Mass Debugger  
Interim Report

Niko Simonson  
December 13th, 2011

Summary
The initial conceptual challenges in the development of the MASS Debugger are being addressed. These were:
• conceptualization – what needs to be written in a debugger
• communication between the debugger, the user's class, and MASS
• development of an interactive graphical interface – how to display graphics, and handle events

Additional and ongoing difficulties are as follows:
• Further details in the interactivity of the graphical interface – avoiding busy waits, coding further intricacies of a graphical interface, and dealing with new problems as they crop up
• Testing in a UNIX environment – difficulty with the xll display variable and tunneling, which may be unrelated to the code itself
• Refactoring of existing code – the class contains large amounts of exploratory coding and comments that need to be eliminated before a beta release

The current state of the project:
• Communication with MASS is very smooth.
• Able to display MASS data with 2D graphics.
• Some interactivity, but not fully functional.
• Basic debugger features included in code, but not fully functional.
• Remaining question of where a debugging loop should be implemented: by the user or by the debugger. Currently possible to do both, but may wish to eliminate one or the other.
• No 3D graphics.
• No means to display exchange of data or agent information.
• Messy code, not ready for release outside of the team.
In terms of outlook for the next quarter:

- Prospects for the completion of non-3D basic debugging features are excellent.
- Prospects for the completion of 3D interactive display are fair -- more analysis is needed.
- Prospects for the completion of data exchange features is very good, but must use methods beyond the basic MASS wrapper to implement.
- Completion of agent and agent migration features are good in the long term, but not favorable next quarter due to the time to implement the above. The current plan is to hand off the project to a successor at the end of winter.

**Goals**

For this quarter the project had the following goals:

- Complete initial project analysis, identifying risks.
  - Lifecycle Analysis Document
  - Expanded Use Cases
- Complete initial project design
  - Rough System Design Document, method calls
  - Expanded System Design Document
- Development - Code priority 1 features, as reflected by the following functional requirements:
  - 2.2.1.1 Incorporate debugger into code.
  - 2.2.1.2 Set a break point.
  - 2.2.1.3 Choose between 2D or 3D visualization.
  - 2.2.1.4 Open GUI
  - 2.2.1.5 Close GUI
  - 2.2.1.6 Step through data.
  - 2.2.1.7 Navigate to desired 3D cell.
  - 2.2.1.8 See contents of a selected cell.
  - 2.2.1.9 Hide contents of a selected cell.
- Interim Report
- Final System Design Document
**Status**

Here is the current state of the project in relation to the schedule for this quarter.

- Complete initial project analysis, identifying risks.
  - Lifecycle Analysis Document – completed successfully
  - Expanded Use Cases – completed successfully
- Complete initial project design
  - Rough System Design Document, method calls – notes only
  - Expanded System Design Document – not done
- Development - Code priority 1 features, as reflected by the following functional requirements:
  - 2.2.1.1 Incorporate debugger into code. – completed successfully
  - 2.2.1.2 Set a break point. - method and data members allow for a break point but not fully functional
  - 2.2.1.3 Choose between 2D or 3D visualization. - currently 2D visualization only
  - 2.2.1.4 Open GUI – completed successfully
  - 2.2.1.5 Close GUI – completed, some disposal issues remain
  - 2.2.1.6 Step through data. - method and data members allow step by step iteration, but not fully functional
  - 2.2.1.7 Navigate to desired 3D cell – design approach exists (layer buttons on the side of each "slice") but not implemented
  - 2.2.1.8 See contents of a selected cell. - implemented but not fully functional
  - 2.2.1.9 Hide contents of a selected cell. - completed, some disposal issues remain, further controls needed

- Interim Report – this document
- Final System Design Document – not done

**Green** highlighted items are complete.
**Yellow** highlighted items are mostly complete or nearing completion but have remaining issues.
**Orange** highlighted items are partially complete, but significant conceptual issues remain, requiring further analysis
**Red** highlighted items were not done this quarter.
In terms of the initial schedule

- Analysis was completed on schedule
- Design is incomplete – I had difficulty putting the design portion of my classwork, specifically CSS 370 which I'm taking concurrently, into practice. Design focused on much larger structures than the current project.
- Development is two weeks behind schedule – The Zero Feature release was completed one week behind schedule, but the project is not yet feature complete.

Iterative development will continue to identify risks and problems requiring further solutions.

**Project Information**

**Compilation – General**
Files required:
- Debugger.java
- MASS-Thread.jar
- a Java application that uses the debugger (Driver.java)

**Compilation – Unix**
Compile in the command line prompt of the local directory:
javac -cp MASS-Thread.jar:. Debugger.java
javac -cp MASS-Thread.jar:. Driver.java

**Compilation – Windows**
Compile in chosen development environment, with MASS-Thread.jar added as an external jar. Otherwise, compile in the command line prompt of the local directory:
javac -cp MASS-Thread.jar; Debugger.java
javac -cp MASS-Thread.jar; Debugger.java
Execution – General
The Debugger does not run on its own. A driver program is required to instantiate a debugger object and call its methods. The driver program must be written in a way that utilizes MASS:
• extends Place
• calls the super class in the constructor
• implements callMethod

The driver then instantiates a Debugger object, and uses Debugger methods instead of MASS methods, including calling loops and setting breakpoints. The Debugger initializes and finalizes MASS, and runs MASS methods.

Execution – Unix
Execute from the command line prompt of the local directory:
java –cp MASS-Thread.jar:. Debugger

Execution – Windows
Execute from chosen programming environment. Otherwise, execute from the command line prompt of the local directory:
java –cp MASS-Thread.jar; Debugger

Difficulties and Resolutions
The main challenge this quarter was developing a conceptual approach to the implementation of a debugger. For several weeks, the implementation of what was necessary seemed opaque. This was compounded by a certain amount of confusion over the nature of MASS, itself, and anxiety about the weight of the project.

I opted for an incredible approach that emphasized analysis and design, running parallel with the CSS 370 in Analysis and Design I’m taking this quarter. I believe this aided me in the analysis area, as I was able to identify the major difficulties early and focus my resources towards resolving them. In fact, at the current time, I believe the last major conceptual difficulty has been resolved.

Design is proving to be more problematic. The design portion of CSS 370 seems to be focused on larger-scale projects than what I’m working on. At the moment, everything is fitting into a single class. Breaking the design up into multiple classes will probably be necessary in the near future, but right now I am prioritizing functionality. It is important that everything work so that I can address risk.
The big risks were, in order, the conceptualization of debugger functions, the ability of the debugger to gain information from MASS, and the development of a graphical interface. All of these have been largely resolved.

Conceptualization of debugger functions was solved thanks to the regular analysis of the project. The debugger can gain information from MASS by wrapping MASS commands and acting as a broker between a MASS-friendly application and MASS. Development of a graphical interface for display was aided by the example programs already present, especially wave 2d. The interactive graphical interface was dependent on a great deal of personal research, but did not constitute any ground-breaking activity.

Initially this communication was thought to include Java Reflection, but for CallAll data, a wrapper is sufficient. However, when it comes time to report on data exchanges, the lack of a return value for ExchangeAll and ExchangeSome indicates that some other approach is needed.

A large breakthrough came when I found exactly how the MASS code can be divided between two different classes. The debugger truly exists as an independent class that can be instantiated in a MASS-friendly class, now.

In recent weeks, a lot of this confusion has lifted, thanks to the fact that code is now performing. The remaining challenges for priority 1 features, other than the 3d implementation are largely dependent on design details, such as where to put a debug loop.

**Looking Ahead**
Here is a proposed schedule for next quarter based on the current state of the project.

- Completion of Priority 1 Alpha release (not including 3d implementation) at the beginning of the quarter
- 3D implementation in January
- Priority 1 Beta in early February
- Priority 2 Alpha release at end of February
- Priority 2 Beta release at the end of the quarter
- Initial implementation of priority 3 features and hand off to successor in March.
The remaining major risks are as follows:

- 3D interactive implementation
- Capturing data exchange information (Priority 2 Features)
- Creation of handoff documents to minimize repeat work.