UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA UNIVERSIDAD DE MÁLAGA

Grado en Ingeniería del Software

Aplicación de Diseño de Diagramas de Estados para EDAG Production Solutions

State Diagram Design Application for EDAG Production Solutions

Realizado por

Manuel Bermudo Maeso

Tutorizado por

Eduardo Guzmán de los Riscos

Konstantin Ruppel

Departamento
Departamento de Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, FEBRERO 2020

Confidentiality clause

Málaga, 31st January, 2020

[illegible]

Manuel Bermudo Maeso
-Author-

i.V. Chr. [illegible]

Christopher Reuß
-Head of Department-

Abstract (Spanish)

En este trabajo de fin de grado se presenta el desarrollo realizado de una aplicación de escritorio para la empresa *EDAG Production Solutions GmbH & Co. KG*, dedicada a implementar soluciones de producción para sus clientes.

Dentro de la empresa, los desarrolladores de software del departamento de producción IT emplean distintas herramientas software al trabajar en las diferentes fases que un proyecto requiere, y, como le ocurre a otras empresas, para ciertos casos el mercado no ofrece soluciones accesibles cuando se trata de resolver problemas concretos. En concreto, *EDAG Production Solutions GmbH & Co. KG* necesita para el departamento de producción IT una aplicación de escritorio para diseñar diagramas que representan el comportamiento de un sistema y que, a partir de este diseño, genere automáticamente una plantilla del diagrama escrito en Java, ya que esta tarea se realiza manualmente y el tiempo de desarrollo de un proyecto se reducirá notablemente con esta herramienta. Actualmente, la empresa no ve adecuadas las soluciones existentes ya que o no son software libre o no permiten generar código a partir de un diagrama de la forma que necesitan. Además, estas herramientas no son intuitivas y demasiado complejas para lo que realmente necesita la empresa, por ello buscan una solución más adecuada a sus necesidades.

La aplicación desarrollada como trabajo final de grado permite esta funcionalidad que *EDAG Production Solutions GmbH & Co. KG* necesita. Además de la importación y exportación de diagramas en formato GraphML. Esta aplicación se ha desarrollado utilizando Angular, un framework para desarrollar aplicaciones web, sobre Electron, un framework de código abierto que permite ejecutar componentes para aplicaciones web en una aplicación de escritorio compatible con los sistemas operativos para ordenador más utilizados.

Palabras clave: Angular, aplicación, Electron, diagrama, mxGraph, GraphML.

Abstract (English)

This bachelor's thesis describes the development of a desktop application for the company *EDAG Production Solutions GmbH & Co. KG*, which aims to implement production solutions for its customers.

Within the company, the software developers of the productions IT department use different software tools to work on the many phases that a project requires, and as with other companies, in some cases, the market does not offer accessible solutions when it comes to solving specific problems.

More specifically, what *EDAG Production Solutions GmbH & Co. KG* needs for the IT production department a desktop application to design diagrams that represent the behavior of a system and, from this design, automatically generate a template of the diagram written in Java code, because this task is currently done manually and the development time of a project will be significantly reduced with this tool. At the moment, the company does not find suitable the existing solutions since they are either not free software or they can not generate code from a diagram in the way they need. Besides, these tools are not intuitive and much too complex for what the company needs, so they look for a solution more appropriate to their needs.

The application developed for the bachelor's thesis provides this functionality that the company needs. Additionally, it imports and exports diagrams in GraphML format. This application has been built using Angular, a framework for developing web applications, on Electron, an open-source framework that allows running components for web applications in a desktop application compatible with the most common desktop operating systems.

Keywords: Angular, application, Electron, diagram, mxGraph, GraphML.

Contents

1 Introduction

1.1 Motivation

The company *EDAG P. S.* [1], but in particular the IT production department, is responsible for developing digital manufacturing solutions for its customers, within the tasks they carry out when developing a project, there are generic activities that are normally put into practice. These are, according to Pressman [2]:

- **Communication:** in this stage, it is studied what the customer wants.
- **Planning:** here the tasks to be developed are described.
- **Modeling:** in this phase models are created and designed to understand and carry out product requirements.
- **Construction:** in this step programs are generated and tested to verify that they meet the requirements.
- **Implementation:** here the product is delivered to the customer.

In the construction phase, the models designed in the modeling phase are translated into code, including class diagrams, object diagrams, activity diagrams, sequence diagrams, and state diagrams, among many others, although this depends on the type of project to develop.

In *EDAG P. S.* they usually work in the modeling phase with a simplified concept of state machine diagrams [3], explained in section 2.1. To write these kind of models into code, *EDAG P. S.* requires a tool that makes it possible for its software developers to make state machine diagrams and from these designs, automatically generate a template written in Java [4], since this task is done manually and the development time of any project will be significantly reduced. Currently *EDAG P. S.* does not see adequate actual solutions in the market, primarily because these facilities have a cost that can not be absorbed. Moreover, the seemingly viable solutions for this are either too complex or do not generate an acceptable code template, therefore they prefer to develop their own solution.

1.2 Objectives

This bachelor thesis consists in developing a desktop application to design a simplified concept of state machine diagrams for the Windows operating system that allows drawing state machine diagrams, and from these graphs, generates a Java file containing a template that represents these diagrams. Also, any drawn state machine diagram can be imported and exported in GraphML format [5]. CRUD operations can also be performed on a state machine diagram, meaning that a graph and its elements can be viewed, created, edited and deleted using the tool.

1.3 Document structure

This document is divided into four chapters, each of them contains sections and subsections.

1. **Introduction**. This part gives a brief overview of the motivation and the aims of the project, the structure of this document and the working procedure followed to develop the application.
2. **Background**. In this chapter the technologies and tools utilized to implement this thesis and necessary concepts to understand some applied terms have been summarized.
3. **Requirements**. This chapter establishes the functional requirements, the non-functional requirements, and the use cases that the application must be able to perform.
4. **Design**. The formalization in the diagrammatic format of the project structure, the architecture of the application concerning the technologies used and the data model are defined in this chapter.
5. **Development and verification**. Here is explained how the application has been written and tested.
6. **Summary**. In this final chapter conclusions and further guidance of this project are drawn. Here it is also included any problems encountered during the development.

1.4 Working procedure

Every software project must be carried out by applying a certain methodology if stability and positive results are desired, but depending on the type of project and team, a more appropriate technique must be chosen. In this case, as it is an individual project without the possibility of assigning tasks to other members of a team, the more suitable option would be to apply the waterfall model [6]. This model consists of four main working phases, although depending on the type of project and how the tasks in each phase are defined, they may vary:

- Requirements collection and analysis.
- Design.
- Development.
- Validation and testing.

These phases do not need to be established as the model said, but it is necessary to categorize all the tasks in their respective phases and that the execution order is respected.

There were no specific roles for this project, but the responsibilities of each person involved were clear and the following roles could be established based on those responsibilities:

- **Customer**. This role has been represented by *EDAG P. S.*. Their main task was to specify what requirements the final product must meet. Those requirements have been communicated to what could be called the "project manager".
- **Project Manager**. This role has been performed by the author of this thesis, who has been in charge of managing the tasks to be performed by the developer.
- **Developer**. Role performed also by the author of the project, in charge of implementing each task needed to fulfill the customer's requirements.

For this project, these are the stages carried out:

1. **Requirements collection**. Here the application's requirements to be implemented are defined.
2. **Requirements analysis**. Once the previous phase is concluded, the requirements are formalized.

3. **Technologies analysis**. When the preceding phase is finished, it is studied and decided which technologies are suitable for the requirements analyzed.
4. **System design**. In this phase, the application structure and its components are defined. The following phase will start once these models are finished.
5. **Learning technologies**. After analyzing the possible technologies, several guides are made to understand how to implement the systems designed with the chosen technologies.
6. **Development**. In this phase, the source code of the application is written. Within this phase the following sub-tasks have been established:
   - Environment configuration.
   - Implementation of the project architecture for the technologies chosen in phase three of the working procedure.
   - Implementation of the defined entities in phase four of the working procedure.
   - Implementation of simple UI-elements (windows, headers, panels...).
   - Implementation of the functional requirements established in phase two of the working procedure.
7. **Validation and tests**. In this phase, the software implemented is tested using sample inputs and checking the outputs generated to further corrections.
8. **Project documentation**. Finally, in this last phase, the whole project is reported in this document.

2 Background

Here are explained briefly the core elements this project is dependent on. Sections 2.1 and 2.2 are the main concepts that must be clearly explained in order to abstract what is to be represented and how its elements can be linked to generate the template described in section 2.3.

2.1 State diagram

When it comes to defining the behavior of a system, the process to establish an abstract representation of it can be performed through various techniques. In the software development industry, this phase of abstraction and specification is normally conducted using the UML (Unified Modeling Language) notation [7] to design state diagrams.
A state diagram is a type of chart used to describe the behavior of a complex, composed of a finite number of states interconnected by transitions. This definition was set by Claude E. Shannon and Warren Weaver in their book *A Mathematical Theory of Communication* in 1949 [8].

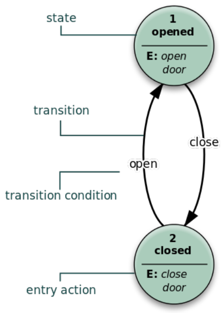


Figure 2.1: Example of a state machine.

The main and most common elements that a state diagram usually contains are:

- **State:** this element represents a defined situation within a system. Usually, the systems represented by these diagrams contain several states that define different situations.

- **Transition:** this component is a directed relationship between a source state and a target state. It models the event that changes the current state of a system and the following one.

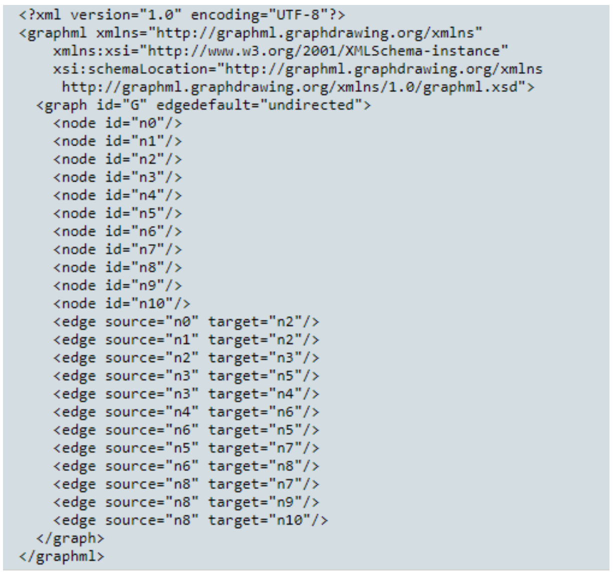
The UML notation defines other components in its documentation that originate from these two previous ones. However, they are not necessary to understand this concept.

After explaining the concept of the state machine, it is presented what kind of state diagrams *EDAG P. S.* wants to design using this application:

- State: A state represents a particular situation in a system. Its attributes should only identify it, and these attributes will be an **identifier** and a **name**.

- Transition: A transition indicates under what circumstances a system changes its state. When drawing one using the application, it should be clear what states are connected through a transition (source and target state). No guards are necessary when representing transitions, but specific input parameters to each transition. These parameters must be identified with a name and a type, as this information will be needed when generating a Java class.

2.2 Graph Markup Language (GraphML)



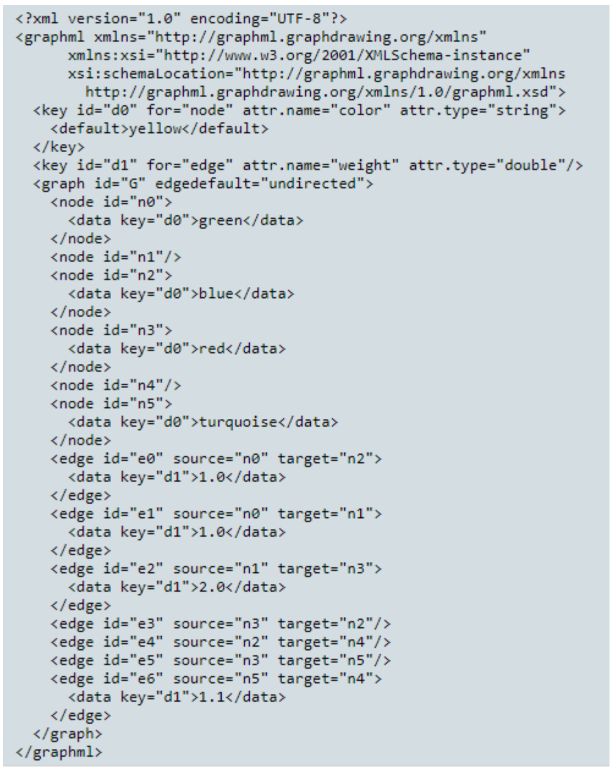
```xml
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
     http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <graph id="G" edgedefault="undirected">
    <node id="n0"/>
    <node id="n1"/>
    <node id="n2"/>
    <node id="n3"/>
    <node id="n4"/>
    <node id="n5"/>
    <node id="n6"/>
    <node id="n7"/>
    <node id="n8"/>
    <node id="n9"/>
    <node id="n10"/>
    <edge source="n0" target="n2"/>
    <edge source="n1" target="n2"/>
    <edge source="n2" target="n3"/>
    <edge source="n3" target="n5"/>
    <edge source="n3" target="n4"/>
    <edge source="n4" target="n6"/>
    <edge source="n6" target="n5"/>
    <edge source="n5" target="n7"/>
    <edge source="n6" target="n8"/>
    <edge source="n8" target="n7"/>
    <edge source="n8" target="n9"/>
    <edge source="n8" target="n10"/>
  </graph>
</graphml>
```

Figure 2.2: Representation of a graph and its attributes.

This application works only with a kind of file format called GraphML based on XML [9]. It is specifically designed to describe the structure of a graph and its properties. As explained in its official site [10], the main features of the format include support of:

- directed, undirected and mixed graphs.
- hypergraphs.
- hierarchical graphs.
- graphical representations.
- references to external data.
- application-specific attribute data.
- light-weight parsers.

As shown in figure 2.2, a GraphML file is composed of a `<graph>` tag that contains `<edge>`and `<edge>` elements, not necessary ordered. Each `<node>` and `<edge>` tag must contain a unique `id` attribute, also the `<edge>` tags must have a `source` and a `target` attribute containing existing `id` values of the `<edge>` at the endpoints of the `<edge>`.



```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
        http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <key id="d0" for="node" attr.name="color" attr.type="string">
    <default>yellow</default>
  </key>
  <key id="d1" for="edge" attr.name="weight" attr.type="double"/>
  <graph id="G" edgedefault="undirected">
    <node id="n0">
      <data key="d0">green</data>
    </node>
    <node id="n1"/>
    <node id="n2">
      <data key="d0">blue</data>
    </node>
    <node id="n3">
      <data key="d0">red</data>
    </node>
    <node id="n4"/>
    <node id="n5">
      <data key="d0">turquoise</data>
    </node>
    <edge id="e0" source="n0" target="n2">
      <data key="d1">1.0</data>
    </edge>
    <edge id="e1" source="n0" target="n1">
      <data key="d1">1.0</data>
    </edge>
    <edge id="e2" source="n1" target="n3">
      <data key="d1">2.0</data>
    </edge>
    <edge id="e3" source="n3" target="n2"/>
    <edge id="e4" source="n2" target="n4"/>
    <edge id="e5" source="n3" target="n5"/>
    <edge id="e6" source="n5" target="n4">
      <data key="d1">1.1</data>
    </edge>
  </graph>
</graphml>
```

Figure 2.3: Representation of a graph with `<key>` and `<data>` components.

Some other optional elements of GraphML needed to make this project possible are the `<data>` and `<key>` tags. As specified in the official documentation, it is possible to find `<data>` objects within other elements such as `<edge>`, `<edge>`, `<graph>`, etc. These objects are declared in `<key>` elements after the `<graphml>` tag and defined by `<data>` items.

2.3 Java Class

The main objective of the application is that it can convert a state diagram into a Java class. The template that *EDAG P. S.* wants to be generated is the following, which is also explained below.

```java
class StateMachine {

  private State currentState;

  private State getState() {
    return currentState;
  }

  private void setCurrentState(State s) {
    currentState.onExit();
    currentState = s;
    currentState.onEntry();
  }

  public void changeState() {
    // TODO Implement auto-generated
    currentState.changeState();
  }


  private abstract class State {

    void onEntry() {
      // TODO Implement auto-generatedö
    }

    void onExit() {
      // TODO Implement auto-generated
    }

    void changeState() {
      // TODO Implement auto-generated
    }

35   }
36 
37   private class State1 extends State {
38 
39     @Override
40     void onEntry() {
41       // TODO Implement auto-generated
42     }
43 
44     @Override
45     void onExit() {
46       // TODO Implement auto-generated
47     }
48 
49     @Override
50     void changeState() {
51       // TODO Implement auto-generated
52       setCurrentState(new State2());
53     }
54 
55   }
56 
57   private class State2 extends State {
58 
59     @Override
60     void onEntry() {
61       // TODO Implement auto-generated
62     }
63 
64     @Override
65     void onExit() {
66       // TODO Implement auto-generated
67     }
68 
69   }
70 
71 }
```

Listing 2.1: Template that *EDAG P. S.* wants to be generated

This class is the one to be generated for a state diagram with a source state ”State1” and a target state ”State2”, connected by a transition called ”changeState” without input parameters. For each transition in a diagram, a method with the same name as the transition must be generated within the main class (lines 15 to 18). If there are states present in the diagram, an abstract class ”State” is generated from which the other states in the diagram are inherited (lines 21 to 35). This class will contain all the transitions of the diagram.
After the abstract class ”State”, the states of the diagram are generated. If a state is the source state of a transition, it will contain a method with the name of that transition, as

shown in the code block 2.1 from line 50 to 53.

2.4 Study of technologies

This section briefly discusses which libraries and tools are used in this project. The fact of choosing the framework of subsection 2.4.2 has required a little research to know the different options offered by the market. If the reader is interested in knowing the reasons that have led to the use of this technology, he can consult the appendix 6.

2.4.1 Angular

Since Angular is one of the technologies that *EDAG P. S.* uses in many of its projects, it was decided to implement this application using this framework.
Angular [11] is a development platform for building mobile and desktop web applications. It is based on TypeScript and is developed and maintained by Google and a community of developers that contribute to it. This framework is a complete rewrite of AngularJS [12].

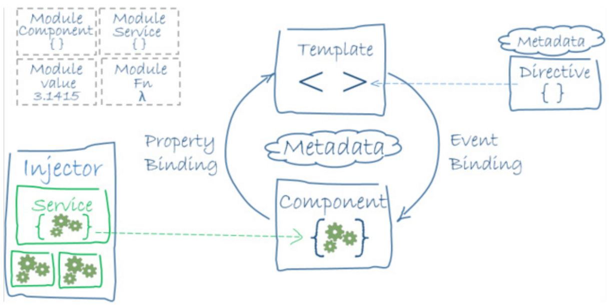
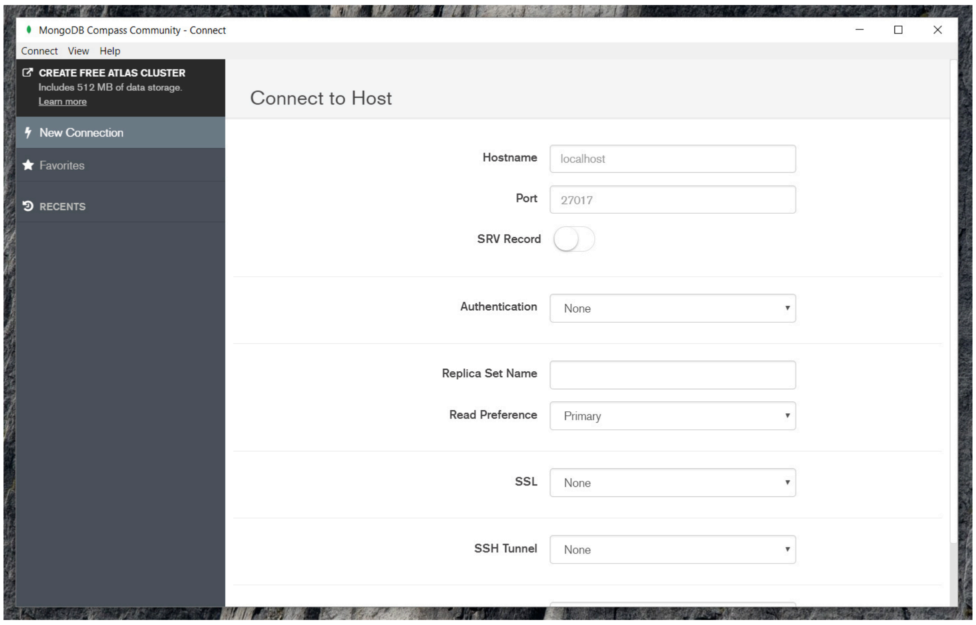


Figure 2.4: Architecture of an Angular application [13]

Some advantages of using this framework are its declarative UI when writing HTML templates, its simplified MVC (Model-View-Controller), its modular structure that allows the developer to reuse angular components, to maintain and escalate web applications easily, and its simplified unit-testing independent.

2.4.2 Electron


MongoDB Compass Community - Connect
Connect View Help
CREATE FREE ATLAS CLUSTER
Includes 512 MB of data storage.
Learn more
New Connection
Favorites
RECENTS
Connect to Host

Hostname: localhost
Port: 27017
SRV Record
Authentication: None
Replica Set Name:
Read Preference: Primary
SSL: None
SSH Tunnel: None

Figure 2.5: MongoDB Compass desktop application, developed with Electron

Electron [14] is a library developed by GitHub for building desktop applications for different supported operating systems with web application technologies, such as HTML, CSS, and JavaScript. Electron does this combining Chromium [15] on the client-side and Node.js [16] on the server-side into a single run-time and the applications can be built for Windows, Linux, and Mac-OS. Some examples of applications built using this framework are Visual Studio Code [17] (the code editor used for this project), Discord [18], MongoDB Compass [19] or Slack [20].

2.4.3 mxGraph

To implement the application and its functions without developing methods and features to draw and create diagrams, it is necessary to use a library that already implements these functionalities. For this purpose, mxGraph has been chosen.

The framework mxGraph [21] is a JavaScript diagramming library that allows the user to create interactive graphs. The framework is divided into 8 packages:

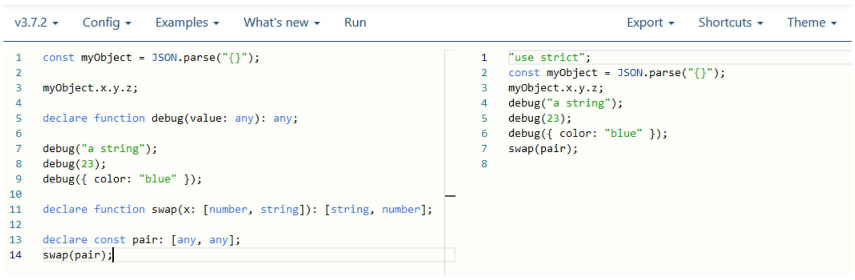
- **Editor**. It provides classes and methods to implement a diagram editor.

- **View and Model**. These packages implement the graph component and its elements.
- **Handler, Layout, and Shape**. They contain event listeners, layout algorithms and shapes structures.
- **Util**. This package contains utility classes for various features, such as the commands *copy* and *paste*, constants for stylesheets or functions to translate content.
- **IO**. This part of the framework implements functions to convert from JavaScript objects to XML and the other way around and functions to register custom converters.

The use of a library written in JavaScript in an Angular application may seem incompatible at first glance, but it can be adapted as explained in section 5.1.

2.4.4 TypeScript

Since version 2 of Angular, this framework is based on the TypeScript programming language. TypeScript [22] is an open-source programming language released in 2012 by Microsoft [23] and actively developed and maintained. It is a superset of JavaScript that adds new features like variable types, interfaces, classes, modules, "arrow" syntax for inline functions, generic objects...



```
const myObject = JSON.parse("{}");

myObject.x.y.z;

declare function debug(value: any): any;

debug("a string");
debug(23);
debug({ color: "blue" });

declare function swap(x: [number, string]): [string, number];

declare const pair: [any, any];
swap(pair);
```

```
"use strict";
const myObject = JSON.parse("{}");
myObject.x.y.z;
debug("a string");
debug(23);
debug({ color: "blue" });
swap(pair);
```

Figure 2.6: On the left, script written in TypeScript. On the right, same script on the left compiled to JavaScript

2.4.5 Visual Studio Code

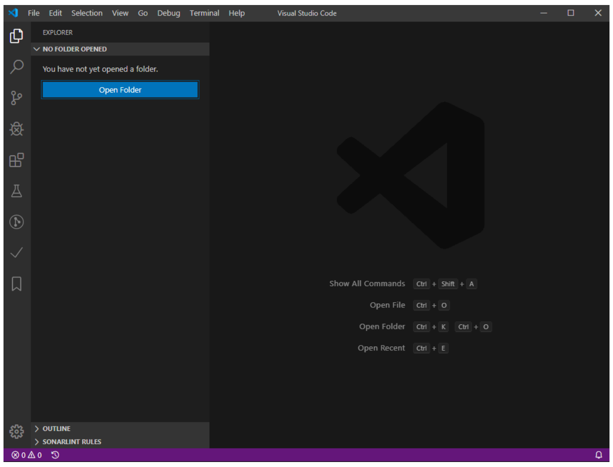

File Edit Selection View Go Debug Terminal Help
Visual Studio Code
EXPLORER
NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
Show All Commands Ctrl + Shift + A
Open File Ctrl + O
Open Folder Ctrl + K Ctrl + O
Open Recent Ctrl + E
OUTLINE
SONARLINT RULES

Figure 2.7: Desktop application of Visual Studio Code

To write code and install packages to develop this project, the Visual Studio Code [17] editor has been the tool of choice, an open-source product of Microsoft. The main reason for choosing this editor and not other similar ones is because it includes native support for TypeScript, the possibility to add extensions to develop Angular application and it is one of the most used code editors by the developer community, which ensures that in case of configuration failures or improve productivity when implementing the project, it is straightforward to find a solution.

2.4.6 Git

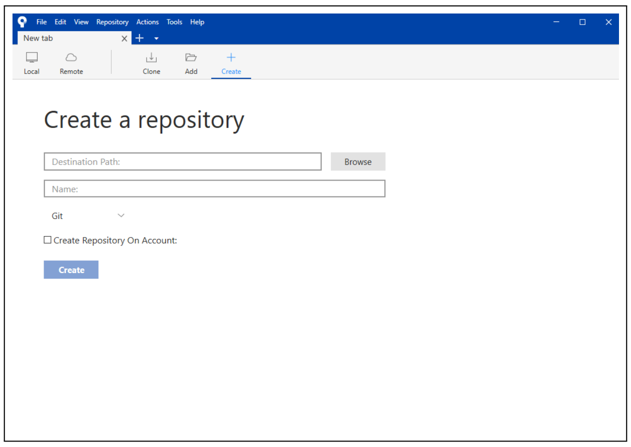


Figure 2.8: Sourcetree, a desktop client to use Git

All the code written for this project, including the LaTeX [24] files to write this bachelor thesis, has been uploaded regularly to a private repository of *EDAG P. S.* through Git. Git [25] is a version-control system for tracking changes in source code. Its main feature is the support for non-linear development, with options to reverse changes and to merge new content with old ones avoiding conflicts. Aside from Git's own provided software to update code and resolve conflicts, some vendors offer intuitive tools to use the full potential of Git, such as Atlassian's Sourcetree application, which has been used throughout the development process.

2.4.7 LaTeX

```
\documentclass[a4paper, 12pt, oneside]{scrbook}
\input{settings/settings.tex}
\bibliography{management/bibliography.bib}
\input{settings/bib_labels.tex}
\begin{document}
  \mainmatter %Main part numbering
  \insertblankpage
  \insertblankpage
  \input{management/variables.tex}
  \input{chapters/FrontMatter/combined.tex}
  \input{chapters/00_mainMatter.tex}
  \appendix
  \input{chapters/99_Anhang.tex}
  \printbibliography
\end{document}
```

Listing 2.2: Root file used to develop the current thesis

LaTeX is a high-quality typesetting system for developing reports, articles, books or similar document formats. This free software prevents the user from spending time editing the appearance of the document and can focus on the quality of the work. Before beginning a LaTeX project, the author would have to configure the format of the document once (or change it while writing) and some other properties of the paper.

3 Requirements

The requirements have been obtained through meetings between the customer and the project manager. It must be clarified that the client has never required any specification outside the functional requirements, which are defined hereafter, and the functionalities apart from these have been accepted by the client.

3.1 Overall Description

The system to be developed will be a Windows desktop application, and no Internet connection is necessary. It has to be a responsive application so that it will adapt to different display sizes. It will be possible to use it from an executable compatible with Windows 7 or superior.

3.1.1 Design and Implementation Constraints

- The application is limited to platforms supported by Electron.
- Electron provides a run-time to build desktop applications with JavaScript, that forces us to use only JavaScript or some framework based on JavaScript to develop the product, such as Vue.js, Angular or React.
- Electron applications need sufficient free disk space and it is required at least 200 MB of free memory. The customer does not impose a maximum size for the application.
- Because it will be an Electron application, the framework style guidelines must be followed for the application to be maintainable.

3.1.2 Assumptions and Dependencies

- Assumptions
    - It is assumed that the application will be executed only on desktop devices

running Windows 7 or higher.

  – Another assumption is that the user is familiar with handling the keyboard, mouse and the operating system where the application is running.
  – It is presumed that the user is familiar with the concept of a state diagram and its elements.

- Dependencies
  – It is intended to use an open-source framework with which it is possible to develop desktop applications for Windows, therefore the programming environment must be compatible with that framework.
  – When the user opens a GraphML file, the diagram generation will depend entirely on how this file is written. If there are some inconsistencies in the file, these will be displayed to the user.
  – In case that it is necessary to use a library to convert a GraphML file to any other element, for example, an XML file or a Java object, the software application will be dependent on that.

3.2 Functional Requirements

ID	SF1
Title	Open diagram
Description	The user may open a file in GraphML format to the application by clicking the option “Open diagram” located on the application’s toolbar. After this event, the content of the file will be displayed in the drawing area of the application.
Priority	HIGH

ID	SF2
Title	Save diagram
Description	The user may generate a file in GraphML format from a designed diagram by clicking a button called “Save diagram” located on the application’s toolbar. After this action, a pop-up window will open, then the user may choose wherein his computer to save the file.
Priority	HIGH

ID	SF3
Title	Generate Java class
Description	The user may generate a file in Java format from a designed diagram by clicking the button called “Generate Java class” located on the application’s toolbar. After this action, a pop-up window will open, then the user may choose wherein his computer to save the file containing the generated Java class.
Priority	HIGH

ID	SF4
Title	View diagram
Description	The user will be able to see the contents of a diagram in the application’s drawing panel.
Priority	MEDIUM

ID	SF5
Title	Create diagram
Description	The user will be able to create an empty diagram.
Priority	MEDIUM

ID	SF6
Title	Edit diagram
Description	The user will be able to edit an existing diagram. Specifically, the user will be able to drag elements from the items panel to the drawing area, to move and resize any item inside the drawing area, to create transitions between nodes, and to edit information of the items in the drawing area.
Priority	MEDIUM

ID	SF7
Title	Delete elements
Description	The user will be able to remove elements from an existing diagram.
Priority	MEDIUM

ID	SF8
Title	Perform CRUD operations on parameters.
Description	The user may perform CRUD (create, read, update and delete) operations on the parameters of a transition between nodes in the diagram.
Priority	HIGH

The functional requirements SF1, SF2, SF3, and SF8 are the only indispensable requirements for this application. The CRUD (create, read, update, and delete) type requirements formalize that a diagram can be drawn and edited as expected from a standard editor.

3.3 Non-functional Requirements

ID	NF1
Title	Speed of operations
Description	Each operation of the application must not take more than five seconds to produce a result.
Priority	MEDIUM

ID	NF2
Title	Personal information
Description	The application must not store any kind of personal information of the user or the computer.
Priority	MEDIUM

ID	NF3
Title	Reliability
Description	The application must be stable when being executed. This means that it must be fail-safe.
Priority	MEDIUM

ID	NF4
Title	Testability
Description	The system must be easy to test and find defects.
Priority	MEDIUM

ID	NF5
Title	Portability
Description	The application must run on every device that works with Windows 7 or higher.
Priority	MEDIUM

ID	NF6
Title	Usability
Description	The application must be user-friendly and easy to learn.
Priority	MEDIUM

ID	NF7
Title	Correctness
Description	The application must adhere to functional requirements.
Priority	MEDIUM

3.4 Actors

This application does not involve many types of actors except for the main user, which represents any user who uses the application. This actor can perform all the functionalities offered by the tool.

3.5 Use cases

ID	UC1
Title	Open existing diagram
Actor	Main user
Related	SF1
Description	The user may open a GraphML file with the application.
Preconditions	The GraphML file must be valid, without inconsistencies.
Main Scenario	1. The user clicks on the button “Open diagram”. 2. The program opens a new window with the device files. 3. The user selects a suitable file (“.graphml” ending). 4. The user clicks the button called “Open”. 5. The program represents the file’s content in the drawing area.
Postconditions	The content of the file will be displayed as a graph.
Alternative scenario	3. The user closes the opened window. 4. The application throws a pop-up message which says that no file has been selected.

ID	UC2
Title	Save existing diagram
Actor	Main user
Relates	SF2
Description	The user may save the diagram drawn in the drawing area into a GraphML file.
Preconditions	
Main Scenario	1. The user clicks on the button “Save diagram”. 2. The program opens a new window with the device files. 3. The user selects a device folder and writes in the corresponding field the name under which he wants the diagram to be saved. 4. The user clicks the button called “Save”. 5. The program informs with a pop-up message that the file was saved.
Postconditions	The diagram content is saved under a GraphML file.
Alternative scenario	3. The user clicks the button called “Cancel”. 4. The program informs with a pop-up message that the file was not saved.

ID	UC3
Title	Generate Java file
Actor	Main user
Related	SF3, SF4
Description	The user may save the diagram drawn in the drawing area into a Java file.
Preconditions	A diagram is drawn in the drawing area.
Main Scenario	1. The user clicks on the button "Generate Java file". 2. The program opens a new window with the device files. 3. The user selects a device folder and writes in the corresponding field the name under which he wants the diagram to be saved. 4. The user clicks the button called "Save". 5. The program informs with a pop-up message that the file was saved.
Postconditions	The diagram content is saved under a Java file.
Alternative scenario	3. The user clicks the button called "Cancel". 4. The program informs with a pop-up message that the file was not saved.

ID	UC4
Title	Drag and drop states
Actor	Main user
Related	SF4, SF6
Description	The user may drag elements from the items panel and drop them in the drawing area.
Preconditions	
Main Scenario	1. The user clicks and holds an element from the left panel. 2. The user moves the mouse until it points to the drawing while holding the click button of the mouse. 3. The user releases the click button of the mouse.
Postconditions	The drawing area displays a new node with the same shape as the item selected at the beginning.
Alternative scenario	3. The user releases the click button on the mouse when it is not pointing to the drawing area. 4. Nothing happens.

ID	UC5
Title	Link nodes
Actor	Main user
Related	SF4, SF6
Description	The user may link two different nodes together.
Preconditions	There are at least two nodes in the drawing area.
Main Scenario	1. The user points the mouse over a node in the drawing area. 2. The pointed node shows an arrow icon in its center. 3. The user clicks and holds the click button of the mouse on the arrow icon displayed. 4. the user moves the mouse while holding the click button until the mouse is over a different node. 5. The drawing area shows an arrow connecting both nodes.
Postconditions	The drawing area displays a link between the source node and the target node.
Alternative scenario	3. The user releases the click button on the mouse when it is not pointing to a node. 4. No arrow is created.

ID	UC6
Title	Delete elements
Actor	Main user
Related	SF4, SF6
Description	The user may delete any element in the drawing area.
Preconditions	There is at least one element in the drawing area.
Main Scenario	1. The user clicks on any element in the drawing area. 2. The clicked element changes its shape, representing that has been selected. 3. The user presses the key "Delete" of the computer keyboard.
Postconditions	The selected element is removed from the drawing area.
Alternative scenario	

ID	UC7
Title	Add parameter
Actor	Main user
Related	SF4, SF6, SF8
Description	The user may add parameters to an existing transition.
Preconditions	There is at least a transition in the drawing area.
Main Scenario	1. The user right-clicks on a transition. 2. A panel with the label “Parameters” is displayed close to the transition. 3. The user clicks the label “Parameters”. 4. A dialog is displayed with the name of the transition and a table with the actual parameters of the transition. 5. The user clicks on the button called “Add parameter”. 6. A new entry is added to the parameter list of the transition 7. The user changes the fields of the new entry in the table inside the dialog after double-clicking on them. 8. The user clicks the button called “Save changes”.
Postconditions	The parameter added is saved.
Alternative scenario	7. The user clicks outside of the dialog. 8. The changes are not saved.

ID	UC8
Title	View parameters
Actor	Main user
Related	SF4, SF6, SF8
Description	The user may view parameters of an existing transition.
Preconditions	There is at least a transition in the drawing area.
Main Scenario	1. The user right-clicks on a transition. 2. A panel with the label "Parameters" is displayed close to the transition. 3. The user clicks the label "Parameters".
Postconditions	The parameters of the transition selected are displayed in a table inside a dialog element. If the transition does not have any parameters, this information will be shown.
Alternative scenario	

ID	UC9
Title	Update parameters
Actor	Main user
Related	SF4, SF6, SF8
Description	The user may edit parameters of an existing transition.
Preconditions	There is at least a transition with parameters defined in the drawing area.
Main Scenario	1. The user right-clicks on a transition. 2. A panel with the label "Parameters" is displayed close to the transition. 3. The user clicks the label "Parameters". 4. A dialog is displayed with the name of the transition and a table with the actual parameters of the transition. 5. The user changes the fields of the new entry in the table inside the dialog after double-clicking on them. 6. The user clicks the button called "Save changes".
Postconditions	The parameters edited are saved.
Alternative scenario	6. The user clicks outside of the dialog. 7. The changes are not saved.

ID	UC10
Title	Delete parameters
Actor	Main user
Related	SF4, SF6, SF8
Description	The user may delete parameters of an existing transition.
Preconditions	There is at least a transition with parameters defined in the drawing area.
Main Scenario	1. The user right-clicks on a transition. 2. A panel with the label “Parameters” is displayed close to the transition. 3. The user clicks the label “Parameters”. 4. A dialog is displayed with the name of the transition and a table with the actual parameters of the transition. 5. The user clicks on the button “Delete” located on every row of the parameters table in the dialog.
Postconditions	The parameters are definitely removed.
Alternative scenario	

4 Design

The architecture of this application is single-page since it only requires an interface that does not change except for the diagrams that the user wants to create. There is no need to navigate between different views and only one controller is needed to process the different commands that the user wants to execute.

4.1 Electron architecture

This application uses the Electron framework to run and therefore depends exclusively on how Electron interacts with the system. This communication can be seen in the figure 4.1.

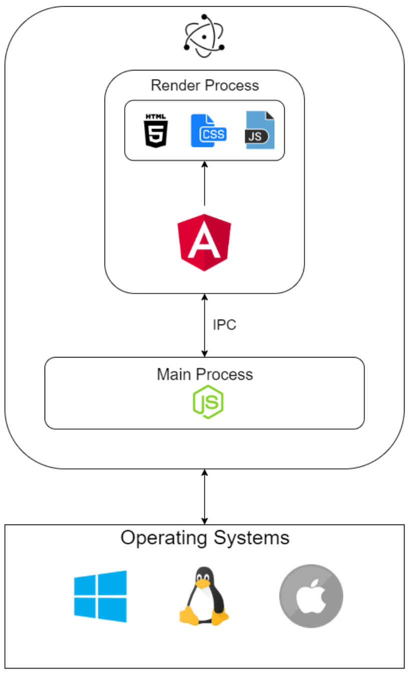


Figure 4.1: Architecture of an Electron application

Electron executes what is called the “main process”, which is responsible for interacting with the operating system where it is running. This process creates the application GUI by creating *BrowserWindow* instances which, at the same time, execute their own “renderer process". This renderer process takes the **index.html** file that, in this case, Angular provides as a starting point of the web application and renders it in the window. More about how Electron works can be found in its official documentation.

4.2 Model-View-ViewModel

This project has been designed as a single-page application applying the design pattern "Model-View-ViewModel" [26] since it only requires an interface that does not change except for the diagrams that the user wants to create. There is no need to navigate between different views and only one controller is needed to process the different commands that the user wants to execute.

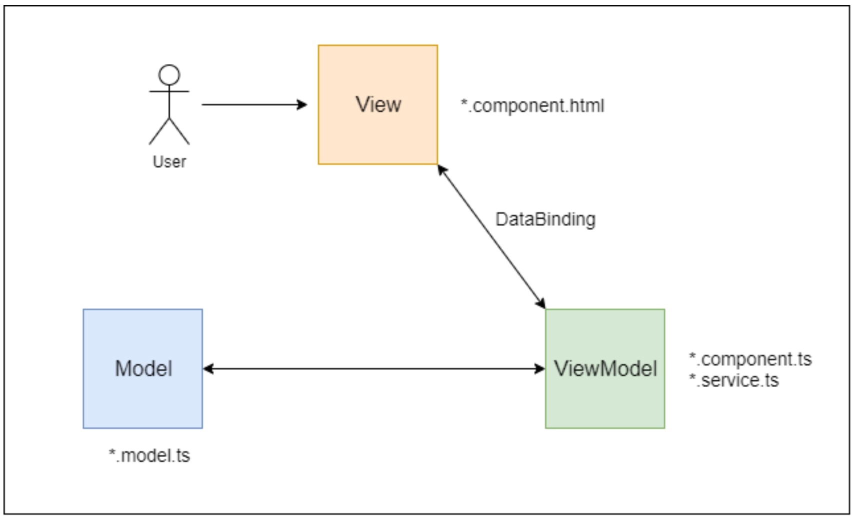
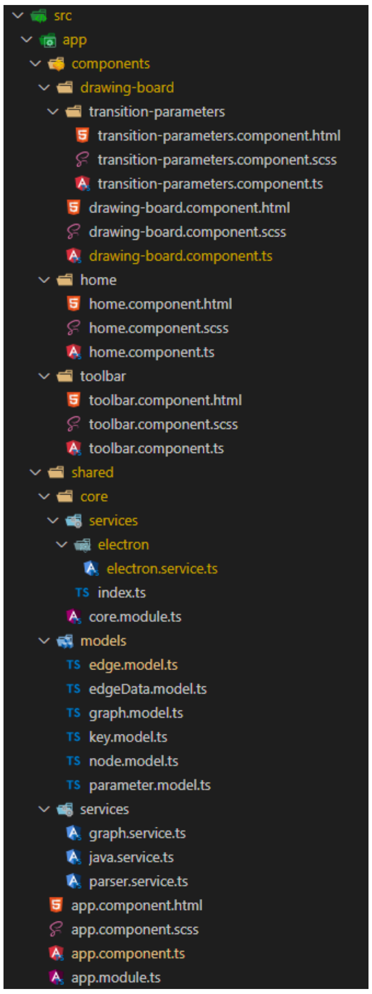


Figure 4.2: Pattern MVVM adapted to Angular

As Strawn [27] explains, this pattern separates objects into three groups:

- **Models** represent the application data (classes and interfaces in Angular).
- **Views** are the visual elements (HTML components).
- **View models** convert model information into usable data (TypeScript components and services).

This pattern has been applied as figure 4.3 shows, where the "component.html" files belong to *View*, "component.ts" and "service.ts" to *ViewModel* and "model.ts" to *Model*.



```
src
  app
    components
      drawing-board
        transition-parameters
          transition-parameters.component.html
          transition-parameters.component.scss
          transition-parameters.component.ts
        drawing-board.component.html
        drawing-board.component.scss
        drawing-board.component.ts
      home
        home.component.html
        home.component.scss
        home.component.ts
      toolbar
        toolbar.component.html
        toolbar.component.scss
        toolbar.component.ts
    shared
      core
        services
          electron
            electron.service.ts
          index.ts
        core.module.ts
      models
        edge.model.ts
        edgeData.model.ts
        graph.model.ts
        key.model.ts
        node.model.ts
        parameter.model.ts
      services
        graph.service.ts
        java.service.ts
        parser.service.ts
    app.component.html
    app.component.scss
    app.component.ts
    app.module.ts
```

Figure 4.3: Structure of the project

4.3 Data model

To convert diagrams drawn with the mxGraph framework and those drawn from a file in GraphML format in the simplest way, a model is needed that allows such operations without using overly complicated methods. For this reason, the model of figure 4.4 has been defined.

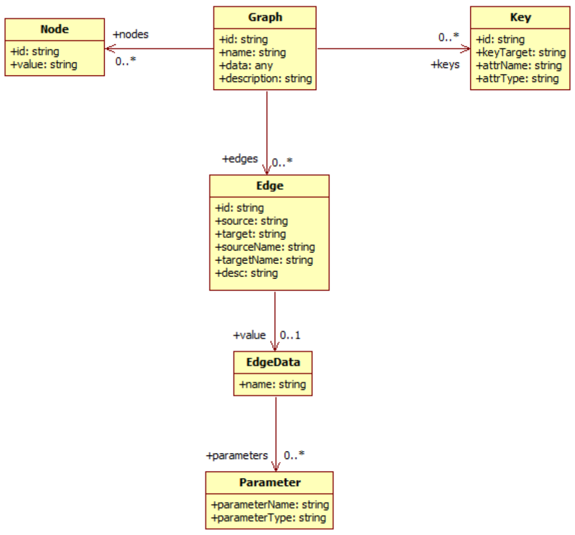


Figure 4.4: Graph class diagram

The Graph class diagram combines the basic elements of GraphML from section 2.2 and a "Parameter" interface to represent the parameters that transitions must contain.

5 Development and verification

This section explains what has been done to implement the functions to be performed by the application, starting from how the project was generated until what has been done to test the functionalities once completed.

5.1 Development

To create the structure of this project by combining two different technologies such as Electron and Angular, Electron's documentation has been consulted since what has been developed is a web application that runs on Electron. The documentation redirects to a GitHub repository where any user can find resources related to the Electron framework. From this repository, a template has been used that already provides an initial configuration for an Angular project.

An initial problem that, due to a lack of prior knowledge, had to be faced was integrating the mxGraph framework into Angular. The mxGraph library is written in JavaScript, but Angular uses TypeScript, which initially means that the mxGraph library cannot be used in an Angular application. To use any library in Angular, it must either be developed in TypeScript or written in JavaScript and have a file (commonly known as "type definition") that allows it to be used in Angular. By exploring possible solutions, a project called "mxgraph-typings" [28] hosted in GitHub with an MIT license [29] has been used that already defines these types.

After configuring the environment, the model of figure 4.4 has been written. The current attributes are not final, but they are sufficient to be able to perform the conversion functions between the GraphML format and the objects of the mxGraph library.

To implement the structure MVVM shown in image 4.2, a distinction has been made between components, which represent the "View-ViewModel" communication, and services, which, within the "ViewModel" entity in figure 4.2, are responsible for processing the component requests. The following files have been written:

- **Components**. In Angular, in the same component the relation ”View-ViewModel” is established where the HTML file (*.component.html) would be the View and the TypeScript file (*component.ts) would be the ViewModel. Four components have been established:
  - **toolbar**. The toolbar component contains the buttons from which the user can execute the different functions of the application.

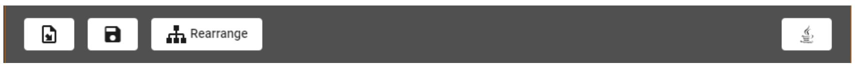


Figure 5.1: From left to right: open diagram button, save diagram button, rearrange layout button and generate Java file button

  - **drawing-board**. The drawing board component is the main component of the application. It consists of two panels, one on the left and one on the right. The left panel contains the elements that can be dragged to the right panel, each with a different shape when dropped on the right panel. The right panel contains a kind of whiteboard where the elements inside can be moved, deleted and edited. This component communicates with the services to open diagrams, save them and generate a Java.

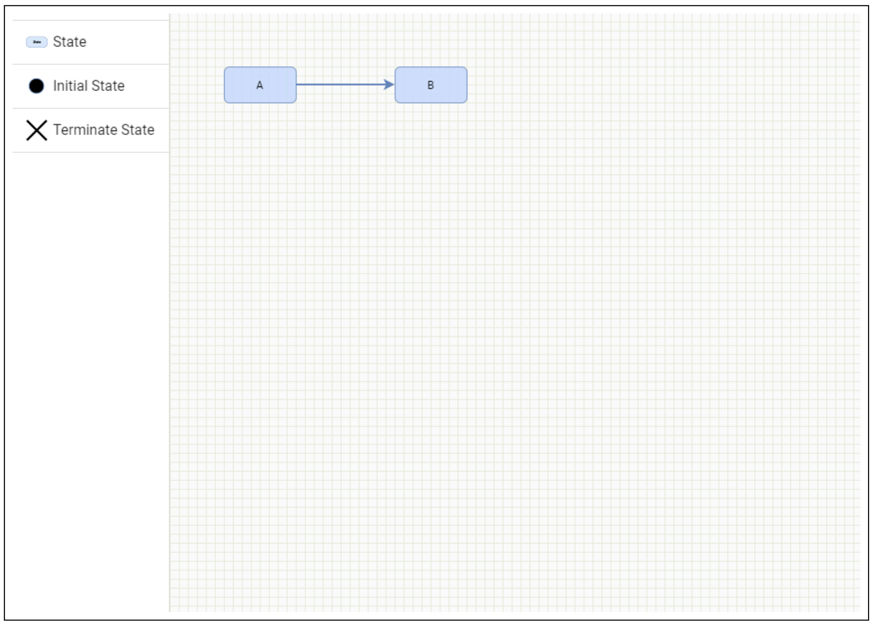


Figure 5.2: drawing-board component view

- **transition-parameters**. The transition-parameters component is a dialog that opens inside the application when right-clicking on a transition drawn in the right panel of the drawing-board component.

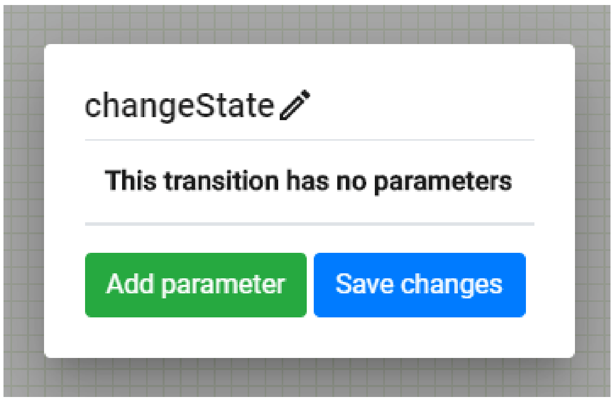


Figure 5.3: transition-parameters component view

- **Services**. Three services have been defined:
  - **graph.service.ts**. This service is responsible for saving and opening diagrams when the drawing-board component invokes them.
  - **parser.service.ts**. This service is in charge of converting objects for the other components and services. Within it, methods are implemented to convert from a mxGraph instance to GraphML and from GraphML to an instance of mxGraph.
  - **java.service.ts**. This service is used to generate a java class from a mxGraph instance. Together with the parser service, they are the ones that have had the most workload, as there are no technologies that can convert between mxGraph, GraphML and the Java class that *EDAG P. S.* requires.

Electron provides a package called “electron-builder” to build a compatible executable for the operating systems it supports. The script “**npm run electron:windows**” defined in the **package.json** file of the project generates an executable for Windows.

5.2 Verification

To verify that the application meets the functional requirements, the resulting outputs have been checked after executing the use cases defined in section 3.5, although each time a function was completed during the development phase, it was tested with different input values and errors were fixed on site. Unfortunately, if the use cases involving the interface had been tested, it would have been necessary to investigate how this could have been done with mxGraph, but this would have exceeded the workload for a bachelor thesis.

After the correction of the bugs found during the development of the methods, the results generated by the different input values tested have been satisfactory.

6 Summary

Conclusiones y líneas futuras

Objetivos alcanzados

Los requisitos funcionales definidos en la sección 3.2 se cumplen satisfactoriamente. La solución generada es apta para que *EDAG P. S.* pueda utilizarla a la hora de desarrollar proyectos. Excluyendo los resultados positivos que se han logrado, se han obtenido otras conclusiones:

- El mercado no ofrece una variedad de herramientas para el tipo de aplicación y tecnología utilizados. Esto tiene ventajas, como la realización de un producto único y potente y el aprendizaje de tecnologías muy útiles a medio y largo plazo en la industria del desarrollo de software, e inconvenientes, como la falta de recursos a los que recurrir y la inmadurez de las tecnologías utilizadas. Electron es una herramienta viable a nivel empresarial, pero las versiones estables publicadas durante el desarrollo de esta aplicación causaron errores que prolongaron esta fase respecto a la planificación establecida.
- El framework mxGraph es muy potente y visual, pero a su vez complejo y requirió más tiempo del deseado para poder entender las funciones necesarias para la aplicación. Una funcionalidad que se consideró una opción viable a la hora de guardar diagramas no fue posible finalmente. La clase mxCodecRegistry proporcionada por la librería mxGraph contiene un bug al registrar codecs personalizados, impidiendo su uso cuando se requería y se invirtió mucho tiempo sin lograr un resultado positivo en este aspecto.
- Si una librería escrita en JavaScript no contiene tipos definidos para poder ser usada en Angular, considero que no es productivo utilizarla y sería conveniente buscar una alternativa viable. Fue cuestión de suerte encontrar una opción de código abierto que permitiera poder usar mxGraph en Angular, de lo contrario se hubiera empleado mucho más tiempo del previsto.

Líneas futuras

La aplicación tiene margen de mejora en lo que a uso de mxGraph se refiere. Existen muchas funcionalidades proporcionadas por este framework que no se han utilizado por falta de tiempo y estudio previo. Es posible que, si *EDAG P. S.* requiere nuevas características de la aplicación, se puedan implementar sin esfuerzo y sin necesidad de realizar un análisis previo.

Conclusions and future extensions

Achieved objectives

The functional requirements defined in section 3.2 are satisfied. The solution generated is suitable for use by *EDAG P. S.* when developing projects. Excluding the positive results that have been achieved, other conclusions have been drawn:

- The market does not offer a variety of tools for the type of application and technology used. This has advantages, such as making a unique and robust product and learning technologies that are valuable in the software development industry in the medium and long term, and disadvantages, such as the lack of resources to draw on and the immaturity of the technologies used. Electron is a viable tool at a business level, but the stable versions published during the development of this application caused errors that prolonged this phase compared to the established planning.
- The mxGraph framework is powerful and visual, but at the same time quite complex and required more time than desired to understand the functions required for the application. A functionality that was considered a viable option when it came to saving diagrams was not finally possible. The mxCodecRegistry class provided by the mxGraph library contains a bug when registering custom codecs, causing it to be unusable when required and to take a long time without achieving a positive result in this regard.
- If a library written in JavaScript does not contain defined types to be used in Angular, I consider that it is not profitable to use it and it would be convenient to look for a viable alternative. It was a matter of luck to find an open-source option that would allow mxGraph to be used in Angular, otherwise, it would have taken much longer than expected.

Future extensions

The application has room for improvement when it comes to using mxGraph. There are many features provided by this framework that has not been used due to lack of time and study. It is possible that, if the company requires new application features, they can be implemented effortlessly and without prior analysis.

Bibliography

[1] EDAG Production Solutions GmbH & Co. KG, *Making ideas perform*, 2019. [Online]. Available: `www.edag-ps.de/en/company` (visited on 03/12/2019).

[2] R. S. Pressman, *Software engineering: a practitioner's approach.* McGraw-Hill, 2010, New York, USA.

[3] K. Fakhroutdinov, *State machine diagrams*, 2019. [Online]. Available: `www.uml-diagrams.org/state-machine-diagrams.html` (visited on 03/12/2019).

[4] B. J. Evans and D. Flanagan, *Java in a Nutshell.* O' Reilly, 2019, 7th edition.

[5] U. Brandes, M. Eiglsperger, J. Lerner and C. Pich, 'Graph markup language (graphml)', in *Handbook of Graph Drawing and Visualization*, R. Tamassia, Ed., CRC Press, Jun. 2013, ch. 16, pp. 517–541.

[6] H. D. Benington, *Production of Large Computer Programs*, I. E. A. Department, Ed. 1st Oct. 1983, ch. 5, pp. 350–361.

[7] OMG, *Omg unified modeling language (omg uml), superstructure*, Feb. 2009. [Online]. Available: `www.omg.org/spec/UML/2.2/Superstructure/PDF` (visited on 25/11/2019).

[8] C. E. Shannon and W. Weaver, *A Mathematical Theory of Communication.* University of Illinois Press, 1949.

[9] W3C, *Extensible markup language (xml) 1.0*, 5th Edition, 2008. [Online]. Available: `www.w3.org/TR/REC-xml` (visited on 25/11/2019).

[10] G. Team, *What is graphml?* [Online]. Available: `www.graphml.graphdrawing.org/` (visited on 25/11/2019).

[11] A. Team, *Angular*, 2019. [Online]. Available: `www.angular.io` (visited on 10/11/2019).

[12] A. Team, *Angularjs.* [Online]. Available: `www.angularjs.org` (visited on 10/11/2019).

[13] A. Team, *Angular, architecture overview*, 2019. [Online]. Available: `www.angular.io/guide/architecture` (visited on 10/11/2019).

[14] GitHub, *About electron.* [Online]. Available: `www.electronjs.org/docs/tutorial/about` (visited on 10/11/2019).

[15] *Chromium.* [Online]. Available: `www.chromium.org/Home` (visited on 21/01/2020).

[16] *About node.js*. [Online]. Available: `www.nodejs.org/en/about/` (visited on 21/01/2020).

[17] *Visual studio code*. [Online]. Available: `www.code.visualstudio.com` (visited on 21/01/2020).

[18] *Discord*. [Online]. Available: `www.discordapp.com` (visited on 21/01/2020).

[19] *Mongodb compass*. [Online]. Available: `www.mongodb.com/products/compass` (visited on 21/01/2020).

[20] *Slack*. [Online]. Available: `www.slack.com` (visited on 21/01/2020).

[21] *Mxgraph*, Nov. 2019. [Online]. Available: `www.jgraph.github.io/mxgraph` (visited on 10/12/2019).

[22] Microsoft, *Typescript*, 2012. [Online]. Available: `www.github.com/microsoft/TypeScript` (visited on 03/11/2019).

[23] ——, *Our company*. [Online]. Available: `www.microsoft.com/en-us/about` (visited on 22/01/2020).

[24] L. Lamport, *LATEX: a document preparation system*. Addison-Wesley Pub. Co., 1986, Reading, MA, USA.

[25] Git, 2005. [Online]. Available: `www.git-scm.com` (visited on 22/01/2020).

[26] Wikipedia, *Model-view-viewmodel*, 2019. [Online]. Available: `en.wikipedia.org/wiki/Model-view-viewmodel` (visited on 15/10/2019).

[27] J. Strawn, *Design patterns by tutorials: Mvvm*, 2018. [Online]. Available: `www.raywenderlich.com/34-design-patterns-by-tutorials-mvvm` (visited on 15/10/2019).

[28] *Typescript type definitions / typings for mxgraph*, 2019. [Online]. Available: `www.github.com/lgleim/mxgraph-typings` (visited on 05/11/2019).

[29] Wikipedia, *Mit license*, 2019. [Online]. Available: `en.wikipedia.org/wiki/MIT_License` (visited on 05/11/2019).

[30] ——, *List of unified modeling language tools*, 2019. [Online]. Available: `en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools` (visited on 03/10/2019).

Annex A. Suitable technologies for a state diagram application

When it comes to choosing between different alternatives for developing an application, it is necessary to analyze individually and compare the varied suite of tools that best suits the application we want to make. In this situation, our main goal is to create an application to design state machine diagrams through a graphical interface, and also it has to be able to be used in different operating systems. For this purpose, the options that are offered to us by the market and the different communities of developers, are focused on web applications and desktop applications. Therefore, following the most popular and supported technologies, our goal will be easier to achieve and also more valuable in a mid-term period.

- **Web application technologies**
  A web application is an application program that runs on a remote server, and they are accessible through web browsers. Because of the massive use of the web in the last years is one of the best ways, along with mobile applications, for businesses to influence their presence online.

- **Desktop application technologies**
  A desktop application is an application program that runs stand-alone in a desktop or laptop computer. This solution to develop applications was a standard to many companies, but its use is decreasing due to the massive popularity of web and mobile applications. Despite that, it is a reliable, mature and simple way to develop an application.

In the context of the application to be developed, it is more interesting to write a desktop application than a web application. For this program a client-server model is not necessary, as the intention is for the user to use the application locally on a computer. But the languages and libraries for web development are ideal for designing a graphical interface, because they are used and maintained by large companies and communities of developers. After some research, it is possible to use this type of technology to develop desktop applications, developing a hybrid desktop application (native desktop application with a web GUI), a PWA (Progressive Web Application), or through a dedicated framework (Electron.js or NW.js for example).

PWAs are a type of application delivered through the web designed to work on any platform that uses a web browser. To simplify this, PWAs are web applications that can appear to the user like traditional applications. In spite of being the option with most future in the development of applications, it is focused in generating a desktop application from a web application, which is not what we want to do.

About the hybrid approach, it is a suitable option for our objective, but the process of integrating a WebView component (embedded browser) that renders JavaScript code will make the project too complex. Also, using some of the frameworks mentioned above we would avoid wasting a lot of time in that part.

In order to avoid learning a new framework or another programming language, it has been decided to focus the development of the application using either Java or some JavaScript framework. Both languages are among the most popular for creating web and desktop applications, so it is interesting, apart from the knowledge previously acquired, to compare the available options of both languages.

There are more applicable libraries as a solution to what is wanted to be done, but as previously explained, following the most popular technologies will reduce the complexity of the project. The options listed below are the most commonly used.

- **JavaFX**
  JavaFX is a Java library used to build desktop applications. The applications created with JavaFX can be executed in different devices, such as smartphones, tablets or laptops. Some of the benefits of this framework are:
  - Ease of interface development.
  - Supports Properties and data binding.
  - Custom styling with CSS.
  - Support touch devices.
- **Swing**
  Swing is the standard framework in creating a user interface in Java. As with

JavaFX, Swing components are implemented in Java, which makes it independent of any platform. It also provides a set of advanced components to design a complex GUI.

JavaFX is going to replace Swing as the standard GUI toolkit for Java. Oracle (the company in charge of Java development) answers the following in the JavaFX FAQ section, and because of this reason, we should follow Oracle's recommendation to develop this project using Java:

Is JavaFX replacing Swing as the new client UI library for Java SE?
Yes. However, Swing will remain part of the Java SE specification for the foreseeable future, and therefore included in the JRE. While we recommend developers to leverage JavaFX APIs as much as possible when building new applications, it is possible to extend a Swing application with JavaFX, allowing for a smoother transition.

- **Electron**
  Electron is a platform that makes it possible to develop and build cross-platform desktop apps using web technologies like HTML, CSS and JavaScript. In combination with these tools, it is possible to use different JavaScript libraries and frameworks.

- **NW.js**
  NW.js (previously known as node-webkit) is an app runtime based on Chromium and Node.js, with which it is possible to write native applications in HTML and JavaScript. It supports Node.js APIs and its third party modules.

These ones are the best solutions to develop a cross-platform desktop application using JavaScript or any framework based on JavaScript. Some of the most important differences are shown in this table:

Electron	**NW.js**
• Auto-update mechanism. • Easy build process. • Entry point is a JavaScript script. • Crash-reporting integrated tool.	• Separated or mixed JavaScript contexts. • Integrated source code protection tool. • Entry point can be an HTML or a Node.js file. • Chrome extensions and APIs support. • Print preview support.

But intangibles make Electron more interesting. Community response, well designed documentation and applications like Discord, Visual Studio Code or Slack (made with Electron) are the reasons for opting for Electron.

After these brief analysis, we must choose between software developed with technologies for web applications or with a language such as Java, which is robust, mature and has many references. Based on the list of UML diagram design tools offered by the market [30], most of the tools are developed with Java, which would not bring anything new nor would it be an alternative for anyone interested in using our tool, as there are more complete tools than the one we intend to make. On the other hand, if we look at the tools developed with JavaScript, there are only three solutions:

- Lucidchart by Lucid Software.
- Gliffy.
- Umple by University of Ottawa.

The first two options are not open source nor free to use, which is an inconvenience to potential users. The last alternative from the list has also a not at all intuitive web application with scarce options. Therefore, developing our project using Electron would be a highly interesting alternative. And because in the future, if we want our desktop application to be also a web or mobile application, thanks to Electron the process is easy to perform.

As a conclusion, for the reasons stated in the document, the project will be developed using Electron.

Annex B. User Manual

"State Diagram Tool" is a desktop application that allows the user to design a simple kind of state diagrams. The user may through this tool open diagrams, save diagrams and generate a Java class from the current design displayed.

Installing

To use the application, run the provided ".exe" file with this project. It does not require any installation process.

Building

To build the application, it is necessary to have installed npm package manager and electron and run the following command:

```
npm install
```

Once finished, run the next command and the application will start on development mode:

```
npm start
```

Tool

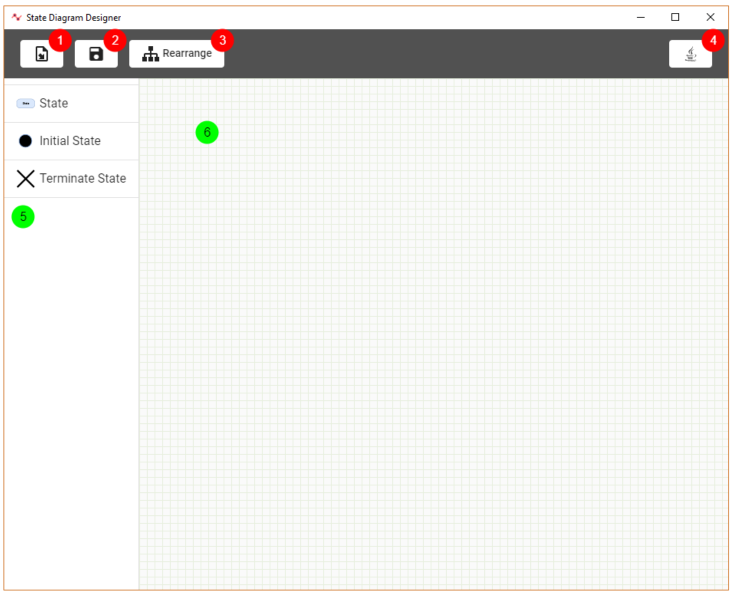
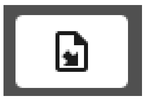
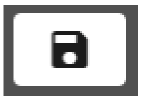


Figure 1: GUI of the application

Figure 1 shows the user interface when opening the application. Red-labeled buttons in the image perform specific functions, green-labeled areas in the figure are the drawing area of the application.

1. 

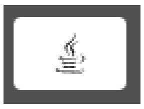
   When clicking on this button, a dialog will open and the user may choose which file should be opened. The tool works only with ".graphml" files without inconsistencies, otherwise, it will throw an error dialog. After selecting a suitable file, the diagram will be displayed.

2. 

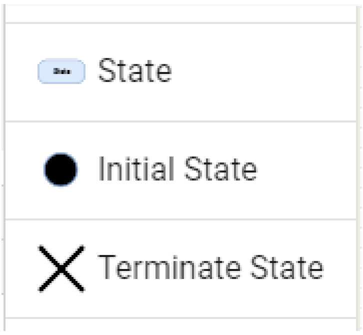
   This button opens a dialog and the user may choose where to save the current diagram in the application. The application will save it in the ".graphml" format.

3.

   When clicking on this button, the current diagram displayed on the drawing area will be reordered.

4. 

   After clicking on this button, a new window will prompt, then the user may select a folder in his computer to save the resulting ".java" file after processing the current diagram displayed on the drawing area.

5. 

   This panel contains the items that can be drop into the drawing area. the user may drag an item of the items panel and then drop it on the drawing area.

6.

   The user may drop and edit dropped elements in this panel on the right side of the items panel. Here the user may perform the coming actions:

   - Change a state name by double-clicking on it.
   - Connect two states by dragging the arrow icon shown when moving the mouse pointer over a state node, and then dropping it on a different state node.
   - Resize a state node by clicking and holding the selection perimeter points displayed when a state node is clicked.
   - Perform the menu actions shown when right-clicking any element on the drawing area, including the drawing area itself.

- Edit the attributes of a transition edge by right-clicking on it and then clicking on the “Parameters” option.