Design Document for Autonomous Multi-cycle Farming in Space

Team Members:

Name	**Contact**
Client & Sponsor: Philip Chan PhD	pkc@cs.fit.edu
Christopher Milsap	cmillsap2013@my.fit.edu
Giampiero Corsbie	gcorsbie2018@my.fit.edu

Design Document v1.0
Created: 9/18/2019
Revised: 9/23/19

Design Document Version Control:

Document ID: DD-000001				
Revised Version	**Date**	**Collaborators**	**Version**	**Reason**
N/A	9/23/2019	All	v1.0	Initial creation

Introduction

Purpose

This document describes the design of the AMCFD. It will cover the various modules that monitor and control the system, along with detail that describes how the actuators and sensors interact with each other.

Scope

This document will cover the requirements of the SRS and describes how they will be implemented. It will include details on module functionality, a sample use case, state diagram for describing how the sensor and actuator modules will interact, and details on configuration file syntax.

Context

This project serves as the software subsystem (SS) of the AMCFD. It will be responsible for the control and observation of the physical mechanisms and crop, along with any interaction with the host vessel and crew. To do this, the Decision Maker (DM) will coordinate with the various modules of the SS to automate the entire process.

Implementation Summary

Based on queries to and from the scheduling module and the sensor module, the DM will interpret the state of the farm and manage the appropriate job queues and sensor cycle period lengths. Based on these job queues, the DM will control any actuators necessary to perform a task to manage the environment of the farm. Sensors will activate and deactivate in cycles to conserve power. Additionally, the DM will be manually configurable to change any details of the SS's operation.

Module Functionality

Sensor

- Inheritable abstract parent classes
- Activation and deactivation methods
- Data abstraction from lower level granularity
- Modifiable threshold values
- Methods to access specific sensors via ID
- Read methods to retrieve a stream of data from low level hardware

Driver

- Query methods for state and stage
- Methods for parsing and applying user configured schedules
- Queues for configurable and sequential job execution
- Methods for signaling listeners to transition
- Methods to prioritize events

Interface

- Methods to visually signal the current stage, via lights or a GUI
- System to monitor power draw
- Manual override

Actuator

- Activation methods with magnitude and positional parameters
- Callback functions for job completion
- Methods to access specific sensors via ID

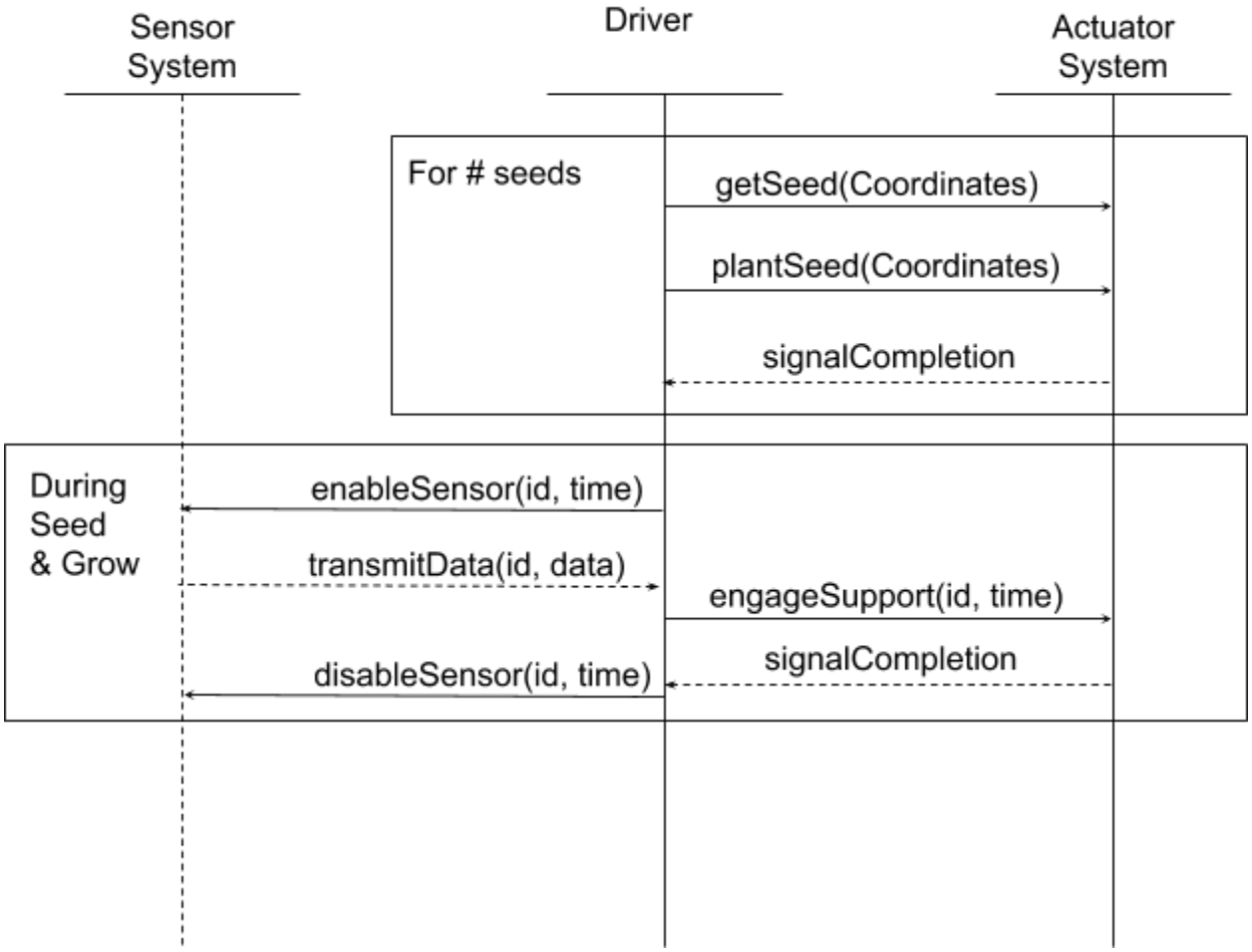
Lighting Array

- Activation and Deactivation methods for individual light fixtures
- Method to set lighting patterns
- Method to set the color of the lights

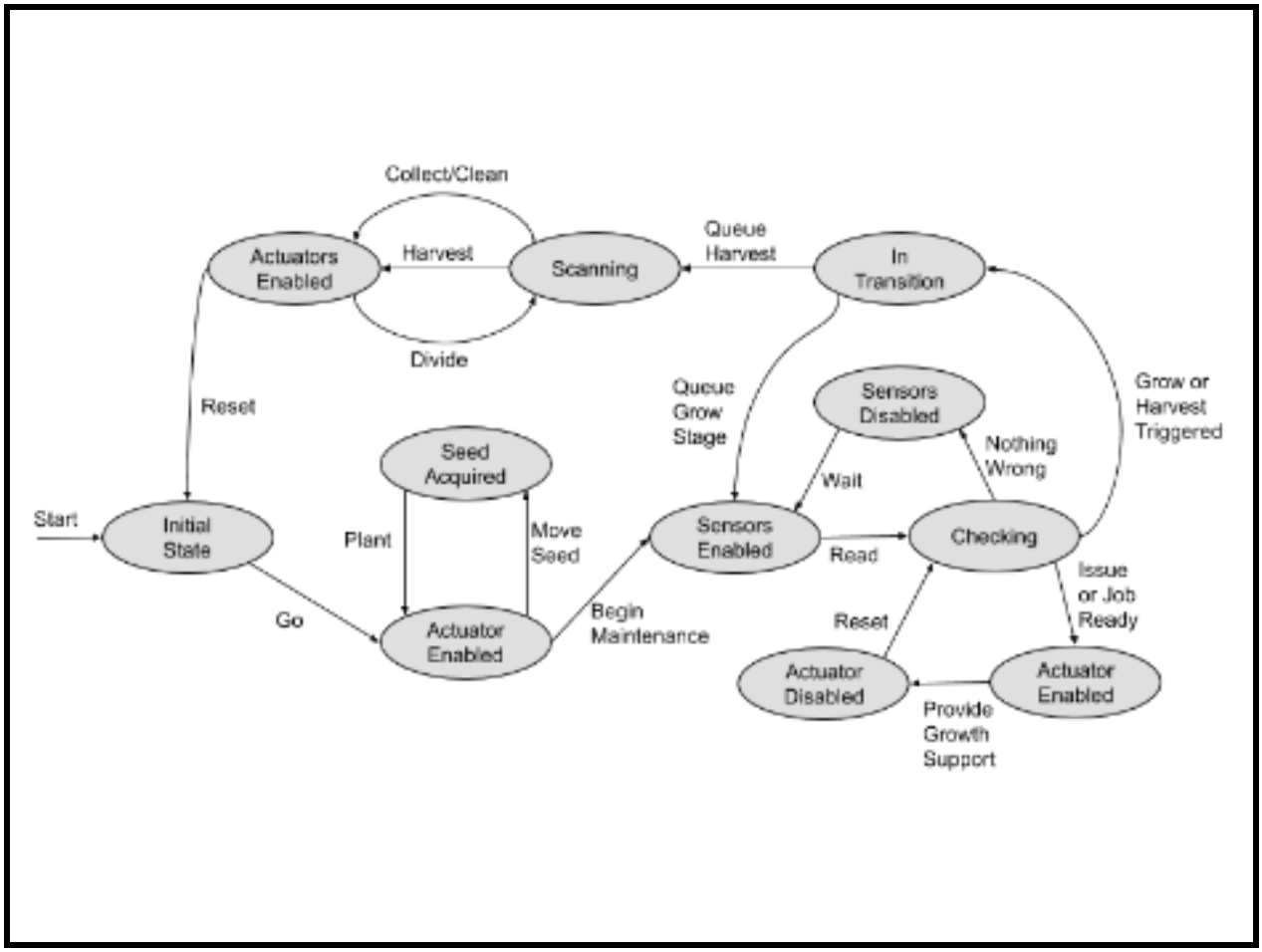
SSD

Use Case Example: Seed and Grow Loops

1. Get seeds
2. Plant seeds
3. Cycle sensor activation
4. Engage support systems

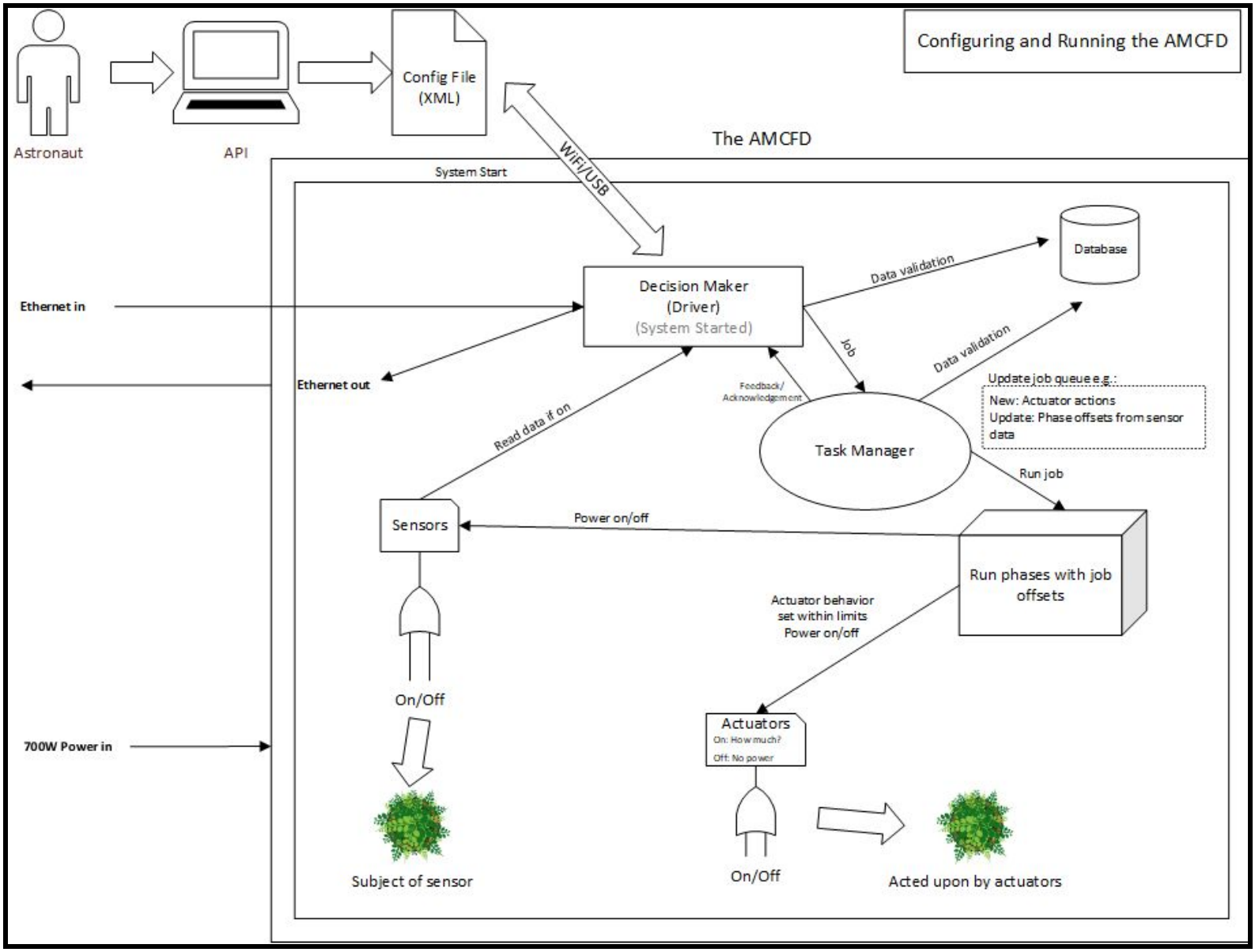


Actuator-Sensor State Diagram



Configuring and Running the AMCFD

- 700w power constraint is subject to change as research continues.
- Config file may or may not use XML
- Ground may have some control of unit TBD



Configuration File Syntax

The configuration file for the AMCFD will follow the following grammar. The SS will use multiple queues for each module/sensor/actuator to store actions and execute them in parallel to each other.

Subqueues will be built for conditionals and these will be loaded when conditions are met. Conflicts will be resolved using constraint satisfaction algorithms. Any actuators and sensors in already in loop will have their loop overwritten if a new one is introduced.

Every transition will engage the “end” subscript. This grammar is not final and will evolve as the system is developed. Lower level definitions will be made available as their respective parts are completed.

<phase-script> ::= <phase-ident> “{“ {step} “}” [<phase-script>]

<step> ::= “step” <step-ident> “:” [“start:” <sequence>]
[“during:” <sequence-loop> { “,” <sequence-loop> }]
[“if:” <conditional-set>]
[“end:” <sequence>] “;”

<sequence> ::= <action> { ”,” <sequence> }

<sequence-loop> ::= “every” <time-const> “(“ <sequence> “)”

<conditional-set> ::= <conditional> [“,” <conditional-set>]

<conditional> ::= (<sensor-type> “is” (<sensor-check> | <sensor-state>)
“then” <sequence> | <actuator-type> “is” (<actuator-check> |
<actuator-state>) “then” (<sequence> | <sequence> <deviation >)

<action> ::= (“activate” (<sensor-type> | <actuator-type> <actuator-params>)
| “deactivate” (<sensor-type> | <actuator-type>)
| “stall” (<sensor-type> | <actuator-type>) “for” <time-const>
| “wait” <time-const>
| “reset” (<sensor-type> | <actuator-type>)

<deviation> ::= (“transition to” <phase-ident> | “go to” <step-ident>)

<actuator-params> ::= “(“ <param> { “,” <param> } “)”

<param> ::= <param-ident> “=” <value>