|  |
| --- |
| TU Wien – Institut für Computertechnik |
| Gardener – Java to plantUML parser |
| Software Requirements Specification (SRS) |

|  |  |  |  |
| --- | --- | --- | --- |
| Gruppe | Autor(en) | | MatrikelNr. |
| 5 | Krevastouli Eirini | | 11845882 |
| Telleria Manuel | | 01429517 |
| Ausgabe-Datum: 16.12.2021 | | Version: 1.0 | |

# Table of contents

[0 Table of contents 1](#_Toc94201079)

[0.1 Version History 2](#_Toc94201080)

[1 Introduction 3](#_Toc94201081)

[2 General Task description 3](#_Toc94201082)

[2.1 Problem Analysis 3](#_Toc94201083)

[2.2 Software purpose 4](#_Toc94201084)

[2.3 External relations 4](#_Toc94201085)

[2.4 Restrictions and conditions 4](#_Toc94201086)

[3 REQUIREMENTS 6](#_Toc94201087)

[3.1 UML Use-Case diagram 6](#_Toc94201088)

[3.2 Use Case Reports 6](#_Toc94201089)

[3.2.1 Case: Provide source code 7](#_Toc94201090)

[3.2.2 Case: Generate PLANTUML SYNTAX 8](#_Toc94201091)

[3.2.3 Case: Generate DIAGRAM IMAGE 9](#_Toc94201092)

[3.3 Domain model 11](#_Toc94201093)

[3.4 Functional requirements 11](#_Toc94201094)

[3.4.1 Set source path: 11](#_Toc94201095)

[3.4.2 Choose method 11](#_Toc94201096)

[3.4.3 Set parser options 11](#_Toc94201097)

[3.4.4 Parse method 11](#_Toc94201098)

[3.4.5 Generate plantUML syntax 12](#_Toc94201099)

[3.4.6 Generate File 12](#_Toc94201100)

[3.4.7 Generate Image 12](#_Toc94201101)

[3.5 Restrictions 12](#_Toc94201102)

[4 Responsibilites of stakeholder 12](#_Toc94201103)

[5 External References 13](#_Toc94201104)

[6 Annex 13](#_Toc94201105)

## Version History

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Release notes** |
| 0.1.0 | 2021-11-20 | Creation of the Document |
| 0.1.1 | 2021-12-15 | Preparing document for first revision |
| 1.0 | 2021-27-01 | Final Revision |

Tabelle 1: Versions History

# Introduction

The PlantUML language provides an easy way for developers to quickly generate UML diagrams from text files. The goal of this project is to further simplify this process by providing a java software library to automatically generate UML sequence diagrams from the source code of valid java (1.8 extension) methods.

The purpose of this document is to precisely specify the expected capabilities of the Gardener library as well as the functionality required to fulfill the task at hand as defined by the primary stakeholders. To accomplish this, it will explain the features of the library, the interfaces of the library, the purpose of the library, as well as the actual capabilities of the library along with the constraints under which it will operate.

This document also deals with the general description of the Project and the definition of the specific requirements for the solution to be developed. Furthermore, the framework conditions for the development process are determined and an overview of the use of the solution in the defined domain is given.

The document is intended for both the stakeholders and the developers of the Gardener library and will be proposed to the primary stakeholders for approval.

# General Task description

The core task of the project consists in the Creation of Sequence Diagrams for java methods.

It will be necessary to parse information from the source code of a valid java (1.8 extension) method and map the obtained information into the proper PlantUML syntax for generating UML sequence diagrams while requiring as little user interaction as possible.

## Problem Analysis

PlantUML is an open-source tool used to create diagrams from plain text language. The result of this project should be a text file with the PlantUML syntax necessary for the creation of the corresponding sequence diagram.

The task at be broadly divided into two main subtasks:

1. Parsing the source code of a java method for information.
2. Creating a text file with the proper PlantUML syntax for generating a sequence diagram.

The task of parsing the source code will primarily be carried out by parsing the AST (Abstract syntax tree) of the chosen method as generated by the JavaParser API ([JavaParser - Home](https://javaparser.org/)). Information about the method, its scope and context can be obtained by traversing the generated AST. This information should then be stored in a format that allows the second subtask to be easily accomplished

The task of Generating the valid PlantUML syntax consists of Implementing as many functions of the PlantUML syntax as necessary to display all the information obtained from the method source code. This requires mapping the information obtained in the parsing step to generate the appropriate strings as needed to create the corresponding UML sequence diagram.

## Software purpose

The purpose of the project is to help stakeholders generate UML sequence diagrams from the source code of valid (1.8 extension) java methods. Furthermore, the project aims to make the process of generating UML sequence diagrams automatically from valid java source code easier.

The solution to be developed is intended to derive and collect information from the given source code and to save this information in such a way that a plantUML sequence diagram can be generated.

## External relations

The project exists at the boundary of OOD (object-oriented design) and OOA (object-oriented analysis) since it deals with the abstraction (analysis) of concrete implementations (designs) of java methods and the display of the extracted information such that it may be used by the stakeholder to carry out further design or analysis of the system under test.

The project must interact with a potential user in such a way that allows the stakeholder to quickly and easily generated a valid sequence diagram from a chosen java method.

The project must interact with the JavaParser API to obtain a valid AST that can then be parsed for information on method, its scope and context.

The project must interact with the plantUML language, and its implementation of a syntax designed to quickly generate UML sequence diagrams.

## Restrictions and conditions

The project must use a three-tier architecture during the software development process.

The solution must be developed as part of an iterative/incremental development process, which consists of -at least- two Iterations. After each Iteration a meeting will be held between the developers and the stakeholders to clarify, explain and answer questions and eventually give feedback and recommendations for improvement.

The first development phase will take place from November 15th to December 20th.  
The goal of the first development phase is to complete the Software Requirements Specification (SRS) Document.

Between 20-12-2021 and 22-12-2021 a mid-submission appointment will take place with the stakeholders to discuss progress made and suggest changes if necessary.

The second development phase will take place from December 22nd to January 22nd is to complete The Software Design Description (SDD) Document.

After the successful completion of the second development phase, a meeting must follow where the developers submit the final source code along with the two required documents (SRS and SDD). The stakeholders will evaluate the submitted work and perform a review session with the developers.

The project must primarily use the Java (1.8 extension) language. The project will be developed as a java (1.8 extension) library.

The process of generating a text file with the proper syntax for a PlantUML sequence diagram is not constrained to the Java language.

The project must have a time to market of less than 1 business year.

The project will be developed as an open-source application using github for hosting and version control

The project will be delivered to stakeholders in the form of a Java (1.8 extension) software library packaged in a JAR file. Management of to the github repository will be handed over to the primary stakeholder at the end of the development cycle.

Any external libraries used must be open-source and have the appropriate licenses as approved by the primary stakeholder

# REQUIREMENTS

## UML Use-Case diagram

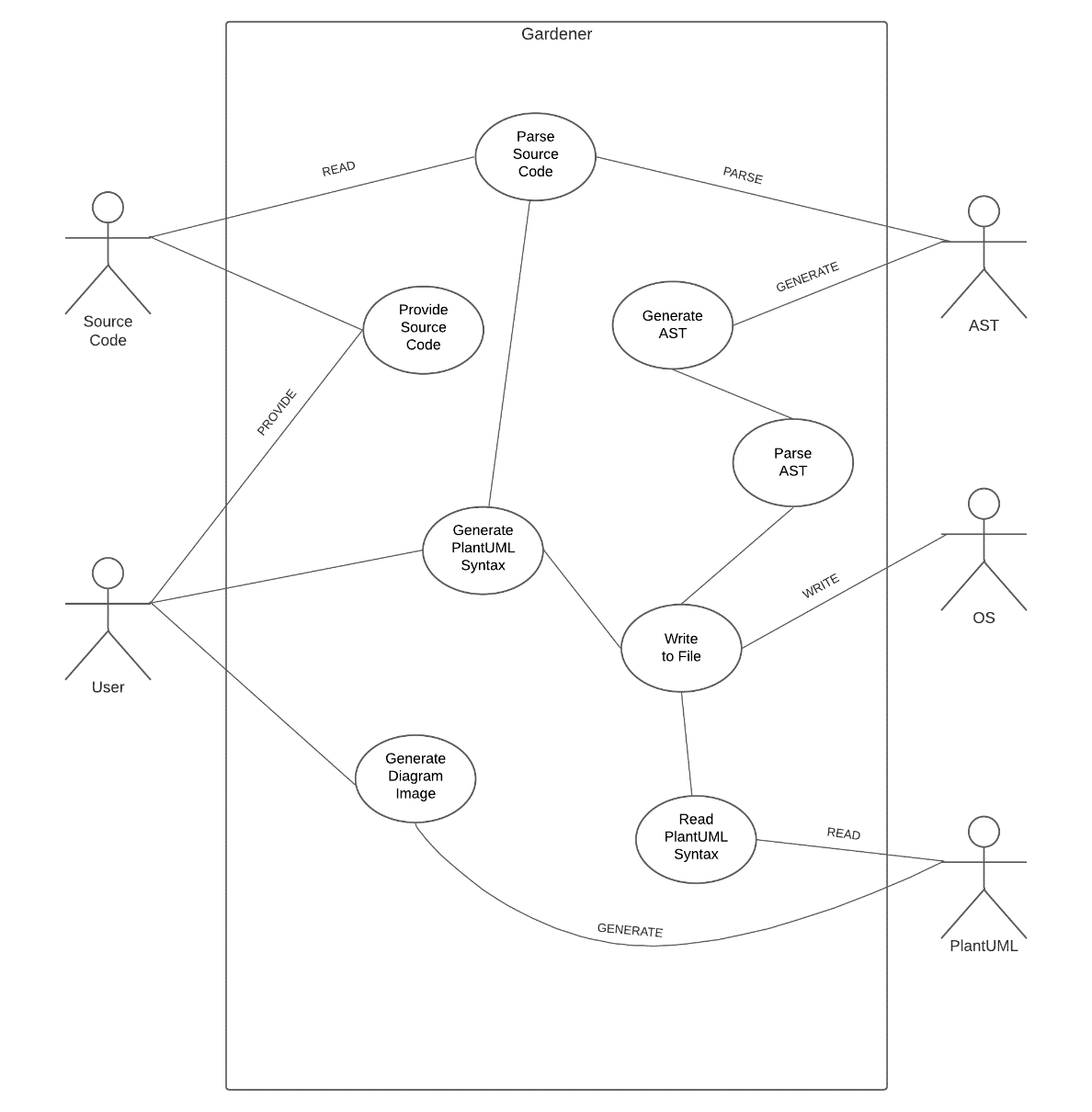


Diagram 1: use case diagram

## Use Case Reports

All relationships between the individual use cases are shown in Diagram 1  
In this Diagram we see five actors:

* User: represents the user that provides the system with the source code.
* Source Code: holds the needed information.
* AST(Abstract Syntax Tree): generates the AST and parses the source code.
* OS(Operating System): writes the PlantUML syntax to file.
* PlantUML: reads the plantUML syntax and generates a diagram image.

### Case: Provide source code

**Description:**

The user wants to provide the program access to the source code where the method to be analyzed is implemented in.

**Pre-requisites:**

The method to be analyzed must exist as valid Java 1.8 extension source code and the user must have access to this source code and be able to provide the program the correct system path to the source code.

**Scenario:**

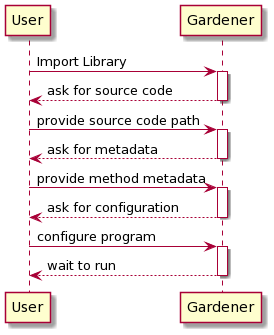
****

Figure 1:Provide Source code case

**Alternatives:**

Should the user fail to provide the path to a valid source code file the program should warn the user in the form of an exception and then terminate safely

**Post-requisites:**

A reference to a file which contains valid java 1.8 extension source code implementing the method the user wants to analyze.

A reference to the metadata needed to parse the chosen method.

The valid configuration of program parameters.

**Goal:**

The program holds a reference to a file *By-Scenario* Provide source code path

The program holds a reference to the necessary metadata *By-Scenario* Provide method metadata

The program has a valid configuration *By-Scenario* configure program

**Extension Points:**

None

**Includes:**

None

**Functional Requirements:**

Provide source code path *By-Function* Set source path

Provide method metadata *By-Function* Choose method

Configure the program *By-Function* Set parser options

### Case: Generate PLANTUML SYNTAX

**Description:**

The user wants to generate a text file with the plantUML syntax necessary to generate a sequence diagram of the chosen method

**Pre-requisites:**

The program must hold a reference to a file with a valid java source code implementation of the chosen method.

The program must hold a reference to the metadata needed to parse the chosen method

**Scenario:**

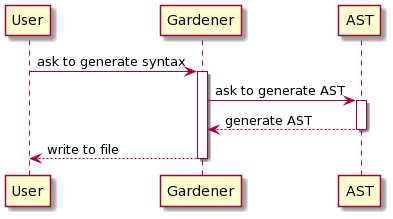
****

Figure 2: Generate Syntax case

**Alternatives:**

Should the program not have access to valid source code the program should warn the user in the form of an exception and then terminate safely.

Should the program not have access to the necessary metadata the program should log the error and terminate silently

**Post-requisites:**

A file containing the necessary plantUML syntax to generate a sequence diagram of the method chosen by the user

**Goal:**

A file containing the necessary plantUML syntax is written to disk file *By-Scenario* write to file

**Extension Points:**

Different parsing strategies can be implemented

**Includes:**

Provide source code.

Parse source code

Generate AST

Parse AST

Write to file

**Functional Requirements:**

Parse AST *By-Function* Parse method

Generate syntax *By-Function* Generate plantUML syntax

Save syntax to file *By-Function* Generate File

### Case: Generate DIAGRAM IMAGE

**Description:**

The user wants to generate an image file from the plantUML syntax necessary to generate a sequence diagram of the chosen method

**Pre-requisites:**

The program must hold a reference to a file with the plantUML syntax necessary to generate a sequence diagram of the chosen method

**Scenario:**

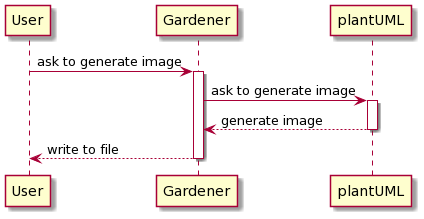
****

Figure 3: Generate image case

**Alternatives:**

Should the program not have access to a file with plantUML syntax the program should warn the user in the form of an exception and then terminate safely

Should the plantUML syntax be invalid the program should terminate safely. Further error handling is then delegated to the chosen plantUML implementation

**Post-requisites:**

A file containing an image of the sequence diagram corresponding to the method chosen by the user

**Goal:**

Provide the user an image of the sequence diagram *By-Scenario* write to file

**Extension Points:**

none

**Includes:**

Read plantUML syntax

**Functional Requirements:**

Save image to file *By-Function* Generate Image

## Domain model

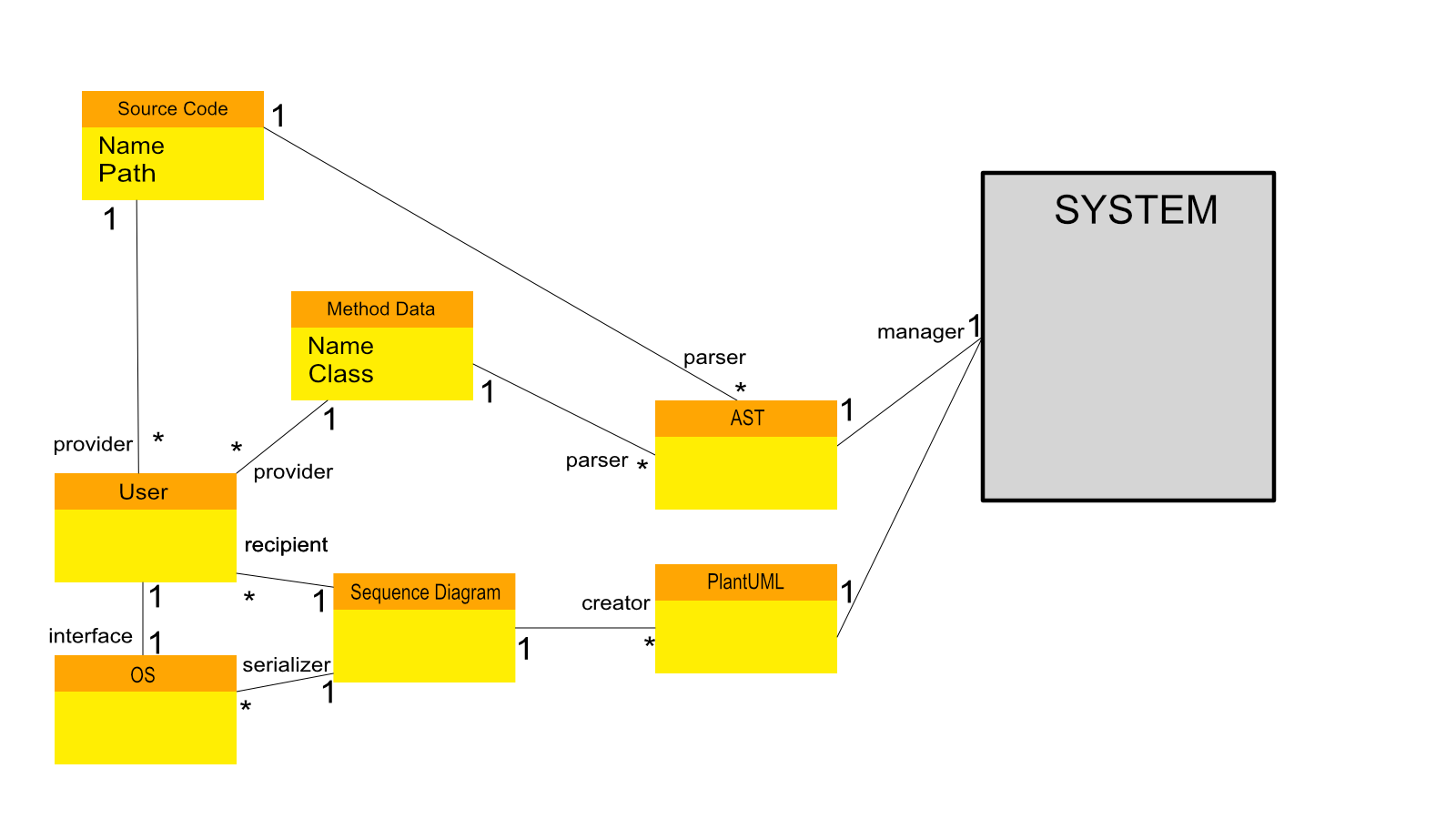


Figure : Domain Model

## Functional requirements

### Set source path:

The user shall provide the system path to a file containing valid 1.8 java source code implementing the method the user wishes to create a sequence diagram of.

### Choose method

The user shall provide the necessary metadata for the parser to traverse the AST and extract information on the method the user wishes to create a sequence diagram of.

### Set parser options

The user shall configure the parser as necessary for the creation of the sequence diagram.

### Parse method

The parser shall traverse the AST and extract any information necessary for the creation of the text file containing the plantUML syntax necessary to create the sequence diagram of the method chosen by the user.

### Generate plantUML syntax

The program shall generate the plantUML syntax necessary to create the sequence diagram of the method chosen by the user.

### Generate File

The program shall save the plantUML syntax to a file at the system path chosen by the user

### Generate Image

The program shall provide the chosen plantUML implementation with the file containing the syntax necessary to create an image of the sequence diagram of the method chosen by the user

## Restrictions

The program will only work on valid methods implemented using the java 1.8 extension language.

The program is limited by the functions provided by the plantUML language for sequence diagram generator.

The program is limited by the information that can be obtained from the AST provided by the JavaParser library.

The program will not parse lambda expressions.

The program will not resolve generic data types.

# Responsibilites of stakeholder

The Stakeholder is committed to providing the developers one round of feedback after the first iteration phase of the project.

# External References

General Information about the external library JavaParser used for generating abstract syntax trees can be obtained at [JavaParser - Home](https://javaparser.org/).

The specific functionality of the JavaParser library is documented at [javaparser-core 3.24.0 javadoc (com.github.javaparser)](https://javadoc.io/doc/com.github.javaparser/javaparser-core/latest/index.html)

General Information about plantUML and the plantuml external library used for generating the sequence diagram images can be obtained at [plantuml.com](https://plantuml.com/)

The software design specifications document should always be provided as a package together with this document.

# Annex

The following documents and files should always be provided as a package together with this document:

Domain model diagram:

domainmodel.png

Use case diagram:

usecase.png

Provide source code use case diagram:

provide.png

Generate PLANTUML SYNTAX use case diagram:

syntax.png

Generate DIAGRAM IMAGE use case diagram:

diagramfile.png

Software design specifications document:

SDD.doc