International Journal of Agent Technologies and Systems

October-December 2012, Vol. 4, No. 4

Table of Contents

Editorial Preface

Research Articles

IJATS is currently listed or indexed in: ACM Digital Library; Bacon's Media Directory; Cabell's Directories; DBLP; GetCited; Google Scholar; INSPEC; JournalTOCs; MediaFinder; The Standard Periodical Directory; Ulrich's Periodicals Directory

On Modeling and Verification of Agent-Based Traffic Simulation Properties in Alloy

Junia Valente, Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA

Frederico Araujo, Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA

Rym Z. Wenkstern, Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA

ABSTRACT

The advances in Intelligent Transportation Systems (ITS) call for a new generation of traffic simulation models that support connectivity and collaboration among simulated vehicles and traffic infrastructure. In this paper we introduce MATISSE, a complex, large scale agent-based framework for the modeling and simulation of ITS and discuss how Alloy, a modeling language based on set theory and first order logic, was used to specify, verify, and analyze MATISSE's traffic models.

Keywords: *Alloy, Formal Specification, Intelligent Transportation Systems (ITS), Multi-Agent Systems, Traffic Simulation, Verification*

1. INTRODUCTION

For the past twenty years, Intelligent Transportation Systems (ITS) have been considered as possible solutions for traffic safety and congestion problems. An ITS is defined as "the application of advanced sensor, computer, electronics, and communication technologies and management strategies in an integrated manner to increase the safety and efficiency of the surface transportation system" (Meyer, 1997). The work presented in this paper is based on a novel, multilayered integrated ITS for safety improvement and congestion reduction (Boyraz et al., 2009a; Wenkstern, Steel, Daescu, Hansen, & Boyraz, 2009a). In this ITS infrastructure traffic is viewed as a bottom-up phenomenon that is the consequence of individual decisions at the micro-level, and traffic management as a top-down activity that is the result of decisions taken at the macro-level. Both macro- and micro-levels consist of multi-agent based infrastructures where autonomous traffic entities continuously communicate and interact with each other to achieve traffic safety and efficiency goals. Even though some of the

DOI: 10.4018/jats.2012100103

proposed ITS components have already been implemented, the overall infrastructure is still in its conceptual phase.

Given the critical role of interactions among ITS components and their independent decision making capabilities, it is essential to simulate traffic scenarios under nominal and extreme conditions before deploying the physical infrastructure on roads and highways.

MATISSE (**M**ulti-**A**gent based **T**raff**I**c **S**afety **S**imulation syst**E**m) is an agent-based "tailor made" simulation framework designed to provide a platform for the execution of such scenarios. MATISSE provides means to analyze and evaluate different ITS configuration, collaboration, and control strategies. Before embarking on the full-scale development of this large-scale, distributed, multi-agent based simulation framework, the specification and validation of MATISSE's properties proved to be necessary.

Alloy is a modeling language based on set theory and first order logic that has been used in both industry and academia to validate a wide variety of systems (Coppit & Sullivan, 2000; Dolby, Vaziri, & Tip, 2007; Jackson & Vaziri, 2000). The language has a simple and concise syntax that comes with a powerful, integrated tool for compiling and analyzing models.

The purpose of this paper is to present a formalization of the MATISSE model in Alloy, and discuss how the model's core properties are verified using Alloy's Analyzer. In particular, we discuss an approach to produce execution traces from the specification. These traces serve two purposes: they allow for a thorough analysis and evaluation of the traffic model; and demonstrate the suitability of MATISSE for the simulation of ITS scenarios.

In the following section we give an overview of traffic simulation systems. In Section 3 we briefly present the proposed ITS and MATISSE's high level architecture. In Section 4 and section 5 we discuss how Alloy has been used to specify, verify, and analyze MATISSE's model. In Section 6 we present an evaluation of the approach. Finally, in Section 7 we give an overview of related works.

2. TRAFFIC SIMULATION

There are two major approaches to simulate traffic scenarios. Macroscopic models (Babin, Florian, James-Lefebvre, & Spiess, 1982; Lieu, Santiago, & Kanaan, 1992) describe traffic as a physical flow of fluid and make use of mathematical equations relating macroscopic quantities (e.g., traffic density, flow rate and average velocity). These models assume rational driving behavior and fairly consistent traffic streams and thus are unfit to model real traffic operations.

In contrast, microscopic models consider the characteristics of individual traffic elements (e.g., vehicles, traffic lights, traffic signals, driver behavior) and their interactions. Typical microscopic models are based on analytical techniques such as queuing analysis and shock-wave analysis (Helbing & Tilch, 1998). They assume traffic elements with predefined behavioral models. This is a limitation since realistic traffic simulation scenarios call for the modeling of unexpected behavior and unforeseen environmental conditions. The multi-agent paradigm alleviates this limitation by providing means to address non-deterministic behavior in non-deterministic, unpredictable environments.

Over the last decade, a large number of agent-based traffic simulation systems have been proposed. Some focus on specific small scale traffic problems such as driver behavioral modeling, tactical driving, and intersection management (Dresner & Stone, 2008; Rossetti & Liu, 2005; Sukthankar, Hancock, & Thorpe, 1998) while others attempt to tackle complex large scale traffic scenarios (Balmer et al., 2009; Cetin, Nagel, Raney, & Voellmy, 2002; Galland, Gaud, Demange, & Koukam, 2009). In this section we restrict our discussion to those that best compare to MATISSE, namely MatSim (Balmer et al., 2009), and Transims (Cetin et al., 2002).

MatSim (Balmer et al., 2009) is an agent-based framework for modeling transport demand. MatSim represents individual travelers as agents endowed with predefined plans. These agents follow a utility based strategy to

determine their optimal daily plan. Interactions among agents are implicitly encoded into the agent's utility function. In its current version, traveler agents cannot directly interact with other agents. In addition, agents are not capable of perceiving their environment dynamically. They act upon global environmental knowledge seeded at initialization time.

Similarly, Transims (Cetin et al., 2002) is a large-scale microscopic simulation system for transportation planning and congestion evaluation. In Transims travelers are modeled as agents which can walk, drive cars, or use buses. Traveler agents can decide which plan to select depending on their current state but they cannot dynamically perceive their environment. It is also unclear whether they can interact with other agents. Transims' environment is static and fully observable, thus reducing its capabilities to model complex and realistic scenarios.

Our work enhances the conventional urban traffic simulation by proposing a multi-agent based framework that simulates macro- and micro-level traffic entities and their interactions within and across levels. The unique characteristics of MATISSE are: 1) the simulation environment is open, i.e., non-deterministic, dynamic, inaccessible and continuous (Russell & Norvig, 1995). The environment has mechanisms that allow the simulation of event propagation. 2) The agents are not given global environmental knowledge to act upon. They dynamically perceive their surroundings through various senses. 3) At run-time, the user can change the properties of the simulated agents (e.g, driver "awake" to driver "asleep", disable agent sensors) and the environment (e.g., change the laws that govern the environment) without interrupting the simulation. To the best of our knowledge, no other existing framework offers this integrated set of features.

A recent system called JaSim (Galland et al., 2009) was developed along the same premises as MATISSE. Even though it shares the same environment structure and similar agent perception mechanisms, it lacks the advanced simulation features of event propagation and dynamic property modification discussed above.

3. OVERVIEW OF ITS AND MATISSE

In this section, we briefly present the main components of the proposed ITS and discuss MATISSE's architecture. More detailed discussions on these topics can be found in (Boyraz et al., 2009a; Wenkstern et al., 2009a; Wenkstern, Steel, & Leask, 2009b).

3.1. Elements of a Novel ITS

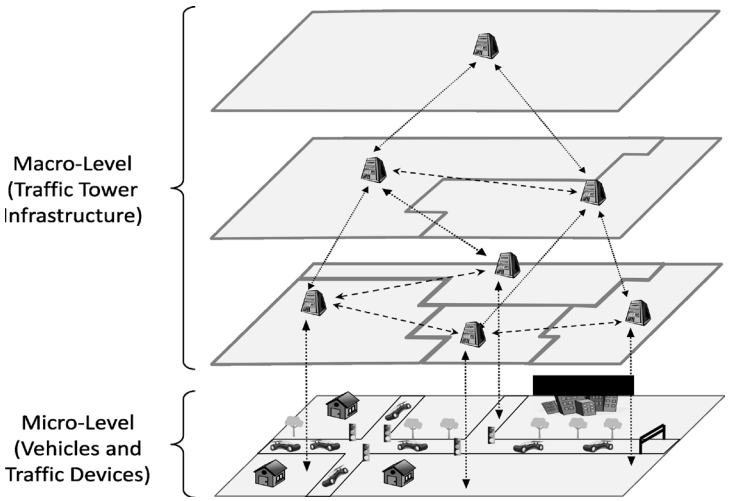
The proposed ITS aims at enforcing communication, interaction, and collaboration between various types of elements defined at various levels of abstraction. In the remainder of this paper we will use the word "micro-level" element to refer to an entity that has very limited knowledge of the state of the world. In contrast, a "macro-level" element refers to one that is aware of a larger portion of the world.

The infrastructure is based upon two underlying concepts:

- In order to manage a large environment efficiently, it is necessary to partition the space into smaller defined regions called *traffic area*;
- Each traffic area is assigned a *tower*. A tower is required to: 1) autonomously manage environmental information about its traffic area; 2) be aware of the traffic elements (e.g., vehicles, traffic devices) located in its defined area; 3) be able to interact with local traffic elements to inform them about changes in their surroundings; 4) be able to communicate with other towers to inform them of external events.

In order to manage traffic information efficiently, traffic towers are organized as a hierarchy (see Figure 1). This structure is particularly important for the case when towers need a higher level of knowledge to properly manage their traffic areas. For example, if congestion is caused by an accident in an area, and the micro-level information is insufficient for the tower to determine the best exit route

Figure 1. ITS super-infrastructure



for its local vehicles, it will communicate with a higher level traffic tower to obtain a broader image of the traffic.

The micro-level entities are classified in two categories:

- **Mobile Context-Aware Intelligent (CAI) vehicles** (Boyraz, Yang, Sathyanarayana, & Hansen, 2009b). These are vehicles equipped with devices that allow them to 1) monitor the driver's behavior in order to prevent possible accidents; 2) communicate with other vehicles and traffic devices; and 3) interact with the traffic tower infrastructure to obtain traffic information and guidance in real time;
- **Stationary Context-Aware Intelligent (CAI) traffic devices.** These include traffic lights, traffic collection devices, and relay units. They serve the purpose of improving safety and traffic flow on roads and highways by providing information about the physical traffic infrastructure and congestion condition. Traffic lights are equipped with adaptive systems that allow them to 1) interact with the traffic tower infrastructure to obtain traffic information in real time, 2) communicate with vehicles for intersection coordination, and 3) communicate with other traffic light controllers to improve traffic flow when necessary. Traffic collection devices are used on highways to collect information about traffic, and communicate the information to the traffic management system for further analysis (e.g., identification of a drunk driver on the highway). Relay units are used to pass on information between the various communicating entities when the physical distance is too great.

3.2. MATISSE Architecture

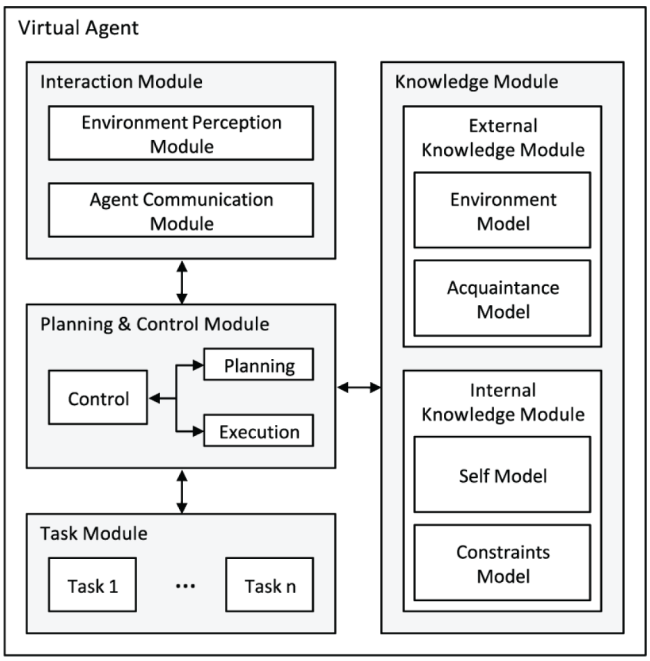
MATISSE is a "tailor made" multi-agent based simulation platform designed to specify and execute simulation models for the above-mentioned ITS. We define an agent as a software entity which (Mili, Steiner, & Oladimeji, 2006): 1) is driven by a set of tendencies in the form of individual objectives; 2) can communicate, collaborate, coordinate and negotiate with other agents; 3) possesses resources of its own; 4) executes in an environment that is partially perceived; 5) possesses skills and can offer services. A *virtual agent* is an application specific agent that represents a real world concept (e.g., vehicle, traffic device).

As shown in Figure 2, a virtual agent consists of four main modules (Mili et al., 2006). The **Interaction Module** handles an agent's interaction with external entities and separates environment interaction from agent interaction. The *Environment Perception Module* contains various perception modules emulating the agent's senses and is responsible for perceiving information about an agent's environment. The *Agent Communication Module* provides an interface for agent-to-agent communication. The **Knowledge Module** is partitioned into *External Knowledge Module (EKM)* and *Internal Knowledge Module (IKM)*. The EKM serves as the portion of the agent's memory dedicated to maintaining knowledge about entities external to the agent, such as acquaintances and objects situated in the environment. The IKM serves as the portion of the agent's memory dedicated for keeping information that the agent knows about itself, including its current state, physical constraints, and social limitations. The **Task Module** manages the specification of the atomic tasks that the agent can perform and the **Planning and Control Module** serves as the brain of the agent; it uses information provided by the other modules to plan, initiate tasks, make decisions, and achieve the agent's goals.

MATISSE defines virtual agents for each micro- and macro-level element used in the ITS. *Vehicle agents* simulate the behavior of human drivers; have individual goals such as arriving at some destination in a reasonably short time; influence other agents such as turning signals and changing lanes; and are governed by environmental norms and constraints such as speed limits and traffic signals. *Traffic light* and *traffic collection agents* are aware of and influence nearby vehicles; are able to perceive and adapt to changing conditions; and work collaboratively to achieve certain objectives. Finally, *traffic tower agents* autonomously manage and control their traffic area, including the vehicles and traffic devices they enclose.

In addition to these virtual agents, and for software design purposes, it is necessary to introduce two design related concepts: a *cell* is a repository that encompasses all informa-

Figure 2. Virtual agent architecture



tion related to a traffic area. A *cell controller* is a special purpose agent whose main role is to consistently provide virtual agents located within its cell with a correct perception of their surroundings. This is a complex and critical role in any realistic simulation. More information on this topic can be found in (Mili & Steiner, 2007). It is important to note that a cell controller does not correspond to a real world concept.

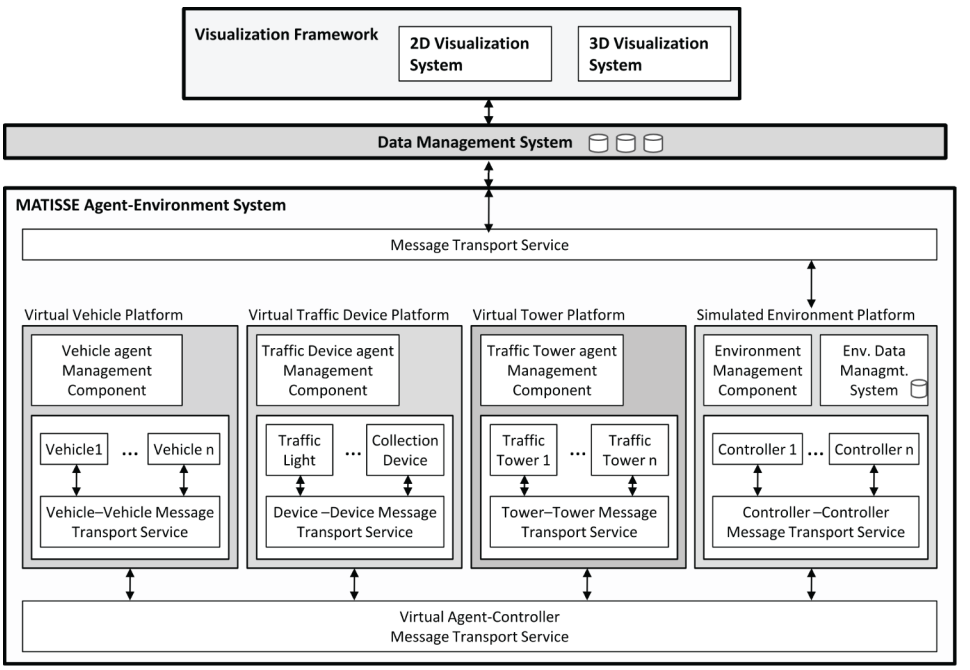
3.2.1. High Level Architecture

As shown in Figure 3, MATISSE's high level architecture includes three main components: the Agent-Environment System (AES) creates simulation instances; the Data Management System (DMS) stores and processes information collected from the AES; and the Visualization Framework receives information from the DMS and creates 2D or 3D images of the simulation.

3.2.2. MATISSE's Virtual Agent Platforms

The four types of agents identified by MATISSE are naturally managed by four distinct agent platforms within the Agent-Environment System component. The **Virtual Vehicle Platform** manages mobile agents that represent vehicles. Vehicle-agents are created by the *Vehicle-Agent Management Component*, and vehicle-agents communicate with each other through the *Vehicle-Vehicle Message Transport Service*. The **Virtual Traffic Device Platform** manages stationary agents that represent traffic lights, relays and information collection devices. The *Traffic-Device-Agent Management Component* creates and manages traffic-device-agents within the simulation while *Device-Device Message Transport Service* handles communication between these stationary traffic-agents. The

Figure 3. Matisse high level architecture



Virtual Tower Platform creates and manages the hierarchical infrastructure of traffic-tower-agents. Finally the **Simulated Environment Platform** creates and manages cell controllers. The *Environment Agent Management Component* creates cell controllers, assigns them to a cell, and maintains the cell controller hierarchy for the simulation.

4. SPECIFYING MATISSE IN ALLOY

Due to the scale and complexity of the simulation architecture, from a software engineering perspective, we found it necessary to formally specify and validate various simulation properties before starting the implementation of MATISSE. In this section we briefly introduce the Alloy language (Jackson, 2002) and present a specification of the simulation properties of MATISSE in Alloy.

4.1. Overview of Alloy

In the past two decades, several formalisms have been proposed for multi-agent systems (e.g., temporal logic, multi-modal logic). These formalisms are generally abstract and not related to concrete computational models (D'Inverno et al., 1997). Other approaches have used traditional formal languages such as Z and CSP (Brazier, Dunin-Keplicz, Jennings, Treur, & Lesser, 1995; Luck & D'Inverno, 2001). While providing an accessible notation, these formalisms lack the diagrammatic representation and tool support necessary to effectively analyze models.

Alloy is a specification language based on set theory and first-order relational logic (Jackson, 2002). The language has a simple and concise syntax that can represent complex structural properties and behavior. It comes with an Analyzer, a powerful, integrated tool for compiling and analyzing models. The Analyzer supports two types of automatic analysis: 1) the search for an instance that satisfies all the constraints and relations specified in a model; 2) the identification of a counterexample that violates the assertions specified in a model. Both analysis are performed within a user defined *scope* that bounds the cardinality of entity sets in instances of the model. Outputs can be graphically depicted using the visualizer and evaluated using the command-line evaluator.

Alloy has been used in both industry and academia (Coppit & Sullivan, 2000; Dolby et al., 2007; Jackson & Vaziri, 2000). Jackson and Vaziri (2000) have proposed an approach to verify Java methods in Alloy. At IBM, a subset of Alloy has been used to develop a technique for efficient checking of data structure invariants (Dolby et al., 2007). Alloy was also used in (Coppit, Yang, Khurshid, Le, & Sullivan, 2005) to test and find bugs in Galileo, a dynamic fault tree analysis tool used at NASA (Coppit & Sullivan, 2000).

4.2. MATISSE Metamodel

For the purpose of specifying MATISSE in Alloy, we introduce a set of related concepts based on the discussion presented in Section 3. Figure 4 depicts the traffic domain concepts of the simulation. It describes the different types of virtual agents, their environment and organizational relationships. In this model, the *Virtual Environment* consists of *Traffic Areas* and represents the environment where *Virtual Agents* are situated in. A *Tower* provides guidance to virtual agents within its managed traffic area while being able to collaborate with other towers.

Figure 5 depicts *Cells* and *Cell Controllers* as previously discussed in Section 3. In addition, it defines *Communication Medium* as an abstraction of the communication mechanisms for vehicle to vehicle and vehicle to traffic infrastructure interactions. The relation *vicinity* represents a virtual agent's range of communication.

4.3. Specification of MATISSE Static Properties

The static properties of a model describe entities and their relationships. In Alloy, these are specified through the *signature* declaration.

Figure 4. Traffic metamodel

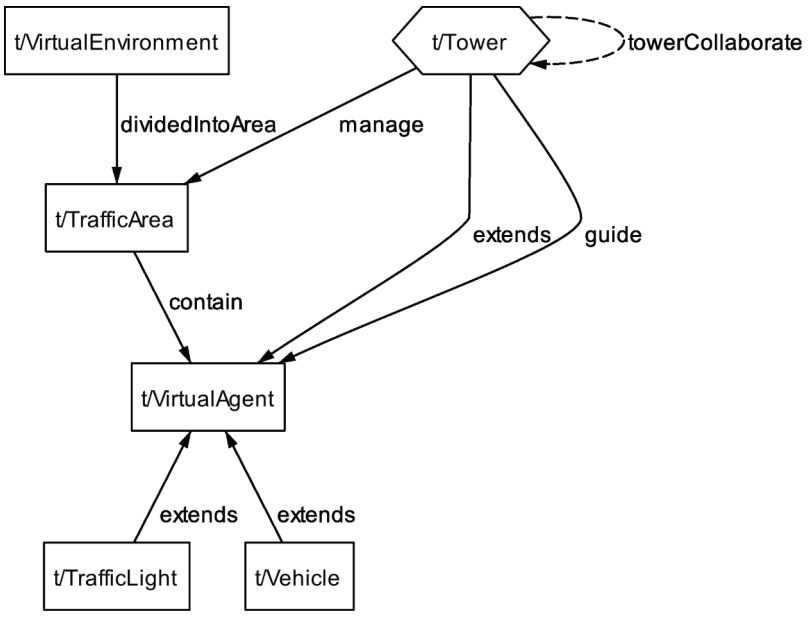
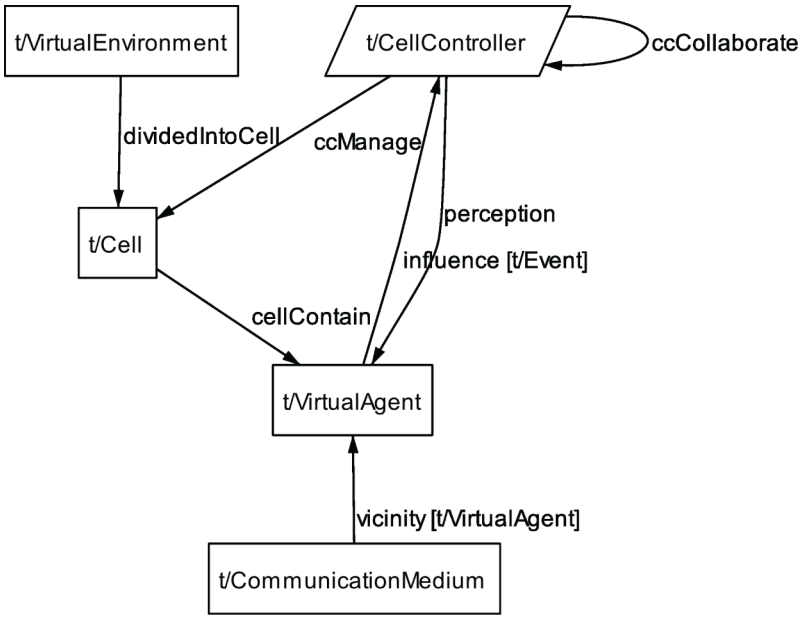


Figure 5. Simulation metamodel



For example, in the following specification excerpt, **module** TrafficSimulation Entity specifies vehicle-agents, traffic-light-agents, and tower-agents. It also specifies VirtualEnvironment, TrafficArea, Cell, CellController, and CommunicationMedium. The **one** keyword constraints the model to one virtual environment. Simulation events (both external and internal) are specified by Event and emergency alerts are specified by Alert.

```
1 module TrafficSimulationEntity
2 abstract sig VirtualAgent{}
3 sig Vehicle extends VirtualAgent{}
4 sig TrafficLight extends VirtualAgent{}
5 sig Tower extends VirtualAgent{}
6 one sig VirtualEnvironment{}
7 sig TrafficArea{}
8 sig Cell{}
9 sig CellController{}
10 sig CommunicationMedium{}
11 sig Event{}
12 sig Alert extends Event{}
```

The following specification fragment shows a partial specification of MATISSE's model. **module** TrafficSimulation makes use of the elements defined in **module** TrafficSimulationEntity to specify the relations and constrains of the model. An example of a relation, in **sig** Simulation, is guide that corresponds to the relationship between tower-agents and virtual-agents (e.g., vehicle-agents, traffic-light-agents). The aggregation of **module** TrafficSimulationEntity and TrafficSimulation makes up the complete MATISSE simulation model.

```
1 module TrafficSimulation
2 open TrafficSimulationEntity
3 sig Simulation{
4 dividedIntoArea: VirtualEnvironment one
5  → TrafficArea,
6  guide: Tower lone → VirtualAgent,
7  manage: Tower one → one TrafficArea,
8  contain: TrafficArea one → VirtualAgent,
9  towerCollaborate: Tower → Tower,
10  ccCollaborate: CellController → Cell-
    Controller,
11  dividedIntoCell: VirtualEnvironment one
    → Cell,
12 ccManage: CellController one → one Cell,
13  cellContain: Cell lone → VirtualAgent,
14  influence: VirtualAgent → Event → Cell-
    Controller,
15 perception: CellController → VirtualAgent,
16  knows: VirtualAgent → Event,
17   vicinity: CommunicationMedium →
    VirtualAgent
18   → VirtualAgent,
19  transmitted: VirtualAgent → Event
20   → CommunicationMedium,
21 relayed: CommunicationMedium → Event
    →VirtualAgent,
22  sent: VirtualAgent → Event → Tower,
23  notified: Tower → Event → VirtualAgent,
24  propagated: Tower → Event → Tower
25 }{
26 ...
27  contain - (TrafficArea \
28   → (VirtualAgent - Tower)) = ~manage
29 all va: (VirtualAgent - Tower) | one t: Tower |
30   t → va in guide
31  no t: Tower | t → t in guide
32  all t, t': Tower |
33  not ((t → t' in guide) and (t'→ t in guide))
34  towerCollaborate = ~towerCollaborate
35  ...
36 }
```

Alloy enables the precise specification of static properties such as "each *tower-agent* manages a *virtual traffic area*". Using relation multiplicities, manage: Tower **one** → **one** TrafficArea specifies a one-to-one relation between tower and traffic area elements. Further, the constraint contain − (TrafficArea → (VirtualAgent − Tower)) = ~manage ensures that each tower-agent is assigned to a unique traffic area, and that each area is uniquely associated to its tower-agent.

4.4. Specification of MATISSE Dynamic Properties

In Alloy, operations are specified through *predicates*, which relate valid instances of Simulation through a change in its composition. For instance, **pred** sendMessage makes use of the *function* **fun** getTransmittedMessage to add the relation between a virtual agent and the communication medium to s in order to produce s', in which s and s' denote the before and after states of Simulation.

```
1 pred sendMessage[va: VirtualAgent, s, s': Simulation]{
2  s'.transmitted = s.transmitted
3  + getTransmittedMessage[va, s]
4 }
5
6  fun getTransmittedMessage[va: VirtualAgent,
7  s:Simulation]: VirtualAgent → Event
8  → CommunicationMedium {
9  (s.knows → CommunicationMedium)
10  - ((VirtualAgent - va) → Event → CommunicationMedium)
11 }
```

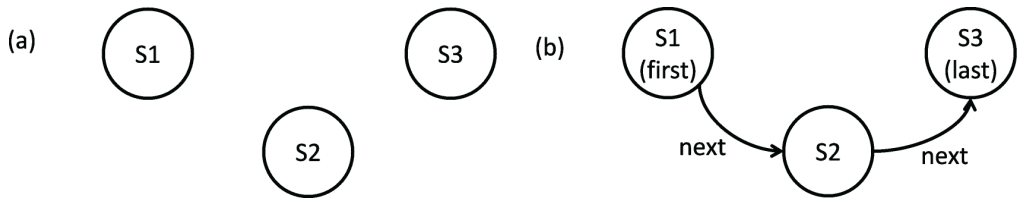
Thus far, the presented specification produces arbitrary, unrelated instances of the MATISSE simulation model. In order to model the system behavior, it is necessary to define relations between instances and make use of *execution traces*. To produce execution traces, we specify a linear ordering over Simulation elements (see Figure 6).

This is achieved by importing the library module util/ordering. This module includes functions first, next, and last. As depicted by Figure 6 (b), first returns the first element *S*1, s1.next returns *S*2 and s2.next returns *S*3, and last returns the last element *S*3.

The following fragments of MATISSE's specification illustrate the new constraints added to the model to enable execution traces. The **pred** init defines the initial conditions (i.e., the initial composition) and **pred** inv defines invariants (i.e., properties that never change during an execution trace) of Simulation. Any adjacent Simulation in the ordering is related by **fact** traces. For instance, the following trace fragment specifies short-range *vehicle-to-vehicle* and *vehicle-to-infrastructure* interactions. If a vehicle has transmitted a message in s, then the message is relayed to its recipients in s' through operation relayMessage (lines 18 to 23). Similarly, if a message has been relayed in s, then the message is stored in each recipient's knowledge base in s' through operation receiveMessage (lines 25 to 30).

```
1  module TrafficSimulation
2  open util/ordering[Simulation] as t
3
4  pred init[s:Simulation]{ ... }
5  pred inv[s, s':Simulation]{
6   s'.dividedIntoArea = s.dividedIntoArea
7   s'.guide = s.guide
8   ...
9 }
10 fact traces {
11   init[first]
12
13   all s:Simulation - last | let s' = s.next {
14    inv[s,s']
15   …
```

Figure 6. (a) Unrelated instances of the model and (b) Execution trace of the model



```
16   // vehicle-to-vehicle interactions
17   // and vehicle-to-traffic-devices interac-
     tions
18   let va=((s.transmitted).Communication-
     Medium).Event | {
19   (va.(getVicinity[va, s])
20   not in Event.(CommunicationMedium.
     (s.relayed)))
21   => relayMessage[va, s, s']
22   else s'.relayed = s.relayed
23 }
24
25 let va = (Event.(CommunicationMedium.
     (s.relayed))
26   + Event.(Tower.(s.notified))
27   - (s.knows).Event) | {
28   (va != none)
29   => receiveMessage[va, s, s']
30 }
31
32   // Information Passing within an area
33   …
34
35   // Event Propagation across traffic areas
36   ...
37   }
38 }
```

The next specification excerpt describes how information is passed within a traffic area. For example, upon facing an unpredicted event in s, a vehicle sends an alert to its traffic tower in s' through operation sendEvent (lines 2 to 11). This operation modifies the state of Simulation by adding a new element to relation sent. In a subsequent step s, this information is used in s' to pass the event information to all virtual agents located within the tower's traffic area through operation notifyEvent (lines 13 to 18).

```
1 // Information Passing within an area
2 let va = (s.knows.Event - Tower)
3    - Event.(Tower.(s.notified)) | {
4    (va not in ((s.sent).Tower).Event) =>
5    (sendEvent[va, s, s']
6    and (towers[va,s'].va).(s'.notified)
7    = (towers[va,s].va).(s.notified)
8    and (s'.knows - Tower → Event)
9    = (s.knows - Tower → Event))
10 else (s'.sent = s.sent)
11 }
12
13 let t = (Event.(VirtualAgent.(s.sent))) | {
14   (t not in ((s.notified).VirtualAgent).Event)
     =>
15   (notifyEvent[t, s, s'] and (s'.knows - t →
     Event)
16   = (s.knows - t → Event))
17   else t.(s'.notified) = t.(s.notified)
18 }
```

For the purpose of propagating information across traffic areas, MATISSE defines interactions between traffic towers. As specified in the following excerpt, if a traffic tower is aware of an event in s, then the event is propagated in s' to its adjacent towers through operation propagateEvent. This operation modifies the state of Simulation by adding new elements to relation propagated. In a subsequent step s, each tower uses this information in s' to pass the event to its virtual agents through operation notifyEvent.

```
1 // Event Propagation across traffic areas
2 let t = s.knows.Event - (VirtualAgent - Tower)
     | {
3    (t not in ((s.propagated).Tower).Event) =>
4      (propagateEvent[t, s, s'])
5    and (t.(s'.towerCollaborate)).(s'.notified)
6  = (t.(s.towerCollaborate)).(s.notified)
7  else (s'.propagated = s.propagated)
8  }
9
10 let t = Event.(Tower.(s.propagated)) | {
11  (t not in ((s.notified).VirtualAgent).Event)
     =>
12  notifyEvent[t, s, s']
13  else t.(s'.notified) = t.(s.notified)
14 }
```

This complete specification allows the analysis of the static and dynamic properties of MATISSE. In addition, a number of ITS traffic scenarios involving collaboration, information dissemination, and event propagation can be planned and designed to validate MATISSE's traffic model.

5. ANALYZING MATISSE'S PROPERTIES

In this section, we discuss how the above mentioned Alloy models are verified and validated. In the remainder of this paper, the term *verification* is used to refer to consistency checking between Alloy specification and MATISSE's design. On the other hand, the term *validation* is used to refer to conformance checking between Alloy specification and MATISSE's high level requirements.

5.1. MATISSE's Properties Verification

In this section, we discuss how static and dynamic properties of MATISSE are verified. In addition to providing model type checking features, the Alloy Analyzer allows us to define *assertions*, verify their correctness, and identify constraint violations if they exist.

The assertion VehicleSendEventOnlyToTowerGuidingIt states that for all instances of the Simulation a vehicle or traffic light can send an event (through operation sendEvent) only to the tower guiding it. No counterexample is found for this static property following the constraining facts specified in Simulation.

```
1 assert VehicleSendEventOnlyToTower-
    GuidingIt {
2   all s: Simulation | {
3     let va = ((s.sent).Tower).Event |
4       (va != none) => Event.(va.(s.sent)) =
    (s.guide).va
5   }
6 }
```

The assertion MessageIsRelayed is an example of verification of a dynamic property of the model, in which we ensure consistency between adjacent instances of Simulation. It states that if a vehicle has transmitted a message in s, then the message must be relayed to the vehicle's vicinity in s'. No counterexample is found for this property.

```
1 assert MessageIsRelayed {
2   all s: Simulation, s': s.next | {
3   let va = ((s.transmitted).Communication-
    Medium).Event |
4   let e = (va.(s.transmitted)).Communica-
    tionMedium |
5     let vicinity = getVicinity[va, s] |
6       (va != none) => va.vicinity
7   in e.(CommunicationMedium.(s'.relayed))
8   }
9 }
```

The assertion receiveMessageIsDeterministic ensures that **pred** receiveMessage is determinitic (i.e., each simulation s is associated with at most one simulation s'). It states that if a vehicle has received a message in both s' and s" after the message was relayed in s, then its knowledge base in s' must be the same as in s" (i.e., s'.knows = s".knows). No counterexample is found for this property.

```
1 assert receiveMessageIsDeterministic{
2   all s, s',s": Simulation, va: VirtualAgent |
3   receiveMessage[va,s,s'] and
    receiveMessage[va,s,s"]
4   => (s'.knows = s".knows)
5 }
```

The assertion initImpliesInv checks that the invariants of the model hold for all initial instances. This assertion states that **pred** init on any instance s implies the invariants between instances s and s.next in the execution trace. No counterexample is found for this property.

```
1 assert initImpliesInv{
2     all s:Simulation, s': s.next | init [s] =>
      inv[s,s']
3 }
```

Finally, the assertion sendMessagePreservesInv states that operation **pred** sendMessage preserves the invariants between ordered instances specified in the model. No counter-example is found for this property.

```
1 assert sendMessagePreservesInv{
2     all s:Simulation, s': s.next, va: VirtualAgent |
3     sendMessage[va,s,s'] => inv[s,s']
4 }
```

5.2. Traffic Scenarios Validation

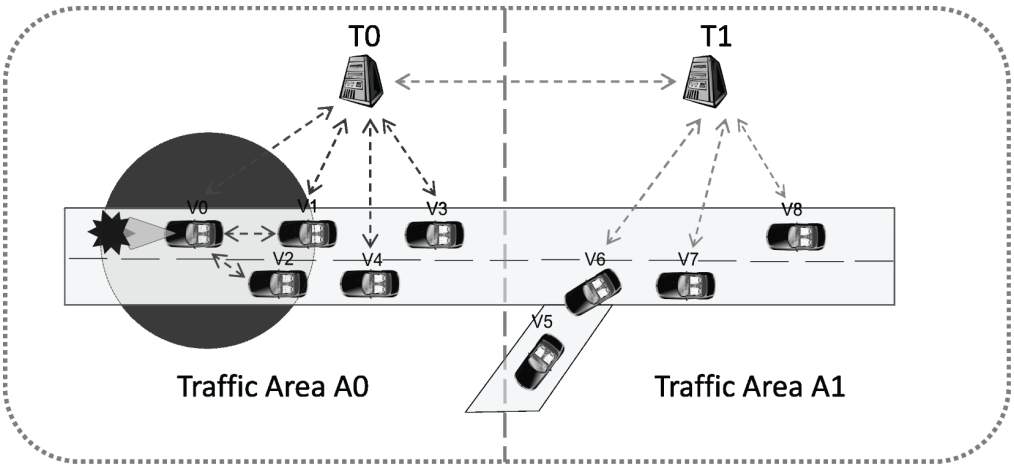
The following ITS scenarios are used to validate the simulation properties of MATISSE. For each case, we present a textual description of the scenario followed by an analysis of the execution traces generated from the Alloy specification. These execution traces validate simulation properties such as virtual agent perception, agent-to-agent interaction, and event propagation.

5.2.1. Scenario 1: Safety Enhancement and Congestion Reduction on a one-way road

The scenario depicted in Figure 7 demonstrates the suitability of MATISSE for safety improvement and congestion reduction. This ITS scenario consists of vehicles driving on a one-way road. An event (e.g., an accident, an obstruction on the road, or any other abnormal condition) has occurred in Traffic Area A0, and vehicle V0 perceives the event within its field of vision shown as a cone. Under this scenario, V0 takes the following steps: 1) it informs all vehicles located in its *close vicinity* about the perceived event through *vehicle-to-vehicle* interactions. The notified vehicles are able to take the necessary actions to avoid a major accident. 2) It informs traffic tower T0 about the perceived event via *vehicle-to-infrastructure* interactions.

After deliberation and based on the event characteristics, T0 alerts the vehicles located in A0 (i.e., V1 to V4) about the event. T0 also determines the potential impact of this event on neighboring traffic areas and informs the adjacent traffic tower T1 of the event. T1 de-

Figure 7. Scenario for safety enhancement and congestion reduction



liberates, and informs all vehicles located within Traffic Area A1 of the event and guides them in their choice of the best alternate route to follow (to avoid congestion). All vehicles in the traffic area make use of the broader traffic information to improve the overall safety condition and avoid traffic congestion.

The execution of the Alloy model for this scenario produces the execution trace consisting of the following sets of elements:

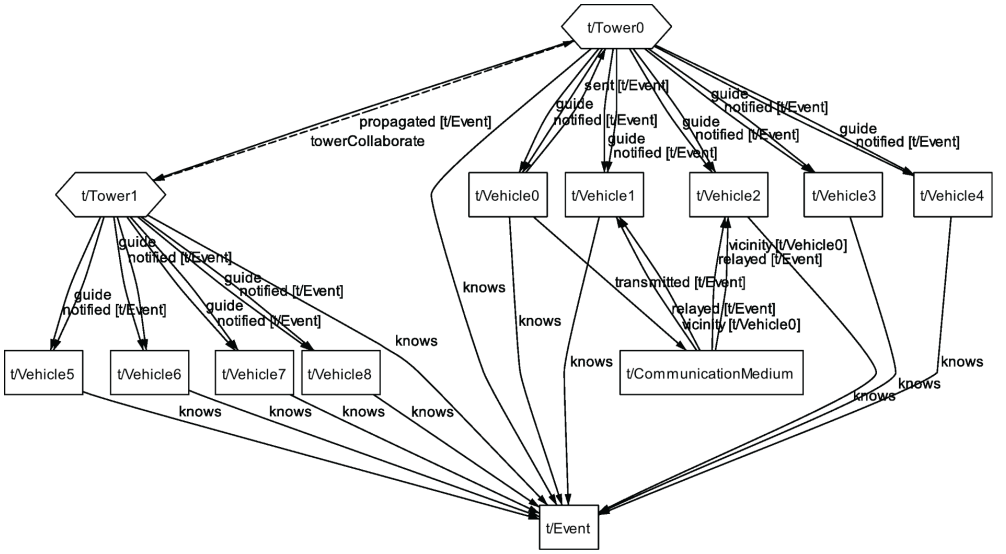
this/Simulation = {Simulation0, Simulation1, Simulation2,
Simulation3, Simulation4, Simulation5}
t/VirtualAgent = {t/Tower0, t/Tower1, t/Vehicle0,
t/Vehicle1, t/Vehicle2, t/Vehicle3,
t/Vehicle4, t/Vehicle5, t/Vehicle6,
t/Vehicle7, t/Vehicle8}
t/TrafficArea = {t/TrafficArea0, t/TrafficArea1}
t/CommunicationMedium = {t/CommunicationMedium0}
t/Event = {t/Event0}

These sets (e.g., t/VirtualAgent, t/TrafficArea) correspond to the signatures defined in the specification (e.g., **sig** VirtualAgent, **sig** TrafficArea) and the elements (e.g., t/Tower0, t/TrafficArea0) are arbitrarily assigned to the sets at execution time. For the purpose of this section, sets such as VirtualEnvironment, CellController, and Cell are omitted from the discussion.

Figure 8 shows the visual representation of the last instance of the execution trace (i.e., *Simulation*5). Each element is depicted as a geometric figure, and each relation (e.g., *knows*, *guide*) as an arrow. The visual representations of the intermediate instances of the trace (i.e., *Simulation*0, *Simulation*1, *Simulation*2, *Simulation*3, and *Simulation*4) are omitted. The following steps describe in detail the complete execution trace.

In *Simulation*0, *Vehicle*0 perceives an *Event* through its sensors. It stores this information into its knowledge base as reflected by relation *knows*. In *Simulation*1, *Vehicle*0 broadcasts the event to the *CommunicationMedium*, and communicates the event information to its virtual traffic tower. This is reflected by relation *transmitted*[*Event*] between *Vehicle*0 and *CommunicationMedium*, and relation *sent*[*Event*] between *Vehicle*0 and *Tower*0.

Figure 8. Last instance of the execution trace for Scenario 1



In Simulation2, Tower0 stores the event information into its knowledge base as reflected by relation knows between Tower0 and Event. The CommunicationMedium proceeds by relaying the event to Vehicle0's vicinity (i.e., Vehicle1, Vehicle2). This is represented by relations vicinity[Vehicle0] and relayed[Event] between Communication Medium and vehicles Vehicle1 and Vehicle2. Also, Tower0 communicates Event to all vehicles in its traffic area as depicted by relations notified[Event].

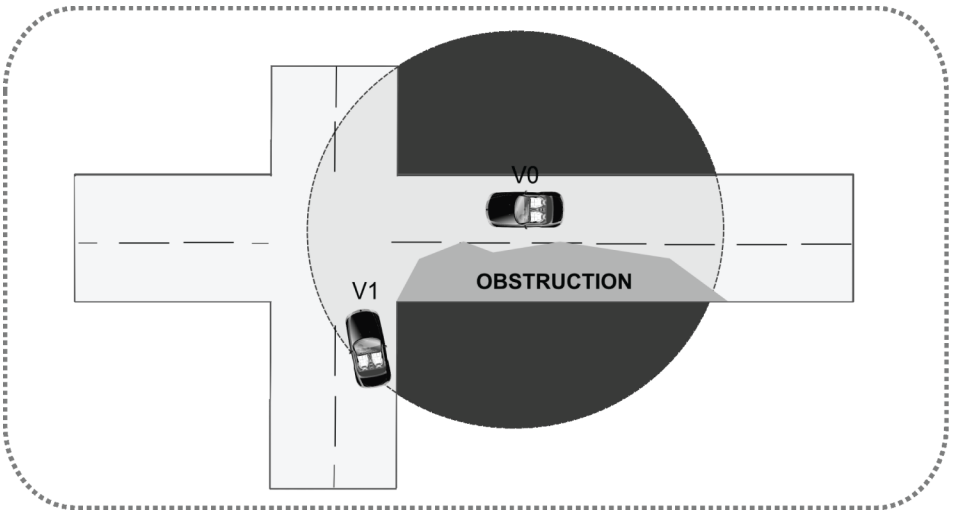
In *Simulation*3, vehicles *vehicle*1 and *vehicle*2 store the relayed event (received via *CommunicationMedium*) into their knowledge bases. In addition, all vehicles within *Tower*0's traffic area store the event notification (received via *Tower*0) into their knowledge bases. This is reflected by relations *knows* between *Vehicle*0, *Vehicle*1, *Vehicle*2, *Vehicle*3, *Vehicle*4, *Vehicle*5 and *Event*. Also, *Tower*0 uses its acquaintance model represented by relation *towerCollaborate* to identify the neighboring tower that might be affected by the event (in this case, *Tower*1) and passes *Event* on to it. This is reflected by relation *propagated*[*Event*] between *Tower*0 and *Tower*1.

In *Simulation*4, *Tower*1 stores *Event* into its knowledge base as reflected by relation *knows* between *Tower*1 and *Event*. Also, *Tower*1 communicates *Event* to its local vehicles as represented by relation *notified*[*Event*]. Finally, in *Simulation*5, all vehicles within *Tower*1's traffic area store the event information received into their knowledge bases. This is reflected by the relations *knows* between *Vehicle*5, *Vehicle*6, *Vehicle*7, *Vehicle*8, and *Event*.

5.2.2. Scenario 2: Intersection Collision Avoidance

The scenario depicted in Figure 9 demonstrates the suitability of MATISSE for the simulation of short-range communication between vehicles (Xu, Mak, Ko, & Sengupta, 2004) where upstream communication with towers is not necessary. In this scenario, vehicle V1 wants to make a left turn while there is an obstruction blocking its vision from oncoming vehicles on

Figure 9. Scenario for intersection collision avoidance



the right of the intersection. Using *vehicle-to-vehicle* communication, vehicle V1 is alerted about vehicle V0's presence, thus allowing vehicle V1 to take necessary actions to avoid a potential accident.

The execution of the Alloy model for this scenario produces the execution trace consisting of the following sets of elements:

this/Simulation={Simulation0, Simulation1, Simulation2,
Simulation3}
t/VirtualAgent={t/Tower0, t/Vehicle0, t/Vehicle1}
t/TrafficArea={t/TrafficArea0}
t/CommunicationMedium={t/CommunicationMedium0}
t/Alert={t/Alert0}

Figure 10 shows a visual representation of the complete execution trace. In *Simulation*0, *Vehicle*0 contains *Vehicle*1 in its vicinity as reflected by relation *vicinity*[*Vehicle*0]. In *Simulation*1, *Vehicle*0 broadcasts an alert (using a standard dedicated short-range communication mechanism) warning the surrounding vehicles of its presence near the road intersection. This is reflected by relation *transmitted*[*Alert*] between *Vehicle*0 and *CommunicationMedium*. In *Simulation*2, the *CommunicationMedium* uses *Vehicle*0's vicinity to relay the alert to *Vehicle*1, as depicted by relation *relayed*[*Alert*]. Finally, in *Simulation*3, although *Vehicle*0 is out of *Vehicle*1's field of vision, *Vehicle*1 becomes aware that *Vehicle*0 is approaching the intersection, as reflected by relation *knows* between *Vehicle*1 and *Alert*.

6. EVALUATION OF THE APPROACH

In this section, we discuss our evaluation of the approach used to specify and analyze MATISSE's properties.

Abstraction. By abstracting from implementation details, Alloy helped us focus on the most important aspects of system design and explore design alternatives. For instance, it allowed us to revisit various responsibilities assigned to agents when modeling agent-to-agent interactions.

Process. We have started the design activity with a small model containing just a few signatures and constraints, and progressed by adding detail iteratively. At each step, key aspects of the system were modeled, checked, and simulated without writing a single line of code.

Model Execution. The execution of traces is a valuable tool that allowed us to identify several conceptual inconsistencies of the model such as a traffic tower incorrectly passing an event information to a vehicle outside its traffic area. The step-by-step scenario execution enabled us to analyze the various states in which the traffic model can be and validates its high level properties.

Analysis. To make analysis feasible, it is necessary to define a scope that restricts the number of elements of each signature. This restriction comes at a cost: counterexamples might go undetected if they are outside of the scope. Nevertheless, as mentioned by Jackson (2011), it is likely that invalid assertions have counterexamples within small scopes. For the purposes of verifying MATISSE's properties, we found that the analysis provided by Alloy was sufficient to produce meaningful results. For instance, the verification of the assertions VehicleSendEventOnlyToTowerGuidingIt (A1), MessageIsRelayed (A2) and receiveMessageIsDeterministic (A3) produce the logical clauses shown in Table 1 and the total analysis time depicted in Figure 11. As stated earlier no counterexamples were found for these assertions. Despite the exponential growth of the search space and time, our experimental results show that even small scopes produce large search spaces. This is sufficient to increase our confidence in the correctness of the model. To illustrate this, we introduced inconsistencies in the model and identified counterexamples within scopes of six elements per signature.

Separation of Concerns. The notion of scope is decoupled from the model, allowing us to analyze different scenarios without

Figure 10. Complete execution trace for scenario 2

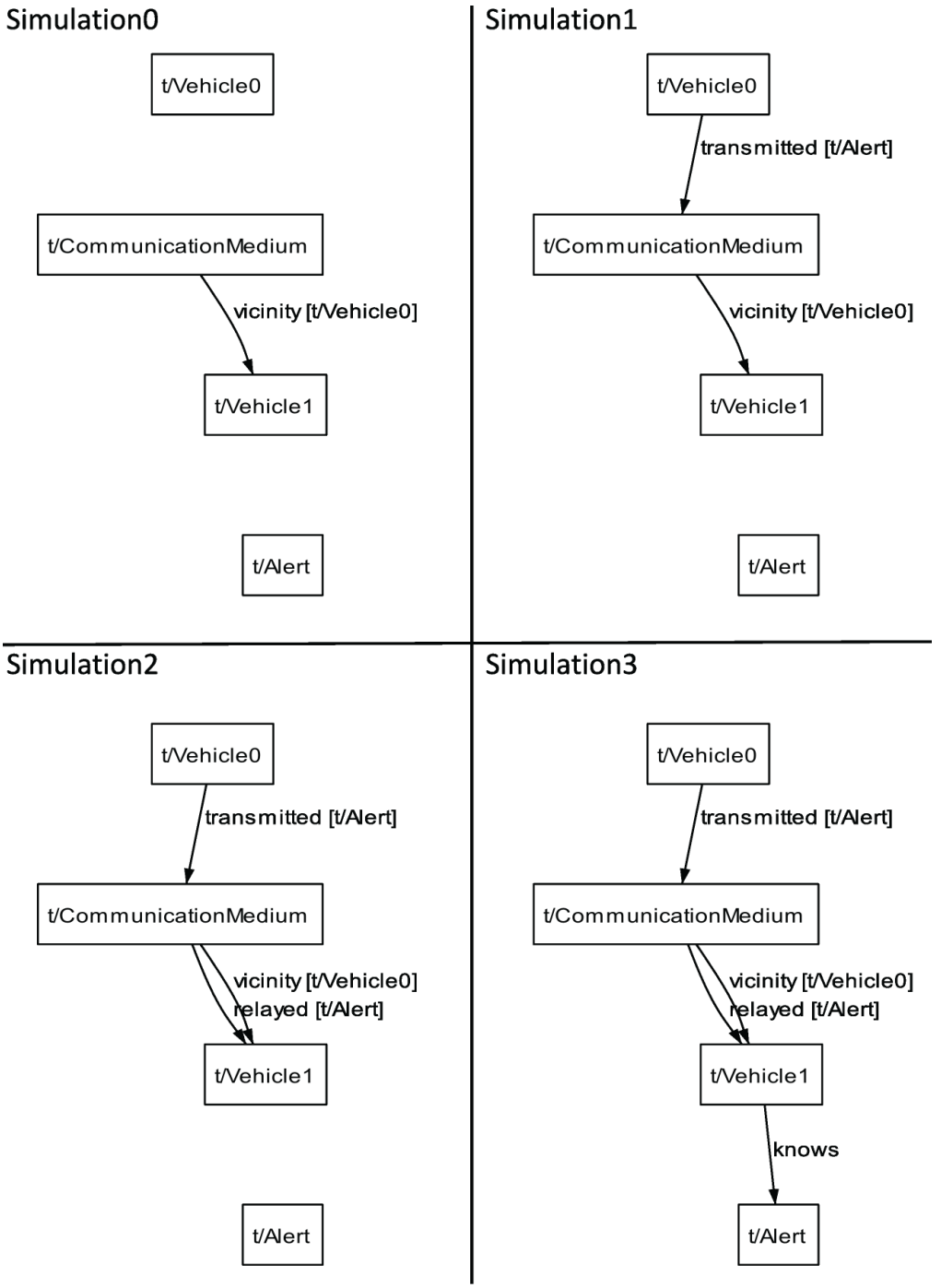
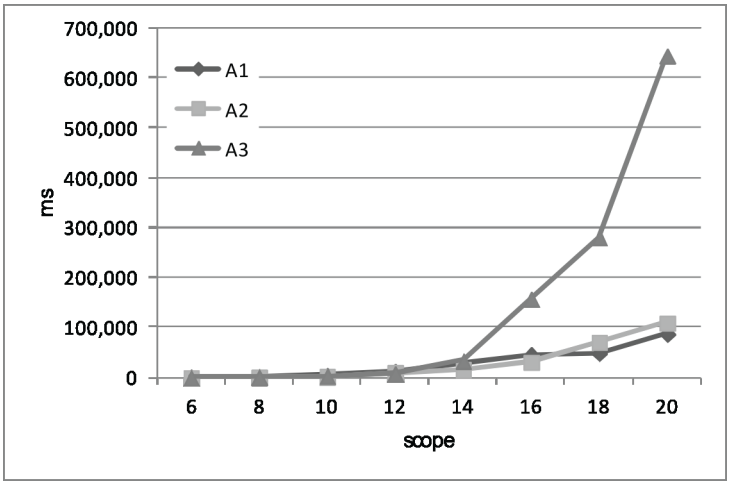


Table 1. Number of logical clauses generated for assertions checked on the MATISSE model

Scope	A1	A2	A3
6	491,653	484,616	494,540
8	1,074,997	1,057,730	1,079,548
10	2,011,318	1,976,925	2,017,573
12	3,370,218	3,310,027	3,378,025
14	5,239,118	5,142,681	5,248,133
16	7,620,086	7,475,179	7,629,773
18	10,774,265	10,566,888	10,783,896
20	14,521,253	14,235,630	14,529,908

Figure 11. Scope vs. analysis time for assertions checked on the MATISSE model



modifying the model itself. For example, the following excerpt shows the *run* commands used to produce the execution traces for each of the scenarios discussed in Section 5. In Scenario 1 (line 2) we instructed the Analyzer to produce an execution trace with 2 Towers and 9 Vehicles in 6 Simulation steps, while in Scenario 2 (line 5) we instructed the Analyzer to produce an execution trace with 1 Tower and 2 Vehicles in 4 Simulation steps. This separation resulted in more robust models enabling us to specify MATISSE's properties independently of analysis concerns.

```
1  // Scenario 1
2  run Show for 2 Tower, 9 Vehicle but 6
   Simulation
3
4  // Scenario 2
5  run Show for 1 Tower, 2 Vehicle but 4
   Simulation
```

Formalism. Despite its apparent simplicity, a strong foundation in set theory and logic is essential to specify complex interactions involving communication and cooperation among agents in Alloy.

Real-time Properties. We experienced difficulty coping with the timing aspects of the simulation. For instance, there is no built in mechanism in Alloy to specify and verify real-time properties. Hence, it is not possible to verify that a given communication occurs in due time among agents. Although Alloy includes some basic temporal constraints, it is less expressive than traditional temporal logics for specifying complex temporal properties such as instantaneous transitions and interval properties of system states. For these properties, approaches such as duration calculus, timed petri-nets, and timed process algebras proved to be more suitable (Bihler & Vogler, 2004; Cacciagrano & Corradini, 2004; Zhou & Hansen, 2004).

Based on these observations, we believe that in order to specify complex real-time systems, it is necessary to complement our modeling approach with a notation such as Statecharts (Harel, 1987). This follows the traditional belief that functional and temporal behaviors are orthogonal concerns that can be naturally modeled using different notations (Broy, 2007; Cacciagrano & Corradini, 2004).

Statecharts is a visual diagrammatic notation with a formal semantic foundation that has been extended to tackle hard real-time systems (Burmester & Giese, 2005; Giese & Burmester, 2003). It provides mechanisms to model state hierarchy and concurrency, and thus allows for the definition of compact and expressive models.

7. RELATED WORK

7.1. Multi-Agent Systems Formalisms

As stated in Section 4, several formalisms have emerged for the specification of multi-agent systems. In general, each of them provides a different emphasis and no single formalism seems to be suitable to model all aspects of the problem (e.g., functional, reactive, intentional). As identified in (D'Inverno et al., 1997), these formalisms can be grouped under three categories: temporal logic, modal logic, and well-known formal specification languages from traditional software engineering.

Temporal logic has been used to reason about liveness and safety properties of agent systems (Wooldridge, 2009). For instance, branching time temporal logic such as CTL and its extensions have been used to specify and verify dynamic properties of multi-agent systems (Fisher & Wooldridge, 1997; Lomuscio, Qu, & Raimondi, 2009; Wooldridge & Jennings, 1999). However, their inherent complexity hinders their wide adoption as a tool for the specification of large-scale multi-agent systems (Wooldridge, Jennings, & Kinny, 2000).

Another approach uses modal logic to specify cognitive properties (beliefs, goals, tasks) of agents. In Wooldridge and Jennings (1999), a framework based on a quantified multi-modal logic has been proposed to characterize the agent's mental states (e.g., states triggering agents to engage in cooperative actions). These formalisms are generally abstract and not related to concrete computational models, making their application for mainstream multi-agent system development difficult (D'Inverno et al., 1997).

In contrast, well-known formal specification approaches are more accessible and appropriate for describing software systems at different levels of abstraction. For example, Z has been used to define the basis of a framework for specifying functional properties of multi-agent systems (D'Inverno & Luck, 1998; Luck & D'Inverno, 2001). However, these approaches do not naturally lend themselves to executable specifications and lack complete tool suites for automated analysis. Alloy overcomes these limitations by providing model type checking and animation directly built-in in the Analyzer.

7.2. Verification of Multi-Agent Systems

Besides specifying the static and dynamic properties of a complex system, it is important to consider the problem of verifying such specifications. Model checking (Clarke, 1997) has been widely used for the verification of MAS systems (Benerecetti & Cimatti, 2002;

Bordini, Fisher, Visser, & Wooldridge, 2006; Rao & Georgeff, 1993; Van Der Hoek & Wooldridge, 2002). Given a finite state model M of a system and a property P expressed via a logical formula φ_P to be checked, the model checking problem is to verify whether or not $M \vDash \varphi_P$.

Although inspired by model checking, analysis in Alloy relies on recent advances in SAT (Boolean satisfiability) techniques. The Analyzer translates Alloy specifications into Boolean constraints which are given as input to SAT solvers. As opposed to model checkers, the Analyzer does not exhaustively search the entire state space. Also, since only traces of bounded length are considered, the Analyzer finds counterexamples faster than a model checker due to the depth-first nature of the SAT solver (Jackson, 2011).

8. CONCLUSION

MATISSE is a multi-agent based simulation platform designed to specify and execute traffic simulations for a new generation of ITS. MATISSE's unique features include its open, decentralized and distributed environment; the ability of agents to perceive their surroundings in simulated real time; and the ability to execute micro- and macro-level ITS scenarios within the same framework.

In this paper we discussed how Alloy, a modeling language based on set theory and first order logic, was used to specify and analyze the static and dynamic properties of MATISSE. Future work includes determining how to integrate Alloy models with Statecharts and evaluating the scalability of Alloy specifications for complex interaction patterns.

ACKNOWLEDGMENTS

This project is partially supported by Rockwell Collins under the grant number 5-25143.

REFERENCES

Babin, A., Florian, M., James-Lefebvre, L., & Spiess, H. (1982). Emme/2: Interactive graphic method for road and transit planning. *Transportation Research Record*, 866.

Balmer, M., Rieser, M., Meister, K., Charypar, D., Lefebvre, N., & Nagel, K. (2009). Matsim-t: Architecture and simulation times. *Multi-agent Systems for Traffic and Transportation Engineering*, 57–78.

Benerecetti, M., & Cimatti, A. (2002). Symbolic model checking for multi-agent systems. *Proceedings of the MoChart*, *2*, 1–8.

Bihler, E., Vogler, W., & Bernardo, M. (2004). Timed petri nets: Efficiency of asynchronous systems. In B. Marco, F. Corradini (Eds.), *Formal Methods for the Design of Real-Time Systems* (*LNCS* 3185, pp. 105-106). Berlin/Heidelberg, Germany: Springer Berlin / Heidelberg.

Bordini, R., Fisher, M., Visser, W., & Wooldridge, M. (2006). Verifying multi-agent programs by model checking. *Autonomous Agents and Multi-Agent Systems*, *12*(2), 239–256. doi:10.1007/s10458-006-5955-7

Boyraz, P., Daescu, O., Fumagalli, A., Hansen, J., Trumper, K., & Wenkstern, R. (2009a). *Soteria: An integrated macro-micro transportation super-infrastructure system for management and safety (Tech. Rep.)*. Dallas, TX: University of Texas at Dallas, Erik Jonsson School of Engineering and Computer Science.

Boyraz, P., Yang, X., Sathyanarayana, A., & Hansen, J. (2009b). Computer vision systems for "context-aware" active vehicle safety and driver assistance. In *Proceedings of the 21st International Technical Conference on the Enhanced Safety of Vehicles*. Stuttgart, Germany: National Highway Traffic Safety Administration.

Brazier, F., Dunin-Keplicz, B., Jennings, N., Treur, J., & Lesser, V. (1995). Formal specification of multi-agent systems: A real world case. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS)*. Cambridge, MA: MIT Press.

Broy, M. (2007). From "formal methods" to system modeling. In C. B. Jones, Z. Liu, & J. Woodcock (Eds.), *Formal methods and hybrid real-time systems* (LNCS 4700, pp. 24-44). Berlin/Heidelberg, Germany: Springer Berlin / Heidelberg.

Burmester, S., Giese, H., & Schäfer, W. (2005). Model-driven architecture for hard real-time systems: From platform independent models to code. In A. Hartman, & D. Kreische (Eds.), *Model driven architecture foundations and applications* (LNCS 3748, pp. 25-40). Berlin/Heidelberg, Germany: Springer Berlin / Heidelberg.

Cacciagrano, D., & Corradini, F. (2004). Expressiveness of timed events and timed languages. In M. Bernardo & F. Corradini (Eds.), *Formal methods for the design of real-time systems*, (LNCS 3185, pp. 51-53). Berlin/Heidelberg, Germany: Springer Berlin / Heidelberg.

Cetin, N., Nagel, K., Raney, B., & Voellmy, A. (2002). Large-scale multi-agent transportation simulations. *Computer Physics Communications*, *147*(1-2), 559–564. doi:10.1016/S0010-4655(02)00353-3

Clarke, E. (1997). Model checking. In S. Ramesh & G. Sivakumar (Eds.), *Foundations of Software Technology and Theoretical Computer Science* (Vol. 1346, pp. 54–56). Berlin/Heidelberg, Germany: Springer Berlin / Heidelberg.

Coppit, D., & Sullivan, K. J. (2000). Galileo: A tool built from mass-market applications. In *Proceedings of the 22nd International Conference on Software Engineering.* New York, NY: ACM.

Coppit, D., Yang, J., Khurshid, S., Le, W., & Sullivan, K. (2005). Software assurance by bounded exhaustive testing. *IEEE Transactions on Software Engineering*, *31*(4), 328–339. doi:10.1109/TSE.2005.52

D'Inverno, M., Fisher, M., Lomuscio, A., Luck, M., De Rijke, M., Ryan, M., & Wooldridge, M. (1997). Formalisms for multi-agent systems. *The Knowledge Engineering Review*, *12*(3), 315–321. doi:10.1017/S0269888997003068

D'Inverno, M., & Luck, M. (1998). Engineering AgentSpeak (L): A formal computational model. *Journal of Logic and Computation*, *8*(3), 233–260. doi:10.1093/logcom/8.3.233

Dolby, J., Vaziri, M., & Tip, F. (2007). Finding bugs efficiently with a SAT solver. In *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering* (pp. 195–204). New York, NY: ACM.

Dresner, K., & Stone, P. (2008). A multiagent approach to autonomous intersection management. *Journal of Artificial Intelligence Research*, *31*(1), 591–656.

Fisher, M., & Wooldridge, M. (1997). On the formal specification and verification of multi-agent systems. *International Journal of Cooperative Information Systems*, *6*(1), 37–66. doi:10.1142/S0218843097000057

Galland, S., Gaud, N., Demange, J., & Koukam, A. (2009). Environment model for multiagent-based simulation of 3d urban systems. In *Proceedings of the 7th European Workshop on Multi-Agent Systems*, Ayia Napa, Cyprus.

Giese, H., & Burmester, S. (2003). *Real-time statechart semantics (Tech. Rep. tr-ri-03-239)*. Paderborn, Germany: University of Paderborn.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, *8*(3), 231–274. doi:10.1016/0167-6423(87)90035-9

Helbing, D., & Tilch, B. (1998). Generalized force model of traffic dynamics. *Physical Review E: Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics*, *58*(1), 133. doi:10.1103/PhysRevE.58.133

Jackson, D. (2002). Alloy: A lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology*, *11*(2), 256–290. doi:10.1145/505145.505149

Jackson, D. (2011). *Software abstractions: Logic, language and analysis*. Cambridge, MA: The MIT Press.

Jackson, D., & Vaziri, M. (2000). Finding bugs with a constraint solver. *ACM SIGSOFT Software Engineering Notes*, *25*(5), 14–25. doi:10.1145/347636.383378

Lieu, H., Santiago, A., & Kanaan, A. (1992). Corflo: An integrated traffic simulation system for corridors. In *Proceedings of the Engineering Foundation Conference.* Palm Coast, FL.

Lomuscio, A., Qu, H., & Raimondi, F. (2009). MCMAS: A model checker for the verification of multi-agent systems. In A. Bouajjani & O. Maler (Eds.), *Computer aided Verification* (Vol. 5643, pp. 682-688). Berlin/Heidelberg, Germany: Springer Berlin / Heidelberg.

Luck, M., & D'Inverno, M. (2001). A conceptual framework for agent definition and development. *The Computer Journal*, *44*(1), 1–20. doi:10.1093/comjnl/44.1.1

Meyer, M. (1997). *A toolbox for alleviating traffic congestion and enhancing mobility*. Washington, DC: Institute of Transportation Engineers.

Mili, R. Z., & Steiner, R. (2008). Modeling agent-environment interactions in adaptive MAS. In D. Weyns, S. Brueckner, & Y. Demazeau (Eds.), *Engineering environment-mediated multi-agent systems* (Vol. 5049, pp. 135–147). Berlin/Heidelberg, Germany: Springer Berlin / Heidelberg.

Mili, R. Z., Steiner, R., & Oladimeji, E. (2006). DIVAs: Illustrating an abstract architecture for agent-environment simulation systems. *Multiagent and Grid Systems, Special Issue on Agent-oriented Software Development Methodologies*, *2*(4), 505–525.

Rao, A., & Georgeff, M. (1993). A model-theoretic approach to the verification of situated reasoning systems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence* (Vol. 1, pp. 318–324).

Rossetti, R., & Liu, R. (2005). An agent-based approach to assess drivers' interaction with pre-trip information systems. *Journal of Intelligent Transportation Systems*, *9*(1), 1–10. doi:10.1080/15472450590912529

Russell, S., & Norvig, P. (1995). *Artificial intelligence: A modern approach*. Englewood Cliffs, NJ: Prentice Hall.

Sukthankar, R., Hancock, J., & Thorpe, C. (1998). Tactical-level simulation for intelligent transportation systems. *Mathematical and Computer Modelling*, *27*(9-11), 229–242. doi:10.1016/S0895-7177(98)00062-4

Van Der Hoek, W., & Wooldridge, M. (2002). Model checking knowledge and time. *Model Checking Software*, 25–26.

Wenkstern, R. Z., Steel, T., Daescu, O., Hansen, J., & Boyraz, P. (2009a). MATISSE: A large scale multi-agent system for simulating traffic safety scenarios. In *Proceedings of IEEE 4th Biennial Workshop on DSP for In-Vehicle Systems and Safety.*

Wenkstern, R. Z., Steel, T., & Leask, G. (2009b). A self-organizing architecture for traffic management. In *Proceedings of Workshop on Self-Organizing Architectures, Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture.*

Wooldridge, M. (2009). *An introduction to multiagent systems*. Chichester, UK: Wiley.

Wooldridge, M., & Jennings, N. (1999). The cooperative problem-solving process. *Journal of Logic and Computation*, *9*(4), 563–592. doi:10.1093/logcom/9.4.563

Wooldridge, M., Jennings, N., & Kinny, D. (2000). The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, *3*(3), 285–312. doi:10.1023/A:1010071910869

Xu, Q., Mak, T., Ko, J., & Sengupta, R. (2004). Vehicle-to-vehicle safety messaging in DSRC. In *Proceedings of the 1st ACM International Workshop on Vehicular Ad Hoc Networks* (pp. 19–28). New York, NY: ACM.

Zhou, C., & Hansen, M. (2004). *Duration calculus: A formal approach to real-time systems*. Berlin, Germany: Springer-Verlag.

Junia Valente is a PhD candidate at the School of Engineering and Computer Science at the University of Texas at Dallas. She is a researcher at the UTD Multi-Agent & Visualization Systems Lab where she is currently involved with several research projects in multi-agent based simulation systems. Her research interests include self-adaptive and self-organizing systems, multi-agent systems, and modeling and simulation of agent-based social and intelligent transportation systems. She holds a MS degree in Computer Science with Major in Software Engineering and a BS degree in Software Engineering with Minor in Music from the University of Texas at Dallas.

Frederico Araujo is a PhD candidate in Software Engineering at the University of Texas at Dallas. He holds a BS degree in Electrical Engineering from the University of São Paulo and a MS degree from the École Centrale Paris, France. He also holds a MS degree in Computer Science from the University of Texas at Dallas. His current research interests include self-adaptive and self-organizing systems and agent-based simulation of intelligent transportation systems. He is also interested in modern software paradigms and empirical software engineering. Past academic research includes the study of the application of Artificial Neural Networks to support safety analysis and fault location on distributed systems.

Rym Zalila-Wenkstern is an Associate Professor at the School of Engineering and Computer Science, University of Texas at Dallas. She holds a PhD in Computer Science from the University of Ottawa, Canada, and the Doctorat de Spécialité in Computer Science from the University of Tunis, Tunisia. She is the director of the Multi Agent and Visualization Systems lab. Her research projects have been sponsored by several organizations including the National Science Foundation, Sandia National Laboratories, Rockwell Collins and the Department of Education. Dr. Zalila-Wenkstern has served on several international conference organizing committees and numerous program committees. She has worked as a consultant for U.S. and European organizations. She is the CEO of ZW Corp, a startup specializing in the development of web-based multi-agent systems and the director of the Executive Masters of Science in Software Engineering program at UT Dallas. Dr. Zalila-Wenkstern is a member of ACM, IEEE and SWE professional societies.