**System Architecture Design**

For Multiagent Control of Traffic Signals

Version 1.0

Submitted in partial fulfillment of the requirements of the degree of MSE

Bryan Nehl

CIS 895 – MSE Project

Kansas State University

Table of Contents

[1 Introduction 3](#_Toc321173998)

[2 References 3](#_Toc321173999)

[3 Architecture 3](#_Toc321174000)

[3.1 System Analysis 3](#_Toc321174001)

[3.2 System Context Diagram 4](#_Toc321174002)

[3.3 Component Design 5](#_Toc321174003)

[3.4 Component Interface Specification 6](#_Toc321174004)

[3.5 System Design Rationale 8](#_Toc321174005)

[3.6 High-Level Design 8](#_Toc321174006)

[3.7 Mid-Level Design 10](#_Toc321174007)

[4 Component Interaction 11](#_Toc321174008)

[4.1 Start up interaction 12](#_Toc321174009)

[4.2 Processing interaction 13](#_Toc321174010)

[4.3 Simulation end interaction 14](#_Toc321174011)

[4.4 Discovery Protocol 15](#_Toc321174012)

[5 USE/OCL Model 16](#_Toc321174013)

[5.1 Overview 16](#_Toc321174014)

[5.2 USE OCL Code 16](#_Toc321174015)

# Introduction

This document provides system design information for the MultiAgent Control of Traffic Signals (MACTS) system. This system is used to simulate agent based control of traffic light signals. This document covers the system components and component interfaces. However, it does not cover all of the interfaces methods in detail. A system analysis diagram as well as a high-level overview of the whole system is included in this document. Mid-Level design is also included for all of the components. A sequence diagram is included which shows how the system components interact during run time.

# References

1. “Vision Document” available at <http://people.cis.ksu.edu/~bnehl/>.

# Architecture

This section documents the system component design, the interfaces of those components and provides high-level design with rationale for design within the system context.

## System Analysis

Clearly describe the high level relationship between model elements

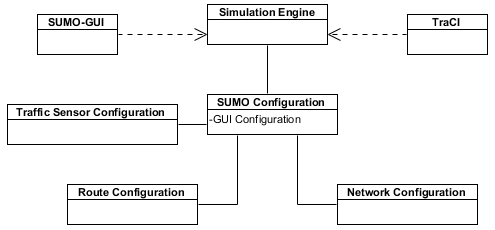


Figure System Analysis Diagram

Referring to Figure 1, the Simulation Engine takes care of the work of simulating the movement of the vehicles in the system. The SUMO-GUI is the front-end graphical user interface that displays the state of the simulation. TraCI is the TCP/IP interface to the simulation engine. TraCI is how external entities can interact with the simulation. The Simulation Engine relies on a SUMO Configuration file. The SUMO Configuration file includes specific settings information for the GUI as well as references or pointers to three other configuration files. The Traffic Sensor Configuration file contains information about sensors like the e1 inductor that are on the road network. The Route Configuration includes information about the routes that cars take. Details regarding the types of vehicles, vehicle distribution are specified. For the routes, the flow rates and probabilities are specified. The Network Configuration file is the result of running three files: Nodes, Edges and Connectors through the NETCONVERT utility. NETCONVERT is a SUMO utility. The Nodes, Edges and Connectors files detail where connections happen (nodes), streets are described by the edges and connectors handle the mapping from one edge to another at a junction node.

## System Context Diagram

This system context diagram shows how the components of the MACTS system interact with each other and with the external systems.

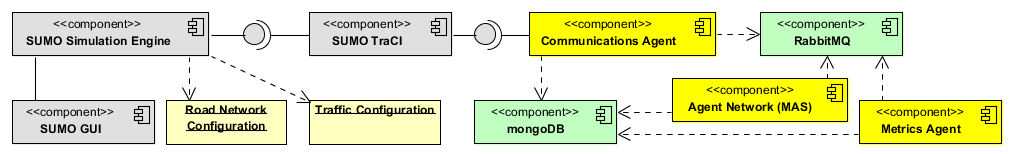


Figure System Context Diagram

In Figure 2, the System Context Diagram, grey components are SUMO components, the ivory components are SUMO configuration files created by me. The green components represent third party infrastructure servers. The yellow components (Communications Agent, Agent Network (MAS) and Metrics Agent) are the aspects that I will be creating. What we don’t see at this level is the possibility for multiple Agent Networks. That would come in to play when we have a MAS working at every intersection.

In Figure 3, the basic processing for a single simulation step is shown. Data is received from the simulator and sent to RabbitMQ. From there, there the Metrics Processing pulls data from its queue and does its own parallel operations. In the Analyze Data step, the MAS node planning agent uses sensor information that it received from specific queues. The planning agent then creates a suggested plan and sends it to the safety officer for checking that the command is safe. At that point, if the plan is safe, operation continues on the happy path sending the commands to the communications agent. Otherwise, the planning agent is informed that the plan isn’t safe.

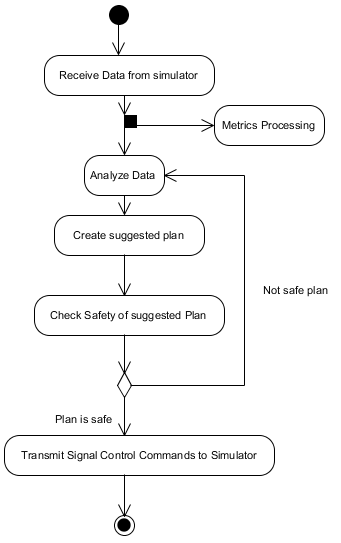


Figure Basic Processing for single simulation step

## Component Design

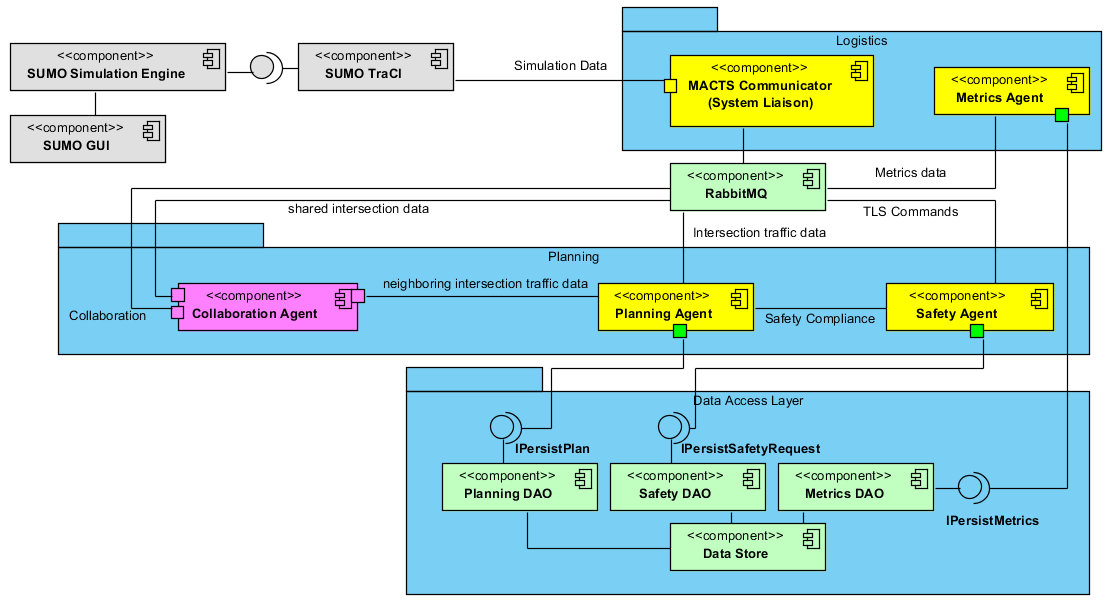


Figure MACTS with single MAS Node

In the Figure 4 the MACTS with a single MAS is portrayed. The grey components, SUMO Simulation Engine, SUMO GUI and SUMO TraCI are all part of the Simulation for Urban MObility software. The light green components RabbitMQ and DataStore are third party servers for message queuing and data persistence. The yellow components Communicator or System Liaison and the Metrics agent are part of the MACTS system. No matter how many MAS Nodes are in the system, there will only be one of each of these.

The communicator component has the responsibilities of: initiating a session, retrieving and publishing sensor data, retrieving and publishing metrics data, retrieving and submitting simulation commands, telling the simulation to proceed with the next step and finally of notifying all participants that the simulation has ended.

The components that constitute a MAS Node are enclosed in the light blue container. They are the yellow Planning Agent and the yellow Safety Agent. The pink component, Collaboration Agent is an optional part of the MAS Node. Concrete instances of the planning agent will propose a signal light plan given sensor data. The concrete safety agent will make the determination if the proposed plan is safe for the intersection.

In figure five we see a multiple MAS node system. Some lines from the second MAS node to the RabbitMQ and Data Store servers have been eliminated for clarity. If draw in, they would originate and terminate the same as the connections from the first MAS node. In a multi-node system with collaboration, the pink collaboration agents share or publish information about what is happening in this intersection with neighboring collaboration agents.

Because I know the topology of the test network, I know that there will at most be two MAS nodes operating simultaneously. If a collaboration agent is enabled, it will be configured to share relevant information with its neighbor. It will also be configured to receive shared information from a predetermined queue.

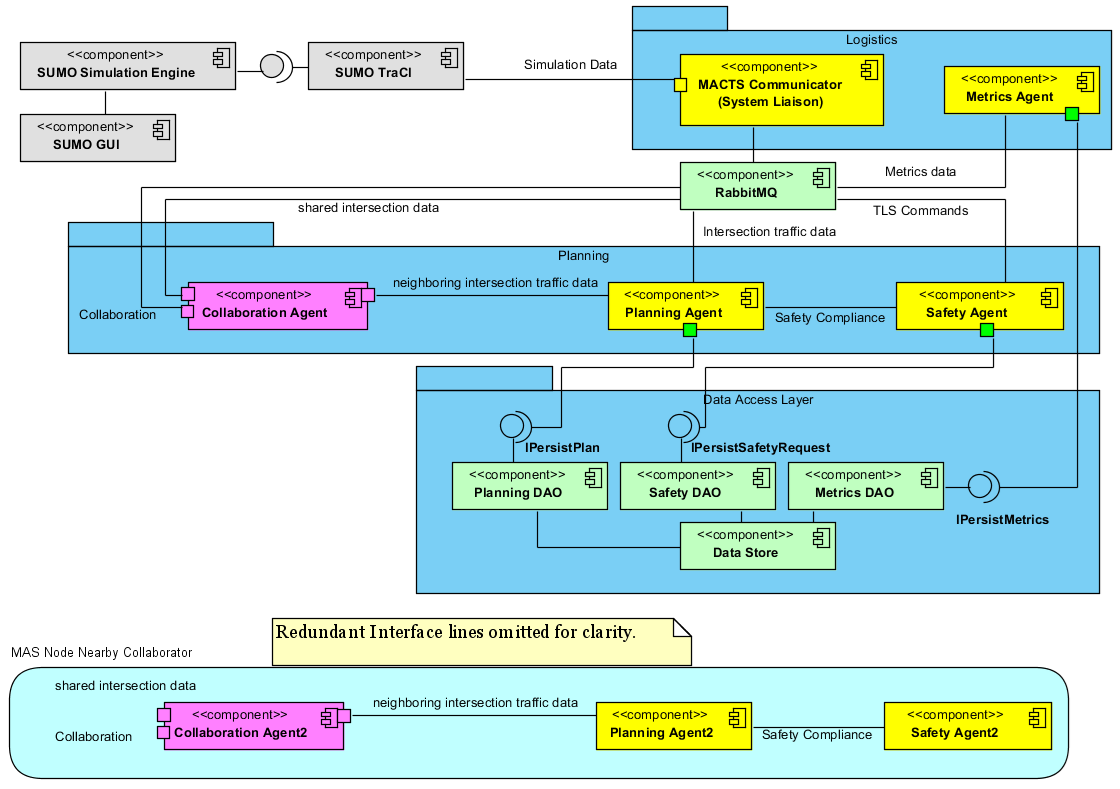


Figure MACTS with collaborating MAS Node

## Component Interface Specification

*Documentation of key interface members for system components here.*

*While there are traditional class interfaces in the project, MACTS interfaces are more service oriented application programming interfaces, SOA APIs. In the forthcoming Component Design Document I will expand on the message exchanges used, their purpose and the type of messages being passed.*

**IPersistPlan**

+persistPlan(simulationId : string, step : integer, agentId : string, plan : string)

**IPersistSafetyRequest**

+persistSafety(simulationId: string, step: integer, agentId: string, request:string,response:string)

**IPersistMetrics**

+persistStep(simulationId : string, step : integer, simulationStepMetrics : Metrics)

+persistSimulation(simulationId : string, aggregatedMetrics : Metrics, systemConfiguration : List<string>)

|  |  |
| --- | --- |
| **Signature** |  |
| **Purpose** |  |
| **Pre-Conditions** |  |
| **Post-Conditions** |  |

## System Design Rationale

There were several design criteria which led me to existing architecture. A key problem that had to be resolved was how to enable a distributed architecture to work with the direct connection, single client interface of TraCI. This design makes that possible by using a Communications Agent/System Liaison which interacts with the other parts of the system in a decoupled way through RabbitMQ message queues.

I chose to work with RabbitMQ and MongoDB because of their easy interfaces and ability to work with JSON documents. Also, since the core of this project isn’t messaging or data persistence this seemed like a good way to avoid reinventing the wheel.

I knew that I wanted to be able to test various types of agents as well. So, I wanted to be able to easily configure a network. This led me to a decoupled design where the agents communicate with each other through message queues. Because there is some shared behavior amongst agents, I thought it would be useful to have an abstract Agent class which could be used by the other types of agents. Planning Agent, Collaboration Agent and Safety Agent are all more specialized, yet still abstract classes that build upon the Agent class.

Why didn’t I have all of the planning agents going directly to the Communication Agent for shared information? Because I wanted to reflect a bit of reality in that at a given intersection there likely won’t be the connectivity back to a central all-knowing authority. The agent has to interact with its neighbors.

To obtain system metrics, TraCI must be used to query the simulation for a specific metric on every lane in the system. To me, it made the most sense for the Communications Agent to gather all of this information and publish it to its own queue. From there, the Metrics Agent gathers it, does any necessary analysis and aggregation and saves the data. The Metrics Agent stores the network configuration along with all of the aggregated metrics at the end of a simulation run.

## High-Level Design

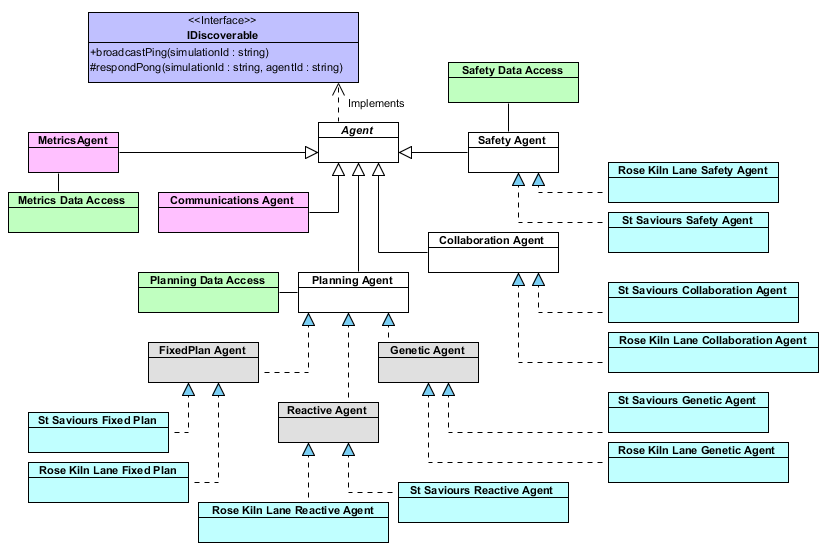


Figure High Level Class Diagram

In this high level view of the design we can see how it provides for reusability and scalability.

## Mid-Level Design

*Class Diagrams and discussion here*

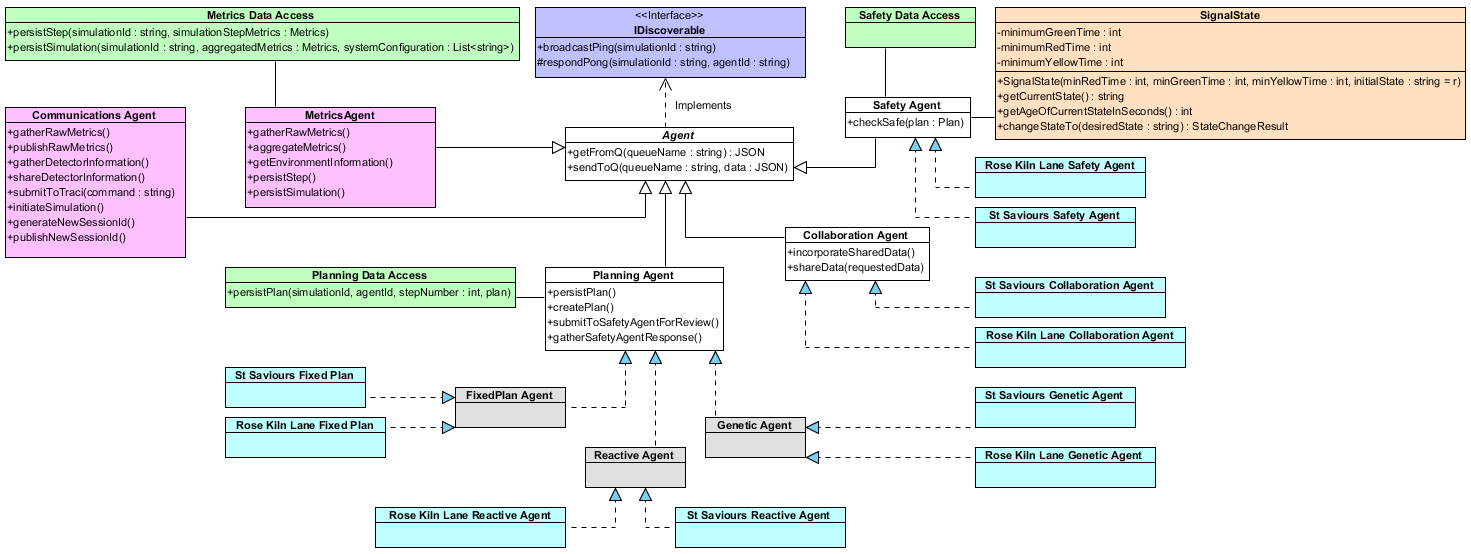


Figure Class Diagram

# Component Interaction

*In this section….*

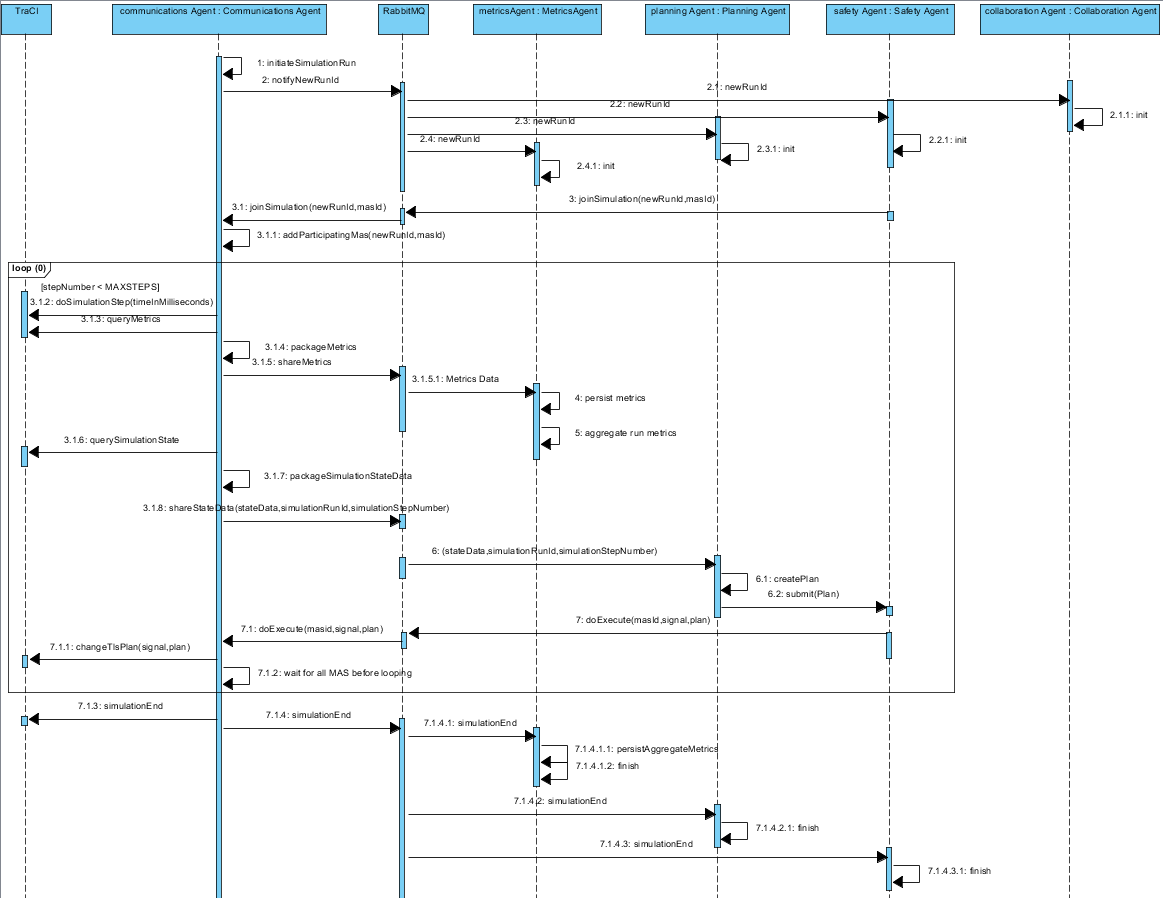
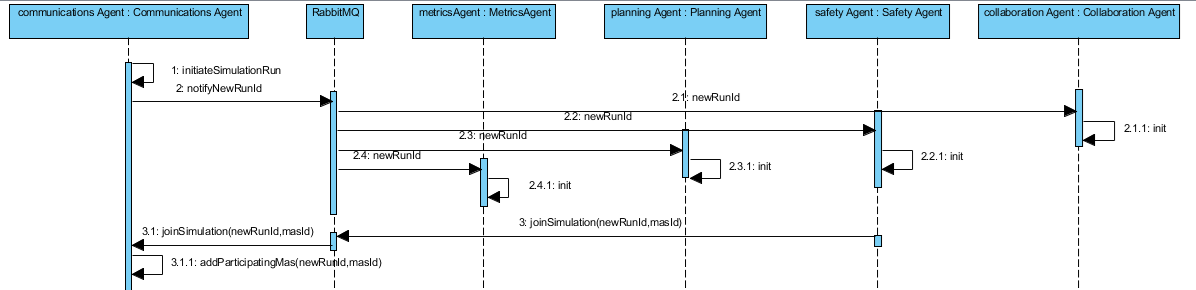


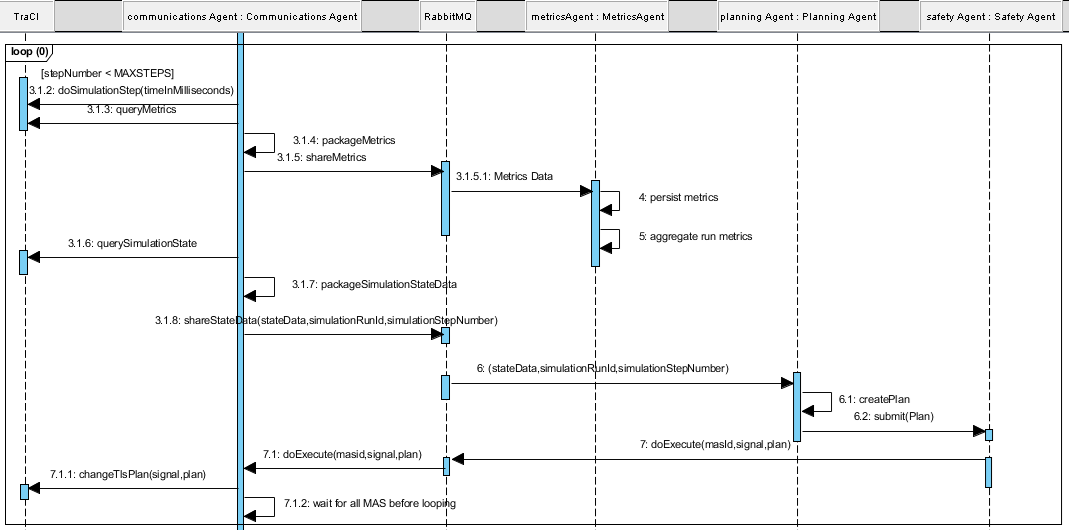
Figure Sequence diagram for process interactions

## Start up interaction



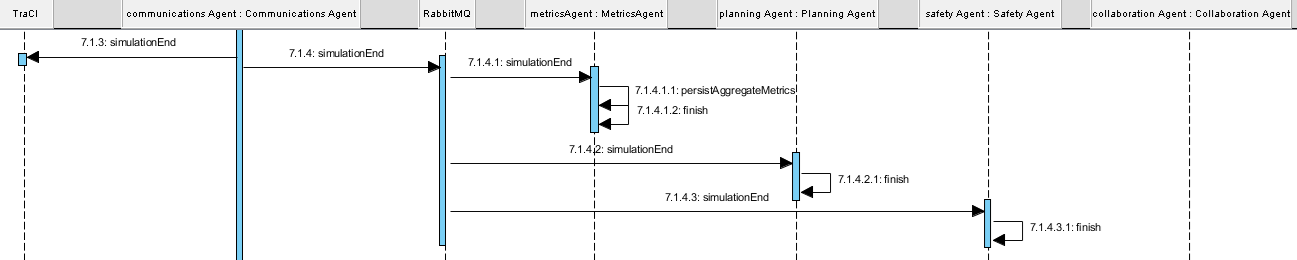
*Lorem ipsum*

## Processing interaction



*Lorem ipsum*

## Simulation end interaction



*Lorem ipsum*

## Discovery Protocol

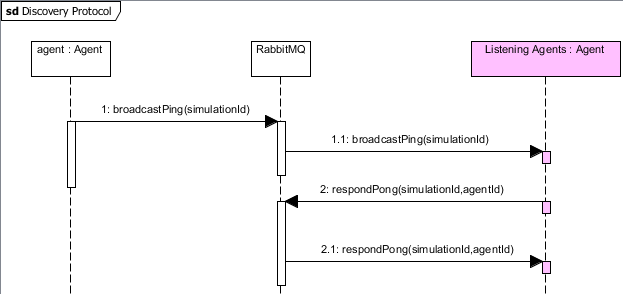


Figure 9 Discovery Protocol

In the discovery protocol, an agent sends a broadcast ping along with the current simulation Id to a discovery queue. All agents are monitoring the discovery queue. On “hearing” a ping, the listening agent responds with the simulation Id as well as its agentId. The agentId is made up of the type of Agent and the MAS Node Id. Any agent that is listening to the pong response queue will “hear” other agents on the network.

# USE/OCL Model

## Overview

This section will provide a formal specification that the safety agent enforces minimum time per light color and that the lights must change in a rotation of green, yellow, red. The system combines all traffic light signals at an intersection into a single command.

The interactions involved are Planning Agent sends plan to Safety Agent. Safety Agent evaluates.  If ok, the Safety Agent sends the plan on to the Communications Agent.  If not ok, the Safety Agent notifies the planning agent that the plan is not acceptable and the cause/reason why.

## USE OCL Code

-- CIS 895 MSE Project Formal Specification MACTS Architecture