Technical University of Denmark

Matthias Larsen, s103437
Joachim Jensen, s103430

UML

A Graphical User Interface (GUI) for weather radar and wind energy data visualization and analysis

02350 Windows Programming using C# and .Net, December 2012

DTU Informatics
Department of Informatics and Mathematical Modelling

Matthias Larsen, s103437
Joachim Jensen, s103430

UML

A Graphical User Interface (GUI) for weather radar and wind energy data visualization and analysis

02350 Windows Programming using C# and .Net, December 2012

UML,A Graphical User Interface (GUI) for weather radar and wind energy data visualization and analysis

This report was prepared by
Matthias Larsen, s103437
Joachim Jensen, s103430

Supervisors
Bjarne Poulsen

Release date:	December, 2012
Category:	1 (public)
Edition:	First

Department of Informatics and Mathematical Modelling
Technical University of Denmark
Asmussens Alle building 305
DK-2800 Kgs. Lyngby
Denmark

www.imm.dtu.dk
Tel: (+45) 45 25 33 51
Fax: (+45) 45 88 26 73
E-mail: reception@imm.dtu.dk

Abstract

Data for weather analysis and forecasts consists of enormous amounts of data.

Comparing these huge amounts of data quickly becomes difficult. Most people interpret images easier than numbers, hence the need for some sort of visualization for the data sets.

The data come from multiple sources (e.g. on site observations from measuring stations, meteorological forecasts from Numerical Weather Prediction models) and, consequently have very diverse formats (time series, georeferenced data, gridded data).

These diverse formats raise an important issue because no common or efficient platform for visualizing and analyzing these data exists except for two earlier attempts: one in MATLAB and one with Google Maps. These earlier application were not flexible enough to be used in a bigger context.

This paper shows the development and considerations throughout the project of an application that might serve as the beginning of a new common open source platform for analyzing, visualizing and comparing weather related data.

Contents

List of Figures

List of Tables

Listings

1 Introduction

The problem in this project is to analyse and develop an C# .NET application to manage UML class diagrams in a way so that it can be extended with other types of UML diagrams in the future.

This is interesting because different design patterns and methods can be used

Since the prototype demonstration on December 12, the following have been done:

- Save and load uses its own thread
- Text added on relations between classes
- Relations are removed when their respective classes are removed

2 Analysis

A lot of considerations have gone into the development process of the application. The following chapter explains the considerations with the most impact on the application.

TÆLLER 5

2.1 Choice of diagram

While state and sequence diagrams show how (part of) an application works in a detailed way, class diagrams give an overview over all or some classes in the application, what properties and methods they contain and how they are related to each other. In this way, class diagrams can be used to easily see how an application is modelled and thus how it can be altered or extended. Therefore, class diagrams are used more often than state and sequence diagrams when it comes to software architecture, and therefore an application to manage class diagrams has been chosen.

2.1.0.1 Solution strategy

Due to limited time, not all specifications for a UML class diagram can be implemented in the application, and thus we have made a prioritized specification list, an iteration plan, where the first element is of highest priority and will be designed and implemented first.

1. Different types of classes
2. Undo and redo functionality
3. CRUD for classes
4. Movable classes
5. Relations between classes
6. Save and load functionality
7. Automatic save functionality

One should think that *CRUD for classes* and *Movable classes* is more important than *Undo and redo functionality*, but as they are highly dependent on *Undo and redo functionality*, this has higher priority. E.g. if *CRUD for classes* was made before *Undo and redo functionality*, one could risk having to modify it all again in order to get undo and redo to work properly in the end.

2.1.1 Risks

Because of the solution strategy where a functionality has been given a priority corresponding to its importance, the risks of not implementing one of the specifications due to limited time have been minimized. If *Automatic save functionality* was left out, it would not cause the application to work improperly as it has no impact on the actual purpose - that is, creating an application that can manage UML class diagrams.
This is an advantage of using an iteration plan in the process of software development.

3

Design & Implementation

3.1 GUI

TÆLLER 35

The GUI consists of a menu, a two-column grid and a status bar. The features that can be interacted with in the GUI have been outlined below.

3.1.1 Editable classes

There are different gestures for creating, updating and deleting classes, each chosen to get the best user experience. In the left grid column, one can click on a button to create a specific type of class in the application. At the moment there are four types to choose among:

- Class
- Abstract Class
- Enum
- Struct

Clicking one of the buttons will also insert a box in the right grid column representing the class in the application. The color of the boxes are determined by the type of the classes.

Deleting a class is done by right clicking on the specific box representing that class.

If one wants to edit a class, this is done by double (left) clicking on the specific box. This will cause the box to switch to an editable field where one can type in the name, properties and methods for the class in accordance with the syntax of UML class diagrams. Clicking on the same box again or double clicking on another box will save the changes, and the boxes will automatically resize to fit its content. Thus, only one class can be edited at a time.
Choosing inline text edit instead of a pop-up window or likewise is because the UML class diagram content syntax is so simple and understandable and because it

(for a software architect's point of view) is much faster to use this text based UI than pick the different properties and methods etc. in a GUI.

3.1.1.1 Regex

3.1.2 Movable classes

It is possible to freely move the boxes around in the right grid column, but only there. That means that it is not possible to e.g. move the boxes out of the window or into the left grid column.
When editing a box, one cannot move it, but it is still possible to move other boxes.

3.1.3 Undo and redo

3.1.4 Relations

Relations will be created, updated and deleted automatically based on the classes (boxes) present. Also, a text will say what kind of relation

3.1.5 Menu and shortcuts

In the top menu, one can find various options and functions. Each of them has been given a shortcut key combination that correspond to the default shortcut for its type (if it exists). The options and their shortcuts can be seen in table 3.1.

Option	Shortcut
New file	Ctrl+N
Open file	Ctrl+O
Save file	Ctrl+S
Save as	Ctrl+A
Undo	Ctrl+Z
Redo	Ctrl+Y

Table 3.1: Shortcuts of menu options

3.1.6 Status bar

On the bottom of the GUI a status bar can be seen. Here, the user gets messages about what the status is, i.e. what is going on in the application. E.g. if a box is being edited, removed, moved or if a file is being (automatically) saved or loaded.

3.2 Application layer

TÆLLER 20

3.2.1 dll

models

3.2.2 Save and load

The save and load functionality has been implemented in its own thread, because the application could otherwise be halted as these operations were executed.

When saving a file FORMAT

After a file has been saved the first time, the application will automatically save it continuously in a specific time interval. This way, the user does not have to worry about losing hours of work if the computer crashes or shuts down.

3.3 Design patterns

2-3

TÆLLER 10

3.4 Tests

The application has been tested with various use cases that can be found in appendix A.

3.5 Noget der ikke er udnervist i

TÆLLER 5

regex

4

Conclusion

An application to manage UML class diagrams was created successfully. It has been made so that other UML diagrams can be implemented in the future and because all models are saved in a dll, this layer can easily be reused in other applications.

All specifications from the iteration plan REF? have been designed and implemented

The final product ended up being a satisfactorily solution for a UML class diagram application that can manage the content in an intuitive way.

4.1 Future work

4.1.1 Code scaffold

4.1.2 Export

4.1.3 Code scaffold

A

Use cases

Actors:

- User: Person using the application.

A.1 Create a class

Actor: User **Scenario:**

- User left clicks once on Class in the left grid column
- Class *New_Class* is created and shown in the right grid column

A.2 Remove a class

Actor: User **Scenario:**

- User right clicks once on a specific class in the right grid column
- The specific class is deleted and removed from the right grid column

A.3 Edit a class

Actor: User **Scenario:**

- User left clicks twice on a specific class in the right grid column
- The specific class becomes editable
- User types in some valid content for the class
- User left clicks once on the specific class
- The specific class returns to normal state and changes are saved

Alternative scenario:

1. User types in some invalid content for the class
2. Invalid content gets ignored on save

A.4 Move a class

Actor: User **Scenario:**

- User left clicks once and hold on a specific class in the right grid column
- User drags class around in the right grid column
- User releases mouse button and the new class position is saved

Alternative scenario:

1. User tries to move the class out of right grid column
2. The class is stopped by the edges

A.5 Save a file

Actor: User **Scenario:**

- User goes to File > Save (or clicks Ctrl+S)
- A Save File dialogue pops up
- User specifies a name and clicks Save
- File is saved

Alternative scenario:

1. User cancels Save File dialogue
2. File is not saved

A.6 Open a file

Actor: User **Scenario:**

- User goes to File > Open (or clicks Ctrl+O)
- A Open File dialogue pops up
- User picks a relevant file and clicks Open

- File is opened and content shown in the application

Alternative scenario:

1. User cancels Open File dialogue
2. File is not opened and current content remains in the application

www.imm.dtu.dk

Department of Informatics and Mathematical Modelling
Technical University of Denmark
Asmussens Alle building 325
DK-2800 Kgs. Lyngby
Denmark
Tel: (+45) 45 25 33 51
Fax: (+45) 45 88 26 73
E-mail: reception@imm.dtu.dk