HelloUML
Top Level Design
Author: Joshua Tasman (jmt)
Rev 1

1 Introduction

The top-level design for a visual design editor, helloUML, is presented. This project is partially inspired by the *ROSE* system; we hope that helloUML will be much easier, intuitive, and helpful to use.

1.1 Requirements

The original requirements document can be found at: http://www.cs.brown.edu/courses/cs190/asgns/2-2/mjn2.txt. These evolved as the project developed.

1.2 Specifications

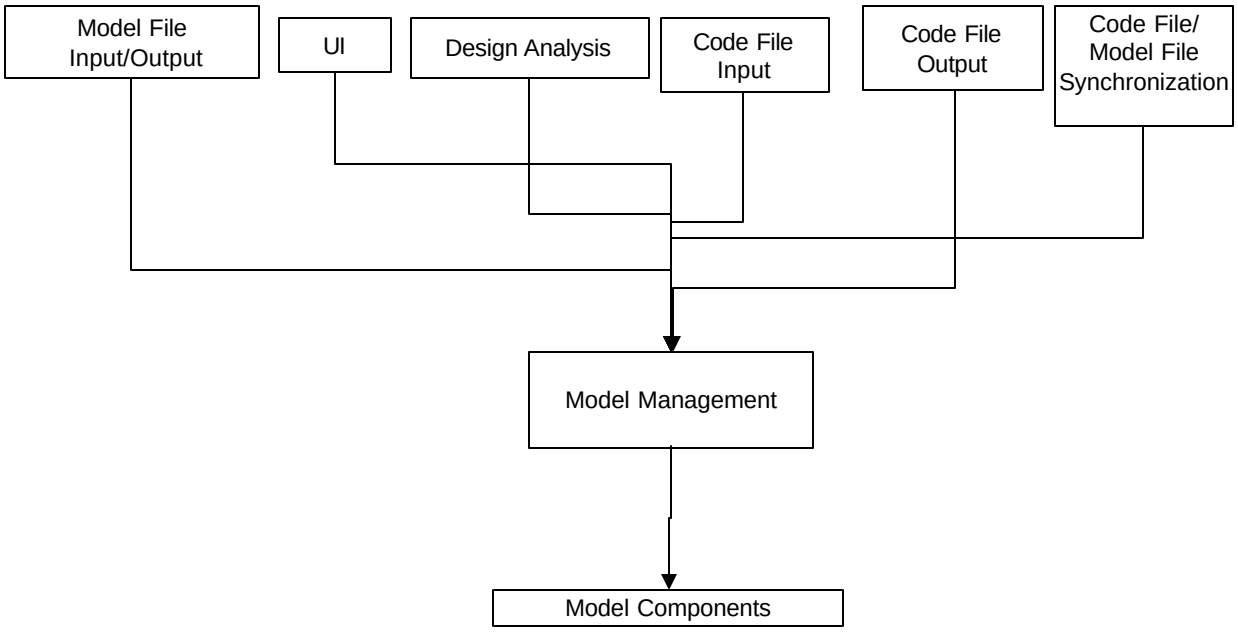
The document http://www.cs.brown.edu/courses/cs190/HelloUML/HelloUML.pdf describes the specification of the current version of the project.

1.3 Changes

In this document, I propose adding the functionality of loading existing code files into the system and deriving a modeled representation. This will be explained in more detail throughout the rest of this document.

2 High-level Diagrams and Description of Components

Figure 1- Levelized Top-Level Component Diagram



Three levels of the system are presented in fig. 1 in relation of *dependency*. At the base are the *model components.* The inheritance diagram for these classes is displayed in fig. 2. These components contain information about the visual presentation and layout as well as *functional* information. For example, a *class* object contains the screen coordinates for display as well as links to all of the *connections* and *groups* it is associated with.

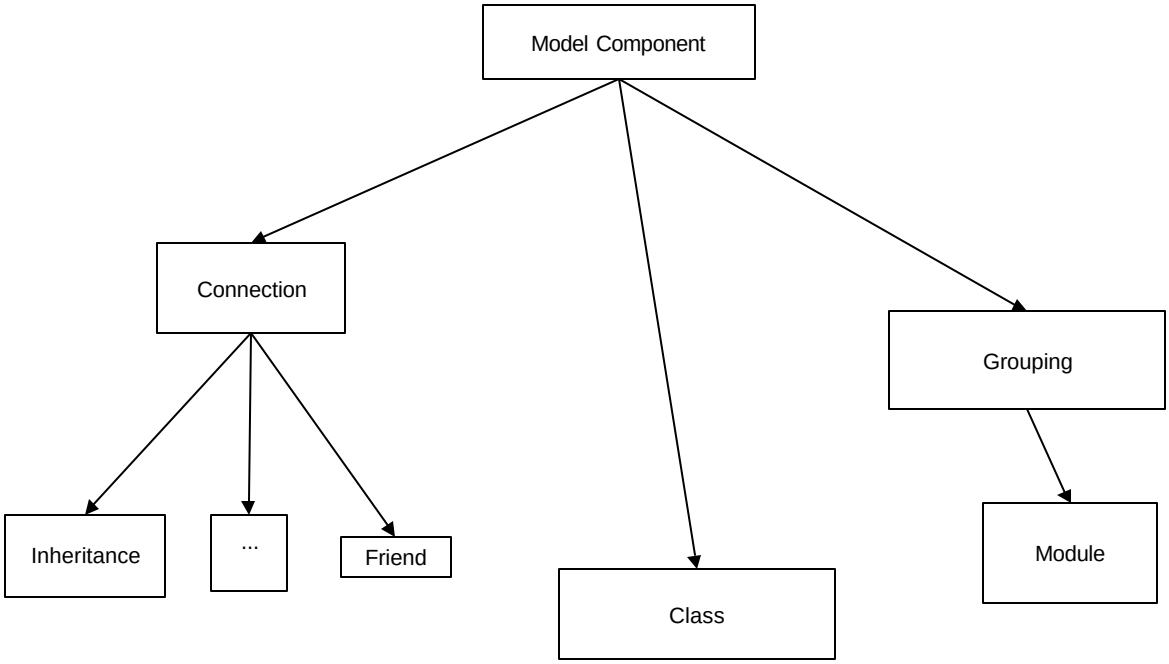
At the middle level, the *Model Management* component contains functionality for maintaining the model- such as automatic layout of parts. It handles communication between the model data structures and the top-level components; it also maintains a list of the most recent actions to the model, and their inverse actions, for *undo* functionality. At the highest level, the model is not stored as a graph, as items may be drawn on the screen in any order, and are not required to be

connected [see note below under *Design Analysis*]; the model is merely a list of model components. The components themselves contain the connection information, as described above.

The top level consists of several components: The *UI* subsystem allows the user to interact with a visual representation of the model, and access the rest of the system. *Model File Input/Output* reads and saves files describing the complete state of the model, including visual information (screen location, &c.) The *Code File Output* components generates stub code for the selected language (Java or C++) in a safe manner (as discussed in the specifications).

The remaining components are the most complicated. The *Code File Input* system is my contribution to the specifications. This system allows an alternate method of model entry: the user to input a pre-existing C++ or Java file. This file is parsed and analyzed, and a model is created directly from the file. The *Design Analysis* system logically reviews the model, checking for "illegal" situations (such as a class inheriting from itself), and generates a list of errors and warnings. [Please note: the specifications were not entirely clear on this point and I have elaborated.] I believe that is it important for the user to be able to draw anything in the editor, in any order, regardless of the correctness of the model. So, the *Design Analysis* system may, at the user's discretion, be an interactive part of the experience (for example, not allowing a user to make an illegal connection in the first place), or run as a "check" on a model. The *Design Analysis* system also is used for reviewing models and suggesting conceptual patterns and groupings. Lastly, the *Code/Model File Synchronization* system ensures that model and code files match up, as code files may have been edited between runs helloUML.

Figure 2- Model Components Inheritance Diagram



3 External dependencies

We can use the Qt library for the GUI- this should help with any future cross-platform development.

The proposed addition of the code reader section will require use of a standard scanner/parser generator such as *LEXX* and *YACC* (or their derivatives.)

4 Task breakdown and Group Organization

Members:

Joshua Tasman
Michelle Neuringer
Greg Getchell

Toan Pham
George Quievryn
Christopher Filippis
Di Wang
Freddie O'Connell

[ ] indicates possible group member assignments.
{ } indicated expected involvement with coding.

Administrator: In charge of the entire project. Final authority for decisions about the project. It is the administrators job to schedule meetings, keep the project on track, personnel issues, &c. {Some coding.} *[Toan Pham]*

Documenter/Librarian: In charge of maintaining the code library- understanding and knowing the current versions of all aspects of the system; also maintains internal documentation and user manuals. This person should make sure that the architect's vision is communicated to the group, and that everyone can keep an up-to-date idea of what changes occur. {Minimal coding, if desired- this role is one of the most important and will require a large time investment.} *[Joshua Tasman]*

Tool support: Creates function libraries, scripts, &c. Sets up systems such as source control. Should be a Unix/C++ guru. {Minimal coding outside of tools.} *[Freddie O'Connell]*

Architect: In charge of system design. The architect must keep administration, testing, and coding talking and working together. {Moderate coding.} *[Michelle Neuringer]*

Testing: In charge of overall testing. A large part of this can be working with coders to make sure they self-test their code. Another aspect is to set up and run test cases on the rest of the groups' code. {Minimal coding- testing will require a lot of time to do correctly} *[Greg Getchell]*

[code group: These people do the bulk of implementation. The lead coder does most of the interfacing with the architect and is in charge of making sure this group works together.]

Lead Coder: Decides on the specifics of implementing the architect's design. Primary coding.} *[Di Wang]*

Coder (x2): Support for the lead coder. Focuses on specific tasks sketched out by the lead coder. {Primary coding.} *[Christopher Filippis]*, *[George Quievryn]*

5 Schedule

m 3/6: Group Roles Assigned
w 3/8:
f 3/10: Initial group design completed.

m 3/13: Version Control set up
w 3/15: GUI prototype drawn out.
f 3/17: Interfaces prototyped
m 3/20: Initial Internal Documentation
w 3/22: [break]
f 3/24: [break]
m 4/3: Interfaces Completed
w 4/5: Detailed designs completed- Coding Begins
f 4/7:
m 4/10:
w 4/12: Design Freeze
f 4/14:
m 4/17: Initial integration.
w 4/19: External Testing Phase begins
f 4/21: Initial User Documentation
m 4/24:
w 4/26:
f 4/28: Functionality freeze, in class demo.
m 5/1:
w 5/3: Public Demos.
f 5/5: Wrap-up/Evaluations.
m 5/8: Final Code Freeze
f 5/12: Final Demos, Final Documentation.

6. Summary

To review, I have added the functionality of working with pre-existing code files, and described a top-level design for a UML editor which contains these changes. Testing occurs at all levels of development, and is supervised by a team member who works with the coders in addition to being in charge of the external testing. The schedule is designed so that a large time is devoted to designing before any code work occurs; testing is given an equally large division. Documentation, both internal and external (user manuals) will be created and updated throughout the project.