ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη εργαλείου μετατροπής διαγραμμάτων μεταξύ μορφοτύπων PlantUML και XML για το περιβάλλον μοντελοποίησης Visual Paradigm

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ναταλία Ν. Μπούρδη

Επιβλέπων: Βασίλειος Βεσκούκης
Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2025

Εθνικο Μετσοβιο Πολυτεχνειο
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΥΠΟ-
ΛΟΓΙΣΤΩΝ

Ανάπτυξη εργαλείου μετατροπής διαγραμμάτων μεταξύ μορφοτύπων PlantUML και XML για το περιβάλλον μοντελοποίησης Visual Paradigm

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ναταλία Ν. Μπούρδη

Επιβλέπων: Βασίλειος Βεσκούκης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 28 Φεβρουαρίου 2025.

.....	.....	.....
Βασίλειος Βεσκούκης Καθηγητής Ε.Μ.Π.	Παρασκευοπούλου Ζωή Επίκουρη Καθηγήτρια Ε.Μ.Π.	Παπασπύρου Νικόλαος Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2025.

.....

Ναταλία Ν. Μπούρδη
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η UML αποτελεί θεμελιώδες μέρος του σχεδιασμού και της τεκμηρίωσης συστημάτων λογισμικού, προσφέροντας ένα τρόπο αναπαράστασης της δομής και της λειτουργίας τους διαγραμματικά. Το Visual Paradigm είναι από τα πιο διαδεδομένα εργαλεία για τη δημιουργία και διαχείριση UML διαγραμμάτων, παρέχοντας εκτεταμένες δυνατότητες μοντελοποίησης. Ωστόσο, παρατηρείται αυξανόμενη τάση προς τη χρήση εργαλείων UML που βασίζονται στη δημιουργία διαγραμμάτων με απλό κείμενο, ιδιαίτερα του PlantUML. Για τη τάση αυτή ευθύνεται εν μέρει η εξέλιξη των Μεγάλων Γλωσσικών Μοντέλων (LLMs), τα οποία μπορούν να δημιουργήσουν κώδικα PlantUML, αλλά αντιμετωπίζουν δυσκολίες με πιο περίπλοκες μορφές, όπως το XMI, που αποτελεί το XML πρότυπο παράστασης και μέρος της προδιαγραφής της UML από το OMG.

Η παρούσα εργασία παρουσιάζει την ανάπτυξη μίας επέκτασης (plugin) για το Visual Paradigm, το οποίο επιτρέπει την εισαγωγή και εξαγωγή UML διαγραμμάτων από και προς το μορφότυπο κειμένου PlantUML, καθώς και ενός εργαλείου γραμμής εντολών (CLI) προς διευκόλυνση αυτοματοποιήσεων. Δόθηκε έμφαση στη διατήρηση του σημασιολογικού περιεχομένου του Visual Paradigm, ώστε η μετατροπή των διαγραμμάτων να αποδίδει όσο το δυνατόν πιο πιστά την αρχική τους μορφή και την έννοια του ολοκληρωμένου UML μοντέλου που διέπει τέτοια εργαλεία. Επίσης, έγιναν σχεδιαστικές επιλογές με γνώμονα τόσο τη βελτιστοποίηση της εισαγωγής διαγραμμάτων που δημιουργήθηκαν από LLMs όσο και της εξαγωγής τους σε μορφή που προορίζεται για επεξεργασία από αυτά.

Λέξεις Κλειδιά

Visual Paradigm, PlantUML, Διαγράμματα UML, XML, XMI, Διαλειτουργικότητα UML, Επέκταση Visual Paradigm

Abstract

UML is a fundamental part of software system design and documentation, offering a way to represent their structure and functionality diagrammatically. Visual Paradigm is one of the most widely used tools for creating and managing UML diagrams, providing extensive modeling capabilities. However, there is a growing trend toward using UML tools that generate diagrams from plain text, particularly PlantUML. This trend is partly driven by the evolution of Large Language Models (LLMs), which can generate PlantUML code but struggle with more complex formats such as XMI, the XML format specified by OMG for UML representation.

This thesis presents the development of a plugin for Visual Paradigm that enables the import and export of UML diagrams to and from the PlantUML text format, as well as a command-line tool (CLI) to facilitate automation. Emphasis was placed on preserving the semantic content of Visual Paradigm, ensuring that the diagram transformation represents the original form as accurately as possible. Additionally, design choices were made with a focus on optimizing both the import of diagrams generated by LLMs and their export in a format intended for further processing by such models.

Keywords

Visual Paradigm, PlantUML, UML Diagrams, XML, XMI, UML Interoperability, Visual Paradigm Plugin

Ευχαριστίες

Θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον επιβλέποντα καθηγητή μου, κ. Βασίλη Βεσκούκη, για την καθοδήγηση και το ενδιαφέρον του κατά την εκπόνηση της παρούσας εργασίας. Ευχαριστώ επίσης τον συμφοιτητή Γ. Σωτηρόπουλο, που μοιράστηκε μέρος της εργασίας του, συμβάλλοντας στην παρούσα διπλωματική.

Ευχαριστώ την οικογένειά μου, που με στήριξε αδιάκοπα αυτά τα χρόνια, καθώς και τους φίλους μου και τον Μάρκο για την πολύτιμη συντροφιά καθ΄ όλη τη διάρκεια των σπουδών μου.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

1.1 Έννοιες και εργαλεία

1.1.1 UML

Η Unified Modeling Language (UML) είναι μία πρότυπη γλώσσα μοντελοποίησης για τη δημιουργία σχεδιαγραμμάτων συστήματων λογισμικού. Χρησιμοποιείται ευρέως για την απεικόνιση/οπτικοποίηση, τον καθορισμό, την κατασκευή και την τεκμηρίωση των συστατικών ενός συστήματος που βασίζεται σε λογισμικό [6]. Προσφέρει έναν ενιαίο και τυποποιημένο τρόπο περιγραφής συστημάτων λογισμικού μέσω διαγραμμάτων, καλύπτοντας όλα τα επίπεδα λεπτομέρειας και αφαιρετικότητας.

Ακόμη και κατά τον πρόχειρο σχεδιασμό λογισμικού στο χαρτί, η UML έχει ασκήσει σημαντική επιρροή. Οι αρχές και οι συμβάσεις της έχουν διαμορφώσει τον τρόπο που οι σχεδιαστές σκέφτονται και επικοινωνούν τα συστήματα λογισμικού.

Η UML ορίζει και περιγράφει 9 τύπους διαγραμμάτων, αν και, θεωρητικά αυτοί μπορούν να αναμειχθούν. Κάθε ένας από τους τύπους εξυπηρετεί στην οπτικοποίηση και επικοινωνία μιας διαφορετικής όψης του συστήματος λογισμικού [1]. Οι τύποι διαγραμμάτων ανήκουν σε δύο γενικότερες κατηγορίες: διαγράμματα δομής αν περιγράφουν στατική όψη του συστήματος ή διαγράμματα συμπεριφοράς αν περιγράφουν δυναμική όψη και αλληλεπίδραση στοιχείων.

Τύπος διαγράμματος	**Περιγραφή**
Διαγράμματα Δομής	
Class Diagram	Αντιπροσωπεύει τη στατική δομή ενός συστήματος, παρουσιάζοντας κλάσεις, χαρακτηριστικά, λειτουργίες και σχέσεις.

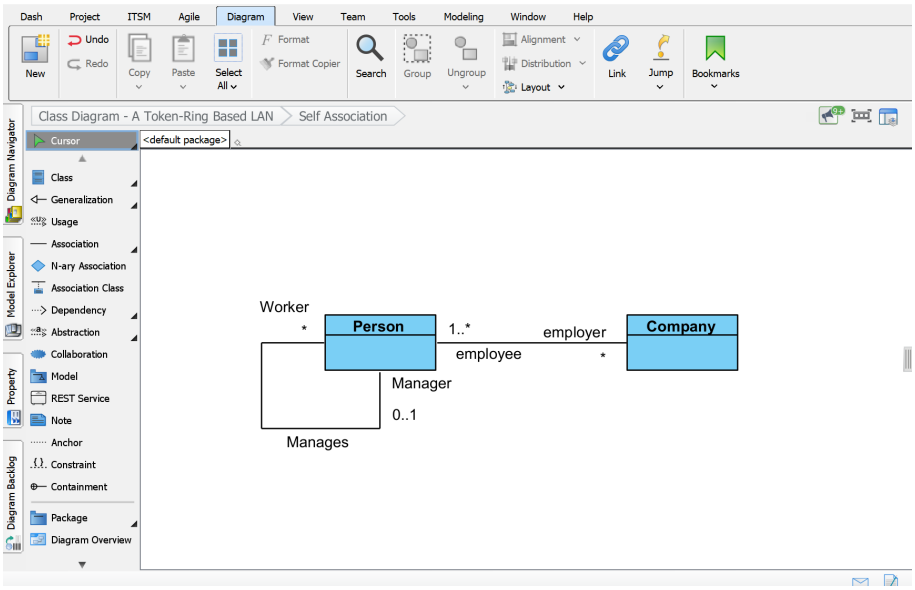
Object Diagram	Απεικονίζει στιγμιότυπα κλάσεων και τις σχέσεις τους, δηλαδή της κατάστασης του συστήματος σε μια συγκεκριμένη χρονική στιγμή.
Component Diagram	Δείχνει τα φυσικά συστατικά ενός συστήματος, όπως βιβλιοθήκες, εκτελέσιμα αρχεία και εξαρτήσεις.
Deployment Diagram	Απεικονίζει τη φυσική ανάπτυξη αντικειμένων (artifacts) σε κόμβους, παρουσιάζοντας την τοπολογία υλικού του συστήματος.
Διαγράμματα συμπεριφοράς	
Use Case Diagram	Περιγράφει τις λειτουργικές απαιτήσεις ενός συστήματος, συμπεριλαμβανομένων των χρηστών (actors) και των περιπτώσεων χρήσης που αλληλεπιδρούν μαζί τους.
Sequence Diagram	Δείχνει τη διαδοχή των αλληλεπιδράσεων μεταξύ αντικειμένων με την πάροδο του χρόνου για την επίτευξη ενός συγκεκριμένου στόχου.
Collaboration Diagram	Εστιάζει στην οργανωτική δομή των αντικειμένων και στις αλληλεπιδράσεις τους για την ολοκλήρωση ενός συγκεκριμένου έργου.
State (Machine) Diagram	Μοντελοποιεί τη δυναμική συμπεριφορά ενός συστήματος, απεικονίζοντας καταστάσεις και μεταβάσεις που προκαλούνται από γεγονότα.
Activity Diagram	Παρουσιάζει τη ροή ελέγχου ή δεδομένων μέσα σε ένα σύστημα, αναπαριστώντας ροές εργασιών ή επιχειρηματικές διαδικασίες.

Πίνακας 1.1: Τύποι Διαγραμμάτων UML και Περιγραφές

Φυσικά, η UML αποτελεί μόνο την γλώσσα μοντελοποίησης των διαγραμμάτων, και δεν υπάρχει επίσημο ή μοναδικό εργαλείο για την κατασκευή τους. Προς αυτό το σκοπό, έχουν αναπτυχθεί πολλά εργαλεία λογισμικού με διαφορετικές φιλοσοφίες και προσεγγίσεις.

1.1.2 Visual Paradigm

Το Visual Paradigm είναι ένα από αυτά τα εργαλεία που διατίθενται στην αγορά για τη δημιουργία και διαχείριση διαγραμμάτων UML (Unified Modeling Language). Ανήκει στη κατηγορία των γραφικών εργαλείων (GUI-based) με αντίστοιχους ανταγωνιστές το Visio της Microsoft, το ArgoUML, Sparx, Modelio, StarUML, Eclipse Papyrus, Enterprise Architect. Είναι εμπορικό προϊόν που αποτελεί εκτενή πλατφόρμα, υπερβαίνοντας κατά πολύ την απλή δημιουργία διαγραμμάτων. Πρόκειται για ολοκληρωμένη σουίτα λειτουργιών για τη μοντελοποίηση, τον σχεδιασμό και τη διαχείριση σύνθετων συστημάτων.



Σχήμα 1.1: Το γραφικό περιβάλλον του Visual Paradigm - ένα class diagram

Κάθε αρχείο/project του Visual Paradigm μπορεί να περιέχει αόριστο αριθμό μοντέλων και διαγραμμάτων, εμπλουτισμένα με μεταδεδομένα και συσχετίσεις μεταξύ μοντέλων που ανήκουν σε διαφορετικά διαγράμματα (semantics). Τέτοια πρόσθετα σημασιολογικά στοιχεία είναι αναφορές σε άλλα διαγράμματα, σε αρχεία, ιστοσελίδες ή συγκεκριμένα μοντέλα του έργου, υποδιαγράμματα σε μοντέλα, περιγραφές μοντέλων και άλλα.

Δεν υποστηρίζει μόνο τη δημιουργία διαγραμμάτων UML, αλλά ενσωματώνεται με διάφορα στάδια του κύκλου ζωής ανάπτυξης λογισμικού. Ενδεικτικά, επιτρέπει τη δημιουργία Wireframes για σχεδιασμό UI, υποστηρίζει διαγράμματα Business Process Model and Notation (BPMN), Entity-Relationship Diagram (ERD) κ.α. που παρέχουν σημαντική αξία ως επέκταση του συνόλου τεκμηρίων για τη περιγραφή ενός συστήματος. Ως αποτέλεσμα, το καθιστούν ιδιαίτερα ελκυστικό για τη μοντελοποίηση μεγάλων συστημάτων σε επίπεδο επιχείρησης.

Ιδιαίτερα πλεονεκτήματα του Visual Paradigm σε σχέση με άλλα εργαλεία αποτελεί η αυτόματη παραγωγή κώδικα, η αντίστροφη μηχανική κώδικα (reverse engineering) και η ενσωμάτωση με εργαλεία ομαδικής συνεργασίας και version control, διευκολύνοντας τις ροές εργασίας και φέρνοντας τον σχεδιασμό πιο κοντά στην υλοποίηση.

Στην desktop έκδοσή του, το Visual Paradigm επιτρέπει και την εγκατάσταση επεκτασεων (plugin) για την ανάπτυξη των οποίων προσφέρει μία προγραμματιστική διεπαφή. Η παρούσα εργασία βασίστηκε στην εκμετάλλευση αυτής της διεπαφής.

1.1.3 PlantUML

Στη κατηγορία των μη γραφικών λύσεων για τη δημιουργία διαγραμμάτων κατατάσσεται και το open-source εργαλείο PlantUML, όπως και άλλες πλατφόρμες σαν το MermaidJS [18] και το Δ2 [17]. Αντί να σχεδιάζονται σε γραφικό περιβάλλον, τα διαγράμματα ορίζονται μέσω συνοπτικών αρχείων κειμένου. Τέτοιες λύσεις γνωρίζουν ολοένα και μεγαλύτερη αποδοχή, συμβάλλοντας στην ευρύτερη διάδοση της UML ως πρότυπο και ενισχύοντας τη χρήση της γενικότερα [15].

Η PlantUML στοχεύει στην εύκολη, συνεκτική δήλωση των στοιχείων των διαγραμμάτων και των σχέσεων μεταξύ τους για τη διαισθητική συγγραφή των κειμένων περιγραφής τους.[9]. Η διαισθητική της σύνταξη αποτελείται κυρίως από διαδοχικές εντολές μίας γραμμής ή μερικών γραμμών, που δηλώνουν συνήθως ένα αντικείμενο ή μια σχέση μεταξύ των αντικειμένων.

Ένα σύντομο παράδειγμα διαγράμματος κλάσεων που μαρτυρά την απλότητα της σύνταξης της PlantUML παρατίθεται παρακάτω, το αποτέλεσμα του οποίου φαίνεται στο Σχήμα 1.1

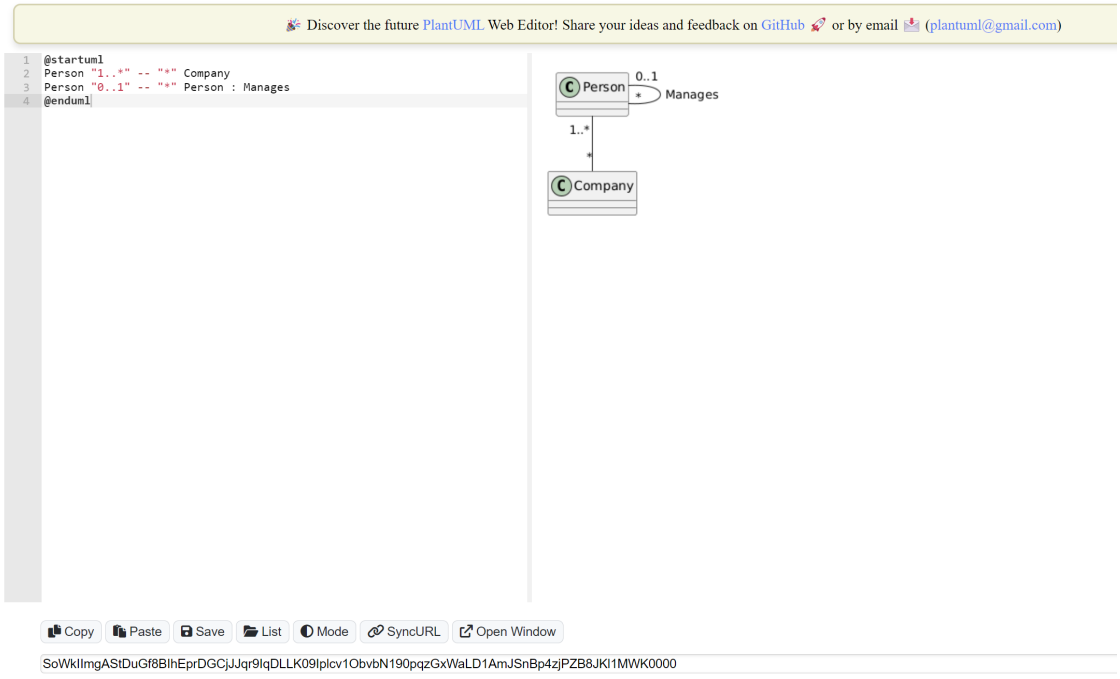
Listing 1.1: PlantUML class diagram

```
@startuml
Person "1..*" -- "*" Company
Person "0..1" -- "*" Person : Manages
@enduml
```

Διατίθεται ως web εφαρμογή προς εγκατάσταση σε υπολογιστή τοπικά του χρήστη ή ως desktop εφαρμογή, ενώ υπάρχει και PlantUML Server στο διαδίκτυο, ο οποίος είναι σαφώς λίγο αργότερος και δεν προτείνεται για εκτεταμένη χρήση. [19]

Η έξοδος του συστήματος του PlantUML είναι στατική εικόνα-διάγραμμα των αντικειμένων και σχέσεων σε μορφή PNG, SVG, LaTeX, EPS (Encapsulated PostScript) για χρήση με LaTeX ή ASCII art (αποκλειστικά για sequence diagrams). Αυτό αποτελεί μεγάλη αντίθεση σε σχέση με το Visual Paradigm και άλλα πιο ολοκληρωμένα συστήματα, των οποίων η έξοδος είναι ένα πολύπλοκο αρχείο-μοντέλο (*.vpp για το Visual Paradigm) όλου του συστήματος. Να σημειωθεί επίσης ότι δεν είναι δυνατή ούτε η δημιουργία πολλαπλών διαγραμμάτων από ένα αρχείο. Κάθε αρχείο μπορεί να ορίσει ένα μοναδικό διάγραμμα και δεν υπάρχουν μεταδεδομένα, ό,τι βλέπει κανείς στο παραγόμενο διάγραμμα είναι όλη η πληροφορία που περιέχει.

Οι υποστηριζόμενοι τύποι και για το PlantUML δεν περιορίζονται μόνο στη UML, αλλά και σε κάποια ακόμη όπως Timing Diagram, οπτικοποίηση JSON και YAML δεδομένων, Wireframes, Gantt και μερικά ακόμη.



Σχήμα 1.2: Το web περιβάλλον του PlantUML - ένα class diagram

1.1.4 MermaidJS

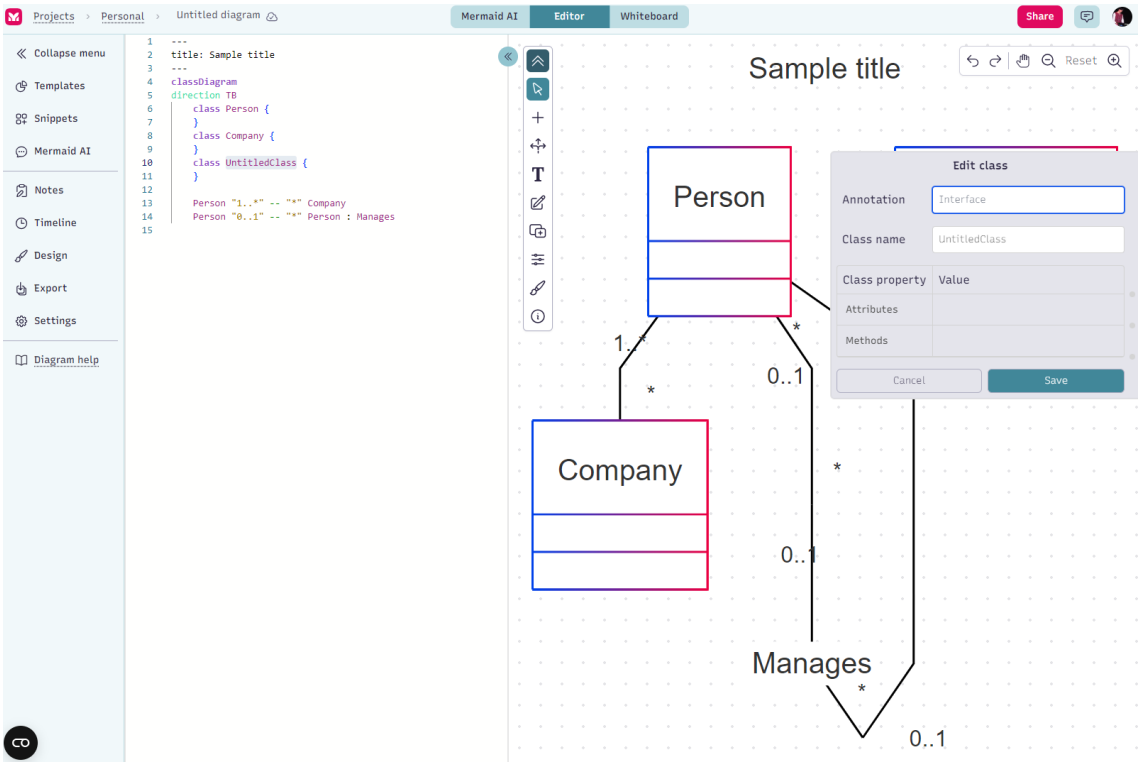
Για την εύρεση της καταλληλότερης text-based πλατφόρμας UML για την περίπτωση χρήσης μας εξετάσαμε διάφορες εναλλακτικές του PlantUML. Το Mermaid (ή mermaidjs) είναι ένα ακόμη εργαλείο διαγραμμάτων που χειρίζεται είσοδο σε μορφή απλού κειμένου, βασισμένο στη Javascript. Υποστήρίζει μόνο τα πιο συνήθη διαγράμματα UML, συγκεκριμένα τα Class, State και Sequence Diagrams, καθώς και αρκετά περισσότερα που δεν ανήκουν στο σύνολό της UML [18]. Η συνταξή των διαγραμμάτων στο Mermaid παρουσιάζει κοινά χαρακτηριστικά με τη PlantUML. Για παράδειγμα, το Person-Company class diagram που παρουσιάσαμε παραπάνω έχει ακριβώς την ίδια σύνταξη στην MermaidJS όπως και στην PlantUML.

Listing 1.2: MermaidJS class diagram

```
classDiagram
    Person "1..*" -- "*" Company
    Person "0..1" -- "*" Person : Manages
```

Παρόμοια με το PlantUML, υπάρχει επιλογή τοπικής εγκατάστασης ή χρήσης του διαδικτυακού τους εργαλείου για απεικόνιση μόνο ενός διαγράμματος.

Το εργαλείο Mermaid ξεχωρίζει από τα υπόλοιπα εργαλεία κειμένου λόγω της διαδραστικής του λειτουργίας. Επιχειρεί με αυτό τον τρόπο να συνδυάσει τα πλεονεκτήματα των γραφικών πλατφόρμων με των κειμενικών. Αν και όχι πλήρως διαδραστικό, προσφέρει τις βασικές λειτουργίες προσθήκης κάποιου διαγραμματικού στοιχείου, προσαρμογής του layout και της παρουσίασης (στυλ, χρώματα), εισαγωγής σημειώσεων και κειμένου μεταξύ άλλων. Αυτόματα ενημερώνει και το αρχείο κειμένου καθώς η οπτικοποίηση επεξεργάζεται από τον χρήστη.



Σχήμα 1.3: Η web εφαρμογή του Mermaid

1.2 Διαλειτουργικότητα συστημάτων μοντελοποίησης UML

Η μετατροπή μεταξύ συστημάτων δημιουργίας διαγραμμάτων UML, όπως το Visual Paradigm Και το PlantUML, είναι ιδιαίτερα επιθυμητή στον τομέα της ανάπτυξης λογισμικού. Διαφορετικά εργαλεία προσφέρουν μοναδικά πλεονεκτήματα, αλλά η έλλειψη δυνατότητας ομαλής μετατροπής μεταξύ τους ¨κλειδώνει' τους χρήστες σε συστήματα που δε τους εξυπηρετούν πλήρως.

Μεγάλα project δε μπορούν να αντιγραφούν από ένα σύστημα σε άλλο χειροκίνητα. Ως αποτέλεσμα, χωρίς επαρκείς επιλογές εξαγωγής και μετατροπής των μοντέλων, ο χρήστης αναγκάζεται να δεσμευτεί σε μία μόνο πλατφόρμα την οποία πρέπει να εμπιστευτεί για τη καλή λειτουργία και συντήρηση του λογισμικού της.

Ευτυχώς, παρά τις άστοχίες και ελλείψεις στις υλοποιήσεις, μετατροπές μεταξύ αρκετών μορφών υπάρχουν και έχουν γίνει προσπάθειες για τη τυποποιημένη ανταλλαγή μοντέλων.

1.2.1 XMI

Η XML Metadata Interchange (XMI) είναι ένα πρότυπο βασισμένο στήν XML που προτάθηκε ως απάντηση σε αίτημα του Object Management Group (OMG), του ίδιου οργανισμού που όρισε την UML, για τη δημιουργία μιας μορφής ανταλλαγής μοντέλων που βασίζεται σε ροές δεδομένων (stream-based) [5].

Ο κύριος σκοπός και η χρήση του XMI είναι στη πράξη να διευκολύνει την ανταλλαγή δεδομένων και μεταδεδομένων μεταξύ εργαλείων μοντελοποίησης UML. Η τρέχουσα έκδοση του XMI είναι η 2.5, ωστόσο πολλά εργαλεία UML εξακολουθούν να χρησιμοποιούν παλαιότερες εκδόσεις, με αρκετά από αυτά να διατηρούν υποστήριξη μόνο για τη σειρά 1.x.

Για τη δημιουργία του αρχείου XMI, διατρέχονται τα μοντέλα και συγγράφεται με βάση το σχήμα (schema) του XMI. Ομοίως, για την εισαγωγή του σε ένα εργαλείο, το XMI αρχείο αναλύεται συντακτικά και αποσπώνται τα περιγραφόμενα μοντέλα προς δημιουργία [5]. Σημειώνεται ότι πληροφορία σχετικά με την παρουσίαση και τη διάταξη των διαγραμμάτων δεν υποστηριζόταν στην έκδοση 1.x. Βέβαια, τα εργαλεία μπορούν να επεκτείνουν την XMI υλοποίηση τους με XML tags που αποσκοπούν στην διατήρηση αυτής της πληροφορίας (π.χ. "diagramming.diagram" tag).

Όμως, στην πράξη, το XMI έχει αποτύχει σε μεγάλο βαθμό με ιδιαίτερα περιορισμένη χρήση του προτύπου, παρά την ύπαρξη των λειτουργιών μετατροπής. Το κύριο πρόβλημα εντοπίζεται στις ασυνεπείες υλοποιήσεις του προτύπου XMI από διαφορετικά εργαλεία UML, οδηγώντας σε ασυμβατότητες και απώλεια δεδομένων κατά την ανταλλαγή μοντέλων[3] [13]. Επιπλέον, μελέτες αποκαλύπτουν ότι περίπου το μισό των ευρέως χρησιμοποιούμενων εργαλείων UML δεν διαθέτουν την απαραίτητη υποστήριξη για εξαγωγή/εισαγωγή XMI [7], ενώ τα εργαλεία που το υποστηρίζουν συχνά υλοποιούν το πρότυπο με ειδικές επεκτάσεις ή ερμηνεύουν τις προδιαγραφές με διαφορετικό τρόπο, καταργώντας την υποσχόμενη καθολικότητα του XMI και, ως εκ τούτου, ενώ το XMI προσφέρει θεωρητικά δυνατότητες για διαλειτουργικότητα, η πρακτική εφαρμογή είναι περιορισμένη.

Οι Persson et al. (2005) διερεύνησαν τη διαλειτουργικότητα του XMI μεταξύ εννέα εργαλείων UML και διαπίστωσαν ότι η ανταλλαγή μοντέλων ήταν σε μεγάλο βαθμό ανεπιτυχής. Κανένας συνδυασμός εργαλείων δεν υποστήριξε πλήρη αμφίδρομη μεταφορά, ενώ στις περισσότερες περιπτώσεις σημειώθηκαν απώλειες σχέσεων, συσχετίσεων και πληθικοτήτων. Σε αρκετές δοκιμές, δεν μεταφέρθηκε καμία πληροφορία, καταδεικνύοντας την αποτυχία του XMI να λειτουργήσει ως αξιόπιστο πρότυπο διαλειτουργικότητας. Πρόβλημα αποτέλεσε και η υιοθέτηση διαφορετικών εκδόσεων του XMI ανάμεσα σε εργαλεία [2].

Visual Paradigm

Το Visual Paradigm θεωρητικά υποστηρίζει πλήρως την εισαγωγή/εξαγωγή XMI αρχείων. Στο [4] εξετάστηκε η διαλειτουργικότητα του Visual Paradigm με δύο ακόμη εργαλείων UML. Δημιουργήθηκαν διαγράμματα κλάσεων σε Visual Paradigm, MagicDraw και Ar-

goUML, όπου τα δύο πρώτα εργαλεία χρησιμοποιούσαν XMI 2.1 (η τελευταία έκδοση που μέχρι σήμερα υποστηρίζεται από το Visual Paradigm), ενώ το ArgoUML βασιζόταν στην παλαιότερη έκδοση 1.1. Η σύγκριση των παραγόμενων XMI αρχείων αποκάλυψε ότι τα εργαλεία χρησιμοποιούν διαφορετικά XMI tags για την αναπαράσταση των διαγραμμάτων, επιβεβαιώνοντας έτσι την έλλειψη συνέπειας στην υλοποίηση του προτύπου και αναγκάζοντας απώλειες πληροφορίας.

Η διπλωματική εργασία του Αγερίδη Γ. [13] ανέλυσε τη διαλειτουργικότητα μεταξύ εργαλείων UML μέσω XMI πειραματικά, με το Visual Paradigm να επιτυγχάνει αρκετές επιτυχημένες μεταφορές κυρίως στην εξαγωγή, αν και με απώλειες σε περιγραφές στοιχείων και σημασιολογικές πληροφορίες. Ωστόσο, στην εισαγωγή, οι ακμές των διαγραμμάτων συχνά χάνονταν ή απλοποιούνταν σημαντικά, επηρεάζοντας την οπτική αναπαράσταση. Συνολικά, η συμβατότητα διέφερε μεταξύ εργαλείων, με το Eclipse Papyrus να έχει την υψηλότερη επιτυχία, ενώ άλλες μεταφορές, όπως από το Enterprise Architect ή το StarUML, παρουσίασαν περιορισμούς.

PlantUML

Η υποστήριξη του XMI από το PlantUML είναι ιδιαίτερα περιορισμένη. Βρίσκεται ακόμη σε beta έκδοση και δεν παρατηρείται ενεργή προσπάθεια για την ολοκλήρωση, με υλοποίηση μόνο για τα διαγράμματα κλάσεων. Απόδειξη της αποτυχίας υιοθέτησης του προτύπου είναι η ύπαρξη δύο διαφορετικών λειτουργιών εξαγωγής αναπτυγμένο από το PlantUML, μία που προορίζεται για εισαγωγή στο εργαλείο StarUML και μία ακόμη για ArgoUML (plantuml.com/xmi).

1.2.2 Υπάρχουσες μετατροπές Visual Paradigm, PlantUML

Ειδικές λύσεις, λοιπόν, χρησιμοποιούνται για τη μετατροπή (εξαγωγή και εισαγωγή) διάφορετικών πλατφόρμων που εστιάζουν και αντιμετωπίζουν τις αναπόφευκτες ασυμβατότητες μεταξύ τους. Βέβαια, αυτό σημαίνει ότι τέτοιες λύσεις προσφέρονται βάσει ζήτησης και δημοφιλίας εργαλείων.

Visual Paradigm

Το Visual Paradigm παρέχει υποστήριξη για την εισαγωγή έργων από πολλές δημοφιλείς εφαρμογές ανταγωνιστών. Τη στιγμή της συγγραφής, υποστηρίζονται αρχεία από Rose, Visio, Enterprise Architect, NetBeans UML, Rational Rhapsody & Architect, Visual UML, ERWin, Bizagi, PowerDesigner DataArchitect, Excel, TextBased Sequence Diagram.

Ακόμη και σε αυτές τις υλοποιήσεις, ωστόσο, γίνονται προφανείς οι ασυμβατότητες μεταξύ των εργαλείων. Ιδιαίτερα στην εισαγωγή από άλλους μορφότυπους, όπου οι εισαγωγές γίνονται σε επίπεδο μοντέλων και όχι διαγραμματικών στοιχείων, μέρος της πληροφορίας των αρχικων διαγραμμάτων (π.χ. οι ακμές) γίνεται κρυφό ενώ αποθηκεύεται στη δομή μοντέλου και χρειάζεται εργασία από το χρήστη για την ανάκτηση της στην απεικόνιση [13].

Όσον αφορά την εξαγωγή (export), η συλλογή επιλογων είναι πολύ φτώχότερη με υποστήριξη μόνο εξαγωγής σε Microsoft Excel, πέρα από τα ήδη αναφερόμενα XMI/XML.

PlantUML

Επίσημες λύσεις μετατροπής σε άλλα έργαλεία δεν υπάρχουν από το PlantUML, πέρα από την εγκαταλλειμένη προσπάθεια για XMI support.

Εκμεταλλευόμενοι τις οπτικές μορφές εξαγωγής (PNG, SVG) κάποιες εισαγωγές σε τρίτα εργαλεία, όπως το γενικής χρήσης διαγραμματικό εργαλείο Draw.io, είναι δυνατές. Ωστόσο, αυτές αφορούν μόνο την απεικόνιση των μοντέλων, χωρίς semantics, πράγμα εκτός πεδίου ενδιαφέροντος για την παρούσα εργασία.

1.2.3 Αμφίδρομή Μετατροπή Visual Paradigm και PlantUML

Ούτε άμεσος ούτε έμμεσος τρόπος μετατροπής (π.χ. μέσω XMI ή διαδοχικών μετατροπών με τη χρήση ενός κοινού, υποστηριζόμενου εργαλείου) έχει αναπτυχθεί για τη σύνδεση των δύο δημοφιλών πλατφορμών. Αυτό οφείλεται κυρίως στο ότι τα δύο εργαλεία διαφέρουν ριζικά, τόσο στη φιλοσοφία τους όσο και στις περιπτώσεις χρήσης τους. Η πολυπλοκότητα του Visual Paradigm, με την πλούσια σημασιολογία των μοντέλων του, χάνεται εντελώς όταν μετατρέπεται στις απλοϊκές, μονοδιάστατες αναπαραστάσεις του PlantUML.

Παρά τις δυσκολίες αυτές, η ανάπτυξη ενός τέτοιου εργαλείου είναι επιθυμητή, κυρίως λόγω της αυξανόμενης ανάγκης για αυτοματοποίηση δημιουργίας UML διαγραμμάτων. Το PlantUML χάρη στη χρήση απλού κειμένου για την περιγραφή διαγραμμάτων, έχει καταστεί ιδιαίτερα αποτελεσματικό εργαλείο για την αυτοματοποίηση ειδικά με τη βοήθεια LLMs, όπως θα αποδείξουμε στο κεφάλαιο 2.1.

Δυστυχώς, το PlantUML, από κατασκευής του προσφέρεται για γρήγορη και πρόχειρη μοντελοποίηση μικρών συστημάτων, και παραμένει ανεπαρκές για την υποστήριξη μεγάλων ή σύνθετων συστημάτων. Το περιβάλλον του Visual Paradigm είναι σαφώς περισσότερο ολοκληρωμένο, καθιστώντας το απαραίτητο για τέτοιες περιπτώσεις. Ένα εργαλείο που γεφυρώνει το PlantUML με το Visual Paradigm μπορεί να επιτύχει κάτι εξαιρετικά ισχυρό: την αυτοματοποίηση της δημιουργίας πλούσιων αρχείων Visual Paradigm, ξεπερνώντας κατά πολύ τις δυνατότητες της του PlantUML.

1.3 Προτεινόμενο εργαλείο

Σκοπός της εργασίας αυτής είναι η κάλυψη αυτού ακριβώς του κενού μέσω της **ανάπτυξης ενός εργαλείου που επιτρέπει τη μετατροπή UML μοντέλων μεταξύ των δύο εργαλείων**. Ιδιαίτερη έμφαση δόθηκε στη διατήρηση της πολυπλοκότητας και των semantics (σημασιολογία) των ολοκληρωμένων μοντέλων που υποστηρίζει το Visual Paradigm, ενώ το εργαλείο σχεδιάστηκε με γνώμονα τη περίπτωση χρήσης όπου αυτόματα

παραγόμενος κώδικας PlantUML δημιουργείται με τη βοήθεια τεχνητής νοημοσύνης και στη συνέχεια εισάγεται στο Visual Paradigm για να αποτελέσει τμήμα σε ένα ολοκληρωμένο μοντέλο UML.

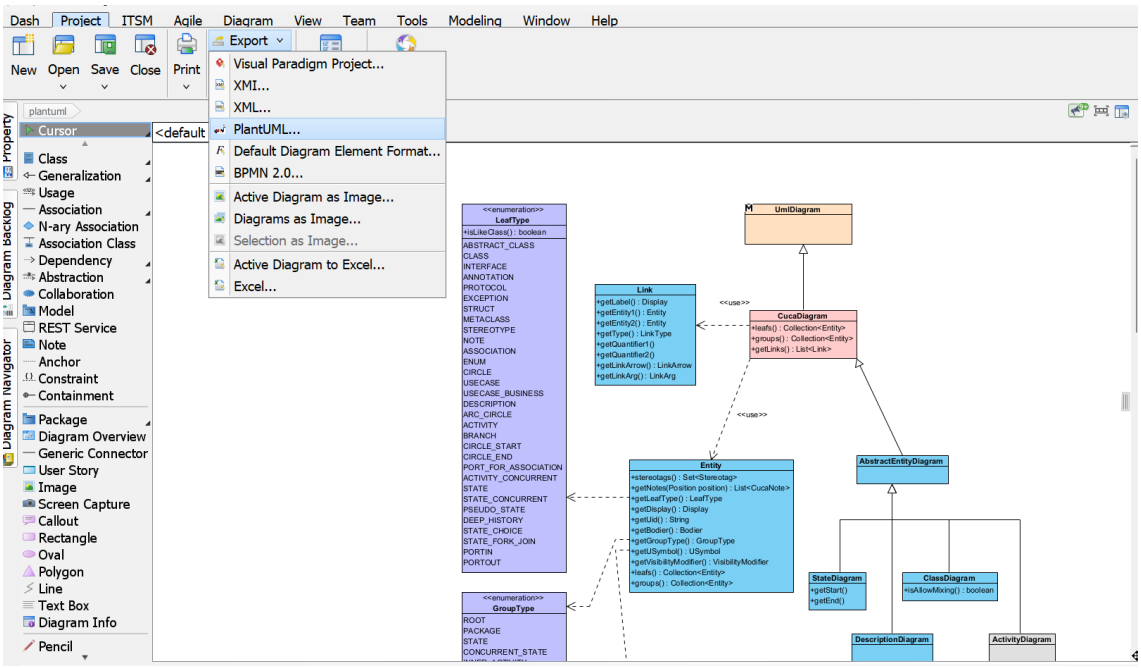
Plugin: Visual Paradigm από/προς PlantUML Η επιλεγμένη μορφή του εργαλείου είναι η ανάπτυξη ενός Plugin, μιας προσθήκης που μπορεί να εγκατασταθεί στο Visual Paradigm. Το plugin, μετά την εγκατάστασή του, επεκτείνει τις δυνατότητες εισαγωγής και εξαγωγής project από και προς PlantUML, προσφέροντας μια διαισθητική εμπειρία μέσα από το γραφικό περιβάλλον του Visual Paradigm.

Μία δικτυακή εφαρμογή[1] απεικόνισης PlantUML με χρήση διατιθέμενου ανοικτού κώδικα και ένα εργαλείο γραμμής εντολών που βασίζονται πάνω στο plugin αναπτύχθηκαν συμπληρωματικά.

Από τα εννέα διαγράμματα UML, όπως αναφέρθηκαν στον Πίνακα 1.1 υποστηρίζονται προς μετατροπή τα εξής (εισαγωγή και εξαγωγή):

- Class Diagram
- Use Case Diagram
- State (Machine) Diagram
- Activity Diagram (μόνο από και πρός τη νέα σύνταξη της PlantUML)
- Sequence Diagram
- Component Diagram
- Deployment Diagram

[1] https://davinci.softlab.ntua.gr/plantuml-viewer



Σχήμα 1.4: Επιλογή μενού "PlantUML..." της λειτουργίας εξαγωγής διαγραμμάτων στο Visual Paradigm

Κεφάλαιο 2

Αυτόματη παραγωγή UML από LLM

Σε αυτό το κεφάλαιο θα τεκμηριώσουμε την επιλογή του PlantUML ως στόχου μετατροπής από και προς το Visual Paradigm σχετικά με την αξιοποίηση του στην εφαρμογή της αυτόματης παραγωγής UML διαγραμμάτων από μεγάλα γλωσσικά μοντέλα (LLMs).

Θα παρουσιάσουμε επιπλέον δύο προτεινόμενες περιπτώσεις χρήσης και ροές αξιοποίησης της επέκτασης (plugin) του Visual Paradigm που αναπτύχθηκε. Με αυτές μπορούν παρακαμφθούν προκλήσεις που σχετίζονται με την ενσωμάτωση και χρήση της τεχνητής νοημοσύνης στο περίπλοκο περιβάλλον του Visual Paradigm, ενσωματώνοντας στο ενδιάμεσο το PlantUML συνδυάζοντας τελικά τα πλεονεκτήματά τους.

2.1 Πειραματισμός με LLM

Η βιβλιογραφία σχετικά με την αυτόματη παραγωγή UML διαγραμμάτων μέσω γλωσσικών μοντέλων (LLMs) παραμένει περιορισμένη, καθώς πρόκειται για νέο ερευνητικό πεδίο. Ωστόσο, διάφορες μελέτες έχουν αρχίσει να εξερευνούν το ζήτημα, αντανακλώντας τη γενικότερη τάση ενσωμάτωσης των LLMs σε διάφορες πτυχές της ανάπτυξης λογισμικού, πέρα από τη δημιουργία κώδικα. Υπάρχει αισιοδοξία σχετικά με την αξιοποίηση αυτών των τεχνικών για τη βελτίωση της διαδικασίας τεκμηρίωσης και της αυτόματης παραγωγής UML διαγραμμάτων, ιδιαίτερα καθώς τα γλωσσικά μοντέλα συνεχίζουν να εξελίσσονται και να βελτιώνονται.

Το άρθρο [8] δείχνει ότι τα γλωσσικά μοντέλα μπορούν να παράγουν UML διαγράμματα με γενικά ορθή σύνταξη, αν και παρουσιάζουν μικρά σφάλματα, τα οποία εξαρτώνται από τη χρησιμοποιούμενη σημειογραφία. Ειδικότερα, η παραγωγή διαγραμμάτων σε **PlantUML ήταν αισθητά πιο συντακτικά ορθή σε σύγκριση με άλλα εργαλεία**, όπως

το USE (UML-based Specification Environment). Παρά τις αδυναμίες τους, τα LLMs διαχειρίζονται αρκετά καλά βασικές έννοιες μοντελοποίησης, όπως συσχετίσεις, συναθροίσεις και απλή κληρονομικότητα, ενώ δυσκολεύονται με πιο σύνθετες έννοιες. Συνολικά, αν και δεν τα θεωρούν ακόμη αξιόπιστα εργαλεία για μοντελοποίηση, η μελέτη προτείνει ότι θα πρέπει να επενδυθεί προσπάθεια στη βελτίωση των δεξιοτήτων τους, καθώς μπορούν να διαδραματίσουν σημαντικό ρόλο στο μέλλον της μοντελοποίησης λογισμικού.

Παρόμοια, οι Speth et al. πειραματίστηκαν με τη κατανοήση και παραγωγή UML διαγραμμάτων σε μορφή κειμένου από το ChatGPT. Χρησιμοποίησαν PlantUML και Mermaid αλλά και υλικό σε μορφή εικόνας για τα πειράματά τους. Τα αποτελέσματα ήταν πολύ θετικά για τη UML σε μορφή κειμένου, με το μοντέλο να κατανοεί πολύ καλύτερα PlantUML και Mermaid αρχεία σε σχέση με γραφική είσοδο. Η παραγωγή των UML κειμένων κρίθηκε επίσης επιτυχής, ενώ η παραγωγή ορθών διαγραμμάτων σε μορφή εικόνας απέτυχε πλήρως [12].

Οι Wang et al [10] πειραματίστηκαν με την αυτόματη αξιολόγηση μοντέλων UML σπουδαστών από γλωσσικά μοντέλα. Έδειξαν ότι τα γλωσσικά μοντέλα (εξετάζοντας το ChatGPT) μπορούν να αξιολογήσουν UML διαγράμματα με ακρίβεια συγκρίσιμη με εκείνη ειδικών. Παρόλο που οι βαθμολογίες που αποδίδει είναι ελαφρώς χαμηλότερες από εκείνες των ειδικών, η συνέπεια στην αξιολόγηση των use case, class και sequence διαγραμμάτων υποδεικνύει ότι τέτοια μοντέλα μπορούν να χρησιμοποιηθούν ως υποστηρικτικά εργαλεία για την αυτόματη αξιολόγηση UML μοντέλωω, ενισχύοντας την άποψη ότι τα γλωσσικά μοντέλα έχουν τη δυνατότητα να κατανοούν αφαιρετικές δομές λογισμικού, συμβάλλοντας στην αυτοματοποίηση και την τυποποίηση σχεδιαστικών μοντέλων σε εκπαιδευτικά και επαγγελματικά περιβάλλοντα.

Στο πλαίσιο της εκπόνησης παράλληλης διπλωματικής εργασίας από το εργαστήριο, διεξήχθησαν πειράματα σχετικά με την επίδοση διάφορων μεγάλων γλωσσικών μοντέλων (LLM) στη παραγωγή UML βάση λεπτομερών εντολών και καθοδήγησης (prompt). Αξιολογήθηκαν στη παραγωγή απλών εικόνων, κώδικα **PlantUML**, κώδικα **MermaidJS** και **XMI**. Αξιοποιήθηκαν τα μοντέλα **ChatGPT-3.5, ChatGPT-4, Claude3.5** και **NotebookLM-Gemini-2.0-Flash** από τα οποία ζητήθηκαν διαγράμματα κλάσεων σχετιζόμενα με αρχιτεκτονικές λογισμικού τύπου Client-Server, Model-View-Controller (MVC) και Three-Tier.

Οι έξοδοι των μοντέλων αξιολογήθηκαν σε κλίματα 0-5 βάσει συνδυασμού κριτηρίων ποιότητας και ακρίβειας των αποτελεσμάτων, σύμφωνα με δικτυακό εργαλείο που υλοποιήθηκε σε προηγούμενη διπλωματική εργασία του εργαστηρίου για την απεικόνιση και αξιολόγηση παραγόμενης UML από LLMs του Μ. Τσιλιμιγκουνάκη [14]. Σε περίπτωση άκυρης εξόδου δίνεται η ειδική βαθμολογία -1.

Οι κατά μέσον όρο βαθμολογίες από σύνολο 66 πειραμάτων των εξόδον ανά μορφότυπο και μοντέλο παρουσιάζονται στον Πίνακα 2.1.

Από τις αρχιτεκτονικές που ζητήθηκαν να υλοποιηθούν από τα μοντέλα, δεν υπήρξε μεγάλη διακύμανση βαθμολογίας. Ο μέσος όρος αξιολόγησης ανά ζητούμενη αρχιτεκτονική ανεξαρ-

Μοντέλο	**Εικόνα**	**XMI**	**Mermaid**	**PlantUML**
ChatGPT-4	1.17	-1	3	3.5
Claude3.5	2.67	3.83	4	4.5
NotebookLM-Gemini-2.0-Flash	-	1.17	2.5	3.67
Συνολικά	**1.92**	**1.33**	**3.16**	**3.89**

Πίνακας 2.1: Επιδόσεις μοντέλων στη παραγωγή UML ανά μορφότυπο κατά μέσο όρο

Αρχιτεκτονική	**Συλονική Μ.Ο. βαθμολογία**
Three-Tier	2.67
MVC	2.6
Client-Server	2.7

Πίνακας 2.2: Συνολική επίδοση των μοντέλων στη παραγωγή UML ανά ζητούμενη αρχιτεκτονική κατά μέσο όρο

τήτως μοντέλου και μοροφοτύπου παρουσιάζεται στον πίνακα 2.2.

Αναφορικά με τη συνολική απόδοση των διαφόρων μοντέλων, όπως υπολογίστηκε κατά μέσο όρο για κάθε μορφότυπο, η κατάταξη των μορφότυπων διαμορφώνεται ως εξής, από τον καλύτερο στον χειρότερο:

1. PlantUML
2. Mermaid
3. Απλή εικόνα
4. Αρχείο XMI

Από τα αποτελέσματα, προκύπτει ότι το PlantUML αποτελεί την καλύτερη επιλογή για όλα τα μοντέλα που εξετάστηκαν, ενώ το Mermaid κατατάσσεται ως η δεύτερη καλύτερη επιλογή για όλα τα μοντέλα. Η συγκριτική υπεροχή αυτών των εργαλείων υπογραμμίζει τη σημασία της χρήσης εργαλείων που βασίζονται σε απλό κείμενο για την περιγραφή και παραγωγή διαγραμμάτων, καθώς επιτυγχάνουν μεγαλύτερη ευχέρεια και ακρίβεια στην αυτόματη δημιουργία των εν λόγω διαγραμμάτων.

Από την αρνητική βαθμολογία που καταγράφηκε, παρατηρούμε δυστυχώς ότι το πιο ευέλικτο και ευρέως αποδεκτό XMI δεν μπορεί, στη παρούσα φάση της εξέλιξης των μοντέλων, να παραχθεί με ορθό τρόπο αυτοματα, ο οποίος να τηρεί πλήρως τη δομή του μορφότυπου. Ως αποτέλεσμα, τα παραγόμενα XMI αρχεία δεν είναι ικανά να εισαχθούν σε προγράμματα και εργαλεία που δέχονται και επεξεργάζονται αρχεία XMI, περιορίζοντας τη χρησιμότητά τους.

Τέλος, η ποιότητα των παραγόμενων εικόνων ήταν απογοητευτική. Παρά το γεγονός ότι οι εικόνες προσφέρουν μια γρήγορη και εύκολη λύση, το περιεχόμενό τους δεν είναι σε

επίπεδο που να επιτρέπει την χρήση τους σε σοβαρές αναλύσεις ή παρουσιάσεις.

2.1.1 Παραδείγματα εξόδων παραγόμενης UML από LLMs

Παρακάτω παρατίθενται οι εξοδοι των πειραμάτων στη προτροπή (prompt) για την ίδια αρχιτεκτονική **Client-Server**, **που είχε τις καλύτερες βαθμολογίες** σύμφωνα με το εργαλείο της [14], όπως αυτές παράχθηκαν από το μοντέλο με τις **καλύτερες συνολικά αποδόσεις Claude3.5**.

Το prompt. Οι εξοδοι παράχθηκαν με prompt το οποίο αποτελείται από ένα αρχείο SRS (Software Requirement Specification) το οποίο περιγράφει την λειτουργία ενός λογισμικού "Dummy Coordination Conversion (DCC)" και μία περιγραφή της εργασίας που ζητήται να εκτελέσει το μοντέλο. Το DCC επιτρέπει στους χρήστες να διαχειρίζονται ομάδες συντεταγμένων σε καρτεσιανές και πολικές μορφές. Μέσω ενός γραφικού περιβάλλοντος χρήστη (GUI) που έχει αναπτυχθεί με Java, οι χρήστες μπορούν να εκτελούν λειτουργίες όπως η δημιουργία, τροποποίηση, διαγραφή και προβολή των ομάδων συντεταγμένων.

Οι χρήστες αλληλεπιδρούν με την εφαρμογή εισάγοντας τον τύπο των συντεταγμένων (καρτεσιανές ή πολικές), τις αντίστοιχες τιμές και την ετικέτα της ομάδας συντεταγμένων. Οι ομάδες αποθηκεύονται στη βάση δεδομένων του συστήματος και οι συντεταγμένες μετατρέπονται μεταξύ των δύο μορφών, κατά περίπτωση.

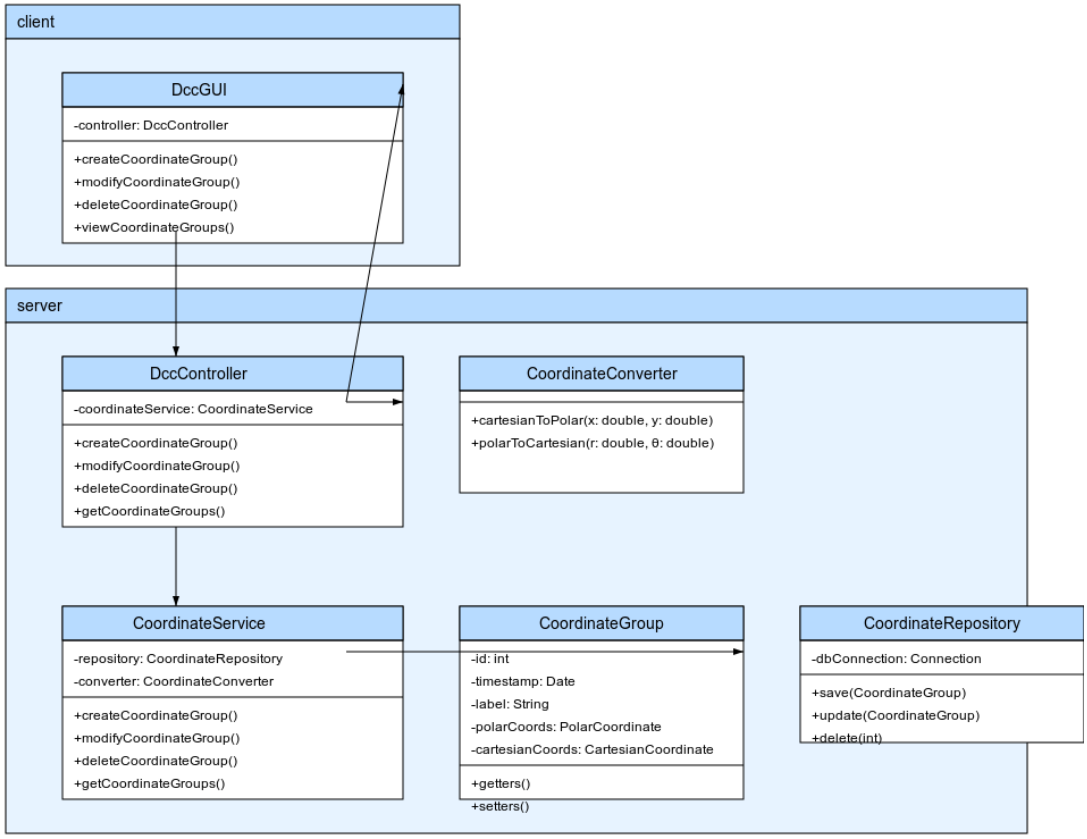
Το σύστημα υποστηρίζει:

- Δημιουργία ομάδας συντεταγμένων: Εισάγοντας τις συντεταγμένες και την ετικέτα, η νέα ομάδα αποθηκεύεται και οι συντεταγμένες μετατρέπονται στη δεύτερη μορφή.
- Προβολή όλων των ομάδων: Η εφαρμογή επιτρέπει στους χρήστες να δουν όλες τις αποθηκευμένες ομάδες, ενώ αν δεν υπάρχουν, εμφανίζεται μια κενή λίστα.
- Προβολή ομάδων ανά ετικέτα: Ο χρήστης μπορεί να αναζητήσει ομάδες με βάση την ετικέτα τους και να δει τα αποτελέσματα ή μια κενή λίστα αν δεν υπάρχουν αντιστοιχίες.
- Διαγραφή ομάδας συντεταγμένων: Οι χρήστες μπορούν να επιλέξουν μια ομάδα συντεταγμένων και να τη διαγράψουν από τη βάση δεδομένων.

Η βάση δεδομένων του συστήματος αποθηκεύει τα απαραίτητα στοιχεία για κάθε ομάδα συντεταγμένων, συγκεκριμένα αναγνωριστικό (ID), χρόνο δημιουργίας (timestamp), ετικέτα της ομάδας, τιμές σε καρτεσιανές και πολικές συντεταγμένες.

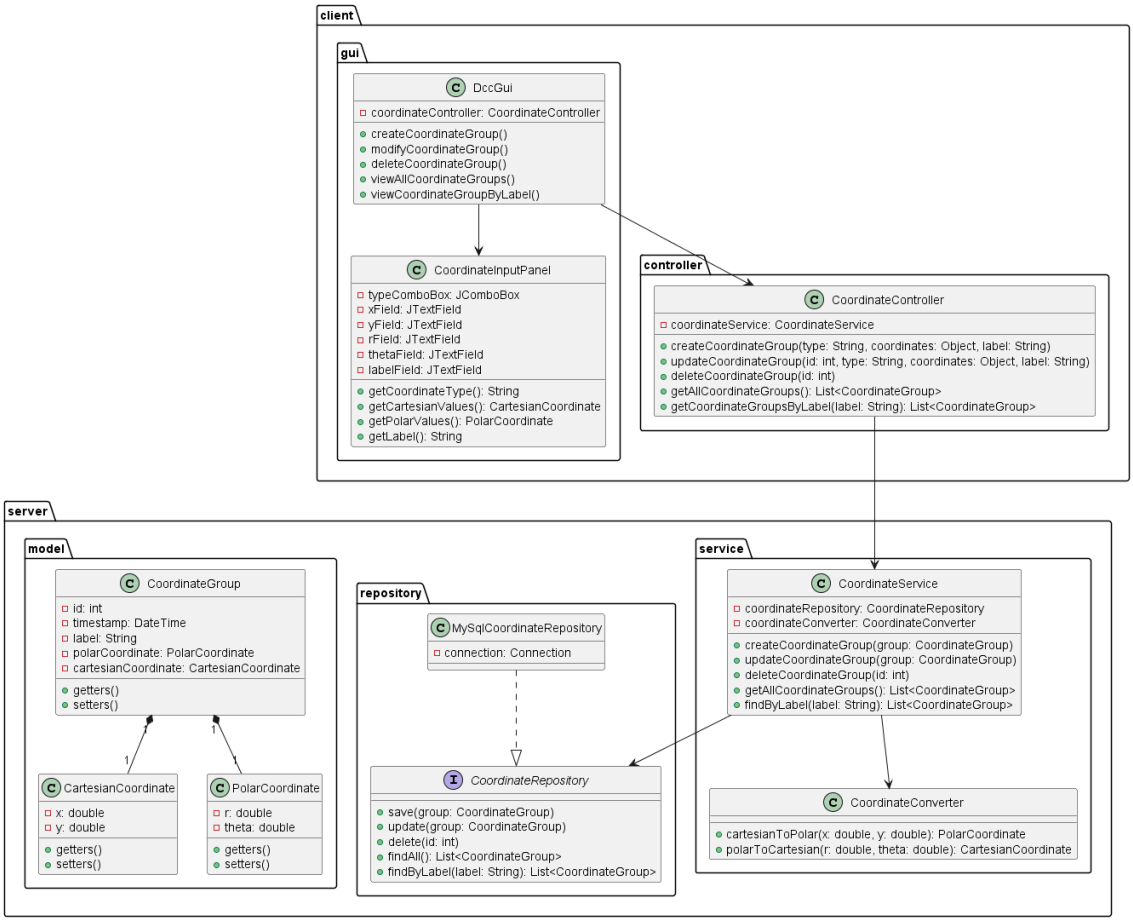
Στο SRS αναφέρονταιτα παραπάνω, οι περιπτώσεις χρήσης και ροές αλληλεπίδρασης του χρήστη με την εφαρμογή, συνοδευόμενα από διαγράμματα UML σε PlantUML μορφότυπο. Η περιγραφή της εργασίας που ζητήται να γίνει από το μοντέλο περιλαμβάνει εντολή για δημιουργία ενός διαγράμματος κλάσεων και την επιθυμητή αρχιτεκτονική (π.χ. Client-Server).

Εικόνα (βαθμολογία 2) Από τις καλύτερες εξόδους εικόνας, παρουσιάζει κάποια συνοχή και σχετικές μεθόδους και κλάσεις με έγκυρο κείμενο. Λείπει ακρίβεια, και οι σύνδεσμοι είναι άκυροι.



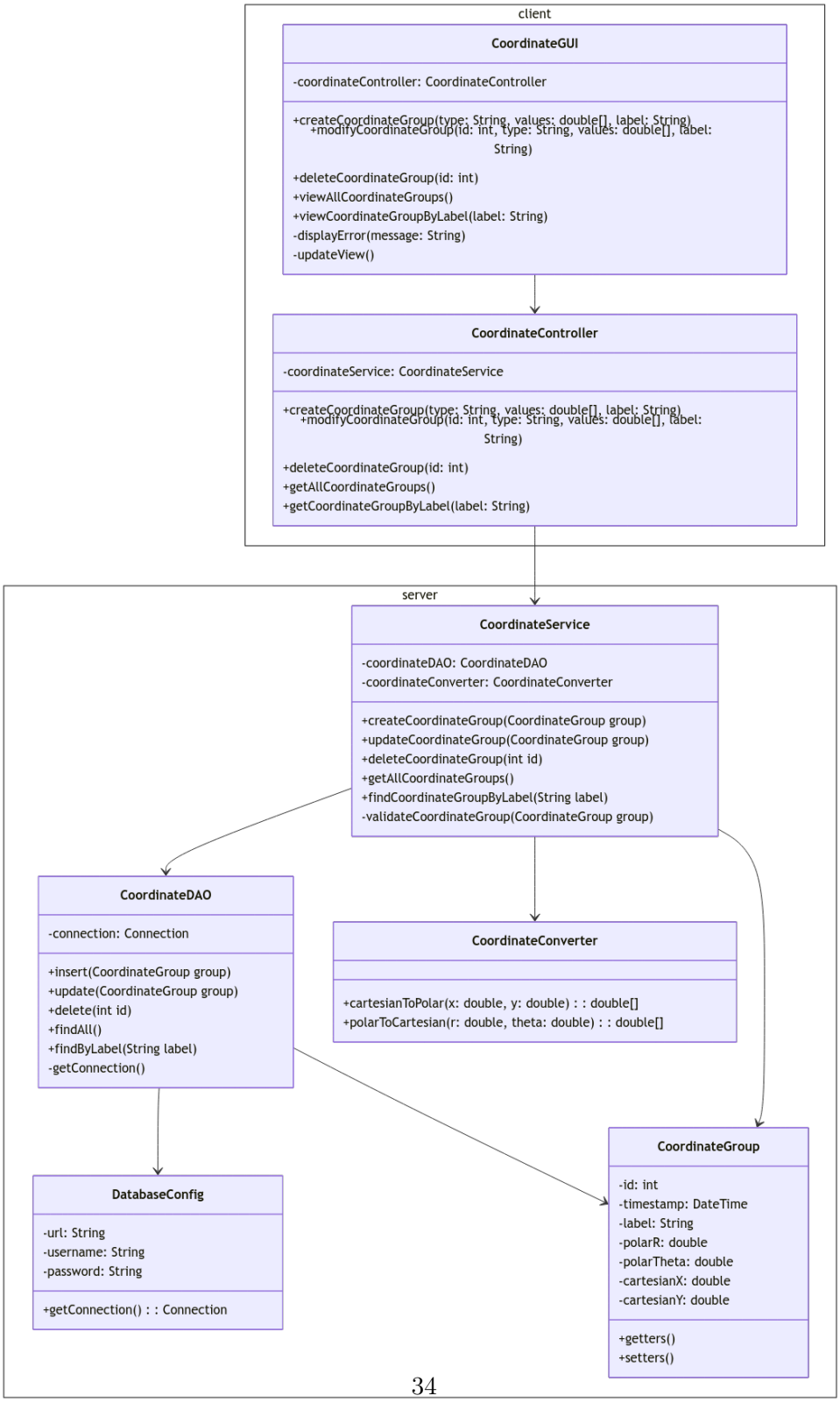
Σχήμα 2.1: Αρχιτεκτονική Client-Server παραγόμενη από το Claude3.5 σε μορφή εικόνας

PlantUML (βαθμολογία 4) Η πιο ακριβής και πλήρης αναπαράσταση της αρχιτεκτονικής βάσει της περιγραφής του συστήματος.



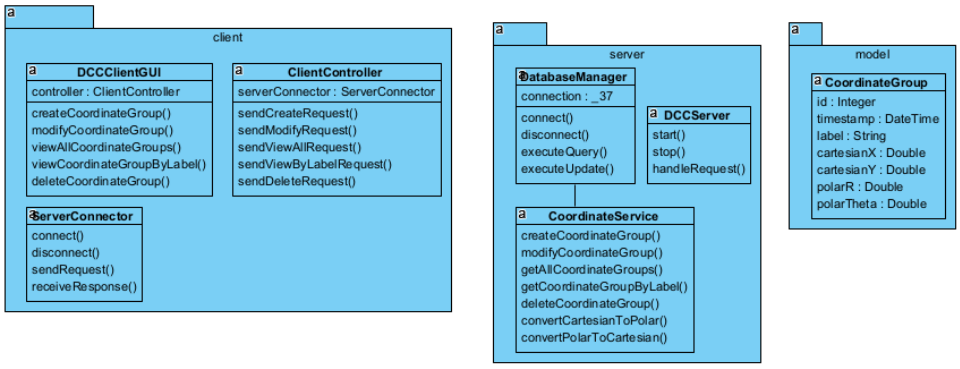
Σχήμα 2.2: Αρχιτεκτονική Client-Server παραγόμενη από το Claude3.5 σε PlantUML

Mermaid (βαθμολογία 3) Σχετικά καλή απεικόνιση της αρχιτεκτονικής, λίγο κατώτερης του PlantUML.



Σχήμα 2.3: Αρχιτεκτονική Client-Server παραγόμενη από το Claude3.5 σε Mermaid

XMI (βαθμολογία 3) Στο σχήμα 2.4 περιέχεται το διάγραμμα που προέκυψε από την επιτυχή εισαγωγή του XMI αρχείου που παράχθηκε από το μοντέλο. Παρατηρούμε πιστή αναπαράσταση, έλλειψη συνδέσμων ως διαγραμματικά στοιχεία (εισάχθηκε ένας μόνο σύνδεσμος). Πρόκειται για μία από τις καλύτερες επιδόσεις σε XMI, με τις περισσότερες δοκιμές να αποτυχάνουν στην εισαγωγή στο Visual Paradigm.



Σχήμα 2.4: Αρχιτεκτονική Client-Server παραγόμενη από το Claude3.5 σε XMI μετά από εισαγωγή στο Visual Paradigm

2.1.2 Η επιλογή του PlantUML ως καταλληλότερη για τη περίπτωση χρήσης

Με βάση τα αποτελέσματα των πειραμάτων που εκτελέστηκαν και παρουσιάστηκαν στην υποενότητα 2.1, οι μόνες εφικτές επιλογές μεταξύ των μορφοτύπων όσον αφορά την αυτοματοποίηση του σχεδιασμού UML διαγραμμάτων είναι τα βασισμένα σε κείμενο **PlantUML** και **Mermaid**.

Η συνδυασμένη αξιολόγηση και σύγκρισή τους στον πίνακα 2.3 προκύπτει από τα πειράματα και τα βασικά χαρακτηριστικά τους όπως παρουσιάστηκαν στις αντίστοιχες υποενότητες.

PlantUML	**Mermaid**
Βαθμολογία επίδοσης παραγωγής απο LLM **3.89**	Βαθμολογία επίδοσης παραγωγής απο LLM **3.16**
Υποστηρίζει όλα τα διαγράμματα UML	Υποστηρίζει Class, Sequence, State διαγράμματα
Παράγει στατικά διαγράμματα	Είναι δυνατή αλληλεπίδραση με το παραγόμενο διάγραμμα

Πίνακας 2.3: Σύγκριση PlantUML και Mermaid

Ο μορφότυπος PlantUML **συνδυάζει τη μέγιστη βαθμολογία** ποιότητας παραγωγής από τα γλωσσικά μοντέλα με την **πλήρη υποστήριξη των κλασικών διαγραμμάτων UML**. Το Mermaid παρέχει μία πιο σύγχρονη εμπειρία χρήστη σε σχέση με το PlantUML αλλά προσφέρεται περισσότερο για την δημιουργία κοινών (όχι αυστηρά τυποποιημένων όπως η UML) διαγραμμάτων έχοντας περιορισμένη υποστήριξη διαγραμμάτων UML αλλά και λιγότερο ικανοποιητικά πειραματικά αποτελέσματα.

Με βάση την υπεροχή του PlantUML στη περίπτωση χρήσης της αυτοματοποίησης που αποτέλεσε το κίνητρο για τη δημιουργία του εργαλείου της εργασίας, επιλέχθηκε αυτό αντί των υπόλοιπων λύσεων που εξετάστηκαν.

2.2 Παραδείγματα αξιοποίησης της διαλειτουργικότητας PlantUML και Visual Paradigm

Μία απολύτως έγκυρη περίπτωση χρήσης του προτεινόμενου εργαλείου διαλειτουργικότητας μεταξύ του PlantUML και του Visual Paradigm είναι η χρήση του από έναν κοινό χρήστη που θέλει να μετατρέψει τα διαγράμματά του ή να αλλάξει περιβάλλον μοντελοποίησης.

Πιθανώς επιθυμεί να μεταναστεύσει τα πρόχειρα διαγράμματά του από το PlantUML στο πιο ολοκληρωμένο και σημασιολογικά πλούσιο, ενσωματωμένο περιβάλλον του Visual Paradigm. Ίσως αντίστοιχα θέλει να απομακρυνθεί από το Visual Paradigm μετατρέποντας το έργο του σε ένα πιο ελαφρύ μορφότυπο, το οποίο μπορεί να τρέξει σε οποιονδήποτε υπολογιστή χωρίς την ανάγκη για την εφαρμογή του Visual Paradigm, και του αρκεί η απλότητα του PlantUML.

Ωστόσο, σε αυτή την υποενότητα θα εστιάσουμε στην ενσωμάτωση του εργαλείου για την περίπτωση χρήσης **αξιοποίησης LLM** για τη παραγωγή διαγραμμάτων, παρουσιάζοντας δύο σχετιζόμενες ροές χρήσης.

2.2.1 Εισαγωγή αυτόματα παραγόμενου PlantUML κώδικα στο περιβάλλον του Visual Paradigm

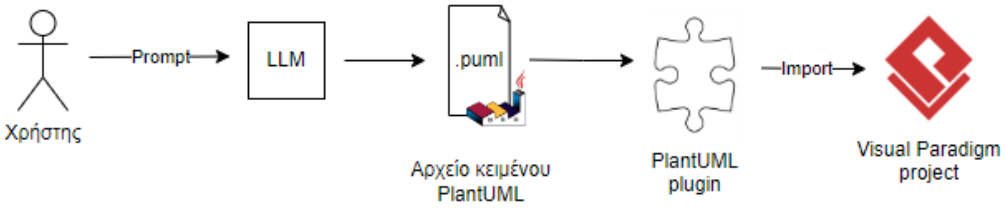
Η πρώτη περιπτωση χρήσης που προτείνουμε αξιοποιεί τη **λειτουργία εισαγωγής** του Plugin στο Visual Paradigm.

Σε αυτήν, ο χρήστης καθοδηγεί ένα γλωσσικό μοντέλο δίνοντας του ένα λεπτομερές prompt με τη περιγραφή του προβλήματος ή συστήματος που επιχειρεί να μοντελοποιήσει με UML. Καθορίζει τον επιθυμητό τύπο ή τύπους διαγράμματος σε PlantUML και την όψη του συστήματος που θέλει να αναπαραστήσει με όσο το δυνατόν πιο λεπτομερή τρόπο για καλύτερα αποτελέσματα.

Στη συνέχεια, αν είναι ικανοποιημένος από το παραγόμενο διαγραμμα, το αποθηκεύει σε αρχείο κειμένου. Στην εφαρμογή του Visual Paradigm όπου είναι εγκαταστημένη η επέκταση

PlantUML plugin, εκτελεί import (εισαγωγή) του συνόλου των διαγραμμάτων PlantUML που συγκέντρωσε από το μοντέλο.

Στο τέλος, εφόσον η εισαγωγή δεν εμφάνισε κάποιο συντακτικό ή άλλο πρόβλημα, τα διαγράμματα είναι σε μορφή Visual Paradigm και μπορούν να τελειοποιηθούν με επεξεργασία από το χρήστη και να εμπλουτιστούν σημασιολογικά με τις πρόσθετες λειτουργίες του περιβάλλοντος.



Σχήμα 2.5: Ροή περίπτωσης χρήσης εισαγωγής αυτόματα παραγόμενου PlantUML κώδικα στο Visual Paradigm

2.2.2 Εξαγωγή διαγραμμάτων Visual Paradigm προς χρήση από LLMs

Η δεύτερη περίπτωση χρήσης αφορά τη **λειτουργία εξαγωγής** (export) από το Visual Paradigm προς PlantUML μορφή.

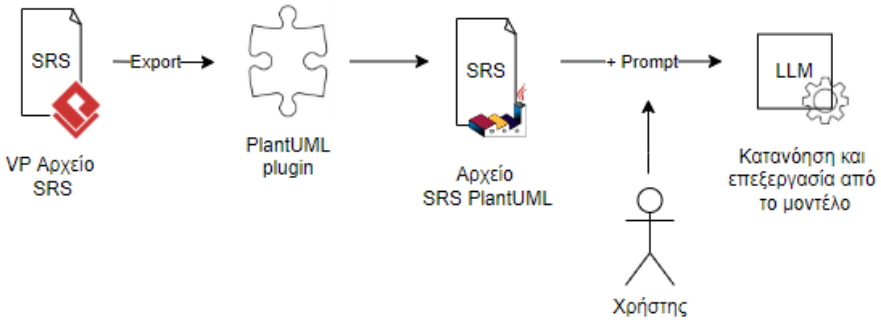
Στη προηγούμενη περίπτωση χρήσης ήταν αναγκαία η λεπτομερής περιγραφή του συστήματος και της επιθυμητής εξόδου του μοντέλου στο LLM για τη καλύτερη επίδοση του. Σε περίπτωση όμως που υπάρχει ένα αρχείο SRS (Software Requirements Specification) που περιγράφει ήδη το σύστημα, συνήθως με όλες τις απαραίτητες πληροφορίες και κάποια διαγράμματα UML ενσωματωμένα σε αυτο, η διαδικασία συγγραφής του κατάλληλου prompt για το μοντέλο μπορεί να απλουστευτεί σημαντικά.

Έστω πως υπάρχει ένα έργο Visual Paradigm (*.vpp project) που μοντελοποιεί ένα περίπλοκο και εκτενές σύστημα, το οποίο περιέχει και το αντίστοιχο αρχείο SRS που περιγράφει τις απαιτήσεις του λογισμικού. Κάποια από τα πιο σημαντικά και αντιπροσωπευτικά διαγράμματα του έργου θα είναι ενσωματωμένα σε αυτό, σε μορφή διαγράμματος του Visual Paradigm.

Το αρχείο αυτό αν δοθεί συννημένο σε ένα γλωσσικό μοντέλο έχει ιδιαίτερη αξία για τη κατανόηση του συνόλου του συστήματος. Δεν νοείται ωστόσο να παραληφθούν τα διαγράμματα από αυτό, καθώς είναι θεμελιώδη. Μία λύση που αξιοποιεί το plugin βρίσκεται στην

μετατροπή αυτών των διαγραμμάτων σε PlantUML και η αντικατάσταση των αντίστοιχων Visual Paradigm διαγραμμάτων με αυτά.

Έτσι, μετατρέπουμε ένα **Visual Paradigm SRS σε ένα PlantUML SRS** το οποίο μπορεί να διοχετευθεί σε ένα LLM. Το μοντέλο στη συνέχεια μπορεί να επεξεργαστεί τη πληροφορία όπως το κατευθύνουμε, π.χ. για να δημιουργήσει κάποιο συμπληρωματικό διάγραμμα σε PlantUML για το έργο. Το συμπληρωματικό διάγραμμα μπορεί με τη σειρά του να ενταχθεί μέσω της λειτουργίας εισαγωγής στο Visual Paradigm project και να τροποποιηθεί όπως χρειάζεται.



Σχήμα 2.6: Ροή περίπτωσης χρήσης εξαγωγής SRS με Visual Paradigm διαγράμματα σε SRS με PlantUML κώδικα

Φυσικά, εξίσου χρήσιμο μπορεί να είναι το εργαλείο και για τη περίπτωση που θέλουμε να περάσουμε ένα Visual Paradigm διάγραμμα σε ένα γλωσσικό μοντέλο ώστε να το αξιολογήσει ή να το επεξεργαστεί. Σε αυτή τη περίπτωση, είναι προτιμότερο να μεταφραστεί σε διάγραμμα PlantUML (αντί για εξαγωγή ως εικόνα) ώστε να μπορεί να το κατανοήσει ή/και να το τροποποιήσει κατάλληλα.

Κεφάλαιο 3

Σχεδιασμός και αρχιτεκτονική του plugin

3.1 Η διεπαφή Visual Paradigm Open API

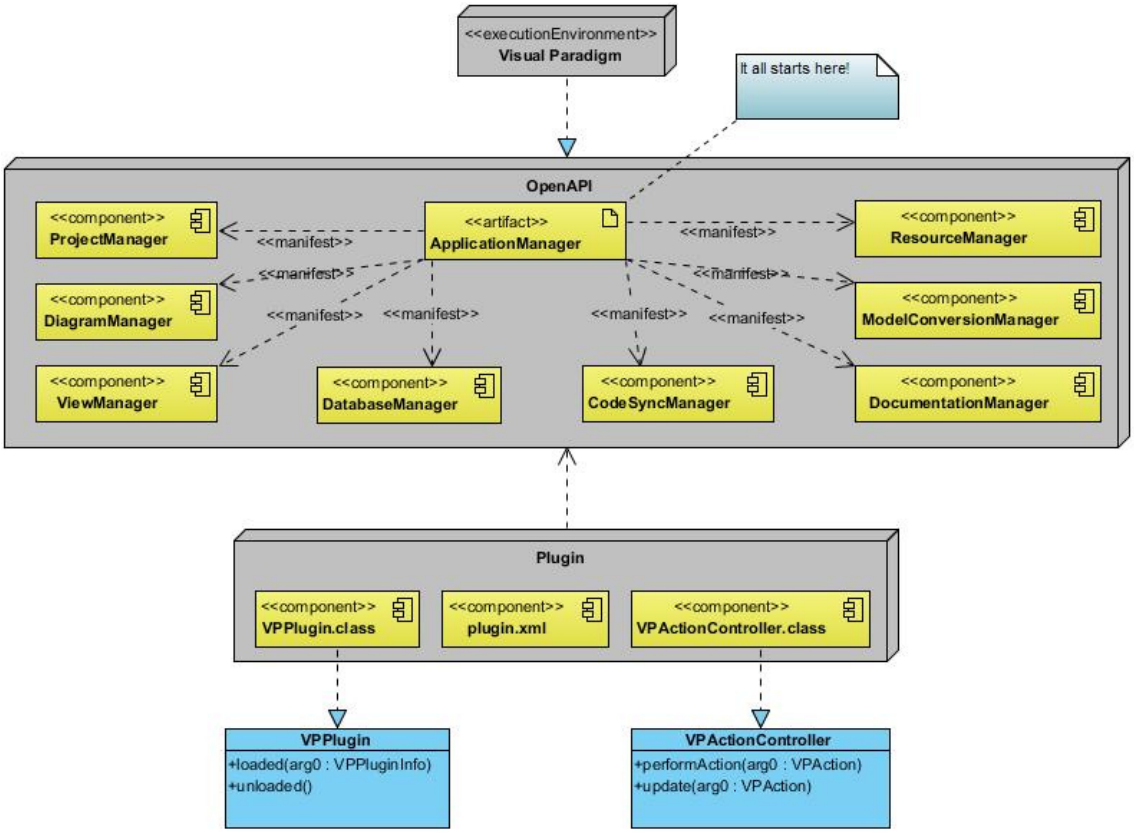
Το Visual Paradigm παρέχει το *Visual Paradigm Open API*, μία εκτενέστατη προγραμματιστική διεπαφή σε Java που δίνει τη δυνατότητα σε προγραμματιστές να επεκτείνουν τις λειτουργίες της desktop εφαρμογής μέσω plugins. Με αυτό το API, μπορούμε να δημιουργούμε, να διαγράφουμε και να μετατρέπουμε κάθε μοντέλο ή διάγραμμα που επιθυμούμε σε ένα project. [20].
Δυστυχώς, η τεκμηρίωση της διεπαφής είναι ελλιπής, γεγονός που έχει οδηγήσει σε σχετικά λίγες προσπάθειες ανάπτυξης προσθηκών όπως και σε περιορισμούς στην εκμετάλλευση των πλήρη δυνατοτήτων της διεπαφής στη παρούσα εργασία. Βασιστήκαμε κυρίως σε επίσημα παραδείγματα και οδηγούς από το Visual Paradigm καθώς και παραδείγματα άλλων χρηστών.
Στο Σχήμα 3.1 παρουσιάζονται τα βασικά συστατικά της διεπαφής που προσφέρει το Visual Paradigm καθώς και τα στοιχεία που πρέπει κατ΄ ελάχιστο να υλοποιεί ένα plugin, ώστε να εγκατασταθεί με επιτυχία στην εφαρμογή.

3.1.1 Δομή API

Γενικά

Όπως φαίνεται στο Σχήμα 3.1, το API αξιοποιείται μέσα από τον ApplicationManager. Οι περισσότερες δυνατότητες του OpenAPI παρέχονται μέσω διεπαφών. Η ανάπτυξη ενός plugin ξεκινά συνήθως από τον ApplicationManager..

[1]Πηγή: https://forums.visual-paradigm.com/t/guide-getting-started-with-plugin-development/13915/2



Σχήμα 3.1: Visual Paradigm Open API - διάγραμμα component[1]

Το OpenAPI αποτελείται από διάφορα μέρη, καθένα από τα οποία ελέγχεται από μια διαχειριστική κλάση. Συνοπτικά:

- **ProjectManager**: Παρέχει έλεγχο στο τρέχον έργο (αρχείο *.vpp). Μέσω του project manager, ο προγραμματιστής μπορεί να ανοίξει ή να αποθηκεύσει ένα έργο, καθώς και να αποκτήσει πρόσβαση στην κλάση IProject, η οποία υποστηρίζει εργασίες όπως η ανάκτηση των διαγραμμάτων, η λήψη του ονόματος του έργου κ.α.
- **DiagramManager**: Είναι υπεύθυνος για τη διαχείριση των διαγραμμάτων. Με αυτόν, μπορείτε να δημιουργήσετε νέα διαγράμματα, να έχετε μια συνολική εικόνα των ανοιχτών διαγραμμάτων και να δημιουργήσετε συγκεκριμένα στοιχεία διαγραμμάτων, όπως shapes.
- **ViewManager**: Ελέγχει όσα αφορούν το γραφικό περιβάλλον χρήστη. Χρησιμοποιείται για την προβολή μηνυμάτων, την εμφάνιση διαλόγων και την ενημέρωση του χρήστη με πληροφορίες μέσω της γραμμής μηνυμάτων ή απλών παραθύρων διαλόγου.

Για την ανάπτυξη του προιόντος της παρούσας εργασίας αξιοποιήθηκαν μόνο οι παραπάνω κλάσεις.

Ωστόσο, το OpenAPI περιλαμβάνει και άλλους διαχειριστές, όπως τους databaseManager, codeSyncManager, resourceManager, modelConversionManager και documentationManager, που καλύπτουν εξειδικευμένες λειτουργίες όπως η δημιουργία SQL κώδικα, η παραγωγή πηγαίου κώδικα, η διαχείριση εξωτερικών πόρων και η δημιουργία αναφορών.

Μοντέλα - IModelElement

Τα μοντέλα αποτελούν την αφηρημένη βάση πίσω από την οπτική αναπαράσταση ενός στοιχείου του διαγράμματος, είτε αυτό είναι μία κλάση σε διάγραμμα κλάσεων είτε μία περίπτωση χρήσης σε ένα Use Case diagram. Με άλλα λόγια, ενώ ένα στοιχείο διαγράμματος προσφέρει μια οπτική απεικόνιση, το αντίστοιχο μοντέλο περιέχει τις βασικές πληροφορίες, τις ιδιότητες και τις σχέσεις που καθορίζουν τη λογική του δομή. Αυτή η διάκριση επιτρέπει την ανεξάρτητη διαχείριση της επιχειρησιακής λογικής από την απεικόνιση.

Κάθε στοιχείο ενός διαγράμματος πρέπει να αναφέρεται σε κάποιο υπάρχον μοντέλο σε ένα Visual Paradigm project.

Το API παρέχει ολοκληρωμένες διεπαφές για τη δημιουργία, ανάκτηση, ενημέρωση και διαγραφή μοντέλων. Η βασική κλάση που αντιπροσωπεύει ένα μοντέλο είναι η IModelElement, όλες οι κλάσεις που εξειδικεύουν μοντέλα κληρονομούν από αυτή (π.χ. IClass, IActivityAction). Όλα τα μοντέλα βρίσκονται εντός ενός IProject.

Δημιουργία IModelElement

- Για τη δημιουργία IModelElement, ο προγραμματιστής μπορεί να χρησιμοποιήσει τη IModelElementFactory. Η IModelElementFactory παρέχει μεθόδους για τη δημιουρ-

γία όλων των τύπων μοντέλων μέσω της κλήσης της μεθόδου του instance() της, π.χ. IModelElementFactory.instance().createClass().

- Σε εξειδικευμένες περιπτώσεις, όπως για παράδειγμα ένα IClass (που κληρονομεί από IModelElementParent), υπάρχουν επιπλέον μέθοδοι, π.χ. createOperation(), για τη δημιουργία ενός IOperation, το οποίο είναι επίσης υποκλάση του IModelElement.

Ανάκτηση IModelElement

- Τα IModelElements μπορούν να ανακτηθούν μέσω του IProject, το οποίο περιέχει όλα τα μοντέλα, διαγράμματα και τα στοιχεία διαγραμμάτων, μέσω iterator ή άλλων μεθόδων.
- Εναλλακτικά, μέσω του VPContext που παρέχεται από τον ActionController, ο προγραμματιστής μπορεί να λάβει το ενεργό μοντέλο σχετικό με το περιβάλλον (π.χ. από ένα popup menu σε ένα διάγραμμα).

Ενημέρωση IModelElement

- Κάθε model element διαθέτει μεθόδους get/set για την ενημέρωση των ιδιοτήτων του.
- Σε περιπτώσεις που απαιτείται πρόσβαση σε εξειδικευμένες λειτουργίες, το IModelElement μπορεί να μετατραπεί στην αντίστοιχη υποκλάση του (π.χ. IClass) ώστε να κληθούν οι κατάλληλες μέθοδοι.

Διαγραφή IModelElement

- Η διαγραφή ενός μοντέλου γίνεται με την κλήση της μεθόδου delete() στο αντίστοιχο αντικείμενο.

Δράση	**Κύριες Διεπαφές**	**Περιγραφή**
Δημιουργία	IModelElementFactory, IModelElementParent	Χρήση του IModelElementFactory για τη δημιουργία νέων μοντέλων ή δημιουργία child model σε ένα γονικό.
Ανάκτηση	IProject, VPContext	Λήψη μοντέλων από το έργο ή από το περιβάλλον του ποπυπ μενυ.
Ενημέρωση	IModelElement (και υποκλάσεις)	Ενημέρωση ιδιοτήτων, μετατροπή σε εξειδικευμένη υποκλάση για πρόσβαση σε επιπλέον λειτουργίες.
Διαγραφή	IModelElement	Διαγραφή του μοντέλου με την κλήση της μεθόδου delete().

Πίνακας 3.1: Βασικές λειτουργίες με IModelElements

Στοιχεία διαγραμμάτων

Το API παρέχει επίσης λειτουργίες για τη δημιουργία, ανάκτηση, ενημέρωση και διαγραφή διαγραμμάτων και των στοιχείων τους. Αυτά αφορούν μόνο την όψη των στοιχείων και δεν επηρεάζουν τα μοντέλα που απεικονίζουν.

Δημιουργία IDiagram και IDiagramElement

- Η δημιουργία διαγραμμάτων (αντιπροσωπεύονται από την IDiagramUIModel) και των αντίστοιχων στοιχείων diagram elements (αντιπροσωπεύονται από την IDiagramElement) πραγματοποιείται μέσω του DiagramManager.
- Η πρόσβαση στον DiagramManager γίνεται μέσω της μεθόδου `ApplicationManager.instance().getDiagramManager()`.
- Σε ορισμένες περιπτώσεις, το αντίστοιχο μοντέλο για ένα diagram element δημιουργείται αυτόματα κατά τη δημιουργία του.

Ανάκτηση IDiagram και IDiagramElement

- Τα διαγράμματα μπορούν να ανακτηθούν μέσω του IProject με την κλήση της μεθόδου diagramIterator().
- Επιπλέον, από κάθε διάγραμμα (IDiagramUIModel), μπορούν να εξαχθούν τα στοιχεία του μέσω της μεθόδου diagramElementIterator().
- Εναλλακτικά, το VPContext που παρέχεται από τον ActionController μπορεί να δώσει πρόσβαση στο ενεργό διάγραμμα ή στα επιλεγμένα diagram elements.

Ενημέρωση IDiagram και IDiagramElement

- Η ενημέρωση της εμφάνισης και της συμπεριφοράς των διαγραμμάτων γίνεται μέσω των μεθόδων που παρέχονται στις διεπαφές IDiagramUIModel, IShapeUIModel και IConnectorUIModel.
- Τα diagram elements διαχωρίζονται σε σχήματα (IShapeUIModel) και συνδέσμους (IConnectorUIModel), τα οποία προσφέρουν διαφορετικές λειτουργίες ενημέρωσης.

Διαγραφή Diagrams και Diagram Elements

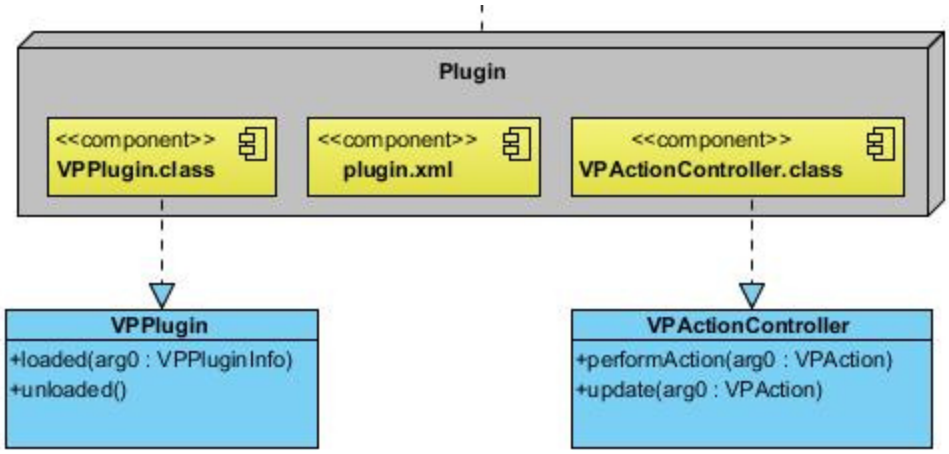
- Η διαγραφή πραγματοποιείται με την κλήση της μεθόδου delete() στην αντίστοιχη υλοποίηση, είτε πρόκειται για IDiagramUIModel είτε για IDiagramElement.

3.1.2 Δομή Plugin

Ένα plugin απαιτεί τουλάχιστον τρία βασικά συστατικά, όπως αυτά φαίνονται στο component 'Plugin' του σχήματος 3.2.

Δράση	Κύριες Διεπαφές	Περιγραφή
Δημιουργία	DiagramManager, IDiagramUIModel, IDiagramElement	Δημιουργία νέων διαγραμμάτων και στοιχείων, με ή χωρίς αυτόματη δημιουργία του αντίστοιχου μοντέλου.
Ανάκτηση	IProject, VPContext	Ανάκτηση διαγραμμάτων του έργου.
Ενημέρωση	IDiagramUIModel, IShapeUIModel, IConnectorUIModel	Ενημέρωση ιδιοτήτων εμφάνισης και λειτουργίας των διαγραμμάτων και των στοιχείων τους.
Διαγραφή	IDiagramUIModel, IDiagramElement	Διαγραφή των διαγραμμάτων ή των στοιχείων μέσω της μεθόδου delete().

Πίνακας 3.2: Βασικές λειτουργίες με IDiagramUIModel και IDiagramElement



Σχήμα 3.2: VP OpenAPI εστίαση στο Plugin - διάγραμμα component

Plugin.xml Το αρχείο plugin.xml αποτελεί τη βάση της υλοποίησης, όπου ορίζονται το αναγνωριστικό, το όνομα και άλλα στοιχεία του plugin, καθώς και τα ονόματα και οι τοποθεσίες των βασικών κλάσεων που κάνουν implement τις διεπαφές VPPlugin και VPActionController. Αν απαιτούνται εξωτερικές βιβλιοθήκες, αυτές επίσης δηλώνονται στο XML αρχείο. Ανάλογα με την επιθυμητή συμπεριφορά του plugin υπό ανάπτυξη, δηλώνονται πολλά διαφορετικά XML elements. Τα βασικά, αυτά που αφορούν την υλοποίηση του PlantUML plugin, περιγράφονται στον Πίνακα 3.3.

VPPlugin Η διεπαφή VPPlugin απαιτείται να υλοποιηθεί από μία κλάση του plugin και συνήθως δεν χρειάζεται να τροποποιηθούν οι υλοποιήσεις των abstract μεθόδων loaded() και unloaded(), οι οποίες καλούνται όταν το plugin φορτώνεται ή ξεφορτώνεται στο Visual Paradigm.

Element	Attribute	Περιγραφή
plugin		Το ριζικό στοιχείο του plugin.xml, που ορίζει τις βασικές πληροφορίες του πλυγιν (π.χ. id, name, provider, κ.ά.).
plugin	class	Η κλάση του plugin, η οποία απαιτείται να υλοποιεί το com.vp.plugin.VPPlugin.
runtime		Το στοιχείο που ορίζει το περιβάλλον εκτέλεσης του plugin.
library (1..*)		Ορίζει το αρχείο .jar ή το φάκελο που περιέχει τις κλάσεις του plugin καθώς και τις βιβλιοθήκες που απαιτούνται.
library (1..*)	Path	Η διαδρομή προς το αρχείο .jar ή το φάκελο.
library (1..*)	relativePath (προαιρετικό)	Καθορίζει εάν η διαδρομή είναι σχετική.
actionSets		actionSet και contextSensitiveActionSet. Το actionSet αφορά ενέργειες που εμφανίζονται στο μενού, ενώ το contextSensitiveActionSet ενέργειες που εμφανίζονται στο αναδυόμενο μενού.

Πίνακας 3.3: Περιγραφή βασικών στοιχείων του plugin.xml[2]

```
1  package sample.plugin;
2
3  public class SamplePlugin implements com.vp.plugin.VPPlugin {
4      // make sure there is a constructor without any parameters
5      public SamplePlugin() {
6          // default constructor
7      }
8
9      public void loaded(com.vp.plugin.VPPluginInfo info) {
10         // called when the plugin is loaded
11         // developer can get the current plugin's id and the
12         // current plugin directory (default: %Visual-Paradigm%/
               plugins)
13         // of Visual-Paradigm from the VPPluginInfo.
14     }
15
16     public void unloaded() {
17         // called when the plugin is unloaded (when Visual Paradigm
               will be exited)
18     }
19 }
```

Listing 3.1: VPPlugin Sample implementation

VPActionController Το τρίτο και πιο ουσιώδες στοιχείο στην ανάπτυξη ενός plugin αφορά την υλοποίηση της διεπαφής VPActionController. Για κάθε επιθυμητή λειτουργία απαιτείται η δημιουργία ενός implementation της διεπαφής αυτής, το οποίο στη συνέχεια δηλώνεται μέσα σε ένα στοιχείο <actionSet> ως <action> στο αρχείο plugin.xml.

Η συγκεκριμένη κλάση ορίζει τη συμπεριφορά που θα εκτελεστεί όταν ενεργοποιηθεί η αντίστοιχη ενέργεια, όπως έχει προδιαγραφεί στο actionSet. Συγκεκριμένα, όταν η ενέργεια πυροδοτηθεί, καλείται η μέθοδος performAction(), η οποία υλοποιείται από τον προγραμματιστή ώστε να εκτελεί την επιθυμητή λειτουργία.

Για την παρούσα εργασία, έχουν υλοποιηθεί δύο ActionControllers, συγκεκριμένα τα PlantUMLImportController και PlantUMLExportController, που διαχειρίζονται αντίστοιχα τις διαδικασίες εισαγωγής και εξαγωγής.

Η τελική μορφή του plugin.xml παρουσιάζεται στο 3.2. Εστιάζουμε στα actionSets, τα οποία καθορίζουν την τοποθεσία και την εμφάνιση των προστιθέμενων επιλογών στο μενού. Για παράδειγμα, το ribbonPath="Project/Import/XML" υποδηλώνει ότι η επιλογή εισαγωγής τοποθετείται στο μενού των εισαγωγών, κάτω από την ήδη υπάρχουσα επιλογή "XML", ενώ το ribbonPath="Project/Export/XML" προσδιορίζει την τοποθεσία της επιλογής εξαγωγής.

Όταν ο χρήστης επιλέγει μία από τις παραπάνω επιλογές, καλείται η μέθοδος *performAction()* της αντίστοιχης κλάσης. Η συγκεκριμένη κλάση δηλώνεται μέσω της ιδιότητας actionController.class στο plugin.xml και υλοποιεί το VPActionController. Η μέθοδος *performAction()* είναι υπεύθυνη για την εκτέλεση της επιθυμητής λειτουργίας που έχει καθοριστεί, ανταποκρινόμενη στο συμβάν που προκαλείται από την επιλογή του χρήστη.

```
<plugin
    id="plugins.plantUML"
    name="PlantUML Support"
    description="Import from / Export to PlantUML"
    provider="Natalia Bourdi"
    class="plugins.plantUML.PlantUML">

    <actionSets>
        <actionSet id="plugins.plantUML.actionset">
            <action
                id="ImportAction"
                actionType="generalAction"
                label="PlantUML..."
```

```
                tooltip="PlantUML..."
                style="normal"
                icon="icons/puml-icon.png"
                ribbonPath="Project/Import/XML">
                <actionController
                    class="plugins.plantUML.actions.
                        PlantUMLImportController"/>
            </action>
            <action
                id="ExportAction"
                actionType="generalAction"
                label="PlantUML..."
                tooltip="PlantUML..."
                style="normal"
                icon="icons/puml-icon.png"
                ribbonPath="Project/Export/XML">
                <actionController
                    class="plugins.plantUML.actions.
                        PlantUMLExportController"/>
            </action>
        </actionSet>
    </actionSets>
</plugin>
```

Listing 3.2: PlantUML plugin.xml

3.2 Το μοντέλο του PlantUML

Για την υλοποίηση της λειτουργίας εισαγωγής (import) του plugin, απαιτείται σαφώς η επεξεργασία αρχείων PlantUML με σκοπό την αναπαράσταση των υποκείμενων μοντέλων που περιγράφουν στο κείμενο. Αφού τα μοντέλα ανακτηθούν, μπορούν στη συνέχεια να αναδημιουργηθούν στο Visual Paradigm, αξιοποιώντας το Open API.

Η προφανής προσέγγιση είναι η αναζήτηση ενός έτοιμου αναλυτή κώδικα (parser) για τη γλώσσα PlantUML ή η ανάπτυξη ενός αν δεν υπάρχει διαθέσιμος.

Ωστόσο, διαπιστώθηκε ότι δεν υφίσταται κάποιος πλήρης και αυτόνομος αναλυτής για τη σύνταξη του. Επιπλέον, η δημιουργία ενός νέου αναλυτή κρίθηκε μη εφικτή, κυρίως λόγω της απουσίας τυπικής γραμματικής (grammar), ζήτημα που θα αναλυθεί στην υποενότητα 3.2.1.

Ως εκ τούτου, η λύση που υιοθετήθηκε είναι η αξιοποίηση του ανοιχτού κώδικα του PlantUML ως βιβλιοθήκη (JAR), επιτρέποντας στο ίδιο το PlantUML να διαχειρίζεται την ε-

[2]Πηγή: https://www.visual-paradigm.com/support/documents/vpuserguide/124/254/7040_implementing.html

πεξεργασία του κειμένου. Μέσω αυτής της προσέγγισης, η ανάλυση του κώδικα πραγματοποιείται από τον εσωτερικό μηχανισμό του PlantUML, και στη συνέχεια τα απαραίτητα δεδομένα εξάγονται από την εσωτερική αναπαράσταση των μοντέλων του, προκειμένου τελικά να δομηθούν τα αντίστοιχα μοντέλα στο Visual Paradigm.

3.2.1 Έρευνα για μια PlantUML γραμματική

Κατά τη διάρκεια της ερευνητικής φάσης, πραγματοποιήθηκε εκτενής αναζήτηση για την εύρεση υφιστάμενων αναλυτών (parsers) που θα μπορούσαν να επεξεργαστούν τη σύνταξη του PlantUML. Εξετάστηκαν δημόσια αποθετήρια σε πλατφόρμες όπως το GitHub και το GitLab, καθώς και η επίσημη ιστοσελίδα και ο διαθέσιμος πηγαίος κώδικας του PlantUML. Παρά την αξιολόγηση πολλών έργων, **κανένα δεν αποδείχθηκε κατάλληλο** για τις ανάγκες της συγκεκριμένης υλοποίησης.

Ο βασικός λόγος είναι το γεγονός ότι όλοι οι parsers που εντοπίστηκαν βασίζονται σε μια τυπική γραμματική (grammar) για τη σύνταξη του PlantUML. Ωστόσο, η ίδια η PlantUML δεν διαθέτει επισήμως ορισμένη γραμματική. Η ανάλυση της γλώσσας πραγματοποιείται απευθείας στον πηγαίο της κώδικα, μέσω ενός συνδυασμού ευρετικών μεθόδων και κανονικών εκφράσεων (regular expressions), αντί για πλήρη γραμματικό αναλυτή. Κατά συνέπεια, οι αναλυτές που έχουν δημιουργηθεί από τρίτους πάσχουν από σοβαρούς περιορισμούς, καθώς η γραμματική που χρησιμοποιούν είναι συχνά ελλιπής και προσαρμοσμένη σε συγκεκριμένες ανάγκες , αντί να καλύπτει το σύνολο της γλώσσας.

Ακόμη και μεταξύ των πιο προηγμένων έργων που εντοπίστηκαν [11], κανένα δεν προσέφερε πλήρη υποστήριξη για το PlantUML. Παρά τη σημαντική προσπάθεια, οι υλοποιήσεις δεν καταφερνουν να καλύψουν όλους τους τύπους διαγραμμάτων και τις πιθανές παραλλαγές στη σύνταξη. Επιπλέον, όπως αναφέρεται σε ένα ζήτημα (issue) που δημοσιεύτηκε στο επίσημο repository του PlantUML[16], οι προγραμματιστές του έργου απάντησαν ότι μια επίσημη γραμματική δεν υπάρχει και ότι η κατασκευή της θα ήταν εξαιρετικά δύσκολη, ενώ ένα φιλόδοξο έργο με σκοπό τον ορισμό γραμματικής ετοιμάζεται προς δημιουργία. Παρόλο που η σύνταξη του PlantUML είναι σχετικά απλή και εύκολα κατανοητή από τον άνθρωπο, **η ευελιξία της οδηγεί σε πολυάριθμες ιδιαιτερότητες, εναλλακτικές εντολές και επεκτάσεις**, καθιστώντας την κατασκευή μίας πλήρους και διατηρήσιμης γραμματικής μια ιδιαίτερα απαιτητική.

Η χρήση οποιουδήποτε υπάρχοντος αναλυτή θα περιόριζε σημαντικά τη λειτουργικότητα εισαγωγής του προσθέτου, καθώς θα υποστήριζε μόνο ένα περιορισμένο υποσύνολο της σύνταξης του PlantUML. Αυτό θα σήμαινε ότι **έγκυρα διαγράμματα θα μπορούσαν να απορρίπτονται λανθασμένα**, λόγω των περιορισμών του parser. Επιπλέον, στο πλαίσιο αυτής της υλοποίησης, όπου χρησιμοποιούνται Large Language Models (LLMs) για τη δημιουργία αρχείων PlantUML, ένας τέτοιος περιορισμός θα αποτελούσε σοβαρό μειονέκτημα. Τα LLMs συχνά παράγουν συντακτικά σφάλματα, ακόμη και όταν ακολουθούν το πλήρες συντακτικό του PlantUML. **Η επιβολή επιπλέον περιορισμών με βάση**

μια ελλιπή γραμματική θα περιέπλεκε περαιτέρω τη διαδικασία.

Με βάση αυτά τα ευρήματα, η κατασκευή μιας πλήρους γραμματικής για το PlantUML κρίθηκε ανέφικτη στο πλαίσιο της παρούσες εργασίας. Υφιστάμενες προσπάθειες προς αυτήν την κατεύθυνση βρίσκονται σε εξέλιξη εδώ και χρόνια και παραμένουν ημιτελείς. Συνεπώς, αντί της δημιουργίας ενός νέου συντακτικού αναλυτή, ακολουθήσαμε διαφορετική προσέγγιση, η οποία βασίζεται στην αξιοποίηση του ίδιου του πηγαίου κώδικα του PlantUML για την εξαγωγή και επεξεργασία των απαραίτητων πληροφοριών.

3.2.2 Η δομή της βάσης κώδικα PlantUML

Ο πηγαίος κώδικας του PlantUML δεν έχει σχεδιαστεί ώστε να λειτουργεί ως επαναχρησιμοποιήσιμη βιβλιοθήκη. Ο κύριος σκοπός του είναι η απόδοση διαγραμμάτων σε διάφορες μορφές εικόνας, και όχι η παροχή μιας δομημένης διεπαφής (API) για προγραμματιστική πρόσβαση. Ως εκ τούτου, δεν διαθέτει επίσημη τεκμηρίωση πέρα από την Javadoc, ενώ ορισμένες εσωτερικές υλοποιήσεις του δεν είναι κατάλληλες για απευθείας χρήση σε εξωτερικά έργα, όπως η παρούσα εργασία. Αυτός είναι και ο κύριος λόγος για τον οποίο εξετάστηκε ενδελεχώς η εναλλακτική του parser.

Ένα βασικό στοιχείο της φιλοσοφίας σχεδιασμού του PlantUML είναι ότι δίνει **προτεραιότητα στην οπτική αναπαράσταση των διαγραμμάτων**, αντί για την εξαγωγή δομημένων δεδομένων. Το σύστημα είναι βελτιστοποιημένο για τη δημιουργία εικόνων διαγραμμάτων και, επομένως, οποιαδήποτε πληροφορία δεν επηρεάζει άμεσα την οπτική απόδοση ενδέχεται να μην αποθηκεύεται ή να μην επεξεργάζεται με συστηματικό τρόπο. Αυτή η αρχιτεκτονική επιλογή εισάγει περιορισμούς στην εξαγωγή σημασιολογικών δεδομένων, καθώς η ανάλυση πληροφοριών που δεν σχετίζονται με την απόδοση των διαγραμμάτων θεωρείται μη αναγκαία.

Για παράδειγμα, σε ένα διάγραμμα κλάσεων μια τυπική δήλωση μεθόδου όπως `operation(int parameter) : void` αποδίδεται απλώς ως κείμενο στο τελικό διάγραμμα. Δεδομένου ότι το PlantUML δεν έχει λόγο να επεξεργαστεί ή να αποθηκεύσει τη δήλωση πέρα από την οπτική της εμφάνιση, δεν αναλύει περαιτέρω τη σημασία της—δεν αναγνωρίζει, δηλαδή, ότι το `int` είναι ο τύπος της παραμέτρου ή ότι το `void` είναι ο τύπος επιστροφής της μεθόδου. Αυτή η συμπεριφορά είναι σκόπιμη, καθώς το σύστημα αντιμετωπίζει τέτοιες πληροφορίες απλώς ως μορφοποιημένο κείμενο για απόδοση, και όχι ως δομημένα μεταδεδομένα.

Αν και αυτή η προσέγγιση είναι επαρκής για τη δημιουργία διαγραμμάτων, προκαλεί προβλήματα στην επαναχρησιμοποίηση και επέκταση του κώδικα ιδιαίτερα όταν απαιτείται προσαρμογή του για άλλες εφαρμογές, όπως η εξαγωγή διαγραμμάτων σε άλλες UML πλατφόρμες ή η μετατροπή τους σε μορφή XMI. Η απουσία δομημένης ανάλυσης των στοιχείων του διαγράμματος σημαίνει ότι απαιτείται επιπλέον επεξεργασία για την εξαγωγή ουσιαστικών πληροφοριών, καθιστώντας την ενοποίηση του PlantUML με άλλα εργαλεία πιο περίπλοκη.

Παρά τους εγγενείς αυτούς περιορισμούς, ο πηγαίος κώδικας του PlantUML προσαρμόστη-

κε με επιτυχία για τις ανάγκες της παρούσας υλοποίησης, επιτρέποντας την εξαγωγή των απαραίτητων μοντέλων και τη μετατροπή τους σε κατάλληλη μορφή για περαιτέρω επεξεργασία.

Αρχιτεκτονική του PlantUML

Το PlantUML είναι ένα εκτενές έργο με πλήθος κλάσεων και σύνθετη αρχιτεκτονική. Εδώ, θα επιχειρήσουμε να αναλύσουμε **ορισμένα από τα βασικά στοιχεία του από αυτά αξιοποιήθηκαν** στο πλαίσιο της παρούσας υλοποίησης. Στον πίνακα 3.4 παρουσιάζεται η αντιστοίχιση των τύπων διαγραμμάτων με τις αντίστοιχες κλάσεις που χρησιμοποιούνται για την αποθήκευση των πληροφοριών τους, αφού ολοκληρωθεί η ανάλυση του κώδικα PlantUML, ενώ στα σχήματα .

Ένα ιδιαίτερο χαρακτηριστικό του PlantUML είναι ότι δεν διακρίνει αυστηρά μεταξύ των διαγραμμάτων Component, Deployment και Use Case. Αντίθετα, τα στοιχεία που συνήθως ανήκουν σε κάποιον από αυτούς τους τύπους μπορούν να συνυπάρχουν μέσα σε ένα ενιαίο Description Diagram. Αυτό σημαίνει ότι ένα τέτοιο διάγραμμα μπορεί να περιλαμβάνει actors, components, nodes, use cases και άλλα στοιχεία ταυτόχρονα. Αυτή η ενιαία προσέγγιση αντικατοπτρίζεται και στον πηγαίο κώδικα, όπου τα σχετικά αντικείμενα αναπαρίστανται ως DescriptionDiagram αντί να κατηγοριοποιούνται σε ξεχωριστές κλάσεις.

Τύπος Διαγράμματος	**Κλάση PlantUML**	**Σχετική υπερκλάση**
Class Diagram	ClassDiagram	CucaDiagram
State Diagram	StateDiagram	CucaDiagram
Activity Diagram	ActivityDiagram3	UmlDiagram
Sequence Diagram	SequenceDiagram	UmlDiagram
Component Diagram	DescriptionDiagram	CucaDiagram
Deployment Diagram	DescriptionDiagram	CucaDiagram
Use Case Diagram	DescriptionDiagram	CucaDiagram

Πίνακας 3.4: Τύποι διαγραμμάτων και οι αντίστοιχες κλάσεις τους στο PlantUML

Η περισσότερη χρήσιμη πληροφορία που είναι αποθηκευμένη σε ένα άπο τα αντικείμενα της δεύτερης στήλης του πίνακα 3.4 βρίσκονται σε κληροδοτημένα στοιχεία από τις υπερκλάσεις τους και αποσπώνται μέσω public μεθόδων τους. Για αυτό, έχουν συμπεριληφθεί και οι σχετικές υπερκλάσεις τους.

CucaDiagram Όλα τα διαγράμματα UML εκτός από το Activity Diagram και το Sequence Diagram επεκτείνουν την κλάση AbstractEntityDiagram, η οποία με τη σειρά της επεκτείνει τη κλάση **CucaDiagram (CUCA = Class UseCase Activity)** όπως φαίνεται στα σχήματα 3.3 και 3.4. Αυτά τα διαγράμματα μοιράζονται μια κοινή θεμελιώδη δομή, όπου κάθε βασικό διαγραμματικό στοιχείο—όπως μια κλάση, μια κατάσταση σε ένα διάγραμμα καταστάσεων ή μια περίπτωση χρήσης—αντιπροσωπεύεται ως αντικείμενο τύπου

Entity. Η κλάση Entity λειτουργεί ως γενική αναπαράσταση για όλους τους τύπους στοιχείων αυτών των διαγραμμάτων, ανεξάρτητα από το αν πρόκειται για στοιχεία που μπορούν να περιέχουν άλλα στοιχεία στο εσωτερικό τους (Group Types), όπως τα πακέτα και τα components, ή για αυτόνομα στοιχεία (Leaf Types) ¨φύλλα¨.

Οι τρεις βασικές μέθοδοι που χρησιμοποιούνται για την απόσπαση όλων των στοιχείων ενός CucaDiagram είναι οι εξής:

- leafs(): Επιστρέφει όλα τα στοιχεία φύλλα του διαγράμματος, δηλαδή τα μεμονωμένα διαγραμματικά στοιχεία που δεν περιέχουν άλλα.
- groups(): Επιστρέφει όλα τα ομαδικά στοιχεία του διαγράμματος, δηλαδή εκείνα που μπορούν να περιέχουν άλλα στοιχεία. Για την απόσπαση των στοιχείων που περιλαμβάνονται σε ένα group, καλείται αναδρομικά η μέθοδος leafs() του αντίστοιχου group entity.
- getLinks(): Επιστρέφει όλους τους συνδέσμους (links) του διαγράμματος. Από τα αντικείμενα τύπου Link που επιστρέφονται, μπορούμε να αξιοποιήσουμε τις διαθέσιμες μεθόδους της αντίστοιχης κλάσης για να προσδιορίσουμε τον τύπο τους, καθώς και τα διαγραμματικά στοιχεία (entities) που συνδέουν μεταξύ τους.

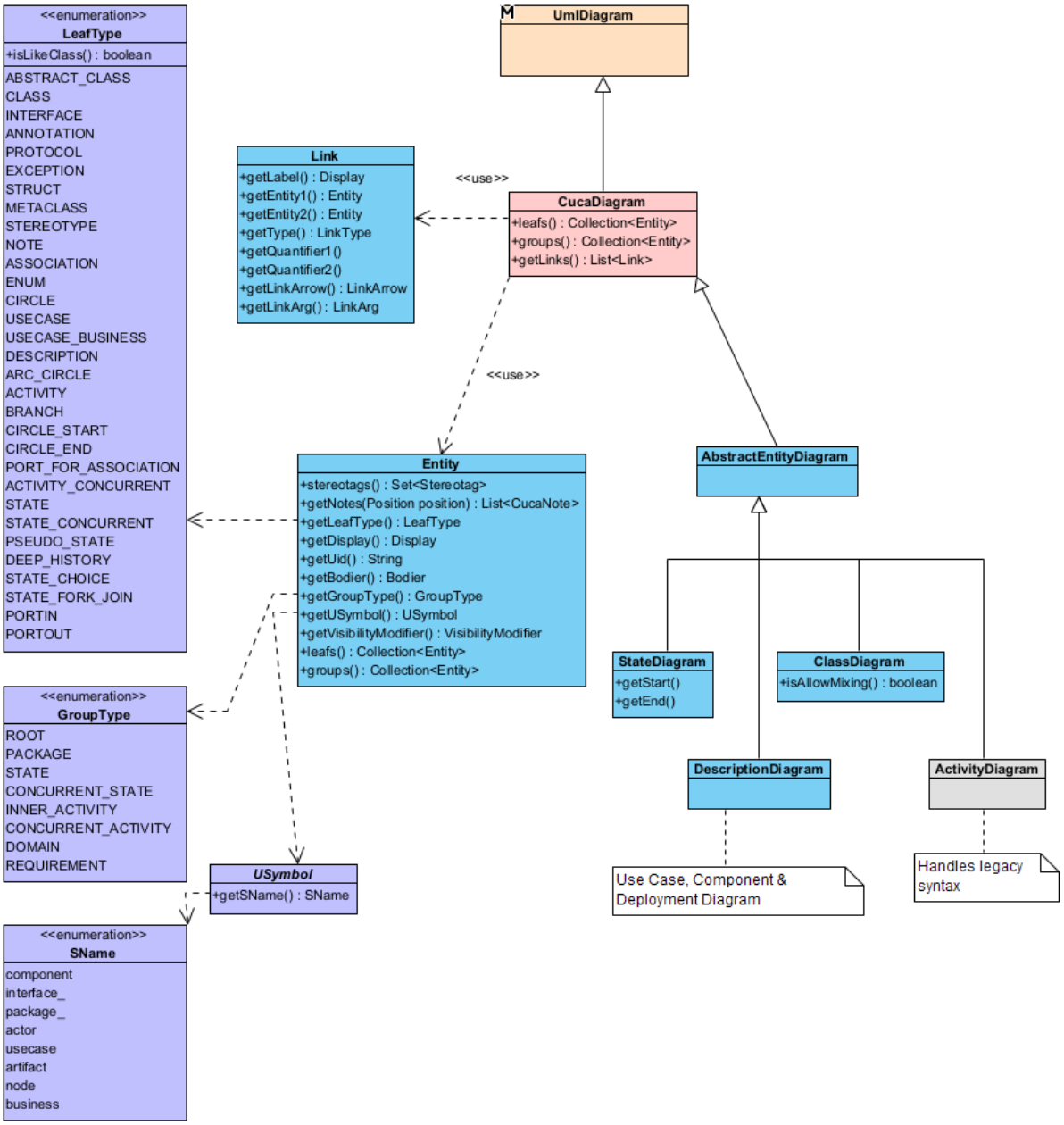
Σχεδόν όλα τα στοιχεία αυτών των διαγραμμάτων διαθέτουν έναν τύπο LeafType ή GroupType, ο οποίος προέρχεται από τα αντίστοιχα enums LeafType και GroupType. Ωστόσο, για άγνωστο λόγο, ορισμένα στοιχεία των Description Diagrams αποτελούν εξαίρεση και δεν ακολουθούν αυτή τη λογική.

Για τα περισσότερα στοιχεία, ο προσδιορισμός του τύπου Entity πραγματοποιείται μέσω των μεθόδων getLeafType() και getGroupType(), οι οποίες επιστρέφουν τις αντίστοιχες πληροφορίες. Ωστόσο, στα στοιχεία των Description Diagrams που δεν διαθέτουν LeafType, η αναγνώριση του τύπου τους βασίστηκε στο SName του USymbol τους. Το SName (Style Name) αποτελεί μια πληροφορία που σχετίζεται με την εμφάνιση του διαγραμματικού στοιχείου και χρησιμοποιείται κυρίως για το rendering κατά την απεικόνισή του.

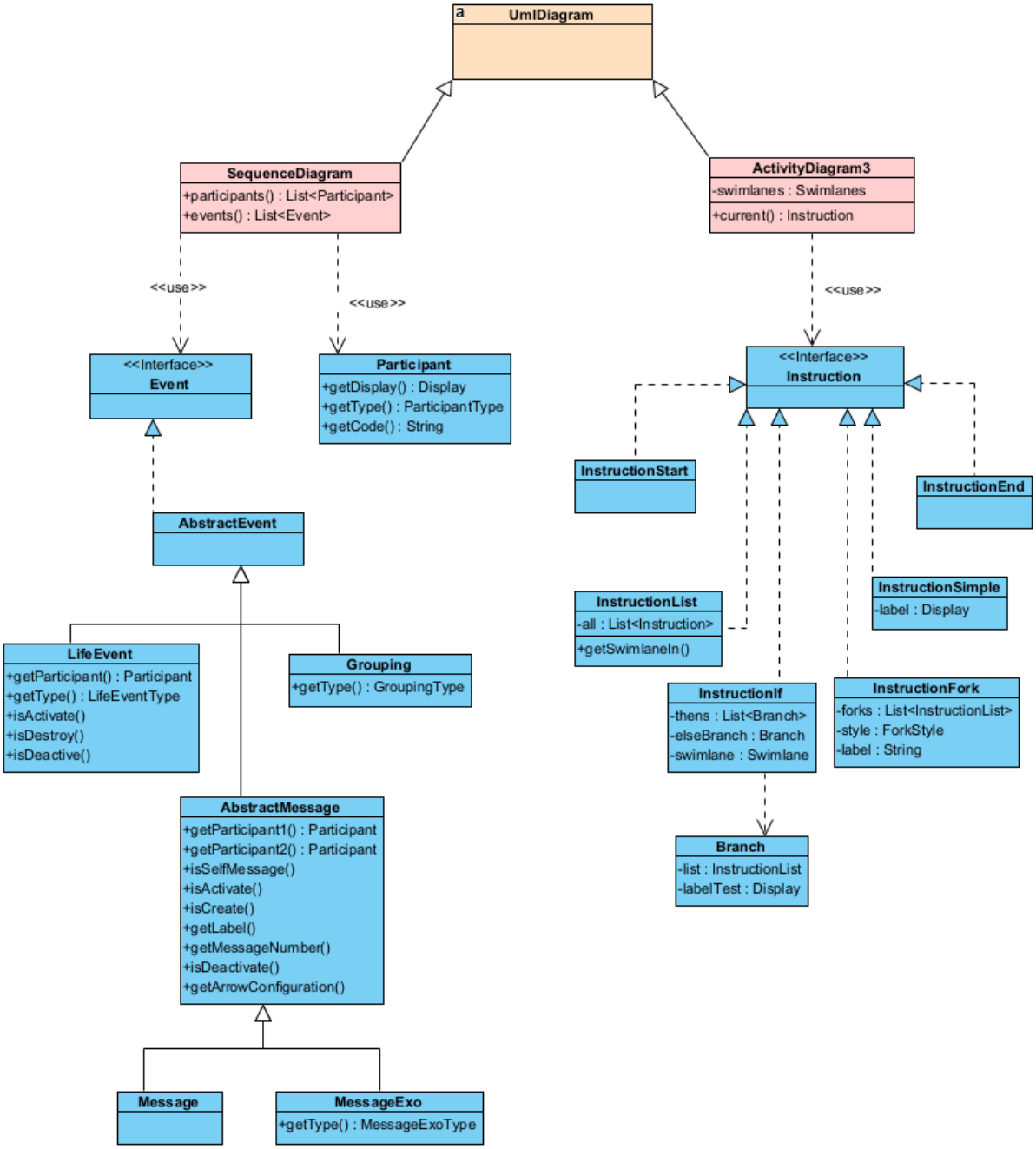
Όλα τα Entityy διαθέτουν SName, όχι μόνο αυτά που εμφανίζονται στο σχήμα 3.3. Στο διάγραμμα αποτυπώνονται μόνο τα στοιχεία που και χρησιμοποιήθηκαν από τον κώδικα κατά την υλοποίηση της παρούσας εργασίας, εφόσον εκείνα δεν διέθεταν LeafType. Η χρήση του SName για την αναγνώριση του τύπου αυτών των στοιχείων αποτελεί workaround, γι' αυτό και δεν υιοθετήθηκε ως γενική προσέγγιση για τους υπόλοιπους τύπους στοιχείων.

SequenceDiagram Λόγω της ριζικά διαφορετικής συνταξης και φιλοσοφίας του διαγράμματος Sequence, η κλάση SequenceDiagram δεν επεκτείνει την AbstractEntityDiagram και δεν αναπαριστά τα στοιχεία του με αντικείμενα Entity. Αντίθετα, όπως φαίνεται και στο διάγραμμα του σχήματος 3.4 τα στοιχεία κατηγοριοποιούνται σε Participants και Events.

Η κλάση Participant αντιστοιχεί σε στοιχεία που, στο Visual Paradigm, αναγνωρίζονται



Σχήμα 3.3: Κώδικας του PlantUML : Διαγράμματα CUCA - διάγραμμα Class



Σχήμα 3.4: Κώδικας του PlantUML : Activity & Sequence - διάγραμμα Class

ως Actor και Lifeline. Αυτή η κλάση χρησιμοποιείται για την αναπαράσταση των βασικών οντοτήτων που συμμετέχουν σε ένα Sequence Diagram.

Η διεπαφή Event περιγράφει όλα τα υπόλοιπα στοιχεία που μπορούν να εμφανιστούν σε ένα διάγραμμα sequence. Η AbstractEvent αποτελεί μια αφηρημένη κλάση που επεκτείνεται από επιμέρους κλάσεις, οι οποίες αναπαριστούν διαφορετικά είδη συμβάντων. Συγκεκριμένα, περιλαμβάνει κλάσεις που περιγράφουν μεταβολές σε lifelines (π.χ. deactivation, activation), όλους τους τύπους μηνυμάτων (Message ή MessageExo για χαμένα μηνύματα), τα οποία κληρονομούν από την AbstractMessage, καθώς και δομές Groupings που χρησιμοποιούνται για την αναπαράσταση των combined fragments. Τα combined fragments περιλαμβάνουν στοιχεία όπως το ref (αναφορά σε άλλο διάγραμμα), το alt/else (εναλλακτικές ροές), το loop και άλλες συνθετικές κατασκευές που επηρεάζουν τη ροή εκτέλεσης του διαγράμματος.

ActivityDiagram3 Υπάρχουν δύο συντάξεις για τη δημιουργία Activity Diagrams στη PlantUML: μια παλαιότερη (legacy syntax) και μια νεότερη, η οποία αποτελεί την προεπιλεγμένη σύνταξη που προβάλλεται στην επίσημη σελίδα της PlantUML. Στο πλαίσιο της παρούσας εργασίας, επιλέχθηκε η υποστήριξη αποκλειστικά της νέας σύνταξης, ωστόσο η απόφαση αυτή συνοδεύτηκε από προκλήσεις στην υλοποίηση, λόγω της δυσκολίας αξιοποίησης του πηγαίου κώδικα της PlantUML.

Η παλαιότερη σύνταξη διαχειρίζεται από την κλάση ActivityDiagram, η οποία επεκτείνει την CucaDiagram, όπως συμβαίνει και με τα υπόλοιπα διαγράμματα, εκτός από τα Sequence. Αυτό σημαίνει ότι η υλοποίηση υποστήριξης για τη legacy syntax θα ήταν σημαντικά πιο απλή, καθώς ενσωματώνεται στη γενικότερη δομή της PlantUML. Αντίθετα, η νέα σύνταξη ακολουθεί μια ανεξάρτητη προσέγγιση, επεκτείνοντας απευθείας την κλάση UmlDiagram και αποθηκεύεται σε αντικείμενα της κλάσης ActivityDiagram3.

Σε ένα ActivityDiagram3, κάθε Activity (ή Action) αναπαρίσταται ως αντικείμενο τύπου Instruction. Η διεπαφή Instruction υλοποιείται από όλους τους τύπους στοιχείων που μπορεί να περιλαμβάνει ένα Activity Diagram. Για παράδειγμα:

- Ο αρχικός κόμβος ενός διαγράμματος υλοποιείται από την κλάση InstructionStart και ο τελικός από την InstructionStop ή InstructionEnd.
- Ένας κόμβος απόφασης αναπαρίσταται ως InstructionIf, ο οποίος περιέχει αντικείμενα τύπου Branch.
- Κάθε Branch περιλαμβάνει μια λίστα InstructionList, η οποία με τη σειρά της αποτελείται από επιμέρους Instruction αντικείμενα.

Η διαδικασία ανάλυσης του ActivityDiagram3 ξεκινά μέσω της δημόσιας μεθόδου current(), η οποία επιστρέφει το πρώτο Instruction, που συνήθως είναι αντικείμενο τύπου InstructionList. Ωστόσο, σε αυτό το σημείο προκύπτουν σημαντικές δυσκολίες, καθώς όλες οι υπόλοιπες μέθοδοι και ιδιότητες (attributes) που είναι απαραίτητες για την εξαγωγή των δεδομένων είναι νιδιωτικές (private).

Συγκεκριμένα, για να αποκτήσουμε πρόσβαση στη λίστα των Instructions που περιέχονται σε ένα InstructionList, απαιτείται η ανάκτηση του πεδίου "all", το οποίο όμως δεν είναι απευθείας προσβάσιμο. Ο μόνος εφικτός τρόπος ήταν η αξιοποίηση μηχανισμών reflection, προκειμένου να αποκτήσουμε πρόσβαση στις μεθόδους και τις μεταβλητές που χρειαζόμασταν. Μέσω αυτής της προσέγγισης κατέστη δυνατή η απόσπαση των απαραίτητων λεπτομερειών για την επεξεργασία του Activity Diagram.

Όμως, λόγω αυτών των περιορισμών, η υποστήριξη των Activity Diagrams μέσω του plugin δεν είναι τόσο ολοκληρωμένη όσο για άλλους τύπους διαγραμμάτων. Η ανάγκη παρέμβασης σε κώδικα που δεν έχει σχεδιαστεί για χρήση από τρίτους καθιστά οποιαδήποτε τροποποίηση δύσκολη και ενέχει υψηλό ρίσκο εμφάνισης σφαλμάτων. Ως αποτέλεσμα, η πλήρης υποστήριξη όλων των δυνατοτήτων των Activity Diagrams δεν ήταν εφικτή, καθώς θα απαιτούσε σημαντικές τροποποιήσεις σε κώδικα που δεν προορίζεται για αυτό.

3.3 Αρχιτεκτονική του plugin

Σύμφωνα με τις απαιτήσεις και πληροφορίες από το Visual Paradigm Open API και τον πηγαίο κώδικα του PlantUML, διαμορφώθηκε η αρχιτεκτονική του προσθέτου που αναπτύχθηκε. Η τελική, απλοποιημένη μορφή της παρουσιάζεται στα διαγράμματα component και class αντίστοιχα στα Σχήματα 3.5 και 3.6.

Η σχεδίαση της αρχιτεκτονικής βασίστηκε στα συστατικά που απαιτεί το Visual Paradigm για τη σωστή ενσωμάτωση οποιουδήποτε plugin. Όπως φαίνεται και στις απεικονίσεις, οι απαραίτητες κλάσεις του προσθέτου υλοποιήθηκαν σύμφωνα με τις προδιαγραφές του Visual Paradigm όπως παρουσιάστηκαν στο σχήμα 3.2, εξασφαλίζοντας τη σωστή διασύνδεση με το υπόλοιπο περιβάλλον του λογισμικού.

Διάγραμμα Component

Ας αναλύσουμε το Component Diagram (Σχήμα 3.5) που αποτυπώνει τη σχεδίαση του plugin.

- Η κλάση PlantUML υλοποιεί την κλάση VPPlugin, η οποία αποτελεί το entry point του προσθέτου και πρέπει να υλοποιηθεί, όπως αναλύθηκε προηγουμένως. Επιπλέον, υλοποιεί το interface VPPluginCommandLineSupport, το οποίο δεν ήταν απαραίτητο, αλλά χρησιμοποιήθηκε για την ανάπτυξη του CLI εργαλείου του προσθέτου.
- Οι κλάσεις PlantUMLExportController και PlantUMLImportController υλοποιούν το VPActionController και είναι υπεύθυνες για την εκκίνηση της λειτουργίας του προσθέτου όταν ο χρήστης επιλέγει τις αντίστοιχες ενέργειες export ή import από το μενού μέσω των κουμπιά αυτά προστέθηκαν στο Visual Paradigm για την επέκταση του υπάρχοντος μενού.

- Το αρχείο plugin.xml περιέχει τις απαραίτητες δηλώσεις για τη δομή και τα αναγνωριστικά του plugin, σύμφωνα με τη δομή που απαιτεί το API, όπως έχει ήδη αναλυθεί.
- Οι export και import controllers αξιοποιούν τις κλάσεις που δημιουργήθηκαν για την λογική των λειτουργιών. Η δομή αυτών των κλάσεων θα παρουσιαστεί συνόπτικά παρακάτω.
- Οι κλάσεις που υλοποιούν τη λογική του Import εξαρτώνται και από τον πηγαίο κώδικα του PlantUML (component "PlantUML JAR") και τον χρησιμοποιούν για την ανάλυση των αρχείων προς εισαγωγή στο Visual Paradigm.
- Όλη η διαδικασία εκτελείται εντός της εφαρμογής του Visual Paradigm, η οποία βασίζεται στη Java.

Για λόγους απλότητας, έχει παραληφθεί να αποτυπωθεί στο διάγραμμα component η **εκτενής χρήση του VP Open API σε όλες τις διεργασίες** των εσωτερικών κλάσεων που υλοποιούν τη λογική.

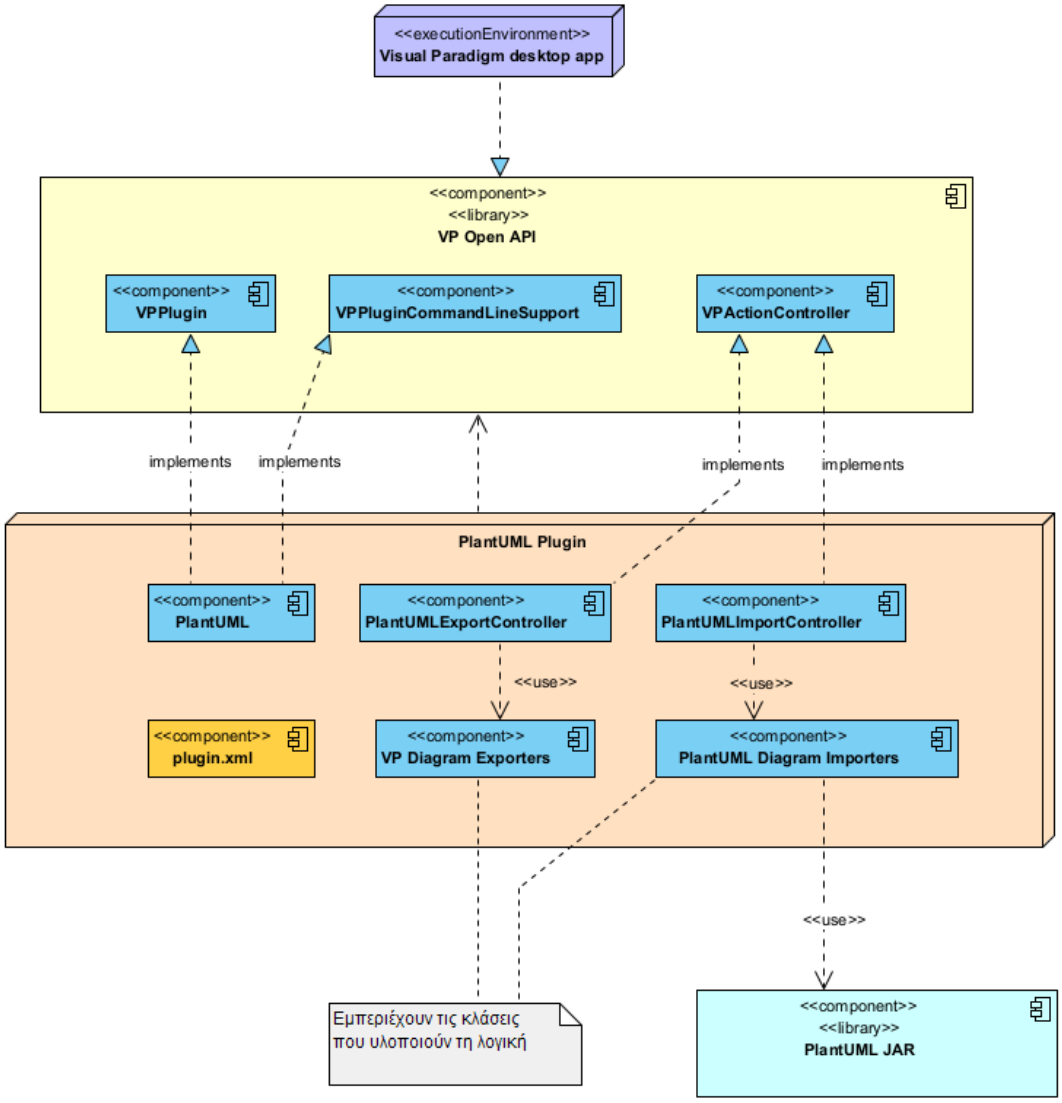
Διάγραμμα Class

Το Class Diagram του Σχήματος 3.6 αποτυπώνει τη δομή και τη λειτουργικότητα των κύριων κλάσεων του plugin.

Όπως και στο Component Diagram, παρατηρούμε τις κλάσεις ActionControllers, οι οποίες υλοποιούν τη μέθοδο performAction(). Αυτή η μέθοδος ενεργοποιείται όταν ο χρήστης επιλέγει μία από τις διαθέσιμες ενέργειες εισαγωγής ή εξαγωγής μέσω του μενού του Visual Paradigm.

Διαδικασία εξαγωγής (Export) Η διαδικασία εξαγωγής διαγραμμάτων ακολουθεί την ακόλουθη ροή εκτέλεσης ξεκινώντας από τον PlantUMLExportController:

- Η μέθοδος performAction() καλεί την κλάση ExportDialogHandler, η οποία αποτελεί υλοποίηση του interface IDialogHandler του VP OpenAPI. Αυτή η κλάση διαχειρίζεται την εμφάνιση ενός διαλόγου επιλογής (popup menu), όπου ο χρήστης καθορίζει ποια διαγράμματα του τρέχοντος έργου (.vpp αρχείο) επιθυμεί να εξάγει σε μορφή PlantUML, καθώς και τον προορισμό αποθήκευσης των παραγόμενων αρχείων.
- Μετά την επιλογή από τον χρήστη, η κλάση ExportDialogHandler καλεί είτε τη μέθοδο export() της κλάσης DiagramExportPipeline, εάν έχει επιλεγεί ένα μόνο διάγραμμα, είτε τη exportDiagramList() για την εξαγωγή πολλαπλών διαγραμμάτων. Σε κάθε περίπτωση, περνά ως παράμετρο τα επιλεγμένα IDiagramUIModel αντικείμενα, τα οποία αντιπροσωπεύουν τα επιλεγμένα διαγράμματα του έργου.
- Η κλάση DiagramExportPipeline αναλαμβάνει τη συνολική διαχείριση της διαδικασίας εξαγωγής, καλώντας τις κατάλληλες μεθόδους extract() των αντίστοιχων υποκλάσεων της DiagramExporter.



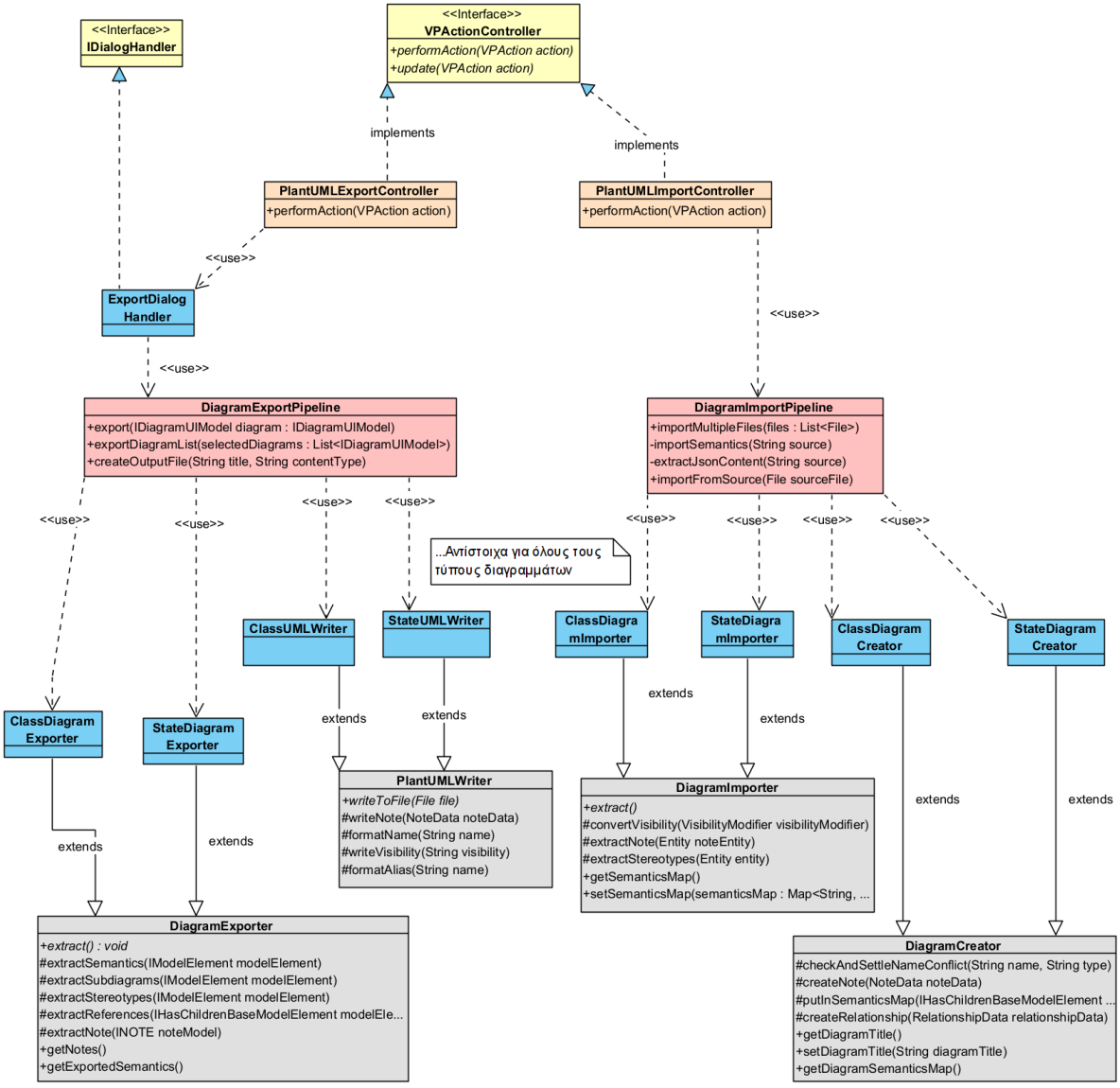
Σχήμα 3.5: Αρχιτεκτονική του PlantUML Plugin - διάγραμμα Component

- Η DiagramExporter αποτελεί την υπερκλάση όλων των εξειδικευμένων εξαγωγέων και περιλαμβάνει βασικές μεθόδους που είναι κοινές για όλα τα είδη διαγραμμάτων.
- Οι μέθοδοι extract() των εξειδικευμένων εξαγωγέων (π.χ. ClassDiagramExporter) υλοποιούν τη λογική απόσπασης των ουσιωδών πληροφοριών από τα διαγράμματα. Οι πληροφορίες αυτές μετατρέπονται σε ελαφρύτερες εσωτερικές αναπαραστάσεις, αντί να διατηρούνται ως πλήρη IModelElements.
- Οι εξαγόμενες πληροφορίες μεταφέρονται από την DiagramExportPipeline στις κλάσεις PlantUMLWriters, οι οποίες είναι υπεύθυνες για τη δημιουργία των τελικών αρχείων.
- Οι εξειδικευμένοι PlantUMLWriters (π.χ. ClassUMLWriter) υλοποιούν την αφηρημένη μέθοδο **writeToFile() και αναλαμβάνουν τη συγγραφή των αρχείων** PlantUML με σωστή σύνταξη, χρησιμοποιώντας τα δεδομένα που παρείχαν οι αντίστοιχοι DiagramExporters.

Με ανάλογη λογική υλοποιήθηκε και η διαδικασία εισαγωγής. Στο διάγραμμα παρουσιάζεται μόνο η αρχιτεκτονική για τα διαγράμματα Class και State, καθώς η δομή είναι παρόμοια και για τους υπόλοιπους τύπους διαγραμμάτων.

Διαδικασία εισαγωγής (Import) Για τη διαδικασία εισαγωγής διαγραμμάτων ακολουθείται η παρακάτω ροή:

- Η μέθοδος performAction() της κλάσης PlantUMLImportController εμφανίζει έναν διάλογο επιλογής αρχείου (file chooser popup), μέσω του οποίου ο χρήστης καθορίζει το αρχείο κειμένου ή τον φάκελο που περιέχει αρχεία PlantUML προς εισαγωγή στο Visual Paradigm.
- Ανάλογα με την επιλογή του χρήστη, η performAction() καλεί είτε τη μέθοδο importMultipleFiles() είτε την importFromSource() της κλάσης DiagramImportPipeline.
- Η κλάση DiagramImportPipeline είναι υπεύθυνη για την ανάθεση της επεξεργασίας στα κατάλληλα υποσυστήματα. Συγκεκριμένα, καλεί τη μέθοδο extract(), η οποία υλοποιείται από εξειδικευμένους εισαγωγείς (DiagramImporters) που επεκτείνουν την υπερκλάση DiagramImporter.
- Η μέθοδος extract() **αξιοποιεί κλάσεις από τον πηγαίο κώδικα του PlantUML** για την ανάλυση των αρχείων και την απόσπαση των απαραίτητων πληροφοριών που περιγράφουν τα υποκείμενα μοντέλα.
- Τελικά, η DiagramImportPipeline μεταφέρει τις πληροφορίες που εξήχθησαν στις κλάσεις DiagramCreators, οι οποίες, χρησιμοποιώντας το VP OpenAPI, κατασκευάζουν τα αντίστοιχα διαγράμματα στο περιβάλλον του Visual Paradigm.



Σχήμα 3.6: Αρχιτεκτονική του PlantUML Plugin - διάγραμμα Class

Κεφάλαιο 4

Το Plugin αμφίδρομης μετατροπής

Στα προηγούμενα κεφάλαια έγινε αναφορά στη λειτουργικότητα του Visual Paradigm plugin που αναπτύχθηκε στο πλαίσιο της παρούσας εργασίας, το οποίο κάνει δυνατή τη μετατροπή αρχείων PlantUML σε διαγράμματα Visual Paradigm και το αντίστροφο. Ωστόσο, μέχρι στιγμής η περιγραφή του ήταν συνοπτική και επικεντρωμένη στη γενική λειτουργία του.

Στο παρόν κεφάλαιο, πραγματοποιείται μια αναλυτική παρουσίαση του plugin, δείχνοντας τις επιλογές και τις παραδοχές που έγιναν κατά την ανάπτυξή του. Επιπλέον, παρουσιάζονται παραδείγματα χρήσης, καθώς και οι περιορισμοί και οι αδυναμίες του.

4.1 Στόχοι

Η σχεδίαση του PlantUML plugin βασίστηκε σε στόχους οι οποίοι διαμορφώθηκαν με γνώμονα τη ρεαλιστική υλοποίηση της μετατροπής μεταξύ PlantUML και Visual Paradigm. Οι βασικοί στόχοι συνοψίζονται ακολουθως:

- **Αμφίδρομη μετατροπή όλων των βασικών UML διαγραμμάτων τύπου: Class, Use Case, State, Activity, Sequence, Component και Deployment**
- **Προτεραιοποίηση δομών και χαρακτηριστικών που μπορούν να παραχθούν απο γλωσσικά μοντέλα**
  Δεδομένου ότι η αυτόματη παραγωγή διαγραμμάτων μέσω γλωσσικών μοντέλων αποτέλεσε το κύριο κίνητρο για την ανάπτυξη του εργαλείου, δόθηκε προτεραιότητα σε PlantUML που μπορούν ρεαλιστικά να παραχθούν από μοντέλα.
- **Παράκαμψη περιορισμών του PlantUML κατά την εξαγωγή**
  Το PlantUML υποστηρίζει περιορισμένο εύρος χαρακτηριστικών UML σε σύγκριση με το Visual Paradigm, γεγονός που θα μπορούσε να οδηγήσει σε απώλειες πληροφορίας κατά τη μετατροπή. Για την αντιμετώπιση αυτού του ζητήματος, εφαρμόστηκαν

workarounds, τα οποία, αν και δεν διασφαλίζουν απόλυτη αντιστοίχιση 1:1 μεταξύ των διαγραμματων, ελαχιστοποιούν τις σημαντικές απώλειες σημαντικού περιεχομένου

- **Μερική διατήρηση των semantics του Visual Paradigm**
  Το Visual Paradigm αποθηκεύει πληροφορίες που δεν είναι αναπαραστάσιμες σε μέσω του PlantUML, όπως περιγραφές μοντέλων, υποδιαγράμματα και αναφορές (τα "semantics"). Για τη διατήρηση αυτής της πληροφορίας, υιοθετήθηκε μια δομημένη προσέγγιση, ώστε να είναι δυνατή η ανακτήση αυτών των δεδομένων.
- **Αυτόματη διαχείριση της διάταξης των διαγραμμάτων (layout)**
  Η πιστή αντιγραφή του διαγράμματος ως προς τη διάταξη των στοιχείων του δεν τέθηκε προτεραιότητα, καθώς το PlantUML βασίζεται σε αυτόματη διάταξη (auto layout), η οποία επιδέχεται μόνο περιορισμένο έλεγχο. Επίσης κατά την εισαγωγή (import), δεν επιχειρήθηκε η διατήρηση του αρχικού layout των διαγραμμάτων, καθώς η βελτιστοποίηση της θέσης των στοιχείων στο Visual Paradigm αποτελεί ιδιαίτερα περίπλοκο πρόβλημα και αντιμετωπίστηκαν περιορισμοί προγραμματιστικά. Αντί αυτού, χρησιμοποιήθηκαν αυτόματα layouts, τα οποία, αν και δεν ανταποκρίνονται πάντα σωστά, μπορούν να τροποποιηθούν από τον χρήστη όπου κρίνεται απαραίτητο.
- **Υποστήριξη ατελών μετατροπών με προειδοποιήσεις (warnings)**
  Η ύπαρξη ασυμβατοτήτων μεταξύ εργαλείων UML είναι αναμενόμενη και καθίσταται ιδιαίτερα έντονη όταν συγκρίνονται δύο τόσο διαφορετικά όπως το Visual Paradigm και το PlantUML. Για αυτό, το εργαλείο επιτρέπει μερικές ή ατελείς μετατροπές, συνοδευόμενες από warnings, όπου αυτό γίνεται. Σε περιπτώσεις σοβαρής ασυμβατότητας, το εργαλείο απορρίπτει τη μετατροπή του διαγράμματος, διασφαλίζοντας ότι το τελικό αποτέλεσμα δεν θα οδηγήσει σε πολύ λανθασμένες αναπαραστάσεις.

4.2 Περιγραφή

Ακολουθεί η περιγραφή των εισόδων και εξόδων των δύο λειτουργιών του plugin (import from PlantUML και export to PlantUML). Και οι δύο λειτουργίες αφορούν τύπους διαγραμμάτων Class, Use Case, State, Activity, Sequence, Component και Deployment.

4.2.1 Import

Είσοδος

Είσοδος για τη λειτουργία import μπορεί να είναι ένα διάγραμμα PlantUML ή ένας φάκελος που περιέχει PlantUML αρχεία σε μορφή κειμένου και επεκτάσεις .txt, .plantuml, .puml. Προαιρετικά, μπορεί ο φάκελος να περιέχει ένα αρχείο JSON σε έγκυρη μορφή όπως θα περιγραφεί στην αντίστοιχη υποενότητα το οποίο περιέχει πληροφορία semantics για τα μοντέλα που εμφανίζονται στα διαγράμματα. Συμφωνα με αυτό το αρχείο θα εμπλουτιστεί σημασιολογικά το Visual Paradigm project.

Έξοδος

Αποτέλεσμα της επιτυχούς εισαγωγής στο Visual Paradigm είναι η εμφάνιση των διαγραμμάτων στο περιβάλλον του, με τίτλους που αντιστοιχούν στον τίτλο των αρχείων. Σε περίπτωση που κατά την εισαγωγή βρέθηκαν συγκρούσεις ονομάτων, τα μοντέλα που προκάλεσαν τη σύγκρουση κατά την εισαγωγή τους διατηρούν το όνομα τους και μετονομάζονται τα ήδη υπάρχοντα στο έργο. Για παράδειγμα, αν στο μοντέλο υπήρχε ήδη η κλάση `"demo_class"` και εισήχθηκε κλάση ονόματος `"demo_class"`, τότε η πρώτη μετονομάζεται σε `"demo_class_renamed_on`
Σε αυτή τη περίπτωση, σε ένα pop-up μήνυμα γνωστοποιούνται στο χρήστη οι μετονομασίες που έγιναν. Αν η μετονομασία δεν ήταν επιθυμητή, μπορεί να κλείσει το έργο χωρίς αποθήκευση για να απορριφθούν οι αλλαγές πιο εύκολα αν ήταν πολλές.

Οποιοδήποτε άλλο σφάλμα ή προειδοποίηση εμφανίζεται επίσης σε pop-up μήνυμα, και, για λόγους διατήρησης αυτής της πληροφορίας, εμφανίζεται και στα plugin logs της εφαρμογής του Visual Paradigm.

4.2.2 Export

Είσοδος

Είσοδος για τη λειτουργία export είναι, μέσω επιλογής από γραφικό μενού το οποίο προστέθηκε στην εφαρμογή, ένα ή περισσότερα διαγράμματα από αυτά που υποστηρίζονται. Από το ίδιο μενού, επιλέγεται και ο επιθυμητός φάκελος εξόδου στον οποίο θα γραφτούν τα διαγράμματα.

Έξοδος

Μετά την επιτυχή εξαγωγή (export) των επιλεγμένων διαγραμμάτων (ή όσων από αυτά ήταν έγκυρα προς μετατροπή), στο φάκελο εξόδου θα εμφανιστούν .txt αρχεία με ονόματα τους τίτλους των διαγραμμάτων. Θα δημιουργηθεί επίσης ένα json αρχείο (σε μορφή που μπορεί να διαβαστεί και αναπαρασταθεί από το PlantUML) όπως θα περιγραφεί σε επόμενη υποενότητα, που περιέχει όλη την επιπρόσθετη πληροφορία (semantics) που δε μπορεί να αποδοθεί σε PlantUML. Αυτό, αναλόγως τη περίπτωση χρήσης, μπορεί να αγνοηθεί ή να διαγραφεί αν δεν εξυπηρετεί κάποιο σκοπό για το χρήστη.

4.3 Παραδοχές και περιορισμοί

Κατά την ανάπτυξη προέκυψαν ζητήματα που επιλύθηκαν με συμβιβασμούς, παραδοχές και περιορισμούς για τις μετατροπές.

4.3.1 Γενικά

Κάποιοι γενικοί συμβιβασμοί είναι οι εξής:

- Η μετατροπή είναι (κατά περίπτωση) ασύμμετρη, εννοώντας πως δεν υποστηρίζονται σε όλα τα διαγράμματα οι ίδιες δομές ή δεν αποδίδονται με τον ίδιο τρόπο. Αυτό σημαίνει ότι, αν ένα διάγραμμα εξαχθεί με export από το Visual Paradigm σε μορφή PlantUML και στη συνέχεια εισαχθεί εκ νέου με import στο Visual Paradigm, το αρχικό διάγραμμα μπορεί να διαφέρει από το νέο.
- Οι ακμές (σύνδεσμοι) μπορεί να μην είναι οι ορθότεροι σύμφωνα με τις συμβάσεις και τη σημασιολογία που ακολουθεί το κάθε εργαλείο. Το PlantUML δεν έχει περιορισμούς στο τύπο συνδέσμου μεταξύ δύο οποιονδήποτε στοιχείων διαγραμμάτος. Η σημασιολογία τους δεν είναι σαφώς ορισμένη και τέτοια αυστηρότητα θα ήταν αντίθετη με την αρχή της ευελιξίας του PlantUML. Δεν ίσχύει ωστόσο το ίδιο για το Visual Paradigm το οποίο αντιστοιχεί ρητά την εμφάνιση ενός συνδέσμου με το σκοπό του (π.χ. Generalization, Anchor) και δεν επιτρέπει σημασιολογικά παράλογους συνδέσμους μεταξύ στοιχείων (π.χ. Actor -(Generalization)|> Use Case) . Αυτό μας οδήγησε σε προσπάθεια οπτικής αντιστοίχησης των τύπων ακμών, πράγμα που κάποιες φορές οδηγεί σε παράνομες (σπάνια) ή ατελείς αναπαραστάσεις.
- Ίσχύουν οι περιορισμοί διάταξης που συζητήθηκαν στους στόχους, ενώ ιδιοτροπίες του Visual Paradigm OpenAPI γέννησαν και άλλους περιορισμούς σε επίπεδο μοντέλων λόγω διαγραμματικών προβλημάτων και προβλημάτων διάταξης.
- Εννοούνται περιορισμοί που έχουν να κάνουν με την έλλειψη αντίστοιχων δομών. Για παράδειγμα, η έννοια του Activity που περιέχει Actions όπως γίνεται στο Visual Paradigm δε βρίσκει αντιστοιχία στο PlantUML, για το οποίο όλα είναι Actions.

Συγκεκριμένα για κάποια διαγράμματα οι συμβιβασμοί ήταν περισσότεροι, λόγω αυξημένων ασυμβατότητων και περιορισμών από τις προγραμματιστικές διεπαφές των εργαλείων. Εδώ θα συζητήσουμε μερικές από τις επιλογές που έγιναν ως αποτέλεσμα αυτών.

4.3.2 Component / Deployment Diagram

Components και Interfaces Στο περιβάλλον του Visual Paradigm, τα μοντέλα τύπου component έχουν παρόμοια χαρακτηριστικά με τις κλάσεις. Μέθοδοι (operations) και ιδιότητες (attributes) είναι βασικά στοιχεία που μπορούν να εισαχθούν σε ένα component. Παρόμοια, όπως και στα διαγράμματα κλάσεων, οι διεπαφές είναι μοντέλα τύπου κλάσης με στερεότυπο «interface» που επίσης δέχονται μεθόδους και ιδιότητες.

Παρά το γεγονός ότι δε φαίνονται σε μία στατική εικόνα του διαγράμματος, η περιγραφή των μοντέλων (specification) περιέχει τη πληροφορία. Αντίθετα, το PlantUML δεν επιτρέπει σε components ή interfaces να έχουν μεθόδους και ιδιότητες. Με σκοπό να ξεπεράσουμε αυτό το περιορισμό κατά το export, επιλέγουμε μία πιο χαλαρή μετάφραση του διαγράμματος με αποτέλεσμα μία μίξη τύπων διαγραμμάτων.

Στο PlantUML, η παράμετρος `allowmixing`, η οποία δηλώνεται στην αρχή του αρχείου, επιτρέπει την ανάμειξη διαφορετικών τύπων διαγραμμάτων. Αξιοποιώντας αυτή τη δυνατότητα,

επιλέγουμε μια προσαρμοσμένη μετατροπή των component interfaces ώστε να μπορούν να περιέχουν μεθόδους.

Συγκεκριμένα, τα interface μοντέλα ενός component diagram (που αναπαρίστανται με κύκλο) μετατρέπονται σε δηλώσεις κλάσεων με στερεότυπο «interface» στο PlantUML, επιτρέποντας έτσι την προσθήκη μεθόδων.

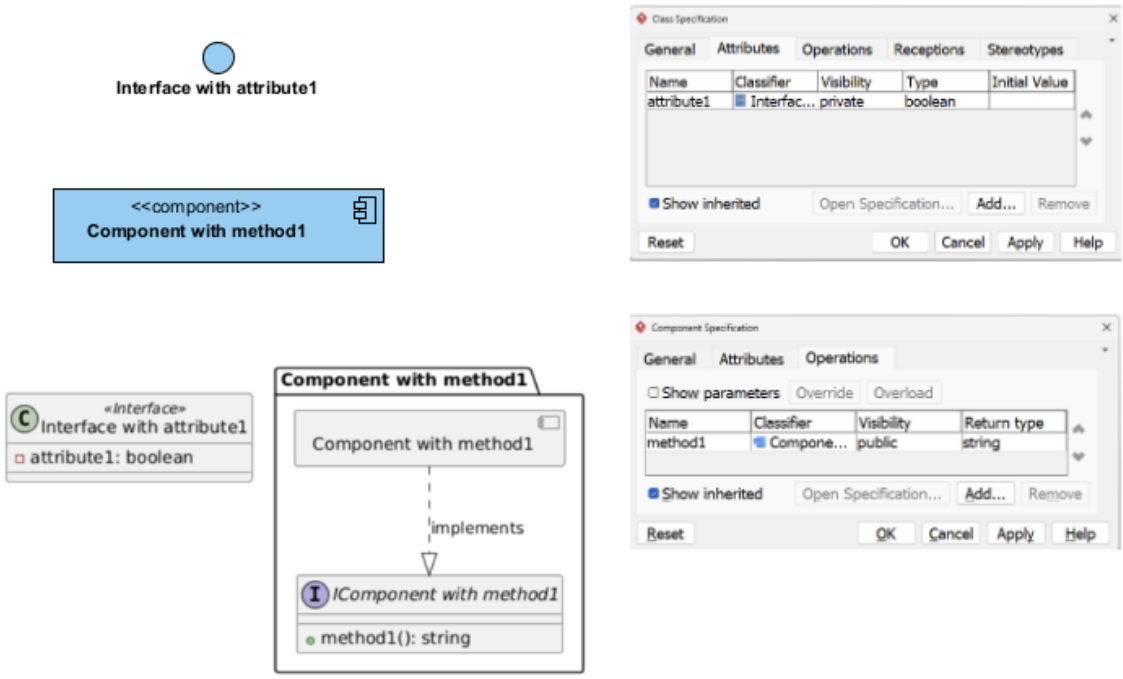
Για τα components που περιέχουν μεθόδους, εφαρμόζεται μια εναλλακτική προσέγγιση κατά την εξαγωγή (export).

Το component "διαχωρίζεται" σε δύο στοιχεία:

1. Ένα component χωρίς μεθόδους.
2. Μια κλάση με στερεότυπο «interface», στην οποία μεταφέρονται οι μέθοδοι του αρχικού component.
3. Component και κλάση interface συνδέονται με σχέση Realization.
4. Τα δύο αυτά στοιχεία τοποθετούνται μέσα σε ένα κοινό package, διατηρώντας τη λογική σύνδεσή τους.

Με αυτόν τον τρόπο, **ξεπερνάμε τον περιορισμό του PlantUML**, διατηρώντας όσο το δυνατόν περισσότερες πληροφορίες από το αρχικό διάγραμμα.

Παράδειγμα που αναδεικνύει την αντιμετώπιση που περιγράφηκε φαίνεται στο σχήμα 4.1 όπου αριστερά και πάνω Visual Paradigm διαγραμμα component με "κρυφές" μεθόδους και ιδιότητες (φαίνονται στα παράθυρα δεξιά) και κάτω το PlantUML διάγραμμα που προέκυψε από το export.



Σχήμα 4.1: Αντιμετώπιση έλλειψης μεθόδων σε PlantUML component διαγράμματα

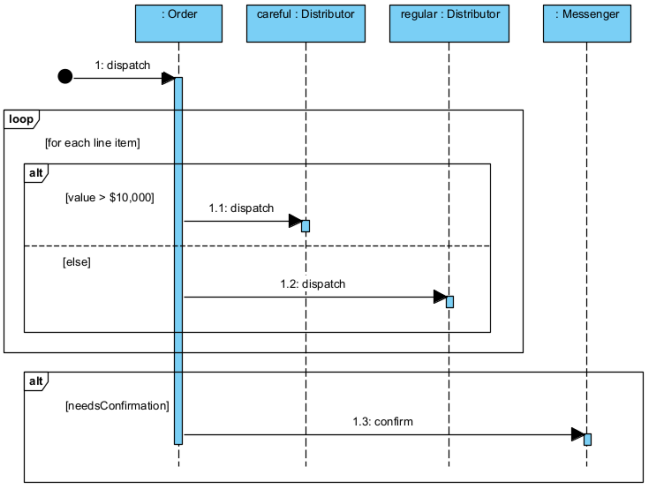
Κατά το import, όλα αυτά τα στοιχεία παίρνουν μορφή αμιγώς τύπου component diagram.

Σημειώνεται ότι αυτή η χρήση αποτελεί τη μόνη επιτρεπτή ανάμειξη διαγραμμάτων και οποιαδήποτε άλλη χρήση του allowmixing δεν είναι έγκυρη προς import με το εργαλείο.

4.3.3 Sequence Diagram

Classifiers Ένα σημαντικό χαρακτηριστικό των Lifeline μοντέλων είναι ο τύπος τους (classifier). Ο classifier είναι ένα άλλο μοντέλο class ή component του οποίου το Lifeline αποτελεί ένα instance. Ωστόχο, δεν υπάρχει αντίστοιχη δομή στο PlantUML εφόσον διαχειρίζεται μόνο ένα διάγραμμα τη φορά και δε χωράει αναφορές σε άλλα μοντέλα, όπως κλάσεις. Για να μη χαθεί η πληροφορία του classifier:

- Κατά το export από το Visual Paradigm, στο παραγόμενο διάγραμμα PlantUML προστίθεται ένα σημείωμα (note) πάνω από το αντίστοιχο Lifeline που δηλώνει το classifier του.
- Κατά το import από PlantUML, το όνομα του Lifeline αναζητείται σε ονόματα κλάσεων και component στοιχείων (με προτεραιότητα στις κλάσεις). Αν βρεθεί το όνομα σε υποψήφιο classifier, τότε στο Visual Paradigm εισάγεται η πληροφορία ότι το Life-



Σχήμα 4.2: Διάγραμμα Sequence με πολλά στοιχεία Combined Fragment

line έχει το classifier που ταίριαξε. Επιπροσθέτως, για τα εξερχόμενα μηνύματα των lifeline που απέκτησαν classifier εξετάζεται αν επικαλούνται κάποια από τις μεθόδους του, οπότε και γίνεται το μήνυμα τύπου "Call" με σωστή αναφορά στη μέθοδο της κλάσης/component.

Combined Fragments Τα Combined Fragments είναι τα στοιχεία alt/else, loop, ref (βλέπε σχήμα 4.2) και άλλα τα οποία ορίζουν ειδικές αλληλουχίες αλληλεπιδράσεων. Είναι σημαντικό στοιχείο των διαγραμμάτων Sequence και υποστηρίζονται και από τα δύο εργαλεία. Δυστυχώς, ενώ κατά το export ήταν δυνατή η μεταφρασή τους, η κατεύθυνση PlantUML προς Visual Paradigm δεν ήταν δυνατή. Αυτό οφείλεται στην ευαίσθητη σύνδεση αυτών των στοιχείων οπτικά (σε επίπεδο διαγράμματος) στο Visual Paradigm με τα υποκείμενα μοντέλα τους. Η εισαγωγή τους ως μοντέλα (model elements) ήταν επιτυχής, αλλά η αυτόματη διάταξη των στοιχείων στο διάγραμμα αποτυγχάνει πλήρως, κάτι που τελικά επηρεάζει και τα μοντέλα καθώς αυτά προσαρμόζουν τα στοιχεία που περιέχουν δυναμικά από τη διάταξη του διαγράμματος.

4.3.4 Activity Diagram

Τα διαγράμματα τύπου Activity ήταν η μεγαλύτερη πρόκληση λόγω της δυσκολίας απόσπασης των δομών από τον κώδικα του PlantUML που αφορούν τη νέα σύνταξή του. Όντας σχετικά νέος και βασισμένος σε ιδιωτικές μεθόδους και στοιχεία η αξιοποίηση του ήταν ιδιαίτερη πρόκληση όπως συζητήθηκε και σε προηγούμενο κεφάλαιο.

Ως αποτέλεσμα αποτελεί το πλέον ημιτελές στην υποστήριξή του. Συγκεκριμένα, δεν υλοποιήθηκαν

όσα αναφέρονται στο πίνακα 4.1.

Στοιχείο	**Αίτια μη υποστήριξης**	**Λειτουργία**
Object nodes / flows	Δεν υπάρχει αντιστοιχία	export
Πολλαπλά start nodes / flows	Σύνταξη PlantUML	import, export
Join στον ίδιο κόμβο ροών που έγιναν fork σε διαφορετικό κόμβο	Αδύνατο σε PlantUML, υπό ανάπτυξη	export
"repeat" λούπες	Ανεπιτυχία απόσπασης από κώδικα PlantUML	import, export
Σύνδεση στοιχείου με προηγούμενο (εκτός decision κόμβου)	Αδύνατο σε PlantUML	export
Ορισμένα ονόματα συνδέσμων	Ανεπιτυχία απόσπασης από κώδικα PlantUML	import
Swimlane σε επίπεδο διαγράμματος	Μεγάλο πρόβλημα διάταξης του VP OpenAPI	import

Πίνακας 4.1: Μη υποστηριζόμενα για μετατροπή στοιχεία των Activity Diagrams

4.4 Το συμπληρωματικό αρχείο JSON

Ένας από τους στόχους κατά την ανάπτυξη του εργαλείου μετατροπής, όπως προαναφέρθηκε, ήταν η διατήρηση του σημασιολογικού πλούτου του Visual Paradigm κατά τη διαδικασία μετατροπής προς PlantUML.

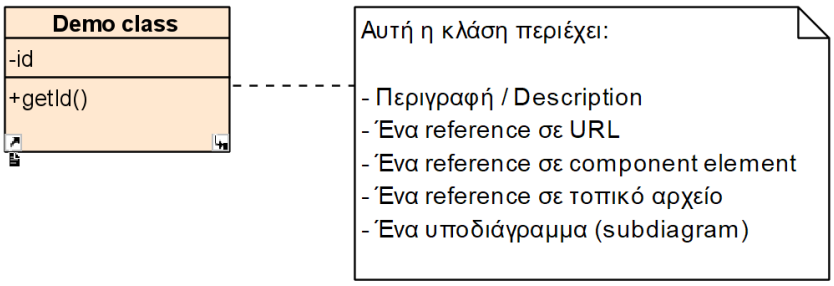
Πιο σημαντικές κρίθηκαν τρεις δομές που μπορεί να υποστηρίξουν όλα τα μοντέλα στο Visual Paradigm (model elements): **αναφορές (references), υποδιαγράμματα (subdiagrams) και περιγραφές**.

- **Αναφορές**. Πρόκειται για αναφορές εντός μοντέλου (έργου Visual Paradigm), όπως αναφορά σε άλλο διάγραμμα ή άλλο μοντέλο διαγράμματος, αναφορές σε τοπικά αρχεία ή φακέλους του υπολογιστή εγκατάστασης, σύνδεσμοι URL.
- **Υποδιαγράμματα**. Διαγράμματα εντός ενός έργου Visual Paradigm μπορούν, εκτός από αναφορές, να οριστούν ως υποδιάγραμμα οποιοδήποτε μοντέλου που εμφανίζεται σε άλλο διάγραμμα (ανεξαρτήτως τύπου).
- **Περιγραφές**. Κάθε μοντέλο μπορεί επίσης να συνοδεύεται από μία περιγραφή.

Αυτές οι πληροφορίες δε μπορούν με κανένα τρόπο να χωρέσουν στις απλές αναπαραστάσεις του PlantUML. Ωστόσο, κρίθηκε σκόπιμο να τις εξάγουμε σε μορφή που επιτρέπει τη μηχανική επεξεργασία, είτε από λογισμικά που μπορούν να κατακευαστούν, είτε από LLMs. Σε κάθε περίπτωση, δίνει τη δομή των ολοκληρωμένων μοντέλων UML που υποστηρίζονται από τα Visual Paradigm.

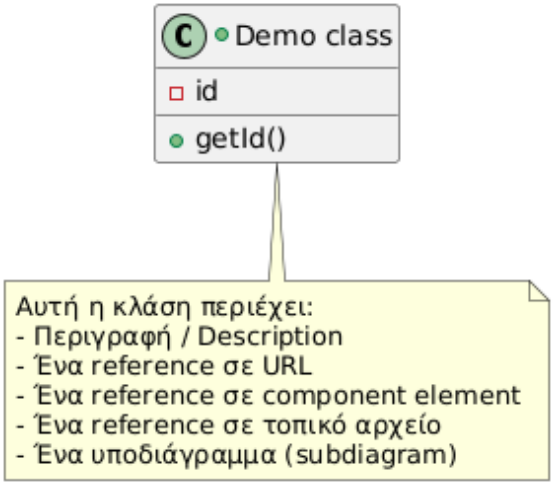
Επιλέχθηκε λοιπόν να δημιουργείται ενα συμπληρωματικό αρχείο JSON κατά τη μετατροπή στη κατεύθυνση Visual Paradigm προς PlantUML, σε ειδική μορφή που μπορεί να διαβαστεί και να αναπαρασταθεί σε PlantUML. Η δομημένη αυτή πληροφορία μπορεί να εισαχθεί επίσης στο Visual Paradigm με τις πληροφορίες να τοποθετούνται στις δομές των δεδομένων κανονικά.

Κατά τη διαδικασία export, κάθε μοντέλο (από αυτά των οποίων διαγραμματικά στοιχεία μετατράπηκαν σε PlantUML) εξετάζεται για ύπαρξη έστω ενός από τα τρία χαρακτηριστικά που αποθηκεύουμε (περιγραφή, αναφορές, υποδιαγράμματα). Αν βρεθεί χαρακτηριστικό προς αποθήκευση, προστίθεται στη λίστα JSON συνοδευόμενο από το όνομά του, το τύπο του, τη περιγραφή του (αν υπάρχει) και λίστες με αναφορές και υποδιαγράμματα (αν υπάρχουν). Κάθε αναφορά συνοδεύεται από τον τύπο της (url, diagram, model, file, folder), τη περιγραφή της και το όνομά της. Αντίστοιχα, για κάθε υποδιάγραμμα συμπεριλαμβάνεται ο τίτλος του και ο τύπος του.



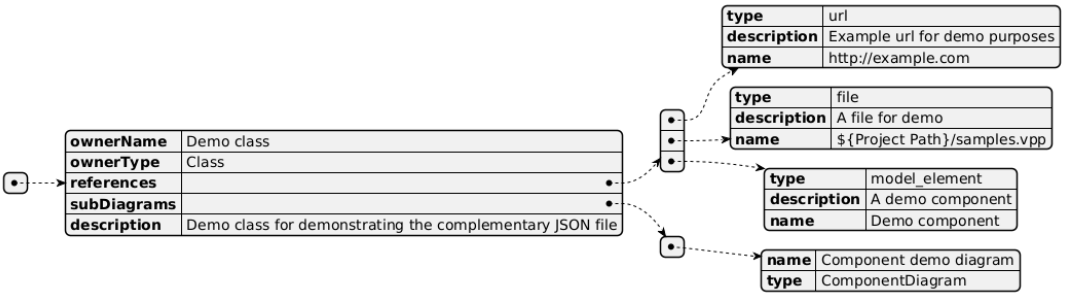
Σχήμα 4.3: Demo Class: Μία κλάση με semantics στο Visual Paradigm

Το παράδειγμα "Demo class" του σχήματος 4.3 περιέχει μία απλή κλάση με όλα τα παραπάνω χαρακτηριστικά. Μέσω του περιβάλλοντος του Visual Paradigm προστέθηκαν μία περιγραφή, τρείς αναφορές διαφορετικού τύπου και ένα υποδιάγραμμα, όπως περιγράφει και το περιεχόμενο του σημειώματος.



Σχήμα 4.4: Demo Class: Μία κλάση με semantics στο μετά από export σε PlantUML

Αποτέλεσμα της εξαγωγής προς PlantUML είναι δύο αρχεία αναγνώσιμα από το PlantUML: ένα Class Diagram (σχήμα 4.4), και ένα JSON Diagram (rendered στο σχήμα 4.5).



Σχήμα 4.5: Demo Class: Το συμπληρωματικό semantics JSON σε αναπαράσταση από το PlantUML

Η επιλογή χρήσης JSON σε σύνταξη PlantUML επιτρέπει την ομοιογενή αναπαράσταση των δεδομένων, διατηρώντας συνοχή με τα υπόλοιπα PlantUML αρχεία. Επιπλέον, παρέχει μια οπτικά κατανοητή αναπαράσταση, κάνοντας τη δομή του JSON πιο προσιτή στον χρήστη μέσω της γραφικής απεικόνισής του. Παράλληλα, η δομημένη μορφή εξασφαλίζει ότι το αρχείο παραμένει εύκολα αναγνώσιμο και επεξεργάσιμο από κώδικα, διευκολύνοντας τη διαδικασία Import στο Visual Paradigm.

Listing 4.1: Complementary JSON PlantUML

```
@startjson
```

```
[
  {
    "ownerName": "Demo class",
    "ownerType": "Class",
    "references": [
      {
        "type": "url",
        "description": "Example url for demo purposes",
        "name": "http://example.com"
      },
      {
        "type": "file",
        "description": "A file for demo",
        "name": "${Project Path}/samples.vpp"
      },
      {
        "type": "model_element",
        "description": "A demo component",
        "name": "Demo component"
      }
    ],
    "subDiagrams": [
      {
        "name": "Component demo diagram",
        "type": "ComponentDiagram"
      }
    ],
    "description": "Demo class for demonstrating the complementary
    JSON file"
  }
]
@endjson
```

4.5 Εγκατάσταση του Plugin

Η εγκατάσταση του PlantUML Plugin γίνεται όπως οποιαδήποτε άλλη επέκταση του Visual Paradigm.

Μοναδική προϋπόθεση είναι η διάθεση της desktop εφαρμογής του Visual Paradigm με οποιοδήποτε επίπεδο άδειας (π.χ. Community, Modeler, Enterprise).

Βήματα εγκατάστασης

1. Κατεβάστε το συμπιεσμένο φάκελο .zip της τελευταίας έκδοσης στο Github reposi-

tory[1] (μην αποσυμπιέσετε).

2. Ανοίξτε την εφαρμογή του Visual Paradigm και επιλέξτε τη καρτέλα *Help* από το μενού.
3. Πατήστε στο κουμπί *Install Plugin*.
4. Στο αναδυόμενο μενού, επιλέξτε *Install from a zip of plugin* και πατήστε *Next*.
5. Δώστε τη τοποθεσία του αρχείου .zip.
6. Επανεκκινήστε την εφαρμογή ωστέ να ολοκληρωθεί η εγκατάσταση.

Μετά από το βήμα 5, αν η εγκατάσταση ήταν επιτυχής, θα πρέπει να εμφανιστεί το σχετικό μήνυμα σε διάλογο pop-up.

Αν εμφανιστεί μήνυμα αποτυχίας, δοκιμάστε να αποσυμπιέσετε τον φάκελο και να τον εγκαταστήσετε με έναν από τους υπόλοιπους διαθέσιμους τρόπους κατά το βήμα 4. Στο τέλος, επανεκκινήστε την εφαρμογή.

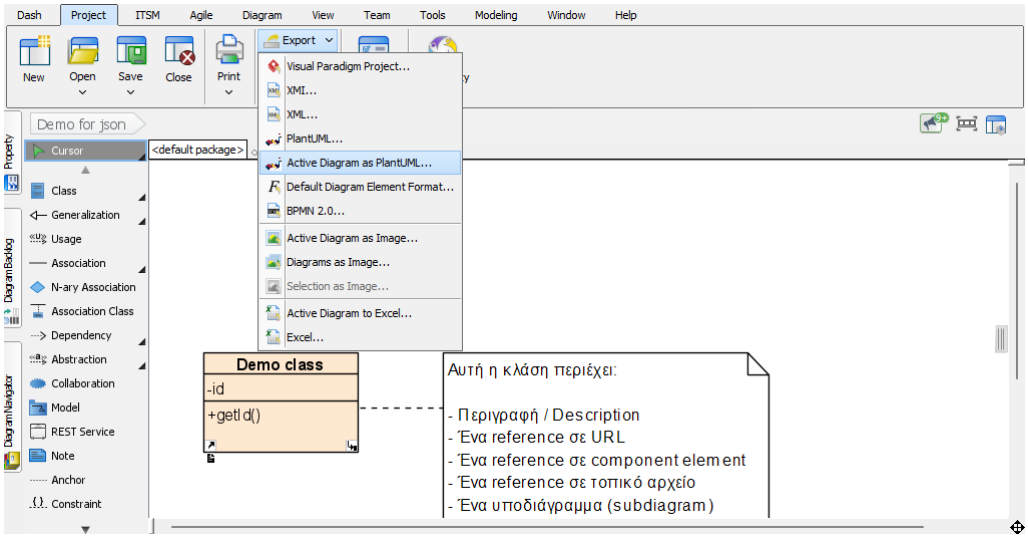
4.6 Οδηγός χρήσης των λειτουργιών

Στη παρούσα ενότητα θα παρουσιάσουμε τη χρήση του Plugin στη πράξη, μέσω της αλληλεπίδρασης με τα προστιθέμενα γραφικά στοιχεία στο Visual Paradigm.

4.6.1 Λειτουργία export

Η λειτουργία export του plugin ενσωματώθηκε στο drop-down μενού "Export" του tab "Project", όπου βρίσκονται και οι υπόλοιπες ενσωματωμένες επιλογές εξαγωγής του Visual Paradigm. Η επιλογή εμφανίζεται με την ονομασία "*PlantUML...*" (βλ. Σχημα 4.6). Συμπληρωματικά, προστέθηκε και επιλογή "*Active Diagram as PlantUML...*" η οποία εξάγει το ενεργό διάγραμμα.

[1] https://github.com/ntua/plantuml-vp

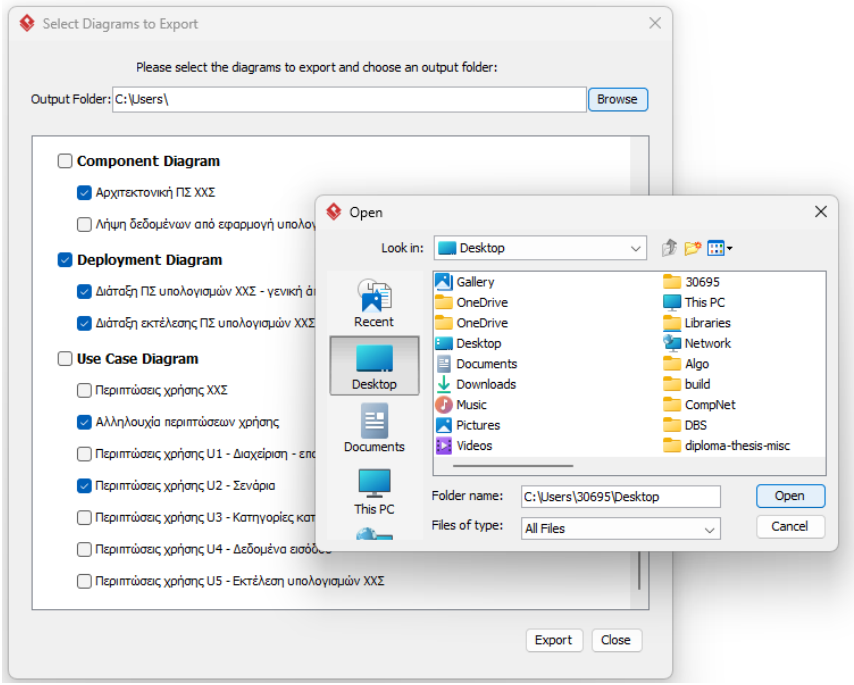


Σχήμα 4.6: Οι ενσωματωμένες επιλογές PlantUML export στο Visual Paradigm

Όταν επιλεγεί η λειτουργία export, εμφανίζεται ένα popup παράθυρο με τη μορφή του σχήματος 4.7, στο οποίο ο χρήστης καλείται να διαμορφώσει τις ρυθμίσεις εξαγωγής.

Συγκεκριμένα, το παράθυρο περιέχει:

- Λίστα με όλα τα διαθέσιμα διαγράμματα του έργου, όπου ο χρήστης μπορεί να επιλέξει ποια διαγράμματα επιθυμεί να εξάγει, χρησιμοποιώντας checkboxes.
- Πεδίο επιλογής φακέλου εξαγωγής (output folder), το οποίο υλοποιείται μέσω file chooser, επιτρέποντας στον χρήστη να καθορίσει τη διαδρομή αποθήκευσης των εξαγόμενων αρχείων.

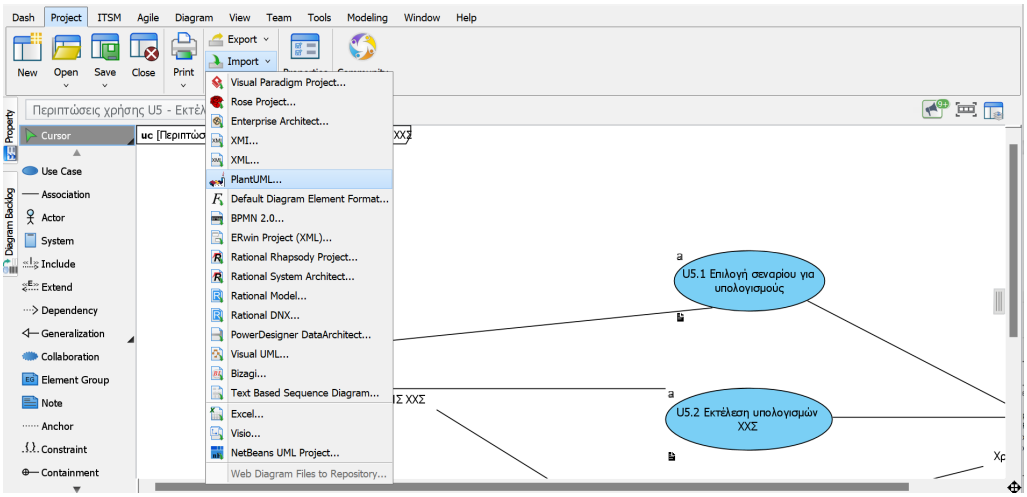


Σχήμα 4.7: Επιλογή διαγραμμάτων για export και φακέλου εξόδου

Τέλος, με την οριστικοποίηση των επιλογών και την επίλογη "Export" θα πρέπει να εμφανιστούν μήνυμα ολοκλήρωσης σε συνδυασμό με τυχόν ειδοποιήσεις για ημιτελείς εξαγωγές, και να εμφανιστούν στον επιλεγμένο φάκελο εξόδου τα αντίστοιχα διαγράμματα σε PlantUML μορφή.

4.6.2 Λειτουργία import

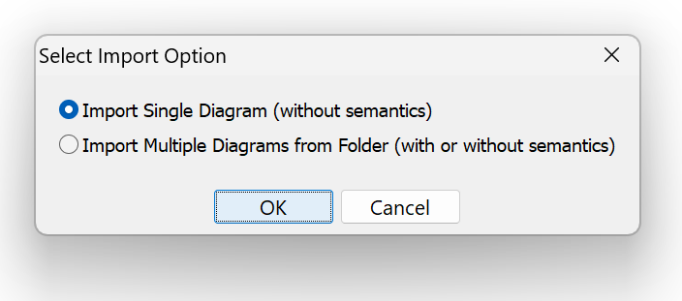
Παρόμοια, η λειτουργία import του plugin ενσωματώθηκε στο drop-down μενού "Import" του tab "Project", όπου βρίσκονται και οι υπόλοιπες ενσωματωμένες επιλογές εισαγωγής. Η επιλογή εμφανίζεται και αυτή με την ονομασία "PlantUML..." (βλ. Σχημα 4.8).



Σχήμα 4.8: Η ενσωματωμένη επιλογή PlantUML import στο Visual Paradigm

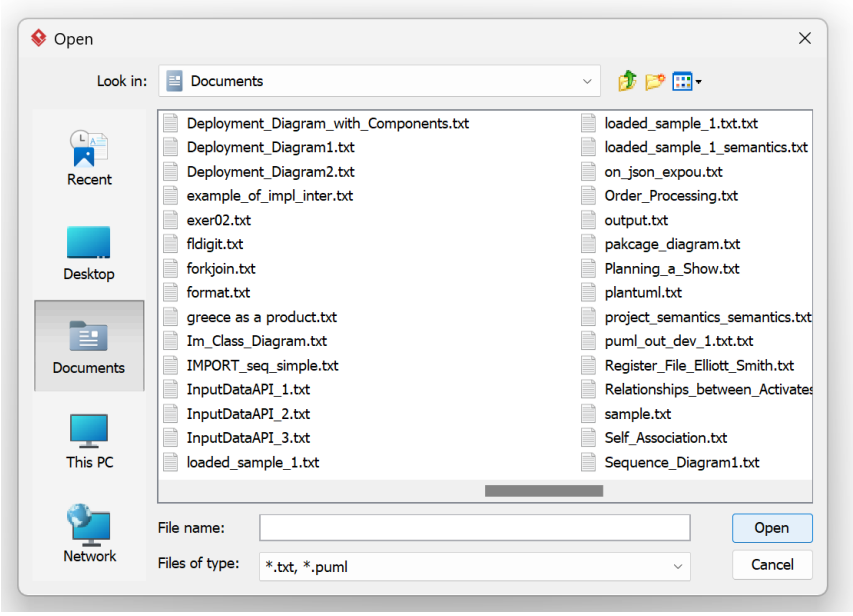
Στη συνέχεια, εμφανίζεται ένα popup παράθυρο επιλογής λειτουργίας (4.9, στο οποίο ο χρήστης καλείται να επιλέξει ανάμεσα σε δύο διαθέσιμες επιλογές εισαγωγής:

1. Εισαγωγή ενός μόνο διαγράμματος από αρχείο PlantUML
    - Σε αυτήν την περίπτωση, ο χρήστης μπορεί να επιλέξει ένα μεμονωμένο αρχείο PlantUML για εισαγωγή.
    - Δεν υποστηρίζεται η εισαγωγή semantics για αυτήν την επιλογή, καθώς αφορά αποκλειστικά τη μετατροπή ενός και μόνο αρχείου.
2. Εισαγωγή πολλών αρχείων (με ή χωρίς JSON αρχείο)
    - Ο χρήστης μπορεί να επιλέξει φάκελο με πολλαπλά αρχεία PlantUML προς εισαγωγή.
    - Σε αυτήν την περίπτωση, υποστηρίζεται και η εισαγωγή JSON αρχείου το οποίο περιέχει συμπληρωματικές πληροφορίες.



Σχήμα 4.9: Επιλογές import του Plugin

Ανάλογα την επίλογη, εμφανίζεται file chooser για το αρχείο ή τον φάκελο εισόδου. Τέλος, εμφανίζεται μήνυμα επιτυχίας (ή επιτυχίας με προειδοποιήσεις) ή αποτυχίας εισαγωγής.



Σχήμα 4.10: Επιλογή PlantUML αρχείου ή φακέλου για import

4.7 Παραδείγματα μετατροπών από και προς PlantUML

Ακολουθούν κάποια χαρακτηριστικά παραδείγματα των αποτελεσμάτων της χρήσης του plugin για τις δύο κατευθύνσεις μετατροπής. Επιλέχθηκαν με τρόπο ώστε να **αναδεικνύονται οι δυνατότητες αλλά και οι περιορισμοί** της υλοποίησης ενώ σχολιάζονται σημαντικές λεπτομέρειες που αφορούν κάθε τύπο διαγράμματος. **Έγινε προσπάθεια**

όπου μπορεί να υπάρξει απώλεια πληροφορίας, αυτό να φανεί στα παραδείγματα.

Τα περισσότερα διαγράμματα χρησιμοποιούν ως βάση κάποια έτοιμα δείγματα του Visual Paradigm και του PlantUML, με προσθήκες και αλλαγές ώστε να καλύπτουν όσο το δυνατό περισσότερα στοιχεία διαγραμμάτων χωρίς να γίνονται δυσνόητα και δυσανάγνωστα.

4.7.1 Class Diagram

Export

Στο επιλεγμένο παράδειγμα στο Σχήμα 4.11 για την εξαγωγή περιλαμβάνονται πολλά από τα στοιχεία των διαγραμμάτων κλάσεων. Συγκεκριμένα, περιέχει κλάσεις, κλάσεις με stereotype, interfaces, association classes, σημειώσεις (notes), controls, n-ary nodes, αφηρημένες κλάσεις (abstract classes, όπως η Payment) και πακέτα (packages).

Όσον αφορά τους συνδέσμους, περιλαμβάνονται associations, aggregations, realizations, generalizations, anchors και containments. Οι κλάσεις διαθέτουν ιδιότητες και μεθόδους, τόσο με όσο και χωρίς παραμέτρους, ενώ ορισμένοι σύνδεσμοι φέρουν ετικέτες ονομάτων.

Μετά τη μετατροπή σε PlantUML, όλη η πληροφορία διατηρήθηκε επιτυχώς. Το στερεότυπο Control, το οποίο δεν υποστηρίζεται εγγενώς στο PlantUML, δεν διατηρεί το ίδιο σχήμα, αλλά μετατράπηκε σε κλάση με στερεότυπο, όπως επιλέξαμε.

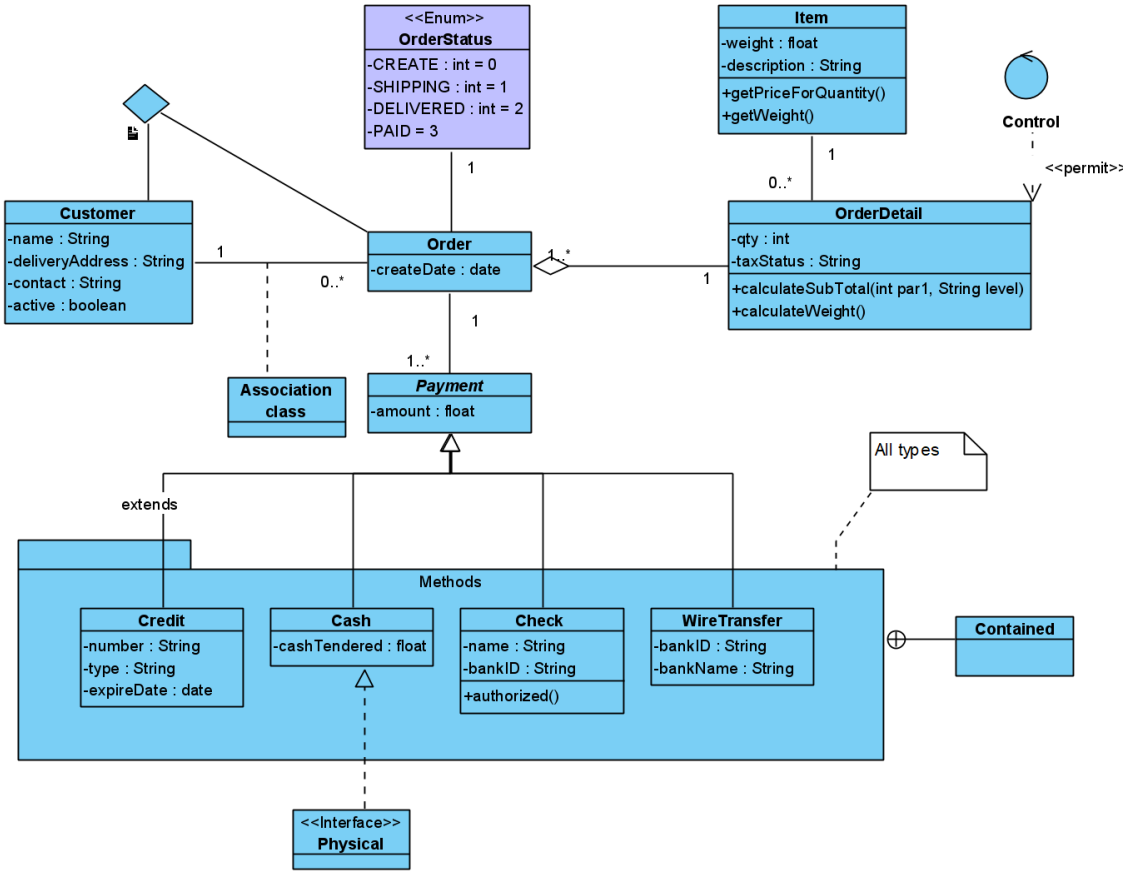
Μια ακόμη προσαρμογή αφορά τον σύνδεσμο containment, όπου, σε περιπτώσεις που σχετίζεται με ένα package, πέρα από την αναπαράσταση του containment, το τοποθετούμε εντός του package για ακόμα πιο ξεκάθαρη αποτύπωση της σχέσης.

Import

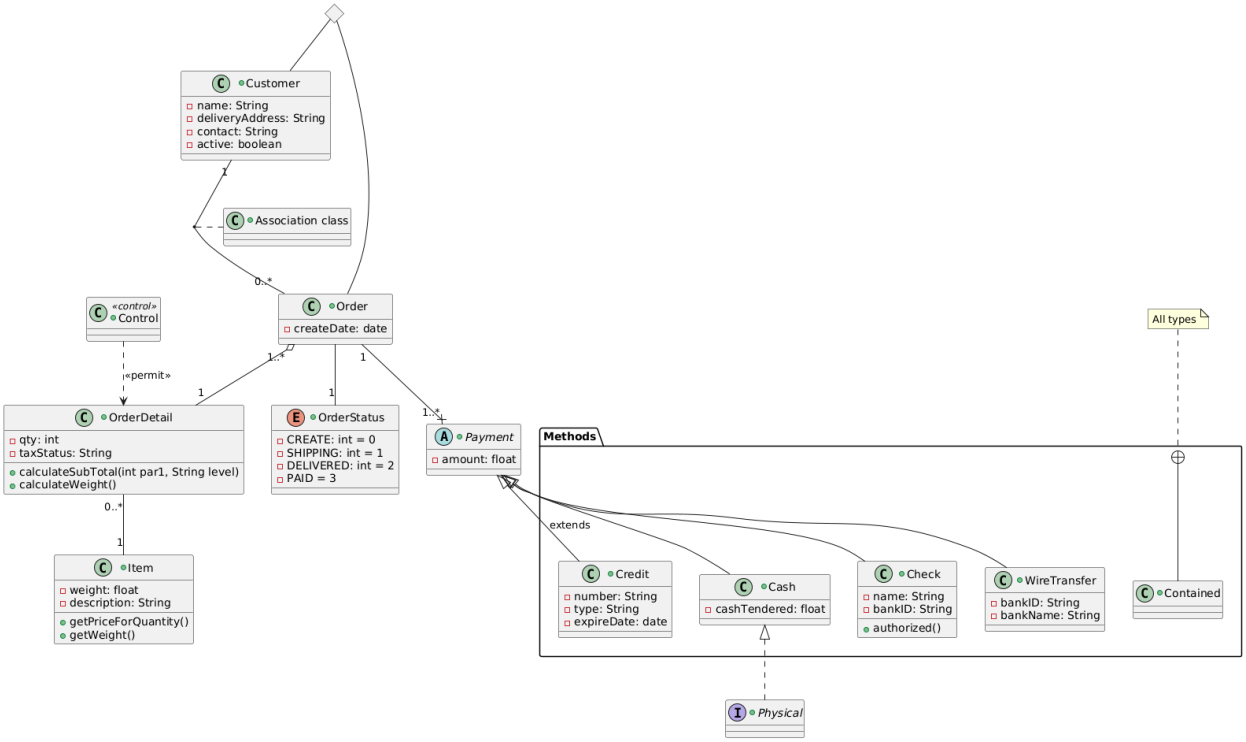
Για το import θα χρησιμοποιήσουμε το ίδιο διάγραμμα, εισάγοντας στο Visual Paradigm το PlantUML αρχείο που προέκυψε (βλ. Σχήμα 4.12).

Από το αποτέλεσμα της εισαγωγής στο Visual Paradigm, όπως φαίνεται στο Σχήμα 4.13, παρατηρούμε τις εξής διαφορές με το πρωτότυπο:

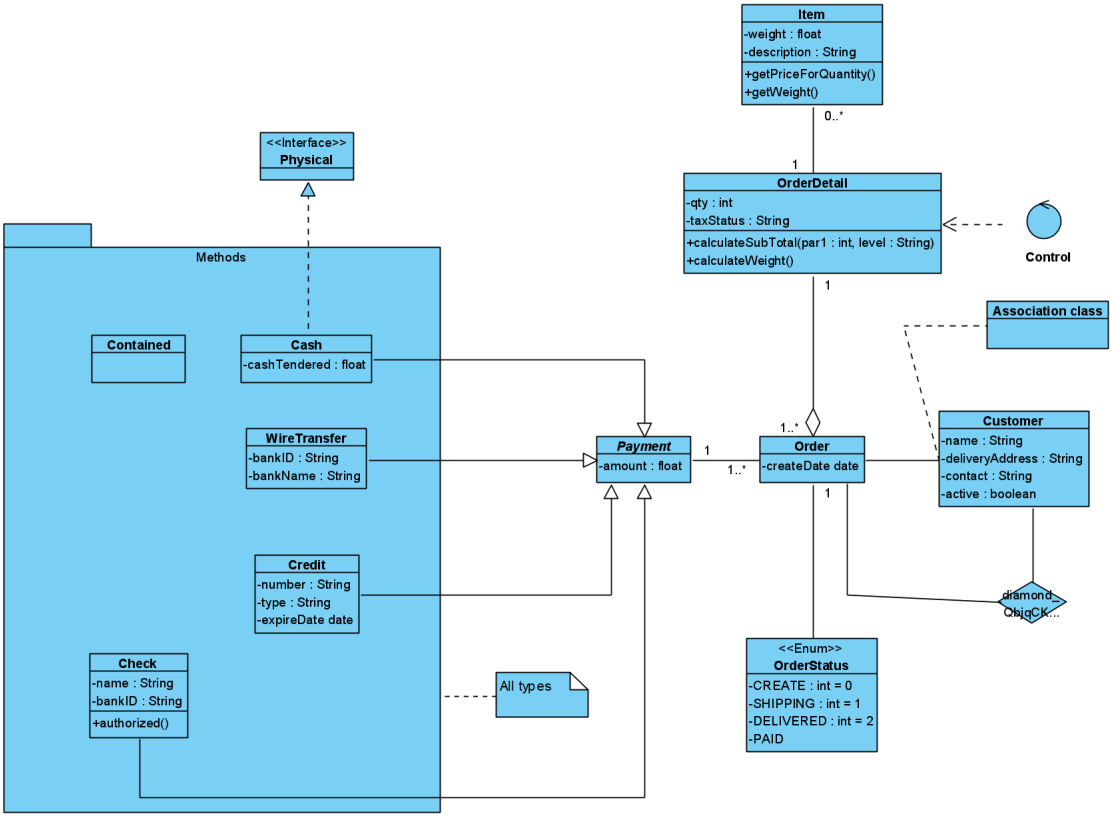
- Η δύο μετατροπές ήταν σχεδόν συμμετρικές.
- Υπήρξαν αναμενόμενες αλλαγές στη διάταξη και το στυλ.
- Η αυτόματη διάταξη δίνει κακό αποτέλεσμα οπτικά στον Association class συνδεσμο, αλλά έχει εισαχθεί κανονικά.
- Χάθηκε ο τύπος «permit» του συνδέσμου μεταξύ Control και OrderDetail. Αυτό συνέβη διότι στο PlantUML ήταν απλά μια ετικέτα. Εισάχθηκε ως dependency.
- Ο κόμβος n-ary πήρε όνομα, λόγω της ανάγκης για alias στο PlantUML που μεταφέρθηκε και στο Visual Paradigm.



Σχήμα 4.11: Παράδειγμα Class Diagram προς εξαγωγή



Σχήμα 4.12: Παράδειγμα Class Diagram μετά από εξαγωγή



Σχήμα 4.13: Το παράδειγμα Class Diagram μετά από εισαγωγή στο Visual Paradigm

4.7.2 Use Case Diagram

Export

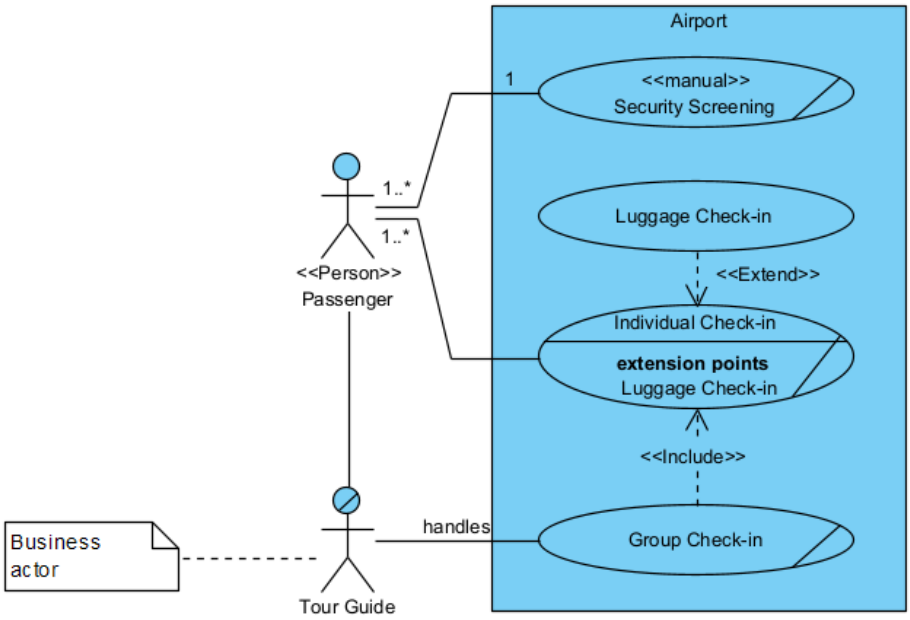
Για την παρουσίαση της εξαγωγής ενός Use Case Diagram, επιλέχθηκε το διάγραμμα της Εικόνας 4.14. Το διάγραμμα αυτό περιλαμβάνει:

- Απλές περιπτώσεις χρήσης (Use Cases) και απλός Actor
- Επιχειρησιακά στοιχεία Use Case και Actor (Business Models, τα οποία χαρακτηρίζονται από τη λοξή γραμμή)
- Actor και Use Case με στερεότυπο
- Πακέτο-σύστημα (Airport)
- Σημείωμα (Note)

- Use Case με "extension points"

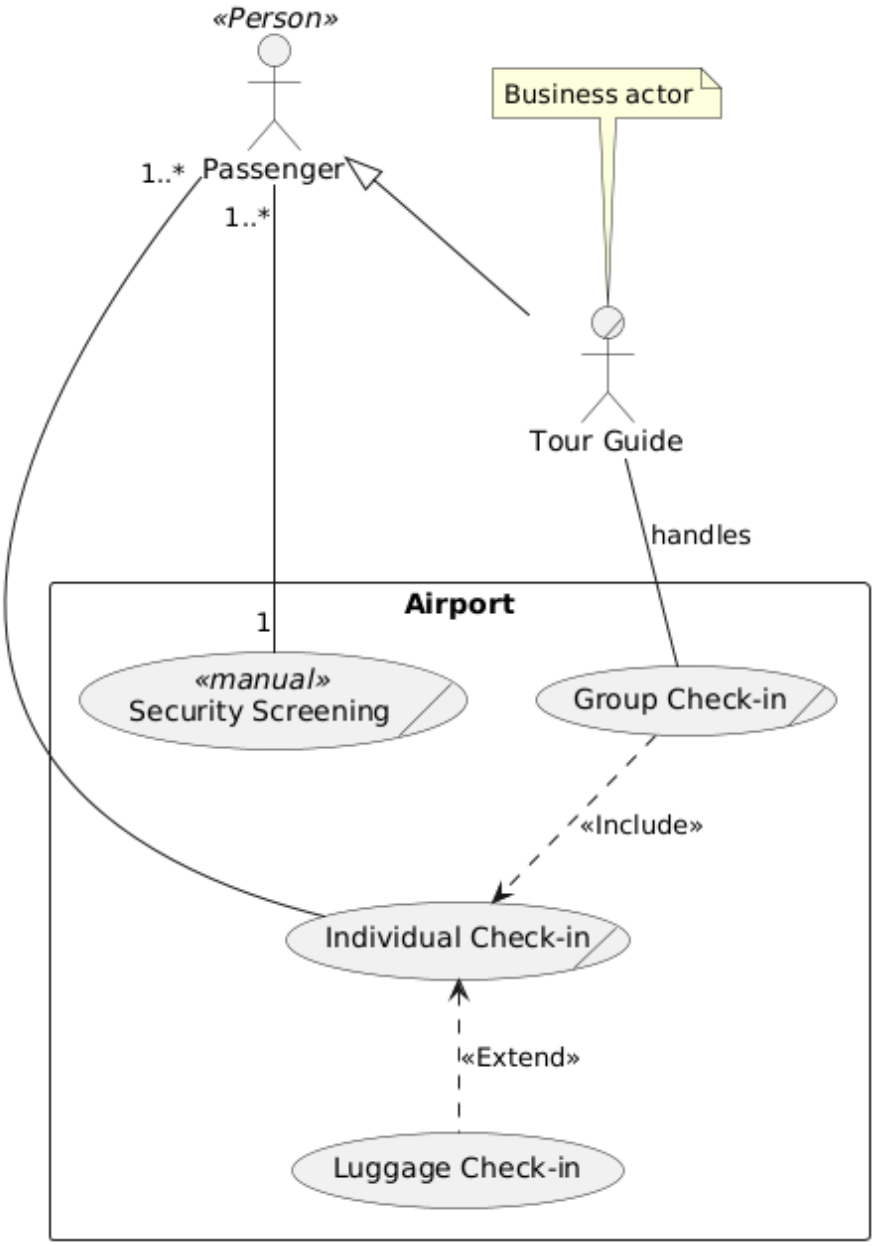
Όσον αφορά τους συνδέσμους, περιλαμβάνονται:

- Associations με πληθυκότητα (multiplicity)
- Απλα associations
- Σχέσεις «extend» και «include»
- Generalization μεταξύ actors
- Σύνδεσμος με ετικέτα ("handles")
- Anchor προς σημείωμα (note)



Σχήμα 4.14: Παράδειγμα Use Case Diagram προς εξαγωγή

Το αποτέλεσμα της μετατροπής φαίνεται στο Σχήμα 4.15 οπού παρατηρούμε σχεδόν πλήρη διατήρηση της πληροφορίας. **Δε μεταφέρθηκε μόνο το extension point** του Use Case "Individual Check-in" καθώς δεν υποστηρίζεται από το PlantUML.



Σχήμα 4.15: Παράδειγμα Use Case Diagram μετά από εξαγωγή

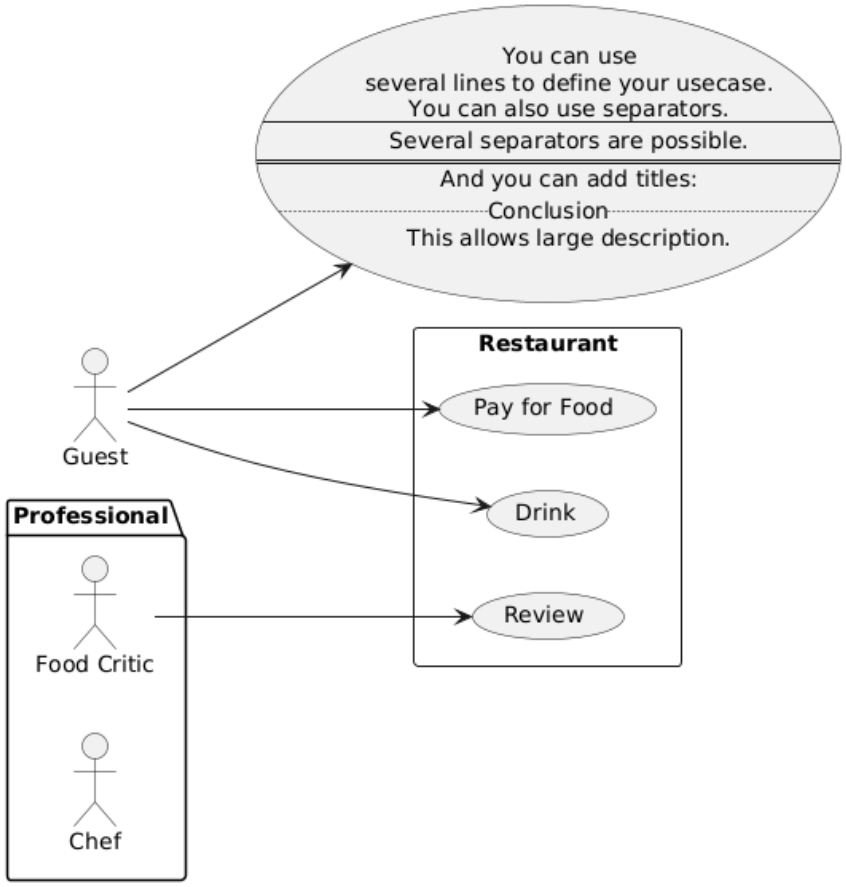
Import

Για την επίδειξη εισαγωγής Use Case διαγράμματος επιλέγουμε ένα που χρησιμοποιεί τις συμβάσεις και το ειδικό στυλ του PlantUML. Στο σχήμα 4.16, έχουμε περιπτώσεις χρήσης

μέσα σε σύστημα και μέσα σε Package, καθώς και μία περίπτωση χρήσης με διάφορα τμήματα περιγραφής σε στυλ ειδικό στο PlantUML.

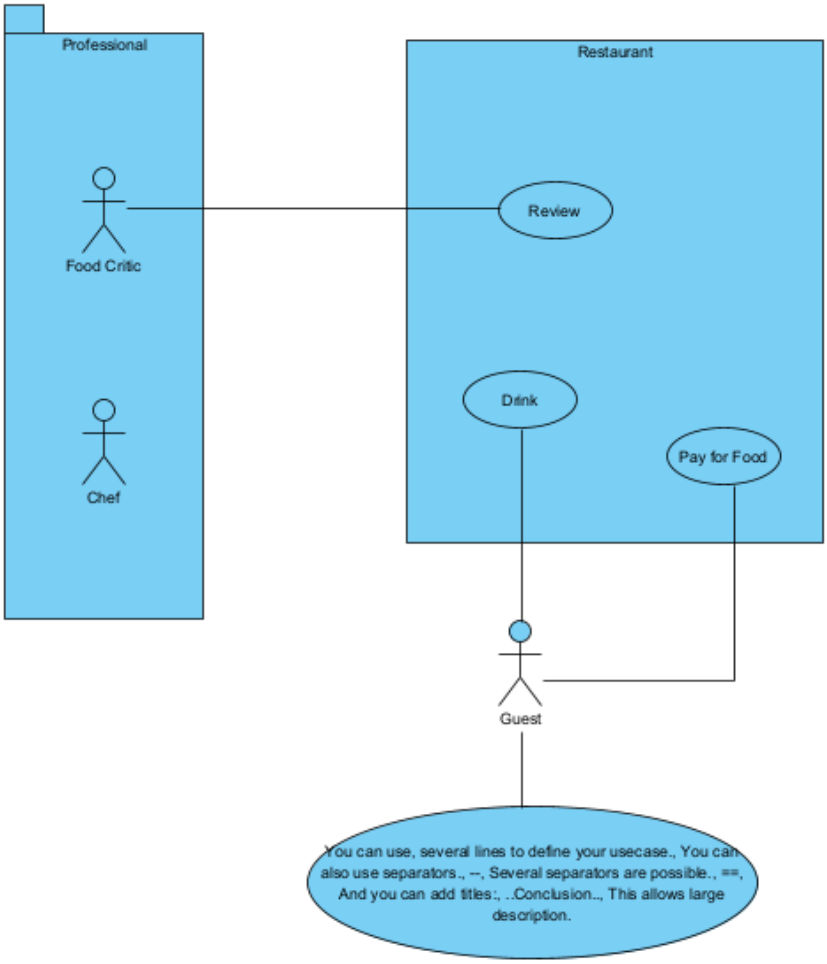
Η PlantUML, σύμφωνα με τη σύμβασή της, δεν συνδέει τα Use Cases με τους Actors μέσω απλών associations (οπτικά απλή γραμμή), αλλά χρησιμοποιεί συμπαγή σύνδεσμο (solid line) με βέλος προς το Use Case.

Αντίστοιχος ή **παρόμοιος σύνδεσμος δεν εντοπίζεται στο Visual Paradigm**. Για αυτόν τον λόγο, κατά την εισαγωγή των διαγραμμάτων μετατρέπονται σε associations, ώστε να ταιριάζουν με τη συμβατική αναπαράσταση του Visual Paradigm.



Σχήμα 4.16: Παράδειγμα Use Case Diagram για εισαγωγή

Τα υπόλοιπα στοιχεία βρίσκουν **πλήρη αντιστοιχία** στο διάγραμμα που προέκυψε με την εισαγωγή (βλ. Σχήμα 4.17). Το στυλ του Use Case με τη μεγάλη περιγραφή σε τμήματα δεν αποδόθηκε με τον ίδιο τρόπο αλλά ως απλό κείμενο.

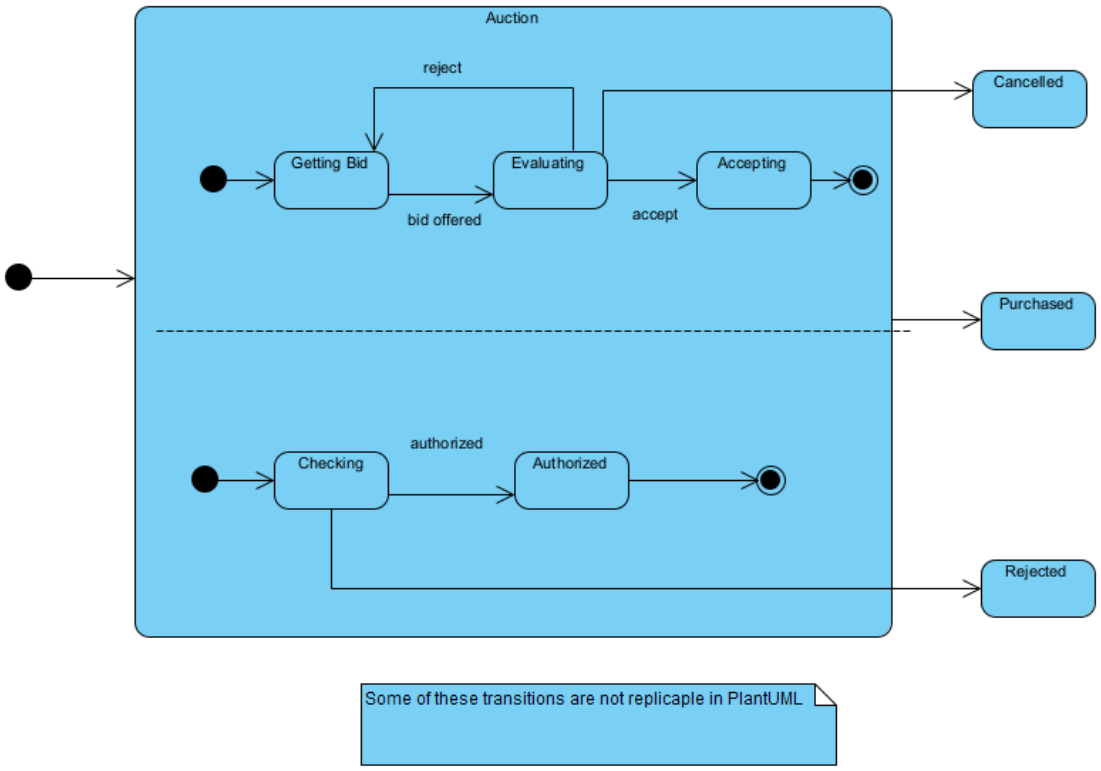


Σχήμα 4.17: Παράδειγμα Use Case Diagram μετά την εισαγωγή

4.7.3 State Diagram

Export

Για το export δημιουργήθηκε σύνθετο παράδειγμα State Diagram. Το State "Auction" του σχήματος 4.18 περιέχει δύο State Regions με εναλλακτικές αλληλουχίες καταστάσεων. Αυτές μπορούν να καταλήξουν και σε εξωτερικές καταστάσεις του διαγράμματος (Cancelled, Rejected). Το διάγραμμα περιέχει και τα βασικά στοιχεία όπως αρχικός και τελικός κόμβος, πολλαπλούς συνδέσμους αλλαγής κατάστασης και ετικέτες, σημείωμα.



Σχήμα 4.18: Παράδειγμα State Diagram προς εξαγωγή

Η εξαγωγή του σύνθετου διαγράμματος είχε ως αποτέλεσμα τη PlantUML απεικόνιση του σχήματος 4.19. Υπήρξαν επίσης δύο επισημάνσεις-προειδοποιήσεις σχετικά με τη μετατροπή: *Warning: Transition " lost during export because it's illegal in PlantUML. Reason: One state is not in a region, but the other belongs to a multi-region state.*

Όπως φαίνεται από το διάγραμμα, **δύο ακμές λείπουν**, όπως επισημαίνουν οι προειδοποιήσεις. Αυτό συμβαίνει διότι σύμφωνα με τη σύμβαση που ακολουθεί το PlantUML, οι δύο ροές καταστάσεων μέσα στη κατάσταση "Auction" είναι ταυτόχρονες (τις αποκαλεί concurrent states). Για αυτό, δε μπορούν να μεταβούν σε κατάσταση εκτός της "Auction" ούτε σε κατάσταση κάποιας άλλης ροής μεταβάσεων μέσα στην "Auction".

Πέρα από τη παραπάνω **ασυμβατότητα μεταξύ PlantUML Concurrent States και Visual Paradigm State with Regions**, τα υπόλοιπα στοιχεία αποδόθηκαν κανονικά.

Import

Για τον έλεγχο της διαδικασίας import χρησιμοποιήθηκε το διάγραμμα του Σχήματος 4.20. Περιέχει:

- Εμφωλευμένες καταστάσεις (μέχρι και δύο επίπεδα)
- Fork και join κόμβους
- Βαθιά ιστορία (deep history H*)
- Κατάσταση με περιγραφή (TaskA)
- Αρχικούς και τελικούς κόμβους
- Συνδέσμους με ετικέτες
- Σημείωμα (note)

Το αποτέλεσμα της εισαγωγής στο Visual Paradigm στο Σχήμα 4.21 είναι άρτιο και διατηρεί όλα τα στοιχεία του αρχικού διαγράμματος. Η μόνη διαφορά βρίσκεται σε ετικέτα ονόματος για τον κόμβο απόφασης "Decision Point", που είναι κρυφή στο PlantUML διάγραμμα αλλά έγκυρη.

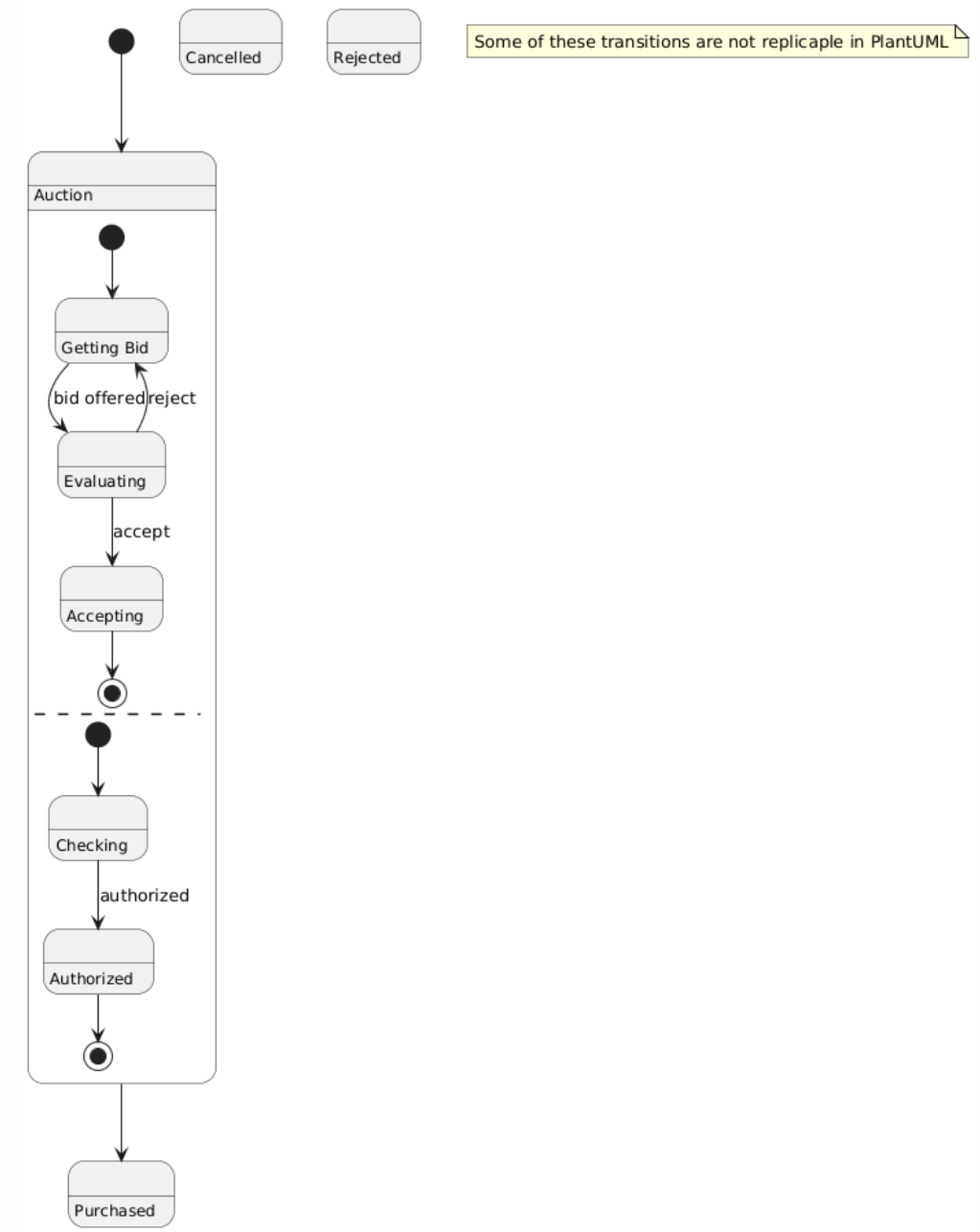
Λόγω πληρότητας παρουσιάζουμε και τα προβληματικά Concurrent States (ταυτόχρονες ροές καταστάσεων) κατά την εισαγωγή. Το διάγραμμα PlantUML του σχήματος 4.22 μεταφράζεται στο διάγραμμα Visual Paradigm 4.23. Η οπτική πληροφορία σε αυτό το δείγμα είναι ακριβώς η ίδια μετά τη μετατροπή.

Σημείωση: η αυτόματη διάταξη δε δούλεψε καλά για το διάγραμμα κατά την εισαγωγή, με τις ακμές να είναι παράλογα μεγάλες και δυσανάγνωστες. Σε τέτοιες περιπτώσεις προτείνεται το περαιτέρω Auto Layout option "**Route connectors orthogonally**" από το μενού.

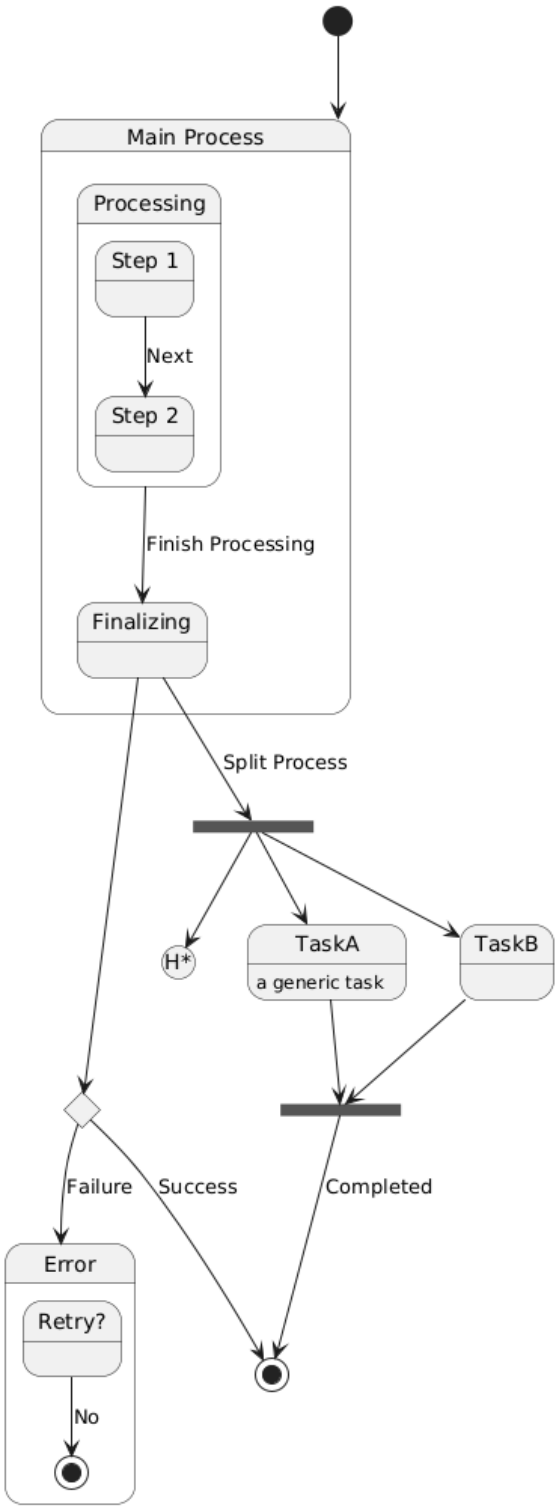
4.7.4 Component Diagram

Export

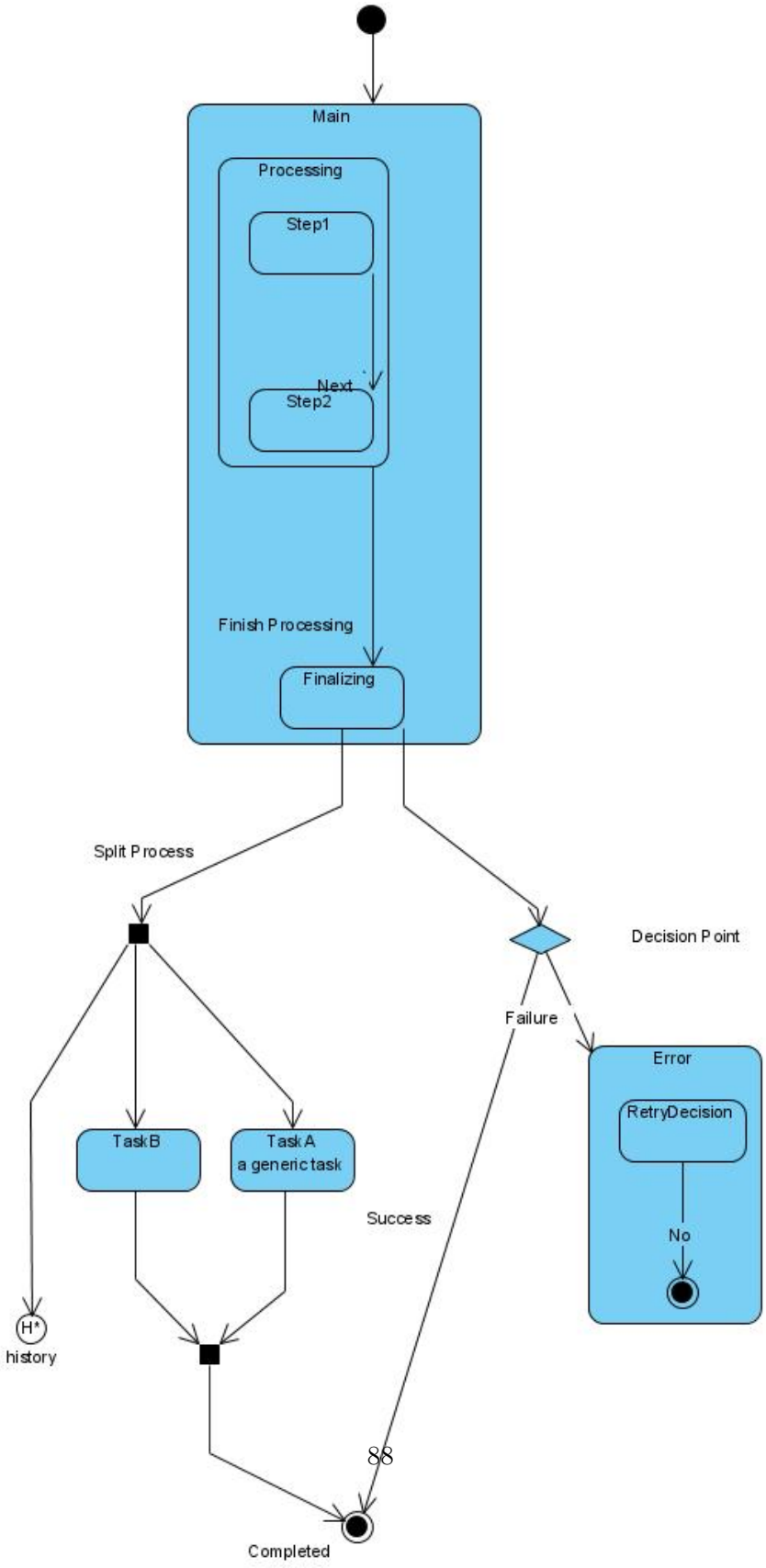
Το διάγραμμα Component του σχήματος 4.24 καλύπτει όλες τις βασικές δομές του τύπου, αφού εμφανίζονται components (με ή χωρίς στερεότυπα), εμφωλευμένα components, ports,



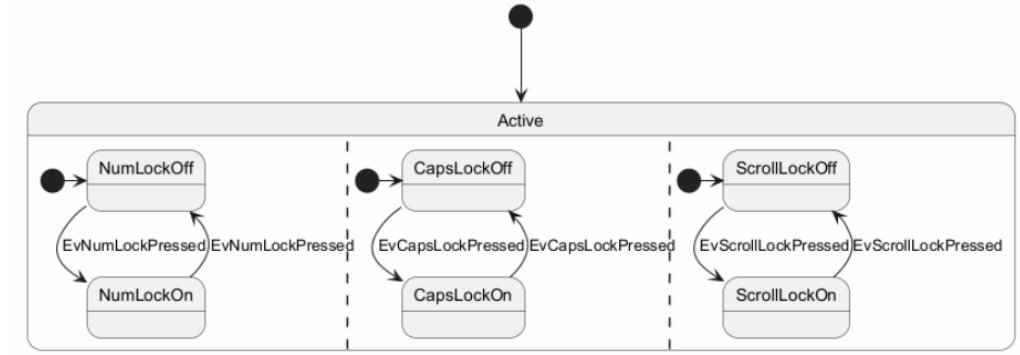
Σχήμα 4.19: Παράδειγμα State Diagram μετά την εξαγωγή



Σχήμα 4.20: Παράδειγμα State Diagram για εισαγωγή



Σχήμα 4.21: Παράδειγμα State Diagram μετά την εισαγωγή



Σχήμα 4.22: State Diagram με "ταυτόχρονα" States για εισαγωγή

interfaces, package και note. Επίσης, το component *ClerkInterface* περιέχει ένα attribute και ένα operation (τα στοιχεία αυτά δεν οπτικοποιούνται), όπως δηλώνει και το σημείωμα.

Οι σύνδεσμοι που περιέχονται στο διάγραμμα είναι τύπου **Usage** («use») και **Realization** (εξαιρείται το Anchor στο σημείωμα). Το Visual Paradigm αναπαριστά τις σχέσεις αυτές ως απλές γραμμές όταν συνδέονται με interfaces, δε πρόκειται όμως για Associations.

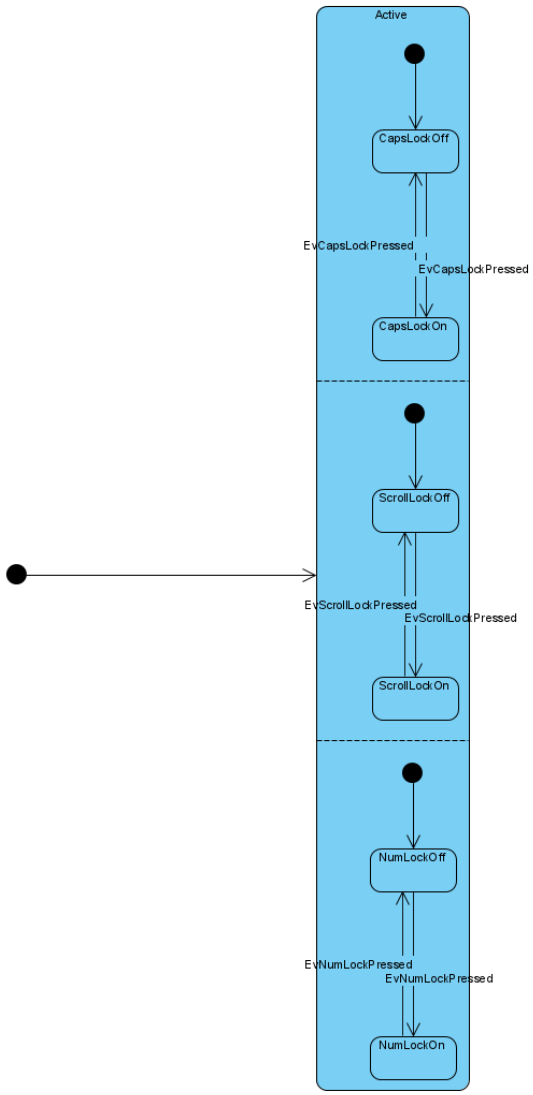
Το αποτέλεσμα του export είναι το PlantUML Component/Class διάγραμμα του σχήματος 4.25. Το διάγραμμα αποτελεί ανάμειξη των τύπων component και class με τη παράμετρο *allowmixing*. Αυτό βοηθά στη παράκαμψη του περιορισμού του PlantUML για τα attributes και operations των components, όπως συζητήθηκε στην υποενότητα 4.3.2 και δείξαμε στο Σχήμα 4.1.

Αν και οπτικά οι σύνδεσμοι δεν είναι ίδιοι με το αρχικό διάγραμμα του Visual Paradigm, είναι ορθοί και **σύμφωνοι με τις συμβάσεις και τη σημασιολογία**. Τα υπόλοιπα στοιχεία **αποδίδονται πιστά**, και η αντιμετώπιση σχετικά με τη μέθοδο και την ιδιότητα του *ClerkInterface* δημιούργησε το Package ClerkInterface.

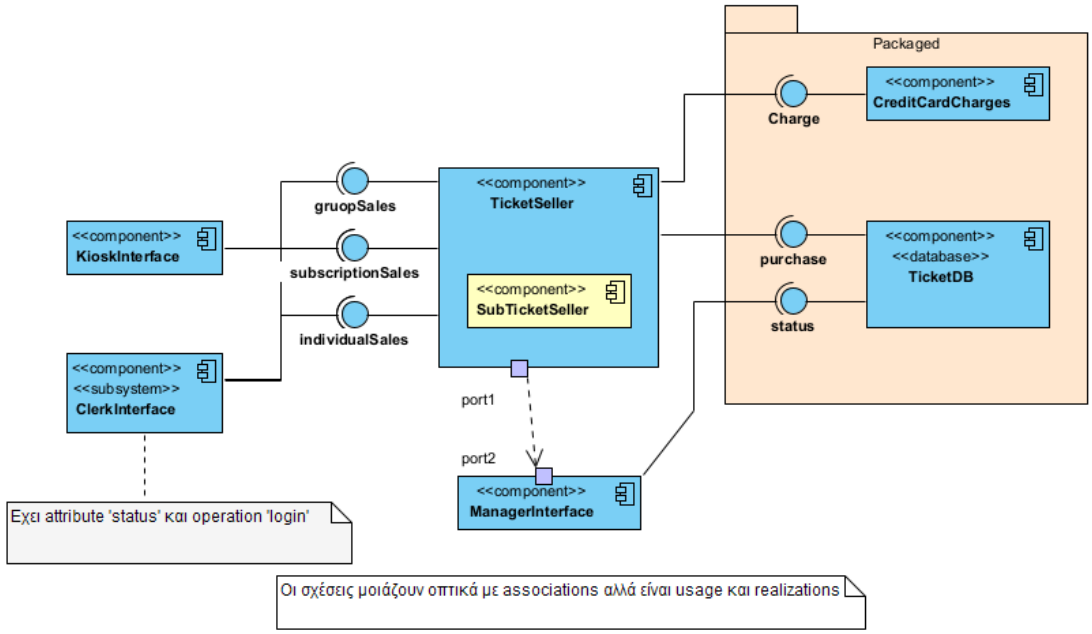
Import

Για τη λειτουργία import χρησιμοποιήθηκε παράδειγμα (Σχήμα 4.26) που ακολουθεί τους τύπους συνδέσμων που συνήθως βρίσκονται σε PlantUML διαγράμματα. Υπενθυμίζουμε ότι η σημασία των συνδέμσων στο PlantUML δεν είναι σαφώς ορισμένη ή αυστηρή. Αυτό κάνει τη μετάφρασή τους δύσκολη καθώς θα πρεπε να γνωρίζουμε τη πρόθεση του συγγραφέα του διαγράμματος ώστε να είμαστε σίγουροι για τη σημασία τους.

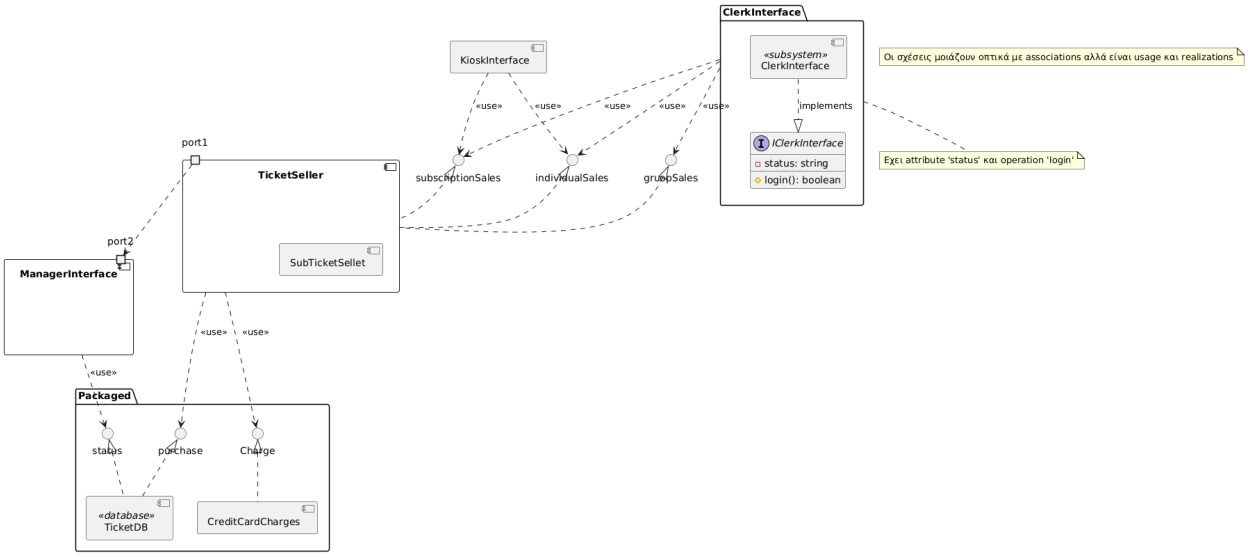
Στο διάγραμμα του παραδείγματος χρησιμοποιούνται απλές γραμμές, και διακεκομμένες με βέλος. Οι οπτικά αντίστοιχες ακμές σε Visual Paradigm είναι Association και Dependency αντίστοιχα, που είναι επιτρεπτές και "νόμιμες" σχέσεις μεταξύ των στοιχείων component και interface. Για αυτό, προς αποφυγή πρωτοβουλιών στην ερμηνεία που πιθανόν βασίζονται σε λανθασμένες υποθέσεις προθέσεων, μετατρέπουμε στα οπτικά αντίστοιχα στοιχεία και



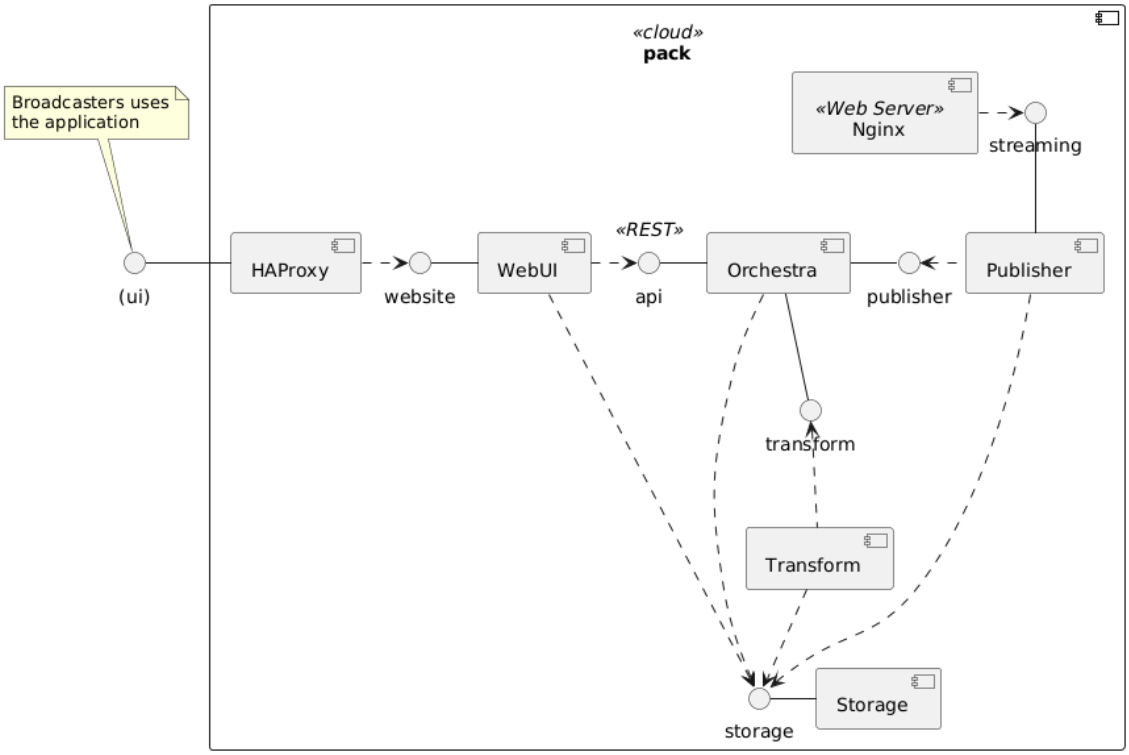
Σχήμα 4.23: State Diagram με "ταυτόχρονα" States μετά την εισαγωγή



Σχήμα 4.24: Παράδειγμα Component Diagram για εξαγωγή



Σχήμα 4.25: Παράδειγμα Component Diagram μετά την εξαγωγή



Σχήμα 4.26: Παράδειγμα Component Diagram για εισαγωγή

δεν μετατρέπουμε σε Realizations ή Usages. Αυτό μπορεί να παράξει "λάθος" διαγράμματα αλλά ήταν αναγκαίος συμβιβασμός.

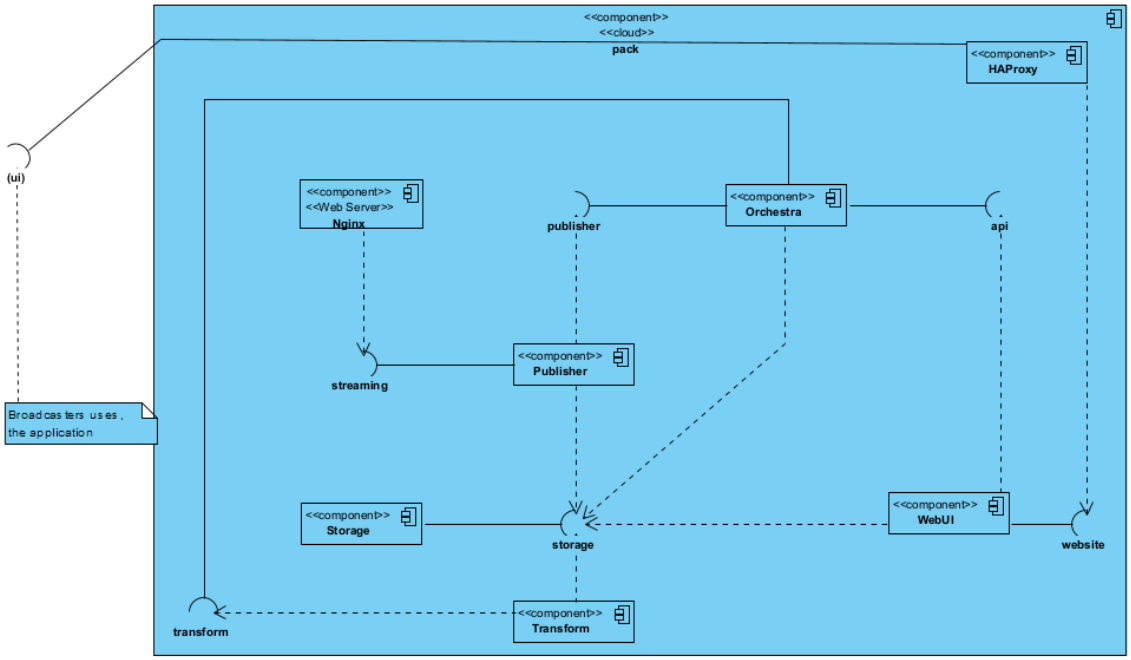
Το αποτέλεσμα της εισαγωγής στο Visual Paradigm είναι το Σχήμα 4.27.

4.7.5 Deployment Diagram

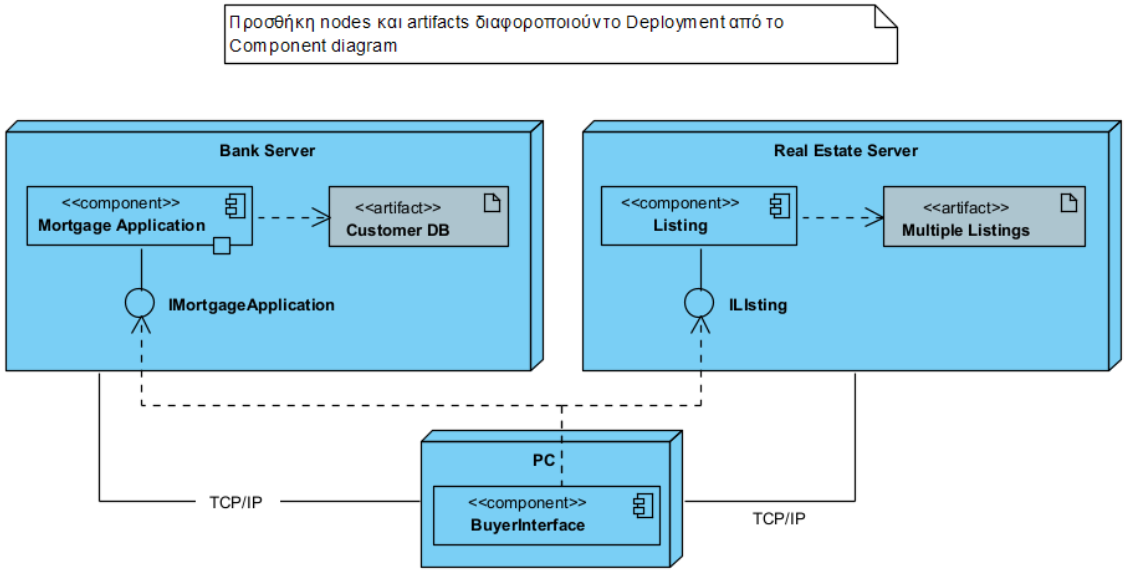
Export

Τα διαγράμματα τύπου Deployment δε διαφέρουν σημαντικά από τα Component. Τυπικά είναι υπερσύνολο των Component, με προστιθέμενα στοιχεία τύπου Node και Artifact για την περιγραφή της δομής, της εγκατάστασης και τα περιβάλλοντα των components. Στο διάγραμμα του σχήματος 4.28 παρουσιάζουμε ένα απλό παράδειγμα Deployment που περιέχει αυτά τα στοιχεία με σκοπό τον έλεγχο της λειτουργίας εξαγωγής.

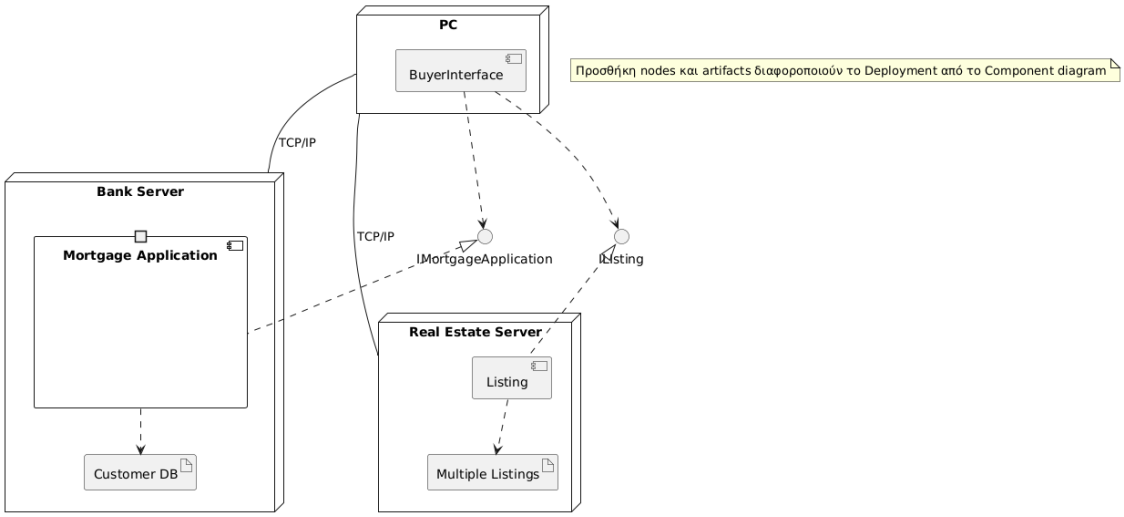
Στο σχήμα 4.29 φαίνεται το αποτέλεσμα της μετατροπής σε PlantUML όπου όλα τα στοιχεία έχουν μεταφραστεί πιστά.



Σχήμα 4.27: Παράδειγμα Component Diagram μετά την εισαγωγή



Σχήμα 4.28: Παράδειγμα Deployment Diagram για εξαγωγή



Σχήμα 4.29: Παράδειγμα Deployment Diagram μετά την εξαγωγή

Import

Για το PlantUML, ένα Deployment διαγραμμα είναι οποιαδήποτε μείξη των στοιχείων διαγραμμάτων τύπου Component, Use Case, Deployment. Έτσι υπόστηρίζουμε αυτό το συνδυασμό και για τη λειτουργία import θα χρησιμοποιήσουμε ένα τέτοιο σύνθετο διάγραμμα, όπως το Σχήμα 4.30.

Μετά την εισαγωγή, όλη η πληροφορία έχει αποδοθεί πιστά στο Visual Paradigm (βλ. Σχήμα 4.31).

Σημειώνεται ότι το PlantUML ορίζει πολλά στοιχεία εκτός από node, system, package που περιέχουν άλλα στοιχεία. Ορισμένα από αυτά παρουσιάζονται στο Σχήμα 4.32. Αυτά δεν υποστηρίζονται προς εισαγωγή, εκτός από εξαιρέσεις για τις οποίες το στοιχείο μετατρέπεται σε Node με κατάλληλο στερεότυπο.

4.7.6 Sequence Diagram

Export

Όπως είδαμε στην υποενότητα 4.3.3, τα διαγράμματα Sequence δημιουργούν προβλήματα στην ομαλή μετατροπή και στις δύο κατευθυνσεις. Για το export επιλέξαμε μετατροπή του διαγράμματος στο σχήμα 4.33. Αυτό περιέχει:

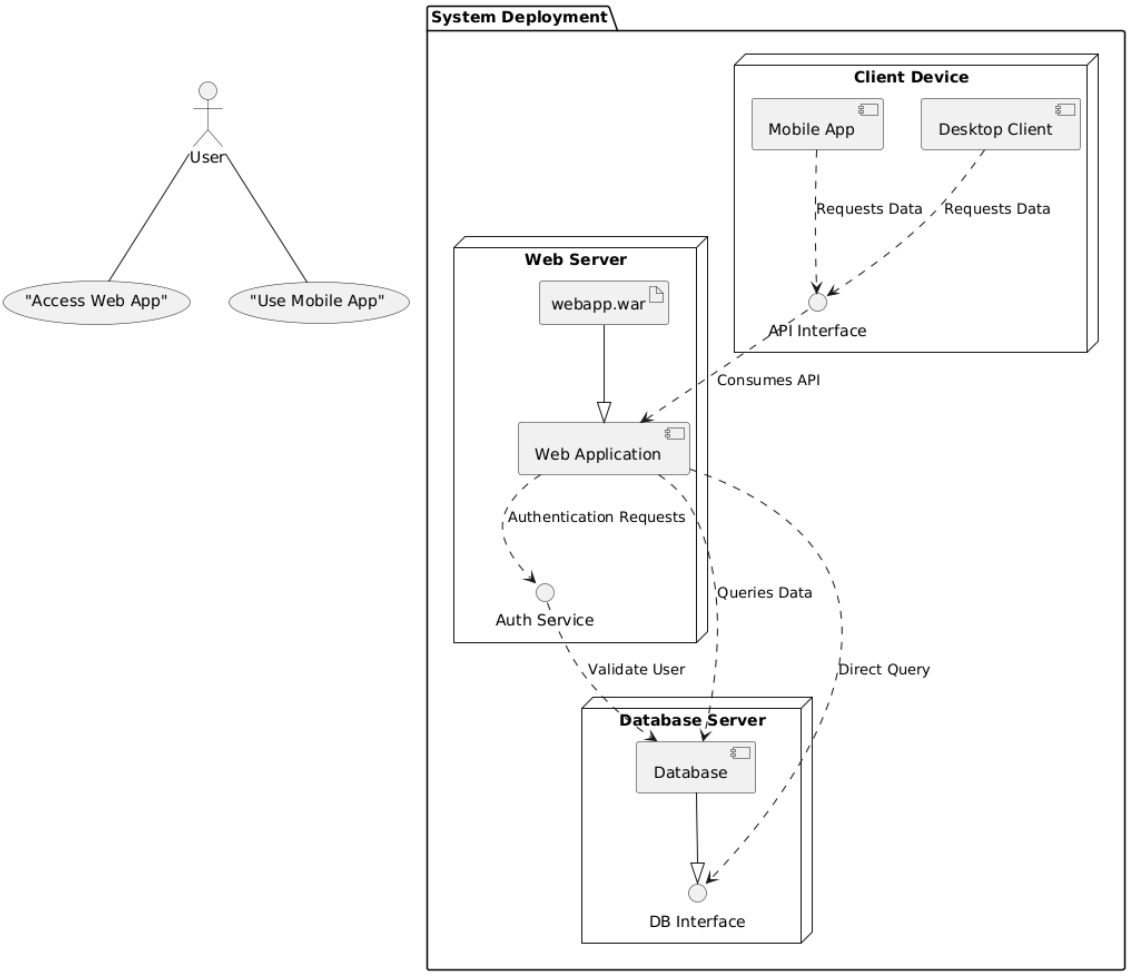
- Lifelines με Base Classifier (φαίνονται μετά την άνω και κάτω τελεία)
- Μήνυμα που βρέθηκε και μήνυμα που χάθηκε (τα 1 και 2 αντίστοιχα)
- Μήνυμα με διάρκεια (το λοξό 1.3)
- Ένα loop fragment
- Ένα alt/else fragment μέσα στο loop
- Ένα alt fragment

Το αποτέλεσμα της εξαγωγής είναι το δίαγραμμα PlantUML του σχήματος 4.34.

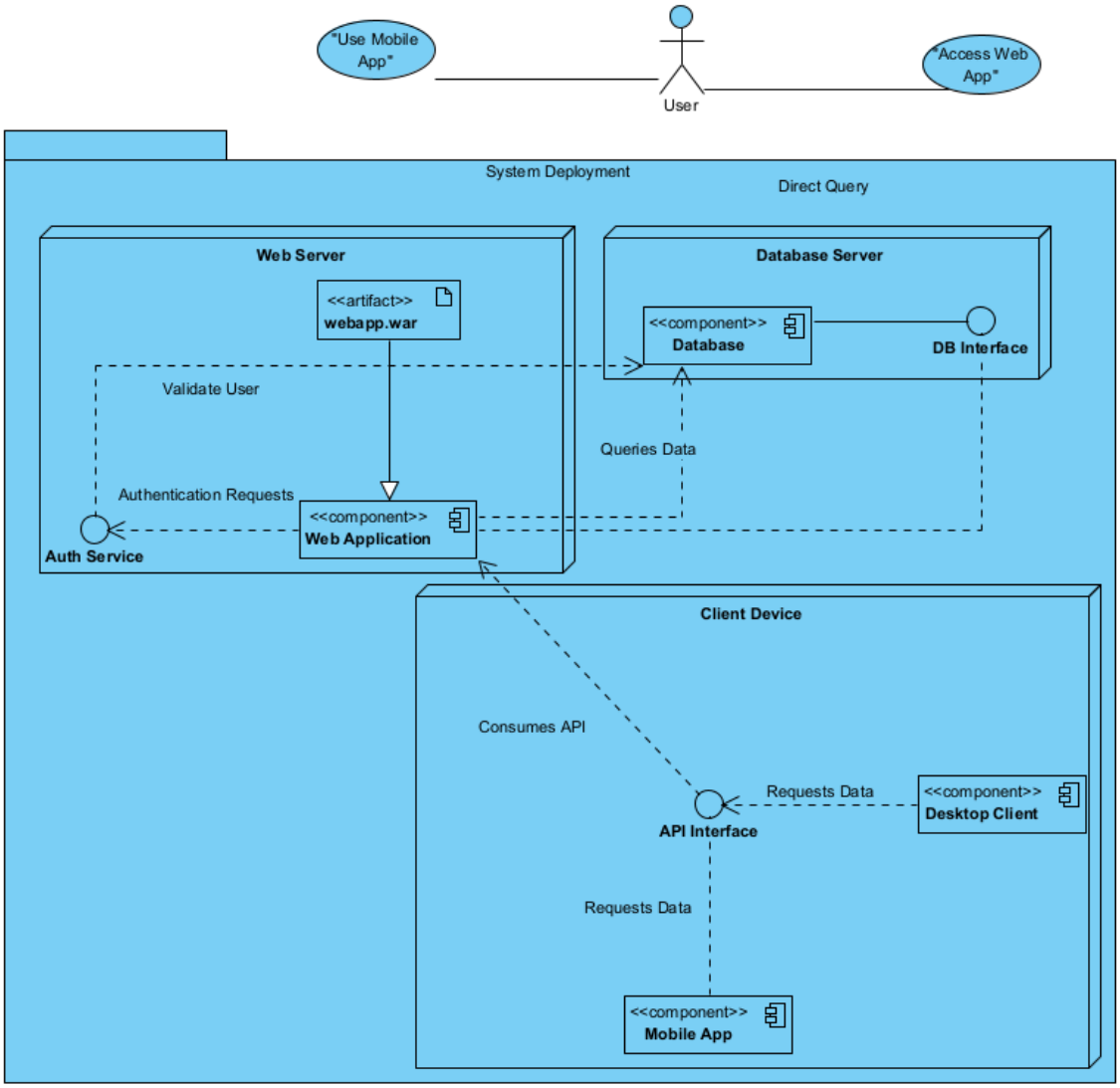
Κάποια από τα Lifelines του αρχικού διαγράμματος του visual Paradigm δεν είχαν ονόματα, μόνο Classifiers. Κάτι τέτοιο δε μπορεί να γίνει στο PlantUML, οπότε **ορίστηκαν ονόματα τα ονόματα των Classifiers** τους. Επίσης, πάνω από κάθε Lifeline προστέθηκε ένα σημείωμα που δηλώνει τον classifier του. Εδώ δε φαίνεται να έχει πολύ νόημα, αλλά σε περίπτωση που τα ονόματα των lifeline ήταν διαφορέτικα από των classifier θα ήταν χρήσιμη πληροφορία και θεωρήθηκε σημαντική να διατηρηθεί.

Τα μηνύματα διατήρησαν την αρίθμηση τους, την ετικέτα, τον τύπο και τη διάρκειά τους.

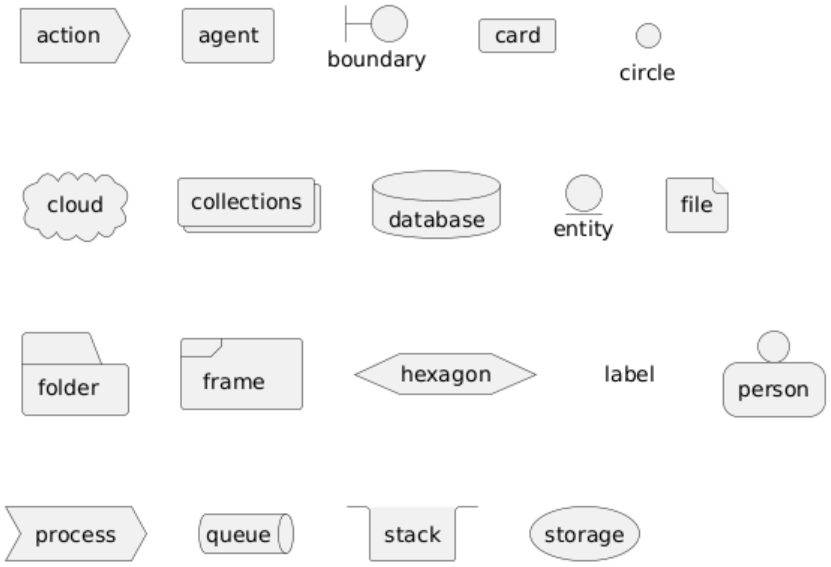
Η διάταξη είναι λίγο διαφορετική κάτι που ήταν αναμενόμενο καθώς αφήνεται να γίνει αυτόματα.



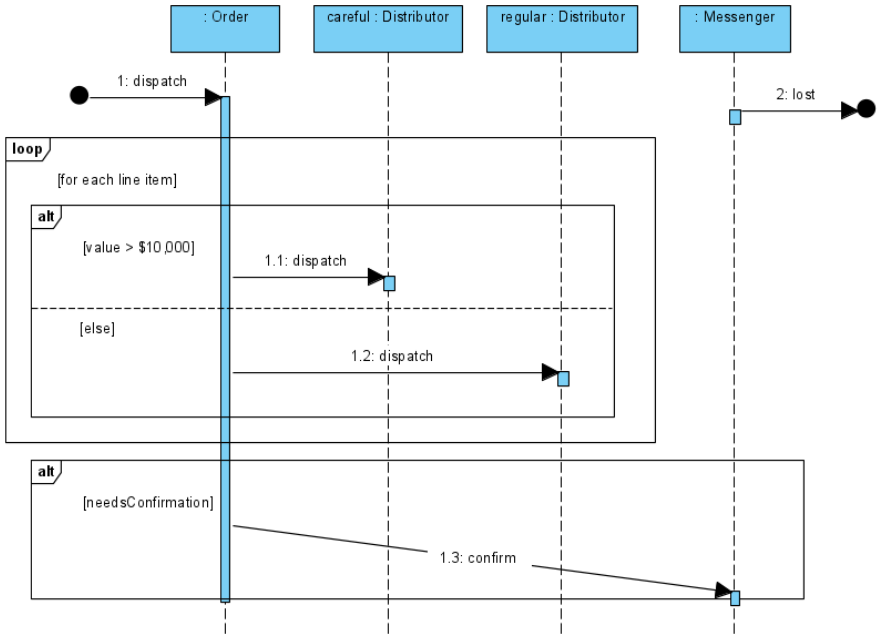
Σχήμα 4.30: Σύνθετο παράδειγμα Deployment Diagram για εισαγωγή



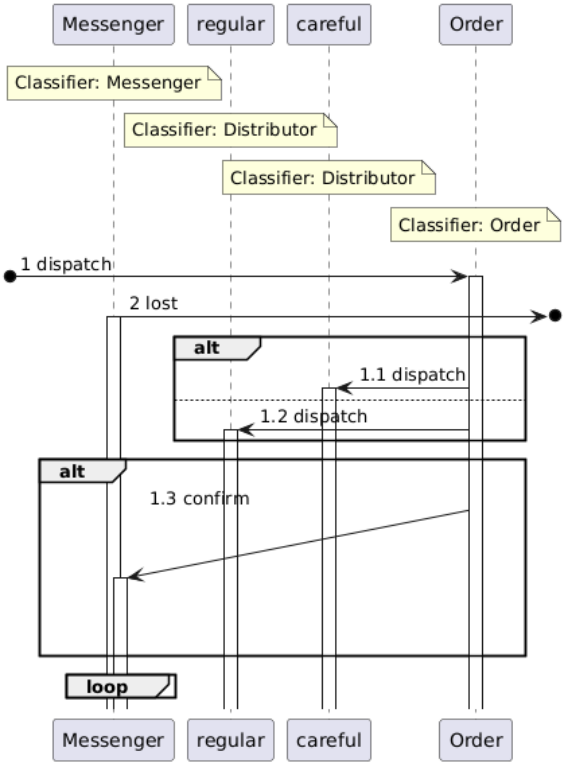
Σχήμα 4.31: Το σύνθετο παράδειγμα Deployment Diagram μετά από εισαγωγή



Σχήμα 4.32: Ειδικά στοιχεία PlantUML deployment που δεν υποστηρίζονται από το Visual Paradigm



Σχήμα 4.33: Παράδειγμα Sequence Diagram για εξαγωγή



Σχήμα 4.34: Παράδειγμα Sequence Diagram μετά την εξαγωγή

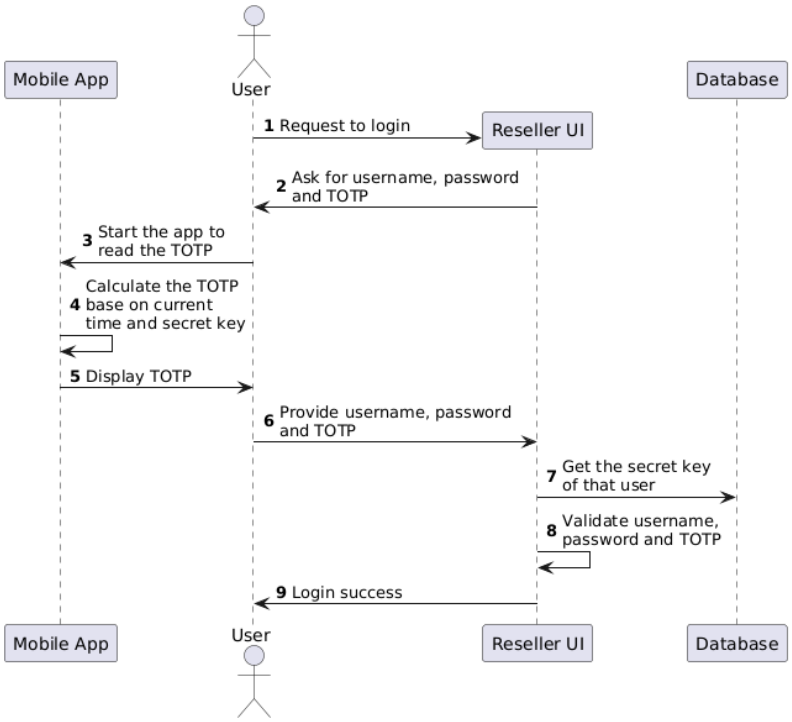
Τα combined fragments (alt/else, loop, alt) έχουν κάποιες ατέλειες στη μετατροπή. Τα μηνύματα που ανήκουν στα alt/else μεταφράστηκαν σωστά και περιέχονται στα fragments. Το loop fragment δε μετατράπηκε σωστά διότι περιέχει άλλο fragment και όχι μηνύματα. Δυστυχώς, δε βρέθηκε λύση για τη σωστή μετάφρασή του.

Σημείωση: για να μπορούν να μετατραπούν alt/else σωστά, πρέπει στο διάγραμμα του Visual Paradigm να είναι σωστά **τοποθετημένα όλα τα μηνύματα που περιέχονται σε αυτά τα fragments σε επίπεδο μοντέλων**. Συχνά, όταν δημιουργούμε Visual Paradigm διαγράμματα sequence με το χέρι, τοποθετούμε τα μηνύματα μέσα στο fragment μόνο διαγραμματικά/σχηματικά και δεν ενημερώνεται ανάλογα το μοντέλο.

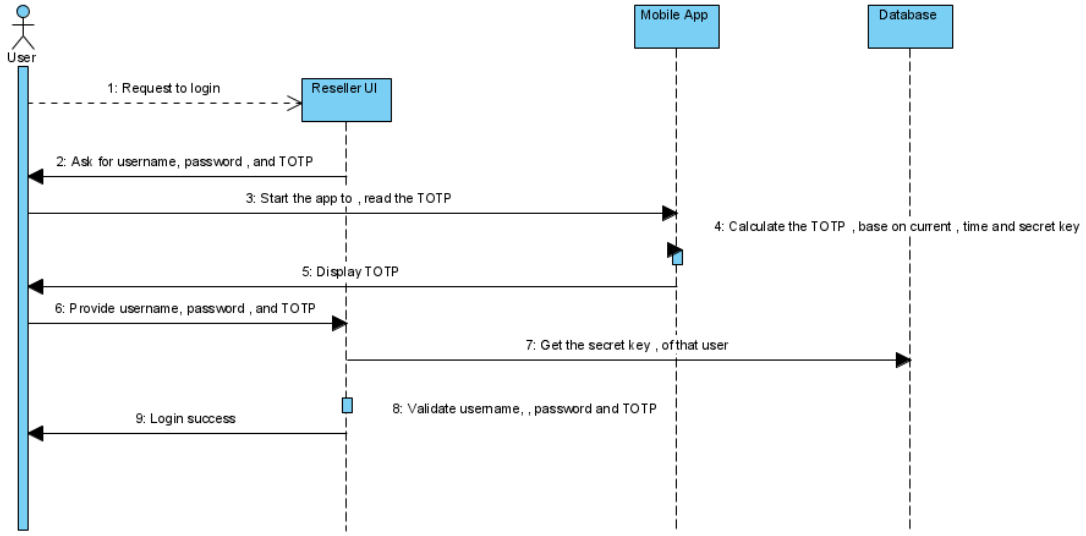
Import

Για το import δημιουργήθηκε το διάγραμμα του σχήματος 4.35. Όπως έχουμε ήδη αναφέρει, combined fragments κατά το import δε μπορούν να διαταχθούν σωστά οπότε δεν υποστηρίζονται.

Το αποτέλεσμα της εισαγωγής είναι το διάγραμμα στο Σχήμα **??**. Όλοι οι τύποι μηνυμάτων,



Σχήμα 4.35: Παράδειγμα Sequence Diagram για εισαγωγή



Σχήμα 4.36: Παράδειγμα Sequence Diagram μετά την εισαγωγή

οι αριθμήσεις, ετικέτες και άλλα στοιχεία έχουν εισαχθεί σωστά. Το μήνυμα τύπου "Create" (1) που δημιουργεί το lifeline "Reseller UI" έγινε διακεκομμένης γραμμής λόγω της αναπαράστασης του Visual Paradigm.

Προκύπτει πρόβλημα διάταξης μόνο στα μηνύματα τύπου self message (αύξοντες αριθμοί 4 και 8) τα οποία δεν είναι καλά ορατά και χρειάζονται χειροκίνητη διόρθωση καθώς **τα βέλη υπάρχουν αλλά είναι "κολλημένα" πάνω στο lifeline**. Καμία επιλογή αυτόματης διάταξης διαγράμματος ή συνδέσμων δεν έλυσε το πρόβλημα.

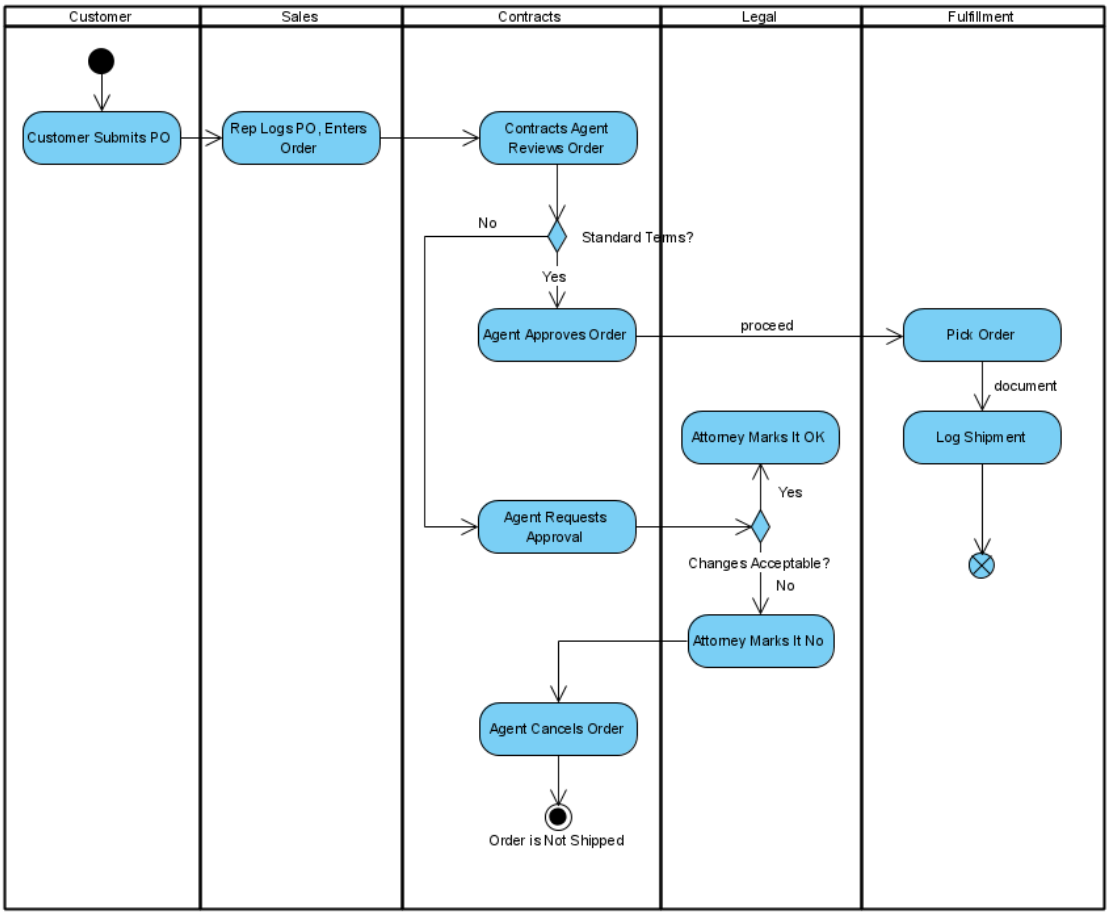
4.7.7 Activity Diagram

Τα διαγράμματα Activity παρουσιάζουν πολλές ασυμβατότητες και δυσκολίες στη μετατροπή. Ισχύουν οι περιορισμοί της υλοποίησης όπως παρουσιάστηκαν στον Πίνακα 4.1.

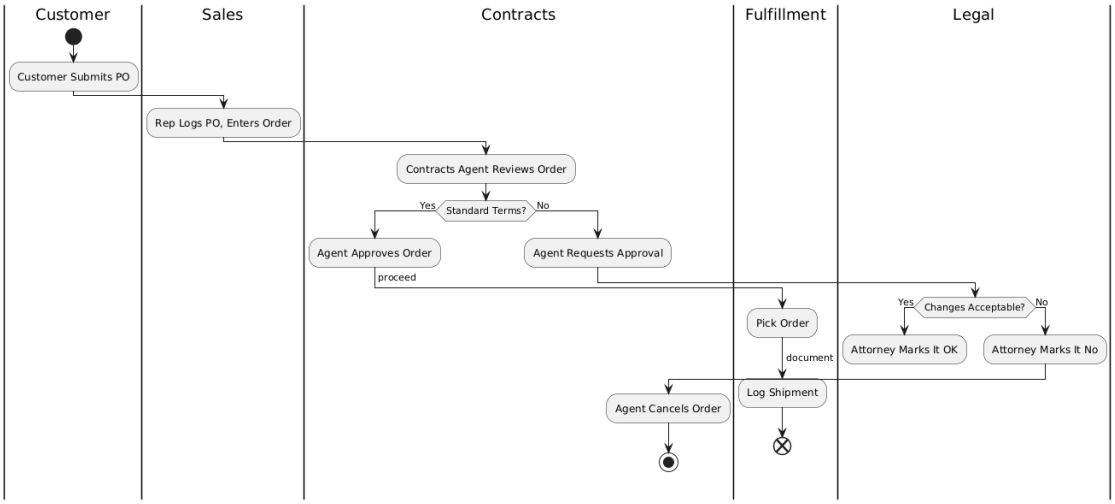
Export

Για το export χρησιμοποιήθηκε το σύνθετο διάγραμμα του σχήματος 4.37 με swimlanes, start, stop, end, decision, ετικέτες σε συνδέσμους.

Στο PlantUML διάγραμμα που προέκυψε από τη λειτουργία export παρατηρούμε διατήρηση της πληροφορίας όλων των στοιχείων. Δε προστέθηκε μόνο ετικέτα ονόματος στο τελικό κόμβο "Order is Not Shipped" καθώς δεν υπάρχει η επιλογή στο PlantUML.



Σχήμα 4.37: Παράδειγμα Activity Diagram για την εξαγωγή



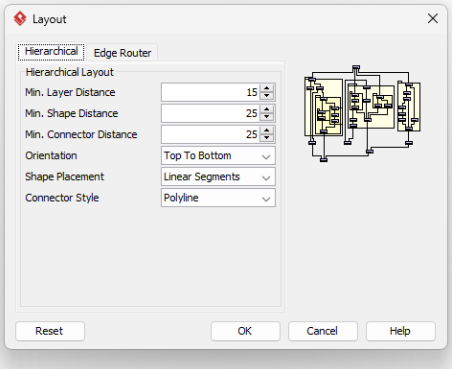
Σχήμα 4.38: Παράδειγμα Activity Diagram μετά την εξαγωγή

Import

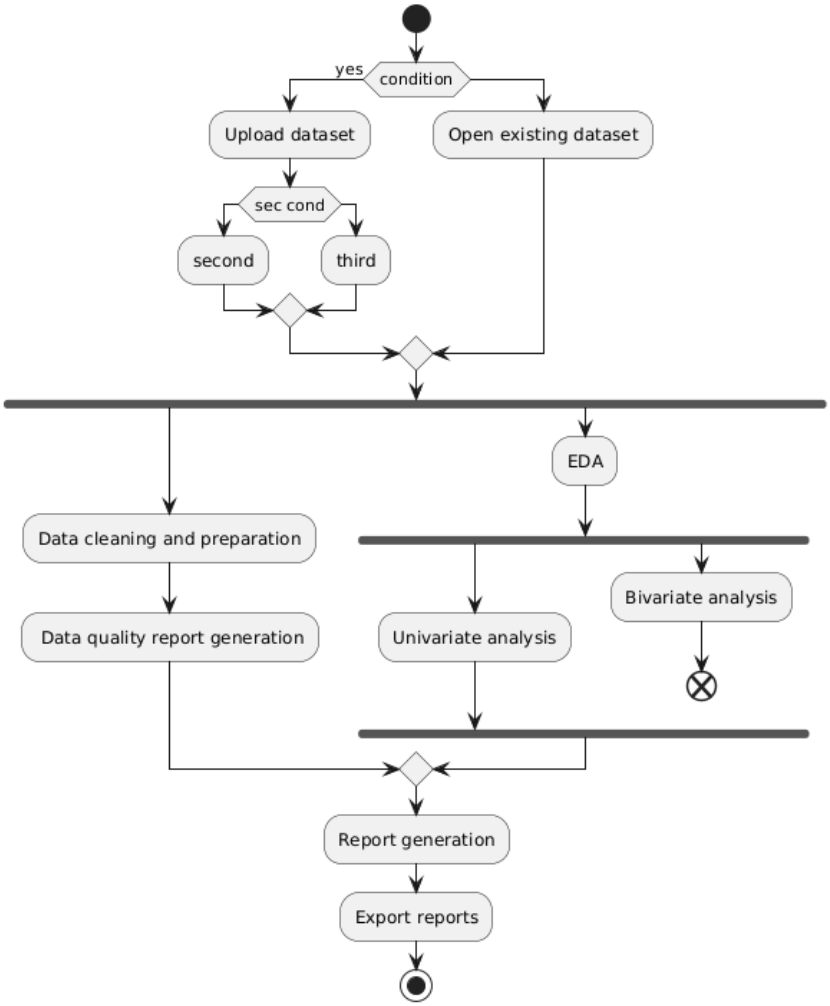
Η αυτόματη διάταξη κατά το import διαγραμμάτων Activity είναι ορισμένες φορές δυσανάγνωστη. **Προτείνεται σε τέτοιες περιπτώσεις να εφαρμόζεται custom layout** μετά την εισαγωγή σύμφωνα με τις ρυθμίσεις που φαίνονται στην Εικόνα 4.39. Το μενού εμφανίζεται με δεξί κλικ σε λευκό χώρο του διαγράμματος, στην επιλογή *Layout* > *Customized...* .

Στο σχήμα 4.40 φαίνεται το διάγραμμα PlantUML Activity προς εισαγωγή στο Visual Paradigm. Περιέχει ροή με κόμβους απόφασης, fork, join, merge, end, start, stop.

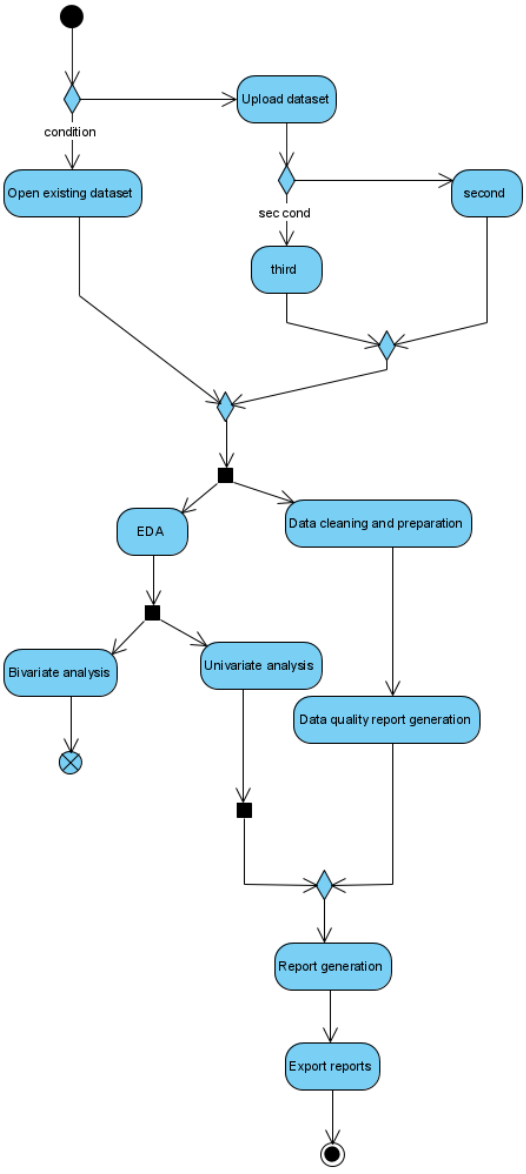
Το αποτέλεσμα της εισαγωγής στο Σχήμα 4.41 είναι πιστή μετατροπή της ροής του αρχικού διαγράμματος.



Σχήμα 4.39: Προτεινόμενες ρυθμίσεις layout για Acivity import



Σχήμα 4.40: Παράδειγμα Activity Diagram για την εισαγωγή



Σχήμα 4.41: Παράδειγμα Activity Diagram μετά την εισαγωγή

Κεφάλαιο 5

Συμπληρωματικά εργαλεία που υλοποιήθηκαν

5.1 PlantUML Plugin CLI (εργαλείο γραμμής εντολών)

Προς υποστήριξη των περιπτώσεων χρήσης της αυτοματοποίησης στις ροές εργασίας που προτείναμε, κρίθηκε αναγκαία η υλοποίηση ενός βοηθητικού εργαλείου γραμμής εντολών (CLI) για την εκτέλεση των μετατροπών, διευκολύνοντας έτσι την ενσωμάτωσή τους σε αυτοματοποιημένες διαδικασίες.

Το εργαλείο αυτό βασίζεται στη λειτουργικότητα του PlantUML plugin, και επομένως απαιτείται η εγκαταστασή του στο περιβάλλον του Visual Paradigm καθώς και τουλάχιστον ένα σχετικό έργο Visual Paradigm σε μορφή .vpp για τη χρήση του.

Η υλοποίηση βασίστηκε στην υλοποίηση του VPPCommandLineSupport interface του Visual Paradigm OpenAPI, μέσω του οποίου αναπτύχθηκαν δύο **βασικές εντολές που επιτρέπουν την εισαγωγή και εξαγωγή διαγραμμάτων** μεταξύ Visual Paradigm και PlantUML. Οι εντολές αυτές παρέχουν έναν αποδοτικό τρόπο μετατροπής, καθιστώντας εφικτή την ενσωμάτωσή τους σε μεγαλύτερα αυτοματοποιημένα συστήματα.

Για τη χρήση του PlantUML Plugin CLI, δεν απαιτείται ξεχωριστή ή πρόσθετη εγκατάσταση καθώς εγκαθιστάται πακεταρισμένο με το Plugin όταν αυτό εισαχθεί στο Visual Paradigm. Απαιτείται όμως η αντιγραφή του εκτελέσιμου. Το εκτελέσιμο που χρησιμοποιείται, Plugin.bat ή Plugin.sh αναλόγως περιβάλλοντος λειτουργικού, βρίσκεται στη τοποθεσία εγκατάστασης του Visual Paradigm στο φάκελο με τα script αρχεία (VP_installation_folder/script) και πρέπει να αντιγραφεί στον φάκελο με τα binaries (VP_installation_folder/bin).

Ο πίνακας 5.1 παρουσιάζει τα απαραίτητα ορίσματα για την εκτέλεση οποιουδήποτε plugin σύμφωνα με τις προδιαγραφές του Visual Paradigm. Ειδικά ορίσματα σχετικά με τη λειτουργία συγκεκριμένου plugin που ορίζονται από τον προγραμματιστή τοποθετούνται ως

Όρισμα	Περιγραφή	Τιμές
`-project`	Η τοποθεσία του σχετικού `.vpp` project	π.χ. `"example/testcli.vpp"`
`-pluginid`	Το αναγνωριστικό του Plugin	`"plugins.plantUML"`
`-pluginargs`	Τα ειδικά ορίσματα που σχετίζονται με συγκεκριμένο plugin	

Πίνακας 5.1: Τα απαραίτητα ορίσματα CLI για οποιοδήποτε Visual Paradigm plugin

τιμή του ορίσματος `pluginargs`. Στη περίπτωση του PlantUML plugin ήταν αναγκάια η προσθήκη ορισμάτων για την εκτέλεση των λειτουργιών όπως παρουσιάζονται στον πίνακα 5.2.

Όρισμα	Σχετική Λειτουργία	Περιγραφή	Τιμές
`-action`	`import/ export`	Επιλογή μεταξύ λειτουργίας εξαγωγής ή εισαγωγής	`"import"` ή `"export"`
`-path`	`import/ export`	Ο φάκελος ή το αρχείο όπου βρίσκονται τα διαγράμματα προς εισαγωγή ή εξαγωγή	`"test/diagram.txt"`
`-list`	`export`	Τυπώνει τη λίστα των διαγραμμάτων με το αναγνωριστικό τους	-
`-target`	`export`	Καθορίζει το διάγραμμα στόχο για εξαγωγή	ID διαγράμματος ή `"all"` για εξαγωγή όλων

Πίνακας 5.2: Τα ειδικά υλοποιημένα ορίσματα CLI για το PlantUML plugin

Παραδείγματα εντολών

Import όλων των διαγραμμάτων PlantUML του φακέλου:

```
Plugin.bat -project "C:/Demo/demo_project.vpp" -pluginid
"plugins.plantUML"
-pluginargs -action "import" -path "C:/Demo/plant_diagrams"
```

Import του διαγράμματος από αρχείο:

```
Plugin.bat -project "C:/Demo/demo_project.vpp" -pluginid
"plugins.plantUML"
-pluginargs -action "import" -path
"C:/Demo/plant_diagrams/sequence_1.txt"
```

Λίστα διαγραμμάτων του έργου:

```
Plugin.bat -project "C:/Demo/demo_project.vpp" -pluginid
"plugins.plantUML"
-pluginargs -action "export" -list

Listing available diagrams in the project:
Client Server Component | id:  lpaFx8mGAqACcQas
Class in a Package (Airline) | id:  wpVuGcmGAqAEBQ3L
Authentication Process | id:  lAJWSCmGAqACKRNG
```

Export συγκεκριμένου διαγράμματος:

```
Plugin.bat -project "C:/Demo/demo_project.vpp" -pluginid
"plugins.plantUML"
-pluginargs -action "export" -path "C:/Demo/output"
-target "lAJWSCmGAqACKRNG"
```

Export όλων των διαγραμμάτων:

```
Plugin.bat -project "C:/Demo/demo_project.vpp" -pluginid
"plugins.plantUML"
-pluginargs -action "export" -path "C:/Demo/output"
-target "all"
```

Σημείωση Υπό λειτουργία του Plugin μέσα από το γραφικό περιβάλλον του Visual Paradigm, όταν γίνεται import ενός ή περισσότερων αρχείων το project δεν αποθηκεύεται αυτόματα σε περίπτωση που ο χρήστης θελήσει να απορρίψει όλες τις αλλαγές που έγιναν κλείνοντάς το χωρίς αποθήκευση. Αυτό είναι σημαντικό γιατί το import, ιδίως σε περίπτωση σύγκρουση ονομάτων με ήδη υπάρχοντα μοντέλα στο έργο, θα επιφέρει αλλαγές στα υπάρχοντα μοντέλα που δύσκολα εντοπίζονται και απορρίπτονται με άλλο τρόπο. Αυτή η δικλείδα προστασίας σαφώς δε θα μπορούσε να λειτουργήσει από το CLI, καθώς δε θα αποθηκευόταν τίποτα στο project και η εντολή δε θα είχε κανένα αποτέλεσμα. Έτσι, όταν καλείται από γραμμή εντολών, η εισαγωγή αλλάζει "οριστικά" το αρχείο. Αυτή είναι η **μόναδική διαφορά στο τρόπο εκτέλεσης των λειτουργιών μεταξύ CLI/GUI**.

5.2 PlantUML web viewer

Η διαδικτυακή εφαρμογή του PlantUML, η οποία φιλοξενείται από την ομάδα ανάπτυξής του, αποτελεί μια λύση για όσους επιθυμούν να δοκιμάσουν τις δυνατότητες του εργαλείου και να εξοικειωθούν με τη λειτουργία του. Παρέχει ένα εύχρηστο περιβάλλον όπου οι χρήστες μπορούν να γράψουν κώδικα PlantUML και να δουν άμεσα το αντίστοιχο διάγραμμα, χωρίς να απαιτείται εγκατάσταση λογισμικού.

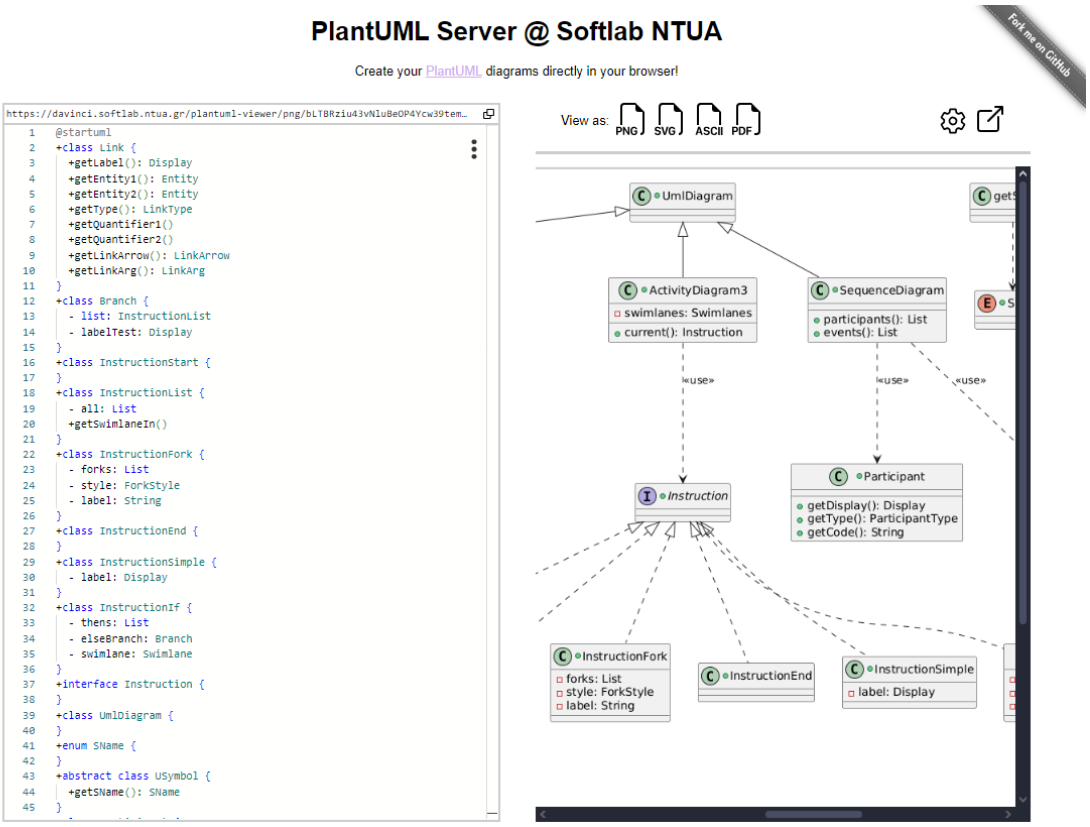
Για όσους επιθυμούν μεγαλύτερη ευελιξία και απόδοση, το PlantUML προσφέρεται και για τοπική εγκατάσταση, επιτρέποντας τη χρήση όλων των δυνατοτήτων του απευθείας από τον υπολογιστή του χρήστη. Η τοπική εκτέλεση είναι προτιμότερη για τη ταχύτερη απόκριση, με τα διαγράμματα να αποδίδονται πιο γρήγορα, ενώ παράλληλα προστατεύεται τα δεδομένα, καθώς δεν μεταφέρονται σε εξωτερικούς διακομιστές. Επιπλέον, δίνεται η δυνατότητα ρύθμισης του χρόνου ανανέωσης του διαγράμματος, ώστε ο χρήστης να προσαρμόσει το πόσο συχνά θα ενημερώνεται η οπτική αναπαράσταση του κώδικα. Ένα ακόμη χρήσιμο χαρακτηριστικό είναι η υποστήριξη μεταφόρτωσης αρχείων PlantUML, επιτρέποντας στον χρήστη να ανεβάσει έτοιμα αρχεία αντί να πληκτρολογεί τον κώδικα στον ενσωματωμένο επεξεργαστή κειμένου.

Η εφαρμογή υποστηρίζει εξαγωγή των διαγραμμάτων σε όλες τις μορφές που προσφέρονται και από τη διαδικτυακή, **PNG, SVG, ASCII και PDF**.

Στο πλαίσιο της παρούσας εργασίας και με σκοπό τη διευκόλυνση των φοιτητών που επιθυμούν να αξιοποιήσουν το PlantUML για τις εργασίες τους, **η εφαρμογή εγκαταστάθηκε και φιλοξενείται πλέον σε διακομιστή του εργαστηρίου**, στη διεύθυνση davinci.softlab.ntua.gr/plantuml-viewer/.

Στο Σχήμα 5.1 παρουσιάζεται η διεπαφή της εφαρμογής: στα δεξιά βρίσκεται ο επεξεργαστής κειμένου, όπου ο χρήστης πληκτρολογεί την περιγραφή του διαγράμματος, και στα αριστερά εμφανίζεται το παραγόμενο διάγραμμα (ή το σφάλμα που πιθανόν προέκυψε), το οποίο ανανεώνεται δυναμικά με βάση τις αλλαγές του κώδικα.

Ανέβασμα αρχείου γίνεται από τις τρεις κάθετες τελείες στο δεξιά πάνω μέρος του επεξεργαστή κειμένου, ενώ ρυθμίσεις σχετικά με το χρόνο ανανέωσης του viewer διαγράμματος και ρυθμίσεις στυλ βρίσκονται στο γρανάζι πάνω από τον viewer.



Σχήμα 5.1: PlantUML web εφαρμογή φιλοξενούμενη σε υπολογιστή του Softlab

Κεφάλαιο 6

Συμπεράσματα και Μελλοντική Εργασία

6.1 Συμπεράσματα

Συνολικά, η εργασία πέτυχε τους στόχους που τέθηκαν, καταλήγοντας σε ένα ισχυρό εργαλείο για την αμφιδρομη μετατροπή μεταξύ PlantUML και Visual Paradigm. Το plugin συμβάλλει ουσιαστικά στη διαλειτουργικότητα μεταξύ των δύο συστημάτων για την οποία δεν έχει υπάρξει προηγούμενη προσπάθεια, και ανοίγει το δρόμο για τη χρήση του PlantUML ως ενδιάμεσου μέσου για την αξιοποίηση τεχνητής νοημοσύνης στην παραγωγή UML.

Ένα από τα κύρια χαρακτηριστικά του εργαλείου είναι η προσαρμοσμένη μεταχείριση των μετατροπών, δίνοντας προτεραιότητα στην όσο το δυνατόν ακριβέστερη και πληρέστερη αντιστοίχιση μεταξύ των μοντέλων των δύο περιβαλλόντων. Αυτό είναι ιδιαίτερα σημαντικό, καθώς η διαλειτουργικότητα μεταξύ εργαλείων UML συχνά πάσχει από απώλειες πληροφορίας ή μη ακριβείς μετατροπές, κάτι που στο παρόν έργο επιχειρήσαμε να αποφύγουμε στο μέγιστο δυνατό βαθμό.

Βέβαια, η υλοποίηση αντιμετώπισε και συγκεκριμένους περιορισμούς, οι οποίοι προέρχονται τόσο από τα ίδια τα UML περιβάλλοντα μοντελοποίησης όσο και από τις ιδιαιτερότητες των τεχνολογιών που χρησιμοποιήθηκαν. Συγκεκριμένα:

- Το Visual Paradigm API παρουσιάζει ορισμένες ιδιοτροπίες, ιδιαίτερα όσον αφορά το επίπεδο της διάταξης διαγραμμάτων, οι οποίες δυσχεραίνουν την άμεση μετατροπή των διαγραμμάτων.
- Το PlantUML δεν διαθέτει ακόμη επίσημη γραμματική ή κατάλληλη προγραμματιστική διεπαφή, γεγονός που περιπλέκει και περιορίζει την ανάλυση και επεξεργασία των διαγραμμάτων του.

- Περιορισμοί και ασυμβαρότητες στη σχεδίαση των UML διαγραμμάτων και στα δυο εργαλεία δημιουργούν δεν επιτρέπουν την πλήρη αντιστοίχιση ορισμένων στοιχείων.

Παρά τις δυσκολίες αυτές, το εργαλείο αποδείχθηκε ιδιαίτερα ισχυρό μέσα από σειρά δοκιμών. Οι κυριότερες αδυναμίες εντοπίζονται στην υποστήριξη διαγραμμάτων τύπου Sequence και Activity, όπου η υλοποίηση βρήκε τα μεγαλύτερα εμπόδια λόγω των προαναφερθέντων περιορισμών του PlantUML και του Visual Paradigm.

Ωστόσο, αν εξετάσουμε τη βασική περίπτωση χρήσης, δηλαδή την μετατροπή από αυτόματα παραγμένο PlantUML κώδικα από LLMs, παρατηρούμε ότι ο κώδικας αυτός είναι συνήθως σχετικά απλός και δεν περιλαμβάνει ιδιαίτερα πολύπλοκες ή εξειδικευμένες δυνατότητες του PlantUML. Συνεπώς, οι αδυναμίες που προκύπτουν από ειδικές περιπτώσεις σύνθετων διαγραμμάτων δεν έχουν τόσο μεγάλη πρακτική σημασία.

Γενικά, το εργαλείο παρέχει μια λειτουργική λύση για τη μετατροπή μεταξύ των δύο περιβάλλοντων μοντελοποίησης, αποτελώντας μια πρώτη απόπειρα για τη διαλειτουργικότητα των δύο UML εργαλείων και ανοίγοντας νέες δυνατότητες για την αξιοποίηση γλωσσικών μοντέλων στον σχεδιασμό και τη τεκμηρίωση λογισμικού με UML.

6.2 Μελλοντική εργασία

Πιθανή μελλοντική εργασία μπορεί να σχετίζεται με τη χρήση του συγκεκριμένου εργαλείου για μεταφορά μοντέλων UML από/προς εργαλεία τεχνητής νοημοσύνης, η οποία είναι από μόνη της μια μη- παγιωμένη περίπτωση αξιοποίησης της ΤΝ στη Μηχανική Λογισμικού. Αυτό οφείλεται στο γεγονός πως, όπως έχει παρατηρηθεί σε προηγούμενες εργασίες και εργασίες που βρίσκονται σε εξέλιξη (π.χ. [14]), οι επιδόσεις των γλωσσικών μοντέλων τη στιγμή συγγραφής της παρούσας εργασίας, στην παραγωγή αρχιτεκτονικών λογισμικού και γενικότερα περιγραφών με UML, απέχουν από τις εξαιρετικές επιδόσεις τους σε προβλήματα παραγωγής πηγαίου κώδικα.

Η σύνδεση διαφορετικών διαγραμμάτων PlantUML σε ένα ενιαίο μοντέλο UML για το Visual Paradigm ή οποιοδήποτε άλλο εργαλείο που υποστηρίζει την έννοια του UML μοντέλου και τα στοιχεία του οποίου έχουν μοναδική ταυτότητα και πολλές απεικονίσεις (views), είναι μια πρόκληση για το μέλλον αυτού του έργου, η οποία μπορεί να αντιμετωπιστεί μόνο με παραδοχές και συμβάσεις χρήσης (π.χ. ονοματολογία) των διαγραμμάτων (και όχι μοντέλων) PlantUML.

Τέλος, η χρήση μιας γραμματικής για τα κείμενα PlantUML θα συνέβαλλε αναμφίβολα σε αυτή την κατεύθυνση, ωστόσο κάτι τέτοιο αποδείχθηκε σύνθετο πρόβλημα και η ίδια η ομάδα ανάπτυξης του συγκεκριμένου εργαλείου δεν το έχει επιχειρήσει.

Βιβλιογραφία

[1] G. Booch, J. Rumbaugh, and I. Jacobson, *Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley, 1999, ISBN: 0321267974.

[2] Anna Persson and Henrik Gustavsson and Brian Lings and Björn Lundell, "OSS Tools in a Heterogeneous Environment for Embedded Systems Modelling: An Analysis of Adoptions of XMI", in *Proceedings of the Fifth Workshop on Open Source Software Engineering (5-WOSSE)*, Association for Computing Machinery (ACM), 2005, pp. 39–42. DOI: 10.1145/1083258.1083267. [Online]. Available: https://doi.org/10.1145/1083258.1083267.

[3] S. Huang et al, "Towards Interoperability of UML Tools for Exchanging High-Fidelity Diagrams", in *Proceedings of the 25th Annual ACM International Conference on Design of Communication (SIGDOC '07)*, ACM, 2007, pp. 129–134. DOI: 10.1145/1297144.1297172.

[4] G. Bansal and D. Vijayvargiya and S. Garg and S. K. Singh, "An Approach to Identify and Manage Interoperability of Class Diagrams in Visual Paradigm and MagicDraw Tools", in *Contemporary Computing*, Springer, 2010, pp. 142–154. DOI: 10.1007/978-3-642-14825-5_13.

[5] Object Management Group, *XML Metadata Interchange (XMI) Specification*. OMG, 2015.

[6] Object Management Group, *Unified Modeling Language Specification*. OMG, 2017.

[7] Ozkaya, Mert, "Are the UML modelling tools powerful enough for practitioners? A literature review", *IET Software*, vol. 13, no. 5, pp. 338–354, 2019. DOI: https://doi.org/10.1049/iet-sen.2018.5409. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-sen.2018.5409.

[8] Javier Cámara and Javier Troya and Lola Burgueño and Antonio Vallecillo, "On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML", *Software and Systems Modeling*, vol. 22, pp. 781–793, 2023. DOI: 10.1007/s10270-023-01105-5.

[9] PlantUML, *PlantUML Language Reference Guide*. PlantUML, 2023.

[10] Chong Wang and Beian Wang and Peng Liang and Jie Liang, "Assessing UML Models by ChatGPT: Implications for Education", *arXiv preprint*, vol. 2412.17200, 2024. DOI: 10.48550/arXiv.2412.17200. arXiv: 2412.17200.

[11] Enteee, *Plantuml parser*, GitHub repository, Accessed: 2025-02-27, 2024. [Online]. Available: https://github.com/Enteee/plantuml-parser.

[12] Speth and Sandro and Meißner, Niklas and Becker, Steffen, "ChatGPT's Aptitude in Utilizing UML Diagrams for Software Engineering Exercise Generation", in *2024 36th International Conference on Software Engineering Education and Training (CSEET)*, 2024, pp. 1–5. DOI: 10.1109/CSEET62301.2024.10663027.

[13] Γ. Αγερίδης, "Διερεύνηση διαλειτουργικότητας εργαλείων UML", M.S. thesis, Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Ε.Μ.Π, 2024. [Online]. Available: https://dspace.lib.ntua.gr/xmlui/handle/123456789/58680.

[14] Μ. Τσιλιμιγκουνάκης, "Διερεύνηση αξιοποίησης εργαλείων LLM στην αρχιτεκτονική λογισμικού", M.S. thesis, Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Ε.Μ.Π, 2024. [Online]. Available: http://artemis.cslab.ece.ntua.gr:8080/jspui/handle/123456789/19432.

[15] Joseph Romeo and Marco Raglianti and Csaba Nagy and Michele Lanza, "UML is back. Or is it? : investigating the past, present, and future of UML in open source software", in *ICSE 2025 47th International Conference on Software Engineering*, Istituto del software (SI), Facoltà di scienze informatiche, Università della Svizzera italiana, 2025. [Online]. Available: https://ark:/12658/srd1330716.

[16] PlantUML, *Ebnf-like grammar*, GitHub issue, Accessed: 2025-02-27, 2025. [Online]. Available: https://github.com/plantuml/plantuml/issues/2039.

[17] D2, *D2 Documentation*, Accessed: January 5, 2025. [Online]. Available: https://d2lang.com.

[18] MermaidJS, *About Mermaid*, Accessed: January 4, 2025. [Online]. Available: https://mermaid.js.org/intro/.

[19] PlantUML, *PlantUML*, Accessed: January 4, 2025. [Online]. Available: https://plantuml.com.

[20] Visual Paradigm, *Open API Archives*, Accessed: January 5, 2025. [Online]. Available: https://knowhow.visual-paradigm.com/openapi/.