Rasler, Habegger, Parker

**EARL (Egg Alert and Real-Time Logistics)**

12/14/2011 Version 1.0

Software Design Document

**TABLE OF CONTENTS**

**1. INTRODUCTION 3**

1.1 Purpose 3

1.2 Scope 3

1.3 Overview 3

1.4 Reference Material 3

1.5 Definitions and Acronyms 3

**2. SYSTEM OVERVIEW 4**

**3. SYSTEM ARCHITECTURE 4**

3.1 Architectural Design 4

3.2 Decomposition Description 6

3.3 Design Rationale 6

**4. DATA DESIGN 3**

4.1 Data Description 3

4.2 Data Dictionary 3

**5. COMPONENT DESIGN 3**

**6. HUMAN INTERFACE DESIGN 4**

6.1 Overview of User Interface 4

6.2 Screen Images 4

6.3 Screen Objects and Actions 4

**7. REQUIREMENTS MATRIX 4**

**8. APPENDICES 4**

**1. INTRODUCTION**

**1.1 Purpose**

This software design document describes the architecture and design of the Egg Alert and Real-time Logistics (EARL) System. It is intended to provide an overview of its inherent structure and the connectivity of its software elements, to be used as a guide for development.

**1.2 Scope**

The goal of this project is the creation of a system that automates the process of determining when and where a chicken egg flow problem (egg jam) occurs on a system of conveyors through the chicken egg packaging process. In the case of abnormal flow, the system will alert the user in real-time as to which specific line the problem has occurred on. In a typical poultry operation, thousands of feet of conveyor lines would need to be searched manually to locate a jam. This system would minimize the searching, thus reducing the labor cost needed to fix the problem. The system also eliminates unnecessary loss in performance by alerting users even when the packaging system is not in use.

**1.3 Overview**

Section 1 (Introduction) provides a short description of the document and identifies information useful for understanding the other sections. Section 2 (System Overview) provides a description of the EARL system. Section 3 (System Architecture) explains the architecture of the EARL system and breaks down the individual subsystems. Section 4 (Data Design) describes the organization of data structures used in the project. Section 5 (Component Design) further details the individual components of each subsystem mentioned in Section 3. Section 6 (Human Interface Design) provides an overview of the software’s graphical user interface (GUI). Section 7 (Requirements Matrix) shows the FR-DP matrix and its translation into software components. Section 8 (Appendices) includes any other information we have deemed necessary.

**1.4 Reference Material**

The SDD makes use of information from the following documents:

EARL Software Requirements Specification

EARL Project Management Plan

EARL System Architecture

EARL Functional Requirements Document

EARL UML Architecture

**1.5 Definitions and Acronyms**

EARL: Egg Alert and Real-time Logistics

UART: Universal Asynchronous Receiver/Transmitter

BS2P40: BASIC Stamp 2p 40-Pin Microcontroller Module

CAT-5: Serial Twisted Pair Category-5 RJ45 Style Cabling

GUI: Graphical User Interface

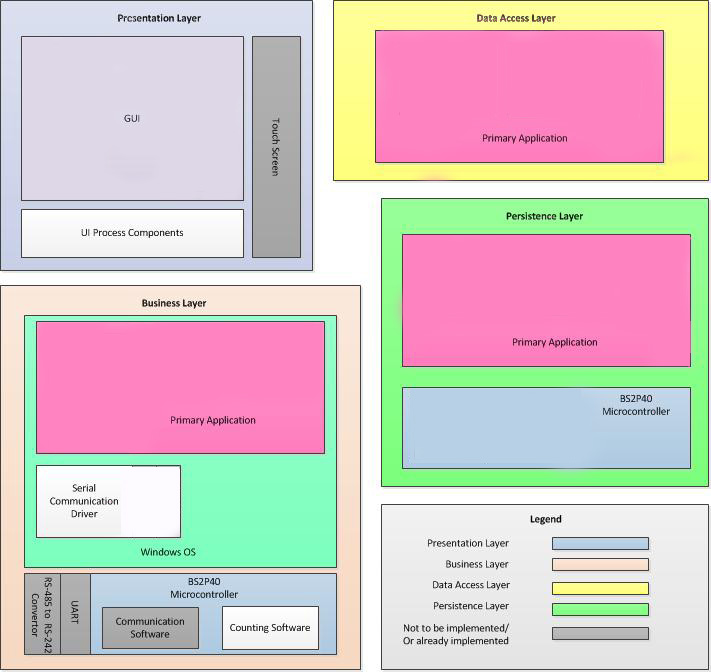
**2. SYSTEM OVERVIEW**

The EARL system incorporates physical hardware and software components. Several sensor units consisting of armatures and magnetic triggers are installed onto the egg conveyor system. As eggs pass by, they raise the armatures and activate the trigger. Each activation is recorded by the BS2P40 associated with the sensor unit (one per ten armatures), which is passed on to a connected UART.

The main program polls each UART to determine which sensor units are active. When a sensor unit remains inactive for a period of time, the main program judges whether a jam has developed. If so, it activates a visual alert on the GUI and an audio alert either through a local speaker system or via a remote transmission (to a distant stereo system or possibly a cell phone). It will display the assumed location of the jam on the GUI and record instance data in the log. The alarms will continue until suspended by an operator, after which the alert system will remain dormant until it is reset.

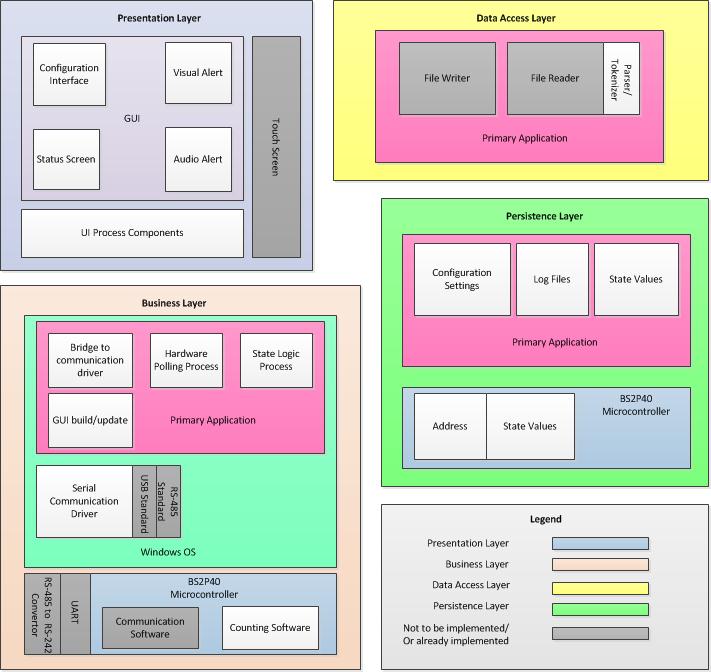
**3. SYSTEM ARCHITECTURE**

**3.1 Architectural Design**



* Presentation Layer:
  + GUI: Provides user interaction to the primary application via touch screen. The GUI displays alerts and allows the user to suspend or reset them. It also allows the user to set configuration options (alert volume and style, etc.) and to call up log files, which are displayed on the touch screen
* Data Access Layer:
  + File Access (Primary Application): Relays data between the primary application and the hard disk, reading and writing log files and configuration values.
* Business Layer:
  + Primary Application: Polls each UART to get the buffered data from the BASIC Stamps. It then interprets the data to determine if a jam has occurred. In the event of a jam, the primary application calls for the alert and tells the GUI to display it. The primary application also suspends and resets the alarm at the command of the user, relays through the GUI.
  + Serial Communication Driver: Manages I/O between the primary application and the UARTs through the serial port.
  + Microcontroller Logic: Maintains a count of eggs that pass through the sensors and relays sensor data to the UART, which buffers the data until polled by the main application.
* Persistence Layer:
  + Configuration: Records user settings (alert volume, style, and methods of transmission) for future start-ups.
  + Log Files: Records data regarding jams.
  + Microcontroller Configuration: Stores its own serial address and the current state of its sensor array.

**3.2 Decomposition Description**



**3.3 Design Rationale**

We decided to adopt a layered architecture because we felt this best represented and separated the functions of our software. The GUI incorporates touchscreen capability because one is already installed at the location where the software will be used. The serial communication driver accommodates RS-485 due to its ability to transmit signals across thousands of feet without requiring additional hardware – a necessary feature for covering the entire conveyor system without unnecessary additional costs.

**4. DATA DESIGN**

**4.1 Data Description**

The information domain of the system is transformed into data structures using outputs to modulate serial lines. The data is transmitted though serial communication, buffered using a universal asynchronous receiver/transmitter (UART), and organized using a state machine. Local data storage items include an industrial I2C serial EEPROM that stores firmware. The BS2P40’s EEPROM size is 16 kilobytes. It stores approximately 4,000 PBASIC instructions.

**4.2 Data Dictionary**

**4.2.1 Host**

* mStandardFile – file
* mStandardPort – serial port
* states – integer array

**4.2.2 BS2P40**

* Address – unsigned character
* States – integer array

**5. COMPONENT DESIGN**

In this section, we take a closer look at what each component does in a more systematic way. Below is a summary of each object member function for all the objects listed in 3.2 in pseudocode.

**5.1 Presentation Layer**

This layer consists of components designed to interact with the user. The configuration interface behaves as follows:

For-each *configuration setting* in *configuration interface*

* If (*configuration interface is visible*) draw *configuration setting’s* *UI component*
* If (*configuration setting changes*)
  + Record changes to settings file
  + Change related program behavior
  + Refresh the GUI

The visual alert behaves as follows:

If (*there is an egg jam* AND *status screen is not displayed*) Display the Status Screen

The status screen behaves as follows:

If (*communication failure*) Display communication failure graphic

Else For-each *collector* in *building*

* If (collector is disabled) Display disabled collector graphic
* Else For-each *lane* in the *collector*
  + If (*lane is jammed*) Display *lane* as jammed graphic
  + Else If (*lane is disabled*) Display *lane* as disabled graphic
  + Else Display *lane* as operable graphic
  + If (*lane* *is clicked*) Display the configuration interface for *lane*
* If (*collector* *is clicked*) Display the configuration interface for *collector*

The UI Process Components behaves as follows:

Asynchronously update the GUI

**5.2 Business Layer**

This layer consists of communication components. The bridge to communication driver behaves as follows:

Asynchronously loop until program ends:

* If (*port* *isn’t open*)
  + Open the port
  + Throw faulty port exception
* Else For-each *string* in *serial buffer*
  + Send string to State Logic Process

The hardware polling process behaves as follows:

Asynchronously loop until program ends:

* For-each *module* in *module collection*
  + request *module* status
  + Listen for response
  + If (*no response*)
    - If (*two seconds have passed*) Report *module* as faulty
    - Else try again
  + Else update *module* status

The state logic process behaves as follows:

If (*input string* *is a status response*) Assign parsed status to last queried module

Else Dismiss *input string* as random chatter

The GUI build/update behaves as follows:

Set configuration settings from serialized data

Update GUI

The serial communication driver behaves as follows:

If (*new* *out-going query string*)

* Format and transmit *query string*
* Log transmissions

The counting software behaves as follows:

For-each *sensor* in *valid sensors*

* If (*sensor status changed*) Increment *sensor’s lane* count

**5.3 Data Access Layer**

This layer consists of raw file I/O components. The parser/tokenizer behaves as follows:

For-each *line* in the *file*

* If (*line starts with* ‘[‘ *and ends with* ‘]’) Center substring of *line* is a section title
* Else If (*line starts with ‘;’*) *line* is a comment
* Else If (*line contains ‘=’*)
  + Substring of *line* up to the ‘=’ is the key name
  + Substring of *line* beyond the ‘=’ is the key value

**5.4 Persistence Layer**

This layer consists of formatted file I/O components. The configuration settings behave as follows:

Load *all* *configuration settings* at program start

For-each *configuration setting changed event*

* Serialize *configuration setting* in settings.ini

The log files behave as follows:

For-each *error* *event*

* Serialize *error* in error.txt

The state values behave as follows:

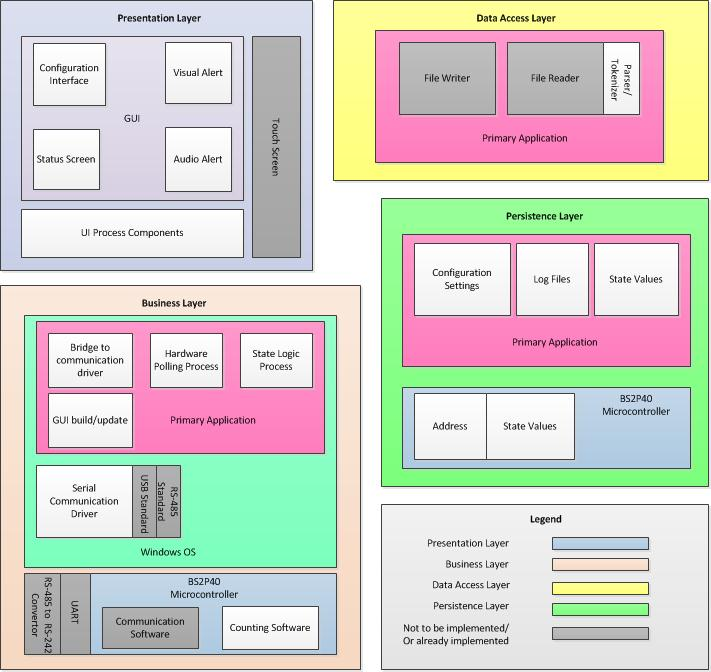
Initialize space for module states

Report module state changes to log file

5.5 Architectures

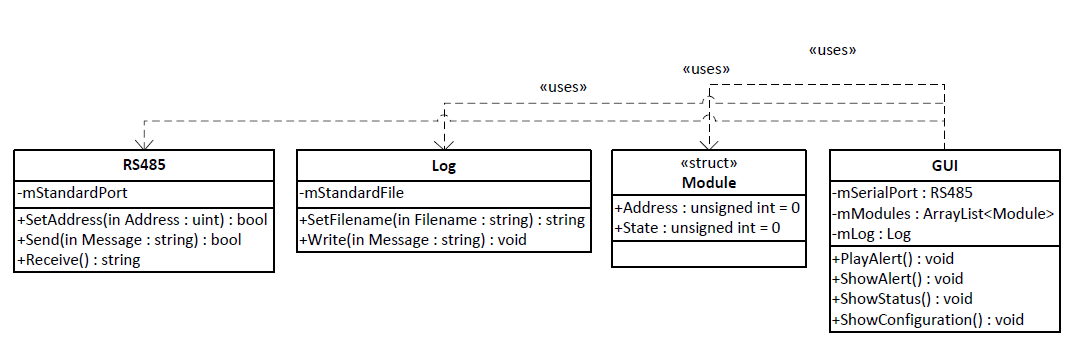
5.5.1

**Application Architecture**

****

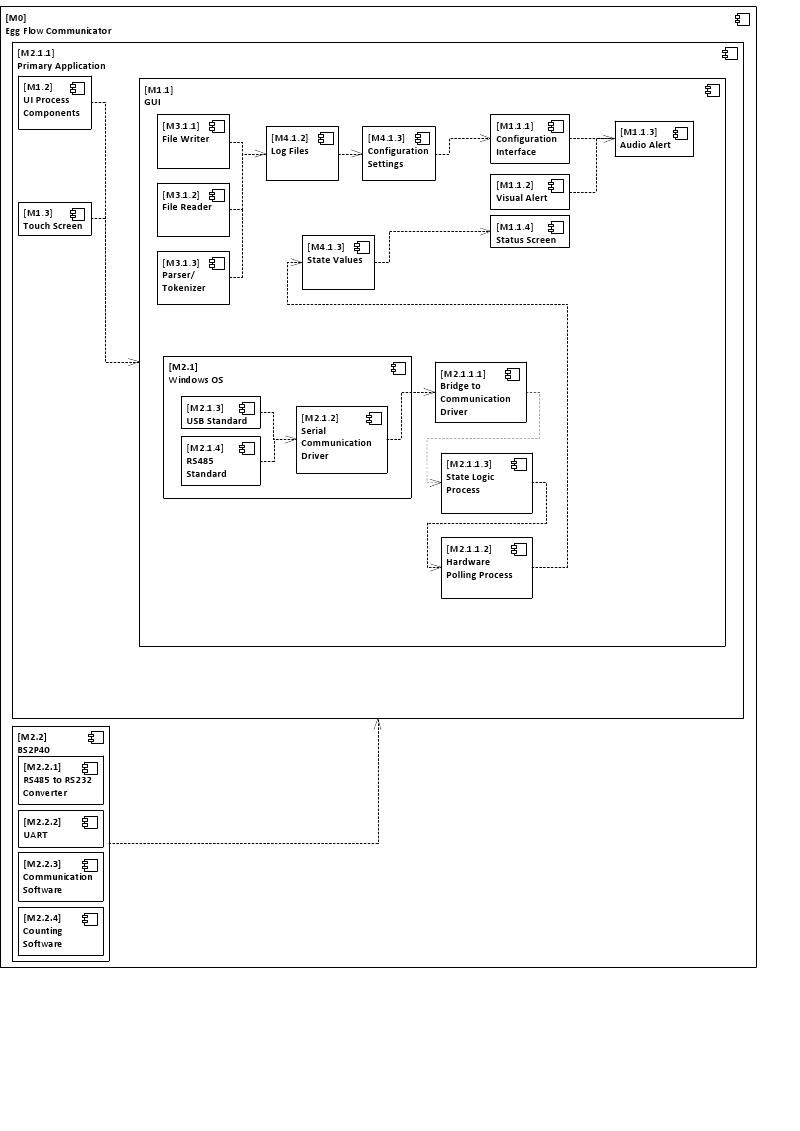
**5.5.2**

**UML Class Diagram**



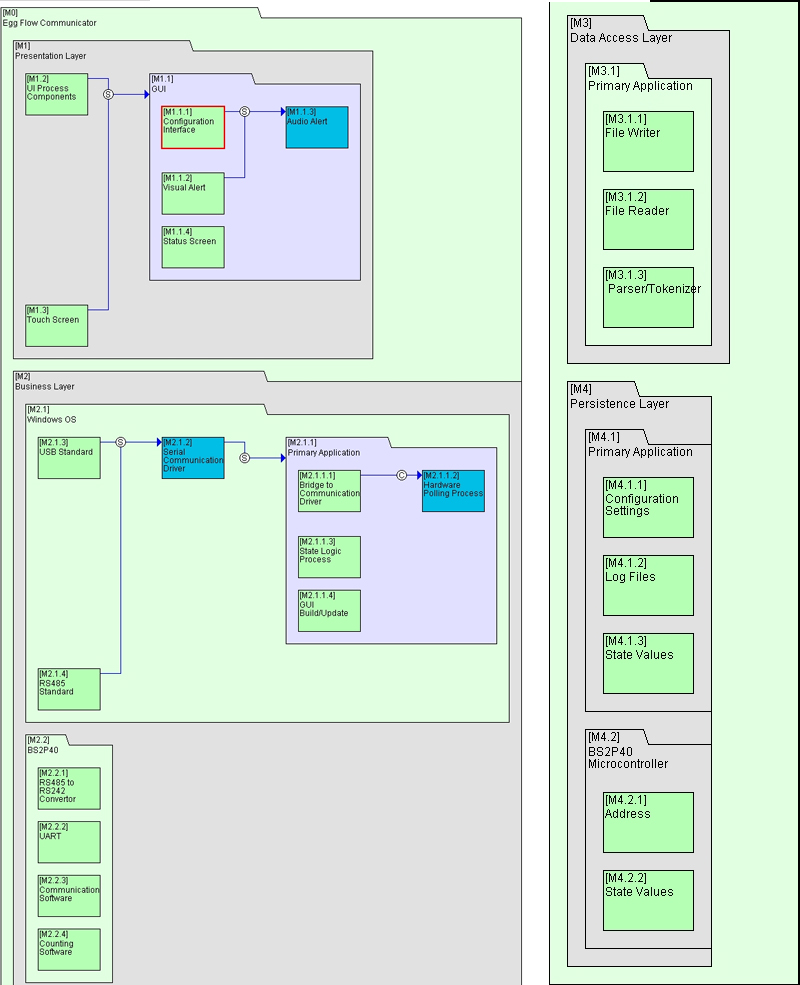
**5.5.3**

**UML Component Diagram**



**5.5.4**

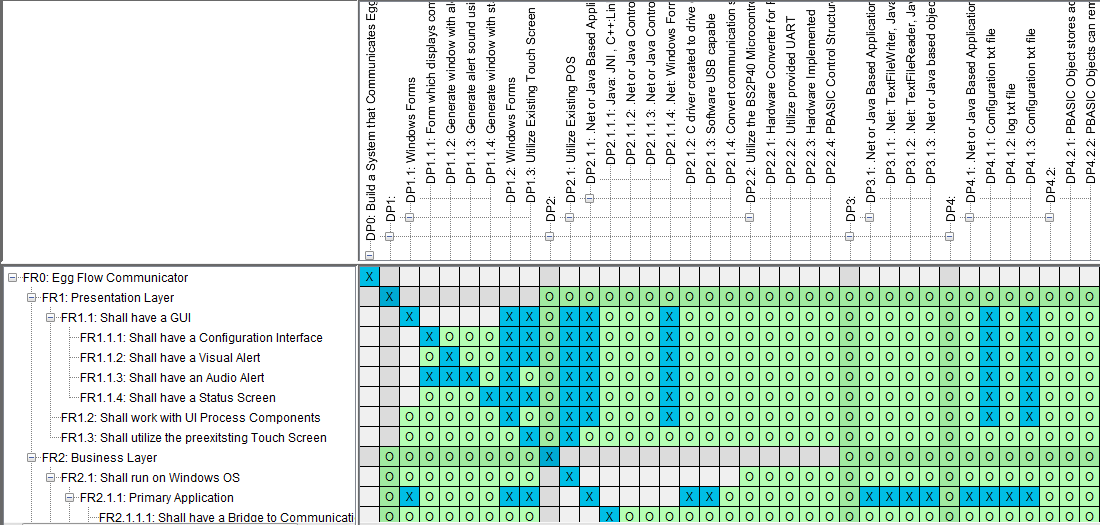
**System Architecture**

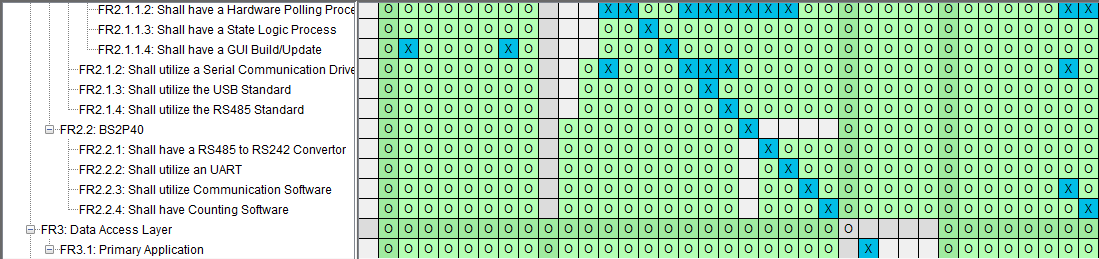
****

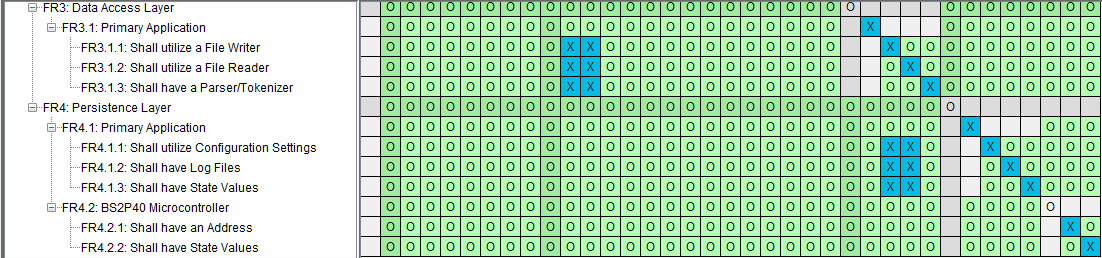
**5.5.5**

**Information Architecture**

**Design Matrix**

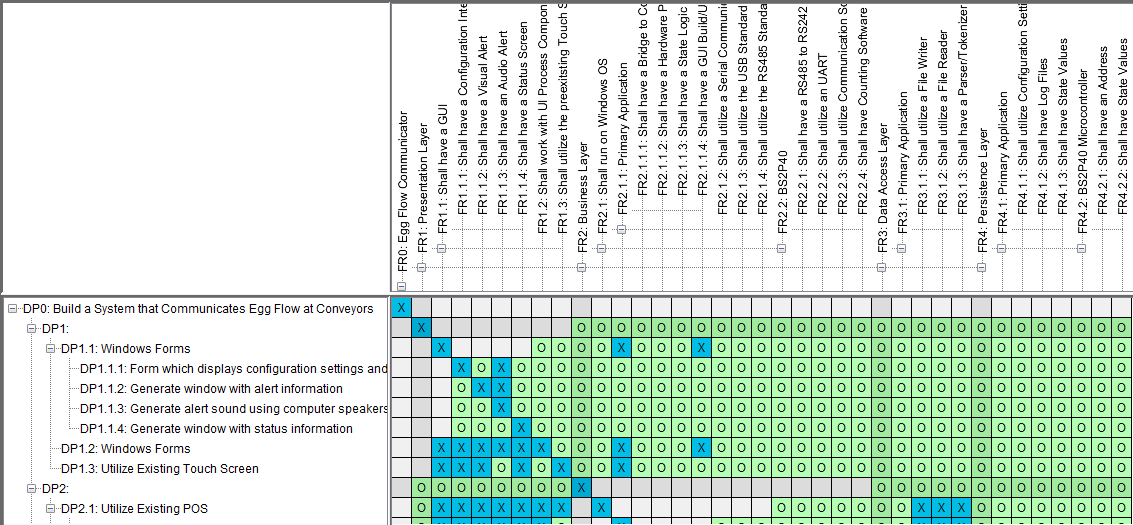


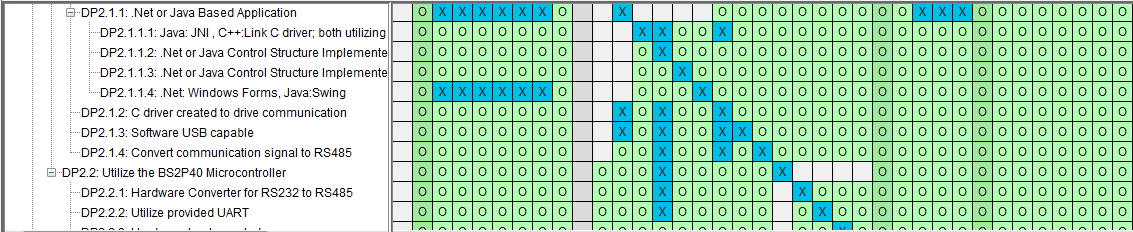


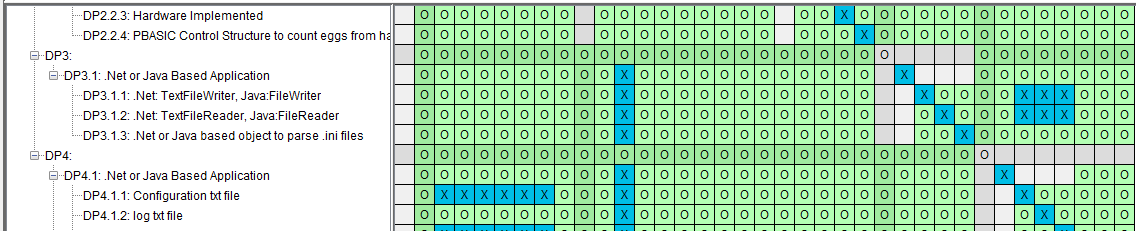


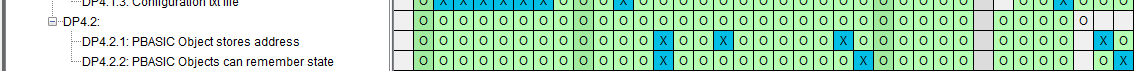
**Information Architecture**

**DSM**









**6. HUMAN INTERFACE DESIGN**

**6.1 Overview of User Interface**

The main application will contain an informative status feature that can be checked at will and during normal conditions (no egg jams) this application should be minimizable. During normal conditions there should be some informative feature that graphically displays the current counting status of the modules (a line chart,…) along with a textual display of the system’s status.

There should be drop down menus under the title bar for adjusting parameters. These parameters should include:

Audio alert mode on/off

Volume adjuster

Tolerance of system

Main chart adjustments:

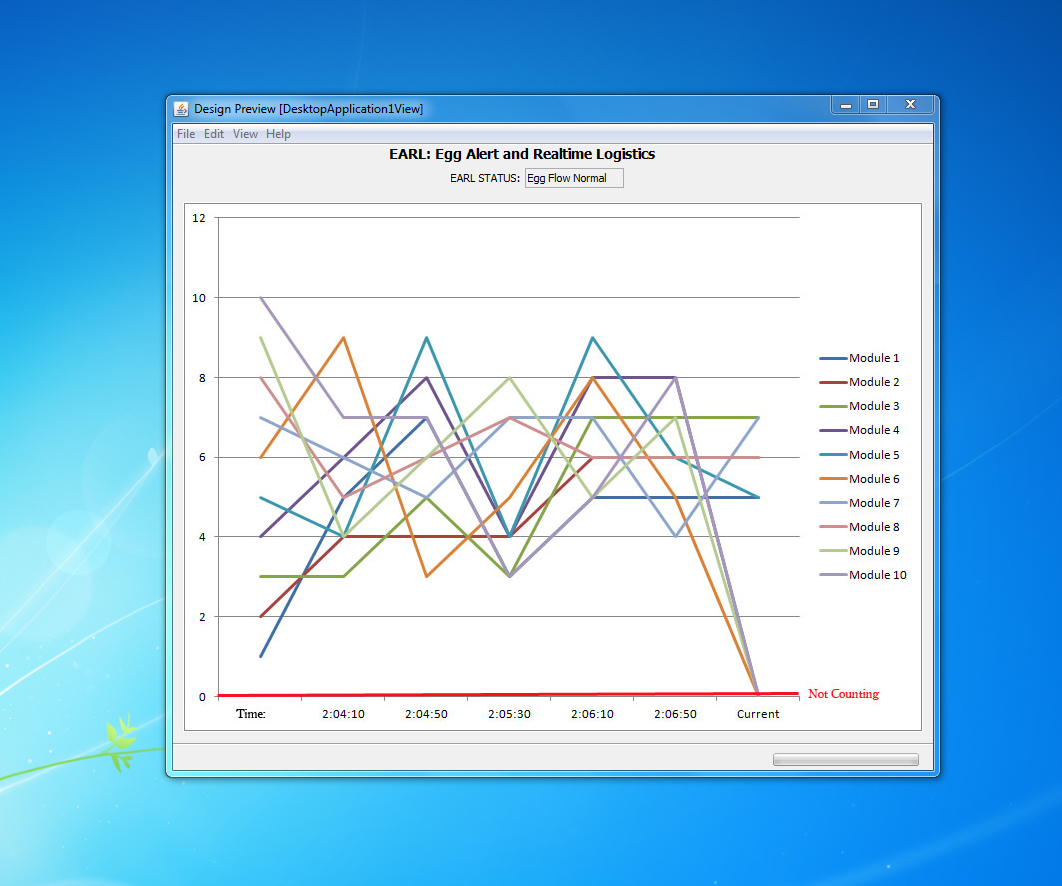
Interval frequency

Modules displayed

Under alert conditions this application should take focus and be maximized at the top of all other application, with a noticeable red alert screen, possibly flashing, with a highly visual cue as to which module is detecting a jam or other alert situations. The status text should be changed to reflect this.

**6.2 Screen Images**

The main user interface in normal conditions mode:



Menu Bar options:





Status condition label:



**6.3 Screen Objects and Actions**

The primary application in non-alert mode is depicted above with status displayed and modules 1-10 showing counting amounts between time intervals associated. The menu options include a preferences option, alert history option, and a flow history option.

The alert history option should populate another window with historical alert data. The flow history option should populate another window with historical data of module counts.

In Alert mode the EARL STATUS should be indicated as Egg Flow Alert, or other Alert modes (module, sensor failures,…). The module label associated with the alert should flash in red or in some highly visible fashion.

**8. APPENDICES**

*Not Included.*