Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Τμήμα Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Μονάδα Επεξεργασίας Σήματος
και Βιοϊατρικής Τεχνολογίας

Διπλωματική Εργασία

Υλοποίηση διεπαφής έξυπνου καθρέφτη για ανάπτυξη εφαρμογών lifestyle και εξατομικευμένης υγείας

Ζαχαρόπουλος Φίλιππος 8559
filipposz@ece.auth.gr

Παπαγεωργίου Δημήτριος 8884
dim_papag@windowslive.com

Επιβλέποντες:

Χατζηλεοντιάδης Λεόντιος
Καθηγητής Α.Π.Θ.

Ντράχα Αναστασία
Μεταδιδακτορική Ερευνήτρια

Χατζηδημητρίου Στέλιος
Μεταδιδακτορικός Ερευνητής

21 Ιανουαρίου 2022

Ευχαριστιες

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

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

Περίληψη

Το διαδίκτυο των πραγμάτων (*Internet of Things ή IoT*) είναι ένας κλάδος που εξελίσσεται ραγδαία ειδικά τα τελευταία χρόνια. Υπάρχει η δυνατότητα ανάπτυξης όλο και περισσότερων εφαρμογών, χρήσιμες για πολλούς ανθρώπους, είτε έχουν να κάνουν με απλές λειτουργίες σε συστήματα αυτοματισμού, είτε με μεγαλύτερης κλίμακας εφαρμογές στη βιομηχανία. Επομένως, όλο και περισσότερος κόσμος επιθυμεί να ασχοληθεί με το αντικείμενο αυτό.

Η διαδικασία υλοποίησης ενός IoT συστήματος περιλαμβάνει την ανάπτυξη κώδικα για τον έλεγχο των συσκευών. Μάλιστα, στις περισσότερες περιπτώσεις η γρήγορη απόκριση είναι υψίστης σημασίας, επομένως απαιτείται η ανάπτυξη χαμηλού επιπέδου κώδικα, καθώς και η χρήση λειτουργικών συστημάτων πραγματικού χρόνου (*Real Time Operating System ή RTOS*). Επίσης, λόγω της μεγάλης ετερογένειας IoT συσκευών που υπάρχουν στην αγορά, κρίνεται αναγκαία η κατανόηση των δυνατοτήτων που η εκάστοτε συσκευή μπορεί να προσφέρει, ώστε να γίνεται η κατάλληλη επιλογή τους, προσαρμοσμένη στις ανάγκες του συστήματος προς υλοποίηση.

Οι ενέργειες αυτές είναι λογικό να φαίνονται περίπλοκες σε κάποιους χρήστες, ειδικότερα στα άτομα που είναι τεχνολογικά ακατάρτιστα, δεν έχουν δηλαδή τις απαραίτητες προγραμματιστικές γνώσεις, αλλά παρόλα αυτά επιθυμούν να κατασκευάσουν ένα IoT σύστημα π.χ. για προσωπική τους χρήση. Αυτό έχει ως αποτέλεσμα μεγάλη μερίδα κόσμου που θέλει να ασχοληθεί με το IoT να αποθαρρύνεται.

Η μοντελοστρεφής μηχανική (*Model Driven Engineering ή MDE*), έρχεται να δώσει λύση στα προβλήματα που μπορεί να αντιμετωπίσουν όσοι/ες θέλουν να ασχοληθούν με το IoT, αλλά και γενικότερα να απλοποιήσει τη διαδικασία παραγωγής λογισμικού, καθώς μπορεί να παρέχει την ανάπτυξη IoT συστημάτων σε ένα πιο αφαιρετικό επίπεδο, το οποίο είναι πιο φιλικό προς τον απλό χρήστη.

Μέσω της παρούσας διπλωματικής εργασίας, δίνεται η δυνατότητα σε κάποιον/α να περιγράψει, με χρήση μοντέλων, IoT συσκευές, μέσω δύο γλωσσών συγκεκριμένου πεδίου (*Domain Specific Language ή DSL*) που αναπτύχθηκαν, για την περιγραφή των συσκευών και των μεταξύ τους συνδέσεων. Από τα μοντέλα πραγματοποιείται ένας Model-to-Text μετασχηματισμός για την αυτόματη παραγωγή λογισμικού, για μια πληθώρα IoT συσκευών, προσαρμοσμένη στα χαρακτηριστικά που επιθυμεί ο χρήστης να έχει. Το λογισμικό ελέγχου των IoT συσκευών που παράγεται υλοποιεί την λήψη μετρήσεων από αισθητήρες και την αποστολή τους σε κάποιον μεσολαβητή (*broker*), αλλά και τον έλεγχο ενεργοποιητών μέσω του broker. Επίσης αποτελείται από χαμηλού επιπέδου κώδικα, καθώς έχει σχεδιαστεί σύμφωνα με τις απαιτήσεις ενός λειτουργικού συστήματος πραγματικού χρόνου, το RIOT. Τέλος, πραγματοποιείται και ένας Model-to-Model μετασχηματισμός για την παραγωγή διαγραμμάτων τα οποία βοηθούν στην οπτικοποίηση και άρα καλύτερη αντίληψη από τον χρήστη για τη συνδεσμολογία και ενδοεπικοινωνία του συστήματός του.

Title

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

Abstract

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

Internet of Things (*IoT*) is a field that is evolving rapidly, especially in recent years. There is the possibility of developing even more applications which prove to be useful for many people, whether they have to do with simple functions in automation systems, or with larger scale applications in the industry. Therefore, more and more people want to work in this field.

The process of developing an IoT system involves code development to control the system's devices. In fact, in most cases fast response is of the utmost importance, so low-level code development is required, as well as the use of real-time operating systems (*RTOS*). Also, due to the great heterogeneity of IoT devices on the market, it is necessary to understand the capabilities that each device can offer, in order to make the appropriate choice of one, tailored to the needs of the system to be implemented.

These requirements may seem complicated to some users, especially to people who are technologically untrained, i.e. do not have the necessary programming skills, but still want to build an IoT system e.g. for their personal use. This results in a large portion of people wanting to get involved with IoT, being discouraged to do so.

Model Driven Engineering (*MDE*) is here to solve the problems that, those who want to get involved with IoT, may face, but also to simplify the software production process in general, as it can provide the developing of IoT systems to a more abstract level, which is more user friendly.

Through this diploma thesis, one is given the opportunity to describe, using models, IoT devices, through two domain specific Languages (*DSL*) developed for the description of devices and the connections between them. From the models, a Model-to-Text (*M2T*) transformation is performed for the automated code generation, for a variety of IoT devices, adapted to the characteristics that the user wishes for it to have. The software for controlling the IoT devices that is produced implements the process of taking measurements from sensors, and sending them to a *broker*, but also the process of controlling actuators through the broker. It also consists of low-level code, as it has been designed according to the requirements of a real time operating system, named RIOT. Finally, a Model-to-Model (*M2M*) transformation takes place in order

to produce diagrams that provide a visualization and thus a better understanding by the user, of the wiring and intercommunication of their system.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος πινάκων

Ακρωνύμια Εγγράφου

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

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

1 Εισαγωγή

Αν και υπήρχε ως ιδέα εδώ και περίπου 50 χρόνια, το διαδίκτυο των πραγμάτων, και ως όρος αλλά και ως προς τη χρήση του, ήρθε στο επίκεντρο του ενδιαφέροντος τα τελευταία 10 χρόνια και καθημερινά γίνεται ολοένα και πιο διαδεδομένο. Πλέον μάλιστα ο συνολικός αριθμός συνδεδεμένων συσκευών στο διαδίκτυο είναι μεγαλύτερος από αυτόν του πληθυσμού της Γης[1].

Το IoT βρίσκει εφαρμογή στην ανάπτυξη έξυπνων υποδομών και αυτοματοποίησης διαδικασιών. Από την δημιουργία ενός ''Έξυπνου Σπιτιού'' μέχρι και σε κάτι τόσο ουσιώδες όπως την καλύτερη παρακολούθηση ασθενών σε νοσοκομεία και άρα την πιο σωστή περίθαλψή τους. Άλλα παραδείγματα εφαρμογής είναι οι ''Έξυπνες πόλεις'' (διαχείρηση κυκλοφορίας στους δρόμους, διαχείριση απορριμάτων, διανομή νερού κ.α.), τα αυτοκινούμενα οχήματα, οι αυτοματισμοί στη γεωργία, και γενικότερα στη βιομηχανία.

Η ολοένα και μεγαλύτερη διάδοση του IoT, συνεπάγεται και την αξιοποίησή του από μεγαλύτερο κοινό, στο οποίο ανήκουν και άτομα τα οποία μπορεί να γνωρίζουν σε βάθος ένα συγκεκριμένο αντικείμενο στο οποίο βρίσκει εφαρμογή το IoT, όμως δεν έχουν επαρκείς, ή και καθόλου γνώσεις προγραμματισμού (οι λεγόμενοι citizen developers). Κρίνεται σκόπιμη λοιπόν η ανάπτυξη μεθόδων που θα μετατρέπουν την δημιουργία ενός συστήμος IoT σε διαδικασία πιο φιλική προς τα άτομα αυτά. Ταυτόχρονα, καθώς η αυτοματοποίηση διαδικασιών φαίνεται να χρησιμεύει σε όλο και περισσότερους τομείς, έχει αρχίσει να εμφανίζεται η *ανάπτυξη χαμηλού-κώδικα* (*low-code development*), η οποία αποσκοπεί σε όσο το δυνατό λιγότερη χρήση κώδικα για την παραγωγή λογισμικού.

Εδώ βρίσκει άμεση εφαρμογή η Μοντελοστρεφής Μηχανική, η οποία προσφέ-

[1] *https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/*

ρει γρήγορη και πιο αυτοματοποιημένη ανάπτυξη λογισμικού. Στο πλαίσιο αυτής, οι γλώσσες συγκεκριμένου πεδίου προσφέρουν ένα πιο αφαιρετικό επίπεδο για την παραγωγή λογισμικού, επομένως αποτελούν ένα πολύ σημαντικό εργαλείο για προγραμματιστές και μη, είτε για να απλοποιηθεί η διαδικασία παραγωγής για τους πρώτους, είτε για να καλυφθεί το κενό προγραμματιστικών γνώσεων για τους δεύτερους.

1.1 Περιγραφη του Προβληματος

Πέρα από τα προβλήματα που αναλύθηκαν στην προηγούμενη παράγραφο, στα οποία δίνει λύση η MDE, ένα ακόμη σημαντικό θέμα που εμφανίζεται με την ανάπτυξη του κλάδου του IoT είναι η κατασκευή όλο και περισσότερων διαφορετικών IoT συσκευών. Φυσικά, λόγω αυτού από τη μία επεκτείνονται οι δυνατότητες που ένα IoT σύστημα μπορεί να έχει, από την άλλη όμως αυξάνεται η πολυπλοκότητα και ετερογένεια στο IoT.

Υπάρχει πληθώρα IoT συσκευών στην αγορά, όπως π.χ. τα έξυπνα ρολόγια, που διανέμονται έτοιμες για χρήση. Σε αυτές τις περιπτώσεις, οι χρήστες μπορούν να ακολουθήσουν σαφείς οδηγίες χρήσης από τον κατασκευαστή, και άρα πολύ εύκολα να αξιοποιήσουν τις δυνατότητες που η εκάστοτε συσκευή προσφέρει. Επομένως, το πρόβλημα της πολυπλοκότητας δεν εμφανίζεται σε τέτοιου είδους IoT συσκευές.

Στην περίπτωση όμως που κάποιος/α επιθυμεί να αναπτύξει ένα σύστημα με IoT συσκευές από την αρχή, επειδή π.χ. θέλει να πειραματιστεί ή να υλοποιήσει κάποιες λειτουργίες πιο εξειδικευμένες, τότε απαιτείται μια μεγάλη διαδικασία για την κατασκευή και άρα πολλές γνώσεις. Το πρώτο βήμα είναι η επιλογή των κατάλληλων μικροελεγκτών, αισθητήρων, ενεργοποιητών για την υλοποίηση της ιδέας. Απαιτείται λοιπόν γνώση πάνω στον τρόπο λειτουργίας των συσκευών αυτών, καθώς και στον τρόπο διασύνδεσης και επικοινωνίας τους. Ακολουθεί η ανάπτυξη λογισμικού για την υλοποίηση των επιθυμητών λειτουργιών, κάτι το οποίο από μόνο του σημαίνει πως πρέπει να υπάρχει εμπειρία με προγραμματισμό και πρωτόκολλα επικοινωνίας. Επίσης, σε πολλες περιπτώσεις, στο σύστημα που υλοποιείται απαιτείται η ύπαρξη ιδιοτήτων όπως η ακρίβεια στον χρόνο απόκρισης ή η χαμηλή κατανάλωση ενέργειας. Επομένως, απαιτούνται και οι γνώσεις των ιδιοτήτων των RTOS, καθώς και της κατάλληλης χρήσης τους.

1.2 Σκοπος - Συνεισφορα της Διπλωματικης Εργασιας

Η παρούσα διπλωματική έχει ως στόχο την ανάπτυξη μιας μηχανής λογισμικού μοντελοστρεφούς λογικής, με την οποία οι χρήστες θα μπορούν να μοντελοποιούν συσκευές καθώς και την διασύνδεσή τους. Οι συσκευές αυτές θα μπορούν να είναι

είτε μικροελεγκτές, είτε περιφερειακά (αισθητήρες και ενεργοποιητές), και όλα μαζί θα συνδέονται κατάλληλα για να συνθέσουν ένα σύστημα.

Αρχικά υλοποιήθηκαν δύο DSL για την περιγραφή των συσκευών και των μεταξύ τους συνδέσεων. Στην μία περιγράφονται τα χαρακτηριστικά των συσκευών (μνήμη, μονάδα επεξεργασίας, δικτύωση, ακροδέκτες κ.α.) και στην άλλη οι μεταξύ τους συνδέσεις (συνδέσεις ακροδεκτών, πρωτόκολλα επικοινωνίας που χρησιμοποιούνται κ.α.). Μέσω αυτών, δημιουργούνται τα κατάλληλα μοντέλα για τις συσκευές και συνδέσεις.

Από τα μοντέλα, πραγματοποιούνται δύο μετασχηματισμοί, ένας Model-to-Text (M2T) και ένας Model-to-Model (M2M). Ο M2M έχει ως αποτέλεσμα την παραγωγή διαγραμμάτων, τα οποία βοηθούν στην οπτικοποίηση της συνδεσμολογίας και ενδοεπικοινωνίας του συστήματος. Μέσω του M2T, παράγονται αυτόματα τμήματα λογισμικού που θα υλοποιούν κάποιες βασικές λειτουργίες (λήψη μετρήσεων από αισθητήρες, έλεγχος ενεργοποιητών κ.α.). Ο παραγόμενος κώδικας θα αφορά συσκευές που υποστηρίζονται από το λειτουργικό σύστημα πραγματικού χρόνου RIOT.

1.3 Διαρθρωση της Αναφορας

Η διάρθρωση της παρούσας διπλωματικής εργασίας είναι η εξής:

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

2

Θεωρητικό Υπόβαθρο - Αρχιτεκτονική

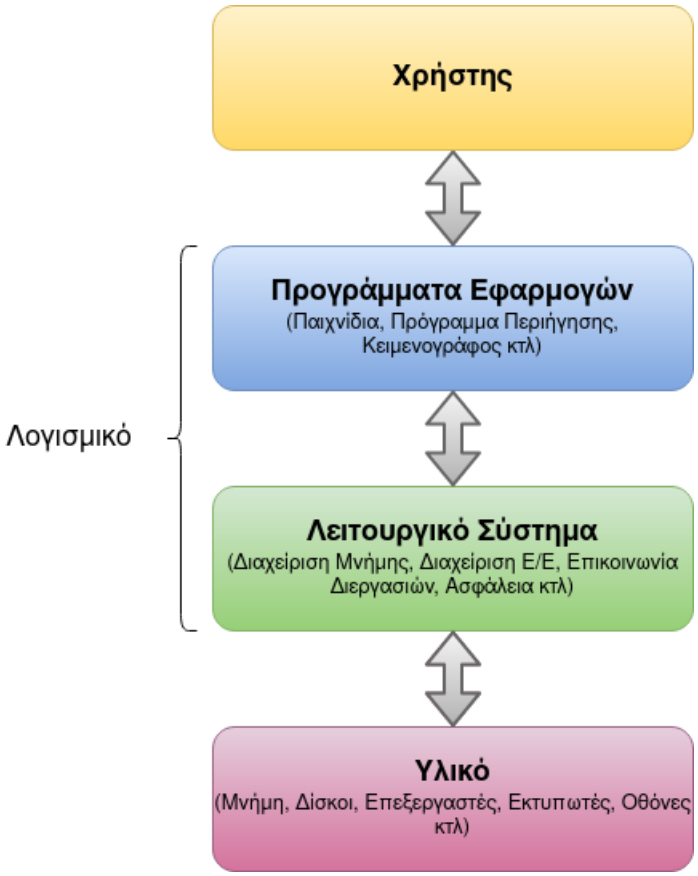
2.1 Λειτουργικα Συστηματα

Σύμφωνα με το [2]: Τα **Λειτουργικά Συστήματα** (εν συντομία **ΛΣ**) είναι προγράμματα που ελέγχουν την εκτέλεση προγραμμάτων εφαρμογών και δρουν ως διεπαφή ανάμεσα στις εφαρμογές και το υλικό του υπολογιστή. Θα λέγαμε ότι έχουν τρεις στόχους:

- **Ευκολία**: Τα ΛΣ κάνουν ευκολότερη τη χρήση ενός υπολογιστή.
- **Αποτελεσματικότητα**: Τα ΛΣ επιτρέπουν την αποτελεσματική χρήση των πόρων ενός υπολογιστικού συστήματος.
- **Ικανότητα Εξέλιξης**: Τα ΛΣ πρέπει να είναι κατασκευασμένα με τέτοιο τρόπο, ώστε να επιτρέπουν την αποτελεσματική ανάπτυξη, τον έλεγχο και την εισαγωγή νέων λειτουργιών συστήματος, χωρίς να παρεμβαίνουν στην παροχή υπηρεσιών.

2.1.1 Το ΛΣ ως Διεπαφή Χρήστη/Υπολογιστή

Ένας υπολογιστής είναι ένα σύστημα που αποτελείται από το υλικό (hardware) και το λογισμικό (software). Το υλικό είναι ο φυσικός εξοπλισμός (οθόνες, μνήμες, εκτυπωτές κτλ) ενώ το λογισμικό είναι μια συλλογή προγραμμάτων που επιτρέπουν στο υλικό να λειτουργεί [3]. Το λογισμικό διαιρείται σε 2 κατηγορίες: στα προγράμματα εφαρμογών και στα λειτουργικά συστήματα. Το υλικό και το λογι-



Σχήμα 2.1: Δομή Υλικού και Λογισμικού Υπολογιστή

σμικό που αποτελούν ένα υπολογιστικό σύστημα μπορεί να αναπαρασταθεί από μια ιεραρχική δομή ή μια δομή επιπέδων, όπως φαίνεται στο Σχήμα 2.1.

Το ΛΣ, επομένως, «κάθεται» ανάμεσα στο υλικό και στις εφαρμογές που χρησιμοποιεί ο χρήστης ο οποίος συνήθως δεν ενδιαφέρεται για λεπτομέρειες του υλικού του υπολογιστή. Δρα, δηλαδή, ως ένας διαμεσολαβητής που διευκολύνει την επικοινωνία του χρήστη με τον υπολογιστή χωρίς ο πρώτος να χρειάζεται να γνωρίζει την γλώσσα του τελευταίου, ενώ είναι υπεύθυνο για την ομαλή λειτουργία των προγραμμάτων εφαρμογών. Για παράδειγμα, προκειμένου να εμφανιστεί το κείμενο «Χαίρε Κόσμε» στην οθόνη, πρέπει μερικές εκατοντάδες πίξελ να λειτουργήσουν σε συγκεκριμένες θέσεις. Αυτό μπορεί να γίνει με το να διαβάζει κανείς τις προδιαγραφές του υλικού και να γράψει κώδικα που χειρίζεται τα σωστά μπιτς στην μνήμη κάτι που είναι αρκετά επίπονο. Οι περισσότεροι χρήστες όμως θέλουν απλά να γράψουν την εντολή `print("Hello World")` χωρίς να νοιαστούν για περαιτέρω λεπτομέρειες. Εκεί είναι που μπαίνει το ΛΣ για να δώσει λύση [4].

Πιο αναλυτικά, μερικές από τις περιοχές στις οποίες το ΛΣ παρέχει υπηρεσίες περιγράφονται παρακάτω σύμφωνα με το [2]:

- **Στην ανάπτυξη προγραμμάτων**: Το ΛΣ παρέχει πληθώρα υπηρεσιών, όπως επεξεργαστές κειμένου και διορθωτές λαθών (debuggers), προκειμένου να

βοηθήσει τον προγραμματιστή στην ανάπτυξη εφαρμογών. Συνήθως, οι υπηρεσίες αυτές έχουν τη μορφή βοηθητικών προγραμμάτων, τα οποία αν και δεν αποτελούν αυστηρά τμήμα του πυρήνα του λειτουργικού συστήματος, παρέχονται μαζί με το ΛΣ και αναφέρονται ως εργαλεία ανάπτυξης προγραμμάτων εφαρμογών.

- **Στην εκτέλεση προγραμμάτων**: Για την εκτέλεση ενός προγράμματος απαιτείται πλήθος βημάτων. Εντολές και δεδομένα πρέπει να φορτωθούν στην κύρια μνήμη, συσκευές Εισόδου/Εξόδου (Ε/Ε) και αρχεία πρέπει να αρχικοποιηθούν και διάφοροι άλλοι πόροι πρέπει να ετοιμαστούν. Το ΛΣ διαχειρίζεται αυτές τις υποχρεώσεις χρονοδρομολόγησης για λογαριασμό του χρήστη.
- **Στην πρόσβαση σε συσκευές Εισόδου/Εξόδου(Ε/Ε)**: Κάθε συσκευή Ε/Ε απαιτεί το δικό της ιδιαίτερο σύνολο εντολών ή σημάτων ελέγχου για τη λειτουργία της. Το ΛΣ παρέχει ενιαία διεπαφή που αποκρύπτει αυτές τις λεπτομέρειες, έτσι ώστε οι προγραμματιστές να έχουν πρόσβαση σε τέτοιες συσκευές χρησιμοποιώντας απλές αναγνώσεις (reads) και εγγραφές (writes).
- **Στην ελεγχόμενη πρόσβαση σε αρχεία**: Το ΛΣ πρέπει να αναπαριστά τη λεπτομερή κατανόηση, όχι μόνο της φύσης των συσκευών Ε/Ε (οδηγού δίσκου, οδηγού ταινίας), αλλά και της δομής των δεδομένων που βρίσκονται σε αρχεία στο αποθηκευτικό μέσο. Επίσης, στην περίπτωση συστημάτων πολλαπλών χρηστών, το ΛΣ μπορεί να παρέχει μηχανισμούς προστασίας για τον έλεγχο της πρόσβασης σε αρχεία.
- **Στην πρόσβαση στο σύστημα**: Σε ό,τι αφορά διαμοιραζόμενα ή δημόσια συστήματα, το ΛΣ ελέγχει συνολικά την πρόσβαση στο σύστημα και σε συγκεκριμένους πόρους του. Η λειτουργία πρόσβασης πρέπει να παρέχει προστασία των πόρων και των δεδομένων από μη εξουσιοδοτημένους χρήστες και να διευθετεί θέματα συγκρούσεων και διεκδίκησης πόρων.
- **Στην ανίχνευση σφαλμάτων και στην απόκριση**: Διάφορα σφάλματα μπορούν να προκύψουν κατά τη διάρκεια λειτουργίας ενός υπολογιστικού συστήματος. Μεταξύ αυτών περιλαμβάνονται εσωτερικά και εξωτερικά σφάλματα υλικού, όπως σφάλματα μνήμης ή δυσλειτουργία κάποιας συσκευής, καθώς και διάφορα σφάλματα λογισμικού, όπως η διαίρεση με το μηδέν και η προσπάθεια προσπέλασης απαγορευμένης θέσης μνήμης. Σε κάθε περίπτωση, το ΛΣ πρέπει να παρέχει μια απόκριση, η οποία απαλείφει τη συνθήκη σφάλματος, με το μικρότερο δυνατό αντίκτυπο στις εφαρμογές που εκείνη την ώρα εκτελούνται. Απόκριση μπορεί να αποτελεί η λήξη του προγράμματος που προκάλεσε το σφάλμα, η επανέναρξη λειτουργίας ή ακόμα και η απλή αναφορά σφάλματος στην εφαρμογή.
- **Στην λογιστική**: Ένα καλό ΛΣ συλλέγει στατιστικά χρήσης διάφορων πόρων και παρακολουθεί παραμέτρους απόδοσης, όπως είναι ο χρόνος απόκρισης. Σε κάθε σύστημα, οι πληροφορίες αυτές είναι χρήσιμες για τη λήψη αποφάσεων σχετικών με μελλοντικές αναβαθμίσεις και με τις ρυθμίσεις του συστήματος, ώστε να επιτυγχάνεται η βελτίωση της απόδοσης. Σε συστήματα πολλαπλών χρηστών, οι πληροφορίες αυτές μπορούν να χρησιμοποιηθούν και για λόγους χρέωσης.

2.1.2 Το ΛΣ ως Διαχειριστής Πόρων

Όπως αναφέρεται και παραπάνω το ΛΣ είναι μεταξύ άλλων υπεύθυνο για τον έλεγχο των πόρων του υπολογιστή, δηλαδή τις μνήμες, τους δίσκους, τον επεξεργαστή κτλ. Αναφέρθηκε, επίσης, ότι το ΛΣ είναι και αυτό ένα λογισμικό και άρα λειτουργεί με τον ίδιο τρόπο που λειτουργεί οποιαδήποτε άλλη εφαρμογή, δηλαδή είναι μια ακολουθία εντολών που εκτελούνται από τον επεξεργαστή. Κατά την φάση εκτέλεσης, το ΛΣ κατανέμει τον χρόνο του επεξεργαστή και τους πόρους του υπολογιστή στα προγράμματα που πρόκειται να τρέξουν. Για να εκτελεστεί όμως μια εφαρμογή ή ένα πρόγραμμα ο επεξεργαστής πρέπει να διακόψει την εκτέλεση του ΛΣ, ενώ μετά το πέρας των εντολών επαναφέρει το ΛΣ στον έλεγχο προκειμένου να γίνουν οι απαραίτητες προετοιμασίες για τις επόμενες εργασίες [2].

Ακριβώς επειδή το ΛΣ βοηθάει την εκτέλεση σχεδόν κάθε προγράμματος πρέπει να είναι και αρκετά αποδοτικό. Για παράδειγμα, τα προγράμματα εφαρμογών δημιουργούν αντικείμενα και πίνακες συνέχεια οπότε είναι σημαντικό αυτή η δουλειά να γίνεται γρήγορα και με εξοικονόμηση πόρων. Οποιοδήποτε κέρδος του ΛΣ, είτε σε ταχύτητα είτε σε μνήμη, μπορεί να επηρεάσει δραματικά την απόδοση του υπολογιστικού συστήματος [4].

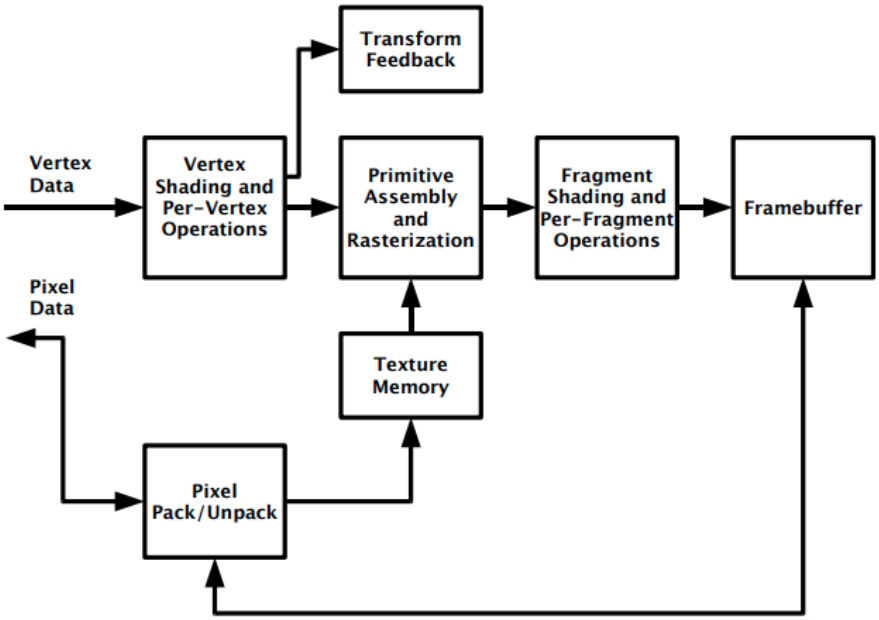
2.2 Open Graphics Library

Αντίστοιχα με το ΛΣ αλλά ανεξάρτητο από αυτό, η **Open Graphics Library**[5] (εν συντομία **OpenGL**) αποτελεί μία *Διεπαφή Προγραμματισμού Εφαρμογών* για τη βέλτιστη χρήση των πόρων της κάρτας γραφικών με σκοπό την οικονομική και γρήγορη απόδοση 2D και 3D γραφικών. Η διεπαφή περιλαμβάνει ένα σύνολο εντολών οι οποίες επιτρέπουν στον χρήστη να καθορίσει τα απαραίτητα αντικείμενα και διεργασίες για την παραγωγή υψηλής ποιότητας έγχρωμων εικόνων δισδιάστατων ή τρισδιάστατων αντικειμένων.

2.2.1 OpenGL

Η OpenGL είναι υπεύθυνη για την επεξεργασία των δεδομένων στην μνήμη της κάρτας γραφικών, την εγγραφή δεδομένων στον framebuffer και την ανάγνωση του. Ο framebuffer απαρτίζεται από ένα σύνολο pixels διατεταγμένων σε ένα δισδιάστατο πίνακα. Κάθε στοιχείο του πίνακα αποτελείται από έναν αριθμό bits, ανάλογα με την υλοποίηση της OpenGL, τα οποία καθορίζουν το χρώμα, το βάθος και το στένσιλ για κάθε pixel.

Στο Σχήμα 2.2 φαίνεται η διαδικασία απεικόνισης γραφικών της OpenGL. Αρχικά, δέχεται ως είσοδο τα δεδομένα των κορυφών των αντικειμένων προς προβολή,



Σχήμα 2.2: Διαδικασία απεικόνισης γραφικών της OpenGL

τα οποία συνθέτονται σε πρωτόγονα σχήματα όπως κορυφές, τμήματα γραμμών, επιφάνειες και πολύγωνα. Στη συνέχεια, οι κορυφές μεταμορφώνονται σε γεωμετρικά πρωτόγονα σχήματα, συνήθως τρίγωνα ή πολύγωνα, τα οποία μέσω ψηφίδωσης μπορούν να παράξουν περισσότερα πρωτόγονα σχήματα από μία είσοδο. Προαιρετικά, τα αποτελέσματα από αυτά τα στάδια δύναται να ανατροφοδοτήσουν ενδιάμεσες μνήμες για μετέπειτα χρήση.

Τα τελικά πρωτόγονα σχήματα περικόπτονται από έναν καθορισμένο όγκο για να προετοιμαστούν για το στάδιο της ψηφιοποίησης. Η διαδικασία αυτή παράγει σειρές από διευθύνσεις του framebuffer συνοδευόμενες από τιμές που περιγράφουν τη δισδιάστατη απεικόνιση των αρχικών τρισδιάστατων πρωτόγονων σχημάτων. Κάθε τμήμα που παράγεται με αυτό τον τρόπο υπόκειται σε περαιτέρω διεργασίες ξεχωριστά. Οι διεργασίες περιλαμβάνουν υπό συνθήκη ενημέρωση των τιμών σε συνάρτηση με εισερχόμενες ή αποθηκευμένες τιμές του βάθους ή του στένσιλ, ανάμειξη των εισερχόμενων χρωμάτων με αποθηκευμένα χρώματα ή άλλες λογικές πράξεις στις τιμές των τμημάτων. Τέλος, ο framebuffer επικαιροποιείται με τις τιμές των τμημάτων, προβάλλοντας έτσι την τελική εικόνα στην οθόνη.

2.3 Διαδικτυο των Πραγματων

Το **Διαδίκτυο των Πραγμάτων** (μτφ. **Internet of Things**, εν συντομία **IoT**) είναι ένα σύνολο φυσικών αντικειμένων τα οποία έχουν αισθητήρες, λογισμικό, υπολογιστική ισχύ κτλ. και τα οποία ανταλλάσσουν δεδομένα μεταξύ τους μέσω του

Διαδικτύου ή ενός οποιουδήποτε άλλου δικτύου επικοινωνίας. Τα αντικείμενα αυτά μπορεί να είναι ένας άνθρωπος με βηματοδότη στην καρδιά, ένα έξυπνο ρολόι που μετράει παλμούς, ένας καθρέφτης που διαβάζει δεδομένα από αισθητήρες και εμφανίζει πληροφορίες για την κατάσταση του χρήστη.

2.3.1 Πώς δουλεύει το IoT

Ένα σύστημα IoT περιέχει πολλές φυσικές μονάδες (hardware) οι οποίες μπορούν να χωριστούν στις ακόλουθες κατηγορίες [6]:

- Αισθητήρες & Ενεργοποιητές (Sensors & Actuators)
- Μονάδες Επεξεργασίας (Processing Units)
- Μονάδες Αποθήκευσης (Storage Units)
- Μονάδες Επικοινωνίας (Communication Units)

Σε ό,τι αφορά το λογισμικό των IoT συσκευών, δεν υπάρχει κάποια συγκεκριμένη αρχιτεκτονική που να ακολουθείται καθολικά. Έχουν προταθεί αρκετές, ενώ παρακάτω παρουσιάζονται 2 από τις πιο διαδεδομένες, η αρχιτεκτονική 3 επιπέδων και η αρχιτεκτονική 5 επιπέδων [7].

Αρχιτεκτονική 3 επιπέδων

Η αρχιτεκτονική 3 επιπέδων είναι από τις πιο βασικές και πρώιμες αρχιτεκτονικές που αναπτύχθηκαν και απεικονίζεται στο Σχήμα 2.3.

Όπως προκύπτει και από το όνομα αποτελείται από 3 επίπεδα:

- **Επίπεδο Αντίληψης**: Το επίπεδο αντίληψης είναι το φυσικό επίπεδο (hardware) το οποίο περιέχει αισθητήρες για λήψη πληροφοριών και ανίχνευση παραμέτρων από τον περιβάλλοντα χώρο.
- **Επίπεδο Δικτύου**: Το επίπεδο δικτύου είναι υπεύθυνο για την διασύνδεση με άλλες συσκευές, έξυπνα πράγματα κτλ. Χρησιμοποιείται επίσης για την μετάδοση και επεξεργασία δεδομένων των αισθητήρων.
- **Επίπεδο Εφαρμογής**: Το επίπεδο εφαρμογής είναι υπεύθυνο για την παροχή υπηρεσιών συγκεκριμένης εφαρμογής (application specific services) στον χρήστη. Καθορίζει τις διάφορες εφαρμογές στις οποίες μπορεί να αναπτυχθεί το IoT, όπως το έξυπνο σπίτι, ο έξυπνος καθρέφτης κ.α.

Αρχιτεκτονική 5 επιπέδων

Η αρχιτεκτονική των 3 επιπέδων περιγράφει την βασική ιδέα πίσω από το IoT, αλλά στην πράξη δεν επαρκή. Για το λόγο αυτό, υπάρχουν αρχιτεκτονικές με πε-

Σχήμα 2.3: Αρχιτεκτονική 3 Επιπέδων

ρισσότερα επίπεδα στην βιβλιογραφία. Μία από αυτές είναι η αρχιτεκτονική 5 επιπέδων που φαίνεται στο Σχήμα 2.4.

Στην περίπτωση αυτή, τα επίπεδα Αντίληψης και Εφαρμογής παραμένουν ίδια με αυτά της αρχιτεκτονικής 3 επιπέδων ενώ επεξηγούνται και τα υπόλοιπα 3 επίπεδα:

- **Επίπεδο Μεταφοράς**: Το επίπεδο μεταφοράς, μεταδίδει τα δεδομένα των αισθητήρων από το επίπεδο αντίληψης στο επίπεδο επεξεργασίας και αντίστροφα μέσω δικτύων όπως 3G, LAN, RFID, NFC κτλ.

- **Επίπεδο Επεξεργασίας**: Το επίπεδο επεξεργασίας, γνωστό και ως middleware, αποθηκεύει, αναλύει και επεξεργάζεται μεγάλο όγκο δεδομένων που προέρχονται από το επίπεδο μεταφοράς. Προσφέρει μια ευρεία γκάμα υπηρεσιών στα χαμηλότερα επίπεδα. Χρησιμοποιεί επίσης πολλές τεχνολογίες όπως βάσεις δεδομένων, υπολογιστική νέφους και ενότητες επεξεργασίας μεγάλων δεδομένων.

- **Επίπεδο Επιχείρησης**: Το επίπεδο επιχείρησης διαχειρίζεται όλο το IoT σύστημα, συμπεριλαμβανομένων των εφαρμογών του, των επιχειρηματικών του μοντέλων και της ιδιωτικότητας των χρηστών.

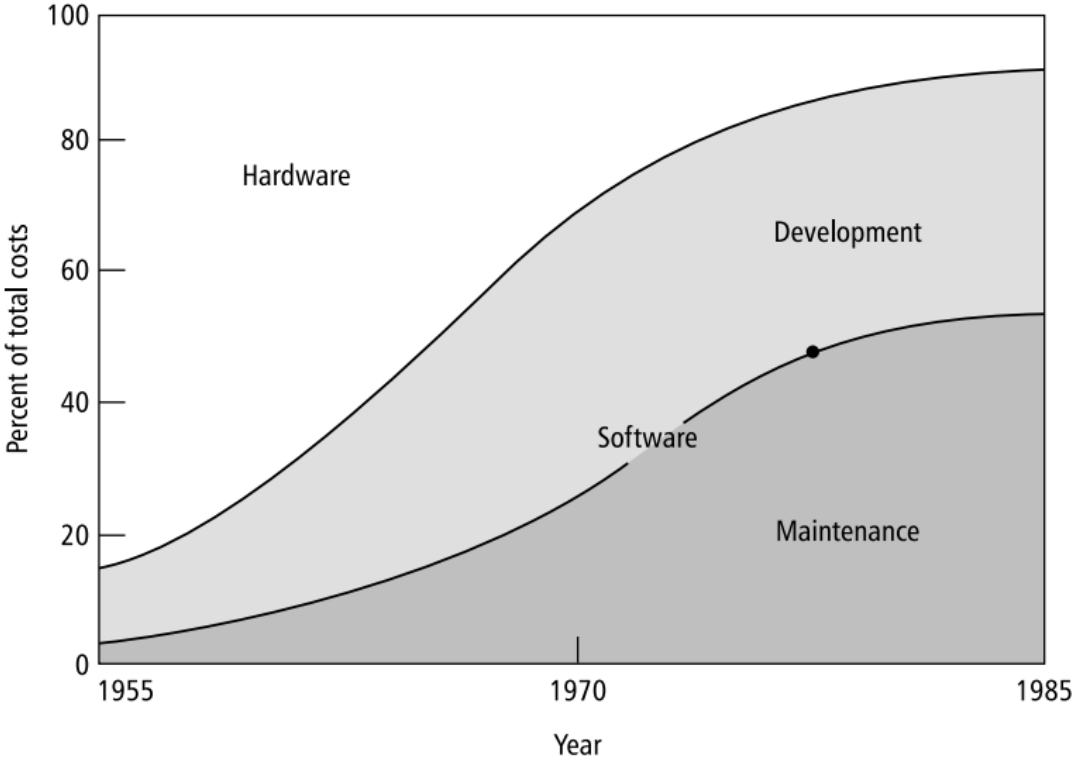
Σχήμα 2.4: Αρχιτεκτονική 5 Επιπέδων

2.4 Ανάπτυξη Λογισμικού

Στη σύγχρονη πραγματικότητα, το λογισμικό έχει κυριαρχήσει στην καθημερινή ζωή, καθιστώντας το μία από τις σημαντικότερες τεχνολογίες. Παίζοντας ταυτόχρονα το ρόλο προϊόντος και εργαλείου, αποτελεί τη βάση της επιστημονικής έρευνας και της επίλυσης προβλημάτων μηχανικής, καθιστώντας δυνατή την δημιουργία καινούργιων και την επέκταση υπαρχουσών τεχνολογιών (π.χ. γενετική μηχανική και τηλεπικοινωνίες αντίστοιχα). Ταυτόχρονα, έχει διεισδύσει σε συστήματα κάθε είδους: μεταφοράς, ιατρικά, τηλεπικοινωνιών, στρατιωτικά, βιομηχανικά, ψυχαγωγίας κ.α. αλλάζοντας τον τρόπο που αντιλαμβανόμαστε και αλληλεπιδράμε με τον κόσμο.

Έτσι, τα λογισμικά προγράμματα αντιμετωπίζουν ολοένα και περισσότερα προβλήματα της καθημερινής ζωής, αυξάνοντας την αναγκαιότητα και το κόστος τους. Πράγματι, το συνολικό κόστος λογισμικού υπολογίζεται στα €500 δισεκατομμύρια

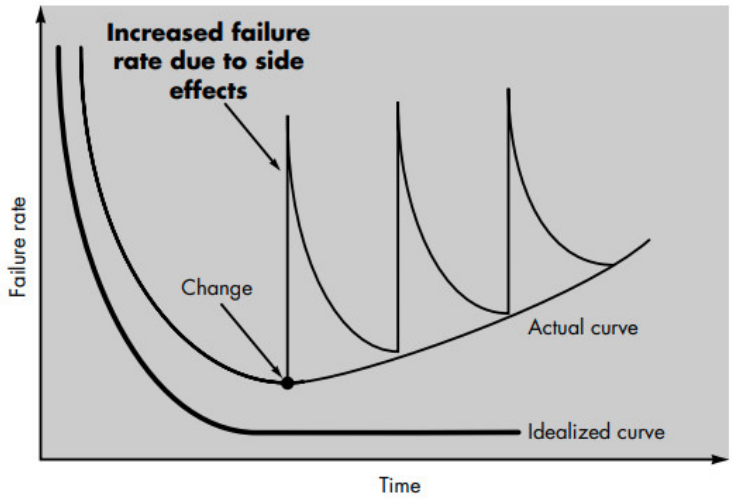
στην Αμερική και διπλάσιο παγκοσμίως [8]. Αυτό αναφέρεται τόσο στο κόστος ανάπτυξης του λογισμικού όσο και συντήρησής του αφού έχει παραδοθεί στον πελάτη. Ταυτόχρονα, το κόστος του υλικού έχει μειωθεί δραματικά, αποτελώντας λιγότερο από το 20% των συνολικών εξόδων ενός συστήματος όπως φαίνεται στο Σχήμα 2.5.



Σχήμα 2.5: Σχετική κατανομή κόστους υλικού/λογισμικού (*Πηγή: [1]*)

Η διαφορά κόστους έγκειται στο γεγονός ότι τα σύγχρονα προγράμματα λογισμικού είναι μεγάλα και περίπλοκα, απαιτώντας ομάδες υψηλής εξειδίκευσης, δεν έχουν περιορισμούς (δηλαδή έχουν περισσότερους βαθμούς ελευθερίας) και τέλος επειδή υφίστανται συνεχόμενες αλλαγές. Μάλιστα, στη διάρκεια ζωής του λογισμικού οι αλλαγές αυτές ενδέχεται να εισάγουν σφάλματα, αυξάνοντας τον κίνδυνο αποτυχίας του συστήματος, όπως φαίνεται στο Σχήμα 2.6. Επιπλέον, σε αντίθεση με την αποτυχία του υλικού, όπου αντιμετωπίζεται με αντικατάσταση του χαλασμένου μέρους, η αποτυχία του λογισμικού έγκειται σε σφάλμα σχεδιασμού, καθιστώντας τη συντήρηση του σημαντικά πιο περίπλοκη και ακριβή διαδικασία.

Εντούτοις, η αυξανόμενη εξάρτηση των καθημερινών δραστηριοτήτων από περίπλοκα συστήματα λογισμικού απαιτεί την εύρωστη και αλάνθαστη λειτουργία τους καθ' όλη τη διάρκεια ζωής του λογισμικού, το οποίο όμως αλλάζει συνεχώς αυξάνοντας τον κίνδυνο αποτυχίας του. Τοιουτοτρόπως, για την μείωση του ρίσκου αποτυχίας και την απλούστευση της ανάπτυξης και συντήρησης υψηλής ποιότητας έργων λογισμικού, έπρεπε να τυποποιηθούν οι μέθοδοι και διαδικασίες που διέπουν ολόκληρο τον κύκλο ζωής του. Αυτές οι μεθοδολογίες εμπεριέχονται και αναλύονται από την επιστήμη της τεχνολογίας λογισμικού.



Σχήμα 2.6: Καμπύλη αποτυχίας για το λογισμικό

2.4.1 Τεχνολογία Λογισμικού

Η τεχνολογία λογισμικού είναι η συστηματική, πειθαρχημένη και μετρήσιμη προσέγγιση με σκοπό την ανάπτυξη, λειτουργία και συντήρηση υψηλής ποιότητας έργων λογισμικού. Οι διαδικασίες της τεχνολογίας λογισμικού παρέχουν ένα ευρύτερο πλαίσιο, μέσα στο οποίο εφαρμόζονται οι τεχνικές μέθοδοι, παράγονται προϊόντα δουλειάς (μοντέλα, έγγραφα, δεδομένα, αναφορές κ.α.), εξακριβώνονται σημεία σταθμοί, εξασφαλίζεται η ποιότητα και γίνεται κατάλληλη διαχείριση των αλλαγών των απαιτήσεων.

Οι μέθοδοι της τεχνολογίας λογισμικού είναι το τεχνικό εγχειρίδιο για την ανάπτυξη λογισμικού. Οι μέθοδοι εγκολπώνουν ένα ευρύ σύνολο λειτουργιών συμπεριλαμβανομένων την ανάλυση απαιτήσεων, σχεδιασμού, δημιουργία προγράμματος, δοκιμής και υποστήριξης. Οι μέθοδοι της τεχνολογίας λογισμικού βασίζονται σε βασικές αρχές και πρακτικές που διέπουν κάθε τομέα της τεχνολογίας και περιλαμβάνουν δραστηριότητες μοντελοποίησης και άλλες περιγραφικές τεχνικές.

Πλαίσιο διαδικασίας

Το πλαίσιο διαδικασίας της τεχνολογίας λογισμικού απαρτίζεται από δραστηριότητες που είναι κοινές για οποιοδήποτε έργο λογισμικού ανεξάρτητα από το μέγεθος και την περιπλοκότητα του. Ένα γενικό πλαίσιο διαδικασίας περιλαμβάνει πέντε κύριες δραστηριότητες [9]:

- **Επικοινωνία** (Communication): Πριν την έναρξη οποιασδήποτε τεχνικής εργασίας, είναι άκρως σημαντική η επικοινωνία και συνεργασία με τον πελάτη (και άλλα ενδιαφερόμενα μέλη). Η πρόθεση είναι η κατανόηση των στόχων των ενδιαφερόμενων μελών για το έργο λογισμικού και η συλλογή απαιτή-

σεων, βοηθώντας τον καθορισμό των λειτουργιών και των χαρακτηριστικών του λογισμικού.

- **Προγραμματισμός** (Planning): Ένα έργο λογισμικού είναι ένα περίπλοκο ταξίδι και η διαδικασία του προγραμματισμού είναι ο χάρτης ο οποίος οδηγεί την ομάδα ανάπτυξης. Το σχέδιο του έργου αποσαφηνίζει την απαιτούμενη δουλειά, περιγράφοντας τις τεχνικές εργασίες προς διεκπεραίωση, τα πιθανά ρίσκα, τους απαιτούμενους πόρους, τα προϊόντα προς παραγωγή και το πρόγραμμα εργασίας.
- **Μοντελοποίηση** (Modelling): Δημιουργούνται μοντέλα ώστε να γίνει εφικτή η καλύτερη κατανόηση των απαιτήσεων του λογισμικού και του σχεδιασμού ο οποίος δύναται να ικανοποιήσει αυτές τις απαιτήσεις. Γίνεται εμφανής η αρχιτεκτονική μορφή του έργου, ο τρόπος αλληλεξάρτησης των μεμονωμένων μερών και πολλά άλλα χαρακτηριστικά, στη προσπάθεια κατανόησης του προβλήματος και του τρόπου επίλυσης του.
- **Κατασκευή** (Construction): Η δραστηριότητα αυτή περιλαμβάνει την υλοποίηση του έργου παράγοντας κώδικα (είτε με χειροκίνητο ή αυτόματο τρόπο) και τον έλεγχο του κώδικα για την ανίχνευση σφαλμάτων.
- **Εγκατάσταση** (Deployment): Το λογισμικό ως ολόκληρη οντότητα ή σε επιμέρους διαδοχικά στάδια παραδίνεται στον πελάτη ο οποίος αξιολογεί το παραδοθέν προϊόν και παρέχει ανατροφοδότηση.

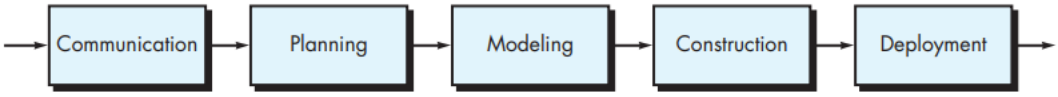
Σε πολλά έργα λογισμικού, οι δραστηριότητες πλαισίου εφαρμόζονται επαναληπτικά καθώς αναπτύσσεται το έργο. Κάθε επανάληψη παράγει μία επαύξηση του λογισμικού η οποία θέτει στην διάθεση των ενδιαφερόμενων μελών ένα υποσύνολο των τελικών χαρακτηριστικών και λειτουργιών του λογισμικού. Τοιουτοτρόπως, το λογισμικό ολοκληρώνεται σταδιακά με το πέρας κάθε επανάληψης.

2.4.2 Ροές διαδικασίας Τεχνολογίας Λογισμικού

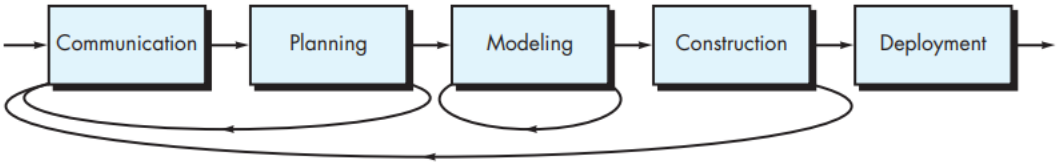
Μία διαδικασία ορίζεται ως το άθροισμα των δραστηριοτήτων, ενεργειών και εργασιών που πρέπει να διεκπεραιωθούν με σκοπό την παραγωγή ενός προϊόντος εργασίας. Όπως αναφέρθηκε παραπάνω, το γενικό πλαίσιο διαδικασίας του λογισμικού αποτελείται από πέντε δραστηριότητες σε συνδυασμό με άλλες ενέργειες - όπως παρακολούθηση και έλεγχος έργου, εκτίμηση και διαχείριση ρίσκου, διασφάλιση ποιότητας, ρύθμιση παραμέτρων κ.α. - που εφαρμόζονται καθ' όλη την διαδικασία.

Η ροή διαδικασίας περιγράφει τον τρόπο με τον οποίο οι δραστηριότητες πλαισίου, οι ενέργειες και οι εργασίες εντός κάθε δραστηριότητας οργανώνονται σε σχέση με την αλληλουχία τους και τον χρόνο όπως φαίνεται στο Σχήμα 2.7.

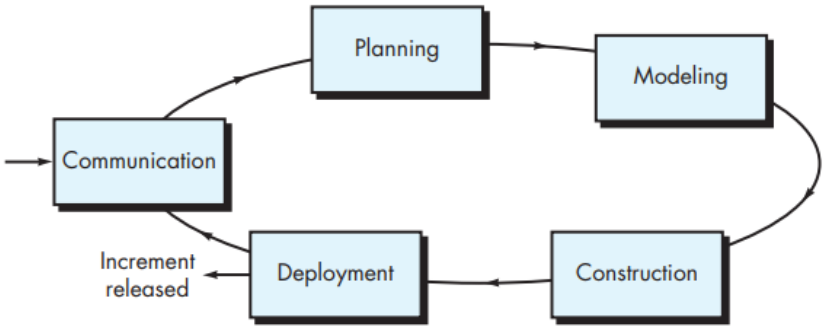
Η γραμμική ροή εκτελεί τις δραστηριότητες διαδοχικά, ξεκινώντας την κάθε δραστηριότητα με το πέρας της προηγούμενης (2.7α΄). Η επαναληπτική ροή επαναλαμβάνει μία ή περισσότερες δραστηριότητες πριν προχωρήσει στην επόμενη



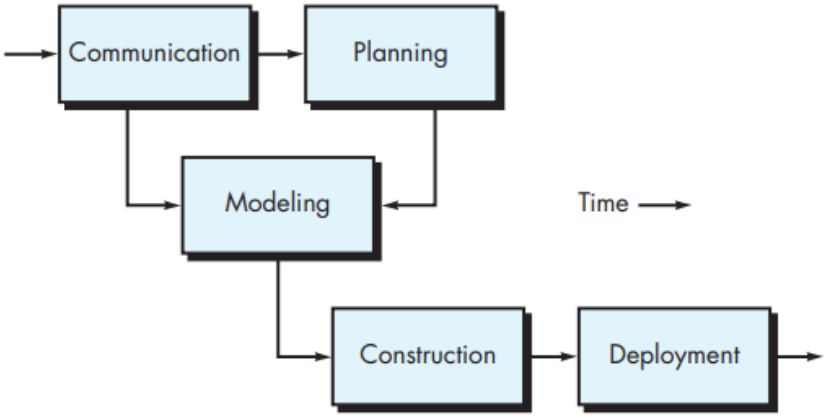
(α΄) Γραμμική ροή



(β΄) Επαναληπτική ροή



(γ΄) Εξελικτική ροή



(δ΄) Παράλληλή ροή

Σχήμα 2.7: Ροές διαδικασίας λογισμικού

(2.7β΄).Η εξελικτική ροή εκτελεί τις δραστηριότητες κυκλικά. Στο πέρας της διεκπεραίωσης των δραστηριοτήτων παράγεται σταδιακά η εφαρμογή ολοένα και πιο ολοκληρωμένη (2.7γ΄). Η παράλληλη ροή (2.7δ΄) εκτελεί μία ή περισσότερες δραστηριότητες παράλληλα με άλλες δραστηριότητες (για παράδειγμα η μοντελοποίηση μιας πτυχής του λογισμικού μπορεί να γίνεται ταυτόχρονα με την κατασκευή μιας άλλης πτυχής του).

2.4.3 Μοντέλα ανάπτυξης λογισμικού

Μοντέλο Καταρράκτη

Το μοντέλο ανάπτυξης *καταρράκτη* ακολουθεί την γραμμική ροή, δηλαδή οι δραστηριότητες από την *επικοινωνία* μέχρι την *εγκατάσταση* διεκπεραιώνονται διαδοχικά. Το μοντέλο αυτό επιλέγεται όταν οι απαιτήσεις του λογισμικού είναι ξεκάθαρα ορισμένες και επαρκώς σταθερές.

Το μοντέλο *καταρράκτη* είναι το παλαιότερο μοντέλο της τεχνολογίας λογισμικού. Εντούτοις, με την εξέλιξη της τεχνολογίας και των απαιτήσεων έχει παρουσιάσει πληθώρα προβλημάτων.

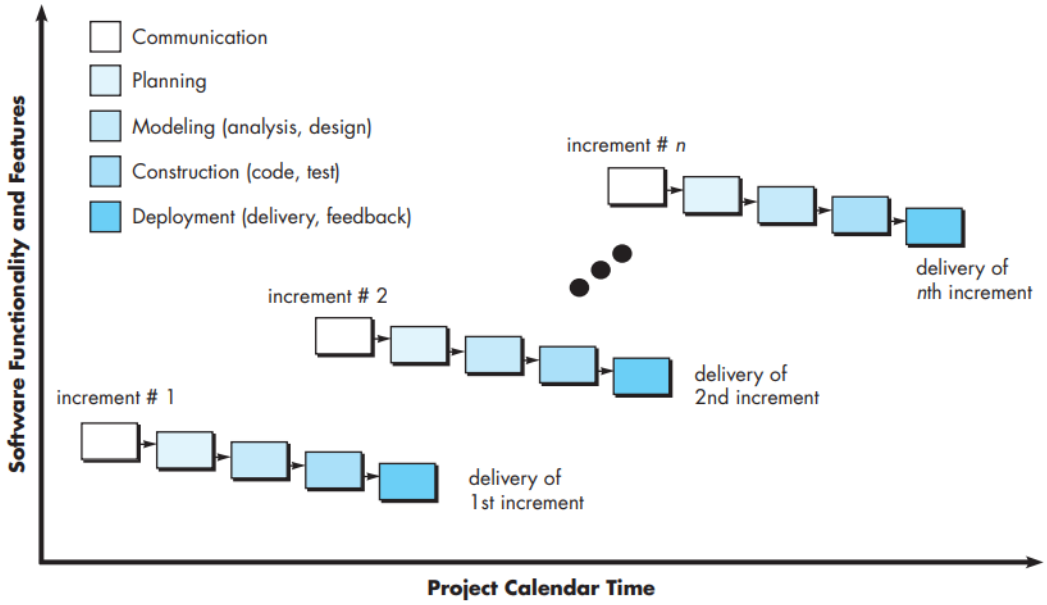
- Τα έργα λογισμικού σπάνια ακολουθούν την προτεινόμενη γραμμική ροή. Τοιουτοτρόπως, οι αλλαγές δεν είναι εύκολα διαχειρίσιμες και μπορεί να προκαλέσουν σύγχυση.
- Συνήθως είναι δύσκολο για τον πελάτη να καθορίσει όλες τις απαιτήσεις αναλυτικά. Το μοντέλο *καταρράκτη* όμως το απαιτεί και έτσι δεν μπορεί να διαχειριστεί την αβεβαιότητα που έγκειται στην έναρξη οποιουδήποτε έργου λογισμικού.
- Μία λειτουργική έκδοση του λογισμικού γίνεται διαθέσιμη αρκετά αργά στην διάρκεια ζωής του έργου. Έτσι, από την μία ο πελάτης πρέπει να είναι υπομονετικός και από την άλλη μία αστοχία σχεδιασμού που θα ανιχνευτεί κατά την αξιολόγηση της λειτουργικής έκδοσης μπορεί να είναι καταστροφική.

Αυξητικό Μοντέλο

Το *αυξητικό* μοντέλο συνδυάζει την γραμμική και την παράλληλη ροή διαδικασιών. Όπως φαίνεται στο Σχήμα 2.8, το αυξητικό μοντέλο χωρίζει τις λειτουργίες και τα χαρακτηριστικά του λογισμικού σε μικρές επαυξήσεις και για κάθε επαύξηση διεκπεραιώνει γραμμικά τις δραστηριότητες, στο πέρας των οποίων παραδίδεται η λειτουργία του λογισμικού.

Οι πρώτες επαυξήσεις υλοποιούν τις κυριότερες λειτουργίες του λογισμικού, ικανοποιώντας τις αρχικές απαιτήσεις. Το προϊόν παραδίδεται στον πελάτη ή υπόκειται σε λεπτομερή αξιολόγηση. Τοιουτοτρόπως, αναπτύσσεται ένα σχέδιο για τις επόμενες επαυξήσεις. Το σχέδιο αφορά τις αλλαγές του κύριου προϊόντος, με σκοπό την καλύτερη ικανοποίηση των απαιτήσεων του πελάτη, και την ανάπτυξη επιπλέον χαρακτηριστικών και λειτουργιών. Η διαδικασία επαναλαμβάνεται μετά από την παράδοση κάθε επαύξησης έως ότου ολοκληρωθεί το λογισμικό.

Το αυξητικό μοντέλο είναι ιδανικό όταν υπάρχουν κάποιες πλήρης και καλά καθορισμένες απαιτήσεις με χαλαρά συζευγμένα μέρη. Έτσι, οι αρχικές επαυξήσεις μπορούν να προγραμματιστούν και να παραλληλοποιηθούν επαρκώς.



Σχήμα 2.8: Αυξητικό Μοντέλο

Σπειροειδές Μοντέλο

Το *σπειροειδές* μοντέλο είναι ένα εξελικτικό μοντέλο με κύριο γνώμονα το ρίσκο. Όπως και στο *αυξητικό* μοντέλο το λογισμικό παραδίδεται σε επαναλήψεις, όμως σε αντίθεση με αυτό τα βήματα δεν είναι δραστηριότητες αλλά φάσεις, με σκοπό την αντιμετώπιση του προβλήματος με το μεγαλύτερο ρίσκο να προκαλέσει αποτυχία.

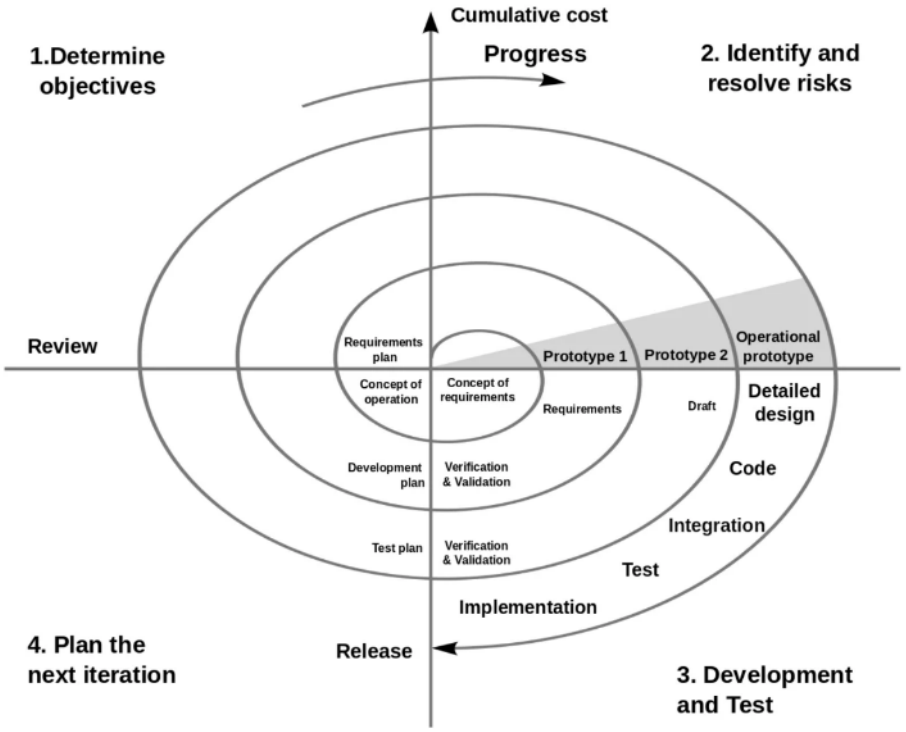
Όπως φαίνεται στο Σχήμα 2.9, οι φάσεις σε κάθε επανάληψη είναι:

1. Εξακρίβωση του προβλήματος με το μεγαλύτερο ρίσκο, καθορισμός των στόχων και εναλλακτικών λύσεων.
2. Αξιολόγηση των εναλλακτικών και προσδιορισμός των ρίσκων, σχεδιασμός των πιθανών λύσεων.
3. Ανάπτυξη μίας λύσης, επιβεβαίωση καταλληλότητας και δοκιμή της
4. Προετοιμασία για την επόμενη επανάληψη με βάση την ανατροφοδότηση από την παραδοθείσα επαύξηση.

Το σπειροειδής μοντέλο μπορεί να διαχειριστεί την αβεβαιότητα εξαιρετικά καλά. Έτσι, είναι ιδανικό για έργα λογισμικού με ασαφή απαιτήσεις και για έργα που βρίσκονται στην έρευνα και ανάπτυξη.

Ευέλικτο Μοντέλο

Στη σύγχρονη πραγματικότητα, η εργασία στην παραγωγή λογισμικού έχει ταχύς ρυθμούς και είναι υποκείμενο σε συνεχή ροή αλλαγών (χαρακτηριστικών, λει-



Σχήμα 2.9: Σπειροειδές Μοντέλο

τουργιών και πληροφοριακού περιεχόμενου). Έτσι, το *ευέλικτο* μοντέλο αποτελεί μία λογική εναλλακτική στις παραδοσιακές μεθόδους της τεχνολογίας λογισμικού.

Το μοντέλο ενθαρρύνει συνεχόμενες επαναλήψεις από ανάπτυξη και δοκιμή των λειτουργιών. Σε κάθε επανάληψη παράγεται μία επαύξηση του λογισμικού αποτελούμενη από ένα μικρό σύνολο από λειτουργίες πλήρως ολοκληρωμένες. Επίσης, η κάθε επανάληψη είναι σχεδιασμένη ώστε να είναι μικρή, διαχειρίσιμη και ολοκληρώσιμη σε λίγες εβδομάδες. Συμπεριλαμβάνει των πελάτη στην διαδικασία της ανάπτυξης και ελαχιστοποιεί τα έγγραφα χρησιμοποιώντας ανεπίσημη επικοινωνία.

Το ευέλικτο μοντέλο αν και αποτελεί ρεαλιστική προσέγγιση στην ανάπτυξη λογισμικού, βρίσκεται σε μειονεκτική θέση όταν το λογισμικό είναι σύνθετο. Επίσης, αδυνατεί να διαχειριστεί τις μεταβιβάσεις λόγω των λίγων εγγράφων, αλλά από την άλλη μπορεί να διαχειριστεί τις μεταβαλλόμενες απαιτήσεις.

Οι πιο συνηθισμένες ευέλικτες μεθοδολογίες είναι:

- **Scrum**: Αποτελείται από επαναλήψεις που λέγονται sprints. Κάθε sprint διαρκεί 2 με 4 εβδομάδες και πριν την έναρξη του γίνεται προγραμματισμός. Αφού οριστούν οι δραστηριότητες και οι εργασίες για το sprint δεν μεταβάλλονται κατά την διάρκειά του.

- **Extreme Programming (XP)**: Σε αυτό το μοντέλο η επανάληψη διαρκεί 1 με 2 εβδομάδες. Χαρακτηριστικά αυτού του μοντέλου είναι ο προγραμματισμός σε ζεύγη, η ανάπτυξη οδηγούμενη από έλεγχο (test-driven development), ο αυτοματοποιημένος έλεγχος, η απλή σχεδίασμη λογισμικού και τέλος οι μικρές εκδόσεις με συνεχόμενη ενσωμάτωση στο σύστημα.
- **Kanban**: Το μοντέλο Kanban επικεντρώνεται στην οπτικοποίηση, και αν υπάρχουν επαναλήψεις είναι πολύ μικρές. Στο πίνακα Kanban παριστάνονται όλες οι δραστηριότητες και εργασίες του έργου, συνοδευόμενες από τα υπεύθυνα άτομα για την διεκπεραίωσή τους και την πρόοδό τους.

2.4.4 Σχεδιαστικά Πρότυπα

Πράγματι, οι σύγχρονες επιχειρήσεις στρέφονται ολοένα και περισσότερο προς τα *ευέλικτα μοντέλα* ανάπτυξης λογισμικού για την διαχείριση των μεταβαλλόμενων απαιτήσεων. Εντούτοις, τα *ευέλικτα μοντέλα* απαιτούν απλό σχεδιασμό του λογισμικού έτσι ώστε να μπορούν να επεκταθούν. Προς επίτευξη αυτού του σκοπού, καθίσταται επιτακτική η ανάγκη αναζήτησης και εφαρμογής σχεδιαστικών προτύπων.

Ο σκοπός των σχεδιαστικών προτύπων είναι να παρέχουν γενικές λύσεις σε συνηθισμένα προβλήματα του σχεδιασμού λογισμικού. Τυποποιώντας επίσημα τις λύσεις και τις σχέσεις μεταξύ τους καθίσταται

2.5 Εκτιμηση Ποζας

Η εκτίμηση πόζας είναι μία διεργασία της υπολογιστικής όρασης όπου εκτιμάται η πόζα ενός αντικειμένου ή ανθρώπου σε μία εικόνα ή βίντεο. Το πρόβλημα της εκτίμησης πόζας περιλαμβάνει, επίσης, τον καθορισμό της θέσης και του προσανατολισμού της κάμερας σε σχέση με το αντικείμενο ή τον άνθρωπο.

Συνήθως αυτό γίνεται με την αναγνώριση, εκτίμηση θέσης και παρακολούθησης ενός αριθμού σημείων κλειδιών του αντικειμένου ή του ανθρώπου. Για τα αντικείμενα, αυτά μπορεί να είναι γωνίες ή άλλα σημαντικά χαρακτηριστικά ενώ για τον άνθρωπο τα σημεία κλειδιά αναπαριστούν κύριες αρθρώσεις όπως οι αγκώνες ή τα γόνατα.

Χαρακτηριστική είναι η διάκριση μεταξύ της 2D και της 3D εκτίμησης πόζας. Στην δισδιάστατη εκτίμηση πόζας εκτιμούνται οι θέσεις των σημείων στο 2D χώρο σε σχέση με το πλαίσιο της εικόνας ή του βίντεο. Αντιθέτως, η 3D εκτίμηση πόζας προβλέπει τις συντεταγμένες των σημείων κλειδιών σε ένα τρισδιάστατο σύστημα συντεταγμένων όπως φαίνεται στο Σχήμα 2.10.

(α΄) 2D σημεία με συντεταγμένες (x_i, y_i) (β΄) 3D σημεία με συντεταγμένες (x_i, y_i, z_i)

Σχήμα 2.10: Παράδειγμα εκτίμησης πόζας

Επιπλέον, η εκτίμηση πόζας διαφοροποιείται ως προς την αναγνώριση ενός ή περισσότερων αντικειμένων. Οι δύο προσεγγίσεις αποκαλούνται μεμονωμένη εκτίμηση πόζας, όπου αναγνωρίζεται και παρακολουθείται μόνο ένα αντικείμενο ή άνθρωπος, και πολλαπλή εκτίμηση πόζας, όπου αναγνωρίζονται και παρακολουθούνται πολλαπλά.

2.5.1 Μεθοδολογίες εκτίμησης πόζας

Το πρόβλημα της εκτίμηση πόζας μπορεί να λυθεί με ποικίλους τρόπους αναλόγως με την διαρρύθμιση των αισθητήρων εικόνας και την επιλογή της μεθοδολογίας. Διακρίνονται τρεις κλάσεις μεθοδολογιών [10]:

- **Αναλυτικές ή γεωμετρικές μέθοδοι**: Απαιτούν την ρύθμιση της κάμερας ώστε η αντιστοίχηση των τρισδιάστατων σημείων της σκηνής και των δισδιάστατων σημείων της εικόνας να είναι γνωστή. Τοιουτοτρόπως, γνωρίζοντας την γεωμετρία του αντικειμένου η προβολή του στην εικόνα της κάμερας είναι μια γνωστή συνάρτηση της πόζας του αντικειμένου. Έτσι, το πρόβλημα της εκτίμησης πόζας λύνεται αναγνωρίζοντας τα σημεία κλειδιά και στην συνέχεια λύνοντας το σύνολο των εξισώσεων που αντιστοιχίζουν τις τρισδιάστατες συντεταγμένες των σημείων με τις δισδιάστατες συντεταγμένες της εικόνας.
- **Γενετικοί αλγόριθμοι**: Στην περίπτωση που η πόζα του αντικειμένου ή του ανθρώπου δεν απαιτεί τον υπολογισμό της σε πραγματικό χρόνο μπορεί να χρησιμοποιηθεί ένας *γενετικός αλγόριθμος*. Η πόζα περιγράφεται από μεταβλητές, όπως για παράδειγμα οι περιστροφές των αρθρώσεων από μία στάση αναφοράς, οι οποίες χρησιμοποιούνται ως παράμετροι εισόδου του *γενετικού*, ορίζοντας έτσι τη συνάρτηση καταλληλότητας (fitness function) ως το σφάλμα της προβολής των εκτιμώμενων τρισδιάστατων σημείων κλειδιών από τις πραγματικές συντεταγμένες τους στο δισδιάστατο πλαίσιο της εικόνας.
- **Μηχανική μάθηση**: Οι μέθοδοι *μηχανικής μάθησης* χρησιμοποιούν ένα σύστημα τεχνητής νοημοσύνης το οποίο μαθαίνει την αντιστοίχηση των 2D χαρακτηριστικών της εικόνας με τις παραμέτρους μοντελοποίησης που περιγράφουν με σαφήνεια την πόζα. Εν συντομία, ένα επαρκώς μεγάλο σύνολο

από φωτογραφίες ή βίντεο αντικειμένων ή ανθρώπων σε διαφορετικές πόζες χρησιμοποιείται ως είσοδος στο σύστημα κατά την διάρκεια της φάσης εκμάθησης. Με το πέρας της εκμάθησης, το σύστημα μπορεί να εκτιμήσει την πόζα του αντικειμένου ή ανθρώπου δεδομένης της 2D εικόνας του.

Στη συγκεκριμένη εφαρμογή, απαιτείται η εκτίμηση της ανθρώπινης πόζας στον τρισδιάστατο χώρο σε πραγματικό χρόνο. Το πρόβλημα αυτό κρίνεται άκρως περίπλοκο, με την δυσκολία του να έγκειται στο γεγονός ότι η εκτίμηση της τρισδιάστατης πόζας από μία δισδιάστατη εικόνα είναι εξ ορισμού κακώς ορισμένο πρόβλημα. Αυτό γίνεται εμφανές αν σκεφτούμε ότι μία δισδιάστατη πόζα μπορεί να προκύψει από πολυάριθμες τρισδιάστατες πόζες. Ταυτόχρονα, υπάρχουν πολλές επιπλέον προκλήσεις όπως η αβεβαιότητα του περιβάλλοντος, του φόντου, του φωτισμού, η κίνηση της κάμερας, οι γρήγορες κινήσεις, οι μεταβολές των ρούχων και των σκιών που μπορεί να κάνουν την μορφή και το σχήμα του ανθρώπου να αλλάζει δραματικά με την πάροδο του χρόνου. Επιπλέον, οι ανθρώπινες αρθρώσεις είναι μικρές, οριακά εμφανής και έχουν πολλούς βαθμούς ελευθερίας δυσχεραίνοντας ακόμα περισσότερο το πρόβλημα της εκτίμησης πόζας. Ως εκ τούτου, η καταλληλότερη μεθοδολογία για την επίλυση του προβλήματος είναι η *Βαθιά Μηχανική Μάθηση*, ικανοποιώντας τόσο την απαίτηση της σχετικά ακριβής εκτίμησης των σημείων κλειδιών αλλά και διεξάγοντας την εκτίμηση σε πραγματικό χρόνο. Στη συνέχεια του κεφαλαίου λοιπόν, θα αναλύσουμε το θεωρητικό υπόβαθρο που συναντάει κανείς στις διάφορες προσεγγίσεις επίλυσης του προβλήματος εκτίμησης ανθρώπινης πόζας.

2.5.2 Μοντελοποίηση ανθρώπινου σώματος

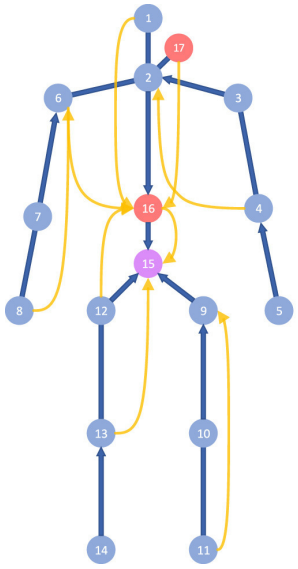
Όπως αναφέρθηκε, το ανθρώπινο σώμα είναι περίπλοκο με πολλαπλές αρθρώσεις πολλών βαθμών ελευθερίας και μεγάλου εύρους κίνησης. Τοιουτοτρόπως, κρίνεται αναγκαία η σαφής μοντελοποίηση της υποβόσκουσας δομής του, μειώνοντας αφενός την εγγενή αβεβαιότητα του προβλήματος εκτίμησης πόζας και αφετέρου αποτελώντας το μέσο για την περιγραφή της τελικής πόζας.

Οι διαφορετικές μέθοδοι εκτίμησης πόζας χρησιμοποιούν διαφορετικές μοντελοποιήσεις του ανθρώπινου σώματος ανάλογα με τις ανάγκες της εκάστοτε μεθόδου. Εντούτοις, οι κατά κόρον χρησιμοποιούμενες μοντελοποιήσεις είναι τα σκελετικά μοντέλα (*skeleton models*) και τα μοντέλα σχήματος (*shape models*). Ταυτόχρονα, αξίζει να σημειωθεί μία καινούργια μορφή αναπαράστασης του ανθρώπινου σώματος που βασίζεται στην αναπαράσταση των σημείων της επιφάνειας του σώματος. [11]

Μοντέλο σκελετού

Το Μοντέλο σκελετού είναι μια δενδρική δομή αποτελούμενη από σημεία κλειδιά του ανθρώπινου σώματος, συνδέοντας τις φυσικές παρακείμενες αρθρώσεις, τα σημεία κλειδιά, με ακμές (για παράδειγμα η ακμή του βραχίονα συνδέει τις

αρθρώσεις-σημεία κλειδιά του καρπού και του αγκώνα). Ανάλογα με την μοντελοποίηση, ορίζονται οι γονείς πρώτης τάξης του κάθε σημείου κλειδιού ή ενδεχομένως και οι γονείς δεύτερης τάξης, όπως φαίνεται στο Σχήμα 2.11, ανάλογα με τις ανάγκες του χρησιμοποιούμενου μοντέλου μηχανικής μάθησης.



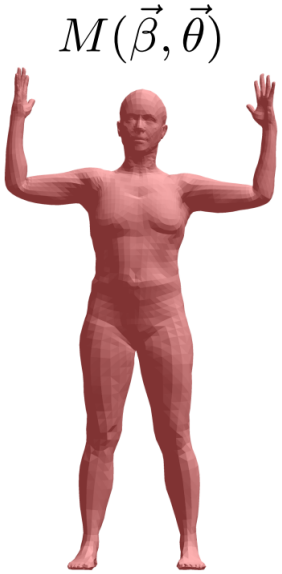
Σχήμα 2.11: *Μοντέλο Σκελετού ανθρώπινου σώματος*. Το σημείο κλειδί αναφοράς είναι το 15. Οι μπλε ακμές δείχνουν τους γονείς πρώτης τάξης του κάθε σημείου κλειδιού, ενώ οι κίτρινες ακμές τους γονείς δεύτερης τάξης (παραλείπονται κάποιοι για ευκρίνεια της εικόνας).

Μοντέλο σχήματος

Το μοντέλο σχήματος αναπαριστά το ανθρώπινο σώμα ως ένα τριγωνικό πλέγμα καθορίζοντας έτσι πλήρως το σχήμα του σε οποιαδήποτε πόζα. Η μοντελοποίηση που χρησιμοποιείται στην πλειοψηφία των άρθρων εκτίμησης πόζας των τελευταίων χρόνων είναι το *skinned multi-person linear model* (SMPL) [12]. Το μοντέλο SMPL, το οποίο φαίνεται στο Σχήμα 2.12, περιγράφει το ανθρώπινο δέρμα με 6890 κορυφές που σχηματίζουν το τριγωνικό πλέγμα. Οι κορυφές παραμετροποιούνται από τις παραμέτρους σχήματος $\vec{\beta}$ που καθορίζουν τις αναλογίες του σώματος, όπως το ύψος και τα κιλά, και τις παραμέτρους πόζας $\vec{\theta}$ που περιγράφουν τις περιστροφές των αρθρώσεων από μία πόζα αναφοράς.

Επιφανειακό μοντέλο

Μία ακόμα πιο πρόσφατη μοντελοποίηση του ανθρώπινου σώματος για το πρόβλημα της εκτίμησης πόζας, αποτελεί το επιφανειακό μοντέλο. Αναπτύχθηκε με την εικασία ότι μία αραιή συσχέτιση της εικόνας με τα σημεία κλειδιά του ανθρώπινου σώματος δεν είναι αρκετή για να αντικατοπτρίσει τη πλήρη κατάσταση του.



Σχήμα 2.12: *Μοντελοποίηση SMPL*. Το ανθρώπινο σώμα περιγράφεται από τις παραμέτρους σχήματος $\vec{\beta}$ και πόζας $\vec{\theta}$.

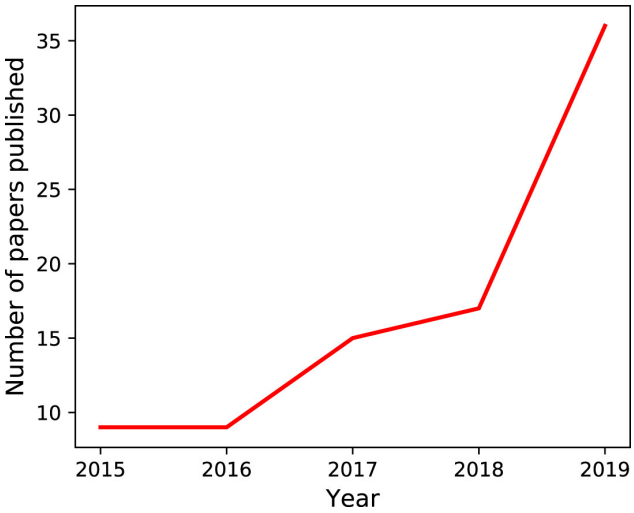
Τοιουτοτρόπως, στο άρθρο [11], προτείνεται η επιφανειακή μοντελοποίηση, αποκαλούμενη DensePose, όπου εγκαθιδρύεται μία πυκνή συσχέτιση των πίξελ της εικόνας με μία επιφανειακή αντιπροσώπευση του ανθρώπινου σώματος, όπως φαίνεται στο Σχήμα 2.13

Σχήμα 2.13: *Μοντελοποίηση DensePose*. Στα αριστερά φαίνεται η επιφανειακή τρισδιάστατη αναπαράσταση του ανθρώπινου σώματος. Αντιστοιχίζοντας τα σημεία της εικόνας δεξιά στην αριστερή επιφάνεια γίνεται δυνατή η εκτίμηση της πόζας.

2.5.3 Κατηγοριοποιήσεις λύσεων του προβλήματος εκτίμησης πόζας

Για την επίλυση του προβλήματος εκτίμησης πόζας τα τελευταία χρόνια παρουσιάζονται ολοένα και περισσότερα άρθρα όπως φαίνεται στο Σχήμα 2.14. Προφανώς, η κάθε προσέγγιση επίλυσης του προβλήματος έχει μοναδικά χαρακτηριστικά, ικανοποιώντας διαφορετικές απαιτήσεις, αλλά υπάρχουν κοινές κατευθυντήριες γραμμές πάνω στις οποίες κινούνται οι ερευνητές. Με αυτόν τον τρόπο, γίνεται

δυνατή η κατηγοριοποίηση των λύσεων του προβλήματος ως προς τα διάφορα χαρακτηριστικά τους.



Σχήμα 2.14: Αριθμός άρθρων εκτίμησης ανθρώπινης πόζας (κατακόρυφος άξονας) ανά χρόνο (οριζόντιος άξονας)

Η γενικότερη κατηγοριοποίηση αφορά την μορφή της εισόδου του μοντέλου μηχανικής μάθησης. Με άλλα λόγια, το μοντέλο μπορεί να χρησιμοποιεί για είσοδο μόνο μία εικόνα ή πολλαπλές εικόνες από διαφορετικές οπτικές (της ίδιας χρονικής στιγμής). Αντίθετα, για την αξιοποίηση της χρονικής πληροφορίας, μπορεί να χρησιμοποιηθεί ως είσοδος αλληλουχίες εικόνων, είτε μεμονωμένων είτε από διαφορετικές οπτικές. Προφανώς, οι πιο περίπλοκες είσοδοι απαιτούν περισσότερους πόρους τόσο για την ανάκτηση των εικόνων όσο και για την επεξεργασία τους, μειώνοντας ωστόσο τις ασάφειες του προβλήματος.

Επιπλέον, οι μεθοδολογίες της εκτίμησης πόζας μπορούν να διαχωριστούν ανάλογα με την ικανότητα τους στην πρόβλεψη πόζας ενός ή περισσότερων ανθρώπων. Στην πρώτη περίπτωση, ορίζεται η πρόβλεψη της πόζας μόνο για έναν άνθρωπο μέσα στην εικόνα και είναι σαφώς η απλούστερη. Στην περίπτωση εκτίμησης πόζας περισσότερων ανθρώπων το πρόβλημα δυσχεραίνεται λόγω του πολύ μεγαλύτερου χώρου καταστάσεων, της απόκρυψης ή και σύγχυσης των σημείων κλειδιών και της αναγκαιότητας διαφοροποίησης τους.

Τοιουτοτρόπως, συγκεκριμένα για το πρόβλημα εκτίμησης πόζας πολλαπλών ανθρώπων, διακρίνουμε την κατηγοριοποίηση με βάση τον τρόπο ομαδοποίησης των σημείων κλειδιών στους ανθρώπους στους οποίους ανήκουν. Οι δύο κύριες προσεγγίσεις είναι η *από κάτω προς τα πάνω* και η *από πάνω προς τα κάτω*.

- *Από κάτω προς τα πάνω*: Το μοντέλο ανιχνεύει κάθε εμφάνιση ενός συγκεκριμένου σημείου κλειδιού (όπως όλα τα αριστερά χέρια) σε μία εικόνα και στην συνέχεια προσπαθεί να ομαδοποιήσει τα σημεία κλειδιά με βάση τους ανθρώπους στους οποίους ανήκουν.
- *Από πάνω προς τα κάτω*: Αντίθετα, σε αυτή την μέθοδο, το μοντέλο αρχικά χρησιμοποιεί έναν ανιχνευτή αντικειμένων, καθορίζοντας έτσι ένα πλαί-

σιο οριοθέτησης γύρω από κάθε άνθρωπο, και έπειτα εκτιμά την θέση των σημείων κλειδιών του κάθε ανθρώπου εσωτερικά σε κάθε περικομμένη περιοχή.

Μία άλλη πιθανή κατηγοριοποίηση σχετίζεται με τα χρησιμοποιούμενα στάδια προβλέψεων έως ότου προκύψει η τελική εκτίμηση πόζας. Αναλυτικότερα, θα αναλύσουμε τις μεθόδους από τρεις κατηγορίες: εκτίμηση τρισδιάστατης πόζας απευθείας από εικόνες, ανύψωση από δισδιάστατες σε τρισδιάστατες προβλέψεις και μεθόδους βασισμένες στο μοντέλο SMPL.

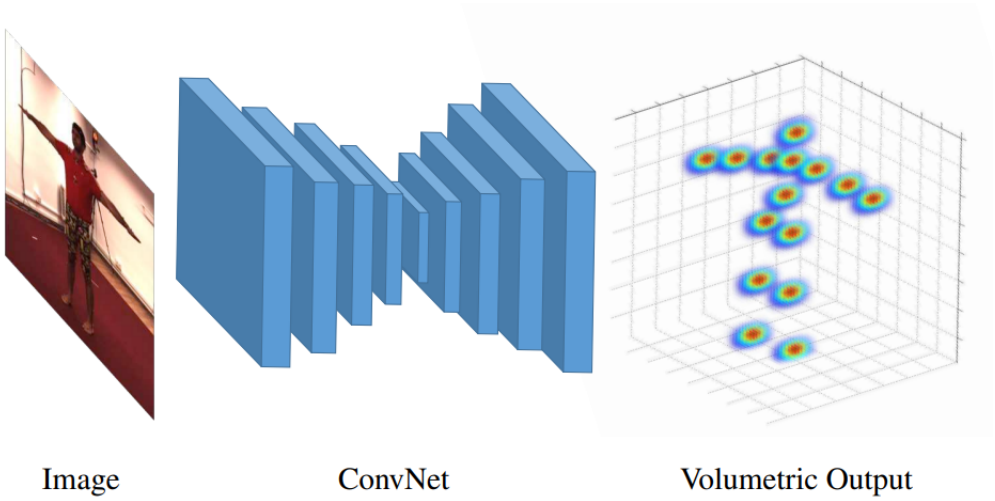
Εκτίμηση πόζας απευθείας από εικόνες

Η πιο άμεση μεθοδολογία για την εκτίμηση της τρισδιάστατης πόζας είναι ο σχεδιασμός συνεχόμενων δικτύων για την πρόβλεψη των 3D συντεταγμένων των σημείων κλειδιών ή αρθρώσεων. Οι μέθοδοι της απευθείας εκτίμησης από εικόνες μπορούν να διαχωριστούν περαιτέρω σε δύο κλάσεις: μέθοδοι βασισμένες στον εντοπισμό και μέθοδοι βασισμένες στην παλινδρόμηση. Αξίζει να σημειωθεί ότι έχουν γίνει προσπάθειες για ενοποίηση των δύο αυτών προσεγγίσεων.

Οι μέθοδοι βασισμένες στον εντοπισμό προβλέπουν έναν πιθανοτικό θερμικό χάρτη για κάθε σημείο κλειδί και παίρνοντας την μέγιστη πιθανότητα του χάρτη καθορίζουν την θέση του σημείου κλειδιού. Για παράδειγμα στο [13] αναπαριστάται ο τρισδιάστατος χώρος σε έναν όγκο και εκπαιδεύεται ένα μοντέλο Συνελικτικού Νευρωνικού Δικτύου (*Convolutional Neural Network*, *CNN* στο καθεξής) ώστε να προβλέπονται οι ογκομετρικοί πιθανοτικοί θερμικοί χάρτες για κάθε σημείο κλειδί, όπως φαίνεται στο Σχήμα 2.15. Παρόμοιες μέθοδοι βασίζονται σε επιπλέον βήματα για την μετατροπή των θερμικών χαρτών στην τελική εκτίμηση των θέσεων των σημείων κλειδιών, συνήθως παίρνοντας την μέγιστη πιθανότητα του χάρτη, δηλαδή εφαρμόζοντας την συνάρτηση argmax (βλ. στο παράρτημα τον ορισμό 1). Βέβαια, η διαδικασία αυτή δεν είναι διαφορίσιμη, αποτελώντας τροχοπέδη στον μηχανισμό εκμάθησης των νευρωνικών δικτύων. Ταυτόχρονα, η ακρίβεια των προβλεπόμενων σημείων κλειδιών είναι ανάλογη της διακριτότητας των θερμικών χαρτών ο οποίοι έχουν εγγενή αδυναμία στη χωρική γενίκευση. Για την αύξηση της ακρίβειας των προβλέψεων, οι παραγόμενοι θερμικοί χάρτες απαιτούν αύξηση της χωρικής διακριτότητας, αυξάνοντας όμως τετραγωνικά τις απαιτήσεις σε υπολογιστική ισχύ και κατανάλωση μνήμης.

Από την άλλη μεριά, το πρόβλημα εκτίμησης πόζας μπορεί να θεωρηθεί διαισθητικά ως ένα πρόβλημα παλινδρόμησης, όπου εκτιμούνται οι συντεταγμένες θέσεων των σημείων κλειδιών ως προς την θέση ενός σημείου κλειδιού αναφοράς[2]. Σε αυτή την προσέγγιση, γίνεται εμφανής η καταλληλότητα επιλογής των *μοντέλων σκελετού* ή παρεμφερή μοντελοποιήσεις για την αναπαράσταση του ανθρώπινου σώματος. Για παράδειγμα, για να ενσωματώσουν προηγούμενη γνώση για την δομή

[2]Στο μοντέλο σκελετού *SMPL*, για παράδειγμα, το σημείο αναφοράς, σημείο με δείκτη (*index*) 0, είναι η λεκάνη



Σχήμα 2.15: *Μέθοδος εκτίμησης πόζας βασισμένη στον εντοπισμό*. Η εικόνα ως είσοδος περνάει από την αρχιτεκτονική του CNN και παράγει πιθανοτικούς χάρτες για κάθε σημείο κλειδί στον 3D χώρο.

του ανθρώπινου σώματος, στο [14], εισάγουν ένα *κινηματικό μοντέλο σκελετού*, αποτελούμενο από αρθρώσεις και κόκαλα, τα οποία κόκαλα έχουν σταθερό μήκος και μπορούν να περιστραφούν γύρω από τις αρθρώσεις. Εντούτοις, το σταθερό μήκος των κοκάλων δεν ανταποκρίνεται επαρκώς στη μεταβλητότητα του ανθρώπινου σκελετού, γεγονός που υποβαθμίζει την ικανότητα γενικοποίησης του μοντέλου. Από την άλλη μεριά, στο [15] υποθέτουν ότι στην εκτίμηση πόζας είναι πιο λογική η εκτίμηση της θέσεως των κοκάλων αντί για τις αρθρώσεις διότι η αναπαράσταση των κοκάλων καθιστά ευκολότερη την εκμάθηση του *μοντέλου βαθιάς μάθησης* και αντικατοπτρίζουν καλύτερα τους γεωμετρικούς περιορισμούς του ανθρώπινου σκελετού. Φυσικά, αυτή η προσέγγιση απαιτεί την μετατροπή των δεδομένων εκπαίδευσης στην σχετική μορφή αναπαράστασης τις θέσεις των κοκάλων.

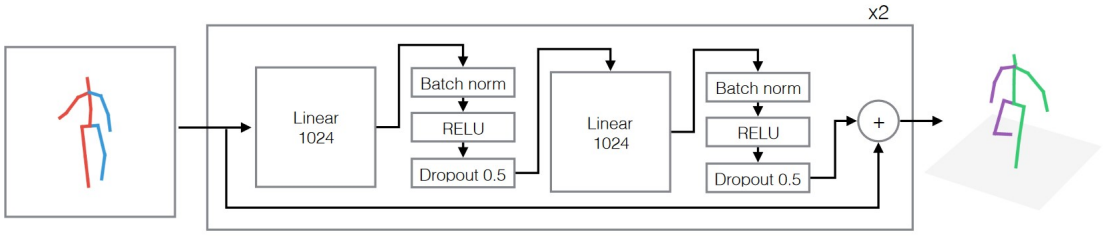
Εν κατακλείδι, οι μέθοδοι εντοπισμού με θερμικούς χάρτες, αν και πιο αποδοτικοί από τις μεθόδους παλινδρόμησης, έχουν κάποια μειονεκτήματα. Η διαδικασία επιλογής της μέγιστης πιθανότητας δεν είναι διαφορίσιμη και δεν επιτρέπει την από άκρη σε άκρη εκμάθηση του μοντέλου, σε αντίθεση με τις μεθόδους παλινδρόμησης όπου αυτή η εκμάθηση είναι εφικτή.

Προς αντιμετώπιση των προαναφερθέντων προβλημάτων έχουν γίνει προσπάθειες συνδυασμού των δύο παραπάνω μεθόδων. Για παράδειγμα, στο άρθρο [16], προτείνεται η χρήση της συνάρτησης soft-argmax (αναλυτικότερα βλ. ορισμό 2) για την μετατροπή των χαρτών χαρακτηριστικών (feature maps), ή των θερμικών χαρτών, σε συντεταγμένες των σημείων κλειδιών, έχοντας ως αποτέλεσμα ένα πλήρως διαφορίσιμο σύστημα. Με παρόμοια λογική με την συνάρτηση soft-argmax, στο άρθρο [17], εισάγουν ένα καινούργιο επίπεδο, αποκαλούμενο διαφορίσιμη χωρική σε αριθμητική μετατροπή (differentiable spatial to numerical transform, DSNT), για να διατηρήσουν την από από άκρη σε άκρη δυνατότητα εκμάθησης του μοντέλου και ταυτόχρονα βελτιώνοντας την ικανότητα γενικοποίησης του.

Ανύψωση προβλέψεων από 2D σε 3D

Εμπνευσμένοι από την ραγδαία εξέλιξη των αλγορίθμων για εκτίμηση της 2D πόζας, αρκετοί ερευνητές προσπάθησαν να χρησιμοποιήσουν τα αποτελέσματα της 2D εκτίμησης πόζας για να βελτιώσουν την ικανότητα γενικοποίησης σε δεδομένα αποκτημένα υπό καθημερινές συνθήκες.

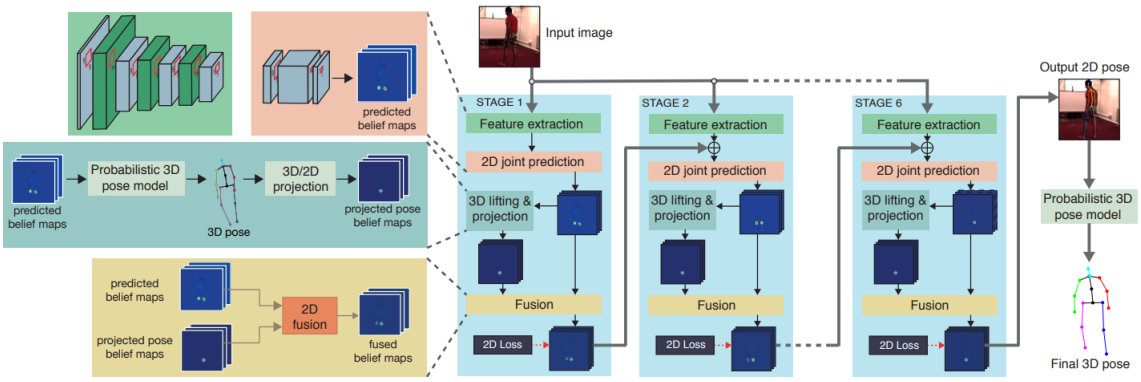
Χαρακτηριστικό παράδειγμα αποτελεί η έρευνα στο [18] όπου προτείνεται μία αρχιτεκτονική, όπως φαίνεται στο Σχήμα 2.16, επικεντρωμένη στην ανύψωση της 2D πόζας σε 3D με ένα απλό αλλά αποδοτικό νευρωνικό δίκτυο, εμφυσώντας ενδιαφέρον για περαιτέρω έρευνα στην ανύψωση της 2D πόζας στον 3D χώρο.



Σχήμα 2.16: Αρχιτεκτονική simple baseline για την εκτίμηση πόζας. Ακρογωνιαίος λίθος του νευρωνικού αποτελεί ένα γραμμικό επίπεδο, ακολουθούμενο από ένα επίπεδο κανονικοποίησης παρτίδας (batch normalization), ένα επίπεδο ενεργοποίησης με συνάρτηση ReLU (βλ. παράρτημα 3) και ένα επίπεδο εγκατάλειψης μονάδων εισόδου (dropout). Η είσοδος του συστήματος είναι ένας πίνακας με τις 2D θέσεις των σημείων κλειδιών και η έξοδος είναι οι θέσεις των σημείων κλειδιών σε 3D συντεταγμένες.

Για την αντιμετώπιση της δυσκολίας παραγωγής και σχολιασμού εξολοκλήρου 3D συνόλων δεδομένων σε διάφορες έρευνες χρησιμοποιούνται επιπλέον πληροφορίες ως ενδιάμεση επίβλεψη του συστήματος. Ένα είδος πληροφορίας που χρησιμοποιείται συχνά είναι η προβολή της προβλεπόμενης 3D πόζας στον 2D χώρο της εικόνας. Για παράδειγμα στο [19], αρχικά εκτιμάται η 2D πόζα η οποία χρησιμοποιείται για την εκτίμηση της 3D πόζας. Έπειτα, οι προβλέψεις της 3D πόζας προβάλλονται στον 2D χώρο, συνδυάζονται με τις αρχικές 2D προβλέψεις και τέλος συγκρίνονται με τους 2D σχολιασμούς του συνόλου δεδομένων (όπως φαίνεται στο Σχήμα 2.17), εκπαιδεύοντας έτσι το σύστημα. Αξίζει να σημειωθεί ότι στο στάδιο ανύψωσης των 2D προβλέψεων σε 3D ενσωματώνεται ένα επιπλέον επίπεδο στο CNN όπου αναπαριστώνται οι 3D γεωμετρικοί περιορισμοί του σώματος, ενός *μοντέλου σκελετού* (όπως αναπτύχθηκε στην ενότητα 2.5.2), εξασφαλίζοντας ότι οι προβλεπόμενες πόζες βρίσκονται στον χώρο των επιτρεπόμενων εκ φύσεως ποζών.

Αντίστοιχα για την αποφυγή σχολιασμού των 3D συνόλων δεδομένων, στο [20], προτείνεται μία εναλλακτική, πιο ασθενής μέθοδος επίβλεψης του συστήματος. Πιο συγκεκριμένα, για την επίβλεψη χρησιμοποιούνται οι ήδη υπάρχοντες σχολιασμοί των 2D σημείων κλειδιών σε συνδυασμό με τις σχέσεις βάθους (πιο κοντά - πιο



Σχήμα 2.17: *Lifting from the deep: Παράδειγμα αρχιτεκτονικής ανύψωσης 2D σε 3D.* Κάθε στάδιο παράγει ως έξοδο ένα σύνολο 2D πιθανοτικών χαρτών για κάθε σημείο κλειδί, οι οποίοι μαζί με την αρχική εικόνα αποτελούν την είσοδο του επόμενο σταδίου. Κάθε στάδιο μαθαίνει να συνδυάζει τους πιθανοτικούς χάρτες της 2D εκτίμησης πόζας με τους προβαλλόμενους χάρτες από την 3D εκτίμηση πόζας.

μακριά) των σημείων κλειδιών, όπως φαίνεται στο Σχήμα 2.18. Μάλιστα, παρουσιάζεται η ευελιξία αυτής της μεθόδου επίβλεψης, ενσωματώνοντάς την σε διαφορετικές διαρρυθμίσεις νευρωνικών δικτύων όπου επιτυγχάνονται ανταγωνιστικά αποτελέσματα με την επίβλεψη με πραγματικούς 3D σχολιασμούς του συνόλου δεδομένων.

Μέθοδοι βασισμένες στο μοντέλο σχήματος SMPL

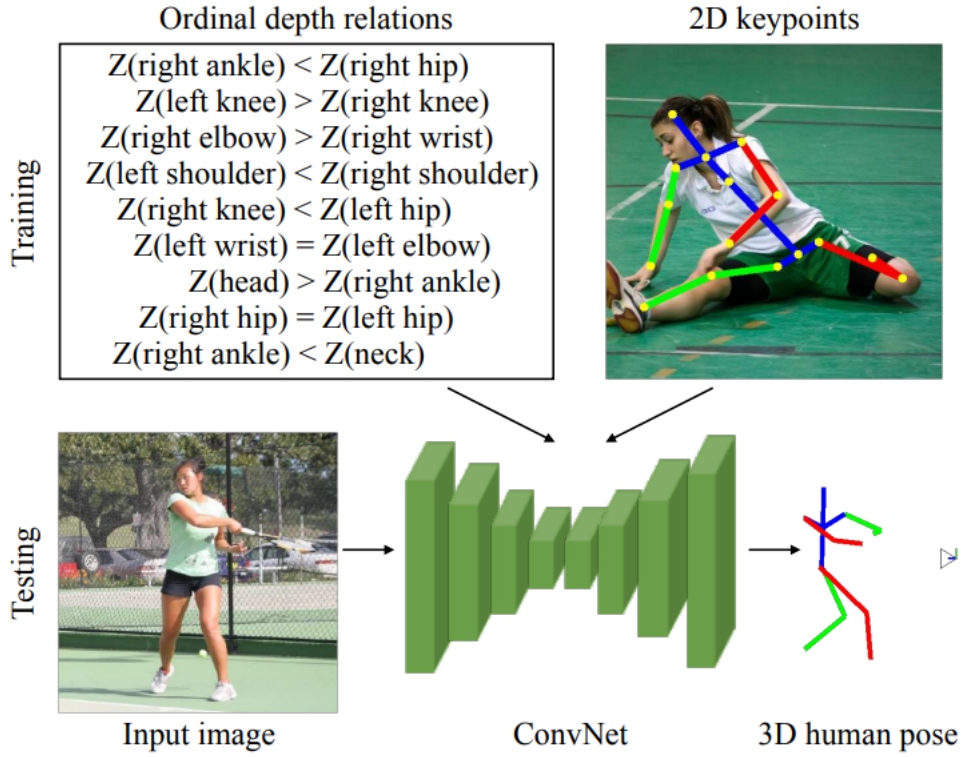
2.5.4 Αρχιτεκτονικές μοντέλων μηχανικής μάθησης

Οι αρχιτεκτονικές *βαθιάς μηχανικής μάθησης* για την εκτίμηση πόζας παίρνουν πολλαπλές μορφές. Οι περισσότερες χρησιμοποιούν αρχικά ένα κωδικοποιητή ο οποίος δέχεται ως είσοδο την εικόνα ή πλαίσια αυτής και εξάγει κάποια χαρακτηριστικά (*features*). Το επόμενο βήμα, εξαρτάται κατά κόρον από την εκάστοτε υλοποίηση.

Η πιο απλή τεχνική χρησιμοποιεί έναν *παλινδρομητή* (*regressor*), ο οποίος δέχεται ως είσοδο τα εξαγόμενα χαρακτηριστικά από τον κωδικοποιητή και παράγει τις τελικές εκτιμήσεις των τρισδιάστατων συντεταγμένων για κάθε σημείο κλειδί.

Μία ελαφρώς πιο περίπλοκη τεχνική χρησιμοποιεί αρχιτεκτονική κωδικοποιητή - αποκωδικοποιητή. Τα εξαγόμενα χαρακτηριστικά του κωδικοποιητή εισάγονται σε έναν αποκωδικοποιητή ο οποίος παράγει θερμικούς χάρτες αναπαριστώντας την πιθανότητα ενός σημείου κλειδιού να βρίσκεται σε κάποια περιοχή της εικόνας ή του χώρου.

Οι δύο προαναφερθείσες τεχνικές μπορούν να εφαρμοστούν στην εκτίμηση τόσο



Σχήμα 2.18: *Ordinal Depth Supervision: Παράδειγμα αρχιτεκτονικής ανύψωσης 2D σε 3D*. Όταν εκλείπουν οι 3D σχολιασμοί του συνόλου δεδομένου, μπορούν να χρησιμοποιηθούν οι 2D σχολιασμοί των σημείων κλειδιών σε συνδυασμό με τις σχέσεις βάθους των σημείων κλειδιών.

της δισδιάστατης όσο και της τρισδιάστατης πόζας. Στην περίπτωση όμως της τρισδιάστατης εκτίμησης πόζας υφίσταται μία ακόμα επιλογή. Αντί για την εκτίμηση των συντεταγμένων των σημείων κλειδιών, είναι πιθανό να εκτιμηθούν οι τιμές των παραμέτρων κάθε βαθμού ελευθερίας όλων των αρθρώσεων και στην συνέχεια να εφαρμοστούν οι μετασχηματισμοί στις αρθρώσεις από μία πόζα αναφοράς.

Και στις δύο προσεγγίσεις, οι αρχιτεκτονικές *βαθιάς μηχανικής μάθησης* κατάλληλες για ανάλυση της εικόνας έχουν ως ακρογωνιαίο λίθο κάποια παραλλαγή ενός *Συνελικτικού Νευρωνικού Δικτύου* (*Convolutional Neural Network*, *CNN*).

3

Εργαλεία

3.1 Kivy

Σχήμα 3.1: Kivy

Το Kivy[3] είναι μια βιβλιοθήκη ανοιχτού κώδικα της Python για την γρήγορη ανάπτυξη εφαρμογών που χρησιμοποιούν γραφικά για υλοποίηση καινοτόμων διεπαφών χρηστών. Επιλέξαμε την εν λόγω βιβλιοθήκη για την υλοποίηση της διεπαφής του καθρέφτη επειδή είναι:

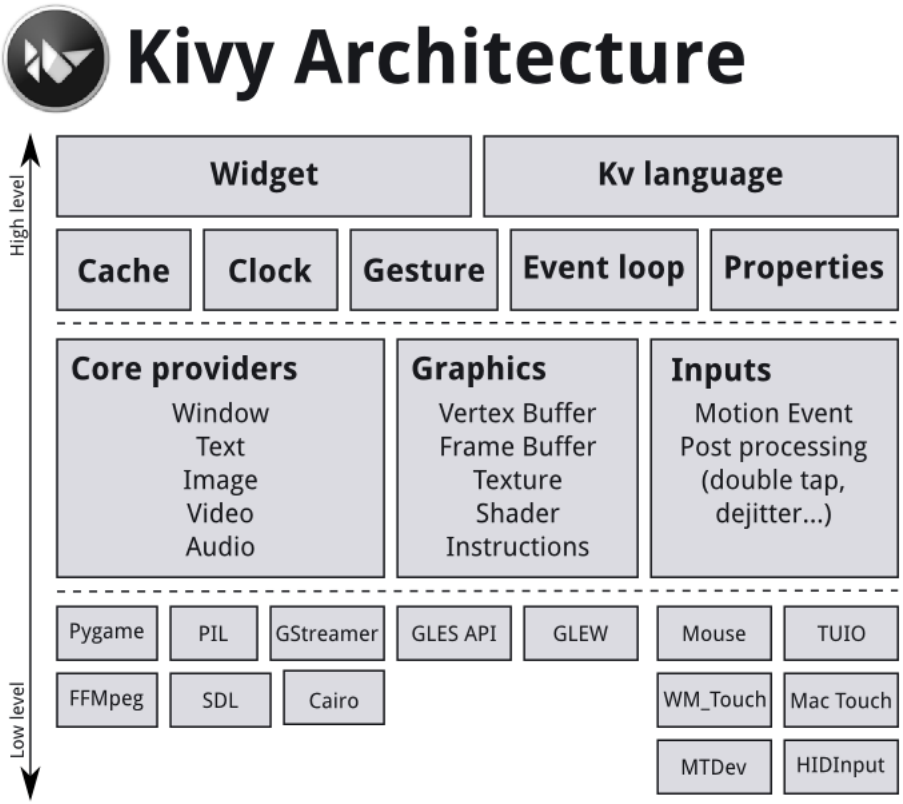
- **Cross Platform**: Το Kivy τρέχει σε διαφορετικά λειτουργικά συστήματα και πλατφόρμες όπως Windows, Linux, Android, OS X, Raspberry Pi με τον ίδιο κώδικα.
- **GPU Accelerated**: Η μηχανή γραφικών έχει υλοποιηθεί πάνω στην OpenGL ES 2 χρησιμοποιώντας μια σύγχρονη και γρήγορη γραμμή γραφικών.

[3] https://kivy.org/

- **Εύκολη στη Χρήση**: Περιλαμβάνει μια ευρεία γκάμα έτοιμων widgets ενώ χρησιμοποιεί μια απλή ενδιάμεση γλώσσα (kv) για τον εύκολο σχεδιασμό widget από τον χρήστη.

3.1.1 Αρχιτεκτονική του Kivy

Το Kivy αποτελείται από πολλά δομικά κομμάτια όπως φαίνεται στο Σχήμα 3.2. Κάποια βασικά από αυτά τα δομικά κομμάτια περιγράφονται στη συνέχεια [21].



Σχήμα 3.2: Αρχιτεκτονική της βιβλιοθήκης Kivy

Core και Input Providers

Ένας Core Provider είναι ένα κομμάτι κώδικα που λειτουργεί ως διαμεσολαβητής ανάμεσα στο ΛΣ και το Kivy. Η βασική ιδέα είναι να χωριστούν εργασίες όπως το άνοιγμα ενός παραθύρου, η εμφάνιση μια εικόνας ή κειμένου, η αναπαραγωγή ήχου κτλ. έτσι ώστε το API να είναι εύχρηστο και επεκτάσιμο. Το βασικότερο πλεονέκτημα όμως είναι ότι με αυτόν τον τρόπο επιτρέπεται να χρησιμοποιηθούν συγκεκριμένοι providers ανάλογα με τα σενάρια χρήσης της εφαρμογής. Για παράδειγμα, τα OS X, Linux και Windows χρησιμοποιούν διαφορετικούς providers

για τις παραπάνω εργασίες. Με τη χρήση εξειδικευμένων providers για κάθε πλατφόρμα είναι εφικτή η πλήρης αξιοποίηση του ΛΣ με αποδοτικό τρόπο, ενώ είναι εφικτή η μεταφορά της εφαρμογής σε άλλες πλατφόρμες αρκετά εύκολα.

Με αντίστοιχο τρόπο δουλεύουν και οι providers εισόδων. Αν κάποιος θέλει να υποστηρίξει μια νέα συσκευή εισόδου χρειάζεται απλά να γράψει μία κλάση η οποία διαβάζει τα δεδομένα της συγκεκριμένης συσκευής και τα μεταφράζει και γεγονότα του Kivy (Kivy Events).

Γραφικά

Το γραφικό API του Kivy είναι μια αφαιρετικότητα της OpenGL. Στο χαμηλότερο επίπεδο το Kivy δίνει hardware accelerated εντολές σχεδιασμού χρησιμοποιώντας OpenGL. Αλλά επειδή το να γράφει κανείς σε OpenGL είναι επίπονο, ειδικά για τους νέους χρήστες, είναι εφικτό να σχεδιαστού γραφικά χρησιμοποιώντας μεταφορικές εντολές (Canvas, Rectangle κτλ.) που δεν υπάρχουν στην OpenGL.

Πυρήνας

Ο κώδικας του πακέτου του πυρήνα παρέχει συχνά χρησιμοποιούμενα χαρακτηριστικά όπως:

- Το **ρολόι** που χρησιμοποιείται για να προγραμματιστούν χρονικά γεγονότα.
- Την **cache** σε περίπτωση που κάποιος θέλει να αποθηκεύσει κάτι που χρησιμοποιείται συχνά.
- Την **αναγνώριση χειρονομίας** για αποθήκευση και ανίχνευση διαφόρων σχημάτων, όπως κύκλοι, ενώ μπορεί να εκπαιδευτεί για σχήματα που καθορίζει ο χρήστης.
- Την **γλώσσα Kivy** η οποία χρησιμοποιείται για να περιγραφούν με ευκολία και αποδοτικότητα οι διεπαφές χρηστών
- Τις **ιδιότητες** οι οποίες είναι κλάσεις που συνδέουν τον κώδικα ενός widget με την περιγραφή της διεπαφής του χρήστη.

UIX

Το συγκεκριμένο πακέτο περιέχει τα πιο συχνά χρησιμοποιούμενα widgets και layouts για την γρήγορη ανάπτυξη μια διεπαφής χρήστη.

- Τα **widgets** είναι στοιχεία διεπαφής τα οποία εισάγονται στο πρόγραμμα και παρέχουν κάποια λειτουργία όπως πχ. κουμπιά, λίστες, επιγραφές κ.α.
- Τα **layouts** χρησιμοποιούνται για την διάταξη των widgets.

3.2 Wit.ai

Σχήμα 3.3: Wit.ai

Το Wit[4] είναι ένα framework ανοιχτού κώδικα που επιτρέπει του χρήστες να αλληλεπιδρούν με την εφαρμογή μέσως φωνής και βασίζεται στη χρήση ισχυρού NLP (Natural Language Processing). Η λειτουργία του Wit μπορεί να χωριστεί σε 2 βασικές κατηγορίες, την **ταξινόμηση πρόθεσης** (intent classification) και την **εξαγωγή οντοτήτων** (entity extraction).

Το Wit επιτρέπει μια εφαρμογή να καταλάβει τους χρήστες. Αρχικά μπορεί ο προγραμματιστής μέσω της πλατφόρμας να δώσει κάποιες **εκφράσεις** (utterances) ως παραδείγματα στις οποίες ορίζει την πρόθεση και τις οντότητες που περιέχουν προκειμένου να εκπαιδευτεί το μοντέλο. Για παράδειγμα μπορούμε να δώσουμε ως έκφραση την παρακάτω πρόταση:

«What's the temperature?»

και να ορίσουμε ως πρόθεση `temperature_get`. Με αυτόν τον τρόπο μαθαίνουμε στο Wit ότι με αυτήν την έκφραση ο χρήστης δηλώνει την πρόθεσή του να μάθει την θερμοκρασία. Όταν επομένως ρωτήσει ο χρήστης το Wit για την θερμοκρασία (είτε με την παραπάνω έκφραση είτε με μια παρόμοια όπως «I would like to know the temperature») το Wit θα προσπαθήσει να προβλέψει την πρόθεση του χρήστη δίνοντας και μία μετρική της σιγουριάς (confidence) από 0 έως 1. Όσα περισσότερα παραδείγματα δώσουμε τόσο μεγαλύτερη γίνεται η σιγουριά και η ικανότητα του Wit για την πρόβλεψη της πρόθεσης.

Έστω ότι δίνεται ως είσοδος στο Wit η έκφραση «set the temperature to 70 degrees». Η απάντηση που θα δοθεί θα έχει την εξής μορφή:

```
{
    "text": "set the temperature to 70 degrees",
    "intents": [
        {
            "id": "226127658493500",
            "name": "temperature_get",
            "confidence": 0.9953
        }
    ],
    "entities": {},
    "traits": {}
}
```

[4] https://wit.ai/

Η απάντηση όμως δεν είναι ικανοποιητική αφού η πρόθεση του χρήστη είναι να αλλάξει η θερμοκρασία και όχι να πάρει την θερμοκρασία κάτι που είναι λογικό αφού το Wit καταλαβαίνει μόνο μία πρόθεση. Μέσω της πλατφόρμας όμως είναι εύκολο να αλλάξουμε την πρόθεση μιας ήδη δοσμένης έκφρασης και να ξαναεκπαιδευτεί το μοντέλο θέτοντας αυτή τη φορά την πρόθεση `temperature_set`. Δίνεται επίσης η επιλογή να ορίστει ένα κατώφλι σιγουριάς έτσι ώστε να μην επιστρέφονται προθέσεις κάτω από αυτό το όριο.

Στο παραπάνω παράδειγμα, θέλουμε να αποσπάσουμε μαζί με την πρόθεση και το νούμερο της επιθυμητής θερμοκρασίας. Κάτι τέτοιο είναι εύκολο μέσω της πλατφόρμας όπου μπορούμε να ορίσουμε το `70 degrees` να είναι μια οντότητα (πχ `temperature_degrees`). Έτσι στην απάντηση που θα λάβει η εφαρμογή από το Wit θα μπορεί να εξάγει και τον αριθμό των βαθμών έτσι ώστε να προβεί στην κατάλληλη πράξη για να θέσει την θερμοκρασία στην επιθυμητή τιμή.

Η επιλογή του Wit έγινε ανάμεσα σε αρκετά αντίστοιχα εργαλεία ανοιχτού κώδικα που υπάρχουν διαθέσιμα όπως το spaCy[5] ή το Rasa[6]. Ο βασικός λόγος επιλογής του Wit είναι η ευκολία εκπαίδευσης και προσαρμογής του μέσω της πλατφόρμας. Επίσης, η ενσωμάτωση στην εφαρμογή ήταν πολύ εύκολη με τη χρήση της βιβλιοθήκης[7] που παρέχει το Wit για Python, το documentation ήταν αρκετά κατατοπιστικό και η δομή των δεδομένων στην απάντηση του Wit το έκαναν εύκολο στην χρήση. Τέλος, είναι δυνατό κανείς να κατεβάσει τα δεδομένα ενός ήδη εκπαιδευμένου μοντέλου, και να δημιουργήσει ένα με βάση αυτά όποτε και να μπορεί στη συνέχει να το επεκτείνει με δικές του εντολές και να το προσαρμόσει στις ανάγκες των δικών του εφαρμογών, βοηθώντας κατά αυτόν τον τρόπο στην επεκτασιμότητα της διεπαφής του καθρέφτη.

3.3 Χειρισμος Φωνης

Για την αλληλεπίδραση του χρήστη με τον καθρέφτη επιλέξαμε ως μέσο την φωνή του. Το κομμάτι της φωνής μπορεί να χωριστεί σε δύο μέρη, την είσοδο κατά την οποία θέλουμε να μετατρέψουμε τη φωνή σε κείμενο ούτως ώστε η εφαρμογή να καταλάβει την εντολή και να εκτελέσει την κατάλληλη ενέργεια (**Speech to Text**) και την έξοδο όπου σε αρκετές περιπτώσεις θα θέλαμε ο καθρέφτης να μιλάει στον χρήστη δίνοντας του πληροφορίες και κατά την οποία μετατρέπουμε το κείμενο σε φωνή (**Text to Speech**).

[5] https://spacy.io/
[6] https://rasa.com/open-source/
[7] https://github.com/wit-ai/pywit

3.3.1 SpeechRecognition

Το SpeechRecognition[8] είναι μια βιβλιοθήκη για την εκτέλεση αναγνώρισης φωνής όπου υποστηρίζει διάφορες μηχανές και APIs. Για τον καθρέφτη χρησιμοποιήθηκε το API της Google[9].

Για την λειτουργία της αναγνώρισης φωνής, απαιτείται η δημιουργία 2 αντικειμένων, ένα τύπου `Microphone` και ένα τύπου `Recognizer`. Το πρώτο αντικείμενο περιέχει μεθόδους για την είσοδο φωνής μέσω του μικροφώνου αλλά και παραμετροποίησης διαφόρων τιμών όπως η μέγιστη διάρκεια που θα ακούει το μικρόφωνο, τιμές σχετικές με τον θόρυβο φόντου κτλ, ενώ επιστρέφει ένα αντικείμενο που μπορεί να χρησιμοποιηθεί ως πηγή (source) στη συνέχεια.

Το δεύτερο αντικείμενο, περιέχει μεθόδους για την ανάγνωση του ήχου και την μετατροπή του σε κείμενο μέσω του επιθυμητού υποστηριζόμενου API. Πιο συγκεκριμένα, μέσω της συνάρτησης `listen` μπορούμε να ηχογραφήσουμε από την πηγή (στον καθρέφτη η πηγή μας είναι το μικρόφωνο) και να αποθηκεύσουμε τα δεδομένα σε κατάλληλη μορφή για την αναγνώρισή τους αργότερα. Αυτό επιτυγχάνεται περιμένοντας το επίπεδο ενέργειας του ήχου να ξεπεράσει ένα κατώφλι (ο χρήστης ξεκίνησε να μιλάει) και σταματάει όταν η ενέργεια πέσει χαμηλά ή όταν δεν υπάρχει άλλη είσοδος ήχου. Η μέθοδος `listen()` περιέχει επίσης κάποιες χρήσιμες παραμέτρους για τον καθορισμό του μέγιστου χρόνου που θα περιμένει η εφαρμογή μέχρι ο χρήστης αρχίζει να μιλάει (`timeout`) και τον καθορισμό του μέγιστου χρόνου που θα περιμένει την φράση να τελειώσει (`phrase_timeout`). Επομένως αν και οι 2 παράμετροι είναι ορισμένοι ο συνολικός χρόνος που θα ακούει η εφαρμογή είναι `timeout` + `phrase_timeout`.

Τέλος, τα δεδομένα που επιστρέφονται από την συνάρτηση `listen()` δίνονται ως είσοδο στην κατάλληλη συνάρτηση αναγνώρισης για το επιθυμητό API. Στην εφαρμογή του καθρέφτη επιλέξαμε την μέθοδο `recognize_google()` η οποία στέλνει τα δεδομένα στο API της Google και αυτό με τη σειρά του επιστρέφει σε κείμενο τα λόγια που ειπώθηκαν από τον χρήστη.

3.3.2 Text To Speech

Για την υλοποίηση της μετατροπής κειμένου σε φωνή χρησιμοποιήθηκαν δύο βιβλιοθήκες, η `gTTS`[10] και η `pydub`[11].

Η gTTS είναι μια βιβλιοθήκη που επιτρέπει την αλληλεπίδραση της εφαρμογής με το Text-To-Speech API[12] της Google και αποθηκεύει τον ήχο ομιλίας σε ένα

[8] https://github.com/Uberi/speech_recognition
[9] https://cloud.google.com/speech-to-text
[10] https://gtts.readthedocs.io/en/latest/
[11] https://pydub.com/
[12] https://cloud.google.com/text-to-speech

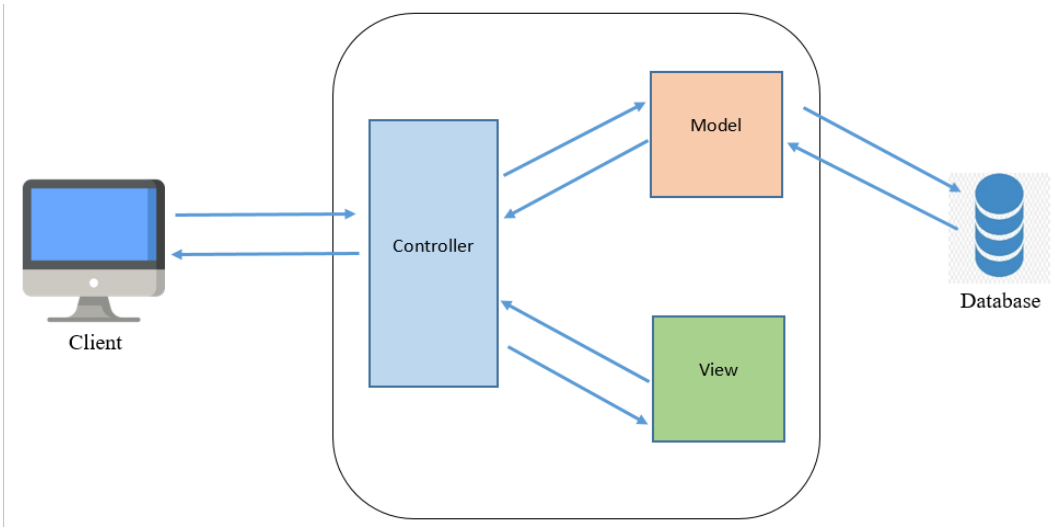
προσωρινό mp3 αρχείο. Στη συνέχεια, γίνεται χρήση της pydub βιβλιοθήκης για την αναπαραγωγή του ήχου στον χρήστη.

4

Υλοποίηση Καθρέφτη

4.1 Model-View-Controller

Το Model-View-Controller (εν συντομία **MVC**) είναι ένα σχεδιαστικό πρότυπο ευρέως διαδεδομένο, ειδικά στην ανάπτυξη Web εφαρμογών και απεικονίζεται στο Σχήμα 4.1. Όπως προδίδει και το όνομά του, το MVC χρησιμοποιεί 3 είδη κλάσεων οι οποίες περιγράφονται παρακάτω [22].



Σχήμα 4.1: Αρχιτεκτονική MVC

- **Model**: Οι συγκεκριμένες κλάσεις χρησιμοποιούνται για την αλληλεπίδραση με τα δεδομένα, δηλαδή την εισαγωγή, την ανάκτηση και την ενημέρωση των

βάσεων δεδομένων της εφαρμογής

- **View**: Υλοποιούν με γραφικό τρόπο την διεπαφή της εφαρμογής μέσω της οποίας αλληλεπιδρούν οι χρήστες.

- **Controller**: Ακούνε τα αιτήματα των χρηστών και εκτελούν τις κατάλληλες ενέργειες. Οι κλάσεις αυτές αλληλεπιδρούν με τις Model και επιλέγουν το κατάλληλο View για να εμφανίσουν στον χρήστη ανάλογα με τις ενέργειες που απαιτεί.

Βλέπουμε επομένως ότι το MVC είναι ένα πρότυπο 3 επιπέδων το οποίο ορίζει εκτός από τα επίπεδα και τις σχέσεις μεταξύ αυτών (Σχήμα 4.1). Πιο συγκεκριμένα, το επίπεδο του Controller αλληλεπιδρά με τον χρήστη λαμβάνοντας δεδομένα τα οποία τα αποστέλλει στο Model επίπεδο αφού πρώτα τα επικυρώσει. Επίσης, ανάλογα με το αίτημα του χρήστη θα ζητήσει τα κατάλληλα δεδομένα από το Model προκειμένου να τα αποστείλει στο επίπεδο View για να δημιουργηθεί η κατάλληλη απεικόνιση για τον χρήστη.

Τέλος, το συγκεκριμένο πρότυπο προσφέρει αρκετά πλεονεκτήματα όπως:

- Μας βοηθάει να ελέγξουμε την πολυπλοκότητα της εφαρμογής χωρίζοντας την σε 3 επίπεδα.

- Επιτρέπει ευκολότερες δοκιμές (testing) της εφαρμογής.

- Διαφορετικοί προγραμματιστές μπορούν να δουλεύουν ταυτόχρονα στα διαφορετικά επίπεδα κάτι που μειώνει τον χρόνο ανάπτυξης της εφαρμογής.

4.2 Απαιτησεις Συστηματος

Οι απαιτήσεις συστήματος εκφράζουν τις ανάγκες και τους περιορισμούς που τίθενται σε ένα σύστημα το οποίο προσπαθεί να λύσει ένα πραγματικό πρόβλημα [23].

Οι απαιτήσεις από το σύστημα χωρίζονται σε δύο κατηγορίες, τις λειτουργικές (**ΛΑ**) και τις μη λειτουργικές (**ΜΛΑ**) απαιτήσεις. Οι λειτουργικές απαιτήσεις αντανακλούν τη δυνατότητα εκτέλεσης ενεργειών που παρέχει το σύστημα στον χρήστη, ενώ οι μη λειτουργικές απαιτήσεις είναι αυτές που χαρακτηρίζουν το σύστημα και προσδιορίζουν ποιοτικά χαρακτηριστικά στις λειτουργίες του. Μπορεί, επίσης, να τεθούν περιορισμοί και στόχοι ως προς τη συμπεριφορά του συστήματος, όπως και απαιτήσεις ως προς την εξελιξιμότητά του.

4.2.1 Λειτουργικές Απαιτήσεις

ΛΑ-1

Ο χρήστης πρέπει να μπορεί να δίνει φωνητικές εντολές.

Περιγραφή: Ο χρήστης πρέπει να μπορεί να δίνει φωνητικές εντολής μέσω της φωνής του τις οποίες ο καθρέφτης θα εκτελεί.

User Priority (5/5): Η δυνατότητα φωνητικών εντολών είναι πάρα πολύ σημαντική για τον χρήστη αφού θα είναι ο κύριος τρόπος επικοινωνίας και αλληλεπίδρασής του με τον καθρέφτη.

Technical Priority (5/5): Η δυνατότητα φωνητικών εντολών είναι υψίστης σημασίας για το σύστημα, καθώς πρέπει να είναι σε θέση να αναγνωρίζει τις προθέσεις του χρήστη και να εκτελεί τις κατάλληλες ενέργειες.

ΛΑ-2

Ο χρήστης πρέπει να μπορεί να ανοίγει διαφορετικές εφαρμογές.

Περιγραφή: Ο χρήστης πρέπει να μπορεί να χρησιμοποιεί τις διαφορετικές εφαρμογές που είναι εγκατεστημένες στον καθρέφτη.

User Priority(5/5): Η δυνατότητα χρήστης διαφορετικών εφαρμογών είναι πολύ σημαντική για τον χρήστη, καθώς με αυτόν τον τρόπο μπορεί να εκτελέσει τις διαφορετικές λειτουργίες που του παρέχει ο καθρέφτης

Technical Priority (5/5): Η δυνατότητα χρήσης διαφορετικών εφαρμογών είναι πολύ σημαντική για το σύστημα, αφού θα πρέπει να είναι σε θέση να εκτελεί τις διαφορετικές εφαρμογές που είναι εγκατεστημένες στον καθρέφτη.

ΛΑ-3

Ο χρήστης πρέπει να μπορεί να εγκαθιστά διαφορετικές εφαρμογές.

Περιγραφή: Ο χρήστης πρέπει να μπορεί να εγκαθιστά στον καθρέφτη εφαρμογές που αναπτύσσει αυτός ή κάποιος τρίτος.

User Priority (4/5): Η δυνατότητα εγκατάστασης διαφορετικών εφαρμογών είναι αρκετά σημαντική για τον χρήστη, καθώς με αυτόν τον τρόπο θα μπορεί να επεκτείνει τις δυνατότητες του καθρέφτη.

Technical Priority (5/5): Η δυνατότητα εγκατάστασης διαφορετικών εφαρμογών εί-

ναι πολύ σημαντική για το σύστημα, αφού θα πρέπει να δέχεται εύκολα εξωτερικές εφαρμογές και να τις εκτελεί χωρίς προβλήματα.

ΛΑ-4

Ο χρήστης πρέπει να μπορεί να ενεργοποιεί και να απενεργοποιεί τον καθρέφτη με φωνητικές εντολές.

Περιγραφή: Ο χρήστης πρέπει να μπορεί να ενεργοποιεί και να απενεργοποιεί τον καθρέφτη δίνοντάς του κατάλληλες εντολές ανοίγματος/κλεισίματος.

User Priority (5/5): Η δυνατότητα ενεργοποίησης/απενεργοποίησης του καθρέφτη με εντολές είναι πολύ σημαντική για τον χρήστη, καθώς θα μπορεί να ελέγξει πότε ο καθρέφτης να είναι σε λειτουργία

Technical Priority (5/5): Η δυνατότητα ενεργοποίησης/απενεργοποίησης του καθρέφτη με εντολές είναι πολύ σημαντική για το σύστημα, καθώς θα μπορεί να ελεγχθεί η κατάσταση λειτουργίας του.

ΛΑ-5

Ο χρήστης πρέπει να μπορεί να αλλάζει τις ρυθμίσεις του καθρέφτη.

Περιγραφή: Ο χρήστης πρέπει να μπορεί να αλλάζει τις ρυθμίσεις του καθρέφτη είτε μέσω εντολών είτε μέσω κειμένου.

User Priority (5/5): Η δυνατότητα αλλαγής των ρυθμίσεων είναι πολύ σημαντική για τον χρήστη, καθώς θα μπορεί να εξατομικεύσει τις λειτουργίες του καθρέφτη ανάλογα με τα θέλω του.

Technical Priority (5/5): Η δυνατότητα αλλαγής των ρυθμίσεων είναι πολύ σημαντική για το σύστημα, καθώς θα είναι εφικτή η ορθή και εξατομικευμένη λειτουργία των εφαρμογών του καθρέφτη.

ΛΑ-6

Ο χρήστης πρέπει να μπορεί να δει την ημερομηνία και ώρα

Περιγραφή: Ο χρήστης πρέπει να μπορεί να δει την τωρινή ημερομηνία και την ώρα στην αρχική οθόνη του καθρέφτη

User Priority (5/5): Η δυνατότητα ενημέρωσης για την ημερομηνία και ώρα είναι πολύ σημαντική για τον χρήστη προκειμένου να μπορεί να οργανώσει καλύτερα το

πρόγραμμά του.

Technical Priority (5/5): Η δυνατότητα ενημέρωσης για την ημερομηνία και ώρα είναι πολύ σημαντική για το σύστημα αφού με βάση αυτά μπορεί να επηρεαστεί η λειτουργία του.

ΛΑ-7

Ο χρήστης πρέπει να μπορεί να δει τον καιρό σε ζωντανό χρόνο.

Περιγραφή: Ο χρήστης πρέπει να μπορεί να δει τον καιρό σε ζωντανό χρόνο στην αρχική οθόνη του καθρέφτη.

User Priority (5/5): Η δυνατότητα ζωντανής ενημέρωσης του καιρού είναι πολύ σημαντική για τον χρήστη, καθώς μπορεί να οργανώσει καλύτερα το πρόγραμμά του.

Technical Priority (4/5): Η δυνατότητα ζωντανής ενημέρωσης του καιρού είναι αρκετά σημαντική για το σύστημα αφού μπορεί να χρησιμοποιήσει τις πληροφορίες για βελτίωση της λειτουργίας του

ΛΑ-8

Ο χρήστης πρέπει να μπορεί να κάνει ερωτήσεις σχετικά με τον καιρό.

Περιγραφή: Ο χρήστης πρέπει να μπορεί να κάνει ερωτήσεις και να ενημερώνεται σχετικά με τον καιρό.

User Priority (4/5): Η δυνατότητα ερωτήσεων σχετικά με τον καιρό είναι αρκετά σημαντική για τον χρήστη αφού μπορεί να ενημερωθεί και να οργανώσει καλύτερα το πρόγραμμά του.

Technical Priority (2/5): Η δυνατότητα ερωτήσεων σχετικά με τον καιρό δεν είναι πολύ σημαντική για την εύρυθμη λειτουργία του συστήματος αφού μπορεί να λειτουργήσει ομαλά και χωρίς αυτήν.

ΛΑ-9

Ο χρήστης πρέπει να μπορεί να

Περιγραφή:

User Priority (5/5):

Technical Priority (4/5):

ΛΑ-10

Ο χρήστης πρέπει να μπορεί να

Περιγραφή:

User Priority (5/5):

Technical Priority (4/5):

ΛΑ-10

Ο χρήστης πρέπει να μπορεί να

Περιγραφή:

User Priority (5/5):

Technical Priority (4/5):

ΛΑ-10

Ο χρήστης πρέπει να μπορεί να

Περιγραφή:

User Priority (5/5):

Technical Priority (4/5):

4.2.2 Μη Λειτουργικές Απαιτήσεις

ΜΛΑ-1

Το σύστημα πρέπει να αποκρίνεται στις εντολές του χρήστη σε λιγότερο από 500ms.

Περιγραφή: Το σύστημα πρέπει να αναγνωρίζει τις προθέσεις του χρήστη και να ανταποκρίνεται κατάλληλα σε λιγότερο από 500ms.

User Priority (5/5): Η απαίτηση αυτή είναι πολύ σημαντική για τον χρήστη, καθώς θέλει ο καθρέφτης να του προσφέρει μια διαδραστική εμπειρία πραγματικού χρόνου.

Technical Priority (5/5): Η απαίτηση αυτή είναι πολύ σημαντική για το σύστημα, καθώς θα καθορίσει τα τεχνικά χαρακτηριστικά και τη δομή του, προκειμένου να καλύπτεται η απαιτούμενη ταχύτητα απόκρισης (500ms).

ΜΛΑ-2

Το σύστημα πρέπει να ανταποκρίνεται ορθά όταν η εντολή του χρήστη δεν αναγνωρίζεται

Περιγραφή: Το σύστημα πρέπει να χειρίζεται εντολές του χρήστη που δεν αναγνωρίζονται και να μην καταρρέει.

User Priority (5/5): Η απαίτηση αυτή είναι πολύ σημαντική για τον χρήστη, καθώς θέλει ο καθρέφτης να λειτουργεί χωρίς σφάλματα.

Technical Priority (5/5): Η απαίτηση αυτή είναι πολύ σημαντική για το σύστημα, καθώς ο χειρισμός των ''κακών'' εισόδων θα βοηθήσει τον καθρέφτη να λειτουργεί ορθά χωρίς σφάλματα.

ΜΛΑ-3

Το σύστημα πρέπει να επιτρέπει την προσάρτηση εφαρμογών.

Περιγραφή: Το σύστημα πρέπει να επιτρέπει την προσάρτηση εφαρμογών και την εύκολη ενσωμάτωσή τους στην ροή του προγράμματος.

User Priority (5/5): Η απαίτηση αυτή είναι πολύ σημαντική για τον χρήστη, καθώς η ενσωμάτωση εξωτερικών εφαρμογών θα επεκτείνει τις δυνατότητες που θα του δίνει ο καθρέφτης.

Technical Priority (5/5): Η απαίτηση αυτή είναι πολύ σημαντική για το σύστημα, καθώς θα πρέπει να σχεδιαστεί με τέτοιο τρόπο ώστε να είναι εύκολη η επεκτασιμότητα των εφαρμογών του καθρέφτη.

ΜΛΑ-4

Το σύστημα πρέπει να επιτρέπει την ταυτόχρονη εμφάνιση εφαρμογών και του ειδώλου του χρήστη.

Περιγραφή: Το σύστημα πρέπει να έχει ανακλαστική οθόνη στην οποία ο χρήστης να μπορεί να δει τόσο τις εφαρμογές του καθρέφτη όσο και το είδωλό του.

User Priority (5/5): Η απαίτηση αυτή είναι πολύ σημαντική για τον χρήστη, καθώς θέλει να μπορεί να αλληλεπιδρά με τον καθρέφτη καθώς βλέπει το είδωλό του.

Technical Priority (5/5): Η απαίτηση αυτήν είναι πολύ σημαντική για το σύστημα, καθώς θα καθορίσει το είδος της οθόνης που θα χρησιμοποιηθεί και την τοποθέτηση των εφαρμογών πάνω σε αυτήν.

ΜΛΑ-5

Το σύστημα πρέπει να υποστηρίζει σύνδεση με το Wit.ai.

Περιγραφή: Το σύστημα πρέπει να υποστηρίζει σύνδεση με το Wit.ai προκειμένου να επιτρέπεται η αναγνώριση πρόθεσης των εντολών του χρήστη.

User Priority (5/5): Η απαίτηση αυτή είναι πολύ σημαντική για τον χρήστη, καθώς ο καθρέφτης πρέπει να αναγνωρίζει τις προθέσεις του και να εκτελεί τις κατάλληλες ενέργειες.

Technical Priority (5/5): Η απαίτηση αυτή είναι πολύ σημαντική για το σύστημα, καθώς πρέπει να μπορεί να αναγνωρίζει τις προθέσεις του χρήστη για να εκτέλεση των επιθυμητών ενεργειών.

ΜΛΑ-6

Το σύστημα πρέπει να λειτουργεί μόνο όταν ανιχνεύει πρόσωπο ανθρώπου.

Περιγραφή: Το σύστημα πρέπει να ανιχνεύει την ύπαρξη ανθρώπου μπροστά στον καθρέφτη και να σταματά τη λειτουργία του όταν δεν βρίσκει τίποτα.

User Priority (4/5): Η απαίτηση αυτή είναι σημαντική για τον χρήστη, καθώς δεν επιθυμεί ο καθρέφτης να ανιχνεύει τα λόγια του σε ανύποπτο χρόνο και να πιάνει εντολές χωρίς πρόθεση.

Technical Priority (5/5): Η απαίτηση αυτήν είναι πολύ σημαντική για το σύστημα, καθώς θα πρέπει να ελέγχει συνεχώς την ύπαρξη χρήστη μπροστά στον καθρέφτη και να σταματά τη λειτουργία του. Η απαίτηση αυτή θα βοηθήσει και στην εξοικονόμηση πόρων του συστήματος.

ΜΛΑ-7

Το σύστημα πρέπει να συνδέεται στο διαδίκτυο με ικανοποιητικό εύρος σύνδεσης.

Περιγραφή: Το σύστημα πρέπει να υποστηρίζει σύνδεση στο διαδίκτυο προκειμένου να μπορεί να αλληλεπιδρά με τα διαφορετικά APIs και να λειτουργεί ορθά. Επομένως πρέπει να υπάρχει η απαραίτητη υποστήριξη σύνδεσης στο διαδίκτυο από το υλικό του συστήματος.

User Priority (5/5): Η απαίτηση αυτή είναι πολύ σημαντική για τον χρήστη, καθώς χωρίς σύνδεση διαδικτύου ο καθρέφτης δεν θα είναι σε θέση να εκτελέσει τις εφαρμογές του και να αναγνωρίσει τις εντολές του χρήστη.

Technical Priority (5/5): Η απαίτηση αυτή είναι πολύ σημαντική για το σύστημα, καθώς χρειάζεται σύνδεση στο διαδίκτυο για να λειτουργήσει ο καθρέφτης τις εφαρμογές του.

ΜΛΑ-8

Το σύστημα πρέπει να

Περιγραφή:

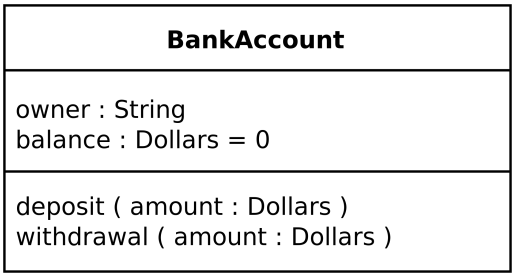
User Priority (5/5):

Technical Priority (4/5):

4.3 Μοντελοποιηση Κλασεων

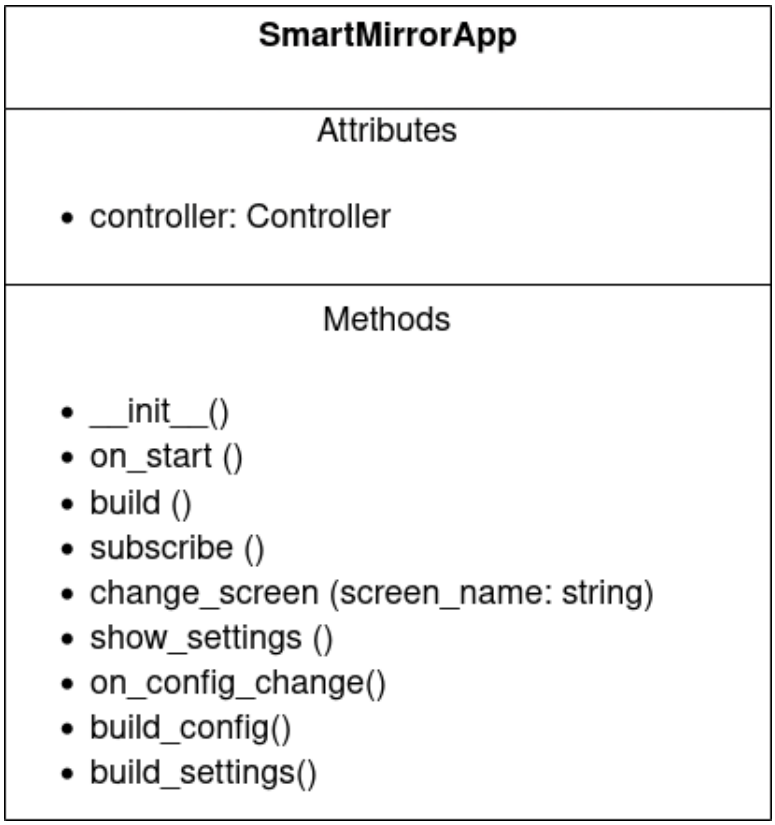
Η οργάνωση του κώδικα έγινε με την μορφή κλάσεων στη γλώσσα Python ενώ όπως έχει αναφερθεί παραπάνω για την γραφική διεπαφή χρησιμοποιήθηκε η βιβλιοθήκη Kivy. Στο παρόν κεφάλαιο θα παρουσιαστούν όλες οι κλάσεις που υλοποιήθηκαν για το λογισμικό του καθρέφτη μαζί με τις μεταβλητές και μεθόδους που ανήκουν σε κάθε κλάση, ενώ θα γίνει και μια περιγραφή του κάθε μέλους. Με τέτοιου είδους διαγράμματα είναι εφικτό να μοντελοποιηθεί το λογισμικό σε ανώτερο επίπεδο και χωρίς να χρειαστεί να κοιτάξουμε άμεσα τον κώδικα [24]. Η περιγραφή των κλάσεων θα γίνει στην γλώσσα UML[13] κατά την οποία, όπως φαίνεται στο Σχήμα 4.2, μια κλάση αναπαριστάται από 3 μέρη, το όνομά της, τα χαρακτηριστικά (μεταβλητές) της και τέλος οι μέθοδοί της.

[13] https://en.wikipedia.org/wiki/Unified_Modeling_Language



Σχήμα 4.2: Παράδειγμα αναπαράστασης κλάσης στη UML

4.3.1 SmartMirrorApp



Σχήμα 4.3: UML της κλάσης SmartMirrorApp

Περιγραφή Κλάσης

Η κλάση αυτή κληρονομεί από την κλάση kivy.app.App[14] η οποία είναι η βασική κλάση για την δημιουργία μιας Kivy εφαρμογής. Μέσως αυτής της κλάσης ελέγχουμε την γραφική εφαρμογή και όταν όλα είναι έτοιμα ξεκινάμε τον κύκλο ζωής της εφαρμογής καλώντας την μέθοδο `SmartMirrorApp().run()`.

[14] https://kivy.org/doc/stable/api-kivy.app.html#kivy.app.App

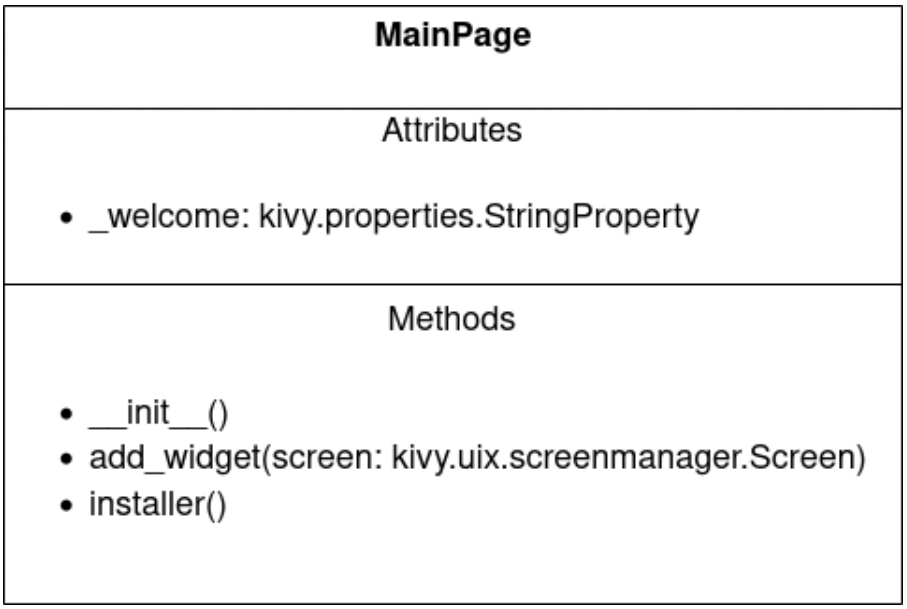
Χαρακτηριστικά Κλάσης

- `controller`: Αντικείμενο τύπου Controller για την ενεργοποίηση του thread ελέγχου.

Μέθοδοι Κλάσης

- `__init__()`: Συνάρτηση δόμησης της κλάσης η οποία καλή την συνάρτηση δόμησης της κλάσης `kivy.app.App` μέσω του `super()` και αρχικοποιεί τη μεταβλητή `controller` να είναι `None`.
- `on_start()`: Ειδική μέθοδος της κλάσης `kivy.app.App` που εκτελείται λίγο πριν την εκκίνηση της εφαρμογής (αμέσως μετά την συνάρτηση `build()`. Δημιουργεί το αντικείμενο `Controller` και ξεκινάει το thread ελέγχου.
- `build()`: Ειδική μέθοδος της κλάσης `kivy.app.App` η οποία αρχικοποιεί την εφαρμογή και εκτελείται μόνο μία φορά. Φορτώνει όλες τις εγκατεστημένες εφαρμογές και επιστρέφει το βασικό Widget (ένα στιγμιότυπο της κλάσης MainPage η οποία θα εξηγηθεί παρακάτω).
- `subscribe()`: Μέθοδος η οποία επιστρέφει ένα Python dictionary όπου γίνεται μια αντιστοίχηση ενός intent σε μια συνάρτηση. Η μέθοδος αυτή χρησιμοποιείται προκειμένου να μπορεί να εκτελεστεί η σωστή συνάρτηση όταν γίνει αναγνώριση της πρόθεσης της εντολής του χρήστη από το Wit.ai.
- `change_screen(screen_name: string)`: Συνάρτηση που χρησιμοποείται για να γίνει αλλαγή Screen δηλαδή να φορτώσει μια άλλη εφαρμογή με όνομα `screen_name`.
- `show_settings()`: Μέθοδος η οποία καλεί την συνάρτηση `open_settings` της κλάσης `kivy.app.App` για να ανοίξει το μενού ρυθμίσεων της εφαρμογής.
- `on_config_change()`: Ειδική μέθοδος της κλάσης `kivy.app.App` η οποία εκτελείται όταν γίνει κάποια αλλαγή ρύθμισης. Για κάθε ένα από τα εγκατεστημένα Widgets ανανεώνει τις ρυθμίσεις τους.
- `build_config()`: Ειδική μέθοδος της κλάσης `kivy.app.App` η οποία εκτελείται πριν την αρχικοποίηση της εφαρμογής και δημιουργεί default ρυθμίσεις για κάθε ένα από τα εγκατεστημένα Widgets.
- `build_settings()`: Ειδική μέθοδος της κλάσης `kivy.app.App` η οποία εκτελείται όταν θέλουμε να δείξουμε τις ρυθμίσεις της εφαρμογής. Διαβάζει από κάθε εγκατεστημένο Widget τις ρυθμίσεις τους (οι οποίες βρίσκονται σε ένα αρχείο ”settings.json” και τις προσθέτει στο μενού ρυθμίσεων.

4.3.2 MainPage



Σχήμα 4.4: UML της κλάσης MainPage

Περιγραφή Κλάσης

Η κλάση αυτή κληρονομεί από την κλάση ScreenManager[15] που περιέχεται στο πακέτο `kivy.uix.screenmanager`. Η κλάση αυτή είναι το πρωτεύον Widget που επιστρέφει η εφαρμογή και κάτω από αυτήν μπαίνουν όλες οι εφαρμογές που εγκαθιστά ο χρήστης (σαν καινούργια Screens).

Χαρακτηριστικά Κλάσης

- `_welcome`: Μεταβλητή που χρησιμοποιείται για να αποθηκεύσει ένα μήνυμα που καλωσορίζει τον χρήστη.

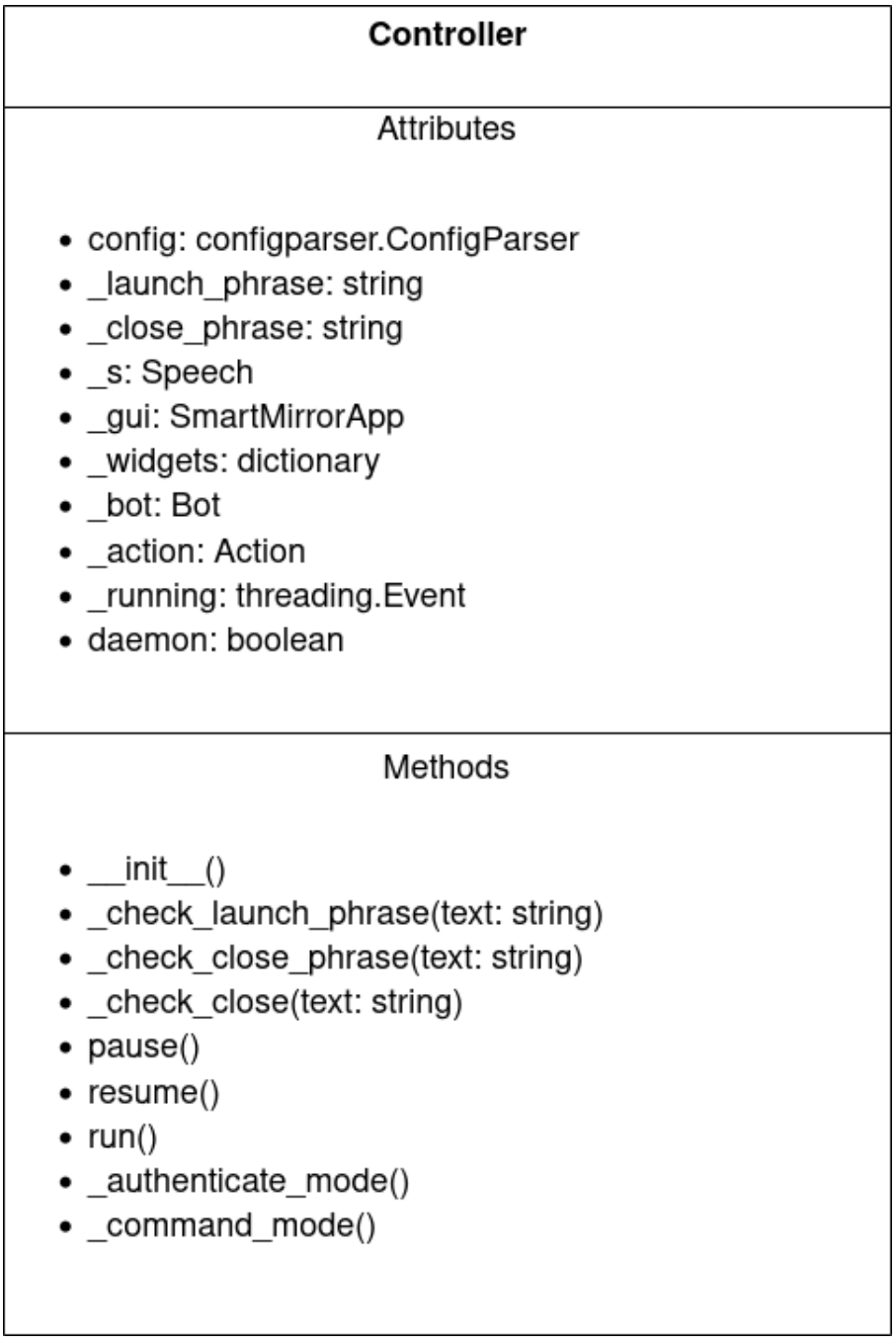
Μέθοδοι Κλάσης

- `__init__()`: Η συνάρτηση δόμησης της κλάσης η οποία καλή την συνάρτηση δόμησης της κλάσης `kivy.uix.screenmanager.ScreenManager` μέσω του `super` και αρχικοποιεί τη μεταβλητή `_welcome` να πάρει μία τυχαία επιλογή από μια λίστα μηνυμάτων. Τέλος καλεί την εσωτερική συνάρτηση `installer` προκειμένου να εγκαταστήσει όλες τις εφαρμογές που βρίσκονται στον φάκελο `/widgets` σαν ξεχωριστά Screens.
- `add_widget(screen: kivy.uix.screenmanager.Screen)`: Ειδική συνάρτηση της κλάσης `kivy.uix.screenmanager.ScreenManager` η οποία προσθέτει ένα καινούργιο Widget στα παιδιά του τρέχοντος Widget. Στην υπερφορτωμένη της έκδοση, επιτρέπεται η πρόσθεση μόνο ενός Widget τύπου `kivy.uix.screenmanager.Screen` για περισσότερη ασφάλεια.

[15] https://kivy.org/doc/stable/api-kivy.uix.screenmanager.html#kivy.uix.screenmanager.ScreenManager

- `installer()`: Συνάρτηση η οποία για κάθε ένα από τα παιδιά της `MainPage` καλεί την συνάρτηση `install` την οποία πρέπει να ορίσουν οι εξωτερικές εφαρμογές προκειμένου να μπορούν να εγκατασταθούν από τον καθρέφτη.

4.3.3 Controller



Σχήμα 4.5: UML της κλάσης Controller

Περιγραφή Κλάσης

Η κλάση αυτή κληρονομεί από την κλάση threading.Thread[16]. Ο controller είναι ένα ανεξάρτητο thread από το main thread στο οποίο τρέχει η γραφική εφαρμογή και είναι υπεύθυνος για τον έλεγχο των λειτουργιών του συστήματος. Μέσα από αυτήν την κλάση ελέγχεται η ροή του προγράμματος, επιλέγονται οι ενέργειες που θα εκτελέσει ο καθρέφτης και ανανεώνεται το γραφικό περιβάλλον με βάση τις εντολές του χρήστη.

Χαρακτηριστικά Κλάσης

- `config`: Αντικείμενο τύπου `configparser.ConfigParser` για την ανάγνωση ρυθμίσεων.
- `_launch_phrase`: Μεταβλητή για την αποθήκευση της φράσης ενεργοποίησης του καθρέφτη.
- `_close_phrase`: Μεταβλητή για την αποθήκευση της φράσης απενεργοποίησης του καθρέφτη.
- `_s`: Αντικείμενο τύπου `Speech` το οποίο χρησιμοποιείται για την αναγνώριση της φωνής του χρήστη.
- `_gui`: Αντικείμενο τύπου `SmartMirrorApp` το οποίο χρησιμοποιείται για να γίνεται αλληλεπίδραση μεταξύ του Controller και του γραφικού περιβάλλοντος.
- `_widgets`: Dictionary το οποίο αποθηκεύει όλα τα Widgets του γραφικού περιβάλοντος με τα ids τους ως κλειδιά.
- `_bot`: Αντικείμενο τύπου `Bot` το οποίο χρησιμοποιείται για την αλληλεπίδραση με το Wit.ai.
- `_action`: Αντικείμενο τύπου `Action` το οποίο χρησιμοποιείται ως βοηθημα για την εκτέλεση της εντολής του χρήστη.
- `_running`: Μεταβλητή τύπου `threading.Event` η οποία χρησιμοποιείται για τον έλεγχο του Thread του Controller.
- `daemon`: Μεταβλητή που καθορίζει αν το Thread θα εκτελεστεί στο προσκήνιο ή στο παρασκήνιο.

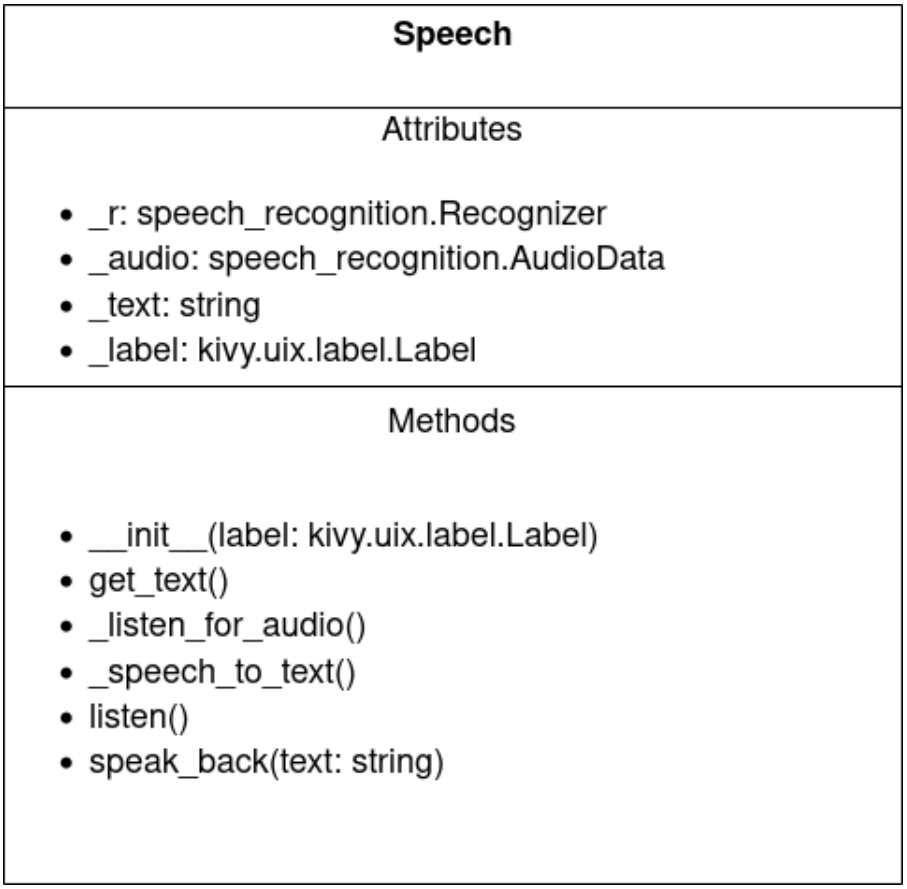
Μέθοδοι Κλάσης

- `__init__()`: Η συνάρτηση δόμησης της κλάσης διαβάζει από τις ρυθμίσεις τις φράσεις ενεργοποίησης/απενεργοποίησης, αρχικοποιεί τα αντικείμενα των βοηθητικών κλάσεων που θα χρησιμοποιηθούν για τον έλεγχο του καθρέφτη και τέλος ξεκινάει ένα Thread στο παρασκήνιο στο οποίο θα γίνεται ο λογικός έλεγχος της ροής του προγράμματος.
- `_check_launch_phrase(text: String)`: Βοηθητική συνάρτηση όπου επιστρέφει True αν η μεταβλητή `text` ταυτίζεται με την φράση έναρξης και False σε αντίθετη περίπτωση.

[16] https://docs.python.org/3/library/threading.html#threading.Thread

- `_check_close_phrase(text: String)`: Βοηθητική συνάρτηση όπου επιστρέφει True αν η μεταβλητή `text` ταυτίζεται με την φράση απενεργοποίησης και False σε αντίθετη περίπτωση.
- `_check_close(text: String)`: Βοηθητική συνάρτηση που ελέγχει εαν η μεταβλητή `text` είναι ίση με `"quit"` ή `"exit"` προκειμένου να διακόψει τη λειτουργία του καθρέφτη.
- `pause()`: Βοηθητική συνάρτηση η οποία σταματάει το Thread του Controller.
- `resume()`: Βοηθητική συνάρτηση η οποία ενεργοποιεί το Thread του Controller.
- `run()`: Η συνάρτηση αυτή ενεργοποιεί το Thread του Controller και καλεί τις συναρτήσεις `_authenticate_mode()` και `_command_mode()` η οποίες ελέγχουν την κατάσταση του καθρέφτη.
- `_authenticate_mode()`: Στη συγκεκριμένη κατάσταση ο καθρέφτης είναι αδρανής και περιμένει να ακούσει την φράση ενεργοποίησης προκειμένου να δώσει πρόσβαση στον χρήστη και να ακολουθεί τις εντολές του. Με αυτόν τον τρόπο ο καθρέφτης θα ακούει τις εντολές του χρήστη μόνο όταν αυτός το θέλει.
- `_command_mode()`: Στη συγκεκριμένη κατάσταση γίνεται ο έλεγχος της ροής του καθρέφτη όπου γίνεται επαναληπτικά η ακοή της εντολής του χρήστη, η αλληλεπίδραση με το Wit.ai για την αναγνώριση της πρόθεσης του χρήστη και η εκτέλεση της εντολής αν αυτή είναι αποδεκτή. Η επανάληψη συνεχίζεται έως ότου ο χρήστης πει την φράση απενεργοποίησης ή μία εκ των εντολών ”quit” και ”exit” οι οποίες απενεργοποιούν τον καθρέφτη.

4.3.4 Speech



Σχήμα 4.6: UML της κλάσης Speech

Περιγραφή Κλάσης

Η κλάση αυτή υλοποιεί όλες τις μεθόδους και την διεπαφή για την ανάγνωση και ομιλία ήχου. Μέσω αυτής της κλάσης ο καθρέφτης μπορεί να ακούει τον χρήστη από το μικρόφωνο ανά τακτά διαστήματα και να μετατρέπει τις εντολές του σε κείμενο το οποίο θα χρησιμοποιηθεί αργότερα για την αναγνώριση της πρόθεσης. Τέλος, μέσω αυτής της κλάσης ο καθρέφτης δύναται να μιλήσει στον χρήστη, λέγοντάς του αποτελέσματα ενεργειών ή πληροφορίες.

Χαρακτηριστικά Κλάσης

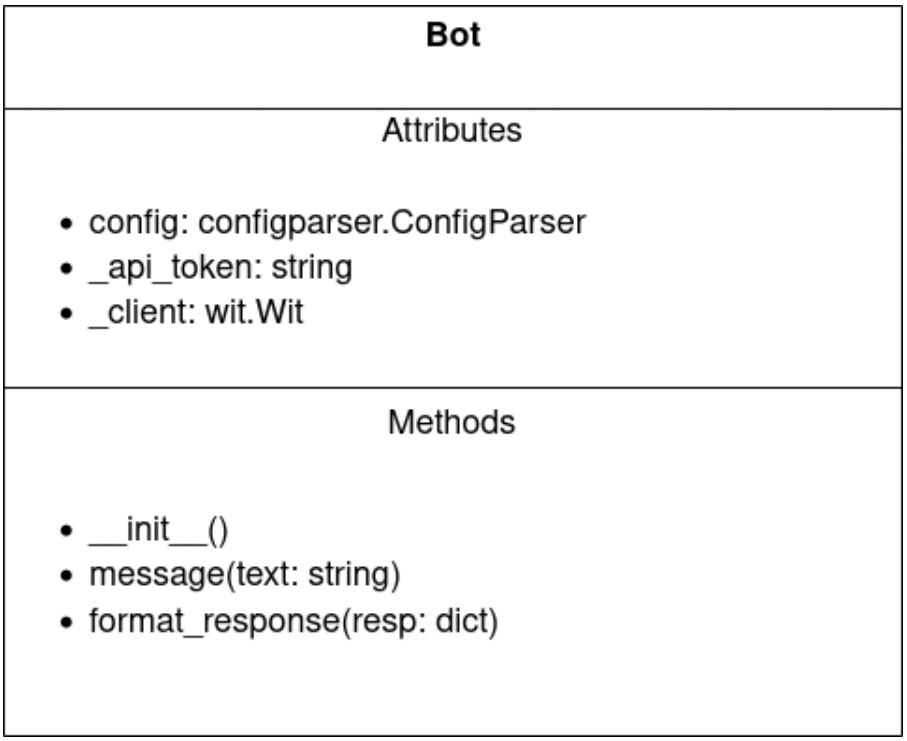
- `_r`: Αντικείμενο τύπου `speech_recognition.Recognizer` μέσω του οποίο γίνεται η ανάγνωση της φωνής και η αναγνώρισή της σε κείμενο.
- `_audio`: Αντικείμενο τύπου `speech_recognition.AudioData` το οποίο αποθηκεύει τα δεδομένα του ήχου τα οποία χρησιμοποιούνται για την αναγνώριση και μετατροπή σε κείμενο.
- `_text`: Μεταβλητή στην οποία αποθηκεύεται σε γραπτή μορφή η εντολή του χρήστη μετά την αναγνώριση.

- `_label`: Αντικείμενο τύπου `kivy.uix.label.Label` το οποίο χρησιμοποιείται προκειμένου να εμφανίζεται στην οθόνη του καθρέφτη πότε το σύστημα ακούει εντολές από τον χρήστη.

Μέθοδοι Κλάσης

- `__init__(kivy.uix.label.Label)`: Συνάρτηση δόμησης της κλάσης η οποία αρχικοποιεί τις μεταβλητές που χρησιμποιεί η κλάση και δέχεται ως είσοδο το Label στο οποίο θα υποδηλώνεται πότε ο καθρέφτης ακούει για εντολές.
- `get_text()`: Βοηθητική συνάρτηση η οποία επιστρέφει την τιμή της μεταβλητής `_text`.
- `_listen_for_audio()`: Βοηθητική συνάρτηση η οποία ακούει από το μικρόφωνο τις εντολές του χρήστη. Η λειτουργία ανάγνωσης του ήχου από το μικρόφωνο του συστήματος γίνεται με την αρχικοποίηση ενός αντικειμένου τύπου `speech_recognition.Microphone`. Τα δεδομένα ήχου αποθηκεύονται στο αντικείμενο `_audio` ενώ γίνεται και εγγραφή στο Label του καθρέφτη για την ενημέρωση του χρήστη ότι το σύστημα περιμένει εντολή.
- `_speech_to_text()`: Συνάρτηση που καλεί την μέθοδο `recognize_google()` του αντικειμένου `_r` η οποία αλληλεπιδρά με το API της Google για την μετατροπή των δεδομένων ήχου σε κείμενο. Σε περίπτωση που δεν γίνεται αναγνώριση το κείμενο τίθεται κενό.
- `listen()`: Συνάρτηση η οποία καλείται από τον Controller και καλεί τις 2 συναρτήσεις για ανάγνωση του ήχου και μετατροπή σε κείμενο.
- `speak_back(text: string)`: Συνάρτηση η οποία καλείται εξωτερικά για την ομιλία του καθρέφτη προς τον χρήστη. Μέσω της βιβλιοθήκης gTTS μετατρέπεται το κειμενο εισόδου σε ομιλία, αποθηκεύεται ως ένα προσωρινό αρχείο .mp3 και αναπαράγεται μέσω της βιβλιοθήκης pydub. Τέλος, το προσωρινό αρχείο διαγράφεται.

4.3.5 Bot



Σχήμα 4.7: UML της κλάσης Bot

Περιγραφή Κλάσης

Η κλάση αυτή αποτελεί την διεπαφή για την αλληλεπίδραση με το Wit.ai και την εξαγωγή της πρόθεσης της εντολής του χρήστη αλλά και των οντοτήτων οι οποίες θα δοθούν ως ορίσματα κατά την εκτέλεση των εντολών.

Χαρακτηριστικά Κλάσης

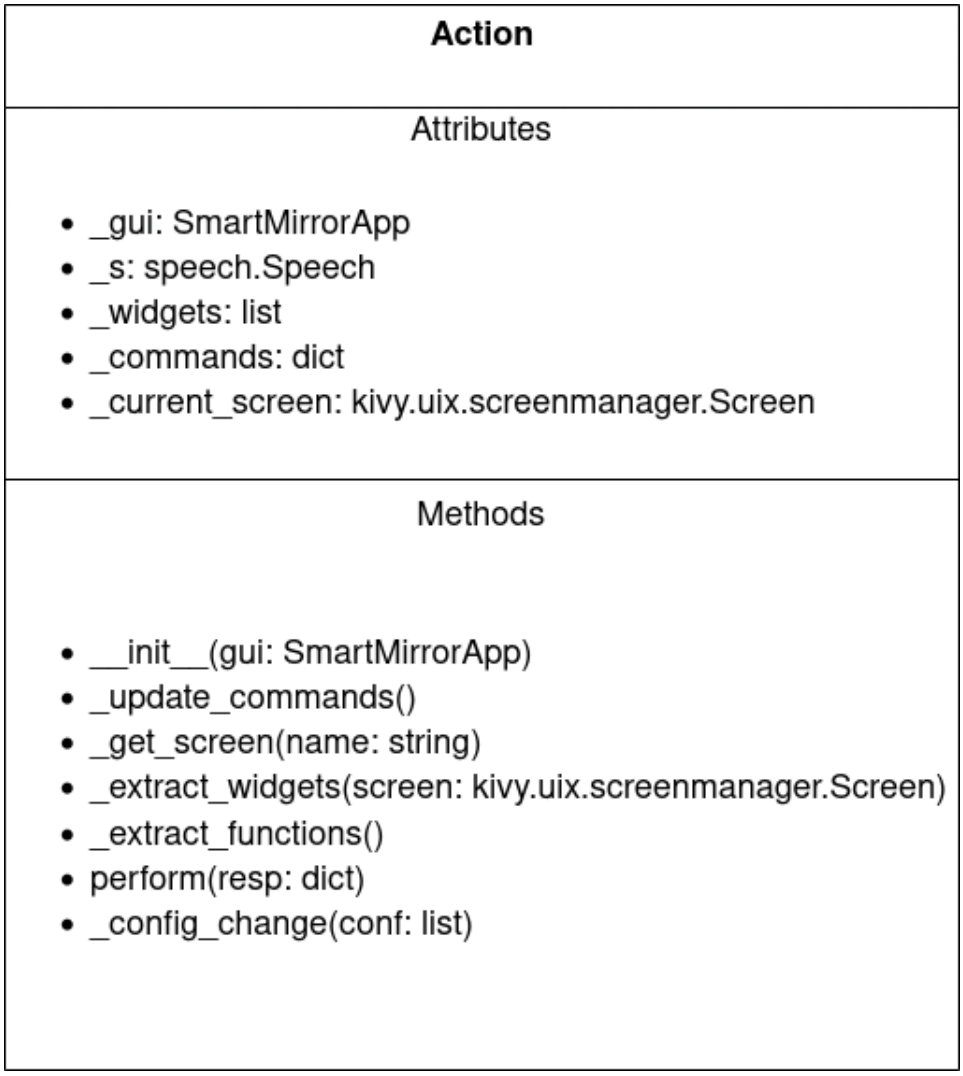
- `config`: Αντικείμενο τύπου `configparser.ConfigParser` για την ανάγνωση ρυθμίσεων.
- `_api_token`: Μεταβλητή η οποία αποθηκεύει το api_key που χρησιμοποιείται για να αποκτήσει η εφαρμογή πρόσβαση στο Wit.ai.
- `_client`: Αντικείμενο τύπου `wit.Wit` μέσω του οποίου θα γίνεται η αλληλεπίδραση με το Wit.ai.

Μέθοδοι Κλάσης

- `__init__()`: Συνάρτηση δόμησης της κλάσης η οποία διαβάζει το κλειδί από τις ρυθμίσεις και αρχικοποιεί ένα αντικείμενο τύπου `wit.Wit`.
- `message(text: string)`: Η συνάρτηση που καλείται από τον Controller, στέλνει την εντολή σε μορφή κειμένου στο Wit.ai και επιστρέφει την πρόθεση και τις οντότητες που εξάγονται. Σε περίπτωση που δεν αναγνωριστεί η εντολή γίνεται κατάλληλος έλεγχος σφάλματος.

- `format_response(resp: dict)`: Βοηθητική συνάρτηση η οποία δέχεται ως είσοδο το αποτέλεσμα του Wit.ai και μορφοποιεί την πρόθεση και τις οντότητες στην επιθυμητή μορφή ενός dictionary.

4.3.6 Action



Σχήμα 4.8: UML της κλάσης Action

Περιγραφή Κλάσης

Η κλάση αυτή αποτελεί την διεπαφή για την εκτέλεση των εντολών του χρήστη και είναι στην ουσία μια επέκταση του Controller. Μέσω της συγκεκριμένης κλάσης, γίνεται η μετατροπή της πρόθεσης που παίρνει το σύστημα από το Wit.ai σε εκτέλεση της κατάλληλης πράξης που υποδηλώνεται από την πρόθεση (αν αυτή υπάρχει και είναι προσβάσιμη).

Χαρακτηριστικά Κλάσης

- `_gui`: Αντικείμενο τύπου `SmartMirrorApp` το οποίο χρησιμοποιείται για να γίνεται αλληλεπίδραση μεταξύ του Controller και του γραφικού περιβάλλοντος.
- `_s`: Αντικείμενο τύπου `Speech` το οποίο χρησιμοποιείται για την ομιλία του καθρέφτη προς τον χρήστη.
- `_widgets`: Μία λίστα που αποθηκεύει όλα τα Widgets που βρίσκονται στην τωρινή οθόνη και το αντικείμενο _gui.
- `_commands`: Ένα dictionary που αποθηκεύει τις δυνατές εντολές που εξάγει κάθε Widget ως κλειδιά μαζί με τις αντίστοιχες συναρτήσεις που θα κληθούν για την κάθε εντολή ως τιμές.
- `_current_screen`: Αντικείμενο τύπου kivy.uix.screenmanager.Screen το οποίο έχει την τιμή της οθόνης στην οποία βρίσκεται κάθε στιγμή ο καθρέφτης.

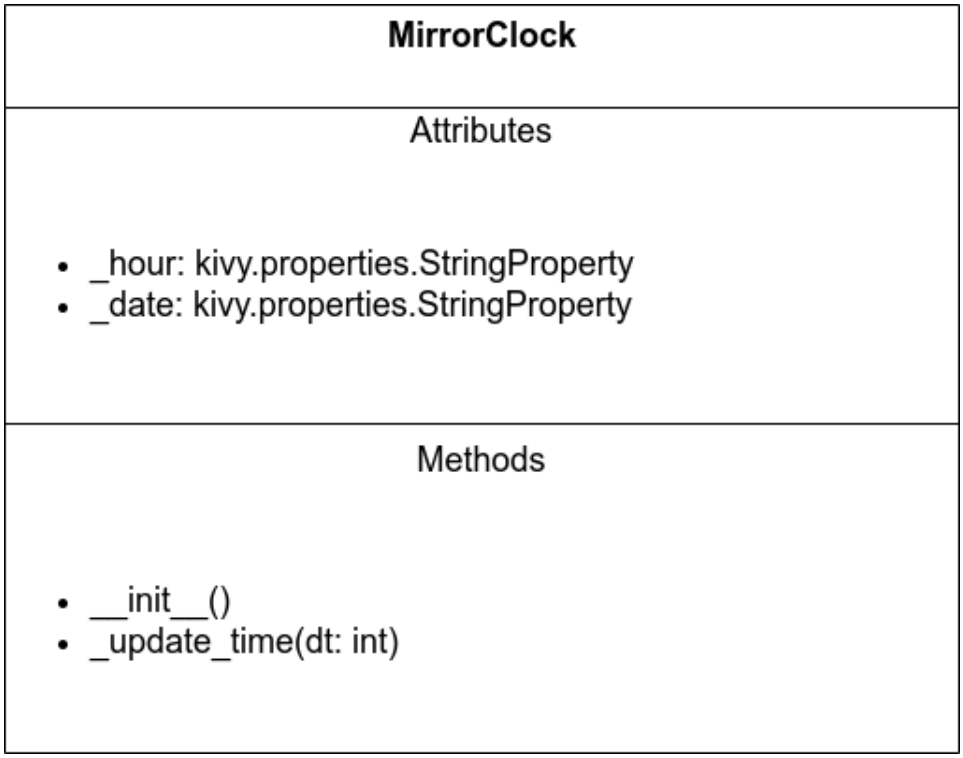
Μέθοδοι Κλάσης

- `__init__(gui: SmartMirrorApp)`: Συνάρτηση δόμησης της κλάσης η οποία δέχεται ως είσοδο ένα αντικείμενο τύπου `SmartMirrorApp` για τον έλεγχο της γραφικής εφαρμογής, αρχικοποιεί ένα αντικείμενο τύπου `Speech` για την διαχείριση της φωνής του καθρέφτη και τέλος καλεί την βοηθητική συνάρτηση `_update_commands` για να αρχικοποιηθούν οι διαθέσιμες εντολές στην τωρινή κατάσταση του καθρέφτη.
- `_update_commands()`: Βοηθητική συνάρτηση η οποία καθορίζει τις διαθέσιμες εντολές που μπορεί να εκτελέσει ο καθρέφτης. Αρχικά εξάγεται η τωρινή οθόνη, στην συνέχεια εξάγονται όλα τα Widgets που είναι διαθέσιμα στην οθόνη και διαθέτουν την μέθοδο `subscribe` ενώ πάντα μετά προστίθεται το αντικείμενο της εφαρμογής προκειμένου να είναι διαθέσιμες οι κάποιες βασικές εντολές, ενώ τέλος αποθηκεύονται οι συναρτήσεις που εκθέτουν τα παραπάνω Widgets μέσω της συνάρτησης subscribe.
- `_get_screen(name: string)`: Βοηθητική συνάρτηση η οποία δέχεται το όνομα μιας οθόνης και επιστρέφει ένα αντικείμενο αυτής.
- `_extract_widgets(screen: kivy.uix.screenmanager.Screen)`: Βοηθητική συνάρτηση η οποία δέχεται ένα αντικείμενο τύπου `Screen` και επιστρέφει μια λίστα με όλα τα Widgets που βρίσκονται στην οθόνη και διαθέτουν συνάρτηση subscribe.
- `_extract_functions()`: Βοηθητική συνάρτηση που ελέγχει όλα τα διαθέσιμα widgets και διαβάζει τις εντολές που εκθέτουν μέσω της συνάρτησης subscribe. Οι εντολές έχουν ως κλειδί την πρόθεση που θα έχει η εντολή και το αντικείμενο της συνάρτησης που θα κληθεί ως τιμή.
- `perform(resp: dict)`: Συνάρτηση που δέχεται ως είσοδο ένα dictionary με την πρόθεση και τα ορίσματα από το Wit και εκτελεί την εντολή. Η πρόθεση ελέγχεται αν βρίσκεται στα κλειδιά της μεταβλητής _commands και εκτελεί την συνάρτηση στην οποία δείχνει το συγκεκριμένο κλειδί. Εαν η συνάρτηση επιστρέφει μία από τις τιμές ”config”, ”update” ή ”speech” τότε ο καθρέφτης

ανανεώνει τις ρυθμίσεις ή ανανεώνει τις διαθέσιμες εντολές (συνήθως όταν γίνεται αλλαγή οθόνης) ή ο καθρέφτης μιλάει στον χρήστη.

- `_config_change(conf: list)`: Βοηθητική συνάρτηση η οποία καλείται όταν κάποια εντολή του χρήστη θα αλλάξει τις ρυθμίσεις του καθρέφτη. Δέχεται μια ως είσοδο μια λίστα η οποία καθορίζει τις νέες ρυθμίσεις.

4.3.7 MirrorClock



Σχήμα 4.9: UML της κλάσης MirrorClock

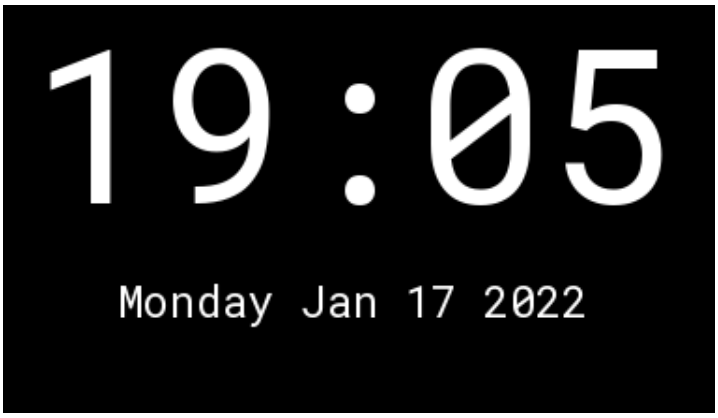
Περιγραφή Κλάσης

Η κλάση αυτή κληρονομεί από την κλάση Widget [17] που περιέχεται στο πακέτο `kivy.uix.widget` και υλοποιεί ένα ρολόι και ημερολόγιο στην αρχική οθόνη του καθρέφτη. Στο Σχήμα 4.10 φαίνεται γραφικά το συγκεκριμένο Widget.

Χαρακτηριστικά Κλάσης

- `_hour`: Μεταβλητή τύπου kivy.properties.StringProperty η οποία αποθηκεύει την ώρα σε μορφή ”HH:MM”.
- `_date`: Μεταβλητή τύπου kivy.properties.StringProperty η οποία αποθηκεύει την ημερομηνία σε μορφή ”Μέρα εβδομάδας, Μήνας, Μέρα του μήνα, Χρόνος”.

[17] https://kivy.org/doc/stable/api-kivy.uix.widget.html



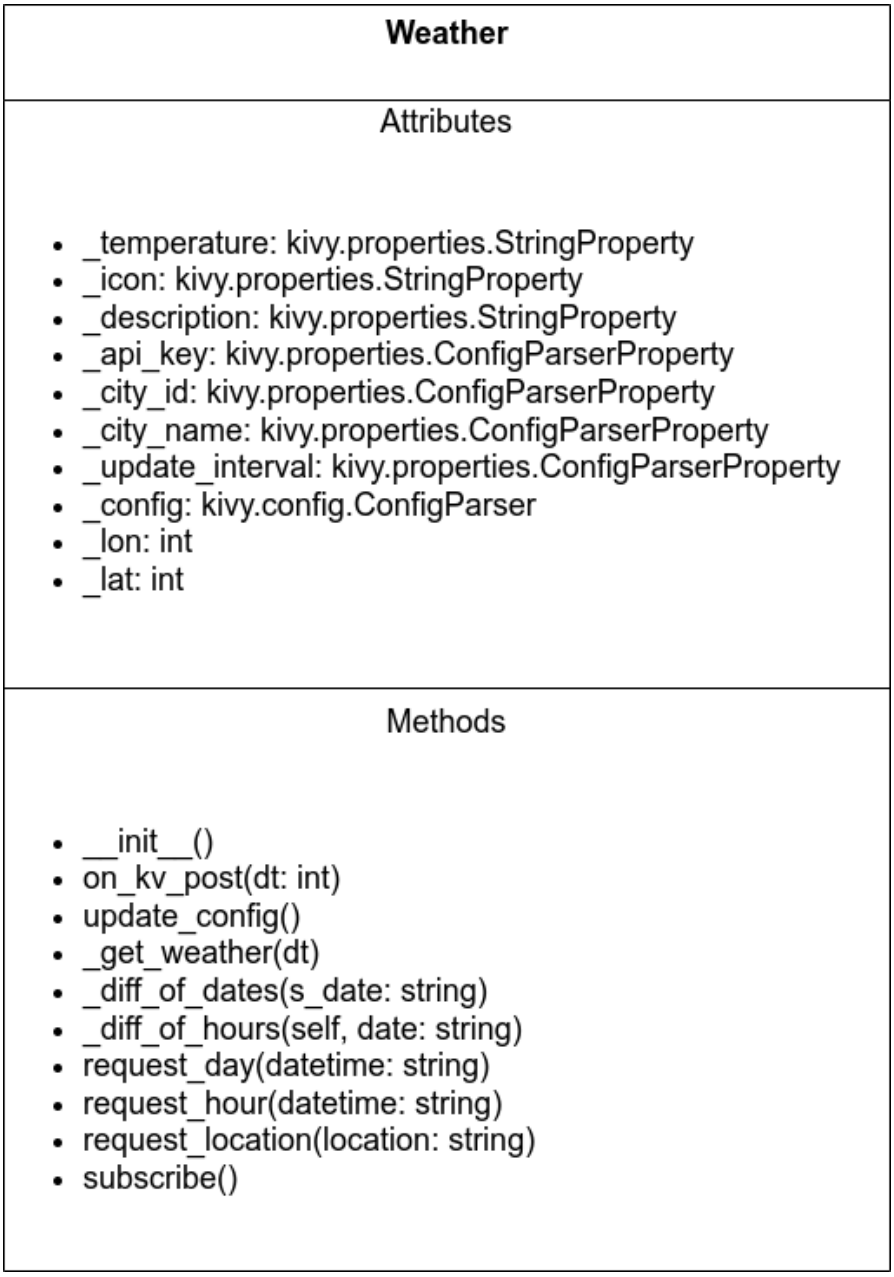
Σχήμα 4.10: Γραφική αναπαράσταση του Widget MirrorClock

Μέθοδοι Κλάσης

- `__init__()`: Συνάρτηση δόμησης της κλάσης η οποία καλή την συνάρτηση δόμησης της κλάσης `kivy.uix.widget.Widget` μέσω του `super` και καλεί την συνάρτηση `schedule_interval`[18] της κλάσης `kivy.clock.Clock` με χρόνο 1 sec η οποία καλεί με την σειρά της περιοδικά την συνάρτηση για την ανανέωση του χρόνου. Με αυτόν τον τρόπο η ώρα ανανεώνεται κάθε δευτερόλεπτο.
- `_update_time(dt: int)`: Συνάρτηση η οποία διαβάζει την ώρα μέσω της standard βιβλιοθήκης datetime[19] της Python και θέτει τις δύο μεταβλητές `_hour` και `_date` στις κατάλληλες τιμές.

[18] https://kivy.org/doc/stable/api-kivy.clock.html#kivy.clock.CyClockBase.schedule_interval
[19] https://docs.python.org/3/library/datetime.html

4.3.8 Weather



Σχήμα 4.11: UML της κλάσης Weather

Περιγραφή Κλάσης

Η κλάση αυτή κληρονομεί από την κλάση Widget [20] που περιέχεται στο πακέτο `kivy.uix.widget` και υλοποιεί ένα παράθυρο που δείχνει τον καιρό στην αρχική οθόνη του καθρέφτη αλλά και διάφορες συναρτήσεις για την πληροφόρηση σχετικά με τις καιρικές συνθήκες. Στο Σχήμα 4.12 φαίνεται γραφικά το συγκεκριμένο Widget.

[20] https://kivy.org/doc/stable/api-kivy.uix.widget.html

Σχήμα 4.12: Γραφική αναπαράσταση του Widget Weather

Χαρακτηριστικά Κλάσης

- `_temperature`: Μεταβλητή που αποθηκεύει την τιμή της θερμοκρασίας για να εμφανιστεί στο Widget.
- `_icon`: Μεταβλητή που αποθηκεύει το URL της εικόνας που αντιστοιχεί στον καιρό.
- `_description`: Μεταβλητή που αποθηκεύει μια πρόταση η οποία περιγράφει τον καιρό.
- `_api_key`: Μεταβλητή τύπου kivy.properties.ConfigParserProperty[21] η οποία διαβάζει από τις ρυθμίσει το κλειδί πρόσβασης για την αλληλεπίδραση με το OpenWeather[22] API.
- `_city_id`: Μεταβλητή τύπου kivy.properties.ConfigParserProperty η οποία διαβάζει το id της πόλης που επιθυμεί ο χρήστης να ενημερωθεί για τον καιρό από τις ρυθμίσεις.
- `_city_name`: Μεταβλητή τύπου kivy.properties.ConfigParserProperty η οποία διαβάζει το όνομα της πόλης που επιθυμεί ο χρήστης να ενημερωθεί για τον καιρό από τις ρυθμίσεις.
- `_update_interval`: Μεταβλητή τύπου kivy.properties.ConfigParserProperty η οποία διαβάζει από τις ρυθμίσεις τη χρονική περίοδο στην οποία επιθυμεί ο χρήστης να γίνεται ανανέωση του καιρού.
- `_update_interval`: Μεταβλητή τύπου kivy.config.ConfigParser η οποία χρησιμοποιείται για την ανάγνωση των ρυθμίσεων του καθρέφτη.
- `_lon`: Μεταβλητή τύπου int που αποθηκεύει το γεωγραφικό μήκος της πόλης για την οποία επιθυμεί ο χρήστης να ενημερωθεί για τον καιρό.
- `_lat`: Μεταβλητή τύπου int που αποθηκεύει το γεωγραφικό πλάτος της πόλης για την οποία επιθυμεί ο χρήστης να ενημερωθεί για τον καιρό.

[21] https://kivy.org/doc/stable/api-kivy.properties.html#kivy.properties.ConfigParserProperty
[22] https://openweathermap.org/

Μέθοδοι Κλάσης

- `__init__()`: Συνάρτηση δόμησης της κλάσης η οποία αρχικοποιεί το Widget.
- `on_kivy_post(dt: int)`: Ειδική συνάρτηση η οποία εκτελείται όταν το συγκεκριμένο Widget εμφανιστεί. Καλεί την συνάρτηση `update_config` και αρχικοποιεί ένα ρολόι το οποία καλεί την συνάρτηση `_get_weather` περιοδικά ανάλογα την τιμή της μεταβλητής `_update_interval`.
- `update_config()`: Συνάρτηση η οποία διαβάζει τις ρυθμίσεις και ανανενώνει κατάλληλα τις μεταβλητές μέλη της κλάσης και καλεί τις συνάρτησεις `_city_to_coord` και `_get_weather`.
- `_get_weather(dt: int)`: Συνάρτηση η οποία αλληλεπιδρά με το API της OpenWeather και ανανεώνει τα στοιχεία του Widget ώστε να δείχνουν την τελευταία ανάγνωση του καιρού.
- `_city_to_coord()`: Συνάρτηση η οποία διαβάζει ένα αρχείο με τις τοποθεσίες κάθε πόλης και επιστρέφει την τοποθεσία της πόλης που είναι αποθηκευμένη στην μεταβλητή `_city_name`.
- `_diff_of_dates(s_date: string)`: Συνάρτηση η οποία δέχεται μια ημερομηνία και υπολογίζει την διαφορά σε μέρες από την σημερινή.
- `_diff_of_hours(date: string)`: Συνάρτηση η οποία δέχεται μια ημερομηνία και υπολογίζει την διαφορά σε ώρες από την τωρινή ώρα.
- `request_date(datetime: string)`: Συνάρτηση η οποία δέχεται μια ημερομηνία ως όρισμα και λέει στον χρήστη τον καιρό για την εισαγώμενη μέρα.
- `request_hour(datetime: string)`: Συνάρτηση η οποία δέχεται μια ημερομηνία ως όρισμα και λέει στον χρήστη τον καιρό που θα κάνει την ζητούμενη ώρα.
- `request_location(location: string)`: Συνάρτηση η οποία δέχεται μια τοποθεσία, ενημερώνει τον χρήστη για τον καιρό στην συγκεκριμένη τοποθεσία και ενημερώνει τις ρυθμίσεις του καθρέφτει να δείχνει πλέον την εισαγώμενη τοποθεσία.
- `subscribe()`: Συνάρτηση η οποία εκθέτει στον Controller τις εντολές που δέχεται το Widget και ποιες συναρτήσεις θα εκτελεστούν.

4.4 Αναπτυξη Εφαρμογων

Η παρούσα παράγραφος περιγράφει την διαδικασία που πρέπει να ακολουθηθεί για να αναπτύξει κάποιος μια εφαρμογή η οποία θα προσαρτηθεί πάνω στον καθρέφτη. Η οργάνωση του συστήματος έχει γίνει με την χρήση των κλάσεων kivy.uix.screenmanager.ScreenManager και kivy.uix.screenmanager.Screen. Τα σημεία που πρέπει να προσέξει κάποιος για να προσαρτήσει την δική του εφαρμογή είναι στην δομή των φακέλων του κώδικα, στην ορθή ανάγνωση του Widget από

το λειτουργικό του καθρέφτη, οι σωστή δομή των ρυθμίσεων της προσαρτούμενης εφαρμογής και τέλος οι φωνητικές εντολές που θα είναι διαθέσιμες για τον χρήστη.

4.4.1 Δομή Φακέλων

Η δομή που έχει ο κώδικας του καθρέφτη φαίνεται στο παρακάτω δέντρο.

```
SmartMirror
├── smartmirror.py
├── smartmirror.kv
├── mirror_settings.py
├── modules/
│   ├── action.py
│   ├── basedir.py
│   ├── bot.py
│   ├── controller.py
│   └── speech.py
└── widgets/
    ├── clock/
    ├── exercisor/
    └── weather/
```

Οποιαδήποτε καινούργια εφαρμογή αναπτύσει ο χρήστης πρέπει να μπει κάτω από τον φάκελο widgets με την παρακάτω δομή.

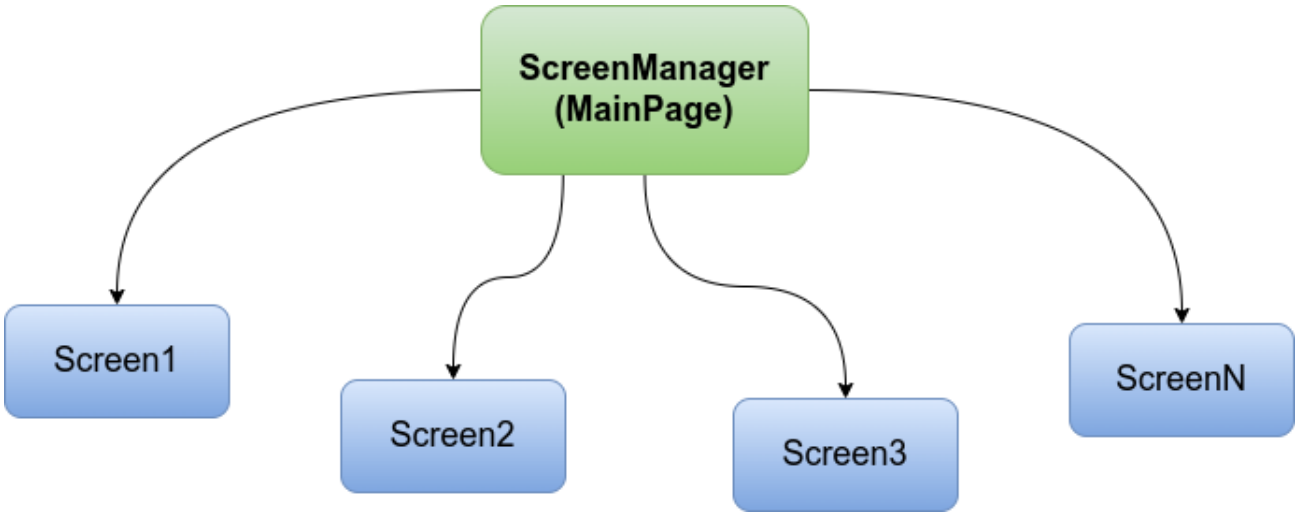
```
widgets/
└── <widget_name>/
    ├── <widget_name>.py
    ├── <widget_name>.kv
    └── settings.json
```

Με την παραπάνω οργάνωση το λειτουργικό είναι σε θέση να βρει αυτόματα και να προσαρτήσει την εφαρμογή.

4.4.2 Εγκατάσταση Οθόνης

Η γενική οργάνωση του καθρέφτη όσον αφορά την λογική των εξωτερικών εφαρμογών φαίνεται στο Σχήμα 4.13. Η κλάση MainPage είναι ένας ScreenManager ο οποίος διαβάζει όλες τις οθόνες που βρίσκονται στον φάκελο widgets/ και τις προσαρτίζει σε αυτόν.

Προκειμένου να γίνει η προσάρτηση, κάθε widget πρέπει να ορίσει μία συνάρτηση install(manager) η οποία καλεί την εντολή `manager.add_widget(<Screen>)`.



Σχήμα 4.13: Αναπαράσταση δομής του καθρέφτη

Κατά την εκκίνηση της εφαρμογής η κλάση MainPage καλεί τις συναρτήσεις install του κάθε widget και έτσι προσθέτει τις συγκεκριμένες οθόνες.

4.4.3 Ρυθμίσεις

Βιβλιογραφία

[1] Boehm. "*Software Engineering*". IEEE Transactions on Computers, C-25(12): 1226–1241, 1976.

[2] William Stallings. "*Operating Systems: Internals and Design Principles*". Pearson, 9 edition, 2018.

[3] Behrouz Forouzan and Firouz Mosharraf. "*Foundations of computer science*". Cengage Learning EMEA, 2 edition, 2008.

[4] Noam Nisan and Shimon Schocken. "*The Elements of Computing Systems: Building a Modern Computer from First Principles*". The MIT Press, 2 edition, 2021.

[5] Mark Segal and Kurt Akeley. "*The OpenGL Graphics System: A Specification (Version 4.6 (Core Profile) - October 22, 2019)*", 2021. URL https://www.khronos.org/registry/OpenGL/specs/gl/glspec46.core.pdf.

[6] Abiy Biru Chebudie, Roberto Minerva, and Domenico Rotondi. "*Towards a definition of the Internet of Things (IoT)*". PhD thesis, 08 2014.

[7] Pallavi Sethi and Smruti R. Sarangi. "*Internet of Things: Architectures, Protocols, and Applications*". JECE, 2017, January 2017. ISSN 2090-0147.

[8] Hans van Vliet. "*Software Engineering: Principles and Practice*". Wiley, 2007.

[9] Roger S. Pressman and Bruce R. Maxim. "*Software Engineering: A practitioner's approach*". McGraw-Hill Education, 8 edition, 2014. ISBN 978-0-07-802212-8.

[10] Linda G. Shapiro and George C. Stockman. "*Computer Vision*". Prentice Hall, 2001. ISBN 0-13-030796-3.

[11] Rıza Alp Güler, Natalia Neverova, and Iasonas Kokkinos. "*DensePose: Dense Human Pose Estimation in the Wild*". In "*Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*", June 2018.

[12] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. "*SMPL: A Skinned Multi-Person Linear Model*". ACM Transactions on Graphics, (Proc. SIGGRAPH Asia), 34(6):248:1–248:16, October 2015.

[13] Georgios Pavlakos, Xiaowei Zhou, Konstantinos G. Derpanis, and Kostas Daniilidis. "*Coarse-To-Fine Volumetric Prediction for Single-Image 3D Human Pose*". In "*Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*", July 2017.

[14] Xingyi Zhou, Xiao Sun, Wei Zhang, Shuang Liang, and Yichen Wei. "*Deep Kinematic Pose Regression*", 2016.

[15] Xiao Sun, Jiaxiang Shang, Shuang Liang, and Yichen Wei. "*Compositional Human Pose Regression*". In "*Proceedings of the IEEE International Conference on Computer Vision (ICCV)*", October 2017.

[16] Diogo C. Luvizon, Hedi Tabia, and David Picard. "*Human pose regression by combining indirect part detection and contextual information*". Computers & Graphics, 85:15–22, 2019. ISSN 0097-8493.

[17] Aiden Nibali, Zhen He, Stuart Morgan, and Luke Prendergast. "*Numerical Coordinate Regression with Convolutional Neural Networks*". arXiv preprint arXiv:1801.07372, 2018.

[18] Julieta Martinez, Rayat Hossain, Javier Romero, and James J. Little. "*A Simple yet Effective Baseline for 3D Human Pose Estimation*". In "*Proceedings of the IEEE International Conference on Computer Vision (ICCV)*", October 2017.

[19] Denis Tome, Chris Russell, and Lourdes Agapito. "*Lifting From the Deep: Convolutional 3D Pose Estimation From a Single Image*". In "*Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*", July 2017.

[20] Georgios Pavlakos, Xiaowei Zhou, and Kostas Daniilidis. "*Ordinal Depth Supervision for 3D Human Pose Estimation*". In "*Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*", June 2018.

[21] "*Kivy Documentation*". URL https://kivy.org/doc/stable/.

[22] Abdul Majeed and Ibtisam Rauf. "*MVC Architecture: A Detailed Insight to theModern Web Applications Development*". 2018.

[23] P. Bourque and R.E. Fairley. "*Guide to the Software Engineering Body of Knowledge*". 2014.

[24] Alireza Souri, Mohammad Shariffloo, and Monire Norouzi. "*Formalizing class diagram in UML*". 07 2011.

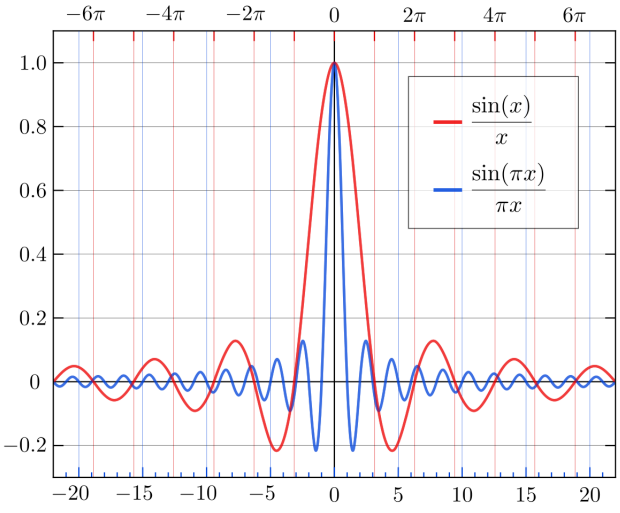
Παράρτημα

Μαθηματικοι Ορισμοι

Ορισμός 1 (Συνάρτηση argmax). *Δεδομένων ενός τυχαίου συνόλου X, ενός συνόλου Y και μιας συνάρτησης $f : X \to Y$, το μέγιστο όρισμα, argmax, πάνω σε ένα υποσύνολο $S \subseteq X$ ορίζεται ως*

$$argmax_S\, f := \underset{x \in S}{argmax} f(x) := \{x \in S \mid f(s) \leq f(x)\, \forall s \in S\}$$

Με άλλα λόγια, η συνάρτηση argmax είναι το σύνολο των σημείων x για τα οποία η $f(x)$ παίρνει την μέγιστη τιμή της.



Σχήμα 4.14: Για παράδειγμα, οι συναρτήσεις $sinc$ της γραφικής παράστασης έχουν και οι δύο $argmax = 0$ επειδή έχουν μέγιστο το 1 στο $x = 0$.

Ορισμός 2 (Συνάρτηση soft-argmax). *Η συνάρτηση soft-argmax $f : \mathbb{R}^K \to [0, 1)^K$ ορίζεται όταν $\|K\| > 1$ ως*

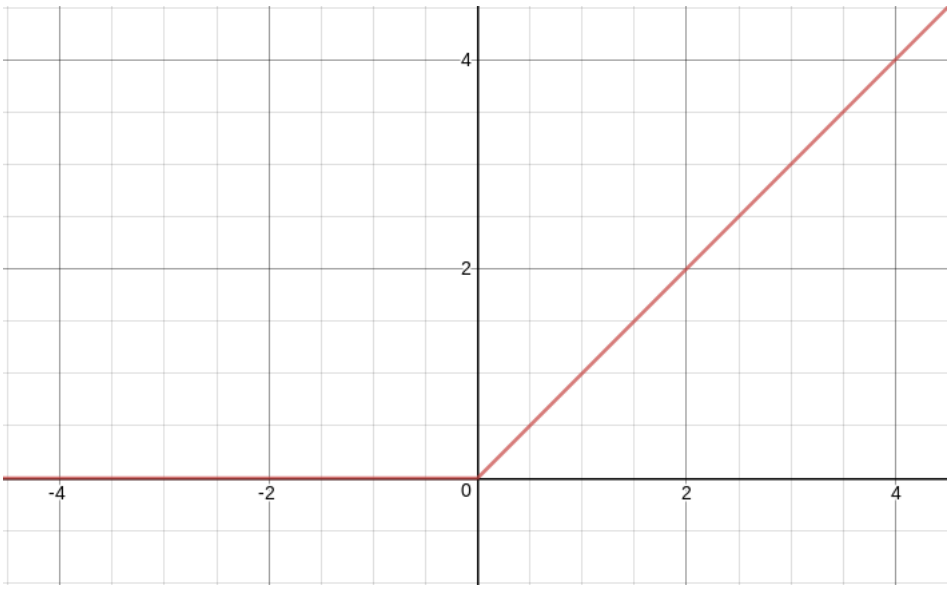
$$soft\_argmax = \sigma(\boldsymbol{x})_i := \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \text{ για } i = 1, \cdots, K \text{ και } \boldsymbol{x} = (x_1, \cdots, x_K) \in \mathbb{R}^K$$

Ουσιαστικά, η συνάρτηση soft-argmax εφαρμόζει την κανονική εκθετική συνάρτηση σε κάθε στοιχείο x_i του διανύσματος $\boldsymbol{x}$ και κανονικοποιεί τις τιμές διαιρώντας με το άθροισμα όλων των εκθετικών. Η κανονικοποίηση αυτή εξασφαλίζει ότι το άθροισμα των στοιχείων του διανύσματος εξόδου $\sigma(\boldsymbol{x})$ είναι 1.

Ορισμός 3 (Συνάρτηση ReLU). *Η συνάρτηση ReLU* $f : \mathbb{R} \to \mathbb{R}^+$ *ορίζεται ως*

$$f(x) = max(0, x)$$

Η συνάρτηση ReLU έχει ως έξοδο το όρισμα της όταν $x > 0$, ενώ όταν $x \leq 0$ η έξοδος είναι 0.



Σχήμα 4.15: Η συνάρτηση ReLU