Directed Reading Report - High-Level Architecture

HARMANJOT SINGH MALHI, Brock University, Canada

1 INTRODUCTION

1.1 SEE(Space Exploration Experience)

The Simulation Exploration Experience (SEE) is an educational initiative organized by nasa aimed at equipping students with hands-on expertise in distributed simulation technologies, particularly focusing on the High-Level Architecture (HLA) framework. SEE emphasizes collaboration between students, academic institutions, and industry professionals to design, develop, and execute federated simulations that emulate complex real-world scenarios. With a strong focus on interoperability and real-time synchronization, the program fosters critical skills in modeling, simulation integration, and teamwork. Participants often engage in space-themed projects, such as simulating Lunar Rovers or space exploration missions, which serve as practical platforms to apply HLA principles[3].

SEE promotes experiential learning by utilizing tools like Pitch RTI and predefined software development kits (e.g., SEE kits). It enables participants to explore concepts such as time management, federation design, and object modeling in a controlled but innovative environment. The initiative's space exploration focus provides a unique opportunity for students to simulate the challenges of real-world aerospace applications, preparing them for careers in defense, aerospace, or advanced simulation engineering. By bridging academia and industry, SEE enhances the participants' understanding of HLA standards and their application in creating interoperable systems across diverse domains[2].

1.2 Distributed Simulation

Distributed simulation involves executing simulations throughout computing platforms that are geographically dispersed, unlike parallel simulations that execute on the same machine. These simulations may additionally leverage nearby location networks, the Internet, or even physical environments like sensor networks monitoring traffic. The number one intention is to make use of distributed resources, like equipment or people in different locations, and to position simulations near live data streams for predicting the actual operational structures. A key motivation for disbursed simulation has been the combination of numerous simulators. For example, early use of distributed simulations was seen in defense, for instance, tank simulators, flight simulators, and computer generated forces which could be combined to create distributed virtual environments for training[5].

1.3 High-Level Architecture

High-Level Architecture (HLA) is a framework that permits big-scale, distributed simulations by providing communication to the federates—individual simulation structures—working collectively within a federation. There are two primary players in the HLA system, the federates and the RTI. Federates are entities which may be deployed on different machines and they communicate with each other with the help of RTI to simulate a system. RTI(Run Time Infrastructure) is used by the federates to communicate with each other. RTI manages the time management among the federates which ensures the federates run in the correct

Author's address: Harmanjot Singh Malhi, Brock University, St. Catharines, ON, L2S 3A1, Canada.

sequence and it optimizes communication by using mechanism like publish and subscribe to relay the information. This allows federate to establish connection and communicate seamlessly and allows for simulations for defense training, predictive modeling, or traffic monitoring. As the want for scalable and interoperable simulations grows, particularly in fields like aerospace, defense applications, and independent motors, knowledge and enforcing HLA will become vital[1].

1.4 This Project and SEE

This project is also possible due to the SEE program. As a university student who was interested in doing a research project in distributed simulations, I was able to do that with the help of my project supervisor, Professor Robson De Grande. This project became possible because of the folks at the SMASH lab at University of Calabria. They guided me through the project and provided me with SEE resources and other resources needed to complete the project. This project is possible due to SEE because it allowed me to collaborate with SMASH lab and learn about Distributed Simulations and develop a project around HLA.

1.5 This Project and The Report

This project focuses on developing a lunar rover federate by using the SEE environment and deploy in a distributed environment. The primary goal of this project was to research and learn about HLA, understand its components, and develop a federate that might be integrated into a distributed simulation. The Lunar Rover Federate is designed to interact with different federates in actual-time, simulating the rover's actions and position updates in the surroundings.

The project is based as follows: the Background section affords a top level view of key HLA concepts, consisting of federation, federates, and time management. The Environment Setup segment offers a step-by-use-step guide for replicating the setup and implementing the Lunar Rover Federate. Finally, the Conclusion summarizes the findings and displays the project's implications for future research in simulation technology.

2 BACKGROUND

2.1 HLA Topology

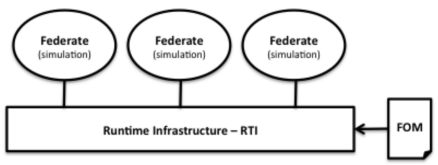


Fig. 1. High-Level Architecture

[7]

2.2 Distributed Simulation

Distributed Simulation is an abstract concept where entities/computational models deployed on geographically dispersed systems or nodes work together to simulate complex processes and systems. They are used in a variety of fields, for instance, defence, healthcare, and

gaming, to simulate scenarios that might not be possible on a single machine because of geographical constraints or computational constraints.

2.3 HLA

HLA is a concrete example of Distributed Simulations. The distributed entities are called federates and there is a central system called RTI which is responsible for providing communication between these federates. The whole environment is called the federation which is made up of the federates and the RTI. [7] The HLA is a very flexible system in the sense that federates can join and leave at any time and RTI is capable of handling the joining and leaving of the federates. HLA can be used to simulate scenarios like car simulation, restaurant simulation and even moon and a lunar rover.

2.4 Federation

A federation represents the collective machine of federates connected through the RTI, running synchronously to achieve a collective federation objective. The federation is governed via a Federation Agreement, a document outlining the rules and requirements for interaction, along with the Federation Object Model (FOM) that defines the types of gadgets, attributes, and interactions exchanged. Federation executions are periods where the federates actively perform actions together. Each execution is candid, allowing federates to enroll in or leave without disrupting the general federation. This method allows for flexible configuration of the federation objective.

2.5 Federate

A federate is a machine, simulator, or utility that connects to the RTI to take part in a HLA federation. Each federate can model unique entities, consisting of vehicles, environments, or gear, depending on its function within the simulation. Federates are accountable for publishing and subscribing to item attributes and interactions as defined inside the Federation Object Model (FOM), ensuring their contributions align with the federation's objectives. They additionally take care of callbacks from the RTI to system updates and interactions in actual time. Federates can range from records loggers and visualizers to complex simulators, making them flexible constructing building blocks for creating interoperable systems.

2.6 Time Management

Time management in HLA guarantees consistent event ordering and synchronization throughout federates, allowing simulations to run in real-time, quicker, or slower than real-international time. Logical time is valuable to this concept, enabling federates to increase their simulation time at varying rates at the same time as maintaining a cohesive timeline.. Federates use time-law and time-confined modes to manipulate how they produce and consume time sensitive information. Synchronization points also are employed to align all federates at crucial simulation milestones which facilitates time coordination.

2.7 Federation Object Model

The Federation Object Model (FOM) serves as the "language" of the federation, describing the objects, interactions, attributes, that federates use to communicate. Structured as an XML document following the HLA Object Model Template, the FOM offers readability and consistency in information representation throughout the federation. It includes item instructions , interaction training , and data types, which may be simple (integers, strings) or

complicated (data, enumerations). FOMs can be modular, allowing the reuse of standardized additives while permitting area specific customizations.

2.8 Federation Agreement

The Federation Agreement is a layout document that specifies how federates will interact within a federation. It includes detailed policies, objectives, and the FOM to make certain all collaborating federates align with the federation's necessities. Different domains, which includes protection, healthcare, or aerospace, can also require specific agreements due to varying specific requirements. This document promotes interoperability and minimizes conflicts, permitting federates from diverse businesses or systems to work cohesively. Over time, the Federation Agreement can evolve to house new federates or up to date requirements without disrupting existing configurations.

2.9 Management Object Model

The Management Object Model (MOM) is a predefined set of items and interactions in HLA that facilitates administrative and monitoring capabilities inside a federation. Federates can use MOM to query and manipulate the state of the federation, look into the overall performance and conduct of other federates, and control RTI configurations. MOM gives metrics about the system, such as the variety of federates, their joint reputation, and attribute ebook info, presenting transparency and enabling efficient debugging and system control. It plays an essential role in keeping the state of the federation in a good state.

2.10 RTI

The Runtime Infrastructure (RTI) is the backbone of HLA, acting as a middleware that allows communication and coordination among federates in a federation. It offers crucial services, inclusive of information alternate, synchronization, and management, making sure federates can seamlessly engage with each other while not having direct connections to each other. The RTI makes use of a push/subscribe mechanism to determine which federates produce and which federates consume information. Additionally, it manages logical time, synchronization points, and store/repair operations, critical for federations running in varied time scales or requiring regular states.

2.11 Central RTI Component

The Central RTI Component (CRC) is a key a part of the RTI, serving as the critical coordinator for federation executions. It manages federates' connections, tracks which federates have joined or resigned, and guarantees easy operation of the federation. The CRC is responsible for tasks like synchronization, distributing initialization steps, and keeping a steady view of federation. Running on the principal node, the CRC facilitates efficient communication and simplifies federation management.

2.12 Local RTI Component

The Local RTI Component (LRC) resides in each federate, performing as an intermediary between the federate and the RTI. It handles all communication, which includes sending and receiving updates, coping with time improvements, and processing callbacks from the RTI to the federate. The LRC guarantees federates perform independently while being synchronized with the federation. By running these things locally, HLA allows scalable simulations, allowing federates to run seamlessly even in different geographical locations.

The LRC is essential in enabling communication in geographically dispersed locations for a federation.

2.13 Summary

HLA works by connecting many computational entities called federates by using the middleware component called RTI. Federates are responsible for running their part in the whole simulation and use RTI to exchange data among them, and the RTI is responsible for maintaining communication, ensuring consistency and synchronization among the federates. The HLA defines a Federation Object Model(FOM) which is used to ensure the interoperability factor in the Federation because it defines the list of objects which are exchanged between the federates. This allows for a setup to develop flexible simulation where federates can dynamically join and leave, and enabling complex, scalable solutions to run seamlessly across distributed systems.

3 SEE AND DEMO

3.1 Demo

Section 4 explains the pre-requisites and shows how to setup the environment and develop a basic lunar rover federate which interacts with the environment. The demo is about developing a federate and running it with the environment to develop a practical understanding of HLA and developing federates and working with federations. In the setup, we develop a lunar rover federate and its components like the FOM, and the federate ambassador and use the libraries to code the components and then we run it to show a basic lunar rover simulation which just shows the position of the lunar rover being updated relative to the moon environment over time.

4 ENVIRONMENT SETUP

4.1 Pre-requisites

The machine used for this project is M1 Macbook air with 256 GB of storage, 16 GB of ram, and 8 cores. To install the pitchRTI software, you need to go to pitchRTI website, there will be software for different OS, install the latest one for mac and choose the free starter version. The Java version is needed is 1.7 or higher. Use eclipse as the IDE. For the SEE2015 folder needed for this project, you need to contact folks from the SMASH lab from University of Calabria.

4.2 Tools

(1) Eclipse [4]
(2) Pitch RTI [6]
(3) SEE [3]
(4) Java

4.3 Setup

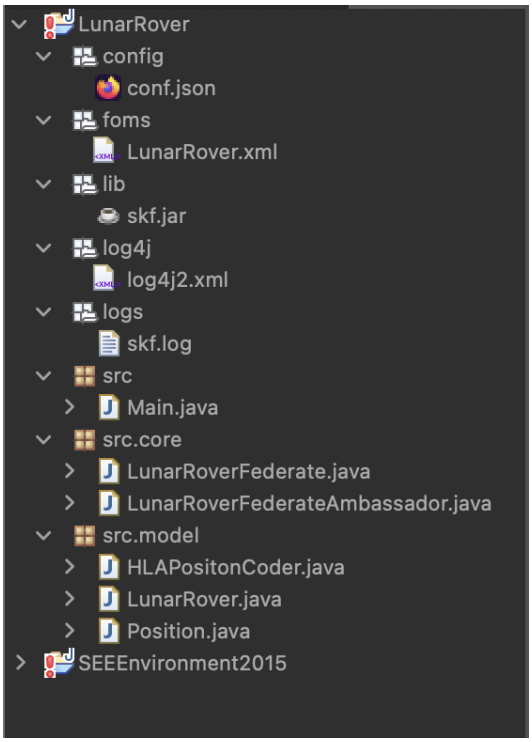


Fig. 2. Project Structure

(1) Above is the project structure which will be mentioned in the demo setup below.

(2) Launch eclipse

(3) Import the SEE2015_Environment kit. Move the SEE2015 into the current workspace

(4) Use jdk 1.7 or higher for the project.

(5) Create a new java project “LunarRover” in the same package.

(6) Create a new folder named lib under LunarRover which will be used to store all the libraries i.e. SKF and HLA jars files.

(7) Create a new config under LunarRover where will contain the .json configuration file.

(8) Create a log4j folder under LunarRover which will contain the .xml file according to Apache log4j specifications.

(9) Create a foms folder under LunarRover which will contain the FOM files related to the LunarRover

(10) Import the conf.json into config folder from SEE kit

(11) Change the host address to “localhost” in conf.json

(12) Import the LunarRover.xml file into foms.xml kit

(13) Import the skf.jar file into lib folder

(14) Then navigate to prti1516 —> lib and select booster1516.jar, prti.jar, prti1516e.jar and prticore.jar files and import them into lib folder under LunarRover

(15) Add the previously imported libraries to the build path.

(16) Import the log4j2.xml file to the log4j folder

(17) Change the filename path to “logs/LunarRoverLog” after importing the log4j2.xml

(18) Create a package named “model” under the src folder.

(19) A new class named “LunarRover.java” under the model package. Define attributes for the lunar rover like the entity name, parent reference frame and the entity type, and the position.
   (a) Create a new class named position which records the position of the lunar rover.

   (b) In the position class define the variables such as x, y, z which hold the position of the lunar rover. Include the default getters, setters, and toString method.

   (c) Create a new “HLAPositionCoder.java” class which implements the “Coder” interface from the skf library.

   (d) Create an EncoderFactory object and HLAfixedRecord object. Define three variables x, y, z with the type of HLAfloat64LE from the pitchRTI imported library.

   (e) In the constructor, get an EncoderFactory object from the RtiFactoryFactory class and assign it to the EncoderFactory instance variable.

   (f) Then use that factory object to create HLAfloat64LE() variables and assign it to the x,y,z instance variables.

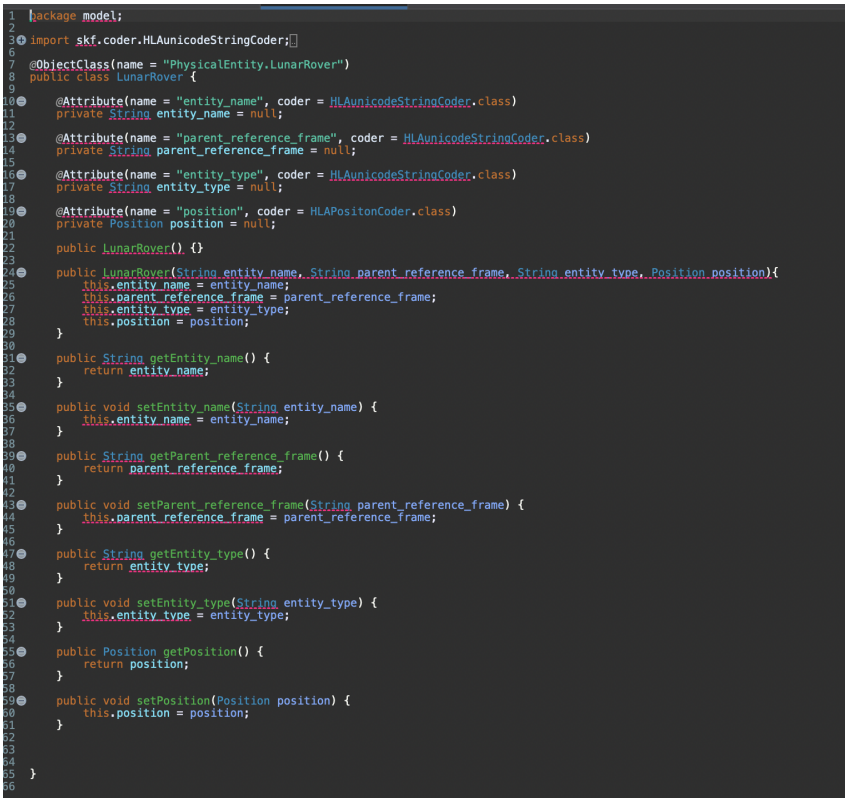
   (g) Then add them to the HLAfixedRecord object.

   (h) Implement the decode method by using the decode method from the HLAfixedRecord object and returning a new position object with the values set to the current x, y, z

   (i) In the encode method, set the x, y, z to the passed in position object’s x, y, z and return the HLAfixedRecord.toByteArray();

   (j) For the allowed type method, return Position.class

   (k) Include the getters, setters, toString() in the LunarRover.java class and create the constructor setting up the variables.

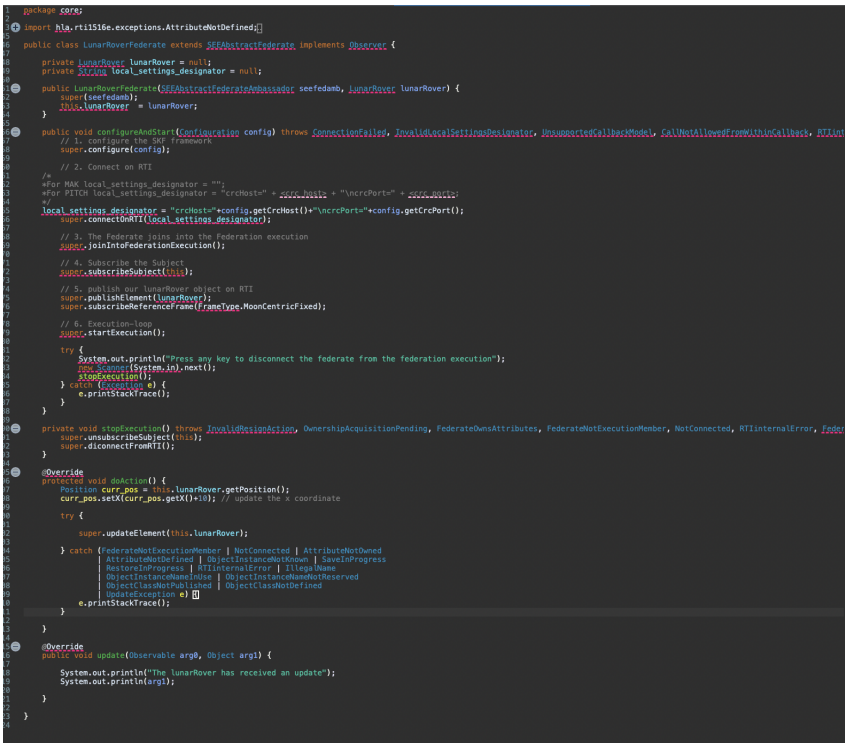


```java
package model;

import skf.coder.HLAunicodeStringCoder;

@ObjectClass(name = "PhysicalEntity.LunarRover")
public class LunarRover {

    @Attribute(name = "entity_name", coder = HLAunicodeStringCoder.class)
    private String entity_name = null;

    @Attribute(name = "parent_reference_frame", coder = HLAunicodeStringCoder.class)
    private String parent_reference_frame = null;

    @Attribute(name = "entity_type", coder = HLAunicodeStringCoder.class)
    private String entity_type = null;

    @Attribute(name = "position", coder = HLAPositonCoder.class)
    private Position position = null;

    public LunarRover() {}

    public LunarRover(String entity_name, String parent_reference_frame, String entity_type, Position position){
        this.entity_name = entity_name;
        this.parent_reference_frame = parent_reference_frame;
        this.entity_type = entity_type;
        this.position = position;
    }

    public String getEntity_name() {
        return entity_name;
    }

    public void setEntity_name(String entity_name) {
        this.entity_name = entity_name;
    }

    public String getParent_reference_frame() {
        return parent_reference_frame;
    }

    public void setParent_reference_frame(String parent_reference_frame) {
        this.parent_reference_frame = parent_reference_frame;
    }

    public String getEntity_type() {
        return entity_type;
    }

    public void setEntity_type(String entity_type) {
        this.entity_type = entity_type;
    }

    public Position getPosition() {
        return position;
    }

    public void setPosition(Position position) {
        this.position = position;
    }


}
```

Fig. 3. LunarRover.Java

(20) Create a new package under src folder where we deploy the core of the federation

(21) Create a new class “LunarRoverFederate” with its superclass being SEEAbstractFederate and interface being observer from java util’s library.



```
package core;

import hla.rti1516e.exceptions.AttributeNotDefined;

public class LunarRoverFederate extends SEEAbstractFederate implements Observer {

    private LunarRover lunarRover = null;
    private String local_settings_designator = null;

    public LunarRoverFederate(SEEAbstractFederateAmbassador seefedamb, LunarRover lunarRover) {
        super(seefedamb);
        this.lunarRover  = lunarRover;
    }

    public void configureAndStart(Configuration config) throws ConnectionFailed, InvalidLocalSettingsDesignator, UnsupportedCallbackModel, CallNotAllowedFromWithinCallback, RTIin
        // 1. configure the SKF framework
        super.configure(config);

        // 2. Connect on RTI
    /*
    *For MAK local_settings_designator = "";
    *For PITCH local_settings_designator = "crcHost=" + args.hosts + "\ncrcPort=" + args.ports;
    */
    local_settings_designator = "crcHost="+config.getCrcHost()+"\ncrcPort="+config.getCrcPort();
        super.connectOnRTI(local_settings_designator);

        // 3. The Federate joins into the Federation execution
        super.joinIntoFederationExecution();

        // 4. Subscribe the Subject
        super.subscribeSubject(this);

        // 5. publish our lunarRover object on RTI
        super.publishElement(lunarRover);
        super.subscribeReferenceFrame(FrameType.MoonCentricFixed);

        // 6. Execution-loop
        super.startExecution();

        try {
            System.out.println("Press any key to disconnect the federate from the federation execution");
            new Scanner(System.in).next();
            stopExecution();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void stopExecution() throws InvalidResignAction, OwnershipAcquisitionPending, FederateOwnsAttributes, FederateNotExecutionMember, NotConnected, RTIinternalError, Fedec
        super.unsubscribeSubject(this);
        super.diconnectFromRTI();
    }

    @Override
    protected void doAction() {
        Position curr_pos = this.lunarRover.getPosition();
        curr_pos.setX(curr_pos.getX()+10); // update the x coordinate

        try {

            super.updateElement(this.lunarRover);

        } catch (FederateNotExecutionMember | NotConnected | AttributeNotOwned
                | AttributeNotDefined | ObjectInstanceNotKnown | SaveInProgress
                | RestoreInProgress | RTIinternalError | IllegalName
                | ObjectInstanceNameInUse | ObjectInstanceNameNotReserved
                | ObjectClassNotPublished | ObjectClassNotDefined
                | UpdateException e) {
            e.printStackTrace();
        }

    }

    @Override
    public void update(Observable arg0, Object arg1) {

        System.out.println("The lunarRover has received an update");
        System.out.println(arg1);

    }

}
```

Fig. 4. Lunar rover federate

(a) Define a LunarRover object in the class.

(b) In the configureAndStart method, take these steps to connect to the federation: Configure the skf library by using the configure method from the parent abstract class.

(c) Define a string as an instance variable, which holds the crc host address and crc port number.

(d) Use the parent's "joinIntoFederationExecution" method to join SEE Federation Execution. Use the parent's "subscribeSubject" method to subscribe to the subject.

(e) Use the "publishElement" method to publish the lunarRover object which is the instance variable.

(f) Use the "subscribeReferenceFrame" method to subscribe to the "MoonCentricFixed ReferenceFrame.

(g) Then start the execution loop with the method "startExecution" from the parent class. So that you can disconnect from the federation. Use a scanner object to read any key and when any key is pressed use the "unsubscribeSubject" method to unsubscribe from the federation followed by "disconnectFromRTI" method to disconnect from the RTI. Put this in a try catch block.

(h) In the doAction method, get the "lunarRover" position by using the "getPosition" method and set it to a position object. Update the x variable of the position object by adding 10 using the "setX" method on the position object. Then update the lunarRover object by calling parent's "updateElement" method.

Create a new class "LunarRoverFederateAmbassador" which will communicate with the federation for the federate, LunarRoverFederate. Create the constructor and call super in the constructor.

Fig. 5. Main class

(a) Create a new class LunarRoverMain where we will combine all the components and run the federate.

(b) Create a "LunarRover" object.

(c) Create a "LunarRoverFederateAmbassador" object

(d) Create a "LunarRoverFederate" object by passing in the LunarRover and LunarRoverFederateAmbassador object

(e) Start the federate by using the configureAndStart on the "LunarRoverFederate" object

(22) Now we are ready to run the federation.

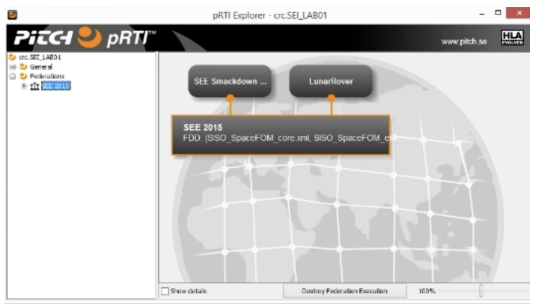


Fig. 6. PitchRTI Diagram

(a) Run the PitchRTI software by going to the prti1516e directory. Then go inside the bin folder. After that launch the PitchRTI software by double clicking pRTI application.

(b) Run the “Environment.java” file and then the “LunarRoverMain.java” file as “Run and Configuration” and change the classpath in the User Entries to include the log4j folder.

(c) Then apply and run.

(d) Now you can see the results in the terminal of rover’s position being updated and check out the PitchRTI software to see the federates running.

After running the files, you should be able to see the results in the console where the position of the rover is being updated with respect to the moon. There was also sample 3d visualization of the rover on the moon for the simulation but it belongs to nasa and we could not get permission to use that video. During the 2024 eclipse event at Brock University, we were able to present our work where we showed other people at the event about the project and HLA through a presentation. We explained how we collaborated with SMASH and what we were developing, and also explaining the benefits and practical uses of the simulation. We also showed the demo by running the code in Java and were able to demonstrate the code and the output.

5 CONCLUSION

In conclusion, this project gave an in-depth exploration of the High-Level Architecture (HLA) and gave an opportunity to work with distributed simulations via the development of a Lunar Rover federate. By integrating the concepts of federation design, time management, and interoperability the use of tools which includes Pitch RTI and the SEE framework, the project demonstrated the practical challenges and opportunities in a distributed simulation environment. The process of constructing the Lunar Rover federate offered precious insights into object modeling, synchronization, and real-time conversation within a federation, showcasing the flexibility and scalability of HLA for diverse simulation scenarios. Through hands-on implementation and a deep dive into the principles of HLA, this project showed the importance of standardized processes for collaborating and integration in general. The learnings from this project not only enhance the knowledge of HLA but also show the importance of distributed simulations in a wide variety of fields, including defense, aerospace, development, etc.

REFERENCES

[1] 2010. IEEE Standard for Modeling and Simulation (MS) High Level Architecture (HLA)– Framework and Rules. *IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000)* (2010), 1–38. DOI:http://dx.doi.org/10.1109/IEEESTD.2010.5553440

[2] Agostino G. Bruzzone, Luciano Dato, and Angelo Ferrando. 2014. Simulation Exploration Experience: Providing Effective Surveillance and Defense for a Moon Base Against Threats from Outer Space. In *2014 IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications*. 121–126. DOI:http://dx.doi.org/10.1109/DS-RT.2014.23

[3] Simulation Exploration Experience. 2023. Simulation Exploration Experience (SEE) 2023. (2023). A global initiative uniting students, industry, and NASA for collaborative space mission simulations. The event focuses on lunar and Mars exploration, using distributed simulation technology for modeling and testing mission elements. Available at https://simulationexplorationexperience.org/.

[4] Eclipse Foundation. 2023. Eclipse IDE for Java Developers (includes Incubating components). (2023). Copyright Eclipse contributors and others, 2000-2023. Eclipse and the Eclipse logo are trademarks of the Eclipse Foundation, Inc. Oracle and Java are trademarks or registered trademarks of Oracle and/or its affiliates. This product includes software developed by the Apache Software Foundation.

[5] Richard Fujimoto. 2015. Parallel and distributed simulation. In *2015 Winter Simulation Conference (WSC)*. 45–59. DOI:http://dx.doi.org/10.1109/WSC.2015.7408152

[6] Pitch Systems. 2023. Pitch pRTI 5.5.2. (2023). Version 5.5.2, available from https://www2.pitch.se/pRTI1516e/Releases/v5.3.2.1/MxdozAZrF/install.asp.

[7] Pitch Technologies. 2023. HLA Tutorial: Introduction to High-Level Architecture (HLA) for Simulation. *Pitch Technologies Documentation* (2023). Available from https://www.pitchtechnologies.com/.