθήκες ορισμένες στο επακόλουθο κομμάτι. Μέσω της SWRL επιτυγχάνεται ο συλλογισμός που επιτρέπει ένα αυτοματοποιημένο σύστημα να λειτουργήσει.

2.4.12 Σημασιολογικός Γράφος

Ένας σημασιολογικός γράφος (ή γράφος RDF) είναι ένα δίκτυο κόμβων, το οποίο αναπαριστά σημασιολογικές σχέσεις μεταξύ εννοιών και απαρτίζεται από ένα σύνολο τριάδων RDF. Στην παρακάτω εικόνα φαίνεται ένα στιγμιότυπο (από το περιβάλλον Protege, βλ. 2.4.19) που απεικονίζει σχηματικά το σημασιολογικό γράφο μιας οντολογίας - συγκεκριμένα αυτής που αναπτύχθηκε για τις ανάγκες των εφαρμογών του συστήματος - και εστιάζει στην κλάση System της οντολογίας (και τις υποκλάσεις της).



Εικόνα 2.5: Ένα μέρος του σημασιολογικού γράφου της οντολογίας του συστήματος (δείχνει την ιεραρχία ορισμένων κλάσεων).

2.4.13 SPARQL

Η SPARQL είναι μια γλώσσα ερωτήσεων RDF (Resource Description Framework), δηλαδή μια σημασιολογική γλώσσα ερωτήσεων για βάσεις δεδομένων, η οποία χρησιμοποιείται για την ανάκτηση και το χειρισμό δεδομένων που έχουν αποθηκευτεί σε μορφή RDF. Αυτό σημαίνει ότι είναι δυνατή η επεξεργασία δεδομένων τα οποία βρίσκονται αποθηκευμένα σε βάσεις δεδομένων ως σύνολα τριπλετών με τη μορφή υποκείμενο - κατηγορία - αντικείμενο (subject - predicate - object). Συγκεκριμένα, με τη χρήση της γλώσσας SPARQL, μπορούν να προστεθούν νέα δεδομένα σε γράφους, καθώς και να ανακτηθούν, να ενημερωθούν ή να διαγραφούν υπάρχοντα δεδομένα.

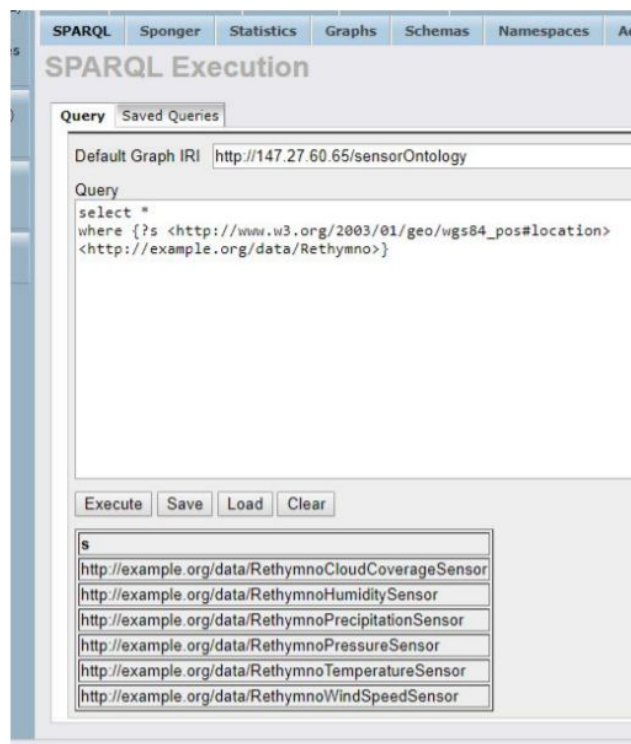
2.4.14 Virtuoso

Ο Virtuoso (Openlink) Universal Server είναι ένα ενδιάμεσο λογισμικό (middleware) που συνδυάζει τη λειτουργικότητα ενός παραδοσιακού Σχεσιακού συστήματος βάσεων δεδομένων (RDBMS) με τη λειτουργικότητα μιας Αντικειμενο-σχεσιακής βάσης δεδομένων (ORDBMS), μιας εικονικής βάσης δεδομένων, του προτύπου RDF, του προτύπου XML, ενός εξυπηρετητή διαδικτυακής εφαρμογής (web application server) και ενός εξυπηρετητή αρχείων (file server) σε ένα ενιαίο σύστημα. Αντί για να έχουμε διαφορετικούς εξυπηρετητές, απ' τους οποίους ο καθένας θα είναι αφιερωμένος στο να πραγματοποιεί μια λειτουργικότητα από τις προαναφερθείσες, ο Virtuoso είναι ένας “universal server” (καθολικός εξυπηρετητής). Αυτό σημαίνει ότι επιτρέπει μια ενιαία πολυνηματική (multithreaded) διαδικασία εξυπηρετητή που εφαρμόζει πολλαπλά πρωτόκολλα. Η ελεύθερη και ανοικτού κώδικα έκδοση του Virtuoso Universal Server είναι επίσης γνωστή ως Openlink Virtuoso.

Χάρη στο τερματικό της Virtuoso (Virtuoso SPARQL Endpoint), μπορούμε να εκτελέσουμε SPARQL ερωτήματα για οποιοδήποτε γράφο από αυτούς που φιλοξενούνται στην Virtuoso. Για παράδειγμα, μπορούμε να τρέξουμε ένα ερώτημα το οποίο θα ζητάει από το γράφο της οντολογίας όλες τις οντότητες (π.χ. αισθητήρες) που βρίσκονται στην τοποθεσία Ρέθυμνο. Αυτό επιτυγχάνεται με ένα απλό SPARQL ερώτημα και προκύπτει το αποτέλεσμα που φαίνεται στην παρακάτω εικόνα, δηλαδή μια σειρά από αισθητήρες που βρίσκονται (σύμφωνα με τα δεδομένα που έχουν καταχωρηθεί στην οντολογία) στο Ρέθυμνο. Αυτή η πληροφορία φιλοξενείται στο γράφο σε μορφή τριπλετών, όπου, σε κάθε τριπλέτα που αναφέρεται στην τοποθεσία, έχουμε ότι:

- το υποκείμενο είναι το όνομα του αισθητήρα,
- το κατηγορήμα είναι ο υπεрсύνδεσμος:
[“http://www.w3.org/2003/01/geo/wgs84_pos#location”](http://www.w3.org/2003/01/geo/wgs84_pos#location)
- και το αντικείμενο είναι ο υπεрсύνδεσμος
[“http://example.org/data/Rethymno”](http://example.org/data/Rethymno).

Έτσι, παίρνουμε αποτελέσματα που φαίνονται στην εικόνα 2.6:



Εικόνα 2.6: Αναζήτηση μέσω SPARQL στην Virtuoso

2.4.15 Java EE (Enterprise Edition)

Η Java EE είναι πλατφόρμα που βασίζεται στη γλώσσα προγραμματισμού Java και προορίζεται για προγραμματισμό εξυπηρετητών (server programming) με προδιαγραφές για enterprise features όπως οι υπηρεσίες ιστού (web services). Η Java EE περιλαμβάνει πολλές προδιαγραφές που εξυπηρετούν διαφορετικές σκοπιμότητες, όπως η δημιουργία ιστοσελίδων, το διάβασμα και το γράψιμο από μια βάση δεδομένων με ένα συναλλακτικό τρόπο (transactional way), η διαχείριση αιτημάτων HTTP, η διεπαφή προγραμματισμού εφαρμογών Java για Restful υπηρεσίες Ιστού και η επεξεργασία μορφότυπου JSON.

2.4.16 Spring Boot

Πρόκειται για ένα πλαίσιο εφαρμογών ανοικτού κώδικα για την πλατφόρμα Java το οποίο παρέχει πολλές επεκτάσεις για τη δημιουργία διαδικτυακών εφαρμογών.

Αποτελεί την λύση του Spring¹⁶ για να δημιουργήσουμε αυτόνομες εφαρμογές (βασισμένες σε Spring) οι οποίες μπορούν να εγκατασταθούν και να τρέξουν πάνω σε έναν server. Τα βασικά χαρακτηριστικά του Spring Boot είναι:

- Να δημιουργεί αυτόνομες εφαρμογές Spring
- Ο ενσωματωμένος Tomcat server
- Να διαμορφώνει αυτόματα (automatically configure) το Spring όταν είναι δυνατόν
- Να παρέχει δογματικά (opinionated) “starter” Project Object Models (POMs) για να απλοποιήσει τη Maven¹⁷ διαμόρφωσή σου

2.4.17 Apache Jena

Το Apache Jena Semantic Web Framework αποτελεί ένα πλαίσιο εφαρμογών Σημασιολογικού Ιστού βασισμένο σε γλώσσα Java. Παρέχει ένα προγραμματιστικό περιβάλλον για επεξεργασία μεταδεδομένων RDF, RDFS, OWL, SPARQL και ένα μηχανισμό αιτίας - συλλογισμού (reasoner), δηλαδή μια μηχανή εξαγωγής λογικών συμπερασμάτων βασισμένη σε κανόνες. Αξιοποιεί την έννοια του Μοντέλου, δηλαδή μιας διεπαφής η οποία ορίζει τον τρόπο αποθήκευσης γράφων RDF καθώς και τις μεθόδους που μπορούν να κληθούν πάνω στο Μοντέλο. Οι μέθοδοι αφορούν λειτουργίες όπως η προσθήκη τριπλετών (triples) δεδομένων, ή η προσθήκη πόρων και ιδιοτήτων σε πόρους, η ανάκτηση τριάδων, πόρων και ιδιοτήτων, η εκτύπωση μεταδεδομένων RDF στην έξοδο (π.χ. σε αρχείο RDF/XML), κλπ.

2.4.18 Μηχανισμοί συλλογισμού

Η σημασιολογική αιτίαση ή αλλιώς συλλογισμός (reasoning) αποτελεί μια διαδικασία αυτοματοποιημένου συλλογισμού, που χρησιμοποιείται για την εύρεση ασυνεπειών και για την εξαγωγή νέων πληροφοριών από τις πληροφορίες που αντιπροσωπεύονται σε μια οντολογία. Χαρακτηριστικοί μηχανισμοί αιτίας ή συλλογισμού (reasoners) που χρησιμοποιούνται ευρέως σήμερα είναι ο Pellet, ο FaCT++ και ο HermiT.

¹⁶ <https://spring.io/>

¹⁷ https://en.wikipedia.org/wiki/Apache_Maven

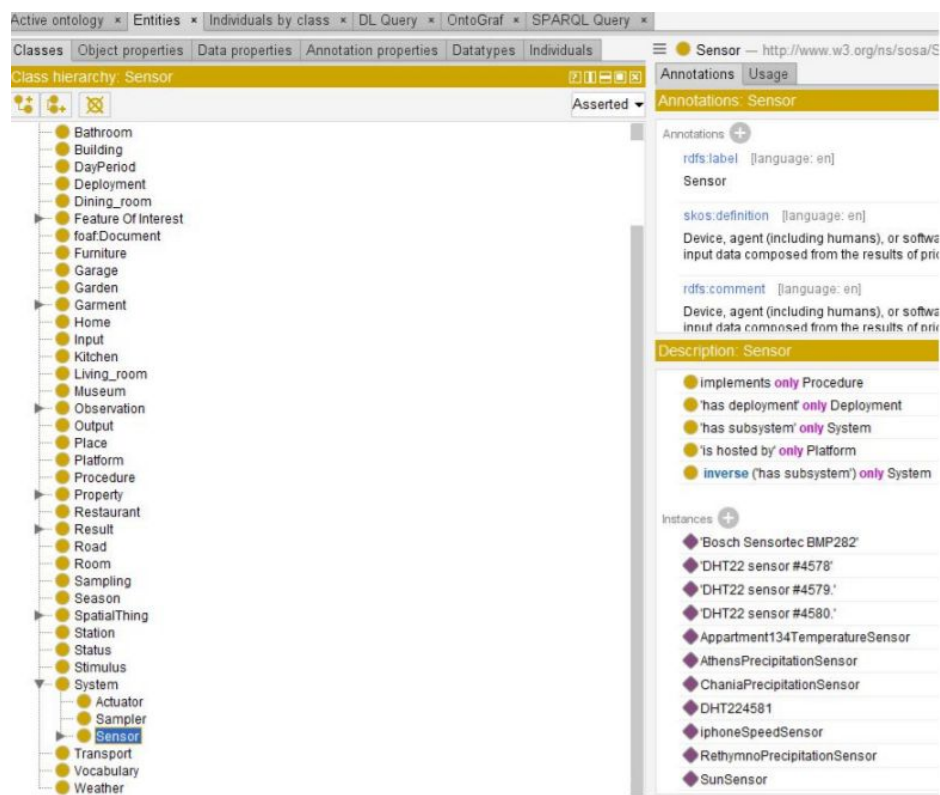
Ο **Pellet Reasoner**¹⁸ αποτελεί έναν ανοικτού κώδικα OWL 2 μηχανισμό αιτίασης (reasoner), ο οποίος είναι βασισμένος σε Java. Μπορεί να χρησιμοποιηθεί σε συνδυασμό με τις βιβλιοθήκες του Apache Jena και του OWL API, καθώς και να ενσωματωθεί σε άλλες εφαρμογές. Όπως κάθε μηχανισμός αιτίασης, μπορεί να αξιοποιηθεί για την εύρεση ασυνεπειών αλλά και για την εξαγωγή νέων πληροφοριών από τις πληροφορίες που αντιπροσωπεύονται σε κάποια οντολογία.

2.4.19 Protege

Το Protege¹⁹ είναι μια ελεύθερη, ανοικτού κώδικα πλατφόρμα δημιουργίας και επεξεργασίας οντολογιών και ένα σύστημα διαχείρισης γνώσεων. Διαθέτει μια γραφική διεπαφή χρήστη για ορισμό οντολογιών. Σε αυτήν τη διεπαφή, ο χρήστης μπορεί να εξερευνήσει μια ήδη υπάρχουσα οντολογία (να ανακαλύψει όλες τις κλάσεις και τις ιδιότητές της) και να την επεξεργαστεί ή να δημιουργήσει μια νέα οντολογία. Επιπλέον, το Protege έχει τη δυνατότητα ενσωμάτωσης επεκτάσεων όπως είναι οι μηχανισμοί αιτίασης/συλλογισμού (π.χ. Pellet reasoner) που έχουν σκοπό να επικυρώσουν ότι τα μοντέλα οντολογιών είναι έγκυρα καθώς και να εξαγάγουν νέες πληροφορίες και συμπεράσματα που βασίζονται στην ανάλυση των οντολογιών. Μετά τη σύνταξη μιας οντολογίας, υπάρχει η δυνατότητα εξαγωγής της (export) σε διαφορετικές μορφές αναπαράστασης όπως RDF, RDFS, OWL και XML. Στην εικόνα 2.7 μπορούμε να δούμε ένα στιγμιότυπο από το περιβάλλον Protege, όπου φαίνεται η οντολογία που έχει δημιουργηθεί για τις ανάγκες των εφαρμογών του Mashup Service

¹⁸ <https://www.w3.org/2001/sw/wiki/Pellet>

¹⁹ <https://protege.stanford.edu/>



Εικόνα 2.7: Στιγμιότυπο από την πλατφόρμα Protege, στο οποίο διακρίνονται οι κλάσεις της οντολογίας (αυτής που αναπτύχθηκε για την παρούσα εργασία).

2.4.20 PHP

Η PHP²⁰ (Hypertext Processor) αποτελεί μια server-side γλώσσα προγραμματισμού που χρησιμοποιείται για τη δημιουργία σελίδων web με δυναμικό περιεχόμενο. Στο κομμάτι της υλοποίησης του Back-End της παρούσας εργασίας έγινε χρήση εξυπηρετητών Apache (διακομιστής του Παγκόσμιου Ιστού που είναι συμβατός με την PHP) για τη διαχείριση REST αιτημάτων μεταξύ υπηρεσιών. Συνεπώς, χρησιμοποιήθηκε η PHP, καθώς και ορισμένες επεκτάσεις της, όπως η cURL²¹. Η cURL είναι μια βιβλιοθήκη της PHP που επιτρέπει την εκτέλεση HTTP αιτημάτων και συνεπώς τη μεταφορά δεδομένων μεταξύ των υπηρεσιών με τη χρήση διαφορετικών πρωτοκόλλων (DICT, FTP, FTPS, HTTP, HTTPS κ.ά.). Σε αυτήν την εργασία η

²⁰ <https://www.php.net/>

²¹ <https://curl.haxx.se/>

βιβλιοθήκη cURL αξιοποιήθηκε για την κλήση μεθόδων του πρωτοκόλλου HTTP κατευθείαν μέσα από τον κώδικα της γλώσσας PHP.

2.4.21 Google Charts

Πρόκειται για μία βιβλιοθήκη από την Google²² η οποία παρέχει έτοιμες μεθόδους για την απεικόνιση δεδομένων σε ένα ιστότοπο. Από απλά γραφήματα γραμμών έως πολύπλοκους ιεραρχικούς χάρτες δέντρων, η συλλογή γραφημάτων παρέχει έναν μεγάλο αριθμό έτοιμων τύπων γραφημάτων. Ο τρόπος χρήσης των Google Charts είναι με το JavaScript, φορτώνοντας απλά την βιβλιοθήκη στην ιστοσελίδα. Τα διαγράμματα αποδίδονται χρησιμοποιώντας την τεχνολογία HTML5 / SVG για την παροχή συμβατότητας μεταξύ των προγραμμάτων περιήγησης και της φορητότητας πλατφόρμας σε iPhones, iPads και Android.

2.4.22 Υπηρεσία Mashup - Node RED

Πρόκειται για μία υπηρεσία Mashup η οποία παρέχει την δυνατότητα δημιουργίας εκτελέσιμων εφαρμογών ως σύνθεση κόμβων που ανταλλάσσουν δεδομένα σε προκαθορισμένες συνδέσεις. Ένας κόμβος για το Node-RED²³ είναι μια ανεξάρτητη και αυτοτελής μονάδα με σαφώς καθορισμένη λειτουργικότητα. Διαθέτει ένα runtime περιβάλλον²⁴ σε Node.js μέσω του οποίου εκτελούνται οι εφαρμογές. Η υπηρεσία προσφέρει μια RESTful διεπαφή με μεθόδους για την δημιουργία/επεξεργασία και διαγραφή εκτελέσιμων εφαρμογών.

²² <https://developers.google.com/chart/>

²³ <https://nodered.org/>

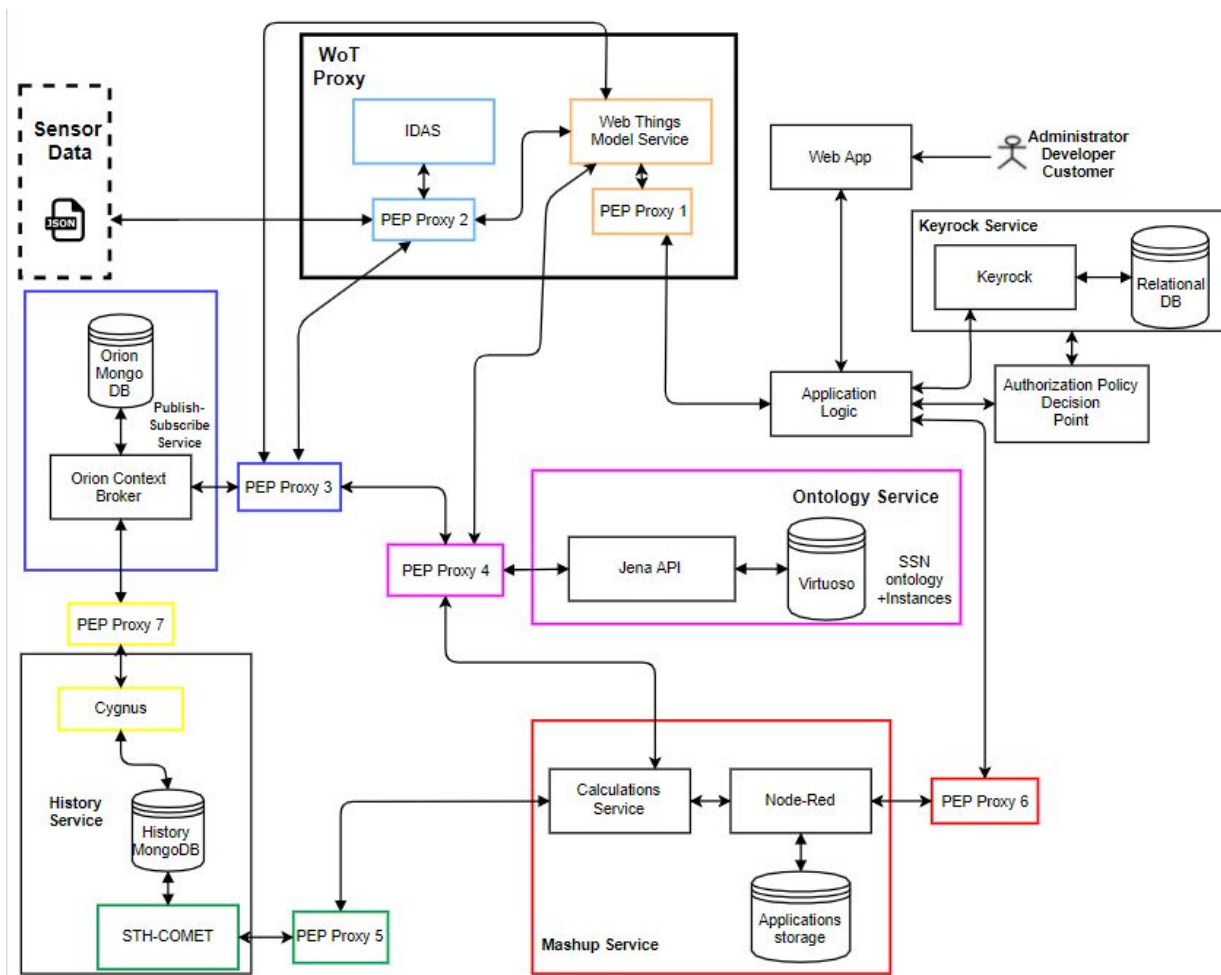
²⁴ <https://techterms.com/definition/rte>

Κεφάλαιο 3

Αρχιτεκτονική Συστήματος

3.1 Διάγραμμα Αρχιτεκτονικής

Ακολουθεί το διάγραμμα αρχιτεκτονικής του συστήματος στο οποίο στηρίχθηκε η υλοποίησή του. Έχουμε ακολουθήσει μια υπηρεσιοκεντρική αρχιτεκτονική, η οποία υλοποιεί μία πλατφόρμα Ιστού των Πραγμάτων. Στο σχήμα φαίνονται οι υπηρεσίες που εμπλουτίζουν με σημασιολογία τις οντότητες του συστήματος και καθιστούν την παρούσα αρχιτεκτονική μέρος του Σημασιολογικού Ιστού των Πραγμάτων.



Εικόνα 3.1: Διάγραμμα αρχιτεκτονικής συστήματος

Στη συνέχεια, γίνεται μια εκτενής περιγραφή των επιμέρους υπηρεσιών του συστήματος, καθώς και της αλληλεπίδρασης που υπάρχει μεταξύ τους προκειμένου να πραγματοποιηθούν οι ζητούμενες λειτουργίες.

3.2 Διαχείριση Δεδομένων

Το σύστημά μας διαχειρίζεται πολλά διαφορετική είδη πληροφορίας και για κάθε είδος υπάρχει μια αντίστοιχη βάση δεδομένων στην οποία αποθηκεύονται τα αντίστοιχα δεδομένα.

Τα αποτελέσματα των μετρήσεων των αισθητήρων αποθηκεύονται σε μια ιστορική βάση MongoDB (History MongoDB). Αυτή η βάση είναι μια μη-σχεσιακή βάση δεδομένων κι έτσι μπορεί να εξυπηρετήσει το μεγάλο όγκο ιστορικών δεδομένων μετρήσεων (χρονικής σειράς) των αισθητήρων/συσκευών που έχουν συνδεθεί στο σύστημα. Ακόμα, η MongoDB είναι μια απλή στη χρήση και γρήγορη βάση δεδομένων, ενώ είναι και συμβατή με δεδομένα της μορφής JSON (NGSI2). Επομένως, ικανοποιεί τις ανάγκες της ιστορικής βάσης δεδομένων. Η ανάκτηση των ιστορικών μετρήσεων γίνεται μέσω μιας άλλης υπηρεσίας. Η υπηρεσία ανάκτησης πληροφοριών (COMET) με τις REST μεθόδους που αυτή παρέχει, η πληροφορία, που έχει αποθηκευτεί στην ιστορική βάση και αφορά το ιστορικό μετρήσεων, ανακτάται πολύ εύκολα. Ο ρόλος της υπηρεσίας Comet είναι η ανάκτηση των στατιστικών τιμών δεδομένων (μέσος όρος, μέγιστη, ελάχιστη τιμή πάνω σε ορισμένο διάστημα όπως ημέρα, ώρα, μήνας κλπ) τα οποία τα παίρνει από την ιστορική αυτή βάση MongoDB.

Η σχεσιακή βάση δεδομένων του συστήματος (Relational DB) χρησιμοποιείται για την αποθήκευση πληροφοριών σχετικών με τους χρήστες. Αυτή η βάση χρησιμοποιείται από την υπηρεσία Ταυτοποίησης και Εξουσιοδότησης Χρηστών (βλ. 3.7) και αποθηκεύει την πληροφορία που χρειάζεται ένας χρήστης για να συνδεθεί στο σύστημα όπως “όνομα χρήστη” (username) και κωδικό πρόσβασης (password). Λόγω της φύσης της πληροφορίας που αφορά το προφίλ ενός χρήστη, απαιτείται η χρήση σχεσιακής βάσης δεδομένων για την αποθήκευση αυτής της πληροφορίας.

Η βάση δεδομένων “Applications Storage” χρησιμοποιείται από την υπηρεσία Σύνθεσης Εφαρμογών για να αποθηκεύονται οι εφαρμογές που κατασκευάζουν οι προγραμματιστές. Πρόκειται για μη-σχεσιακή βάση δεδομένων MongoDB, διότι οι εφαρμογές αποθηκεύονται σε μορφή JSON.

Η βάση δεδομένων Orion MongoDB χρησιμοποιείται από την Υπηρεσία Διαχείρισης Συμβάντων και Συνδρομών (Orion Context Broker), με σκοπό αυτή να αποθηκεύει τις οντότητες πληροφορίας που πρέπει να διαχειριστεί. Πάλι πρόκειται για μη-σχεσιακή βάση δεδομένων MongoDB, αφού οι οντότητες πληροφορίας που διαχειρίζεται η υπηρεσία έχουν μορφή JSON.

Τέλος, στο Σύστημα Διαχείρισης Βάσεων Δεδομένων (ΣΔΒΔ) της Virtuoso περιέχεται ο γράφος RDF, στον οποίο έχει αποθηκευτεί η οντολογία του συστήματος.

Το Virtuoso αποτελεί ένα ένα σύστημα αποθήκευσης τριπλετών (triple store) της οντολογίας. Η οντολογία που δημιουργήθηκε και χρησιμοποιείται από το σύστημα περιέχει (σε μορφή τριπλετών) τόσο το αρχικό μοντέλο οντολογίας όσο και τα δεδομένα αλλά και τα μεταδεδομένα που αφορούν τους αισθητήρες / συσκευές, όπως είναι οι μετρήσεις τους, τα σπίτια και οι πόλεις όπου αυτοί βρίσκονται, κλπ.

Υπηρεσία Αποθήκευσης των Ιστορικών Δεδομένων (History Service)

Η Υπηρεσία Αποθήκευσης των Ιστορικών Δεδομένων αναλαμβάνει να δέχεται ροές δεδομένων που προέρχονται από μετρήσεις αισθητήρων. Όπως και οι υπόλοιπες υπηρεσίες, περιλαμβάνει μικρότερες υπηρεσίες, καθώς και μια βάση δεδομένων MongoDB (βλ. διάγραμμα αρχιτεκτονικής), που συνεργάζονται μεταξύ τους για να εξυπηρετήσουν τις ανάγκες της υπηρεσίας που διαχειρίζεται τα ιστορικά δεδομένα. Συγκεκριμένα, η παρούσα υπηρεσία αποτελείται από την υπηρεσία Cygnus (βλ. Κεφάλαιο 2), που αποτελεί σύνδεσμο μεταξύ του Orion Context Broker (στέλνει δεδομένα μετρήσεων σε μορφή NGSI2) και της βάσης δεδομένων MongoDB. Το Cygnus δέχεται τις ροές δεδομένων NGSI2 και τις αποθηκεύει στην History MongoDB, δηλαδή στην ιστορική βάση δεδομένων του συστήματος (βλ. “Βάσεις δεδομένων στο σύστημα”). Επιπλέον, έχει τη δυνατότητα να αποθηκεύει τα ιστορικά δεδομένα τόσο ακατέργαστα (Raw) όσο και συγκεντρωτικά (Aggregated), στην ιστορική βάση δεδομένων. Με αυτόν τον τρόπο, διατηρείται το ιστορικό των μετρήσεων των αισθητήρων. Στη συνέχεια, η υπηρεσία COMET (βλ. Κεφάλαιο 2) χρησιμοποιείται για την ανάκτηση των δεδομένων που έχουν αποθηκευτεί στην ιστορική βάση MongoDB. Η υπηρεσία COMET έχει το πλεονέκτημα ότι παρέχει REST μεθόδους για εξειδικευμένα ερωτήματα (queries) που αφορούν στατιστικά (π.χ. για ανάκτηση του αθροίσματος, του μέγιστου ή του ελάχιστου από ένα πλήθος τιμών), καθώς και ερωτήματα για συγκεκριμένα χρονικά διαστήματα (π.χ. για ανάκτηση δεδομένων κατά το διάστημα ενός ή παραπάνω λεπτών, ωρών, ημερών ή ακόμα και μηνών). Τα δεδομένα που λαμβάνονται μέσω της υπηρεσίας COMET, και συγκεκριμένα τα στατιστικά, αξιοποιούνται στη συνέχεια από άλλη υπηρεσία (Mashup Service), όπως φαίνεται και στο διάγραμμα αρχιτεκτονικής του συστήματος. Το Mashup Service μπορεί να εκμεταλλευτεί κατάλληλα το γεγονός ότι λαμβάνει στατιστικά για επιλεγμένες χρονικές περιόδους, ούτως ώστε να εξυπηρετήσει τις ανάγκες των εφαρμογών.

3.3 Υπηρεσία Διακομιστή μεσολάβησης του Ιστού των Πραγμάτων (Web of Things Proxy)

Η υπηρεσία Διακομιστή μεσολάβησης του Ιστού των Πραγμάτων απαρτίζεται από δύο υπηρεσίες: την υπηρεσία Υλοποίησης του Μοντέλου του Ιστού των Πραγμάτων (Web Things Model Service) και την Υπηρεσία Διαχείρισης Συσκευών Backend IDAS. Ο Διακομιστής μεσολάβησης επικοινωνεί με τις εξωτερικές οντότητες του Ιστού των Πραγμάτων (τα ερεθίσματα από τον “έξω κόσμο”), δηλαδή με τους χρήστες (χάρη στην πρώτη υπηρεσία) αλλά και με τις συσκευές - αισθητήρες (μέσω της δεύτερης). Με αυτόν τον τρόπο, μεσολαβεί ανάμεσα στο Υπολογιστικό Νέφος και τον έξω κόσμο, αποτελώντας είσοδο δεδομένων από τον έξω κόσμο, καθώς και έξοδο προς αυτόν (π.χ. όταν δίνεται εντολή προς μια συσκευή ή κάποιος χρήστης ανακτά μια πληροφορία). Συγκεκριμένα, η υπηρεσία Υλοποίησης του Μοντέλου του Ιστού των Πραγμάτων είναι υπεύθυνη για την πραγματοποίηση όλων των λειτουργιών που προβλέπει το Web Thing Model. Γι’ αυτό το λόγο, επικοινωνεί άμεσα με την Διαδικτυακή Εφαρμογή (Web Application) του συστήματος, δηλαδή μια γραφική διεπαφή για τους χρήστες, με σκοπό να εξυπηρετήσει όλες τις λειτουργίες που επιλέγει ο χρήστης μέσω αυτής. Παράλληλα, αλληλεπιδρά τόσο με την Υπηρεσία Διαχείρισης Συμβάντων και Συνδρομών του συστήματος όσο και με την Υπηρεσία Οντολογίας, με τις οποίες έχει αμφίδρομη σχέση, καθώς και λαμβάνει και στέλνει δεδομένα προς αυτές. Για παράδειγμα, μπορεί να ανακτά πληροφορίες (σχετικές με τις συσκευές) από την οντολογία ή από τη βάση δεδομένων της Υπηρεσίας Διαχείρισης Συμβάντων και Συνδρομών και να τις εκτυπώνει στο χρήστη μέσω της Διαδικτυακής Εφαρμογής. Και αντίστροφα, μπορεί να προσθέτει / ενημερώνει / διαγράφει πληροφορίες στις προαναφερθείσες αποθηκευτικές δομές, σύμφωνα με τις επιλογές του χρήστη στη Διαδικτυακή Εφαρμογή. Τέλος, η υπηρεσία Υλοποίησης του Μοντέλου του Ιστού των Πραγμάτων επικοινωνεί και με την Υπηρεσία Διαχείρισης Συσκευών Backend IDAS, είτε στην περίπτωση που ο χρήστης επιθυμεί να εγγράψει μια συσκευή στη λίστα των συσκευών της υπηρεσίας IDAS (ούτως ώστε το σύστημα να λαμβάνει στη συνέχεια δεδομένα από αυτήν τη συσκευή) είτε στην περίπτωση που απλά θέλει να ελέγξει αν μια συσκευή βρίσκεται στην παραπάνω λίστα συσκευών (επομένως ζητάει πληροφορία από την Υπηρεσία Διαχείρισης Συσκευών).

Όπως προαναφέρθηκε, η Υπηρεσία Διαχείρισης Συσκευών Backend IDAS του οικοσυστήματος FIWARE, αναλαμβάνει να εξυπηρετήσει την επικοινωνία των συσκευών με το Υπολογιστικό Νέφος, εφόσον αυτές έχουν συνδεθεί στην υπηρεσία και μεταδίδουν πληροφορίες. Η εν λόγω υπηρεσία έχει έναν πολύ ειδικό ρόλο: λαμβάνει τα

δεδομένα που στέλνει κάθε συσκευή, ανεξάρτητα από το πρωτόκολλο κάθε συσκευής, και μεταβιβάζει την πληροφορία στην Υπηρεσία Διαχείρισης Συμβάντων και Συνδρομών αποκλειστικά σε μορφή JSON-LD. Επομένως, το έργο της υπηρεσίας IDAS είναι πολύ σημαντικό, διότι πετυχαίνει προσαρμογή πρωτοκόλλου (protocol adaptation) και καταφέρνει να καταστήσει την επικοινωνία ανεξάρτητη από το πρωτόκολλο μετάδοσης της συσκευής. Με αυτόν τον τρόπο, η πληροφορία από συσκευές διαφορετικού είδους και πρωτοκόλλων αποθηκεύεται στο σύστημα με την ίδια μορφή και η διαχείρισή της γίνεται ακριβώς με τον ίδιο τρόπο. Σε αυτήν την πληροφορία, συμπεριλαμβάνονται μεταδεδομένα, όπως η τοποθεσία της συσκευής, η ιδιότητα που υφίσταται μέτρηση, το πρωτόκολλο της συσκευής, κλπ. Συμπεραίνουμε, δηλαδή, ότι αυτή η υπηρεσία είναι το σημείο εισόδου της πληροφορίας των συσκευών στο Υπολογιστικό Νέφος

3.4 Υπηρεσία Διαχείρισης Συμβάντων και Συνδρομών (Publish - Subscribe Service)

Το σύστημα που υλοποιήσαμε σχεδιάστηκε έτσι ώστε να επιτρέπει αλληλεπίδραση διαφόρων οντοτήτων. Αυτές οι οντότητες μπορεί να είναι να συσκευές, όπως αισθητήρες ή ενεργοποιητές αυτοματισμών, τις μετρήσεις που κάνουν οι αισθητήρες αυτοί (Παρατηρήσεις), τα χαρακτηριστικά που έχει η κάθε συσκευή (μεταδεδομένα), φυσικές οντότητες όπως πόλη ή σπίτι τις οποίες αφορούν οι μετρήσεις, εφαρμογές, συνδρομές σε εφαρμογές ή συσκευές. Όλες οι οντότητες του συστήματος περιγράφονται από την οντολογία που έχει σχεδιαστεί (θα αναλυθεί σε άλλη ενότητα) και εφόσον μία οντότητα ανήκει στο σύστημα και περιγράφεται από την οντολογία τότε η υπηρεσία Διαχείρισης Συμβάντων και Συνδρομών αναλαμβάνει την αποθήκευση αυτής της οντότητας στο σύστημα.

Η Υπηρεσία Διαχείρισης Συμβάντων και Συνδρομών δρα ως μεσολαβητής για τις παρακάτω οντότητες:

- “Αισθητήρων” (Sensors)
- “Μετρήσεων” (Observations)
- “Σπιτιών” (Homes)
- “Εφαρμογών” (Applications)
- “Εκτελέσεων ενέργειας” (Actuations)
- “Μετρούμενων ιδιοτήτων” (Observable properties)
- “Χωρικών οντοτήτων” (Spatial things)

Οι οντότητες που ανήκουν στις παραπάνω κατηγορίες αποθηκεύονται σε μορφή JSON-LD (NGSI-LD) στη βάση δεδομένων Orion MongoDB (βλ. “Βάσεις Δεδομένων του Συστήματος”), που είναι συμβεβλημένη με την υπηρεσία Orion Context Broker του FIWARE. Αυτό έπρεπε να γίνει έτσι ώστε η πληροφορία που παρέχει η οντολογία SSN για τις οντότητες του συστήματος να απεικονίζεται και στην υπηρεσία Publish-Subscribe, έτσι ώστε οι οντότητες του συστήματος που δημοσιεύονται από την υπηρεσία αυτή να ανήκουν ταυτόχρονα και να περιγράφονται από την οντολογία SSN.

Ένα πρόβλημα που κληθήκαμε να αντιμετωπίσουμε είναι το γεγονός ότι το μοντέλο NGSI2 αποτελεί ένα JSON schema και όχι ένα JSON-LD schema, δηλαδή μορφοποίηση Διασυνδεδεμένων Δεδομένων που μας ήταν απαραίτητη, ώστε να εκφραστούν σωστά μέσω της οντολογίας οι οντότητες του συστήματος και να δημιουργηθεί μια σύνδεση μεταξύ τους, διαμορφώνοντας έτσι έναν Ιστό από Διασυνδεδεμένα Δεδομένα. Αυτό όμως δεν υποστηρίζεται από την παρούσα υλοποίηση της υπηρεσίας Δημοσίευσης και Συνδρομών. Έτσι κληθήκαμε να κάνουμε κάποιες αλλαγές στον τρόπο αναπαράστασης των οντοτήτων μας στην υπηρεσία δημοσίευσης και συνδρομών. Ένα τυπικό μοντέλο NGSI2 παρουσιάζεται στην εικόνα 3.2. Μπορεί εύκολα να διαπιστωθεί ότι αυτή η αναπαράσταση οντότητας δεν αποτελεί ένα JSON-LD schema καθώς:

1. Το μοναδικό αναγνωριστικό “id” δεν αποτελεί υπερσύνδεσμο (hyperlink) σε ένα URL το οποίο δείχνει την περιγραφή της οντότητας.
2. Τα attributes της οντότητας (“refPointOfInterest”) δεν περιέχουν υπερσυνδέσμους σε άλλες οντότητες με τις οποίες σχετίζεται η παρακάτω οντότητα.

```
{
  "id": "AirQualityObserved:RZ:Obsv4567",
  "type": "AirQualityObserved",
  "dateObserved": {
    "type": "DateTime",
    "value": "2018-08-07T12:00:00"
  },
  "N02": {
    "type": "Number",
    "value": 22,
    "metadata": {
      "unitCode": {
        "type": "Text",
        "value": "GP"
      }
    }
  },
  "refPointOfInterest": {
    "type": "Reference",
    "value": "PointOfInterest:RZ:MainSquare"
  }
}
```

Εικόνα 3.2: Αναπαράσταση NGSI2(JSON) οντότητας.

Αυτή η αναπαράσταση έπρεπε να μετατραπεί σε NGSI-LD, δηλαδή ορθή JSON-LD αναπαράσταση. Για την μετατροπή ακολουθήσαμε τις συμβουλές που παρέχει το Fiware (Fiware Data Models) για την αναπαράσταση του NGSI-LD μοντέλου του²⁵. Αυτό έγινε με την προσθήκη του πεδίου “**context**”, το οποίο ουσιαστικά παρέχει τον σύνδεσμο στην οντολογία ο οποίος εκφράζει την κάθε οντότητα. Επίσης, έγινε η αλλαγή κάθε attribute (όπως τα “dateObserved”, “NO2” και “refPointOfInterest” που έχουμε εδώ) να ανήκει επίσης στην οντολογία και αποτελεί ουσιαστικά έναν JSON-LD κόμβο τύπου **Relationship**, αν αναφέρεται σε ένα *Object Property* (“refPointOfInterest”), είτε ένα κόμβο τύπου **Property** αν αναφέρεται σε ένα *Datatype Property* (“dateObserved”, “NO2”), τα οποία εκφράζονται μέσω της προσθήκης επίσης ενός “**context**” για το κάθε ένα. Αυτή η αλλαγή φαίνεται στην εικόνα 3.3 και σε όλες τις οντότητες που παρουσιάζονται παρακάτω. Στην εικόνα 3.3 φαίνεται η προσθήκη των “**Property**” και “**Relationship**” όπου χρειάζεται καθώς και η προσθήκη του πεδίου “**context**”, το οποίο παρέχει τους υπερσυνδέσμους που είναι απαραίτητοι σε ένα JSON-LD μοντέλο, οι οποίοι δείχνουν στην οντολογία SSN η οποία περιγράφει την εκάστοτε οντότητα, καθώς και τις οντότητες με τις οποίες σχετίζεται. Τέλος, στην περίπτωση που έχουμε Object Property (δηλ. “Relationship”), με το attribute “**object**” δηλώνεται ένα URL το οποίο δείχνει στην οντότητα με την οποία σχετίζεται η εκάστοτε οντότητα που το περιέχει, ενώ όταν έχουμε Datatype Property (δηλ. “Property”) με το attribute “value” υποδηλώνεται η τιμή που έχει.

²⁵ https://fiware-datamodels.readthedocs.io/en/latest/ngsi-ld_howto/index.html

```

{
  "id": "urn:ngsi-ld:AirQualityObserved:RZ:Obsv4567",
  "type": "AirQualityObserved",
  "dateObserved": {
    "type": "Property",
    "value": {
      "@type": "DateTime",
      "@value": "2018-08-07T12:00:00Z"
    }
  },
  "NO2": {
    "type": "Property",
    "value": 22,
    "unitCode": "GP",
    "accuracy": {
      "type": "Property",
      "value": 0.95
    }
  },
  "refPointOfInterest": {
    "type": "Relationship",
    "object": "urn:ngsi-ld:PointOfInterest:RZ:MainSquare"
  },
  "@context": [
    "https://schema.lab.fiware.org/ld/context",
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
  ]
}

```

Εικόνα 3.3: Αναπαράσταση NGSI-LD(JSON-LD) οντότητας.

Η παραπάνω σχεδιαστική λογική ακολουθήθηκε για όλες τις οντότητες του συστήματος.

Η αναπαράσταση JSON-LD ενός στιγμιότυπου της οντότητας “**Αισθητήρα**” (που είναι στην ουσία ένα μοντέλο συσκευής, δηλαδή ένα Web Thing Model) περιέχει δεδομένα και μεταδεδομένα για τα χαρακτηριστικά μιας οποιασδήποτε συσκευής, όπως μοναδικό αναγνωριστικό (έχει το όνομα του αντίστοιχου Πράγματος), context (υπερσύνδεσμο σε τοποθεσία που προσφέρει πληροφορία για το Πράγμα), τοποθεσία όπου βρίσκεται τοποθετημένη η συσκευή (π.χ. Σπίτι, πόλη ή γεωγραφικό σημείο), χαρακτηριστικά μέτρησης (π.χ. ότι μετράει θερμοκρασία), κατασκευάστρια εταιρεία της συσκευής, κλπ.

Η αναπαράσταση JSON-LD ενός στιγμιότυπου της οντότητας “**Μέτρηση**” (Observation) περιέχει πληροφορία για τα χαρακτηριστικά μιας συγκεκριμένης μέτρησης ενός αισθητήρα, όπως την τιμή της μέτρησης, τη φύση της μέτρησης, κλπ.

Η αναπαράσταση JSON-LD ενός στιγμιότυπου της οντότητας “**Σπίτι**” (Home) περιέχει πληροφορία για τα χαρακτηριστικά ενός συγκεκριμένου σπιτιού, όπως την τοποθεσία του, τι αισθητήρες έχει εγκατεστημένους, τι μετράνε κλπ.

Η αναπαράσταση JSON-LD ενός στιγμιότυπου της οντότητας “**Εφαρμογή**” (Application) αναφέρεται σε ένα συγκεκριμένο προγραμματιστή εφαρμογών και συνιστά μια περιγραφή της εφαρμογής που αυτός έχει δημιουργήσει. Περιλαμβάνει, δηλαδή,

χαρακτηριστικά όπως μοναδικό αναγνωριστικό στιγμιότυπου, όνομα εφαρμογής και πεδίο ενδιαφέροντος εφαρμογής.

Η αναπαράσταση JSON-LD ενός στιγμιότυπου της οντότητας **“Εκτέλεση ενέργειας”** (Actuation) περιγράφει τους αυτοματισμούς που υπάρχουν σε ένα σπίτι. Περιλαμβάνει, δηλαδή, χαρακτηριστικά όπως μοναδικό αναγνωριστικό στιγμιότυπου, μια χαρακτηριστική τιμή που φανερώνει την ενέργεια (π.χ. 0 για “off” ή 1 για “on”), ένα σημασιολογικό πλαίσιο (context) για τη συγκεκριμένη εκτέλεση ενέργειας, καθώς και την ιδιότητα μιας οντότητας στην οποία έχει επίδραση η ενέργεια (π.χ. η εκτέλεση ενέργειας με αναγνωριστικό “Room245SwitchOnLights” έχει επίδραση στη φωτεινότητα της οντότητας Room245). Επιπλέον, μπορεί να περιλαμβάνει την πληροφορία για το ποια συσκευή εκτέλεσε την ενέργεια.

Η αναπαράσταση JSON-LD ενός στιγμιότυπου της οντότητας **“Χωρική οντότητα”** (SpatialThing) περιγράφει χαρακτηριστικά μιας συγκεκριμένης τοποθεσίας (πόλης στην περίπτωση μας), όπως είναι οι καιρικές και ατμοσφαιρικές συνθήκες της.

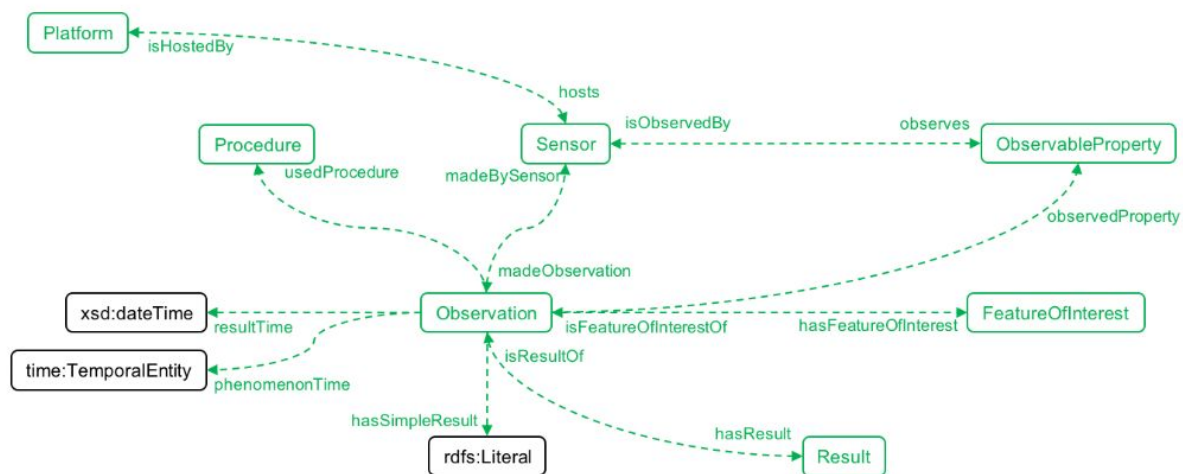
Χάρη στη RESTful διεπαφή που παρέχει η Υπηρεσία Διαχείρισης Συμβάντων και Συνδρομών, η πληροφορία που σχετίζεται με τις παραπάνω οντότητες είναι δυνατό να αποθηκεύεται - παραμετροποιείται, να ανακτάται, να ενημερώνεται αλλά και να διαγράφεται. Η λειτουργία συνδρομής (subscription) προς οντότητες που διαχειρίζεται η υπηρεσία είναι μια αξιοσημείωτη μέθοδος της RESTful διεπαφής. Συγκεκριμένα, στην περίπτωση που υπάρξει κάποια αλλαγή στις τιμές των χαρακτηριστικών της οντότητας, πυροδοτείται μια ενημέρωση με περιεχόμενο αυτήν την αλλαγή, από την εν λόγω υπηρεσία στον κάτοχο της συνδρομής. Στην παρούσα εργασία, έγινε χρήση της λειτουργίας αυτής με σκοπό τη διατήρηση του ιστορικού χρονικής σειράς για τις αλλαγές στις μετρήσεις των συσκευών (καθώς η πληροφορία για κάθε αλλαγή τιμής φτάνει στην ιστορική βάση δεδομένων), ενώ αξιοποιήθηκε η ίδια λειτουργία και για την ενημέρωση της οντολογίας του συστήματος σχετικά με τις αλλαγές στις τιμές των συσκευών. Δηλαδή, κάθε φορά που μια μέτρηση (type: Observation) αλλάζει τιμή, ένα αίτημα HTTP με σώμα την πληροφορία αυτής της αλλαγής φτάνει στην Virtuoso κι ενημερώνει τις τιμές των συσκευών που έχουν αποθηκευτεί στο γράφο της οντολογίας.

3.5 Υπηρεσία Οντολογίας (Ontology Service)

Η Υπηρεσία Οντολογίας είναι υπεύθυνη για την αλληλεπίδραση της οντολογίας (ή των οντολογιών) του συστήματος με τις υπόλοιπες υπηρεσίες της πλατφόρμας. Περιλαμβάνει το Σύστημα Διαχείρισης Βάσεων Δεδομένων της Virtuoso, στο οποίο φιλοξενείται ο γράφος της οντολογίας, καθώς και τη διεπαφή προγραμματισμού

εφαρμογών Jena API, η οποία μεσολαβεί σε κάθε επικοινωνία της οντολογίας με το υπόλοιπο σύστημα. Ο γράφος περιέχει την οντολογία που εκφράζει το σύστημα, σε μορφή OWL, δηλαδή τις τριπλέτες και τους κανόνες που διέπουν τις οντότητες του συστήματος. Η διεπαφή Jena API είναι το εργαλείο που επιτρέπει την ανάκτηση, την προσθήκη, την ενημέρωση αλλά και τη διαγραφή δεδομένων από την οντολογία, καθώς και την αξιοποίηση κάποιου μηχανισμού αιτίσσης (στην περίπτωση μας του Pellet Reasoner), ώστε να γίνει ο συλλογισμός πάνω στα δεδομένα και να συμπεράνει το σύστημα χρήσιμη πληροφορία.

Για τις ανάγκες περιγραφής των συσκευών / αισθητήρων επιλέχθηκε η οντολογία SSN, η οποία παρέχει μεγάλη εκφραστικότητα στην περιγραφή των οντοτήτων (όπως εξηγήθηκε και στο Κεφάλαιο 2). Στην εικόνα 3.4 φαίνονται οι κύριες κλάσεις και οι συσχετίσεις στην οντολογία SSN.



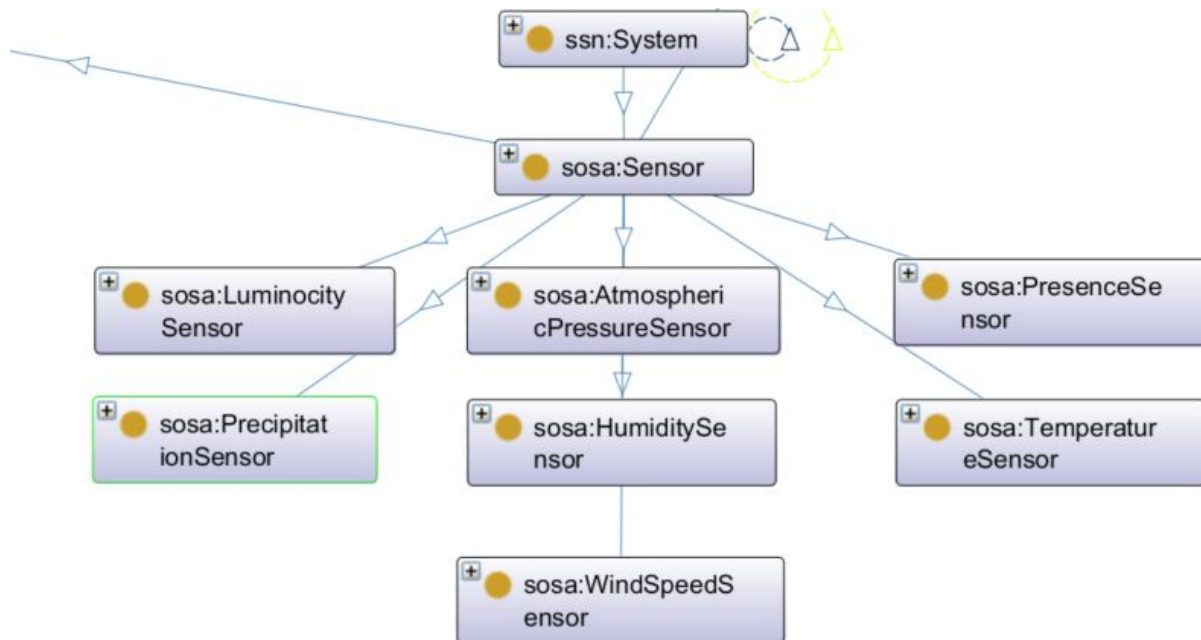
Εικόνα 3.4: Η οντολογία SSN..

Μπορούμε να δούμε πως η οντολογία αυτή είναι γενικής χρήσης και ανεξάρτητη από εφαρμογές και προσφέρει εκφραστικότητα για τις έννοιες:

1. Αισθητήρας (Sensor).
2. Ιδιότητα που παρατηρεί, δηλαδή τι μετράει ο αισθητήρας (ObservableProperty).
3. Παρατήρηση, δηλαδή την μέτρηση που έκανε ο αισθητήρας (Observation).
4. Οντότητα την οποία αφορά η μέτρηση, δηλαδή η ιδιότητα που μετράει ο αισθητήρας σε τι ανήκει (FeatureOfInterest).

Χρειάστηκε όμως να προστεθεί εκφραστικότητα στην οντολογία σύμφωνα με τις προδιαγραφές του συστήματός μας, η οποία προσφέρει επιπλέον λειτουργικότητα στις εφαρμογές.

Παρακάτω παρατίθενται εικόνες με την επιπλέον εκφραστικότητα, με την μορφή κλάσεων, υποκλάσεων, ιδιοτήτων και κανόνων που υλοποιήθηκαν.

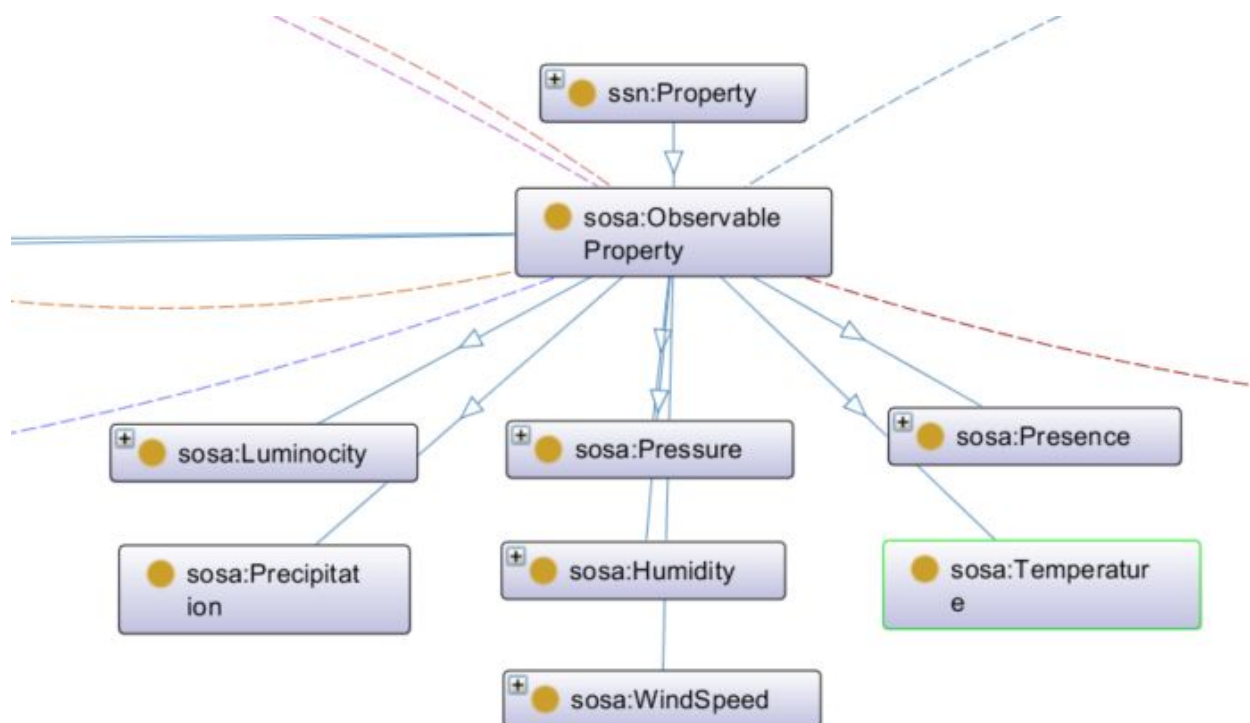


Εικόνα 3.5: Κατηγορίες αισθητήρων.

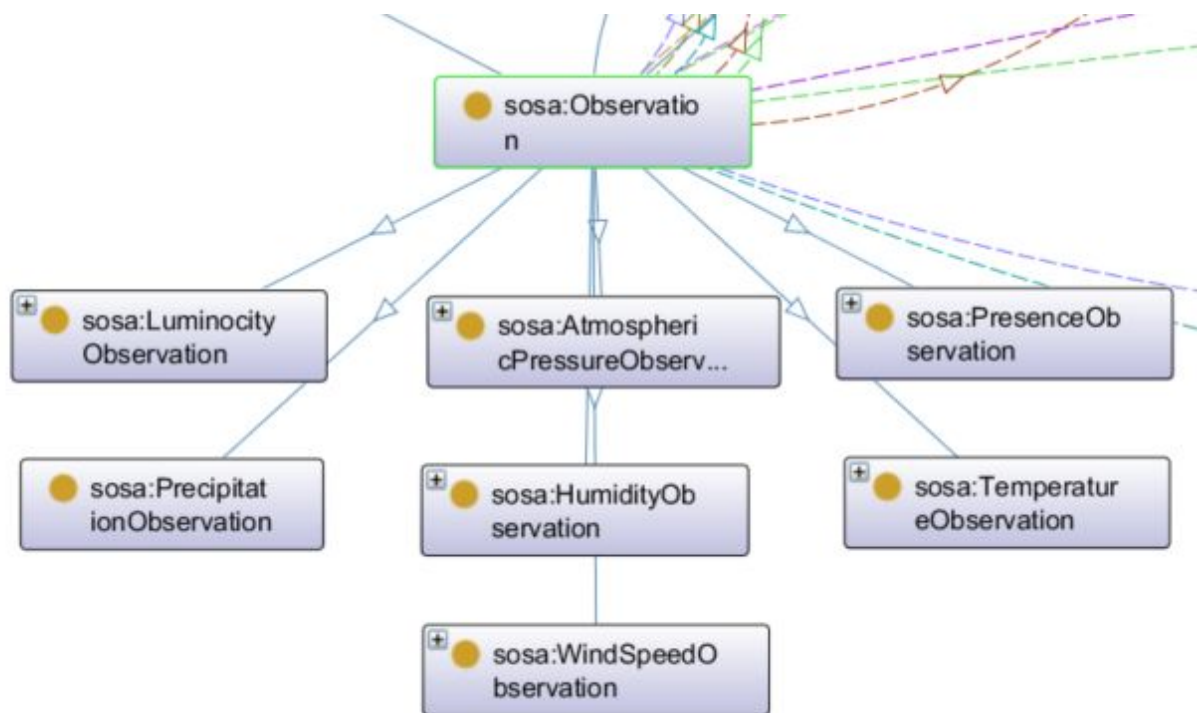
Το σύστημά μας υποστηρίζει 7 είδη αισθητήρων, τα οποία φαίνονται στην εικόνα 3.5 και τα οποία προστέθηκαν σαν υποκλάση της ήδη υπάρχουσας Κλάσης “sosa:Sensor” που υπήρχε στην οντολογία SSN, όπως και η υπερκλάση αυτής που είναι η “ssn:System”. Οι κατηγορίες είναι:

- Ατμοσφαιρικής Πίεσης (πόλη)
- Θερμοκρασίας (εσωτερική για σπίτι - εξωτερική για πόλη)
- Υγρασίας (εσωτερική για σπίτι - εξωτερική για πόλη)
- Φωτεινότητας (εσωτερική για σπίτι - νεφοκάλυψη για πόλη)
- Βροχόπτωσης (πόλη)
- Παρουσίας ατόμων (σπίτι)
- Ταχύτητα ανέμου (πόλη)

Αντίστοιχα, για κάθε κατηγορία αισθητήρα που δημιουργήθηκε, δημιουργήθηκαν και οι υποκλάσεις που φαίνονται στις εικόνες 3.6 και 3.7 για τις κλάσεις “sosa:ObservableProperty”, “sosa:Observation” αντίστοιχα. Η λογική ήταν ότι ένας αισθητήρας θερμοκρασίας (κλάση “sosa:Sensor”) παρατηρεί την μετρήσιμη ιδιότητα θερμοκρασία (κλάση “sosa:ObservableProperty”) και κάνει μετρήσεις θερμοκρασίας (κλάση “sosa:Observation”), άρα όπως γίνεται κατανοητό έπρεπε να δημιουργηθούν οι υποκλάσεις που αντιστοιχούν σε κάθε κατηγορία (θερμοκρασία, υγρασία κλπ).

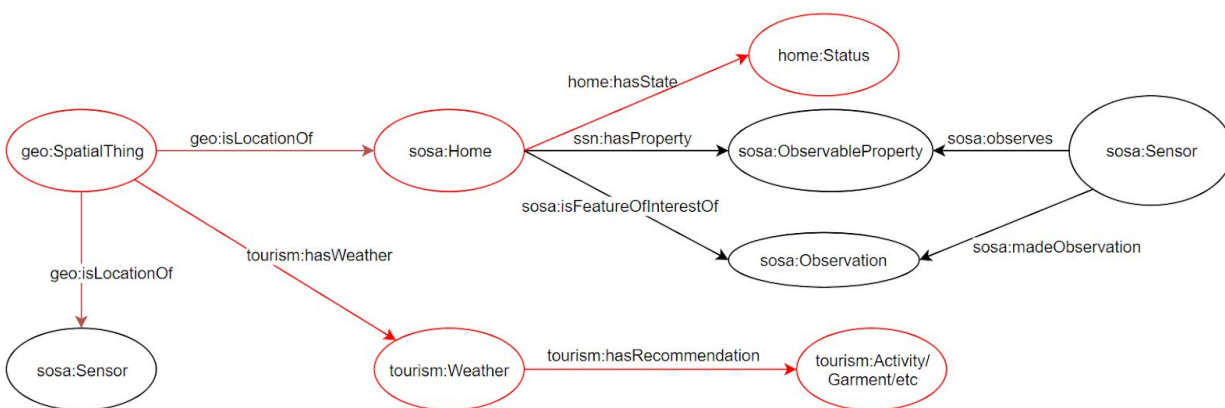


Εικόνα 3.6: Κατηγορίες μετρήσιμων ιδιοτήτων



Εικόνα 3.7: Κατηγορίες παρατηρήσεων.

Στην οντολογία μας, στα πλαίσια των εφαρμογών και των δυνατοτήτων που προσφέρουν χρειάστηκε να ενσωματωθούν στην αρχική οντολογία SSN κάποιες μικρότερες οντολογίες και να δημιουργηθούν κάποιες κλάσεις και ιδιότητες από εμάς. Η ενσωμάτωση των επιπλέον υπο-οντολογιών έγινε δημιουργώντας τα κατάλληλα Object Properties πάνω στην οντολογία. Αυτή η ενσωμάτωση φαίνεται στο σχήμα 3.8 και αναλύεται στις επόμενες παραγράφους. Με κόκκινο χρώμα φαίνονται οι καινούριες κλάσεις και ιδιότητες που ενσωματώθηκαν στην οντολογία SSN. Με μαύρο χρώμα είναι οι κλάσεις και ιδιότητες της οντολογίας SSN.



Εικόνα 3.8: Καινούριες κλάσεις και σύνδεση με υπάρχουσες.

Στην οντολογία μας, χρειάστηκε να προστεθεί η περιγραφή των οντότητα οντοτήτων σπιτιού και πόλης. Για την οντότητα πόλης χρησιμοποιήθηκε η οντολογία “geo”²⁶. Η κλάση “**geo:SpatialThing**” είναι που εκφράζει μια πόλη. Το Object Property “**geo:isLocationOf**” είναι αυτό που συνδέει μια πόλη με τις υπόλοιπες οντότητες. Εκφράζει δηλαδή ότι ένα σπίτι ή ένας αισθητήρας ανήκουν σε μια πόλη. Επίσης, για την οντότητα σπιτιού κατασκευάστηκε η κλάση “**sosa:Home**” η οποία ενσωματώθηκε στην υπάρχουσα οντολογία SSN. Πλέον δηλαδή υπάρχουν δηλωμένες οι τριπλέτες:

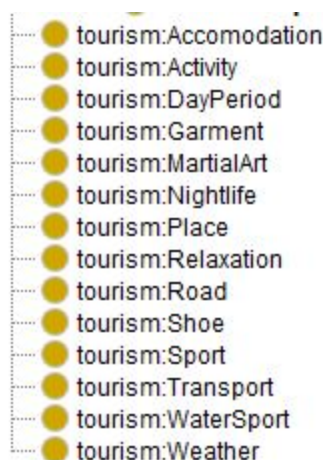
geo:SpatialThing geo:isLocationOf sosa:Home

geo:SpatialThing geo:isLocationOf sosa:Sensor

Επιπλέον, είναι σημαντικό να αναφερθεί ότι η κλάση sosa:Home, συμπίπτει με την υπάρχουσα κλάση της οντολογίας SSN “sosa:FeatureOfInterest”, η οποία εκφράζει την οντότητα στην οποία ανήκει η μετρήσιμη ιδιότητα (sosa:ObservableProperty) που παρατηρείται. Έτσι, έγινε η σχεδιαστική επιλογή κάθε οντότητα της κλάσης “sosa:Home”, δηλαδή τα σπίτια, να ανήκουν και στην κλάση “sosa:FeatureOfInterest”. Αυτό επιτυγχάνει την ένταξη της κλάσης στην υπάρχουσα οντολογία SSN.

²⁶ http://www.w3.org/2003/01/geo/wgs84_pos#

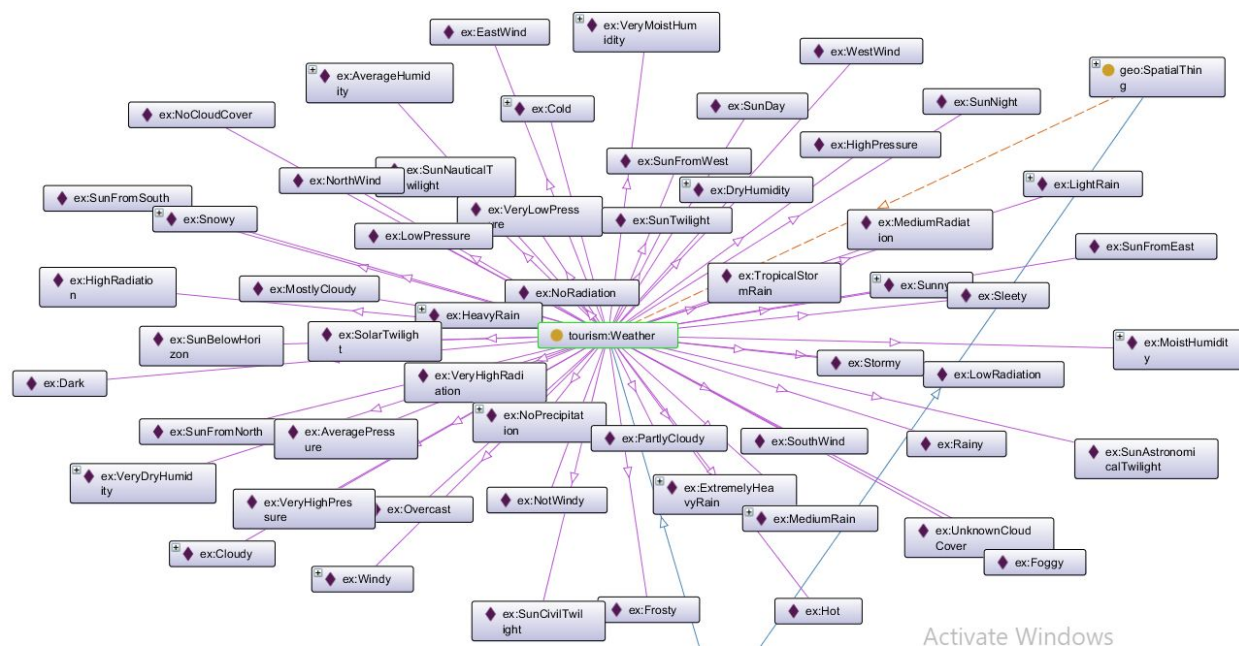
Ακόμα, για την περάτωση της υπηρεσίας δημιουργίας εφαρμογών χρησιμοποιήθηκαν κομμάτια από δύο ακόμα υπο-οντολογίες, την **“tourism”** που περιγράφει καιρικές συνθήκες σε πόλη και την **“home”** που εκφράζει αντίστοιχες “καιρικές” καταστάσεις σε σπίτι, οι οποίες εντάχθηκαν στην αρχική οντολογία SSN. Αυτές οι οντολογίες χρησιμοποιήθηκαν έτοιμες από το ιντερνετ²⁷. Στην εικόνα 3.9 φαίνονται οι κλάσεις της οντολογίας “tourism”.



Εικόνα 3.9 Κλάσεις που εκφράζουν τομείς της καθημερινότητας.

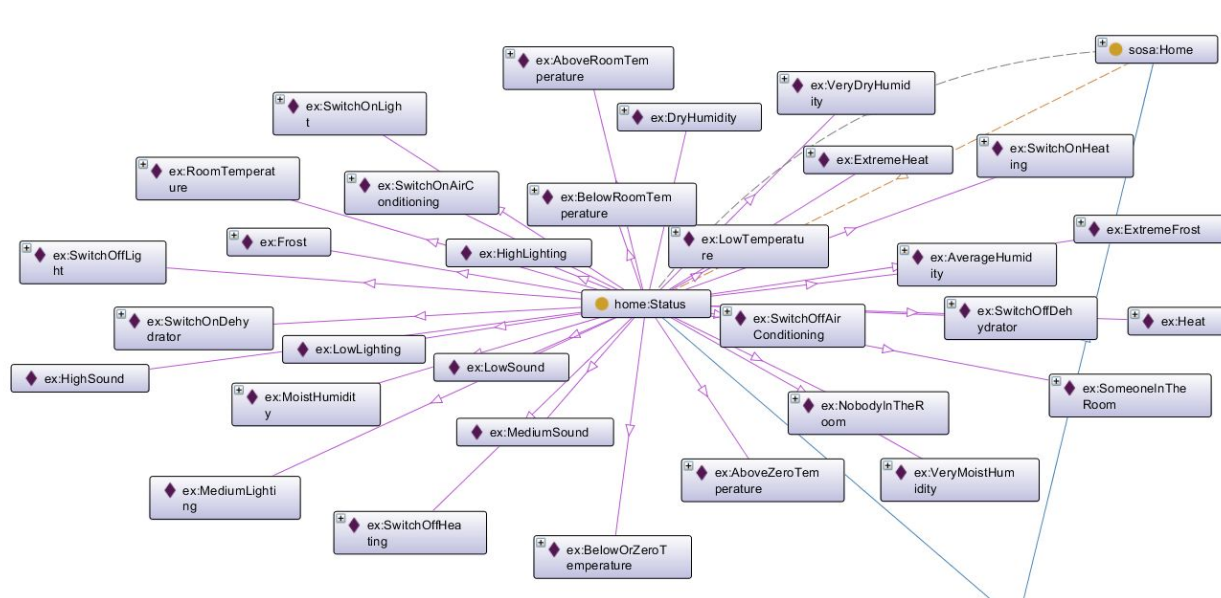
Η οντολογία “tourism” καιρικών συνθηκών πόλης μας προσφέρει την δυνατότητα να εκφράσουμε τις μετρήσεις των αισθητήρων με έναν φιλικό προς τον άνθρωπο τρόπο. Έχουμε την δυνατότητα δηλαδή να κάνουμε το σύστημα να καταλαβαίνει και να συμπεραίνει σωστά τι σημαίνουν οι μετρήσεις που παίρνει. Στην εικόνα 3.10 μπορεί να φανεί οπτικά η δομή της οντολογίας “tourism” που χρησιμοποιήθηκε. Η κλάση **“tourism:Weather”** εκφράζει τον καιρό και έχει σαν στιγμιότυπα τις καιρικές συνθήκες που φαίνονται στην εικόνα. Μπορεί να φανεί και η κλάση **“geo:SpatialThing”**, η οποία περιγράφει τις πόλεις και συνδέεται μέσω του Object Property που δημιουργήθηκε **“tourism:hasWeather”** με οποιοδήποτε στιγμιότυπο της κλάσης “tourism:Weather”.

²⁷ <http://sensormeasurement.appspot.com/>



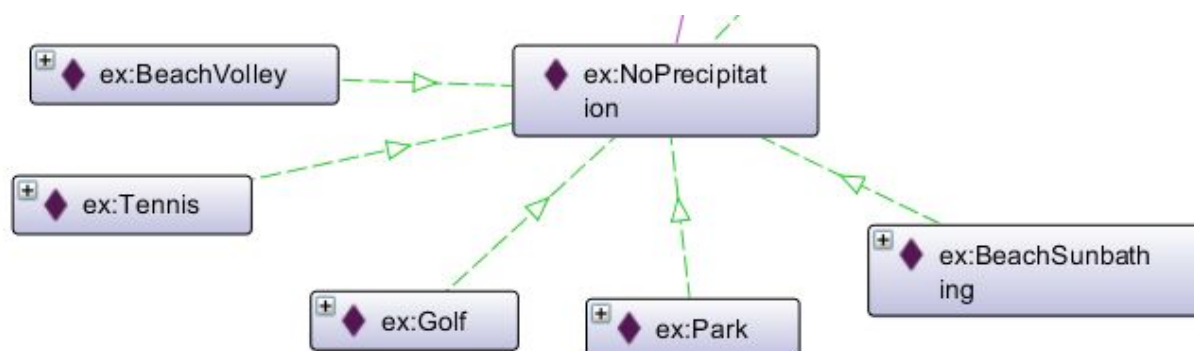
Εικόνα 3.10: Καιρικές συνθήκες πόλης σαν στιγμιότυπα της κλάσης *tourism:Weather*.

Η δεύτερη οντολογία “home” συνθηκών σπιτιού, κάνει το ίδιο με την πρώτη, αναφερόμενη σε σπίτια αυτή τη φορά. Στην εικόνα 3.11 μπορεί να φανεί οπτικά η δομή της οντολογίας. Η κλάση “**home:Status**” εκφράζει τον καιρό και έχει σαν στιγμιότυπα τις καιρικές συνθήκες που φαίνονται στην εικόνα. Μπορεί να φανεί και η κλάση “**sosa:Home**”, η οποία περιγράφει σπίτια και συνδέεται μέσω του Object Property που δημιουργήθηκε “**home:hasState**” με οποιοδήποτε στιγμιότυπο της κλάσης “home:Status”.



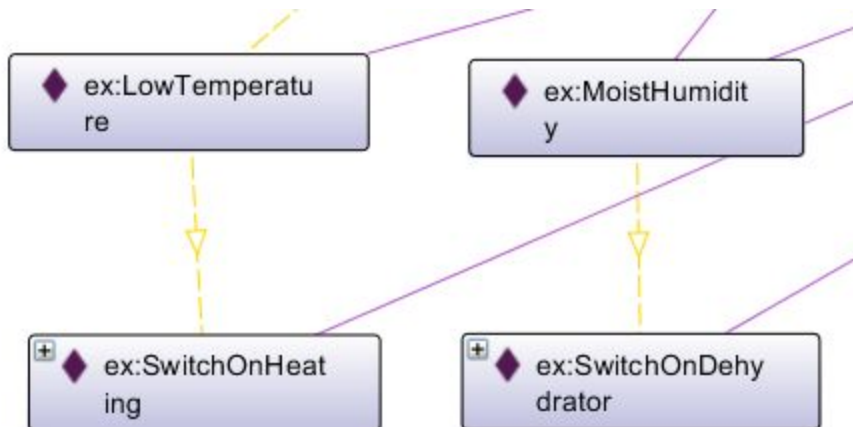
Εικόνα 3.11: Συνθήκες σπιτιού σαν στιγμιότυπα της κλάσης `home:Status`.

Τώρα, αν πάρουμε μια οποιαδήποτε καιρική συνθήκη πόλης από την πρώτη οντολογία, θα δούμε ότι μέσω του Object Property **“tourism:hasRecommendation”**, σχετίζεται με κάποιες άλλες οντότητες οι οποίες αποτελούν στιγμιότυπα των υπόλοιπων κλάσεων της οντολογίας που φαίνονται στην εικόνα 3.9, (δηλαδή των κλάσεων “tourism:Accommodation”, “tourism:Activity”, “tourism:Garment” κλπ.). Στην ουσία εδώ φαίνεται πως η οντολογία όχι μόνο περιγράφει καιρικές συνθήκες αλλά και τομείς της καθημερινότητας οι οποίοι σχετίζονται με την εκάστοτε καιρική συνθήκη, όπως εξωτερική δραστηριότητα (tourism:Activity) που μπορεί να γίνει, ρούχο (tourism:Garment) που μπορεί να φορεθεί, σπορ (tourism:Sport) και άλλα. Για παράδειγμα, στην εικόνα 3.12 μπορεί να φανεί τι συμβουλεύει η οντολογία στην περίπτωση που η συνθήκη που επικρατεί σε μια πόλη είναι η “ex:NoPrecipitation”, δηλαδή δεν βρέχει.



Εικόνα 3.12 Συμβουλές οντολογίας για περίπτωση που δεν υπάρχει βροχή.

Αντίστοιχα, στην οντολογία “home” για συνθήκες σπιτιού, υπάρχουν αντίστοιχες σχέσεις που περιγράφουν αυτοματισμούς που πρέπει να γίνουν σε κάθε περίπτωση. Για παράδειγμα αν σε ένα σπίτι επικρατεί η συνθήκη “LowTemperature”, δηλαδή κρύο (χαμηλή θερμοκρασία), τότε ο αυτοματισμός που πρέπει να γίνει είναι να ξεκινήσει να λειτουργεί η θέρμανση. Στην εικόνα 3.13 φαίνεται η σχέση αυτή μαζί με την περίπτωση της υψηλής υγρασίας (“ex:MoistHumidity”).



Εικόνα 3.13: Συμβουλές οντολογίας για περίπτωση χαμηλής θερμοκρασίας και υψηλής υγρασίας..

Τα παραπάνω δεδομένα συνδυάστηκαν για να δημιουργηθούν οι SWRL κανόνες που χρησιμοποιούνται για τις σημασιολογικές εφαρμογές. Η ιδέα για την δημιουργία των σημασιολογικών εφαρμογών ήταν το σύστημα να παίρνει τις τελευταίες μετρήσεις των αισθητήρων, να τις ερμηνεύει μέσω του reasoner Pellet και να παρουσιάζουν ένα αποτέλεσμα που προκύπτει από τον συλλογισμό πάνω στις μετρήσεις αυτές του συστήματος. Άρα οι εφαρμογές μέσω των κανόνων SWRL έπρεπε να υλοποιούν τα παρακάτω:

1. Να καταλαβαίνουν και να παρουσιάζουν με ένα φιλικό προς τον χρήστη τρόπο την ερμηνεία πάνω στις μετρήσεις των αισθητήρων.
2. Να δίνουν συμβουλές για άλλα πεδία της καθημερινότητας στον χρήστη ανάλογα με τις καιρικές συνθήκες που επικρατούν σε μια πόλη.
3. Να ενεργοποιούν αυτοματισμούς ανάλογα με τις συνθήκες που επικρατούν σε ένα σπίτι.
4. Τελικά να παρουσιάζεται ένα τελικό συμπέρασμα το οποίο μπορεί να προκύψει και είναι ενδιαφέρον προς τον χρήστη, παλι μέσω ενεργοποίησης του reasoner.

Συγκεκριμένα παραδείγματα παρουσιάζονται παρακάτω.

Καιρικές Συνθήκες πόλης:

Κανόνας SWRL με όνομα “*HeavyRain*”:

```
sosa:PrecipitationObservation(?observation) ^ sosa:hasSimpleResult(?observation, ?result) ^
swrlb:greaterThanOrEqual(?result, 20) ^ swrlb:lessThan(?result, 50) -> sosa:deducesPrecipitation(?observation,
ex:HeavyRain)
```

Αυτός ο κανόνας παίρνει το αποτέλεσμα μιας μέτρησης βροχόπτωσης σε μια πόλη, ελέγχει την τιμή της και ανάλογα με αυτήν (αν είναι μεγαλύτερη ή ίση με 20) συμπεραίνει ότι η πόλη στην οποία ανήκει ο αισθητήρας επικρατεί η αντίστοιχη συνθήκη (βρέχει πολύ). Υπάρχουν κανόνες που κάνουν το αντίστοιχο για όλες τις πιθανές τιμές βροχόπτωσης και για όλα τα μετρήσιμα χαρακτηριστικά μιας πόλης. Στη συνέχεια έχει οριστεί στην οντολογία η τριπλέτα:

ex:HeavyRain tourism:isRecommendation ex:Umbrella

Η οποία εκφράζει το ότι αν βρέχει πολύ τότε πρέπει κάποιος να πάρει ομπρέλα, το οποίο ανήκει στην κατηγορία κλάσης “*tourism:Garment*”.

Συνθήκες σπιτιού(Αυτοματισμοί):

Κανόνας SWRL με όνομα “*VeryMoistHumidity*”:

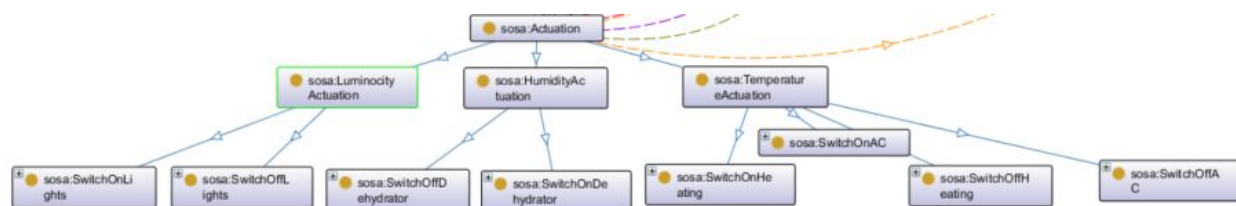
```
sosa:hasSimpleResult(?observation, ?result) ^ sosa:HumidityObservation(?observation) ^ swrlb:greaterThan(?result,
80) -> sosa:roomHumidityState(?observation, ex:VeryMoistHumidity)
```

Αυτός ο κανόνας παίρνει το αποτέλεσμα μιας μέτρησης υγρασίας ενός σπιτιού, ελέγχει την τιμή της και ανάλογα με αυτήν (αν είναι πάνω από 80) συμπεραίνει ότι το σπίτι στο οποίο ανήκει ο αισθητήρας επικρατεί η αντίστοιχη συνθήκη (πολύ υψηλή υγρασία). Στη συνέχεια έχει οριστεί στην οντολογία η τριπλέτα:

ex:VeryMoistHumidity home:hasRecommendation ex:SwitchOnDehydrator

Η οποία εκφράζει το ότι αν επικρατεί πολύ υψηλή υγρασία τότε σε ένα σπίτι πρέπει να ενεργοποιηθεί ο αφυγραντήρας(dehydrator).

Στην οντολογία μας έχουμε προσθέσει ένα Datatype Property με το όνομα: “**sosa:ActuationEnabled**”, και το έχουμε συσχετίσει με όλες τις οντότητες αυτοματισμού (Actuation). Κάναμε την παραδοχή ότι όταν δουλεύει ένας αυτοματισμός (Actuation), η τιμή αυτού του Datatype Property θα είναι “1” και όταν δεν δουλεύει θα είναι “0”. Άρα πρέπει να ελέγχουμε πότε χρειάζεται να ενεργοποιηθεί ένας αυτοματισμός στο σύστημά μας. Αυτό γίνεται με κάποιους SWRL κανόνες που κατασκευάστηκαν, ένας για κάθε τύπο αυτοματισμού όπως φαίνεται στην εικόνα 3.14.



Εικόνα 3.14: Κλάσεις που εκφράζουν αυτοματισμούς σε σπίτι.

Για παράδειγμα για τον παραπάνω αυτοματισμό υγρασίας ενός σπιτιού (ex:SwitchOnDehydrator) ο κανοντας που ορίζει ότι στο αντίστοιχο σπίτι θα ενεργοποιηθεί ο αυτοματισμός είναι ο παρακάτω:

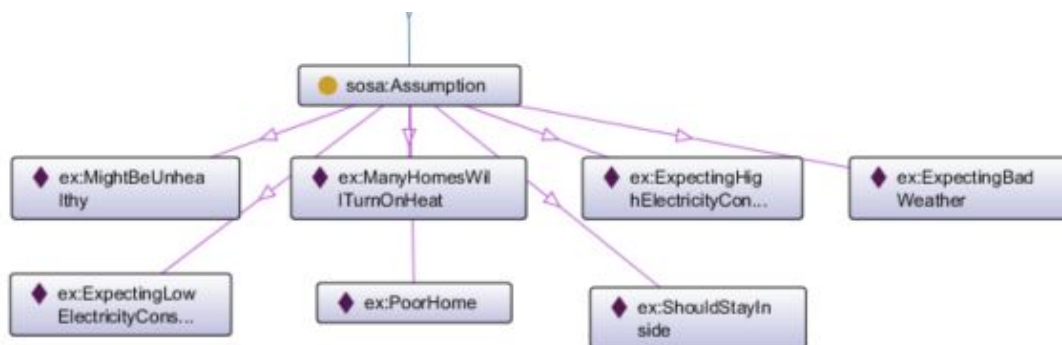
Κανόνας SWRL με όνομα “SwitchOnDehydratorActuation”:

```
home:hasActuationState(?foi, ex:SwitchOnDehydrator) ^ sosa:hasFeatureOfInterest(?act, ?foi) ^
sosa:SwitchOnDehydrator(?act) -> sosa:actuationEnabled(?act, 1)
```

Αυτός ο κανόνας ελέγχει αν σε ένα σπίτι αν η κατάστασή του απαιτεί την ενεργοποίηση του αυτοματισμού “sosa:SwitchOnDehydrator”, και αν ισχύει αλλάζει την τιμή του Datatype Property “sosa:ActuationEnabled” του αυτοματισμού “sosa:SwitchOnDehydrator” σε “1”. Ένα σπίτι συνδέεται με έναν αυτοματισμό μέσω του Object Property “sosa:isFeatureOfInterestOf”.

sosa:Home sosa:isFeatureOfInterestOf sosa:Actuation

Στην οντολογία μας έχουμε προσθέσει ακόμα μια κλάση με το όνομα “**sosa:Assumption**”. Σκοπός αυτής της κλάσης είναι να περιγράψει χρήσιμα επιπλέον συμπεράσματα που μπορεί να παραχθούν λαμβάνοντας υπόψη τις παρατηρήσεις των αισθητήρων. Για παράδειγμα ένα χρήσιμο συμπέρασμα είναι το γεγονός πως αν μια πόλη παρουσιάζει χαμηλή ατμοσφαιρική πίεση τότε περιμένουμε να χαλάσει ο καιρός. Στην εικόνα 3.15 φαίνονται τα στιγμιότυπα της κλάσης “sosa:Assumption”.



Εικόνα 3.15: Κλάση sosa:Assumption με στιγμιότυπα της.

Για κάθε ένα από τα στιγμιότυπα κατασκευάστηκε ένας SWRL κανόνας που ελέγχει αν πληρούνται οι προϋποθέσεις για να ισχύουν. Για παράδειγμα ο κανόνας που συμπεραίνει ότι μια πόλη θα έχει κακό καιρό αν υπάρχει χαμηλή ατμοσφαιρική πίεση είναι:

```
geo:SpatialThing(?city) ^ tourism:hasPressure(?city, ex:LowPressure) -> sosa:assumes(?city, ex:ExpectingBadWeather)
```

Στον πίνακα 3.1 παρατίθενται όλοι οι κανόνες SWRL. Για κάθε μια από τις παρακάτω περιπτώσεις υπάρχει ένας κανόνας SWRL που την ελέγχει.

Πίνακας 3.1: Κανόνες SWRL.

Κανόνες Βροχής (mm/h)	
>100	TropicalStormRain
>50 - 100	ExtremelyHeavyRain
>20 - 50	HeavyRain
>5 - 20	MediumRain
>0 - 5	LightRain
=0	NoPrecipitation
Κανόνες ατμοσφαιρικής πίεσης (Pascal)	
<998000	VeryLowPressure
>998000 - 1008000	LowPressure
>100800 - 1018000	AveragePressure
>1018000 - 1028000	HighPressure
>1028000	VeryHighPressure
Κανόνες νεφοκάλυψης (Lux)	
>9900	Sunny
>4900 - 5100	Cloudy
=0	Dark

Κανόνες υγρασίας (%)	
<30	VeryDryHumidity
>30 - 40	DryHumidity
>40 - 70	AverageHumidity
>70 - 80	MoistHumidity
>80	VeryMoistHumidity
Κανόνες φωτεινότητας (Lux)	
>20 - 150	LowLighting
>150 - 750	MediumLighting
>750 - 2000	HighLighting
Κανόνες θερμοκρασίας (Celsius)	
<20	BelowRoomTemperature
>20 - 25	RoomTemperature
>25	AboveRoomTemperature
Κανόνες παρουσίας ατόμου (0 ή 1)	
=0	NobodyInTheRoom
=1	SomeoneInTheRoom

Στον πίνακα 3.2 παρουσιάζονται οι επιπλέον κανόνες SWRL περιγραφικά οι οποίοι ελέγχουν με τη σειρά τους κάποιες συνθήκες και ανάλογα παρουσιάζουν σαν αποτέλεσμα ένα χρήσιμο συμπέρασμα που προκύπτει. Για παράδειγμα ο κανόνας με όνομα “ExpectingBadWeather” ελέγχει αν σε μια πόλη επικρατεί χαμηλή πίεση (LowPressure), το οποίο σημαίνει ότι έρχεται κακοκαιρία. Αντίστοιχα ο κανόνας “ExpectingHighElectricityConsumption” ελέγχει αν σε ένα σπίτι είναι ενεργοποιημένος ο κλιματισμός και αν ισχύει τότε συμπεραίνει ότι το σπίτι θα έχει υψηλή κατανάλωση ρεύματος. Το ίδιο ισχύει και την περίπτωση που η θέρμανση είναι ενεργοποιημένη και ταυτόχρονα είναι αναμμένα τα φώτα. Ο κανόνας “ManyHomesWillTurnOnHeat” ελέγχει

αν σε μια πόλη επικρατεί χαμηλή θερμοκρασία (BelowRoomTemperature), κάτι το οποίο σημαίνει ότι αναμένεται πολλά σπίτια να ενεργοποιήσουν την θέρμανση.

Ακόμα, δημιουργήθηκε το Datatype Property “sosa:homeCount”, το οποίο υποδηλώνει το ποσοστό των σπιτιών σε μια πόλη που δεν έχουν ενεργοποιημένη την θέρμανση. Με βάση αυτή την πληροφορία, δημιουργήθηκε ο κανόνας SWRL με όνομα PoorCity ο οποίος ελέγχει το ποσοστό αυτό και αν είναι πάνω από 70% και αν επικρατεί χαμηλή θερμοκρασία στην πόλη (BelowRoomTemperature). Τότε οδηγεί στο συμπέρασμα ότι η πόλη είναι φτωχή γενικά και δεν έχουν τα σπίτια την δυνατότητα για έχουν θέρμανση, κάτι το οποίο μπορεί να είναι χρήσιμη πληροφορία για πολιτικούς κλπ. Κανόνας PoorCity:

```
swrlb:greaterThan(?result, 0.70) ^ sosa:homeCount(?city, ?result) ^ tourism:hasTemperature(?city, ex:BelowRoomTemperature) -> sosa:assumes(?city, ex:PoorCityManyHomesDoNotHaveHeat)
```

Πίνακας 3.2: Κανόνες SWRL για επιπλέον συμπεράσματα οντολογίας..

Συνθήκη	Όνομα κανόνα/Χρήσιμο συμπέρασμα
LowPressure (πόλη)	ExpectingBadWeather
SwitchOnAirConditioning & SwitchOnLight (σπίτι)	ExpectingHighElectricityConsumption
SwitchOffAirConditioning (σπίτι)	ExpectingHighElectricityConsumption
VeryMoistHumidity (σπίτι)	MightBeUnhealthy
BelowRoomTemperature (πόλη)	ManyHomesWillTurnOnHeat
BelowRoomTemperature & VeryMoistHumidity (σπίτι)	PoorHome
BelowRoomTemperature & Rainy (πόλη)	ShouldStayInside
>70% σπιτιών δεν έχουν θέρμανση σε μια πόλη & BelowRoomTemperature (πόλη)	PoorCityManyHomesDoNotHaveHeat

3.6 Υπηρεσία Σύνθεσης Εφαρμογών (Mashup Service)

Η υπηρεσία Mashup υποστηρίζει την δυνατότητα δημιουργίας εφαρμογών, υποκείμενες στο Σημασιολογικό Ιστό, από προγραμματιστές του συστήματος. Ο

προγραμματιστής μέσω γραφικής διεπαφής μπορεί να επιλέξει το είδος της εφαρμογής, δηλαδή αν η εφαρμογή αφορά ένα σπίτι ή μία πόλη από αυτές που υποστηρίζει το σύστημα. Στη συνέχεια επιλέγει για ποιο σπίτι ή πόλη συγκεκριμένα επιθυμεί να δει πληροφορίες, δηλαδή, να δημιουργηθεί μια εφαρμογή για αυτό. Εν τέλει, επιλέγει αν η εφαρμογή θα είναι τύπου ιστορικών δεδομένων ή σημασιολογικού τύπου τρεχουσών τιμών. Προκειμένου να παραχθεί η εκτελέσιμη εφαρμογή, η λειτουργικότητα της όπως την διαμόρφωσε ο προγραμματιστής, περιγράφεται σε μία πρότυπη δομή JSON. Στην εικόνα 3.16 φαίνεται μια δομή JSON για εφαρμογή πόλης ενώ στην εικόνα 3.17 εφαρμογή σπιτιού.

```
{
  "appname": "rethymnotemprec",
  "appscope": "weather",
  "info": [
    {
      "attributes": [
        "temperature"
      ],
      "city": "Rethymno",
      "domains": [
        "Activity",
        "Garment"
      ],
      "func": "LIVE"
    }
  ]
}
```

Εικόνα 3.16: Αναπαράσταση των *mashup* μίας εφαρμογής πόλης σε JSON.

```
{
  "appname": "apartment134",
  "appscope": "home",
  "info": [
    {
      "attributes": [
        "temperature",
        "humidity",
        "luminocity",
        "presence"
      ],
      "city": "Appartment134",
      "domains": [],
      "func": "LIVE"
    }
  ]
}
```

Εικόνα 3.17: Αναπαράσταση των mashup μίας εφαρμογής σπιτιού σε JSON.

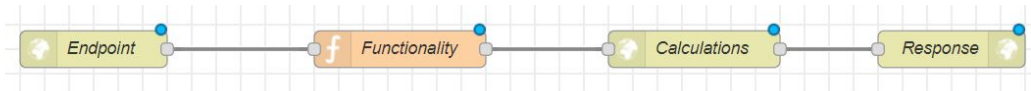
Αναλυτικά, τα χαρακτηριστικά που φαίνονται στις εικόνες 3.16 και 3.16 έχουν την εξής σημασία:

- **“appname”** : Το όνομα της εφαρμογής.
- **“appscope”** : Ο τύπος της εφαρμογής όσον αφορά το αν είναι για σπίτι ή πόλη(**weather** για πόλη, **home** για σπίτι).
- **“info”** : Πίνακας ο οποίος περιέχει το σύνολο των mashup της εφαρμογής.
- **“attributes”**: Πρόκειται για το υποσύνολο των χαρακτηριστικών μέτρησης των αισθητήρων ενός σπιτιού ή πόλης που περιλαμβάνονται στο mashup και για τα οποία ενδιαφέρεται ο χρήστης να λάβει πληροφορία (πχ. *Temperature, humidity*), δηλαδή το mashup αυτό θα απεικονίζει μία σύνθεση από δεδομένα που αφορούν τα συγκεκριμένα χαρακτηριστικά.
- **“city”** : Αφορά το όνομα (id) του σπιτιού ή της πόλης για το οποίο μας ενδιαφέρει να λάβουμε πληροφορίες από τους αισθητήρες που είναι εγκατεστημένοι εκεί.
- **“domains”**: Στην περίπτωση που η εφαρμογή αφορά ταυτόχρονα πόλη και τρέχουσες τιμές, ο χρήστης μπορεί να ρωτήσει για έναν τομέα της πραγματικής ζωής, να του απεικονιστεί συμβουλή, η οποία παράγεται ανάλογα με την τιμή που έχουν οι αισθητήρες που μετρούν τα χαρακτηριστικά που επέλεξε.

- **“func”**: Πρόκειται για τον «τύπο» του mashup, μία κωδική λέξη (συμβολοσειρά) η οποία ορίζει τι κατάσταση (ή συμβάν) απεικονίζει το mashup με την σύνθεση των δεδομένων. Κάθε διαφορετικός «τύπος» mashup που υποστηρίζει το σύστημα αντιστοιχείται με μία -μοναδική- κωδική λέξη όπως βλέπουμε παρακάτω:
- **“MAX_24hr”**: Γράφημα γραμμής (Line Chart) που απεικονίζει τη μέγιστη τιμή του **“attribute”** ανά ώρα, για τις τελευταίες 24 ώρες.
- **“MIN_24hr”**: Γράφημα γραμμής που απεικονίζει την ελάχιστη τιμή του **“attribute”** ανά ώρα, για τις τελευταίες 24 ώρες.
- **“AVERAGE_24hr”**: Γράφημα γραμμής που απεικονίζει τη μέση τιμή του **“attribute”** ανά ώρα, για τις τελευταίες 24 ώρες.
- **“MAX_7D”**: Γράφημα γραμμής που απεικονίζει τη μέγιστη τιμή του **“attribute”** ανά μέρα, για τις τελευταίες 7 μέρες.
- **“MIN_7D”**: Γράφημα γραμμής που απεικονίζει την ελάχιστη τιμή του **“attribute”** ανά μέρα.
- **“AVERAGE_7D”**: Γράφημα γραμμής που απεικονίζει τη μέση τιμή του **“attribute”** ανά μέρα, για τις τελευταίες 7 μέρες.
- **“LIVE”**: Έχουμε κάνει την επιλογή αυτή η κωδική λέξη να δηλώνει ότι η εφαρμογή είναι σημασιολογικού τύπου, ενώ όλες οι προηγούμενες αποτελούν ιστορικές εφαρμογές. Αναπαριστάται η σημασιολογική ερμηνεία της οντολογίας πάνω στις τρέχουσες τιμές των **“attributes”** της πόλης που επιλέχθηκε. Επίσης αναπαριστώνται και οι συμβουλές της οντολογίας ανάλογα με τις επιλογές που έγιναν στο πεδίο **“domains”**, καθώς επίσης και επιπλέον συμπεράσματα που μπορεί να ερμηνεύσει η οντολογία.

Για κάθε διαφορετική κωδική λέξη από τις παραπάνω, υπάρχει ένας αντίστοιχος αλγόριθμος που υλοποιεί το mashup που ορίζει η κωδική λέξη (βλ. παρακάτω).

Εντός της υπηρεσίας Mashup του συστήματος, η εφαρμογή υφίσταται σαν σύνθεση τεσσάρων κόμβων (nodes) της υπηρεσίας Node-RED που ανταλλάσσουν δεδομένα σε προκαθορισμένες συνδέσεις. Κάθε κόμβος έχει μία συγκεκριμένη λειτουργικότητα. Οι κόμβοι εκτελούνται με προκαθορισμένη σειρά ροής (flow) και η σύνθεση τους σαν σύνολο συνιστούν την εκτελέσιμη εφαρμογή. Ουσιαστικά, κάθε εφαρμογή υφίσταται αρχικά σαν μια ροή (flow) η οποία αποθηκεύεται στην βάση αποθήκευσης εφαρμογών της υπηρεσίας Node-RED, όπου μετά είναι έτοιμη προς εκτέλεση. Το πρότυπο ροής που εφαρμόζεται στην εργασία φαίνεται στην Εικόνα 3.18.



Εικόνα 3.18: Πρότυπο ροής (flow) σύνθεσης εκτελέσιμης εφαρμογής.

Παρακάτω περιγράφεται κάθε κόμβος της ροής, από αριστερά προς τα δεξιά όπως φαίνονται στην Εικόνα 3.18:

1. Ο πρώτος κόμβος “Endpoint”, πρόκειται για ένα ακροατή (Listener) ο οποίος δέχεται HTTP αιτήματα και αποτελεί το σημείο επαφής (endpoint) της εφαρμογής. Το URL του τελικού σημείου, διαμορφώνεται ως : {δημόσια διεύθυνση IP της υπηρεσίας Mashup:port / όνομα εφαρμογής }. Η προσπέλαση του γίνεται με την μέθοδο GET του πρωτοκόλλου HTTP. Για παράδειγμα, η εφαρμογή με όνομα “apartment134”, είναι προσβάσιμη με ένα HTTP αίτημα της μεθόδου GET στο URL: “http://147.27.60.65:1880/apartment134 ”. Με την προσπέλαση του τελικού σημείου της εφαρμογής η εκτέλεση της ροής προχωράει στο δεύτερο κόμβο.
2. Ο δεύτερος κόμβος “Functionality” περιέχει αποθηκευμένη την πληροφορία JSON που αναπαριστά τα mashup της εφαρμογής (Αναλύθηκε στην προηγούμενη παράγραφο. Για παράδειγμα εδώ θα μπορούσε να υπάρχει αποθηκευμένη η πληροφορία JSON της εικόνας 3.15 ή 3.16 που είδαμε νωρίτερα). Σκοπός αυτού του κόμβου είναι να μεταδώσει την πληροφορία JSON στον επόμενο κόμβο της ροής “Calculations”.
3. Ο Τρίτος κόμβος “Calculations” δέχεται σαν είσοδο την πληροφορία JSON από τον κόμβο “Functionality”. Σε αυτό το στάδιο, εκτελούνται οι αλγόριθμοι οι οποίοι υλοποιούν τα mashup που ορίζει η πληροφορία JSON που ελήφθη στην είσοδο. Πιο συγκεκριμένα η πληροφορία JSON αναλύεται, κάθε διαφορετικό mashup καλεί αρμόδια συνάρτηση αλγορίθμου που το υλοποιεί ανάλογα με την τιμή που έχει το πεδίο “func” που περιέχει. Οι πιθανές τιμές είναι οι αλγόριθμοι που περιγράφηκαν προηγουμένως:
 - MAX_24hr
 - MIN_24hr
 - AVERAGE_24hr

- MAX_7D
- MIN_7D
- AVERAGE_7D
- LIVE: Αν είναι αυτή η τιμή του func πρόκειται για σημασιολογική εφαρμογή και όχι ιστορική και τότε ελέγχεται η τιμή του πεδίου “appscope”.
 - Αν η τιμή είναι “weather”, τότε εκτελείται ο αλγόριθμος που προορίζεται για τις πόλεις και υπολογίζει τον καιρό στην πόλη, καθώς και τις προτάσεις που παράγουν τα αποτελέσματα αυτά για κάθε τύπο συμβουλής που έχει ζητήσει ο χρήστης στο πεδίο “domains”.
 - Αν η τιμή είναι “home”, τότε εκτελείται ο αλγόριθμος που προορίζεται για τα σπίτια και υπολογίζει την κατάσταση μέσα στο σπίτι, και ανάλογα δείχνει ποιοι αυτοματισμοί θα ενεργοποιηθούν σύμφωνα με τα προηγούμενα αποτελέσματα.

Τα δεδομένα (ιστορικές μετρήσεις των αισθητήρων) που είναι απαραίτητα για τα mashup, ανακτώνται από τους αλγορίθμους μέσω REST αιτημάτων στην υπηρεσία ανάκτησης ιστορικού δεδομένων (βλ.3.3), ενώ στην περίπτωση που έχουμε σημασιολογική εφαρμογή ανατρέχουμε με SPARQL ερωτήματα στην υπηρεσία οντολογίας (βλ.3.5). Όταν οι υπολογισμοί ολοκληρωθούν, παράγεται στην έξοδο του κόμβου ο προσαρμοσμένος κώδικας HTML/Javascript που περιέχει την γραφική αναπαράσταση των mashup της εφαρμογής. Ο κώδικας αυτός προωθείται στον επόμενο κόμβο της ροής “Response”.

4. Ο τελικός κόμβος “Response” προωθεί τον κώδικα -HTML/Javascript- που έλαβε στην είσοδο, στο πρόγραμμα περιήγησης του τελικού χρήστη. Ο τελικός χρήστης ο οποίος προσπέρασε αρχικά το τελικό σημείο της εφαρμογής, βλέπει πλέον στην οθόνη του την πληροφορία της εφαρμογής (την γραφική απεικόνιση των mashup).

Κατά την δημιουργία μιας εφαρμογής από ένα προγραμματιστή, ανεξαρτήτως του τύπου της, παράγεται η σύνθεση κόμβων που υλοποιεί την εκτελέσιμη εφαρμογή και αποθηκεύεται στην βάση δεδομένων της υπηρεσίας Mashup σαν μια ροή (flow) του Node-RED μέσω της Restful διεπαφής που διαθέτει. Έπειτα η εφαρμογή είναι προσβάσιμη από τους τελικούς χρήστες του συστήματος. Η διαφοροποίηση των τύπων

των εφαρμογών υπάρχει στην υπηρεσία Calculations, όπου εκεί γίνεται ανάλυση της JSON περιγραφής της εφαρμογής και δρομολογούνται τα κατάλληλα αιτήματα στις υπηρεσίες (οντολογίας ή ιστορική) που πρέπει.

Όπως βλέπουμε η εκτελέσιμη εφαρμογή υφίσταται ως REST μέθοδος της υπηρεσίας Mashup την οποία «παράγει» ο προγραμματιστής. Παράλληλα ο τελικός χρήστης μπορεί να έχει πρόσβαση σε αυτή από οποιοδήποτε πρόγραμμα περιήγησης.

3.7 Υπηρεσία Ταυτοποίησης και Εξουσιοδότησης Χρηστών (User Authentication - Authorization)

Η υπηρεσία αυτή συνιστά αφετηρία για το σύστημα, καθώς αναλαμβάνει την εγγραφή και τη σύνδεση των χρηστών. Κατά την εγγραφή του χρήστη, ορίζονται τα χαρακτηριστικά που συνθέτουν το προσωπικό του προφίλ όπως όνομα, email, κωδικό πρόσβασης. Για τη σύνδεσή του στο σύστημα πληκτρολογεί στη σελίδα εισόδου το email του και τον κωδικό πρόσβασης. Το αίτημα σύνδεσης δρομολογείται στην Υπηρεσία Ταυτοποίησης και Εξουσιοδότησης Χρηστών (μέσω της RESTful διεπαφής της), η οποία ελέγχει την ταυτότητα του χρήστη. Η ταυτότητα αυτή ελέγχεται χάρη στην αξιοποίηση του μηχανισμού OAuth2.0 και του OAuth2 token (η διαδικασία θα παρουσιαστεί αναλυτικά σε επόμενο κεφάλαιο).

3.8 Υπηρεσία Λήψης Απόφασης Εξουσιοδότησης (Authorization Policy Decision Point).

Η υπηρεσία αυτή ευθύνεται για την λήψη αποφάσεων έγκρισης ή απόρριψης αιτημάτων πρόσβασης που γίνονται από χρήστες προς υπηρεσίες του συστήματος, με κριτήριο τις εφαρμοστέες πολιτικές πρόσβασης που συνιστούν οι ρόλοι εντός της υπηρεσίας Ταυτοποίησης και Εξουσιοδότησης Χρηστών.

Οι αποφάσεις έγκρισης ή απόρριψης αιτημάτων πρόσβασης, προκύπτουν από τους κανόνες ελέγχου πρόσβασης. Ένας κανόνας ελέγχου πρόσβασης σχετίζεται άρρηκτα με ένα ρόλο εντός της υπηρεσίας Ταυτοποίησης και Εξουσιοδότησης Χρηστών καθώς περιγράφει τις προϋποθέσεις που πρέπει να ακολουθεί το αίτημα πρόσβασης οποιουδήποτε χρήστη με αυτό το ρόλο, προκειμένου εγκριθεί. Οι κανόνες ελέγχου πρόσβασης ακολουθούν το πρότυπο XACML και δημιουργούνται στην υπηρεσία Λήψης

Απόφασης Εξουσιοδότησης από την υπηρεσία Ταυτοποίησης και Εξουσιοδότησης Χρηστών, μέσω της Restful επικοινωνίας τους. Η δημιουργία ενός νέου κανόνα ελέγχου πρόσβασης, προκύπτει όταν δημιουργείται στην υπηρεσία Ταυτοποίησης και Εξουσιοδότησης Χρηστών ένας νέος ρόλος που ενσωματώνει μία άδεια πρόσβασης.

Τα αιτήματα των χρηστών, προωθούνται για αξιολόγηση στην υπηρεσία Λήψης Απόφασης Εξουσιοδότησης μέσω της Restful διεπαφής που διαθέτει, από τον Διακομιστή Μεσολάβησης Επιβολής Πολιτικής.

3.9 Διακομιστής Μεσολάβησης Επιβολής Πολιτικής (Policy Enforcement Point Proxy)

Ένας διακομιστής μεσολάβησης είναι ένας διακομιστής που ενεργεί ως ενδιάμεσος για τις αιτήσεις από χρήστες που αναζητούν πόρους από άλλους διακομιστές. Ένας χρήστης συνδέεται με τον διακομιστή μεσολάβησης, ζητώντας κάποια υπηρεσία, όπως ένα αρχείο, μια σύνδεση, μια ιστοσελίδα ή άλλο πόρο που διατίθεται από διαφορετικό διακομιστή. Ο διακομιστής μεσολάβησης αναλαμβάνει να προωθήσει το αίτημα στον διακομιστή που διαθέτει την συγκεκριμένη υπηρεσία και αφού λάβει την απάντηση του ερωτήματος την προωθεί στον αιτούντα.

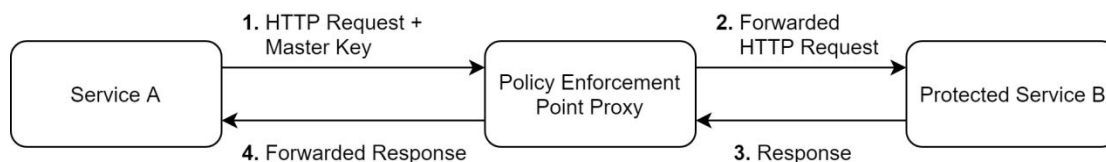
Οι υπηρεσίες της αρχιτεκτονικής που διαθέτουν πόρους οι οποίοι δεν είναι θεμιτό να είναι προσβάσιμοι από μη εξουσιοδοτημένες υπηρεσίες ή χρήστες, δεν προσφέρουν δημόσια την REST διεπαφή τους. Επομένως, κάθε υπηρεσία προκειμένου να μπορεί να εξυπηρετεί εξωτερικά (Σε σχέση με την εικονική μηχανή στην οποία «τρέχει») αιτήματα, διαθέτει ένα τοπικό διακομιστή μεσολάβησης ο οποίος αναλαμβάνει να δέχεται στο δημόσιο τελικό του σημείο τα αιτήματα που προορίζονται για εκείνη και της τα προωθεί.

Ο Διακομιστής Μεσολάβησης Επιβολής Πολιτικής, είναι ένας διακομιστής μεσολάβησης που απαιτεί υποχρεωτικά στην κεφαλίδα των HTTP αιτημάτων που λαμβάνει, ένα από τα δύο παρακάτω διακριτικά, διαφορετικά τα αιτήματα αγνοούνται:

- **OAuth2 token:** Ένα έγκυρο OAuth2 token αντιστοιχεί σε ένα χρήστη και έχει δημιουργηθεί από την υπηρεσία Ταυτοποίησης και Εξουσιοδότησης Χρηστών κατά την είσοδο του στο σύστημα.
- **Master Key:** Πρόκειται για ένα μυστικό κωδικό ο οποίος καθορίζεται κατά την αρχικοποίηση του Διακομιστή Μεσολάβησης Επιβολής Πολιτικής. Κάθε διαφορετικός Διακομιστής Μεσολάβησης Επιβολής Πολιτικής της αρχιτεκτονικής έχει το δικό του -μοναδικό- κωδικό Master key.

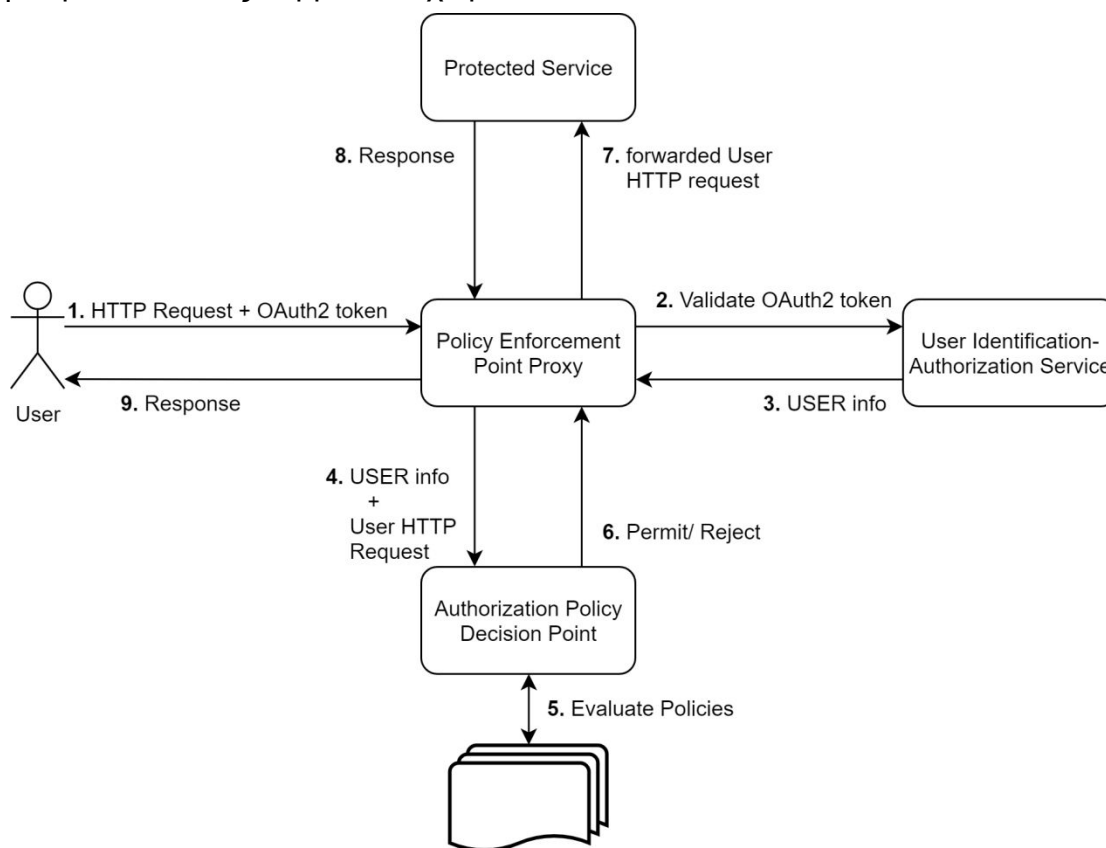
Στην περίπτωση όπου το αίτημα που δέχεται, φέρει στην κεφαλίδα του τον μυστικό κωδικό “Master Key”, εφόσον ο κωδικός Master key είναι σωστός, ο

Διακομιστής PEP το προωθεί στην υπηρεσία για την οποία μεσολαβεί και επιστρέφει την απάντηση της στον αιτούντα (Εικόνα 3.11).



Εικόνα 3.19: Λειτουργία Διακομιστή PEP με την χρήση του μυστικού κωδικού Master Key.

Στην επόμενη περίπτωση αιτημάτων με χρήση OAuth2 token, η διαδικασία απεικονίζεται στην Εικόνα 3.20 και παρακάτω αναλύονται οι ενέργειες που εκτελούνται με τη σειρά που αυτές λαμβάνουν χώρα.



Εικόνα 3.12: Λειτουργία Διακομιστή PEP με την χρήση του OAuth2 token.

1. Ο Διακομιστής PEP δέχεται ένα HTTP αίτημα που περιλαμβάνει ένα **OAuth2 token**.
2. Ελέγχει αν το διακριτικό που φέρει στην κεφαλίδα του το αίτημα που δέχτηκε, αποτελεί ένα έγκυρο **OAuth2 token**. Ο έλεγχος αυτός συμβαίνει μέσω REST επικοινωνίας με την υπηρεσία ταυτοποίησης και εξουσιοδότησης χρηστών.
3. Εφόσον η εγκυρότητα του **OAuth2 token** επιβεβαιωθεί, η υπηρεσία Ταυτοποίησης και Εξουσιοδότησης, επιστρέφει στον Διακομιστή PEP, τις

πληροφορίες που σχετίζονται με την ταυτότητα και τους ρόλους του χρήστη που αφορά το OAuth2 token. Σε περίπτωση που το OAuth2 token δεν είναι έγκυρο η διαδικασία σταματάει εδώ.

4. Σειρά έχει να αξιολογηθεί αν το αίτημα το οποίο έκανε ο χρήστης στο βήμα 1), εγκρίνεται βάσει των ρόλων που διαθέτει. Ο Διακομιστής PEP μέσω REST επικοινωνίας, προωθεί τις πληροφορίες που έλαβε στο βήμα 3) και το αίτημα που έλαβε στο βήμα 1) στην υπηρεσία Λήψης Απόφασης Εξουσιοδότησης προκειμένου η υπηρεσία να αποφασίσει αν το αίτημα εγκρίνεται ή όχι.
5. Η υπηρεσία αξιολογεί αν το αίτημα του χρήστη εγκρίνεται ή απορρίπτεται.
6. Ο Διακομιστής PEP ενημερώνεται για το αποτέλεσμα της αξιολόγησης.
7. Εφόσον το αίτημα εγκρίνεται, ο Διακομιστής PEP προωθεί το αίτημα στην προστατευόμενη υπηρεσία για την οποία μεσολαβεί.
8. Η υπηρεσία επιστρέφει στον Διακομιστή PEP την απάντηση του αιτήματος.
9. Ο Διακομιστής PEP προωθεί την απάντηση της υπηρεσίας στον χρήστη.

3.10 Υπηρεσία Λογική Εφαρμογής – Σύστημα Διεπαφής Χρηστών (Application Logic- Web Application)

Η Λογική Εφαρμογής αποτελεί την καρδιά του συστήματος καθώς περιλαμβάνει τον κώδικα για την ενορχήστρωση των επιμέρους υπηρεσιών, ώστε το σύστημα να υλοποιεί την λειτουργικότητα που ορίζεται στις προδιαγραφές που έχουν οριστεί. Το σύστημα διεπαφής χρηστών (Web Application) θεωρείται τμήμα της λογικής εφαρμογής καθώς περιλαμβάνει τον απαραίτητο κώδικα για την υλοποίηση των γραφικών διεπαφών (για όλους τους διαφορετικούς τύπους χρηστών) του συστήματος. Τα αιτήματα των χρηστών του συστήματος προκύπτουν από το σύστημα διεπαφής χρηστών και προωθούνται στην λογική εφαρμογής για την κατάλληλη δρομολόγηση τους. Ακολουθούν δύο σχετικά παραδείγματα της λειτουργίας τους:

- Ένας χρήστης μέσω του συστήματος διεπαφής χρηστών κάνει αίτημα εισόδου στο σύστημα (“Log in” με την χρήση Username και Password). Το αίτημα δρομολογείται στην υπηρεσία «Λογική εφαρμογής», η οποία με την σειρά της το δρομολογεί στην υπηρεσία ταυτοποίησης και εξουσιοδότησης του συστήματος (Μέσω του OAuth2 API που παρέχει η υπηρεσία ταυτοποίησης και εξουσιοδότησης για την επαλήθευση στοιχείων χρήστη). Εφόσον τα στοιχεία του επαληθευτούν, ο χρήστης θα εισέλθει στο σύστημα. Κατόπιν, η υπηρεσία «Λογική εφαρμογής» δημιουργεί μία συνεδρία σύνδεσης για το χρήστη, η οποία περιλαμβάνει τα στοιχεία εισόδου του και το OAuth2 token που του αντιστοιχεί.
- Ένας τελικός χρήστης μέσω του συστήματος διεπαφής χρηστών, ζητάει πρόσβαση σε μία εφαρμογή της συνδρομής του. Το HTTP αίτημα του, δρομολογείται στην υπηρεσία «Λογική Εφαρμογής». Η «Λογική εφαρμογής» δρομολογεί το αίτημα στην υπηρεσία Mashup (Υπηρεσία που εκτελεί τις

εφαρμογές του συστήματος, βλ. 3.6.7). Η «λογική εφαρμογής» προωθεί την απάντηση στο χρήστη και εκείνος βλέπει στην οθόνη του την πληροφορία της εφαρμογής.

Ακολουθώντας αυτή τη λογική, όλα τα αιτήματα που προκύπτουν από τις γραφικές διεπαφές του συστήματος διεπαφής χρηστών δρομολογούνται μέσω της υπηρεσίας «λογική εφαρμογής» στις υπηρεσίες που προορίζονται.

Κεφάλαιο 4

Υλοποίηση Συστήματος

4.1 Υλοποίηση Υπηρεσιών

Είδαμε τον σχεδιασμό του συστήματος και παρακάτω παρατίθεται η υλοποίηση των υπηρεσιών του συστήματος που αναπτύχθηκαν, ενώ γίνεται και μια αναλυτική περιγραφή της χρήσης και της λειτουργίας για την κάθε υπηρεσία ξεχωριστά.

4.1.1 Υπηρεσία Ταυτοποίησης και Εξουσιοδότησης Χρηστών - Keyrock Identity Management (Keyrock IdM)

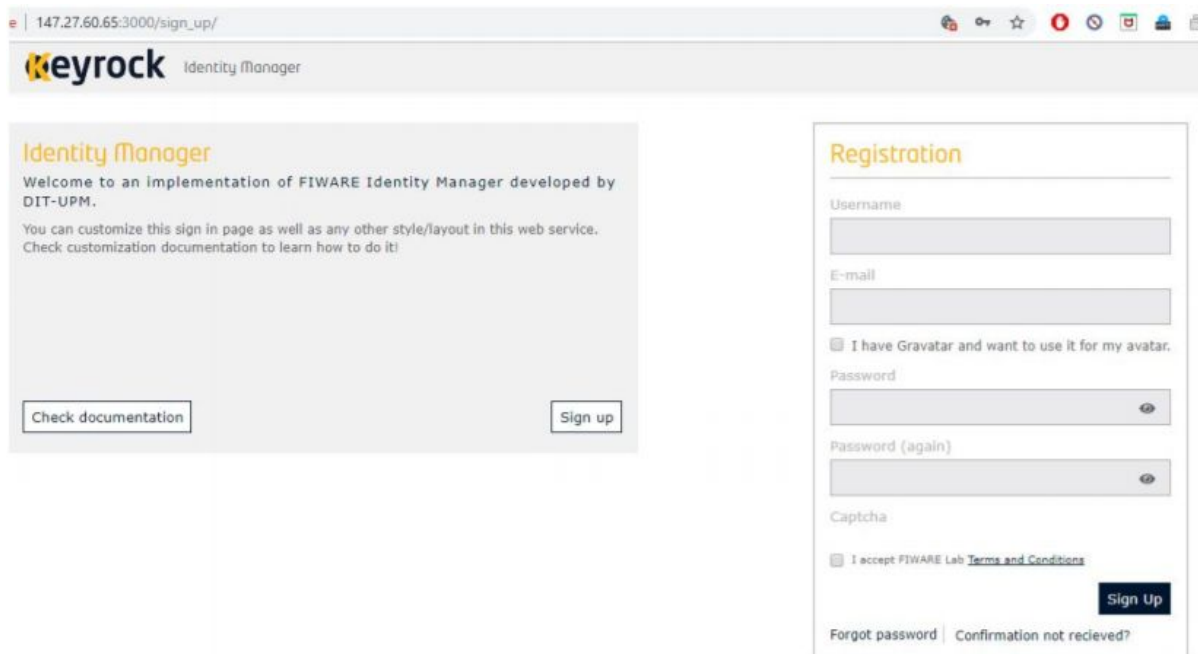
Η υπηρεσία αυτή έχει αναπτυχθεί και παρέχεται από το FIWARE. Η λειτουργία της βασίζεται στη χρήση του πρωτοκόλλου εξουσιοδότησης OAuth2, χάρη στο οποίο γίνεται η ταυτοποίηση των χρηστών και η παροχή εξουσιοδότησης (authorization) πρόσβασης σε υπηρεσίες. Παρακάτω δίνεται μια περιγραφή του τρόπου λειτουργίας της υπηρεσίας, ο οποίος περιλαμβάνει τα εξής τρία στάδια:

1) την εγγραφή του συστήματος από κάποιο διαχειριστή (administrator) στην υπηρεσία Keyrock IdM,

2) την εγγραφή ενός νέου χρήστη στο σύστημα ούτως ώστε να έχει πρόσβαση σε αυτό.
3) τη διαδικασία ταυτοποίησης ενός διαχειριστή ή χρήστη μέσω της εν λόγω υπηρεσίας.
Στη συνέχεια, αναλύονται τα τρία παραπάνω στάδια:

1) Αρχικά, πραγματοποιείται η διαδικασία εγγραφής του συστήματος στην υπηρεσία Keyrock IdM από το διαχειριστή. Συγκεκριμένα, ο πίνακας ελέγχου της υπηρεσίας Keyrock IdM παρέχει ένα γραφικό περιβάλλον, στο οποίο μπορεί εύκολα να επιτευχθεί η εγγραφή, με αποτέλεσμα την αυτόματη παραγωγή δύο μοναδικών αναγνωριστικών που σχετίζονται με την ταυτοποίηση (`client_id` και `client_secret`). Τα δύο αυτά αναγνωριστικά συνδυάζονται με τη μέθοδο κωδικοποίησης `base64` (`client_id:client_secret`) και οδηγούν στην παραγωγή ενός επιπλέον αναγνωριστικού που ονομάζεται “`Authorization_Basic`”. Αυτό το αναγνωριστικό συνιστά για την υπηρεσία Keyrock IdM την ταυτότητα του συστήματος. Επομένως, κάθε αίτημα σύνδεσης στο σύστημα πρέπει οπωσδήποτε να περιλαμβάνει στην κεφαλίδα του (`request header`) το αναγνωριστικό “`Authorization_Basic`”.

2) Στη συνέχεια, κάποιος χρήστης που επιθυμεί να αποκτήσει πρόσβαση στο σύστημα, θα πρέπει πρωτίστως να δημιουργήσει ένα νέο λογαριασμό στην υπηρεσία Keyrock IdM. Το γραφικό περιβάλλον, όπου γίνεται η εγγραφή (`sign up`) και το οποίο παρέχεται από τον πίνακα ελέγχου της υπηρεσίας Keyrock IdM, φαίνεται στην εικόνα 4.1:

The image shows a web browser window with the URL '147.27.60.65:3000/sign_up/'. The page header displays the 'keyrock Identity Manager' logo. The main content area is split into two panels. The left panel, titled 'Identity Manager', contains a welcome message: 'Welcome to an implementation of FIWARE Identity Manager developed by DIT-UPM. You can customize this sign in page as well as any other style/layout in this web service. Check customization documentation to learn how to do it!'. It features two buttons: 'Check documentation' and 'Sign up'. The right panel, titled 'Registration', contains a form with the following fields: 'Username' (text input), 'E-mail' (text input), a checkbox labeled 'I have Gravatar and want to use it for my avatar.', 'Password' (password input with an eye icon), 'Password (again)' (password input with an eye icon), and a 'Captcha' section. Below the captcha is a checkbox labeled 'I accept FIWARE Lab Terms and Conditions'. At the bottom of the form is a 'Sign Up' button. At the very bottom of the registration panel, there are links for 'Forgot password' and 'Confirmation not recieved?'. The browser's address bar and various icons are visible at the top of the window.

Εικόνα 4.1: Στιγμιότυπο από το γραφικό περιβάλλον όπου γίνεται η εγγραφή (`sign up`) ενός χρήστη στο σύστημα.

3) Τέλος, οποιοσδήποτε χρήστης επιθυμεί να εισέλθει στο σύστημα, πρέπει να περάσει επιτυχώς από τη διαδικασία ταυτοποίησης. Αυτή πραγματοποιείται κατά το αίτημα εισόδου του χρήστη στο σύστημα, με το οποίο δίνει τα απαραίτητα στοιχεία του λογαριασμού του (e-mail και password). Ειδικότερα, η Λογική Εφαρμογής (Application Logic) πραγματοποιεί ένα αίτημα REST στην υπηρεσία Keyrock IdM. Η κεφαλίδα του αιτήματος περιέχει το διακριτικό “Authorization_Basic” και το σώμα του (request body) τα στοιχεία εισόδου του χρήστη, προκειμένου να γίνει η ταυτοποίηση. Μόλις αυτή ολοκληρωθεί επιτυχώς, επιστρέφεται ένα OAuth2 token από την υπηρεσία Keyrock IdM στη Λογική Εφαρμογής και η είσοδος του χρήστη πραγματοποιείται με επιτυχία. Αμέσως μετά, δημιουργείται στον εξυπηρετητή του χρήστη μια συνεδρία (session) που αποθηκεύει το OAuth2 token του χρήστη κι έτσι ο χρήστης μπορεί να παραμείνει και να περιηγηθεί σε κάθε σελίδα της Διαδικτυακής Εφαρμογής του Συστήματος για όλο το διάστημα κατά το οποίο η συνεδρία παραμένει ενεργή. Στην παρούσα εργασία, η λειτουργία αυτή υλοποιήθηκε με τη χρήση της μεταβλητής \$_SESSION[OAuth2_token] (στη γλώσσα προγραμματισμού php), η οποία αποθήκευσε την τιμή του token. Στον πίνακα που ακολουθεί (REST table) γίνεται μια περιγραφή της HTTP μεθόδου της υπηρεσίας Keyrock IdM που αναλύθηκε παραπάνω:

Μέθοδος	URL Μεθόδου	Header	Σώμα αιτήματος	Περιγραφή μεθόδου
POST	/oauth2/token	Authorization: base64 (client_id: client_secret)	{ &username = "username" &password = "user_password" }	Δίνεται ένα έγκυρο username και password και επιστρέφεται ένα OAuth2 token.

Πίνακας 4.1: HTTP μέθοδος της υπηρεσίας Keyrock IdM.

4.1.2 Υπηρεσία Λήψης Απόφασης Εξουσιοδότησης (Authorization PDP) – AuthZForce

Το AuthZForce παρέχεται από το FIWARE και αποτελεί την υπηρεσία λήψης απόφασης εξουσιοδότησης του συστήματος (Authorization Policy Decision Point). Οι προδιαγραφές λειτουργίας του είναι εκείνες που αναλύθηκαν στην ενότητα 3.8.

Η υπηρεσία παίρνει αποφάσεις έγκρισης ή απόρριψης αιτημάτων πρόσβασης από χρήστες σε πόρους του συστήματος. Τα αποτελέσματα των αποφάσεων (έγκριση ή απόρριψη) προκύπτουν από τους αποθηκευμένους κανόνες πρόσβασης της υπηρεσίας, οι οποίοι ακολουθούν το πρότυπο XACML. Ένας κανόνας πρόσβασης

σχετίζεται άρρηκτα με ένα ρόλο εντός της υπηρεσίας Keyrock, καθώς περιγράφει τις προϋποθέσεις (μέθοδο αιτήματος και URL) που πρέπει να ακολουθεί το αίτημα πρόσβασης ενός χρήστη -με αυτό το ρόλο- προκειμένου εγκριθεί. Το AuthZForce, διαθέτει μία RESTful διεπαφή μέσω της οποίας:

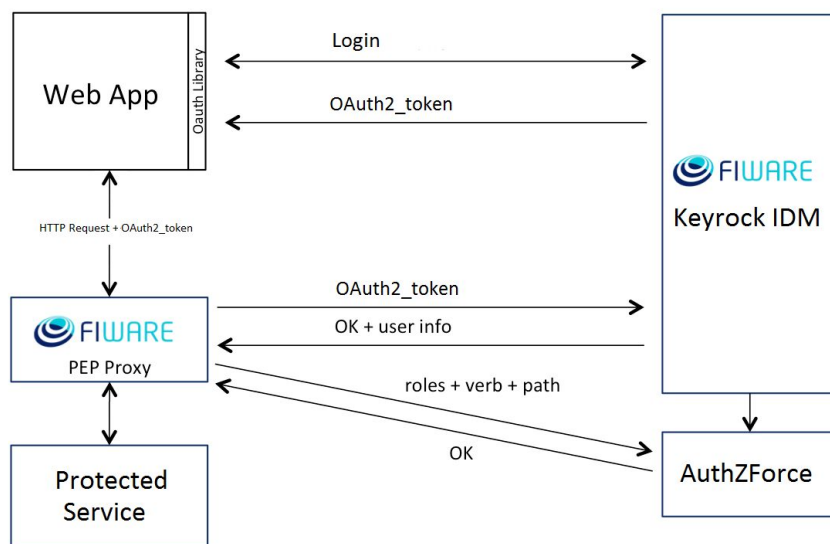
- Με την δημιουργία ενός νέου ρόλου στην υπηρεσία Keyrock δρομολογείται –αυτόματα- το REST αίτημα, από την υπηρεσία Keyrock στην υπηρεσία AuthZForce το οποίο δημιουργεί τον νέο κανόνα πρόσβασης που συνιστά ο ρόλος.
- Προωθούνται από ένα διακομιστή PEP στην υπηρεσία AuthZForce, τα REST αιτήματα πρόσβασης χρηστών σε πόρους του συστήματος, προκειμένου να αξιολογηθούν προς έγκριση ή απόρριψη (βλ. παρακάτω).

4.1.3 Διακομιστής Μεσολάβησης Επιβολή Πολιτικής - Pep Proxy Wilma

Πρόκειται για μία υπηρεσία που παρέχεται από το FIWARE ανεπτυγμένη ώστε να συνεργάζεται με τις υπηρεσίες Keyrock IDM και AuthZforce παρέχοντας ασφάλεια στις υπόλοιπες υπηρεσίες του συστήματος. Το Pep Proxy-Wilma, λειτουργεί σύμφωνα με τις προδιαγραφές του διακομιστή μεσολάβησης επιβολής πολιτικής (Policy Enforcement Point Proxy), όπως αυτές διατυπώθηκαν στην ενότητα 3.9. Η υλοποίηση του συστήματος περιλαμβάνει 7 διαφορετικά PEP Proxy-Wilma με το κάθε ένα να «προστατεύει» αντίστοιχα μια υπηρεσία.

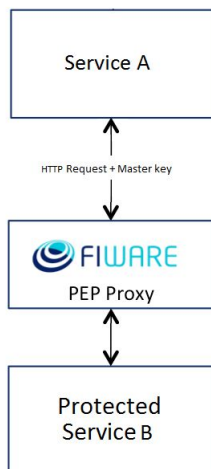
Τα PEP Proxy-Wilma που μεσολαβούν μεταξύ υπηρεσιών και αιτημάτων από χρήστες (User generated) του συστήματος (Web application ->App Logic

-> Web Things Model Service, Web application ->App Logic -> Mashup Service) απαιτούν το OAuth2 token του χρήστη στην κεφαλίδα του αιτήματος. Με αυτό τον τρόπο το PEP Proxy-Wilma μέσω επικοινωνίας με τον Keyrock IDM, επιβεβαιώνει την ταυτότητα και τους ρόλους του χρήστη στον οποίο ανήκει το OAuth2 token. Στην συνέχεια επικοινωνεί με το AuthZforce και ενημερώνεται για το αν το αίτημα του χρήστη πρέπει να προωθηθεί στην υπηρεσία για την οποία μεσολαβεί ή να απορριφθεί. Η διαδικασία αυτή αποτυπώνεται στην εικόνα 4.1.



Εικόνα 4.1: PEP Proxy-Wilma με την χρήση OAuth2 token

Στο σενάριο επικοινωνίας μεταξύ υπηρεσιών του συστήματος τα PEP Proxy-Wilma απαιτούν στην κεφαλίδα των αιτημάτων που λαμβάνουν τον μυστικό κωδικό Master key (κάθε PEP Proxy-Wilma έχει τον δικό του μυστικό κωδικό Master key). Εφόσον ένα αίτημα περιλαμβάνει τον σωστό κωδικό Master key, το PEP Proxy-Wilma προωθεί την επικοινωνία στην υπηρεσία για την οποία μεσολαβεί. Η διαδικασία αυτή απεικονίζεται στην εικόνα 4.2.



Εικόνα 4.2: PEP Proxy-Wilma με την χρήση κωδικού Master Key.

4.1.3 Υπηρεσία Διαχείρισης Συμβάντων και Συνδρομών

Ο τρόπος λειτουργίας της υπηρεσίας Orion Context Broker (η οποία επίσης είναι παροχή του FIWARE) βασίζεται στα πρότυπα του μοντέλου πληροφορίας NGSI-2 και στόχος της υπηρεσίας είναι η διαχείριση δεδομένων πλαισίου. Αυτή επιτυγχάνεται μέσω της RESTful διεπαφής του Orion Context Broker. Παρακάτω γίνεται μια περιγραφή της χρήσης και των λειτουργιών της υπηρεσίας στην υλοποίηση του συστήματος.

Orion Context Broker - Δημιουργία και ενημέρωση οντότητας

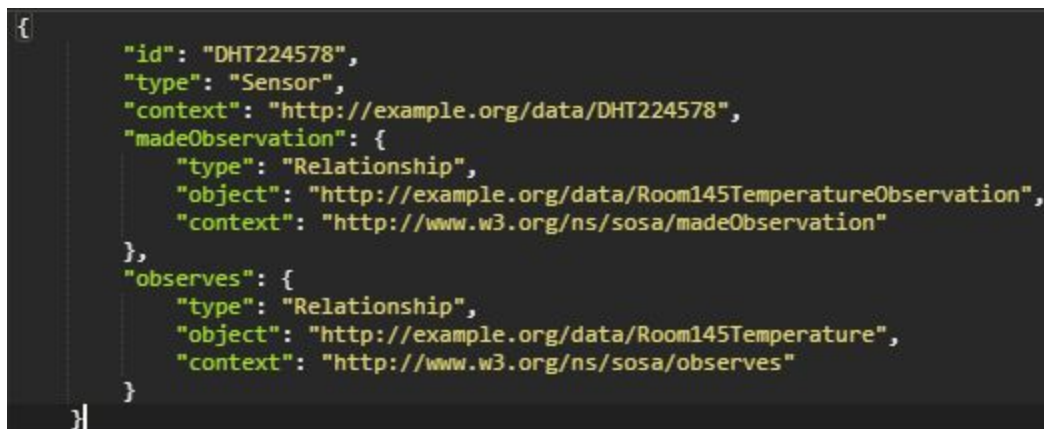
Στην παρούσα εργασία έχει γίνει η σχεδιαστική επιλογή να διατηρούνται στον Orion Context Broker οι οντότητες (entities) της μορφής NGSI-LD οι οποίες περιγράφουν:

- Κάθε διαφορετικό αισθητήρα (Sensor).
- Την τελευταία μέτρηση (Observation) που γίνεται από αισθητήρα.
- Κάθε διαφορετικό σπίτι (Home) που περιέχει αισθητήρες/συσσκευές.
- Κάθε διαφορετική εφαρμογή που έχει δημιουργηθεί στο σύστημα από κάποιο προγραμματιστή εφαρμογών.
- Κάθε διαφορετική εκτέλεση ενέργειας (Actuation) που έχει πραγματοποιηθεί από κάποια συσκευή.
- Κάθε διαφορετική μετρούμενη ιδιότητα (Observable Property) που έχει αποθηκευτεί στην πλατφόρμα.
- Κάθε διαφορετική χωρική οντότητα (Spatial Thing), δηλαδή τοποθεσία - για παράδειγμα, η πόλη του Ρεθύμνου - για την οποία έχουν καταγραφεί πληροφορίες στο σύστημα.

Όλες οι οντότητες που περιγράφονται παρακάτω παρουσιάζονται σε μορφή NGSI-LD, δηλαδή JSON-LD. Όπως αναφέρθηκε και το κεφάλαιο 3.4, παρόλο που η μορφή τους είναι NGSI-LD η υπηρεσία Orion Context Broker για να λειτουργήσει τις μεταφράζει σαν NGSI2. Με την NGSI-LD (JSON-LD) μορφοποίηση όμως επιτυγχάνεται μέσω του attribute “context” η αναφορά στην οντολογία και μέσω των συνδέσμων που έχουν επίσης τα attribute “object” επιτυγχάνεται η σύνδεση με τις υπόλοιπες οντότητες.

Προκειμένου να δημιουργηθεί μια νέα NGSI οντότητα στην υπηρεσία Orion Context Broker, πραγματοποιείται ένα HTTP αίτημα της μεθόδου POST, ενώ για την ενημέρωση μιας οντότητας χρησιμοποιείται η μέθοδος PUT. Ακολουθεί ένα παράδειγμα της οντότητας **“Αισθητήρα” (Sensor)**. Η οντότητα αυτή γενικά περιέχει πληροφορία για

το μοντέλο ενός Πράγματος, δηλαδή ενός αισθητήρα ή γενικότερα μιας συσκευής. Στο συγκεκριμένο παράδειγμα, πρόκειται για το μοντέλο ενός υποθετικού αισθητήρα που μετράει την υγρασία στην πόλη των Χανίων.



```
{
  "id": "DHT224578",
  "type": "Sensor",
  "context": "http://example.org/data/DHT224578",
  "madeObservation": {
    "type": "Relationship",
    "object": "http://example.org/data/Room145TemperatureObservation",
    "context": "http://www.w3.org/ns/sosa/madeObservation"
  },
  "observes": {
    "type": "Relationship",
    "object": "http://example.org/data/Room145Temperature",
    "context": "http://www.w3.org/ns/sosa/observes"
  }
}
```

Εικόνα 4.2: Αναπαράσταση JSON-LD της οντότητας “DHT224578” στον Orion Context Broker.

Τα χαρακτηριστικά που φαίνονται στην αναπαράσταση JSON-LD της εικόνας 4.2 περιγράφονται παρακάτω:

- **“id”**: Πρόκειται για το μοναδικό αναγνωριστικό του συγκεκριμένου μοντέλου συσκευών.
- **“type”**: Δηλώνει τον τύπο της οντότητας, ο οποίος στην περίπτωση αυτή είναι “Sensor” επομένως πρόκειται για μοντέλο αισθητήρα (Πράγματος).
- **“context”**: Αυτό το χαρακτηριστικό αποτελεί τον υπερσύνδεσμο (link) με τον οποίο αυτή η οντότητα συνδέεται με άλλες οντότητες του συστήματος και την καθιστά JSON-LD. Οδηγεί στην σημασιολογική πληροφορία για το συγκεκριμένο μοντέλο αισθητήρα, δηλαδή το μοντέλο έχει τη σημασία που συνιστά ο διαδικτυακός σύνδεσμος “http://example.org/data/DHT224578” (που είναι η τιμή του χαρακτηριστικού context). Χάρη στον παραπάνω υπερσύνδεσμο, με άλλα λόγια, μπορούμε να πάρουμε επιπλέον πληροφορίες για το παρόν μοντέλο.
- **“observes”**: Πρόκειται για ένα χαρακτηριστικό που δηλώνει ότι ο συγκεκριμένος τύπος αισθητήρα μετράει μια συγκεκριμένη μετρήσιμη ιδιότητα, η οποία στην περίπτωση αυτή είναι η υγρασία στην πόλη των Χανίων. Αποτελεί ουσιαστικά την σύνδεση του συγκεκριμένου αισθητήρα με το συγκεκριμένο μετρήσιμο μέγεθος που αυτός μετράει. Το “forProperty” συνιστά ένα αντικείμενο JSON, το οποίο θα μετατραπεί στη συνέχεια (στην Υπηρεσία Οντολογίας) σε μια σημασιολογική τριπλέτα και θα αποθηκευτεί στην Οντολογία. Όπως παρατηρούμε, στο αντικείμενο JSON “observes” (δηλαδή στα χαρακτηριστικά που περιλαμβάνονται στα άγκιστρα) αποτελείται από το:

- Το **“type”**: έχει προφανώς την τιμή “Relationship”, όπως περιγράφηκε στο κεφάλαιο 3.4, καθώς το συγκεκριμένο attribute αποτελεί *Object Property*.
- Το εσωτερικό **“context”**: διαφέρει από αυτό που αναφέρθηκε παραπάνω καθώς περιέχει τον υπεрсύνδεσμο στην οντολογία SSN, δηλαδή την επίσημη περιγραφή του Object Property αυτού. Αν ανατρέξουμε στην οντολογία SSN, από την οποία προέρχεται το συγκεκριμένο *“context”* ([“http://www.w3.org/ns/ssn/observes”](http://www.w3.org/ns/ssn/observes)), θα δούμε ότι πρόκειται για ένα *Object Property* της Οντολογίας που υποδηλώνει ότι ένας αισθητήρας παίρνει μετρήσεις ενός συγκεκριμένου μετρήσιμου μεγέθους.
- Το **“object”**: μας δίνει την πληροφορία ότι ο συγκεκριμένος αισθητήρας μετράει την “θερμοκρασία στο σπίτι Room145”. Αυτό το κάνει περιέχοντας ένα υπεрсύνδεσμο (link) στην οντότητα [“http://example.org/data/Room145Temperature”](http://example.org/data/Room145Temperature), η οποία υποδηλώνει την θερμοκρασία στο σπίτι αυτό.
- **“madeObservation”**: Όπως και προηγουμένως με το “observes”, πρόκειται για ένα χαρακτηριστικό που δηλώνει τη σχέση ανάμεσα σε έναν αισθητήρα και σε μια οντότητα μέτρησης (παρατήρησης - Observation) και παρέχει τον σύνδεσμο (link) στην οντότητα μέτρησης (παρατήρησης - Observation) *“Room145TemperatureObservation”*, δηλαδή στο μέγεθος που περιγράφει την μέτρηση υγρασίας στα Χανιά. Αυτή η διασύνδεση γίνεται με την τιμή του attribute *“object”*, που επίσης είναι αποθηκευμένη στον Orion ως *“Room145TemperatureObservation”*, άρα έχει επιτευχθεί η σύνδεση ανάμεσα στις δύο οντότητες.

Στη συνέχεια, παρατίθενται στιγμιότυπα με παραδείγματα από τους υπόλοιπους τύπους οντοτήτων (μέτρηση, σπίτι κλπ) που φιλοξενούνται στον Orion Context Broker:

- Οντότητα σπιτιού (home):

```
{
  "id": "Room145",
  "type": "Home",
  "context": "http://example.org/data/Room145",
  "hasProperty": {
    "type": "Relationship",
    "object": "http://example.org/data/Room145Temperature",
    "context": "http://www.w3.org/ns/ssn/hasProperty"
  },
  "hasTemperatureState": {
    "type": "Relationship",
    "object": "http://example.org/data/BelowRoomTemperature",
    "context": "http://sensormeasurement.appspot.com/ont/m3/home#hasTemperatureState"
  },
  "isFeatureOfInterestOf": {
    "type": "Relationship",
    "object": "http://example.org/data/Room145TemperatureObservation",
    "context": "http://www.w3.org/ns/sosa/isFeatureOfInterestOf"
  }
}
```

Εικόνα 4.3: Στιγμιότυπο οντότητας σπιτιού (Room145) όπου φαίνονται σημασιολογικές πληροφορίες σχετικά με τη θερμοκρασία του σπιτιού.

Τα χαρακτηριστικά που φαίνονται στην αναπαράσταση JSON-LD της εικόνας 4.3 περιγράφονται παρακάτω:

- **“id”**: Πρόκειται για το μοναδικό αναγνωριστικό του συγκεκριμένου σπιτιού.
 - **“type”**: έχει τιμή “Home”.
 - **“hasTemperatureState”**: δείχνει ποια είναι η κατάσταση αυτή την στιγμή μέσα στο σπίτι για κάθε μετρήσιμο μέγεθος από τους αισθητήρες(χαμηλή θερμοκρασία).
 - **“hasProperty”**: περιέχει συνδεσμο στις ιδιότητες που μετράνε οι αισθητήρες του σπιτιού (θερμοκρασία σπιτιού).
 - **“isFeatureOfInterestOf”**: περιέχει συνδεσμο στην οντότητα Παρατηρήσεων (Μέτρησης) που αφορά το συγκεκριμένο σπίτι και περιέχει την τελευταία μέτρηση (θερμοκρασίας) για το σπίτι αυτό..
-
- **Οντότητα μετρήσιμης ιδιότητας (observable property):**

```
{
  "id": "Room145Temperature",
  "type": "ObservableProperty",
  "context": "http://example.org/data/Room145Temperature",
  "isObservedBy": {
    "type": "Relationship",
    "object": "http://example.org/data/DHT224578",
    "context": "http://www.w3.org/ns/sosa/isObservedBy"
  },
  "isPropertyOf": {
    "type": "Relationship",
    "object": "http://example.org/data/Room145",
    "context": "http://www.w3.org/ns/ssn/isPropertyOf"
  }
}
```

Εικόνα 4.4: Στιγμιότυπο της οντότητας μετρήσιμης ιδιότητας (Room145Temperature).

Η οντότητα της μετρήσιμης ιδιότητας είναι η οντότητα που περιγράφει το μέγεθος το οποίο ανήκει σε πόλη ή σπίτι και το μετράει ο αισθητήρας (πχ θερμοκρασία σε πόλη, υγρασία μέσα σε σπίτι). Περιγράφεται από τα εξής πεδία:

- **“id”**: Πρόκειται για το μοναδικό αναγνωριστικό της συγκεκριμένης ιδιότητας.
 - **“type”**: έχει τιμή “ObservableProperty”.
 - **“context”**: περιέχει το σύνδεσμο (link) της οντότητας με τον οποίο συνδέεται σε άλλες οντότητες.
 - **“isObservedBy”**: περιέχει τον σύνδεσμο που οδηγεί στον αισθητήρα ο οποίος παίρνει μετρήσεις του συγκεκριμένου μεγέθους.
 - **“isPropertyOf”**: περιέχει τον σύνδεσμο στην οντότητα σπιτιού ή πόλης στην οποία ανήκει το συγκεκριμένο μετρήσιμο μέγεθος (ιδιότητα).
-
- **Οντότητα μέτρησης (observation):**


```

{id": "Room145TemperatureObservation",
"type": "Observation",
"context": "http://example.org/data/Room145TemperatureObservation",
"deducesTemperature": {
  "type": "Relationship",
  "object": "http://example.org/data/BelowRoomTemperature",
  "context": "http://www.w3.org/ns/sosa/deducesTemperature"
},
"hasFeatureOfInterest": {
  "type": "Relationship",
  "object": "http://example.org/data/Room145",
  "context": "http://www.w3.org/ns/sosa/hasFeatureOfInterest"
},
"hasSimpleResult": 15,
"madeBySensor": {
  "type": "Relationship",
  "object": "http://example.org/data/DHT224578",
  "context": "http://www.w3.org/ns/sosa/madeBySensor"
},
"observedProperty": {
  "type": "Relationship",
  "object": "http://example.org/data/Room145Temperature",
  "context": "http://www.w3.org/ns/sosa/observedProperty"
},
"resultTime": {
  "type": "DateTime",
  "value": "2019-06-03T11:12:00Z"
},

```

Εικόνα 4.5: Στιγμιότυπο οντότητας μέτρησης (Room145temperatureObservation).

Η οντότητα των Παρατηρήσεων αποτελεί πολύ σημαντικό στοιχείο της οντολογίας καθώς είναι η οντότητα που περιγράφει τα αποτελέσματα των μετρήσεων που εκτελούν οι αισθητήρες. Περιγράφεται από τα εξής πεδία:

- **“id”**: Πρόκειται για το μοναδικό αναγνωριστικό του συγκεκριμένου σπιτιού, και μέσω αυτού του πεδίου καταλαβαίνουμε για ποια οντότητα έχει παρθεί η μέτρηση, έχουμε κάνει την παραδοχή δηλαδή, ότι μέσα από το “id” υποδηλώνεται το σπίτι ή η πόλη της οποίας μία μετρήσιμη ιδιότητα(observable property) έχουμε πάρει μία τιμή.
- **“type”**: έχει τιμή “Observation”.
- **“context”**: περιέχει το σύνδεσμο (link) της οντότητας με τον οποίο συνδέεται σε άλλες οντότητες.

- **“deducesTemperature”**: περιέχει την παρούσα κατάσταση στο σπίτι ή πόλη σύμφωνα με την τελευταία μέτρηση. Στο κεφάλαιο 3.5 περιγράφεται πως παράγεται αυτή η πληροφορία.
 - **“hasFeatureOfInterest”**: περιέχει τον σύνδεσμο που οδηγεί στην οντότητα σπιτιού ή πόλης στην οποία αναφέρεται η μέτρηση.
 - **“hasSimpleResult”**: έχει την τελευταία τιμή που μέτρησε ο αισθητήρας.
 - **“madeBySensor”**: περιέχει τον σύνδεσμο που οδηγεί στον αισθητήρα που παίρνει της μετρήσεις.
 - **“observedProperty”**: περιέχει τον σύνδεσμο που οδηγεί στο μέγεθος (Οντότητα Μετρήσιμη Ιδιότητα) το οποίο μετράει ο αισθητήρας.
 - **“resultTime”**: έχει τον χρόνο που πάρθηκε η τελευταία μέτρηση του αισθητήρα.
- **Οντότητα (εκτέλεσης) ενέργειας - αυτοματισμός:**

```
{
  "id": "Room145SwitchOffHeat",
  "type": "Actuation",
  "context": "http://example.org/data/Room145SwitchOffHeat",
  "actsOnProperty": {
    "type": "Relationship",
    "object": "http://example.org/data/Room145Temperature",
    "context": "http://www.w3.org/ns/sosa/actsOnProperty"
  },
  "actuationEnabled": 0,
  "hasFeatureOfInterest": {
    "type": "Relationship",
    "object": "http://example.org/data/Room145",
    "context": "http://www.w3.org/ns/sosa/hasFeatureOfInterest"
  }
}
```

Εικόνα 4.6: Στιγμιότυπο οντότητας εκτέλεσης ενέργειας (Appartment134SwitchOnAC).

Τα χαρακτηριστικά που φαίνονται στην αναπαράσταση JSON-LD της εικόνας 4.6 περιγράφονται παρακάτω:

- **“id”**: Πρόκειται για το μοναδικό αναγνωριστικό της συγκεκριμένης ενέργειας - αυτοματισμού.
- **“type”**: έχει τιμή “Actuation”.
- **“context”**: περιέχει το σύνδεσμο (link) της οντότητας με τον οποίο συνδέεται σε άλλες οντότητες.
- **“actsOnProperty”**: περιέχει τον σύνδεσμο στην μετρήσιμη ιδιότητα την οποία επηρεάζει (θερμοκρασία).

- **“hasFeatureOfInterest”**: περιέχει τον σύνδεσμο που οδηγεί στην οντότητα σπιτιού στην οποία είναι εγκατεστημένος ο αυτοματισμός.
- **“actuationEnabled”**: έχει την τιμή “0” αν δεν λειτουργεί αυτή την στιγμή ο αυτοματισμός και “1” αν λειτουργεί.

- **Οντότητα εφαρμογής (application):**

```
{  
  "id": "room145live",  
  "type": "application",  
  "scope": {  
    "value": "home",  
    "type": "Text"  
  }  
},
```

Εικόνα 4.7: Στιγμιότυπο οντότητας εφαρμογής (room145live) που αφορά τις συνθήκες (όπως η θερμοκρασία και η φωτεινότητα) οι οποίες επικρατούν μέσα σε ένα σπίτι.

- **Οντότητα Πόλης(SpatialThing):**

```
{
  "id": "Rethymno",
  "type": "SpatialThing",
  "context": "http://example.org/data/Rethymno",
  "hasCloudCoverage": {
    "type": "Relationship",
    "object": "http://example.org/data/Dark",
    "context": "http://sensormeasurement.appspot.com/ont/m3/tourism#hasCloudCoverage"
  },
  "hasHumidity": {
    "type": "Relationship",
    "object": "http://example.org/data/VeryMoistHumidity",
    "context": "http://sensormeasurement.appspot.com/ont/m3/tourism#hasHumidity"
  },
  "hasPrecipitation": {
    "type": "Relationship",
    "object": "http://example.org/data/ExtremelyHeavyRain",
    "context": "http://sensormeasurement.appspot.com/ont/m3/tourism#hasPrecipitation"
  },
  "hasPressure": {
    "type": "Relationship",
    "object": "http://example.org/data/VeryHighPressure",
    "context": "http://sensormeasurement.appspot.com/ont/m3/tourism#hasPressure"
  },
  "hasTemperature": {
    "type": "Relationship",
    "object": "http://example.org/data/RoomTemperature",
    "context": "http://sensormeasurement.appspot.com/ont/m3/tourism#hasTemperature"
  },
  "hasWeather": {
    "type": "Relationship",
    "object": "http://example.org/data/VeryMoistHumidity",
    "context": "http://sensormeasurement.appspot.com/ont/m3/tourism#hasWeather"
  },
  "hasWindSpeed": {
    "type": "Relationship",
    "object": "http://example.org/data/NotWindy",
    "context": "http://sensormeasurement.appspot.com/ont/m3/tourism#hasWindSpeed"
  },
  "isLocationOf": {
    "type": "Relationship",
    "object": "http://example.org/data/RethymnoTemperatureSensor",
    "context": "http://www.w3.org/2003/01/geo/wgs84_pos#isLocationOf"
  }
}
```

Εικόνα 4.8: Στιγμιότυπο οντότητας χώρου που αφορά την πόλη του Ρεθύμνου.

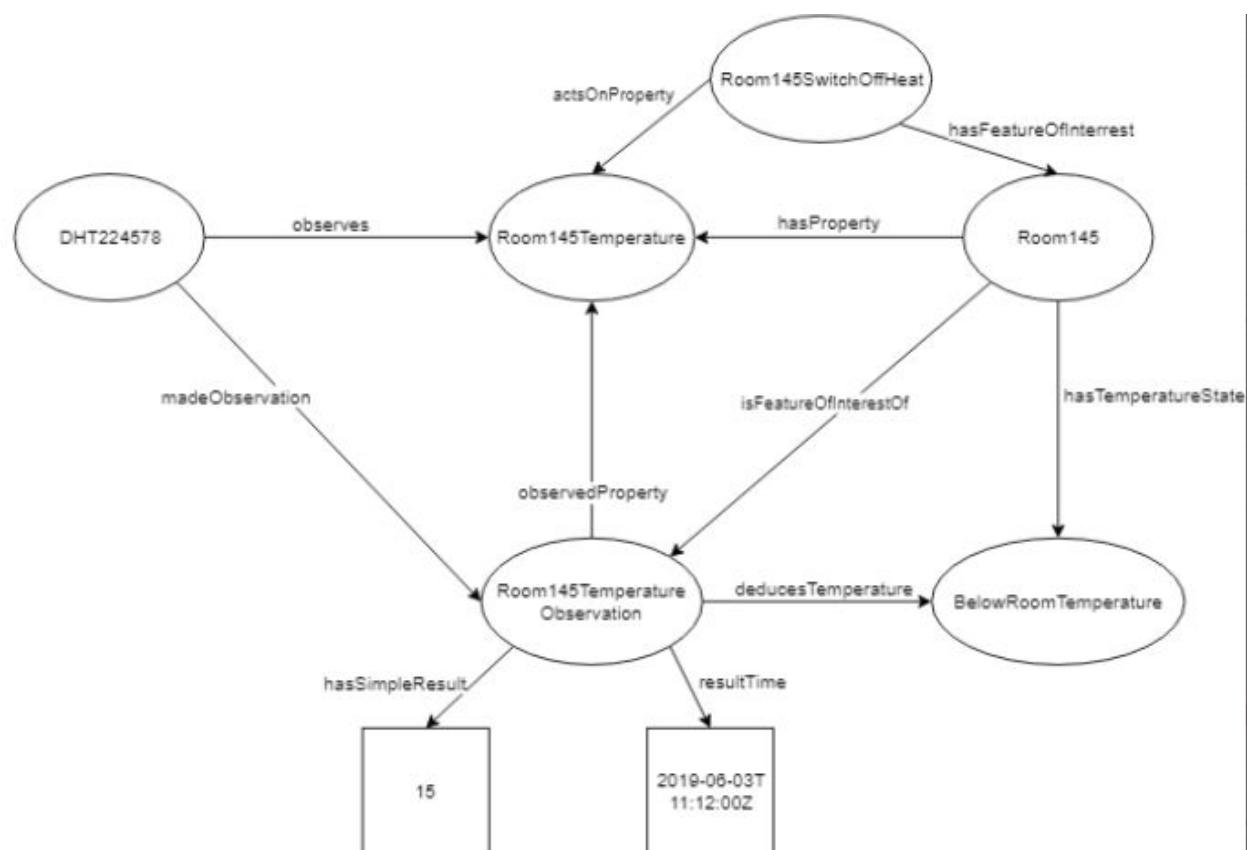
Τα χαρακτηριστικά που φαίνονται στην αναπαράσταση JSON-LD της εικόνας 4.8 περιγράφονται παρακάτω:

- **“id”**: Πρόκειται για το μοναδικό αναγνωριστικό της πόλης.
- **“type”**: έχει τιμή “SpatialThing”, καθώς έτσι είναι ορισμένες οι πόλεις στην οντολογία μας.
- **“context”**: περιέχει το σύνδεσμο (link) της οντότητας με τον οποίο συνδέεται σε άλλες οντότητες.
- **“hasHumidity - hasCloudCoverage - hasTemperature κλπ.”**: κάθε πεδίο από αυτά περιέχει την αντίστοιχη καιρική συνθήκη που επικρατεί στην πόλη όπως έχει μετρήσει ο κάθε αισθητήρας που ανήκει στην πόλη και παρακολουθεί μια μετρήσιμη ιδιότητα.
- **“isLocationOf”**: περιέχει το σύνδεσμο (link) της οντότητας στους αισθητήρες που είναι εγκατεστημένοι στην συγκεκριμένη πόλη.

Όταν στο σύστημα εισέρχονται νέες τιμές που προέρχονται από συσκευές (π.χ. νέες μετρήσεις θερμοκρασίας) ή όταν κάποιος χρήστης κάνει - μέσω της Διαδικτυακής Εφαρμογής - ενημέρωση μιας οντότητας που υπάρχει ήδη στο σύστημα (π.χ. ενός μοντέλου συσκευής), οι αντίστοιχες οντότητες της υπηρεσίας Orion Context Broker υφίστανται τις αντίστοιχες αλλαγές, δηλαδή ενημερώνονται.

Στην περίπτωση των οντοτήτων τύπου “Observation” που αφορούν τις μετρήσεις, κάθε φορά που εισέρχεται στο σύστημα η πληροφορία για μια νέα μέτρηση (που αφορά ήδη υπάρχουσα οντότητα), αλλάζει η τιμή του χαρακτηριστικού “hasSimpleResult”, καθώς και η τιμή του “resultTime” που περιλαμβάνει την πληροφορία για τη χρονική στιγμή και την ημερομηνία που έγινε η μέτρηση. Επομένως, οι τιμές αυτές ενημερώνονται στον Orion Context Broker. Επίσης, όταν αλλάζει η τιμή “hasSimpleResult” ενός **Observation**, απαιτείται ενημέρωση πολλών άλλων πεδίων άλλων οντοτήτων στον Context Broker, η οποίες αλλαγές γίνονται μέσω της Υπηρεσίας Οντολογίας και να αναλυθούν εκτενέστερα παρακάτω.

Στην εικόνα 4.9 μπορούν να φανούν οι σχέσεις μεταξύ των οντοτήτων που περιγράφηκαν παραπάνω σχηματικά. Φαίνονται οι οντότητες και η σύνδεσή μεταξύ τους όπως είναι δηλωμένες στην υπηρεσία Orion Context Broker σε μορφή NGSI-LD. Κάποιες από τα attributes που είναι δηλωμένα δεν φαίνονται στο σχήμα διότι φαίνεται η ανάστροφή τους (inverse Object Property).



Εικόνα 4.9: Σχηματική απεικόνιση της σύνδεσης των οντοτήτων μέσω της μορφοποίησης JSON-LD και της περιγραφής από την οντολογία.

Orion Context Broker - Δημιουργία συνδρομής σε οντότητες

Ο Orion Context Broker έχει - μεταξύ άλλων - μια πολύ σημαντική λειτουργία: μπορεί να δημιουργεί συνδρομή σε συγκεκριμένες (ή συγκεκριμένου τύπου) οντότητες, γεγονός που του δίνει τη δυνατότητα να πυροδοτεί ειδοποιήσεις για οποιαδήποτε αλλαγή συμβεί στα χαρακτηριστικά (attributes) κάποιας οντότητας. Η εκάστοτε ενημέρωση θα αποσταλεί από την υπηρεσία Orion Context Broker σε ένα URI που έχει προκαθοριστεί (από τον συνδρομητή), μέσω ενός REST αιτήματος της μεθόδου POST. Το αίτημα αυτό περιέχει στο σώμα του την πληροφορία της αλλαγής (των τιμών των χαρακτηριστικών), στη μορφή που ορίζει το πρότυπο NGSI-2.

Η δυνατότητα αυτή αξιοποιείται τόσο από την Υπηρεσία Αποθήκευσης των Ιστορικών Δεδομένων όσο και από την Υπηρεσία Οντολογίας (με ξεχωριστή συνδρομή από κάθε υπηρεσία). Αμφότερες οι υπηρεσίες, δρώντας ως συνδρομητές, δέχονται

ειδοποιήσεις σχετικά με τις αλλαγές στις μετρήσεις των αισθητήρων. Η πρώτη υπηρεσία αποθηκεύει τις αλλαγές αυτές στην ιστορική βάση δεδομένων του συστήματος, ενώ η δεύτερη τις αποθηκεύει στην οντολογία του συστήματος (κατά τη διαδικασία αυτή πραγματοποιείται και η διαδικασία του reasoning ώστε να ελεγχθούν οι νέες τιμές π.χ. σχετικά με τα όρια στα οποία κυμαίνονται). Έτσι, με κάθε αλλαγή στις τιμές των μετρήσεων, η ιστορική βάση προσθέτει τις νέες τιμές, διατηρώντας το ιστορικό μετρήσεων χρονικής σειράς, ενώ η οντολογία ενημερώνει τις υπάρχουσες τιμές που έχουν αποθηκευτεί για κάθε μέτρηση (δηλαδή κρατάει μόνο τις τρέχουσες τιμές, όπως συμβαίνει και στον Orion Context Broker).

Για να επιτευχθεί η καθεμία από τις παραπάνω συνδρομές, απαιτείται η αποστολή ενός αιτήματος στην υπηρεσία Orion Context Broker, το οποίο έχει ως αποτέλεσμα τη δημιουργία συνδρομής προς τις οντότητες των μετρήσεων των αισθητήρων. Για παράδειγμα, το αίτημα για να δημιουργηθεί η συνδρομή της Υπηρεσίας Αποθήκευσης των Ιστορικών Δεδομένων προς τις οντότητες των μετρήσεων είναι αυτό που φαίνεται στην εικόνα 4.9:

```
{
  "description": "Notify Cygnus of all observation changes",
  "subject": {
    "entities": [
      {
        "idPattern": ".*",
        "type": "Observation"
      }
    ],
    "condition": {
      "attrs": [ "hasSimpleResult" ]
    }
  },
  "notification": {
    "http": {
      "url": "http://localhost:5050/notify"
    },
    "attrs": [ "hasSimpleResult" ],
    "attrsFormat": "legacy"
  }
}
```

Εικόνα 4.9: Στιγμιότυπο της συνδρομής της Υπηρεσίας Αποθήκευσης των Ιστορικών Δεδομένων προς τις οντότητες των μετρήσεων των αισθητήρων (σε μορφή JSON).

Η παραπάνω αναπαράσταση NGSI2 αποτελείται από κάποια πεδία:

- **“description”**: Μια σύντομη περιγραφή της συνδρομής.
- **“subject”**: περιλαμβάνει τα χαρακτηριστικά
- **“entities”**: δηλαδή τις οντότητες στις οποίες δημιουργείται η συνδρομή (στην περίπτωση αυτή επιλέχθηκε τύπος οντότητας)
- **“condition”**: υποδηλώνει τη συνθήκη αλλαγής της οντότητας. Συγκεκριμένα, το “entities” υποδεικνύει μέσω του attribute “type” ότι μας ενδιαφέρει ο τύπος Observation και

- **“idPattern”**: η τιμή “.*” του υποδηλώνει ότι μας ενδιαφέρουν όλα τα διαφορετικά στιγμιότυπα μετρήσεων του συστήματος.
- **“condition”**: εμπεριέχει ένα ακόμα χαρακτηριστικό, το
 - **“attributes” (“attrs”)**: το οποίο υποδεικνύει για ποιο ή για ποια χαρακτηριστικά μας ενδιαφέρει να λαμβάνουμε ενημερώσεις σχετικά με τις αλλαγές στα δεδομένα τους. Στην παρούσα περίπτωση, ο τύπος “Observation” εκπροσωπεί πάντα μια συγκεκριμένου είδους μέτρηση και αυτό που μας ενδιαφέρει είναι το attribute “hasSimpleResult” που κρατάει την τιμή της μέτρησης.
- **“notification”**: υποδηλώνεται το URI (<http://localhost:5050/notify>) στο οποίο θα γίνεται η αποστολή των ειδοποιήσεων - πρόκειται για το τελικό σημείο μέσω του οποίου θα λαμβάνει τις ενημερώσεις η Υπηρεσία Αποθήκευσης των Ιστορικών Δεδομένων και ειδικότερα η υπηρεσία Cygnus.

Το αίτημα για να δημιουργηθεί η αντίστοιχη συνδρομή της Υπηρεσίας Οντολογίας είναι παρόμοιο με το προηγούμενο (εκείνο στην Εικόνα 4.9), αλλά στα attributes του “condition” προστίθεται το “resultTime” (καθώς μας ενδιαφέρει η αλλαγή στη χρονική στιγμή της μέτρησης), ενώ αλλάζει και το URI στο οποίο αποστέλλονται οι ενημερώσεις. Συγκεκριμένα, αυτό αντικαθίσταται από ένα URI που αποτελεί τελικό σημείο της Υπηρεσίας Οντολογίας και, μόλις η πληροφορία φτάσει σε αυτό, γίνεται η ενημέρωση των τιμών στην οντολογία. Ακόμα, στα attributes του “notification” προστίθεται το “id” (το μοναδικό αναγνωριστικό της κάθε οντότητας), γεγονός που διευκολύνει την υλοποίηση της Υπηρεσίας Οντολογίας.

```
{
  "description": "Observation sub",
  "subject": {
    "entities": [
      {
        "idPattern": ".*",
        "type": "Observation"
      }
    ],
    "condition": {
      "attrs": [
        "hasSimpleResult",
        "resultTime"
      ]
    }
  },
  "notification": {
    "attrs": [
      "id",
      "hasSimpleResult",
      "resultTime"
    ],
    "http": {
      "url": "http://147.27.60.182:8080/jena-examples-0.0.1-SNAPSHOT/update"
    }
  }
}
```

Εικόνα 4.10: Στιγμιότυπο της συνδρομής της Υπηρεσίας Οντολογίας προς τις οντότητες των μετρήσεων των αισθητήρων (σε μορφή JSON).

Orion Context Broker - Διαγραφή Οντοτήτων

Η διαγραφή οντοτήτων (π.χ. η διαγραφή συνδρομών που προβλέπεται ως λειτουργία από την Υπηρεσία Υλοποίησης του Μοντέλου του Ιστού των Πραγμάτων) επιτυγχάνεται μέσω ενός REST αιτήματος που χρησιμοποιεί τη μέθοδο DELETE του πρωτοκόλλου HTTP.

Orion Context Broker - Ανάκτηση Οντοτήτων

Η ανάκτηση οντοτήτων πραγματοποιείται μέσω ενός REST αιτήματος που χρησιμοποιεί τη μέθοδο GET του πρωτοκόλλου HTTP (βλ πίνακα 4.2). Οι REST μέθοδοι που υλοποιούν όλες τις λειτουργίες της υπηρεσίας Orion Context Broker που αναλύονται παραπάνω (αλλά και στη συνέχεια του παρόντος Κεφαλαίου, όπου παρουσιάζονται αναλυτικά οι λειτουργίες του Web Thing Model) παρατίθενται στον παρακάτω πίνακα:

Πίνακας 4.2: HTTP μέθοδοι της υπηρεσίας Orion Context Broker.

Μέθοδος	URL Μεθόδου	Σώμα αιτήματος	Περιγραφή μεθόδου
POST	v2/entities	Εικόνα 4.2	Δημιουργία μιας οποιασδήποτε από τις οντότητες που προαναφέρθηκαν, για παράδειγμα της οντότητας “Αισθητήρα” (μοντέλου συσκευής)
POST	v2/subscriptions	Εικόνα 4.9 / Εικόνα 4.10	Δημιουργία συνδρομής προς όλες τις οντότητες μετρήσεων (είτε από την Υπηρεσία Αποθήκευσης των Ιστορικών Δεδομένων είτε από την Υπηρεσία Οντολογίας).
GET	v2/entities/{entity_id}?type=Thing		Ανάκτηση της περιγραφής συγκεκριμένης οντότητας Πράγματος (Web Thing).
GET	v2/entities?type=Sensor		Ανάκτηση όλων των οντοτήτων “Αισθητήρων”, δηλαδή όλων των μοντέλων συσκευών.

GET	v2/entities/{entity_id}/attrs?type=Sensor		Ανάκτηση όλων των χαρακτηριστικών ιδιοτήτων (properties) ενός συγκεκριμένου μοντέλου αισθητήρα.
GET	v2/entities/{entity_id}/attrs?type=Sensor& attrs=observes		Ανάκτηση μετρήσιμης ιδιότητας ενός συγκεκριμένου μοντέλου αισθητήρα.
GET	v2/entities/{entity_id}?type=Actuation& attrs=actuationEnabled,hasFeatureOfInterest, actuationMadeBy		Ανάκτηση σημαντικών χαρακτηριστικών μιας συγκεκριμένης εκτέλεσης ενέργειας (actuation) από συσκευή.
PUT	v2/entities/{entity_id}/attrs/hasSimpleResult	{ "value": {the_value} }	Αλλαγή τιμής μέτρησης ενός Observation
DELETE	v2/entities/{entity_id}?type=Sensor		Διαγραφή οντότητας "Αισθητήρα" (χρησιμοποιείται στη λειτουργία ενημέρωσης μοντέλου αισθητήρα).
DELETE	v2/entities/{entity_id}?type=Thing		Διαγραφή οντότητας "Πράγματος" (χρησιμοποιείται στη λειτουργία ενημέρωσης οντότητας Πράγματος).
DELETE	v2/subscriptions/{entity_id}		Διαγραφή συνδρομής.

4.1.4 Υπηρεσία Οντολογίας

Η Υπηρεσία Οντολογίας, όπως έχει ήδη αναφερθεί, αποτελείται από το Jena API και την οντολογία, όπως περιγράφηκε στο κεφάλαιο 3.5, που είναι αποθηκευμένη στο ΣΔΒΔ Virtuoso. Προκειμένου να εξυπηρετήσει τα αιτήματα που δέχεται από τις υπόλοιπες υπηρεσίες, βασίζεται στο Jena API, που είναι υπεύθυνο για κάθε είδους επικοινωνία της οντολογίας με τις υπηρεσίες του υπόλοιπου συστήματος - το Jena API αποτελεί μια "γέφυρα" μεταξύ της οντολογίας και των υπόλοιπων υπηρεσιών. Χάρη σε αυτό, μπορεί να γίνει ανάκτηση πληροφοριών από την οντολογία, εισαγωγή νέων πληροφοριών σε αυτήν, ενημέρωση ή ακόμα και διαγραφή τους. Ακόμα, το Jena είναι υπεύθυνο για την πραγματοποίηση της σημασιολογικής αιτίασης / συλλογισμού

(reasoning), το οποίο γίνεται κυρίως κατά την προσθήκη νέων δεδομένων, ούτως ώστε να βρεθούν όλες οι πιθανές ασυνέπειες που μπορεί να προκύψουν, καθώς η εξαγωγή νέων πληροφοριών από αυτές που ήδη υπάρχουν στην οντολογία. Η Υπηρεσία Οντολογίας, και συνεπώς όλες οι λειτουργίες που πραγματοποιούνται μέσω του Jena API, υλοποιήθηκαν με τη χρήση του JavaEE και του Spring Framework. Η εν λόγω υπηρεσία έχει ενσωματώσει έναν Tomcat server (εξυπηρετητή) και για το τελικό σημείο του Tomcat ορίζουμε ένα ειδικό port στην εικονική μηχανή που φιλοξενεί το Jena API (το port 8080). Οι REST μέθοδοι που υλοποιούν όλες τις λειτουργίες της υπηρεσίας Οντολογίας που αναλύονται παραπάνω παρατίθενται στον παρακάτω πίνακα:

Πίνακας 4.3: HTTP μέθοδος της Υπηρεσίας Οντολογίας (Jena API).

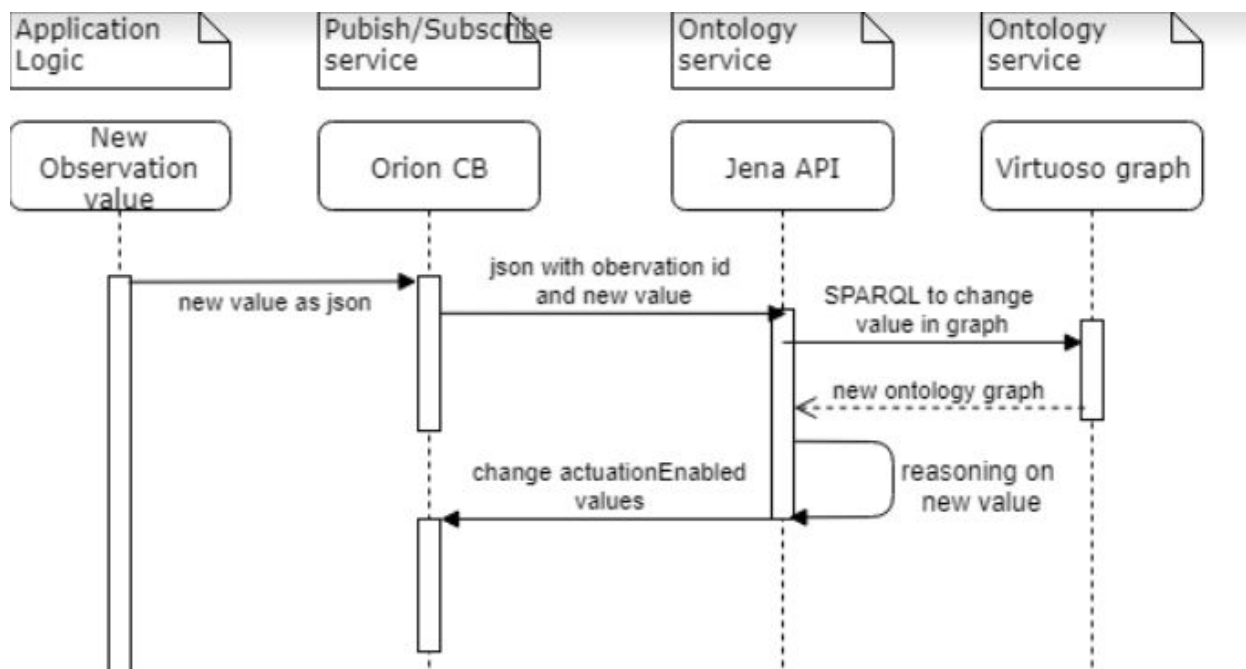
Μέθοδος	URL Μεθόδου	Σώμα αιτήματος	Περιγραφή μεθόδου
POST	/jena-project-0.0.1-SNAPSHOT/ create_wt_add_to_virt	Η περιγραφή JSON του Πράγματος, η οποία δίνεται από το χρήστη μέσω του Web Application.	Αίτημα εισαγωγής περιγραφής JSON ενός Πράγματος (Web Thing) στην οντολογία.
POST	/jena-project-0.0.1-SNAPSHOT/ update_wt	Η περιγραφή JSON του Πράγματος, η οποία δίνεται από το χρήστη μέσω του Web Application.	Αίτημα ενημέρωσης περιγραφής JSON ενός Πράγματος (Web Thing) στην οντολογία.
POST	/jena-project-0.0.1-SNAPSHOT/ create_model_add_to_virt	Η περιγραφή JSON-LD του μοντέλου του Πράγματος, η οποία δίνεται από το χρήστη μέσω του Web Application.	Αίτημα εισαγωγής περιγραφής JSON-LD του μοντέλου ενός Πράγματος (Web Thing Model) στην οντολογία.
POST	/jena-project-0.0.1-SNAPSHOT/ update_model	Η περιγραφή JSON-LD του μοντέλου του Πράγματος, η οποία δίνεται από το χρήστη μέσω του Web Application.	Αίτημα ενημέρωσης περιγραφής JSON-LD του μοντέλου ενός Πράγματος (Web Thing Model) στην οντολογία.
POST	/jena-project-0.0.1-SNAPSHOT/ update	Οι τιμές που αλλάζουν και στέλνονται από	Αίτημα προσθήκης των νέων τιμών των

		τον Orion (βλ. Εικόνα 4.8).	μετρήσεων αισθητήρων στην οντολογία (μετά την ενημέρωση που πυροδοτείται από τον Orion Context Broker).
POST	/jena-project-0.0.1-SNAPSHOT/execute_action	Η περιγραφή JSON της ενέργειας που πρόκειται να εκτελεστεί.	Αίτημα εκτέλεσης ενέργειας (σε μια συσκευή). Η πληροφορία της ενέργειας που θα εκτελεστεί αποθηκεύεται στην οντολογία.
GET	/jena-project-0.0.1-SNAPSHOT/properties		Αίτημα εύρεσης όλων των μετρήσιμων ιδιοτήτων των μοντέλων αισθητήρων που έχουν αποθηκευτεί στην οντολογία.
GET	/jena-project-0.0.1-SNAPSHOT/action		Αίτημα εύρεσης όλων των δυνατών ενεργειών που πραγματοποιούνται από αισθητήρες.
GET	/jena-project-0.0.1-SNAPSHOT/actuators		Αίτημα εύρεσης όλων των ενεργοποιητών (actuators) που έχουν αποθηκευτεί στην οντολογία.
GET	/jena-examples-0.0.1-SNAPSHOT/select?thequery={the_query}	Το SPARQL ερώτημα τύπου SELECT QUERY.	SELECT QUERY (SPARQL) μέσω reasoning στους γράφους του Virtuoso.
GET	/jena-examples-0.0.1-SNAPSHOT/ask?thequery={the_query}	Το SPARQL ερώτημα τύπου ASK QUERY.	ASK QUERY (SPARQL) μέσω reasoning στους γράφους του Virtuoso.
POST	/jena-examples-0.0.1-SNAPSHOT/updateactuators		Αίτημα που πυροδοτεί συγχρονισμό τιμών ενεργοποιητών στον

			Orion σύμφωνα με την Οντολογία.
POST	/jena-examples-0.0.1-SNAPSHOT/update?response={the_response}	Το JSON notification που πυροδοτείται από τον Orion στην αλλαγή μίας τιμής και περιέχει id του Observation που άλλαξε, την νέα τιμή και την χρονική στιγμή.	Αίτημα που στέλνεται αυτόματα από τον Orion μέσω του Subscription που έχει γίνει, που περιέχει ποιο entity άλλαξε τιμή, ποια είναι η τιμή και τότε άλλαξε, ώστε να αποθηκευτούν στην Οντολογία, να γίνει reasoning και να αλλάξουν τυχόν τιμές ενεργοποιητών στον Orion.

Διαδικασία αλλαγής τιμών - Νέες μετρήσεις.

Στην εικόνα 4.11 φαίνεται ποιες υπηρεσίες και πως συνεργάζονται όταν έρχονται νέες μετρήσεις από τους αισθητήρες, δηλαδή αλλάζει το πεδίο “**hasSimpleResult**” ενός **Observation entity** στον Orion.



Εικόνα 4.11: Διάγραμμα ροής αλλαγής τιμών.

Όπως αναφέρθηκε προηγουμένως, έχει δημιουργηθεί ένα Subscription στον Orion, το οποίο μόλις έρθει νέα τιμή, πυροδοτεί ένα notification το οποίο περιέχει:

- **id** του Observation που αλλάζει τιμή.
- Την νέα τιμή του πεδίου **“hasSimpleResult”** (νέα μέτρηση).
- Τη χρονική τιμή του πεδίου **“resultTime”** δηλαδή τη χρονική στιγμή μέτρησης.

Το notification στέλνεται υπό την μορφή JSON, με HTTP POST αίτημα στο URL: [“http://147.27.60.182:8080/jena-examples-0.0.1-SNAPSHOT/update”](http://147.27.60.182:8080/jena-examples-0.0.1-SNAPSHOT/update) δηλαδή στην υπηρεσία Οντολογίας (Ontology Service) και συγκεκριμένα στο Jena API το οποίο ακούει και δέχεται την πληροφορία JSON. Στη συνέχεια, μέσω Jena, γίνονται οι εξής λειτουργίες:

1. Αλλαγή της σωστής τιμής μέτρησης στον αντίστοιχο γράφο στη Virtuoso.
2. Reasoning μετά την αλλαγή τιμής για έλεγχο ορθότητας στην οντολογία και
3. Για έλεγχο για άνοιγμα ή κλείσιμο ενεργοποιητών, εφόσον αναφερόμαστε σε σπίτι, και αλλαγής των αντίστοιχων τιμών στον Orion μέσω HTTP PUT αιτημάτων. Δηλαδή, μετά το reasoning που γίνεται ελέγχουμε ποιοι ενεργοποιητές κλείνουν και ποιοι ανοίγουν και αλλάζουμε κατάλληλα την τιμή **“actuationEnabled”** των οντοτήτων αυτών (Actuations) αντίστοιχα στον Orion.

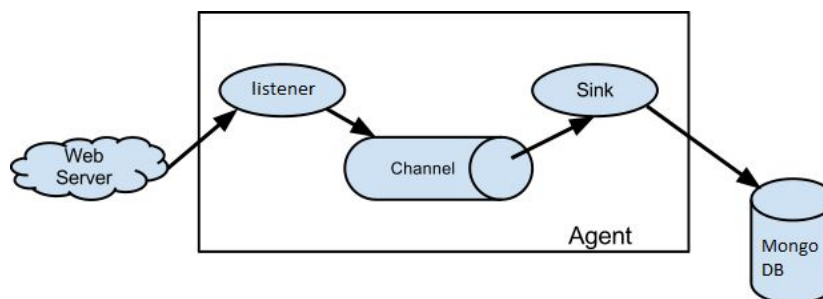
4.1.5 Υπηρεσία Αποθήκευσης των Ιστορικών Δεδομένων

Η υπηρεσία αυτή (όπως είδαμε και στο Κεφάλαιο 3) απαρτίζεται από την υπηρεσία Cygnus του FIWARE, καθώς και από την υπηρεσία STH Comet του FIWARE. Αμφότερες επικοινωνούν με την ιστορική βάση δεδομένων του συστήματος (MongoDB).

FIWARE Cygnus

Η υπηρεσία Cygnus είναι βασισμένη στην αρχιτεκτονική “Apache Flume” και παρέχει διάφορους πράκτορες (agents). Αυτοί είναι υπεύθυνοι για τη συλλογή ροών δεδομένων NGSI-2 και για την αποθήκευσή τους σε κάποια εξωτερική βάση δεδομένων που έχει προκαθοριστεί. Ένας πράκτορας (Εικόνα 4.12) απαρτίζεται από ένα “ακροατή” (listener), που είναι υπεύθυνος για τη λήψη των δεδομένων, ένα “κανάλι” (Channel) στο οποίο ο ακροατής προωθεί τα δεδομένα και έναν “προορισμό” (Sink), ο οποίος

“παραλαμβάνει” τα δεδομένα με σκοπό να τα αποθηκεύσει σε μία εξωτερική βάση δεδομένων (η οποία ενδέχεται να είναι σε ξεχωριστή εικονική μηχανή από τον Πράκτορα).



Εικόνα 4.12: Apache flume Agent.

Το Cygnus παρέχει εξειδικευμένους πράκτορες οι οποίοι έχουν τη δυνατότητα να υποστηρίξουν τη συλλογή και τη διατήρηση NGSΙ δεδομένων στις εξής υπηρεσίες βάσης δεδομένων - αποθετηρίων:

- HDFS, το σύστημα κατανομής αρχείων Hadoop.
- MySQL, το γνωστό σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων.
- CKAN, μια πλατφόρμα Open data.
- MongoDB, η NoSQL βάση δεδομένων για έγγραφα.
- Kafka, ο μεσίτης μηνυμάτων δημοσίευσης - εγγραφής.
- DynamoDB, μια NoSQL βασισμένη στο υπολογιστικό νέφος βάση δεδομένων της Amazon Web Services,
- PostgreSQL, το γνωστό σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων.
- CarTo, μια βάση δεδομένων που ειδικεύεται σε γεωγραφικά δεδομένα.

Για παράδειγμα, στο σενάριο κατά το οποίο πρέπει να αποθηκεύσουμε ροές δεδομένων NGSΙ σε μία βάση δεδομένων MongoDB (όπως συμβαίνει στην παρούσα εργασία), απαιτείται η χρήση ενός εξειδικευμένου “MongoDB πράκτορα” της υπηρεσίας Cygnus. Ομοίως, για το σενάριο αποθήκευσης σε βάση δεδομένων MySQL, θα γίνει χρήση ενός εξειδικευμένου “MySQL πράκτορα”. Στα πλαίσια της παρούσας εργασίας, έχει τεθεί σε λειτουργία ένας εξειδικευμένος πράκτορας του Cygnus, με σκοπό την αποθήκευση τόσο ακατέργαστων (Raw) όσο και συγκεντρωτικών (Aggregated) δεδομένων στην ιστορική βάση MongoDB του συστήματος. Αξιοποιώντας τη λειτουργία συνδρομής που παρέχει η υπηρεσία Orion Context Broker, ο πράκτορας αποτελεί συνδρομητή για όλες τις οντότητες μετρήσεων του Orion. Με αυτόν τον τρόπο, όπως αναλύθηκε και παραπάνω, οποιαδήποτε αλλαγή συμβαίνει στο χαρακτηριστικό (attribute) “hasSimpleResult” σε οντότητα του τύπου Observation - δηλαδή

οποιαδήποτε νέα μέτρηση - πυροδοτεί μια ενημέρωση από τον Ορίον προς το τελικό σημείο του συνδρομητή πράκτορα. Συνεπώς, ο πράκτορας λαμβάνει όλες τις νέες μετρήσεις των αισθητήρων (όταν αυτές προκύπτουν). Αμέσως μετά, αναλαμβάνει να αποθηκεύσει τις μετρήσεις αυτές ως ακατέργαστα και συγκεντρωτικά δεδομένα (σημαντική λειτουργία που επιτυγχάνεται από τον Cygnus) στην ιστορική βάση δεδομένων, διατηρώντας έτσι ένα ιστορικό δεδομένων χρονικής σειράς για τις μετρήσεις όλων των αισθητήρων που έχουν συνδεθεί στο σύστημα.

Προκειμένου να αποθηκευτούν οι συγκεντρωτικές πληροφορίες στην ιστορική βάση δεδομένων, διατηρούνται σε αυτήν ξεχωριστές μεταβλητές που αφορούν τα εξής δεδομένα:

1. Την μέγιστη τιμή των μετρήσεων του κάθε αισθητήρα για συγκεκριμένα χρονικά διαστήματα (την τελευταία ώρα / μέρα / μήνα).
2. Την ελάχιστη τιμή των μετρήσεων του κάθε αισθητήρα για συγκεκριμένα χρονικά διαστήματα (την τελευταία ώρα / μέρα / μήνα).
3. Την αθροιστική τιμή των μετρήσεων του κάθε αισθητήρα για συγκεκριμένα χρονικά διαστήματα (την τελευταία ώρα / μέρα / μήνα), η οποία μπορεί στη συνέχεια να αξιοποιηθεί από το Mashup Service για εύρεση του μέσου όρου των τιμών των μετρήσεων.

Σε αυτό το σημείο, καθώς η λειτουργία ενός πράκτορα της υπηρεσίας Cygnus έχει γίνει πιο κατανοητή, αξίζει να σημειωθεί ένα ακόμα θετικό που προσφέρει σε επίπεδο αρχιτεκτονικής. Όπως εξηγήθηκε στην αρχή της ενότητας ο πράκτορας αποτελείται από ένα ακροατή, ένα κανάλι και ένα “sink”. Το κανάλι λειτουργεί σαν προσωρινή αποθήκη δεδομένων που λαμβάνονται από τον ακροατή (Το μέγεθος της μνήμης του καναλιού καθορίζεται κατά την αρχικοποίηση του πράκτορα), το Sink αναλαμβάνει να «πάρει» δεδομένα που βρίσκονται προσωρινά στο κανάλι και να τα αποθηκεύσει στην εξωτερική βάση δεδομένων. Εφόσον τα δεδομένα αποθηκευτούν με επιτυχία στην εξωτερική βάση δεδομένων, διαγράφονται από το κανάλι, διαφορετικά τα δεδομένα συνεχίζουν να διατηρούνται στο κανάλι. Με αυτόν τον τρόπο, μία αποτυχημένη εγγραφή (πχ. λόγω καθυστέρησης δικτύου, λόγου προσωρινής υπερφόρτωσης της βάσης δεδομένων κλπ) μπορεί να επαναληφθεί χωρίς να χαθούν τα δεδομένα. Το διάγραμμα ροής της αλλαγής και αποθήκευσης των παλαιών τιμών φαίνεται στην εικόνα 4.11 (βλ. Διαδικασία αλλαγής τιμών - Νέες τιμές).

Στον παρακάτω πίνακα περιγράφεται η REST μέθοδος που παρέχεται από την υπηρεσία Cygnus για τη λήψη και την αποθήκευση ροών δεδομένων NGSI-2:

Πίνακας 4.4: HTTP μέθοδος της υπηρεσίας Cygnus.

Μέθοδος	URL Μεθόδου	Περιγραφή Μεθόδου
POST	/notify	Πρόκειται για το τελικό σημείο του συνδρομητή πράκτορα της υπηρεσίας στο οποίο αποστέλλονται ροές δεδομένων NGSi για αποθήκευση. (Το είδαμε και προηγουμένως στην εικόνα 4.7, στην δημιουργία συνδρομής προς οντότητες αισθητήρων

FIWARE - COMET

Όπως έχει ήδη επισημανθεί, η υπηρεσία COMET παρέχεται επίσης από το FIWARE και συνδέεται τοπικά με την ιστορική βάση δεδομένων MongoDB του συστήματος (History MongoDB). Η υπηρεσία φιλοξενείται μάλιστα στην ίδια εικονική μηχανή με τη βάση. Χρησιμοποιώντας τη RESTful διεπαφή της, η υπηρεσία επιτρέπει την ανάκτηση ακατέργαστων (Raw) και συγκεντρωτικών (Aggregated) δεδομένων που έχουν αποθηκευτεί στην ιστορική βάση δεδομένων. Η αποθήκευση αυτών των δεδομένων, όπως αναφέρεται και παραπάνω, έχει επιτευχθεί από τον πράκτορα του Cygnus. Στον πίνακα που ακολουθεί περιγράφεται η μέθοδος της υπηρεσίας COMET που χρησιμοποιήθηκε κατά την υλοποίηση της παρούσας εργασίας:

Πίνακας 4.5: HTTP μέθοδοι υπηρεσίας FIWARE-COMET.

Μέθοδος	URL Μεθόδου	Περιγραφή Μεθόδου
GET	/STH/v1/contextEntities/ type/Observation/ id/{Observation_id} /attributes/hasSimpleResult &dateFrom ={2019-01-01T00:00:00.000Z} &dateTo ={2019-01-2T23:59:59.999Z}	Αίτημα ανάκτησης ακατέργαστης ιστορικής πληροφορίας, που αφορά τις μετρήσεις της παρατήρησης με αναγνωριστικό {Observation_id}, στο διάστημα περιόδου από {2019-01-01T00:00:00.000Z} μέχρι {2019-01-2T23:59:59.999Z} Όπως αναγράφονται στο URL

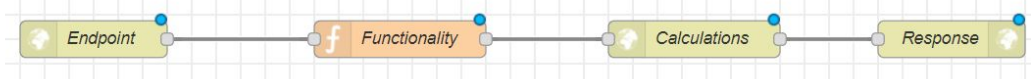
GET	/STH/v1/contextEntities/ type/Observation/ id/{Observation_id} /attributes/hasSimpleResult &LastN=1	Αίτημα ανάκτησης τελευταίας εγγεγραμμένης τιμής (τρέχουσα τιμή), που αφορά την μέτρηση της παρατήρησης με αναγνωριστικό {Observation_id} Όπως αναγράφονται στο URL.
GET	/STH/v1/contextEntities/ type/Observation/ id/{Observation_id} /attributes/hasSimpleResult &aggrMethod={max} &aggrPeriod= {Hour} &dateFrom ={2018-01-01T00:00:00.000Z} &dateTo ={2018-01-2T23:59:59.999Z}	Αίτημα ανάκτησης συγκεντρωτικής ιστορικής πληροφορίας, που αφορά την μέγιστη μέτρηση ανά ώρα της παρατήρησης με αναγνωριστικό {Observation_id}, στο διάστημα περιόδου από : {2019-01-01T00:00:00.000Z} μέχρι : {2019-01-2T23:59:59.999Z}. Όπως αναγράφονται στο URL. Το “aggrMethod” μπορεί να λάβει τις τιμές “sum”, “max”, “min”. Το “aggrPeriod” μπορεί να λάβει τις τιμές “month”, “day”, “hour”. Για παράδειγμα με aggrMethod: “min” και aggrPeriod: “day” δηλώνουμε ότι θέλουμε την ελάχιστη μέτρηση ανά μέρα.

4.1.6 Υπηρεσία Mashup

Η υπηρεσία Mashup υλοποιείται ως σύνθεση των επιμέρους υπηρεσιών Node-RED και Calculations και ακολουθεί τις προδιαγραφές σχεδιασμού που αποτυπώθηκαν στην ενότητα 3.6. Παρακάτω γίνεται μία αναλυτική περιγραφή της λειτουργίας-συνεργασίας των υπηρεσιών αυτών.

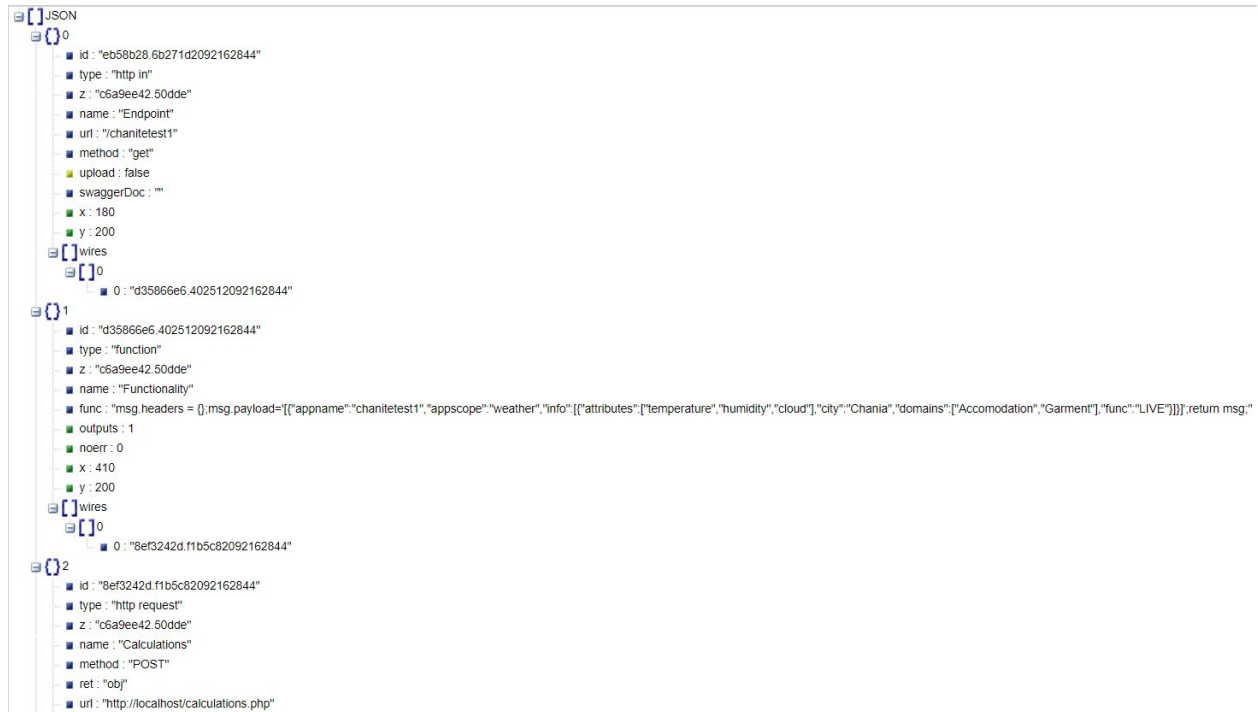
Node-RED

Η υπηρεσία Node-RED έχει τον πρωταρχικό ρόλο για την δημιουργία εκτελέσιμων εφαρμογών από προγραμματιστές, παρέχοντας παράλληλα ένα runtime περιβάλλον που επιτρέπει την εκτέλεση τους, από τελικούς χρήστες του συστήματος. Το Node-RED διαθέτει εγκατεστημένους και διαθέσιμους προς χρήση τους κόμβους οι οποίοι συνθέτουν το πρότυπο της εκτελέσιμης εφαρμογής (εικόνα 4.13) όπως το έχουμε ορίσει στις προδιαγραφές του συστήματος (ενότητα 3.6).



Εικόνα 4.13: Πρότυπο σύνθεσης εκτελέσιμης εφαρμογής.

Μέσω ενός REST αιτήματος προς το Node-RED, μπορούμε να δημιουργήσουμε μία εκτελέσιμη εφαρμογή περιγράφοντας στο σώμα του αιτήματος, τους κόμβους που θα συνθέτουν την εφαρμογή αλλά και το πώς αυτοί θα συνδέονται μεταξύ τους. Η περιγραφή της εφαρμογής στο σώμα του αιτήματος γίνεται σε JSON η οποία ακολουθεί τους κανόνες σύνταξης που ορίζει το Node-RED για την δημιουργία εφαρμογών μέσω του REST API. Η σύνταξη της περιγραφής JSON για την δημιουργία εκτελέσιμης εφαρμογής που ακολουθεί το πρότυπο σύνθεσης της εργασίας (εικόνας 4.13), παρουσιάζεται στην εικόνα 4.14 (Η JSON της εικόνας 4.14, προβάλλεται υπό την μορφή «πίνακα» για την ευκολότερη κατανόηση της). Η εφαρμογή έπειτα από το αίτημα δημιουργίας της, αποθηκεύεται στην βάση δεδομένων MongoDB της υπηρεσίας και είναι διαθέσιμη προς εκτέλεση στους τελικούς χρήστες του συστήματος.



Εικόνα 4.14: Σώμα JSON για την δημιουργία εφαρμογής στην υπηρεσία Node-RED .

Όπως φαίνεται στην εικόνα 4.14, η δομή JSON περιλαμβάνει τέσσερα εμφωλευμένα στοιχεία (element {0}, {1}, {2}, {3}). Τα στοιχεία αυτά αποτελούν την περιγραφή των κόμβων που θα συνθέτουν την εφαρμογή. Παρακάτω αναλύουμε την σύνταξη και το περιεχόμενο της εικόνας 4.14:

Κόμβος πρώτος {0} - Endpoint

"id": "eb58b28.6b271d2092162844" Το αναγνωριστικό "Id" του κόμβου.

"type": "http in" Δηλώνουμε τον τύπο του κόμβου σε "http in", προκειμένου να έχει την λειτουργικότητα ακροατή (Listener) που θα λαμβάνει HTTP αιτήματα στο τελικό σημείο του.

"method": "GET" Το τελικό σημείο θα προσπελάζεται με την μέθοδο GET του πρωτοκόλλου HTTP.

"url": "/chaniatest1" Ορίζουμε το URL του τελικού σημείου σε *"/chaniatest1"*. Ένας τελικός χρήστης θα έχει πρόσβαση στην παρούσα εφαρμογή μεταβαίνοντας στο URL: *"http://147.27.60.65:1880/chanitetest1"*.

"name": "Endpoint" Το όνομα που δίνουμε στον παρών κόμβο είναι "Endpoint".

"wires": "d35866e6.402512092162844 " Η έξοδος του παρών κόμβου θα συνδεθεί στην είσοδο του κόμβου με αναγνωριστικό Id: "d35866e6 " (κόμβο **Functionality**).

Κόμβος δεύτερος {1} - Functionality

"id": "d35866e6.402512092162844" Το αναγνωριστικό "Id" του κόμβου.

"type": "function" Δηλώνουμε τον τύπο του κόμβου σε "function" προκειμένου -ο κόμβος- να εκτελεί μία λειτουργικότητα Javascript.

"func": Πρόκειται για την συνάρτηση Javascript που θα εκτελέσει ο παρών κόμβος. Η συνάρτηση αυτή, όπως φαίνεται και στην εικόνα 4.14 έχει την εξής λειτουργικότητα: Η περιγραφή JSON των mashup της συγκεκριμένης εφαρμογής, υπάρχει αποθηκευμένη σαν συμβολοσειρά (String) στην μεταβλητή **"msg.payload"**

```
msg.payload={{
  "Appname":"chanitetest1",
  "Appscope":"weather",
  "Info":{
    "Attributes":[
      "Temperature",
      "Humidity",
      "Cloud"
    ],
    "city":"Chania",
    "Domains":[
      "Accommodation",
      "Garment"
    ],
    "func":"LIVE"
  }};
```

Μέσω της εντολής **"return msg;"** η περιγραφή JSON επιστρέφεται στην έξοδο του παρών κόμβου, προκειμένου να μεταδοθεί στην είσοδο του επόμενου κόμβου με τον οποίο συνδέεται.

"wires": "8ef3242d.f1b5c82092162844"

" Η έξοδος του παρών κόμβου θα συνδεθεί στην είσοδο του κόμβου με αναγνωριστικό Id: "8ef3242d.f1b5c82092162844 " (κόμβο **Calculations**).

"name": " functionality " Το όνομα που δίνουμε στον κόμβο είναι "Functionality".

Κόμβος τρίτος {2} - Calculations

"id": "8ef3242d.f1b5c82092162844"

" Το αναγνωριστικό Id του κόμβου.

"type": "http request" Δηλώνουμε τον τύπο του κόμβου σε "http request", προκειμένου -ο κόμβος- να λειτουργεί σαν αποστολέας HTTP αιτημάτων.

"method": "POST" Το αίτημα που θα αποστέλλει ο παρών κόμβος θα είναι της μεθόδου POST με σώμα -αιτήματος- την πληροφορία JSON που έλαβε στην είσοδο του, από τον προηγούμενο κόμβο *"Functionality"* (Δηλαδή την δομή JSON που περιγράφει τα mashup της εφαρμογής).

"url": "http://localhost/calculations" Ορίζουμε το τελικό σημείο στο οποίο ο παρών κόμβος θα στείλει το αίτημα. Το URL: *"http://localhost/calculations"* πρόκειται για το τελικό σημείο της υπηρεσίας Calculations (αναλύσαμε στην προηγούμενη ενότητα). Ουσιαστικά ο παρών κόμβος αποστέλλει μέσω ενός POST αιτήματος, την πληροφορία JSON (έλαβε στην είσοδο) που περιγράφει τα mashup της εφαρμογής στην υπηρεσία Calculations. Η υπηρεσία Calculations λαμβάνει το αίτημα του κόμβου και του απαντάει τον προσαρμοσμένο κώδικα HTML/Javascript, που υλοποιεί την γραφική απεικόνιση των mashup (όπως τα περιέγραφε το αίτημα που δέχτηκε).

"ret": "obj" Η απάντηση που θα λάβει ο παρών κόμβος για το HTTP αίτημα στην υπηρεσία Calculations, θα επιστραφεί στην έξοδο του σαν "αντικείμενο" (object) προκειμένου να μεταδοθεί στην είσοδο του επόμενου κόμβου με τον οποίο συνδέεται.

"wires": "784c5a17.4f59b42092162844"

" Η έξοδος του παρών κόμβου θα συνδεθεί στην είσοδο του κόμβου με αναγνωριστικό ID "784c5a17" (κόμβο **Response**).

"name": "Calculations" Το όνομα που δίνουμε στον παρών κόμβο είναι "Calculations".

Κόμβος τέταρτος {3} - Response

"id": "784c5a17.4f59b42092162844"

" Το αναγνωριστικό Id του κόμβου.

"type": "http Response" Δηλώνουμε τον τύπο του κόμβου σε "http Response". Ο κόμβος προωθεί τον κώδικα -HTML/Javascript- που λαμβάνει στην είσοδο του (από τον προηγούμενο κόμβο), στο πρόγραμμα περιήγησης του τελικού χρήστη που προσπέλασε αρχικά το τελικό σημείο της εφαρμογής. Ο κώδικας εκτελείται στο πρόγραμμα περιήγησης του τελικού χρήστη και παράγει στην οθόνη του την γραφική αναπαράσταση της εφαρμογής.

"name": "Response" Το όνομα που δίνουμε στον κόμβο είναι "Response".

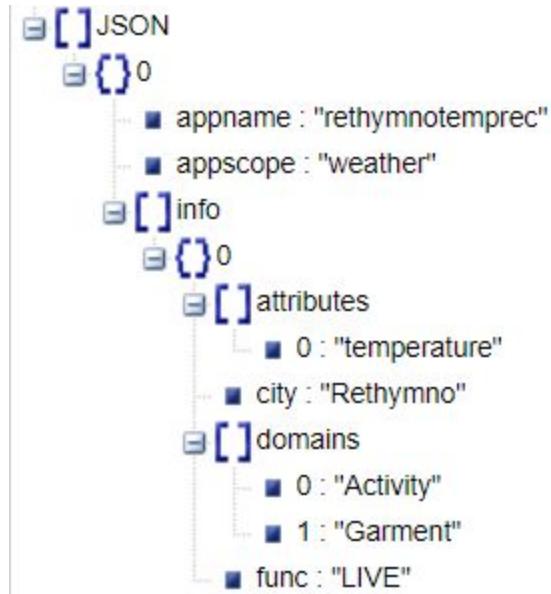
Calculations

Πρόκειται για ένα διακομιστή Apache ο οποίος περιέχει όλους τους αλγορίθμους (σε γλώσσα PHP) που υλοποιούν τα διαφορετικά mashup δεδομένων που υποστηρίζει το σύστημα. Είναι προγραμματισμένος ώστε να εκτελεί την παρακάτω διαδικασία:

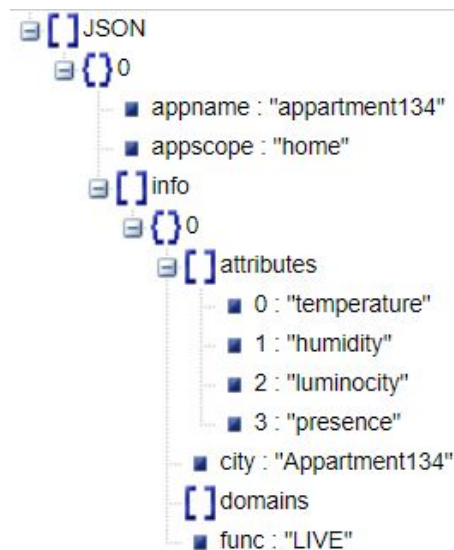
1. Δέχεται HTTP αιτήματα της μεθόδου POST, με σώμα -αιτήματος- την πρότυπη δομή JSON η οποία περιγράφει τα mashup που περιλαμβάνει μία εφαρμογή. Τα αιτήματα αυτά έρχονται από την υπηρεσία Node-RED και συγκεκριμένα από τον τρίτο κόμβο με όνομα Calculations, ο οποίος αναλύθηκε προηγουμένως (υποενότητα Node-RED). Το πρότυπο περιγραφής των mashup μιας εφαρμογής σε JSON, παρουσιάστηκε αναλυτικά στην ενότητα 3.6, ενώ παρατίθενται ξανά δύο παραδείγματα στις εικόνες 4.15, 4.16.
2. Απαντάει στην έξοδο του, τον προσαρμοσμένο κώδικα HTML/Javascript ο οποίος σχεδιάζει την γραφική αναπαράσταση των mashup που περιγράφονται στην δομή JSON που έλαβε στην είσοδο (βλ. παρακάτω). Η έξοδος του είναι πάλι ο τρίτος κόμβος Calculations του Node-RED ο οποίος στη συνέχεια προωθεί την πληροφορία στον τελευταίο κόμβο Response.

Με την άφιξη ενός αιτήματος (Για παράδειγμα εικόνες 4.15, 4.16) γίνεται ανάλυση της δομής JSON. Για κάθε διαφορετικό mashup που ζητείται θα καλεστεί ο κατάλληλος αλγόριθμος για τον υπολογισμό του με βάση τα πεδία "appscope" και "func" που περιέχει. Ο αλγόριθμος θα λάβει ως ορίσματα εισόδου:

1. την λίστα των μετρήσιμων χαρακτηριστικών "attributes" (πχ. Temperature, humidity) που περιλαμβάνονται στο mashup,
2. το σπίτι ή πόλη στο οποίο ανήκουν τα παραπάνω χαρακτηριστικά "city" (πχ. Athens, Room145) που θα απεικονίζει το mashup (βλ. παρακάτω).
3. Προαιρετικά τους τομείς της ζωής που ο χρήστης ενδιαφέρεται να λάβει απάντηση "domains".



Εικόνα 4.15: Αναπαράσταση των mashup μίας εφαρμογής πόλης σε JSON.



Εικόνα 4.16: Αναπαράσταση των mashup μίας εφαρμογής σπιτιού σε JSON.

Οι δυνατοί αλγόριθμοι mashup που υποστηρίζει η υλοποίηση του συστήματος μέσω της υπηρεσίας Calculations είναι οι παρακάτω:

1. **Function MAX_24hr(Array attributes, String city):** Ο αλγόριθμος αυτός καλείται για την υλοποίηση ενός mashup με κωδική λέξη “operation”: “MAX_24hr” και η λειτουργία του είναι η εξής: Για κάθε διαφορετικό χαρακτηριστικό στη λίστα “attributes” (πχ. Temperature, humidity), πραγματοποιείται ένα HTTP αίτημα

στην υπηρεσία FIWARE-COMET με το οποίο ανακτάται για το δοθέν “attribute” (πχ. temperature), η μέγιστη τιμή μέτρησης του της παρατήρησης που έκανε ο αισθητήρας που παρακολουθεί το συγκεκριμένο χαρακτηριστικό ανά ώρα για τις τελευταίες 24 ώρες. Έχοντας ανακτήσει αυτά τα δεδομένα για κάθε διαφορετικό αισθητήρα της λίστας, ο αλγόριθμος δημιουργεί ένα πίνακα (Array) με τις ολικές μέγιστες τιμές -ανά ώρα- που μετρήθηκαν ανάμεσα στο συγκεκριμένο υποσύνολο αισθητήρων τις τελευταίες 24 ώρες. Τα δεδομένα του πίνακα θα αναπαρασταθούν γραφικά σε ένα διάγραμμα γραμμής (Line chart), το οποίο θα απεικονίζει μια κυματομορφή με την ολική μέγιστη μέτρηση ανά ώρα, για τις τελευταίες 24 ώρες.

2. **Function MIN_24hr(Array attributes, String city):** Ο αλγόριθμος αυτός καλείται για την υλοποίηση ενός mashup με κωδική λέξη “operation”: “MIN_24hr”. Έχει ακριβώς την ίδια λειτουργικότητα όπως περιγράφηκε στην (1), με την διαφορά ότι η διαδικασία αφορά ελάχιστες τιμές μετρήσεων ανά ώρα αντί για μέγιστες.
3. **Function AVERAGE_24hr(Array attributes, String city):** Ο αλγόριθμος αυτός καλείται για την υλοποίηση ενός mashup με κωδική λέξη “operation”: “AVERAGE_24hr”. Έχει ακριβώς την ίδια λειτουργικότητα όπως περιγράφηκε στην (1), με την διαφορά ότι η διαδικασία αφορά μέσες τιμές μετρήσεων ανά ώρα αντί για μέγιστες.
4. **Function MAX_7D(Array attributes, String city):** Ο αλγόριθμος αυτός καλείται για την υλοποίηση ενός mashup με κωδική λέξη “operation”: “MAX_7D”, η λειτουργία του είναι η εξής: Για κάθε διαφορετικό χαρακτηριστικό στη λίστα “attributes” (πχ. Temperature, humidity), πραγματοποιείται ένα HTTP αίτημα στην υπηρεσία FIWARE-COMET με το οποίο ανακτάται για το δοθέν “attribute” (πχ. temperature), η μέγιστη τιμή μέτρησης του της παρατήρησης που έκανε ο αισθητήρας που παρακολουθεί το συγκεκριμένο χαρακτηριστικό ανά μέρα για τις τελευταίες 7 μέρες. Έχοντας ανακτήσει αυτά τα δεδομένα για κάθε διαφορετικό αισθητήρα της λίστας, ο αλγόριθμος δημιουργεί ένα πίνακα (Array) με τις ολικές μέγιστες τιμές ανά μέρα που μετρήθηκαν ανάμεσα στο συγκεκριμένο υποσύνολο αισθητήρων τις τελευταίες 7 μέρες. Τα δεδομένα του πίνακα θα αναπαρασταθούν γραφικά σε ένα διάγραμμα γραμμής (Line Chart), το οποίο θα απεικονίζει μια κυματομορφή με την ολική μέγιστη μέτρηση ανά μέρα, για τις τελευταίες 7 μέρες.
5. **Function MIN_7D(Array attributes, String city):** Ο αλγόριθμος αυτός καλείται για την υλοποίηση ενός mashup με κωδική λέξη “operation”: “MIN_7D”. Έχει

ακριβώς την ίδια λειτουργικότητα όπως περιγράφηκε στην (4) με την διαφορά ότι η διαδικασία αφορά ελάχιστες τιμές μετρήσεων ανά μέρα αντί για μέγιστες.

6. **Function AVERAGE_7D(Array attributes, String city):** Ο αλγόριθμος αυτός καλείται για την υλοποίηση ενός mashup με κωδική λέξη “operation”: “AVERAGE_7D” . Έχει ακριβώς την ίδια λειτουργικότητα όπως περιγράφηκε στην (4) με την διαφορά ότι η διαδικασία αφορά μέσες τιμές μετρήσεων ανά μέρα αντί για μέγιστες.
7. **Function LIVE(Array attributes, String city, String appscope):** Πρόκειται για τον αλγόριθμο που αποτελεί το σημασιολογικό κομμάτι της υπηρεσίας εφαρμογών. Ο αλγόριθμος αυτός καλείται για την υλοποίηση mashup με κωδική λέξη “operation”: “LIVE”. Σε αυτή την περίπτωση για κάθε διαφορετικό χαρακτηριστικό στη λίστα “attributes” (πχ. Temperature, humidity), πραγματοποιείται ένα HTTP αίτημα, που περιέχει μία ερώτηση SPARQL στην υπηρεσία Οντολογίας με το οποίο ανακτάται για το δοθέν “attribute” (πχ. temperature) η τρέχουσα τιμή μέτρησης του αισθητήρα που το παρακολουθεί. Επιστρέφει όμως, τη σημασιολογική έννοια της μέτρησης και όχι το νούμερο, δηλαδή επιστρέφει το αποτέλεσμα του reasoning που γίνεται πάνω στις τιμές αυτές, το οποίο είναι μία περιγραφή κατανοητή από τον άνθρωπο που όμως έχει παραχθεί από την μηχανή. Επίσης, αν η εφαρμογή αφορά πόλεις, επιστρέφεται πάλι μέσω της υπηρεσίας Οντολογίας, η απάντηση της οντολογίας όσον αφορά τα “domains” που έχει επιλέξει να ρωτήσει ο χρήστης, η τιμή των οποίων παράγεται πάλι μέσω reasoning. Το τελευταίο αντικείμενο που επιστρέφεται είναι μια ερμηνεία της οντολογίας για τα αποτελέσματα αυτά.

Εφόσον η εκτέλεση των αλγορίθμων ολοκληρωθεί, ο διακομιστής Calculations επιστρέφει στην έξοδο του ένα προσαρμοσμένο κώδικα HTML/Javascript. Ο κώδικας αυτός είναι προσαρμοσμένος ώστε να σχεδιάζει γραφικά τα αποτελέσματα των αλγορίθμων σε διαγράμματα(όπου έχει ζητηθεί). Για την γραφική απεικόνιση ενός διαγράμματος γίνεται χρήση της βιβλιοθήκης “Google Charts”. Η βιβλιοθήκη αυτή παρέχει προκατασκευασμένες -javascript- μεθόδους (functions) για τον σχεδιασμό γραφημάτων διαφόρων ειδών (γραμμής, στήλης, πίτας κλπ). Για παράδειγμα, η μέθοδος για τον σχεδιασμό ενός διαγράμματος γραμμής (Line Chart) λαμβάνει ως όρισμα εισόδου ένα πίνακα (Array) δεδομένων και παράγει ως έξοδο στην οθόνη μία γραφική παράσταση κυματομορφής που απεικονίζει την διακύμανση των δεδομένων αυτών.

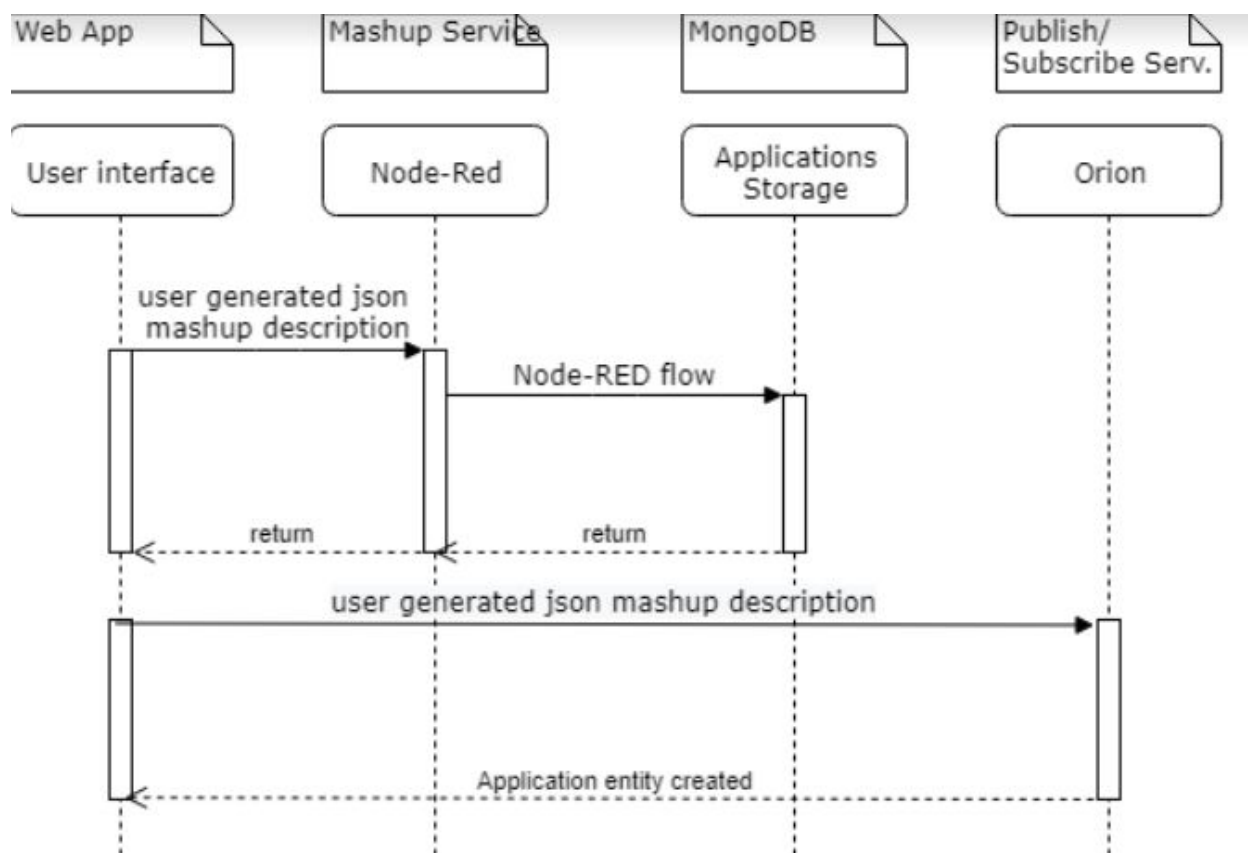
Οι HTTP μέθοδοι της υπηρεσίας Node-RED περιγράφονται στον παρακάτω πίνακα:

Πίνακας 4.6: HTTP μέθοδοι υπηρεσίας: Node Red

Μέθοδος	URL Μεθόδου	Headers	Σώμα αιτήματος	Περιγραφή Μεθόδου
POST	/flow	Node-RED-Deployment-Type: flows	Η σύνθεση κόμβων της εκτελέσιμης εφαρμογής σε περιγραφή JSON όπως είδαμε στην εικόνα 4.13	Δημιουργείται η εκτελέσιμη εφαρμογή όπως αυτή περιγράφεται στο σώμα της μεθόδου. Πλέον είναι διαθέσιμη προς εκτέλεση. Η μέθοδος επιστρέφει το μοναδικό αναγνωριστικό "appid" της δημιουργηθέντος εφαρμογής.
PUT	/flow/{appid}	Node-RED-Deployment-Type: flows	Η σύνθεση κόμβων της εκτελέσιμης εφαρμογής σε περιγραφή JSON όπως είδαμε στην εικόνα 4.13	Ενημερώνεται η υπάρχουσα εφαρμογή με μοναδικό αναγνωριστικό {appid}, όπως περιγράφεται στο σώμα της μεθόδου.
GET	/flow/{appid}			Γίνεται ανάκτηση της δομής JSON που περιγράφει την σύνθεση της εκτελέσιμης εφαρμογής με μοναδικό αναγνωριστικό {appid}.

Δημιουργία εφαρμογής

Στην εικόνα 4.17 φαίνεται το διάγραμμα ροής για την δημιουργία και αποθήκευση των εφαρμογών στο σύστημα.

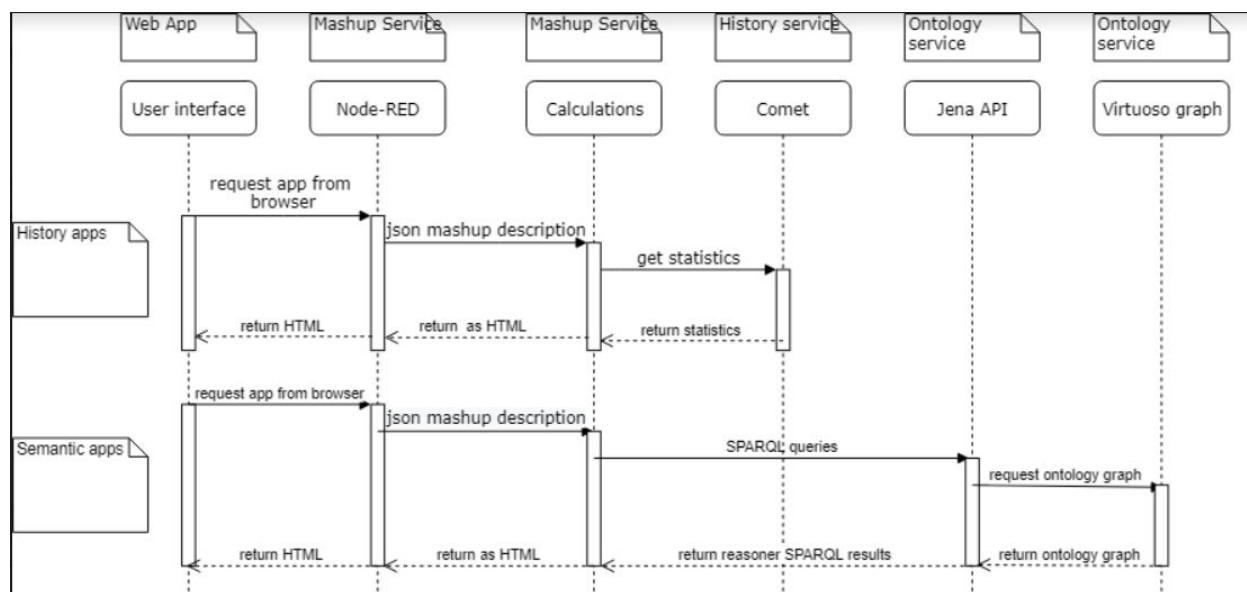


Εικόνα 4.17: Sequence diagram δημιουργίας εφαρμογής.

1. Ο χρήστης αρχικά επιλέγει τα mashup και την λειτουργία που θα έχει η εφαρμογή μέσω γραφικής διεπαφής. Η πληροφορία αυτή (εικόνες 4.15, 4.16) αποστέλλεται μετά στην υπηρεσία Node-RED μέσω ενός POST αιτήματος (φαίνεται στον πίνακα 4.6)
2. Στην υπηρεσία Node-RED δημιουργείται η ροή (flow - εικόνα 4.13) που αντιστοιχεί στην εκάστοτε εφαρμογή.
3. Η ροή που δημιουργήθηκε αποθηκεύεται στην βάση αποθήκευσης των ροών του Node-RED.
4. Ταυτόχρονα, αποστέλλεται και άλλο ένα POST αίτημα στην υπηρεσία Δημοσίευσης - Συνδρομής (Orion Context Broker) ώστε να δημιουργηθεί η οντότητα εφαρμογής (εικόνα 4.9) στο σύστημα και να δημοσιοποιηθεί. Πλέον, η εφαρμογή αποκτά ένα μοναδικό αναγνωριστικό και μπορεί να ανακτηθεί μέσω του συνδέσμου (URL) που οδηγεί σε αυτή.

Ανάκτηση - εκτέλεση εφαρμογής

Στην εικόνα 4.18 φαίνεται το διάγραμμα ροής για την ανάκτηση των εφαρμογών από τον χρήστη. Και οι ιστορικές και οι σημασιολογικές εφαρμογές έχουν ίδια διαδρομή εκτέλεσης μέχρι το σημείο που αναλύεται ο τύπος της εφαρμογής από την υπηρεσία Calculations.



Εικόνα 4.18: Sequence diagram ανάκτησης εφαρμογής.

1. Ο χρήστης μέσω ενός προγράμματος περιήγησης μπορεί να ανατρέξει σε μια υπάρχουσα εφαρμογή πληκτρολογώντας τον σύνδεσμό της (URL). Στη συνέχεια απεικονίζεται μία HTML απεικόνιση της εφαρμογής (βλ. Κεφάλαιο 4.2 Γραφική διεπαφή εφαρμογών). Ουσιαστικά στέλνεται ένα GET αίτημα στην υπηρεσία Node-RED και συγκεκριμένα στην ροή (flow) που αντιστοιχεί στην εφαρμογή.
2. Γίνεται επεξεργασία από τους κόμβους της ροής και στέλνεται ένα POST αίτημα στην υπηρεσία Calculations με τα δεδομένα που ζητάει ο χρήστης.
3. Στη συνέχεια, όπως έχει αναφερθεί, οι εφαρμογές μπορούν να είναι δύο ειδών: ιστορικού περιεχομένου και σημασιολογικού περιεχομένου. Η υπηρεσία Calculations αναλύει την περιγραφή mashup της εφαρμογής, αποφασίζει τι είδους εφαρμογή είναι και ποιες υπηρεσίες θα χρησιμοποιηθούν στη συνέχεια για την εκτέλεση της εφαρμογής.
4. Αν πρόκειται για ιστορική εφαρμογή στέλνονται τα κατάλληλα αιτήματα στην ιστορική υπηρεσία Comet.
5. Αν πρόκειται για σημασιολογική εφαρμογή στέλνονται τα κατάλληλα αιτήματα στην υπηρεσία οντολογίας. Στην υπηρεσία αυτή:
 - a. Αρχικά η υπηρεσία Jena API ζητάει από την υπηρεσία Virtuoso τον γράφο της οντολογίας.

- b. Μόλις πάρει τον γράφο εκτελείται συλλογισμός μέσω του Pellet reasoner, οι κανόνες SWRL παράγουν τα αποτελέσματά τους και εν τέλει με SPARQL ερωτήματα ανακτούνται τα αποτελέσματα των κανόνων SWRL της οντολογίας.
- 6. Μόλις επιστραφούν τα αποτελέσματα στην υπηρεσία Calculations, παίρνουν την HTML μορφή που θα απεικονιστεί στο παράθυρο περιήγησης του χρήστη. Η τελική μορφοποίηση αυτή επιστρέφει στην υπηρεσία Node-RED, η οποία την προωθεί στο παράθυρο περιήγησης του χρήστη.

4.2 Σύστημα διεπαφής χρηστών

Με τον όρο “Σύστημα διεπαφής χρηστών” αναφερόμαστε στην γραφική διεπαφή στην οποία ο χρήστης έχει πρόσβαση από το πρόγραμμα περιήγησής του, καθώς και τον κώδικα που τη δημιουργήσε και τρέχει τοπικά σε αυτήν. Προκειμένου να υλοποιηθούν οι γραφικές διεπαφές, χρησιμοποιήθηκαν γνωστές τεχνολογίες Ιστού: HTML , CSS , Javascript και AJAX .

Γραφική διεπαφή εισόδου

Όταν κάποιος χρήστης προσπαθήσει να μεταβεί για πρώτη φορά στην κεντρική σελίδα της Διαδικτυακής Εφαρμογής, ανακατευθύνεται αυτόματα στη σελίδα εισόδου του συστήματος, όπου καλείται να εισάγει τα στοιχεία (username και password) του λογαριασμού του, όπως φαίνεται στην εικόνα 4.1. Δεν επιτρέπεται στο χρήστη να έχει οποιαδήποτε αλληλεπίδραση με το σύστημα αν δεν περάσει επιτυχώς από αυτό το βήμα. Μόλις επιτευχθεί η είσοδος, δημιουργείται μια συνεδρία (session) με τα στοιχεία του χρήστη. Η παραπάνω διαδικασία δεν επαναλαμβάνεται όσο η συνεδρία αυτή είναι ενεργή. Οποιαδήποτε προσπάθεια του χρήστη για μετάβαση σε κάποια γραφική διεπαφή του συστήματος, χωρίς την ύπαρξη ενεργής συνεδρίας εισόδου στο πρόγραμμα περιήγησης, έχει ως αποτέλεσμα την επιστροφή του χρήστη στη σελίδα εισόδου.



Εικόνα 4.19: Σελίδα εισόδου του συστήματος

Γραφική διεπαφή χρήστη

Η γραφική διεπαφή που αναπτύχθηκε αποτελεί αφετηρία για το χρήστη του συστήματος (Διαδικτυακή Εφαρμογή). Στη σελίδα αυτή υπάρχει ένα κεντρικό μενού που περιλαμβάνει όλες τις λειτουργίες που περιγράφονται στο Web Thing Model της W3C, όπως υλοποιήθηκαν από τον συμφοιτητή Αιμίλιο Τζαβάρα στην εργασία του[2], καθώς και κάποιες που προστέθηκαν στα πλαίσια της παρούσας εργασίας.

Στα πλαίσια της δημιουργίας εφαρμογών αναπτύχθηκε ένα γραφικό μενού όπου προσφέρει όλες τις πιθανές επιλογές που μπορούν να προστεθούν σε ένα mashup.

The screenshot shows the 'Create application' interface of the Mashup Service. At the top, there is a blue header with the text 'Mashup Service' and 'Create application'. Below the header, the form is organized into several sections. The first section is 'Select Application domain:', which contains a dropdown menu labeled 'Select Domain'. The second section is 'Application name', which contains a text input field with the placeholder text 'e.g. myapp'. The third section is 'Select City:', which contains a dropdown menu labeled 'Rethymno'. The fourth section is 'Select attributes:', which contains a list of attributes with checkboxes: Accomodation, Activity, Garment, MartialArt, Nightlife, Place, Relaxation, Road, Shoe, and Sport. The fifth section is 'Select recommendation to apply semantics:', which is currently empty. The sixth section is 'Select calculation to execute', which contains a dropdown menu. In the bottom right corner, there is a watermark that says 'Activate Windows Go to Settings to activate Windows.'

Εικόνα 4.20: Σελίδα δημιουργίας εφαρμογών.

Σαν πρώτη επιλογή εμφανίζεται το πεδίο στο οποίο θα αναφέρεται η εφαρμογή δηλαδή σε πόλη ή σπίτι. Ανάλογα με την επιλογή αυτή αλλάζει και η μορφοποίηση της σελίδας, διότι αλλάζουν οι διαθέσιμες επιλογές.

Select Application domain:

City Weather and Recommendations ▼

Application name

e.g. myapp

Select City: **Rethymno** ▼

Select attributes:

- ☐ Atmosphere temperature
- ☐ Atmosphere humidity
- ☐ Wind speed
- ☐ Cloud coverage
- ☐ Precipitation
- ☐ Atmospheric pressure

Select recommendation to apply semantics:

- ☐ Accomodation
- ☐ Activity
- ☐ Garment
- ☐ MartialArt
- ☐ Nightlife
- ☐ Place
- ☐ Relaxation
- ☐ Road
- ☐ Shoe
- ☐ Sport
- ☐ Transport
- ☐ WaterSport
- ☐ Weather

Select calculation to execute ▼

Εικόνα 4.21: Επιλογές δημιουργίας εφαρμογών για πόλη.

Στην εικόνα 4.21 φαίνονται οι διαθέσιμες επιλογές για μία εφαρμογή πόλης. Αρχικά επιλέγεται η συγκεκριμένη πόλη που μας ενδιαφέρει, στη συνέχεια τα μετρήσιμα χαρακτηριστικά που μας ενδιαφέρουν, μετά τον τομέα της καθημερινότητας για τον οποίο θα θέλαμε συμβουλή, σύμφωνα με την επιλογή των μετρήσιμων χαρακτηριστικών, και τελευταία επιλογή είναι οι διαφορετικές επιλογές mashup που υποστηρίζει το σύστημα. (Διάγραμμα γραμμής/στήλης, με την μέση/μέγιστη/ελάχιστη μέτρηση των επιλεγμένων αισθητήρων για τις τελευταίες 24 ώρες/7 μέρες κλπ). Ο χρήστης επιλέγει μία από τις διαθέσιμες επιλογές.

The screenshot shows a web interface titled 'Home Automations'. At the top, there is a dropdown menu labeled 'Home Automations'. Below it is a text input field for 'Application name' with the placeholder text 'e.g: myapp'. To the left, under 'Select Home:', there is a button labeled 'WindowHouse'. To the right, under 'Select attributes:', there are four checkboxes: 'Room temperature', 'Room humidity', 'Luminocity', and 'Presence'. Further right is a dropdown menu labeled 'Select calculation to execute'. Below these options is a button labeled 'See actuations'. At the bottom, there is a section titled 'Actuations enabled in building:' with two buttons: 'Add Mashups' and 'Generate Application'.

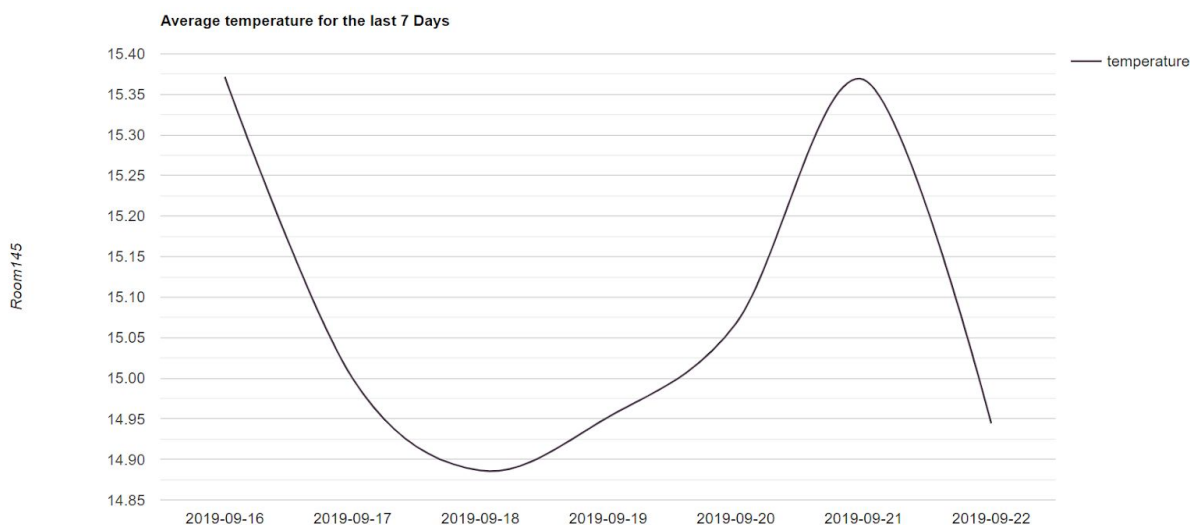
Εικόνα 4.22: Επιλογές δημιουργίας εφαρμογών για σπίτι.

Αν ο χρήστης επιλέξει να δημιουργήσει εφαρμογή για σπίτι οι διαθέσιμες επιλογές αλλάζουν. Πάλι σαν πρώτη επιλογή εμφανίζεται το σπίτι που τον ενδιαφέρει, στη συνέχεια η μετρήσιμη ιδιότητα όπως προηγουμένως και τέλος μόνο οι διαφορετικές επιλογές mashup που υποστηρίζει το σύστημα. (Διάγραμμα γραμμής/στήλης, με την μέση/μέγιστη/ελάχιστη μέτρηση των επιλεγμένων αισθητήρων για τις τελευταίες 24 ώρες/7 μέρες κλπ). Πατώντας “Add” προστίθεται το mashup των αισθητήρων στην εφαρμογή όπως αυτό διαμορφώθηκε από το χρήστη.

Γραφική διεπαφή εφαρμογών

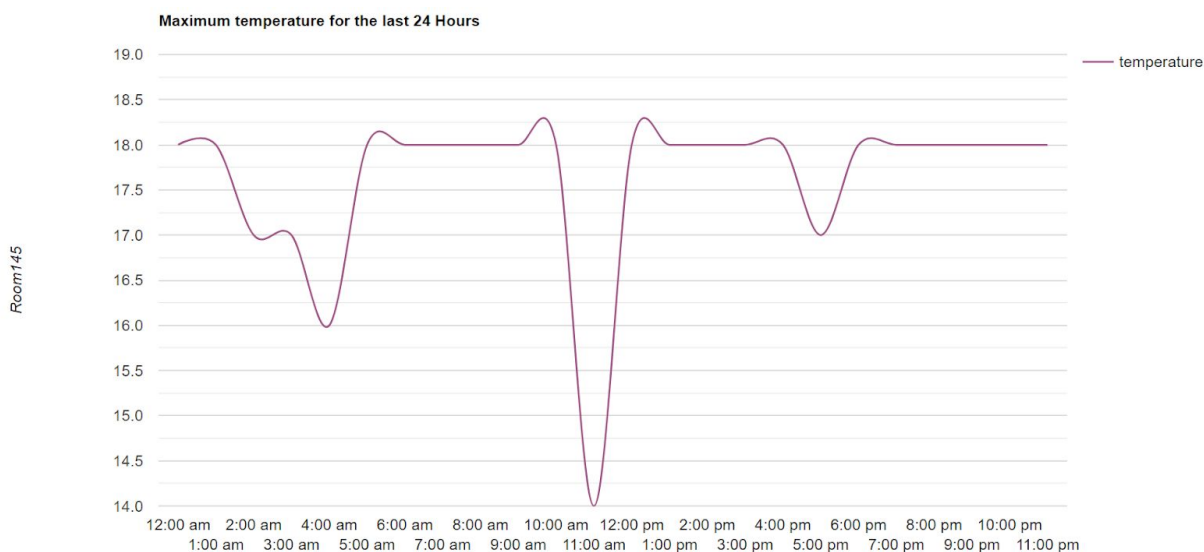
Παρακάτω παρουσιάζεται η μορφή που μπορεί να έχει μία εφαρμογή ανάλογα με τις επιλογές που έχει κάνει ο χρήστης.

Στην εικόνα 4.23 φαίνεται η σελίδα μιας ιστορικής εφαρμογής. Η συγκεκριμένη εφαρμογή αποτελεί μια ιστορική εφαρμογή η οποία παρουσιάζει την μέση θερμοκρασία ανά ημέρα για τις τελευταίες 7 μέρες για το σπίτι με αναγνωριστικό “Room145”.



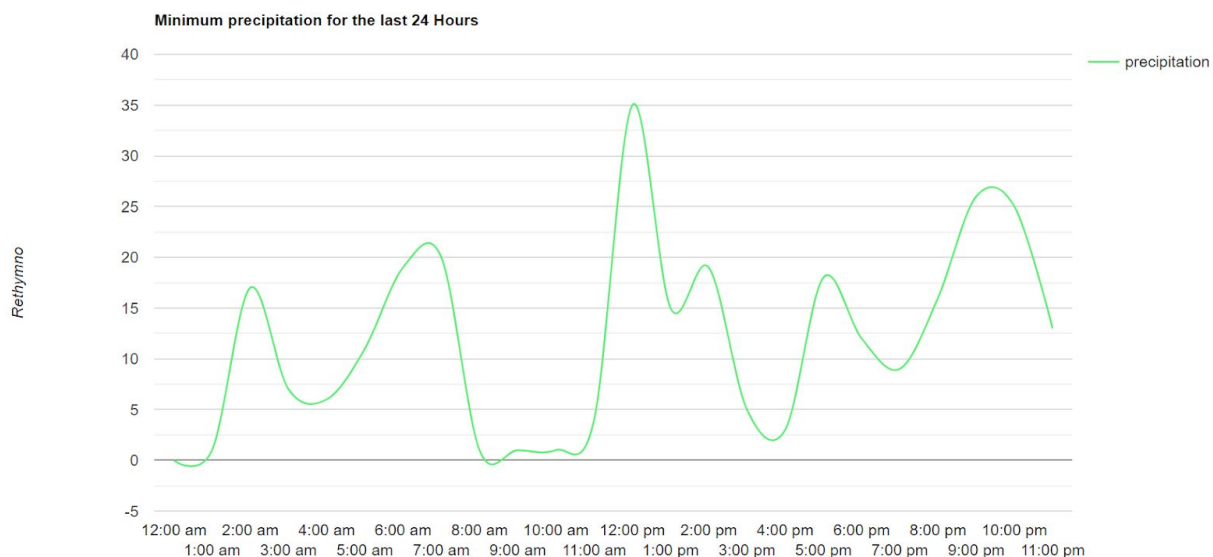
Εικόνα 4.23: Διάγραμμα ιστορικής εφαρμογής που αφορά μέση τιμή θερμοκρασίας ανά ημέρα για 7 μέρες σε ένα σπίτι.

Στην εικόνα 4.24 αντίστοιχα φαίνεται η HTML απεικόνιση πάλι μιας ιστορικής εφαρμογής η οποία όμως δείχνει την μέγιστη θερμοκρασία ανά ώρα για τις τελευταίες 24 ώρες του σπιτιού με αναγνωριστικό “Room145”.



Εικόνα 4.24: Διάγραμμα ιστορικής εφαρμογής που αφορά μέγιστη τιμή θερμοκρασίας ανά ώρα για 24 ώρες σε ένα σπίτι.

Στην εικόνα 4.25 φαίνεται πάλι μια ιστορική εφαρμογή η οποία παρουσιάζει την ελάχιστη βροχόπτωση ανά ώρα για τις τελευταίες 24 ώρες στην πόλη με αναγνωριστικό “Rethymno”.



Εικόνα 4.25: Διάγραμμα ιστορικής εφαρμογής που αφορά ελάχιστη τιμή βροχόπτωσης ανά ώρα για 24 ώρες σε μια πόλη.

room245test

Live state

AboveNormalTemperature
AverageHumidity

Recommendations based on current state

SwitchOnAirConditioning
SwitchOffHeating
SwitchOffDehydrator

Further assumptions

ExpectingHighElectricityConsumption

Εικόνα 4.26: Διάγραμμα σημασιολογικής εφαρμογής που αφορά LIVE τιμές ενός σπιτιού(θερμοκρασία, υγρασία).

Στην εικόνα 4.26 φαίνεται μια εφαρμογή σπιτιού. Οι εφαρμογές που αφορούν LIVE τιμές διαφέρουν ως προς την σημασιολογική δυνατότητά τους καθώς ενεργοποιείται reasoner. Η σημασιολογικές εφαρμογές επιστρέφουν:

1. Κάτω από την στήλη Live state: Την τρέχουσα τιμή των αισθητήρων, ερμηνευμένες όμως μέσω της οντολογίας σε μια περιγραφή προσιτή στον άνθρωπο. Η ανάκτηση γίνεται με το κατάλληλο SPARQL ερώτημα πάνω στην οντολογία. Ένα τέτοιο ερώτημα μπορεί να είναι το παρακατω:

```
SELECT ?p WHERE {ex:Room145 home:hasTemperatureState ?p}
```

Το οποίο επιστρέφει την τιμή “AboveNormalTemperature”, όπως φαίνεται στην εφαρμογή. Η τιμή “AverageHumidity” αντίστοιχα επιτρέφεται μέσω του SPARQL ερωτήματος:

```
SELECT ?p WHERE {ex:Room145 home:hasHumidityState ?p}
```

Ουσιαστικά, τα αποτελέσματα αυτά είναι τα παράγωγα των SWRL κανόνων πάνω στην οντολογία, οι οποίοι κανόνες ελέγχουν την τιμή των μετρήσεων των αισθητήρων, γίνεται συλλογισμός με Pellet (reasoning) και επιστρέφουν την συνθήκη που επικρατεί με την περιγραφή όμως της δευτερεύουσας οντολογίας συνθηκών σπιτιού (βλ. 3.5).

2. Κάτω από την στήλη Recommendations based on current state: Τις ενεργοποιήσεις οι οποίες πρέπει να γίνουν σύμφωνα με τα προηγούμενα δεδομένα (πχ. Να λειτουργήσει AC καθώς κάνει ζέστη, να κλείσει ο αφυγραντήρας εφόσον ήταν ανοιχτός καθώς έχουμε χαμηλή υγρασία). Αυτά τα αποτελέσματα ανακτώνται πάλι με κατάλληλα SPARQL ερωτήματα:

```
SELECT ?p WHERE {ex:AboveNormalTemperature home:hasRecommendation ?p}
```

```
SELECT ?p WHERE {ex:AverageHumidity home:hasRecommendation ?p}
```

Εδώ βλέπουμε ότι χρησιμοποιούμε τα προηγούμενα αποτέλεσμα για να πάρουμε τους αυτοματισμούς που αντιστοιχούν στο καθένα και συνδέονται με το Object Property “home:hasRecommendation”.

3. Κάτω από την στήλη Further Assumptions: Τις ερμηνεύσεις που κάνει η οντολογία σύμφωνα με τις προηγούμενες ενεργοποιήσεις καθώς μπορεί να είναι χρήσιμες προς τον χρήστη (πχ. Ανοικτό AC άρα μεγάλη κατανάλωση ρεύματος). Εδώ το SPARQL ερώτημα είναι:

```
SELECT ?o WHERE { ex:Room145 sosa:assumes ?o }
```

Αυτό που ρωτάμε και θέλουμε να μάθουμε είναι αν υπάρχουν επιπλέον χρήσιμες πληροφορίες που τυχόν έχει συμπεράνει η οντολογία. Στη περίπτωση αυτή στην οντολογία είναι ορισμένος ο κανόνας SWRL ο οποίος λέει:

```
sosa:Home(?home) ^ home:hasActuationState(?home, ex:SwitchOnAirConditioning) ^
home:hasActuationState(?home, ex:SwitchOnLight) -> sosa:assumes(?home,
ex:ExpectingHighElectricityConsumption)
```

Λόγω του ότι στο σπίτι επικρατούν οι συνθήκες “SwitchOnAirConditioning” και “SwitchOnLight” τότε η τιμή που επιστρέφεται είναι “ExpectingHighElectricityConsumption”.

chanitetest1

Live state	Recommendations based on current state	Further assumptions
BelowNormalTemperature DryHumidity Sunny	Short Hot Swimsuit SunGlass	ManyHomesWillTurnOnHeat ExpectingBadWeather

Εικόνα 4.27: Διάγραμμα εφαρμογής που αφορά LIVE τιμές μιας πόλης (θερμοκρασία, υγρασία).

Σε μία εφαρμογή πόλης, όπως της εικόνας 4.27, επιστρέφονται τα ίδια αποτελέσματα αλλά η στήλη Recommendations based on current state περιέχει τις συμβουλές την οντολογίας για τα πεδία της καθημερινότητας που επέλεξε να λάβει ο χρήστης, σύμφωνα με τις τρέχουσες καιρικές συνθήκες. Δηλαδή:

1. Κάτω από την στήλη Live state: Την τρέχουσα τιμή των αισθητήρων, ερμηνευμένες όμως μέσω της οντολογίας σε μια περιγραφή προσιτή στον άνθρωπο. Η ανάκτηση γίνεται με το κατάλληλο SPARQL ερώτημα πάνω στην οντολογία. Ένα τέτοιο ερώτημα μπορεί να είναι το παρακάτω:

```
SELECT ?p WHERE {ex:Chania tourism:hasTemperature ?p}
```

Το οποίο επιστρέφει την τιμή “BelowNormalTemperature”, όπως φαίνεται στην εφαρμογή. Η τιμή “DryHumidity” αντίστοιχα επιτρέφεται μέσω του SPARQL ερωτήματος:

```
SELECT ?p WHERE {ex:Chania tourism:hasHumidity ?p}
```

Η τιμή “Sunny” αντίστοιχα επιτρέφεται μέσω του SPARQL ερωτήματος:

```
SELECT ?p WHERE {ex:Chania tourism:hasCloudCoverage ?p}
```

Όπως και στις εφαρμογές σπιτιών, τα αποτελέσματα αυτά είναι τα παράγωγα των SWRL κανόνων πάνω στην οντολογία, οι οποίοι κανόνες ελέγχουν την τιμή των μετρήσεων των αισθητήρων, γίνεται συλλογισμός με Pellet (reasoning) και επιστρέφουν την συνθήκη που επικρατεί με την περιγραφή όμως της δευτερεύουσας οντολογίας συνθηκών σπιτιού (βλ. 3.5).

2. Κάτω από την στήλη Recommendations based on current state: Τις συμβουλές της οντολογίας σύμφωνα με τα προηγούμενα δεδομένα (πχ. τι ρούχο είναι κατάλληλο ανάλογα με τον καιρό ή τι δραστηριότητα μπορεί να γίνει). Αυτά τα αποτελέσματα ανακτώνται πάλι με κατάλληλα SPARQL ερωτήματα:

```
SELECT ?p WHERE {ex:Sunny tourism:isRecommendation ?p. ?p rdf:type tourism:Garment}
```

Το οποίο επιστρέφει τις τιμές: “Short”, “Hat”, “Swimsuit”, “SunGlass”.

3. Κάτω από την στήλη Further Assumptions: Τις ερμηνεύσεις που κάνει η οντολογία σύμφωνα με τις προηγούμενες ενεργοποιήσεις καθώς μπορεί να είναι χρήσιμες προς τον χρήστη. Εδώ το SPARQL ερώτημα είναι:

```
SELECT ?o WHERE { ex:Chania sosa:assumes ?o }
```

Αυτό που ρωτάμε και θέλουμε να μάθουμε είναι αν υπάρχουν επιπλέον χρήσιμες πληροφορίες που τυχόν έχει συμπεράνει η οντολογία. Στη περίπτωση αυτή στην οντολογία είναι ορισμένος ο κανόνας SWRL ο οποίος λέει:

```
geo:SpatialThing(?city) ^ tourism:hasWeather(?city, ex:BelowRoomTemperature) -> sosa:assumes(?city, ex:ManyHomesWillTurnOnHeat)
```

Λόγω του ότι στο σπίτι επικρατεί η συνθήκη “BelowRoomTemperature” τότε η τιμή που επιστρέφεται είναι “ManyHomesWillTurnOnHeat”. Αντίστοιχα λόγω του ότι επικρατεί η συνθήκη “LowPressure” μέσω του ερωτήματος:

```
geo:SpatialThing(?city) ^ tourism:hasPressure(?city, ex:LowPressure) -> sosa:assumes(?city, ex:ExpectingBadWeather)
```

Επιστρέφει η τιμή “ExpectingBadWeather”.

Κεφάλαιο 5

Ανάλυση απόδοσης

Η υποδομή του συστήματος που υλοποιήθηκε στην παρούσα εργασία στεγάζεται στο περιβάλλον Intellicloud του Πολυτεχνείου Κρήτης και απαρτίζεται από 5 εικονικές μηχανές (virtual machines). Στην **πρώτη εικονική μηχανή (1)** εκτελούνται:

- Η Υπηρεσία Διαχείρισης Συμβάντων και Συνδρομών (Orion Context Broker).
- Η υπηρεσία Cygnus, που χρησιμοποιείται με σκοπό την αποθήκευση ιστορικού ακατέργαστων / συγκεντρωτικών μετρήσεων χρονικής σειράς στην ιστορική βάση δεδομένων.
- Η Υπηρεσία Οντολογίας και συγκεκριμένα το Jena API, που φιλοξενείται στον Apache Tomcat (Tomcat Server).

Τα τεχνικά χαρακτηριστικά της παραπάνω εικονικής μηχανής (η μόνη με 2 CPU cores) είναι τα εξής:

CPU	2 CPU cores, x86_64@2.8GHz
Memory	4096 MB
HDD	40GB
OS	CentOS 7

Στη **δεύτερη εικονική μηχανή (2)** εκτελείται η υπηρεσία Διαχείρισης Συσκευών (Device Management) Backend IDAS. Τα τεχνικά χαρακτηριστικά της παραπάνω εικονικής μηχανής είναι τα εξής:

CPU	1 CPU core, x86_64@2.8GHz
Memory	2048 MB
HDD	20GB
OS	CentOS 7

Στην τρίτη εικονική μηχανή (3) εκτελούνται:

- Η Υπηρεσία Υλοποίησης του Μοντέλου του Ιστού των Πραγμάτων (Web Things Model Service).
- Η υπηρεσία Λογικής Εφαρμογής (Application Logic).
- Το Mashup Service.

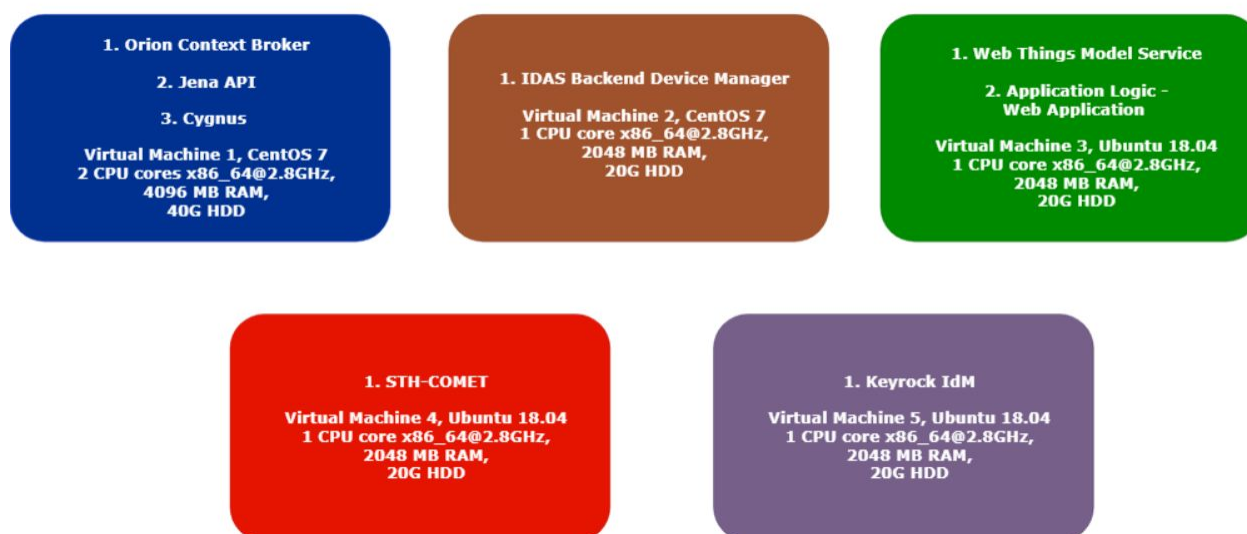
Στην **τέταρτη εικονική μηχανή (4)** εκτελείται η υπηρεσία FIWARE-COMET που χρησιμοποιείται για την ανάκτηση ιστορικών δεδομένων χρονικής σειράς από την ιστορική βάση δεδομένων.

Στην **πέμπτη εικονική μηχανή (5)** εκτελείται η υπηρεσία Keyrock IDM, που είναι η υπηρεσία ταυτοποίησης και εξουσιοδότησης χρηστών. Τα τεχνικά χαρακτηριστικά των παραπάνω εικονικών μηχανών είναι τα εξής:

CPU	1 CPU core, x86_64@2.8GHz
Memory	2048 MB
HDD	20GB
OS	Ubuntu 18.04

Ο στόχος της πειραματικής διαδικασίας είναι να αποδειχθεί ότι το σύστημα που υλοποιήθηκε είναι αποδοτικό σε πραγματικές συνθήκες. Για αυτό το λόγο, χρησιμοποιήθηκε το εργαλείο Apache Bench²⁸, το οποίο έχει τη δυνατότητα δημιουργίας αρκετών ταυτόχρονων αιτημάτων (concurrent requests) και παράλληλα υπολογισμού σημαντικών παραμέτρων (όπως ο χρόνος απόκρισης για συγκεκριμένο αριθμό αιτημάτων) σε ένα διακομιστή HTTP. Χάρη στο Apache Bench, μπορούμε να δημιουργήσουμε συνθήκες φόρτου σε κάθε υπηρεσία του συστήματος ξεχωριστά, ενώ παράλληλα μας δίνεται η δυνατότητα να καθορίσουμε τον αριθμό των αιτημάτων που θα πρέπει να εξυπηρετήσει η κάθε υπηρεσία, καθώς και τον αριθμό αυτών που θα εκτελεστούν ταυτόχρονα. Κάθε εντολή του Apache Bench πρέπει να περιέχει την IP που πρόκειται να “χτυπήσει”, το συνολικό αριθμό των αιτημάτων, τον αριθμό των ταυτόχρονων αιτημάτων, αλλά και την είσοδο που θα έχει η εκάστοτε εφαρμογή της υπηρεσίας. Τα τεχνικά χαρακτηριστικά συνολικά των 5 εικονικών μηχανών φαίνονται συγκεντρωμένα στην παρακάτω εικόνα:

²⁸ <https://httpd.apache.org/docs/2.4/programs/ab.html>



Εικόνα 5.1: Υποδομή Συστήματος.

Προκειμένου να παρακολουθήσουμε τους φυσικούς πόρους των εικονικών μηχανών κατά τη διάρκεια των πειραμάτων, αξιοποιήσαμε το εργαλείο HTOP. Το πρόγραμμα αυτό εκτελείται κατευθείαν από τη γραμμή εντολών και μας ενημερώνει σχετικά με την κατανάλωση πόρων ανά διεργασία, καθώς και για τη συνολική κατανάλωση πόρων από το σύστημα σε πραγματικό χρόνο.

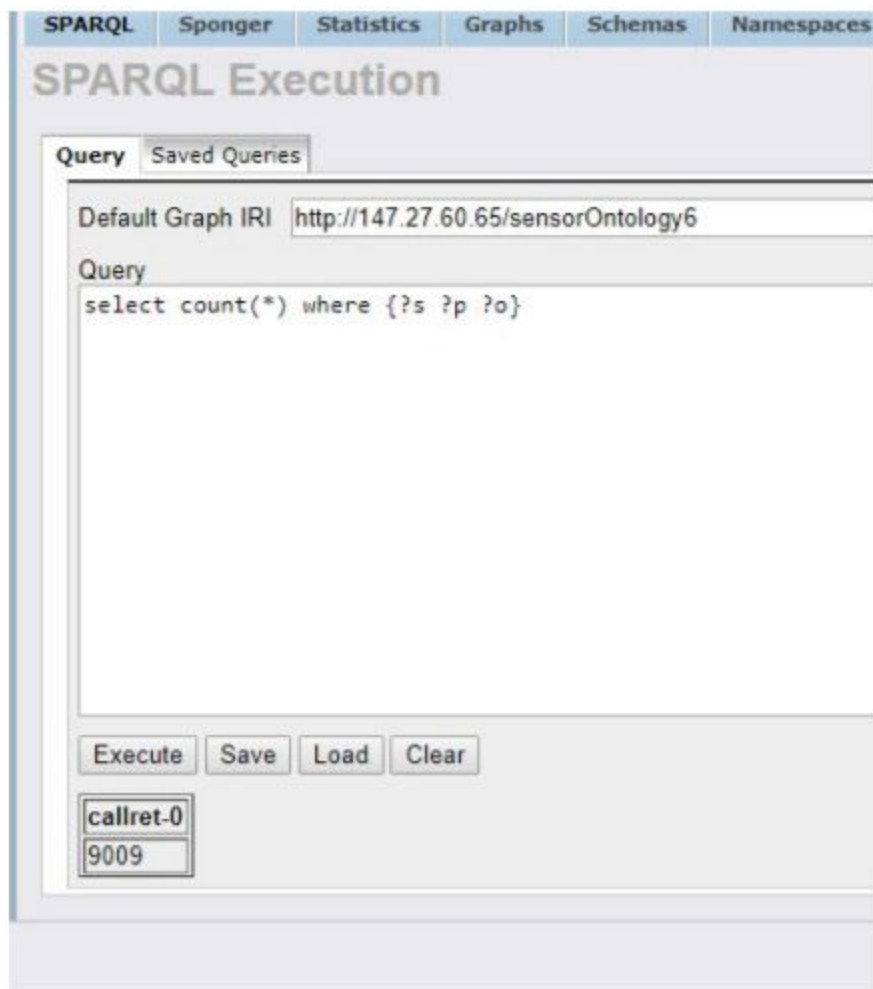
Κάθε πείραμα πραγματοποιείται στο ίδιο μοτίβο, δηλαδή περιλαμβάνει 1000 αιτήματα με σκοπό να εξεταστεί η λειτουργία της εκάστοτε υπηρεσίας του συστήματος. Όλα τα πειράματα επαναλαμβάνονται με χρήση διαφορετικού αριθμού ταυτόχρονων αιτημάτων κάθε φορά. Οι μετρήσεις δείχνουν το μέσο χρόνο εξυπηρέτησης ανά αίτημα και διακρίνονται σε κατηγορίες ανάλογα με τον ταυτοχρονισμό που χρησιμοποιήθηκε για να σταλούν τα αιτήματα. Συγκεκριμένα, τα αιτήματα στέλνονται ανά ένα, ανά πενήντα, ανά εκατό, ανά εκατόν πενήντα και ανά διακόσια. Με ταυτοχρονισμό ανά ένα σημαίνει ότι τα αιτήματα στέλνονται το ένα μετά το άλλο, διαδοχικά, με ταυτοχρονισμό ανά πενήντα σημαίνει ότι στέλνονται διαδοχικά σε ομάδες των πενήντα αιτημάτων, με ταυτοχρονισμό ανά εκατό σημαίνει ότι στέλνονται διαδοχικά σε ομάδες των εκατό αιτημάτων, και ούτω καθεξής. Για κάθε κατηγορία ταυτοχρονισμού εξετάζεται - εκτός από το μέσο χρόνο εξυπηρέτησης ανά αίτημα - και η κατανάλωση σε CPU και μνήμη RAM σε μορφή ποσοστού. Τα αποτελέσματα αυτά έχουν καταγραφεί τόσο σε πίνακες όσο και διαγράμματα, τα οποία φαίνονται στη συνέχεια του Κεφαλαίου.

Για να δημιουργήσουμε συνθήκες φόρτου στο σύστημα, φτιάξαμε ένα υποθετικό σενάριο με εικονικές οντότητες 1050 σπιτιών, οι οποίες “φιλοξενούν” αισθητήρες, και βρίσκονται σε μία συγκεκριμένη πόλη. Άρα πρόκειται για μια πόλη, στην οποία έχουμε 1050 σπίτια, με 3 είδη αισθητήρων (υγρασίας, θερμοκρασίας, φωτεινότητας) και σε όλα

τα σπίτια 1 αισθητήρα από κάθε είδος. Οι οντότητες αυτές αποθηκεύτηκαν τόσο στην οντολογία όσο και στη βάση δεδομένων του Orion Context Broker, που φιλοξενεί τις πιο πρόσφατες τιμές.

Λόγω του γεγονότος ότι ο αποθηκευτικός χώρος ενός γράφου δεν επαρκούσε για την αποθήκευση όλων των δεδομένων (μετρήσεις και άλλες πληροφορίες) και των 1050 σπιτιών, χωρίσαμε την οντολογία σε 7 διαφορετικούς γράφους, οι οποίοι όμως συντελούν μια ενιαία οντολογία που αφορά μία πόλη (υποθέσαμε την Αθήνα).

Στην Εικόνα 5.2 φαίνεται ο υπολογισμός (πάλι με χρήση της γλώσσας SPARQL) των συνολικών τριπλετών που φιλοξενεί ο κάθε γράφος της οντολογίας, συγκεκριμένα 9009 τριπλέτες. Εφόσον οι γράφοι είναι όλοι ίδιοι σε μέγεθος, συνολικά στους 7 γράφους έχουμε: $7 \times 9009 = 63063$ τριπλέτες δεδομένων. Τώρα υπάρχει διαφοροποίηση όσον αφορά τις τριπλέτες που ανήκουν στην οντολογία και την περιγράφουν γενικά και τις τριπλέτες οι οποίες είναι στιγμιότυπα (instances) της οντολογίας, δηλαδή πραγματικά δεδομένα. Σε κάθε γράφο έχουμε 3434 τριπλέτες οντολογίας και 5575 τριπλέτες που αναφέρονται σε στιγμιότυπα.



Εικόνα 5.2: Στιγμιότυπο όπου φαίνεται ο υπολογισμός των τριπλετών του ενός γράφου της οντολογίας. στο πεδίο εκτέλεσης ερωτημάτων SPARQL που παρέχει το Virtuoso.

5.1 Πείραμα 1

Σενάριο: Ένας χρήστης μέσω γραφικής διεπαφής θέλει να δει την ελάχιστη τιμή της θερμοκρασίας ανά ώρα για 24 ώρες σε ένα σπίτι. Για τα πλαίσια του πειράματος, δημιουργήθηκε μία εφαρμογή με όνομα “room145mintemp”. Στην εικόνα 5.3 φαίνεται η JSON περιγραφή της εφαρμογής όπως είναι αποθηκευμένη στην υπηρεσία Node-RED.

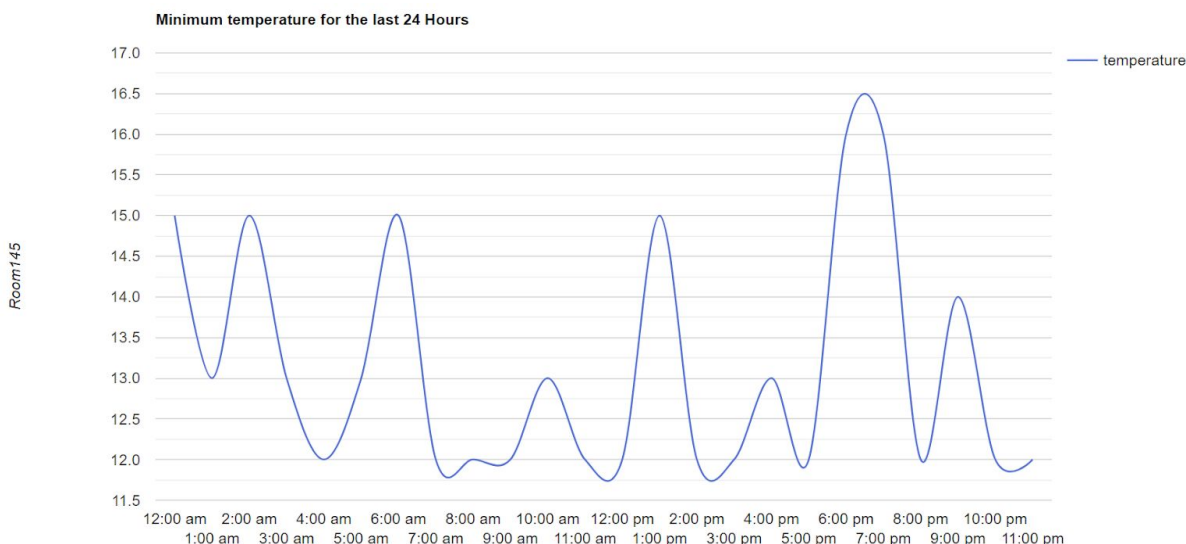
```
{
  "appname": "room145mintemp",
  "appscope": "home",
  "info": [
    {
      "attributes": [
        "temperature"
      ],
      "city": "Room145",
      "domains": [],
      "func": "MIN_24hr"
    }
  ]
}
```

Εικόνα 5.3: JSON περιγραφή εφαρμογής “chaniaevery”.

Στην εικόνα φαίνονται τα εξής πεδία:

- “Appname”: Όνομα εφαρμογής.
- “Appscope”: Ο τύπος της εφαρμογής. Με την τιμή “home” δείχνει ότι αναφέρεται σε εφαρμογή σπιτιού.
- “Attributes”: Οι μετρήσιμες ιδιότητες για τις οποίες ο χρήστης ζητάει πληροφορία. Εδώ ζητήθηκε η θερμοκρασία.
- “City”: Με την τιμή “Room145” δείχνει το μοναδικό αναγνωριστικό του σπιτιού στο οποίο αναφέρεται η εφαρμογή.
- “Domains”: Είναι χωρίς τιμή καθώς αυτό το πεδίο χρησιμοποιείται μόνο σε σημασιολογικές εφαρμογές.
- “Func”: Έχει τιμή MIN_24hr άρα πρόκειται για ιστορική εφαρμογή και ο αλγόριθμος που θα εκτελεστεί έχει αυτή την κωδική λέξη.

Εφόσον η εφαρμογή υπάρχει, μπορεί να του παρουσιάσει την γραφική απεικόνιση των τιμών όπως φαίνεται στην εικόνα 5.3.

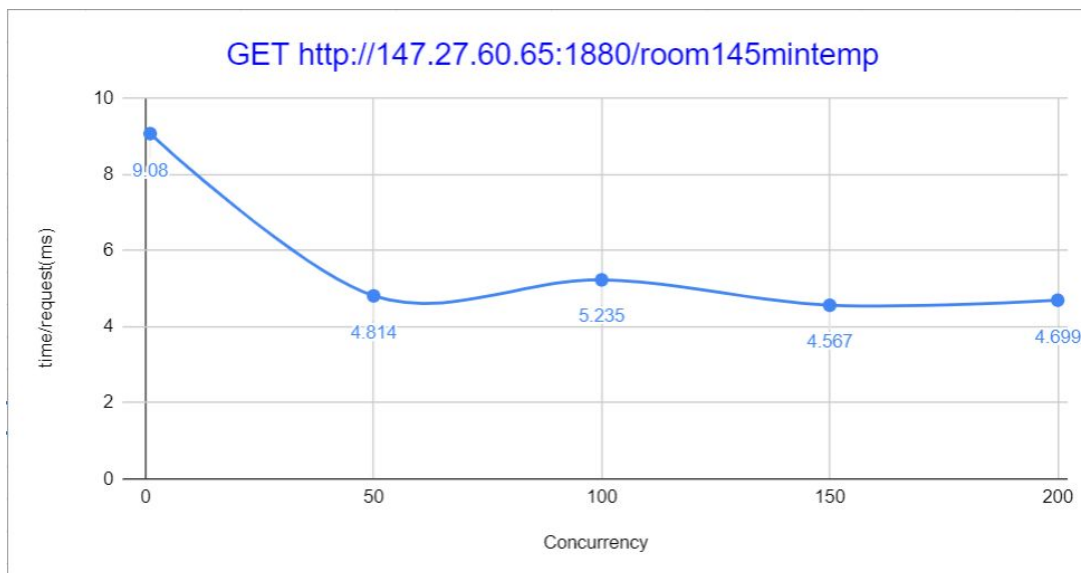


Εικόνα 5.3: Απεικόνιση εφαρμογής “room145mintemp” που βλέπει ο χρήστης.

Υπηρεσίες: Αυτή η εφαρμογή αποτελείται μόνο από ιστορικά δεδομένα άρα η απόδοσή της εξαρτάται από την απόδοση της Ιστορική υπηρεσίας Comet και της υπηρεσίας Node-red, καθώς εκεί είναι αποθηκευμένη η εφαρμογή.

Αίτημα REST: GET <http://147.27.60.65:1880/room145mintemp>

Αποτελέσματα και σχολιασμός: Τα αποτελέσματα για το μέσο χρόνο εκτέλεσης και την κατανάλωση πόρων (CPU και μνήμης RAM) φαίνονται στο Σχήμα 5.4 και αναγράφονται στον Πίνακα 5.1. Παρατηρούμε ότι πάλι ο μέγιστος από τους μέσους χρόνους εκτέλεσης παρατηρείται στην πρώτη περίπτωση, όπου τα αιτήματα στέλνονται ανά ένα (9,08 ms), ενώ ο ελάχιστος από τους μέσους χρόνους εκτέλεσης παρατηρείται στην περίπτωση όπου τα αιτήματα στέλνονται ανά 150 (4,567 ms), με μικρή διαφορά από τις περιπτώσεις των 50, 100 και 200. Παρατηρούμε ότι και στο παρόν πείραμα ο μέσος χρόνος απόκρισης, καθώς και η κατανάλωση της CPU και της RAM, βρίσκονται σε πολύ χαμηλά επίπεδα. Ένα βασικό παράγοντα για αυτά τα αποτελέσματα μπορούμε να θεωρήσουμε το γεγονός ότι πρόκειται για ερώτημα που πραγματοποιεί αναζήτηση σε μία υπηρεσία (σε μία μόνο εικονική μηχανή), για συγκεκριμένη μέτρηση αισθητήρα. Ακόμα, παρόλο που πρόκειται για την ιστορική βάση δεδομένων (που διατηρεί ιστορικό μετρήσεων χρονικής σειράς), σε αυτήν αποθηκεύονται μόνο οι μετρήσεις (οντότητες τύπου “Observation”), επομένως δεν έχει επιβαρυνθεί με πολλά επιπλέον δεδομένα όπως ο Orion Context Broker και η οντολογία. Επίσης, όλες οι τιμές που επιστρέφει ο Comet πρόκειται και προυπολογισμένα, cached δεδομένα και δεν τα υπολογίζει την στιγμή του αιτήματος.



Εικόνα 5.4: Χρόνος απόκρισης σε 1000 ερωτήματα ιστορικής εφαρμογής (Πείραμα 1).

concurrency	CPU Load	RAM Load	Time/request (ms)
1	20.60%	3.80%	9.08
50	37.90%	3.20%	4.814
100	41.80%	2.90%	5.235
150	38.90%	3.80%	4.567
200	34.20%	3.40%	4.699

Πίνακας 5.1: Χρόνος απόκρισης, κατανάλωση CPU και κατανάλωση RAM σε 1000 ερωτήματα ιστορικής εφαρμογής (Πείραμα 1).

5.2 Πείραμα 2

Σενάριο: Ένας χρήστης μέσω γραφικής διεπαφής θέλει να δει τις τρέχουσες τιμές κάποιων χαρακτηριστικών για μια πόλη, καθώς και τις συμβουλές της οντολογίας για κάποιους τομείς που τον ενδιαφέρουν. Για τα πλαίσια του πειράματος, δημιουργήθηκε μία εφαρμογή με όνομα “chaniaevery”, στην οποία έχουν επιλεγθεί για χάρη βαρύτητας, όλα τα μετρήσιμα χαρακτηριστικά και 3 τομείς της καθημερινότητας, καθιστώντας την έτσι πολύ βαριά, καθώς εκτελούνται πολλά ερωτήματα SPARQL στην

υπηρεσία Οντολογίας. Στην εικόνα 5.4 φαίνεται η JSON περιγραφή της εφαρμογής όπως είναι αποθηκευμένη στην υπηρεσία Node-RED.

```
{
  "appname": "chaniaevery",
  "appscope": "weather",
  "info": [
    {
      "attributes": [
        "temperature",
        "humidity",
        "windSpeed",
        "cloud",
        "precipitation",
        "pressure"
      ],
      "city": "Chania",
      "domains": [
        "Activity",
        "Garment",
        "Transport"
      ],
      "func": "LIVE"
    }
  ]
}
```

Εικόνα 5.4: JSON περιγραφή εφαρμογής “chaniaevery”.

Στην εικόνα φαίνονται τα εξής πεδία:

- “Appname”: Όνομα εφαρμογής.
- “Appscope”: Ο τύπος της εφαρμογής. Με την τιμή “weather” δείχνει ότι αναφέρεται σε εφαρμογή πόλης.
- “Attributes”: Οι μετρήσιμες ιδιότητες για τις οποίες ο χρήστης ζητάει πληροφορία. Εδώ ζητήθηκαν όλες οι πιθανές ιδιότητες (θερμοκρασία, υγρασία, ταχύτητα ανέμου, νεφοκάλυψη, βροχόπτωση, πίεση).
- “City”: Η πόλη στην οποία αναφέρεται η εφαρμογή δηλαδή Χανιά (Chania).
- “Domains”: Τα πεδία της καθημερινότητας για τα οποία ο χρήστης ζητάει συμβουλές, δηλαδή τι Δραστηριότητα (Activity), Ενδυμασία (Garment) και Μεταφορά (Transport) είναι κατάλληλα ανάλογα με τον καιρό της πόλης.
- “Func”: Έχει τιμή LIVE άρα πρόκειται για σημασιολογική εφαρμογή.

Εφόσον η εφαρμογή υπάρχει, στην εικόνα 5.4 φαίνεται το αποτέλεσμα της εφαρμογής.

Για κάθε μετρήσιμη ιδιότητα που ζητάει ο χρήστης (θερμοκρασία, υγρασία, ταχύτητα αέρα, νεφοκάλυψη, βροχόπτωση, πίεση) τρέχει ένας SWRL κανόνας ο οποίος παράγει ένα αποτέλεσμα. Στην περίπτωση αυτή τρέχουν 6 SWRL κανόνες αρχικά οι οποίοι παράγουν τα 6 αποτελέσματα που φαίνονται κάτω από την στήλη Live state. Τα αποτελέσματα αυτά ανακτήθηκαν με SPARQL ερωτήματα μέσω του Jena API στην οντολογία που είναι αποθηκευμένη στο Virtuoso. Στη συνέχεια για κάθε ένα αποτέλεσμα της πρώτης στήλης πάλι με SPARQL ερωτήματα ζητάμε και παίρνουμε τις συμβουλές της οντολογίας για κάθε πεδίο της καθημερινότητας που έχει επιλέξει ο χρήστης. Στην περίπτωση αυτή ο χρήστης επέλεξε 3 πεδία της καθημερινότητας. Άρα για κάθε μετρήσιμη ιδιότητα (6) ελέγχουμε για κάθε πεδίο (3) τι συμβουλή δίνει η οντολογία, δηλαδή 18 ερωτήματα στην οντολογία. Τέλος, κάτω από την στήλη Further assumptions βλέπουμε τα αποτελέσματα των κανόνων που δίνουν επιπλέον συμπεράσματα σύμφωνα με τα αποτελέσματα της πρώτης στήλης. Οι δύο κανόνες που έτρεξαν είναι οι “ManyHomesWillTurnOnHeat” και “ExpectingBadWeather”. Συγκεκριμένα:

- ManyHomesWillTurnOnHeat: Παράχθηκε λόγω του BelowNormalTemperature.
- ExpectingBadWeather: Παράχθηκε λόγω του LowPressure

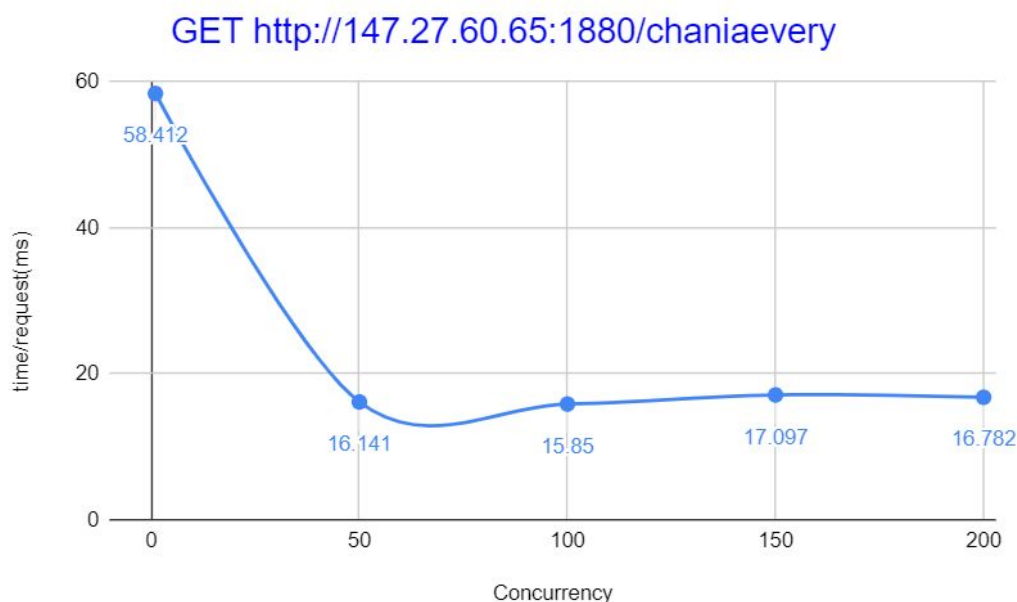
chaniaevery		
Live state	Recommendations based on current state	Further assumptions
BelowNormalTemperature	WindSurfing	ManyHomesWillTurnOnHeat
DryHumidity	KiteSurf	ExpectingBadWeather
Windy	Catamaran	
Sunny	Kite	
LightRain	Surfing	
LowPressure	Turtleneck	
	Raincoat	
	BeachVolley	
	Fishing	
	Tennis	
	Boat	
	Park	
	BeachSunbathing	
	Short	
	Hat	
	Swimsuit	
	SunGlass	
	Squash	
	Opera	
	Paintball	
	Bowling	
	Theater	
	Concert	
		Activate Windows Go to Settings to activate Windows.

Εικόνα 5.4: Απεικόνιση εφαρμογής “chaniaevery” που βλέπει ο χρήστης.

Υπηρεσίες: Αυτή η εφαρμογή αποτελείται από semantic δεδομένα τα οποία υπολογίζονται στην υπηρεσία Οντολογίας, άρα η απόδοση εξαρτάται από αυτή την υπηρεσία και την υπηρεσία Node-red, καθώς εκεί είναι αποθηκευμένη η εφαρμογή.

Αίτημα REST: GET <http://147.27.60.65:1880/chaniaevery>

Αποτελέσματα και σχολιασμός: Τα αποτελέσματα για το μέσο χρόνο εκτέλεσης και την κατανάλωση πόρων (CPU και μνήμης RAM) φαίνονται στο Σχήμα 5.5 και αναγράφονται στον Πίνακα 5.2. Παρατηρούμε ότι πάλι ο μέγιστος από τους μέσους χρόνους εκτέλεσης παρατηρείται στην πρώτη περίπτωση, όπου τα αιτήματα στέλνονται ανά ένα (58.142 ms), ενώ ο ελάχιστος από τους μέσους χρόνους εκτέλεσης παρατηρείται στην περίπτωση όπου τα αιτήματα στέλνονται ανά 100 (15.85 ms), με μικρή διαφορά από τις περιπτώσεις των 50, 150 και 200. Παρατηρούμε ότι και στο η κατανάλωση της CPU κυμαίνεται σε πολύ μεγάλα επίπεδα και της RAM σε μεσαία. Ο λόγος που συμβαίνει αυτό είναι επειδή χρησιμοποιείται reasoner και εκτελούνται συνεχώς και πολλά αιτήματα SPARQL στην οντολογία.



Εικόνα 5.5: Χρόνος απόκρισης σε 1000 ερωτήματα semantic εφαρμογής (Πείραμα 2).

concurrency	CPU Load	RAM Load	Time/request (ms)
1	41%	16%	58.412
50	100%	16%	16.141
100	100%	16%	15.85
150	100%	17%	17.097
200	100%	18%	16.782

Πίνακας 5.2: Χρόνος απόκρισης, κατανάλωση CPU και κατανάλωση RAM σε 1000 ερωτήματα semantic εφαρμογής (Πείραμα 2).

5.3 Πείραμα 3

Σενάριο: Ένας χρήστης μέσω γραφικής διεπαφής θέλει να δει τις τρέχουσες τιμές κάποιων χαρακτηριστικών για ένα σπίτι, καθώς και τις ενεργοποιήσεις των αυτοματισμών που πρέπει να γίνουν σύμφωνα πάνω σε αυτά τα χαρακτηριστικά. Για τα πλαίσια του πειράματος, δημιουργήθηκε μία εφαρμογή με όνομα “app134all”, στην οποία έχουν επιλεγθεί 2 μετρήσιμες ιδιότητες καθιστώντας την έτσι μεσαίου βάρους. Στην εικόνα 5.7 φαίνεται η JSON περιγραφή της εφαρμογής όπως είναι αποθηκευμένη στην υπηρεσία Node-RED.

```
{
  "appname": "app134all",
  "appscope": "home",
  "info": [
    {
      "attributes": [
        "temperature",
        "humidity"
      ],
      "city": "Appartment134",
      "domains": [],
      "func": "LIVE"
    }
  ]
}
```

Εικόνα 5.7: JSON περιγραφή εφαρμογής “app134all”.

Στην εικόνα φαίνονται τα εξής πεδία:

- “Appname”: Όνομα εφαρμογής.
- “Appscope”: Ο τύπος της εφαρμογής. Με την τιμή “home” δείχνει ότι αναφέρεται σε σπίτι.
- “Attributes”: Οι μετρήσιμες ιδιότητες οι οποίες είναι θερμοκρασία και υγρασία.
- “City”: Με την τιμή “Apartment134” δείχνει το μοναδικό αναγνωριστικό του σπιτιού στο οποίο αναφέρεται η εφαρμογή.
- “Domains”: Είναι χωρί τιμή καθώς έχουμε εφαρμογή σπιτιού και άρα δεν απευθύνεται σε πεδία της καθημερινότητας που έχουν να κάνουν με συνθήκες καιρού πόλης.
- “Func”: Έχει τιμή LIVE άρα έχουμε σημασιολογική εφαρμογή.

Εφόσον η εφαρμογή υπάρχει, στην εικόνα 5.6 φαίνεται το αποτέλεσμα της εφαρμογής. Κάτω από την στήλη Live state βλέπουμε τα αποτελέσματα 2 SWRL κανόνων: “AboveNormalTemperature” και “VeryMoistHumidity”. Μέσω των 2 αυτών αποτελεσμάτων παίρνουμε και τα αποτελέσματα κάτω από την στήλη Recommendations based on current state. Τα αποτελέσματα αυτά δείχνουν ποιοι αυτοματισμοί θα ενεργοποιηθούν λόγω της πρώτης στήλης. Συγκεκριμένα:

- SwitchOnAirConditioning και SwitchOffHeating: Παράχθηκαν λόγω του “AboveNormalTemperature”.
- SwitchOnDehydrator : Παράχθηκε λόγω του VeryMoistHumidity.

Τα αποτελέσματα της τρίτης στήλης δείχνουν πάλι 2 SWRL κανόνες που εκτελέστηκαν λόγω των προηγούμενων 2 στηλών. Συγκεκριμένα:

- ExpectingHighElectricityConsumption: Παράχθηκε λόγω του “SwitchOnAirConditioning”.
- MightBeUnhealthy: Παράχθηκε λόγω του VeryMoistHumidity.

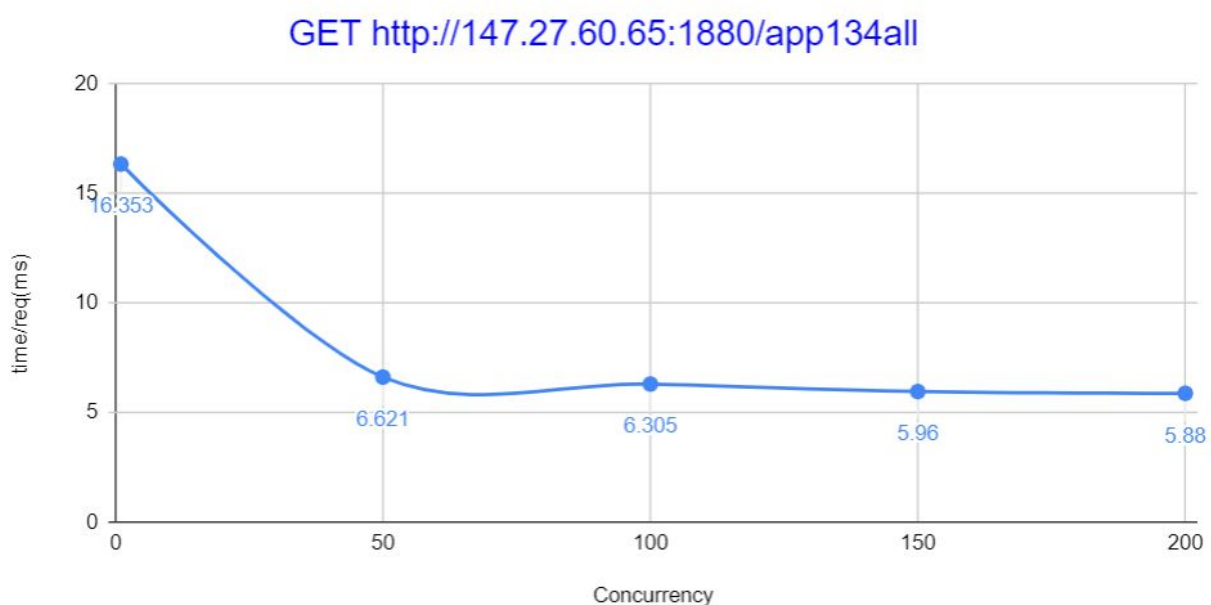
app134all		
Live state	Recommendations based on current state	Further assumptions
NormalTemperature VeryMoistHumidity	SwitchOnAirConditioning SwitchOffHeating SwitchOnDehydrator	ExpectingHighElectricityConsumption MightBeUnhealthy

Εικόνα 5.6: Απεικόνιση εφαρμογής “app134all” που βλέπει ο χρήστης.

Υπηρεσίες: Αυτή η εφαρμογή αποτελείται από semantic δεδομένα τα οποία υπολογίζονται στην υπηρεσία Οντολογίας, άρα η απόδοση εξαρτάται από αυτή την υπηρεσία και την υπηρεσία Node-red, καθώς εκεί είναι αποθηκευμένη η εφαρμογή.

Αίτημα REST: GET <http://147.27.60.65:1880/app134all>

Αποτελέσματα και σχολιασμός: Τα αποτελέσματα για το μέσο χρόνο εκτέλεσης και την κατανάλωση πόρων (CPU και μνήμης RAM) φαίνονται στο Σχήμα 5.7 και αναγράφονται στον Πίνακα 5.3. Παρατηρούμε ότι πάλι ο μέγιστος από τους μέσους χρόνους εκτέλεσης παρατηρείται στην πρώτη περίπτωση, όπου τα αιτήματα στέλνονται ανά ένα 16.153 ms), ενώ ο ελάχιστος από τους μέσους χρόνους εκτέλεσης παρατηρείται στην περίπτωση όπου τα αιτήματα στέλνονται ανά 200(5.88 ms), με μικρή διαφορά από τις περιπτώσεις των 50, 100 και 150. Παρατηρούμε ότι και στο η κατανάλωση της CPU κυμαίνεται σε σχετικά μεγάλα επίπεδα και της RAM σε σχετικά μικρά. Ο λόγος που συμβαίνει αυτό είναι επειδή πάλι χρησιμοποιείται reasoner και εκτελούνται συνεχώς αιτήματα SPARQL στην οντολογία, τα οποία δεν είναι τόσο πολλά σε αριθμό όμως.



Εικόνα 5.7: Χρόνος απόκρισης σε 1000 ερωτήματα semantic εφαρμογής (Πείραμα 3).

concurrency	CPU Load	RAM Load	Time/request (ms)
1	30%	17%	16.353

50	75%	17%	6.621
100	80%	16%	6.305
150	85%	17%	5.96
200	87%	17%	5.88

Πίνακας 5.3: Χρόνος απόκρισης, κατανάλωση CPU και κατανάλωση RAM σε 1000 ερωτήματα semantic εφαρμογής (Πείραμα 3).

Κεφάλαιο 6

Συμπεράσματα - Μελλοντικές Επεκτάσεις

6.1 Συμπεράσματα

Η υιοθέτηση Υπηρεσιοκεντρικής Αρχιτεκτονικής, και συγκεκριμένα των υπηρεσιών που βασίζονται στο πρότυπο REST, διευκόλυνε σε μεγάλο βαθμό την επικοινωνία μεταξύ των υπηρεσιών και συνεπώς τον προγραμματισμό της δομής ενορχήστρωσης των υπηρεσιών (App logic). Μεγάλο πλεονέκτημα αυτής της αρχιτεκτονικής αποτέλεσε η ευελιξία στη χρήση διαφορετικών γλωσσών προγραμματισμού για να πραγματοποιηθούν διαφορετικές λειτουργίες του συστήματος καθώς και ευκολία στην παρέμβαση - τροποποίηση υπηρεσιών μεμονωμένων υπηρεσιών, δίχως να επηρεάζεται το συνολικό σύστημα. Επομένως, το σύστημα βασίζεται σε μια επεκτάσιμη και εύκολα τροποποιήσιμη αρχιτεκτονική.

Το γεγονός ότι το σύστημα αναπτύχθηκε μέσω του οικοσυστήματος FIWARE, προσέφερε το πλεονέκτημα της χρήσης έτοιμων υπηρεσιών (που παρέχει το FIWARE). Η υλοποίηση της εργασίας ξεκίνησε πάνω σε μια πολύ σημαντική βάση, καθώς υπήρχαν διαθέσιμες υπηρεσίες όπως η Υπηρεσία Διαχείρισης Συμβάντων και Συνδρομών, η Υπηρεσία Cygnus, η Υπηρεσία Ταυτοποίησης και Εξουσιοδότησης Χρηστών (Keyrock), κ.ά.

Η υπηρεσία IDAS του FIWARE παρείχε στο σύστημα τη ζητούμενη ανεξαρτησία από τα πρωτόκολλα επικοινωνίας των συσκευών, έτσι ώστε οι προγραμματιστές εφαρμογών να λαμβάνουν κατευθείαν την πληροφορία σε JSON / JSON-LD και να μην ενδιαφέρονται για το εκάστοτε πρωτόκολλο. Συνεπώς, ικανοποιείται αυτή η προδιαγραφή που τέθηκε αρχικά - για ανεξαρτησία από το πρωτόκολλο επικοινωνίας - η οποία ορίζεται από τον Ιστό των Πραγμάτων (Web of Things). Αξιολογώντας την

υπηρεσία Οντολογίας, και συγκεκριμένα την υπηρεσία JENA API και την επικοινωνία της με την οντολογία στο ΣΔΒΔ Virtuoso, αυτή κρίνεται πολύ ικανοποιητική σχετικά με το χρόνο απόκρισης και την κατανάλωση πόρων. Πέραν αυτού, επιτυγχάνει να καταστήσει γρήγορη και απλή την επικοινωνία της οντολογίας του συστήματος με όλο το υπόλοιπο σύστημα κι επομένως να αξιοποιήσει τις τεχνολογίες του Σημασιολογικού Ιστού για τις ανάγκες της πλατφόρμας.

Ακόμα, η υπηρεσία Οντολογίας παρέχει στο σύστημα την απαραίτητη εκφραστικότητα σε όλα τα Things που ανήκουν σε αυτό, καθιστώντας το έτσι ένα Semantic Web of Things σύστημα που ικανοποιεί τις απαιτήσεις του Σημασιολογικού Ιστού και βοηθάει στην δημιουργία semantic εφαρμογών. Σημαντικό στοιχείο που πρέπει να αναφερθεί είναι η δυνατότητα επέκταση της οντολογίας, με άλλες οντολογίες που περιγράφουν συγκεκριμένα πεδία της καθημερινής ζωής, ή με περαιτέρω εκφραστικότητα με διαφορετικά ήδη αισθητήρων, κάτι που προσφέρει ατελείωτη δυνατότητα σχετικά με τους τομείς που μπορεί να καλύπτει το σύστημα. Συγκεκριμένα, δεδομένης μιας οντολογίας που περιγράφει ασθένειες και συμπτώματα, και μιας που περιγράφει αισθητήρες που εντοπίζουν και λειτουργούν στο πεδίο της υγείας, μπορούν εύκολα να ενταχθούν στο σύστημα, επεκτείνοντας έτσι τη δυνατότητα δημιουργίας εφαρμογών που αφορούν τον τομέα της υγείας, και όχι μόνο τον τομέα των περιβαντολλογικών συνθηκών και αυτοματισμών σε σπίτι.

Η υπηρεσία Mashup, η οποία χρησιμοποιεί το εργαλείο Node-RED, έχει τη δυνατότητα δημιουργίας ευέλικτων εφαρμογών, εύκολα με προκατασκευασμένους κόμβους και εύχρηστη RESTful διεπαφή, οι οποίες εφαρμογές είναι εύκολες στην πρόσβαση σε πραγματικό χρόνο. Έτσι, μας διευκόλυνε στη δημιουργία semantic εφαρμογών, που ήταν και το ζητούμενο της εργασίας, και καθιστάται ως ένα χρήσιμο εργαλείο για το Διαδίκτυο των Πραγμάτων.

Συνοψίζοντας τα παραπάνω συμπεράσματα, η πλατφόρμα που αναπτύχθηκε είναι σε θέση να διαχειρίζεται μεγάλο αριθμό συσκευών, ενώ οι λειτουργίες που παρέχει εκτελούνται ικανοποιητικά σε πραγματικό χρόνο. Ακόμα, ο σημασιολογικός χαρακτήρας της υλοποίησης προσδίδει στο σύστημα ένα σημαντικό πλεονέκτημα που εκλείπει από άλλες αρχιτεκτονικές Διαδικτύου των Πραγμάτων.

6.2 Μελλοντικές Επεκτάσεις

Η υλοποίηση του παρόντος συστήματος, όπως αυτό σχεδιάστηκε κατά την εκπόνηση της εργασίας, μπορεί να θεωρηθεί ότι εν τέλει ικανοποιεί τις προδιαγραφές που τέθηκαν, με αποτέλεσμα την ανάπτυξη ενός πλήρως επεκτάσιμου περιβάλλοντος Διαδικτύου των Πραγμάτων για την καταχώρηση, διαχείριση και δημοσίευση συσκευών, που επιτρέπει και διευκολύνει την δημιουργία εφαρμογών με σημασιολογικό χαρακτήρα.

Ωστόσο, καθώς η εργασία εκπονήθηκε μέσα σε ένα συγκεκριμένο χρονικό πλαίσιο, είναι αναπόφευκτο να υπάρχουν ορισμένες αδυναμίες που θα περιγραφούν παρακάτω συνοδευόμενες από πιθανές προτάσεις για τη λύση τους.

Μια βελτίωση που επιδέχεται το σύστημα είναι σίγουρα η προσθήκη επιπλέον υπηρεσιών Ασφάλειας (Security services), για παράδειγμα η χρήση της υπηρεσίας PEP Proxy του FIWARE, που επιτρέπει την πρόσβαση σε πόρους του συστήματος μόνο σε συγκεκριμένους εγγεγραμμένους - εξουσιοδοτημένους χρήστες και υπηρεσίες, κάτι που λείπει από το παρόν σύστημα. Η προσθήκη μιας τέτοιας υπηρεσίας προτείνεται ως μια σημαντική μελλοντική επέκταση, καθώς κρίνεται απαραίτητη η ασφάλεια του συστήματος από αυτήν.

Τα αιτήματα μεταξύ των υπηρεσιών του συστήματος πραγματοποιούνται με τη χρήση του πρωτοκόλλου HTTP. Σημαντική βελτίωση στον τομέα της ασφάλειας του συστήματος θα ήταν η αλλαγή του πρωτοκόλλου επικοινωνίας σε HTTPS καθώς είναι - λόγω της αρχιτεκτονικής του - πιο ασφαλές πρωτόκολλο όσον αφορά στη μετάδοση “ευαίσθητων” πληροφοριών όπως είναι τα OAuth2 tokens.

Ακόμα, εντοπίστηκε το πρόβλημα σχετικά με την επεκτασιμότητα των γράφων της Virtuoso, καθώς παρατηρήσαμε ότι ο αριθμός των δεδομένων που μπορούν να αποθηκεύσουν είναι περιορισμένος, με αποτέλεσμα να χωρίσουμε την οντολογία σε διαφορετικούς γράφους. Μία λύση στο παραπάνω πρόβλημα θα ήταν να κάνουμε μια καλύτερη κατανομή της πληροφορίας σε γράφους, για παράδειγμα να “διακρίνουμε” την κάθε πόλη που μελετάται από το σύστημα σε μικρότερες περιοχές, με αποτέλεσμα την “ελάφρυνση” του κάθε γράφου της οντολογίας. Αυτό θα είναι πολύ σημαντικό για να πραγματοποιείται πιο εύκολα και αποδοτικά ο μηχανισμός συλλογισμού (reasoning) της Υπηρεσίας Οντολογίας του συστήματος.

Γενικά το σημείο δυσκολίας στην απόδοση του συστήματος εντοπίζεται καθαρά στον υπηρεσία Οντολογίας. Η χρήση μιας αποδοτικής βάσης για την αποθήκευση της οντολογίας (triple store) και των Στιγμιотύπων της, είναι εξαιρετικής σημασίας. Επίσης, στο σύστημά μας γίνεται χρήση του Jena API. Αυτό το βοηθητικό λογισμικό αν και προσφέρει πολλές δυνατότητες, αποτελεί μια υπο-υπηρεσία που προσθέτει καθυστέρηση στην επικοινωνία της οντολογίας με το υπόλοιπο σύστημα.

Βιβλιογραφία - Αναφορές

[1] X. Koundourakis: Design and Implementation of Service Oriented Architecture for Deploying IoT Applications in the Cloud, Diploma Thesis, School of Electrical and Computer Engineering, Technical University of Crete, 2019.

<https://dias.library.tuc.gr/view/81121>

[2] Α. Τζαβάρας, Υποστήριξη Λειτουργικότητας Σημασιολογικού Ιστού σε Περιβάλλον Διαδικτύου των Πραγμάτων και Υπολογιστικού Νέφους , Διπλωματική Εργασία, Πολυτεχνείο Κρήτης, Οκτώβριος 2019.

[3] Euripides G.M. Petrakis, Stelios Sotiriadis, Theodoros Soultanopoulos, Pelagia Tsiachri Rentaa, Rajkumar Buyyac, Nik Bessis, Internet of Things as a Service (iTaaS): Challenges and solutions for management of sensor data on the cloud and the fog, September 2018. doi:10.1016/j.iot.2018.09.009.

<http://www.intelligence.tuc.gr/~petrakis/publications/iTaaS.pdf>

[4] Dominique Guinard, Vlad Trifa, Towards the Web of Things: Web Mashups for Embedded Devices, in: MEM 2009 in Proceedings of WWW 2009. ACM

https://webofthings.org/wp-content/uploads/2009/03/guinard_trifa_webofthings_mashup.pdf

[5] Farzad Khodadadi, Richard O. Sinnott, A Semantic-aware Framework for Service Definition and Discovery in the Internet of Things Using CoAP, in: The 8th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN).

<https://www.sciencedirect.com/science/article/pii/S1877050917317441>

[6] Maria Bermudez-Edo, Tarek Elsaleh, Payam Barnaghi, Kerry Taylor, IoT-Lite: a lightweight semantic model for the internet of things and its use with dynamic semantics.

<http://epubs.surrey.ac.uk/841613/1/IoT-Lite.pdf>

[7] Payam Barnaghi, Frieder Ganz, Hamidreza Abangar, Mirko Presser, and Klaus Moessner, Sense2Web: A Linked Data Platform for Semantic Sensor Networks, Centre for Communication Systems Research, University of Surrey, Guildford, GU2 7XH, United Kingdom.

<http://semantic-web-journal.org/sites/default/files/swj189.pdf>

[8] Vlad Trifa, Dominique Guinard, David Carrera, Web Thing Model, W3C Member Submission 24 August 2015

<https://www.w3.org/Submission/wot-model/>

[9] **Cross-Domain Internet of Things Application development: M3 framework and Evaluation.** Gyrard, A.; Datta, S. K.; Bonnet, C.; and Boudaoud, K. In *FICLOUD 2015*, 3rd International Conference on Future Internet of Things and Cloud, August 24-26, Rome, Italy, 2015.