Beulah Works LLC

Software Requirement Specification (SRS) Version 1.1

UML Sequence Diagram File Generator

February 20, 2019

Isis Curiel, Bruno Hnatusko III, Brayden Mccoy,

Dhyey Patel, Jesse Primiani, Jacob Taylor,

and Syed Arshiyan Ali Zaidi

Approvals:

Title	**Signature**	**Date**
Project Manager	Isis Curiel	03-01-19
Requirement Engineer	Jacob Taylor	03-01-19
Requirement Engineer	Sayed Arshiyan Ali Zaidi	03-01-19
Software Architect	Jesse Primiani	03-01-19
Software Architect	Brayden McCoy	03-01-19

Revision History

Date	Revision	Description	Author
02-24-2019	0.00	Created	Jacob Taylor
02-26-2019	0.01	Add sections/ edited table of contents	Isis Curiel
02-28-2019	0.20	Modified Sections 1 and 2 to improve the requirement descriptions given in them.	Jesse Primiani
02-28-2019	0.50	Expanded upon Section 3 and added missing details and class descriptions.	Jesse Primiani
03-01-2019	1.0	Made revisions for final verison	Isis Curiel
03-30-2019	1.1	Added more system requirements (R5_*)	Jesse Primiani

Table of Contents

1 Introduction

1.1. Purpose

The purpose of this document is to give a description of the requirements for the "UML Sequence Diagram File Generator" software. Sections 1 and 2 are intended as a high-level overview of the library and its requirements, and are meant to be easily read by a non-developer. Section 3 is intended as a detailed analysis of the library and its requirements, meant primarily for developers. This document is intended to be a reference for developing the first version of the system for the development team.

1.2. Scope

This document covers the requirements for the "UML Sequence Diagram File Generator" that will be used by Beulah Works LLC. It will illustrate the purpose and complete declaration for the development of the system. It will also explain system constraints, interfaces, and interactions with other external applications.

1.3. Definitions, Acronyms, & Abbreviations

SRS - Software Requirement Specifications.

Requirement Analysis - encompasses those tasks that go into determining the needs or conditions to meet for a new product taking account of the possibly conflicting requirements of the various stakeholders, analyzing, documenting, validating and managing software requirements.

High-level Requirements / **Low-level Requirements** - used to classify, describe and point to specific goals of a systematic operation.

Functional Requirements - desired operations of a program, or system.

Nonfunctional Requirements - specifies criteria that can be used to judge the operation of a system, rather than specific behavior.

1.4. References

Please refer to the Requirement Specification document in Appendix 4.

1.5. Overview

The UML Sequence Diagram File Generator will generate a file from a Java SDM object. From this stage the user will be able to import the file into Lucidchart where they are free to view/edit the sequence diagram.

2 Overall Description

2.1. Product Perspective

The "UML Sequence Diagram File Generator" will be a library packaged as a .jar file for use by Beulah Works LLC.

2.1.1. System Interfaces

The UML Sequence Diagram File Generator will use the SDM and Aspose libraries to create a file from a SDM Java object. The communication between these two entities will be moderated by the classes in this project.

2.1.2. User Interfaces

A Java Object created using this library will be used as the means to generate a file from a SDM Object. See Section 3 for the class name and methods given by this project that are used to accomplish this.

2.1.3. Hardware Interfaces

Since this product does not have any designated hardware, it does not have any direct hardware interfaces.

2.1.4. Software Interfaces

Java has to be installed on the machine that runs any program made to use this library.

2.1.5. Communication Interface

The output file generated by this library must be imported into and readable by Lucidchart.

2.1.6. Memory Constraints

Since this product is just taking in a sequence diagram model object and generating a file, there are no specific primary or secondary memory constraints at this time.

2.1.7. Operations

This library will have the user give a Java SDM model object to its primary interfacing object. Before the software continues with its next steps, it must ensure that it received a valid SDM model object. From there, this product will extract the sequence diagram. From this extraction, this product will generate a file in the requested format and location.

2.1.8. Site Adaptation Requirements

This section is left intentionally blank.

2.2. Product Functions

The "UML Sequence Diagram File Generator" must take in a Java sequence diagram model object as input, extract the sequence diagram information from the model, then generate a file, such as a Visio file. The user then should be able to import the file into Lucidchart where the user can view and edit the sequence diagram. The sequence diagram in Lucidchart may not be well-aligned nor easily readable. It will contain all the information located in the given sequence diagram however. See the Appendixes for more details.

2.3. User Characteristics

Since this is a library for creating files from Java SDM Objects, only software developers associated with Beulah Works LLC will be directly making use of this project.

2.4. Constraints

1. Java will be used as the programming language for this library.
2. This library will only be available for other Java projects.
3. Documentation and Java Library deliverables are only given in English.
4. Only those associated with Beulah Works LLC are permitted to use this software.

2.5. Assumptions and Dependencies

The SDM and Aspose libraries are used in this project. This project should be able to cope with future changes in either of these dependencies, and be able to add new output file formats.

2.6. Apportioning of Requirements

Requirements must be implemented in the order they are listed in Section 3.6. These represent future system tests, and all the functionality required to implement them are considered part of the requirements.

3 Specific Requirements

3.1. External interface requirements

3.1.1. User interfaces

A Java Object created using this library will be used to convert a Java Sequence Diagram Model Object from Beulah Work's SDM project to a file. The class that will implement this functionality is called "SDMtoFile", and its relevant methods are setOutputType(), setOutputFile(), and exportFile().

3.1.2. Hardware interfaces

This product must run on a computer with preferably the latest version of Java installed, or Java 8.0 as a minimum.

3.1.3. Software interfaces

This UML Sequence Diagram File Generator project will access the SDM library and class files located in the Beulah Works repository, and will use the Aspose.Diagram library to create a file from this information. The communication between these two entities will be moderated by the classes in this project as specified in the SDD.

3.1.4. Communication interfaces

The output file generated by this library must be imported into and readable by Lucidchart, since future Beulah Works projects may automate this task. This will allow us to visually test the results of this project and verify whether it will be useful in future Beulah Works projects.

3.2. Classes/Objects

Only the primary interface class used to interact with this library is described here. The SDD contains descriptions for the remainder of the classes created for this project.

3.2.1. SDMtoFile

3.2.1.1. Attributes

Overwrite: Boolean
filePath: String
fileName: String
fileType: OutputType
outputAdapter: OutputAdapter

3.2.1.2. Functional requirements

This class is responsible for converting a SDM object to a file, and serves as the primary interface to the rest of this package, described completely in the SDD.

3.2.1.3. Events

Either the constructor or the setOutputType(OutputType type) and setOutputFile(String path, String name) methods will be used to specify the output file's extension, path, and name. The extension ".ext" part of the filename in the file parameter should be omitted, as it will be appended automatically using the type information given. The setOutputAdapter(OutputAdapter adapter) and exportFile(InputAdapter diagram) methods will then be used to create this file using the information in the object contained in the parameters of these methods.

3.3. Performance requirements

Exporting a file from a SDM object containing 100 elements or less should take no longer than two seconds. A few basic system tests that monitor the library's overall performance must be done to ensure this.

3.4. Design constraints

This project will be designed using UML and object-oriented design methods. Java must be used as the project's programming language, and this project's library deliverable will be used in other Java programs. The Java SDM Model may change its format, as well as Aspose.Diagram. Thus, this project should be able to cope with future changes in either of these dependencies. This project should preferably be able to switch out dependencies and use different libraries to implement the required functionality stated in this document, if necessary. Other output file formats should be relatively easy to be add to this library as well.

3.5. Software system attributes

Reliability: This project should not fail more than once per 100 files.
Availability: This project should be available to be implemented on any application using Java 8 or later by Beulah Works LLC.

Security: Only developers associated with Beulah Works LLC should have access to this project.

Maintainability: It should be relatively straightforward to accommodate changes in either of this projects 2 dependencies, as well as to add new output file formats.

3.6. Other requirements

Below are examples of the intended functionality of this application, expressed as system tests.

R1_*:

R1_1: An empty file should be exported and successfully imported into Lucidchart.

R1_2: A single class without an object name attached to it should be exported from this program and successfully imported into Lucidchart.

R1_3: A single object of a class should be exported from this program and successfully imported into Lucidchart.

R1_4: Two classes without an object name attached to either should be exported from this program and successfully imported into Lucidchart.

R1_5: Two objects of a class should be exported from this program and successfully imported into Lucidchart.

R1_6: A single object of a class and a single class without an attached object should be exported from this program and successfully imported into Lucidchart.

R2_*:

R2_1: A single activation block should be exported from this program and successfully imported into Lucidchart.

R2_2: A single activation block and an object of a class should be exported from this program and successfully imported into Lucidchart.

R2_3: Two activation blocks and an object of a class should be exported from this program and successfully imported into Lucidchart.

R2_4: Two activation blocks and two objects of a class should be exported from this program and successfully imported into Lucidchart.

R3_*:

R3_1: A single activation block and an object of a class, connected via a dotted line, should be exported from this program and successfully imported into Lucidchart.

R3_2: Two activation blocks and an object of a class, connected via a dotted line for the top block and a solid line for the bottom, should be exported from this program and successfully imported into Lucidchart.

R3_3: Two activation blocks and two objects of a class, where each object is connected to a single activation block via a dotted line, should be exported from this program and successfully imported into Lucidchart.

R3_4: Four activation blocks and two objects of a class, where each object is connected to a single activation block via a dotted line, then another with a solid line below the first block, should be exported from this program and successfully imported into Lucidchart.

R4_*:

R4_1: A single activation block and two objects of a class, with the block connected via a solid line to the first object, will have a message arrow passing from the activation block to a solid line below the second object. This should be exported from this program and successfully imported into Lucidchart.

R4_2: Two activation blocks and two objects of a class, where each object is connected to a single activation block via a solid line, will have a message arrow passing from the first activation block to the second. This should be exported from this program and successfully imported into Lucidchart.

R4_3: Two activation blocks and two objects of a class, where each object is connected to a single activation block via a solid line, will have a message arrow passing from the first activation block to the second. This message arrow should have the method name above it. This should be exported from this program and successfully imported into Lucidchart.

R4_4: Two activation blocks and two objects of a class, where the first object is connected to a single activation block via a solid line and the second block to object via a dotted line, will have a message arrow passing from the first activation block to the second. There will be a second message arrow passing from the second activation block to the first as well. Both message arrows should have the method name above them. This should be exported from this program and successfully imported into Lucidchart.

R5_*:

R5_1: An Actor, Two activation blocks, and one object of a class, where the object is connected to a single activation block via a solid line and the second block to object via a dotted line. The actor will have a message arrow passing from its lifeline to the first activation block. There will be a second message arrow passing from the actor's lifeline to the second activation block as well. Both message arrows should have the method name above them. This should be exported from this program and successfully imported into Lucidchart.

R5_2: An Actor, Two activation blocks, and two objects of a class, where the first object is connected to a single activation block via a solid line, and the second block to the second object via a dotted line. The actor will have a message arrow passing from its lifeline to the first activation block. There will be a second message arrow passing from this activation block to the second block. Both message arrows should have the method name above them. The second activation block and method should be contained with a constraint with some internal text. This should be exported from this program and successfully imported into Lucidchart.

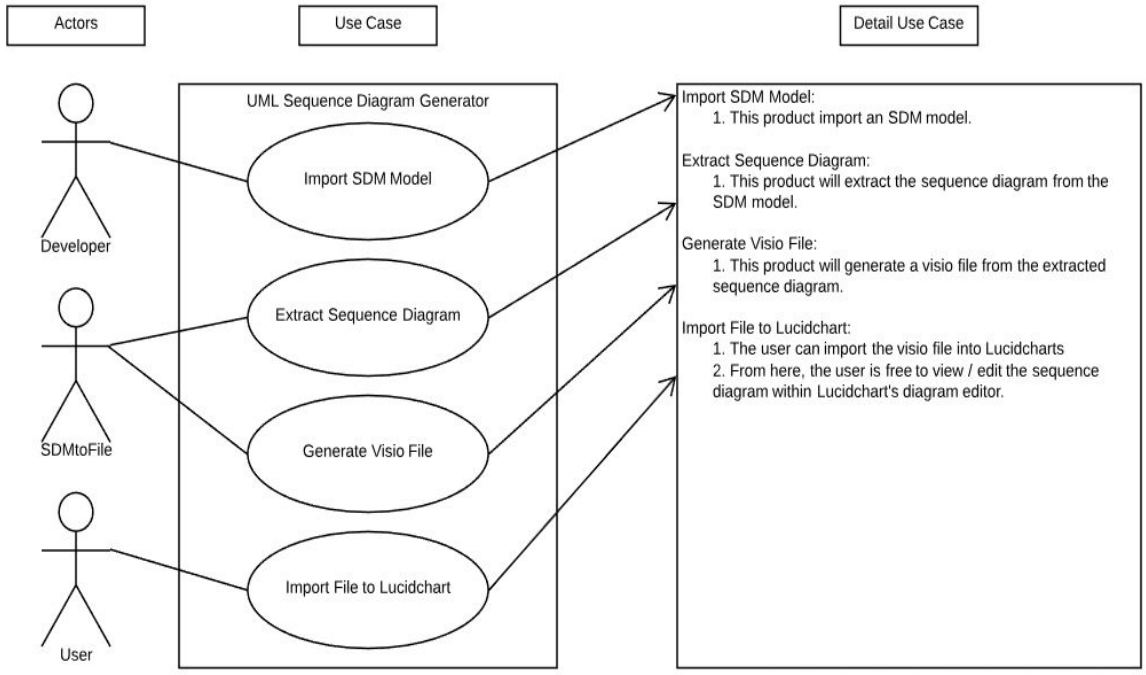
R5_3: An Actor, Two activation blocks, and two objects of a class, where the first object is connected to a single activation block via a solid line, and the second block to the second object via a dotted line. The actor will have a message arrow passing from its lifeline to the first activation block. There will be a second message arrow passing from this activation block to the second block. Both message arrows should have the method name above them. The second activation block and method should be contained within a loop block with some internal constraint text. This should be exported from this program

and successfully imported into Lucidchart.

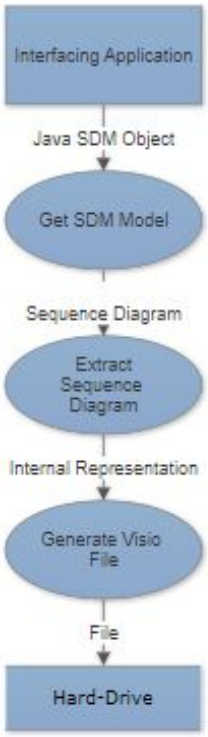
R5_4: An Actor, Two activation blocks, and two objects of a class, where the first object is connected to a single activation block via a solid line, and the second block to the second object via a dotted line. The actor will have a message arrow passing from its lifeline to the first activation block. There will be a second message arrow passing from this activation block to the second block. Both message arrows should have the method name above them. The second activation block and method should be contained within an alternative block with some internal text for both the constraint and the else. This should be exported from this program and successfully imported into Lucidchart.

Appendixes

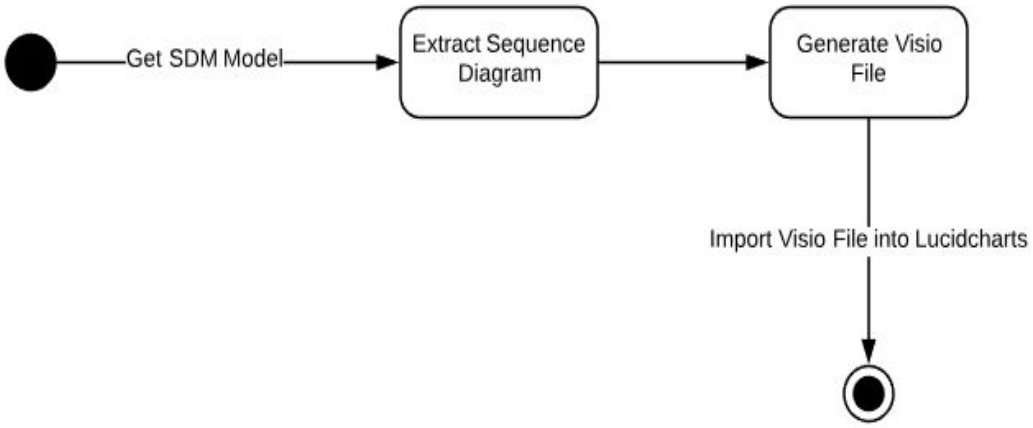
1. Use Case Diagram



2. Data Flow Diagram



3. State Transition Diagram



4. Requirement Specification

Requirement Specification

Project Name: UML Sequence Diagram Generator
Requirement Source: BeulahWorks, LLC.
Contact: Victor Guo admin@beulahworks.com

(Non)Functional Requirement:
A UML Sequence Diagram Model(SDM) is a reusable java component / library developed by BeulahWorks that includes multiple Java classes which model the UML sequence diagram. It can model information such as class/object names, message (invoked method's signature), and message sequence, etc. The model provides APIs (public classes and methods) allowing users to create new sequence diagram models (as Java objects) and read / update / delete information from existing models.

Lucidchart is a web application for diagram drawing. It allows user to draw UML diagrams including the sequence diagram. It also allows user to import diagrams in various formats such as the Microsoft Visio format. The imported diagrams will be visualized and enabled for user editing.

This project shall take the SDM models (java objects) as input, extract the sequence diagram information from the models, and generate Visio file(s). Users shall be able to import the files(s) to Lucidchart, and see / edit the same sequence diagram(s) within Lucidchart's diagram editor. The text styles (e.g. font sizes) and shape styles (e.g. box width and height, relative positions) does not matter. The diagram does not have to be well-aligned and directly human-readable. However all information contained in the sequence diagram must be fully included. E.g. if the SDM model contains 10 object names, then all 10 object names must be seen in the Lucidchart diagram editor once the generated file is imported.

The software design shall consider the possibility that

a) the SDM model may change its format in the future.
b) Lucidchart may add new import file formats in the future.

The design shall ease potential refactoring efforts to support those future changes.

Technology Requirement:

- Java shall be used as the programming language.
- The component shall be packaged as a Java library in a .jar file, so it can be integrated / reused in other products.
- Comprehensive Unit Tests shall be designed and developed on all significant methods (any method that is not simple getter, setter or constructor). JUnit shall be used as the unit test framework.
- System tests shall be designed and developed on the whole component.

- Javadoc comments shall be provided to thoroughly explain the source code.
- Third-party libraries such as Aspose.Diagram can be used to complete the work. Aspose.Diagram is the suggested library. Other alternative libraries must be approved by the requirement source to be used in this project.
- Basic performance tests shall be designed and developed.

Deliverables:

The final deliverables shall include:

1. Java Source Code (including unit test source code)
2. System Test Code
3. Performance Test Code
4. Generated Javadoc
5. Packaged Jar File
6. Engineering Documents