THE UNIVERSITY of ADELAIDE

School of Mathematical and Computer Sciences
COMP SCI 3006 - Software Engineering & Project

Software Design Document

Lunar Mapping Rover

Revision 1.0

1st November 2017

Prepared by UG06

Philip Fai-Yuen Sung
a1211805

Zhongfan Zhang
a1670764

Brent Aaron Poland
a1647000

Charles Jonathon Zyzniewski
a1609619

Christopher Luke Lyndon
a1694039

Sioli Tiafau O'Connell
a1690418

Document Revision Table

Version	Date	Comment
0.1	3/10/17	Initial draft release.
0.2	6/10/17	Section 3.1, 3.2, and 3.3 updated.
0.3	10/10/17	Enhanced resource estimates section. Added requirements achieved sections to 3.2.
1.0	1/11/17	Final release.

Contents

1 Introduction

In order to create a system that is capable of achieving all of the goals outlined by Space Explorations there is a need to provide a succinct summary of the system software and it's implementation decisions.

1.1 Purpose and Scope

The Software Design Document(SDD) aims to provide an insight into the design decision for certain implementation choices whilst providing a deeper dive into the finer details of the system to provide further understanding.

The proposed system sets out to achieve all goals outlined from the Software Requirements Specification(SRS). To do this, each section of this document is collated to provide understanding of the entire system. This document starts with a high level overview and actively hones in on more specific content as the document progresses. The reader of this document should expect to see topics in the scope of system design decisions, components that make up the system, interactions between software systems and also ultimately gain deeper insights into system resource allocations.

1.2 Overview

The SDD is organised into six key areas.

System Overview

To ease the reader into the system in order for the further sections to be more palatable the System Overview section aims to outline the high level structure of the system. This section touches on the major components that make up the system and briefly touches on the interaction between these components.

System Architecture and Components Design

After an initial understanding has been created of the overall system the reader is prompted deeper into each of the components that were discussed from the overview. This also begins a conversation as to the 'why' behind the design choices of each subsystem.

Data Design

Now that the overall system is laid out the reader is prompted with a more technical explanation of the design decisions behind the data structures used to store the Lunar Rover map data.

Design Details

Once the reader understands what data to expect the system is explained in detail starting with class diagrams to understand the connection between classes and moving onto the states of the system that are possible to then finally explain the overall in depth interaction between system components with the interaction diagram.

Human Interface Design

By this time the reader is aware of the flow of the system and with the human interface design section they will then understand the connection made between the Lunar Rover operator and these software systems.

Resource Estimates

Finally after the reader has a full understanding of the entire system and how a Lunar Rover operator will interact with the system they are then shown the estimates of how much system resources and human resources are needed in order for the entire system to operate.

1.3 Constraints

- **Lego Mindstorms EV3 Kit:** We are limited by the hardware we are provided. To expand on this we are specifically limited by the sensors / motors we have available as well as the accuracy of detection / movement of these sensors / motors respectively.
- **Landing Environment:** We are constrained by the environment that the Lunar Rover will arrive to. As this information is unknown to us and our client, Space Explorations, we must develop a system that can be robust enough to handle the unknown features that may be encountered.
- **Human Resource Allocation:** Our progress of development is restricted by the physical man hours that are able to be devoted by each team member. This can result in certain design implementation choices being out of the question because of this time constraint.
- **Development Environment:** Our development implementation is constrained to the libraries we have available to us using the LeJOS Java based replacement firmware for the Lego Mindstorms EV3 brick.

1.4 References

Draw.io

In order to create styled diagrams that are easy to read we used `https://www.draw.io/`.

LeJOS Documentation

To be as accurate as possible with our definitions and system outline we referenced the LeJOS documentation at `http://www.lejos.org` throughout the development and documentation process.

SequenceDiagram Interaction Diagram Generator

With use under the Apache License Version 2.0 we used the IntelliJ IDEA plugin, SequenceDiagram to create interaction diagrams. More information can be found at `http://vanco.github.io/SequencePlugin`

2 System Overview

The Lunar Rover system is being prototyped on a Lego EV3 Mindstorms kit, and will run autonomously or be controlled by a user to explore a designated area. With this in mind, the system has been designed to run on a local device for data processing and logic control. A local network is used to connect to the Lunar Rover and will facilitate the receiving of data and sending of instructions to the Lunar Rover. A general overview of the architecture is shown in Figure 1 below.

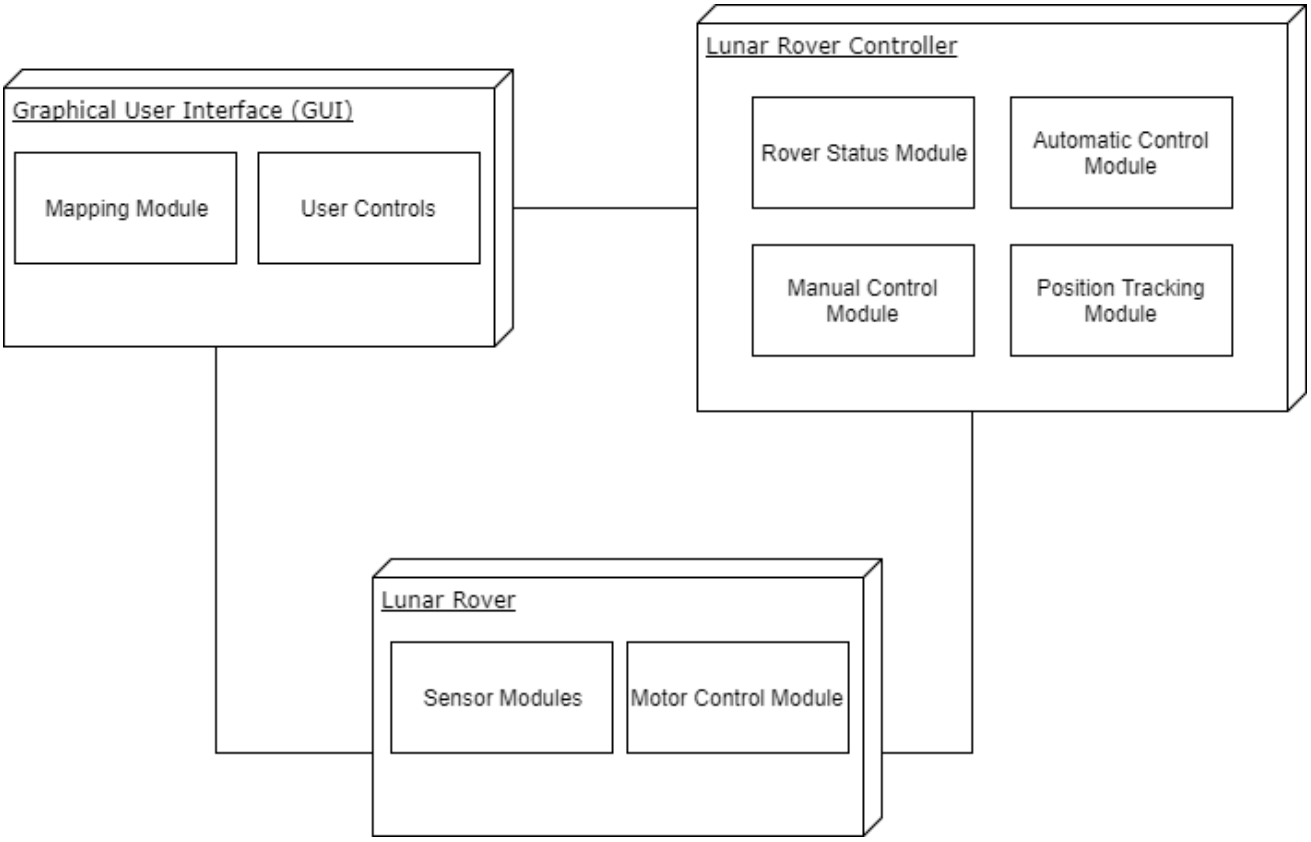


Figure 1: System Overview of the Lunar Rover

3 System Architecture and Components Design

3.1 Architectural Description

As shown in Figure 1, there are three main components, each containing their own subsystems.
The purpose of the GUI is to enable user interaction and viewing of status and map data from the Lunar Rover. The subsystems for this module are shown in Figure 2.
The Lunar Rover Controller is the logic controller that determines how the Lunar Rover should act, and sends this information to the Lunar Rover itself. The subsystems for this module are shown in Figure 3.
The Lunar Rover module refers to the control of the Lunar Rover and parsing of information from the sensors. The subsystems for this module are shown in Figure 4.
Interactions occur between all three components of the Lunar Rover System, for example the user controls send instructions to the motor control module of the Lunar Rover component and the Status Modules of the Lunar Rover Controller component.

For this section, components are given a unique identifier in the form of Cxxxx. Where 'x' is a number from 0-9. Components are not ordered with any significance.

3.2 Component Decomposition Description

C0001: Mapping Module

The Mapping Module parses the XML data of the map to be used in the Lunar Rover Controller and the GUI. It provides functionality to import and export the map data in XML format, and to graphically display the map in the GUI.

Requirements Achieved:

- R0004: Feature Cataloging
- R0005: Import Map
- R0006: Exportable Map Data

C0002: User Controls Module

The User Controls Module allows the Lunar Rover Operator to tell the Lunar Rover to either explore the survey area automatically, semi-automatically, or to wait until it is instructed directly via manual operation mode. In manual operation mode the User Control Module allows the user to turn left, right, or go forward or back for as long as the button is held. The module also provides the ability to draw no go zones on the map.

Requirements Achieved:

- R0005: Import Map
- R0009: Operation Mode Selection
- R0010: Manual Operation Controls
- R0011: Halt Lunar Rover Operation
- R0014: No Go Zone Creation Interface
- R0015: Click For Coordinate

C0003: Lunar Rover Status Module

The Lunar Rover Status Module provides information for other modules to use, such as the connection status of the GUI with the Lunar Rover and more direct messages such as warning messages for the Lunar Rover Operator in manual operation mode.

Requirements Achieved:

- R0008: Object Collision Detection
- R0016: Status Bar

C0004: Automatic Control Module

The Lunar Rover Operator needs to have a way to ensure that the Lunar Rover can map as much of the survey area as possible. The Automatic Control Module allows for traversal without Lunar Rover Operator input.

Requirements Achieved:

- R0001: Automatic Control
- R0004: Feature Cataloging
- R0007: No Go Zone Detection
- R0008: Object Collision Detection
- R0011: Halt Lunar Rover Operation

C0005: Manual Control Module

In a situation where more defined control is requested, the Lunar Rover Operator can take over manual control of the Lunar Rover using the Manual Control Module. This module allows for the Lunar Rover to be controlled in real time using the GUI and ultimately giving the Lunar Rover Operator the utmost control. This module will only warn the user if there is a safety issue and it is the responsibility of the Lunar Rover Operator to avoid any dangerous system states.

Requirements Achieved:

- R0003: Manual Control
- R0004: Feature Cataloging
- R0007: No Go Zone Detection
- R0008: Object Collision Detection
- R0010: Manual Operation Controls
- R0016: Status Bar
- R0017: Warnings for Manual Control

C0006: Position Tracking Module

The Lunar Rover needs to know where it is in relation to any inputted XML data as well as any features that are discovered. The Position Tracking Module works to keep the GUI up to date and in unison with the physical world to ensure correct operation by the Lunar Rover Operator as well as ensuring the correct output XML data for future use.

Requirements Achieved:

- R0001: Automatic Control
- R0002: Semi-Automatic Control
- R0003: Manual Control

- R0004: Feature Cataloging
- R0012: Examine Mapped Data
- R0015: Click For Coordinate
- R0016: Status Bar

C0007: Sensor Modules

In order for the Lunar Rover to operate safely and also to allow for feature discovery and recording the Lunar Rover is equipped with an array of sensor modules. These sensor modules are the Ultrasonic Sensor, Colour Sensor and Gyroscope. The system software will use this hardware to then communicate to the Lunar Rover Operator via the status bar as well as direct the Lunar Rover itself on it's traversal in the automatic and semi-automatic modes away from danger and the boundary of the survey area for example.

Requirements Achieved:

- R0004: Feature Cataloging
- R0007: No Go Zone Detection
- R0008: Object Collision Detection

C0008: Motor Control Module

The key to traversing the survey area is to have precise movement of the Lunar Rover throughout this area. To achieve this the Lunar Rover is equipped with independent motor control modules. This means that not only is the Lunar Rover able to traverse forward it can also rotate on the spot. The motor control modules are supported by an omni-directional rear wheel caster. Systematically depending on the operation mode the software of the system will move these motors forward or backward to achieve the correct traversal.

Requirements Achieved:

- R0001: Automatic Control
- R0002: Semi-Automatic Control
- R0003: Manual Control
- R0009: Operation Mode Selection

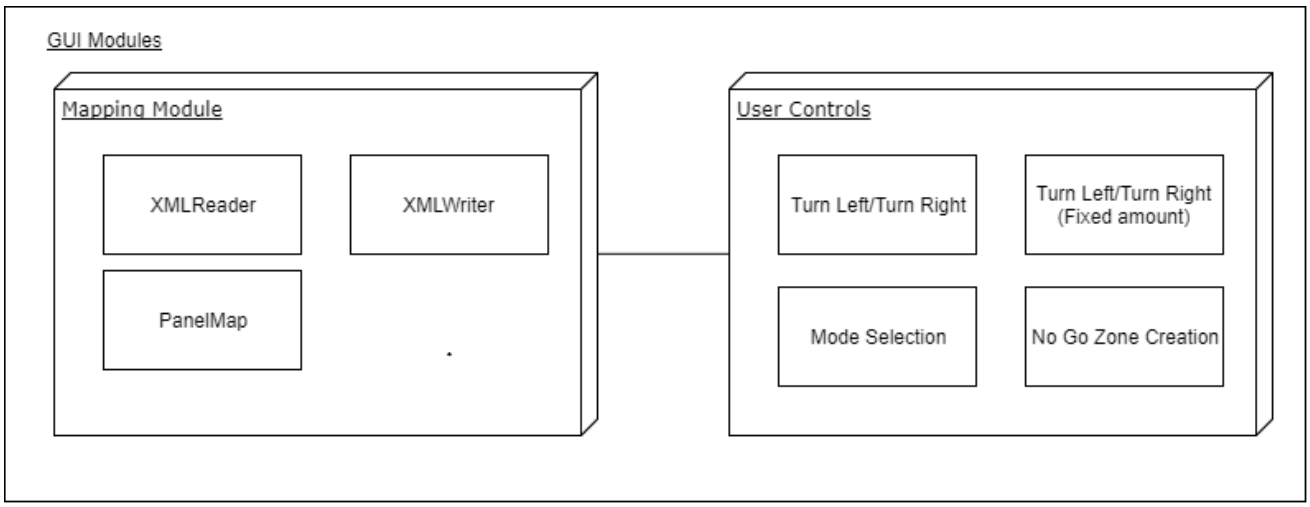


Figure 2: Subsystems of the GUI

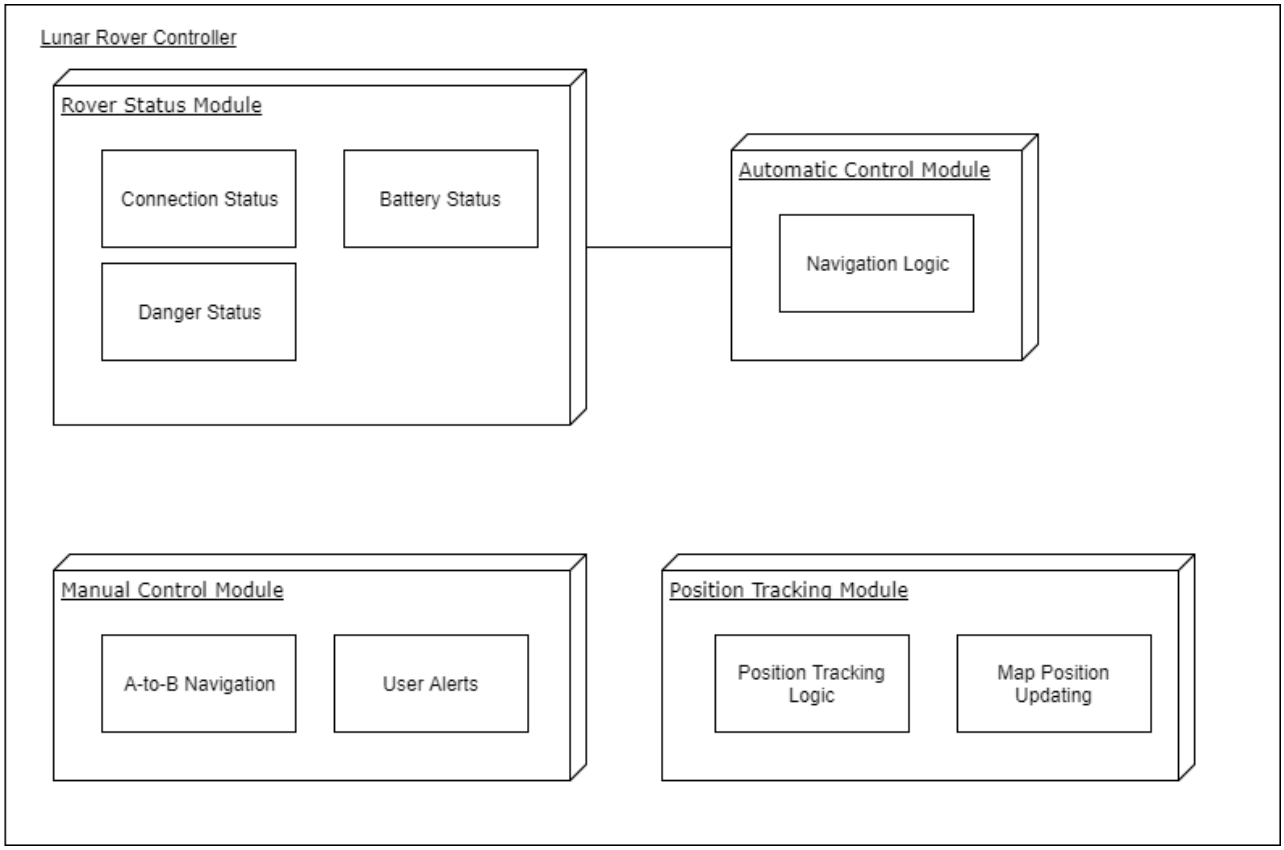


Figure 3: Subsystems of the Lunar Rover Controller

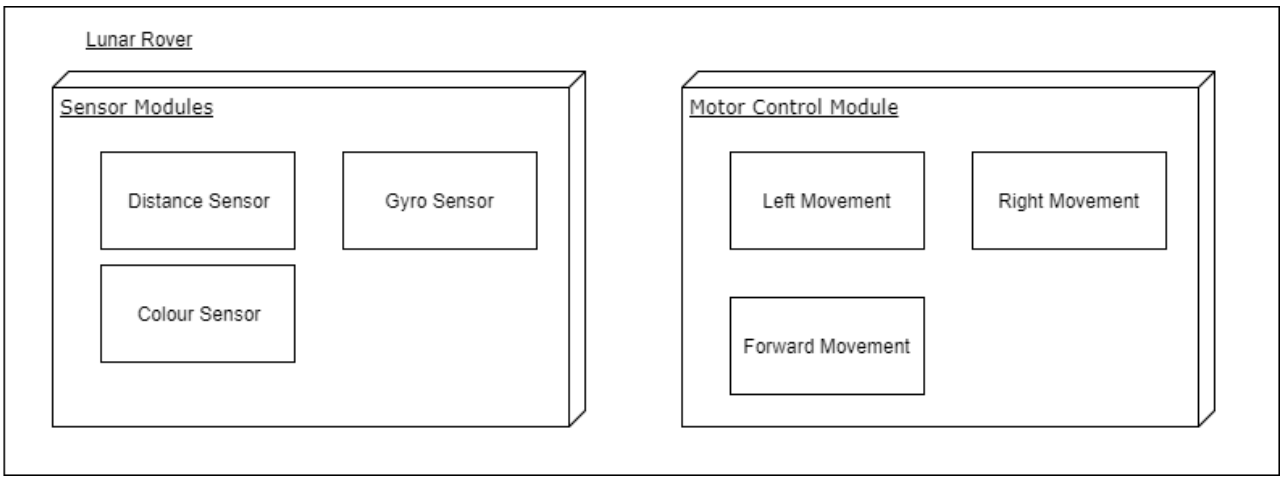


Figure 4: Subsystems of the Lunar Rover

3.3 Detailed Components Design Description

The purpose and function of each component is described in the section above. This section will list the following details of each component, where relevant.

- Subordinates: The components used by this component.
- Dependencies: Constraints placed on this component by other components.
- Interfaces: Control and data flow in and out of the component.
- Data: Description of internal data if there is any.

C0001: Mapping Module

Subordinates

- User Controls Module: No Go Zone creation is painted onto the panel map and is saved as part of the XMLWriter functionality.
- Manual Control Module: A-to-B navigation is reliant on the user viewing the panel map and choosing the location for the Lunar Rover to move toward.
- Sensor Modules: The distance and colour sensors are used to paint features onto the panel map for the operator

Dependencies

- Sensor Modules: Drawing of obstacles and features is dependent on the output from the sensors of the Lunar Rover.
- Motor Control Module: Accurate drawing of the map relies on the movements of the motors to be correct.

Interfaces

- Inputs: Sensor data from the sensor modules, user inputs from the user controls module (such as no go zone creation), and XML data from a file on the local machine (when loading a map).
- Outputs: XML data if the file is saved, and the graphical representation of the map.

Data

- Data for this module is stored in the XML structure detailed in appendix D.

C0002: User Controls Module

Subordinates

- Motor Control Module: Manual control mode involves the operator directly controlling the movement of the Lunar Rover. This requires instructions to be sent to control the left and right motors.
- Sensor Modules: The operator's movement of the Lunar Rover will utilise the sensors to inform them of imminent obstacles or dangers.

Dependencies

- Motor Control Module: The actuation of the operator's instructions is dependent on the motor control provided by this module.

Interfaces

- Inputs: User instructions via the GUI
- Outputs: Motor control instructions, no go zone data, and XML data when a map is saved.

Data

- No go zone creation requires coordinate inputs, the zone type, and zone radius (if a circular zone is created).

C0003: Lunar Rover Status Module

Subordinates

- Position Tracking Module: The location of the Lunar Rover relative to any obstacles and dangerous features is used to determine the danger status.
- User Controls Module: The current mode of Lunar Rover is set by this module.

Interfaces

- Inputs: Coordinates of the Lunar Rover, current mode of the Lunar Rover
- Outputs: Connection status, battery status, and danger status of the Lunar Rover.

Data

- The state of the Lunar Rover is stored as a string to be displayed on the GUI.

C0004: Automatic Control Module

Subordinates

- Lunar Rover Status Module: Reads the status of the Lunar Rover and whether there are any dangers to influence the decision of the exploration algorithm.
- Position Tracking Module: Used to help inform the exploration algorithm of how the Lunar Rover should move.
- Sensor Modules: The outputs from the sensor modules are used in the logic of the module to determine how the Lunar Rover should move.
- Motor Control Module: Used by the exploration algorithm of the automatic control module.

Dependencies

- Sensor Modules: Due to the physical limitations of the sensor hardware, the accuracy of the Automatic Control Module is dependent on the tolerances of the sensors.

Interfaces

- Inputs: Distance sensor status, colour sensor status, and stop or start automatic exploration instructions.

C0005: Manual Control Module

Subordinates

- Lunar Rover Status Module: Reads the status of the Lunar Rover and whether there are any dangers to notify the operator of.
- Position Tracking Module: Used to help with A-to-B navigation if the operator instructs the Lunar Rover to move to a particular location.
- Motor Control Module: The motor control module is instructed directly by the inputs of the operator while in manual control mode.

Dependencies

- Lunar Rover Status Module: Relies heavily on the danger status of this module to alert the user if the Lunar Rover is approaching an obstacle or a dangerous map feature.
- Motor Control Module: The responsiveness and accuracy of the motor control is a limitation to the manual control module. At higher speeds the operator's inputs may not be as accurate as expected, thus lower speeds are required.

Interfaces

- Inputs: The status of the Lunar Rover, operator inputs, and the coordinates of the Lunar Rover.
- Outputs: Instructions to the motor control module to drive the Lunar Rover.

C0006: Position Tracking Module

Subordinates

- Mapping Control Module: Uses the XML data to load the location of the Lunar Rover, or to keep track of its current location.

Dependencies

- Motor Control Module: Relies on the accuracy of the motor control of the Lunar Rover to know where it is relative to its starting position. Inaccuracy in the motor control such as forward movement not being a straight line can cause issues with tracking the location.
- Mapping Control Module: Relies on the accuracy of the XML data from the mapping control module. Assumes that all data makes sense and that the starting location is within the boundaries.

Interfaces

- Inputs: The amount of movement from the motor control module.
- Outputs: The coordinates of the Lunar Rover.

Data

- The current coordinates of the Lunar Rover are stored as a Point2D.

C0007: Sensor Modules

Interfaces

- Outputs: distance from any obstacles, the amount of rotation in degrees, and the colour of any features detected by the colour sensor.

Data

- Tracks the current rotation of the Lunar Rover since startup as a float value.

C0008: Motor Control Module

Interfaces

- Input: Which direction to move the Lunar Rover.
- Output: Activation and deactivation instructions to the Lunar Rover's motors.

3.4 Architectural Alternatives

RMI vs Sockets

Given the use of the LeJOS firmware for the Lego Mindstorms EV3 Kit there are two main ideologies for remote communication between a GUI system and the Lego Mindstorms EV3 brick. Firstly there is Remote Method Invocation (RMI) and secondly there is the use of network sockets. We chose to use the RMI pathway since the support for remote object oriented design is more than adequate for the requirements of the Lunar Rover for Space Explorations. We also decided that sockets left too much room for error in initializing the Lunar Rover connection in areas where there could be multiple sources

of interference causing incorrect port credentials and further connection information. With RMI the connection is handled by the implementation of the LeJOS firmware, leaving less room for error.

3.5 Design Rationale

The system is designed such that each module can be altered without affecting each other. This was done due to the nature of this project. As this is a prototype for a real mission, the EV3 kit will be changed to real hardware and as such having the modules separated will help to facilitate this transition. EV3 specific instructions are used to establish the connection to the Lunar Rover and control the motors, so it is important that these modules can be replaced without causing issue to others. By having this separate from the rest of the system it is easy to modify to suit any future hardware. The main compromise for using this method is that there is a delay between the controller and the Lunar Rover, which in some instances may lead to errors. The controller must take this into consideration during the Lunar Rover's operation.

4 Data Design

There are two main data types for this system. The first is the input and output data which is structured using XML. Secondly to make sure that this data is usable by the Lunar Rover with LeJOS firmware there are an assortment of Java inbuilt data structures. These structures hold map feature information so that this information can be displayed on the GUI as well as used by the Lunar Rover to operate safely.

4.1 Input / Export Data

Space Explorations have requested the use of their own custom generated DTD and in order to adhere to these requirements the system will take input and output data in the XML format from the supplied DTD outlined in Appendix D.

4.2 Data Structures

To hold data that is usable by both the Lunar Rover and the GUI, all data is stored as the member variables of the PanelMap class. To follow is a list outlining the data structures that store this data which lists the data description as well as the rationale behind this structure choice.

Structure	**Description**	**Rationale**
ArrayList<Point2D>	Coordinates relating to the boundary of the survey area.	Due to it's specific implementation as being used for X, Y coordinate systems, a Point2D is a logical choice for storing any coordinates with Java. The points relating to the boundary could be found by the Lunar Rover at any point and as such a dynamic data structure is required hence the use of an ArrayList.
Point2D	A coordinate relating to the last known location of the Lunar Rover.	Similarly for any coordinates a Point2D is the logical choice.
float	A floating point number relating to the last known heading of the Lunar Rover.	Due to the return value of the LeJOS Gyroscope readings a floating point number was the logical choice.
Point2D	A coordinate relating to the first discovered remnants of the Apollo 17 Crash Site.	Similarly for any coordinates a Point2D is the logical choice.
Point2D	A coordinate relating to the landing location of the Lunar Rover	Similarly for any coordinates a Point2D is the logical choice.

ArrayList<Point2D>	As obstacles are detected their location is recorded as a coordinate.	There is an unknown number of obstacles that the Lunar Rover could encounter so an ArrayList is a logical choice to handle the dynamic number of coordinates. Additionally for the GUI to be able to map the obstacles effectively a Point2D was used to represent the coordinate of the encountered obstacle.
Map<String, ArrayList<Zone>>	There are multiple different states of zones that are physical as well as non-physical and they are all mapped from their state to multiple area types such as a circle or a rectangular area.	Given that there are so many different states of zones a custom class was created to store and manipulate zone data effectively. For fast lookup a HashMap was used with the key being the state of zone. As the type of zones may not be unique a separate ArrayList of coordinates is allocated for each zone that is either inputted by the user or found by the robot.
Map<String, ArrayList<Point2D>>	As the Lunar Rover traverses the survey area it may encounter tracks of different types.	Similar to the zones that the Lunar Rover may encounter there is a dynamic number of possible tracks. Tracks result in a combination of connecting points. As the Lunar Rover encounters these points it needs to keep a track each different type as a set of coordinates similarly to the way the system keeps a track of zones, however tracks are not as diverse as zones and therefore don't need their own custom class

5 Design Details

During the requirements elicitation stage of our process we were able to ascertain a set of goals for the Lunar Rover. From these goals we've created a system to accurately achieve them. In the following section the system design shall be outlined in detail through the use of a set of diagrams to show the classes used in the system, the different states of the system and the methods of interaction between each subsystem of the Lunar Rover.

5.1 Class Diagrams

See Appendix A.

5.2 State Diagrams

See Appendix B.

5.3 Interaction Diagrams

See Appendix C.

6 Human Interface Design

6.1 Overview of the User Interface

This document will describe the current interaction of the GUI for the second milestone prototype. The user interface is comprised of multiple parts. On top is the menu bar, on the left is the main control panel for the Lunar Rover, on the right is the area where map data will be displayed, and along the bottom is a status bar which will inform the user of the Lunar Rover's current activities.

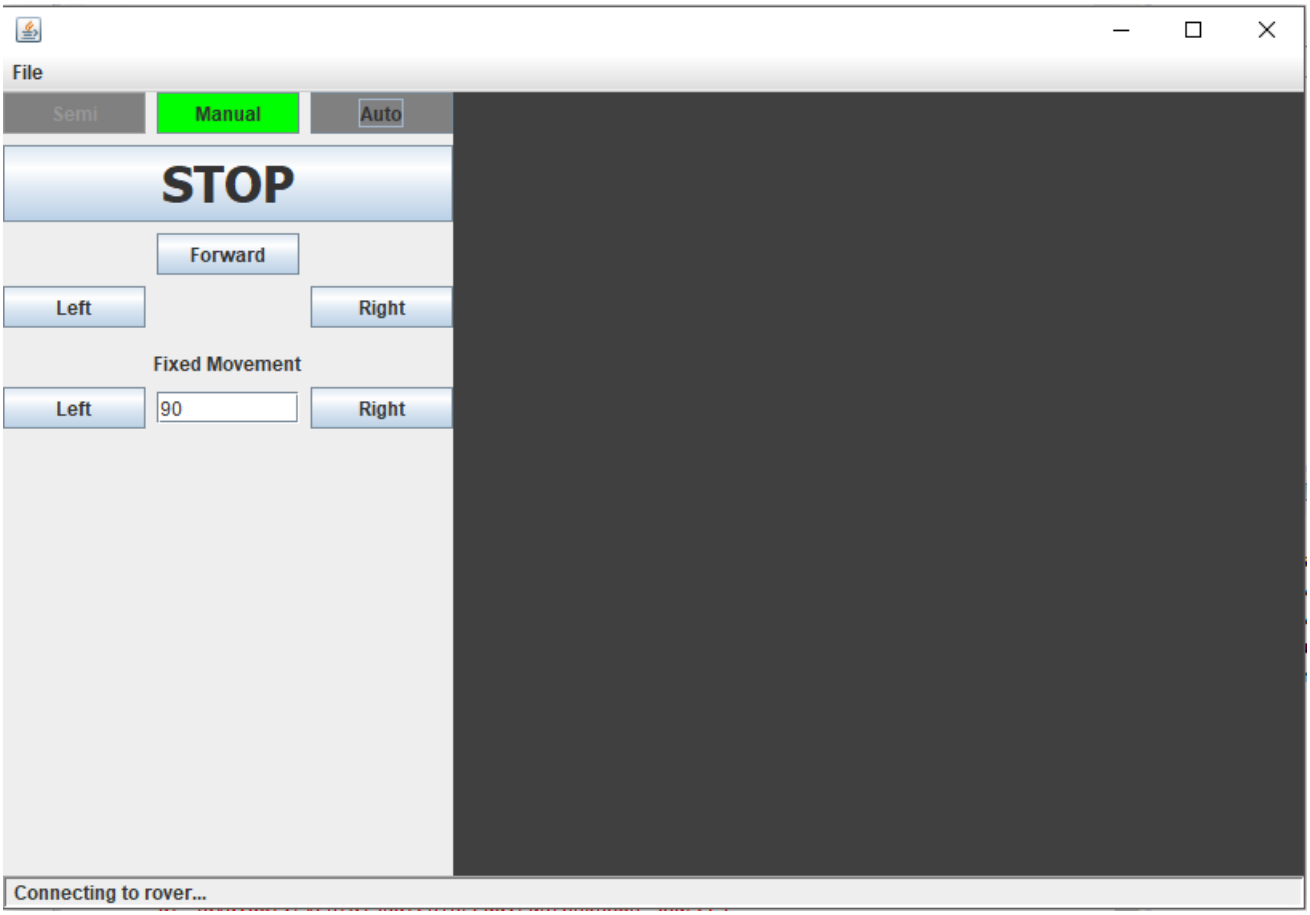


Figure 5: Overview of the Current GUI

6.2 Detailed Design of the User Interface

6.2.1 Menu

The menu allows the operator to import or export map data as well as safely closing the connection to the rover.

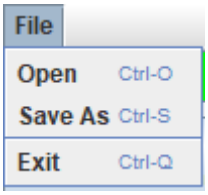


Figure 6: Menu showing Open, Save As and Exit

- **0013: Map Data I/O**: Any map data that has previously been saved can be imported via the Open option in the menu, while map data can be saved using the Save As menu option.

6.2.2 Main Control Panel

The main control panel consists of four control groups. These are the Lunar Rover mode toggle, the STOP button, the course manual controls and the fine manual controls.

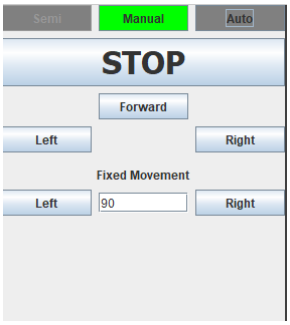


Figure 7: Main Control Panel

- **R0009: Operation Mode Selection**: The mode toggle allows the user to switch between the three operational modes of the Lunar Rover by clicking on one mode they wish to select. The active mode will be highlighted green, and the inactive modes will be greyed out.
- **R0011: Halt Lunar Rover Operation**: The STOP button allows the user to stop movement of the Lunar Rover at any moment. This button is large so that in case of an emergency the operator can quickly click it.
- **R0010: Manual Operation Controls**: The manual controls are split into course and fine control. The course controls allow the operator command the Lunar Rover to move forward and rotate left or right while the button is held. The fine controls allow the user to input the number of degrees the Lunar Rover should turn.

6.2.3 Map Area

The map area shows the map data that has been imported as well as any area that the Lunar Rover has mapped. It also is used in the semi-automatic mode as well as to define the no go zones.

- **R0012: Examine Mapped Data**: The map area gives the operator a view of the current map data by displaying it clearly on screen.
- **R0014: No Go Zone Creation Interface**: No go zone creation is done in the map area by clicking on the map to set points to enclose a no go zone area.
- **R0015: Click For Coordinate**: The operator can click on any area of the map and the GUI will display the coordinate at that point.

6.2.4 Status Bar

The status bar gives the operator important information about the lunar rover such as any warnings as well as the current status of the rover.

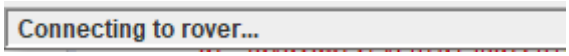


Figure 8: Status Bar Showing the State of the Lunar Rover

- **R0016: Status Bar**: The Status Bar and its information is shown on the bottom of the GUI.
- **R0017: Warnings for Manual Control**: If the Lunar Rover encounters any obstacle while it is in manual mode, a warning will appear in the status bar informing the user of the danger.

7 Resource Estimates

Client System Resources

The minimum requirements for running the Java Runtime Environment 1.7 for Windows and OSX are listed on the Oracle website(`https://java.com/en/download/help/sysreq.xml`). This acts as a guideline for what would be required by the client's system. The client's system must also have a network interface so that it can connect to the Lego Mindstorms EV3 brick. As a backup method of connection it is also recommended to have bluetooth connectivity capabilities. The GUI uses approximately 50MB of RAM while running, the client's system will also need at least 100MB of free hard drive space in order to store the required system information.
The client will also require the hardware of the Lunar Rover itself. Depending on the extent of future involvement with Space Explorations hardware maintenance may also be required.

Client Human Resources

In order for the Lunar Rover to operate there must be a trained Lunar Rover Operator who will be capable of operating the GUI to send commands to the Lunar Rover. So that the designated Lunar Rover Operator(s) can be proficient with the system, approximately 15 minutes of learning time would need to be allocated to have the Lunar Rover Operator(s) read through and become familiar with the Lunar Rover Operator Guide which would lead to a sufficient level of experience to begin traversal. From there and depending on the extent of the goals of the particular mission, the Lunar Rover Operator(s) may spend multiple hours tweaking their respective work-flow to achieve their objectives.

8 Definitions, Acronyms and Abbreviations

8.1 List of Definitions

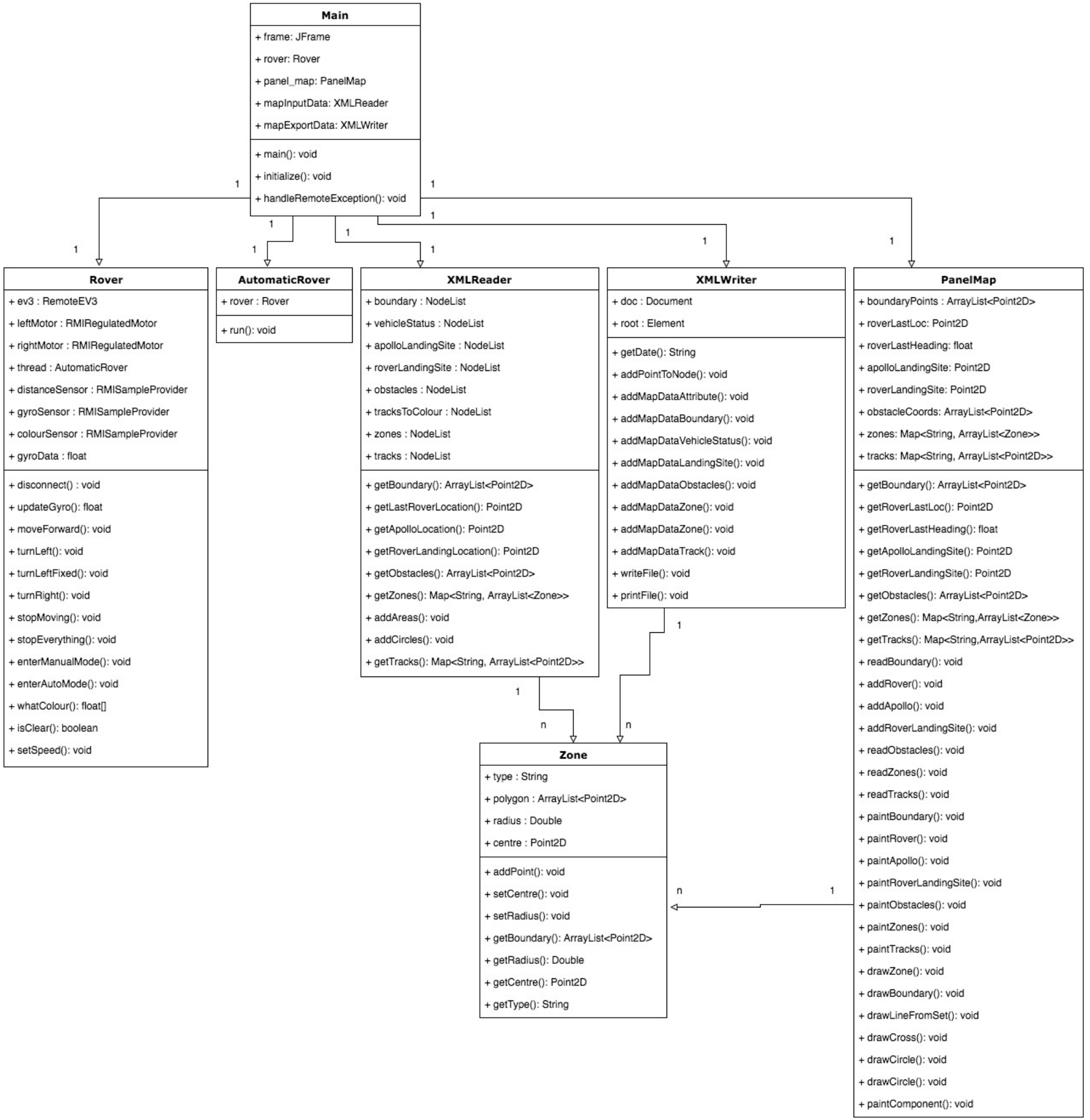
- **Space Explorations:** Space Explorations is our client. The needs of this client are the sole purpose of the creation of this entire system.
- **Lunar Rover / Lego Mindstorms EV3 Kit:** The Lego Mindstorms EV3 Kit is a finite set of Lego components targeted at aspiring robotics engineers. The aim of the kit is to provide an easy and fun to use way to enter into the advanced world of robotics. In this document this acts as the vessel to achieve the goals outlined by Space Explorations.
- **Software Requirements Specification:** The SRS provides feedback to the reader on the requirements elicitation taken from Space Explorations.
- **Software Project Management Plan:** The SPMP is in place to outline the processes used by the team to ensure the creation of the software system designed for Space Explorations.
- **LeJOS - Java Based Replacement Firmware:** LeJOS is an open source firmware designed for multiple Lego Mindstorms bricks. LeJOS is especially useful for more advanced developers in the fact that it allows for more advanced customisation using the Java development language.
- **Parsing:** Parsing is the act of taking one particular input language / style and converting that to usable data or vice versa.
- **Data Structure:** The term data structure in the software engineering sphere refers to the use of a inbuilt data storage implementation available in a given programming language. There are multiple types of structures each with their own benefits and challenges enabling the engineer to take advantage of the best way to store given data depending on the type of data and accesses to that data that are needed.
- **Remote Method Invocation:** Remote Method Invocation is a development ideology that a programmer can exploit where multiple objects may be operated on from multiple systems and more especially across distributed networks.
- **Sockets:** A network socket is an endpoint between a method of communication between two programs that are running simultaneously over a network.

8.2 List of Acronyms and Abbreviations

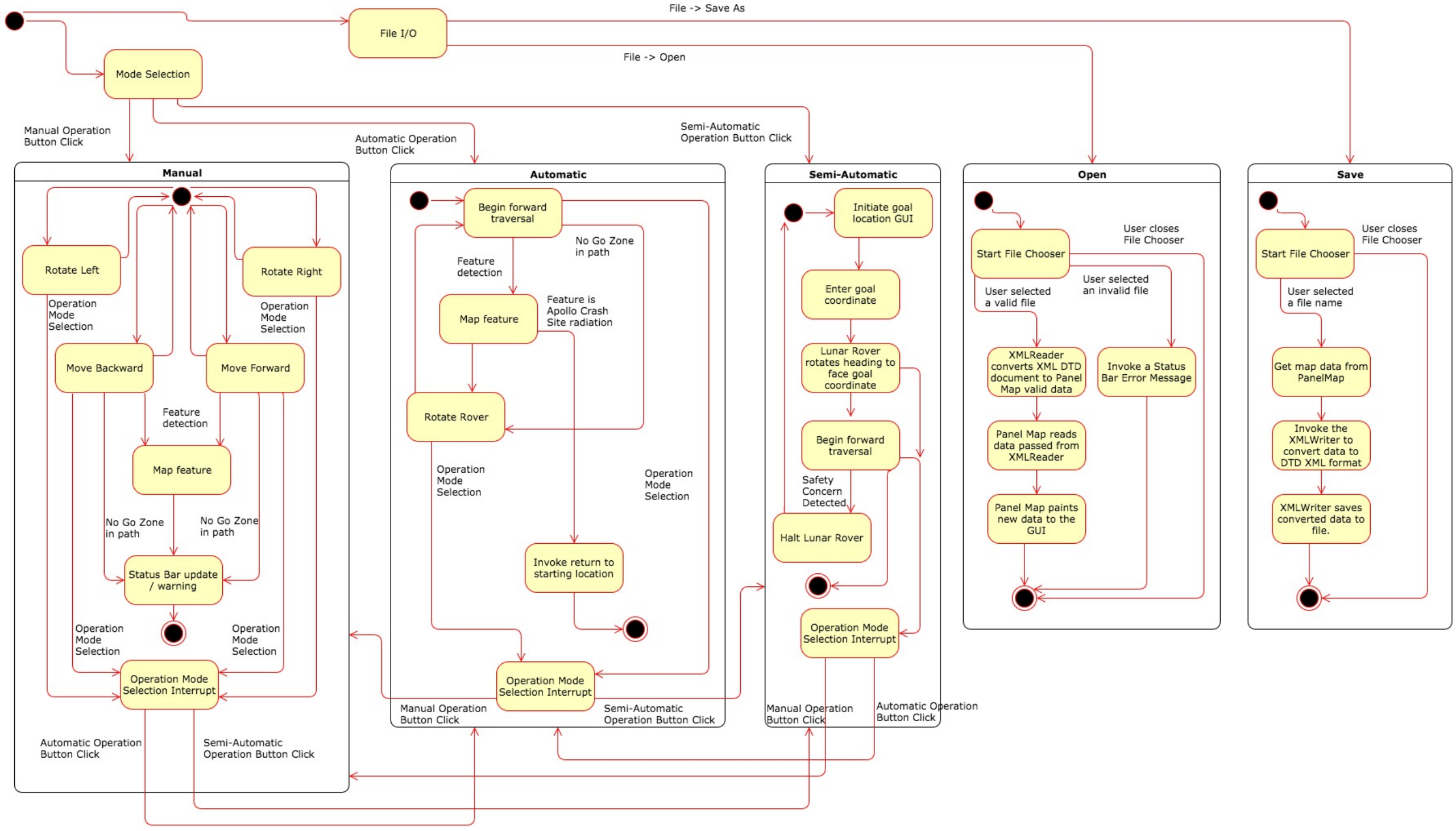
BPM	Backup Project Manager
DET	Development Team
DETL	Development Team Lead
DETBL	Development Team Backup Lead
DOT	Documentation Team
DOTL	Documentation Team Lead
DOTBL	Documentation Team Backup Lead
DTD	Document Type Definition
GUI	Graphical User Interface
IDE	Integrated Development Environment
NGZ	No Go Zone
PM	Project Manager
RMI	Remote Method Invocation
SDD	Software Design Document
SPMP	Software Project Management Plan
SRS	Software Requirements Specification
TL	Team Lead
XML	eXtensible Markup Language

9 Appendix

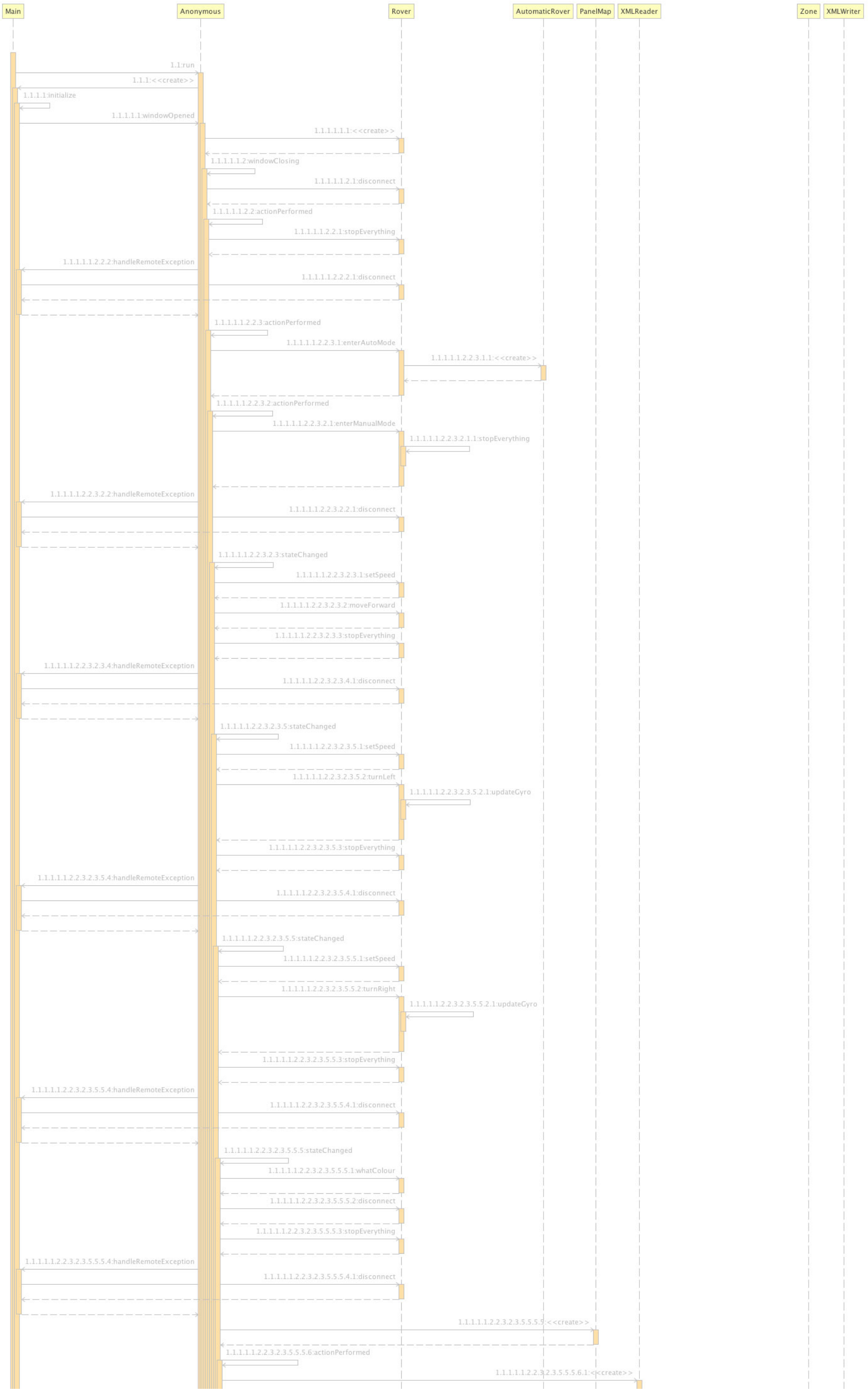
Appendix A - Class Diagram

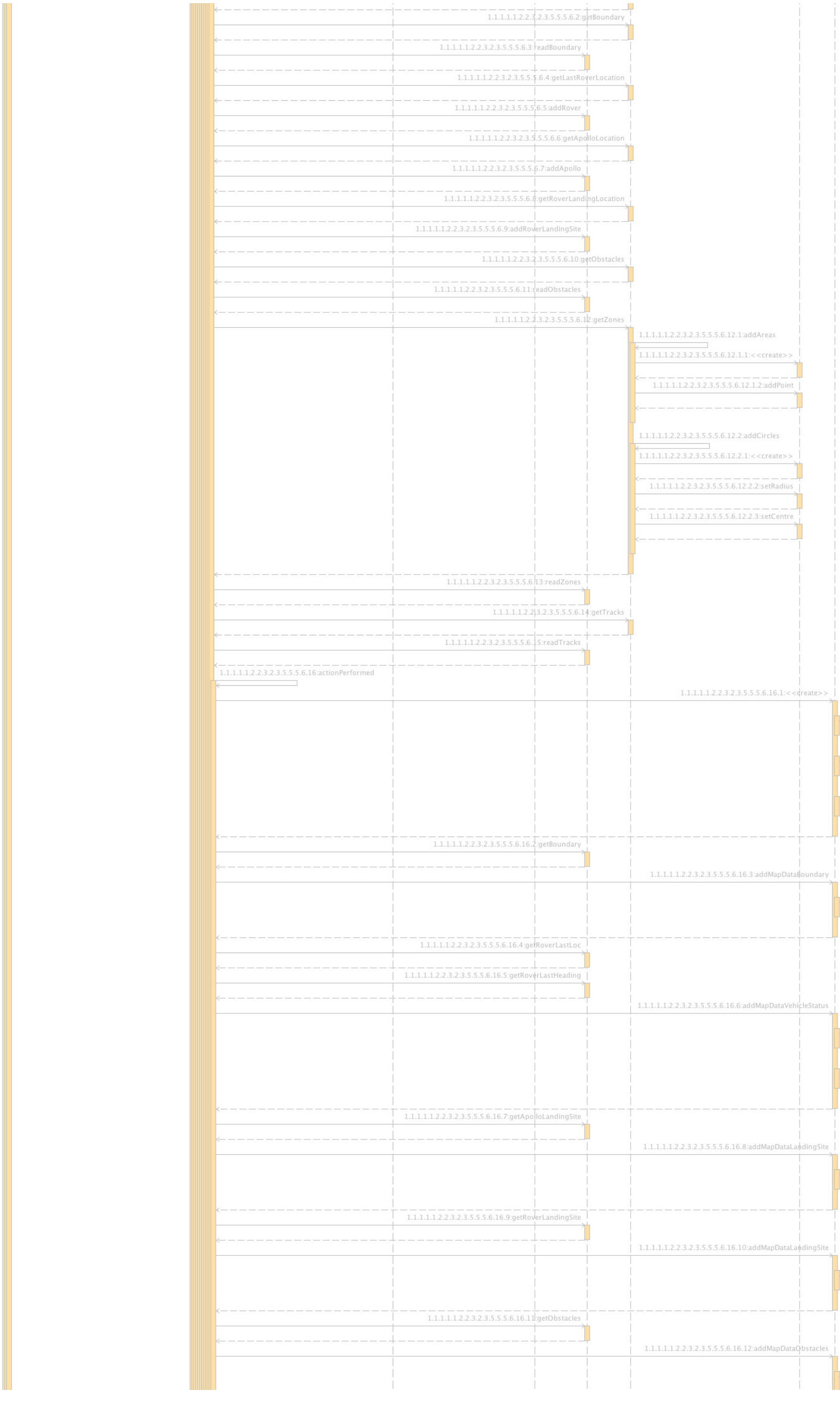


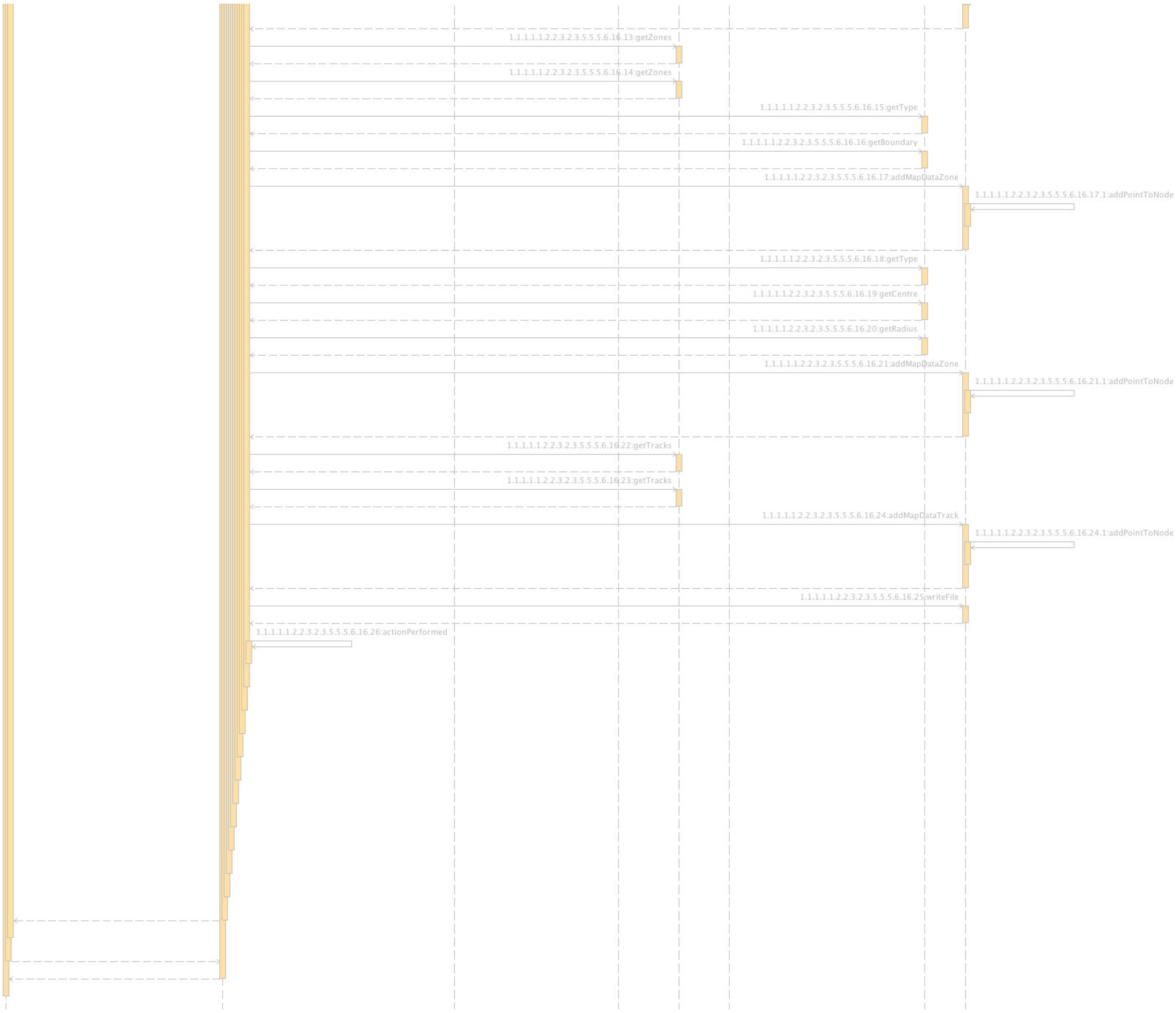
Appendix B - State Diagram



Appendix C - Interaction Diagram




1.1.1.1.1.2.2.3.2.3.5.5.5.6.2:getBoundary
1.1.1.1.1.2.2.3.2.3.5.5.5.6.3:readBoundary
1.1.1.1.1.2.2.3.2.3.5.5.5.6.4:getLastRoverLocation
1.1.1.1.1.2.2.3.2.3.5.5.5.6.5:addRover
1.1.1.1.1.2.2.3.2.3.5.5.5.6.6:getApolloLocation
1.1.1.1.1.2.2.3.2.3.5.5.5.6.7:addApollo
1.1.1.1.1.2.2.3.2.3.5.5.5.6.8:getRoverLandingLocation
1.1.1.1.1.2.2.3.2.3.5.5.5.6.9:addRoverLandingSite
1.1.1.1.1.2.2.3.2.3.5.5.5.6.10:getObstacles
1.1.1.1.1.2.2.3.2.3.5.5.5.6.11:readObstacles
1.1.1.1.1.2.2.3.2.3.5.5.5.6.12:getZones
1.1.1.1.1.2.2.3.2.3.5.5.5.6.12.1:addAreas
1.1.1.1.1.2.2.3.2.3.5.5.5.6.12.1.1:<<create>>
1.1.1.1.1.2.2.3.2.3.5.5.5.6.12.1.2:addPoint
1.1.1.1.1.2.2.3.2.3.5.5.5.6.12.2:addCircles
1.1.1.1.1.2.2.3.2.3.5.5.5.6.12.2.1:<<create>>
1.1.1.1.1.2.2.3.2.3.5.5.5.6.12.2.2:setRadius
1.1.1.1.1.2.2.3.2.3.5.5.5.6.12.2.3:setCentre
1.1.1.1.1.2.2.3.2.3.5.5.5.6.13:readZones
1.1.1.1.1.2.2.3.2.3.5.5.5.6.14:getTracks
1.1.1.1.1.2.2.3.2.3.5.5.5.6.15:readTracks
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16:actionPerformed
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.1:<<create>>
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.1.1:getDate
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.1.2:addMapDataAttribute
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.1.3:addMapDataAttribute
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.2:getBoundary
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.3:addMapDataBoundary
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.3.1:addPointToNode
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.4:getRoverLastLoc
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.5:getRoverLastHeading
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.6:addMapDataVehicleStatus
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.6.1:addPointToNode
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.6.2:addMapDataAttribute
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.7:getApolloLandingSite
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.8:addMapDataLandingSite
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.8.1:addPointToNode
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.9:getRoverLandingSite
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.10:addMapDataLandingSite
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.10.1:addPointToNode
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.11:getObstacles
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.12:addMapDataObstacles
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.12.1:addPointToNode


1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.13:getZones
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.14:getZones
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.15:getType
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.16:getBoundary
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.17:addMapDataZone
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.17.1:addPointToNode
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.18:getType
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.19:getCentre
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.20:getRadius
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.21:addMapDataZone
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.21.1:addPointToNode
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.22:getTracks
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.23:getTracks
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.24:addMapDataTrack
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.24.1:addPointToNode
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.25:writeFile
1.1.1.1.1.2.2.3.2.3.5.5.5.6.16.26:actionPerformed

Appendix D - DTD

```
<!DOCTYPE lunarrovermap [
  <!-- A Lunar Rover map containing one boundary, vehicle status, apollo landing site, rover landing site
  and track to color mapping, with any number of obstacles, zones, tracks and additional attributes.
  The Lunar Rover map uses a global unit of measurement for all measurements. The default unit of
  measurement is metres. -->
  <!ELEMENT lunarrovermap (attribute*, boundary, vehicle-status, apollo-landing-site, rover-landing-site,
  track-to-color, obstacle*, zone*, track*)>
  <!ATTLIST lunarrovermap units (metres|km|cm|mm) "metres">

  <!-- The site boundary described as a rectangular area -->
  <!ELEMENT boundary (area)>

  <!-- The status of the vehicle with its location and heading -->
  <!ELEMENT vehicle-status (point, heading, attribute*)>

  <!-- An obstacle identified as the Apollo landing site -->
  <!ELEMENT apollo-landing-site (obstacle, attribute*)>

  <!-- A location identified as the lunar rover landing site -->
  <!ELEMENT rover-landing-site (point, attribute*)>

  <!-- A mapping of track id to color codes - prototype mapping only -->
  <!ELEMENT track-to-color (attribute*)>

  <!-- An obstacle described as a set of contact points or an area surrounding by the obstacle -->
  <!ELEMENT obstacle ((point+)|area)>

  <!-- A zone with a circle or shape describing the enclosed region of the zone and its state -->
  <!ELEMENT zone (attribute*, (circle|area) )>
  <!ATTLIST zone
    state (nogo|unexplored|explored|radiation|crater) #REQUIRED>

  <!-- A series or points describing the shape and length of a track with sequential point elements -->
  <!ELEMENT track ((point+|area), attribute*)>
  <!ATTLIST track
    type (landing|vehicle|footprint) #REQUIRED>

  <!-- An area with one or more points describing its shape -->
  <!ELEMENT area (point+)>

  <!-- A circle contains a point of origin with a radius attribute -->
  <!ELEMENT circle (point)>
  <!ATTLIST circle
    radius CDATA #REQUIRED>

  <!-- A point is specified on a 2D cartesian plane. X progresses horizontally to the right, y progresses
    vertically and upwards. -->
    <!ELEMENT point EMPTY>
    <!ATTLIST point
        x CDATA #REQUIRED
        y CDATA #REQUIRED>

    <!-- A heading in degrees celsius. -->
    <!ELEMENT heading EMPTY>
    <!ATTLIST heading
        angle CDATA #REQUIRED>

    <!-- General Attributes definition -->
    <!ELEMENT attribute (key,value)>
    <!ELEMENT key (#PCDATA)>
    <!ELEMENT value (#PCDATA)>
]>
```