Vision Document

For Multiagent Control of Traffic Signals

Version 3.0

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

The intent of this project is to create an efficient, scalable, modular Multiagent System (MAS) that controls traffic signals given sensor data. A white paper from the THALES group discusses at a high level using genetic algorithms with a MAS to improve traffic flow. That paper also provides a sample road network with traffic flow data that can be used for comparison.

The purpose of this project is to show a Multiagent System can be used to create a more positive experience for the driver and benefit the environment at the same time.

I intend to show improvements over a baseline typical timing based network with the following metrics: travel time, loss time (# of stops), fuel consumption, hydrocarbon production. To show these improvements I will model timing based behavior with a default times SUMO model. Then I will run simulations with simple reactive MAS, more intelligent reactive MAS, MAS that uses genetic algorithms and finally a goal-oriented MAS that uses mesh network based collaboration.

1.1 Motivation

Most traffic light systems today are strictly timing based. Traffic flow studies are required to create timing plans and atypical conditions cause problems. It is also very frustrating to have to stop for a red light when there is no opposing traffic.

1.2 Terms and Definitions

A software agent is a piece of software that interacts with its environment by sensing, reasoning, making decisions and then effecting a change on the environment.

Multiagent System (MAS) is made up of multiple software agents. Some MAS use communication and coordination between the agents. These are typically cooperative environments.

The Simulation for Urban MObility or SUMO is traffic simulation software that is geared towards micro-simulation.

TraCI is a TCP based interface that is used to interact with a SUMO simulation.

RabbitMQ is Advanced Message Queuing Protocol compliant message-oriented middleware.

MongoDB is a document store oriented database system.

1.3 References

SUMO, “Simulation for Urban MObility,” Sep. 2011; http://sourceforge.net/apps/mediawiki/sumo/index.php?title=Main_Page.

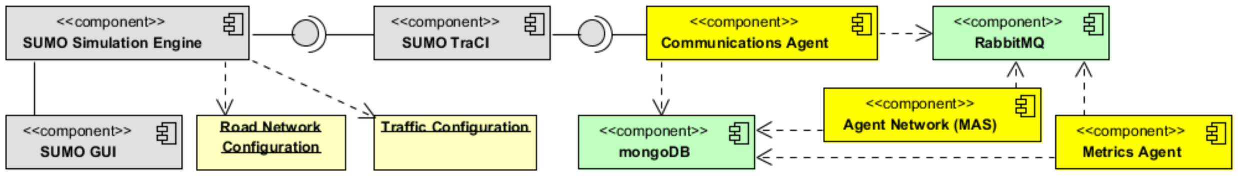
T. Masterton and D. Topiwala, “Multi-Agent Traffic Light Optimisation and Coordination,” white paper, Thales Group, Reference VCS081002, Issue 2, 2008.

2 Project Overview

2.1 Project Goal

The goal of this project is to create a multiagent system that is capable of traffic light signal control which results in an improved travel experience.

2.2 System Context



<<component>> SUMO Simulation Engine
<<component>> SUMO TraCI
<<component>> Communications Agent
<<component>> RabbitMQ
<<component>> SUMO GUI
Road Network Configuration
Traffic Configuration
<<component>> mongoDB
<<component>> Agent Network (MAS)
<<component>> Metrics Agent

Figure 1System Context Diagram

The above system context diagram shows the SUMO simulation engine which will run the micro simulation and provide the metrics I use to evaluate the various solutions. The pale yellow instances of Road Network Configuration and Traffic Configuration are the data files that I will set up for the simulation runs. The Traffic Configuration describes the vehicles and paths that traverse the specified road network configuration. The SUMO GUI provides a visualization of network activity. The SUMO TraCI component provides an application interface which permits external systems to interact with the simulation engine.

The pale green components, RabbitMQ and MongoDB, are third party components. RabbitMQ provides message queuing which will be used for agent communication. MongoDB will be used for any persistent storage needs, it is a document store.

The bright yellow components, the Communications Agent, the Metrics Agent and the Agent Network (MAS) are components that I create for this project. The Communications Agent will interact with the SUMO TraCI component. It will send commands to the simulation including traffic light signal commands. The agent will receive simulation information which it will forward to other agents using RabbitMQ. The Agent Network (MAS) represents a multiagent system. The MAS will make decisions regarding traffic light signal control, gathering of system metrics and collaboration with other intersection networks. The Metrics Agent is responsible for collecting and persisting simulation run metrics.

3 Project Requirements

3.1 Critical Use Cases

3.1.1 Displays Simulation

Description: The Simulation for Urban Mobility will be used to display the active simulation.
Pre-Conditions: A configured and running simulation.
Details: The system hosting the simulation run will be used to display the visualization of system activity. The host system may optionally also host a MAS.
Post Conditions: The simulation is displayed graphically.

Specific Requirements:

3.1.1.1 SR1 [Critical Requirement]

A road network configuration will be created for the simulation.

3.1.1.2 SR2 [Critical Requirement]

Traffic route configurations will be created for the simulation.

3.1.1.3 SR3

The SUMO graphical user interface (GUI) will display simulation activity.

3.1.2 Provides current simulation state data

Description: The TraCI component provides current simulation data to the System Liaison.
Includes: Executes TLS plan
Pre-Conditions: The TraCI component is up and running in conjunction with a simulation run. In addition the System Liaison which consumes the data is also operating.
Details: The system Liaison interacts with the simulation via TraCI to obtain information regarding the current simulation run.
Post Conditions: The liaison has obtained information, checks to see if the MAS has any plans or commands it wishes to submit and then gives an instruction to TraCI to continue with the next simulation step.
Specific Requirements:

3.1.2.1 SR4 [Critical Requirement]

The Liaison (see Use Case diagram in Figure 2.) requests information from the simulation.

3.1.2.2 SR4b [Critical Requirement

The Liaison creates a new run id and shares it with MACTS system.

3.1.2.3 SR5 [Critical Requirement]

The Liaison checks if there are any command requests or plan submissions from the MAS.

3.1.2.4 SR6 [Critical Requirement]

The Liaison submits the commands or plans if there were any provided.

3.1.2.5 SR7 [Critical Requirement]

The Liaison instructs the simulation to continue with the next step once all information has been obtained and commands sent.

3.1.3 Produces individual intersection state data

Description: The System Liaison produced individual intersection state data.
Pre-Conditions: The TraCI component is up and running in conjunction with a simulation run. In addition the System Liaison which consumes the data is also operating.
Details: The System Liaison will translate the incoming simulation data and publish data which corresponds to individual intersections.
Post Conditions: Intersection data is published to RabbitMQ topic queues which correspond to individual intersections.

Specific Requirements:

3.1.3.1 SR8 [Critical Requirement]

Translate incoming simulation data. Parse out information for individual intersections.

3.1.3.2 SR9 [Critical Requirement]

Publish the individual intersection data to RabbitMQ topic queues.

3.1.3.3 SR9b [Critical Requirement]

Gather and Publish metrics data to metrics queue.

3.1.4 Executes traffic light signal plan

Description: The System Liaison passes traffic light signal plans into the simulation engine.
Pre-Conditions: The TraCI component is up and running in conjunction with a simulation run. In addition the System Liaison which communicates with TraCI is also operating.
Details: The MAS has provided the System Liaison with a Traffic Light Signal (TLS) plan which it communicates to the TraCI.
Post Conditions: The SUMO uses the new traffic light signal plan.
Specific Requirements:

3.1.4.1 SR10

See SR5 above. (The Liaison checks if there are any command requests or plan submissions from the MAS.) The Liaison will check a RabbitMQ TLS command queue. The plan will consist of a script of instructions that the Liaison can send to TraCI.

3.1.4.2 SR11

See SR6 above. (The Liaison submits the commands or plans if there were any provided.) The Liaison plays the script of commands, sending them to TraCI.

3.1.5 Information about current intersection status is shared

Description: Neighboring collaboration agents share intersection information.
Includes: Produces traffic data to share with neighboring intersections
Pre-Conditions: A RabbitMQ server which is accessible to both agents is running.
Details: Neighboring collaboration agents share intersection information via a RabbitMQ queue.
Post Conditions: Intersection information has been shared.
Specific Requirements:

3.1.5.1 SR12

Collaboration agents include a discovery interface which is used to self-identify when an agent sends a broadcast querying for agents that are associated with intersections that send traffic into their intersection or receive traffic from their intersection. Agents respond with a list of queues where their outbound traffic information can be found.

3.1.5.2 SR13

Collaboration agents share information regarding traffic leaving their intersection by putting it in distribution queues.

3.1.6 Produces data to share with neighboring intersections

Description: The Collaboration Agent produces intersection data to share with neighboring agents.
Pre-Conditions: A simulation is currently running and the Collaboration Agent has knowledge of current intersection activities.
Details: The collaboration agent examines the current state of the intersection and creates a document consisting of information regarding traffic flow out of the intersection.
Post Conditions: A document consisting of information regarding traffic flow out of the intersection is produced.
Specific Requirements:

3.1.6.1 SR14

The Collaboration agent creates information regarding traffic leaving their intersection.

3.1.7 Create traffic light signal plan

Description: Either the planning agent makes no changes to the operational plan or it sends a plan with commands to change the TLS operation.
Includes: Incorporate data that was shared into planning
Pre-Conditions: A running simulation with which the System Liaison is interacting.
Details: After observing
Post Conditions: Either no action taken or a plan is submitted to the Safety Agent.
Specific Requirements:

3.1.7.1 SR15 [Critical Requirement]

The planning agent examines incoming data and creates a new TLS plan.

3.1.7.2 SR16 [Critical Requirement]

The planning agent submits the plan to the Safety Agent for review.

3.1.8 Incorporate data that was shared into planning

Description: Use data that is provided by neighboring collaboration agents in the planning of TLS control.
Pre-Conditions: A running simulation with neighboring MAS that have collaboration agents that are sending there intersection data.
Details: The agent can adjust the traffic light plan/state based on the input provided from neighboring agents.
Post Conditions: Data incorporated into traffic light signal plan.
Specific Requirements:

3.1.8.1 SR 17

The Planning agent incorporates data that was shared from other collaboration agents regarding traffic that is flowing into this current intersection.

3.1.9 Verify plan is safe

Description: We want to make sure that the MAS are only creating TLS scenarios that are safe.
Pre-Conditions: A running simulation.
Details: The agent examines the TLS plans to verify that there are no states where unsafe conditions are created. For example, the agent would not approve a plan which includes a green

left turn arrow at the same time as a solid green for traffic directly on the other side of the intersection.
Post Conditions: Acceptance or rejection of plan.
Specific Requirements:

3.1.9.1 SR 18 [Critical]

The safety agent examines the TLS plan to verify that there are no simultaneously active paths that will cross each other in such a way as to create an unsafe condition (foes).

3.1.9.2 SR 18b [Critical]

The safety agent enforces minimum times per light color. For example, a light cannot be green for one second then switched to yellow.

3.1.9.3 SR 18c [Critical]

The safety agent enforces proper progression of light changes. That is a light cannot be switched from green to red. The progression must be a rotation of: green, yellow, red.

3.1.9.4 SR 19

If the plan is not safe, it lets the planning agent know and provides the reason why.

3.1.10 Submit plan to be executed

Description: The Safety Agent sends the verified safe plan to the System Liaison to execute.
Pre-Conditions: A verified safe plan provided by the Safety Agent. There should also be a running simulation with an active System Liaison.
Details: The Safety Agent has verified that the plan is safe so it submits it to the System Liaison to run. The plan is put into the TLS command queue for the System Liaison to pick up and execute.
Post Conditions: The verified safe plan has been sent to the TLS command queue. The System Liaison has picked up the plan and executed it.
Specific Requirements:

3.1.10.1 SR 20 [Critical]

Submit verified safe plan to TLS command queue.

3.1.11 Gather Simulation Metrics

Description: The system needs to be able to gather metrics about the simulation in order to evaluate MAS effectiveness.
Pre-Conditions: The TraCI component is up and running in conjunction with a simulation run. In addition the System Liaison which communicates with TraCI is also operating.
Details: The system will gather the metrics either directly or will gather the data necessary for computing the metrics. The metrics to be collected are: average travel time, loss time (# of stops), fuel consumption and hydrocarbon production.
Post Conditions: Simulation Metrics are captured and persisted.
Specific Requirements:

3.1.11.1 SR 21 [Critical]

The Metrics Agent receives simulation metrics from TraCI via the System Liaison.

3.1.11.2 SR 22 [Critical]

Do any internal processing necessary for computing metrics.

3.1.11.3 SR 23 [Critical]

Upon simulation run completion, persist the run metrics to MongoDB. Corresponding network configuration information will be stored with the metrics.

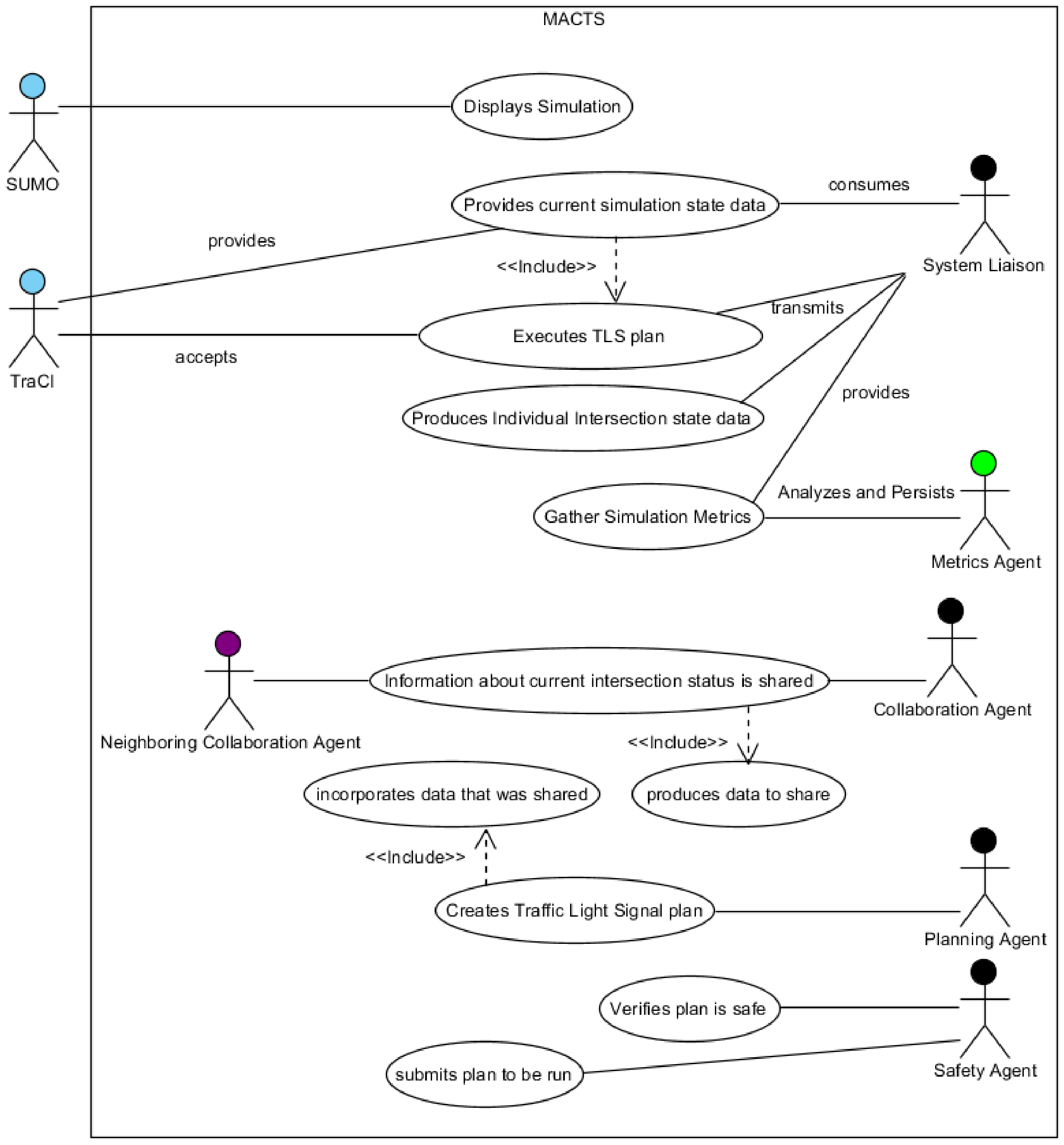


Figure 2Critical Use Cases

3.2 System Activity

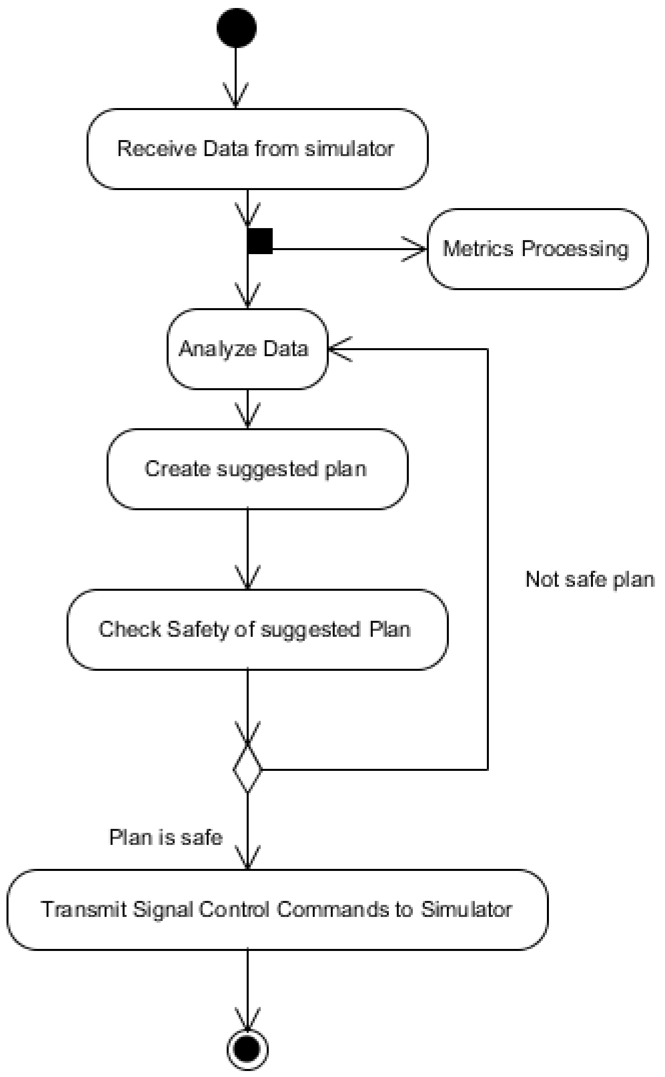

Receive Data from simulator
Metrics Processing
Analyze Data
Create suggested plan
Not safe plan
Check Safety of suggested Plan
Plan is safe
Transmit Signal Control Commands to Simulator

Figure 3UML Activity Diagram for single simulation iteration

The activity diagram above depicts the process that will happen during a typical simulation iteration. I do anticipate that a plan will not be created for every iteration. Once the agent has received enough data to create a plan it does. The plan is checked by the Safety Agent. If the plan is safe it is sent to the Communications Agent to push into the simulation engine via TraCI. If the plan is not safe then that feedback is provided to the agent network that submitted the plan.

4 Assumptions

To be able to run this solution the user is running a computer with Windows 7 64bit Professional edition and will have the following components installed:
Simulation for Urban MObility (SUMO) version 0.14.0, filename sumo-winbin-0.14.0.zip.
Python version 2.7.
Erlang version OTP R14B01. Erlang is a prerequisite for RabbitMQ.
RabbitMQ server version 2.4.1.
MongoDB for Windows version 2.0.2, filename mongodb-win32-x86_64-2.0.2.zip.

5 Constraints

This system will not attempt to model the behavior of individual cars nor will it visualize system behavior. Those aspects are left to the simulation engine SUMO.

6 Environment

6.1 Purpose

The purpose of this section is to describe the development platform, tools, environment practices and processes that will be employed for the Multi-Agent Control of Traffic Signals project.

6.2 Hardware

I will use a Sony laptop as the prime development station. The laptop has an Intel core I7 processor and sufficient RAM to run the simulations. The computer can support the development environment as well as running the simulation. Additional machines may be used to show the distributed nature of the MACTS system. An External Hard Drive will be used for local but off computer repository. A flash drive will be used for OneNote file sync.

6.3 Tools

This section describes tools (software) that will be used for the project and supporting activities.

6.3.1 Process Support

Google docs will be used for maintaining the engineering notebook, for creating the work break down structure, for capturing information about references and for storage of guides, research & reference documents. Google Calendar will be used for scheduling work. The open source tool GanttProject will be used for creating the Gantt chart. OneNote is used for collecting project notes and for putting together working documents. Visual Paradigm for UML will be used for creating UML diagrams and when possible for generation or reverse engineering of application source code. MS Word 2010 will be used to create the documentation. Draft documents will be exported in PDF format.

6.3.2 Development

The Simulation for Urban Mobility or SUMO will be the simulation engine that I interface with.
PyCharm is an integrated development environment that will be used for Python 2.7 coding.
Python 2.7 was chosen since Python 3 is still new enough that I was having difficulty gathering the necessary interface modules.
RabbitMQ is a message queuing server which I will use for inter-agent communication. The Python module pika will be used to work with RabbitMQ.
MongoDB is a document store and will be used for persisting agent configuration information

and for any knowledge base needs. The Python module pymongo is used to work with MongoDB.

6.3.3 Testing

For unit testing, PyUnit the unittest module will be used. If mock objects are needed, mockito-python will be used. However, if mockito-python proves to not be suitable, I'll switch to Michael Foord's Mock. PyCharm's internal code inspection feature will be used to check for common coding errors. To see code coverage of unit tests, Ned Batchelder's Coverage.py module will be used. PyMetrics will be used for code analysis, particularly cyclomatic complexity.

6.3.4 Version Control

The distributed version control system git will be used for all project documents. That is, code and supporting portfolio documents will all be included in to the repository. I will update the git repository on the KSU CIS server, a local on machine repository as well as an external on hard drive repository during development.

6.4 Process

6.4.1 Development

I will create UML diagrams, unit tests and code to meet requirement acceptance criteria. Before committing code to the repository I will run coverage tests and code metrics. I will seek to have 100% coverage of application logic code. I will also strive for a low cyclomatic complexity. As features/requirements are completed, they will be committed.

6.4.2 Releases

There will be a weekly commit to external hard drive and to the KSU CIS server of all current work. Version numbers of Microsoft Word created documents will have their version numbers updated to correspond with the presentations. PDF versions of the Word documents will also be created at the same time. They will be named to relay what presentation they correspond with.

A weekly progress report will be made to my advisor in the form of an email.