Rover Engagement Display (RED)

A telemetry and controls graphical user interface

Tyler Morrow

Introduction

Missouri S&T Mars Rover Design Team (~2 years old)

- Completely volunteer
- Funded by donations

Purpose

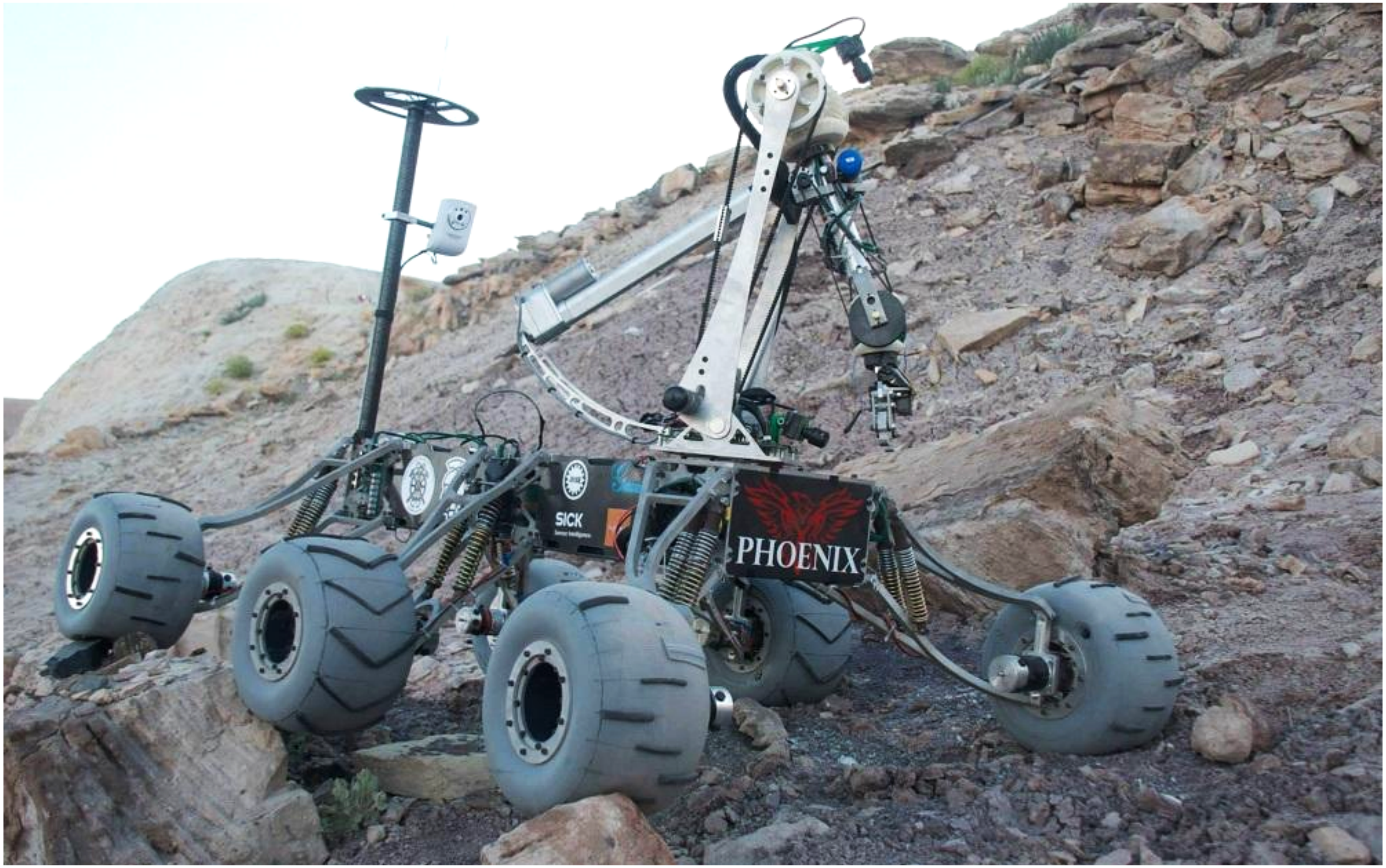
- Compete in the annual University Rover Challenge (URC) held by the Mars Society in Utah.
- Design and build a Mars rover that can perform the tasks set forth by the competition.

Competition Restrictions

- Rover cannot weigh more than 50 kg
- Rover cannot cost more than $15,000

Introduction

Figure 1: Phoenix at URC 2014 in the desert of southern Utah



Requirements

Primary Goal

Control the rover and display telemetry.

This is achieved by meeting the following requirements:

1. **Redundant Network Connection**

   Simultaneously send and receive data on both ends of the connection.

2. **Communication Protocol**

   An agreed upon message format.

3. **Graphical User Interface**

   Displays telemetry and user input coherently and responsively.

4. **Input Retrieval and Translation**

   Converts user input into the appropriate command messages.

Requirements

Solutions for requirements:

1. **Redundant Network Connection**

   Asynchronous TCP Server

2. **Communication Protocol**

   Key->Value pairing adhering to JSON

3. **Graphical User Interface**

   Windows Presentation Foundation (WPF)

4. **Input Retrieval and Interpretation**

   Xbox Controller and DirectX API

All solutions can be implemented with the C# programming language and the .NET framework.

System Overview

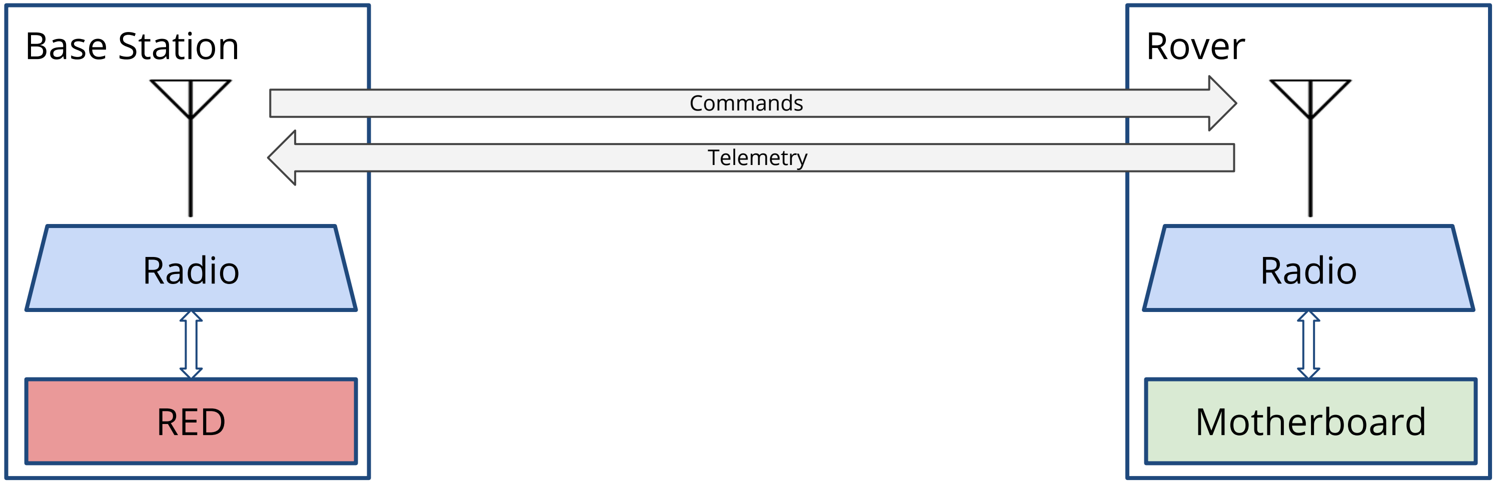
Base Station

The remote location from which the rover is *engaged*. RED exists on the computer located at the base station.

Motherboard

The circuit board onboard the rover with which the base station communicates.

Figure 2: Base Station to Rover Communication Overview



Rover Behavior

Two parallel tasks run continuously:

Telemetry

1. Establish network connection
2. Receive telemetry from each auxiliary system
3. Serialize telemetry as JSON
4. Send serialized data to base station

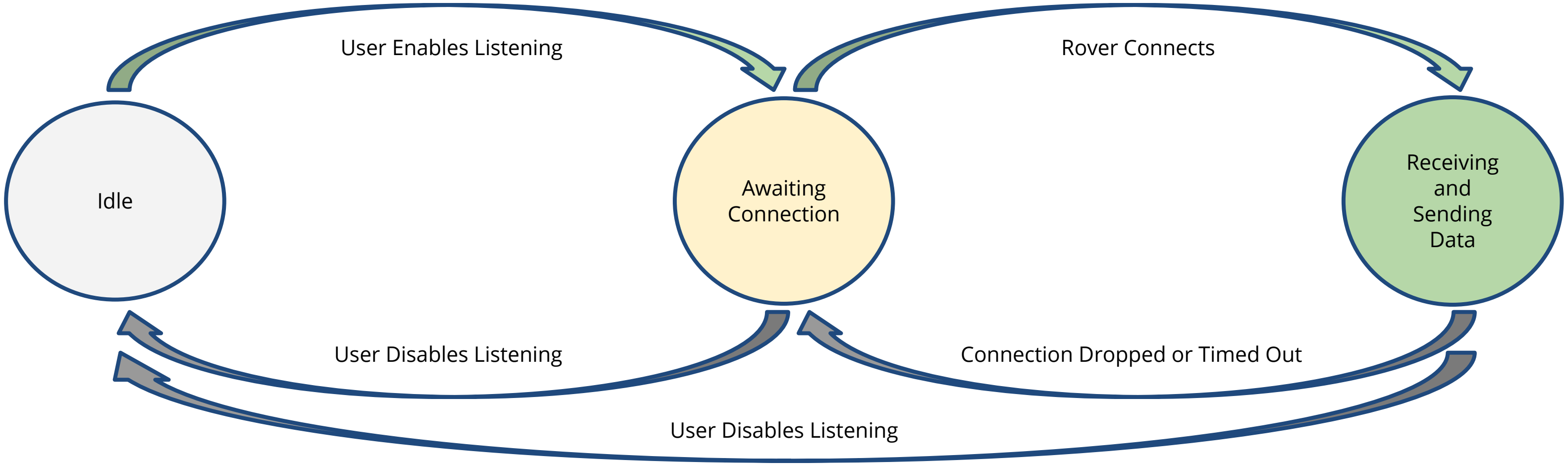
Commands

1. Establish network connection
2. Receive commands from base station
3. Deserialize command
4. Send command to appropriate auxiliary system

- Motherboard sends 150-300 JSON messages per second
- 5 second connection timeout on RED
    - If nothing fatal happened, then the motherboard on the rover will simply reconnect

RED Behavior

Figure 3: RED Network State Diagram



RED Behavior

Network Connection Management

- In order to maintain the responsiveness of the user interface (UI), any long-running operations must not occur in the same ***line of execution*** (thread) as that which renders the UI.
- I.e., the rendering of the UI cannot be dependent on the completion of a long-running task.

Network Connection Management is implemented based on the Asynchronous Programming Model (APM) in .NET

- Asynchronously accepts client connections and immediately goes back to listening for new connections
- Operations
- Utilizes Callbacks and Recursion which increases complexity

RED Behavior

Asynchronous Network Behavior Pseudocode:

```
1.  Start Listening; wait for a connection
    // Connected
2.  Start Receiving; wait for end of message
3.  Deserialize message
4.  Send data to appropriate module
5.  Go to 3
```

- If the connection is dropped or canceled, an error will be thrown and we will cease execution.
- The waiting (1 & 2) and the background work (3 & 4) is potentially blocking, so we utilize the APM to run both networking and ui code in parallel.

RED Behavior

Typical Network Failures:

1. Damaged Cords: poor wiring or stress from operations
2. Power Issues: faulty management of discharging by the Powerboard (blown traces on board)
3. Poor Signal Quality: out of range or blocked by an obstruction

RED Behavior

Message Format

- JSON
- Key->Value pair: Id -> Value
- Agreed upon set of values that have to be kept in sync between RED and the rover's Motherboard

Future Considerations for Message Formatting

- We want to make sure we can define a lot more telemetry and commands without straining the resources on the Motherboard
- A custom protocol with a smaller footprint may save the Motherboard from having to convert string messages to native datatypes.

RED Architecture

Getting User Input

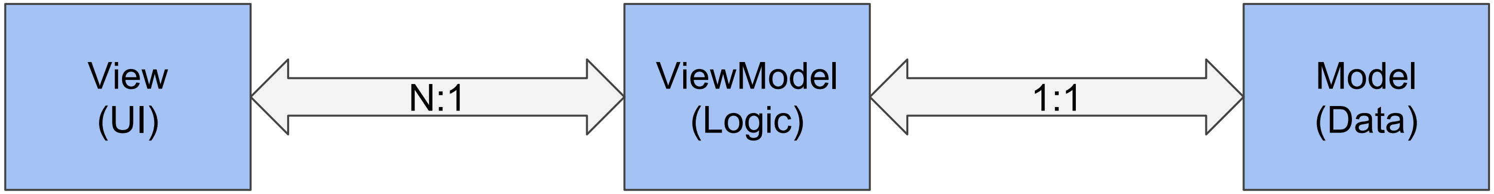
- Xbox controller used to gather commands for driving and operation of robotic arm
  - Implemented by polling the current values of the controller and translating the state of the controller to a particular command
- Mouse and keyboard were also required

RED Architecture - WPF

We utilized the Model-View-ViewModel (MVVM) design pattern with WPF

- Universally defines the organization of, and interaction between, classes within RED

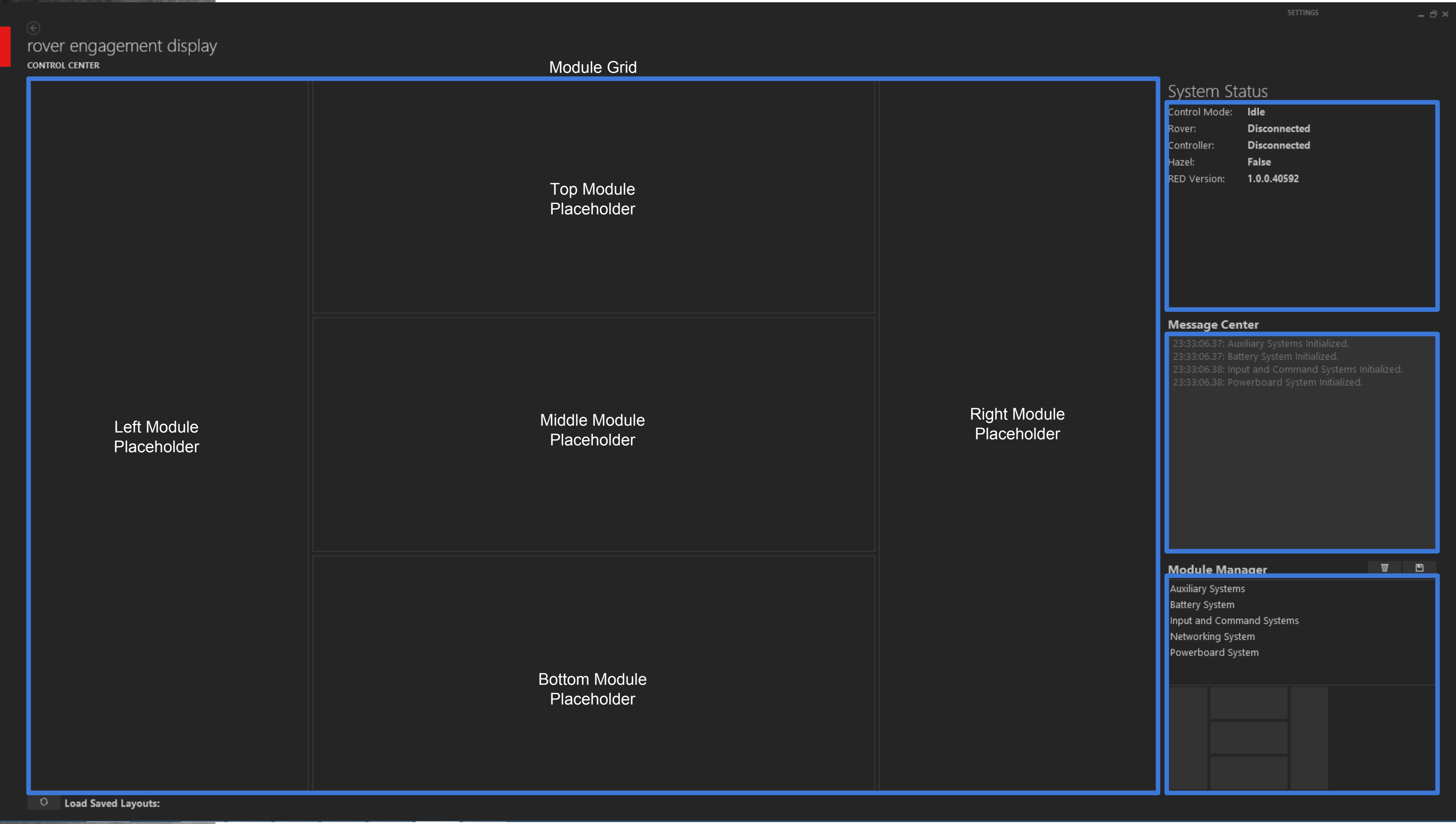
Figure 4: MVVM Entity Relationships



RED Architecture - Class Structure

- **Control Center** ViewModel
  - Control Center Model
  - **System Overview** ViewModel
  - **Console** ViewModel
  - **Module Manager** ViewModel
    - List of Modules
      - **Networking** ViewModel
      - **Battery** ViewModel
      - **Controller Input** ViewModel
      - ...
  - **Module Grid** ViewModel
    - Left Module Placeholder
    - Right Module Placeholder
    - ...
  - **Settings Manager** ViewModel
    - Settings Model
  - ...

Figure 5: RED 1.0 Control Center (no loaded modules)



Control Mode: Idle
Rover: Disconnected
Controller: Disconnected
Hazel: False
RED Version: 1.0.0.40592

```
23:33:06.37: Auxiliary Systems Initialized.
23:33:06.37: Battery System Initialized.
23:33:06.38: Input and Command Systems Initialized.
23:33:06.38: Powerboard System Initialized.
```

RED Architecture - Modules

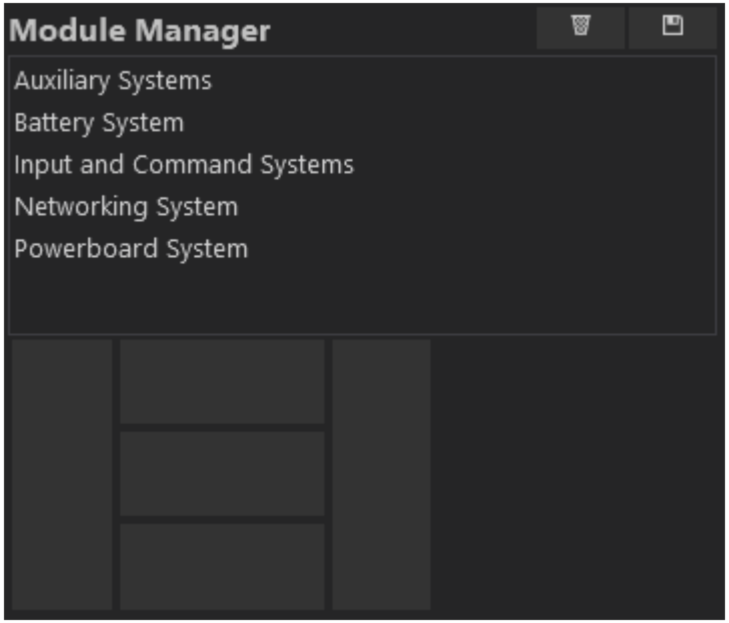
1. **Module Manager**
   1.1. Loads modules onto the grid
2. **Networking Module**
   2.1. Contains buttons to open/close connection and a console to view network-related logs
3. **Battery Module**
   3.1. Displays voltages and temperature for each battery pack as well as the overall pack current
   3.2. Most recent ten data points are displayed visually (histogram) to draw attention to spikes
4. **Auxiliary Systems Module**
   4.1. Displays data from science-related sensors. Contains buttons that send commands for activating auxiliary systems: drill, gripper, science bay door/lights

* Grid placeholders can be resized and module layouts can be saved for easy loading

RED Architecture - Modules

1. **Module Manager**
   1.1. Loads modules onto the grid

Figure 6: Module Manager



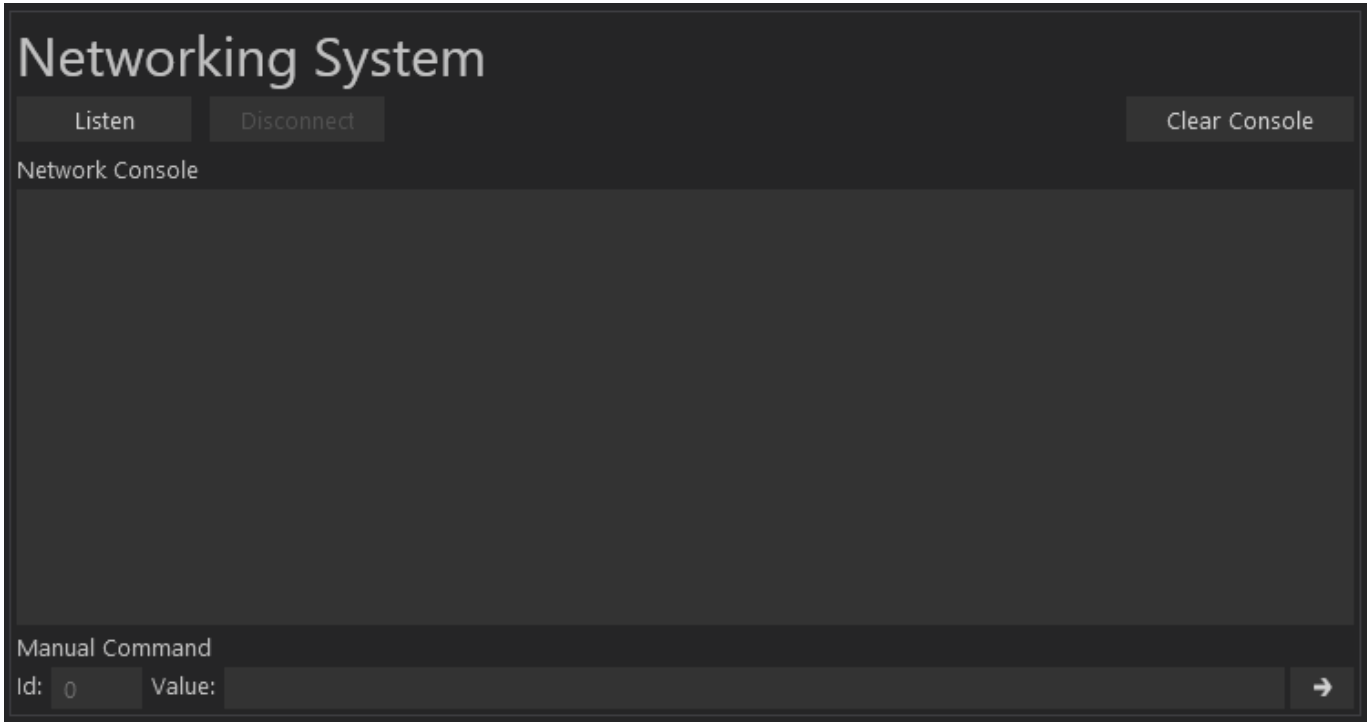
Module Manager

Auxiliary Systems
Battery System
Input and Command Systems
Networking System
Powerboard System

RED Architecture - Modules

2. **Networking Module**
   2.1. Contains buttons to open/close connection and a console to view network-related logs

Figure 7: Networking Module



RED Architecture - Communication

- Within the object-oriented programming paradigm, we employ composition in order to structure our code. This is seen in how the Control Center ***contains*** the Module Manager ViewModel.
- If we want a class to be utilized by multiple classes such as the Console, we pass the Console ViewModel into all of the classes that we desire to have access to it - dependency injection.

Examples

- We inject the Module Grid into the Module Grid Manager so that it can place Modules onto the Grid.
- We inject the Networking Module into the Input Module so that the commands that are translated from controller input can be sent over the network.

RED Architecture - Modules

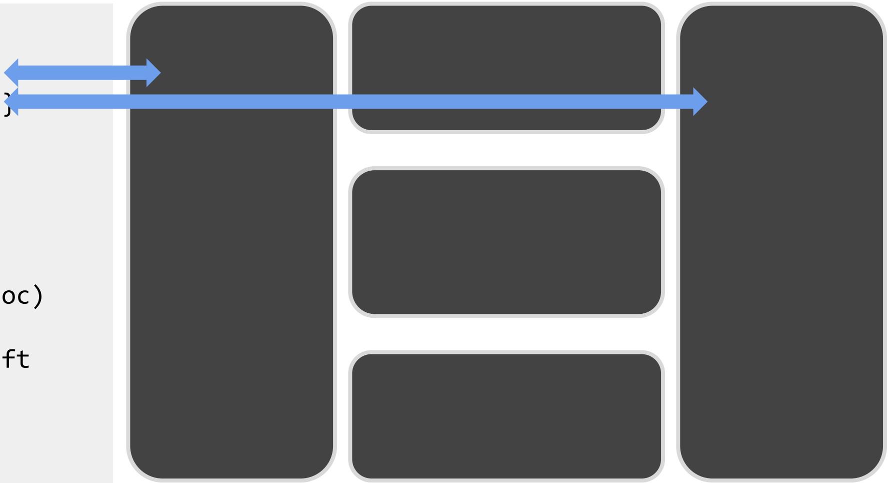
- Modules are the Views that appear in a placeholder.
  - Recall: each View is associated with a ViewModel.

- By using WPF (specifically, Data Binding and Data Templates) and MVVM, assigning a module to the grid and seeing it appear is as easy as <u>binding a UI element to a property and then assigning that property to a ViewModel</u>.

RED Architecture - Modules

Example: Module Grid Binding

```
ModuleGridViewModel (Class)
   public IModule LeftPlaceHolder {get;set;}
   public IModule RightPlaceHolder {get;set;}
   ...
   Constructor()
   ClearGrid()
   LoadState(ModuleGridState savedState)
   ...
   LoadModule(IModule module, GridLocation loc)
   {
      // Assuming loc equals GridLocation.Left
      LeftPlaceholder = module;
   }
```



```
ModuleGridView (UI Control - XAML)
      <UserControl.Resources>
            <DataTemplate DataType="{x:Type ModuleViewModels:BatteryViewModel}">
                  <ModuleViews:BatteryView />
            </DataTemplate>
            <DataTemplate DataType="{x:Type controlCenterViewModels:StateManager}">
                  <ModuleViews:NetworkingView/>
            </DataTemplate>
            ...
      </UserControl.Resources>

      <ContentControl Content={Binding LeftPlaceHolder} />
      ...
```

Future Work for RED

1. Unit Testing
2. Distributed base station architecture with multiple computers each running an instance of RED and they are all communicating with one another
3. Proprietary protocol that doesn't require conversion of string values to native numeric types
4. Allow users to add new key->value pairs for messages within RED (currently hardcoded)
5. Allow users to customize modules within RED (currently hardcoded)
6. Store all telemetry data in a database
7. Offline GPS module
8. Automatic application updates

Learn more @
marsrover.mst.edu
github.com/mst-mrdt

Questions?