This section details a short tutorial on the implementation of self-configuration approach. The following steps can be undertaken to set up the environment. The code is hosted in GitHub repositories in respective folders.

Github Repo: <https://github.com/Hamood564/CodeWalkthrough.git>

## Agent Deployment

**Folders:** Line Simulation, Test Simulation & Lib.

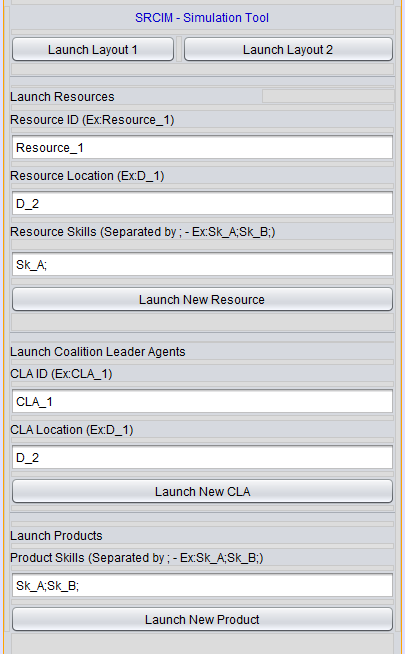
Download these folders**.** The Line Simulation and Test Simulation are separate NetBeans projects compatible with NetBeans IDE 8.2 and JAVA JDK 8. Lib folder contains additional supporting libraries for these projects.



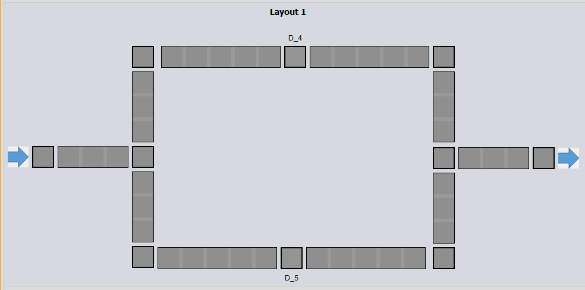
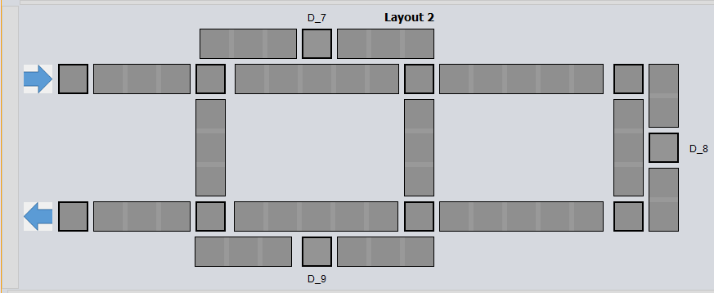
Line Simulation project contains the digital twin model for the functionality execution and the product routing. Test Simulation project contains the agent system for functionality execution. The operation of agent system is detailed in Chapter 8 and Chapter 9 respectively. The inner working of the agent system (architecture, behaviors and protocol) can be understood from the book *Developing Multi-Agent Systems with JADE.*

A brief description of the packages in each project are as follows:

1. **Console Agent:** Contains the console frame and the console agent code that interacts with the frame in the simulation tool. Frame to agent system communication is also present in this package.



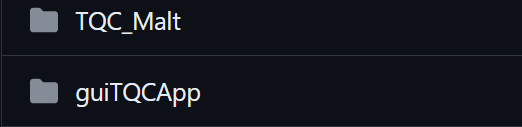
1. **Simulation:** This package contains the two layouts developed for the simulation. The resources can be allocated to resource locations and depending on negotiation criteria product be routed to these resources.

****

1. **Transport:** This package contains transport agent, the transport rules for the product and the relevant hardware interface abstraction. This package also contains the rules for the interface and the classes to map the state of product across the layouts.
2. **Transport Libs:** Thispackage contains thehardware simulation and layout rules.
3. **Utilities:** Contains the constants and directory facilitator interaction mechanism that can be used by the Line Simulation project.
4. **Coalition Leader Agent:** This package contains information on coalition agent. Not implemented in the current strategy.
5. **Product Agent:** This package contains the classes to implement product agent, its behaviours and relevant communication with the simulation & physical environment.
6. **Resource Agent:** This package contains the classes to implement resource agent, its behaviours and relevant communication with the simulation & physical environment.
7. **Mqtt:** This package contains the implementation of mqtt protocol for publish and subscribe. These classes are used to communicate with the physical hardware controller.
8. **WebSockets:** This package contains the implementation of websockts between agents and hardware.
9. **Util:** Contains the constants and directory facilitator interaction mechanism that can be used by the Test Simulation project.

## State Chart and State Machine Functionality Representation

**Folder:** State Chart to State Machine



The folder contains two projects “TQC\_Malt” and “guiTQCApp”. These projects are compatible with Eclipse IDE and JAVA JDK 11. The description of these projects is as follows:

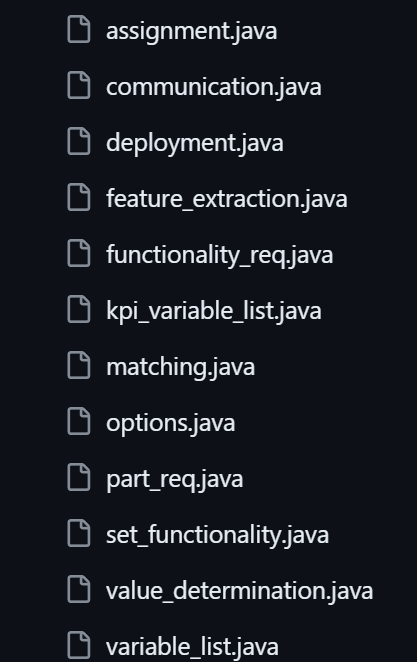
1. **TQC\_Malt:** This project contains the state chart implementation derived from Yakindu state chart modelling environment described in self-configuration approach strategy. The bin folder contains the compiled class files. The src-gen/malt folder contains the implementation of Yakindu state machine. The src folder contains MaltState package that has maltStateMain where the state chart can be executed. In guiTQCApp contains a State Machine package with some example classes on how the state chart can be encapsulated in state machine behaviors.
2. **guiTQCApp :** Contains a basic implementation of tool utility realization through JAVA FX framework. The behaviors can be defined in the State Machine package and leveraged through the app.

More on Yakindu State Chart and its implementation for self-configuration is detailed in relevant thesis chapter.

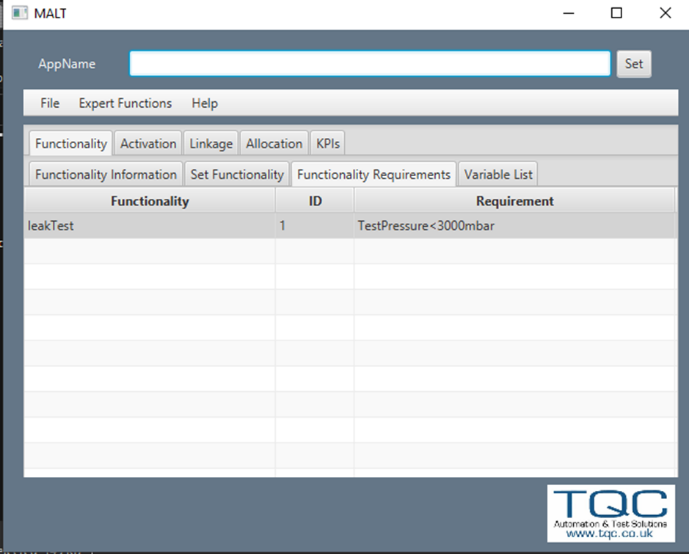
## Tool Utility for Functionality

The tool utility is developed and can be accessed through guiTQCApp project. The TQCMainScene.fxml contains the user interface definition for the app. The TQCMainSceneController class contains the controller classes for the app functions. These functions can be tied to state machines from previous section. The tool utility operates as a coordinator for the functionality execution. An example is present in the code.

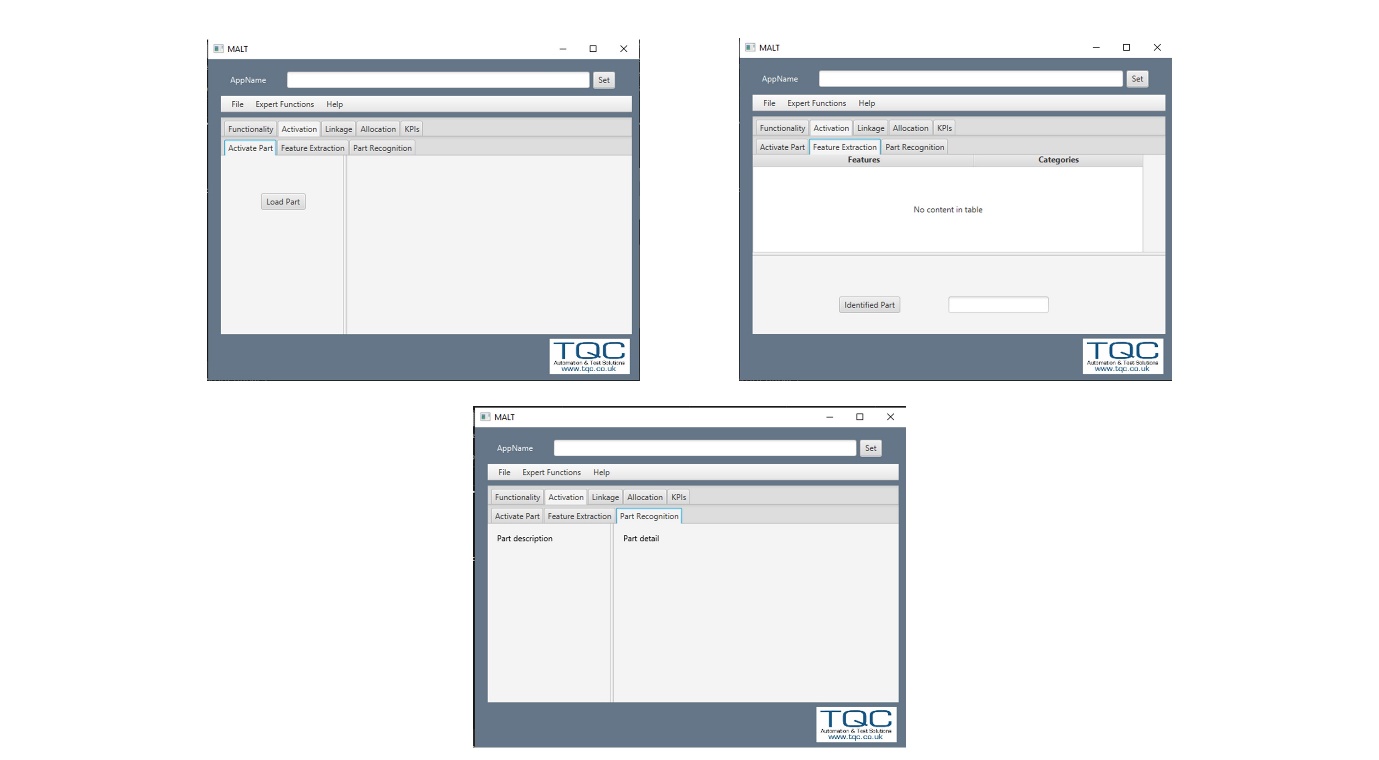
The information capture can be integrated into the guiTQCApp through relevant classes. The information capture is carried out through CAEX engine implementation.



As the information captured is integrated the app can be used to reflect that in real-time, along with coordinating functionality as per state machine behaviors.



The detail on each screen responds to the states in the state machine developed through yakindu environment.



The coordination of functionality execution is detailed in the respective PhD Thesis Chapters.

## Information Capture through CAEX Engine and AAS

**Folders:** CAEX\_Malt, CAEX\_test, CAEX\_testingwithoutinterface and CAEX3

The information capture is carried out through CAEX engine and provides computability to the AAS standard. The detail on mapping between CAEX engine and AAS is present in the respective thesis chapter.

Each folder is a separate JAVA project, with generated classes that can be used to interface with CAEX files. The CAEX files are generated through AAS Explorer, can be referenced through AML editor. The following steps can be followed:

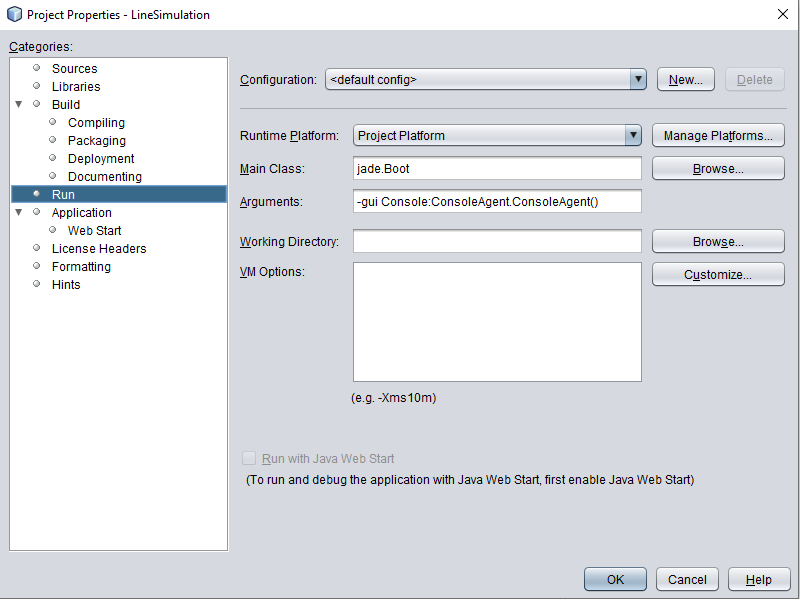
1. Information on modules in a production system can be mapped to the AAS submodels. Then whole AAS of the system can be exported to AML (CAEX) file.
2. The AML editor can be used to access the CAEX file for the system.
3. The folders contain code that can interact with the CAEX files through generated classes. The information can be extracted and updated.
4. The classes can be integrated within state machine behaviors for functionality coordination. Tool utility detailed before can be used together with these state machine behaviors.

A basic implementation example is present in each code project.

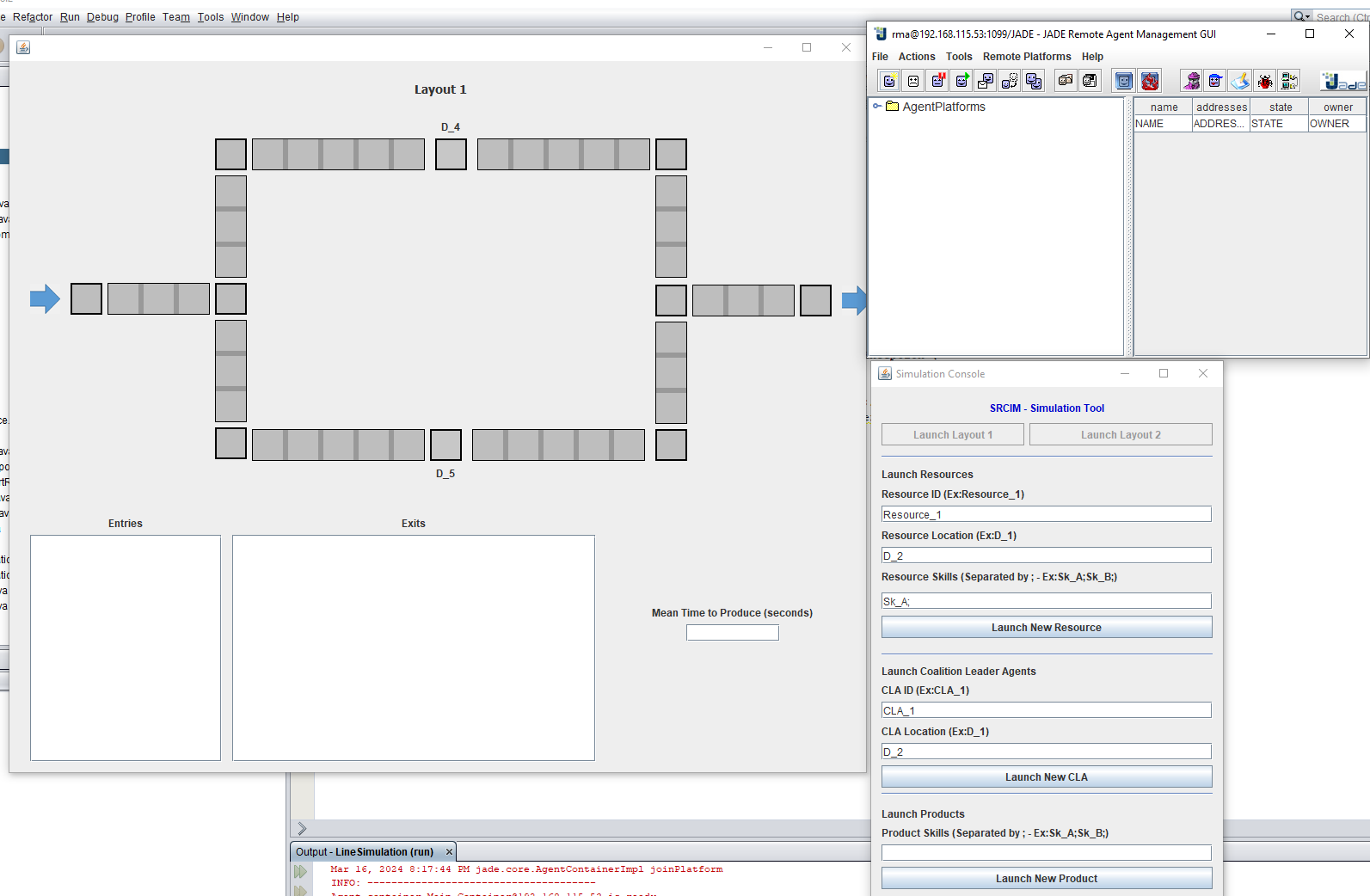
## Agent System Execution

**Folders:** Line Simulation, Test Simulation & Lib.

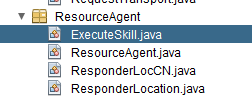
The Agent simulation can be started by run the Line Simulation Project with following configuration:



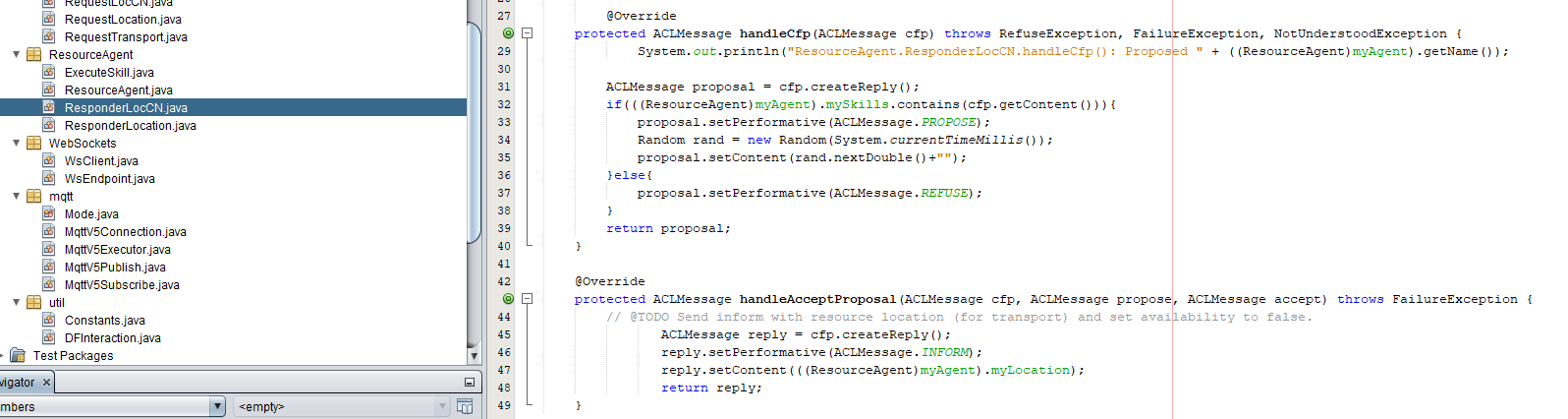
Once the simulation is started the location of resources can be defined through the console screen:



The functionality execution behavior can be triggered through Resource Agent package in Test Simulation. The functionality is executed through Execute Skill class. The Response from location is handled in Responder Location class.



Integrating business objectives (KPIs) and negotiating functionality execution is carried out through Contract Net Protocol (CNP) in ResponderLocCN class.



## ML Pipelines

**Folders:** pass\_fail\_endpoint, collab\_notebooks, endpointStrategy-main and cloud\_functions.

These folders contain the machine learning code for the thesis project. These code implementations rely (depending on application) on JAVA, Python and Google Cloud Platform. The main folders are as follows:

1. **Pass\_Fail\_Endpoint:** This folder contains the pretrained models for leak test and force test prediction. Should be in GCP Storage bucket of same name.
2. **Collab\_Notebooks:** Thisfolder containscode for machine learning applied to datasets. The pretrained models are generated through these.
3. **Cloud\_Functions:**  This folder contains the trigger scripts responsible that query values from pretrained models as they receive a http request. Should be configured in GCP Cloud Functions.
4. **Endpoint\_Strategy:** Contains modular ML endpoint implementation through Google’s Vertex AI. The detail is presented in the code and more insight available through Google official documentation.

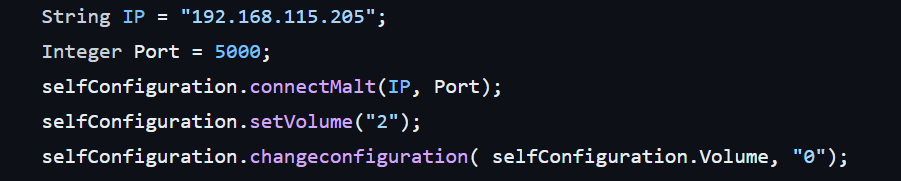
GCP is used to deploy the ML models in cloud storage as well as the trigger functions. The http calls can be integrated in the state machine behaviours and in agent systems to query predictions on settings. GCP official documentation to be followed for deployment. Detail on each implementation is present in respective folders.

## Real Time Control

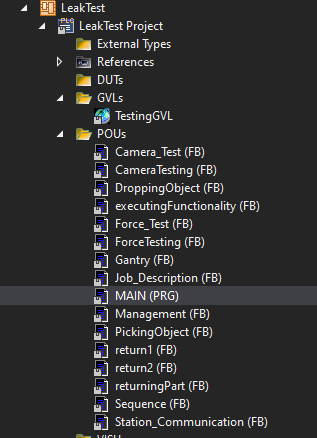
**Folders:** Self-ConfigurationMALT.zip and Testing1.rar

The archived files are the implementation of real time control (RTC) for physical industrial systems. The two industrial systems are Multi-Application Leak Tester (MALT), a bespoke leak tester, developed at TQC and PRIME test station respectively. The RTC is implemented as per following detail:

1. **MALT:** The testingMALT is the main class for the MALT self-configuration RTC. The KPIs can be provided and the configuration be changed through a single call function. Currently it is set to varying volumes. The configuration settings are determined through machine learning model hosted in cloud environment (GCP). This RTC can be integrated with the tool utility for functionality coordination under constraints.

****

1. **PRIME:** The PRIME test station code is a Beckhoff Twincat 3 project that can be loaded into Twincat 3 environment. The variables responsible for settings and functionality execution can be accessed through TestingGVL by OPCUA. The main execution happens through Main FB in the POUs. The configuration settings are determined through machine learning model hosted in cloud environment (GCP) actuated through trigger scripts on storage buckets. This RTC can be integrated with the tool utility for functionality coordination under constraints.

****

*NOTE: For any assistance contact the author of the Thesis at* [*hamood564@outlook.com*](mailto:hamood564@outlook.com)*.*