Vision and Scope Document

for

Ouroboros

Version 1.1 approved

Prepared by Team Cobra

Senior Project 2014-2015

16 September, 2014

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Cobra	9/11/14	Creation of document (draft)	0.1
Cobra	9/16/14	Group revision of document	0.3
Cobra	9/18/14	Initial Feedback Incorporated	1.0
Gabe/Andy	10/28/14	Updated scope of initial release	1.1

1. Business Requirements

1.1. Background

The current state of embedded system solution prototyping at Harris requires for software developers to manually engineer their own prototypes, without much support for automation. In the case of Harris, which creates many different implementations of embedded devices, a lot of time is spent by developers essentially re-writing similar systems. The solution will provide a platform from which developers can configure an embedded system server for their needs, allowing for them to spend more time testing the embedded devices and improving other, external software related to these devices.

1.2. Business Opportunity

Separate software products and components exist that can be integrated in order to form a basic server that can be run from an embedded device. Integrating these different components would normally require manual intervention from developers in order to form a functional prototype, taking time away from other facets of the development process, such as testing the hardware of the embedded devices in question and building and testing software to drive the specific hardware for each embedded device. By providing an open-source platform to automatically generate customized base servers for quick prototyping, Ouroboros would allow engineers and developers to spend more time designing and testing their products and spend less time re-engineering code to configure the prototypes.

1.3. Business Objectives and Success Criteria

The major business objectives for this application revolve around providing a platform for the fast configuration of prototyping servers for embedded devices. This implies that the platform should objectively be able to cut down the time engineers spend setting up the base configuration for the embedded device.

Success for the project will be measured on three factors: deliverables, quality of work, and deployment.

Deliverables pertains to the requirements listed in the Software Requirements Specification document. Requirements are expected to be completed for their assigned release dates. The project teams aims for 100% compliance with release requirements for each software release. Success will be measured on how well the project team complies with this standard.

Quality refers to the overall system complying with the standards for software development put forth by both RIT and Harris. It is the end goal for this project to be deployed at Harris for use in their development process. A high quality implementation with good design should allow for this system to be useful to Harris with minimal updates years into the future. Success will be measured by the project sponsor’s opinion of the system and performance of the team’s work at the completion of releases.

Deployment will refer to the smoothness of the transition of project knowledge and artifacts to

the project sponsor and Harris at the completion of the project. It will also refer to the level of documentation present in the completed system. Success will be measured on the quality of documentation present as well as the sponsor's opinion on the code transition.

1.4. Customer or Market Needs

The basic user of the platform would be a developer tasked with setting up a prototype server for an embedded device in order to allow for testing of the hardware. The developer would simply need to provide a configuration file with parameters describing the state of the device, along with some custom functions, and the platform would then generate a custom server that can, with some minor external additions (for example, 3rd party functions and modules, as well as functions that correspond to the ones described in the configuration file), be run on the embedded device. The goal is that the process of generating the server be as automatic as possible. The generation platform should also be flexible enough that it could be run in cross-platform settings.

1.5. Business Risks

One of the risks involved with the platform is that developers may feel uneasy about having to write configuration files for the platform manually, especially if the embedded device supports a large quantity of states. One way this risk could be mitigated is by designing the configuration file to follow a known document standard to specify configuration, such as XML. This way, developers would be free to write up their own scripts to automate the creation of their configuration files.

What other risks are there?

- Base server being too large
- Base server being able to run in all embedded devices Harris needs (Mongoose seems to need an OS)

Abstract nature of platform could drive developers away. Good documentation would help minimize this risk.

2. Vision of the Solution

2.1. Vision Statement

Ouroboros will provide a platform to customize the Mongoose server to expose functionality specific for the embedded device for which a configuration file given to the platform was written. The generation part of Ouroboros will be cross-platform, and the generated server software package should be runnable. The base business motivation of this system is to increase engineer efficiency by automating what is currently a manual process.

2.2. Major Features

1. Ouroboros will require users to provide it a configuration file describing the parameters and external functions the embedded server will provide.
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 an embedded server, also generate a Web UI using HTML and CSS that can be accessed via a web browser in order to manage the configured parameters in the embedded device.
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. Assumptions and Dependencies

Ouroboros will be using the embedded version of the Mongoose web server. This limits the operating environment of the resulting binary to embedded devices that can run operating systems supported by Mongoose. The resulting binary server generated by the platform can be accessed via a web interface, but a connection to the Internet is not required, since other code interfacing with the C API of the generated server could drive changes in the device. In order to aid in diagnostics, though, a LAN connection to the device would be required in order to, at the very least, view the Web UI exposed by the server in a web browser.

The code generation component of the system may be written in Ruby.

3. Scope and Limitations

3.1. Scope of Initial Release

The initial release will take place --- of December. The following aspects of the system will be release at this time:

1. XML Schema defined for configuration file.
2. Basic code generation tool that supports reading in an XML description file that conforms to the specified XML Schema. The code generation tool will support:
    a. Creating code for a simple custom Mongoose server for an embedded device
    b. Generate code to provide REST API so that users can have limited interact with the device
    c. Generate code to provide HTML pages that the mongoose server will serve

3.2. Scope of Subsequent Releases

The second and final release will take place on May 2015.

1. Code generation tool will be used to:
    a. Interact with and connect with the Mongoose server on an embedded device
    b. Generate code to expose a C++ API for developers to easily hook components of the device with the Mongoose server.
    c. Generate code to allow for custom functions to be handle through requests
    d. Allow for callback to be accessed via both the C++ and REST APIs.
2. Create and provide access to CSS files to style the generated HTML pages.
3. The code generation tool will provide input validation for users so that users will not be able to send bad data as parameters to the device
4. The code generation tool will auto-generate documentation for the code that it also generated

3.3. Limitations and Exclusions

The system will not supply a utility to generate configuration files. In addition, it is up to the developer creating the configuration file given to the platform to also specify what is to be executed by the system when one of the custom functions in the configuration file is called.

4. Business Context

4.1. Stakeholder Profiles

Stakeholder	Major Value	Attitudes	Major Interests	Constraints
Project Sponsor (Harris)	Wants to optimize the time its engineers spend prototyping.	Optimistic about how Ouroboros could reduce time doing manual programming of servers, and looking forward to using system to further automate other processes.	Fast prototyping in order to test hardware radios faster, and thus reduce development costs and reduce time to market.	
Software Developers (at Harris)	Actual engineers prototyping Harris hardware.	(What would this be?)	(Not sure, since we haven't talked to any developers other than Nate… could we base this off Nate?)	
Rochester Institute of Technology				

Us				

4.2. Operating Environment

The main users of this product will originally be centered at Harris, but since this is to be an Open Source project, product usage may spread world-wide. The generation component of the project must be cross-platform in order to allow for any developer with a suitable basic setup to generate the server C/C++ code required for the compilation of the binary web server.