TNM090 — Software Engineering
3. Code Generating Tools

October 9, 2012

1 Introduction

There is an abundance of tools for assisting the development process. In some cases, for some domains, these tools may even assist in writing code. This laboratory exercise is on the topic of tools for generating code. Here you get to try out examples of tools for generating code, as a means to speed up the development process.

1.1 Purpose

In the first part of this exercise you will experiment with UML to structure and understand your program, and as use it as a source model for C/C++ code. The purpose of experimenting with this kind of tools is not primarily to provide you with a code producing tool, but instead to see how UML models at differently detailed levels relates to actual code. Note that in some cases a programmer is capable of producing code from the UML model while the UML software cannot, or produces less detailed code. This is often because the programmer can make assumptions of things not explicitly defined in the model.

In the second part you will experiment with a GUI builder tool. This is more commonly used in real development processes for generating actual software. With Qt, you will, apart from trying out a powerful, cross-platform GUI API, also experience how a strongly structured framework may look and work, and how this will affect the actual program using its API.

There are many steps involved in this exercise and the basic structure is shown in figure 1.

You will be using version control during this exercise. Make sure that you check in all source files that contribute to the final system, even UML structures and GUI configuration files. Everything that is required to build the system from scratch should be version controlled. Therefore, ommit anything that can be automatically generated, such as object files and both images and binaries that come from automatic tools.

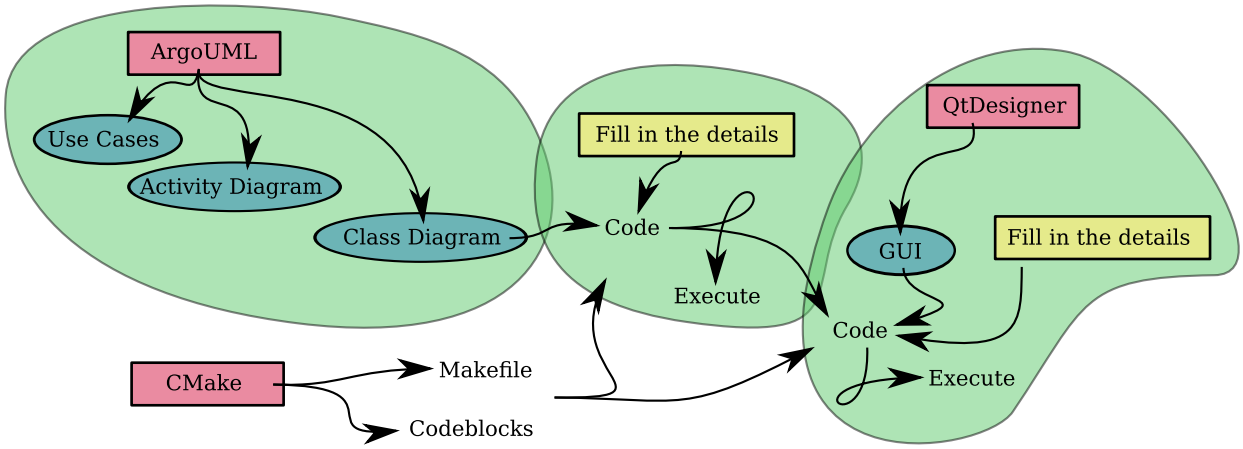

Figure 1: The basic structure of this exercise.

1.2 Documentation

This is not a manual so you are expected to search for information in manuals, tool documentation and command-line help. Useful sources for this exercise are

- ArgoUML guides and documentation at http://argouml.tigris.org/
- Qt Manuals at http://qt-project.org/doc/qt-4.8/
- Qt Designer Manual at http://qt-project.org/doc/qt-4.8/designer-manual.html
- Getting to Know Qt Designer at http://qt-project.org/doc/qt-4.8/designer-to-know.html
- Example code

1.3 System to Implement

For this exercise you will create a basic framework for a 3D Particle System and provide implementations of some functionality and a demonstration user interface. These types of systems are often used in games because of their impressive effect with relatively little code. You will probably end up with several classes, but each will require a small amount of code.

The system should handle particle motions and let particles die after their life-time has expired. It should also have support for sources that generate particles, e.g. explosion source and cone source, and for force effects that pushes them around, e.g. gravity effect and wind effect. Observe that gravity is either a constant force in a direction or a force towards a point in space with magnitude determined by the particle-gravity center distance, while the wind has a speed so that the force will be determined by the *relative* speed of the particle.

In your graphical user interface you will provide a visual rendering of the particle system and means to control some selected parameters for view, effects and sources. Therefore, provide functions to update parameters at run-time and a draw function that upon call will use OpenGL to draw each particle, for example using `GL_POINTS`.

You may select another system to implement, however make sure that it is not too simplistic or too extensive as the main purpose of this exercise is to let you familiarize with these types of tools and learn about their advantages and limitations. Also, read through all instructions to make sure that you can complete the exercise with your own idea.

1.4 Examination

After finishing all tasks in this exercise, show you results to a supervisor and explain what you have done and what conclusions you have come to.

For the successful examination, please make sure that you fully understand what you have done. Also, during the tasks, save all intermediate steps so that you can show and explain the result from one individual task.

2 UML Editor

There are many different tools available for drawing the program design. They provide different features and different levels of strictness. Some are capable of exporting the final diagram as source code. These have to be more strict and can be perceived as cumbersome to work with, impeding the creativeness. The more advanced UML editors can export the structure to at least one programming language, many support a multitude.

There are many different UML editors available for various platforms. In our environment we have installed ArgoUML, however you are welcome to use any tool of your liking as long as you can complete all tasks. Activate the C++ Profile in ArgoUML to expose C++ types. Under File / Project properties / Profile, add the C++ Profile.

To better understand the problem at hand, it can be useful to create a use case diagram showing all typical uses at a very high level. Then, with the use cases known, it is time to analyze possible activities for these use cases. Based on this, you may sketch the basic components and interactions, into which you finally enter the details.

Design your system to have two sources and two force effects, but open for the future inclusion of more sources and effects.

Task 1 — Use Case Diagram:
Create a complete use case diagram over the system. We accept that this diagram might be quite plain.

Task 2 — Activity Diagram:
Create an activity diagram over examples of activities with your system, such as drawing the particle system when one source and one effect are active, or adding an effect. Include possible failure, for example if you have too few operands for the selected operation.

Task 3 — The Basic Class Diagram:
Create a basic class diagram containing the classes that will be involved in the final system and specify associations.

Task 4 — Class Contents:
Add members to the classes, both functions to allow the classes to communicate and attributes that control their internal states and aggregations.

Task 5 — Export the Code:
Use the code export function of the UML editor to create the code structure for the system.

At this point, reduce the system to a subset to actually implement. Select one source and one force effect to include in the final application, while still maintaining the expendable structure.

Task 6 — Final Implementation:
Fill in the details in the code to allow all functions in the system to perform their respective tasks.

[Note: ArgoUML will overwrite your code if you export again! Consider using `kompare` to help patching in old code into an updated structure.]

Task 7 — Execute:
To make sure that you system works as expected, create a main function that calls your system and perform a set of actions. Try both successful actions and unexpected actions that lead to graceful failure.

3 GUI Builder

Now, when you have a working system, the task is to create a user interface for it. To our help we have a GUI designer in which you can easily build the user interface. There are many ways to connect the design from a GUI designer to a program: write all code in the designer, build a GUI resource database that is loaded at run-time, or compile the GUI into separate source files.

These instructions and the example provided for the exercise assume that you use QtDesigner. The result from the designer will end up in header files and by implementing a class that extends the designer's base class, you can fully control the behaviour of buttons. Again, you are welcome to use any tool of your liking as long as you can complete all tasks.

[Note: Use CMake to build your system. It will help you set up and build together with Qt.]

[Note: You will need to include an OpenGL view in your GUI. To do this, first create a header file defining your View-widget that extends QGLWidget. In designer you add a "Widget", right-click on it, select 'promote to' and fill in the form to specify you class name for the widget and in which header file to find it. The base class for QGLWidget is QWidget.]

Task 8 — Draw a user interface:
Use QtDesigner to draw a simple user interface for the external control functions of your program.

Tip: Create a MainWindow and drag-and-drop components into the window. Use a hierarchy of layouts to get the structure you like. The layout of the top level widget (`centralwidget`) is set (after adding components) by right click on the surface and the subsequent selection of layout.

Task 9 — Update Program Structure: (optional)
Go back to the UML diagrams over your program and modify the structure if necessary to accomodate for the connection with the new user interface.

In Qt simple behaviour is implemented by connecting signals from components into slots in other components or in your program. The header files include declaration of signals and slots and the designer outputs code that connect these.

There is an edit mode for drawing connections between components, and a signals/slots editor on the right-hand side of the designer's main window. You can also right-click on the top-level object in the Object Inspector editor to the right, and select 'Change signals/slots'. This allows you to add slots to your program to receive events.

Tip: To better see and draw connections in the designer, temporarily enlarge the window.

To get the source code, the .ui files created by the designer are compiled using the `uic` tool. Manually this is done like this

```
uic thedialog.ui > ui_thedialog.hh
```

however CMake will create a build configuration that automatically does this for you.

Take a look at the resulting file. At the top there is some information and a warning that any changes will be overwritten if you recompile the GUI. At the bottom there is a class definition. This is the class that you should extend.

Take a look at the example implementation provided with this exercise. Create a new document in QtDesigner for you application.

Task 10 — Connect the GUI:
Create a class that extends the base class provided by the designer and implement the connection between the GUI and your program. You will have to declare slots in your program to which you should connect the signals in your GUI.

Task 11 — Implement the Connections:
Fill in the necessary code to implement your previously declared slots, so that your program is properly called upon GUI events.

4 Handling Changes

An important part of the software development process is the handling of changes in the requirements and in the system. At this point you have a working framework with some important features implemented and a working system using this framework. Now it is time to extend the system.

Task 12 — Add a Feature: (optional)
Open up the UML editor again and add a missing feature. This can be a new source or a strange effect. Export the new code structure and incorporate both old and new code into the new structure.

[Note: ArgoUML will overwrite your code if you export again! Consider using `kompare` to help patching in old code into an updated structure.]

Task 13 — Refine the GUI: (optional)
Open the designer again and change the layout, for example by regrouping buttons. Also, add another widget that links to a the new feature in the system.