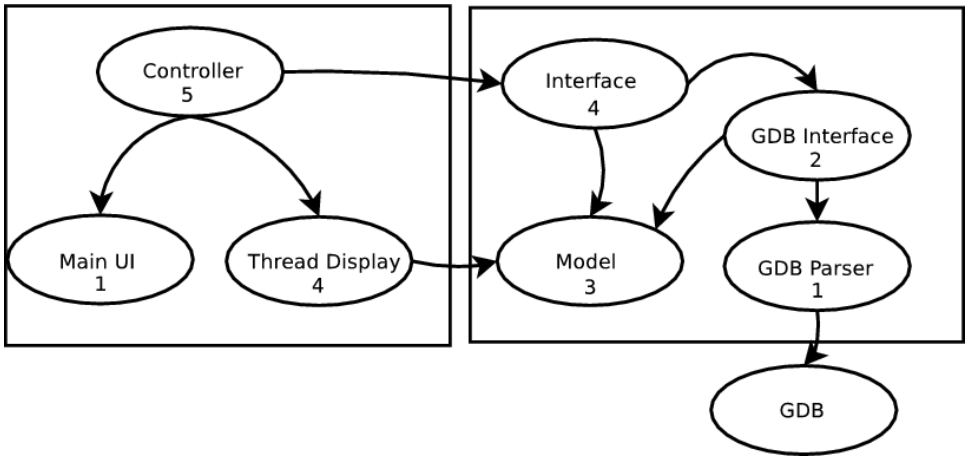
Thread Debugger

1 Components



1.1 Controller

The main program. Interacts with the back-end interface.

1.2 Main UI

The main UI provides unified GUI functionality.

1.3 Thread Display

Generates a display and control based on the information p by the model. The model will push this information when the interface 'orders' it to.

1.4 Interface

All requests to the backend are processed and delegated by the interface. This is the only component of the back-end that the front-end needs to interact with.

1.5 Model

The model is an abstract representation of the state of the threads and files. It isn't dependent on the implementation of the debugger. It tracks breakpoints, control, and other information.

1.6 GDB Interface

Provides communication between the Model and GDB, and the Interface and GDB. Communications are processed through the GDB Parser.

1.7 GDB Parser

Parses the output from GDB.

2 Dependencies

As stressed throughout the specification, the greatest dependency lies in the behavior of GDB. If GDB does not provide the ability to step threads independently, the functionality of Thread Debugger will be severely reduced. Additionally, the output produced by GDB may be insufficient to successfully model the threads. This could make it impossible to detect thread deadlocking.

The specification also assumes that the coding team is familiar with OCaml. It may not be possible to bring the team up to proficiency with the language within the time frame of the project.

3 Task Breakdown

3.1 UI Czars - Nathan, Owen

The UI Czars will work to make ThreadDebugger the most outwardly polished user friendly application since ~~the Gimp~~ Photoshop.

3.2 Thread Display Baron - Owen

The Thread Display Baron will work to process and render the model for user interaction.

3.3 Model Csars - Colin, Brendan

The Model Csars will take the parsed and processed output of GDB and use it to prepare a model of all of the threads. If they get ambitious, they'll extend the model to perform deadlock detection and other worthwhile tasks.

3.4 Parsing Tsars and GDB Interaction Squad - Sean, Josh

The Parsing Tsars are responsible for rationally processing the output of GDB and developing a sensible way to tell GDB what to do.

3.5 Frontend/Backend Integrators - Lincoln, Josh

This group will ensure that there exists seamless interaction between the frontend and backend.

4 Organization

4.1 Project Manager - Dominic

It is the responsibility of the project manager to schedule tasks and track progress to ensure the overall success of the project.

4.2 Architect - Lincoln

The architect directs the overall structure of the project, directing how components should interact. The architect has the final say over software engineering decisions.

4.3 Tool Tyrant - Lincoln

The Tool Tyrant is responsible for supporting the coding team (i.e. svn, makefile).

4.4 Documentation Duke - Dominic

The Documentation Duke will strive to ensure that the question "What the heck does this code do?" will never be intelligently asked.

4.5 Coders - Nathan, Owen, Brendan, Josh, Sean, Colin

Detailed in task breakdown.

4.6 Testing Czarina - Tara

The Testing Czarina will scold any individual who fails to perform rigorous component tests. She will also recommend component tests to the coders and perform systems tests.

5 Schedule

3/9 – Final Design
  Devise preliminary tests
3/23 – Final Interfaces
4/6 – Basically Functional Program
  Lock-in component tests
4/13 – Functional Program (i.e. controller)
  Lock-in systems tests
4/20 – Program for Debugging
4/27 – Class Demo
5/7 – Public Demo
5/17 – Final Demo

6 Assumptions

No intentional assumptions were made about the spec

7 Testing

Rigorous (and possible exhaustive) component tests **must** exist for each part.
No interaction between parts should be assumed correct unless proven.
Each component must have the capability of producing well-formed fake output to prevent testing bottlenecks.