Avera Visibility

System Design Overview

Components:

- **Avera Engine** – communicates with tags, controls hardware, and records data in database
- **Web Engine** – manages tag/asset information, incorporates Capability based security scheme
- **Configuration Manager** – Configures hardware using friendly graphical interface
- **Watchdog** – remote system monitor
- **Database Schema** – database tables used by Avera
- **Remote Console** – Remote on site monitor
- **Blades** – application specific extensions

A complete On-Site Avera system is first broken down into operating sub groups called subsystems. Each subsystem, or controller, has **Logical Device Controllers** (LDC), **Finite State Machines** (FSM), and a location code which provide the functionality required for that group.

Each component of a subsystem communicates via the message queue. The messages passed by the message queue contain the location code and/or the device id of the state machine or logical device controller that dispatched the message.

Logical Device Controllers

The logical device controllers are the interface between Avera and the hardware components. Each logical device is configured in the system via the Devices table in the database. Each LDC also has a set of parameters that modify its behavior.
Due to the underlying control structures, it is possible and even likely that a single physical device will have multiple logical devices associated with it. For example, the IDmicro 4120 interrogator can be partitioned into several zones, each with a unique identity. Also, the solenoids that the interrogator provides are controlled through a separate logical device.

These are the currently supported LDC's:

Detector – IDmicro Microbadge
Detector – IDmicro Microstamp
Detector – RFCode
Detector – Micro controller
Detector – SCS
Detector – Tagmaster
Detector – Intellitag
Display – Digimatics
Fuel – Durant
Fuel – Liquid Controls
Fuel – Gilbarco
Remote Console – Secure
Remote Console – Unsecure
Scale – Rice Lake
Solenoid – Tagmaster
Solenoid – Microstamp
Solenoid – Micro controller
Watchdog – Secure
Watchdog – Unsecure

Finite State Machines

Each finite state machine accepts messages from the message queue. These messages may have been dispatched by other state machines and by logical device controllers. In general, a state machine will only accept messages from the message queue that are dispatched to that state machine's assigned location or from one of that state machine's assigned devices. Each state machine may have up to four devices associated with it. Each state machine also has additional settings which modify its operation.

These are the currently supported state machines:

Actuator
Actuator Pair
Actuator Schedule
Authorizer
Containerizer
Detector
Display
Fuel PAM 1000
Fuel Commercial
Remote Console
Watchdog
Weigh

Remote Console

The remote console component interfaces to the On-Site component via the On-Site remote console socket. Upon connection to the On-Site system, it transmits its configuration information.

Watchdog

The Watchdog process records events in the system that are reported to it and takes actions based on the status of these events. It also maintains a table describing when these reported events last occurred. Some examples of events may be "Gate-3-Opened", "Island -2-heartbeat-Occurred", "Fuel-Pump-Disabled", etc.

The Watchdog can generate two classes of alarms – passive and active. An alarm that is active occurs if the Watchdog is directly told that the alarm event has occurred. An example of an active alarm may be "Fuel pump 3 has no power". An alarm is passive if the Watchdog decides that something is wrong because it has failed to hear otherwise in some defined period of time. An example of a passive alarm may be "Island 4 has not reported for 200 seconds". Passive alarms are useful because they are self-rectifying, namely, if the periodic reporting resumes, the Watchdog will exit the alarm state. Passive events are also able to be "reported" over a broken communication channel since it is the broken communication that causes the alarm condition in the first place.

Database Schema

The Avera database includes tables that describe the configuration of the system; tables that describe the assets controlled by the system; tables that describe the access privileges that those assets have to locations within the system; tables that record activities, detections, and measurements made by the system; and specialized tables.

Web Engine

A website is used for administration of and reporting on the system. The website allows the user to view, update, and edit data in the database. The user can also accomplish specific tasks such as assigning a tag to an asset, or generating reports about the system.

Blades

Blades are additional pieces of software that may be required by the specifications of a particular system. Avera features a neat and easy way to incorporate such additional software as may be required.

System Operations

1. Startup

On system startup, a database connection is established, the message queue is connected, the state machines and devices are loaded from the database, each LDC is constructed, each FSM is constructed, and a startup message is injected into the system.

The database connection is given on the command line along with the controller (specific PC) identifier of this instance of the engine. An OLEDB connection is made to the database. The system parameters are loaded from the Params table. the LDC configuration is loaded from the Devices table into a vector and each is constructed. The System LDC is constructed. The FSM configuration is loaded from the State Machines table into a vector and each is constructed. The System FSM is constructed. As each FSM is constructed it injects its own Initialize message into the message queue.

2. Main processing thread

The message queue is now full of FSM Initialize messages and the startup message. The message processing loop is entered and this is the main thread's primary task. Each message is sent sequentially to all of the FSM's by calling the 'OnMessage' method. Each FSM responds to messages that interest it, ignoring those that do not. During the processing of messages, additional messages may be injected to the end of the queue.

3. Message Queue

The message queue inserts messages only at the end. If the injected message has a delay associated with it, a thread is created which sleeps for the delay period before injecting the message into the queue, calling each of the FSM's OnMessage method, passing the message, and then removing the message from the top of the queue. If no messages are waiting in the queue, the thread is blocked until a message is injected.

4. LDC Connection and Disconnection

The System FSM periodically receives a delayed Reconnect message. When the system FSM receives this message, it queries each LDC for its state of health. If the state of health is bad, the LDC is disconnected from the system. The System FSM then queries

each LDC for its connection status. If an LDC is not connected, the System FSM attempts to connect it.

During connection an LDC attempts to establish a communication channel with its device. IF it is able to communicate, a polling thread is created and a Device Connected message is injected into the message queue.

During disconnection the LDC terminates the polling thread, closes the communication channel to the device and injects a Device Disconnected message into the message queue.

5. LDC Operation

While an LDC is connected its polling thread is either sampling its device for status and data or waiting for communication from its device. The state of health is maintained by the LDC during this process.

Data is collected from the device and injected as messages into the message queue from processing. The LDC cannot communicate directly with FSM's or other LDC's in the system. It may, however, record activities into the database.

Each different type of LDC may present methods for FSM to control. Methods that perform operations such as releasing a solenoid or enabling a fuel pump. The methods will be called directly by an associated FSM. Note that this method call will always come from a different thread than the polling thread.

6. FSM Operation

As each FSM receives a message, it accepts the message if interested. The message may have originated from an LDC or an FSM, including itself. During processing of the accepted message, data may be recorded into the database, queries may be made from the database, and LDC's may be controlled.

7. Shutdown

To shut the system down a Shutdown message is injected into the message queue. When the System FSM receives the message, it disconnects all of the LDC's. When the message queue is empty, the error and output logs are closed, the database connection is closed, and the main thread is terminated.