Software Requirements Specification for Ouroboros

Version 0.2

Prepared by Team Cobra

Senior Project 2014-2015

7 October 2014

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Gabriel Marcano	**09/23/14**	**Initial version**	**0.1**
Cobra	**10/07/14**	**Initial draft version**	**0.2**

1. Introduction

1.1. Purpose

The purpose of this document is to describe in detail the functional and non-functional requirements for the Harris embedded web server project, also known as Ouroboros. This includes but is not limited to use case analysis, requirement specification, project scoping and release specification. This document is written with developers, end users, project sponsors and Harris RF in mind.

1.2. Document Conventions

Every detailed requirement in this document has its own specific priority. Requirements are ordered by overall priority, with higher priority requirements having lower numbers. Major sections start with size 18 bold font with a single digit, and subsections for that section are font 14 with the same digit as the parent with a subsequent digit indicating their positioning in the section.

1.3. Intended Audience and Reading Suggestion

This document is intended for developers, project managers, testers, and installation staff. Contained in this document are the system requirements, constraints, and core features of the system. Efforts have been made to use as little jargon as possible for ease of reading. A listing of commonly used terms and their descriptions can be found in the glossary section. It is recommended that all readers of this document begin by reading the Overall Description and then to branch into other parts of the document as needed.

1.4. Project Scope

The purpose of Ouroboros is to provide a solution for developers to help speed up embedded Linux device prototyping. The scope of this project includes the ability to give the code generation platform a configuration file from which it will generate customized code to interface with the Mongoose web server to handle parameters and functions described in the configuration file. The customized Mongoose server, when compiled and run, will then provide a REST and C++ API to allow for the modification of the parameters and call the functions described by the configuration file used to generate the custom code. Files will be provided explaining the dependencies of the system, as well as how to compile the generated code.

A sample configuration file will be provided, along with the generated code from said configuration file. Compilation scripts and/or instructions will be included in order to create the binary for the sample server. Some sample 3rd party applications interfacing with the sample server will also be provided, along with accompanying compilation scripts and/or instructions.

The overall goal of this project is to provide developers a platform they can use to facilitate and speed up testing of Linux embedded devices through the use of configuration files and code

generation. See the Vision and Scope document for more details.

1.5. References

Title	Corresponding File/Address
Vision and Scope Document User Classes and Characteristics	Vision and Scope.docx User Classes and Characteristics.docx

2. Overall Description

2.1. Product Perspective

Currently there does not exist a platform for the quick configuration of prototyping servers for Linux embedded devices at Harris. Developers have to develop their own solutions for each prototype they wish to test, taking time away from other development tasks, such as actually testing the prototypes. Ouroboros will provide a platform to allow developers to create quickly customized servers for Linux embedded devices, reducing, if not eliminating, the overhead required to set up said prototyping environment.

2.2. Product Features

1. Ouroboros will provide a means for describing the operating state of a system, consisting of parameters and external functions, by means of a configuration file.
2. Ouroboros will use the provided configuration file to generate code that can be compiled and run in an embedded system that can run a Mongoose web server.
3. Ouroboros will, in addition to generating custom embedded server code, also generate a Web UI using HTML and CSS that will be served by the custom server compiled from the generated server code. The Web UI will allow for the modification of the system state via a web interface.
4. Ouroboros will provide a REST API through the compiled embedded server that can be used to modify the state of the device, including changing parameters and running functions specified in the configuration file.
5. Ouroboros will provide a C++ API with the compiled embedded server that can be used by 3rd party software run on the embedded device to modify server parameters and register and execute functions described by the configuration file given to the platform for server creation.
6. Ouroboros will provide a callback mechanism for its C++ and REST APIs in order to allow for 3rd party software to respond to server state changes.

2.3. User Classes and Characteristics

Refer to the User Classes and Characteristics document for details.

2.4. Operating Environment

The system will be comprised of two components, a code generator platform, and the resulting binary produced from the code output by the code generator platform. The code generator platform will run in a developer's development environment, provided they have Ruby available. The developer will need to provide a cross-compiler for the compilation of the generated code, and the resulting binary will be run in the Linux embedded device to be used for prototyping. The binary will run a web server in the Linux embedded device which will allow configuration via a server HTML page. Furthermore, the server can also be accessed and configured via a REST and linked against through a C++ API.

2.5. Design and Implementation Constraints

The code generation aspect of the system must be implemented in Ruby. The web server component of the system must be completed using Mongoose. The configuration file for the system must be in XML, with an accompanying XML schema to validate itself. This configuration file will also specify all aspects of the system functionality such as software state, external references and hardware to control before code generation begins. The web component of the API must conform to REST and be written in C++. The local API to interface with the system must be written in C++.

2.6. User Documentation

To create the XML configuration file for code generation, a Prototyping Developer will be able to refer to the README file in the open source repository. To see documentation on the functionality of the generated code, the Third Party Developer can then refer to the auto-generated documentation in the Web UI. The documentation will also be available offline in a documentation folder generated by the code generation platform.

2.7. Assumptions and Dependencies

Ouroboros will assume that the person using it has some technical experience with software. The user should understand how to make use of the local and web API's as well as how a Mongoose web server works without being explicitly told by the application. The installation process provided for Ouroboros will only install the necessary components to run the code generator and explain how to compile the web server. If third party applications or any other form of external code would like to be used with the system, those dependencies must be installed separately. The initial version of the system will be deployed to a Raspberry Pi. Any dependencies discovered due to this choice of platform shall be noted for future development teams.

Due to the licensing nature of the Mongoose web server, the custom code generated by Ouroboros that links against Mongoose, including the custom C++ API, must be licensed under the GNU GPL v2 license. Third party code linking against the generated C++ API must also be licensed under the GNU GPL v2 license, since the generated C++ API is licensed under the

aforementioned license. The actual code for the Ouroboros's code generation is not limited by this constraint.

3. System Features

Use Case List

Use Cases
OURO_UC_1. User provides configuration file for parser
OURO_UC_2. User generates code for web server
OURO_UC_3. User accesses system component through web API
OURO_UC_4. User accesses system component through local API
OURO_UC_5. A web callback function is triggered automatically under certain conditions
OURO_UC_6. User controls local hardware component with system
OURO_UC_7. Third party application is notified by REST under certain conditions
OURO_UC_8. User adds functionality to the system during run time through the web API

See external Use Cases document for actual use cases with their detailed descriptions.

4. External Interface Requirements

4.1. User Interfaces

For generating the code, there will be a basic interface for providing in an XML configuration file to the code generator to start the process of creating the custom server code. The custom code will compile to a server that exposes a REST API and a C++ API for Third Party Developers. In addition to the software interfaces, the compiled server also provides a web interface (HTML and CSS) allowing a Prototyping Developer to modify the state of the server.

4.2. Hardware Interfaces

Embedded Linux devices will run the server binary compiled from the server code generated by Ouroboros. If the device is on, connected to the Internet, and the compiled server binary is running, any user wanting to check or manipulate the state of the device will need the server address and a device connected to the Internet with web browsing capabilities to do so.

4.3. Software Interfaces

Ouroboros will generate three different software interfaces that will each serve its own purpose.

C++ API:

- Allow for third party developers to hook hardware into the mongoose server

REST API:

- Allows for third party developers to make calls directly to hardware on the device through the mongoose server

Web pages

- Served via HTTP/HTTPS to any connecting user. Allows for the analysis and modification of the system state.

4.4. Communications Interfaces

Accessing the compiled mongoose server remotely will be done via HTTP/HTTPS, either through the provided web interface, or through a REST API that Ouroboros will generate from the configuration file.

The system will require a web browser to access the website to use the basic features of the site. As the system requires a web front HTTP/HTTPS will be necessary.

5. Other Nonfunctional Requirements

5.1. Performance Requirements

There are no specific performance requirements, other than the code generation platform should finish within a reasonable amount of time for the amount of artifacts being generated (approximately, no more than two or three seconds per artifact generated), and that the compiled server should be responsive to requests (effectively, take no more than one second to respond to requests).

5.2. Safety Requirements

There are no specific safety requirements for this system, other than alerting users of the system of a critical failures.

5.3. Security Requirements

The configuration file will provide a way to specify HTTP/HTTPS for accessing the server. Also, the configuration file will provide a way for parameter values to be validated before the generation of the server custom code. The are no specific security requirements for the generation platform of the system. It is the responsibility of the Prototyping Developer to make sure the configuration XML describes a secure server.

5.4. Software Quality Attributes

The system will be designed to be reusable and interchangeable, especially by allowing for the

extensibility of code generation templates used. The system will also be interoperable, the code generation platform being able to be run from any platform having a Ruby interpreter, and also having the custom server be modifiable via a web interface and a REST API, both of which are operating system agnostic. The system will also be testable, allowing for the verification of proper code generation, as well as for testing of the produced server binary. Finally, the system, particularly the compile server binary, shall be usable by providing multiple ways to interface with it, specifically C++ and REST APIs, as well as a Web UI, allowing for varying levels of ease of use and scriptability.

6. Other Requirements

There are no other requirements for the system, outside of the ones mentioned in previous sections.

Appendix A: Glossary

Ouroboros	Prototyping system for developing customized Mongoose web servers based off XML configuration files. System consists of the code generation platform, and the binaries generated from compiling the generated code.
Web UI	Web pages served by the custom Mongoose server exposing a graphical user interface that users can use to alter the state of the running server.
C++ API	Application programming interface exposed by the compiled custom Mongoose web server, allowing access to the state specified by the custom XML configuration file used to generate the custom Mongoose web server.
REST API	Representative State Transfer application programming interface exposed by the custom Mongoose web server, allowing access to the state specified by the custom XML configuration file used to generate the custom Mongoose web server.

Appendix B: Analysis Models

See the Use Cases document for full description of use case, associated stereotype diagrams and sequence diagrams.

Appendix C: Issues List

- There are license issues regarding the code that need to be looked at. Specifically, is the GPL v2 really a problem with regards to third party C++ tools linking with the exposed API, if the C++ API is made from code linked with the Mongoose code, which is GPL v2?
- Use Cases document is not yet prepared, so Appendix B is not accurate yet.