Ατομική Διπλωματική Εργασία

Υλοποίηση και Αποτίμηση του Πλαισίου Εκτέλεσης Κανόνων Προτιμήσεων σε Έξυπνα Κτήρια

Αντώνης Βασιλείου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Υλοποίηση και Αποτίμηση του Πλαισίου Εκτέλεσης Κανόνων Προτιμήσεων σε Έξυπνα Κτήρια

Αντώνης Βασιλείου

Επιβλέπων Καθηγητής

Δημήτρης Ζεϊναλιπούρ

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2021

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου Δρ. Δημήτρη Ζεϊναλιπούρ του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου, για την καθοδήγηση του καθ' όλη τη διάρκεια εκπόνησης αυτής της πτυχιακής εργασίας. Λόγω των αλλόκοτων συνθηκών της πανδημίας στην εκπαίδευση μας, παρουσιάστηκαν δυσκολίες. Παρόλα αυτά μέσω των συχνών συναντήσεων που είχαμε καταφέραμε να πετύχουμε τους στόχους που τέθηκαν. Ακόμη θα ήθελα να τον ευχαριστήσω για τις γνώσεις που απέκτησα στη διάρκεια της συνεργασίας μας.

Θα ήθελα να δώσω ένα μεγάλο ευχαριστώ στην οικογένεια μου για όλη την οικονομική και ψυχολογική υποστήριξη που μου έχει δώσει καθ' όλη τη πορεία μου στο Πανεπιστήμιο Κύπρου. Η οικογένεια μου με στήριξε σε όλες τις δυσκολίες είχα με το μέγιστο δυνατό τρόπο και αυτό με βοήθησε να μην τα παρατήσω ποτέ.

Επίσης θα ήθελα να ευχαριστήσω όλους τους φίλους μου εντός και εκτός του Πανεπιστημίου Κύπρου που στάθηκαν δίπλα μου και με υποστήριξαν.

Τέλος, θα ήθελα να ευχαριστώ τον Σωτήρη Κωνσταντίνου, διδακτορικό φοιτητή του Εργαστηρίου Συστημάτων Διαχείρισης Δεδομένων στο Τμήμα Πληροφορικής του Πανεπιστημίου Κύπρου, για τη συμβολή του στην υποστήριξη και καθοδήγηση της παρούσας διπλωματικής εργασίας.

Η επίδειξη του συστήματος IMCF που περιγράφεται σε αυτήν τη διπλωματική́ εργασία έχει παρουσιαστεί στο πιο κάτω συνέδριο:

•"IMCF: The IoT Meta-Control Firewall for Smart Buildings", Soteris Constantinou, Antonis Vasileiou, Andreas Konstantinidis, Panos K. Chrysanthis, Demetrios Zeinalipour-Yazti, 24th International Conference on Extending Database Technology (EDBT'21), OpenProceedings.org, pp. 658--661, March 23 - March 26, 2021, Nicosia, Cyprus, 2021

Περίληψη

Η υπερθέρμανση του πλανήτη είναι ένα γεγονός το οποίο έχει αναγνωριστεί από όλες τις χώρες παγκόσμια. Τα τελευταία χρόνια γίνονται τεράστιες προσπάθειες από όλα τα κράτη για μείωση ως και εξαφάνιση αυτού του φαινομένου. Η τεράστια ανάπτυξη του Διαδικτύου των πραγμάτων την τελευταία εικοσαετία, έχει αποφέρει ένα καινούργιο τρόπο διαχείρισης των έξυπνων συσκευών σε ένα έξυπνο κτήριο. Αυτές οι συσκευές μπορούν να ενεργοποιηθούν και να απενεργοποιηθούν ανάλογα με τις προτιμήσεις του χρήστη, μέσω ροών αυτοματισμού εργασίας. Με στόχο την μείωση της περιττής κατανάλωσης ηλεκτρικής ενέργειας καθώς και της μείωσης της εκπομπής διοξειδίου του άνθρακα, έχει υλοποιηθεί και αποτιμηθεί το πλαίσιο ικανοποίησης κανόνων προτιμήσεων σε έξυπνα κτήρια.

Στην παρούσα διπλωματική εργασία έχουν υλοποιηθεί οι αλγόριθμοι Energy Planner και Green Planner, οι οποίοι χρησιμοποιούν μέτα-κανόνες από ένα πίνακα μέτα-κανόνων που ορίζονται από χρήστες του συστήματος. Στους κανόνες μπορούν να συμπεριληφθούν προτιμήσεις για τη θερμοκρασία και το επίπεδο φωτός ανά διάστημα ώρας και ένα προτιμώμενο όριο κατανάλωσης ενέργειας. Βάσει αυτών των κανόνων οι αλγόριθμοι προσπαθούν να διατηρήσουν αυτές τις προτιμήσεις και να διατηρήσουν το προτιμώμενο όριο ενέργειας που έχει οριστεί δημιουργώντας ένα πλάνο ενεργοποίησης και απενεργοποίησης των κανόνων. Επίσης ο αλγόριθμος Green Planner προσπαθεί να κρατήσει τις εκπομπές διοξειδίου του άνθρακα στο ελάχιστο. Αφού οριστεί το πλάνο οι αλγόριθμοι συνεχίζουν με την ενεργοποίηση και απενεργοποίηση των διαφόρων συσκευών μέσω ροών αυτοματισμού για την διατήρηση των προτιμήσεων του χρήστη.

Οι αλγόριθμοι βάσει των αποτιμήσεων κατάφεραν να μειώσουν την κατανάλωση ενέργειας κατά 10% από το προτιμώμενο όριο που έχουν θέσει οι χρήστες, ενώ ταυτόχρονα παραμένουν άνετοι κατά 93% στην χειρότερη περίπτωση και 98% στην καλύτερη. Συνεπώς οι αλγόριθμοι που υλοποιήθηκαν κατάφεραν να επιτύχουν το στόχο της μείωσης της κατανάλωσης ενέργειας και της εκπομπής ρίπων θερμοκηπίου.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

1.1 Παρακίνηση

Η παγκόσμια υπερθέρμανση του πλανήτη είναι ένα πλέον γνωστό φαινόμενο κατά το οποίο προβλέπονται διάφορες παγκόσμιες καταστροφές εάν δεν υπάρξει κάποια λύση. Ευτυχώς εδώ και κάποια χρόνια υπήρξε κινητοποίηση από όλα τα έθνη για λύση αυτού του σοβαρού προβλήματος.

Το 2016 με το σύμφωνο “Paris Agreement” [1] το οποίο είναι μια νομικά δεσμευτική διεθνής συνθήκη για την αλλαγή του κλίματος, τέθηκε σε κίνηση. Στόχος του είναι να μειώσει την αύξηση της σε λιγότερο από 2 βαθμούς Κελσίου. Για να επιτύχουν αυτόν τον μακροπρόθεσμο στόχο θερμοκρασίας, οι χώρες στοχεύουν στην επίτευξη της παγκόσμιας κορυφής των εκπομπών αερίων θερμοκηπίου το συντομότερο δυνατό για να επιτύχουν έναν κλιματικά ουδέτερο κόσμο έως τα μέσα του αιώνα. Αυτή η συμφωνία αποτελεί ορόσημο της πολυμερούς διαδικασίας αλλαγής του κλίματος, διότι για πρώτη φορά δεσμεύονται με μια συμφωνία όλα τα έθνη, για την καταπολέμηση της κλιματικής αλλαγής. Για την εκτέλεση αυτής της συμφωνίας απαιτούνται οικονομικές και κοινωνικές αλλαγές.

Η ευρωπαϊκή ένωση με το “Green deal” [2] στοχεύει μέχρι το 2050 να είναι η πρώτη ήπειρος που δεν θα έχει ρίπους αερίων του θερμοκηπίου. Για να το πετύχει αυτό δόθηκε ένα σχέδιο δράσης κατά το οποίο θα ενισχυθεί η αποτελεσματική χρήση των πόρων για μετάβαση σε μια καθαρή και κυκλική οικονομία, καθώς και η αποκατάσταση της βιοποικιλότητας με μείωση της ρύπανσης.

Ο διαγωνισμός με όνομα “Saves” [3], είναι ένας διαγωνισμός που πραγματοποιήθηκε κατά τα ακαδημαϊκά έτη 2014/15 και 2015/16, στον οποίον οι φοιτητές/τριες παρακινήθηκαν να εξοικονομήσουν ενέργεια δημιουργώντας έναν αγώνα μεταξύ κοιτώνων, όπου ανταγωνίζονταν στην εξοικονόμηση της περισσότερης ενέργειας. Στόχος αυτού του έργου ήταν η εγκατάσταση συνηθειών εξοικονόμησης ενέργειας στους μαθητές σε μια βασική στιγμή αλλαγής στη ζωή τους, ώστε να μπορούν να συνεχίσουν τις ενέργειες εξοικονόμησης ενέργειας καθ’ όλη τη διάρκεια της ιδιωτικής τους ζωής.

Με έμπνευση τις παραπάνω κινητοποιήσεις και με σκοπό την εξοικονόμηση ενέργειας και την μείωση των ρίπων του διοξειδίου του άνθρακα, συντάχτηκε η ακόλουθη ατομική διπλωματική εργασία.

1.2 Πρόβλημα

Τα σημερινά συστήματα διαχείρισης ενέργειας στον χώρο του διαδικτύου των πραγμάτων είναι πολύ ανεπτυγμένα. Για παράδειγμα υπάρχουν δυνατότητες προγραμματισμού ενεργοποίησης και απενεργοποίησης των διαφόρων έξυπνων συσκευών σε διάφορες ώρες της ημέρας ή ανάλογα με τον καιρό. Δυστυχώς όμως δεν υπάρχει δυνατότητα σε κανένα σύστημα διαχείρισης συσκευών σε ένα έξυπνο περιβάλλον, κατά την οποία μπορούν να τεθούν και να τηρηθούν μακροχρόνιοι στόχοι εξοικονόμησης ενέργειας.

1.3 Συνεισφορά

Το Πλαίσιο Εκτέλεσης Κανόνων Προτιμήσεων για έξυπνα κτήρια είναι ένα καινοτόμο σύστημα το οποίο στοχεύει να καλύψει το κενό του χειροκίνητου συντονισμού ροών αυτοματισμού κανόνων RAW για την επίτευξη των στόχων κατανάλωσης ενέργειας. Αρχικά ο χρήστης ή μια ομάδα χρηστών καθορίζει ένα διάνυσμα ροών αυτοματισμού

κανόνων σε ένα πίνακα μέτα-κανόνων που ονομάζεται Meta-Rule-Table (MRT) και ενός προφίλ κατανάλωσης ενέργειας που ονομάζεται ECP. Στόχος του είναι να προσδιοριστεί μεταξύ όλων των κανόνων MRT εκείνων που πρέπει να απορριφθούν, ώστε ο χρήστης να παραμείνει εντός του επιθυμητού ενεργειακού προϋπολογισμού σύμφωνα με το ιστορικό του ECP. Συγκεκριμένα το σύστημα χρησιμοποιεί τον αλγόριθμο Energy Planner (EP) climbing και τον αλγόριθμο Green Planner (GP), ο οποίοι είναι εμπνευσμένος από αλγόριθμους τεχνητής νοημοσύνης όπως hill climbing και simulated annealing αντίστοιχα. Ο αλγόριθμος περνά πάνω στον μεγάλο χώρο αναζήτησης N-συνδυασμών όπου το N είναι το σύνολο των κανόνων, αποδίδοντας γρήγορα τους κανόνες που πρέπει να απορριφθούν για να επιτευχθεί ο στόχος του χρήστη. Το πλαίσιο προσαρμόζει τις ροές αυτοματισμού με τέτοιο τρόπο ώστε να μην έρχονται σε αντίθεση με τους μακροπρόθεσμους στόχους των χρηστών, απορρίπτοντας ορισμένους κανόνες βάσει προτεραιότητας προτιμήσεων. Σημαντικό να αναφερθεί ότι οι ροές αυτοματισμού κανόνων διακρίνονται σε δύο κατηγορίες άνεσης και αναγκαιότητας. Οι κανόνες ευκολίας στοχεύουν στην προώθηση της φυσικής άνεσης ενός ατόμου π.χ. θερμοκρασία δωματίου, φωτισμός περιβάλλοντος ή ό, τι θεωρείται προσωρινή άνεση, ενώ οι κανόνες αναγκαιότητας είναι εκείνοι οι κανόνες που πρέπει πάντα να εκτελούνται ανεξάρτητα από το εάν επιτυγχάνεται ο μακροπρόθεσμος στόχος.

1.4 Περίγραμμα εργασίας

Η παρούσα διπλωματική εργασία αποτελείτε από εφτά κεφάλαια. Αυτά είναι η «Εισαγωγή», η «Σχετική Δουλειά», η «Αρχιτεκτονική», το «Υλοποίηση Αλγορίθμων», το «Διεπαφές Διαπροσωπείας», η «Πειραματική Μεθοδολογία & Αποτίμηση» και τα «Συμπεράσματα».

Σε αυτό το κεφάλαιο αναφέρθηκαν οι λόγοι που ώθησαν την υλοποίηση του πλαισίου εκτέλεσης κανόνων προτιμήσεων σε έξυπνα κτήρια. Επίσης, αναφέρθηκε η συνεισφορά που προσφέρει το πλαίσιο αυτό.

Στο δεύτερο κεφάλαιο ακολουθεί περιγραφή σχετικών δουλειών που έχουν γίνει στο πεδίο ενασχόλησης της εργασίας αυτής, των ροών αυτοματισμού ελέγχου και των αλγορίθμων στους οποίους βασίζεται η υλοποίηση.

Στο τρίτο κεφάλαιο δίνεται μια λεπτομερείς περιγραφή της αρχιτεκτονικής του συστήματος στο οποίο έχει αναπτυχθεί το πλαίσιο μας, τα δεδομένα εισόδου του συστήματος και οι αλγόριθμοι οι οποίοι αναπτύχθηκαν.

Στο τέταρτο κεφάλαιο δίνεται μια λεπτομερείς περιγραφή της υλοποίησης των αλγορίθμων του πλαισίου, η οποία περιλαμβάνει τους αλγόριθμους Energy Planner, Green Planner, no-Rule, Meta-Rule και IFTTT.

Στο πέμπτο κεφάλαιο περιγράφεται η γραφική διαπροσωπεία του πλαισίου, στο οποίο μπορεί ένας χρήστης να δει τα αποτελέσματα του αλγόριθμου Energy Planner ή Green Planner αναλόγως, καθώς και να προσθέσει ή να αφαιρέσει κανόνες.

Στο έκτο κεφάλαιο περιγράφεται η μεθοδολογία που χρησιμοποιήθηκε για την εκτέλεση των διαφόρων πειραμάτων. Συγκεκριμένα περιγράφεται το testbed στο οποίο έγιναν τα πειράματα, οι μετρικές που χρησιμοποιήθηκαν και οι αλγόριθμοι που αξιολογήθηκαν. Επίσης περιγράφεται η αξιολόγηση της απόδοσης και τα συμπεράσματα που βγήκαν από την πειραματική αποτίμηση.

Τέλος, στο έβδομο κεφάλαιο αναφέρονται τα γενικά συμπεράσματα που προκύπτουν από την παρούσα διπλωματική εργασία, όπως και παράθεση κάποιων πιθανών μελλοντικών εργασιών για περεταίρω ανάπτυξη του πλαισίου.

Κεφάλαιο 2

Σχετική Εργασία

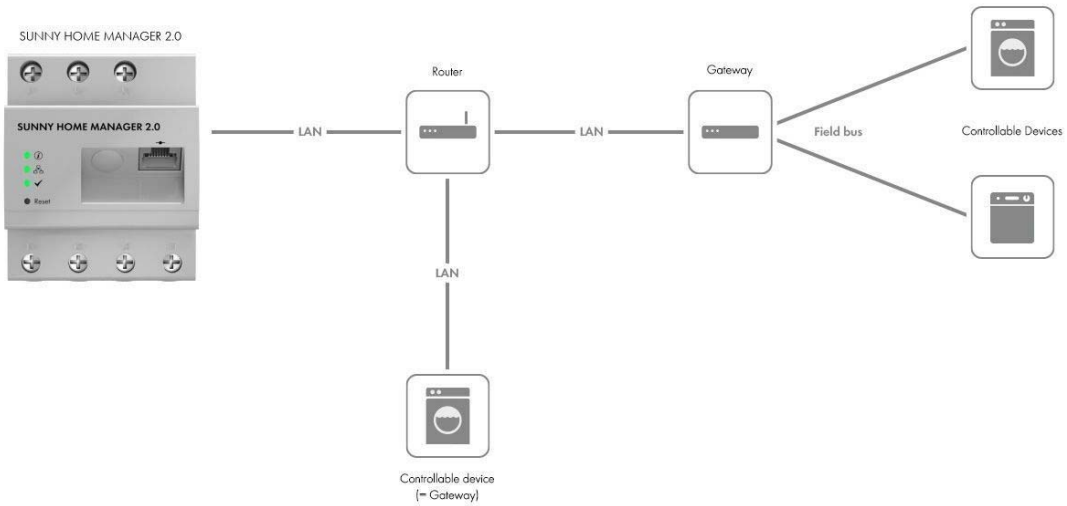
Σε αυτό το κεφάλαιο θα δούμε εν συντομία διάφορες τεχνολογίες οι οποίες χρησιμοποιούνται σήμερα για την διαχείριση έξυπνων συσκευών καθώς και τις αυτών των συστημάτων με το σύστημα που αναπτύχθηκε εντός αυτής της διπλωματικής εργασίας.

2.1 Έξυπνα συστήματα διαχείρισης ενέργειας

Η υιοθέτηση του διαδικτύου των πραγμάτων τα τελευταία χρόνια έφερε ένα αναζωογονημένο ενδιαφέρον για τη διαχείριση δεδομένων και τις λύσεις μηχανικής δεδομένων, τις αρχιτεκτονικές και τις εφαρμογές με έμφαση στην απορρόφηση δεδομένων, αναλυτικές αρχιτεκτονικές για ροή δεδομένων καθώς και σχετική συγκριτική αξιολόγηση. Κάποιες λύσεις αναφέρονται παρακάτω.

2.1.1 Διαχειριζόμενη οικιακή ενέργεια με φωτοβολταϊκά

Οι ελεγκτές Sunny Home Manager [4] από την SMA είναι συνδεδεμένοι με όλες τις συσκευές του σπιτιού. Μπορούν να διαχειριστούν την κατανάλωση ηλεκτρικού ρεύματος από τις συσκευές και να χρησιμοποιήσουν την παραγόμενη ηλεκτρική ενέργεια από τα φωτοβολταϊκά. Συγκεκριμένα παρακολουθούν τις ροές ισχύος, ιδιαίτερα την παραγωγή ισχύος AC από τους μετατροπείς και την κατανάλωση ισχύος AC από τα νοικοκυριά (καταγράφεται από έναν μετρητή ενέργειας). Στη συνέχεια, διαχειρίζεται ανάλογα τους φόρτους εργασίας κατανάλωσης ισχύος (π.χ. πότε πρέπει να λειτουργεί πλυντήριο ρούχων ή ο έξυπνος φορτιστής αυτοκινήτου, έτσι ώστε η αυτοκατανάλωση ηλιακής ενέργειας να βελτιστοποιείται). Οι ελεγκτές SAM χρησιμοποιούν το ανοικτό πρωτόκολλο απλής διαχείρισης ενέργειας (SEMP) [5] ή το EEBUS που αναφέρεται στη συνέχεια.



Σχήμα 2.1 Sunny Home Manager Schema [4]

2.1.2 Έξυπνοι θερμοστάτες

Ένα καλό παράδειγμα έξυπνου θερμοστάτη, είναι ο θερμοστάτης εκμάθησης Google Nest [6], είναι ένας προγραμματιζόμενος και αυτοδίδακτος θερμοστάτης με δυνατότητα Wi-Fi που βελτιστοποιεί την ψύξη και τη θέρμανση για εξοικονόμηση ενέργειας. Επίσης μπορεί να ειδοποιήσει με ηλεκτρονικό μήνυμα αν εντοπίσει πως χρειάζεται περισσότερο χρόνο από τον κανονικό για την θέρμανση ή ψύξη του σπιτιού.

Σχήμα 2.2 Έξυπνος Θερμοστάτης τύπου Nest [6]

2.1.3 Έξυπνες πρίζες με παρακολούθηση ενέργειας

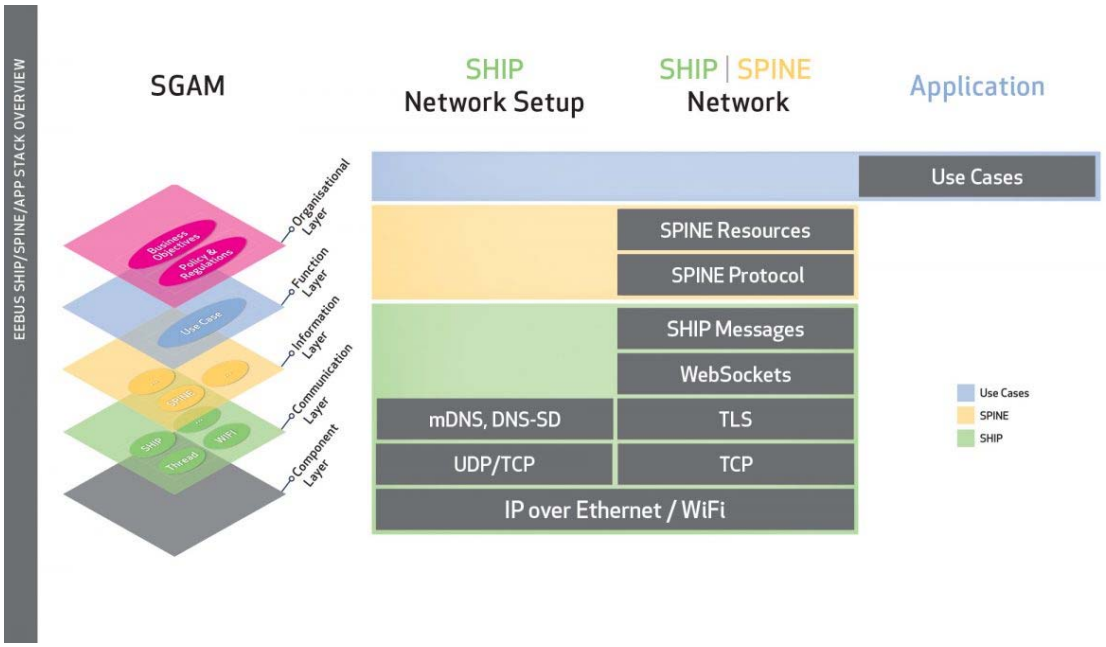
Οι πρίζες τύπου “Kasa Smart WiFi Plug Slim with Energy Monitoring” [7] της εταιρίας tp-link μπορούν να ενεργοποιηθούν από εφαρμογή στο κινητό, μπορούν να προστεθούν προγράμματα ενεργοποίησης και απενεργοποίησης και με την εφαρμογή στο κινητό σου δείχνει την κατανάλωση ενέργειας που έχουν ξοδέψει οι συσκευές που είναι συνδεμένες μέσω τέτοιων πριζών.

Σχήμα 2.3 Έξυπνη πρίζα [7]

2.1.4 Στοίβα πρωτοκόλλου EEBUS

Η στοίβα πρωτοκόλλου EEBUS [8] χρησιμοποιείται στο Διαδίκτυο των πραγμάτων για την επικοινωνία μεταξύ καταναλωτών ηλεκτρικής ενέργειας, παραγωγών, αποθηκών και διαχειριστικών οντοτήτων. Η σουίτα πρωτοκόλλου βασίζεται και αναπτύσσεται πάνω από το υφιστάμενο πρωτόκολλο διαδικτύου, είναι εξαιρετικά γενικό, μπορεί να εφαρμοστεί μεταξύ σε τομείς και είναι ανοιχτό και δωρεάν στο κοινό. Ο κύριος τομέας εφαρμογής του πρωτοκόλλου είναι η διαχείριση όμως ζήτησης ενέργειας, η ανταλλαγή δεδομένων και ο έλεγχος των συσκευών, όμως μπορεί να εφαρμοστεί για οικιακό αυτοματισμό.

Η αρχιτεκτονική του EEBUS βασίζεται στο αρχιτεκτονικό μοντέλο SGAM. Στο παρακάτω σχήμα φαίνεται η στοίβα EEBUS.

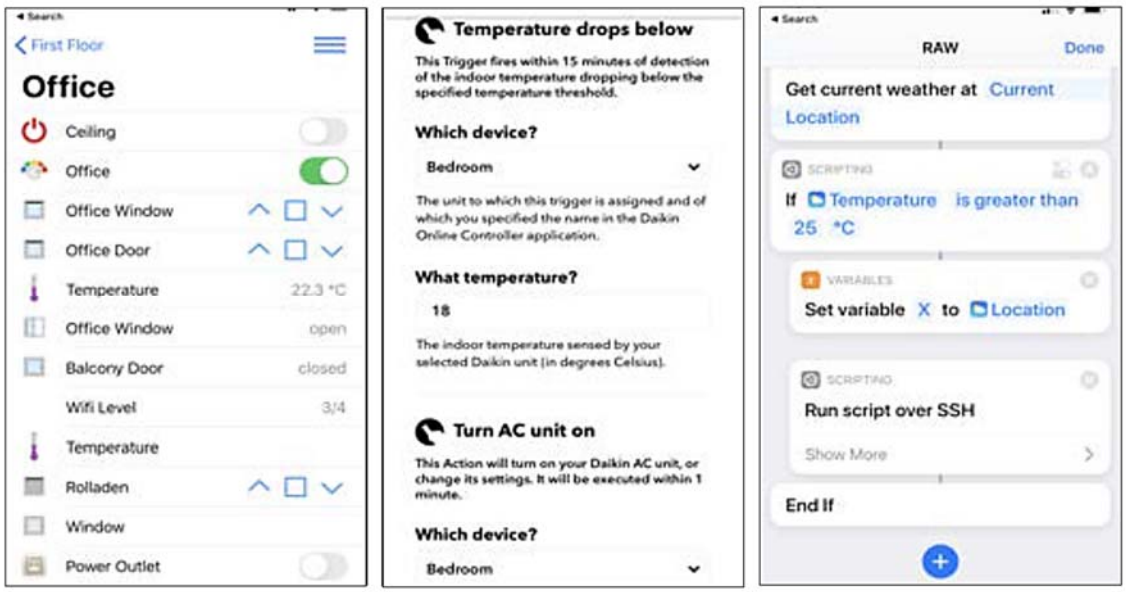


Σχήμα 2.4 EEBUS stack [9]

2.2 Ροές αυτοματισμού εργασίας (RAW)

Οι ροές εργασίας [10] επιτρέπουν των αυτοματισμό συγκεκριμένων καθημερινών εργασιών, ώστε να μην γίνονται χειροκίνητα. Μια ροή εργασίας (κανόνας αυτοματισμού) ορίζεται από κανόνες πυροδότησης (πότε θα ξεκινήσει ο κανόνας), συνθήκες (τα κριτήρια αντιστοίχισης για μια εντολή) και ενέργειες (τι εφαρμόζεται σε μια εντολή που ταιριάζει με τους καθορισμένους κανόνες ετικέτας).

Ένα παράδειγμα ροών εργασίας είναι οι "Κανόνες αποστολής". Οι κανόνες αποστολής επιτρέπουν την αντιστοιχία μεταφορέα, τη μέθοδο αποστολής και τον τύπο πακέτου με βάση την αντίστοιχη χώρα και πολιτεία.



Σχήμα 2.5 Παράδειγμα ροής αυτοματισμού εργασίας [11]

Οι ροές αυτοματισμού εργασίας (RAW) [11], εκτείνονται από απλές δηλώσεις έως διαδικαστικές ροές εργασίας με στόχο τη σύλληψη ενός έξυπνου αγωγού ενεργοποίησης σε εργαλεία που ελέγχουν τις διάφορες έξυπνες συσκευές.

2.2.1 IFTTT σε πραγματικό χρόνο

Το RT-IFTTT [12], είναι μια γλώσσα διαδικτύου των πραγμάτων όπου κατά την εφαρμογή της, χρησιμοποιεί εύκαμπτα διαστήματα ανίχνευσης αισθητήρων συνθηκών ενεργοποίησης σε πραγματικό χρόνο. Η γλώσσα RTIFTTT επεκτείνει την υπάρχουσα σύνταξη IFTTT και επιτρέπει στους χρήστες να καθορίσουν περιορισμούς σε πραγματικό χρόνο για τα τις έξυπνες συσκευές τους.

2.2.2 Μέθοδοι αναζήτησης με ενημέρωση RAW

Οι μέθοδοι RAW [11] χαρακτηρίζονται γενικά από ένα βοηθητικό πρόγραμμα στη σάρωση του χώρου λύσεων για την επίτευξη του στόχου. Αυτοί οι αλγόριθμοι

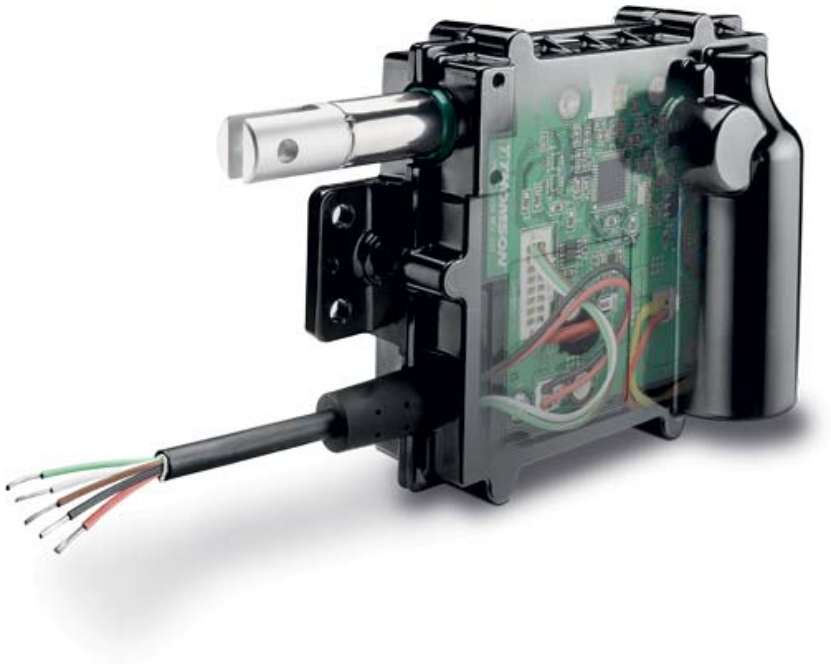
χρησιμοποιούν κάποιο είδος συνάρτησης αξιολόγησης που αξιολογεί λανθασμένα κάποια απόσταση της τρέχουσας λύσης από έναν στόχο.

2.3 Έξυπνη ενεργοποίηση

Οι ενεργοποιητές [13] είναι μηχανικές συσκευές που επιτρέπουν την εφαρμογή μιας συγκεκριμένης κίνησης ή δύναμης έμμεσα σε ένα προϊόν ή μια διαδικασία, και όχι χειροκίνητα. Ο έλεγχος των ενεργοποιητών μπορεί χειροκίνητα ή με τη βοήθεια λογισμικού ή μέσω από διεπαφές σε ένα υπολογιστή. Οι ενεργοποιητές που μπορούν να προγραμματιστούν είναι γνωστοί ως «έξυπνοι» ενεργοποιητές. Δύο συνήθεις τύποι ενεργοποιητών περιλαμβάνουν τη γραμμική ή την ευθεία κίνηση και την περιστροφική ή την κυκλική κίνηση.

Τα οφέλη [14] χρήσης έξυπνων ενεργοποιητών περιλαμβάνουν:

- Αυξημένη αποδοτικότητα και παραγωγικότητα.
- Βελτιωμένες διαγνωστικές δυνατότητες και δυνατότητα ελέγχου.
- Λιγότερα εξαρτήματα και λιγότερη καλωδίωση.
- Ελαχιστοποιημένη πολυπλοκότητα και ευκολότερη εγκατάσταση.
- Μειωμένο κόστος υλικού και λογισμικού.
- Μειωμένος χρόνος και βάρος ανάπτυξης μηχανών.
- Βελτιωμένη λειτουργικότητα και απόδοση του μηχανήματος.



Σχήμα 2.6 Έξυπνος ενεργοποιητής περιστροφικής κίνησης [14]

2.4 Διαφορές έξυπνων συστημάτων με το σύστημα που αναπτύχθηκε

Τα παραπάνω έξυπνα συστήματα διαχείρισης ενέργειας χρησιμοποιούν πρωτόκολλα τα οποία είναι προσανατολισμένα για την διαχείριση φορτίου μέσα σε έξυπνα κτήρια, χωρίς όμως να θέτουν μακροχρόνιους στόχους κατανάλωσης ενέργειας. Αντίθετα η υλοποίηση των αλγορίθμων Energy Planner και Green Planner που αναφέρονται στο κεφάλαιο 4, χρησιμοποιούν κανόνες που θέτονται από τους χρήστες για επίτευξη μακροπρόθεσμου στόχου κατανάλωσης ενέργειας.

2.5 Διαχείριση ζήτησης ενέργειας

Η διαχείριση ζήτησης ενέργειας [15], γνωστή και ως διαχείριση από πλευράς ζήτησης (DSM) ή απόκριση από πλευράς ζήτησης (DSR), είναι η τροποποίηση της ζήτησης ενέργειας για τους καταναλωτές μέσω διαφόρων μεθόδων, όπως οικονομικά κίνητρα και αλλαγή συμπεριφοράς μέσω της εκπαίδευσης.

Συνηθής στόχος της διαχείρισης ζήτησης είναι να ενθαρρύνει τον καταναλωτή να χρησιμοποιεί λιγότερη ενέργεια κατά τις ώρες αιχμής ή να μεταφέρει τον χρόνο χρήσης

ενέργειας σε ώρες εκτός αιχμής, όπως τη νύχτα και τα σαββατοκύριακα. Η μέγιστη διαχείριση της ζήτησης δεν μειώνει απαραίτητα τη συνολική κατανάλωση ενέργειας, αλλά αναμένεται να μειώσει την ανάγκη για επενδύσεις σε δίκτυα ή / και σταθμούς παραγωγής ενέργειας για την κάλυψη των αιχμών.

Με το σύστημα που αναπτύχθηκε επί τους σκοπούς της ατομικής διπλωματικής εργασίας ο στόχος της διαχείρισης ζήτησης, γίνεται πιο εφικτός διότι μειώνεται γενικότερα η κατανάλωση ενεργείας.

Types of DSM
Energy efficiency
Demand response
Dynamic demand
Distributed Energy Resources

Πίνακας 2.1 Τύποι Διαχείρισης Ενέργειας [15]

2.6 Αλγόριθμοι

Σε αυτή την υποενότητα αναφέρονται οι αλγόριθμοι στους οποίους βασίζονται οι αλγόριθμοι Energy Planner και Green Planner που υλοποιήθηκαν στα πλαίσια αυτής της διπλωματικής εργασίας.

2.6.1 Local Search

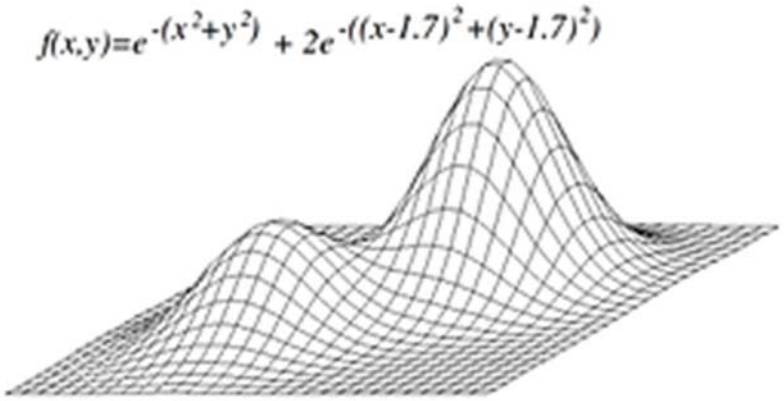
Η τοπική αναζήτηση [16] είναι μια ευρετική μέθοδος για την επίλυση υπολογιστικών σκληρών προβλημάτων βελτιστοποίησης. Η τοπική αναζήτηση μπορεί να χρησιμοποιηθεί σε προβλήματα που μπορούν να διατυπωθούν ως εξεύρεση λύσης μεγιστοποιώντας ένα κριτήριο μεταξύ ορισμένων υποψηφίων λύσεων. Οι τοπικοί αλγόριθμοι αναζήτησης μετακινούνται από λύση σε λύση στο χώρο των υποψηφίων λύσεων (ο χώρος αναζήτησης) εφαρμόζοντας τοπικές αλλαγές, έως ότου βρεθεί μια λύση που θεωρείται βέλτιστη ή έχει παρέλθει ένα χρονικό όριο.

Τα περισσότερα προβλήματα μπορούν να διατυπωθούν από άποψη χώρου αναζήτησης και στόχου με διάφορους τρόπους. Για παράδειγμα, για το πρόβλημα του πωλητή που ταξιδεύει, μια λύση μπορεί να είναι ένας κύκλος και το κριτήριο της μεγιστοποίησης είναι ένας συνδυασμός του αριθμού των κόμβων και της διάρκειας του κύκλου.

2.6.2 Αλγόριθμοι Hill Climbing

Hill climbing [17] είναι μια μαθηματική μέθοδος βελτίωσης η οποία ανήκει στην κατηγορία των τοπικών αναζητήσεων που αναφέρθηκαν πιο πάνω. Είναι ένας επαναληπτικός αλγόριθμος ο οποίος αρχίζει με μια αρχική λύση και σε προσπαθεί να βρει μια καλύτερη λύση κάνοντας μια αυξητική αλλαγή στην αρχική λύση. Ο αλγόριθμος συνεχίζει να προσθέτει αλλαγές μέχρι να μην υπάρχουν άλλες αυξητικές αλλαγές.

Γενικά οι τεχνικές τύπου Hill climbing είναι πολύ καλές σε κυρτά προβλήματα τα οποία έχουν ένα μέγιστο, δηλαδή μόνο μια καλύτερη λύση. Σε προβλήματα που υπάρχουν τοπικά μέγιστα ο αλγόριθμος δεν θα βρει απαραίτητα ως καλύτερη λύση το ολικό μέγιστο αλλά είναι πολύ πιθανόν να βρει ένα αυτά ως καλύτερη λύση. Για την λύση αυτού του προβλήματος χρησιμοποιούνται κάποιες τεχνικές όπως επανεκκινήσεις και simulated annealing.



Σχήμα 2.7 Επιφάνεια με 2 τοπικά μέγιστα [17]

Ένα παράδειγμα χρήσης είναι στο πρόβλημα του πλανόδιου πωλητή [18]. Ο αλγόριθμος ξεκινώντας από μια μη βέλτιστη λύση και κάνει μικρές βελτιώσεις σε αυτή, όπως η αλλαγή της σειράς με την οποία επισκέπτονται δύο πόλεις και τελικά μια πολύ μικρότερη διαδρομή είναι πιθανό να επιτευχθεί.

Αλγόριθμος Simple Hill Climbing [19]:

Βήμα 1: Αξιολογήστε την αρχική κατάσταση, εάν είναι κατάσταση στόχου, τότε επιστρέψτε την επιτυχία και σταματήστε.

Βήμα 2: Βρόχος μέχρι να βρεθεί λύση ή δεν υπάρχει νέος χειριστής για να υποβάλει αίτηση.

Βήμα 3: Επιλέξτε και εφαρμόστε έναν τελεστή στην τρέχουσα κατάσταση.

Βήμα 4: Έλεγχος νέας κατάστασης:

Εάν είναι κατάσταση στόχου, επιστρέψτε την επιτυχία και σταματήστε.

Διαφορετικά, εάν είναι καλύτερη από την τρέχουσα κατάσταση, τότε εκχωρήστε νέα κατάσταση ως τρέχουσα κατάσταση.

Διαφορετικά, αν όχι καλύτερα από την τρέχουσα κατάσταση, επιστρέψτε στο βήμα 2.

Βήμα 5: Έξοδος.

Στον πίνακα παρακάτω αναφέρονται διάφορες παραλλαγές του αλγορίθμου hill climbing.

Παραλλαγές της Μεθόδου Hill Climbing
Simple hill climbing
Stochastic hill climbing
Steepest ascent hill climbing
Random-restart hill climbing

Πίνακας 2.2 Παραλλαγές Hill Climbing [17]

2.6.3 Simulated Annealing

Η προσομοιωμένη ανόπτηση [20] (SA) είναι μια πιθανολογική τεχνική για την προσέγγιση του ολικού βέλτιστου όγκου μιας δεδομένης λειτουργίας. Συγκεκριμένα, είναι μια μεταευριστική προσέγγιση της ολικής βελτιστοποίησης σε ένα μεγάλο χώρο αναζήτησης για ένα πρόβλημα βελτιστοποίησης. Χρησιμοποιείται συχνά όταν ο χώρος αναζήτησης είναι διακριτός (π.χ., το πρόβλημα του πλανόδιου πωλητή).

Το όνομα του αλγορίθμου προέρχεται από την ανόπτηση στη μεταλλουργία, μια τεχνική που περιλαμβάνει θέρμανση και ελεγχόμενη ψύξη ενός υλικού για την αύξηση του μεγέθους των κρυστάλλων του και τη μείωση των ελαττωμάτων τους. Η προσομοιωμένη ανόπτηση μπορεί να χρησιμοποιηθεί για πολύ σκληρά προβλήματα υπολογιστικής βελτιστοποίησης όπου οι ακριβείς αλγόριθμοι αποτυγχάνουν, παρόλο που συνήθως επιτυγχάνει μια κατά προσέγγιση λύση στο ελάχιστο σε παγκόσμιο επίπεδο, θα μπορούσε να είναι αρκετή για πολλά πρακτικά προβλήματα.

Τα προβλήματα που επιλύονται από την SA διατυπώνονται επί του παρόντος από μια αντικειμενική συνάρτηση πολλών μεταβλητών, που υπόκεινται σε διάφορους περιορισμούς. Στην πράξη, ο περιορισμός μπορεί να τιμωρηθεί ως μέρος της αντικειμενικής λειτουργίας.

Σε κάθε βήμα, η SA ευρετική εξετάζει κάποια γειτονική κατάσταση s* της τρέχουσας κατάστασης και αποφασίζει πιθανώς μεταξύ της μετακίνησης του συστήματος στην κατάσταση s* ή της παραμονής στην κατάσταση. Αυτές οι πιθανότητες οδηγούν τελικά το σύστημα να μετακινηθεί σε καταστάσεις χαμηλότερης ενέργειας. Συνήθως αυτό το βήμα επαναλαμβάνεται έως ότου το σύστημα φτάσει σε μια κατάσταση που είναι αρκετά καλή για την εφαρμογή ή έως ότου εξαντληθεί ένας δεδομένος προϋπολογισμός υπολογισμού.

Η βελτιστοποίηση μιας λύσης περιλαμβάνει την αξιολόγηση των γειτόνων μιας κατάστασης του προβλήματος, οι οποίες είναι νέες καταστάσεις που παράγονται μέσω της συντηρητικής αλλαγής μιας δεδομένης κατάστασης. Για παράδειγμα, στο πρόβλημα του πλανόδιου πωλητή κάθε κράτος ορίζεται συνήθως ως παραλλαγή των πόλεων προς επίσκεψη, και οι γείτονες οποιασδήποτε πολιτείας είναι το σύνολο των παραλλαγών που

παράγονται με την ανταλλαγή δύο από αυτές τις πόλεις. Ο καλά καθορισμένος τρόπος με τον οποίο οι καταστάσεις αλλάζουν για να παράγουν γειτονικά κράτη ονομάζεται «κίνηση» και διαφορετικές κινήσεις δίνουν διαφορετικά σύνολα γειτονικών κρατών. Αυτές οι κινήσεις συνήθως οδηγούν σε ελάχιστες αλλαγές της τελευταίας κατάστασης, σε μια προσπάθεια βελτίωσης σταδιακά της λύσης μέσω της επαναληπτικής βελτίωσης των τμημάτων της (όπως οι συνδέσεις πόλεων στο πρόβλημα του ταξιδιώτη πωλητή).

Προκειμένου να εφαρμοστεί η προσομοιωμένη μέθοδος ανόπτησης σε ένα συγκεκριμένο πρόβλημα, πρέπει να προσδιοριστούν οι ακόλουθες παράμετροι: ο χώρος κατάστασης, η ενέργεια (στόχος) συνάρτηση E (), η υποψήφια διαδικασία γεννήτριας γείτονας (), η συνάρτηση πιθανότητας αποδοχής P (), και η θερμοκρασία προγραμματισμού ανόπτησης () και αρχική θερμοκρασία. Αυτές οι επιλογές μπορούν να έχουν σημαντικό αντίκτυπο στην αποτελεσματικότητα της μεθόδου.

Κεφάλαιο 3

Αρχιτεκτονική

Σε αυτό το κεφάλαιο αναλύεται η αρχιτεκτονική του συστήματος IMCF που υλοποιήθηκε το οποίο αποτελείται από 2 βασικά συστατικά τον τοπικό ελεγκτή (LC) και το IMCF συστατικό. Επίσης αναλύονται και τα δεδομένα εισόδου που χρησιμοποιούνται από τα συστατικά.

3.1 Τοπικός Ελεγκτής (LC)

Ο τοπικός ελεγκτής είναι ανεπτυγμένος στην γλώσσα προγραμματισμού JAVA και είναι σχεδιασμένος για εγκατάσταση σε μια μικροσυσκευή όπως το raspberry Pi, ο οποίος θα τρέχει στο τοπικό δίκτυο του χρήστη. Ο τοπικός ελεγκτής βρίσκεται σε άμεση

επικοινωνία με τις συσκευές του διαδικτύου των πραγμάτων για να τους δώσει εντολές βάση τις προτιμήσεις του χρήστη ή της ομάδας χρηστών του συστήματος IMCF. Επίσης ο τοπικός ελεγκτής είναι ανεπτυγμένος ως επέκταση του ανοιχτού λογισμικού για έξυπνο σπίτι, εν ονόματι OpenHab, το οποίο είναι ένα σύστημα μέσα από το οποίο ένας χρήστης μπορεί να αλληλοεπιδράσει με τις διάφορες έξυπνες συσκευές στο περιβάλλον του. Το OpenHab μπορεί να εγκατασταθεί με εφαρμογή στο κινητό, ούτως ώστε ο έλεγχος να γίνεται μέσω κινητού για ευκολία.

Για την καλύτερη κατανόηση της λειτουργίας του τοπικού ελεγκτή μπορεί να θεωρηθεί το ακόλουθο παράδειγμα. Έστω ότι ένας χρήστης θέλει να μειώσει την θερμοκρασία ενός κλιματιστικού στο από 27 βαθμούς κελσίου σε 25. Για επίτευξη αυτής της μείωσης της θερμοκρασίας ο χρήστης πρέπει να αλληλοεπιδράσει μέσω τοπικού ελεγκτή, ο οποίος τελικά θα επικοινωνήσει με το κλιματιστικό.

Για την επικοινωνία με τον τοπικό ελεγκτή εκτός τοπικού δικτύου χρειάζεται η σύνδεση με τον Cloud Controller (CC) ο οποίος ελέγχει τον τοπικό ελεγκτή από απόσταση ή η σύνδεση με το τοπικό δίκτυο μέσω ενός VPN. Ο Cloud Controller δεν έχει υλοποιηθεί στους σκοπούς αυτής της διπλωματικής εργασίας.

3.2 Το σύστημα IMCF

Το συστατικό IMCF είναι μια επέκταση του λογισμικού του τοπικού ελεγκτή, το οποίο επιτρέπει την προσαρμογή των προτιμήσεων ευκολίας για την επίτευξή των μακροπρόθεσμων ενεργειακών στόχων του χρήστη ή της ομάδας χρηστών. Έχει αναπτυχθεί με τέτοιο τρόπο που ενσωματώνει την υλοποίηση του αλγορίθμου Energy Planner και Green Planner, αλλά και της γραφικής διαπροσωπείας του χρήστη και της αποθήκευσης των πληροφοριών, που απαιτούνται για μπορεί ο χρήστης να αλληλοεπιδράσει με το σύστημα. Ο αλγόριθμος EP εφαρμόζεται ως βιβλιοθήκη JAVA και λαμβάνει τις ρυθμίσεις των χρηστών από ένα MariaDB persistence layer. Το επίπεδο της αποθήκευσης κατά τη χρήση του συστήματος, το οποίο έχει διαμορφωθεί ούτως ώστε ο ορισμός μέτα-κανόνων στον πίνακα μέτα-κανόνων μέσω να γίνεται απρόσκοπτα μέσω της γραφικής διαπροσωπείας.

Η γραφική διαπροσωπεία είναι αναπτυγμένη ως μια εφαρμογή Laravel PHP ακολουθώντας το αρχιτεκτονικό μοτίβο μοντέλο-προβολή-ελεγκτή και με χρήση

JavaScript και HTML. Η βιβλιοθήκη JAVA αποτελείται από περίπου 4500 γραμμές κώδικα και η γραφική διαπροσωπεία από 3000. Η εκτέλεση του κώδικα της γραφικής διαπροσωπείας γίνεται μέσω του διακομιστή ιστού NGINX το οποίο είναι διαθέσιμο στο Raspberry PI, ενώ για την Energy Planner και Green Planner τρέχει κάθε 30 λεπτά αξιόπιστα μέσω cronjob daemon.

Το σύστημα IMCF προσφέρει τις ακόλουθες επιλογές σε περίπτωση που πρέπει να γίνει ενεργοποίηση/απενεργοποίηση κάποιων συσκευών:

- Binding mode: χρήση του πλούσιου οικοσυστήματος διαθέσιμων γεφυρών από το OpenHAB για αλληλεπίδραση με τοπικές συσκευές. Αυτή η επιλογή είναι προεπιλεγμένη για το σύστημα IMCF.
- Extended mode: το σύστημα εφαρμόζει τοπικά προσαρμοσμένες οδηγίες για την ενεργοποίηση και απενεργοποίησης των έξυπνων συσκευών.

Το σύστημα IMCF προστατεύεται μέσω της ταυτοποίησης που προσφέρει το OpenHab και Laravel. Επίσης ο τοπικός ελεγκτής βρίσκεται στο τοπικό δίκτυο του χρήστη και προστατεύεται από το firewall του λειτουργικού συστήματος Ubuntu. Συνεπώς, το σύστημα είναι αρκετά ασφαλές, καθώς κάποιος με κακόβουλες προθέσεις θα πρέπει να διαπεράσει την ασφάλεια που προσφέρει το OpenHab και η Laravel για να μπορέσει να κάνει αλλαγές ή να διαπεράσει το firewall.

3.3 Δεδομένα εισόδου

Η υλοποίηση των αλγορίθμων EP και GP χρειάζονται ως δεδομένα εισόδου τους κανόνες από τον πίνακα μέτα-κανόνων MRT και ένα προφίλ κατανάλωσης ενέργειας ECP. Για κάθε εκτέλεση των αλγορίθμων το σύστημα χρησιμοποιεί τους ανανεωμένους μέτα-κανόνες τους οποίους λαμβάνει με αίτημα από το API του συστατικού IMCF. Το προφίλ κατανάλωσης ενέργειας αποτελείται από την μηνιαία κατανάλωση ενέργειας του χρήστη σε kWh. Επίσης για τον αλγόριθμο GP χρειάζονται και προβλέψεις για τον καιρό. Ακολουθεί πίνακας με παράδειγμα για το ECP [11].

Μήνες	kWh τον μήνα	kWh την ώρα
Ιανουάριος	775.50	1.04
Φεβρουάριος	528.75	0.71
Μάρτιος	246.75	0.33
Απρίλιος	141.00	0.19
Μάιος	176.25	0.24
Ιούνιος	211.50	0.28
Ιούλιος	246.75	0.33
Αύγουστος	317.25	0.43
Σεπτέμβριος	211.50	0.28
Οκτώβριος	176.25	0.24
Νοέμβριος	211.50	0.28
Δεκέμβριος	423.00	0.57

Πίνακας 3.1 Παράδειγμα ECP [11]

3.4 Ο αλγόριθμος IMCF

Ο αλγόριθμος IMCF [11] αποτελείται από δύο υπορουτίνες, το σχέδιο απόσβεσης (AP) και το Ενεργειακό Σχέδιο (EP) ή τον αλγόριθμο Green Planner (GP). Το σχέδιο απόσβεσης είναι υπεύθυνο για τον υπολογισμό του μέγιστου περιορισμού του ενεργειακού προϋπολογισμού μέσω ενός προεπιλεγμένου τύπου απόσβεσης. Στη συνέχεια εκτελείται μια προσέγγιση τεχνητής νοημοσύνης κάθε t δευτερόλεπτα για μια χρονική περίοδο p για τη δημιουργία μιας λύσης ενεργειακού σχεδίου s* για τη βελτιστοποίηση του σφάλματος άνεσης.

$$min_{FCE} = \sum_{k=1}^{t}(\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{D} ce_j(MR_i))$$

Εξίσωση 1

Όπου το ce_j είναι μεταξύ της επιθυμητής τιμής εξόδου $\Omega_i^j \epsilon R$ ενός κανόνα που ορίζεται από έναν χρήστη όπως τη θερμοκρασία ή την ένταση φωτός και την πραγματική τιμή $O_i^j \in R$ ορισμένη από τον ελεγκτή, δηλαδή $ce = |\Omega_i^j| - |O_I^j|$.

$$F_E = \sum_{k=1}^{t} (\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{D} e_j(MR_i))$$

Εξίσωση 2

Η κατανάλωση ενέργειας F_E πρέπει να είναι μικρότερη από τον συνολικό διαθέσιμο προϋπολογισμό ενέργειας E_P για το χρονικό διάστημα p κατά το οποίο εκτελείται ο αλγόριθμος. Στην εξίσωση 2 το N είναι αριθμός των μέτα-κανόνων, D είναι ο αριθμός των συσκευών διαδικτύου των πραγμάτων και το e_j είναι η κατανάλωση ενέργειας μιας συσκευής j δοσμένη από τη παραγωγή O_i^j του μέτα-κανόνα MR_i από:

$$E_j = \begin{cases} e_j, & O_i^j \ is \ executed \\ 0, & otherwise \end{cases}$$

Γ_j είναι η εκπομπή CO_2 που παράγεται από την ενεργοποίηση της συσκευής j και για τον υπολογισμό της χρησιμοποιείται η ενεργειακή κατανάλωση E της συσκευής και την ένταση εκπομπής γ CO_2 της χώρας.

$$\Gamma_j = \begin{cases} E_j, & if \ sunlight \\ 0, & otherwise \end{cases}$$

Υπόκειται στην ικανοποίηση του $F_\Gamma \leq \Gamma_p$ όπου,

$$F_\Gamma = \sum_{k=1}^{t} (\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{D} \Gamma_j(E_j(MR_i, \gamma)))$$

Εξίσωση 3

Η εκπομπή διοξειδίου του άνθρακα F_Γ πρέπει να είναι μικρότερη από την μέγιστη επιθυμητή εκπομπή CO_2 Γ_p, για ολόκληρη την περίοδο p κατά την οποία πραγματοποιείται η εκτέλεση του αλγορίθμου.

3.4.1 Σχέδιο απόσβεσης (AP)

Αρχικά εκτελείται το σχέδιο απόσβεσης AP [11] το οποίο είναι μια υπορουτίνα η οποία υπολογίζει τον περιορισμό του ενεργειακού προϋπολογισμού E_p, με χρήση του προφίλ κατανάλωσης ενέργειας ECP. Κάποιες στρατηγικές απόσβεσης έχουν ως ακολούθως:

i. Στρατηγική γραμμικής απόσβεσης (LAF), όπου η συνολική κατανάλωση ενέργειας TE μπορεί να κατανεμηθεί γραμμικά σε μια προκαθορισμένη χρονική περίοδο p, με διάρκεια t, η οποία μπορεί να οριστεί ως ετήσια, μηνιαία, καθημερινή, ωριαία και ούτω καθεξής, δίνοντας τον περιορισμό του ενεργειακού προϋπολογισμού:

$$E_p = \frac{TE}{t}$$

ii. Στρατηγική γραμμικής απόσβεσης μπαλονιών (BLAF), όπου ο χρήστης εξοικονομεί ένα ποσοστό π από τη συνολική κατανάλωση ενέργειας TE για ένα χρονικό διάστημα $\lambda < t$, το λεγόμενο μπαλόνι σ, το οποίο χρησιμοποιείται στην υπόλοιπη χρονική περίοδο $\lambda' = t - \lambda$ όπου η κατανάλωση ενέργειας είναι υψηλότερη. Ο περιορισμός ενεργειακού προϋπολογισμού E_p για μια περίοδο p διάρκειας t υπολογίζεται ως εξής:

$$E_p = \begin{cases} \frac{TE}{t} - \frac{\sigma}{\lambda}, for\ \lambda\ period \\ \frac{TE}{t} + \frac{\sigma}{\lambda}, for\ \lambda' period \end{cases}$$

iii. Στρατηγική απόσβεσης με βάση το ECP (EAF), όπου υπολογίζεται ένα σύνολο βαρών με χρήση του διανύσματος ECP, τα οποία χρησιμοποιούνται για τον καθορισμό των περιορισμών του ενεργειακού προϋπολογισμού για μια περίοδο που καθορίζεται από τον χρήστη σε ένα διαθέσιμο ενεργειακό προϋπολογισμό E:

$$E_p = \left\{\frac{w_i \times E}{t/|ECP|}\right\}, for\ i = 1, \dots, |ECP|,$$

$$\text{where } w_i = \frac{TE}{ECP_i}\ and\ \sum_{i=1}^{|ECP|} w_i = 1$$

Το TE είναι η συνολική κατανάλωση ενέργειας που προέρχεται από το ECP, E είναι ο διαθέσιμος προϋπολογισμός ενέργειας που καθορίζεται από το χρήστη, $|ECP|$ είναι το μέγεθος του διανύσματος του προφίλ κατανάλωσης ενέργειας και $t/|ECP|$ ομαλοποιεί τον ενεργειακό προϋπολογισμό βάσει της χρονικής διάρκειας κοκκοποίησης t.

Για όλες τις στρατηγικές απόσβεσης η μέγιστη εκπομπή CO_2 Γ_p υπολογίζεται ως

$$\Gamma_p = E_p \times \gamma$$

Και ο περιορισμός F_Γ υπολογίζεται απευθείας από την Εξίσωση 3.

3.4.2 Αρχιτεκτονική αλγόριθμου ενεργειακού σχεδίου (EP)

Σε αυτή την υποενότητα αναλύεται ο αλγόριθμος EP και τα σχετικά στοιχεία και παραμέτρους του.

Algorithm 1 $IMCF$: generates an energy-efficient plan

Input: MRT: Meta-Rule Table; k: components to be modified; τ_{max}: max iterations; t: time granularity; apl: amortization plan; ECP: Energy Consumption Profile

Output: An energy plan solution $s^* = (s_1, \ldots, s_N)$

1: **AP**(apl, p, ECP) ▷ **Amortization Plan Routine**
2: **switch** (apl)
3: a: $E_p \leftarrow LAF(t, ECP)$ ▷ use linear Eq. (3)
4: b: $E_p \leftarrow BLAF(t, ECP)$ ▷ use balloon Eq. (4)
5: c: $E_p \leftarrow EAF(t, ECP)$ ▷ use ECP-based Eq. (5)
6:
7: **EP**$(MRT, k, \tau_{max}, i, E_p)$ ▷ **Energy Plan Routine**
8: $s^* \leftarrow init_i(MRT)$ ▷ s^*: initial solution for time i
9: $(F_E, F_{CE}) \leftarrow evaluate(s^*)$ ▷ with Equations (1),(2)
10: **While** $\tau < \tau_{max}$ **do** ▷ τ: current iteration
11: $s \leftarrow optimization(s^*)$ ▷ randomly select k positions and swap their binary value
12: $(F_E, F_{CE}) \leftarrow evaluate(s)$ ▷ with Equations (1),(2)
13: **If** $(F_E(s) \leq E_p)$ && $(F_{CE}(s) < F_C(s^*))$ **then**
14: $s^* \leftarrow s$ ▷ Set s as the current solution s^*
15: **EndIf**
16: $\tau++$ ▷ Increase iterations
17: **EndWhile**
18: return s^* ▷ Return the final energy plan solution
19:
20: $apl \leftarrow AP(apl, t, ECP)$;
21: **return** $(\forall_i^t\ EP(MRT, k, \tau_{max}, i, apl))$

Σχήμα 3.1 Ο αλγόριθμος IMCF με χρήση της EP [11]

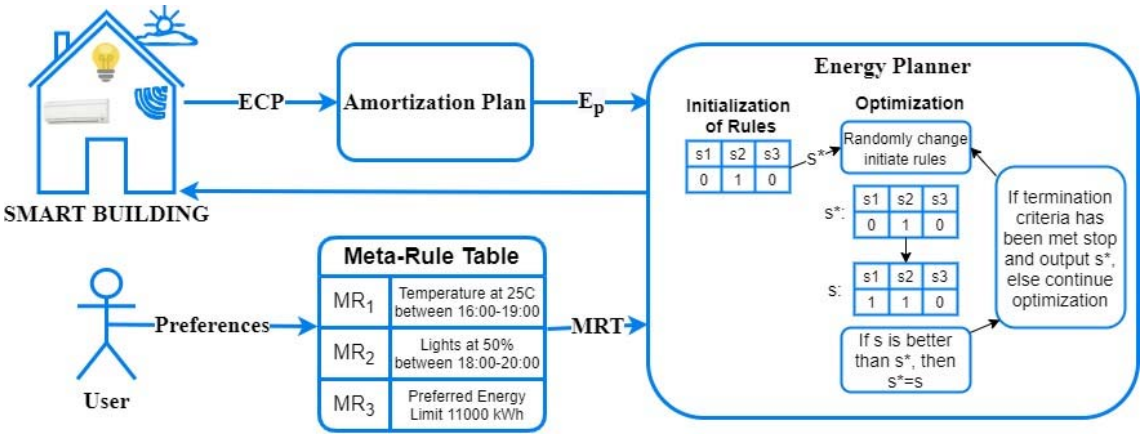
Μια λύση ενεργειακού σχεδίου είναι ένα διάνυσμα $s = \{s_1, s_2, s_3, \ldots, s_N\}$, όπου το N είναι ο αριθμός των μέτα-κανόνων. Ένα στοιχείο s_i αντιπροσωπεύει έναν μέτα-κανόνα στον πίνακα MRT, όπου $s_i = 0$ σημαίνει αγνόηση του μέτα-κανόνα i του πίνακα MRT και $s_i = 1$ σημαίνει χρήση του μέτα-κανόνα i.

Στην αρχή της τοπικής ευρετικής αναζήτησης αναπτύσσεται μια αρχική λύση s* που καθορίζει την αρχική κατάσταση του αλγορίθμου. Μια αρχική λύση μπορεί να δημιουργηθεί τυχαία ή ντετερμινιστικά. Μια ντετερμινιστική λύση για το EP είναι να οριστούν όλοι οι κανόνες s_i σε 1, πράγμα που σημαίνει ότι όλοι οι μέτα-κανόνες θα χρησιμοποιηθούν για την ανάπτυξη της λύσης s*, ευνοώντας με αυτόν τον τρόπο το σφάλμα ευκολίας, αλλά με μεγάλη πιθανότητα παραβίασης του περιορισμού E_p. Στην περίπτωση τυχαίας αρχικοποίησης, οι τιμές όλων των στοιχείων του διανύσματος επιλέγονται τυχαία.

Η βελτιστοποίηση επιτυγχάνεται με ευρετική τοπική αναζήτηση αναρρίχησης λόφου όπου χρησιμοποιείται για τοπική βελτιστοποίηση με γειτονιές που περιλαμβάνουν αλλαγή έως και k συστατικών της λύσης, η οποία αναφέρεται και ως k-opt. Κατά τη διαδικασία βελτιστοποίησης k, οι μέτα-κανόνες επιλέγονται τυχαία από το διάνυσμα s εάν θα αγνοηθούν ή θα χρησιμοποιηθούν.

Κάθε λύση s αξιολογείται χρησιμοποιώντας τις μετρήσεις απόδοσης F_E και F_{CE} των Εξισώσεων Εξίσωση 1 και Εξίσωση 2. Μια λύση s θεωρείται καλύτερη και αντικαθιστά την τρέχουσα καλύτερη λύση s* εάν η κατανάλωση ενέργειας F_E είναι μικρότερη από τον συνολικό διαθέσιμο προϋπολογισμό ενέργειας E_P και αν το συνολικό ποσοστό λάθους για τη λύση s είναι μικρότερο από τη λύση s^*.

Ο αλγόριθμος ενεργειακού σχεδίου τερματίζει όταν φτάσει στις τ_{max} επαναλήψεις. Εναλλακτικά, συνεχίζει μέχρι να μην υπάρχουν καλύτερες λύσεις. Σε περίπτωση που υπάρχουν ελλείψεις στη γνώση σχετικά με τη βέλτιστη λύση, ο αλγόριθμος μπορεί να μπει σε ατέρμονο βρόγχο.



Σχήμα 3.2 Παράδειγμα εκτέλεσης του IMCF με EP

3.4.3 Αρχιτεκτονική αλγόριθμου Green Planner (GP)

Σε αυτή την υποενότητα αναλύεται ο αλγόριθμος GP [21] και τα σχετικά στοιχεία και παραμέτρους του.

Algorithm 1 *IMCF*: generates a green-plan

Input: *MRT*: Meta-Rule Table; k: components to be modified; τ_{max}: max iterations; t: time granularity; *apl*: amortization plan; *ECP*: Energy Consumption Profile; T:Temperature
Output: An green plan solution $s^* = (s_1, \ldots, s_N)$

AP(apl, p, ECP) ▷ **Amortization Plan Routine**
  switch (apl)
  a: $E_p \leftarrow LAF(t, ECP)$ ▷ use linear Eq. (4)
  b: $E_p \leftarrow BLAF(t, ECP)$ ▷ use balloon Eq. (5)
  c: $E_p \leftarrow EAF(t, ECP)$ ▷ use *ECP*-based Eq. (6)
  return $\Gamma_p = E_p \times \gamma$ ▷ max CO_2 emission for γ CO_2 emission intensity

GP$(MRT, k, \tau_{max}, i, \Gamma_p)$ ▷ **Green Plan Routine**
  $s^* \leftarrow init_i(MRT)$ ▷ s^*: initial solution for time i
  $(F_{CE}, F_E, F_T) \leftarrow evaluate(s^*)$ ▷ with Equations (1),(2),(3)
  While $\tau < \tau_{max}$ **do** ▷ τ: current iteration
    $s \leftarrow optimization(s^*)$ ▷ randomly select k positions and swap their binary value
  $(F_{CE}, F_E, F_T) \leftarrow evaluate(s)$ ▷ with Equations (1),(2),(3)
    If $(F_E(s) \leq E_p)$ && $(F_{CE}(s) < F_{CE}(s^*))$ && $(F_T(s) < \Gamma_p)$ **then**
      $s^* \leftarrow s$ ▷ Set s as the current solution s^*
    Else If $rand(0,1) \leq P(F_{CE}(s^*), F_{CE}(s), T)$ && $(F_{CE}(s) < F_{CE}(s^*))$ && $(F_T(s) < \Gamma_p)$ **then**
      $s^* \leftarrow s$ ▷ Set s as the current solution s^*
    EndIf
    $\tau++$ ▷ Increase iterations
  EndWhile
  return s^* ▷ Return the final green plan solution

$\Gamma_p \leftarrow AP(apl, t, ECP)$;
return $(\forall_i^t\ GP(MRT, k, \tau_{max}, i, \Gamma_p))$

Σχήμα 3.3 Ο αλγόριθμος IMCF με χρήση της GP [21]

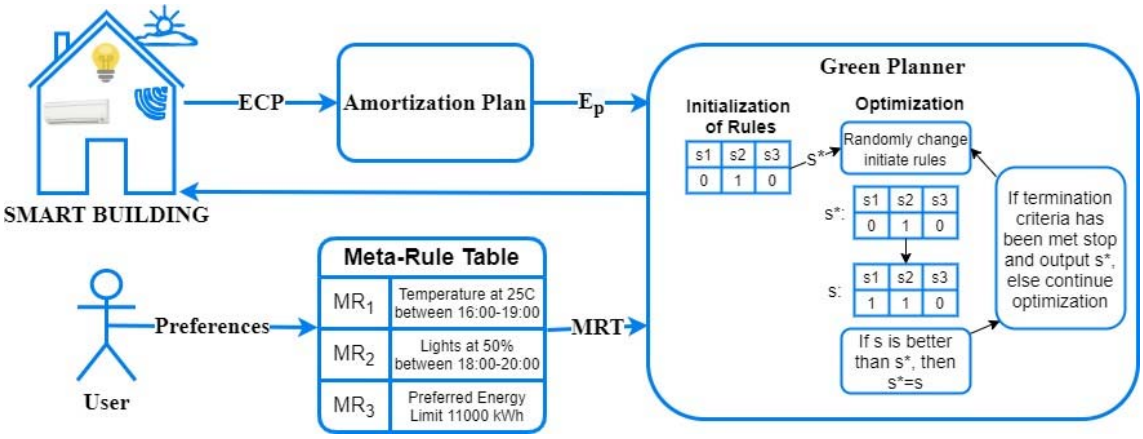
Μια πράσινη λύση ενεργειακού σχεδίου είναι ένα διάνυσμα $s = \{s_1, s_2, s_3, \dots, s_N\}$ όπου το N είναι ο αριθμός των μέτα-κανόνων. Ένα στοιχείο s_i αντιπροσωπεύει έναν μέτα-κανόνα στον πίνακα MRT, όπου $s_i = 0$ σημαίνει αγνόηση του μέτα-κανόνα i του πίνακα MRT και $s_i = 1$ σημαίνει χρήση του μέτα-κανόνα i.

Στην αρχή της αναπτύσσεται μια αρχική λύση s* που καθορίζει την αρχική κατάσταση του αλγορίθμου Μια ντετερμινιστική λύση για το GP είναι να οριστούν όλοι οι κανόνες s_i σε 1, πράγμα που σημαίνει ότι όλοι οι μέτα-κανόνες θα χρησιμοποιηθούν για την ανάπτυξη της λύσης s*, ευνοώντας με αυτόν τον τρόπο το σφάλμα ευκολίας, αλλά με μεγάλη πιθανότητα παραβίασης του περιορισμού Γ_p. Στην περίπτωση τυχαίας αρχικοποίησης, οι τιμές όλων των στοιχείων του διανύσματος επιλέγονται τυχαία.

Για το βήμα βελτιστοποίησης, χρησιμοποιείται μια ευρετική προσομοιωμένης ανόπτησης για τη μετατροπή της λύσης της τρέχουσας κατάστασης s* σε λύση μιας νέας κατάστασης s επιλέγοντας τυχαία και ανταλλάσσοντας έως k στοιχεία του s*. Η πιθανότητα πραγματοποίησης της μετάβασης από την τρέχουσα κατάσταση s* σε μια υποψήφια νέα κατάσταση s καθορίζεται από μια συνάρτηση πιθανότητας αποδοχής P (e (s*), e (s), T), όπου e (s*) και e (s) είναι η ενέργεια (ή αξιολόγηση καταλληλόλητας) των λύσεων που παράγονται σε καταστάσεις s* και s, αντίστοιχα, και T είναι μια παγκόσμια παράμετρος που ποικίλλει από το χρόνο, που ονομάζεται θερμοκρασία. Η θερμοκρασία T παίζει καθοριστικό ρόλο στον έλεγχο της εξέλιξης της κατάστασης s, δεδομένου ότι μια μεγάλη T ευνοεί τις χοντρές παραλλαγές μεταξύ των καταστάσεων, όπου μια μικρή T ευνοεί περισσότερες παραλλαγές λεπτών κόκκων. Η παράμετρος T μπορεί να ρυθμιστεί είτε μέσω ενός προκαθορισμένου χρονοδιαγράμματος ανόπτησης είτε να υπολογιστεί σε σχέση με τον αριθμό των επαναλήψεων που έχουν περάσει (δηλαδή, $T = \frac{\tau+1}{\tau max}$). Σε κάθε περίπτωση, απαιτείται σταδιακή μείωση του T καθώς προχωρά η προσομοίωση. Μια μετάβαση γίνεται πάντα αποδεκτή όταν το s είναι καλύτερο από το s* και είναι πιθανώς αποδεκτό, με πιθανότητα P (e (s^*), e (s), T) όταν s είναι χειρότερο από το s*, επιτρέποντας με αυτόν τον τρόπο την ευρετική αποφυγή τοπικών βέλτιστων.

Κάθε λύση s αξιολογείται χρησιμοποιώντας τις μετρήσεις απόδοσης F_{CE}, F_E και $F_Γ$ από τις εξισώσεις 1, 2, 3 αντίστοιχα. Μια λύση s θεωρείται καλύτερη και αντικαθιστά την τρέχουσα καλύτερη λύση s^* εάν η κατανάλωση ενέργειας F_E είναι μικρότερη από τον συνολικό διαθέσιμο προϋπολογισμό ενέργειας E_P, αν το συνολικό ποσοστό λάθους ευκολίας για τη λύση s είναι μικρότερο από τη λύση s^* και εάν η εκπομπή διοξειδίου του άνθρακα $F_Γ$ είναι μικρότερη από την μέγιστη επιθυμητή εκπομπή $Γ_p$. Διαφορετικά, εάν το s είναι χειρότερο από το s^*, η s αντικαθιστά την s^* αν μια τυχαία τιμή $rand$ στο διάστημα $[0,1]$ είναι μικρότερη από την πιθανότητα $P(e(s^*), e(s), T)$.

Ο αλγόριθμος πράσινου σχεδίου τερματίζει όταν φτάσει στις τ_{max} επαναλήψεις. Εναλλακτικά, συνεχίζει μέχρι να μην υπάρχουν καλύτερες λύσεις. Σε περίπτωση που υπάρχουν ελλείψεις στη γνώση σχετικά με τη βέλτιστη λύση, ο αλγόριθμος μπορεί να μπει σε ατέρμονο βρόγχο.



Σχήμα 3.4 Παράδειγμα εκτέλεσης του IMCF με GP

3.5 Οι βασικές προσεγγίσεις

Για σκοπούς αποτίμησης των αλγορίθμων δημιουργήθηκαν κάποιες βασικές προσεγγίσεις οι οποίες αναφέρονται σε αυτή την υποενότητα.

3.5.1 Βασική προσέγγιση χωρίς κανόνες (NR)

Είναι η πρώτη βασική προσέγγιση, όπου αγνοεί όλους τους μέτα-κανόνες του πίνακα μέτα-κανόνων και δεν τροποποιεί τη συμπεριφορά των αυτόνομων συσκευών, επομένως αυτό έρχεται σε αντίθεση με την ευκολία του χρήστη. Η συνολική κατανάλωση ενέργειας F_E είναι προφανώς πάντα 0, δεδομένου ότι καμία συσκευή διαδικτύου των πραγμάτων δεν είναι ενεργοποιημένη. Από την άλλη, το F_{CE} μετριέται ως ποσοστό σφάλματος ευκολίας που θα είχε ένας χρήστης εάν αυτός ο χρήστης εκτέλεσε όλους τους κανόνες και το F_T είναι μόνο το κόστος της πραγματοποίησης αυτού του υπολογισμού. Κατά συνέπεια, η κατανάλωση ενέργειας αυτής της προσέγγισης είναι ελάχιστη και το σφάλμα ευκολίας είναι το μέγιστο.

3.5.2 Βασική προσέγγιση με χρήση όλων των μέτα-κανόνων (MR)

Είναι η δεύτερη βασική προσέγγιση, όπου χρησιμοποιεί όλους τους μέτα-κανόνες του πίνακα μέτα-κανόνων και ευνοεί την ευκολία του χρήστη. Η συνολική κατανάλωση ενέργειας F_E είναι μέγιστη σε αυτή την περίπτωση ενώ το ποσοστό σφάλματος F_{CE} είναι ελάχιστο.

Κεφάλαιο 4

Υλοποίηση Αλγορίθμων

Σε αυτό το κεφάλαιο αναλύεται η υλοποίηση των διάφορων αλγόριθμων σε βιβλιοθήκη JAVA. Η βιβλιοθήκη αποτελείται από τις κλάσεις importData όπου χρησιμοποιείται για την εισαγωγή των απαραίτητων δεδομένων για εκτέλεση των αλγόριθμων, την κλάση noIFTT, την κλάση EnergyPlanner στην οποία υλοποιήθηκε ο αλγόριθμος EP και την κλάση GreenPlanner στην οποία υλοποιήθηκε ο αλγόριθμος GP.

4.1 Εισαγωγή dataset

Το σύνολο δεδομένων για διαμέρισμα ή κατοικία ή για φοιτητικές εστίες εισάγεται στο σύστημα μέσω ανάγνωσης του αρχείου και τοποθετείτε σε ένα ArrayList το οποίο χρησιμοποιείται από τους αλγόριθμους.

```
1.  while (inputFile.hasNextLine()) {
2.    line = inputFile.nextLine();
3.    String temp[] = new String[7];
4.
5.    if (counter > 1) {
6.      String[] col = line.split("\\|");
7.      temp[0] = col[0].trim();
8.      temp[1] = col[1].trim();
9.      temp[2] = col[2].trim();
10.     temp[3] = col[3].trim();
11.     temp[4] = col[4].trim();
12.     temp[5] = col[5].trim();
13.     temp[6] = col[6].trim();
14.     // fills a list with all the input IoT data
15.     IotData.add(temp);
16.    }
17.
18.   counter++;
19. } // end of while loop
```

Μέσω του While loop διαβάζονται όλες οι γραμμές του αρχείου και χωρίζοντας κάθε γραμμή σε string token όπως ενδείκνυται στην εντολή της γραμμής 6, αναθέτονται σε ένα προσωρινό πίνακα String όπου στη συνέχεια τοποθετείται στο ArrayList στην γραμμή 15. Προχωρώντας στην επόμενη καταχώρηση του συνόλου δεδομένων έχουμε ένα μετρητή με όνομα counter ο οποίος αυξάνεται σταδιακά.

4.2 Ενημέρωση των μέτα-κανόνων από το σύστημα IMCF

Οι μέτα-κανόνες ανανεώνονται πριν κάθε εκτέλεση και για να γίνει αυτό κατεβάζοντας τους από το σύστημα IMCF με χρήση API σε μορφή JSON και μετά από απαραίτητη επεξεργασία γράφονται στο αρχείο μέτα-κανόνων.

```
1.  String urlString = "http://10.16.30.73/api/get-metarules";
2.  StringBuilder result = new StringBuilder();
3.  URL url = new URL(urlString);
4.  HttpURLConnection conn = url.openConnection();
5.  conn.setRequestMethod("GET");
6.  conn.connect();
7.  int responsecode = conn.getResponseCode();
8.
9.  if (responsecode == 200) {
10.   BufferedReader br = new BufferedReader(conn);
11.   String line;
12.   while ((line = br.readLine()) != null) {
13.      result.append(line);
14.   }
15. }
```

Στη γραμμή 1 δηλώνεται ο σύνδεσμος για το API όπου στη συνέχεια με αίτημα GET στο API λαμβάνεται ο πίνακας μέτα-κανόνων και διαβάζεται από το While loop στη γραμμή 12-14. Έπειτα γίνεται επεξεργασία του πίνακα ο οποίος λήφθηκε σε μορφή JSON και αποθηκεύεται στο αρχείο των μέτα-κανόνων. Οι μέτα-κανόνες λαμβάνονται από τη τοπική βάση δεδομένων του πλαισίου μας MariaDB.

4.3 Εισαγωγή των μέτα-κανόνων στο σύστημα

Η εισαγωγή των μέτα-κανόνων γίνεται με την ανάγνωση του αρχείου μέτα-κανόνων αφού ενημερωθεί όπως έχει αναφερθεί πιο πάνω. Οι κανόνες μπαίνουν σε ένα ArrayList για χρήση τους από τους αλγορίθμους.

```
1.  File file = new File(pathMRTrules);
2.  // loop through every line
3.  while (inputFile.hasNextLine()) {
4.    line = inputFile.nextLine();
5.    String[] col = line.split("\\|");
6.    String temp[] = new String[4];
7.
8.    if (counter > 1) {
9.      temp[0] = col[0].trim();
10.     temp[1] = col[2].trim();
11.     temp[2] = col[3].trim();
12.     temp[3] = col[1].trim();
13.     MRTrules.add(temp);
14.   }
15.   counter++;
16. } // end of while loop
```

Αφού ανοιχτεί το αρχείο διαβάζεται κάθε γραμμή από το while loop (γραμμή 3-12) και αφού κατακερματιστεί κάθε γραμμή, τα δεδομένα κάθε γραμμής αποθηκεύονται στο ArrayList στην γραμμή 12, έτσι ώστε να χρησιμοποιηθούν για επεξεργασία από το πλαίσιο μας.

4.4 Υλοποίηση αλγόριθμου noRule / noIFTTT

Ο αλγόριθμος no-Rule υπολογίζει το μέγιστο (F_{CE}). Ο αλγόριθμος εκτελείται πάντα πριν τον αλγόριθμο EP και GP γιατί γίνεται χρήση του μέγιστου F_{CE} από αυτούς. Αυτό γίνεται με χρήση ενός σύνολού δεδομένων για διαμέρισμα ή κατοικία ή για φοιτητικές εστίες και τη χρήση των μέτα-κανόνων. Για κάθε μέτρηση από το σύνολο δεδομένων ελέγχονται όλοι οι μέτα-κανόνες, ούτως ώστε να μετρηθεί το σφάλμα λάθους σε περίπτωση παραβίασης κάποιου από αυτούς.

Οι έλεγχοι έχουν ως ακολούθως:

Αρχικά ελέγχεται αν η μέτρηση περιλαμβάνει θερμοκρασία ή ένταση φωτός.

```
// gets T for temperatures
if (dataSplitted[2].startsWith("T")) {
// gets LS for light sensor
if (dataSplitted[2].startsWith("LS")) {
```

Εάν γίνεται χρήση των μέτα-κανόνων τύπου IFTTT χρησιμοποιείται ο μήνας στον οποίο λήφθηκε η μέτρηση για να βρούμε σε ποια εποχή λήφθηκαν τα δεδομένα.

```
int iSeason = Integer.parseInt(dataSplitted[0]);
```

Στη συνέχεια γίνεται έλεγχος σε ποια εποχή βρισκόμαστε, δηλαδή αν ο μήνας που λήφθηκαν τα δεδομένα είναι χειμώνας ή καλοκαίρι.

```
// checks if SUMMER
if ((iSeason == 6) || (iSeason == 7) || (iSeason == 8)) {
// checks if WINTER
else if ((iSeason == 12) || (iSeason == 1) || (iSeason == 2)) {
```

Αν δεν ισχύει η προτιμώμενη ένταση φωτός ή η προτιμώμενη θερμοκρασία για την ώρα που έχει τεθεί ή για την εποχή που έχει τεθεί τότε αυξάνεται το συνολικό λάθος ευκολίας με τη διαφορά των ποσών από τη μέτρηση και τον μέτα-κανόνα.

4.5 Υλοποίηση του αλγόριθμου Energy Planner

Ο αλγόριθμος EP χρησιμοποιεί το σύνολο δεδομένων για διαμέρισμα ή κατοικία ή για φοιτητικές εστίες, τον πίνακα μέτα-κανόνων και το μέγιστο λάθος που υπολογίζεται από τον αλγόριθμο No-Rule. Ο αλγόριθμος υπολογίζει το σφάλμα ευκολίας (F_{CE}) και την κατανάλωση ενέργειας (F_E).

```
1.  for (int i = 0; i < iotdata.size(); i++) {
2.
3.    String[] dataSplitted = iotdata.get(i);
4.    double tempHourChange = getHour(dataSplitted[0]);
5.
6.    if (hourChange == tempHourChange) {
7.
8.      if (i == iotdata.size() - 1) {
9.        recordsHourly.add(iotdata.get(i));
10.       double hourly_ms = calculateKWH(recordsHourly);
11.       total_ms = total_ms + hourly_ms;
12.       recordsHourly.clear();
13.     } else {
14.       recordsHourly.add(iotdata.get(i));
15.     }
16.
17.   } else {
18.     hourChange = tempHourChange;
19.     hourly_ms = calculateKWH(recordsHourly);
20.     total_ms = total_ms + hourly_ms;
21.     recordsHourly.clear();
22.     recordsHourly.add(iotdata.get(i));
23.   }
24.
25. }
```

Για κάθε καταγραφή από το σύνολο δεδομένων βλέπουμε πότε αλλάζει η ώρα και υπολογίζεται η καλύτερη λύση με χρήση του Amortization Plan. Για κάθε ώρα υπολογίζεται η κατανάλωση ενέργειας και το σφάλμα λάθους για την καλύτερη λύση με χρήση της μεθόδου calculateKW, η οποία είναι εμπνευσμένη από την αναρρίχηση λόφου (hill climbing). Καλύτερη λύση αποτελεί το διάνυσμα των ενεργοποιημένων και απενεργοποιημένων κανόνων, για το οποίο το σφάλμα ευκολίας και η κατανάλωση, είναι ελάχιστα. Η calculateKW είναι recursion μέθοδος όπου εκτελείται για ένα συγκεκριμένο αριθμό επαναλήψεων για κάθε καταγραφή του συνόλου. Σε κάθε πρώτη εκτέλεση της μεθόδου γίνεται τυχαία επιλογή της ενεργοποίησης και απενεργοποίησης των κανόνων όπως έχει ως ακολούθως. Στην εντολή στη γραμμή 4 επιλέγεται τυχαία ένας αριθμός από το 0 μέχρι το 1. Αν είναι 0 τότε ο μέτα-κανόνας θα είναι απενεργοποιημένος και η αντίστοιχη θέση του στον πίνακα με όνομα randomRule είναι false, όπως φαίνεται στην γραμμή εντολής 7 και προσθέτετε σαν απενεργοποιημένος κανόνας στην λίστα με όνομα removedMetaRules, όπως φαίνεται στην γραμμή εντολής 8. Διαφορετικά ο μέτα-κανόνας θα είναι ενεργοποιημένος και η αντίστοιχη θέση του στον πίνακα με όνομα randomRule είναι true όπως φαίνεται στην γραμμή εντολής 10.

```
1.  for (int d = 0; d < randomRule.length; d++) {
2.
3.    // generates random number from 0 to 1
4.    random = rand.nextInt(2);
5.
6.    if (random == 0) {
7.      randomRule[d] = false;
8.      removedMetaRules.add(d);
9.    } else {
10.     randomRule[d] = true;
11.   }
12.
13. }
```

Στις υπόλοιπες επαναλήψεις γίνεται τυχαία επιλογή της ενεργοποίησης και απενεργοποίησης μέχρι k κανόνων το οποίο αντιπροσωπεύετε από τη μεταβλητή numberOfLoops. Στην γραμμή εντολής 4 παρακάτω, παράγεται τυχαία ένας αριθμός από το 0 μέχρι τον αριθμό των μέτα-κανόνων, όπου αντιπροσωπεύει την αλλαγή της κατάστασης του συγκεκριμένου κανόνα. Αν είναι ενεργοποιημένος τότε η θέση του στον πίνακα randomRule από true γίνεται false και αφαιρείται από την λίστα απενεργοποιημένων κανόνων όπως φαίνεται από τις γραμμές εντολών 7-8. Διαφορετικά, ο κανόνας ενεργοποιείται και αφαιρείται από τη λίστα με τους απενεργοποιημένους κανόνες, όπως φαίνεται από τις γραμμές εντολών 10-11.

```
1.  for (int i = 0; i < numberOfLoops; i++) {
2.
3.    // random here is the rule to be removed
4.    random = rand.nextInt(numrules);
5.
6.    if (randomRule[random]) {
7.      randomRule[random] = false;
8.      removedMetaRules.add(random);
9.    } else {
10.     randomRule[random] = true;
11.     removedMetaRules.remove(random);
12.   }
13.
14. }
```

4.6 Υλοποίηση του αλγόριθμου Green Planner

Ο αλγόριθμος GP χρησιμοποιεί το σύνολο δεδομένων για διαμέρισμα ή κατοικία ή για φοιτητικές εστίες, τον πίνακα μέτα-κανόνων και το μέγιστο λάθος που υπολογίζεται από τον αλγόριθμο No-Rule. Ο αλγόριθμος υπολογίζει το σφάλμα ευκολίας (F_{CE}) την κατανάλωση ενέργειας (F_E) και τις εκπομπές CO_2 ($F_Γ$).

```
1.  for (int i = 0; i < iotdata.size(); i++) {
2.
3.    String[] dataSplitted = iotdata.get(i);
4.    double tempDayChange = getDay(dataSplitted[0]);
5.
6.    if (dayChange == tempDayChange) {
7.
8.      if (i == iotdata.size() - 1) {
9.        recordsDaily.add(iotdata.get(i));
10.       double hourly_ms = calculateKWH(recordsHourly);
11.       total_ms = total_ms + hourly_ms;
12.       recordsDaily.clear();
13.     } else {
14.       recordsDaily.add(iotdata.get(i));
15.     }
16.
17.   } else {
18.     dayChange = tempDayChange;
19.     hourly_ms = calculateKWH(recordsDaily);
20.     total_ms = total_ms + hourly_ms;
21.     recordsDaily.clear();
22.     recordsDaily.add(iotdata.get(i));
23.   }
24.
25. }
```

Για κάθε καταγραφή από το σύνολο δεδομένων βλέπουμε πότε αλλάζει η μέρα και υπολογίζεται η καλύτερη λύση ανά ώρα, με χρήση του Amortization Plan. Για κάθε μέρα υπολογίζεται η κατανάλωση ενέργειας, το σφάλμα λάθους και η εκπομπή διοξειδίου του άνθρακα για την καλύτερη ολική λύση με χρήση της μεθόδου calculateKW, η οποία είναι εμπνευσμένη από την προσομοιωμένη ανόπτηση (simulated annealing). Καλύτερη λύση αποτελεί το διάνυσμα των ενεργοποιημένων και απενεργοποιημένων κανόνων, για το οποίο το σφάλμα ευκολίας, η κατανάλωση ενέργειας και η εκπομπή διοξειδίου του άνθρακα, είναι ελάχιστα. Η calculateKW είναι recursion μέθοδος όπου εκτελείται για ένα

συγκεκριμένο αριθμό επαναλήψεων για κάθε καταγραφή. Σε κάθε πρώτη εκτέλεση της μεθόδου γίνεται τυχαία επιλογή της ενεργοποίησης και απενεργοποίησης των κανόνων όπως έχει ως ακολούθως. Στην εντολή στη γραμμή 4 επιλέγεται τυχαία ένας αριθμός από το 0 μέχρι το 1. Αν είναι 0 τότε ο μέτα-κανόνας θα είναι απενεργοποιημένος και η αντίστοιχη θέση του στον πίνακα με όνομα randomRule είναι false, όπως φαίνεται στην γραμμή εντολής 7 και προσθέτετε σαν απενεργοποιημένος κανόνας στην λίστα με όνομα removedMetaRules, όπως φαίνεται στην γραμμή εντολής 8. Διαφορετικά ο μέτα-κανόνας θα είναι ενεργοποιημένος και η αντίστοιχη θέση του στον πίνακα με όνομα randomRule είναι true όπως φαίνεται στην γραμμή εντολής 10.

```
1.  for (int d = 0; d < randomRule.length; d++) {
2.
3.    // generates random number from 0 to 1
4.    random = rand.nextInt(2);
5.
6.    if (random == 0) {
7.      randomRule[d] = false;
8.      removedMetaRules.add(d);
9.    } else {
10.     randomRule[d] = true;
11.   }
12.
13. }
```

Στις υπόλοιπες επαναλήψεις γίνεται τυχαία επιλογή της ενεργοποίησης και απενεργοποίησης μέχρι k κανόνων το οποίο αντιπροσωπεύετε από τη μεταβλητή numberOfLoops. Στην γραμμή εντολής 4 παρακάτω, παράγεται τυχαία ένας αριθμός από το 0 μέχρι τον αριθμό των μέτα-κανόνων, όπου αντιπροσωπεύει την αλλαγή της κατάστασης του συγκεκριμένου κανόνα. Αν είναι ενεργοποιημένος τότε η θέση του στον πίνακα randomRule από true γίνεται false και αφαιρείται από την λίστα απενεργοποιημένων κανόνων όπως φαίνεται από τις γραμμές εντολών 7-8. Διαφορετικά, ο κανόνας ενεργοποιείται και αφαιρείται από τη λίστα με τους απενεργοποιημένους κανόνες, όπως φαίνεται από τις γραμμές εντολών 10-11.

```
for (int i = 0; i < numberOfLoops; i++) {

  // random here is the rule to be removed
  random = rand.nextInt(numrules);

  if (randomRule[random]) {
    randomRule[random] = false;
    removedMetaRules.add(random);
  } else {
    randomRule[random] = true;
    removedMetaRules.remove(random);
  }

}
```

Στον παρακάτω κώδικα φαίνεται ένα παράδειγμα υπολογισμού της εκπομπής διοξειδίου του άνθρακα. Το dt2 αντιπροσωπεύει την ημερομηνία και ώρα της καταγραφής από το σύνολο δεδομένων. Εάν η ώρα καταγραφής είναι μεταξύ του διαστήματος όπου ο μέτα-κανόνας ισχύει (ορίζεται από τη μεταβλητή ruleFrom και ruleTo) και εάν ο καιρός είναι ηλιόλουστος, τότε η συνολική εκπομπή διοξειδίου του άνθρακα αυξάνεται κατά span. Το span είναι η διάρκεια για την οποία τα φώτα ή το κλιματιστικό είναι ενεργοποιημένα. Η παρακάτω αύξηση της εκπομπής του διοξειδίου του άνθρακα γίνεται αν υπάρχει παραβίαση του μέτα-κανόνα διαφορετικά δεν αυξάνεται η εκπομπή διοξειδίου του άνθρακα.

```
if (dt2.getHours()>=ruleFrom && dt2.getHours()<=ruleTo) {

   if (dataSplitted[4].equals("sky is clear")) {
     daily_ms_CO2 += span;
   }

}
```

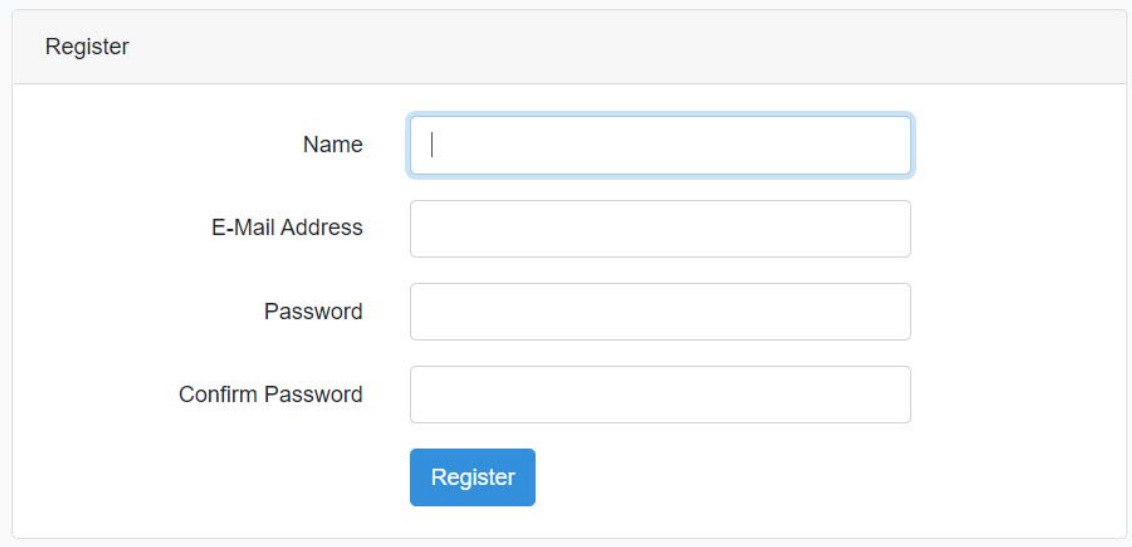
Κεφάλαιο 5

Διεπαφές Διαπροσωπείας

Σε αυτή την ενότητα περιγράφεται το σύστημα στο οποίο ένας χρήστης μπορεί να μπει αφού δημιουργήσει τον λογαριασμό του και εισάγει κάποιους κανόνες (Meta-rules) για τους οποίους χρησιμοποιεί ο αλγόριθμος Energy Planner (EP) και ο Green Planner (GP).

5.1 Δημιουργία λογαριασμού

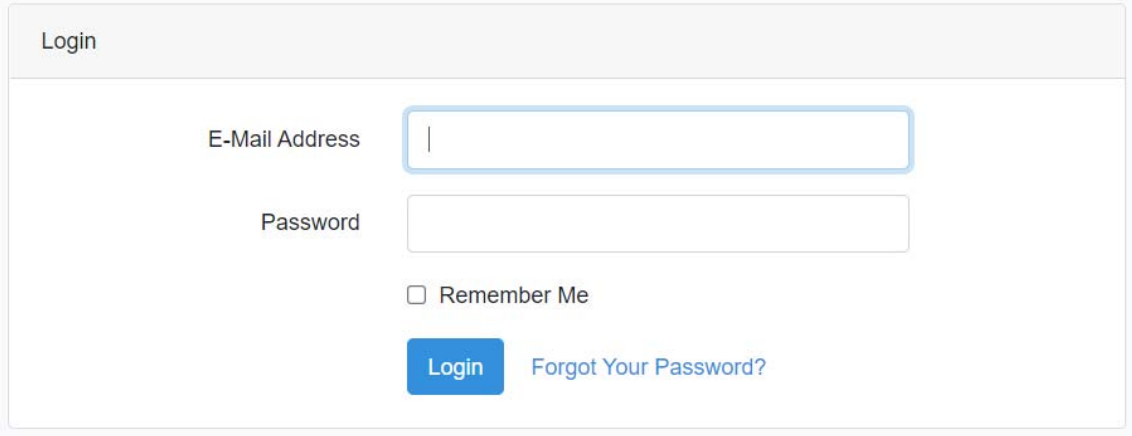
Ο χρήστης για να δημιουργήσει λογαριασμό στο σύστημα με όνομα "Meta-rules Management System" πρέπει να εισάγει στο σύστημα το όνομα του, την ηλεκτρονική του διεύθυνση και τον κωδικό του.



Σχήμα 5.1 Δημιουργία Λογαριασμού

5.2 Εισαγωγή του χρήστη στο σύστημα

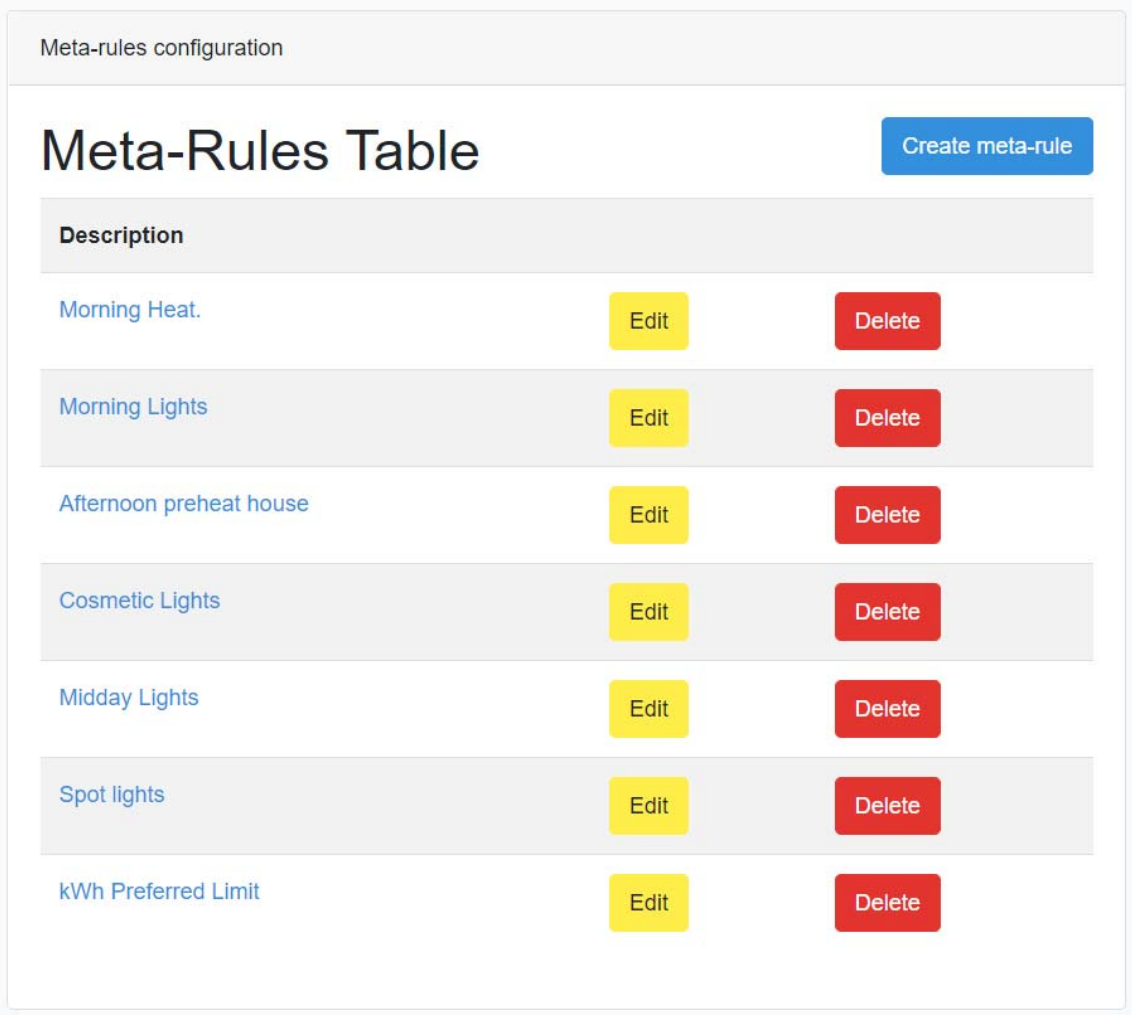
Ο χρήστης για την εισαγωγή του στο σύστημα καλείται να γράψει την ηλεκτρονική του διεύθυνση και τον κωδικό του.



Σχήμα 5.2 Εισαγωγή στο σύστημα

5.3 Σελίδα Διαχείρησης Μέτα-Κανόνων

Στη σελίδα διαχείρισης μέτα-κανόνων ο χρήστης μπορεί να προσθέσει, να αφαιρέσει και να επεξεργαστεί τους κανόνες προτιμήσεων τους.

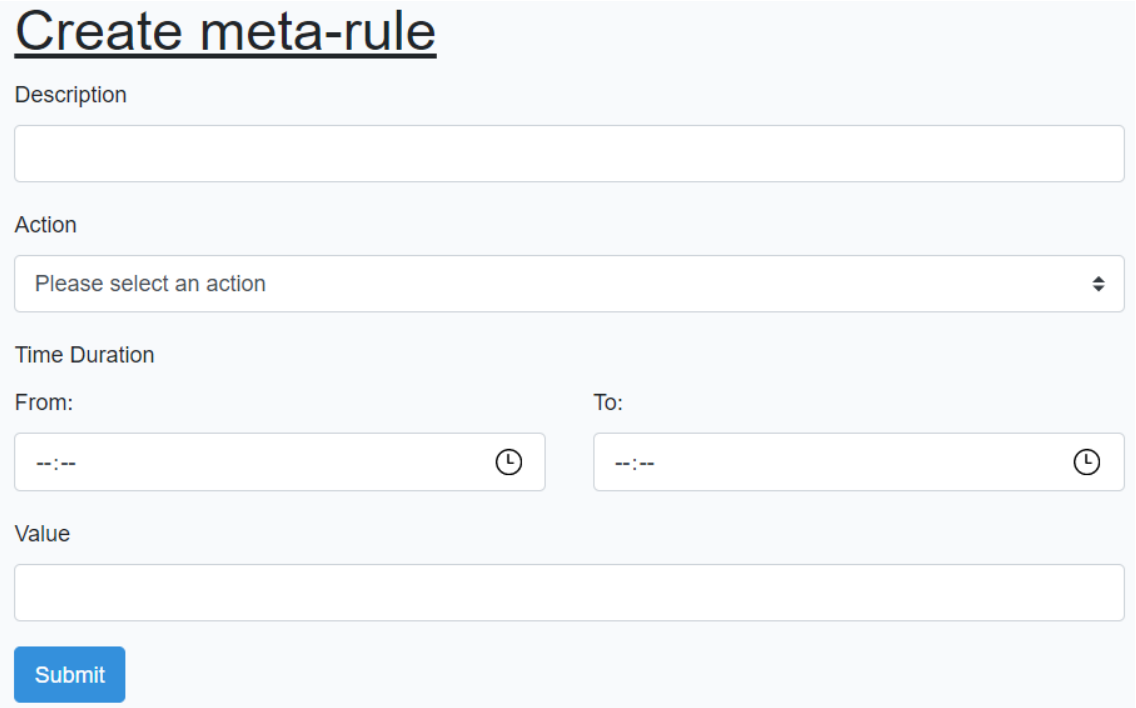


Σχήμα 5.3 Κανόνες Meta-Rules

5.4 Εισαγωγή κανόνα Μέτα-Κανόνα

Ένας χρήστης μπορεί να εισάγει ένα καινούργιο μετα-κανόνα πατώντας το κουμπί "Create meta-rule" από την σελίδα "Meta-Rules", βάζοντας μια περιγραφή του κανόνα,

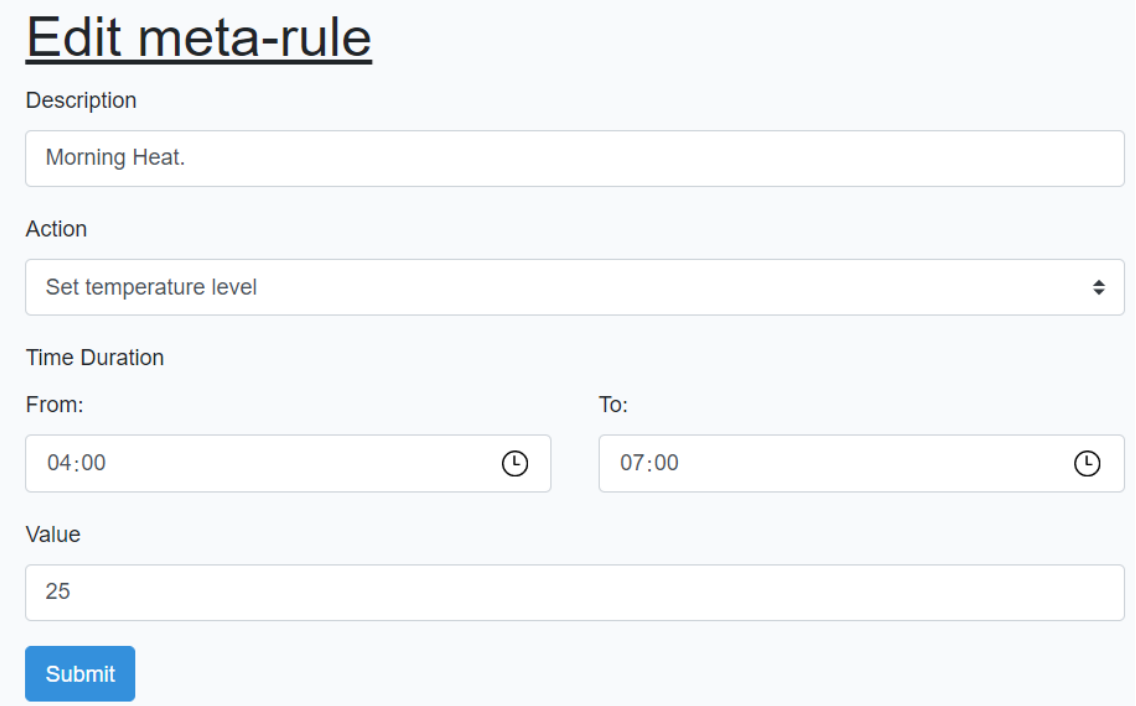
το “action” το οποίο χωρίζεται σε τρείς κατηγορίες, ανάθεσης φωτός, ανάθεσης θερμοκρασίας, ανάθεσης ορίου kWh, ώρα για την οποία πρέπει να ισχύει ο κανόνας (στην περίπτωση ανάθεσης ορίου kWh δεν απαιτείται) και την τιμή.



Σχήμα 5.4 Δημιουργία κανόνα Meta-Rule

5.4 Επεξεργασία Μέτα-Κανόνα

Σε περίπτωση λάθους, ή οποιοδήποτε λόγου ο χρήστης μπορεί να επεξεργαστεί ένα κανόνα πατώντας το κουμπί “Edit” από την σελίδα “Meta-Rules” και να τον αλλάξει. Πατώντας το κουμπί “Submit” μπορεί να αποθηκεύσει οποιαδήποτε αλλαγή έχει κάνει.



Σχήμα 5.5 Επεξεργασία κανόνα Meta-Rule

5.6 Συσκευές συνδεδεμένες στο σύστημα

Ένας χρήστης μπορεί να δει τις διάφορες έξυπνες συσκευές που έχει συνδεδεμένες στο σύστημα IMCF μαζί με την κατάσταση τους (ON/OFF) στην σελίδα με όνομα "OpenHAB Local". Οι διάφορες συσκευές προθέτονται από το OpenHAB και παρουσιάζονται μέσω της χρήσης του Anyplace.

OpenHAB Configurations - Local Controller

OpenHAB Rules Table

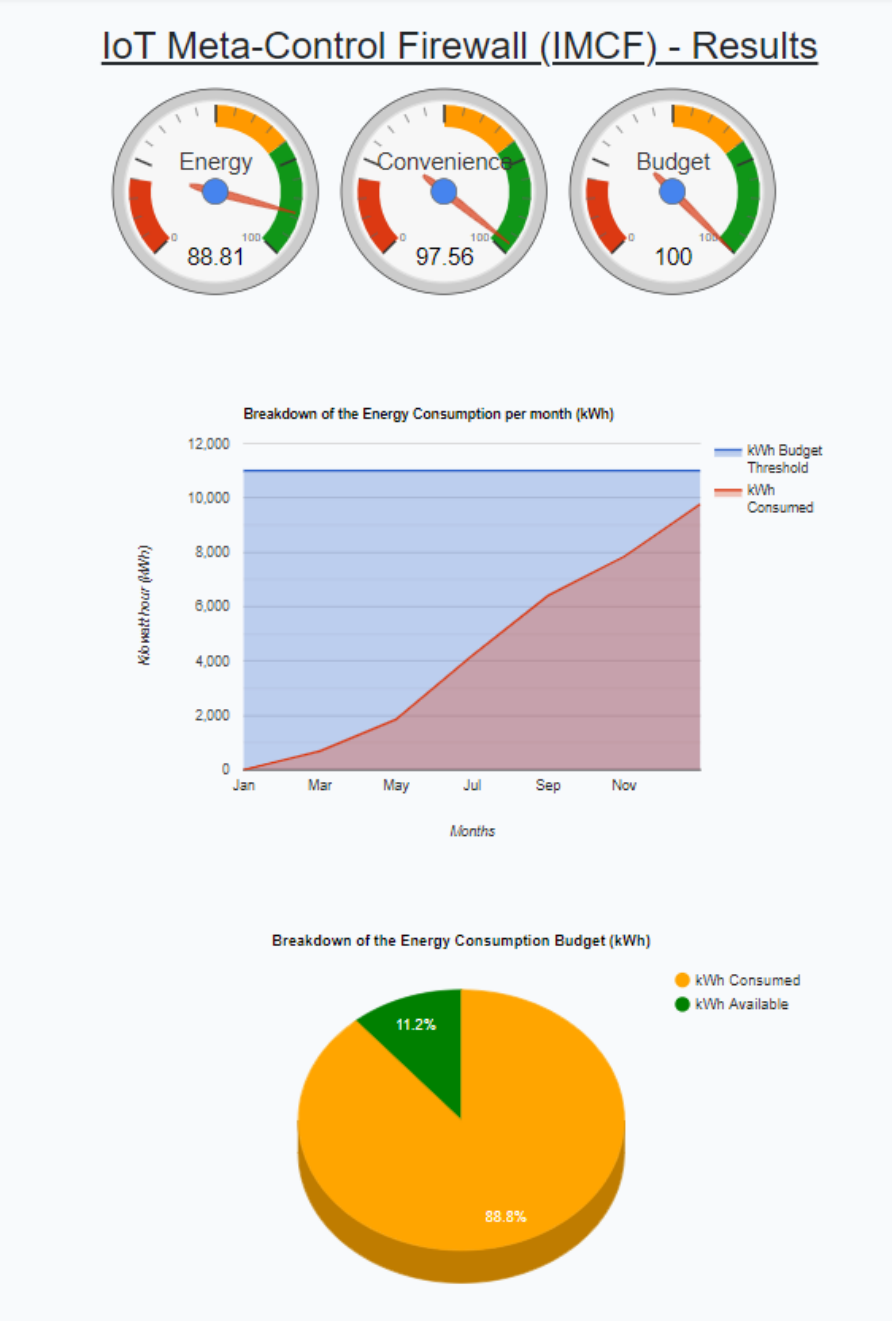
Retrieve OpenHAB Rules

Description	Type	State	Mode
Living room Air-Conditioner	Temperature	19.74 °C	ON
Kitchen Lights	Light	20 %	ON
Bedroom Air-Conditioner	Temperature	18.0 °C	ON
Bedroom Lights	Light	20 %	ON
Kitchen Air-Conditioner	Temperature	17.26 °C	ON
Living Room Lights	Light	40 %	ON

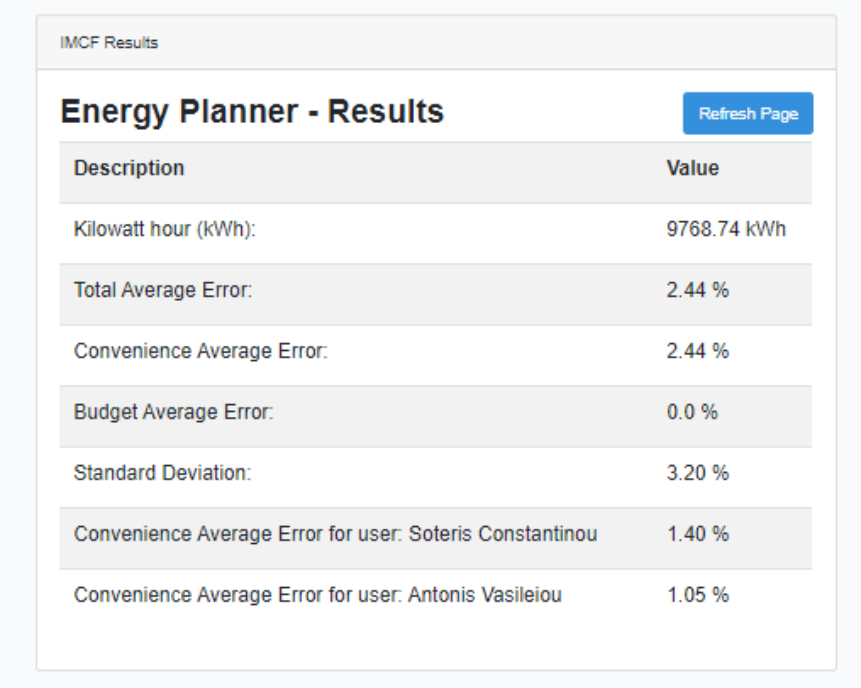
Σχήμα 5.6 Έξυπνες συσκευές συνδεδεμένες στο σύστημα

5.7 Αποτελέσματα εκτέλεσης

Στην σελίδα με όνομα “Results” ο χρήστης μπορεί να δει τα αποτελέσματα που έχει παραγάγει το σύστημα. Στο σχήμα 5.7 μπορούμε να δούμε με την γραφική παράσταση την μηνιαία κατανάλωση ηλεκτρικού ρεύματος, το πόσο τοις εκατό είμαστε άνετα, το πόσο τοις εκατό βρισκόμαστε εντός προϋπολογισμού και πόσο τοις εκατό καταναλώσαμε ενέργεια.

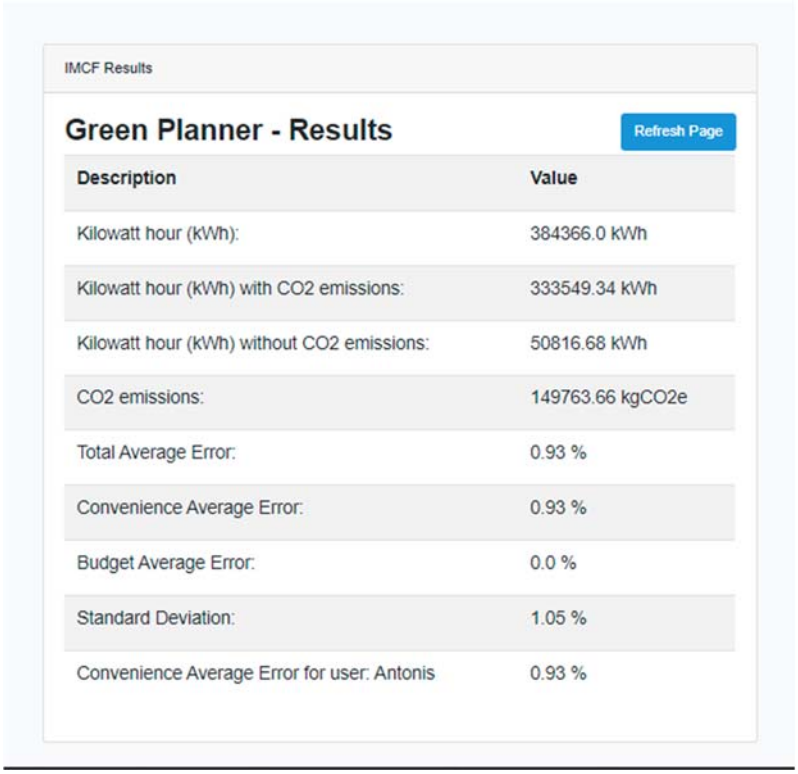


Σχήμα 5.7 Αποτελέσματα Εκτέλεσης 1



Σχήμα 5.8 Αποτελέσματα Εκτέλεσης 2

Στην περίπτωση όπου ο αλγόριθμος που χρησιμοποιήθηκε είναι ο Green Planner τα αποτελέσματα έχουν ως εξής (Τα αποτελέσματα είναι για σενάριο φοιτητικών εστιών).



Σχήμα 5.9 Αποτελέσματα Green Planner

Κεφάλαιο 6

Πειραματική Μεθοδολογία & Αποτίμηση

Σε αυτή την ενότητα παρουσιάζεται η πειραματική μεθοδολογία που χρησιμοποιήθηκε και η αποτίμηση των αλγόριθμων. Αναφέρεται η πλατφόρμα που χρησιμοποιήθηκε την εκτέλεση των διάφορων πειραμάτων και τα διάφορα σύνολα δεδομένων, οι μετρικές που μετρήθηκαν για την πειραματική αποτίμηση καθώς και οι αλγόριθμοι που χρησιμοποιήθηκαν. Επίσης περιγράφετε η αξιολόγηση απόδοσης βάση πειράματος, που εκθέτει τα βασικά οφέλη του πλαισίου μας και του εσωτερικού αλγόριθμου GP και EP σε σύγκριση με τις βασικές τεχνικές.

6.1 Πλατφόρμα

Η αξιολόγησή πραγματοποιήθηκε στο κέντρο δεδομένων DMSL vCenter IaaS, ένα ιδιωτικό cloud, το οποίο περιλαμβάνει 8 IBM System x3550 M3, Lenovo ThinkSystem

SR590 Server και HP Proliant DL 360 G7 rackables με μονή υποδοχή (8 πυρήνες) ή διπλή υποδοχή (16 πυρήνες) Intel (R) Xeon (R) CPU E5620 @ 2.40GHz, αντίστοιχα. Αυτοί οι κεντρικοί υπολογιστές έχουν συλλογικά 300 GB κύριας μνήμης, 25 TB αποθήκευσης RAID-5 σε ένα IBM 3512 και συνδέονται μεταξύ τους μέσω ενός δικτύου Gigabit. Το κέντρο δεδομένων διαχειρίζεται μέσω ενός VMWare VSphere που συνδέεται με τους αντίστοιχους κεντρικούς υπολογιστές VMWare ESXi 6.7.0. Ο κόμβος υπολογιστών μας αποτελείται από μια εικόνα διακομιστή Ubuntu 18.04, που διαθέτει 8 GB RAM με 2 εικονικούς επεξεργαστές (@ 2.40GHz) Η εικόνα χρησιμοποιεί γρήγορους τοπικούς δίσκους RAID-5 LSILogic SCSI 10K PM, μορφοποιημένους με VMFS 6 (μέγεθος μπλοκ 1MB).

6.2 Σύνολα δεδομένων

Υιοθετήθηκε μια πειραματική μεθοδολογία [11] που βασίζεται σε ίχνη στην οποία τα πραγματικά σύνολα δεδομένων τροφοδοτούνται στον προσομοιωτή μας που εκτελείται στο testbed. Αυτό επιτρέπει την επαναλαμβανόμενη εκτέλεση φόρτων εργασίας υπό διαφορετικές παραμέτρους ελέγχου. Πιο συγκεκριμένα, χρησιμοποιήθηκαν ανώνυμες μετρήσεις από ένα πραγματικό διαμέρισμα κατοικιών που αποτελείται από μια ποικιλία αισθητήρων, υπομέτρων και περίπου 5.668.878 αναγνώσεις (1,09 GB συνολικά). Πρόκειται για πραγματικά σύνολα δεδομένων οικιστικών δεδομένων που συλλέγονται από το "Κέντρο Προηγμένων Μελετών σε Προσαρμοσμένα Συστήματα" (CASAS) [22] στο Washington State University στη Σχολή Ηλεκτρολόγων Μηχανικών και Επιστήμης Υπολογιστών. Το CASAS εξυπηρετεί την κάλυψη ερευνητικών αναγκών σχετικά με τη δοκιμή των τεχνολογιών χρησιμοποιώντας πραγματικά δεδομένα μέσω της χρήσης ενός έξυπνου σπιτιού που βρίσκεται στην πανεπιστημιούπολη WSU Pullman. Το πραγματικό σύνολο δεδομένων πρόγνωσης καιρού αποκτήθηκε χρησιμοποιώντας το Weather API στον ιστότοπο του OpenWeatherMap και περιέχει ~ 5 χρόνια (2012-2017) δεδομένων υψηλής χρονικής ανάλυσης βάσει ωριαίων μετρήσεων διαφόρων χαρακτηριστικών καιρού, όπως θερμοκρασία, υγρασία, πίεση αέρα, περιγραφή καιρού, κατεύθυνση και ταχύτητα ανέμου.

Σύνολο δεδομένων θερμοκρασίας

Είναι ένα σύνολο δεδομένων μεγέθους 700 MB και περιέχει 3.555.238 αναγνώσεις σε δεύτερη βάση μεταξύ Οκτωβρίου 2013 και Δεκεμβρίου 2016. Οι μετρήσεις, οι οποίες καταγράφονται σε ένα οικιστικό διαμέρισμα ενός εθελοντή ενήλικα, περιλαμβάνουν μετρήσεις θερμοκρασίας και αισθητήρα πόρτας / παραθύρου.

Σύνολο δεδομένων φωτός

Είναι ένα σύνολο δεδομένων μεγέθους 416 MB και περιέχει 2.113.640 αναγνώσεις σε δεύτερη βάση μεταξύ Οκτωβρίου 2013 και Δεκεμβρίου 2016. Οι μετρήσεις, οι οποίες καταγράφονται σε οικιστικό διαμέρισμα ενός εθελοντή ενήλικα, περιλαμβάνουν μετρήσεις φωτός.

Σύνολο δεδομένων πρόγνωσης καιρού

Είναι ένα σύνολο δεδομένων μεγέθους 71,23 MB και περιέχει 271.561 αναγνώσεις ανά ώρα μεταξύ Οκτωβρίου 2012 και Νοεμβρίου 2017. Τα ιστορικά δεδομένα καιρού είναι διαθέσιμα για 30 πόλεις των ΗΠΑ και του Καναδά, καθώς και για 6 πόλεις του Ισραήλ, συμπεριλαμβανομένης της θερμοκρασίας, της υγρασίας, του αέρα πίεση, περιγραφή καιρού, κατεύθυνση και ταχύτητα ανέμου.

Σύνολο δεδομένων διαμερίσματος

Είναι ένα σύνολο δεδομένων για έναν χρήστη που κατοικεί σε διαμέρισμα το οποίο αποτελείται από ένα υπνοδωμάτιο, ένα μπάνιο και μια κουζίνα. Το διαμέρισμα είναι μεγέθους 50 m^2, διαθέτει μια ενιαία μονάδα για θέρμανση / ψύξη. Το σύνολο δεδομένων έχει μέγεθος 1,09 GB.

Σύνολο δεδομένων κατοικίας

Είναι ένα σύνολο δεδομένων κατοικίας που δημιουργήθηκε με την αναπαραγωγή, την ανάμειξη των μετρήσεων και τον πολλαπλασιασμό του πραγματικού συνόλου δεδομένων (σύνολο διαμερίσματος) επί 4 φορές. Η κατοικία είναι μεγέθους 200 m^2 και αποτελείται από 3 υπνοδωμάτια και 4 ενιαίες μονάδες για θέρμανση / ψύξη.

Σύνολο δεδομένων φοιτητικών εστιών

Είναι ένα σύνολο δεδομένων για μια φοιτητική εστία, το οποίο έχει δημιουργηθεί συνθετικά από τα σύνολα δεδομένων bootstrap. Αποτελείται από 50 κοιτώνες που αποτελούνται από δύο υπνοδωμάτια (10 m^2 / δωμάτιο) με κοινόχρηστο μπάνιο, κουζίνα και δύο χωριστές μονάδες. Το συνολικό μέγεθος των κοιτώνων είναι περίπου $2000m^2$ και το σύνολο δεδομένων έχει μέγεθος 20 GB.

6.3 Μετρικές

Οι μετρικές [11] αποτελούνται από την κατανάλωση ενέργειας F_E, την εκπομπή διοξειδίου του άνθρακα $F_Γ$ και το σφάλμα ευκολίας F_{CE} και ο χρόνος επεξεργαστή F_T όπου χρησιμοποιείται για σύγκριση στη μελέτη απόδοσης. Το F_T είναι ο χρόνος επεξεργασίας που απαιτείται από τον ελεγκτή για την εκτέλεση της λειτουργίας βελτιστοποίησης και τον υπολογισμό της εξόδου για όλους τους μέτα-κανόνες. Ο μέσος όρος και η τυπική απόκλιση των αποτελεσμάτων παρουσιάζονται με γραμμές σφάλματος σε όλες τις πειραματικές μελέτες που ακολουθούν, με βάση δέκα επαναλήψεις.

6.4 Αλγόριθμοι

Σε αυτή την υποενότητα αναφέρονται οι αλγόριθμοι που χρησιμοποιήθηκαν για την πειραματική αξιολόγηση.

6.4.1 Αλγόριθμοι NR και MR

Ο αλγόριθμος No-Rule (NR) [11] αγνοεί όλους τους μέτα-κανόνες του πίνακα μέτα-κανόνων και δεν τροποποιεί τη συμπεριφορά των αυτόνομων συσκευών. Από την άλλη ο αλγόριθμος Meta-Rule (MR) [11] αγνοεί την εξοικονόμηση κατανάλωσης ενέργειας και εκτελεί όλους τους μέτα-κανόνες με απληστία.

6.4.2 Αλγόριθμος If-This-Then-That (IFTTT)

Αυτή ο αλγόριθμος εκτελεί τους κανόνες προτιμήσεων τύπου IFTTT στο σύνολο δεδομένων διαμερίσματος και υπολογίζει το F_E και το F_{CE}.

IF	THIS	THEN	THAT
Season	Summer	Set Temper.	25
Season	Winter	Set Temper.	20
Weather	Sunny	Set Temper.	20
Weather	Cloudy	Set Light.	22
Weather	Sunny	Set Light.	0
Weather	Cloudy	Set Temper.	40
Temperature	>30	Set Temper.	23
Temperature	<10	Set Temper.	24
Light Level	>15	Set Light.	9
Door	Open	Set Light.	0

Πίνακας 6.1 Κανόνες IFTTT [11]

6.4.3 Αλγόριθμος Energy Planner

Για τον αλγόριθμο EP [11] έχει οριστεί αριθμός ενεργοποίησης / απενεργοποίησης κανόνων σε κάθε επανάληψη (k), ένα ποσοστό (τ) εξοικονόμησης και ο αριθμός των επαναλήψεων (τ_{max}).

6.4.4 Αλγόριθμος Green Planner

Για τον αλγόριθμο GP [21] έχει οριστεί αριθμός ενεργοποίησης / απενεργοποίησης κανόνων σε κάθε επανάληψη (k), ένα ποσοστό (τ) εξοικονόμησης και ο αριθμός των επαναλήψεων (τ_{max}).

6.5 Αξιολόγηση απόδοσης

Σε αυτήν την υποενότητα, αξιολογείτε η απόδοση του προτεινόμενου πλαισίου EP και GP έναντι όλων των αλγορίθμων σε όλα τα σύνολα δεδομένων που έχουν εισαχθεί, σε σχέση με την κατανάλωση ενέργειας και το σφάλμα ευκολίας. Το σχήμα 6.2 δείχνει την αντιστάθμιση μεταξύ της κατανάλωσης ενέργειας (F_E), του σφάλματος ευκολίας (F_{CE}) και του χρόνου εκτέλεσης της CPU (F_T) μεταξύ των βασικών προσεγγίσεων και της EP, ενώ το σχήμα 6.3 δείχνει την αντιστάθμιση μεταξύ της κατανάλωσης ενέργειας (F_E), του σφάλματος ευκολίας (F_{CE}), την εκπομπή CO_2 ($F_Γ$) και του χρόνου εκτέλεσης της CPU (F_T) μεταξύ των βασικών προσεγγίσεων και της GP. Η προσέγγιση NR απέκτησε το χειρότερο F_{CE} = 62-72% για όλα τα σύνολα δεδομένων, το καλύτερο F_E = 0 kWh και την καλύτερη εκπομπή CO_2. Ο αλγόριθμος EP έλαβε ένα εντυπωσιακό F_{CE} περίπου 2-4% και το τρίτο χαμηλότερο F_E. Ο αλγόριθμος GP έλαβε ένα F_{CE} περίπου 6.5-8%, το δεύτερο χαμηλότερο F_E και τη δεύτερη μικρότερη εκπομπή CO_2 μετά την NR. Οι αλγόριθμοι IFTTT & MR είναι άπληστοι όσον αφορά την κατανάλωση ενέργειας, επομένως η κατανάλωση kWh και η εκπομπή CO_2 τους είναι πολύ υψηλή. Η κύρια διαφορά μεταξύ των δύο είναι ότι το IFTTT έχει F_{CE} = 26% στο διαμέρισμα, F_{CE} = 29% στην περίπτωση της κατοικίας και F_{CE} = 39% στην περίπτωση των φοιτητικών κοιτώνων, ενώ η MR ικανοποιεί όλους τους μέτα-κανόνες, έτσι το F_{CE} του είναι 0%, το οποίο είναι το καλύτερο δυνατό.

Ο αλγόριθμος EP για την περίπτωση του διαμερίσματος κατάφερε να εξοικονομήσει έως και 10% της ενέργειας, που είναι περίπου 9500 kWh για τον προτιμώμενο ενεργειακό προϋπολογισμό των 11000 kWh και για τα τρία χρόνια, με F_{CE} που έφτασε σε λογικές τιμές των 2-3%. Στην περίπτωση ενός σπιτιού, ο προτιμώμενος ενεργειακός προϋπολογισμός διαμορφώθηκε στα 25500 kWh και για τα τρία χρόνια και η EP κατάφερε να επιτύχει περίπου 22300 kWh, με F_{CE} περίπου 2-2,5%. Στην περίπτωση των κοιτώνων, ο προτιμώμενος προϋπολογισμός ενέργειας διαμορφώθηκε στα 480.000 kWh και για τα τρία χρόνια και η EP κατάφερε να επιτύχει περίπου 410.000 kWh, με την τιμή του F_{CE} να φτάνει την λογική τιμή των 2,5-3%. Εδώ είναι σημαντικό να παρατηρηθεί, ότι η διαφορά μεταξύ του MR και του EP όσον αφορά την κατανάλωση ενέργειας, είναι σχετικά υψηλή και συγκεκριμένα ≈5.000kWh για το σύνολο δεδομένων του

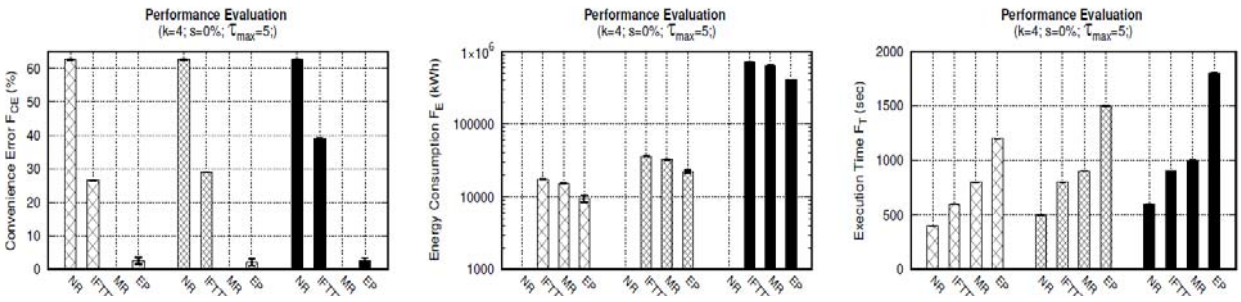
διαμερίσματος, ≈10.000kWh για το σύνολο δεδομένων κατοικίας και ≈150.000kWh για το σύνολο δεδομένων των φοιτητικών κοιτώνων.

Ο αλγόριθμος GP για την περίπτωση του διαμερίσματος κατάφερε να εξοικονομήσει έως και 10% της ενέργειας, που είναι περίπου 9500 kWh για τον προτιμώμενο ενεργειακό προϋπολογισμό των 11000 kWh και για τα τρία χρόνια, με F_{CE} που έφτασε σε λογικές τιμές των 6.5-7% και εκπομπές CO_2 ≈3600 Kg. Στην περίπτωση ενός σπιτιού, ο προτιμώμενος ενεργειακός προϋπολογισμός διαμορφώθηκε στα 25500 kWh και για τα τρία χρόνια και η GP κατάφερε να επιτύχει περίπου 22300 kWh, με F_{CE} περίπου 7-7,5% και εκπομπές CO_2 ≈8700 Kg. Στην περίπτωση των κοιτώνων, ο προτιμώμενος προϋπολογισμός ενέργειας διαμορφώθηκε στα 480.000 kWh και για τα τρία χρόνια και η GP κατάφερε να επιτύχει περίπου 410.000 kWh, με την τιμή του F_{CE} να φτάνει την λογική τιμή των 7,5-8% και εκπομπές CO_2 ≈135,000 Kg. Το κόστος ανά kWh στην Κύπρο είναι 0.181 ευρώ [23], έτσι ο πίνακας που ακολουθεί μας δείχνει την εξοικονόμηση των χρημάτων σε σχέση με το όριο που έχει οριστεί.

Τύπος	**Ενεργειακό Όριο**	**Κατανάλωση με EP/GP**	**Εξοικονόμηση Χρημάτων**
Διαμέρισμα	11000 kWh	9500 kWh	271.5 ευρώ
Κατοικία	25500 kWh	22300 kWh	579.2 ευρώ
Φοιτητικές Εστίες	480000 kWh	410000 kWh	12,670 ευρώ

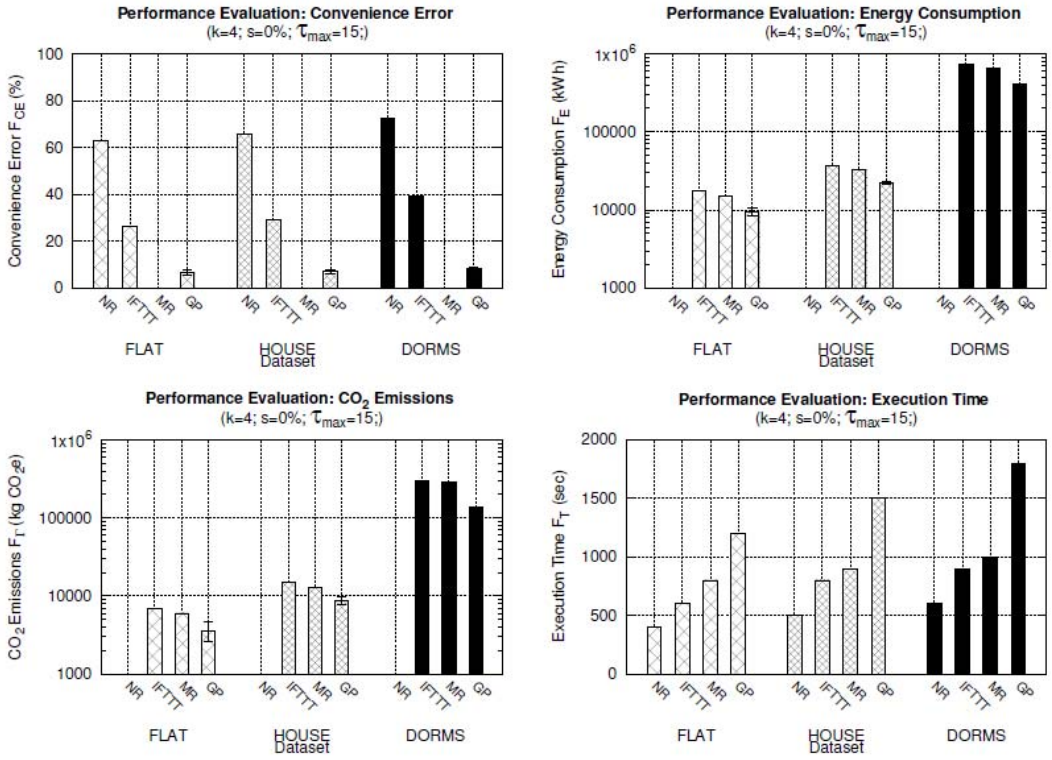
Πίνακας 6.2 Εξοικονόμηση Χρημάτων με χρήση των αλγόριθμων

Ο ταχύτερος χρόνος εκτέλεσης επιτεύχθηκε από την NR, καθώς απλώς εκτελεί έναν υπολογισμό σφάλματος αγνοώντας όλους τους κανόνες. Η προσέγγιση αναρρίχησης του



Σχήμα 6.1 Αξιολόγηση απόδοσης Energy Planner [4]

λόφου της EP, από την άλλη, αναζητά το χώρο λήψης αποφάσεων για μια λύση που θα βελτιστοποιήσει την άνεση του χρήστη και θα ικανοποιήσει τον ενεργειακό περιορισμό, ταυτόχρονα, που είναι μια πολύ πιο χρονοβόρα διαδικασία. Ο αλγόριθμος GP χρησιμοποιεί την προσομοιωμένη προσέγγιση ανόπτησης αναζητώντας μια βέλτιστη λύση για τον χρήστη λαμβάνοντας υπόψη τον προτιμώμενο περιορισμό του επιτρεπόμενου ενεργειακού προϋπολογισμού και τον περιορισμό των εκπομπών CO2, επομένως είναι η πιο χρονοβόρα μέθοδος. Τέλος, η άπληστη προσέγγιση του MR εστιάζει μόνο στην ελαχιστοποίηση του σφάλματος ευκολίας, που σημαίνει την εκτέλεση όλων των μέτα-κανόνων χωρίς επαναλαμβανόμενες διαδικασίες ή υπολογισμούς, αφού F_{CE} = 0%.



Σχήμα 6.2 Αξιολόγηση απόδοσης Green Planner [21]

6.6 Συμπεράσματα από την πειραματική εφαρμογή

Οι αλγόριθμοι έχουν καταφέρει να μείνουν εντός του ενεργειακού στόχου που έθεσαν οι χρήστες και να γίνει εξοικονόμηση 10% της ενέργειας κρατώντας τους χρήστες κατά 2-8% μη ευχαριστημένους. Το σφάλμα ευκολίας F_{CE} στους αλγόριθμους Energy Planner

και Green Planner φτάνει στα 2-3% και 6.5%-8% αντίστοιχα. Αυτό σημαίνει ότι, οι χρήστες μένουν αρκετά ικανοποιημένοι χωρίς να επηρεάσει πολύ την ευκολία στην καθημερινότητα τους, δηλαδή ότι διατηρούνται οι μέτα-κανόνες που έθεσαν ενώ το ενεργειακό όριο που έθεσαν δεν ξεπεράστηκε. Έτσι καταλήγουμε στο συμπέρασμα πως οι αλγόριθμοι έχουν εκπληρώσει με επιτυχία τον στόχο που θέσαμε, το να μειώσουμε την κατανάλωση ενέργειας και τις εκπομπές ρίπων θερμοκηπίου.

Κεφάλαιο 7

Συμπεράσματα

7.1 Συμπεράσματα

Ανακεφαλαιώνοντας, αυτή η διπλωματική εργασία είχε σκοπό την επίλυση του προβλήματος της επίτευξης μακροχρόνιων στόχων ενέργειας σε ένα έξυπνο κτήριο, ούτως ώστε να συνεισφέρουμε θετικά στις διάφορες προσπάθειες των χωρών να μειώσουν την κατανάλωση ενέργειας και τη μείωση ρίπων διοξειδίου του άνθρακα, για τη μείωση του φαινομένου του θερμοκηπίου. Για την επίλυση αυτού του προβλήματος υλοποιήθηκε το πλαίσιο κανόνων προτιμήσεων για επίτευξη μακροχρόνιων στόχων όπως η συνολική κατανάλωση ενέργειας σε ένα έξυπνο κτήριο.

Η υλοποίηση αυτού του πλαισίου συμπεριλαμβάνει την υλοποίηση των αλγορίθμων του συστήματος μέτα-κανόνων διαδικτύου των πραγμάτων (IMCF). Αρχικά μελετήθηκαν οι διάφοροι αλγόριθμοι τεχνητής νοημοσύνης που χρησιμοποιούνται στους αλγόριθμους Energy Planner και Green Planner ούτως ώστε να επιτευχθεί η καλύτερη κατανόηση τους, καθώς και κάποιες έξυπνες συσκευές για τις οποίες μπορεί να εφαρμοστεί το πλαίσιο μας. Επίσης όπως έχουν μελετηθεί οι ροές αυτοματισμού (RAW) και το if-this-then-that που χρησιμοποιήθηκαν για την υλοποίηση των διάφορων αλγορίθμων. Οι αλγόριθμοι κτίστηκαν βάση της αρχιτεκτονικής και ενσωματώθηκαν στο ολοκληρωμένο σύστημα IMCF.

Βάση των διάφορων πειραμάτων και αποτιμήσεων των αλγορίθμων που έγιναν, καταλήξαμε στο συμπέρασμα ότι το πλαίσιο το οποίο αναπτύχθηκε είναι ένας καλός τρόπος για επίτευξη των μακροχρόνιων στόχων βάση κανόνων που μπορεί να θέσουν οι χρήστες, αφού καταφέραμε να επιτύχουμε 10% μείωση στην κατανάλωση ενέργειας κρατώντας τους χρήστες αρκετά ικανοποιημένους καθώς το σφάλμα ευκολίας έφτασε μόνο στο 2-4% στην περίπτωση χρήσης του αλγορίθμου Energy Planner και στο 7-8% στην περίπτωση χρήσης του αλγορίθμου Green Planner. Αποτέλεσμα αυτού είναι η επίτευξη του στόχου μείωσης κατανάλωσης ενέργειας και μείωσης της εκπομπής ρίπων θερμοκηπίου.

7.2 Μελλοντική δουλειά

Το πλαίσιο εκτέλεσης κανόνων προτιμήσεων έχει αποδειχτεί ως ένας αποδοτικός τρόπος για μείωση της κατανάλωσης ενέργειας και μείωσης των ρίπων του διοξειδίου του άνθρακα. Ως εκ τούτου, στο μέλλον το πλαίσιο αυτό μπορεί να εφαρμοστεί σε ένα πραγματικό έξυπνο κτήριο μέσω ενσωμάτωσης σε έξυπνες πλατφόρμες ενεργοποίησης και να μελετηθούν τα πραγματικά οφέλη του. Επίσης στο πλαίσιο, μπορεί να αναπτυχθεί το CMC controller [11] το οποίο μπορεί να ελέγχει το πλαίσιο μέσω ενός cloud, για να μπορούν οι χρήστες να το χρησιμοποιούν εκτός του προσωπικού δικτύου τους. Ακόμα στο μέλλον μπορεί να μελετηθούν μέθοδοι αναγνώρισης φόρτου εργασίας για συσκευές που καταναλώνουν ενέργεια και πώς να αναπρογραμματιστούν με φιλικό προς το περιβάλλον τρόπο.

Αναφορές

[1] «Paris Agreement,» United Nations Climate Change, 2015. [Ηλεκτρονικό]. Available: https://unfccc.int/process-and-meetings/the-paris-agreement/the-paris-agreement.

[2] «Green Deal,» European Commission, 2020. [Ηλεκτρονικό]. Available: https://ec.europa.eu/info/strategy/priorities-2019-2024/european-green-deal_en.

[3] «Saves,» Erasmus Student Network, 2014. [Ηλεκτρονικό]. Available: https://esn.org/saves.

[4] "Sunny home manager 2.0," SMA, 2019. [Online]. Available: https://www.sma.de/en/products/monitoring-control/sunny-home-manager-20.html.

[5] «SMA Developer,» SMA, [Ηλεκτρονικό]. Available: https://www.sma.de/en/products/sma-developer.html.

[6] «Google Nest Thermostat,» Google, [Ηλεκτρονικό]. Available: https://store.google.com/us/product/nest_learning_thermostat_3rd_gen?hl=en-US.

[7] «Kasa Smart WiFi Plug Slim with Energy Monitoring,» Tp-Link, [Ηλεκτρονικό]. Available: https://www.kasasmart.com/us/products/smart-plugs/kasa-smart-plug-slim-energy-monitoring-kp115.

[8] Wikipedia, «EEBUS,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/EEBUS.

[9] «Technology - EEBus Initiative e.V.,» EEBus, [Ηλεκτρονικό]. Available: https://www.eebus.org/technology/.

[10] «Row Automation Workflow,» [Ηλεκτρονικό]. Available: https://intercom.help/mylogiwa/en/articles/4619442-what-is-a-workflow-automation-rule.

[11] Soteris Constantinou, Andreas Konstantinidis, Demetrios Zeinalipour-Yazti and Panos K. Chrysanthis, «The IoT Meta-Control Firewall,» σε 37th IEEE International Conference on Data Engineering, Chania, Crete, pp.1-12, 2021.

[12] S. Heo, S. Song, J. Kim, and H. Kim., «Rt-ifttt: Real-time iot framework,» σε *2017 IEEE Real-Time Systems Symposium (RTSS)*, Paris, 2017.

[13] «What is a Smart Actuator,» [Ηλεκτρονικό]. Available: https://www.infobloom.com/what-is-a-smart-actuator.htm.

[14] «What is smart actuation?,» [Ηλεκτρονικό]. Available: https://www.thomsonlinear.com/micro/smart-actuation_eng/what-is-smart-actuation.html.

[15] «Energy demand management,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Energy_demand_management#Household_scale.

[16] «Local Search,» Wikipedia, [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Local_search_(optimization).

[17] «Hill Climbing,» Wikipedia, [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Hill_climbing.

[18] «Travelling Salesman,» Wikipedia, [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Travelling_salesman_problem.

[19] «Hill climbing algorithm in AI,» javatpoint, [Ηλεκτρονικό]. Available: https://www.javatpoint.com/hill-climbing-algorithm-in-ai.

[20] «Simulated Annealing,» Wikipedia, [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Simulated_annealing.

[21] Soteris Constantinou, Andreas Konstantinidis, Demetrios Zeinalipour-Yazti and Panos K. Chrysanthis, «Green Planning of IoT Rule Automation Workflows,» 2021.

[22] A. Brown, «Interactions,» σε ACM interactions, vol 9, no 2, pp 51-53, 2002.

[23] «Cyprus electricity prices,» GlobalPetrolPrices.com, [Ηλεκτρονικό]. Available: https://www.globalpetrolprices.com/Cyprus/electricity_prices/.

Παράρτημα Α

Στο παράρτημα αυτό παραθέτετε ο ολοκληρωμένος κώδικας των αλγορίθμων του πλαισίου εκτέλεσης κανόνων, που υλοποιήθηκαν στα πλαίσια της παρούσας διπλωματικής εργασίας.

importData.java

```
package iotprojectpackage;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileWriter;
import java.io.InputStreamReader;
import java.net.*;
import java.util.ArrayList;
import java.util.Scanner;

public class importData {

    private ArrayList<String[]> IotData = new ArrayList<String[]>();
    private ArrayList<String[]> IFTTrules = new ArrayList<String[]>();
    private ArrayList<String[]> MRTrules = new ArrayList<String[]>();
    private String pathIFTTrules;
    private String pathMRTrules;
    private String pathIoTdata;

    public importData(String mrtpath, String IFTTpath, String pathData) {
        pathIFTTrules = IFTTpath;
        pathMRTrules = mrtpath;
        pathIoTdata = pathData;
    }

    public void get_updated_MRTrules() {

        String urlString = "http://10.16.30.73/api/get-metarules";

        try {

            StringBuilder result = new StringBuilder();
            URL url = new URL(urlString);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("GET");
            conn.connect();
            int responsecode = conn.getResponseCode();
            if (responsecode != 200) {
                throw new RuntimeException("HttpResponseCode: " + responsecode);
            } else {
                BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream()));
                String line;
                while ((line = br.readLine()) != null) {
                    result.append(line);
                }

                br.close();
```

```
                result.deleteCharAt(result.length() - 1);
                result.deleteCharAt(0);
                String[] stemp = result.toString().split("}");
                StringBuilder stringbld = new StringBuilder();

                for (String s : stemp) {
                    s = s.replace("{", "");
                    s = s.replace("\"", "");
                    String stemp2[] = s.split(",");
                    for (String s1 : stemp2) {
                        // System.out.println(s1);
                        String stemp3[] = s1.split(":");
                        if (stemp3[0].equals("time")) {
                            if (stemp3[1].equals("kWh")) {
                                stringbld.append("kWh");
                                stringbld.append("|");
                            } else {
                                stringbld.append(stemp3[1]);
                                stringbld.append(":");
                                stringbld.append(stemp3[2]);
                                stringbld.append(":");
                                stringbld.append(stemp3[3]);
                                stringbld.append("|");
                            }
                        } else if (stemp3[0].equals("action")) {
                            stringbld.append(stemp3[1]);
                            stringbld.append("|");
                        } else if (stemp3[0].equals("value")) {
                            stringbld.append(stemp3[1]);
                            stringbld.append("|");
                            stringbld.append(stemp3[1]);
                            stringbld.append("\n");
                        }
                    }
                    // System.out.println(s);

                }
                // System.out.println(stringbld);
                FileWriter myWriter = new FileWriter(pathMRTrules);
                myWriter.write("TIME| TEMPER./LIGHT    |    FROM  |  TO     |    \n");
                myWriter.write(
                        "-----------------------------------------------------------------

                myWriter.write(stringbld.toString());
                myWriter.close();

            }

        } catch (Exception e) {
            e.printStackTrace();
            System.out.println(e.getMessage());
        }

    }

    public void get_updated_MRTrules_GP() {
        // call api for data for location and date of today
        // put in dame structure as data I already have
        // put date and T105 and the value of temp the other put whatever
        // First store in text file and then import them like before
        // String API_KEY = "c5c61330eca67b966249ef80724111a6";
        // String LOCATION = "Limassol,CY";
        String urlString = "http://10.16.30.73/imcfgp/api/get-metarules";

        try {

            StringBuilder result = new StringBuilder();
            URL url = new URL(urlString);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("GET");
            conn.connect();
            int responsecode = conn.getResponseCode();
            if (responsecode != 200) {
                throw new RuntimeException("HttpResponseCode: " + responsecode);
            } else {
                BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream()));
                String line;
                while ((line = br.readLine()) != null) {
                    result.append(line);
                }

                br.close();

                result.deleteCharAt(result.length() - 1);
                result.deleteCharAt(0);
                String[] stemp = result.toString().split("}");
                StringBuilder stringbld = new StringBuilder();

                for (String s : stemp) {
                    s = s.replace("{", "");
                    s = s.replace("\"", "");
                    String stemp2[] = s.split(",");
                    for (String s1 : stemp2) {
                        // System.out.println(s1);
                        String stemp3[] = s1.split(":");
                        if (stemp3[0].equals("time")) {
```

```
                        if (stemp3[1].equals("kWh")) {
                            stringbld.append("kWh");
                            stringbld.append("|");
                        } else {
                            stringbld.append(stemp3[1]);
                            stringbld.append(":");
                            stringbld.append(stemp3[2]);
                            stringbld.append(":");
                            stringbld.append(stemp3[3]);
                            stringbld.append("|");
                        }
                    } else if (stemp3[0].equals("action")) {
                        stringbld.append(stemp3[1]);
                        stringbld.append("|");
                    } else if (stemp3[0].equals("value")) {
                        stringbld.append(stemp3[1]);
                        stringbld.append("|");
                        stringbld.append(stemp3[1]);
                        stringbld.append("\n");
                    }
                }
                // System.out.println(s);

            }
            // System.out.println(stringbld);
            FileWriter myWriter = new FileWriter(pathMRTrules);
            myWriter.write("TIME| TEMPER./LIGHT    |    FROM  |  TO     |    \n");
            myWriter.write(
                    "-----------------------------------------------------------------

            myWriter.write(stringbld.toString());
            myWriter.close();

        }

        } catch (Exception e) {
            e.printStackTrace();
            System.out.println(e.getMessage());
        }

    }

    public void import_IoT_data() {

        int counter = 0;
        String line;

        // Read the file and display it line by line.
```

```
        File file = new File(pathIoTdata);

        try {
            System.out.println("Importing Data:\t" + file.getName());
            long startime = System.nanoTime();
            Scanner inputFile = new Scanner(file);
            // loop through every line

            while (inputFile.hasNextLine()) {
                line = inputFile.nextLine();
                // String[] col = line.split("|");

                String temp[] = new String[7];
                if (counter > 1) {
                    String[] col = line.split("\\|");
                    temp[0] = col[0].trim();
                    temp[1] = col[1].trim();
                    temp[2] = col[2].trim();
                    temp[3] = col[3].trim();
                    temp[4] = col[4].trim();
                    temp[5] = col[5].trim();
                    temp[6] = col[6].trim();

                    IotData.add(temp); // fills a list with all the input IoT data
                }

                counter++;
            } // end of while loop

            inputFile.close();

            long finishtime = System.nanoTime() - startime;
            System.out.println("Import Completed!");
            System.out.println("Import Dataset Time:\t" + finishtime / 1000000 + "ms");
        } catch (Exception e) {
            System.out.println("Error with Import of IoT data.");
            System.out.println(e);
        }

        System.out.println("Number of records imported: " + IotData.size());

    }

    public void import_IFTTT_rules() {
        int counter = 0;
        String line;

        // Read the file and display it line by line.
```

```
241         File file = new File(pathIFTTrules);
242 
243         System.out.println("Importing IFTT Rules:\t" + file.getName());
244         try {
245             long startime = System.nanoTime();
246             Scanner inputFile = new Scanner(file);
247 
248             // loop through every line
249             while (inputFile.hasNextLine()) {
250                 line = inputFile.nextLine();
251                 String[] col = line.split("\\|");
252 
253                 String temp[] = new String[8];
254                 if (counter > 1) {
255                     temp[0] = col[0].trim();
256                     temp[1] = col[1].trim();
257                     temp[2] = col[5].trim();
258                     temp[3] = col[3].trim();
259                     temp[4] = col[6].trim();
260                     temp[5] = col[7].trim();
261                     temp[6] = col[8].trim();
262                     temp[7] = col[2].trim();
263                     IFTTrules.add(temp);
264                 } // end of inner while loop
265                 counter++;
266             } // end of outer while loop
267             long finishtime = System.nanoTime() - startime;
268             System.out.println("Import Completed!");
269             System.out.println("Import IFTT rules Time:\t" + finishtime / 1000000 + "ms");
270             inputFile.close();
271             System.out.printf("There were %d lines.\n", counter);
272         } catch (Exception e) {
273             System.out.println("Error with Import of IFTT rules.");
274             System.out.println(e);
275         }
276         // return IFTTrules;
277     }
278 
279     public void import_MRT_rules() {
280         int counter = 0;
281         String line;
282 
283         // Read the file and display it line by line.
284 
285         File file = new File(pathMRTrules);
286         System.out.println("Importing MRT Rules:\t" + file.getName());
287         try {
288             long startime = System.nanoTime();
289             Scanner inputFile = new Scanner(file);
290 
291             // loop through every line
292             while (inputFile.hasNextLine()) {
293                 line = inputFile.nextLine();
294                 String[] col = line.split("\\|");
295 
296                 String temp[] = new String[4];
297                 if (counter > 1) {
298 
299                     temp[0] = col[0].trim();
300                     temp[1] = col[2].trim();
301                     temp[2] = col[3].trim();
302                     temp[3] = col[1].trim();
303                     MRTrules.add(temp);
304 
305                 } // end of if
306                 counter++;
307 
308             } // end of while loop
309 
310 
311 
312             long finishtime = System.nanoTime() - startime;
313             System.out.println("Import Completed!");
314             System.out.println("Import MRT rules Time:\t" + finishtime / 1000000 + "ms");
315             inputFile.close();
316             System.out.printf("There were %d lines.\n", counter);
317         } catch (Exception e) {
318             System.out.println("Error with Import of MRT rules.");
319             System.out.println(e);
320         }
321 
322     }// END OF - Meta-rule Config SCENARIO 2
323 
324     public ArrayList<String[]> getIotData() {
325         return IotData;
326     }
327 
328     public ArrayList<String[]> getIFTTTrules() {
329         return IFTTrules;
330     }
331 
332     public ArrayList<String[]> getMRTrules() {
333         return MRTrules;
334     }
335 
336 }
```

noIFTT.java

```
package iotprojectpackage;

import java.io.FileWriter;
import java.math.RoundingMode;
import java.text.DecimalFormat;
import java.util.ArrayList;

public class noIFTT {

  private static ArrayList<String[]> IotData = new ArrayList<String[]>();
  private ArrayList<String[]> IotData3 = new ArrayList<String[]>(); // for use in noIFTT function
  private ArrayList<String> TempIotData = new ArrayList<String>();
  private static ArrayList<String[]> IFTTrules;
  private static ArrayList<String[]> MRTrules;
  private String errorLine = "";
  private double formaliseMaxError = 0;
  private String pathNoIFTTresults;

  public noIFTT(ArrayList<String[]> ifttrules, ArrayList<String[]> mrtrules, ArrayList<String[]> data,
      String pathForResults) {
    IFTTrules = new ArrayList<String[]>(ifttrules);
    MRTrules = new ArrayList<String[]>(mrtrules);
    IotData = new ArrayList<String[]>(data);
    pathNoIFTTresults = pathForResults;
  }

  public void printLine(String[] array) {
    int length = array.length;
    int count = 0;

    for (String temp : array) {
      if (count == length - 1) {
        System.out.println(temp);
      } else {
        System.out.print(temp + "|");
      }
      count++;
    }
  }

  public String makeDataLine(String[] array) {
    String s = "";
    String seperator = "|";
    int count = 0;
    int length = array.length;

    for (String temp : array) {
      if (count == length - 1) {
        s = s.concat(temp);
      } else {
        s = s.concat(temp);
        s = s.concat(seperator);
      }
      count++;
    }

    return s;
  }

  // returns formaliseMaxError value
  public double method() {

    String listRanges[][] = MRTrules.toArray(new String[0][]);

    double dTotalError = 0;
    double dTempError = 0;
    double dLineError = 0;

    long startime = System.nanoTime();
    double maxError = 0;
    double minError = 100;
    boolean errorFound = false;
    int errorCounter = 0;

    int line = 0;
    try {

      for (int i = 0; i < IotData.size(); i++) {
        // gets the total record line error in regards of all the ranges
        dLineError = 0;

        for (int k = 0; k < listRanges.length; k++) {
          String[] rangesSplitted = listRanges[k];
          errorLine = makeDataLine(IotData.get(i));

          String[] IotDataSplitted2 = IotData.get(i);
          // gets T for temperatures
          if (IotDataSplitted2[2].startsWith("T")) {

            // checks time and temperature and if season is empty
```

```
            if ((!(rangesSplitted[0].equals("SUMMER")) || !(rangesSplitted[0].equals("WINTER")))
                && (rangesSplitted[3].equals("TEMPER."))) {

              if ((Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) >= Double
                  .parseDouble(rangesSplitted[0].substring(0, 2))
                  && Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) <= Double
                      .parseDouble(rangesSplitted[0].substring(6, 8)))
                  && Double.parseDouble(IotDataSplitted2[3]) >= Double.parseDouble(rangesSplitted[1])
                  && Double.parseDouble(IotDataSplitted2[3]) <= Double
                      .parseDouble(rangesSplitted[2])) {
                IotData3.add(IotData.get(i));
                // its already on the list no need to add it twice
                break;

              } else {
                // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                if (Double.parseDouble(IotDataSplitted2[3]) < Double.parseDouble(rangesSplitted[1])) {
                  dTotalError += Double.parseDouble(rangesSplitted[1])
                      - Double.parseDouble(IotDataSplitted2[3]);
                  dLineError += Double.parseDouble(rangesSplitted[1])
                      - Double.parseDouble(IotDataSplitted2[3]);
                  TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
                  if (dLineError > maxError) {
                    maxError = dLineError;
                  }
                  if (dLineError < minError) {
                    minError = dLineError;
                  }
                  errorFound = true;
                } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
                    .parseDouble(rangesSplitted[2])) {
                  dTotalError += Double.parseDouble(IotDataSplitted2[3])
                      - Double.parseDouble(rangesSplitted[2]);
                  dLineError += Double.parseDouble(IotDataSplitted2[3])
                      - Double.parseDouble(rangesSplitted[2]);
                  TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
                  if (dLineError > maxError) {
                    maxError = dLineError;
                  }
                  if (dLineError < minError) {
                    minError = dLineError;
                  }
                  errorFound = true;

                }
```

```
              }
            }

            if (((rangesSplitted[0].equals("SUMMER")) || (rangesSplitted[0].equals("WINTER")))
                && (rangesSplitted[3].equals("TEMPER."))) {
              int iSeason = Integer.parseInt(IotDataSplitted2[0].substring(5, 7));

              // checks if SUMMER
              if ((iSeason == 6) || (iSeason == 7) || (iSeason == 8)) {
                if (Double.parseDouble(IotDataSplitted2[3]) >= Double.parseDouble(rangesSplitted[1])
                    && Double.parseDouble(IotDataSplitted2[3]) <= Double
                        .parseDouble(rangesSplitted[2])) {
                  IotData3.add(IotData.get(i));
                  // its already on the list no need to add it twice
                  break;
                } else {
                  // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                  if (Double.parseDouble(IotDataSplitted2[3]) < Double
                      .parseDouble(rangesSplitted[1])) {
                    dTotalError += Double.parseDouble(rangesSplitted[1])
                        - Double.parseDouble(IotDataSplitted2[3]);
                    dLineError += Double.parseDouble(rangesSplitted[1])
                        - Double.parseDouble(IotDataSplitted2[3]);
                    TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
                    if (dLineError > maxError) {
                      maxError = dLineError;
                    }
                    if (dLineError < minError) {
                      minError = dLineError;
                    }
                    errorFound = true;

                  } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
                      .parseDouble(rangesSplitted[2])) {
                    dTotalError += Double.parseDouble(IotDataSplitted2[3])
                        - Double.parseDouble(rangesSplitted[2]);
                    dLineError += Double.parseDouble(IotDataSplitted2[3])
                        - Double.parseDouble(rangesSplitted[2]);
                    TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
                    // break;
                    if (dLineError > maxError) {
                      maxError = dLineError;
                    }
                    if (dLineError < minError) {
```

```
                errorFound = true;

              }
            }
          }
          // checks if WINTER
          else if ((iSeason == 12) || (iSeason == 1) || (iSeason == 2)) {
            if (Double.parseDouble(IotDataSplitted2[3]) >= Double.parseDouble(rangesSplitted[1])
                && Double.parseDouble(IotDataSplitted2[3]) <= Double
                    .parseDouble(rangesSplitted[2])) {
              IotData3.add(IotData.get(i));
              // its already on the list no need to add it twice
              break;
            } else {

              if (Double.parseDouble(IotDataSplitted2[3]) < Double
                  .parseDouble(rangesSplitted[1])) {
                dTotalError += Double.parseDouble(rangesSplitted[1])
                    - Double.parseDouble(IotDataSplitted2[3]);
                dLineError += Double.parseDouble(rangesSplitted[1])
                    - Double.parseDouble(IotDataSplitted2[3]);
                TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
                // break;
                if (dLineError > maxError) {
                  maxError = dLineError;
                }
                if (dLineError < minError) {
                  minError = dLineError;
                }
                errorFound = true;

              } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
                  .parseDouble(rangesSplitted[2])) {
                dTotalError += Double.parseDouble(IotDataSplitted2[3])
                    - Double.parseDouble(rangesSplitted[2]);
                dLineError += Double.parseDouble(IotDataSplitted2[3])
                    - Double.parseDouble(rangesSplitted[2]);
                TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
                if (dLineError > maxError) {
                  maxError = dLineError;
                }
                if (dLineError < minError) {
                  minError = dLineError;
                }
                errorFound = true;
```

```
              }
            }
          }

        }

      }

      if (IotDataSplitted2[2].startsWith("LS")) // gets LS for light sensor
      {
        if (rangesSplitted[3].equals("LIGHT")) {
          if ((Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) >= Double
              .parseDouble(rangesSplitted[0].substring(0, 2))
              && Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) <= Double
                  .parseDouble(rangesSplitted[0].substring(6, 8)))
              && Integer.parseInt(IotDataSplitted2[3]) >= Integer.parseInt(rangesSplitted[1])
              && Integer.parseInt(IotDataSplitted2[3]) <= Integer.parseInt(rangesSplitted[2])) {
            IotData3.add(IotData.get(i));
            // its already on the list no need to add it twice
            break;
          } else {
            // assigns TOTAL ERROR AND CURRENT RECORD ERROR
            if (Double.parseDouble(IotDataSplitted2[3]) < Double.parseDouble(rangesSplitted[1])) {
              dTotalError += Double.parseDouble(rangesSplitted[1])
                  - Double.parseDouble(IotDataSplitted2[3]);
              dLineError += Double.parseDouble(rangesSplitted[1])
                  - Double.parseDouble(IotDataSplitted2[3]);
              TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
              // break;
              if (dLineError > maxError) {
                maxError = dLineError;
              }
              if (dLineError < minError) {
                minError = dLineError;
              }
              errorFound = true;

            } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
                .parseDouble(rangesSplitted[2])) {
              dTotalError += Double.parseDouble(IotDataSplitted2[3])
                  - Double.parseDouble(rangesSplitted[2]);
              dLineError += Double.parseDouble(IotDataSplitted2[3])
                  - Double.parseDouble(rangesSplitted[2]);
              TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
```

```
                    if (dLineError > maxError) {
                      maxError = dLineError;
                    }
                    if (dLineError < minError) {
                      minError = dLineError;
                    }
                    errorFound = true;

                  }
                }
              }

              if (rangesSplitted[3].equals("SUNNY")) {
                if ((Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) >= Double
                    .parseDouble(rangesSplitted[0].substring(0, 2))
                    && Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) <= Double
                        .parseDouble(rangesSplitted[0].substring(6, 8)))
                    && Integer.parseInt(IotDataSplitted2[3]) >= Integer.parseInt(rangesSplitted[1])
                    && Integer.parseInt(IotDataSplitted2[3]) <= Integer.parseInt(rangesSplitted[2])) {
                  IotData3.add(IotData.get(i));
                  // its already on the list no need to add it twice
                  break;
                } else {
                  // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                  if (Double.parseDouble(IotDataSplitted2[3]) < Double.parseDouble(rangesSplitted[1])) {
                    dTotalError += Double.parseDouble(rangesSplitted[1])
                        - Double.parseDouble(IotDataSplitted2[3]);
                    dLineError += Double.parseDouble(rangesSplitted[1])
                        - Double.parseDouble(IotDataSplitted2[3]);
                    TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
                    if (dLineError > maxError) {
                      maxError = dLineError;
                    }
                    if (dLineError < minError) {
                      minError = dLineError;
                    }
                    errorFound = true;

                  } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
                      .parseDouble(rangesSplitted[2])) {
                    dTotalError += Double.parseDouble(IotDataSplitted2[3])
                        - Double.parseDouble(rangesSplitted[2]);
                    dLineError += Double.parseDouble(IotDataSplitted2[3])
                        - Double.parseDouble(rangesSplitted[2]);
                    TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
```

```
                    if (dLineError > maxError) {
                      maxError = dLineError;
                    }
                    if (dLineError < minError) {
                      minError = dLineError;
                    }
                    errorFound = true;

                  }
                }
              }

              if (rangesSplitted[3].equals("CLOUDY")) {
                if ((Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) >= Double
                    .parseDouble(rangesSplitted[0].substring(0, 2))
                    && Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) <= Double
                        .parseDouble(rangesSplitted[0].substring(6, 8)))
                    && Integer.parseInt(IotDataSplitted2[3]) >= Integer.parseInt(rangesSplitted[1])
                    && Integer.parseInt(IotDataSplitted2[3]) <= Integer.parseInt(rangesSplitted[2])) {
                  IotData3.add(IotData.get(i));
                  // its already on the list no need to add it twice
                  break;
                } else {
                  // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                  if (Double.parseDouble(IotDataSplitted2[3]) < Double.parseDouble(rangesSplitted[1])) {
                    dTotalError += Double.parseDouble(rangesSplitted[1])
                        - Double.parseDouble(IotDataSplitted2[3]);
                    dLineError += Double.parseDouble(rangesSplitted[1])
                        - Double.parseDouble(IotDataSplitted2[3]);
                    TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
                    if (dLineError > maxError) {
                      maxError = dLineError;
                    }
                    if (dLineError < minError) {
                      minError = dLineError;
                    }
                    errorFound = true;

                  } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
                      .parseDouble(rangesSplitted[2])) {
                    dTotalError += Double.parseDouble(IotDataSplitted2[3])
                        - Double.parseDouble(rangesSplitted[2]);
                    dLineError += Double.parseDouble(IotDataSplitted2[3])
                        - Double.parseDouble(rangesSplitted[2]);
                    TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
```

```
                    if (dLineError > maxError) {
                      maxError = dLineError;
                    }
                    if (dLineError < minError) {
                      minError = dLineError;
                    }
                    errorFound = true;

                  }
                }
              }

              if (rangesSplitted[3].equals("DOOR OPEN-LIGHT")) {
                if ((Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) >= Double
                    .parseDouble(rangesSplitted[0].substring(0, 2))
                    && Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) <= Double
                        .parseDouble(rangesSplitted[0].substring(6, 8)))
                    && Integer.parseInt(IotDataSplitted2[3]) >= Integer.parseInt(rangesSplitted[1])
                    && Integer.parseInt(IotDataSplitted2[3]) <= Integer.parseInt(rangesSplitted[2])) {
                  IotData3.add(IotData.get(i));
                  break; // its already on the list no need to add it twice
                } else {
                  // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                  if (Double.parseDouble(IotDataSplitted2[3]) < Double.parseDouble(rangesSplitted[1])) {
                    dTotalError += Double.parseDouble(rangesSplitted[1])
                        - Double.parseDouble(IotDataSplitted2[3]);
                    dLineError += Double.parseDouble(rangesSplitted[1])
                        - Double.parseDouble(IotDataSplitted2[3]);
                    TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
                    if (dLineError > maxError) {
                      maxError = dLineError;
                    }
                    if (dLineError < minError) {
                      minError = dLineError;
                    }
                    errorFound = true;

                  } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
                      .parseDouble(rangesSplitted[2])) {
                    dTotalError += Double.parseDouble(IotDataSplitted2[3])
                        - Double.parseDouble(rangesSplitted[2]);
                    dLineError += Double.parseDouble(IotDataSplitted2[3])
                        - Double.parseDouble(rangesSplitted[2]);
                    TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
                    if (dLineError > maxError) {
                      maxError = dLineError;
                    }
                    if (dLineError < minError) {
                      minError = dLineError;
                    }
                    errorFound = true;

                  }
                }
              }

            }

            if (errorFound == true) {
              errorCounter = errorCounter + 1;
              errorFound = false;
            }

          }
        } catch (Exception ex) {
          System.out
              .println("error first(comparison data with meta-range rules): " + errorLine + ", error msg: " + ex);

        }

        // save formalized max error
        formaliseMaxError = maxError;

        dTotalError = dTotalError * 10;

        DecimalFormat df = new DecimalFormat("#.####");
        df.setRoundingMode(RoundingMode.CEILING);

        String lblError1 = "Total Error: " + Math.round((dTotalError) / formaliseMaxError);
        String label60 = "Max Error: " + df.format((maxError) * 100 / formaliseMaxError) + "%";
        String label59 = "Min Error: " + df.format((minError) * 100 / formaliseMaxError) + "%";

        Double averageErrorTotal = Double.parseDouble(df.format((dTotalError / IotData.size())));
        String label71 = "Total Average Error: " + df.format((averageErrorTotal * 100) / formaliseMaxError) + "%";

        TempIotData.clear();
        IotData3.clear();
```

```
455     System.out.println(lblError1);
456     System.out.println(label60);
457     System.out.println(label59);
458     // System.out.println(label58);
459     System.out.println(label71);
460     // ***********second run***************
461
462     Double meansSum = 0.0;
463     maxError = 0;
464     minError = 100;
465     dTotalError = 0;
466     dTempError = 0;
467
468     try {
469
470       for (int i = 0; i < IotData.size(); i++) {
471         for (int k = 0; k < listRanges.length; k++) {
472           String[] rangesSplitted = listRanges[k];
473
474           errorLine = makeDataLine(IotData.get(i));
475
476           String[] IotDataSplitted2 = IotData.get(i);
477
478           if (IotDataSplitted2[2].startsWith("T")) // gets T for temperatures
479           {
480             // checks time and temperature and if season is empty
481             if ((!(rangesSplitted[0].equals("SUMMER")) || !(rangesSplitted[0].equals("WINTER")))
482                 && (rangesSplitted[3].equals("TEMPER."))) {
483               if ((Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) >= Double
484                   .parseDouble(rangesSplitted[0].substring(0, 2))
485                   && Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) <= Double
486                       .parseDouble(rangesSplitted[0].substring(6, 8)))
487                   && Double.parseDouble(IotDataSplitted2[3]) >= Double.parseDouble(rangesSplitted[1])
488                   && Double.parseDouble(IotDataSplitted2[3]) <= Double
489                       .parseDouble(rangesSplitted[2])) {
490                 IotData3.add(IotData.get(i));
491                 break; // its already on the list no need to add it twice
492               } else {
493                 // assigns TOTAL ERROR AND CURRENT RECORD ERROR
494                 if (Double.parseDouble(IotDataSplitted2[3]) < Double.parseDouble(rangesSplitted[1])) {
495                   TempIotData.add(makeDataLine(IotData.get(i)));
496                   dTotalError += Double.parseDouble(rangesSplitted[1])
497                       - Double.parseDouble(IotDataSplitted2[3]);
498                   dTempError = Double.parseDouble(rangesSplitted[1])
499                       - Double.parseDouble(IotDataSplitted2[3]);
500                   break;
501                 } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
502                     .parseDouble(rangesSplitted[2])) {
503                   TempIotData.add(makeDataLine(IotData.get(i)));
504                   dTotalError += Double.parseDouble(IotDataSplitted2[3])
505                       - Double.parseDouble(rangesSplitted[2]);
506                   dTempError = Double.parseDouble(IotDataSplitted2[3])
507                       - Double.parseDouble(rangesSplitted[2]);
508                   break;
509                 }
510
511               }
512             }
513
514             if (((rangesSplitted[0].equals("SUMMER")) || (rangesSplitted[0].equals("WINTER")))
515                 && (rangesSplitted[3].equals("TEMPER."))) {
516               int iSeason = Integer.parseInt(IotDataSplitted2[0].substring(5, 7));
517
518               // checks if SUMMER
519               if ((iSeason == 6) || (iSeason == 7) || (iSeason == 8)) {
520                 if (Double.parseDouble(IotDataSplitted2[3]) >= Double.parseDouble(rangesSplitted[1])
521                     && Double.parseDouble(IotDataSplitted2[3]) <= Double
522                         .parseDouble(rangesSplitted[2])) {
523                   IotData3.add(IotData.get(i));
524                   break; // its already on the list no need to add it twice
525                 } else {
526                   // assigns TOTAL ERROR AND CURRENT RECORD ERROR
527                   if (Double.parseDouble(IotDataSplitted2[3]) < Double
528                       .parseDouble(rangesSplitted[1])) {
529                     TempIotData.add(makeDataLine(IotData.get(i)));
530                     dTotalError += Double.parseDouble(rangesSplitted[1])
531                         - Double.parseDouble(IotDataSplitted2[3]);
532                     dTempError = Double.parseDouble(rangesSplitted[1])
533                         - Double.parseDouble(IotDataSplitted2[3]);
534                     break;
535                   } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
536                       .parseDouble(rangesSplitted[2])) {
537                     TempIotData.add(makeDataLine(IotData.get(i)));
538                     dTotalError += Double.parseDouble(IotDataSplitted2[3])
539                         - Double.parseDouble(rangesSplitted[2]);
540                     dTempError = Double.parseDouble(IotDataSplitted2[3])
541                         - Double.parseDouble(rangesSplitted[2]);
542                     break;
543                   }
544                 }
```

```
            }
            // checks if WINTER
            else if ((iSeason == 12) || (iSeason == 1) || (iSeason == 2)) {
              if (Double.parseDouble(IotDataSplitted2[3]) >= Double.parseDouble(rangesSplitted[1])
                  && Double.parseDouble(IotDataSplitted2[3]) <= Double
                      .parseDouble(rangesSplitted[2])) {
                IotData3.add(IotData.get(i));
                break; // its already on the list no need to add it twice
              } else {
                // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                if (Double.parseDouble(IotDataSplitted2[3]) < Double
                    .parseDouble(rangesSplitted[1])) {
                  TempIotData.add(makeDataLine(IotData.get(i)));
                  dTotalError += Double.parseDouble(rangesSplitted[1])
                      - Double.parseDouble(IotDataSplitted2[3]);
                  dTempError = Double.parseDouble(rangesSplitted[1])
                      - Double.parseDouble(IotDataSplitted2[3]);
                  break;
                } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
                    .parseDouble(rangesSplitted[2])) {
                  TempIotData.add(makeDataLine(IotData.get(i)));
                  dTotalError += Double.parseDouble(IotDataSplitted2[3])
                      - Double.parseDouble(rangesSplitted[2]);
                  dTempError = Double.parseDouble(IotDataSplitted2[3])
                      - Double.parseDouble(rangesSplitted[2]);
                  break;
                }
              }
            }

          }

        }

        if (IotDataSplitted2[2].startsWith("LS")) // gets LS for light sensor
        {
          if (rangesSplitted[3].equals("LIGHT")) {
            if ((Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) >= Double
                .parseDouble(rangesSplitted[0].substring(0, 2))
                && Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) <= Double
                    .parseDouble(rangesSplitted[0].substring(6, 8)))
                && Integer.parseInt(IotDataSplitted2[3]) >= Integer.parseInt(rangesSplitted[1])
                && Integer.parseInt(IotDataSplitted2[3]) <= Integer.parseInt(rangesSplitted[2])) {
              IotData3.add(IotData.get(i));
              break; // its already on the list no need to add it twice
            } else {
              // assigns TOTAL ERROR AND CURRENT RECORD ERROR
              if (Double.parseDouble(IotDataSplitted2[3]) < Double.parseDouble(rangesSplitted[1])) {
                TempIotData.add(makeDataLine(IotData.get(i)));
                dTotalError += Double.parseDouble(rangesSplitted[1])
                    - Double.parseDouble(IotDataSplitted2[3]);
                dTempError = Double.parseDouble(rangesSplitted[1])
                    - Double.parseDouble(IotDataSplitted2[3]);
                break;
              } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
                  .parseDouble(rangesSplitted[2])) {
                TempIotData.add(makeDataLine(IotData.get(i)));
                dTotalError += Double.parseDouble(IotDataSplitted2[3])
                    - Double.parseDouble(rangesSplitted[2]);
                dTempError = Double.parseDouble(IotDataSplitted2[3])
                    - Double.parseDouble(rangesSplitted[2]);
                break;
              }
            }
          }

          if (rangesSplitted[3].equals("SUNNY")) {
            if ((Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) >= Double
                .parseDouble(rangesSplitted[0].substring(0, 2))
                && Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) <= Double
                    .parseDouble(rangesSplitted[0].substring(6, 8)))
                && Integer.parseInt(IotDataSplitted2[3]) >= Integer.parseInt(rangesSplitted[1])
                && Integer.parseInt(IotDataSplitted2[3]) <= Integer.parseInt(rangesSplitted[2])) {
              IotData3.add(IotData.get(i));
              break; // its already on the list no need to add it twice
            } else {
              // assigns TOTAL ERROR AND CURRENT RECORD ERROR
              if (Double.parseDouble(IotDataSplitted2[3]) < Double.parseDouble(rangesSplitted[1])) {
                TempIotData.add(makeDataLine(IotData.get(i)));
                dTotalError += Double.parseDouble(rangesSplitted[1])
                    - Double.parseDouble(IotDataSplitted2[3]);
                dTempError = Double.parseDouble(rangesSplitted[1])
                    - Double.parseDouble(IotDataSplitted2[3]);
                break;
              } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
                  .parseDouble(rangesSplitted[2])) {
                TempIotData.add(makeDataLine(IotData.get(i)));
                dTotalError += Double.parseDouble(IotDataSplitted2[3])
                    - Double.parseDouble(rangesSplitted[2]);
                dTempError = Double.parseDouble(IotDataSplitted2[3])
```

```
                    - Double.parseDouble(rangesSplitted[2]);
                break;
              }
            }
          }

          if (rangesSplitted[3].equals("CLOUDY")) {
            if ((Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) >= Double
                .parseDouble(rangesSplitted[0].substring(0, 2))
                && Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) <= Double
                    .parseDouble(rangesSplitted[0].substring(6, 8)))
                && Integer.parseInt(IotDataSplitted2[3]) >= Integer.parseInt(rangesSplitted[1])
                && Integer.parseInt(IotDataSplitted2[3]) <= Integer.parseInt(rangesSplitted[2])) {
              IotData3.add(IotData.get(i));
              break; // its already on the list no need to add it twice
            } else {
              // assigns TOTAL ERROR AND CURRENT RECORD ERROR
              if (Double.parseDouble(IotDataSplitted2[3]) < Double.parseDouble(rangesSplitted[1])) {
                TempIotData.add(makeDataLine(IotData.get(i)));
                dTotalError += Double.parseDouble(rangesSplitted[1])
                    - Double.parseDouble(IotDataSplitted2[3]);
                dTempError = Double.parseDouble(rangesSplitted[1])
                    - Double.parseDouble(IotDataSplitted2[3]);
                break;
              } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
                  .parseDouble(rangesSplitted[2])) {
                TempIotData.add(makeDataLine(IotData.get(i)));
                dTotalError += Double.parseDouble(IotDataSplitted2[3])
                    - Double.parseDouble(rangesSplitted[2]);
                dTempError = Double.parseDouble(IotDataSplitted2[3])
                    - Double.parseDouble(rangesSplitted[2]);
                break;
              }
            }
          }

          if (rangesSplitted[3].equals("DOOR OPEN-LIGHT")) {
            if ((Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) >= Double
                .parseDouble(rangesSplitted[0].substring(0, 2))
                && Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) <= Double
                    .parseDouble(rangesSplitted[0].substring(6, 8)))
                && Integer.parseInt(IotDataSplitted2[3]) >= Integer.parseInt(rangesSplitted[1])
                && Integer.parseInt(IotDataSplitted2[3]) <= Integer.parseInt(rangesSplitted[2])) {
              IotData3.add(IotData.get(i));
              break; // its already on the list no need to add it twice
```

```
          if (rangesSplitted[3].equals("DOOR OPEN-LIGHT")) {
            if ((Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) >= Double
                .parseDouble(rangesSplitted[0].substring(0, 2))
                && Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) <= Double
                    .parseDouble(rangesSplitted[0].substring(6, 8)))
                && Integer.parseInt(IotDataSplitted2[3]) >= Integer.parseInt(rangesSplitted[1])
                && Integer.parseInt(IotDataSplitted2[3]) <= Integer.parseInt(rangesSplitted[2])) {
              IotData3.add(IotData.get(i));
              break; // its already on the list no need to add it twice
            } else {
              // assigns TOTAL ERROR AND CURRENT RECORD ERROR
              if (Double.parseDouble(IotDataSplitted2[3]) < Double.parseDouble(rangesSplitted[1])) {
                TempIotData.add(makeDataLine(IotData.get(i)));
                dTotalError += Double.parseDouble(rangesSplitted[1])
                    - Double.parseDouble(IotDataSplitted2[3]);
                dTempError = Double.parseDouble(rangesSplitted[1])
                    - Double.parseDouble(IotDataSplitted2[3]);
                break;
              } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
                  .parseDouble(rangesSplitted[2])) {
                TempIotData.add(makeDataLine(IotData.get(i)));
                dTotalError += Double.parseDouble(IotDataSplitted2[3])
                    - Double.parseDouble(rangesSplitted[2]);
                dTempError = Double.parseDouble(IotDataSplitted2[3])
                    - Double.parseDouble(rangesSplitted[2]);
                break;
              }
            }
          }

        }

      }

      meansSum = meansSum + Math.pow(dLineError - averageErrorTotal, 2);

    }
  } catch (Exception e) {
    System.out.println(
        "error second(comparison data with meta-range rules): " + errorLine + " , Error message: " + e);
    System.out.println("Number of line:\t" + e.getStackTrace()[2].getLineNumber());
  }

  double standardDeviation = Math.sqrt(meansSum / IotData.size());
```

```
716     String label57 = "Standard Deviation: " + df.format((standardDeviation) * 100 / formaliseMaxError) + "%";
717     System.out.println(label57);
718
719     IotData3.clear(); // clears the entire list
720     TempIotData.clear();
721
722     try {
723
724       FileWriter myWriter = new FileWriter(pathNoIFTTresults);
725       myWriter.write(lblError1);
726       myWriter.write("\n");
727       myWriter.write(label60);
728       myWriter.write("\n");
729
730       myWriter.write(label59);
731       myWriter.write("\n");
732
733       // System.out.println(label58);
734       myWriter.write(label71);
735       myWriter.write("\n");
736
737       myWriter.write(label57);
738       myWriter.write("\n");
739       myWriter.close();
740
741     } catch (Exception e) {
742       System.out.println("Unable to write data for noIFTT.");
743     }
744
745     return formaliseMaxError;
746   } // end of comparison imported data and meta rule ranges
747
748 }
749
```

EnergyPlanner.java

```java
/*
    This file implements the Energy Planner algorithm.
    Copyright (C) 2021  Antonis Vasileiou
    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.
    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.
    You should have received a copy of the GNU General Public License
    along with this program.  If not, see https://www.gnu.org/licenses/gpl-3.0.html.
*/

package iotprojectpackage;

import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.FileWriter;
import java.io.InputStreamReader;
import java.math.RoundingMode;
import java.net.HttpURLConnection;
import java.net.URL;
import java.nio.charset.StandardCharsets;
import java.text.DecimalFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.ZoneId;
import java.util.ArrayList;
import java.util.Date;
import java.util.Random;
import java.text.DateFormatSymbols;

import com.google.gson.Gson;
import com.google.gson.JsonArray;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;

public class EnergyPlanner {

    private String errorLine;// global

    private SimpleDateFormat simpleFormatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    private Date dtTemp;
    private double contiguous_ms = 0;
    private String monthlyKWH_listAssignment = "original"; // to help identify allowed kwh list->original, 1%,5%
                                                           // 20%
    private ArrayList<String[]> copiediotdata;
    private ArrayList<String[]> EnergyPlannerData;
    private ArrayList<String[]> copiediotdata3;
    private ArrayList<String[]> MRTrules;
    private String kwhpermonth = "";
    private Boolean solutionsFound;
    private ArrayList<String[]> recursionResults = new ArrayList<String[]>();
    private double public_ms = 0;
    private int recursiveIterator = 0;
    private String urlString = "http://10.16.30.73/api/update-results";
    private double totalaverageerror = 0.0;
    private double budgeterror = 0.0;
    private double convinienceerror = 0.0;
    private double deviation = 0.0;
    private double totalkwh = 0.0;
    private String listRanges[][];

    public EnergyPlanner() {
        solutionsFound = false;
    }

    public void ArrayList_fromString_toStringArray(ArrayList<String> list_from, ArrayList<String[]> list_to) {

        for (int i = 0; i < list_from.size(); i++) {
            String temp[] = list_from.get(i).split("\\|");
            list_to.add(temp);
        }

    }

    // call in line 1313
    public void updateOpenhabRules(String type, String state) {

        String geturl = "http://10.16.30.73/api/get-openhabrules";
        String posturl = "http://10.16.30.73/api/update-openhabrules";
        String postdata = "";
        try {

            StringBuilder result = new StringBuilder();
            URL url = new URL(geturl);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("GET");
            conn.connect();
```

```
92              int responsecode = conn.getResponseCode();
93              if (responsecode != 200) {
94                  throw new RuntimeException("HttpResponseCode: " + responsecode);
95              } else {
96                  BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream()));
97                  String line;
98                  while ((line = br.readLine()) != null) {
99                      result.append(line);
100                 }
101
102                 br.close();
103
104                 String temp = result.toString();
105
106                 JsonArray array = (JsonArray) JsonParser.parseString(temp);
107
108                 int size = array.size();
109
110                 if (type.equals("Light")) {
111
112                     for (int i = 0; i < size; i++) {
113
114                         JsonObject obj = array.get(i).getAsJsonObject();
115
116                         if (obj.get("category").getAsString().equals("Temperature")) {
117                             String state_value = obj.get("state").getAsString().trim();
118                             String[] str = state_value.split("\\s+");
119                             obj.remove("state");
120                             obj.addProperty("state", str[0] + " \u00b0C"); // This is to handle the Celcious sign
121                         }
122
123                         if (obj.get("category").getAsString().equals("Light")) {
124
125                             obj.remove("stateONOFF");
126                             obj.addProperty("stateONOFF", state);
127
128                         }
129
130                         array.set(i, obj);
131
132                     }
133
134                     postdata = array.toString();
135                     String urlParameters = "rules=" + postdata;
136                     byte[] postData = urlParameters.getBytes(StandardCharsets.UTF_8);
137                     int postDataLength = postData.length;
138                     url = new URL(posturl);
139                     conn = (HttpURLConnection) url.openConnection();
140
141                     conn.setDoOutput(true);
142                     conn.setRequestMethod("POST");
143                     conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
144                     conn.setRequestProperty("Charset", "UTF-8");
145                     conn.setRequestProperty("Content-Length", Integer.toString(postDataLength));
146                     conn.setUseCaches(false);
147
148                     conn.connect();
149
150                     DataOutputStream wr = new DataOutputStream(conn.getOutputStream());
151                     wr.write(postData);
152                     conn.getResponseCode();
153
154                 } // End of If
155                 else if (type.equals("Temperature")) {
156                     for (int i = 0; i < size; i++) {
157
158                         JsonObject obj = array.get(i).getAsJsonObject();
159
160                         if (obj.get("category").getAsString().equals("Temperature")) {
161                             String state_value = obj.get("state").getAsString().trim();
162                             String[] str = state_value.split("\\s+");
163                             obj.remove("state");
164                             obj.addProperty("state", str[0] + " \u00b0C"); // This is to handle the Celcious sign
165                             obj.remove("stateONOFF");
166                             obj.addProperty("stateONOFF", state);
167
168                         }
169
170                         array.set(i, obj);
171
172                     }
173
174                     postdata = array.toString();
175                     String urlParameters = "rules=" + postdata;
176                     byte[] postData = urlParameters.getBytes(StandardCharsets.UTF_8);
177                     int postDataLength = postData.length;
178                     url = new URL(posturl);
179                     conn = (HttpURLConnection) url.openConnection();
180
181                     conn.setDoOutput(true);
```

```
                    conn.setRequestMethod("POST");
                    conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
                    conn.setRequestProperty("Charset", "UTF-8");
                    conn.setRequestProperty("Content-Length", Integer.toString(postDataLength));
                    conn.setUseCaches(false);

                    conn.connect();

                    DataOutputStream wr = new DataOutputStream(conn.getOutputStream());
                    wr.write(postData);
                    conn.getResponseCode();
                }

            }

        } catch (Exception e) {
            e.printStackTrace();
            System.out.println(e.getMessage());
        }

    }

    // converts string to double
    public double toDouble(String str) {

        return Double.parseDouble(str);

    }

    // converts string to double
    public int toInt(String str) {

        return Integer.parseInt(str);

    }

    // returns hour from record datetime
    public int getHour(String date) {

        return toInt(date.substring(11, 13));

    }

    // returns month from record datetime
    public int getMonth(String date) {

        return toInt(date.substring(5, 7));

    }

    // returns the hour the rule starts
    public int ruleHourFrom(String ruletime) {

        return toInt(ruletime.substring(0, 2));

    }

    // returns the hour the rule ends
    public int ruleHourTo(String ruletime) {

        return toInt(ruletime.substring(6, 8));

    }

    public void postData() {
        try {

            String urlParameters = "ep_kWh=" + totalkwh + "&ep_totalError=" + totalaverageerror
                    + "&ep_convenienceError=" + convinienceerror + "&ep_budgetError=" + budgeterror
                    + "&ep_standardDeviation=" + deviation + "&kwh_per_month=" + kwhpermonth;
            byte[] postData = urlParameters.getBytes(StandardCharsets.UTF_8);
            int postDataLength = postData.length;

            URL url = new URL(urlString);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();

            conn.setDoOutput(true);
            conn.setRequestMethod("POST");
            conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
            conn.setRequestProperty("Charset", "UTF-8");
            conn.setRequestProperty("Content-Length", Integer.toString(postDataLength));
            conn.setUseCaches(false);

            conn.connect();

            DataOutputStream wr = new DataOutputStream(conn.getOutputStream());
            wr.write(postData);
            conn.getResponseCode();

        } catch (Exception e) {
```

```
            for (StackTraceElement s : e.getStackTrace())
                System.out.println(s);
        }
    }

    public String makeDataLine(String[] array) {
        String s = "";
        String seperator = "|";
        int count = 0;
        int length = array.length;

        for (String temp : array) {
            if (count == length - 1) {
                s = s.concat(temp);
            } else {
                s = s.concat(temp);
                s = s.concat(seperator);
            }
            count++;
        }

        return s;
    }

    public static String[] addX(String myarray[], String ele) {

        int n = myarray.length;
        String newArray[] = new String[n + 1];
        // copy original array into new array
        for (int i = 0; i < n; i++)
            newArray[i] = myarray[i];

        // add element to the new array
        newArray[n] = ele;

        return newArray;
    }

    private double calculateKWH3(ArrayList<String[]> sRecordsHourly) {
        ArrayList<String[]> copiediotdatakwh = new ArrayList<String[]>(sRecordsHourly);
        ArrayList<Integer> removedMetaRules = new ArrayList<Integer>();
        boolean randomRule[] = new boolean[MRTrules.size()];
        Random rand = new Random();
        int numrules = MRTrules.size();
        int random;
```

```
        Boolean startALLzeros = false;
        Boolean startALLrandom = false;
        int numberOfLoops = 3;

        if (removedMetaRules.size() == 0) {
            if (startALLzeros == true) {
                for (int d = 0; d < randomRule.length; d++) {
                    randomRule[d] = false;
                    removedMetaRules.add(d);
                }
            } else if (startALLrandom == true) {
                for (int d = 0; d < randomRule.length; d++) {
                    // generates random number from 0 to 1
                    random = rand.nextInt(2);
                    if (random == 0) {
                        randomRule[d] = false;
                        removedMetaRules.add(d);
                    } else {
                        randomRule[d] = true;
                    }
                }
            } else {
                for (int d = 0; d < randomRule.length; d++) {
                    randomRule[d] = true;
                }
            }

            for (int i = 0; i < numberOfLoops; i++) {
                random = rand.nextInt(numrules);
                // random here is the rule to be removed
                if (randomRule[random]) {
                    randomRule[random] = false;
                    removedMetaRules.add(random);
                } else {
                    randomRule[random] = true;
                    removedMetaRules.remove(Integer.valueOf(random));
                }
            }
        }

        if (solutionsFound == false) {

            for (int i = 0; i < numberOfLoops; i++) {
                random = rand.nextInt(numrules);
                // random here is the rule to be removed
```

```
                if (randomRule[random]) {
                    randomRule[random] = false;
                    removedMetaRules.add(random);
                } else {
                    randomRule[random] = true;
                    removedMetaRules.remove(Integer.valueOf(random));
                }
            }
        } else if (solutionsFound == true) {
            for (int d = 0; d < randomRule.length; d++) {
                randomRule[d] = true;
            }

            for (int i = 0; i < removedMetaRules.size(); i++) {
                randomRule[removedMetaRules.get(i)] = false;
            }
        }

        int IFTTTconstraintsApplied = 0;
        double milliWatts = 0;
        Date dt1 = dtTemp;
        Date dt2;
        double span;
        double ms = 0;
        double localContiguous_ms = contiguous_ms;
        ArrayList<Double> monthlyKWH = new ArrayList<Double>();

        if (monthlyKWH_listAssignment.equals("original")) {
            monthlyKWH = new ArrayList<Double>(); // 0.24
            Double[] d = new Double[] { 1.04, 0.71, 0.33, 0.19, 0.24, 0.28, 0.33, 0.43, 0.28, 0.34, 0.28, 0.57 };
            for (int i = 0; i < d.length; i++)
                monthlyKWH.add(d[i]);
        } else if (monthlyKWH_listAssignment.equals("1")) {
            monthlyKWH = new ArrayList<Double>();
            Double[] d = new Double[] { 1.03, 0.70, 0.33, 0.19, 0.23, 0.28, 0.33, 0.42, 0.28, 0.23, 0.28, 0.56 };
            for (int i = 0; i < d.length; i++)
                monthlyKWH.add(d[i]);
        } else if (monthlyKWH_listAssignment.equals("5")) {
            monthlyKWH = new ArrayList<Double>();

            Double[] d = new Double[] { 0.99, 0.68, 0.32, 0.18, 0.23, 0.27, 0.32, 0.41, 0.27, 0.23, 0.27, 0.54 };
            for (int i = 0; i < d.length; i++)
                monthlyKWH.add(d[i]);
        } else if (monthlyKWH_listAssignment.equals("10")) {
            monthlyKWH = new ArrayList<Double>();
```

```

            Double[] d = new Double[] { 0.94, 0.64, 0.30, 0.17, 0.21, 0.26, 0.30, 0.38, 0.26, 0.21, 0.26, 0.51 };
            for (int i = 0; i < d.length; i++)
                monthlyKWH.add(d[i]);
        } else if (monthlyKWH_listAssignment.equals("20")) {
            monthlyKWH = new ArrayList<Double>();

            Double[] d = new Double[] { 0.83, 0.57, 0.27, 0.15, 0.19, 0.23, 0.27, 0.34, 0.23, 0.19, 0.23, 0.45 };
            for (int i = 0; i < d.length; i++)
                monthlyKWH.add(d[i]);
        } else if (monthlyKWH_listAssignment.equals("30")) {
            monthlyKWH = new ArrayList<Double>();
            Double[] d = new Double[] { 0.73, 0.50, 0.23, 0.13, 0.17, 0.20, 0.23, 0.30, 0.20, 0.17, 0.20, 0.40 };
            for (int i = 0; i < d.length; i++)
                monthlyKWH.add(d[i]);
        } else if (monthlyKWH_listAssignment.equals("40")) {
            monthlyKWH = new ArrayList<Double>();
            Double[] d = new Double[] { 0.63, 0.43, 0.20, 0.11, 0.14, 0.17, 0.20, 0.26, 0.17, 0.14, 0.17, 0.34 };
            for (int i = 0; i < d.length; i++)
                monthlyKWH.add(d[i]);
        }
        try {

            ArrayList<String> listRules = new ArrayList<String>();
            for (int d = 0; d < randomRule.length; d++) {
                if (randomRule[d]) {

                    // temperature
                    if (listRanges[d][3].equals("TEMPER.")) {
                        listRules.add(listRanges[d][0] + "|" + "+" + listRanges[d][1] + "|" + "---" + "|"
                                + listRanges[d][1] + "|" + "---" + "|" + "---" + "|" + "---" + "|" + "SET TEMPERATURE");
                    }
                    // light
                    if (listRanges[d][3].equals("LIGHT")) {
                        listRules.add(listRanges[d][0] + "|" + "---" + "|" + "+" + listRanges[d][1] + "|"
                                + listRanges[d][1] + "|" + "---" + "|" + "---" + "|" + "---" + "|" + "SET LIGHT");
                    }
                    IFTTTconstraintsApplied++;
                }
            }

            for (int i = 0; i < copiediotdatakwh.size(); i++) {

                for (int k = 0; k < listRules.size(); k++) {
                    errorLine = String.join("|", copiediotdatakwh.get(i));
```

```
452
453                     String[] copiediotdataSplitted = copiediotdatakwh.get(i);
454                     String[] rulesSplitted = listRules.get(k).split("\\|");
455
456                     if (copiediotdataSplitted[2].startsWith("T")) // gets T for temperatures
457                     {
458                         // checks time and temperature and if season,wether,door is empty
459                         if ((!rulesSplitted[1].equals("---")) && (rulesSplitted[4].equals("---"))
460                                 && (rulesSplitted[5].equals("---")) && (rulesSplitted[6].equals("---"))) {
461
462                             if ((getHour(copiediotdataSplitted[0]) >= ruleHourFrom(rulesSplitted[0])
463                                     && getHour(copiediotdataSplitted[0]) <= ruleHourTo(rulesSplitted[0]))) {
464                                 dt2 = simpleFormatter.parse(copiediotdataSplitted[0].substring(0, 18));
465                                 span = Math.abs(dt2.getTime() - dt1.getTime());
466                                 ms += span;
467                                 dt1 = dt2;
468
469                                 localContiguous_ms += span;
470                                 double secondsTemp = localContiguous_ms / 1000;
471                                 double hoursTemp = secondsTemp / 3600;
472                                 double kWhTemp = ((hoursTemp * 320) / 1000) * 24;
473                                 copiediotdataSplitted = addX(copiediotdataSplitted, String.valueOf(kWhTemp));
474
475                                 milliWatts = milliWatts + 320;
476                                 copiediotdataSplitted[3] = rulesSplitted[3];
477                                 copiediotdatakwh.set(i, copiediotdataSplitted);
478                                 break; // its already on the list no need to add it twice
479                             }
480
481                         }
482
483                     }
484
485                     if (copiediotdataSplitted[2].startsWith("LS")) // gets LS for light sensor
486                     {
487                         // checks LIGHT , and if weather,door is empty
488                         if ((!rulesSplitted[2].equals("---")) && (rulesSplitted[5].equals("---"))
489                                 && (rulesSplitted[6].equals("---"))) {
490                             // checks time and light value
491                             if ((getHour(copiediotdataSplitted[0]) >= ruleHourFrom(rulesSplitted[0])
492                                     && getHour(copiediotdataSplitted[0]) <= ruleHourTo(rulesSplitted[0]))) {
493                                 dt2 = simpleFormatter.parse(copiediotdataSplitted[0].substring(0, 18));
494                                 span = Math.abs(dt2.getTime() - dt1.getTime());
495                                 ms += span;
496                                 dt1 = dt2;
497
498                                 localContiguous_ms += span;
499                                 double secondsTemp = localContiguous_ms / 1000;
500                                 double hoursTemp = secondsTemp / 3600;
501                                 double kWhTemp = ((hoursTemp * 320) / 1000) * 24;
502                                 copiediotdataSplitted = addX(copiediotdataSplitted, String.valueOf(kWhTemp));
503
504                                 copiediotdataSplitted[3] = rulesSplitted[3];
505                                 copiediotdatakwh.set(i, copiediotdataSplitted);
506                                 break;
507                             }
508
509                         } // end of : //checks LIGHT and if weather is empty
510
511                     }
512
513                     if (k == listRules.size() - 1) {
514                         dt1 = simpleFormatter.parse(copiediotdataSplitted[0].substring(0, 18));
515                     }
516
517                 }
518
519                 String[] copiediotdataSplittedAfterIFTTT = copiediotdatakwh.get(i);
520                 if (copiediotdataSplittedAfterIFTTT.length < 8) {
521
522                     copiediotdatakwh.set(i, addX(copiediotdataSplittedAfterIFTTT, "00"));
523                 }
524             }
525         } catch (Exception ex) {
526             System.out.println("First B: " + errorLine + " => Exception: " + ex);
527         }
528
529         double seconds = ms / 1000;
530         double hours = seconds / 3600;
531
532         double kWh = ((hours * 320) / 1000) * 24;
533
534         String[] copiediotdataSplitted_Month = copiediotdatakwh.get(0);
535         Date getMonth;
536         int currentMonth = 0;
537         try {
538             getMonth = simpleFormatter.parse(copiediotdataSplitted_Month[0].substring(0, 18));
539             currentMonth = getMonth.toInstant().atZone(ZoneId.systemDefault()).toLocalDate().getMonthValue();
540
541         } catch (ParseException e1) {
```

```
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        double allowedMonthly_kWh = toDouble(monthlyKWH.get(currentMonth - 1).toString()) * 6.5;// 6.5, *7.9;

        // **********SECOND AND LAST
        // PHASE*****************************************************************//
        double dTotalError = 0;
        double dLineError = 0;
        double dTotalError_kWh = 0;
        double maxError = 0;
        double minError = 100;
        boolean errorFound = false;
        int errorsize = 0;

        try {

            for (int i = 0; i < copiediotdatakwh.size(); i++) {

                dLineError = 0;// gets the total record line error in regards of all the ranges
                for (int k = 0; k < listRanges.length; k++) {
                    errorLine = String.join("|", copiediotdatakwh.get(i));

                    String[] copiediotdataSplitted2 = copiediotdatakwh.get(i);
                    // System.out.println(copiediotdatakwh.get(i));
                    String[] rangesSplitted = listRanges[k];

                    if (copiediotdataSplitted2[2].startsWith("T")) // gets T for temperatures
                    {

                        // checks time and temperature and if season is empty
                        if (((!rangesSplitted[0].equals("SUMMER")) || (!rangesSplitted[0].equals("WINTER")))
                                && (rangesSplitted[3].equals("TEMPER."))) {
                            if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
                                    && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
                                    && toDouble(copiediotdataSplitted2[3]) >= toDouble(rangesSplitted[1])
                                    && toDouble(copiediotdataSplitted2[3]) <= toDouble(rangesSplitted[2])) {
                                copiediotdata3.add(copiediotdatakwh.get(i));

                            } else {
                                // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                                if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
                                        && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
                                        && toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
                                    dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
                                    dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);

                                    if (dLineError > maxError) {
                                        maxError = dLineError;
                                    }
                                    if (dLineError < minError) {
                                        minError = dLineError;
                                    }
                                    errorFound = true;
                                } else if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
                                        && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
                                        && toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
                                    dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
                                    dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);

                                    if (dLineError > maxError) {
                                        maxError = dLineError;
                                    }
                                    if (dLineError < minError) {
                                        minError = dLineError;
                                    }
                                    errorFound = true;
                                }

                            }
                        }

                        if (((rangesSplitted[0].equals("SUMMER")) || (rangesSplitted[0].equals("WINTER")))
                                && (rangesSplitted[3].equals("TEMPER."))) {
                            int iSeason = getMonth(copiediotdataSplitted2[0]);

                            // checks if SUMMER
                            if ((iSeason == 6) || (iSeason == 7) || (iSeason == 8)) {
                                if (toDouble(copiediotdataSplitted2[3]) >= toDouble(rangesSplitted[1])
                                        && toDouble(copiediotdataSplitted2[3]) <= toDouble(rangesSplitted[2])) {
                                    copiediotdata3.add(copiediotdatakwh.get(i));

                                } else {
                                    // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                                    if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
                                        dTotalError += toDouble(rangesSplitted[1])
                                                - toDouble(copiediotdataSplitted2[3]);
                                        dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);

                                        if (dLineError > maxError) {
```

```
                                    maxError = dLineError;
                                }
                                if (dLineError < minError) {
                                    minError = dLineError;
                                }
                                errorFound = true;
                            } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
                                dTotalError += toDouble(copiediotdataSplitted2[3])
                                        - toDouble(rangesSplitted[2]);
                                dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);

                                if (dLineError > maxError) {
                                    maxError = dLineError;
                                }
                                if (dLineError < minError) {
                                    minError = dLineError;
                                }
                                errorFound = true;
                            }
                        }
                    }
                    // checks if WINTER
                    else if ((iSeason == 12) || (iSeason == 1) || (iSeason == 2)) {
                        if (toDouble(copiediotdataSplitted2[3]) >= toDouble(rangesSplitted[1])
                                && toDouble(copiediotdataSplitted2[3]) <= toDouble(rangesSplitted[2])) {
                            copiediotdata3.add(copiediotdatakwh.get(i));

                        } else {
                            // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                            if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
                                dTotalError += toDouble(rangesSplitted[1])
                                        - toDouble(copiediotdataSplitted2[3]);
                                dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);

                                if (dLineError > maxError) {
                                    maxError = dLineError;
                                }
                                if (dLineError < minError) {
                                    minError = dLineError;
                                }
                                errorFound = true;
                            } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
                                dTotalError += toDouble(copiediotdataSplitted2[3])
                                        - toDouble(rangesSplitted[2]);
                                dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
```

```
                                if (dLineError > maxError) {
                                    maxError = dLineError;
                                }
                                if (dLineError < minError) {
                                    minError = dLineError;
                                }
                                errorFound = true;
                            }
                        }
                    }

                }

                if (rangesSplitted[0].equals("kWh")) {
                    if (toDouble(copiediotdataSplitted2[7]) <= toDouble(rangesSplitted[1])) {
                        copiediotdata3.add(copiediotdatakwh.get(i));
                    } else {
                        dTotalError_kWh += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
                        dTotalError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
                        dLineError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);

                        if (dLincError > maxError) {
                            maxError = dLineError;
                        }
                        if (dLineError < minError) {
                            minError = dLineError;
                        }
                        errorFound = true;
                    }
                }

            }

            if (copiediotdataSplitted2[2].startsWith("LS")) // gets LS for light sensor
            {
                // checks LIGHT
                if (rangesSplitted[3].equals("LIGHT")) {
                    if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
                            && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
                            && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
                            && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
                        copiediotdata3.add(copiediotdatakwh.get(i));

                    } else {
```

```
722                         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
723                         if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
724                                 && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
725                                 && toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
726                             dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
727                             dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
728 
729                             if (dLineError > maxError) {
730                                 maxError = dLineError;
731                             }
732                             if (dLineError < minError) {
733                                 minError = dLineError;
734                             }
735                             errorFound = true;
736                         } else if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
737                                 && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
738                                 && toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
739                             dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
740                             dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
741 
742                             if (dLineError > maxError) {
743                                 maxError = dLineError;
744                             }
745                             if (dLineError < minError) {
746                                 minError = dLineError;
747                             }
748                             errorFound = true;
749                         }
750                     }
751                 } // end of: checks LIGHT
752 
753                 // checks SUNNY
754                 if (rangesSplitted[3].equals("SUNNY")) {
755                     if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
756                             && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
757                             && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
758                             && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
759                         copiediotdata3.add(copiediotdatakwh.get(i));
760 
761                     } else {
762                         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
763                         if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
764                             dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
765                             dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
766 
```

```
767                             if (dLineError > maxError) {
768                                 maxError = dLineError;
769                             }
770                             if (dLineError < minError) {
771                                 minError = dLineError;
772                             }
773                             errorFound = true;
774                         } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
775                             dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
776                             dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
777 
778                             if (dLineError > maxError) {
779                                 maxError = dLineError;
780                             }
781                             if (dLineError < minError) {
782                                 minError = dLineError;
783                             }
784                             errorFound = true;
785                         }
786                     }
787                 } // end of :checks SUNNY
788 
789                 // checks CLOUDY
790                 if (rangesSplitted[3].equals("CLOUDY")) {
791                     if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
792                             && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
793                             && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
794                             && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
795                         copiediotdata3.add(copiediotdatakwh.get(i));
796 
797                     } else {
798                         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
799                         if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
800                             dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
801                             dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
802 
803                             if (dLineError > maxError) {
804                                 maxError = dLineError;
805                             }
806                             if (dLineError < minError) {
807                                 minError = dLineError;
808                             }
809                             errorFound = true;
810                         } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
811                             dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
```

```
                                dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);

                                if (dLineError > maxError) {
                                    maxError = dLineError;
                                }
                                if (dLineError < minError) {
                                    minError = dLineError;
                                }
                                errorFound = true;
                            }
                        }
                    } // end of :checks CLOUDY

                    // checks DOOR OPEN-LIGHT
                    if (rangesSplitted[3].equals("DOOR OPEN-LIGHT")) {
                        if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
                                && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
                                && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
                                && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
                            copiediotdata3.add(copiediotdatakwh.get(i));

                        } else {
                            // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                            if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
                                dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
                                dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);

                                if (dLineError > maxError) {
                                    maxError = dLineError;
                                }
                                if (dLineError < minError) {
                                    minError = dLineError;
                                }
                                errorFound = true;
                            } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
                                dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
                                dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);

                                if (dLineError > maxError) {
                                    maxError = dLineError;
                                }
                                if (dLineError < minError) {
                                    minError = dLineError;
                                }
                                errorFound = true;
```

```
                            }
                        }
                    } // end of :checks DOOR OPEN-LIGHT

                    if (rangesSplitted[0].equals("kWh")) {
                        if (toDouble(copiediotdataSplitted2[7]) <= toDouble(rangesSplitted[1])) {
                            copiediotdata3.add(copiediotdatakwh.get(i));
                        } else {
                            dTotalError_kWh += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
                            dTotalError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
                            dLineError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);

                            if (dLineError > maxError) {
                                maxError = dLineError;
                            }
                            if (dLineError < minError) {
                                minError = dLineError;
                            }
                            errorFound = true;
                        }
                    } // end of kWh check

                } // end of LS

            }

            if (errorFound == true) {
                errorsize++;
                errorFound = false;
            }

        }
    } catch (Exception e) {

        System.out.println("Second B: " + errorLine + "\tException:\t" + e);

        e.printStackTrace();
    }

    // TempIotData.clear();
    copiediotdata3.clear();

    // ******************************end of second phase (calculating Error
    // phase)*********************
```

```
902         if ((kWh > allowedMonthly_kWh) && (solutionsFound == false)) {
903
904             if (removedMetaRules.size() == numrules) {
905                 removedMetaRules.clear();
906             }
907             calculateKWH3(sRecordsHourly);
908             return public_ms;
909
910         } else if (recursiveIterator < 5) {
911             if (recursionResults.size() == 0)// check if result found first time
912             {
913                 String temp[] = new String[3];
914                 String t = "";
915                 boolean first = true;
916                 for (int d = 0; d < removedMetaRules.size(); d++) {
917                     if (first) {
918                         t = removedMetaRules.get(d).toString();
919                         first = false;
920                     } else {
921                         t = t.concat("," + removedMetaRules.get(d));
922                     }
923                 }
924                 temp[0] = t;
925                 temp[1] = String.valueOf(kWh);
926                 temp[2] = String.valueOf(dTotalError);
927                 recursionResults.add(temp);
928             } else {// now check if new result is better than the previous and replace it
929
930                 String[] splitted = recursionResults.get(0);
931                 if (dTotalError < toDouble(splitted[2])) {
932                     String temp[] = new String[3];
933                     String t = "";
934                     boolean first = true;
935                     for (int d = 0; d < removedMetaRules.size(); d++) {
936                         if (first) {
937                             t = removedMetaRules.get(d).toString();
938                             first = false;
939                         } else {
940                             t = t.concat("," + removedMetaRules.get(d));
941                         }
942                     }
943                     temp[0] = t;
944                     temp[1] = String.valueOf(kWh);
945                     temp[2] = String.valueOf(dTotalError);
946                     // recursionResults.set(0, String.join(", ", removedMetaRules) + "|" +
```

```
947                     // String.valueOf(kWh) + "|" + String.valueOf(dTotalError));
948                     recursionResults.set(0, temp);
949                 }
950             }
951
952             recursiveIterator++;
953             if (removedMetaRules.size() == numrules) {
954                 removedMetaRules.clear();
955             }
956             if (recursiveIterator == 5) {
957                 solutionsFound = true;
958                 removedMetaRules.clear();
959                 String[] rs = recursionResults.get(0);
960                 String[] splitted = rs[0].split(",");
961                 for (int i = 0; i < splitted.length; i++) {
962                     if (splitted[0].length() == 0)
963                         break;
964                     removedMetaRules.add(Integer.valueOf(splitted[i]));
965                 }
966
967             }
968             calculateKWH3(sRecordsHourly);
969             return public_ms;
970
971         } else {
972             contiguous_ms = localContiguous_ms;
973             dtTemp = dt1;
974             EnergyPlannerData.addAll(copiediotdatakwh);
975             removedMetaRules.clear();
976             recursiveIterator = 0;
977             recursionResults.clear();
978             solutionsFound = false;
979
980             public_ms = ms;
981             return public_ms;
982
983         }
984
985     }// END OF CALCULATE KWH METHOD - number of modifications
986
987     public void energyplanner(ArrayList<String[]> data, ArrayList<String[]> MRTrulesOriginal,
988             Double formaliseMaxError) {
989         copiediotdata = new ArrayList<String[]>(data); // clone original dataset list
990         MRTrules = new ArrayList<String[]>(MRTrulesOriginal);
991         listRanges = MRTrules.toArray(new String[0][]);
```

```
992
993         try {
994             dtTemp = simpleFormatter.parse("2013-10-22 02:32:34.409"); // temp date
995
996         } catch (Exception e) {
997             System.out.println(e);
998         }
999
1000        contiguous_ms = 0;
1001        ArrayList<String[]> recordsHourly = new ArrayList<String[]>();
1002        double hourChange = 2;
1003        double total_ms = 0;
1004        int monthchange = 0;
1005        int yearchange = 1998;
1006        DecimalFormat df1 = new DecimalFormat("#.##");
1007        df1.setRoundingMode(RoundingMode.CEILING);
1008        String monthString;
1009        EnergyPlannerData = new ArrayList<String[]>();
1010        copiediotdata3 = new ArrayList<String[]>(); // clone original dataset list
1011        boolean first = true;
1012
1013        try {
1014            for (int i = 0; i < copiediotdata.size(); i++) {
1015                // System.out.println(i);
1016                String[] copiediotdataSplitHour = copiediotdata.get(i);
1017                int tempHourChange = getHour(copiediotdataSplitHour[0]);
1018                int tempmonthchange = getMonth(copiediotdataSplitHour[0]);
1019
1020                if (hourChange == tempHourChange) {
1021
1022                    if (i == copiediotdata.size() - 1) {
1023                        recordsHourly.add(copiediotdata.get(i));
1024                        double hourly_ms = calculateKWH3(recordsHourly);
1025                        total_ms = total_ms + hourly_ms;
1026                        recordsHourly.clear();
1027                    } else {
1028                        recordsHourly.add(copiediotdata.get(i));
1029                    }
1030
1031                } else {
1032                    hourChange = tempHourChange;
1033                    double hourly_ms = calculateKWH3(recordsHourly);
1034                    total_ms = total_ms + hourly_ms;
1035                    recordsHourly.clear();
1036                    recordsHourly.add(copiediotdata.get(i));
```

```
1037                }
1038
1039                // this is the calculation for the kwh per month
1040
1041                if (monthchange != tempmonthchange) {
1042                    monthchange = tempmonthchange;
1043                    // first is first month
1044                    if (first) {
1045                        double seconds1 = total_ms / 1000;
1046                        double hours1 = seconds1 / 3600;
1047                        double kWh1 = ((hours1 * 320) / 1000) * 24;
1048
1049                        kwhpermonth = kwhpermonth.concat(df1.format(kWh1));
1050                        first = false;
1051                    } else {
1052                        double seconds1 = total_ms / 1000;
1053                        double hours1 = seconds1 / 3600;
1054                        double kWh1 = ((hours1 * 320) / 1000) * 24;
1055
1056                        kwhpermonth = kwhpermonth.concat("," + df1.format(kWh1));
1057
1058                    }
1059                }
1060                // END of calculation for the kwh per month
1061            }
1062        } catch (Exception ex) {
1063            System.out.println("First A: " + errorLine + " => Exception: " + ex);
1064            ex.printStackTrace();
1065        }
1066
1067        copiediotdata.clear();
1068
1069        copiediotdata = new ArrayList<String[]>(EnergyPlannerData);
1070        EnergyPlannerData.clear();
1071
1072        DecimalFormat df = new DecimalFormat("#.##");
1073        df.setRoundingMode(RoundingMode.CEILING);
1074
1075        double seconds = total_ms / 1000;
1076        double hours = seconds / 3600;
1077        double kWh = ((hours * 320) / 1000) * 24;
1078        String label154 = "Kilowatt hour (kWh): " + df.format(kWh) + " kWh";
1079        System.out.println(label154);
1080        totalkwh = toDouble(df.format(kWh));
1081        double dTotalError = 0;
```

```
1082        double dLineError = 0;
1083        double dTotalError_kWh = 0;
1084
1085        // **********SECOND AND LAST PHASE*******************//
1086
1087        Double maxError = 0.0;
1088        Double minError = 100.0;
1089        Boolean errorFound = false;
1090        int errorsize = 0;
1091
1092        try {
1093
1094            for (int i = 0; i < copiediotdata.size(); i++) {
1095
1096                dLineError = 0;// gets the total record line error in regards of all the ranges
1097                for (int k = 0; k < listRanges.length; k++) {
1098                    errorLine = makeDataLine(copiediotdata.get(i));
1099
1100                    String[] copiediotdataSplitted2 = copiediotdata.get(i);
1101                    String[] rangesSplitted = listRanges[k];
1102
1103                    if (copiediotdataSplitted2[2].startsWith("T")) // gets T for temperatures
1104                    {
1105
1106                        // checks time and temperature and if season is empty
1107                        if (((!rangesSplitted[0].equals("SUMMER")) || (!rangesSplitted[0].equals("WINTER")))
1108                                && (rangesSplitted[3].equals("TEMPER."))) {
1109                            if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1110                                    && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1111                                    && toDouble(copiediotdataSplitted2[3]) >= toDouble(rangesSplitted[1])
1112                                    && toDouble(copiediotdataSplitted2[3]) <= toDouble(rangesSplitted[2])) {
1113                                //updateOpenhabRules("Temperature", "ON");
1114                                copiediotdata3.add(copiediotdata.get(i));
1115
1116                            } else {
1117
1118                                //updateOpenhabRules("Temperature", "OFF");
1119
1120                                // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1121
1122                                if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1123                                        && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1124                                        && toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
1125                                    dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1126                                    dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
```

```
1127
1128                                    if (dLineError > maxError) {
1129                                        maxError = dLineError;
1130                                    }
1131                                    if (dLineError < minError) {
1132                                        minError = dLineError;
1133                                    }
1134                                    errorFound = true;
1135                                } else if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1136                                        && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1137                                        && toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
1138                                    dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1139                                    dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1140
1141                                    if (dLineError > maxError) {
1142                                        maxError = dLineError;
1143                                    }
1144                                    if (dLineError < minError) {
1145                                        minError = dLineError;
1146                                    }
1147                                    errorFound = true;
1148                                }
1149
1150                            }
1151                        }
1152
1153                        if (((rangesSplitted[0].equals("SUMMER")) || (rangesSplitted[0].equals("WINTER")))
1154                                && (rangesSplitted[3].equals("TEMPER."))) {
1155                            int iSeason = getMonth(copiediotdataSplitted2[0]);
1156
1157                            // checks if SUMMER
1158                            if ((iSeason == 6) || (iSeason == 7) || (iSeason == 8)) {
1159                                if (toDouble(copiediotdataSplitted2[3]) >= toDouble(rangesSplitted[1])
1160                                        && toDouble(copiediotdataSplitted2[3]) <= toDouble(rangesSplitted[2])) {
1161                                    copiediotdata3.add(copiediotdata.get(i));
1162
1163                                } else {
1164                                    // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1165                                    if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
1166                                        dTotalError += toDouble(rangesSplitted[1])
1167                                                - toDouble(copiediotdataSplitted2[3]);
1168                                        dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1169
1170                                        if (dLineError > maxError) {
1171                                            maxError = dLineError;
```

```
                        }
                        if (dLineError < minError) {
                            minError = dLineError;
                        }
                        errorFound = true;
                    } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
                        dTotalError += toDouble(copiediotdataSplitted2[3])
                                - toDouble(rangesSplitted[2]);
                        dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);

                        if (dLineError > maxError) {
                            maxError = dLineError;
                        }
                        if (dLineError < minError) {
                            minError = dLineError;
                        }
                        errorFound = true;
                    }
                }
            }
            // checks if WINTER
            else if ((iSeason == 12) || (iSeason == 1) || (iSeason == 2)) {
                if (toDouble(copiediotdataSplitted2[3]) >= toDouble(rangesSplitted[1])
                        && toDouble(copiediotdataSplitted2[3]) <= toDouble(rangesSplitted[2])) {
                    copiediotdata3.add(copiediotdata.get(i));

                } else {
                    // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                    if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
                        dTotalError += toDouble(rangesSplitted[1])
                                - toDouble(copiediotdataSplitted2[3]);
                        dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);

                        if (dLineError > maxError) {
                            maxError = dLineError;
                        }
                        if (dLineError < minError) {
                            minError = dLineError;
                        }
                        errorFound = true;
                    } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
                        dTotalError += toDouble(copiediotdataSplitted2[3])
                                - toDouble(rangesSplitted[2]);
                        dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
```

```
                        if (dLineError > maxError) {
                            maxError = dLineError;
                        }
                        if (dLineError < minError) {
                            minError = dLineError;
                        }
                        errorFound = true;
                    }
                }
            }

        }

        if (rangesSplitted[0].equals("kWh")) {
            if (toDouble(copiediotdataSplitted2[7]) <= toDouble(rangesSplitted[1])) {
                copiediotdata3.add(copiediotdata.get(i));
            } else {
                dTotalError_kWh += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
                dTotalError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
                dLineError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);

                if (dLineError > maxError) {
                    maxError = dLineError;
                }
                if (dLineError < minError) {
                    minError = dLineError;
                }
                errorFound = true;
            }
        }

    }

    if (copiediotdataSplitted2[2].startsWith("LS")) // gets LS for light sensor
    {
        // checks LIGHT
        if (rangesSplitted[3].equals("LIGHT")) {
            if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
                    && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
                    && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
                    && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
                copiediotdata3.add(copiediotdata.get(i));
                //updateOpenhabRules("Light", "ON");

            } else {
```

```
                    //updateOpenhabRules("Light", "OFF");
                    // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                    if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
                            && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
                            && toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
                        dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
                        dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);

                        if (dLineError > maxError) {
                            maxError = dLineError;
                        }
                        if (dLineError < minError) {
                            minError = dLineError;
                        }
                        errorFound = true;
                    } else if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
                            && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
                            && toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
                        dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
                        dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);

                        if (dLineError > maxError) {
                            maxError = dLineError;
                        }
                        if (dLineError < minError) {
                            minError = dLineError;
                        }
                        errorFound = true;
                    }
                }
            } // end of: checks LIGHT

            // checks SUNNY
            if (rangesSplitted[3].equals("SUNNY")) {
                if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
                        && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
                        && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
                        && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
                    copiediotdata3.add(copiediotdata.get(i));

                } else if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
                    // assigns TOTAL ERROR AND CURRENT RECORD ERROR

                    dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
                    dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
```

```
                    if (dLineError > maxError) {
                        maxError = dLineError;
                    }
                    if (dLineError < minError) {
                        minError = dLineError;
                    }
                    errorFound = true;
                } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
                    dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
                    dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);

                    if (dLineError > maxError) {
                        maxError = dLineError;
                    }
                    if (dLineError < minError) {
                        minError = dLineError;
                    }
                    errorFound = true;
                }

            } // end of :checks SUNNY

            // checks CLOUDY
            if (rangesSplitted[3].equals("CLOUDY")) {
                if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
                        && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
                        && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
                        && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
                    copiediotdata3.add(copiediotdata.get(i));

                } else {
                    // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                    if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
                        dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
                        dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);

                        if (dLineError > maxError) {
                            maxError = dLineError;
                        }
                        if (dLineError < minError) {
                            minError = dLineError;
                        }
                        errorFound = true;
                    } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
```

```
                    dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
                    dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);

                    if (dLineError > maxError) {
                        maxError = dLineError;
                    }
                    if (dLineError < minError) {
                        minError = dLineError;
                    }
                    errorFound = true;
                }
            }
        } // end of :checks CLOUDY

        // checks DOOR OPEN-LIGHT
        if (rangesSplitted[3].equals("DOOR OPEN-LIGHT")) {
            if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
                    && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
                    && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
                    && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
                copiediotdata3.add(copiediotdata.get(i));

            } else {
                // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
                    dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
                    dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);

                    if (dLineError > maxError) {
                        maxError = dLineError;
                    }
                    if (dLineError < minError) {
                        minError = dLineError;
                    }
                    errorFound = true;
                } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
                    dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
                    dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);

                    if (dLineError > maxError) {
                        maxError = dLineError;
                    }
                    if (dLineError < minError) {
                        minError = dLineError;
                    }
                    errorFound = true;
                }
            }
        } // end of :checks DOOR OPEN-LIGHT

        if (rangesSplitted[0].equals("kWh")) {
            if (toDouble(copiediotdataSplitted2[7]) <= toDouble(rangesSplitted[1])) {
                copiediotdata3.add(copiediotdata.get(i));
            } else {
                dTotalError_kWh += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
                dTotalError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
                dLineError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);

                if (dLineError > maxError) {
                    maxError = dLineError;
                }
                if (dLineError < minError) {
                    minError = dLineError;
                }
                errorFound = true;
            }
        } // end of kWh check

    } // end of LS

}

if (errorFound == true) {
    errorsize++;
    errorFound = false;
}

    }
} catch (Exception e) {
    System.out.println("Second: " + errorLine + "\tException:\t" + e);

}

df = new DecimalFormat("#.####");
df.setRoundingMode(RoundingMode.CEILING);

dTotalError = dTotalError * 5; // 10 7 5
String label161 = "Total Error: " + df.format((dTotalError * 100) / formaliseMaxError);
String label158 = "Max Error: " + df.format((maxError * 100) / formaliseMaxError) + "%";
String label157 = "Min Error: " + df.format((minError * 100) / formaliseMaxError) + "%";
```

```
1442        // calculate Total Average Error
1443        double averageErrorTotal = toDouble(df.format((dTotalError / copiediotdata.size())));
1444        String label153 = "Total Average Error: " + df.format((averageErrorTotal * 100) / formaliseMaxError) + "%";
1445        totalaverageerror = toDouble(df.format((averageErrorTotal * 100) / formaliseMaxError));
1446
1447        // calculate Budget Average Error
1448        double averageError_Budget = toDouble(df.format((dTotalError_kWh / copiediotdata.size())));
1449        String label151 = "Budget Average Error: " + df.format((averageError_Budget * 100) / formaliseMaxError) + "%";
1450        budgeterror = toDouble(df.format((averageError_Budget * 100) / formaliseMaxError));
1451        // calculate Convenience Average Error
1452        double averageError_Concenience = toDouble(df.format(((dTotalError - dTotalError_kWh) / copiediotdata.size())));
1453        String label152 = "Convenience Average Error: "
1454                + df.format((averageError_Concenience * 100) / formaliseMaxError) + "%";
1455        convinienceerror = toDouble(df.format((averageError_Concenience * 100) / formaliseMaxError));
1456
1457        System.out.println(label161);
1458        System.out.println(label158);
1459        System.out.println(label157);
1460        System.out.println(label153);
1461        System.out.println(label151);
1462        System.out.println(label152);
1463
1464        copiediotdata3.clear();
1465
1466        // ************SECOND RUN*******************************
1467        Double meansSum = 0.0;
1468        maxError = 0.0;
1469        minError = 100.0;
1470        dTotalError = 0;
1471        dLineError = 0;
1472
1473        try {
1474
1475            for (int i = 0; i < copiediotdata.size(); i++) {
1476
1477                dLineError = 0;// gets the total record line error in regards of all the ranges
1478                for (int k = 0; k < listRanges.length; k++) {
1479                    errorLine = makeDataLine(copiediotdata.get(i));
1480
1481                    String[] copiediotdataSplitted2 = copiediotdata.get(i);
1482                    String[] rangesSplitted = listRanges[k];
1483
1484                    if (copiediotdataSplitted2[2].startsWith("T")) // gets T for temperatures
1485                    {
1486                        // checks time and temperature and if season is empty
1487                        if ((!rangesSplitted[0].equals("SUMMER")) || (!rangesSplitted[0].equals("WINTER")))
1488                                && (rangesSplitted[3].equals("TEMPER."))) {
1489                            if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1490                                    && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1491                                    && toDouble(copiediotdataSplitted2[3]) >= toDouble(rangesSplitted[1])
1492                                    && toDouble(copiediotdataSplitted2[3]) <= toDouble(rangesSplitted[2])) {
1493                                copiediotdata3.add(copiediotdata.get(i));
1494
1495                            } else {
1496                                // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1497                                if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1498                                        && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1499                                        && toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
1500                                    dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1501                                    dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1502
1503                                    if (dLineError > maxError) {
1504                                        maxError = dLineError;
1505                                    }
1506                                    if (dLineError < minError) {
1507                                        minError = dLineError;
1508                                    }
1509                                } else if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1510                                        && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1511                                        && toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
1512                                    dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1513                                    dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1514
1515                                    if (dLineError > maxError) {
1516                                        maxError = dLineError;
1517                                    }
1518                                    if (dLineError < minError) {
1519                                        minError = dLineError;
1520                                    }
1521                                }
1522
1523                            }
1524                        }
1525
1526                        if ((rangesSplitted[0].equals("SUMMER")) || (rangesSplitted[0].equals("WINTER")))
1527                                && (rangesSplitted[3].equals("TEMPER."))) {
1528                            int iSeason = getMonth(copiediotdataSplitted2[0]);
1529
1530                            // checks if SUMMER
1531                            if ((iSeason == 6) || (iSeason == 7) || (iSeason == 8)) {
```

```
                    if (toDouble(copiediotdataSplitted2[3]) >= toDouble(rangesSplitted[1])
                            && toDouble(copiediotdataSplitted2[3]) <= toDouble(rangesSplitted[2])) {
                        copiediotdata3.add(copiediotdata.get(i));

                    } else {
                        // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                        if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
                            dTotalError += toDouble(rangesSplitted[1])
                                    - toDouble(copiediotdataSplitted2[3]);
                            dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);

                            if (dLineError > maxError) {
                                maxError = dLineError;
                            }
                            if (dLineError < minError) {
                                minError = dLineError;
                            }
                        } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
                            dTotalError += toDouble(copiediotdataSplitted2[3])
                                    - toDouble(rangesSplitted[2]);
                            dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);

                            if (dLineError > maxError) {
                                maxError = dLineError;
                            }
                            if (dLineError < minError) {
                                minError = dLineError;
                            }
                        }
                    }
                }
                // checks if WINTER
                else if ((iSeason == 12) || (iSeason == 1) || (iSeason == 2)) {
                    if (toDouble(copiediotdataSplitted2[3]) >= toDouble(rangesSplitted[1])
                            && toDouble(copiediotdataSplitted2[3]) <= toDouble(rangesSplitted[2])) {
                        copiediotdata3.add(copiediotdata.get(i));

                    } else {
                        // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                        if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
                            dTotalError += toDouble(rangesSplitted[1])
                                    - toDouble(copiediotdataSplitted2[3]);
                            dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);

                            if (dLineError > maxError) {
                                maxError = dLineError;
                            }
                            if (dLineError < minError) {
                                minError = dLineError;
                            }
                        } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
                            dTotalError += toDouble(copiediotdataSplitted2[3])
                                    - toDouble(rangesSplitted[2]);
                            dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);

                            if (dLineError > maxError) {
                                maxError = dLineError;
                            }
                            if (dLineError < minError) {
                                minError = dLineError;
                            }
                        }
                    }
                }

            }

            if (rangesSplitted[0].equals("kWh")) {
                if (toDouble(copiediotdataSplitted2[7]) <= toDouble(rangesSplitted[1])) {
                    copiediotdata3.add(copiediotdata.get(i));
                } else {
                    dTotalError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
                    dLineError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
                    //

                    if (dLineError > maxError) {
                        maxError = dLineError;
                    }
                    if (dLineError < minError) {
                        minError = dLineError;
                    }
                    errorFound = true;
                }
            }

        }

        if (copiediotdataSplitted2[2].startsWith("LS")) // gets LS for light sensor
        {
            // checks LIGHT
```

```
1622                    if (rangesSplitted[3].equals("LIGHT")) {
1623                        if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1624                                && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1625                                && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
1626                                && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
1627                            copiediotdata3.add(copiediotdata.get(i));
1628
1629                        } else {
1630                            // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1631                            if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1632                                    && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1633                                    && toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
1634                                dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1635                                dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1636
1637                                if (dLineError > maxError) {
1638                                    maxError = dLineError;
1639                                }
1640                                if (dLineError < minError) {
1641                                    minError = dLineError;
1642                                }
1643                            } else if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1644                                    && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1645                                    && toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
1646                                dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1647                                dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1648
1649                                if (dLineError > maxError) {
1650                                    maxError = dLineError;
1651                                }
1652                                if (dLineError < minError) {
1653                                    minError = dLineError;
1654                                }
1655                            }
1656                        }
1657                    } // end of: checks LIGHT
1658
1659                    // checks SUNNY
1660                    if (rangesSplitted[3].equals("SUNNY")) {
1661                        if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1662                                && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1663                                && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
1664                                && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
1665                            copiediotdata3.add(copiediotdata.get(i));
1666
```

```
1667                        } else {
1668                            // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1669                            if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
1670                                dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1671                                dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1672
1673                                if (dLineError > maxError) {
1674                                    maxError = dLineError;
1675                                }
1676                                if (dLineError < minError) {
1677                                    minError = dLineError;
1678                                }
1679                            } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
1680                                dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1681                                dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1682
1683                                if (dLineError > maxError) {
1684                                    maxError = dLineError;
1685                                }
1686                                if (dLineError < minError) {
1687                                    minError = dLineError;
1688                                }
1689                            }
1690                        }
1691                    } // end of :checks SUNNY
1692
1693                    // checks CLOUDY
1694                    if (rangesSplitted[3].equals("CLOUDY")) {
1695                        if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1696                                && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1697                                && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
1698                                && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
1699                            copiediotdata3.add(copiediotdata.get(i));
1700
1701                        } else {
1702                            // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1703                            if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
1704                                dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1705                                dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1706
1707                                if (dLineError > maxError) {
1708                                    maxError = dLineError;
1709                                }
1710                                if (dLineError < minError) {
1711                                    minError = dLineError;
```

```
                                    }
                                } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
                                    dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
                                    dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);

                                    if (dLineError > maxError) {
                                        maxError = dLineError;
                                    }
                                    if (dLineError < minError) {
                                        minError = dLineError;
                                    }
                                }
                            }
                        } // end of :checks CLOUDY

                        // checks DOOR OPEN-LIGHT
                        if (rangesSplitted[3].equals("DOOR OPEN-LIGHT")) {
                            if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
                                    && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
                                    && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
                                    && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
                                copiediotdata3.add(copiediotdata.get(i));

                            } else {
                                // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                                if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
                                    dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
                                    dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);

                                    if (dLineError > maxError) {
                                        maxError = dLineError;
                                    }
                                    if (dLineError < minError) {
                                        minError = dLineError;
                                    }
                                } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
                                    dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
                                    dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);

                                    if (dLineError > maxError) {
                                        maxError = dLineError;
                                    }
                                    if (dLineError < minError) {
                                        minError = dLineError;
                                    }
```

```
                                    }
                                }
                            } // end of :checks DOOR OPEN-LIGHT

                            if (rangesSplitted[0].equals("kWh")) {
                                if (toDouble(copiediotdataSplitted2[7]) <= toDouble(rangesSplitted[1])) {
                                    copiediotdata3.add(copiediotdata.get(i));
                                } else {
                                    dTotalError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
                                    dLineError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);

                                    if (dLineError > maxError) {
                                        maxError = dLineError;
                                    }
                                    if (dLineError < minError) {
                                        minError = dLineError;
                                    }
                                    errorFound = true;
                                }
                            } // end of kWh check

                        } // end of LS

                    }

                    meansSum = meansSum + Math.pow(dLineError - averageErrorTotal, 2);

                }
            } catch (Exception e) {
                System.out.println("Second: " + errorLine + "\tException:\t" + e);

            }

            df = new DecimalFormat("#.####");
            df.setRoundingMode(RoundingMode.CEILING);

            double standardDeviation = Math.sqrt(meansSum / copiediotdata.size());
            String label156 = "Standard Deviation: " + df.format((standardDeviation * 100) / formaliseMaxError) + "%";
            deviation = toDouble(df.format((standardDeviation * 100) / formaliseMaxError));
            System.out.println(label156);
            copiediotdata3.clear();

            postData();
        }// END OF ENERGY PLANNER - number of modifications
```

GreenPlanner.java

```java
/*
    This file implements the Green Planner algorithm.
    Copyright (C) 2021  Antonis Vasileiou
    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.
    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.
    You should have received a copy of the GNU General Public License
    along with this program.  If not, see https://www.gnu.org/licenses/gpl-3.0.html.
*/
package iotprojectpackage;

import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.FileWriter;
import java.io.InputStreamReader;
import java.math.RoundingMode;
import java.net.HttpURLConnection;
import java.net.URL;
import java.nio.charset.StandardCharsets;
import java.text.DecimalFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.ZoneId;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.Random;

import com.google.gson.Gson;
import com.google.gson.JsonArray;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;

public class GreenPlanner {

    private String errorLine;// global

    private SimpleDateFormat simpleFormatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    private SimpleDateFormat getHour = new SimpleDateFormat("HH");
    private Date dtTemp;
```

```java
    private double contiguous_ms = 0;
    private String monthlyKWH_listAssignment = "original"; // to help identify allowed kwh ArrayList->original,
                                                           // 1%,5%,10% or 20%
    private ArrayList<String[]> copiediotdata;
    private ArrayList<String[]> GreenPlannerData;
    private ArrayList<String[]> copiediotdata3;
    private ArrayList<String[]> MRTrules;
    private boolean solutionsFound;
    private ArrayList<String[]> recursionResults = new ArrayList<String[]>();
    private double public_ms = 0;
    private int recursiveIterator = 0;
    private String urlString = "http://10.16.30.73/imcfgp/api/update-results";
    private double totalaverageerror = 0.0;
    private double budgeterror = 0.0;
    private double convinienceerror = 0.0;
    private double deviation = 0.0;
    private double totalkwh = 0.0;
    private double total_daily_ms_CO2 = 0;
    private double kwh_with_CO2 = 0;
    private double kwh_without_CO2 = 0;
    private double CO2_kg_emissions = 0;

    private String listRanges[][];
    private String kwhpermonth = "";

    public GreenPlanner() {
        solutionsFound = false;
    }

    public void ArrayArrayList_fromString_toStringArray(ArrayList<String> ArrayList_from,
            ArrayList<String[]> ArrayList_to) {

        for (int i = 0; i < ArrayList_from.size(); i++) {
            String temp[] = ArrayList_from.get(i).split("\\|");
            ArrayList_to.add(temp);
        }

    }

    public String makeDataLine(String[] array) {
        String s = "";
        String seperator = "|";
        int count = 0;
        int length = array.length;

```

```
        for (String temp : array) {
            if (count == length - 1) {
                s = s.concat(temp);
            } else {
                s = s.concat(temp);
                s = s.concat(seperator);
            }
            count++;
        }

        return s;
    }

    public static String[] addX(String myarray[], String ele) {

        int n = myarray.length;
        String newArray[] = new String[n + 1];
        // copy original array into new array
        for (int i = 0; i < n; i++)
            newArray[i] = myarray[i];

        // add element to the new array
        newArray[n] = ele;

        return newArray;
    }

    // converts string to double
    public double toDouble(String str) {

        return Double.parseDouble(str);

    }

    // converts string to double
    public int toInt(String str) {

        return Integer.parseInt(str);

    }


    // returns hour from record datetime
    public int getHour(String date) {

        return toInt(date.substring(11, 13));

    }


    // returns month from record datetime
    public int getMonth(String date) {

        return toInt(date.substring(5, 7));

    }

    // returns month from record datetime
    public int getDay(String date) {

        return toInt(date.substring(8, 10));

    }

    // returns the hour the rule starts
    public int ruleHourFrom(String ruletime) {

        return toInt(ruletime.substring(0, 2));

    }

    // returns the hour the rule ends
    public int ruleHourTo(String ruletime) {

        return toInt(ruletime.substring(6, 8));

    }


    public void postData() {
        try {

            String urlParameters = "gp_kWh=" + totalkwh + "&gp_totalError=" + totalaverageerror
                    + "&gp_convenienceError=" + convinienceerror + "&gp_budgetError=" + budgeterror
                    + "&gp_standardDeviation=" + deviation + "&gp_kwh_per_month=" + kwhpermonth + "&gp_kwh_with_co2="
                    + kwh_with_CO2 + "&gp_kwh_without_co2=" + kwh_without_CO2 + "&gp_co2_emissions_kg="
                    + CO2_kg_emissions;
            byte[] postData = urlParameters.getBytes(StandardCharsets.UTF_8);
            int postDataLength = postData.length;
```

```
            URL url = new URL(urlString);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();

            conn.setDoOutput(true);
            conn.setRequestMethod("POST");
            conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
            conn.setRequestProperty("Charset", "UTF-8");
            conn.setRequestProperty("Content-Length", Integer.toString(postDataLength));
            conn.setUseCaches(false);

            conn.connect();

            DataOutputStream wr = new DataOutputStream(conn.getOutputStream());
            wr.write(postData);
            conn.getResponseCode();

        } catch (Exception e) {
            for (StackTraceElement s : e.getStackTrace())
                System.out.println(s);
        }
    }

    // START OF CALCULATE KWH METHOD - number of modifications (GREEN PLANNER)
    public double calculateKWH5(ArrayList<String[]> sRecordsDaily) {

        ArrayList<String[]> copiediotdatakwh = new ArrayList<String[]>(sRecordsDaily);
        ArrayList<Integer> removedMetaRules = new ArrayList<Integer>();

        Random rand = new Random();
        boolean startALLzeros = false;
        boolean startALLrandom = false;
        int numofloops = 3;
        boolean randomRule[] = new boolean[listRanges.length];
        int numrules = listRanges.length;
        int random;
        if (removedMetaRules.size() == 0) {
            if (startALLzeros == true) {
                for (int d = 0; d < randomRule.length; d++) {
                    randomRule[d] = false;
                    removedMetaRules.add(d);
                }
            } else if (startALLrandom == true) {
                for (int d = 0; d < randomRule.length; d++) {
                    // generates random number from 0 to 1
                    random = rand.nextInt(2);
```

```
                    if (random == 0) {
                        randomRule[d] = false;
                        removedMetaRules.add(d);
                    } else {
                        randomRule[d] = true;
                    }

                }

            } else {
                for (int d = 0; d < randomRule.length; d++) {
                    randomRule[d] = true;
                }
            }

            // for predifined times add or remove rules
            for (int i = 0; i < numofloops; i++) {
                random = rand.nextInt(numrules);

                if (randomRule[random]) {
                    randomRule[random] = false;
                    removedMetaRules.add(random);
                } else {
                    randomRule[random] = true;
                    removedMetaRules.remove(Integer.valueOf(random));
                }

            }
        }

        if (solutionsFound == false) {

            for (int i = 0; i < numofloops; i++) {
                random = rand.nextInt(numrules);

                if (randomRule[random]) {
                    randomRule[random] = false;
                    removedMetaRules.add(random);
                } else {
                    randomRule[random] = true;
                    removedMetaRules.remove(Integer.valueOf(random));
                }

            }
        } else if (solutionsFound == true) {
```

```
            for (int d = 0; d < randomRule.length; d++) {
                randomRule[d] = true;
            }

            for (int i = 0; i < removedMetaRules.size(); i++) {
                randomRule[removedMetaRules.get(i)] = false;

            }
        }

        int IFTTTconstraintsApplied = 0;
        double milliWatts = 0;
        Date dt1 = dtTemp;
        Date dt2;
        double span;
        double ms = 0;
        double localContiguous_ms = contiguous_ms;
        double daily_ms_CO2 = 0;
        ArrayList<Double> monthlyKWH = new ArrayList<Double>();
        Calendar cdt1 = GregorianCalendar.getInstance(); // creates a new calendar instance
        Calendar cdt2 = GregorianCalendar.getInstance();


        if (monthlyKWH_listAssignment.equals("original")) {
            monthlyKWH = new ArrayList<Double>();
            // [] d= new []{ 25.02, 17.06, 7.96, 4.55, 5.69, 6.82, 7.96, 10.23,
            // 6.82, 5.69, 6.82, 13.65 };
            double[] d = new double[] { 1091.81, 744.42, 347.39, 198.51, 248.14, 297.77, 347.39, 446.65, 297.77, 248.14, 297.77, 595.53 }
            for (int i = 0; i < d.length; i++)
                monthlyKWH.add(d[i]);
        } else if (monthlyKWH_listAssignment.equals("1")) {
            monthlyKWH = new ArrayList<>();
            double[] d = new double[] { 25.02, 17.06, 7.96, 4.55, 5.69, 6.82, 7.96, 10.23, 6.82, 5.69, 6.82, 13.65 };
            for (int i = 0; i < d.length; i++)
                monthlyKWH.add(d[i]);
        } else if (monthlyKWH_listAssignment.equals("5")) {
            monthlyKWH = new ArrayList<>();

            double[] d = new double[] { 23.77, 16.21, 7.56, 4.32, 5.40, 6.48, 7.56, 9.72, 6.48, 5.40, 6.48, 12.96 };
            for (int i = 0; i < d.length; i++)
                monthlyKWH.add(d[i]);
        } else if (monthlyKWH_listAssignment.equals("10")) {
            monthlyKWH = new ArrayList<>();

            double[] d = new double[] { 22.52, 15.35, 7.17, 4.09, 5.12, 6.14, 7.17, 9.21, 6.14, 5.12, 6.14, 12.28 };
            for (int i = 0; i < d.length; i++)
                monthlyKWH.add(d[i]);
        } else if (monthlyKWH_listAssignment.equals("20")) {
            monthlyKWH = new ArrayList<>();

            double[] d = new double[] { 20.01, 13.65, 6.37, 3.64, 4.55, 5.46, 6.37, 8.19, 5.46, 4.55, 5.46, 10.92 };
            for (int i = 0; i < d.length; i++)
                monthlyKWH.add(d[i]);
        } else if (monthlyKWH_listAssignment.equals("30")) {
            monthlyKWH = new ArrayList<>();
            double[] d = new double[] { 17.51, 11.94, 5.57, 3.18, 3.98, 4.78, 5.57, 7.16, 4.78, 3.98, 4.78, 9.55 };
            for (int i = 0; i < d.length; i++)
                monthlyKWH.add(d[i]);
        } else if (monthlyKWH_listAssignment.equals("40")) {
            monthlyKWH = new ArrayList<>();
            double[] d = new double[] { 15.01, 10.24, 4.78, 2.73, 3.41, 4.09, 4.78, 6.14, 4.09, 3.41, 4.09, 8.19 };
            for (int i = 0; i < d.length; i++)
                monthlyKWH.add(d[i]);
        }

        try {

            ArrayList<String> listRules = new ArrayList<String>();

            for (int d = 0; d < randomRule.length; d++) {
                if (randomRule[d]) {
                    // temp
                    if (listRanges[d][3].equals("TEMPER.")) {
                        listRules.add(listRanges[d][0] + "|" + "+" + listRanges[d][1] + "|" + "---" + "|"
                                + listRanges[d][1] + "|" + "---" + "|" + "---" + "|" + "---" + "|" + "SET TEMPERATURE");
                    }
                    // light
                    if (listRanges[d][3].equals("LIGHT")) {
                        listRules.add(listRanges[d][0] + "|" + "---" + "|" + "+" + listRanges[d][1] + "|"
                                + listRanges[d][1] + "|" + "---" + "|" + "---" + "|" + "---" + "|" + "SET LIGHT");
                    }
                    IFTTTconstraintsApplied++;

                }
            }

            for (int i = 0; i < copiediotdatakwh.size(); i++) {

                for (int k = 0; k < listRules.size(); k++) {
                    errorLine = String.join("|", copiediotdatakwh.get(i));
```

```
362
363                     String[] copiediotdataSplitted = copiediotdatakwh.get(i);
364                     String[] rulesSplitted = listRules.get(k).split("\\|");
365
366                     if (copiediotdataSplitted[2].startsWith("T")) // gets T for temperatures
367                     {
368                         // checks time and temperature and if season,wether,door is empty
369                         if (!rulesSplitted[1].equals("---") && rulesSplitted[4].equals("---")
370                                 && rulesSplitted[5].equals("---") && rulesSplitted[6].equals("---")) {
371
372                             if (getHour(copiediotdataSplitted[0]) >=
373                                     ruleHourFrom(rulesSplitted[0])
374                                     && getHour(copiediotdataSplitted[0]) <=
375                                             ruleHourTo(rulesSplitted[0])) {
376                                 dt2 = simpleFormatter.parse(copiediotdataSplitted[0].substring(0, 18));
377                                 span = Math.abs(dt2.getTime() - dt1.getTime());
378                                 ms += span;
379                                 dt1 = dt2;
380                                 cdt2 = GregorianCalendar.getInstance(); // creates a new calendar instance
381                                 cdt2.setTime(dt2);   // assigns calendar to given date
382
383                                 if ( cdt2.get(Calendar.HOUR) >= 8 && cdt2.get(Calendar.HOUR) <= 15
384                                         && copiediotdataSplitted[4].equals("sky is clear")) {
385                                     daily_ms_CO2 += span;
386                                 }
387
388                                 localContiguous_ms += span;
389                                 double secondsTemp = localContiguous_ms / 1000;
390                                 double hoursTemp = secondsTemp / 3600;
391                                 double kWhTemp = ((hoursTemp * 320) / 1000) * 1100;
392                                 copiediotdataSplitted = addX(copiediotdataSplitted, String.valueOf(kWhTemp));
393
394                                 milliWatts = milliWatts + 320;
395                                 copiediotdataSplitted[3] = rulesSplitted[3];
396                                 copiediotdatakwh.set(i, copiediotdataSplitted);
397                                 break; // its already on the ArrayList no need to add it twice
398
399                             }
400
401                         }
402
403                     }
404
405                     if (copiediotdataSplitted[2].startsWith("LS")) // gets LS for light sensor
406                     {
407                         // checks LIGHT , and if weather,door is empty
408                         if (!rulesSplitted[2].equals("---") && rulesSplitted[5].equals("---")
409                                 && rulesSplitted[6].equals("---")) {
410                             // checks time and light value
411                             if (getHour(copiediotdataSplitted[0]) >=
412                                     ruleHourFrom(rulesSplitted[0])
413                                     && getHour(copiediotdataSplitted[0]) <=
414                                             ruleHourTo(rulesSplitted[0])) {
415                                 dt2 = simpleFormatter.parse(copiediotdataSplitted[0].substring(0, 18));
416                                 span = Math.abs(dt2.getTime() - dt1.getTime());
417                                 ms += span;
418                                 dt1 = dt2;
419                                 cdt2 = GregorianCalendar.getInstance(); // creates a new calendar instance
420                                 cdt2.setTime(dt2);   // assigns calendar to given date
421                                 if (cdt2.get(Calendar.HOUR) >= 8 && cdt2.get(Calendar.HOUR) <= 15
422                                         && copiediotdataSplitted[4].equals("sky is clear")) {
423                                     daily_ms_CO2 += span;
424                                 }
425
426                                 localContiguous_ms += span;
427                                 double secondsTemp = localContiguous_ms / 1000;
428                                 double hoursTemp = secondsTemp / 3600;
429                                 double kWhTemp = ((hoursTemp * 320) / 1000) * 1100;
430                                 copiediotdataSplitted = addX(copiediotdataSplitted, String.valueOf(kWhTemp));
431
432                                 copiediotdataSplitted[3] = rulesSplitted[3];
433                                 // copiediotdata2.add(String.Join("|", copiediotdataSplitted));
434                                 copiediotdatakwh.set(i, copiediotdataSplitted);
435                                 break;
436                             }
437
438                         } // end of : //checks LIGHT and if weather is empty
439
440                     }
441
442                     if (k == listRules.size() - 1) {
443                         dt1 = simpleFormatter.parse(copiediotdataSplitted[0].substring(0, 18));
444                     }
445
446                 }
447
448                 String[] copiediotdataSplittedAfterIFTTT = copiediotdatakwh.get(i);
449                 if (copiediotdataSplittedAfterIFTTT.length < 8) {
450                     copiediotdatakwh.set(i, addX(copiediotdataSplittedAfterIFTTT, "00"));
451
```

```
452                 }
453             }
454         } catch (Exception ex) {
455             System.out.println("First B: " + errorLine + " => Exception: " + ex);
456         }
457 
458         // calculations for TIME(hours), watts and kWh
459         double seconds = ms / 1000;
460         double hours = seconds / 3600;
461 
462         double kWh = ((hours * 320) / 1000) * 1100;
463 
464         double seconds_CO2 = daily_ms_CO2 / 1000;
465         double hours_CO2 = seconds_CO2 / 3600;
466         double kWh_CO2 = ((hours_CO2 * 320) / 1000) * 1100;
467 
468         String[] copiediotdataSplitted_Month = copiediotdatakwh.get(0);
469         Date Month;
470         Calendar calendar;
471         int currentMonth = 0;
472         try {
473             Month = simpleFormatter.parse(copiediotdataSplitted_Month[0].substring(0, 18));
474             calendar = GregorianCalendar.getInstance(); // creates a new calendar instance
475             calendar.setTime(Month);   // assigns calendar to given date
476             currentMonth = calendar.get(Calendar.MONTH);
477         } catch (ParseException e1) {
478             e1.printStackTrace();
479         }
480         // months from calendar are indexed from 0 therefore currentMonth=0 is January
481         double allowedMonthly_kWh = monthlyKWH.get(currentMonth) * 6.5;// 6.5, *7.9;
482 
483         // **********SECOND AND LAST
484         // PHASE*****************************************************************************//
485         double dTotalError = 0;
486         double dLineError = 0;
487         double dTotalError_kWh = 0;
488         double maxError = 0;
489         double minError = 100;
490         boolean errorFound = false;
491         int errorsize = 0;
492 
493         try {
494 
495             for (int i = 0; i < copiediotdatakwh.size(); i++) {
496 
```

```
497                 dLineError = 0;// gets the total record line error in regards of all the ranges
498                 for (int k = 0; k < listRanges.length; k++) {
499                     errorLine = String.join("|", copiediotdatakwh.get(i));
500 
501                     String[] copiediotdataSplitted2 = copiediotdatakwh.get(i);
502                     String[] rangesSplitted = listRanges[k];
503 
504                     if (copiediotdataSplitted2[2].startsWith("T")) // gets T for temperatures
505                     {
506 
507                         // checks time and temperature and if season is empty
508                         if ((!rangesSplitted[0].equals("SUMMER") || !rangesSplitted[0].equals("WINTER"))
509                                 && rangesSplitted[3].equals("TEMPER.")) {
510                             if ((getHour(copiediotdataSplitted2[0]) >=
511                                     ruleHourFrom(rangesSplitted[0]))
512                                     && getHour(copiediotdataSplitted2[0]) <=
513                                             ruleHourTo(rangesSplitted[0])
514                                     && toDouble(copiediotdataSplitted2[3]) >=
515                                             toDouble(rangesSplitted[1])
516                                     && toDouble(copiediotdataSplitted2[3]) <=
517                                             toDouble(rangesSplitted[2])) {
518                                 copiediotdata3.add(copiediotdatakwh.get(i));
519                                 // its already on the ArrayList no need to add it twice
520                             } else {
521                                 // assigns TOTAL ERROR AND CURRENT RECORD ERROR
522                                 if ((getHour(copiediotdataSplitted2[0]) >=
523                                         ruleHourFrom(rangesSplitted[0]))
524                                         && getHour(copiediotdataSplitted2[0]) <=
525                                                 ruleHourTo(rangesSplitted[0])
526                                         && toDouble(copiediotdataSplitted2[3]) <
527                                                 toDouble(rangesSplitted[1])) {
528                                     dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
529                                             - toDouble(copiediotdataSplitted2[3])) / 1);
530                                     dLineError = dLineError + ((toDouble(rangesSplitted[1])
531                                             - toDouble(copiediotdataSplitted2[3])) / 1);
532 
533                                     if (dLineError > maxError) {
534                                         maxError = dLineError;
535                                     }
536                                     if (dLineError < minError) {
537                                         minError = dLineError;
538                                     }
539                                     errorFound = true;
540                                 } else if (getHour(copiediotdataSplitted2[0]) >=
541                                         ruleHourFrom(rangesSplitted[0])
```

```
                                && getHour(copiediotdataSplitted2[0]) <=
                                        ruleHourTo(rangesSplitted[0])
                                && toDouble(copiediotdataSplitted2[3]) >
                                        toDouble(rangesSplitted[2]) ) {
                            dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
                                    - toDouble(rangesSplitted[2])) / 1);
                            dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
                                    - toDouble(rangesSplitted[2])) / 1);

                            if (dLineError > maxError) {
                                maxError = dLineError;
                            }
                            if (dLineError < minError) {
                                minError = dLineError;
                            }
                            errorFound = true;
                        }

                    }
                }

                if ((rangesSplitted[0].equals("SUMMER") || rangesSplitted[0].equals("WINTER"))
                        && rangesSplitted[3].equals("TEMPER.")) {
                    int iSeason = getMonth(copiediotdataSplitted2[0]);

                    // checks if SUMMER
                    if ((iSeason == 6) || (iSeason == 7) || (iSeason == 8)) {
                        if (toDouble(copiediotdataSplitted2[3]) >=
                                toDouble(rangesSplitted[1])
                                && toDouble(copiediotdataSplitted2[3]) <=
                                        toDouble(rangesSplitted[2])) {
                            copiediotdata3.add(copiediotdatakwh.get(i));
                            // its already on the ArrayList no need to add it twice
                        } else {
                            // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                            if (toDouble(copiediotdataSplitted2[3]) <
                                    toDouble(rangesSplitted[1])) {
                                dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
                                        - toDouble(copiediotdataSplitted2[3])) / 1);
                                dLineError = dLineError + ((toDouble(rangesSplitted[1])
                                        - toDouble(copiediotdataSplitted2[3])) / 1);

                                if (dLineError > maxError) {
                                    maxError = dLineError;
                                }
```

```
                                if (dLineError < minError) {
                                    minError = dLineError;
                                }
                                errorFound = true;
                            } else if (toDouble(copiediotdataSplitted2[3]) >
                                    toDouble(rangesSplitted[2])) {
                                dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
                                        - toDouble(rangesSplitted[2])) / 1);
                                dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
                                        - toDouble(rangesSplitted[2])) / 1);

                                if (dLineError > maxError) {
                                    maxError = dLineError;
                                }
                                if (dLineError < minError) {
                                    minError = dLineError;
                                }
                                errorFound = true;
                            }
                        }
                    }
                    // checks if WINTER
                    else if ((iSeason == 12) || (iSeason == 1) || (iSeason == 2)) {
                        if (toDouble(copiediotdataSplitted2[3]) >=
                                toDouble(rangesSplitted[1])
                                && toDouble(copiediotdataSplitted2[3]) <=
                                        toDouble(rangesSplitted[2])) {
                            copiediotdata3.add(copiediotdatakwh.get(i));
                            // its already on the ArrayList no need to add it twice
                        } else {
                            // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                            if (toDouble(copiediotdataSplitted2[3]) <
                                    toDouble(rangesSplitted[1])) {
                                dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
                                        - toDouble(copiediotdataSplitted2[3])) / 1);
                                dLineError = dLineError + ((toDouble(rangesSplitted[1])
                                        - toDouble(copiediotdataSplitted2[3])) / 1);

                                if (dLineError > maxError) {
                                    maxError = dLineError;
                                }
                                if (dLineError < minError) {
                                    minError = dLineError;
                                }
                                errorFound = true;
```

```
                                } else if (toDouble(copiediotdataSplitted2[3]) >
                                        toDouble(rangesSplitted[2])) {
                                    dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
                                            - toDouble(rangesSplitted[2])) / 1);
                                    dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
                                            - toDouble(rangesSplitted[2])) / 1);

                                    if (dLineError > maxError) {
                                        maxError = dLineError;
                                    }
                                    if (dLineError < minError) {
                                        minError = dLineError;
                                    }
                                    errorFound = true;
                                }
                            }
                        }

                    }

                    if (rangesSplitted[0].equals("kWh")) {
                        if (toDouble(copiediotdataSplitted2[7]) <=
                                toDouble(rangesSplitted[1])) {
                            copiediotdata3.add(copiediotdatakwh.get(i));
                        } else {
                            dTotalError_kWh = dTotalError_kWh + ((toDouble(copiediotdataSplitted2[7])
                                    - toDouble(rangesSplitted[2])) / 1);
                            dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[7])
                                    - toDouble(rangesSplitted[2])) / 1);
                            dLineError = dLineError + ((toDouble(copiediotdataSplitted2[7])
                                    - toDouble(rangesSplitted[2])) / 1);

                            if (dLineError > maxError) {
                                maxError = dLineError;
                            }
                            if (dLineError < minError) {
                                minError = dLineError;
                            }
                            errorFound = true;
                        }
                    }

                }

                if (copiediotdataSplitted2[2].startsWith("LS")) // gets LS for light sensor
```

```
                {
                    // checks LIGHT
                    if (rangesSplitted[3].equals("LIGHT")) {
                        if (getHour(copiediotdataSplitted2[0]) >=
                                ruleHourFrom(rangesSplitted[0])
                                && getHour(copiediotdataSplitted2[0]) <=
                                        ruleHourTo(rangesSplitted[0])
                                && toInt(copiediotdataSplitted2[3]) >=
                                        toInt(rangesSplitted[1])
                                && toInt(copiediotdataSplitted2[3]) <=
                                        toInt(rangesSplitted[2])) {
                            copiediotdata3.add(copiediotdatakwh.get(i));
                            // its already on the ArrayList no need to add it twice
                        } else {
                            // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                            if (getHour(copiediotdataSplitted2[0]) >=
                                    ruleHourFrom(rangesSplitted[0])
                                    && getHour(copiediotdataSplitted2[0]) <=
                                            ruleHourTo(rangesSplitted[0])
                                    && toDouble(copiediotdataSplitted2[3]) <
                                            toDouble(rangesSplitted[1])) {
                                dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
                                        - toDouble(copiediotdataSplitted2[3])) / 1);
                                dLineError = dLineError + ((toDouble(rangesSplitted[1])
                                        - toDouble(copiediotdataSplitted2[3])) / 1);

                                if (dLineError > maxError) {
                                    maxError = dLineError;
                                }
                                if (dLineError < minError) {
                                    minError = dLineError;
                                }
                                errorFound = true;
                            } else if (getHour(copiediotdataSplitted2[0]) >=
                                    ruleHourFrom(rangesSplitted[0])
                                    && getHour(copiediotdataSplitted2[0]) <=
                                            ruleHourTo(rangesSplitted[0])
                                    && toDouble(copiediotdataSplitted2[3]) >
                                            toDouble(rangesSplitted[2])) {
                                dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
                                        - toDouble(rangesSplitted[2])) / 1);
                                dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
                                        - toDouble(rangesSplitted[2])) / 1);

                                if (dLineError > maxError) {
```

```
                                maxError = dLineError;
                            }
                            if (dLineError < minError) {
                                minError = dLineError;
                            }
                            errorFound = true;
                        }
                    }
                } // end of: checks LIGHT

                // checks SUNNY
                if (rangesSplitted[3].equals("SUNNY")) {
                    if (getHour(copiediotdataSplitted2[0]) >=
                            ruleHourFrom(rangesSplitted[0])
                            && getHour(copiediotdataSplitted2[0]) <=
                                    ruleHourTo(rangesSplitted[0])
                            && toInt(copiediotdataSplitted2[3]) >=
                                    toInt(rangesSplitted[1])
                            && toInt(copiediotdataSplitted2[3]) <=
                                    toInt(rangesSplitted[2])) {
                        copiediotdata3.add(copiediotdatakwh.get(i));
                        // its already on the ArrayList no need to add it twice
                    } else {
                        // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                        if (toDouble(copiediotdataSplitted2[3]) <
                                toDouble(rangesSplitted[1])) {
                            dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
                                    - toDouble(copiediotdataSplitted2[3])) / 1);
                            dLineError = dLineError + ((toDouble(rangesSplitted[1])
                                    - toDouble(copiediotdataSplitted2[3])) / 1);

                            if (dLineError > maxError) {
                                maxError = dLineError;
                            }
                            if (dLineError < minError) {
                                minError = dLineError;
                            }
                            errorFound = true;
                        } else if (toDouble(copiediotdataSplitted2[3]) >
                                toDouble(rangesSplitted[2])) {
                            dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
                                    - toDouble(rangesSplitted[2])) / 1);
                            dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
                                    - toDouble(rangesSplitted[2])) / 1);
```

```
                            if (dLineError > maxError) {
                                maxError = dLineError;
                            }
                            if (dLineError < minError) {
                                minError = dLineError;
                            }
                            errorFound = true;
                        }
                    }
                } // end of :checks SUNNY

                // checks CLOUDY
                if (rangesSplitted[3].equals("CLOUDY")) {
                    if (getHour(copiediotdataSplitted2[0]) >=
                            ruleHourFrom(rangesSplitted[0])
                            && getHour(copiediotdataSplitted2[0]) <=
                                    ruleHourTo(rangesSplitted[0])
                            && toInt(copiediotdataSplitted2[3]) >=
                                    toInt(rangesSplitted[1])
                            && toInt(copiediotdataSplitted2[3]) <=
                                    toInt(rangesSplitted[2])) {
                        copiediotdata3.add(copiediotdatakwh.get(i));
                        // its already on the ArrayList no need to add it twice
                    } else {
                        // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                        if (toDouble(copiediotdataSplitted2[3]) <
                                toDouble(rangesSplitted[1])) {
                            dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
                                    - toDouble(copiediotdataSplitted2[3])) / 1);
                            dLineError = dLineError + ((toDouble(rangesSplitted[1])
                                    - toDouble(copiediotdataSplitted2[3])) / 1);

                            if (dLineError > maxError) {
                                maxError = dLineError;
                            }
                            if (dLineError < minError) {
                                minError = dLineError;
                            }
                            errorFound = true;
                        } else if (toDouble(copiediotdataSplitted2[3]) >
                                toDouble(rangesSplitted[2])) {
                            dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
                                    - toDouble(rangesSplitted[2])) / 1);
                            dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
                                    - toDouble(rangesSplitted[2])) / 1);
```

```
812
813                                 if (dLineError > maxError) {
814                                     maxError = dLineError;
815                                 }
816                                 if (dLineError < minError) {
817                                     minError = dLineError;
818                                 }
819                                 errorFound = true;
820                             }
821                         }
822                     } // end of :checks CLOUDY
823
824                     // checks DOOR OPEN-LIGHT
825                     if (rangesSplitted[3].equals("DOOR OPEN-LIGHT")) {
826                         if (getHour(copiediotdataSplitted2[0]) >=
827                                 ruleHourFrom(rangesSplitted[0])
828                                 && getHour(copiediotdataSplitted2[0]) <=
829                                         ruleHourTo(rangesSplitted[0])
830                                 && toInt(copiediotdataSplitted2[3]) >=
831                                         toInt(rangesSplitted[1])
832                                 && toInt(copiediotdataSplitted2[3]) <=
833                                         toInt(rangesSplitted[2])) {
834                             copiediotdata3.add(copiediotdatakwh.get(i));
835                             // its already on the ArrayList no need to add it twice
836                         } else {
837                             // assigns TOTAL ERROR AND CURRENT RECORD ERROR
838                             if (toDouble(copiediotdataSplitted2[3]) <
839                                     toDouble(rangesSplitted[1])) {
840                                 dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
841                                         - toDouble(copiediotdataSplitted2[3])) / 1);
842                                 dLineError = dLineError + ((toDouble(rangesSplitted[1])
843                                         - toDouble(copiediotdataSplitted2[3])) / 1);
844
845                                 if (dLineError > maxError) {
846                                     maxError = dLineError;
847                                 }
848                                 if (dLineError < minError) {
849                                     minError = dLineError;
850                                 }
851                                 errorFound = true;
852                             } else if (toDouble(copiediotdataSplitted2[3]) >
853                                     toDouble(rangesSplitted[2])) {
854                                 dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
855                                         - toDouble(rangesSplitted[2])) / 1);
856                                 dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
857                                         - toDouble(rangesSplitted[2])) / 1);
858
859                                 if (dLineError > maxError) {
860                                     maxError = dLineError;
861                                 }
862                                 if (dLineError < minError) {
863                                     minError = dLineError;
864                                 }
865                                 errorFound = true;
866                             }
867                         }
868                     } // end of :checks DOOR OPEN-LIGHT
869
870                     if (rangesSplitted[0].equals("kWh")) {
871                         if (toDouble(copiediotdataSplitted2[7]) <=
872                                 toDouble(rangesSplitted[1])) {
873                             copiediotdata3.add(copiediotdatakwh.get(i));
874                         } else {
875                             dTotalError_kWh = dTotalError_kWh + ((toDouble(copiediotdataSplitted2[7])
876                                     - toDouble(rangesSplitted[2])) / 1);
877                             dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[7])
878                                     - toDouble(rangesSplitted[2])) / 1);
879                             dLineError = dLineError + ((toDouble(copiediotdataSplitted2[7])
880                                     - toDouble(rangesSplitted[2])) / 1);
881
882                             if (dLineError > maxError) {
883                                 maxError = dLineError;
884                             }
885                             if (dLineError < minError) {
886                                 minError = dLineError;
887                             }
888                             errorFound = true;
889                         }
890                     } // end of kWh check
891
892                 } // end of LS
893
894             }
895
896             if (errorFound == true) {
897                 errorsize++;
898                 errorFound = false;
899             }
900
901         }
```

```
902         } catch (Exception e) {
903             System.out.println("Second B: " + errorLine);
904 
905         }
906 
907         copiediotdata3.clear();
908 
909         // ******************************end of second phase (calculating Error
910         // phase)**********************
911 
912         if ((kWh > allowedMonthly_kWh) && (solutionsFound == false))// || (recursiveIterator<6))
913         {
914 
915             if (removedMetaRules.size() == numrules) {
916                 removedMetaRules.clear();
917             }
918 
919             calculateKWH5(sRecordsDaily);
920             return public_ms;
921 
922         } else if (recursiveIterator < 15) {
923             if (recursionResults.size() == 0)// check if result found first time
924             {
925                 String temp[] = new String[4];
926                 String t = "";
927                 boolean first = true;
928                 for (int d = 0; d < removedMetaRules.size(); d++) {
929                     if (first) {
930                         t = removedMetaRules.get(d).toString();
931                         first = false;
932                     } else {
933                         t = t.concat("," + removedMetaRules.get(d));
934                     }
935                 }
936                 temp[0] = t;
937                 temp[1] = String.valueOf(kWh);
938                 temp[2] = String.valueOf(dTotalError);
939                 temp[3] = String.valueOf(kWh_CO2);
940                 recursionResults.add(temp);
941             } else {// now check if new result is better than the previous and replace it
942                 String[] splitted = recursionResults.get(0);
943                 int recIterMinusOne = recursiveIterator - 1;
944                 double anneallingValue = recIterMinusOne - recIterMinusOne * (2 / 15);
945                 Random randAnnealing = new Random();
946                 int numRandAnnealing = randAnnealing.nextInt(100);
```

```
947 
948                 if (kWh_CO2 >= toDouble(splitted[3])) {
949 
950                     String temp[] = new String[4];
951                     String t = "";
952                     boolean first = true;
953                     for (int d = 0; d < removedMetaRules.size(); d++) {
954                         if (first) {
955                             t = removedMetaRules.get(d).toString();
956                             first = false;
957                         } else {
958                             t = t.concat("," + removedMetaRules.get(d));
959                         }
960                     }
961                     temp[0] = t;
962                     temp[1] = String.valueOf(kWh);
963                     temp[2] = String.valueOf(dTotalError);
964                     temp[3] = String.valueOf(kWh_CO2);
965                     recursionResults.set(0, temp);
966                 } else if (numRandAnnealing < anneallingValue) {
967                     String temp[] = new String[4];
968                     String t = "";
969                     boolean first = true;
970                     for (int d = 0; d < removedMetaRules.size(); d++) {
971                         if (first) {
972                             t = removedMetaRules.get(d).toString();
973                             first = false;
974                         } else {
975                             t = t.concat("," + removedMetaRules.get(d));
976                         }
977                     }
978                     temp[0] = t;
979                     temp[1] = String.valueOf(kWh);
980                     temp[2] = String.valueOf(dTotalError);
981                     temp[3] = String.valueOf(kWh_CO2);
982                     recursionResults.set(0, temp);
983                 }
984 
985             }
986 
987             recursiveIterator++;
988             if (removedMetaRules.size() == numrules) {
989                 removedMetaRules.clear();
990             }
991             if (recursiveIterator == 15) {
```

```
                solutionsFound = true;
                removedMetaRules.clear();
                String[] rs = recursionResults.get(0);
                String[] splitted = rs[0].split(",");
                for (int i = 0; i < splitted.length; i++) {
                    if (splitted[0].length() == 0)
                        break;
                    removedMetaRules.add(Integer.valueOf(splitted[i]));
                }

            }
            calculateKWH5(sRecordsDaily);
            return public_ms;
            // return 0;
        } else {
            contiguous_ms = localContiguous_ms;
            dtTemp = dt1;
            GreenPlannerData.addAll(copiediotdatakwh);
            removedMetaRules.clear();
            recursiveIterator = 0;
            recursionResults.clear();
            solutionsFound = false;

            public_ms = ms;
            total_daily_ms_CO2 += daily_ms_CO2;
            return public_ms;

        }

    }// END OF CALCULATE KWH METHOD - number of modifications

    // GREEN PLANNER (CALCULATEkwh5 - associated recursion function)
    public void greenplanner(ArrayList<String[]> data, ArrayList<String[]> MRTrulesOriginal, double formaliseMaxError) {

        copiediotdata = new ArrayList<String[]>(data); // clone original dataset ArrayList
        MRTrules = new ArrayList<String[]>(MRTrulesOriginal);
        listRanges = MRTrules.toArray(new String[0][]);
        try {
            dtTemp = simpleFormatter.parse("2013-10-22 02:32:34.409"); // temp date
        } catch (Exception e) {
            System.out.println(e);
        }
        contiguous_ms = 0;
        ArrayList<String[]> recordsDaily = new ArrayList<String[]>();
        double dayChange = 22;
```

```
        double total_ms = 0;
        GreenPlannerData = new ArrayList<String[]>();
        copiediotdata3 = new ArrayList<String[]>();

        int monthchange = 0;
        int yearchange = 1998;
        DecimalFormat df1 = new DecimalFormat("#.##");
        df1.setRoundingMode(RoundingMode.CEILING);
        String monthString;
        boolean first = true;

        try {
            for (int i = 0; i < copiediotdata.size(); i++) {
                String[] copiediotdataSplitHour = copiediotdata.get(i);

                double tempDAYChange = getDay(copiediotdataSplitHour[0]);
                int tempmonthchange = getMonth(copiediotdataSplitHour[0]);

                if (dayChange == tempDAYChange) {
                    if (i == copiediotdata.size() - 1) // only for the last record something...
                    {
                        recordsDaily.add(copiediotdata.get(i));
                        double hourly_ms = calculateKWH5(recordsDaily);

                        total_ms = total_ms + hourly_ms;
                        recordsDaily.clear();
                    } else {
                        recordsDaily.add(copiediotdata.get(i));
                    }

                } else {
                    dayChange = tempDAYChange;
                    double hourly_ms = calculateKWH5(recordsDaily);

                    total_ms = total_ms + hourly_ms;
                    recordsDaily.clear();
                    recordsDaily.add(copiediotdata.get(i));
                }

                if (monthchange != tempmonthchange) {
                    monthchange = tempmonthchange;

                    if (first) {
                        double seconds1 = total_ms / 1000;
                        double hours1 = seconds1 / 3600;
```

```
                    double kWh1 = ((hours1 * 320) / 1000) * 1100;

                    kwhpermonth = kwhpermonth.concat(df1.format(kWh1));
                    first = false;
                } else {
                    double seconds1 = total_ms / 1000;
                    double hours1 = seconds1 / 3600;
                    double kWh1 = ((hours1 * 320) / 1000) * 1100;

                    kwhpermonth = kwhpermonth.concat("," + df1.format(kWh1));

                }
            }
            // END of calculation for the kwh per month

        }
    } catch (Exception ex) {

        System.out.println("First A: " + errorLine + " => Exception: " + ex + "\nAt line:\t");
        ex.printStackTrace();
    }

    copiediotdata.clear();
    copiediotdata.addAll(GreenPlannerData);
    GreenPlannerData.clear();

    double seconds = total_ms / 1000;
    double hours = seconds / 3600;
    double kWh = ((hours * 320) / 1000) * 1100;
    String label183 = "Total Kilowatt hour (kWh): " + Math.round(kWh) + " kWh";
    System.out.println(label183);
    totalkwh = Math.round(kWh);
    double seconds_CO2 = total_daily_ms_CO2 / 1000;
    double hours_CO2 = seconds_CO2 / 3600;
    double kWh_CO2 = ((hours_CO2 * 320) / 1000) * 1100;// 26
    DecimalFormat df = new DecimalFormat("#.##");
    df.setRoundingMode(RoundingMode.CEILING);
    String label176 = "Kilowatt hour (kWh) with CO2 emmisions: " + df.format(kWh - kWh_CO2) + " kWh";
    System.out.println(label176);
    kwh_with_CO2 = toDouble(df.format(kWh - kWh_CO2));
    String label177 = "Kilowatt hour (kWh) without CO2 emmisions: " + df.format(kWh_CO2) + " kWh";
    kwh_without_CO2 = toDouble(df.format(kWh_CO2));
    System.out.println(label177);
    double KgCO2_co2 = 0.449;
    String label179 = "CO2 emissions : " + df.format((kWh - kWh_CO2) * KgCO2_co2) + " kg CO2e";
    CO2_kg_emissions = toDouble(df.format((kWh - kWh_CO2) * KgCO2_co2));
    System.out.println(label179);
    double dTotalError = 0;
    double dLineError = 0;
    double dTotalError_kWh = 0;

    // **********SECOND AND LAST PHASE*******************//

    double maxError = 0;
    double minError = 100;
    boolean errorFound = false;
    int errorsize = 0;

    try {

        for (int i = 0; i < copiediotdata.size(); i++) {

            dLineError = 0;// gets the total record line error in regards of all the ranges
            for (int k = 0; k < listRanges.length; k++) {
                errorLine = String.join("|", copiediotdata.get(i));

                String[] copiediotdataSplitted2 = copiediotdata.get(i);
                String[] rangesSplitted = listRanges[k];

                if (copiediotdataSplitted2[2].startsWith("T")) // gets T for temperatures
                {

                    // checks time and temperature and if season is empty
                    if ((!rangesSplitted[0].equals("SUMMER") || !rangesSplitted[0].equals("WINTER"))
                            && rangesSplitted[3].equals("TEMPER.")) {
                        if (getHour(copiediotdataSplitted2[0]) >=
                                ruleHourFrom(rangesSplitted[0])
                                && getHour(copiediotdataSplitted2[0]) <=
                                        ruleHourTo(rangesSplitted[0])
                                && toDouble(copiediotdataSplitted2[3]) >=
                                        toDouble(rangesSplitted[1])
                                && toDouble(copiediotdataSplitted2[3]) <=
                                        toDouble(rangesSplitted[2])) {
                            copiediotdata3.add(copiediotdata.get(i));
                            // its already on the ArrayList no need to add it twice
                        } else {
                            // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                            if (getHour(copiediotdataSplitted2[0]) >=
                                    ruleHourFrom(rangesSplitted[0])
                                    && getHour(copiediotdataSplitted2[0]) <=
```

```
1172                                    ruleHourTo(rangesSplitted[0])
1173                                && toDouble(copiediotdataSplitted2[3]) <
1174                                    toDouble(rangesSplitted[1])) {
1175                            dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
1176                                - toDouble(copiediotdataSplitted2[3])) / 1);
1177                            dLineError = dLineError + ((toDouble(rangesSplitted[1])
1178                                - toDouble(copiediotdataSplitted2[3])) / 1);
1179
1180                            if (dLineError > maxError) {
1181                                maxError = dLineError;
1182                            }
1183                            if (dLineError < minError) {
1184                                minError = dLineError;
1185                            }
1186                            errorFound = true;
1187                        } else if (getHour(copiediotdataSplitted2[0]) >=
1188                                ruleHourFrom(rangesSplitted[0])
1189                                && getHour(copiediotdataSplitted2[0]) <=
1190                                    ruleHourTo(rangesSplitted[0])
1191                                && toDouble(copiediotdataSplitted2[3]) >
1192                                    toDouble(rangesSplitted[2])) {
1193                            dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
1194                                - toDouble(rangesSplitted[2])) / 1);
1195                            dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
1196                                - toDouble(rangesSplitted[2])) / 1);
1197
1198                            if (dLineError > maxError) {
1199                                maxError = dLineError;
1200                            }
1201                            if (dLineError < minError) {
1202                                minError = dLineError;
1203                            }
1204                            errorFound = true;
1205                        }
1206
1207                    }
1208                }
1209
1210                if ((rangesSplitted[0].equals("SUMMER") || rangesSplitted[0].equals("WINTER"))
1211                        && rangesSplitted[3].equals("TEMPER.")) {
1212                    int iSeason = getMonth(copiediotdataSplitted2[0]);
1213
1214                    // checks if SUMMER
1215                    if ((iSeason == 6) || (iSeason == 7) || (iSeason == 8)) {
1216                        if (toDouble(copiediotdataSplitted2[3]) >=
```

```
1217                                toDouble(rangesSplitted[1])
1218                                && toDouble(copiediotdataSplitted2[3]) <=
1219                                    toDouble(rangesSplitted[2])) {
1220                            copiediotdata3.add(copiediotdata.get(i));
1221                            // its already on the ArrayList no need to add it twice
1222                        } else {
1223                            // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1224                            if (toDouble(copiediotdataSplitted2[3]) <
1225                                    toDouble(rangesSplitted[1])) {
1226                                dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
1227                                    - toDouble(copiediotdataSplitted2[3])) / 1);
1228                                dLineError = dLineError + ((toDouble(rangesSplitted[1])
1229                                    - toDouble(copiediotdataSplitted2[3])) / 1);
1230
1231                                if (dLineError > maxError) {
1232                                    maxError = dLineError;
1233                                }
1234                                if (dLineError < minError) {
1235                                    minError = dLineError;
1236                                }
1237                                errorFound = true;
1238                            } else if (toDouble(copiediotdataSplitted2[3]) >
1239                                    toDouble(rangesSplitted[2])) {
1240                                dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
1241                                    - toDouble(rangesSplitted[2])) / 1);
1242                                dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
1243                                    - toDouble(rangesSplitted[2])) / 1);
1244
1245                                if (dLineError > maxError) {
1246                                    maxError = dLineError;
1247                                }
1248                                if (dLineError < minError) {
1249                                    minError = dLineError;
1250                                }
1251                                errorFound = true;
1252                            }
1253                        }
1254                    }
1255                    // checks if WINTER
1256                    else if ((iSeason == 12) || (iSeason == 1) || (iSeason == 2)) {
1257                        if (toDouble(copiediotdataSplitted2[3]) >=
1258                                toDouble(rangesSplitted[1])
1259                                && toDouble(copiediotdataSplitted2[3]) <=
1260                                    toDouble(rangesSplitted[2])) {
1261                            copiediotdata3.add(copiediotdata.get(i));
```

```
                        // its already on the ArrayList no need to add it twice
                    } else {
                        // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                        if (toDouble(copiediotdataSplitted2[3]) <
                                toDouble(rangesSplitted[1])) {
                            dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
                                    - toDouble(copiediotdataSplitted2[3])) / 1);
                            dLineError = dLineError + ((toDouble(rangesSplitted[1])
                                    - toDouble(copiediotdataSplitted2[3])) / 1);

                            if (dLineError > maxError) {
                                maxError = dLineError;
                            }
                            if (dLineError < minError) {
                                minError = dLineError;
                            }
                            errorFound = true;
                        } else if (toDouble(copiediotdataSplitted2[3]) >
                                toDouble(rangesSplitted[2])) {
                            dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
                                    - toDouble(rangesSplitted[2])) / 1);
                            dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
                                    - toDouble(rangesSplitted[2])) / 1);

                            if (dLineError > maxError) {
                                maxError = dLineError;
                            }
                            if (dLineError < minError) {
                                minError = dLineError;
                            }
                            errorFound = true;
                        }
                    }
                }

            }

            if (rangesSplitted[0].equals("kWh")) {
                if (toDouble(copiediotdataSplitted2[7]) <=
                        toDouble(rangesSplitted[1])) {
                    copiediotdata3.add(copiediotdata.get(i));
                } else {
                    dTotalError_kWh = dTotalError_kWh + ((toDouble(copiediotdataSplitted2[7])
                            - toDouble(rangesSplitted[2])) / 1);
                    dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[7])
                            - toDouble(rangesSplitted[2])) / 1);
                    dLineError = dLineError + ((toDouble(copiediotdataSplitted2[7])
                            - toDouble(rangesSplitted[2])) / 1);

                    if (dLineError > maxError) {
                        maxError = dLineError;
                    }
                    if (dLineError < minError) {
                        minError = dLineError;
                    }
                    errorFound = true;
                }
            }

            if (copiediotdataSplitted2[2].startsWith("LS")) // gets LS for light sensor
            {
                // checks LIGHT
                if (rangesSplitted[3].equals("LIGHT")) {
                    if (getHour(copiediotdataSplitted2[0]) >=
                            ruleHourFrom(rangesSplitted[0])
                            && getHour(copiediotdataSplitted2[0]) <=
                                    ruleHourTo(rangesSplitted[0])
                            && toInt(copiediotdataSplitted2[3]) >=
                                    toInt(rangesSplitted[1])
                            && toInt(copiediotdataSplitted2[3]) <=
                                    toInt(rangesSplitted[2])) {
                        copiediotdata3.add(copiediotdata.get(i));
                        // its already on the ArrayList no need to add it twice
                    } else {
                        // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                        if (getHour(copiediotdataSplitted2[0]) >=
                                ruleHourFrom(rangesSplitted[0])
                                && getHour(copiediotdataSplitted2[0]) <=
                                        ruleHourTo(rangesSplitted[0])
                                && toDouble(copiediotdataSplitted2[3]) <
                                        toDouble(rangesSplitted[1])) {
                            dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
                                    - toDouble(copiediotdataSplitted2[3])) / 1);
                            dLineError = dLineError + ((toDouble(rangesSplitted[1])
                                    - toDouble(copiediotdataSplitted2[3])) / 1);

                            if (dLineError > maxError) {
                                maxError = dLineError;
```

```
                        }
                        if (dLineError < minError) {
                            minError = dLineError;
                        }
                        errorFound = true;
                    } else if (getHour(copiediotdataSplitted2[0]) >=
                            ruleHourFrom(rangesSplitted[0])
                            && getHour(copiediotdataSplitted2[0]) <=
                                    ruleHourTo(rangesSplitted[0])
                            && toDouble(copiediotdataSplitted2[3]) >
                                    toDouble(rangesSplitted[2])) {
                        dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
                                - toDouble(rangesSplitted[2])) / 1);
                        dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
                                - toDouble(rangesSplitted[2])) / 1);

                        if (dLineError > maxError) {
                            maxError = dLineError;
                        }
                        if (dLineError < minError) {
                            minError = dLineError;
                        }
                        errorFound = true;
                    }
                }
            } // end of: checks LIGHT

            // checks SUNNY
            if (rangesSplitted[3].equals("SUNNY")) {
                if (getHour(copiediotdataSplitted2[0]) >=
                        ruleHourFrom(rangesSplitted[0])
                        && getHour(copiediotdataSplitted2[0]) <=
                                ruleHourTo(rangesSplitted[0])
                        && toInt(copiediotdataSplitted2[3]) >=
                                toInt(rangesSplitted[1])
                        && toInt(copiediotdataSplitted2[3]) <=
                                toInt(rangesSplitted[2])) {
                    copiediotdata3.add(copiediotdata.get(i));
                    // its already on the ArrayList no need to add it twice
                } else {
                    // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                    if (toDouble(copiediotdataSplitted2[3]) <
                            toDouble(rangesSplitted[1])) {
                        dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
                                - toDouble(copiediotdataSplitted2[3])) / 1);
```

```
                        dLineError = dLineError + ((toDouble(rangesSplitted[1])
                                - toDouble(copiediotdataSplitted2[3])) / 1);

                        if (dLineError > maxError) {
                            maxError = dLineError;
                        }
                        if (dLineError < minError) {
                            minError = dLineError;
                        }
                        errorFound = true;
                    } else if (toDouble(copiediotdataSplitted2[3]) >
                            toDouble(rangesSplitted[2])) {
                        dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
                                - toDouble(rangesSplitted[2])) / 1);
                        dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
                                - toDouble(rangesSplitted[2])) / 1);

                        if (dLineError > maxError) {
                            maxError = dLineError;
                        }
                        if (dLineError < minError) {
                            minError = dLineError;
                        }
                        errorFound = true;
                    }
                }
            } // end of :checks SUNNY

            // checks CLOUDY
            if (rangesSplitted[3].equals("CLOUDY")) {
                if (getHour(copiediotdataSplitted2[0]) >=
                        ruleHourFrom(rangesSplitted[0])
                        && getHour(copiediotdataSplitted2[0]) <=
                                ruleHourTo(rangesSplitted[0])
                        && toInt(copiediotdataSplitted2[3]) >=
                                toInt(rangesSplitted[1])
                        && toInt(copiediotdataSplitted2[3]) <=
                                toInt(rangesSplitted[2])) {
                    copiediotdata3.add(copiediotdata.get(i));
                    // its already on the ArrayList no need to add it twice
                    // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                } else if (toDouble(copiediotdataSplitted2[3]) <
                        toDouble(rangesSplitted[1])) {

                    dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
```

```
1442                                    - toDouble(copiediotdataSplitted2[3])) / 1);
1443                            dLineError = dLineError + ((toDouble(rangesSplitted[1])
1444                                    - toDouble(copiediotdataSplitted2[3])) / 1);
1445
1446                            if (dLineError > maxError) {
1447                                maxError = dLineError;
1448                            } else if (dLineError < minError) {
1449                                minError = dLineError;
1450                            }
1451                            errorFound = true;
1452                        } else if (toDouble(copiediotdataSplitted2[3]) >
1453                                toDouble(rangesSplitted[2])) {
1454
1455                            dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
1456                                    - toDouble(rangesSplitted[2])) / 1);
1457                            dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
1458                                    - toDouble(rangesSplitted[2])) / 1);
1459
1460                            if (dLineError > maxError) {
1461                                maxError = dLineError;
1462                            } else if (dLineError < minError) {
1463                                minError = dLineError;
1464                            }
1465                            errorFound = true;
1466                        }
1467
1468                    } // end of :checks CLOUDY
1469
1470                    // checks DOOR OPEN-LIGHT
1471                    if (rangesSplitted[3].equals("DOOR OPEN-LIGHT")) {
1472                        if (getHour(copiediotdataSplitted2[0]) >=
1473                                ruleHourFrom(rangesSplitted[0])
1474                                && getHour(copiediotdataSplitted2[0]) <=
1475                                        ruleHourTo(rangesSplitted[0])
1476                                && toInt(copiediotdataSplitted2[3]) >=
1477                                        toInt(rangesSplitted[1])
1478                                && toInt(copiediotdataSplitted2[3]) <=
1479                                        toInt(rangesSplitted[2])) {
1480                            copiediotdata3.add(copiediotdata.get(i));
1481                            // its already on the ArrayList no need to add it twice
1482                        } else {
1483                            // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1484                            if (toDouble(copiediotdataSplitted2[3]) <
1485                                    toDouble(rangesSplitted[1])) {
1486                                dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
```

```
1487                                    - toDouble(copiediotdataSplitted2[3])) / 1);
1488                            dLineError = dLineError + ((toDouble(rangesSplitted[1])
1489                                    - toDouble(copiediotdataSplitted2[3])) / 1);
1490
1491                            if (dLineError > maxError) {
1492                                maxError = dLineError;
1493                            }
1494                            if (dLineError < minError) {
1495                                minError = dLineError;
1496                            }
1497                            errorFound = true;
1498                        } else if (toDouble(copiediotdataSplitted2[3]) >
1499                                toDouble(rangesSplitted[2])) {
1500                            dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
1501                                    - toDouble(rangesSplitted[2])) / 1);
1502                            dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
1503                                    - toDouble(rangesSplitted[2])) / 1);
1504
1505                            if (dLineError > maxError) {
1506                                maxError = dLineError;
1507                            }
1508                            if (dLineError < minError) {
1509                                minError = dLineError;
1510                            }
1511                            errorFound = true;
1512                        }
1513                    }
1514                } // end of :checks DOOR OPEN-LIGHT
1515
1516                if (rangesSplitted[0].equals("kWh")) {
1517                    if (toDouble(copiediotdataSplitted2[7]) <=
1518                            toDouble(rangesSplitted[1])) {
1519                        copiediotdata3.add(copiediotdata.get(i));
1520                    } else {
1521                        dTotalError_kWh = dTotalError_kWh + ((toDouble(copiediotdataSplitted2[7])
1522                                - toDouble(rangesSplitted[2])) / 1);
1523                        dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[7])
1524                                - toDouble(rangesSplitted[2])) / 1);
1525                        dLineError = dLineError + ((toDouble(copiediotdataSplitted2[7])
1526                                - toDouble(rangesSplitted[2])) / 1);
1527
1528                        if (dLineError > maxError) {
1529                            maxError = dLineError;
1530                        }
1531                        if (dLineError < minError) {
```

```
                                    minError = dLineError;
                                }
                                errorFound = true;
                            }
                        } // end of kWh check

                    } // end of LS

                }

                if (errorFound == true) {
                    errorsize++;
                    errorFound = false;
                }

            }
        } catch (Exception e) {
            System.out.println("Second: " + errorLine);

        }

        // ******************************end of second phase *********************

        df = new DecimalFormat("#.####");
        df.setRoundingMode(RoundingMode.CEILING);

        dTotalError = dTotalError * 10;

        String label189 = "Total Error: " + df.format((dTotalError * 100) / formaliseMaxError);
        String label186 = "Max Error: " + df.format((maxError * 100) / formaliseMaxError) + "%";
        String label185 = "Min Error: " + df.format((minError * 100) / formaliseMaxError) + "%";
        System.out.println(label189);
        System.out.println(label186);
        System.out.println(label185);

        // calculate Total Average Error
        double averageErrorTotal = Math.round((dTotalError / copiediotdata.size()));
        String label182 = "Total Average Error: " + df.format((averageErrorTotal * 100) / formaliseMaxError) + "%";
        System.out.println(label182);
        totalaverageerror = toDouble(df.format((averageErrorTotal * 100) / formaliseMaxError));
        // calculate Budget Average Error
        double averageError_Budget = Math.round((dTotalError_kWh / copiediotdata.size()));
        String label180 = "Budget Average Error: " + df.format((averageError_Budget * 100) / formaliseMaxError) + "%";
        System.out.println(label180);
        budgeterror = toDouble(df.format((averageError_Budget * 100) / formaliseMaxError));
```

```
        // calculate Convenience Average Error
        double averageError_Concenience = Math.round(((dTotalError - dTotalError_kWh) / copiediotdata.size()));
        String label181 = "Convenience Average Error: "
                + df.format((averageError_Concenience * 100) / formaliseMaxError) + "%";
        System.out.println(label181);
        convinienceerror = toDouble(df.format((averageError_Concenience * 100) / formaliseMaxError));

        copiediotdata3.clear();

        // ************SECOND RUN*********************************
        double meansSum = 0;
        maxError = 0;
        minError = 100;
        dTotalError = 0;
        dLineError = 0;

        try {

            for (int i = 0; i < copiediotdata.size(); i++) {

                dLineError = 0;// gets the total record line error in regards of all the ranges
                for (int k = 0; k < listRanges.length; k++) {
                    errorLine = String.join("|", copiediotdata.get(i));

                    String[] copiediotdataSplitted2 = copiediotdata.get(i);
                    String[] rangesSplitted = listRanges[k];

                    if (copiediotdataSplitted2[2].startsWith("T")) // gets T for temperatures
                    {
                        // checks time and temperature and if season is empty
                        if ((!rangesSplitted[0].equals("SUMMER") || !rangesSplitted[0].equals("WINTER"))
                                && rangesSplitted[3].equals("TEMPER.")) {
                            if (getHour(copiediotdataSplitted2[0]) >=
                                    ruleHourFrom(rangesSplitted[0])
                                    && getHour(copiediotdataSplitted2[0]) <=
                                            ruleHourTo(rangesSplitted[0])
                                    && toDouble(copiediotdataSplitted2[3]) >=
                                            toDouble(rangesSplitted[1])
                                    && toDouble(copiediotdataSplitted2[3]) <=
                                            toDouble(rangesSplitted[2])) {
                                copiediotdata3.add(copiediotdata.get(i));
                                // its already on the ArrayList no need to add it twice

                            } else {
                                // assigns TOTAL ERROR AND CURRENT RECORD ERROR
```

```
1622                     if (getHour(copiediotdataSplitted2[0]) >=
1623                             ruleHourFrom(rangesSplitted[0])
1624                             && getHour(copiediotdataSplitted2[0]) <=
1625                                     ruleHourTo(rangesSplitted[0])
1626                             && toDouble(copiediotdataSplitted2[3]) <
1627                                     toDouble(rangesSplitted[1])) {
1628                         dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
1629                                 - toDouble(copiediotdataSplitted2[3])) / 1);
1630                         dLineError = dLineError + ((toDouble(rangesSplitted[1])
1631                                 - toDouble(copiediotdataSplitted2[3])) / 1);
1632
1633                         if (dLineError > maxError) {
1634                             maxError = dLineError;
1635                         }
1636                         if (dLineError < minError) {
1637                             minError = dLineError;
1638                         }
1639
1640                     } else if (getHour(copiediotdataSplitted2[0]) >=
1641                             ruleHourFrom(rangesSplitted[0])
1642                             && getHour(copiediotdataSplitted2[0]) <=
1643                                     ruleHourTo(rangesSplitted[0])
1644                             && toDouble(copiediotdataSplitted2[3]) >
1645                                     toDouble(rangesSplitted[2])) {
1646                         dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
1647                                 - toDouble(rangesSplitted[2])) / 1);
1648                         dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
1649                                 - toDouble(rangesSplitted[2])) / 1);
1650
1651                         if (dLineError > maxError) {
1652                             maxError = dLineError;
1653                         }
1654                         if (dLineError < minError) {
1655                             minError = dLineError;
1656                         }
1657
1658                     }
1659
1660                 }
1661             }
1662
1663             if ((rangesSplitted[0].equals("SUMMER") || rangesSplitted[0].equals("WINTER"))
1664                     && rangesSplitted[3].equals("TEMPER.")) {
1665                 int iSeason = getMonth(copiediotdataSplitted2[0]);
1666
```

```
1667                 // checks if SUMMER
1668                 if ((iSeason == 6) || (iSeason == 7) || (iSeason == 8)) {
1669                     if (toDouble(copiediotdataSplitted2[3]) >=
1670                             toDouble(rangesSplitted[1])
1671                             && toDouble(copiediotdataSplitted2[3]) <=
1672                                     toDouble(rangesSplitted[2])) {
1673                         copiediotdata3.add(copiediotdata.get(i));
1674                         // its already on the ArrayList no need to add it twice
1675
1676                     } else {
1677                         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1678                         if (toDouble(copiediotdataSplitted2[3]) <
1679                                 toDouble(rangesSplitted[1])) {
1680                             dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
1681                                     - toDouble(copiediotdataSplitted2[3])) / 1);
1682                             dLineError = dLineError + ((toDouble(rangesSplitted[1])
1683                                     - toDouble(copiediotdataSplitted2[3])) / 1);
1684
1685                             if (dLineError > maxError) {
1686                                 maxError = dLineError;
1687                             }
1688                             if (dLineError < minError) {
1689                                 minError = dLineError;
1690                             }
1691
1692                         } else if (toDouble(copiediotdataSplitted2[3]) >
1693                                 toDouble(rangesSplitted[2])) {
1694                             dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
1695                                     - toDouble(rangesSplitted[2])) / 1);
1696                             dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
1697                                     - toDouble(rangesSplitted[2])) / 1);
1698
1699                             if (dLineError > maxError) {
1700                                 maxError = dLineError;
1701                             }
1702                             if (dLineError < minError) {
1703                                 minError = dLineError;
1704                             }
1705
1706                         }
1707                     }
1708                 }
1709                 // checks if WINTER
1710                 else if ((iSeason == 12) || (iSeason == 1) || (iSeason == 2)) {
1711                     if (toDouble(copiediotdataSplitted2[3]) >=
```

```
                                    toDouble(rangesSplitted[1])
                                    && toDouble(copiediotdataSplitted2[3]) <=
                                            toDouble(rangesSplitted[2])) {
                                copiediotdata3.add(copiediotdata.get(i));
                                // its already on the ArrayList no need to add it twice

                            } else {
                                // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                                if (toDouble(copiediotdataSplitted2[3]) <
                                        toDouble(rangesSplitted[1])) {
                                    dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
                                            - toDouble(copiediotdataSplitted2[3])) / 1);
                                    dLineError = dLineError + ((toDouble(rangesSplitted[1])
                                            - toDouble(copiediotdataSplitted2[3])) / 1);

                                    if (dLineError > maxError) {
                                        maxError = dLineError;
                                    }
                                    if (dLineError < minError) {
                                        minError = dLineError;
                                    }

                                } else if (toDouble(copiediotdataSplitted2[3]) >
                                        toDouble(rangesSplitted[2])) {
                                    dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
                                            - toDouble(rangesSplitted[2])) / 1);
                                    dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
                                            - toDouble(rangesSplitted[2])) / 1);

                                    if (dLineError > maxError) {
                                        maxError = dLineError;
                                    }
                                    if (dLineError < minError) {
                                        minError = dLineError;
                                    }

                                }
                            }
                        }

                    }

                    if (rangesSplitted[0].equals("kWh")) {
                        if (toDouble(copiediotdataSplitted2[7]) <=
                                toDouble(rangesSplitted[1])) {
```

```
                            copiediotdata3.add(copiediotdata.get(i));
                        } else {
                            dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[7])
                                    - toDouble(rangesSplitted[2])) / 1);
                            dLineError = dLineError + ((toDouble(copiediotdataSplitted2[7])
                                    - toDouble(rangesSplitted[2])) / 1);

                            if (dLineError > maxError) {
                                maxError = dLineError;
                            }
                            if (dLineError < minError) {
                                minError = dLineError;
                            }
                            errorFound = true;
                        }
                    }

                }

                if (copiediotdataSplitted2[2].startsWith("LS")) // gets LS for light sensor
                {
                    // checks LIGHT
                    if (rangesSplitted[3].equals("LIGHT")) {
                        if (getHour(copiediotdataSplitted2[0]) >=
                                ruleHourFrom(rangesSplitted[0])
                                && getHour(copiediotdataSplitted2[0]) <=
                                        ruleHourTo(rangesSplitted[0])
                                && toInt(copiediotdataSplitted2[3]) >=
                                        toInt(rangesSplitted[1])
                                && toInt(copiediotdataSplitted2[3]) <=
                                        toInt(rangesSplitted[2])) {
                            copiediotdata3.add(copiediotdata.get(i));
                            // its already on the ArrayList no need to add it twice

                        } else {
                            // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                            if (getHour(copiediotdataSplitted2[0]) >=
                                    ruleHourFrom(rangesSplitted[0])
                                    && getHour(copiediotdataSplitted2[0]) <=
                                            ruleHourTo(rangesSplitted[0])
                                    && toDouble(copiediotdataSplitted2[3]) <
                                            toDouble(rangesSplitted[1])) {
                                dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
                                        - toDouble(copiediotdataSplitted2[3])) / 1);
                                dLineError = dLineError + ((toDouble(rangesSplitted[1])
```

```
                                - toDouble(copiediotdataSplitted2[3])) / 1);

                        if (dLineError > maxError) {
                            maxError = dLineError;
                        }
                        if (dLineError < minError) {
                            minError = dLineError;
                        }

                    } else if (getHour(copiediotdataSplitted2[0]) >=
                            ruleHourFrom(rangesSplitted[0])
                            && getHour(copiediotdataSplitted2[0]) <=
                                    ruleHourTo(rangesSplitted[0])
                            && toDouble(copiediotdataSplitted2[3]) >
                                    toDouble(rangesSplitted[2])) {
                        dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
                                - toDouble(rangesSplitted[2])) / 1);
                        dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
                                - toDouble(rangesSplitted[2])) / 1);

                        if (dLineError > maxError) {
                            maxError = dLineError;
                        }
                        if (dLineError < minError) {
                            minError = dLineError;
                        }

                    }
                }
            } // end of: checks LIGHT

            // checks SUNNY
            if (rangesSplitted[3].equals("SUNNY")) {
                if (getHour(copiediotdataSplitted2[0]) >=
                        ruleHourFrom(rangesSplitted[0])
                        && getHour(copiediotdataSplitted2[0]) <=
                                ruleHourTo(rangesSplitted[0])
                        && toInt(copiediotdataSplitted2[3]) >=
                                toInt(rangesSplitted[1])
                        && toInt(copiediotdataSplitted2[3]) <=
                                toInt(rangesSplitted[2])) {
                    copiediotdata3.add(copiediotdata.get(i));
                    // its already on the ArrayList no need to add it twice

                } else {
```

```
                    // assigns TOTAL ERROR AND CURRENT RECORD ERROR
                    if (toDouble(copiediotdataSplitted2[3]) <
                            toDouble(rangesSplitted[1])) {
                        dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
                                - toDouble(copiediotdataSplitted2[3])) / 1);
                        dLineError = dLineError + ((toDouble(rangesSplitted[1])
                                - toDouble(copiediotdataSplitted2[3])) / 1);

                        if (dLineError > maxError) {
                            maxError = dLineError;
                        }
                        if (dLineError < minError) {
                            minError = dLineError;
                        }

                    } else if (toDouble(copiediotdataSplitted2[3]) >
                            toDouble(rangesSplitted[2])) {
                        dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
                                - toDouble(rangesSplitted[2])) / 1);
                        dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
                                - toDouble(rangesSplitted[2])) / 1);

                        if (dLineError > maxError) {
                            maxError = dLineError;
                        }
                        if (dLineError < minError) {
                            minError = dLineError;
                        }

                    }
                }
            } // end of :checks SUNNY

            // checks CLOUDY
            if (rangesSplitted[3].equals("CLOUDY")) {
                if (getHour(copiediotdataSplitted2[0]) >=
                        ruleHourFrom(rangesSplitted[0])
                        && getHour(copiediotdataSplitted2[0]) <=
                                ruleHourTo(rangesSplitted[0])
                        && toInt(copiediotdataSplitted2[3]) >=
                                toInt(rangesSplitted[1])
                        && toInt(copiediotdataSplitted2[3]) <=
                                toInt(rangesSplitted[2])) {
                    copiediotdata3.add(copiediotdata.get(i));
                    // its already on the ArrayList no need to add it twice
```

```
1892
1893                        } else if (toDouble(copiediotdataSplitted2[3]) <
1894                                toDouble(rangesSplitted[1])) {
1895                            // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1896                            dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
1897                                    - toDouble(copiediotdataSplitted2[3])) / 1);
1898                            dLineError = dLineError + ((toDouble(rangesSplitted[1])
1899                                    - toDouble(copiediotdataSplitted2[3])) / 1);
1900
1901                            if (dLineError > maxError) {
1902                                maxError = dLineError;
1903                            } else if (dLineError < minError) {
1904                                minError = dLineError;
1905                            }
1906
1907                        } else if (toDouble(copiediotdataSplitted2[3]) >
1908                                toDouble(rangesSplitted[2])) {
1909                            dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
1910                                    - toDouble(rangesSplitted[2])) / 1);
1911                            dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
1912                                    - toDouble(rangesSplitted[2])) / 1);
1913
1914                            if (dLineError > maxError) {
1915                                maxError = dLineError;
1916                            } else if (dLineError < minError) {
1917                                minError = dLineError;
1918                            }
1919
1920                        }
1921
1922                    } // end of :checks CLOUDY
1923
1924                    // checks DOOR OPEN-LIGHT
1925                    if (rangesSplitted[3].equals("DOOR OPEN-LIGHT")) {
1926                        if (getHour(copiediotdataSplitted2[0]) >=
1927                                ruleHourFrom(rangesSplitted[0])
1928                                && getHour(copiediotdataSplitted2[0]) <=
1929                                        ruleHourTo(rangesSplitted[0])
1930                                && toInt(copiediotdataSplitted2[3]) >=
1931                                        toInt(rangesSplitted[1])
1932                                && toInt(copiediotdataSplitted2[3]) <=
1933                                        toInt(rangesSplitted[2])) {
1934                            copiediotdata3.add(copiediotdata.get(i));
1935                            // its already on the ArrayList no need to add it twice
1936
```

```
1937                        } else {
1938                            // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1939                            if (toDouble(copiediotdataSplitted2[3]) <
1940                                    toDouble(rangesSplitted[1])) {
1941                                dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
1942                                        - toDouble(copiediotdataSplitted2[3])) / 1);
1943                                dLineError = dLineError + ((toDouble(rangesSplitted[1])
1944                                        - toDouble(copiediotdataSplitted2[3])) / 1);
1945
1946                                if (dLineError > maxError) {
1947                                    maxError = dLineError;
1948                                }
1949                                if (dLineError < minError) {
1950                                    minError = dLineError;
1951                                }
1952
1953                            } else if (toDouble(copiediotdataSplitted2[3]) >
1954                                    toDouble(rangesSplitted[2])) {
1955                                dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
1956                                        - toDouble(rangesSplitted[2])) / 1);
1957                                dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
1958                                        - toDouble(rangesSplitted[2])) / 1);
1959
1960                                if (dLineError > maxError) {
1961                                    maxError = dLineError;
1962                                }
1963                                if (dLineError < minError) {
1964                                    minError = dLineError;
1965                                }
1966
1967                            }
1968                        }
1969                    } // end of :checks DOOR OPEN-LIGHT
1970
1971                    if (rangesSplitted[0].equals("kWh")) {
1972                        if (toDouble(copiediotdataSplitted2[7]) <=
1973                                toDouble(rangesSplitted[1])) {
1974                            copiediotdata3.add(copiediotdata.get(i));
1975                        } else {
1976                            dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[7])
1977                                    - toDouble(rangesSplitted[2])) / 1);
1978                            dLineError = dLineError + ((toDouble(copiediotdataSplitted2[7])
1979                                    - toDouble(rangesSplitted[2])) / 1);
1980
1981                            if (dLineError > maxError) {
```

```
                                    maxError = dLineError;
                                }
                                if (dLineError < minError) {
                                    minError = dLineError;
                                }
                                errorFound = true;
                            }
                        } // end of kWh check

                    } // end of LS

                }

                meansSum = meansSum + Math.pow(dLineError - averageErrorTotal, 2);

            }
        } catch (Exception e) {
            System.out.println("Second: " + errorLine);

        }

        double standardDeviation = Math.sqrt(meansSum / copiediotdata.size());
        String label184 = "Standard Deviation: " + df.format((standardDeviation * 100) / formaliseMaxError) + "%";
        deviation = toDouble(df.format((standardDeviation * 100) / formaliseMaxError));

        copiediotdata3.clear();

        postData();
    }// END OF Green PLANNER - number of modifications

}
```