Εθνικό Μετσόβιο Πολυτεχνείο

ΔΠΜΣ Συστήματα Αυτοματισμού

Κατεύθυνση Β':

Συστήματα Αυτομάτου Ελέγχου και Ρομποτικής

<u>Μεταπτυχιακό Μάθημα:</u>

Τεχνολογίες και Εφαρμογές Προσθετικής Κατασκευής/3D Εκτύπωσης

Ομαδική Εργασία Εξαμήνου

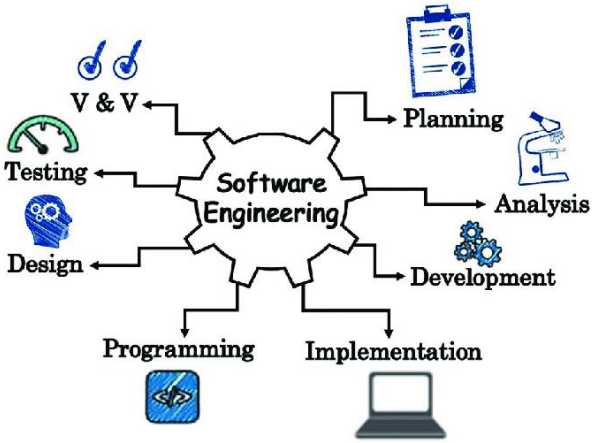
Ανάπτυξη Plugin για το Ultimaker CURA

<u>Ονόματα Φοιτητών – Α.Μ.:</u>

Γεώργιος Κασσαβετάκης - 02121203

Γεώργιος Κρομμύδας – 02121208

Αλέξανδρος Δεληγιαννίδης - 02121214



ΑΘΗΝΑ,

2023

Πίνακας περιεχομένων

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Συντομογραφίες

API - Application Programming Interface

UML – Unified Modeling Language

UC – Use Case

GUI – Graphical User Interface

FDM – Fused Deposition Modeling

FFF – Fused Filament Fabrication

CAD – Computer Aided Design

STL – Standard Triangle Language

3MF – 3D Manufacturing Format

MVC – Model View Controller

UI – User Interface

OOP – Object-Oriented Programming

TCP – Transfer Control Protocol

JSON – JavaScript Object Notation

US – User Story

CBO - Coupling Between Object classes

COF – Coupling Factor

LCOM – Lack of Cohesion of Methods

WMC – Weighted Methods per Class

RFC – Request For a Class

Περίληψη

Η εξέλιξη της τεχνολογίας και συγκεκριμένα του υλισμικού (hardware) και λογισμικού (software) έφερε μεγάλη επανάσταση στον βιομηχανικό κλάδο. Η εξέλιξη αυτή, επηρέασε και τον κλάδο της προσθετικής κατασκευής, ο οποίος ξεκίνησε να αναπτύσσεται από τα μέσα της δεκαετίας του 1980, καθώς έχουν εξελιχθεί οι εκτυπωτές τρισδιάστατης εκτύπωσης. Οι εκτυπωτές αυτοί, εκτός από τον μηχανολογικό τους κορμό, αποτελούνται επίσης και από ηλεκτρονικά συστήματα και εγκαθιδρυμένα λογισμικά, μαζί με ενδοεπικοινωνιακά πρωτόκολλα τα οποία συνεισφέρουν στην ταχεία επεξεργασία της πληροφορίας και την έναρξη εκτύπωσης των αντικειμένων. Ένας πάροχος εκτυπωτών τρισδιάστατης εκτύπωσης είναι η Ultimaker, η οποία προσφέρει το δικό της λογισμικό, το Ultimaker CURA. Στόχος της παρούσας εργασίας είναι να σχεδιαστεί, υλοποιηθεί και ενσωματωθεί ένα εξωτερικό λογισμικό το οποίο διαβάζει τα metadata των αρχείων .3mf και τα προβάλλει στην γραφική διεπαφή (GUI) του λογισμικού Ultimaker CURA.

Λέξεις Κλειδιά: Προσθετική Κατασκευή, Ultimaker CURA, Λογισμικό, Υλισμικό, Τρισδιάστατοι Εκτυπωτές, Μεταδεδομένα, Γραφική Διεπαφή, .3mf Αρχεία, Εξωτερικά Λογισμικό

Abstract

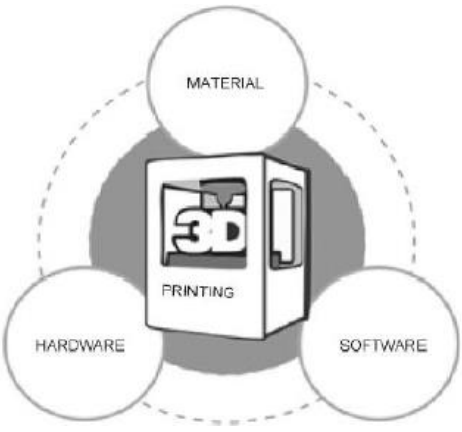
The development of technology, specifically hardware and software, brought a great revolution in the industrial sector. In fact, the field of additive manufacturing, which started developing in the mid-1980s, has also been affected, as 3D printers both hardware and preparation software have evolved. These printers, except from their mechanical properties, consist also of electronic systems and software, as well as intercommunication protocols that contribute to the rapid processing of information and the initiation of printing objects. One provider of 3D printing printers is Ultimaker, which provides its own software, called Ultimaker CURA. The aim of this work is to design, implement and integrate an external software which reads the metadata of .3mf files and displays them into the graphical user interface of the Ultimaker CURA software.

Keywords: Additive Manufacturing, Ultimaker CURA, Software, Hardware, 3D Printers, Metadata, Graphical User Interface, .3mf Files, External Software, Plugin

Κεφάλαιο 1. Εισαγωγή

1.1 Στόχοι Εργασίας

Ο κλάδος της προσθετικής κατασκευής (Additive Manufacturing) ξεκίνησε να αναπτύσσεται στα μέσα της δεκαετίες του 1980 από την εταιρία 3D systems με τον πρώτο εκτυπωτή τεχνολογίας πολυμερισμού με κάδο. Με την πάροδο του χρόνου και την εξέλιξη της τεχνολογίας, καινούργιες μορφές εκτυπωτών αναπτυχθήκαν. Η ανάπτυξη του υλισμικού και λογισμικού επηρέασε σημαντικά την αρχιτεκτονική των σύγχρονων εκτυπωτών. Εκτός από το μηχανολογικό υποσύστημα, πλέον περιέχουν και ηλεκτρονικό υποσύστημα το οποίο διαχειρίζεται την πληροφορία. Το ηλεκτρονικό υποσύστημα αποτελείται από ηλεκτρονικές μονάδες, όπως είναι ο επεξεργαστής και η κάρτα δικτύου. Επίσης, υπάρχουν εγκαθιδρυμένα λογισμικά τοποθετημένα στα ηλεκτρονικά εξαρτήματα (βλ. Σχήμα 1), τα οποία καθιστούν γρήγορη και αποτελεσματική την επικοινωνία μεταξύ τους.



Σχήμα 1. Σύγχρονα μέρη ενός τρισδιάστατου εκτυπωτή

Ένας οίκος που κατασκευάζει τέτοιους εκτυπωτές ονομάζεται Ultimaker. Ο οίκος αυτός κατασκευάζει εκτυπωτές τεχνολογίας

FDM/FFF. Οι συγκεκριμένοι εκτυπωτές διαχειρίζονται κυρίως θερμοπλαστικά υλικά ως πρώτη ύλη. Κάποιοι από τους εκτυπωτές του οίκου αυτού διαθέτουν δεύτερη κεφαλή και επιτρέπουν την χρήση δευτερεύον υλικό δεύτερο υλικό ή υποστήριξη στην κύρια κατασκευή. Ο οίκος αυτός έχει αναπτύξει δικό του λογισμικό προετοιμασίας, το οποίο ονομάζεται Ultimaker CURA. Με βάση αυτό το λογισμικό, ένας χρήστης μπορεί να εισάγει ένα αρχείο CAD, το οποίο έχει σχεδιάσει και να ξεκινήσει τα στάδια προετοιμασίας του αντικειμένου προς εκτύπωση. Κατά το πέρας της εργασίας, το αντικείμενο είναι έτοιμο για εκτύπωση και αποθηκεύεται σε μία μορφή αρχείου. Αυτή η μορφή αποτελείται από κώδικα τον οποίο κατανοεί και εκτελεί ο 3D εκτυπωτής.

Το λογισμικό Ultimaker Cura μπορεί να αναγνωρίσει και να επεξεργαστεί διάφορων τύπων αρχείων που χρησιμοποιούνται από τον χώρο της προσθετικής κατασκευής. Μία μορφή αυτών των αρχείων αποτελούν τα .3mf αρχεία. Είναι μία σύγχρονη μορφή αρχείων που επρόκειτο να αντικαταστήσει την παλιά τεχνολογία αρχείων .stl. Τα αρχεία .3mf αποθηκεύουν μεγαλύτερη ποσότητα πληροφορίας συγκριτικά με το παραδοσιακό αρχείο .stl. Αναλυτικότερα μπορούμε να δούμε και τις διαφορές του στο Σχήμα 2.

	STL	3MF
Creation	1987 by 3D Systems	2015 by the 3MF Consortium
Readable by Humans	No	Yes
Stored Info	Mesh (Polygons)	Units, textures, materials, colors, mesh, nesting, machine, thumbnail...
File Type	Mesh (RAW data)	Archive (Mesh + more info)
Size	Huge (thousands of polygons)	Light (code to express parameters & copies)

Σχήμα 2. Διαφορές μεταξύ .stl και .3mf αρχείων

Διαπιστώνουμε αυτομάτως πως τα .3mf αρχεία είναι καλύτερα για την και περιέχουν περισσότερη και αναγκαία πληροφορία. Ωστόσο εκτός από την κύρια πληροφορία, υπάρχει και η μεταπληροφορία

(metadata) σε ένα αρχείο .3mf, η οποία περιέχει κάποιες πληροφορίες για το σχέδιο, για τα δικαιώματα και το όνομα του κατόχου του αρχείου. Γενικά, τα .3mf αρχεία χρησιμοποιούν την xml μορφή αρχείων αλλά σε μία έκδοση.

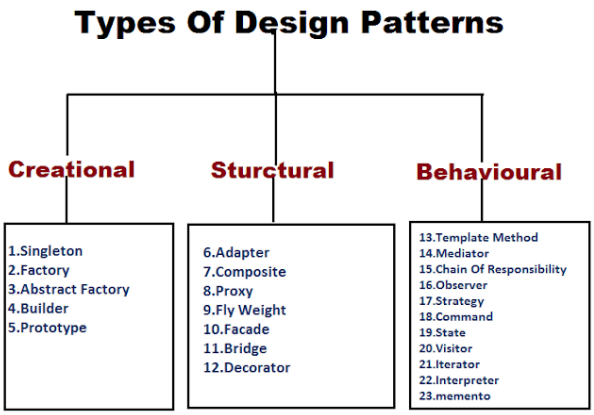
Η παρούσα εργασία ως στόχο έχει να παρουσιάσει ένα εξωτερικό λογισμικό (plugin) το οποίο θα διαβάζει και θα εμφανίζει τα metadata των αρχείων .3mf στο γραφικό περιβάλλον του Ultimaker CURA. Όλη η διαδικασία ανάγνωσης γίνεται με την βοήθεια του framework URANIUM που αποτελεί την κύρια διεπαφή (interface) του λογισμικού CURA. Αποτελεί τον σκελετό του προγράμματος όπου γίνονται όλες οι επικοινωνίες των αντικειμένων μεταξύ τους. Επίσης, υπάρχει αλληλεπίδραση με τον χρήστη που χρησιμοποιεί το συγκεκριμένο λογισμικό.

1.2 Ultimaker CURA και Εξωτερικά Λογισμικά

Το λογισμικό CURA, εκτός από τις κύριες λειτουργίες του, υποστηρίζει και εξωτερικές λειτουργίες, οι οποίες μπορούν να ενσωματωθούν ως εξωτερικά λογισμικά (external software/plugins). Το λογισμικό το ίδιο έχει την δυνατότητα αυτή, καθώς η διεπαφή του είναι σχεδιασμένη και προγραμματισμένη με τέτοια δομή, ώστε να καθιστά εύκολη η ενσωμάτωση αυτών των λογισμικών. Πληθώρα τέτοιων λογισμικών έχουν αναπτυχθεί και μπορούν να εγκατασταθούν στην κύρια εφαρμογή μέσω του marketplace που παρέχει η εταιρία.

Τα λογισμικά αυτά έχουν ως στόχο την επέκταση των λειτουργιών του κύριου προγράμματος. Καθιστώντας έτσι πιο ευέλικτο το πρόγραμμα, με τον χρήστη να παραμετροποιεί την διαδικασία της εκτύπωσης του αντικειμένου του ευκολότερα. Η ενσωμάτωση αυτών των λογισμικών γίνεται μέσω του interface του βασικού προγράμματος. Η σχεδίαση του λογισμικού γίνεται με την χρήση

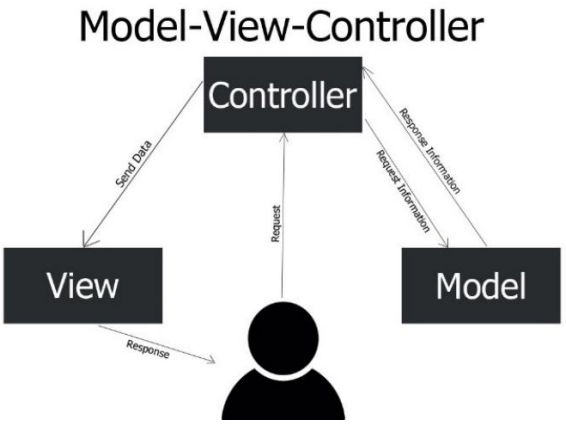
προτύπων (patterns). Τα patterns είναι κομμάτια κώδικα τα οποία προσθέτουν επεκτασιμότητα σε ένα λογισμικό. Δηλαδή, μπορούν να προστεθούν λειτουργίες, χωρίς να επηρεαστεί η πολυπλοκότητα του λογισμικού. Τα patterns μπορούν να χωριστούν σε κατηγορίες ανάλογα με την λειτουργία τους (βλ. Σχήμα 3).



Σχήμα 3. Τύποι προτύπων

Με όμοιο τρόπο λειτουργούν και τα plug-ins σε ένα λογισμικό. Μέσω κάποιων pattern μπορούν να ενσωματωθούν στην κύρια λειτουργία του προγράμματος, προσφέροντας έτσι παραπάνω λειτουργίες στο λογισμικό αυτό. Με αυτόν τον τρόπο το λογισμικό μπορεί να συντηρείται εύκολα με νέες επεκτάσεις. Η πολυπλοκότητα του συστήματος δεν αυξάνεται και έτσι η το σύστημα δεν επιβαρύνεται από νέες λειτουργίες.

Εκτός από τις βασικές λειτουργίες του λογισμικού, το λογισμικό CURA βασίζεται και σε κάποια αρχιτεκτονική. Η αρχιτεκτονική αυτή ονομάζεται MVC (Model – View - Controller). Η κύρια λειτουργίας της μπορεί να περιγραφτεί και από το σχήμα 4. Ένας χρήστης κάνοντας κάποια επιλογή στην γραφική διεπαφή του λογισμικού, δίνει εντολή (command ή control) στο μοντέλο (model) του συστήματος που αποτελεί το back – end κομμάτι του λογισμικού. Το model θα στείλει τα κατάλληλα δεδομένα στην γραφική διεπαφή μέσω μιας εντολής του controller.



Σχήμα 4. Αρχιτεκτονική MVC

Με αυτή την διαδικασία, ο χρήστης μπορεί να επεξεργαστεί ευκολότερα το δικό του αντικείμενο στο λογισμικό, χωρίς να εμπλέκονται καθυστερήσεις και πολυπλοκότητα κατά την συλλογή των δεδομένων από τον controller.

1.3 Δομή Εργασίας

Η παρούσα εργασία αποτελείται από τέσσερα κεφάλαια συνολικά. Το πρώτο κεφάλαιο αποτελείται από την εισαγωγή στην εργασία, θέτοντας τους στόχους για την δημιουργία εξωτερικού λογισμικού και τον τρόπο με τον οποίο ενσωματώνεται αυτό στο κυρίως μέρος του Ultimaker CURA. Επίσης, παρουσιάζεται και η δομή της εργασίας.

Το δεύτερο κεφάλαιο αποτελείται από την περιγραφή του framework URANIUM, μαζί με τους τρόπους λειτουργίας και ενσωμάτωσης των plugin.

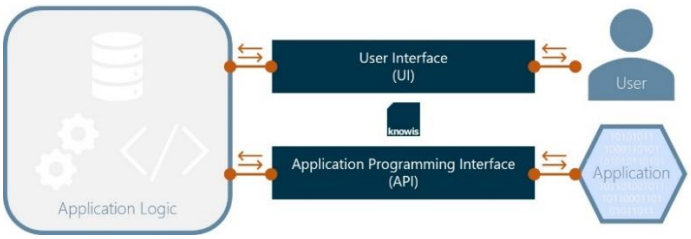
Το τρίτο κεφάλαιο αποτελείται από την σχεδίαση του λογισμικού, σύμφωνα με την ανάλυση απαιτήσεων του λογισμικού και την αρχιτεκτονική του, μέσω των UML διαγραμμάτων.

Τέλος, το τέταρτο κεφάλαιο αποτελείται από την υλοποίηση του λογισμικού και την ενσωμάτωση του στο λογισμικό CURA, με την παραγωγή και εμφάνιση αποτελεσμάτων.

Κεφάλαιο 2. URANIUM Framework

2.1 Application Programming Interface

Η Διεπαφή Προγραμματισμού Εφαρμογών (Application Programming Interface) αποτελεί στην ουσία μία διεπαφή που εγκαθιδρύεται η επικοινωνία μεταξύ προγραμμάτων στο λογισμικό. Η χρήση των APIs αποτελεί θεμελιώδη λίθο ως προς την δημιουργία των λογισμικών, καθώς διασυνδέει όλες τις οντότητες του συστήματος (κλάσεις (classes)) σε επίπεδα αφαίρεσης (abstracting) με στόχο να μπορεί ένας προγραμματιστής να διαχειρίζεται τα αντικείμενα (objects) του λογισμικού. Τα αντικείμενα αυτά περιέχουν την πληροφορία που χρειάζονται που επεξεργάζονται οι μέθοδοι (methods) των κλάσεων.



Σχήμα 5. Σχεδιάγραμμα ενός API

Μία μορφή ενός API φαίνεται και στο σχήμα 5. Παρατηρούμε πως η επικοινωνία γίνεται μεταξύ εσωτερικών εφαρμογών. Ο χρήστης δεν έχει άμεση επαφή με τα δεδομένα που ανταλλάσσονται μεταξύ των εφαρμογών, διότι η διαδικασία αυτή βρίσκεται σε χαμηλότερο αφαιρετικό επίπεδο. Για να μπορέσει ο χρήστης να έχει επαφή, δημιουργείται μία διαφορετική διεπαφή (User Interface) η οποία του παρέχει την ικανότητα να αλληλεπιδρά με την εφαρμογή. Ένα χαρακτηριστικό παράδειγμα είναι η γραφική διεπαφή (GUI), που αποτελεί και το σκέλος της αρχιτεκτονικής View.

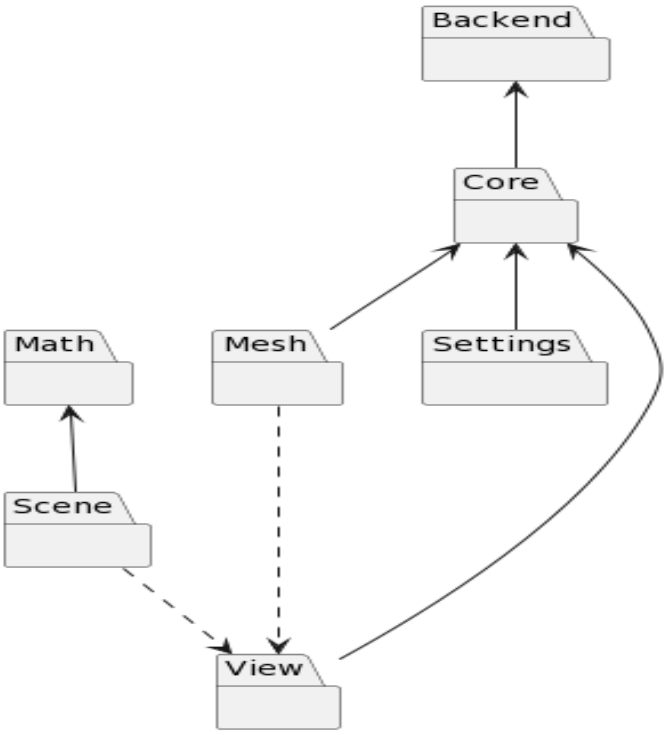
2.2 Framework

Το framework που χρησιμοποιεί το Ultimaker CURA ονομάζεται URANIUM. Αποτελεί στην ουσία το API του συστήματος, με το οποίο επικοινωνούν όλα τα υποσυστήματα του λογισμικού. Έχει σχεδιαστεί και υλοποιηθεί στην γλώσσα προγραμματισμού Python, με χρήση αντικειμενοστρεφή μεθόδων (object oriented methods). Η χρήση των αντικειμένων καθιστά ευκολότερο τον σχεδιασμό και την επικοινωνία των εσωτερικών προγραμμάτων του συστήματος κάνοντας ευκολότερη την επεκτασιμότητα και την συγκεντρωτική παρουσίαση του παραγόμενου λογισμικού.

Αναλυτικότερα, το URANIUM αποτελείται από κομμάτια τα οποία αποτελούν την αρχιτεκτονική του. Τα κομμάτια της αρχιτεκτονικής ονομάζονται διαφορετικά και πακέτα (packages). Τα πακέτα αυτά της αρχιτεκτονικής είναι τα εξής:

- Core Package
- Backend Package
- Math Package
- Mesh Package
- Scene Package
- Settings Package
- View Package

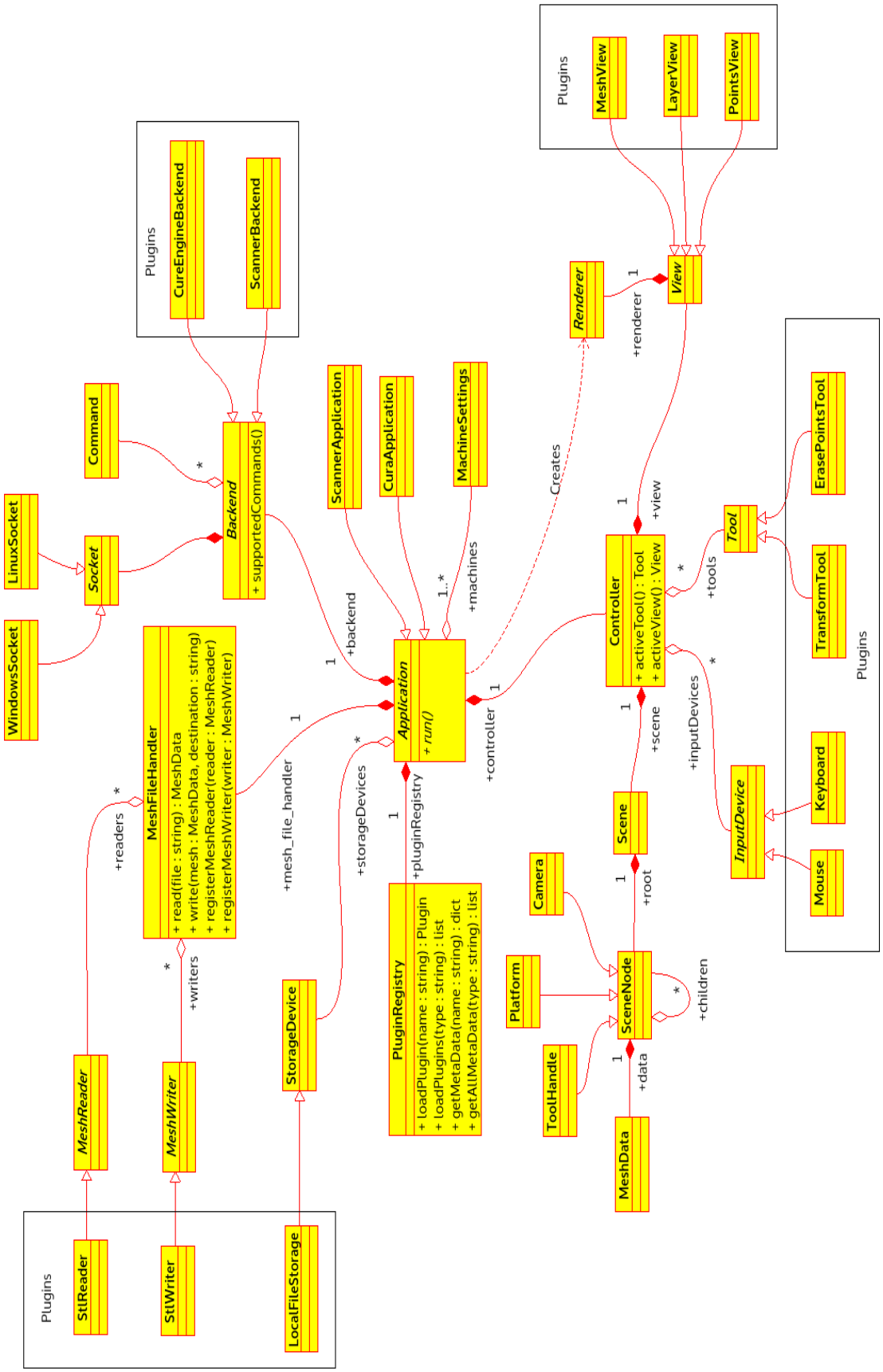
Επίσης, υπάρχουν και δύο ειδικά πακέτα (toolkit specifics) τα οποία υλοποιούν κάποια μέρη του UI της εφαρμογής. Η βασική αρχιτεκτονική του URANIUM παρουσιάζεται και αναλυτικότερα στο Σχήμα 6.



Σχήμα 6. UML Διάγραμμα πακέτων του URANIUM

Παρατηρούμε την βασική αρχιτεκτονική του framework η οποία είναι η MVC. Ως Model έχουμε το πακέτο Backend, το οποίο δέχεται εντολές από τον Controller (Core) και επιστρέφει τα δεδομένα. Ο Controller είναι το πακέτο Core που αποτελεί τον εγκέφαλο του προγράμματος, δίνοντας τις κατάλληλες εντολές στα υπόλοιπα πακέτα. Ως View έχουμε το πακέτο View το οποίο επικοινωνεί με το πακέτο Core, ώστε να εμφανιστούν τα κατάλληλα δεδομένα στην γραφική διεπαφή. Επίσης, το πακέτο Scene και mesh στέλνουν τα δεδομένα στην γραφική διεπαφή. Το πακέτο Math χρησιμοποιείται κυρίως μόνο από το πακέτο Scene, ώστε να γίνονται οι κατάλληλοι υπολογισμοί και μετασχηματισμοί των αντικειμένων. Το πακέτο Mesh χρησιμοποιεί την πληροφορία του των αντικειμένων και τα προβάλει στο πακέτο View. Τέλος, το πακέτο Settings αφορά τις ρυθμίσεις του μηχανήματος (εκτυπωτή) που στέλνονται στο πακέτο Core.

Το συνολικό διάγραμμα κλάσεων του συστήματος φαίνεται στο σχήμα 7. Επίσης, βλέπουμε και τον τρόπο με τον οποίο γίνεται η διασύνδεση των plugins στο σύστημα.



Σχήμα 7. UML Διάγραμμα Κλάσεων του URANIUM

2.2.1 Core Package

Το πακέτο ***Core*** αποτελείται από τις βασικές κλάσεις του συστήματος και τα κεντρικά αντικείμενα του συστήματος, τα οποία επικοινωνούν με όλες τις εσωτερικές εφαρμογές. Οι κύριες κλάσεις αυτού του πακέτου είναι η ***PluginRegistery*** και η ***Application***. Η κλάση ***Application*** αποτελεί το πρωτεύων αντικείμενο για κάθε άλλη κλάση που την καλεί. Ως ρόλο έχει να παρέχει πρόσβαση για τα υπόλοιπα αντικείμενα των κλάσεων, όπως της κλάσης ***Controller*** και της ***MeshFileHandler***.

Η κλάση ***PluginRegistery*** είναι υπεύθυνη για την φόρτωση των plugins στο σύστημα μαζί με τα μεταδεδομένα τους. Ακόμα μία σημαντική κλάση αποτελεί η ***Signal***. Προσφέρει έναν μηχανισμό χειρισμού συμβάντων με την χρήση ειδικών μεθόδων ανάκλησης. Για παράδειγμα, η κλάση ***Controller*** παρέχει ένα σήμα ***activeViewChanged*** που εκπέμπεται κάθε φορά που αλλάζει η σκηνή στην γραφική διεπαφή. Άλλες κατηγορίες μπορούν να συνδεθούν σε αυτό το σήμα για να λαμβάνουν ειδοποιήσεις σχετικά με αυτό το συμβάν.

Άλλες κλάσεις που αποτελούν μέρος του ***Core*** είναι η ***Controller***, η οποία είναι ένα κεντρικό αντικείμενο που παρέχει επικοινωνία (εντολές) μεταξύ των κλάσεων ***Scene*** και ***View***, επιπλέον του χειρισμού συμβάντων μεταξύ ***Scene***, ***View***, ***InputDevice*** και ***Tools***. Η κλάση ***Event*** αντιπροσωπεύει τη βασική κλάση για οποιοδήποτε συμβάν στο σύστημα, με αρκετές υποκλάσεις που παρέχονται για κοινά συμβάντα. Το ***InputDevice*** είναι μια αφηρημένη (abstract) κλάση για συσκευές που παρέχουν αντικείμενα εισόδου στην κλάση ***Tool*** ή/και στην κλάση ***View***. Επίσης, θα πρέπει να κατηγοριοποιείται ανάλογα με την είσοδο. Δηλαδή, ένα προέρχεται από το κύριο πρόγραμμα ή από κάποια plugins. Η κλάση ***Logger*** είναι μια κλάση του πακέτου Core η οποία προσφέρει σε άλλες κλάσεις

σύνδεση σε κάποια έξοδο του συστήματος. Αυτά μπορούν να υποκατηγοριοποιηθούν από plugins για να παρέχουν εναλλακτικές δυνατότητες καταγραφής. Η κλάση ***StorageDevice*** παρέχει την δυνατότητα να ανοίξει/κλείσει το σύστημα αρχεία. Τέλος, η κλάση ***Tool*** είναι η βασική κλάση για κάθε αντικείμενο που λειτουργεί και εμφανίζεται στη σκηνή.

2.2.2 Backend Package

Το πακέτο ***Backend*** εμπεριέχει τις κλάσεις που επικοινωνούν με εξωτερικές εφαρμογές από το σύστημα, ώστε να επεξεργαστεί τα δεδομένα που λαμβάνει. Ένα παράδειγμα επικοινωνίας του backend είναι με την κλάση ***CuraEngine*** για την εφαρμογή του slicing στο αντικείμενο προς εκτύπωση.

Η κλάση ***Backend*** είναι η κύρια κλάση του πακέτου η οποία διευκολύνει την επικοινωνία. Παρέχει μία αφηρημένη βάση που κληρονομεί τα πεδία της σε υποκλάσεις που προέρχονται από κάποια plugins. Επίσης, η κλάση αυτή παρέχει μία λίστα από εντολές, με τις οποίες μεταδίδονται τα δεδομένα στις υπόλοιπες κλάσεις του συστήματος.

Η κλάση ***Command*** ενθυλακώνει αυτές τις εντολές και παρέχει δύο μεθόδους επικοινωνίας την ***send()*** και ***receive()***, από τις οποίες επικοινωνούν οι υπόλοιπες κλάσεις μεταξύ τους. Η επικοινωνία των εντολών μαζί με την κλάση ***Backend*** γίνεται μέσω ενός τοπικού *TCP* πρωτόκολλου, όπου υλοποιείται από την κλάση ***Socket***.

Τέλος, η κλάση ***Socket*** σχηματίζει έναν αγωγό (pipe/socket) που χρησιμοποιείται για την επικοινωνία με την κλάση ***Backend***. Τα command objects είναι υπεύθυνα για την σειριοποίηση και αποσειριοποίηση των δεδομένων για να σταλθούν σε αυτό το socket.

2.2.3 Math Package

Το πακέτο ***Math*** είναι υπεύθυνο με την παροχή μεθόδων της γραμμικής άλγεβρας στις υπόλοιπες κλάσεις του συστήματος.

Η κλάσεις ***Matrix***, ***Vector*** και ***Quaternion*** βασίζονται στην βιβλιοθήκη *NumPy*, όπου κατασκευάζουν πίνακες διαστάσεων 4x4, ώστε να χρησιμοποιούνται οι αφφινικοί μετασχηματισμοί. Χρησιμοποιούνται κυρίως από την κλάση ***SceneNode*** που είναι υπεύθυνη για την κατασκευή του γραφήματος των αντικειμένων της σκηνής. Δηλαδή, τον τρόπο διαχειρισμού των αντικειμένων της σκηνής μέσω αυτών των μετασχηματισμών.

2.2.4 Mesh Package

Το πακέτο ***Mesh*** περιέχει τις κλάσεις που ασχολούνται με τα ***mesh*** δεδομένα. Δηλαδή τα δεδομένα που αφορούν την γεωμετρία, την υφή και τα υπόλοιπα χαρακτηριστικά των τρισδιάστατων αντικειμένων της σκηνής.

Πρωταρχικό ρόλο του πακέτου έχει η κλάση ***MeshFileHandler***, η οποία καλείται από την κεντρική κλάση ***Application***, ώστε να ξεκινήσει η διαδικασία ανάγνωσης και εγγραφής αρχείων ***mesh***. Η συγκεκριμένη κλάση χρησιμοποιεί δύο λίστες αντικειμένων τύπου ***MeshReader*** και ***MeshWriter***. Αυτές οι κλάσεις πρακτικά διαβάζουν και εγγράφουν αντίστοιχα τα ***mesh*** αρχεία. Για να λειτουργήσουν αυτές οι κλάσεις, θα πρέπει να ενσωματωθούν plugins τα οποία διαβάζουν και εγγράφουν συγκεκριμένο τύπο αρχείων. Οι συγκεκριμένες κλάσεις προωθούν τα αντικείμενα τους στις παραπάνω κλάσεις, ώστε η σκηνή να εμφανίσει τα τρισδιάστατα μοντέλα.

Όταν διαβάζεται ένα πλέγμα (mesh), τότε η κλάση ***MeshFileHandler*** επιστρέφει ένα αντικείμενο τύπου ***MeshData***. Το οποίο αποτελείται από τα δεδομένα του ίδιου του πλέγματος. Δηλαδή κορυφές, ακμές και πλευρές. Η κλάση ***MeshData*** δημιουργεί μια λίστα από αντικείμενα τύπου ***Vertex***, που είναι στην ουσία οι ακμές του 3D αντικειμένου. Επίσης, παρέχει μία λίστα με τις ακμές του πλέγματος, που σχηματίζουν τις επιφάνειες τους σχήματος. Επίσης, περιέχει και κάποιες μεθόδους επεξεργασίας των αντικειμένων, όπως για παράδειγμα να μας επιστραφεί το κυρτό περίβλημα των κορυφών για να ελέγξουμε την κυρτότητα του σχήματος ή να εφαρμόσουμε κάποιους μετασχηματισμούς στις κορυφές.

2.2.5 Scene Package

Το πακέτο ***Scene*** περιέχει τις κλάσεις οι οποίες είναι υπεύθυνες για τον χειρισμό όλων των δεδομένων της σκηνής. Η σκήνη αντιπροσωπεύεται από την κλάση ***Scene***. Η διάταξη των αντικειμένων της σκηνής δημιουργείται από μία δενδρική δομή. Η δομή αυτή δημιουργείται από την κλάση ***SceneNode***. Στην ρίζα (root) του δένδρου είναι η το αντικείμενο της σκηνής. Στα υπόλοιπα φύλλα του δένδρου τοποθετούνται τα υπόλοιπα αντικείμενα της σκηνής, έτσι ώστε να σχηματιστεί ένας γράφος. Επιπρόσθετα, στην ρίζα του δένδρου, εκτός από την κλάση ***Scene***, τοποθετείται και το αντικείμενο της κλάσης ***Camera***.

Η κλάση ***Camera***, δημιουργεί το περιβάλλον που βλέπει ο χρήστης στην γραφική επαφή μέσω των προβολικών μετασχηματισμών και με την βοήθεια του πακέτου ***Math***.

2.2.6 Settings Package

Το πακέτο ***Settings*** περιέχει τις κλάσεις που ρυθμίζουν τις παραμέτρους του εκτυπωτή και του αντικειμένου ως προς εκτύπωση. Η κυρίαρχη κλάση της μεθόδου είναι η ***MachineSettings***, η οποία δημιουργεί αντικείμενα τα οποία διαβάζουν ή/και γράφουν τις ρυθμίσεις τις οποίες έδωσε ο χρήστης για την προετοιμασία της εκτύπωσης. Επίσης, η συγκεκριμένη κλάση παρέχει και μία λίστα αντικειμένων τύπου ***SettingsCategory***, η οποία κατηγοριοποιεί την κάθε ρύθμιση. Έτσι, κάθε κατηγορία ρυθμίσεων θα δημιουργεί μία δενδρική δομή με από αντικείμενα τύπου ***Settings***. Τα φύλλα της δενδρικής δομής περιέχουν τις ρυθμίσεις που βλέπει ο χρήστης στην οθόνη. Για παράδειγμα εάν ως Settings αντικείμενο έχουμε την ταχύτητα "Speed", τότε στα φύλλα της δομής θα υπάρχουν όλες οι πιθανές ταχύτητες που θέλουμε να ρυθμίσουμε, όπως για παράδειγμα το "Infill Speed". Τέλος, η κλάση ***MachineSettings*** μπορεί να φορτώσει το δένδρο από ένα JSON αρχείο όπου έχουν αποθηκευτεί οι τιμές που δόθηκαν από τον χρήστη.

Το πακέτο ***Settings***, περιέχει μία υποκατηγορία κλάσεων που ονομάζεται ***Validator***. Αυτές οι κλάσεις χρησιμοποιούνται για να επικυρώσουν τα δεδομένα που εισήχθησαν στο σύστημα μέσω ενός αρχείου τύπου JSON. Αυτοί οι επικυρωτές δημιουργούνται από αντικείμενα τύπου ***MachineSettings***, σύμφωνα με το αρχείο JSON.

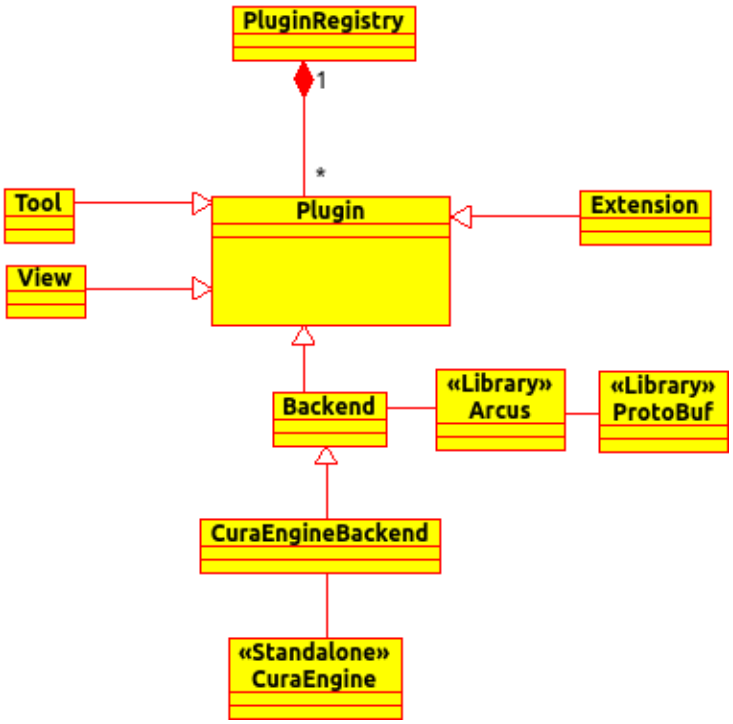
2.2.7 View Package

Το πακέτο ***View*** περιέχει τις κλάσεις οι οποίες είναι υπεύθυνες για την προβολή όλων των αντικειμένων στην γραφική διεπαφή. Η κλάση ***View*** είναι υπεύθυνη για την προβολή των αντικειμένων στην γραφική διεπαφή. Επίσης, αποδίδει (rendering) στην σκηνή τα

αντικείμενα με τα mesh χαρακτηριστικά τους. Η κλάση ***Renderer*** δημιουργείται από την εφαρμογή (Application) και δημιουργεί τα επίπεδα της σκηνής στην γραφική διεπαφή με την βοήθεια της βιβλιοθήκης PyQt6.

2.3 Ενσωμάτωση Plugin στο URANIUM

Το URANIUM έχει την δυνατότητα να ενσωματώνει πολλά plugins σε όλα του τα πακέτα, όπως αναφέρθηκε και προηγουμένως. Στο Σχήμα 8 μπορούμε να δούμε πιο αναλυτικά τον τρόπο εισαγωγής των plugins στο framework.



Σχήμα 8. UML Διάγραμμα Ενσωμάτωσης Plugins

Αυτό που παρατηρούμε είναι πως η σύνδεση γίνεται μέσω της κλάσης ***PluginRegistery*** που ανήκει στο πακέτο ***Core***. Έχουμε σχέση συνάθροισης, με τουλάχιστον ένα πεδίο της κλάσης ***Plugin*** να χρησιμοποιείται από την κλάση ***PluginRegistry***. Ο λόγος της

ύπαρξης της συνάθροισης οφείλεται στο γεγονός πως τα πεδία της κλάσης ***Plugin*** θα πρέπει μόνο να επεξεργάζονται από την κλάση ***PluginRegistry***.

Εκτός από αυτή την κλάση διαπιστώνουμε πως υπάρχει και επικοινωνία με την κλάση ***Backend*** και την κλάση ***View***. Αυτές οι τρεις είναι οι σημαντικότερες συσχετίσεις διότι έτσι το νέο plugin θα μπορέσει να προσφέρει τα αντικείμενά του στο λογισμικό CURA. Οι νέες λειτουργίες που θα προσφέρει το plugin στο λογισμικό, θα το βοηθήσουν ως προς την επέκτασή του. Συνεπώς, ένας χρήστης θα μπορέσει να χρησιμοποιεί παραπάνω λειτουργίες στο πρόγραμμα.

Αναλυτικότερα, η κλάση ***Plugin*** αποτελεί θυγατρική κλάση των υπόλοιπων κλάσεων, ώστε να γίνονται γνωστά τα πεδία της και οι μέθοδοι στα υπόλοιπα δομικά στοιχεία του λογισμικού. Έτσι, τα αντικείμενα που παρέχει το νέο plugin θα μπορούν να επεξεργαστούν από τις υπόλοιπες κλάσεις, χωρίς να μπορούν να τροποποιηθούν.

Η κλάση ***Tool*** δημιουργεί αντικείμενα με την βοήθεια της βιβλιοθήκης PyQt6, ώστε να προβάλλονται στο UI της εφαρμογής. Σε αυτή συνεισφέρει επίσης και η κλάση ***View*** τα δικά της αντικείμενα. Έτσι, το GUI θα εμφανίζει τις εκάστοτε λειτουργίες που προσφέρει το plugin. Το αποτέλεσμα αυτής της σύνδεσης καθιστά πιο ευέλικτο το σύστημα χωρίς να το επηρεάζει σε θέματα δέσμευσης μνήμης ή/και χρόνου εκτέλεσης των υποπρογραμμάτων.

Συνοπτικά, η εγκαθίδρυση νέων plugin στο σύστημα καθιστάτε χωρίς να είναι αναγκαία μία τροποποίηση στον πηγαίο κώδικα του λογισμικού συνολικά. Έτσι, η παραγωγή νέων plugin επεκτείνουν τις λειτουργίες του λογισμικού με ισορροπημένο τρόπο και με ταυτόχρονη αποφυγή δυσάρεστων επιπλοκών, όπως η δέσμευση μνήμης από το σύστημα ή/και οι απαιτήσεις υψηλού πλήθους υπολογιστών πόρων, επιβραδύνοντας έτσι άλλες λειτουργίες.

Κεφάλαιο 3. Σχεδίαση του Plugin

3.1 Ανάλυση Απαιτήσεων του Λογισμικού

Τα .3mf αρχεία εκτός από τα βασικά δεδομένα περιέχουν και κάποια μεταδεδομένα, τα οποία δεν είναι φανερά στον χρήστη. Το plugin που σχεδιάστηκε έχει ως στόχο των εμφάνιση αυτών των μεταδεδομένων στην γραφική διεπαφή του λογισμικού CURA.

Ως πρώτο στάδιο του σχεδίασης του plugin πρέπει να αναλυθούν οι απαιτήσεις του. Δηλαδή, τον σκοπό για την δημιουργία του. Η ανάλυση απαιτήσεων προέρχεται από τις λεγόμενες "Ιστορίες Χρήστη" (User Stories). Πρόκειται για τις επιθυμητές λειτουργίες που θέτονται για νέο λογισμικό. Με βάσει αυτές τις απαιτήσεις μπορούν να ξεκινήσουν τα πρώτα βήματα ως προς την σχεδίαση. Μέσω των User Stories εξάγονται τα Use Cases (επιθυμητές λειτουργίες του λογισμικού). Τα Use Cases αποτελούν μέθοδοι κλάσεων όπου εκτελούν την εκάστοτε λειτουργία.

Στην μηχανική λογισμικού, οι απαιτήσεις διαχωρίζονται σε λειτουργικές (functional requirements) και μη λειτουργικές απαιτήσεις (non-functional requirements). Οι λειτουργικές απαιτήσεις αποτελούνται από τα user stories, ενώ οι μη λειτουργικές απαιτήσεις ενός συστήματος αποτελούν την συντήρηση του προϊόντος, με σκοπό την επέκταση του λογισμικού χωρίς να επιβαρύνεται το λογισμικό. Ακόμα μία μη λειτουργική απαίτηση είναι η απλότητα χρήσης του λογισμικού. Σύμφωνα με αυτήν την απαίτηση, ο χρήστης πρέπει να βρίσκεται σε θέση να το χρησιμοποιήσει και να το μάθει με ευκολία. Τέλος, ένας εξίσου σημαντικός παράγοντας είναι η απόδοση του λογισμικού αυτού. Σύμφωνα με αυτήν, η εμφάνιση των

μεταδεδομένων δεν πρέπει να γίνεται με καθυστέρηση ή να μην καταλαμβάνει σημαντικούς υπολογιστικούς πόρους από το σύστημα.

3.1.1 User Stories

Τα user stories τα οποία τέθηκαν για την ανάπτυξη του plugin ονομάζονται και αλλιώς requirements (απαιτήσεις). Στον Πίνακα 1 μπορούμε να δούμε αναλυτικότερα τα user stories.

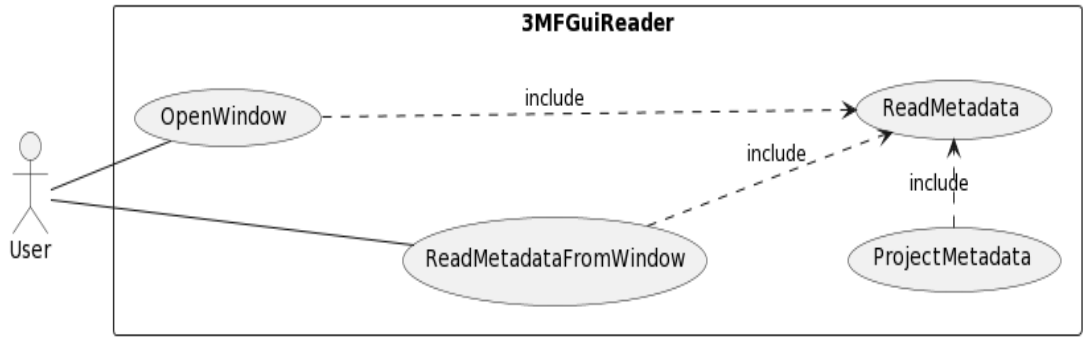
User Story ID	Ως <Τύπος Χρήστη >	Θέλω <Ενέργεια>	Έτσι ώστε <Όφελος/Αξία>
US1	Χρήστης	Να μπορώ να διαβάσω τα μεταδεδομένα ενός αρχείου .3mf	Να μπορώ να δω τις λεπτομέρειες του συγκεκριμένου αρχείου, όπως κάτοχο, μέγεθος και κάποιες ιδιαιτερότητες του αρχείου
US2	Χρήστης	Να μπορώ να βλέπω τα δεδομένα στο κεντρικό παράθυρο του λογισμικού CURA	Να μπορώ να βλέπω και εντός του προγράμματος τα μεταδεδομένα των .3mf αρχείων σε ένα παραθυράκι

Πίνακας 1. Απαιτήσεις Νέου Plugin

Με την χρήση των αυτών των user stories μπορούν να αναπτυχθούν και τα use cases του plugin. Στο σύνολο προκύπτουν τέσσερα use cases, τα οποία θα αναπτυχθούν στην επόμενη υπό-ενότητα.

3.1.2 Use Cases

Από τα user stories θα μπορέσουμε να σχεδιάσουμε τα use cases. Τα use cases θα μας δείξουν μία πιο προσιτή μέθοδο ώστε να κατανοήσουμε πως λειτουργεί ένα σύστημα. Μία αναπαράσταση του συστήματος φαίνεται στο Σχήμα 9



Σχήμα 9. UML use case διάγραμμα plugin

Στο σχήμα βλέπουμε την αλληλεπίδραση που έχει ο χρήστης. Ο χρήστης θα είναι σε θέση να ανοίξει ένα παράθυρο στο οποίο θα έχει την επιλογή να διαβάσει τα μεταδεδομένα ενός .3mf αρχείου. Η μέθοδος ***Open Window*** θα είναι υπεύθυνη για το άνοιγμα του συγκεκριμένου παραθύρου, μέσω ενός κουμπιού που θα πατάει ο χρήστης. Επίσης, Θα διαβάζονται τα μεταδεδομένα από το σύστημα με την βοήθεια της μεθόδου ***Read Metadata***. Εκτός από το σύστημα, σχέση με αυτή την μέθοδο έχει και ο χρήστης, διότι αυτός αποτελεί τον τελικό αποδέκτη και θα αλληλεπιδράσει διαβάζοντας τα μεταδεδομένα (***Read Metadata From Window***). Τέλος, η μέθοδος

Project Metadata είναι υπεύθυνη για την προβολή των μεταδεδομένων στο παράθυρο που έχει ανοίξει ο χρήστης.

Παρακάτω θα δούμε αναλυτικότερα τις τρεις μεθόδους πως λειτουργούν με την μορφή πινάκων. Ο τρόπος γραφής είναι σημαντικός για να κατανοηθεί η εκάστοτε λειτουργία των μεθόδων. Επίσης, κάθε μέθοδος σχεδιάζεται και με κάποιες εξαιρέσεις οι οποίες πρέπει να αναλυθούν.

3.1.2.1 Open Window

Στον πίνακα 1, μπορούμε να δούμε αναλυτικότερα τα στοιχεία της μεθόδου που σχετίζεται με το άνοιγμα του παραθύρου.

Use Case ID	UC1
Actor	Χρήστης
Description & Goals	Το use case αυτό είναι υπεύθυνο για το άνοιγμα του παραθύρου στην γραφική διεπαφή.
Pre-Conditions	1. Η εφαρμογή CURA λειτουργεί.
Main Flow of Events	1. Το use case ξεκινά όταν ο χρήστης επιλέξει την ρύθμιση "Open .3mf Reader". 2. Το σύστημα δέχεται το αίτημα και ανοίγει ένα παράθυρο στην γραφική διεπαφή του CURA.
Alternative Flow	-
Post-Condition	1. Έχει ανοιχθεί το παράθυρο στην γραφική διεπαφή.
Exceptions	-

Πίνακας 2. Open Window Use Case

Το use case με ID UC1 προέκυψε αρχικά από το US1. Επίσης, χρειαζόμαστε την λειτουργία ώστε να ανοίγει το παράθυρο. Το κάθε μέρος του use case περιγράφει τον τρόπο με τον οποίο πρέπει να λειτουργεί σαν οντότητα.

3.1.2.2 Read Metadata

Στον πίνακα 2, μπορούμε να δούμε αναλυτικότερα τα στοιχεία της μεθόδου που σχετίζεται με την ανάγνωση των μεταδεδομένων.

Use Case ID	UC2
Actor	Χρήστης
Description & Goals	Το use case αυτό είναι υπεύθυνο για την συλλογή της πληροφορίας των μεταδεδομένων ενός .3mf αρχείου.
Pre-Conditions	1. Η εφαρμογή CURA λειτουργεί. 2. Το .3mf αρχείο να υπάρχει και να έχει ανοιχτεί από το σύστημα.
Main Flow of Events	1. Το use case ξεκινά όταν ο χρήστης έχει φορτώσει το .3mf αρχείο στην εφαρμογή του CURA. 2. Το σύστημα δέχεται το πλήρες path του αρχείου προέλευσης και διαβάζει τα μεταδεδομένα.
Alternative Flow	-
Post-Condition	1. Έχουν διαβαστεί τα μεταδεδομένα από το σύστημα.
Exceptions	1. Να προβάλλεται μήνυμα σφάλματος στην περίπτωση που το αρχείο είναι κενό.

Πίνακας 3. Read Metadata Use Case

Το use case με ID UC2 προέρχεται και αυτό από το user story 1. Κατά το άνοιγμα του παραθύρου θα πρέπει να διαβάζονται τα μεταδεδομένα από το σύστημα παράλληλα, με την εμφάνισή τους να σχετίζονται με το επόμενο και τελευταίο use case.

3.1.2.3 Project Metadata

Στον πίνακα 3, μπορούμε να δούμε αναλυτικότερα τα στοιχεία της μεθόδου που σχετίζεται με την προβολή των μεταδεδομένων στο νέο παράθυρο της γραφικής διεπαφής.

Use Case ID	UC3
Actor	-
Description & Goals	Το use case αυτό είναι υπεύθυνο για την εμφάνιση των μεταδεδομένων στο παράθυρο που ανοίχθηκε.
Pre-Conditions	1. Η εφαρμογή CURA λειτουργεί. 2. Το .3mf αρχείο να υπάρχει και να έχει διαβαστεί από το σύστημα.
Main Flow of Events	1. Το use case ξεκινά όταν έχουν διαβαστεί τα μεταδεδομένα από το σύστημα. 2. Το σύστημα προβάλει τα μεταδεδομένα ενός αρχείου .3mf στο παράθυρο που ανοίχθηκε στην γραφική διεπαφή.
Alternative Flow	-
Post-Condition	1. Τα μεταδεδομένα έχουν εμφανιστεί στο παράθυρο της γραφικής διεπαφής.
Exceptions	1. Προβάλει μήνυμα λάθος στην περίπτωση που το σύστημα δεν έχει διαβάσει τα μεταδεδομένα του αρχείου.

Πίνακας 4. Project Metadata Use Case

Το use case με ID UC3 προέρχεται από το user story 2. Χρειαζόμαστε έναν τρόπο ώστε τα μεταδεδομένα του αρχείου που θα διαβαστούν από το σύστημα να προβάλλονται στο παράθυρο. Έτσι, ο χρήστης θα μπορέσει να τα βλέπει.

3.2 Σχεδιασμός Λογισμικού

Οι βασικές απαιτήσεις για το λογισμικό έχουν αναλυθεί και δημιουργήθηκαν οι μέθοδοι με τον τρόπο λειτουργίας που έχουν. Το δεύτερο βήμα κατά την ανάπτυξη του plugin είναι η σχεδίαση του και τέλος η υλοποίηση.

Η σχεδίαση του λογισμικού αποτελείται από δύο βασικά στάδια. Το πρώτο στάδιο αποτελεί την αρχιτεκτονική σχεδίαση στην οποία στόχος είναι να κατασκευαστεί ένας σχέδιο του λογισμικού, το οποίο εξηγεί γενικά την λειτουργία του συστήματος, μαζί με τα κομμάτια που θα αποτελείται. Το δεύτερο στάδιο αποτελεί την τεχνική σχεδίαση. Στόχος της τεχνικής σχεδίασης είναι να κατασκευαστεί ένα σχέδιο το οποίο θα εξηγεί σε έναν προγραμματιστή τι θα κάνει το σύστημα και πως θα προγραμματιστεί.

Η αρχιτεκτονική του λογισμικού αποτελείται από κλάσεις, οι οποίες επικοινωνούν μεταξύ τους με βάσει τα αντικείμενα που δημιουργούν. Γενικά μία κλάση αποτελεί κατά κάποιον τρόπο μία δομή δεδομένων για τα αντικείμενα. Οι κλάσεις μεταξύ τους έχουν μία λογική σχέση που τις συνδέει. Η σύνδεση αυτή αφορά τα αντικείμενα που χρησιμοποιεί η κάθε κλάση. Για την υλοποίηση της αρχιτεκτονικής του δικτύου χρησιμοποιείται η γλώσσα περιγραφής UML (βλ. Παράρτημα Α.1).

Η UML αποτελεί πρότυπο κατά την σχεδίαση λογισμικών, με αυτή να έχει πρωταρχικό ως προς την αναπαράσταση των κλάσεων. Με λίγα λόγια παρουσιάζεται η αλληλεπίδραση των κλάσεων μεταξύ

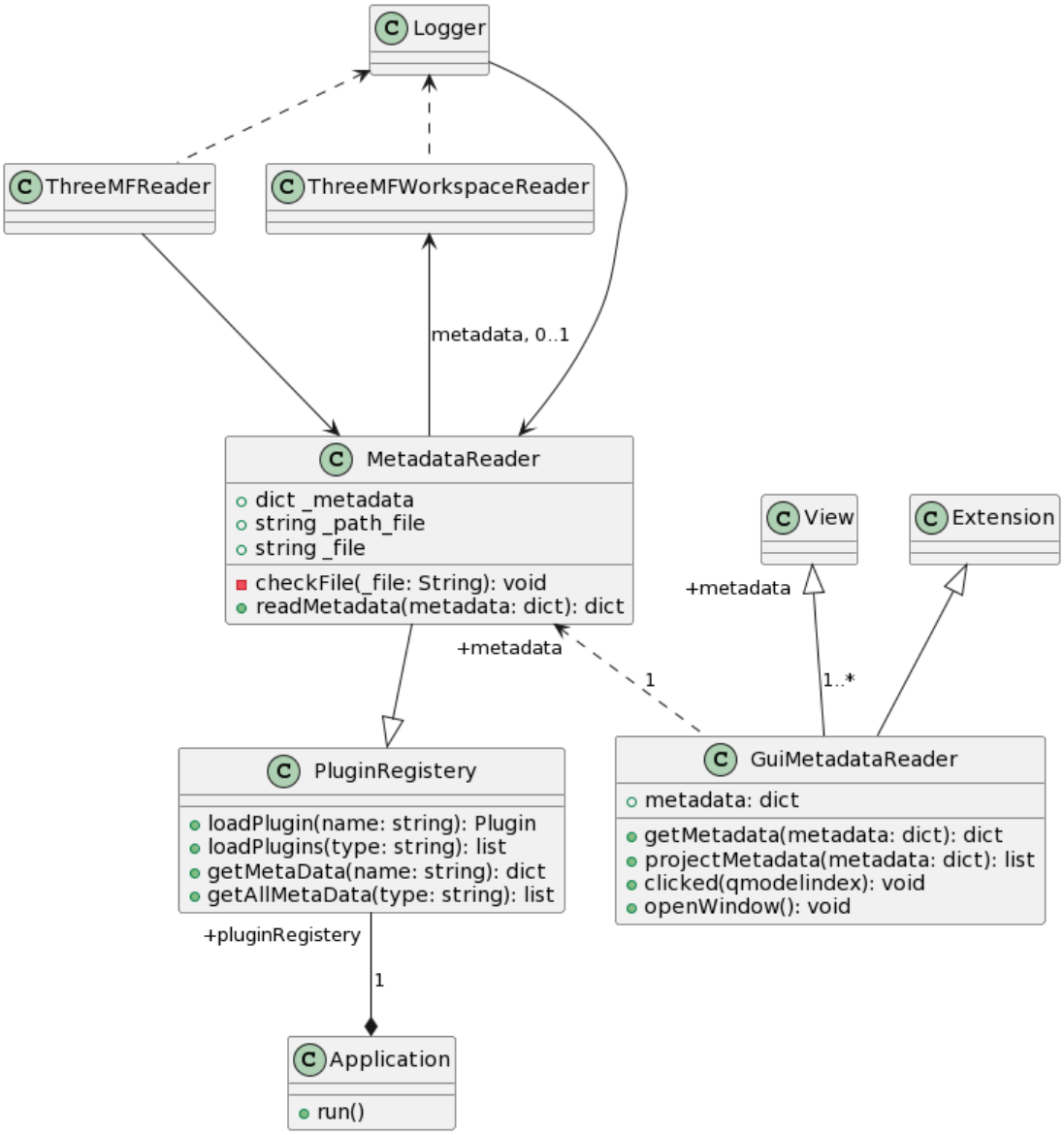
τους. Υπάρχουν πολλοί τρόποι σχεδίασης με τα UML. Οι πιο βασικοί είναι τα διαγράμματα πακέτων (UML Package Diagram) και διαγράμματα κλάσεων (UML Class Diagram). Άλλο ένα βασικό UML διάγραμμα είναι της ακολουθίας (UML Sequence Diagram). Το διάγραμμα πακέτων αναπαριστάτε στο πρώτο στάδιο σχεδίασης, ενώ τα διαγράμματα κλάσεων και ακολουθίας χρησιμοποιούνται στο δεύτερο στάδιο σχεδίασης.

3.2.1 Αρχιτεκτονική Λογισμικού

Η αρχιτεκτονική του λογισμικού βασίζεται στην γλώσσα των UML. Στο σχήμα 10 παρατηρούμε την αρχιτεκτονική του λογισμικού καθώς και την επικοινωνία μεταξύ των αντικειμένων.

Οι πρόσθετες κλάσεις που υλοποιήθηκαν είναι η ***MetadataReader*** και η ***GuiMetadataReader***. Αυτές οι κλάσεις δημιουργούν τα απαραίτητα αντικείμενα που χρειαζόμαστε έτσι ώστε να μπορέσουν να διαβαστούν τα αντικείμενα και να προβληθεί η πληροφορία στην γραφική διεπαφή. Αναλυτικότερα, η κλάση ***MetadataReader*** διαβάζει την απαραίτητη πληροφορία από τις υπάρχουσες κλάσεις του συστήματος ***ThreeMFReader*** και ***ThreeMFWorkspaceReader***. Υπάρχει συσχέτιση μεταξύ των κλάσεων ώστε να στέλνεται η απαραίτητη πληροφορία. Στόχος της είναι να διαβάσει το path file του αρχείου και μετέπειτα να διαβάζει την πληροφορία του αρχείου. Ωστόσο με την διαφορά πως διαβάζει τα μεταδεδομένα του αρχείου και όχι τα mesh δεδομένα. Η κλάση ***GuiMetadataReader*** συνδέεται με την κλάση ***MetadataReader*** μέσω σχέσης εξάρτησης. Η κλάση χρειάζεται τα μεταδομένα ώστε να τα προωθήσει στην κλάση ***View*** για να προβληθούν στην οθόνη.

Τέλος, υπάρχει σχέση κληρονομικότητας με την κλάση ***PluginRegistery***, έτσι ώστε το plugin να ενσωματωθεί με το λογισμικό του CURA.



Σχήμα 10. UML Διάγραμμα Κλάσεων Plugin

Στο UML διάγραμμα παρατηρούμε την επικοινωνία των αντικειμένων μεταξύ των κλάσεων. Ανταλλάσσεται η πληροφορία μεταξύ τους έτσι να υπάρχει η επιθυμητή λειτουργία του plugin. Η κλάση ***MetadataReader*** έχει ως πεδία ένα λεξικό με όνομα **metadata**. Το πεδίο αυτό είναι υπεύθυνο να αποθηκεύει προσωρινά

τα μεταδεδομένα των αρχείων .3mf. Το πεδίο **_path_file** είναι υπεύθυνο να διαβάζει το path του αρχείου. Δηλαδή, την τοποθεσία του στον δίσκο κατά την ανάγνωση του αρχείου. Τέλος, το πεδίο **file** χρησιμοποιεί το path ώστε να ανοίξει το αρχείο. Με την βοήθεια της βιβλιοθήκης της python μπορεί να διαβάσει όλα τα δεδομένα. Ωστόσο, αυτά που θα χρειαστούμε θα είναι μόνο τα μεταδεδομένα και όχι τα mesh δεδομένα του αρχείου.

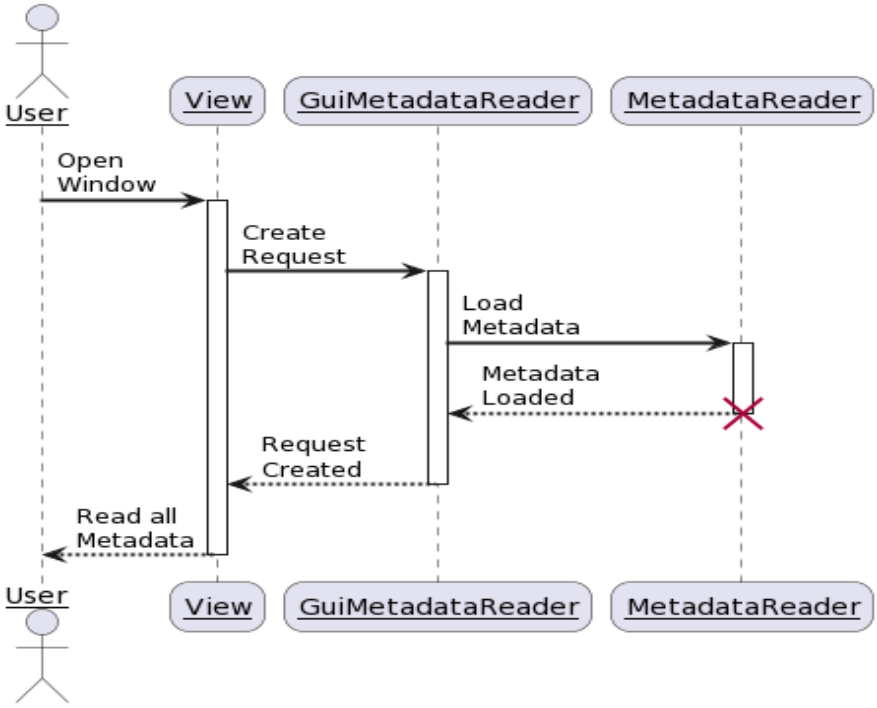
Με το πέρας της ανάγνωσης των μεταδεδομένων, η κλάση αυτή θα επικοινωνήσει με την άλλη κλάση ώστε να τα προβάλει. Η κλάση ***GuiMetadaraReader*** θα δεχτεί τα μεταδεδομένα από την κλάση ***MetadataReader***. Καθώς έχει ανοίξει το παράθυρο, θα ξεκινήσει να τα προβάλει στο γραφικό περιβάλλον. Αυτό γίνεται με την βοήθεια της βιβλιοθήκης PyQt6, με την οποία λειτουργού οι θυγατρικές κλάσεις της, ***View*** και ***Extension***.

3.2.2 Ανάλυση Λειτουργίας Λογισμικού

Καθώς έχει σχεδιαστεί το λογισμικό, το επόμενο βήμα είναι να δούμε κάποιες περιπτώσεις λειτουργίας και αλληλεπίδρασης του χρήστη με τις μεθόδους των κλάσεων. Αυτό μπορούμε να το περιγράψουμε από ένα ακολουθιακό διάγραμμα.

Στο σχήμα 11 βλέπουμε τον τρόπο αλληλεπίδρασης του χρήστη με το σύστημα. Ακολουθούνται οι συγκεκριμένες ενέργειες στο σύστημα, έτσι ώστε να εμφανιστούν τα μεταδεδομένα στο γραφικό περιβάλλον της εφαρμογής. Ο χρήστης είναι αυτό που επιλέγει το άνοιγμα του παραθύρου μέσω των ρυθμίσεων της εφαρμογής. Έπειτα, η εφαρμογή επικοινωνεί με την κλάση ***GuiMetadataReader*** μέσω της δημιουργίας του αιτήματος. Το αίτημα αυτό στέλνεται στην κλάση ***MetadataReader*** με συνέπεια να ξεκινήσει η ανάγνωση των μεταδεδομένων του .3mf αρχείου. Καθώς έχουν διαβαστεί, η κλάση

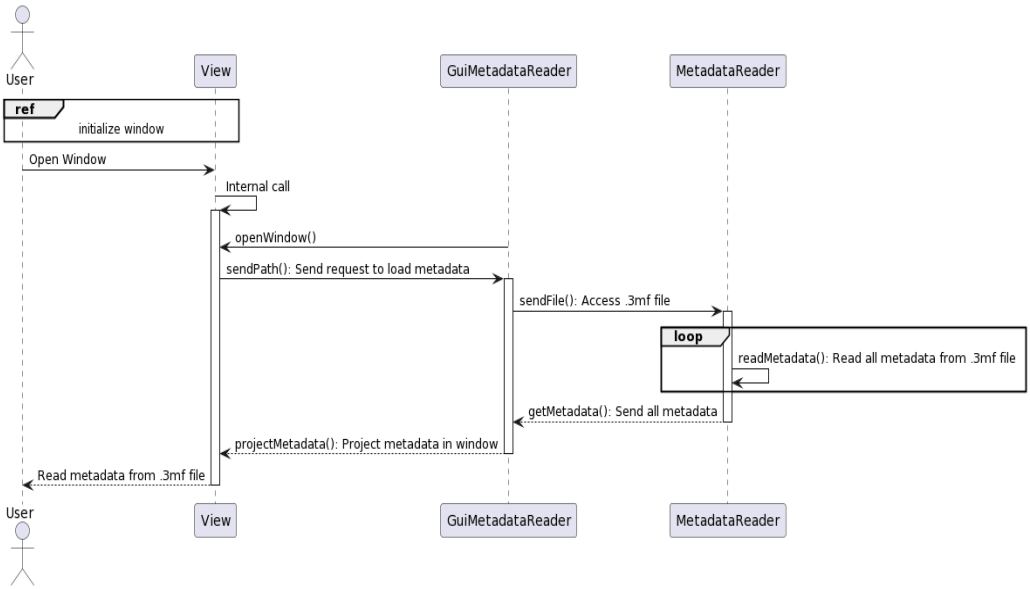
τα προωθεί πίσω στις άλλες κλάσεις, με αποτέλεσμα τελικά να προβάλλεται στην γραφική διεπαφή.



Σχήμα 11. Ακολουθιακό Διάγραμμα Λειτουργίας Κλάσεων κατά την Κλήση του Χρήστη

Υπάρχει και ένας άλλος τρόπος λειτουργίας του ακολουθιακού διαγράμματος που χρησιμοποιούνται και οι μέθοδοι των κλάσεων, ώστε να υπάρχει πιο αναλυτική εξήγηση. Μέσω των μεθόδων, μπορούμε να αντικρίσουμε την διαδρομή των αντικειμένων που ακολουθούν. Για να ξεκινήσει η διαδικασία ανάγνωσης, θα πρέπει ο χρήστης να επιλέξει την εντολή ***Open Metadata File***. Η διαδικασία αυτή γίνεται από την μέθοδο ***openWindow()***. Η συγκεκριμένη μέθοδος χρησιμοποιεί έναν τύπο αντικειμένων **ActionListener** όπου στην ουσία περιμένει να εκτελέσει μια ενέργεια μέσα στο σύστημα. Καθώς ανοίγει το παράθυρο, το επόμενο βήμα είναι να ξεκινήσει η ανάγνωση των μεταδεδομένων. Τα αντικείμενα τύπου ***metadata*** αποθηκεύουν την χρήσιμη πληροφορία μέσα σε ένα λεξικό, στο οποίο κάθε κλειδί (key) περιέχει το index που βρέθηκε στο κείμενο και το

String που διαβάστηκε μέσα στο λεξικό. Ακολουθείται μία επαναληπτική διαδικασία, μέχρις ότου διαβαστούν όλα τα μεταδεδομένα. Τέλος, η μέθοδος αυτή τα προωθεί στην επόμενη μέθοδο που είναι υπεύθυνη για την προβολή τους στην γραφική διεπαφή. Η μέθοδος ***projectMetadata()*** δέχεται τα δεδομένα της προηγούμενης μεθόδου. Στη συνέχεια, με έναν επαναληπτικό βρόχο τα προβάλει στο ανοιγμένο παράθυρο του χρήστη. Με οπτικό τρόπο παρατηρείτε η παραπάνω διαδικασία στο σχήμα 12.



Σχήμα 12. Ακολουθιακό Διάγραμμα Λειτουργίας Plugin

Μέσω της συγκεκριμένης διαδικασίας, θα μπορέσει ο χρήστης να διαβάσει τα μεταδεδομένα του .3mf αρχείου. Δίνοντας την αντίστοιχη εντολή στο CURA, το CURA θα την προωθήσει μέσω του πακέτου View στο πακέτο Core και αυτό με την σειρά του θα επικοινωνήσει με το plugin. Έπειτα, η διαδικασία ανάγνωσης γίνεται μέσω του εξωτερικού λογισμικού, εμφανίζοντας έτσι τα αποτελέσματα στην οθόνη.

Κεφάλαιο 4. Υλοποίηση και Ενσωμάτωση του Plugin

4.1 Υλοποίηση Εξωτερικού Λογισμικού

Η υλοποίηση του λογισμικού βασίστηκε στην σχεδιαστική διαδικασία που αναλύθηκε στο τρίτο κεφάλαιο. Κάθε use case αποτελεί πρακτικά μια μέθοδο μιας κλάσης. Ωστόσο εξαιρείται αυτό που έχει να κάνει με την ανάγνωση του χρήστη, διότι αποτελεί εννοιολογική οντότητα του διαγράμματος.

Κατά την υλοποίηση, προγραμματίστηκαν οι βασικές κλάσεις του συστήματος. Οι δύο κλάσεις από τις οποίες αποτελείται εκτελούν την απαραίτητη εργασία, δίχως να επιβαρύνεται το λογισμικό του ***CURA***. Το ***CURA*** είναι υπεύθυνο να στείλει το κατάλληλο *path* στο λογισμικό, έτσι ώστε να μπορέσει το plugin να επιτελέσει τον σκοπό του. Κατ' επέκτασιν, οι κλάσεις δημιουργούν τα απαραίτητα αντικείμενα και εναλλάσσονται μεταξύ τους. Τέλος, τα αντικείμενα αυτά θα προβληθούν στην οθόνη και έτσι θα είναι διαθέσιμα στον χρήστη.

Οι κλάσεις και οι μέθοδοι ακολούθησαν την αντικειμενοστρεφή λογική, ώστε να είναι συμβατές με το λογισμικό. Οι κώδικες που αναπτύχθηκαν είναι υλοποιημένοι στην γλώσσα προγραμματισμού ***Python***. Οι κλάσεις υλοποιήθηκαν με την χρήση των κατάλληλων βιβλιοθηκών(βλ. Παράρτημα Β). Η επικοινωνία που εγκαθιδρύεται μεταξύ των κλάσεων γίνεται μέσω του πακέτου ***Controller***, το οποίο θα το δέχεται το τελικό αποτέλεσμα και θα το προωθεί στο πακέτο **View**. Έτσι, η υπεύθυνη για την δημιουργία του βασικού παραθύρου του λογισμικού **CURA** είναι υπεύθυνη και για την δημιουργία ενός διαφορετικό παράθυρο με την απαραίτητη πληροφορία.

4.2 Ενσωμάτωση στο CURA

Το τελικό βήμα για το plugin είναι να ενσωματωθεί στο ίδιο το CURA. Η ενσωμάτωση αποτελεί μία απλή και εύκολη διαδικασία η οποία γίνεται αυτόματα από την κλάση ***PluginRegistry***. Η συγκεκριμένη κλάση είναι υπεύθυνη να ενημερώνει την ίδια την εφαρμογή, όταν πρόκειται για εισαγωγή δεδομένων και εντολών από εξωτερικά λογισμικά. Η σχεδίασή του είναι έτσι ώστε να μπορούν να τρέχουν πολλά εξωτερικά λογισμικά. Η λογική αυτή πηγάζει από την ιδέα των προτύπων (patterns). Τα πρότυπα χρησιμοποιούνται για να κάνουν πιο απλή την δομή ενός λογισμικού και επεκτάσιμη (βλ. Παράρτημα Α.2). Συγκεκριμένα, το πρότυπο που ακολουθείται από το CURA είναι το ***Decorator***. Εμπίπτει στην κατηγορία των δομικών προτύπων (Structural Patterns) και χρησιμοποιείται για να εισάγει νέες υπευθυνότητες σε ένα αντικείμενο. Με λίγα λόγια, του παρέχει και άλλες δυνατότητες.

4.2.1 Πρότυπα του CURA

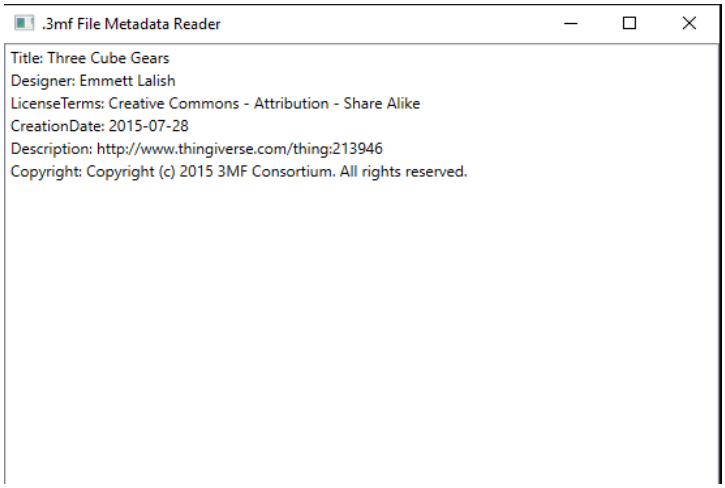
Το λογισμικό **CURA** από την φύση του χρησιμοποιεί πρότυπα, ώστε να μπορεί να επεκτείνει τις λειτουργίες του και να αναβαθμίζεται συνεχώς. Τα πρότυπα που παρατηρήθηκαν στον πηγαίο κώδικά του (source code) είναι κυρίως το ***Decorator*** και το ***Singleton***. Το πρώτο πρότυπο βοηθά στην δυνατότητα των αντικειμένων του προγράμματος να μπορούν να έχουν πολλές λειτουργίες. Ενώ, οι κλάσεις που χρησιμοποιούν το ***Singleton***, είναι κυρίως υπεύθυνες για την εμφάνιση στο γραφικό περιβάλλον. Χρειαζόμαστε αντικείμενα τα οποία να έχουν μία συγκεκριμένη λειτουργικότητα, χωρίς να μπορεί κάποιο άλλο να αποθηκεύσει τα δεδομένα του ή να χρησιμοποιήσει τις

αντίστοιχες μεθόδους. Έτσι, δεν θα μπορούν να γίνουν σφάλματα κατά την λειτουργία του πακέτου ***View***.

4.2.2. Αποτελέσματα Plugin

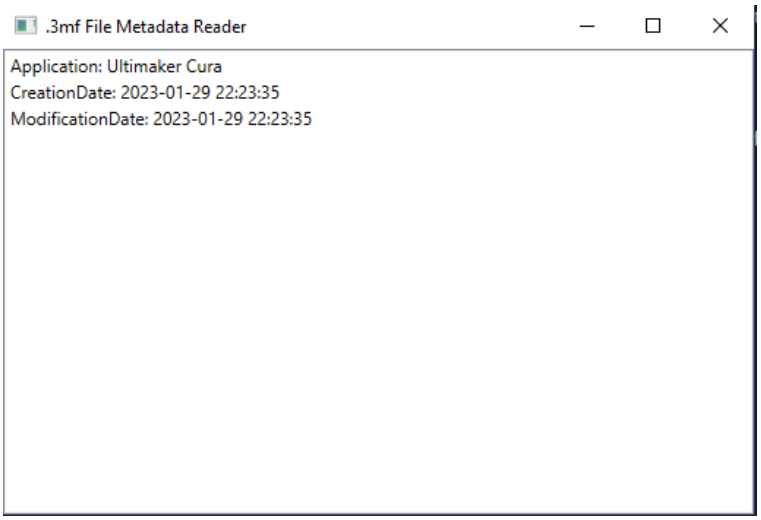
Κατά την υλοποίηση του plugin υπήρχαν πολλά προβλήματα συμβατότητας με την βιβλιοθήκη ***conan*** της ***Python***. Έτσι δεν μπορούσε να τρέξει το πρόγραμμα ***CURA*** από το την γραμμή εντολών, πράγμα που έκανε αδύνατον τον έλεγχο εγκυρότητάς της υλοποίησης εντός του περιβάλλοντος της εφαρμογής. Για αυτό τον λόγο δημιουργήθηκε μία επιπλέον εξωτερική εφαρμογή που εκτελεί την ίδια ακριβώς διαδικασία. Η εφαρμογή αυτή δέχεται ως είσοδο .3mf αρχεία και ανοίγει ένα παράθυρο στο οποίο εμφανίζει τα μεταδεδομένα των.3mf αρχείων. Στα παρακάτω σχήματα εμφανίζονται κάποια αποτελέσματα την εξωτερικής εφαρμογής αυτής.

Έστω ότι έχουμε το αρχείο ***cube_gears.3mf***. Με την εκτέλεση της συγκεκριμένης εφαρμογής δεχόμαστε τα αποτελέσματα του σχήματος 13.



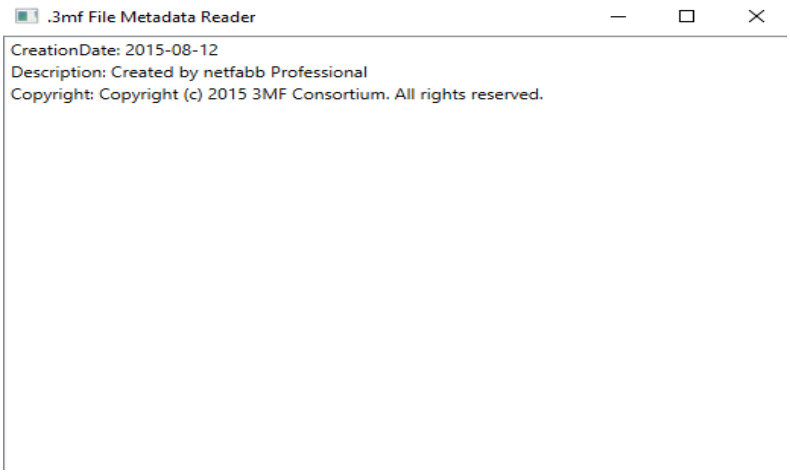
Σχήμα 13. Μεταδεδομένα αρχείου cube_gears.3mf

Το συγκεκριμένο αρχείο αποτελείται από αυτά τα μεταδεδομένα. Βλέπουμε τον κάτοχο του συγκεκριμένου αρχείου και επίσης και τα δικαιώματα που έχει. Άλλο ένα αρχείο που επεξεργάστηκε είναι το ***Mug_project_file.3mf***. Το αρχείο αυτό δεν αποτελεί ένα κλασικό αρχείο τύπου .3mf, καθώς είναι ένα αρχείο project του λογισμικού Cura. Για τον παραπάνω λόγω, εμφανίζονται διαφορετικά μεταδεδομένα. Χαρακτηριστικό μεταδεδομένο των αρχείων που προέρχονται από το λογισμικό ως αρχεία αποθήκευσης project είναι το μεταδεδομένο με όνομα Application, το οποίο διαθέτει την τιμή Ultimaker Cura (βλ. Σχήμα 14).



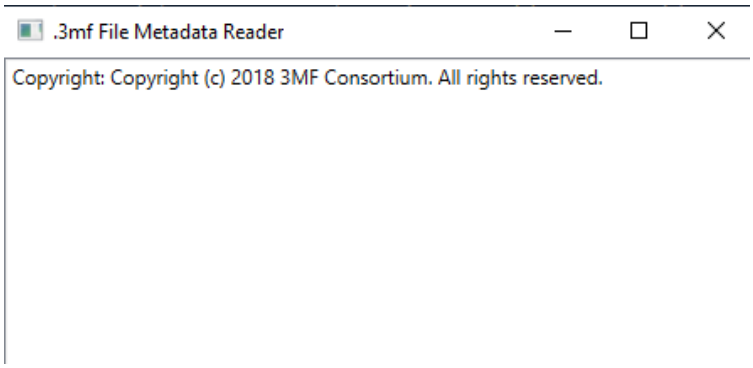
Σχήμα 14. Μεταδεδομένα Mug_project_file.3mf

Με την ανάγνωση δύο διαφορετικών .3mf αρχείων διαπιστώνεται πως το κάθε αρχείο είναι μοναδικό καθώς και τα μεταδεδομένα. Τα μεταδεδομένα αυτά καθορίζονται πρακτικά από τον κάτοχο του αρχείου. Στο σχήμα 15 βλέπουμε τα μεταδεδομένα του αρχείου ***pyramid_vertexcolor.3mf***.



Σχήμα 15. Μεταδεδομένα pyramid_vertexcolor.3mf

Ακόμα ένα παράδειγμα εκτέλεσης είναι το αρχείο ***variable_voronoi.3mf***. Αναλυτικά βλέπουμε τα μεταδεδομένα στο σχήμα 16.



Σχήμα 16. Μεταδεδομένα αρχείου variable_voronoi.3mf

Γενικά, αυτό που παρατηρείται από τα συγκεκριμένα αρχεία είναι πως οι σχεδιαστές ορίζουν το περιεχόμενο και τα πνευματικά δικαιώματα. Δηλαδή, κάποια αρχεία δεν μπορούν θεωρητικά να εκμεταλλευτούν από τον αναγνώστη χωρίς την συγκατάθεση του ιδιοκτήτη ή δημιουργού του αρχείου.

4.3 Απόδοση Λογισμικού

Κατά την ανάπτυξη των λογισμικών χρησιμοποιούνται και κάποιες μετρικές ώστε να εκτιμήσουμε την απόδοσή τους. Έχουν αναπτυχθεί γενικώς πολλές χρήσιμες μετρικές, με τις οποίες εκτιμούμε τον φόρτο εργασίας των κλάσεων. Μία τέτοια μετρική είναι η CBO. Με αυτή την μετρική στην ουσία υπολογίζουμε το πλήθος κλάσεων που χρησιμοποιεί μία κλάση κατά την λειτουργία της. Στο δικό μας λογισμικό μπορούμε να χρησιμοποιήσουμε την συγκεκριμένη μετρική και για τις δύο κλάσεις. Στην κλάση ***MetadataReader*** η μετρική θα έχει την τιμή

$$CBO(MetadataReader) = 0 \qquad \text{(Εξ.1)}$$

καθώς, δεν χρησιμοποιεί καμία πληροφορία από κάποια άλλη κλάση. Η σχέση κληρονομικότητας δεν μετριέται στην συγκεκριμένη μετρική. Ενώ, αντιθέτως η κλάση ***GuiMetadataReader*** έχει

$$CBO(GuiMetadataReader) = 4 \qquad \text{(Εξ.2)}$$

διότι, χρησιμοποιεί αντικείμενα από τέσσερις διαφορετικές κλάσεις (***MetadataReader, QListWidget, QGridLayout, QWidget***). Η συγκεκριμένη μετρική μας δείχνει την σύζευξη μεταξύ κλάσεων. Πράγμα το οποίο διαπιστώνεται και από τις τιμές. Για τον υπολογισμό της σύζευξης χρησιμοποιείται και η μετρική COF. Έστω ότι έχουμε ένα σύνολο $S = \{c_1, c_2, \dots, c_N\}$ που αποτελείται από N κλάσεις. Επίσης, ορίζουμε και μια συνάρτηση $isClient(c_i, c_j): S \times S \to \{0,1\}$. Η συγκεκριμένη συνάρτηση δέχεται την τιμή 1 εφόσον η κλάση c_i χρησιμοποιεί την κλάση c_j. Ενώ αντιθέτως δέχεται την τιμή 0. Στην προκειμένη περίπτωση έχουμε τις παραπάνω 5 κλάσεις $S = \{\boldsymbol{GuiMetadataReader}, \boldsymbol{MetadataReader}, \boldsymbol{QListWidget}, \boldsymbol{QGridLayout}, \boldsymbol{QWidget}\}$. Οπότε η μετρική βγαίνει

$$COF(S) = \frac{Sum\left(isClient(c_i, c_j)\right)}{N * (N-1)} =$$

$$= \frac{(1+1+1+1)+(0+0+0)+(0+0)}{5*4} = \frac{4}{5*4} = \frac{1}{5} = 0.2$$

$$\Rightarrow COF(S) = 0.2 \qquad \text{(Εξ.3)}$$

Με βάσει αυτή την μετρική παρατηρούμε πως υπάρχει μικρή σύζευξη μεταξύ των κλάσεων. Αυτός είναι σημαντικός παράγοντας καθώς έτσι δεν υπάρχει υψηλή πολυπλοκότητα στο σύστημα.

Μία άλλη μετρική που χρησιμοποιείται είναι η LCOM και κατ' επέκτασιν η LCOM2 για να αντικρίσουμε την έλλειψη συνεκτικότητας μεταξύ μεθόδων των κλάσεων. Για την μετρική LCOM θα πρέπει να οριστούν δύο σύνολα P, Q. Το σύνολο Q αποτελείται από τα ζεύγη μεθόδων κλάσεων που χρησιμοποιούν κοινά πεδία. Ενώ το σύνολο P αποτελείται από τις μεθόδους κλάσεων που δεν χρησιμοποιούν κοινά πεδία. Έτσι, θα μπορέσουμε να υπολογίσουμε την μετρική με τον τύπο

$$LCOM(X) = \begin{cases} |P| - |Q| & ,if\ |P| > |Q| \\ 0 & ,otherwise \end{cases} \qquad \text{(Εξ.4)}$$

Στο δικό μας λογισμικό διαπιστώνουμε και για τις δύο μεθόδους η μετρική LCOM είναι 0. Αυτό συμβαίνει καθώς το πλήθος κοινών πεδίων είναι μεγαλύτερο συγκριτικά με το πλήθος των μη κοινών. Ωστόσο, με αυτή την μετρική δεν μπορούμε να διαπιστώσουμε ποσοτικά παρά μόνο ποιοτικά. Έτσι, θα χρησιμοποιήσουμε μία διαφορετική παραλλαγή της μετρικής LCOM. Η μετρική ονομάζεται LCOM2 με την οποία μπορούμε να διαπιστώσουμε καλύτερα εάν υπάρχει συνεκτικότητα μεταξύ των μεθόδων. Συγκεκριμένα δίνεται από τον τύπο

$$LCOM2(X) = 1 - \frac{1}{(m*a)} \sum_{i=1}^{N} ma_i \qquad \text{(Εξ.5)}$$

όπου το α είναι το πλήθος των πεδίων και m το πλήθος των μεθόδων. Επίσης, το ma_i αποτελεί το πλήθος των μεθόδων που χρησιμοποιεί το

πεδίο a_i. Συγκεκριμένα, για την κλάση ***MetadataReader*** βρίσκουμε ότι η μετρική έχει τιμή

$$LCOM2(MetadataReader) = 1 - \frac{(2+2+1)}{3*3} = 1 - \frac{5}{9} = \frac{4}{9} = 0.444 \quad \text{(Εξ.6)}$$

Σύμφωνα με αυτή την μετρική έχουμε κάποια συνοχή μεθόδων ωστόσο όσο μεγάλη. Επιβαρύνεται λίγο η πολυπλοκότητα του λογισμικού καθώς απαιτείται η πληροφορία του αντικειμένου. Αντίστοιχα, για την κλάση ***GuiMetadataReader*** ισχύει

$$LCOM2(GuiMetadataReader) = 1 - \frac{(1+1+1+0+1)}{5*1} = 1 - \frac{4}{5} = \frac{1}{5} = 0.2 \quad \text{(Εξ.7)}$$

Διαπιστώνουμε πως έχουμε μικρή συνεκτικότητα μεταξύ των μεθόδων της συγκεκριμένης κλάσης. Με αυτή την μετρική μπορούμε να διαπιστώσουμε καλύτερα σχετικά με την έλλειψη συνεκτικότητας μεταξύ των μεθόδων συγκριτικά με την προηγούμενη μετρική.

Τέλος, υπάρχουν και κάποιες άλλες μετρικές που δείχνουν την πολυπλοκότητα που υπάρχει στις κλάσεις ατομικά. Μία τέτοια μετρική ονομάζεται WMC. Αυτή υπολογίζεται από την μέθοδο

$$WMC(A) = \sum_{i=1}^{N} C_i \quad \text{(Εξ.8)}$$

όπου το C_i αποτελεί την πολυπλοκότητα τις i μεθόδου στην κλάση. Η συγκριμένη μετρική μπορεί να υπολογιστεί με δύο τρόπους. Είτε με τον υπολογισμό των γραμμών κώδικα, είτε με τον τρόπο McCabe. Εμείς θα χρησιμοποιήσουμε τον δεύτερο τρόπο λόγω της εμφάνισης των for και if loops στον κώδικα. Η μετρική κατά McCabe υπολογίζεται ως

$$WMC(A) = \sum_{i=1}^{N} (number\ of\ conditions\ + 1) \quad \text{(Εξ.9)}$$

όπου

$$number\ of\ \ conditions = \begin{cases} 1 & ,if\ for\ while \\ N-1 & ,switch\ N\ cases \end{cases} \quad \text{(Εξ.10)}$$

Επομένως, στο λογισμικό θα εφαρμοστεί ο παραπάνω τύπος. Για την κλάση ***MetadataReader*** έχουμε ότι

$$WMC(MetadataReader) = (1+1) + (1+1+1) = 5 \quad \text{(Εξ.11)}$$

Για την κλάση ***GuiMetadataReader*** η μετρική έχει τιμή

$$WMC(GuiMetadataReader) = 1 + 1 + (1+1) + 1 + (1+1) = 7 \quad \text{(Εξ.12)}$$

Συγκριτικά με τις δύο μεθόδους αυτές μπορούμε να διαπιστώσουμε πως υπάρχει μεγαλύτερη πολυπλοκότητα στην κλάση ***GuiMetadataReader***. Υπάρχει περισσότερη επεξεργασία, καθώς περιέχει και περισσότερες λειτουργίες.

Μία τελευταία μετρική που χρησιμοποιήθηκε κατά την σχεδίαση του λογισμικού είναι η RFC. Υπολογίζεται από τον τύπο

$$RFC(A) = M + R \quad \text{(Εξ.13)}$$

όπου το R είναι ο αριθμός των μεθόδων που καλεί άλλες μεθόδους κλάσεων και το M είναι το πλήθος των μεθόδων της κλάσης. Για την κλάση ***MetadataReader*** έχουμε

$$RFC(MetadataReader) = 3 + 0 = 3 \quad \text{(Εξ.14)}$$

Για την κλάση ***GuiMetadataReader*** ισχύει ότι

$$RFC(GuiMetadataReader) = 5 + 10 = 15 \quad \text{(Εξ.15)}$$

Παρατηρούμε τεράστια διαφορά ως προς την πολυπλοκότητα των δύο κλάσεων, πράγμα το οποίο είναι λογικό να συμβαίνει καθώς η δεύτερη κλάση διαθέτει και χρησιμοποιεί πολλές μεθόδους.

Παραρτήματα

Παράρτημα Α: Θεωρητικές Έννοιες Τεχνολογίας Λογισμικού

Α.1 Βασικά Πρότυπα UML

Η γλώσσα περιγραφής UML χρησιμοποιείται για να υλοποιηθεί ένα λογισμικό σε αφαιρετικά επίπεδα τα οποία διαχωρίζονται ανάλογα με το άτομο. Δηλαδή, υπάρχουν τρόποι περιγραφής του λογισμικού που γίνεται αντιληπτό στον πελάτη και τόποι περιγραφής του λογισμικού που γίνεται αντιληπτό στον προγραμματιστή. Οι βασικές κατηγορίες UML είναι οι εξής

1. Διάγραμμα Use Case (UML Use Case Diagram)
2. Διάγραμμα Πακέτων (UML Package Diagram)
3. Διάγραμμα Κλάσεων (UML Class Diagram)
4. Ακολουθιακό Διάγραμμα (UML Sequence Diagram)

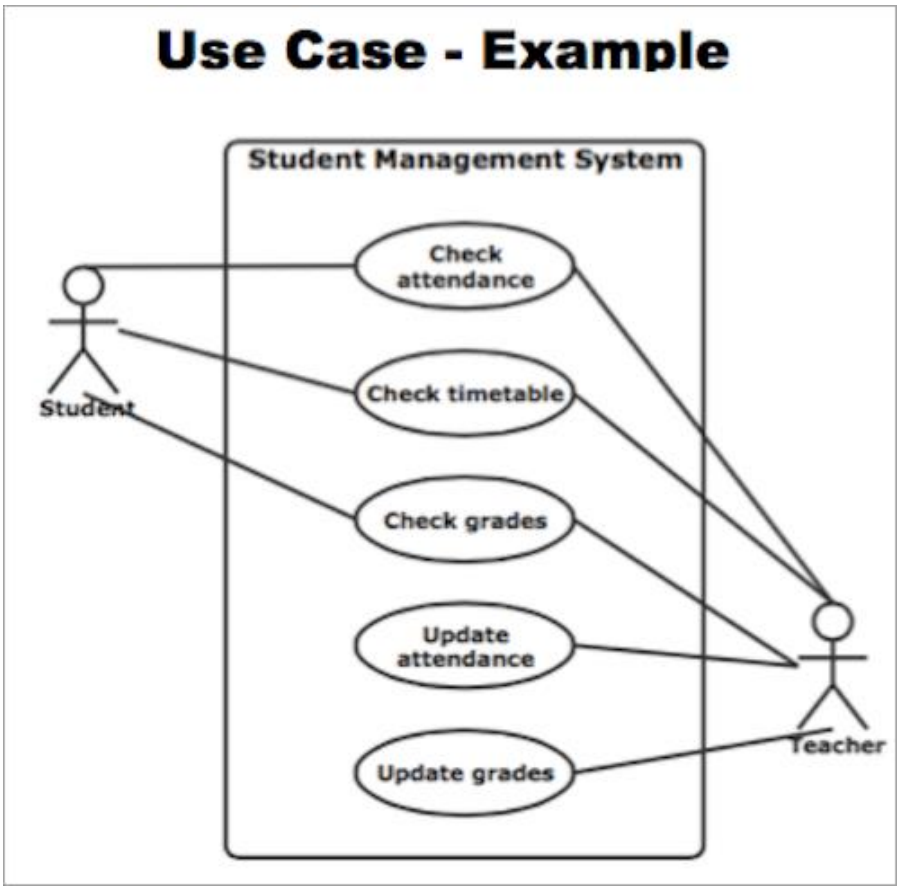
Αυτά τα τέσσερα βασικά διαγράμματα αποτελούν ένα επίπεδο αφαίρεσης και κατανόησης του λογισμικού.

Α.1.1 Διαγράμματα Use Case

Τα use case διαγράμματα αποτελούν μία γραφική αναπαράσταση ενός λογισμικού. Στην αναπαράσταση αυτή δείχνεται ο τρόπος αλληλεπίδρασης του χρήστη με το σύστημα σε υψηλό αφαιρετικό επίπεδο. Ένα τέτοιο διάγραμμα δείχνει την αλληλεπίδραση των use case μέσα στο σύστημα μεταξύ τους. Σε ένα τέτοιο διάγραμμα, με

ελλείψεις παρουσιάζονται τα use cases και με το ανθρωπάκι οι χρήστες.

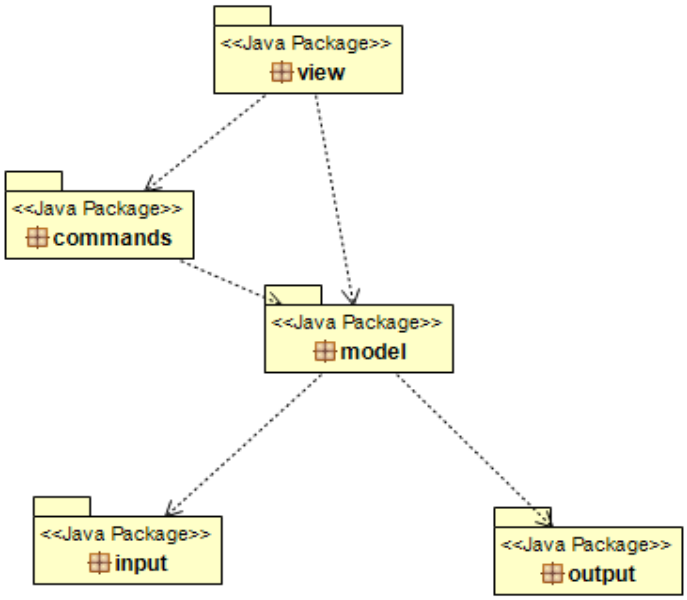
Ενώ ένα use case μπορεί να εμβαθύνει σε πολλές λεπτομέρειες σχετικά με κάθε δυνατότητα που παρέχει στο σύστημα, ένα διάγραμμα use case μπορεί να βοηθήσει στην παροχή ενός σχεδίου υψηλότερου αφαιρετικά επιπέδου του συστήματος, το οποίο γίνεται αντιληπτό από έναν χρήστη. Λόγω της απλοϊκής τους μορφής τους, τα διαγράμματα αυτά μπορούν να αποτελέσουν έναν καλό τρόπο επικοινωνίας μεταξύ χρήστη και σχεδιαστή του λογισμικού. Ένα απλοϊκό παράδειγμα ενός διαγράμματος παρουσιάζεται στο Σχήμα Α.1.



Σχήμα Α.1. Παράδειγμα διαγράμματος use case

Α.1.2 Διαγράμματα Πακέτων

Τα διαγράμματα πακέτων αποτελούν μία μορφής αναπαράστασης των οντοτήτων που αποτελούν το λογισμικό. Ένα πακέτο πρακτικά ομαδοποιεί τις κλάσεις και αντικείμενα. Αυτές οι κλάσεις επικοινωνούν μεταξύ τους με χρήση των αντικειμένων. Στην ουσία τα πακέτα αποτελούν μία πιο αφαιρετική δομή του συστήματος από την σκοπιά του σχεδιαστή. Ένα παράδειγμα διαγράμματος αποτελεί το σχήμα Α.2.

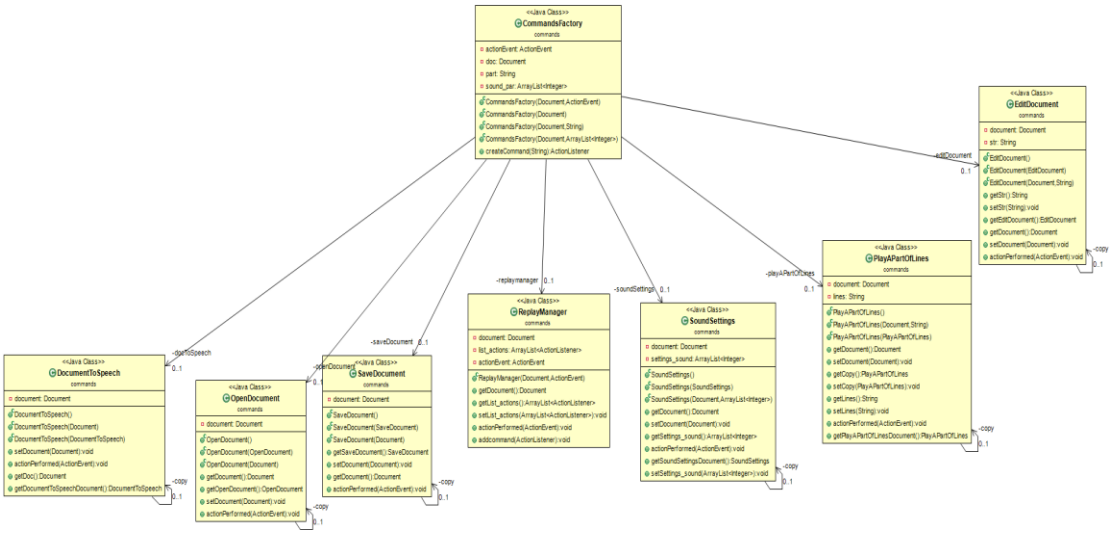


Σχήμα Α.2. Παράδειγμα διαγράμματος πακέτων

Α.1.3 Διαγράμματα κλάσεων

Τα διαγράμματα κλάσεων αποτελούν την βασική σχεδίαση ενός λογισμικού. Σε αυτό το διάγραμμα αποτυπώνονται οι κλάσεις του συστήματος μαζί με τα πεδία (attributes) τις μεθόδους και την σχέση μεταξύ των αντικειμένων τους. Επίσης, αναπαριστάτε η επικοινωνία μεταξύ τους, η οποία δείχνεται από μία συγκεκριμένη σχέση.

Το διάγραμμα κλάσεων αποτελεί το δομικό σχέδιο για την μοντελοποίηση ενός αντικειμενοστρεφούς λογισμικού (object-oriented software). Χρησιμοποιείται ως μια εννοιολογική μοντελοποίηση για την δομή της εφαρμογής και παράλληλα ένα μέσο μετατροπής σε κώδικα. Πρακτικά, ένα διάγραμμα κλάσεων αποτελεί τον σκελετό (αρχικό κώδικα) ενός συστήματος. Ένα παράδειγμα συστήματος αναπαριστάτε στο σχήμα Α.3.



Σχήμα Α.3. Παράδειγμα διαγράμματος κλάσεων

Μία κλάση έχει κάποιες ιδιότητες. Η πρώτη ιδιότητα αφορά την διαφάνεια των πεδίων (μεταβλητών) της. Ένα πεδίο μπορεί να είναι εμφανές σε όλες τις κλάσεις του συστήματος (public attribute) ή να μην είναι σε καμία άλλη εμφανής (private attribute).

Άλλη μία ιδιότητα που έχουν οι κλάσεις είναι η σχέση μεταξύ των αντικειμένων τους. Συνολικά, υπάρχουν έξι διαφορετικές σχέσεις μεταξύ των κλάσεων.

1. Συσχέτιση (Association)
2. Εξάρτηση (Dependency)
3. Συνάθροιση (Aggregation)
4. Σύνθεση (Composition)

5. Περιγραφή/Αναπαράσταση Διεπαφής (Implementation Interface)
6. Κληρονομικότητα (Inheritance)

Οι παραπάνω συσχετίσεις κατηγοριοποιούνται σε δύο κατηγορίες. Την σχέση μεταξύ οντοτήτων (ή πεδίων) [1 έως 4] και σχέση μεταξύ κλάσεων [5, 6].

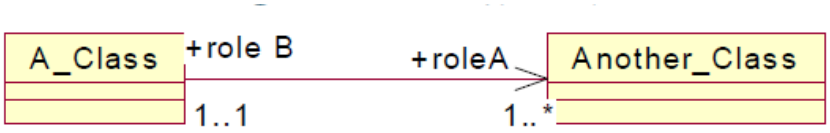
Συσχέτιση:

Μια απλή σχέση συσχέτισης σημαίνει ότι κατά τη διάρκεια εκτέλεσης κάποια αντικείμενα των δύο κλάσεων συνυπάρχουν και συνεργάζονται με κάποιο τρόπο, ρόλο.

Η κατεύθυνση (αν υπάρχει) υποδηλώνει ότι αντικείμενα της ***A_Class*** γνωρίζουν την ύπαρξη και έχουν τη δυνατότητα πρόσβασης στα αντικείμενα της ***Another_Class***.

Το **multiplicity** (αν υπάρχει) υποδηλώνει πόσα αντικείμενα συνεργάζονται: 0..1, 1..*, 0..*, 1..10, 5, 10, n, 1..n, 0..n, κλπ.

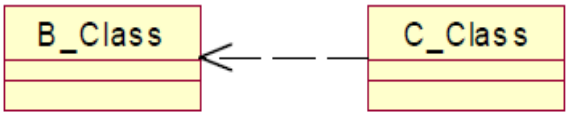
Υπάρχει επίσης δυνατότητα να ονοματίσουμε το ρόλο που παίζουν τα αντικείμενα κάθε κλάσης στη συνεργασία.



Σχήμα Α.4. Σχεδιαστική απεικόνιση συσχέτισης μεταξύ δύο κλάσεων

Εξάρτηση:

Περιγράφεται/Αναπαριστάτε μία σχέσης εξάρτησης μεταξύ δυο κλάσεων. Γενικά η λειτουργία της ***C_Class*** βασίζεται στη ***B_Class***. Αλλαγές στην κλάση ***B_Class*** μπορεί να συνεπάγονται αλλαγές στην κλάση ***C_Class***.

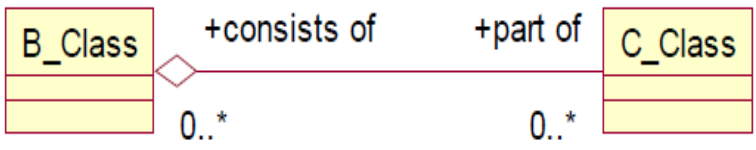


Σχήμα Α.5. Σχεδιαστική απεικόνιση εξάρτησης μεταξύ δύο κλάσεων

Συνάθροιση:

Η σχέση συνάθροισης σημαίνει ότι ένα αντικείμενο της κλάσης ***B_Class*** (πχ ένας άνθρωπος) χαρακτηρίζεται από αντικείμενα της κλάσης ***C_Class*** (πχ μια διεύθυνση). Ένα αντικείμενο της κλάσης ***C_Class*** (πχ η ίδια διεύθυνση) μπορεί να χαρακτηρίζει πολλαπλά αντικείμενα της κλάσης ***B_Class*** (πχ περισσότερους από έναν άνθρωπο που μένουν στο ίδιο σπίτι) ή αντικείμενα (πχ κτίρια) μιας άλλης κλάσης ***Z_Class***.

Επίσης η ύπαρξη των αντικειμένων της ***C_Class*** δεν εξαρτάται άμεσα από την ύπαρξη των αντικειμένων της ***B_Class*** (πχ το πρόγραμμα διαχειρίζεται μια λίστα από διευθύνσεις ακόμα και αν δεν υπάρχουν άνθρωποι που μένουν σε αυτές)

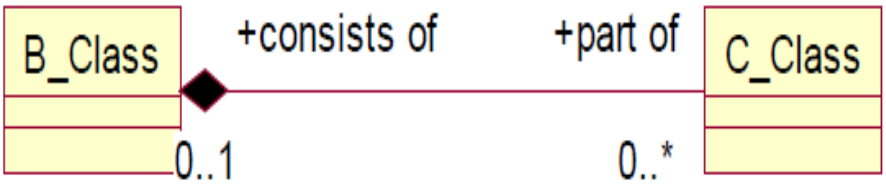


Σχήμα Α.6. Σχεδιαστική απεικόνιση συνάθροισης μεταξύ δύο κλάσεων

Σύνθεση:

Η σχέση σύνθεσης σημαίνει ότι ένα αντικείμενο της κλάσης ***B_Class*** (πχ μια οικογένεια) χαρακτηρίζεται/αποτελείται από αντικείμενα της κλάσης ***C_Class*** (πχ μέλη οικογένειας) και ένα αντικείμενο ***C_Class*** χαρακτηρίζει το πολύ ένα αντικείμενο της ***B_Class*** και εάν ένα αντικείμενο της κλάσης ***C_Class*** χαρακτηρίζει ένα αντικείμενο

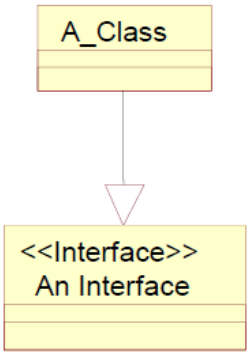
B_Class, τότε δεν έχει λόγο ύπαρξης αν δεν υπάρχει το αντικείμενο της ***B_Class*** το οποίο χαρακτηρίζει.



Σχήμα Α.7. Σχεδιαστική απεικόνιση σύνθεσης μεταξύ δύο κλάσεων

Περιγραφή/Αναπαράσταση Διεπαφής:

Περιγράφει τον τρόπο επικοινωνίας μεταξύ μιας κλάσης και μίας διεπαφής (interface) μέσα σε ένα πακέτο. Η κλάση αυτή υλοποιεί τις μεθόδους της διεπαφής καθώς κληρονομεί τα πεδία της και τις μεθόδους της.

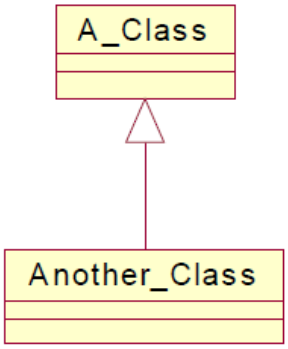


Σχήμα Α.8. Σχεδιαστική απεικόνιση σχέσης κλάσης με διεπαφή

Κληρονομικότητα:

Περιγράφει μία σχέση γενίκευσης/κληρονομικότητας μεταξύ των κλάσεων. Δηλαδή, για παράδειγμα η κλάση ***A_Class*** αποτελεί πρωτεύουσα κλάση, και η κλάση ***Another_Class*** αποτελεί υποκλάση

της άλλης. Η υποκλάση θα κληρονομήσει τα πεδία και τις μεθόδους της πρωτεύουσας κλάσης.



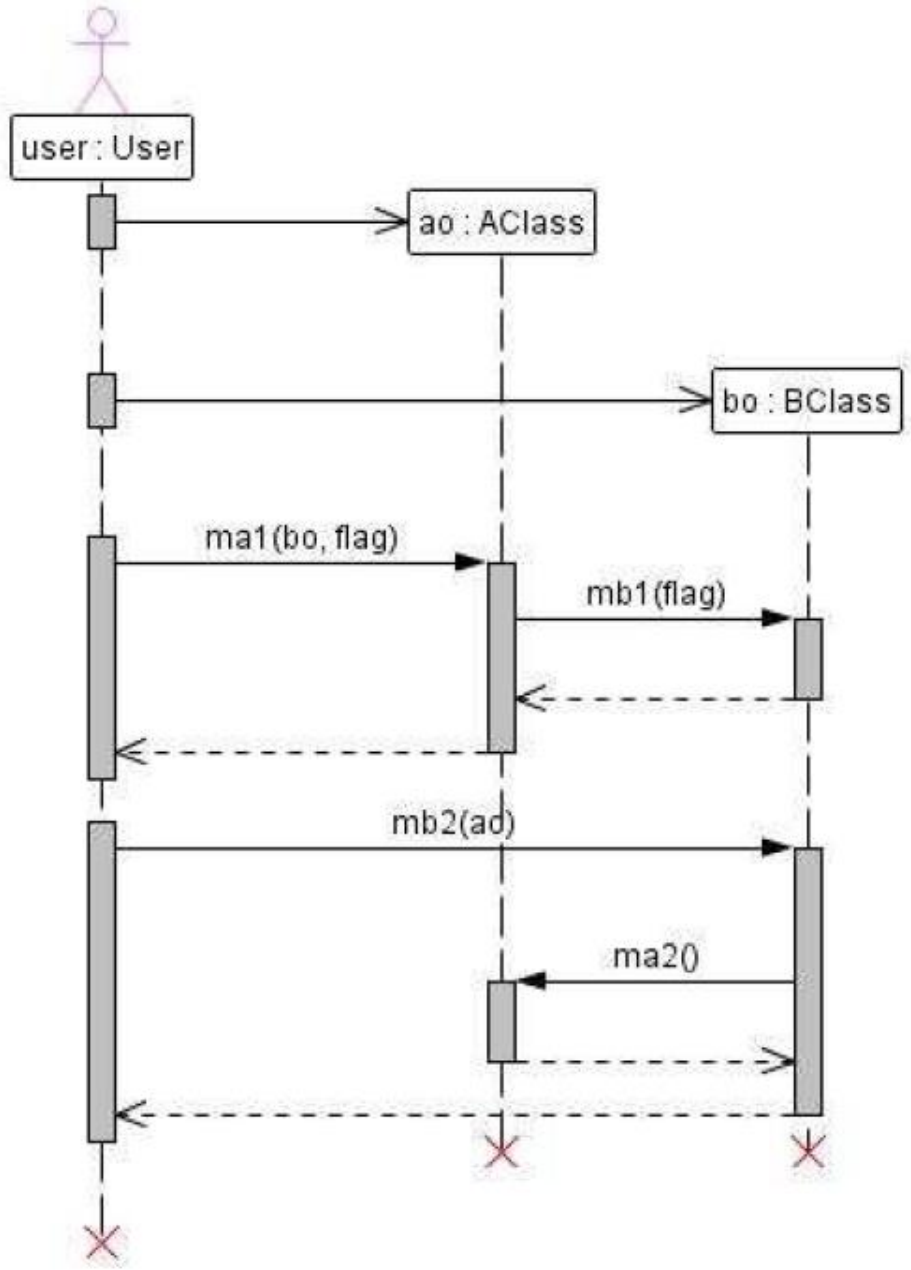
Σχήμα Α.9. Σχεδιαστική απεικόνιση κληρονομικότητας μεταξύ δύο κλάσεων

Α.1.4 Ακολουθιακά Διαγράμματα

Κατά την σχεδίαση ενός λογισμικού, εκτός από την δομή του (κλάσεις, σχέσεις), χρειάζεται να περιγράψουμε και τη συμπεριφορά που έχει λογισμικού. Η περιγραφή αυτή δίνεται από τα ακολουθιακά διαγράμματα. Με το συγκεκριμένο διάγραμμα μπορεί να περιγραφή η αλληλεπίδραση μεταξύ των αντικειμένων κατά την εκτέλεση των διαφόρων περιπτώσεων χρήσης του λογισμικού. Με τον όρο αλληλεπίδραση εννοούμε την κλήση μεθόδων των αντικειμένων. Πιο συνοπτικά μπορούμε να το χαρακτηρίσουμε και σαν ανταλλαγή μηνυμάτων μεταξύ των αντικειμένων των κλάσεων του λογισμικού.

Κατά την σχεδίαση ενός ακολουθιακού διαγράμματος, παρατηρούμε πως η απεικόνιση της αλληλεπίδρασης των αντικειμένων του λογισμικού καταλογίζονται σε δύο άξονες. Στον οριζόντιο άξονα απεικονίζεται ένα σύνολο αντικειμένων κατά την διάρκεια εκτέλεσης μιας περίπτωσης χρήσης του λογισμικού. Στον κατακόρυφο άξονα απεικονίζονται οι αλληλεπιδράσεις που λαμβάνουν χώρα κατά τη

διάρκεια ζωής των αντικειμένων (χρονικό διάστημα το οποίο έχει δεσμευτεί μνήμη της στοίβας για τα αντικείμενα). Ένα παράδειγμα ακολουθιακού διαγράμματος φαίνεται στο σχήμα Α.1.10.



Σχήμα Α.10. Παράδειγμα σχεδίασης ακολουθιακού διαγράμματος

Α.2 Πρότυπα

Σύμφωνα με τους Gamma, Helm, Johnson & Vlissides (1995), «Τα σχεδιαστικά πρότυπα (patterns) περιγράφουν έναν τρόπο επικοινωνίας μεταξύ των αντικειμένων (objects) και των κλάσεων (classes) τα οποία έχουν προσαρμοστεί ώστε να επιλυθεί ένα γενικό σχεδιαστικό πρόβλημα σε συγκεκριμένο πλαίσιο».

Τα πρότυπα μπορούν να χωριστούν σε κατηγορίες ανάλογα με την λειτουργία τους. Υπάρχουν γενικά τρεις κατηγορίες προτύπων.

1. Δημιουργικά (creational) Πρότυπα
2. Δομικά (structural) Πρότυπα
3. Πρότυπα Συμπεριφοράς (Behavioral)

Κάθε κατηγορία περιέχει και διαφορετικά είδη προτύπων τα οποία θα αναλυθούν εκτενώς παρακάτω.

Α.2.1 Δημιουργικά Πρότυπα

Τα δημιουργικά πρότυπα περιγράφουν τη δημιουργία αντικειμένων και βοηθούν στο να κάνουμε το σύστημα ανεξάρτητο του τρόπου του οποίου τα αντικείμενα δημιουργούνται, συνθέτονται και καταστρέφονται. Τα δημιουργικά πρότυπα αποκτούν ιδιαίτερη σημασία κατά την εξέλιξη του συστήματος όταν οι περισσότερες σχέσεις είναι σχέσης συναρμολόγησης και όχι κληρονομικότητας. Αυτό συμβαίνει γιατί σε μεγάλα συστήματα αυτό που μας ενδιαφέρει περισσότερο είναι να ορίσουμε ένα σύνολο βασικών κλάσεων, οι οποίες μπορούν να συναρμολογηθούν με πολλούς διαφορετικούς τρόπους για τη δημιουργία σύνθετων συμπεριφορών αντί για τον ορισμό κλάσεων με πολύ μεγάλη πολυπλοκότητα και σύνθετη

συμπεριφορά. Στην περίπτωση αυτή η δημιουργία ενός αντικειμένου δεν είναι απλά η αρχικοποίησή του.

Δύο είναι τα βασικά θέματα γύρω από τα οποία περιστρέφονται τα πρότυπα αυτής της κατηγορίας. Το πρώτο είναι, η κελυφοποίηση της γνώσης για το πια κλάση χρησιμοποιείται από το σύστημα και δεύτερον πως δημιουργούνται στιγμιότυπα κλάσεων. Ως αποτέλεσμα τα δημιουργικά πρότυπα μας δίνουν μεγάλη ευελιξία στο τι δημιουργούμε, στο ποιος το δημιουργεί, στο πως δημιουργείται και στο πότε δημιουργείται.

Τα πρότυπα της συγκεκριμένης κατηγορίας είναι:

Όνομα Προτύπου	Έννοια
Factory Method	Ορίζει μια διεπαφή για τη δημιουργία ενός αντικειμένου αλλά αφήνει την αρχικοποίηση στις υποκλάσεις
Abstract Factory	Δημιουργεί μια διεπαφή για τη δημιουργία διάφορων οικογενειών κλάσεων
Builder	Χωρίζει την κατασκευή του αντικειμένου από την αναπαράστασή του
Prototype	Δημιουργεί ένα πλήρως αρχικοποιημένο πρωτότυπο αντικείμενο έτοιμο να αντιγραφεί
Singleton	Μια κλάση της οποίας μπορεί να υπάρχει μόνο ένα στιγμιότυπο

Πίνακας 5. Δημιουργικά Πρότυπα

Α.2.2 Δομικά Πρότυπα

Τα δομικά πρότυπα προσδιορίζουν το πως οι κλάσεις και τα αντικείμενα συναρμολογούνται σε μεγαλύτερες δομές. Τα δομικά πρότυπα που εστιάζονται σε κλάσεις, περιγράφουν το πώς

χρησιμοποιείται η σχέση κληρονομικότητας για να παράγουμε χρήσιμες διεπαφές. Ένα απλό παράδειγμα είναι αυτό της πολλαπλής κληρονομικότητας που δημιουργεί μια καινούργια κλάση βασισμένη σε δύο ή περισσότερες άλλες κλάσεις. Η παραγόμενη κλάση έχει τις ιδιότητες όλων των κλάσεων- πατέρας. Το πρακτικό αποτέλεσμα ενός τέτοιου προτύπου είναι ότι μπορούμε να συνδυάζουμε με εύκολο τρόπο API (Application Programming Interface). Τα δομικά πρότυπα αντικειμένων, περιγράφουν το πως από αντικείμενα συναρμολογούμε μεγαλύτερες δομές.

Τα πρότυπα αυτής της κατηγορίας είναι:

Όνομα Προτύπου	Έννοια
Adapter	Δημιουργία κοινής διεπαφής για διαφορετικές κλάσεις
Bridge	Διαχωρισμός της διεπαφής από την υλοποίηση του αντικειμένου
Composite	Χειρισμός δενδρικών δομών απλών ή σύνθετων αντικειμένων
Decorator	Προσθέτει υπευθυνότητες σε ένα αντικείμενο με δυναμικό τρόπο
Façade	Μια κλάση που αντιπροσωπεύει ένα υποσύστημα
Flyweight	Είναι ένα πρότυπο για να βελτιστοποιήσουμε το μοίρασμα των πόρων σε περιπτώσεις που χρειαζόμαστε μεγάλο αριθμό αντικειμένων
Proxy	Ένα αντικείμενο που αντιπροσωπεύει κάποιο άλλο

Πίνακας 6. Δομικά Πρότυπα

Α.2.3 Πρότυπα Συμπεριφοράς

Τα πρότυπα συμπεριφοράς περιέχουν αντικείμενα που χειρίζονται συγκεκριμένες ενέργειες μέσα στο σύστημα. Τα πρότυπα αυτά κελυφοποιούν διαδικασίες όπως μεταγλώττιση κειμένου, εκτέλεση μια

αίτησης, μετακίνηση (iteration) μέσα σε μια ακολουθία (sequence), υλοποίηση ενός αλγορίθμου.

Τα πρότυπα αυτής της κατηγορίας είναι:

Όνομα Προτύπου	Έννοια
Chain of Responsibility	Πρότυπο για το πέρασμα μιας αίτησης από ένα αντικείμενο σε ένα σύνολο αντικειμένων
Command	Κελυφοποίηση μιας αίτησης εντολής σε ένα αντικείμενο
Interpreter	Η αναπαράσταση μιας γλώσσας σε ένα πρόγραμμα
Iterator	Πρότυπο που διατρέχει και προσπελαύνει μια λίστα αντικειμένων
Mediator	Πρότυπο που απλοποιεί την επικοινωνία μεταξύ αντικειμένων
Memento	Καταγράφει την εσωτερική κατάσταση ενός αντικειμένου και το ανακατασκευάζει
Observer	Πρότυπο που ειδοποιεί ένα σύνολο αντικειμένων για μια αλλαγή
State	Αλλάζει τη συμπεριφορά ενός αντικειμένου όταν αλλάξει η κατάσταση του αντικειμένου. Ουσιαστικά αλλάζει την κλάση του αντικειμένου
Strategy	Ορίζει οικογένειες αλγορίθμων και τις κελυφοποιεί σε κλάσεις
Template Method	Ορίζει το σκελετό ενός αλγορίθμου αφήνοντας τον ορισμό ορισμένων βημάτων στις υποκλάσεις
Visitor	Επιτρέπει τον ορισμό καινούργιων μεθόδων σε μια κλάση χωρίς να αλλάξει την κλάση

Πίνακας 7. Πρότυπα Συμπεριφοράς

Παράρτημα Β: Πηγαίος Κώδικας Plugin

Αυτό το παράρτημα αποτελείται από τους πηγαίους κώδικες του Plugin.

```
{
    "name": "3MF Metadata View",
    "author": "Ultimaker B.V.",
    "version": "1.0.0",
    "description": "Provides support for projecting metadata into screen.",
    "api": 8,
    "i18n-catalog": "cura"
}
```

Πίνακας 8. Κώδικας JSON plugin

```
from typing import Dict
import sys
from Uranium.UM.i18n import i18nCatalog
from Uranium.UM.Logger import Logger
catalog = i18nCatalog("cura")
try:
    from . import GuiReader
except ImportError:
    Logger.log("w", "Could not import ThreeMFReader; libSavitar may be missing")


def getMetaData() -> Dict:
    metaData = {}
    workspace_extension = "3mf"

    if "3MFGuiReader.GuiReader"  in sys.modules:
            metaData["file_metadata_reader"] = [
                    {
                        "extension": ".3mf",
                        "description":    catalog.i18nc("@item:inlistbox",    "3MF File")
                    }
            ]
```

```
            metaData["project_metadata"] = [
                {
                    "extension": workspace_extension,
                    "description": catalog.i18nc("@item:inlistbox", "3MF File")
                }
            ]
    return metadata
def register(app):

    if "3MFGuiReader.GuiReader"  in sys.modules:
        return {"file_metadata_reader": GuiReader.GuiMetadataReader(),
                "project_metadata": GuiReader.GuiMetadataReader()}
    else:
        return  {}
```

Πίνακας 9. Κώδικας (__init__.py) Σύνδεσης Με την Κύρια Εφαρμογή

```
# Copyright (c) 2022 Ultimaker B.V.
# Cura is released under the terms of the LGPLv3 or higher. #\

from Cura.cura import CuraApplication
from Uranium.UM import  Extension
from PyQt6.QtWidgets import *
from Uranium.UM.View import View
from Uranium.UM.Logger import Logger
from Uranium.UM.PluginRegistry import PluginRegistry
import sys
import os
import zipfile
import xml.etree.cElementTree as ET
from PyQt6.QtWidgets import (
    QMainWindow, QApplication, QListWidget
)
from PyQt6.QtGui import QAction, QIcon
from PyQt6.QtCore import Qt
try:
    from . import ThreeMFReader
    from . import ThreeMFWorkspaceReader
except ImportError:
    Logger.log("w",     "Could     not     import     ThreeMFReader     and
ThreeMFWorkspaceReader; libSavitar may be missing")

class MetadataReader(PluginRegistry):

    def __init__(self, _metadata, path_file):
        super().__init__()
        self._metadata = {}
        self._path_file = path_file
        self._file = os.path.basename(path_file)

    def checkFile(self, _file):
        if not _file.endswith(".3mf"):
            return False

    def readMetadata(self) -> dict:
```

```
        archive = zipfile.ZipFile(self._file, "r")
        xml_document = archive.open("3D/3dmodel.model")
        archive.close()

        tree = ET.parse(xml_document)

        root = tree.getroot()
        xmlns                                                                    =
"{http://schemas.microsoft.com/3dmanufacturing/core/2015/02}"

        for data in root.findall(xmlns+'metadata'):

            for k in data.attrib:
             self._metadata[data.attrib[k]] = data.text
        return self._metadata

class GuiMetadataReader(QMainWindow,Extension, View):

    def __init__(self, metadata: dict) -> None:
        super().__init__()
        self.metadata = MetadataReader.readMetadata(metadata)

    def getMetadata(self) -> dict:
        return self.metadata

    def projectMetadata(self) -> list:
        attr_val = []

        for i in self.metadata.keys():
            attr_val.append(i + ": " + self.metadata[i])
        return attr_val

    def clicked(self, qmodelindex):
        item = self.listWidget.currentItem()
        print(item.text())

    def openWindow(self):
        self.setWindowTitle(".3mf File Metadata Reader")
        widget = QListWidget()
```

```
        list = GuiMetadataReader.projectMetadata()
        widget.addItems(list)
        widget.currentItemChanged.connect(self.index_changed)
        widget.currentTextChanged.connect(self.text_changed)
        self.setCentralWidget(widget)
        self.window.show()

    def index_changed(self, i): # Not an index, i is a QListWidgetItem
        print(i.text())

    def text_changed(self, s): # s is a str
        print(s)
```

Πίνακας 10. Κώδικας (GuiReader.py) Επεξεργασίας Και Προβολής Μεταδεδομένων

```
@startuml
'https://plantuml.com/use-case-diagram

left to right direction
skinparam packageStyle rectangle
actor User

rectangle 3MFGuiReader {
    User -- (OpenWindow)
    (OpenWindow) ...> (ReadMetadata): include
    User -- (ReadMetadataFromWindow)
    (ReadMetadataFromWindow) ..> (ReadMetadata): include
    (ProjectMetadata) .> (ReadMetadata): include
}
@enduml
```

Πίνακας 11. Κώδικας Παραγωγής UML Use Case Διαγράμματος

```
@startuml
'https://plantuml.com/class-diagram

static class Logger
class View
class Extension
class ThreeMFReader
class ThreeMFWorkspaceReader
class MetadataReader {
    +dict _metadata
    +string _path_file
    +string _file
    -checkFile(_file: String): void
    +readMetadata(metadata: dict): dict
}
class Application {
    +run()
}
class GuiMetadataReader {

    +metadata: dict
    +getMetadata(metadata: dict): dict
    +projectMetadata(metadata: dict): list
    +clicked(qmodelindex): void
    +openWindow(): void



}
class PluginRegistery {
    +loadPlugin(name: string): Plugin
    +loadPlugins(type: string): list
    +getMetaData(name: string): dict
    +getAllMetaData(type: string): list
}
Logger <.. ThreeMFReader
Logger <.. ThreeMFWorkspaceReader
ThreeMFWorkspaceReader <--- MetadataReader : metadata, 0..1
ThreeMFReader ---> MetadataReader
MetadataReader --|> PluginRegistery
```

```
MetadataReader <-- Logger
MetadataReader <.. GuiMetadataReader: "+metadata" 1
View <|-- GuiMetadataReader : " +metadata "  1..*
Extension <|-- GuiMetadataReader
PluginRegistery --* Application: "+pluginRegistery"  1

@enduml
```

Πίνακας 12. Κώδικας Παραγωγής UML Class Διαγράμματος

```
@startuml
'https://plantuml.com/sequence-diagram

skinparam sequenceArrowThickness 2
skinparam roundcorner 20
skinparam maxmessagesize 60
skinparam sequenceParticipant underline
actor User
participant "View" as A
participant "GuiMetadataReader" as B
participant "MetadataReader" as C
User -> A: Open Window
activate A
A -> B: Create Request
activate B
B -> C: Load Metadata
activate C
C --> B: Metadata Loaded
destroy C
B --> A: Request Created
deactivate B
A --> User: Read all Metadata
deactivate A

@enduml
```

Πίνακας 13. Κώδικας Παραγωγής UML Ακολουθιακού Διαγράμματος Αλληλεπίδρασης Κλάσεων

```
@startuml
'https://plantuml.com/sequence-diagram

actor User
participant View
participant GuiMetadataReader
participant MetadataReader
ref over User, View: initialize window

User -> View: Open Window
View -> View: Internal call
activate View
GuiMetadataReader -> View: openWindow()
View -> GuiMetadataReader: sendPath(): Send request to load metadata
activate GuiMetadataReader

MetadataReader <- GuiMetadataReader: sendFile(): Access .3mf file
activate MetadataReader
loop
    MetadataReader -> MetadataReader: readMetadata(): Read all metadata
from .3mf file
end

MetadataReader --> GuiMetadataReader: getMetadata(): Send all metadata
deactivate MetadataReader

GuiMetadataReader --> View: projectMetadata(): Project metadata in window
deactivate GuiMetadataReader
View --> User: Read metadata from .3mf file
deactivate View

@enduml
```

Πίνακας 14. Κώδικας Παραγωγής UML Ακολουθιακού Διαγράμματος Αλληλεπίδρασης Μεθόδων

```
import sys
import zipfile
import xml.etree.cElementTree as ET



from PyQt6.QtWidgets import (
    QMainWindow, QApplication, QListWidget
)
from PyQt6.QtGui import QAction, QIcon
from PyQt6.QtCore import Qt

class Window(QMainWindow):

    def __init__(self, file) -> None:
        super(Window, self).__init__()
        self.file = file
        self.setWindowTitle(".3mf File Metadata Reader")

        widget = QListWidget()
        list = readMetadataList(file)

        widget.addItems(list)

        widget.currentItemChanged.connect(self.index_changed)
        widget.currentTextChanged.connect(self.text_changed)

        self.setCentralWidget(widget)


    def index_changed(self, i): # Not an index, i is a QListWidgetItem
        print(i.text())

    def text_changed(self, s): # s is a str
        print(s)




def readMetadataList(file):
    archive = zipfile.ZipFile(file, "r")
```

```
    xml_document = archive.open("3D/3dmodel.model")
    archive.close()

    tree = ET.parse(xml_document)

    root = tree.getroot()

    listMetadata = {}
    lst = []
    xmlns = "{http://schemas.microsoft.com/3dmanufacturing/core/2015/02}"

    for data in root.findall(xmlns+'metadata'):

        for k in data.attrib:
            listMetadata[data.attrib[k]] = data.text

    for i in listMetadata.keys():
            lst.append(i + ": " + listMetadata[i])
    return lst



if __name__ == "__main__":
    app = QApplication(sys.argv)
    args = sys.argv

    ThreeMFFile = args[1]
    if ThreeMFFile.endswith(".3mf"):
        w = Window(ThreeMFFile)
        w.show()
        app.exec()
    else:
        print("Error: Wrong type of file!")
        sys.exit()
```

Πίνακας 15. Κώδικας αρχείου 3mf.py

Βιβλιογραφία

Uranium Framework, github: https://github.com/Ultimaker/Uranium

Cura Software, github: https://github.com/Ultimaker/Cura

Sommerville, I. (2011). *Software Engineering, 9th Edition*. New York: Pearson Education Inc.

Leithbridge, C. T. & Laganiere, R. (2005). *Object-Oriented Software Engineering: Practical Software Development using UML and Java*. London: McGraw Hill Education.

Bruegge, B. & Dutoit, H. A., (2010). *Object-Oriented Software Engineering: Using UML, Patterns, and Java, 3rd Edition*. New York: Pearson Education Inc., Prentice Hall.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. New York: Addison-Wesley Professional Computing Series.

Mall, R. (2014). *Fundamentals of Software Engineering, 4th Edition*. Dehli: PHI Learning Private Limited.

Pfleeger, L. S. (2011). *Τεχνολογία Λογισμικού: Θεωρία και Πράξη*. Δεύτερη Αμερικανική Έκδοση, Επιστημονική Επιμέλεια: Σταματέλος Γ. Αθήνα: ΕΚΔΟΣΕΙΣ ΚΛΕΙΔΑΡΙΘΜΟΣ ΕΠΕ.

Χατζηγεωργίου, Α. Ν. (2005). *Αντικειμενοστρεφής Σχεδίαση: UML, Πρότυπα και Ευρετικοί Κανόνες*. Αθήνα: ΕΚΔΟΣΕΙΣ ΚΛΕΙΔΑΡΙΘΜΟΣ ΕΠΕ.

Guttag, J. & Liskov, B. (2007). *Ανάπτυξη Προγραμμάτων σε JAVA: Αφαιρέσεις, Προδιαγραφές και Αντικειμενοστρεφής Σχεδιασμός*. Αθήνα: ΕΚΔΟΣΕΙΣ ΚΛΕΙΔΑΡΙΘΜΟΣ ΕΠΕ.

Μανής, Γ. (2016). *Εισαγωγή στον Προγραμματισμό με αρωγό τη γλώσσα Python.* Αθήνα: Εκδόσεις ΣΕΑΒ.

Guttag, J. V. (2015). *Εισαγωγή στον Υπολογισμό και τον Προγραμματισμό με την Python, 2η Έκδοση*. Αθήνα: Α. ΠΑΠΑΣΩΤΗΡΙΟΥ & ΣΙΑ Ι.Κ.Ε.

Μαγκούτης, Κ. & Νικολάου, Χ. (2015). *Εισαγωγή στον Αντικειμενοστραφή Προγραμματισμό με Python: Μια Προσέγγιση από την Πλευρά των Υπολογιστικών Συστημάτων*. Αθήνα: Εκδόσεις ΣΕΑΒ.

Φιτσιλής, Π. (2004). *Σχεδιασμός Λογισμικού, Τεχνολογία Λογισμικού ΙΙ: Πρότυπα Σχεδίασης*, Εκδόσεις ΕΑΠ.