Towards World Model-based Test Generation in Autonomous Systems

Anneliese Andrews, Mahmoud Abdelgawad and Ahmed Gario

Department of Computer Science, University of Denver, Denver, CO 80208 U.S.A.

Keywords: World Model, Behavioral Model, Model-based Testing, Test Generation, Autonomous Systems.

Abstract: This paper describes a model-based test generation approach for testing autonomous systems interacting with their environment (i.e., world). Unlike other approaches that assume a static world with attributes and values, we present and test the world dynamically. We build the world model in two steps: a structural model that constructs environmental factors (i.e., actors) and a behavioral model that describes actors' behaviors over a certain landscape (i.e., snippet). Abstract world behavioral test cases (AWBTCs) are then generated by covering the behavioral model using graph coverage criteria. The world model-based test generation technique (WMBTG) is used on an autonomous ground vehicle (AGV).

1 INTRODUCTION

According to (Cheng, 2011), autonomous systems are those systems that can accomplish entirely or in part certain goals without human intervention. In this paper, we consider autonomous mobile robot platforms (so-called Unmanned Systems). Autonomous systems exist in various applications such as rescue robots, military robots, driverless vehicles, and in-door robots (e.g., robotic vacuum cleaners (Roomba)). Testing the interactions between autonomous systems and world actors- pedestrians, mobile objects, and unknown obstacles- poses a series of challenges, due to the complexity of these systems and the unpredictability of their environment. In order to generate behavioral test cases in the form of concurrent world stimuli, model-based testing (MBT) is able to leverage behavioral models, such as communicating extended finite state machines (CEFSMs) (Li and Wong, 2002; Cheng and Krishnakumar, 1993), Coloured Petri Nets (CPN) (Lill and Saglietti, 2012), Labelled Transition Systems (LTS) (Tretmans, 2008), and sequence and communication diagrams of the Unified Modeling Language (UML) (Shirole and Kumar, 2013), to describe the behavioral scenarios that can occur between the system under test (SUT) and its world. However, MBT can pose challenges due to the state space explosion issues of many behavioral models. This requires testing solutions to deal with the large number of possibilities in behavioral scenarios. Current MBT approaches for testing real-time embedded systems (RTES) interacting with their worlds assume a static world model (Iqbal et al., 2012; Hes-

sel et al., 2008), which does not show the interactions can be occurred in these worlds. However, for autonomous systems, the world cannot be described only statically with attributes and values, the world should also be presented and tested dynamically. To address these challenges, we propose a systematic MBT approach, world model-based test generation (WMBTG), that identifies *what, where and how* to test worlds of autonomous systems, uncovers diverse types of autonomous systems failures, improves scalability issues and avoids state space explosion by using a hierarchical modeling approach instead of flattening all behaviors of actors into a single behavioral model (Andrews et al., 2010). Tests are generated by aggregating test paths in the individual models similar to (Andrews et al., 2010). We apply WMBTG to autonomous city vehicles (ACVs) (Furda and Vlacic, 2010). ACVs are driverless vehicles that share the highway with other traffic participants. We formalize the efficiency of test criteria that are used to generate abstract world behavioral test cases (AWBTCs). This paper uses UML class diagrams to depict the structural model of actors and their relationships. Communicating extended finite state machines (CEFSMs) are used to represent typical landscapes in the environment. We call these landscapes *snippets*. They are used to link behavioral models of various actors in this world together. Our objective is to provide an MBT technique for testing autonomous systems behavior in a dynamic world alongside behavioral testing that is flexible, systematic, scalable, and shows potential of being extendable to other types of applications and types of applicable behavioral models such as CPT

and LTS.

The remainder of this paper is organized as follows. Section 2 gives the state of research about model-based testing, testing autonomous systems, and world model-based testing. The case study is described in section 3. Section 4 presents our approach and applies it to the case study. We also analyze and discuss complexity and efficiency issues in the same section. Section 5 draws conclusions.

2 STATE OF RESEARCH

2.1 Model-Based Testing (MBT)

(Dias-Neto et al., 2007) provide a survey on model-based testing (MBT). MBT uses various models to automatically generate tests. MBT includes three key elements: models that describe software behavior, criteria that guide the test-generation algorithms, and tools that generate supporting infrastructure for the tests. (Zander et al., 2012) define MBT as an algorithm that generates test cases automatically from models instead of creating them manually. (Utting et al., 2012) also provide a survey on MBT. They define six dimensions of MBT approaches (a taxonomy): model scope, characteristics, paradigm, test selection criteria, test generation technology and test execution. They also classify MBT notations as state based, history based, functional, operational, stochastic, and transition based. Transition based notations are graphical node-and-arc notations that focus on defining the transitions between states of the system such as various types of finite state machines. (Li and Wong, 2002) present an MBT approach to generate behavioral test cases. They use CEFSMs to model behavior and events of a system under test (SUT). Events with variables are used to model data while the events' interaction channels are used to model communication. The tests are then generated based on a combination of behavior, data, and communication specifications. (Shirole and Kumar, 2013) present a survey on model-based test generation from behavioral UML specification diagrams. They classify the various research approaches based on formal specifications, graph theory, and direct UML specification processing. Formal specification-based testing is an automated software testing method based on algebraic specifications. UML models are usually translated into a formal notation such as Petri nets, colored Petri nets, concurrent object oriented Petri nets, transition systems, and labeled transition systems, which are used as formal specifications. Test cases are then derived from the formal specification

and applied to the implementation. In graph-based testing, a test case is a path that covers some specific system requirement and data. (Shafique and Labiche, 2013) present a systematic review to determine the current state of the art of MBT tool support. They scope their study to tools which use state-based models: FSMs, EFSMs, abstract state machine (ASM), state-charts, UML state machines, (timed, input/output)-automata, Harel statecharts, Petri Nets, state flow diagram and Markov chains. They grouped MBT tools based on test criteria similarity and divided these criteria into four groups. *Model-flow criteria* refer to state, transition, transition-pair, all-paths and scenario criteria. *Script-flow criteria* refer to interface (function), statement, decision/branch, condition, modified-condition/ decision, and atomic-condition. *Data criteria* refer to the selection of input values when creating concrete test cases from abstract test cases: one-value, all-values, boundary-values and pair-wise values. The *requirement criterion* relies on traceability links between requirements and model elements. Twelve MBT tools are selected as primary studies. A comparison enables tool selection based on project needs.

2.2 Testing Autonomous Systems

Autonomy is the ability to operate independently, without the need for human guidance or intervention (Cheng, 2011). Based on this definition, autonomous systems can automatically achieve certain goals. They can decide which action to take even in unforeseen circumstances. Autonomous systems are now being deployed in safety, mission, and business critical scenarios. Increasingly, modern house-hold, business, and industrial systems incorporate autonomy as well (Fisher et al., 2013). Testing autonomous systems to provide a meaningful assessment of their reliability and robustness with respect to unknown and dynamically changing environments, presents a significant challenge. Although, in this paper, we only consider autonomous mobile robot platforms, our approach may be applicable for multi-agent systems (MAS) due to their autonomous behaviors. (Nguyen et al., 2011) provide a survey on testing methods and techniques in MAS. The authors classify the existing work on MAS testing based on testing levels. Testing in MAS consists of five levels *(unit, agent, integration, system, and acceptance)*. *System* testing intends to test the MAS as a system running in the target operating environment. They also organize testing MAS techniques into two categories, simulation-based techniques (*passive* approaches) and structured testing techniques (*active* approaches). They differ-

entiate between them in terms of test input perspectives. In *passive* techniques, test inputs are often predefined. On the contrary, *active* techniques obtain test inputs while monitoring the output behaviors of the SUT. Despite the classified literature on testing MAS, these approaches do not consider that tests can be generated from a world model. (Rehman and Nadeem, 2013) analyze and evaluate seven techniques based on seven variables (case study, evolution capability, testing framework, tool support, input artifact, artifact coverage, and level of testing). Although the survey is not extensive, the findings indicate that MBT and *Integration* testing of autonomous agent systems requires serious attention. (Jacoff et al., 2003) introduce performance metrics to evaluate capabilities and behaviors of autonomous mobile robots. These robots perform a variety of urban search and rescue (USAR) tasks i.e. explore the maze-like test course, autonomously negotiate obstacles, find simulated victims, identify hazards, deliver sustenance and communications, and generate practical maps of the environment. The presented technique simulates mobile robots' collaboration in realistic situations in a variety of *arenas*. Arenas are collapsed structures that are designed and modeled from buildings in various stages of collapse. The technique uses a *Reference Test Arena for Autonomous Mobile Robots* which was developed through support from DARPA. The technique also provides up to thirty simulated victims placed throughout the arenas. Each victim displays up to five signs of life (form, motion, body heat, sound, and CO_2 emission). Unlike our technique, the test arenas technique is world simulation-based and uses static testing worlds. Similarly, (Arnold and Alexander, 2013) provide a technique to generate automatically a wide range of test situations, which are a combination of maps, peer entities, and missions or objectives. Situations represent simulation environments. Situations are, then, executed against a simulated autonomous robot in the simulator (i.e., Player/Stage robot simulator) to observe how the robot behaves. They generated and ran 500 different situations with randomized map size, obstacle density and minimum route lengths for simulating Pioneer 3-AT robot. These situations involved collisions between the autonomous robot and one of several dumb robots. Although the approach is not fully automated, it requires a human engineer to study accident scenarios, the result shows that faults were uncovered. (Lill and Saglietti, 2012) use a MBT technique for testing autonomous systems. First, the authors compare different modeling notations (Process Algebras like Calculus of Communicating Systems (CCS) and Communicating Sequential Processes (CSP), UML activity diagrams, Petri

Nets (PNs), and Coloured Petri Nets (CPNs)) that are used to model concurrent behavior of cooperating autonomous systems. The comparison is based on four evaluation criteria (understandability, well-definedness, scalability, and testability). The authors then select CPNs to model a factory robot due to its high scalability. The factory robot carries a load from one place to another. Obstacle passing is not considered. They also define coverage criteria tailored to the characteristics of CPNs, such as colour-based and event-based coverage criteria. Although there is no test generation, the authors found CPNs as the most promising option for MBT of autonomous systems. Even though the literature on testing autonomous systems is large, no work found that aims to use behavioral model such as CEFSM and CPN to address testing dynamic worlds.

2.3 World Model-Based Testing

Most approaches in the literature on modeling the world of autonomous systems define the world model as a software control component that represents the autonomous system's view of its world. On the contrary, in our approach, the world is considered as independent actors interacting with the SUT instead of being part of it. Existing approaches mostly have the purpose of increasing the understandability of the autonomous system to the relevant surrounding world in order to implement proper, efficient, and safe behavior, *but* they are not aiming for model-based testing. (Ghete et al., 2010) contribute an intelligent information storage and management system approach for autonomous systems with the aim of modeling the world of an autonomous system. The approach uses a three-pillar information architecture: prior knowledge, world model, and real world (sensory information). Sensory information and prior knowledge are stored as world model, and then are delivered to cognitive processes. The world model is represented as instances of classes with class specific attributes and relations. (Furda and Vlacic, 2010) also present an object-oriented world model approach for the road traffic environment of autonomous vehicles. The main feature of the approach is to build an accurate and real-time world model that is used as input information for a decision-making module in order to execute the most appropriate driving maneuver for any given traffic situation. The authors divide input information into: 1) priori information that comprises all advance information, before the autonomous vehicle starts its journey, such as a planned travel path, 2) real-time information that is obtained from on-board sensors in real-time during the vehicle's movement,

and 3) communication information that is provided through vehicle-to-vehicle (V2V) communication, or through vehicle-to-infrastructure (V2I) communication (e.g., a traffic management center). The approach uses UML class diagrams to represent the structure of the world actors.

A closely related approach for world model-based testing and its extensions is presented in (Iqbal et al., 2012). The approach limits the world model to a static world. It is specified for testing real-time embedded systems (RTES); however, it is not applicable for autonomous systems because their worlds are dynamic. The approach generates black-box test cases automatically based on the static world model. The main characteristics of the approach are: 1) modeling the structural and behavioral world properties, especially real-time properties. Invariants and error states such as unsafe, undesirable, or illegal states are also modeled. They use an extension of UML (MARTE) that models and analyzes real-time embedded systems, to model the world. 2) Test oracles are then generated automatically from the world model. A simulator is used to observe actual response. 3) To identify feasible test cases and maximize possibilities of fault detection, heuristic algorithms are used as test generation strategies. An empirical study is conducted to identify which test case generation approach obtains the best results. The experiment shows that ART is the best among the algorithms.

3 APPLICATION DESCRIPTION

3.1 Autonomous City Vehicles (ACVs)

There are many applications of autonomous system. The autonomous robotic vacuum cleaner is a well-known example. (Couceiro et al., 2014) present a survey on multi-robot systems (MRS). The use of MRS is especially preferable when the development area is either hazardous or inaccessible to humans, e.g., search-and-rescue (SaR) victims in catastrophic scenarios. In the driverless vehicles domain, there are three categories of applications, autonomous underwater vehicles (AUVs), autonomous aerial vehicles (AAVs), and autonomous ground vehicles (AGVs). In this paper, we focus on AGVs. (Cheng, 2011) defines AGVs as consisting of four modules: world perception and modeling (sensors), localization and map building (sensors and communications), path planning and decision-making (intelligent algorithms), and motion control (actuators). The world perception and modeling module includes both image-based sensors like monocular and stereo cameras (monochrome

and color), and range sensing devices like radio detection and ranging (RADAR), Laser detection and ranging (LADAR), and Light detection and ranging (LIDAR). These sensors are responsible for providing a concrete description of the surrounding world, e.g. static obstacles, moving objects, vehicle position, etc. Motion control consists of a set of actuators that are controlled by AVGs autonomously, including throttle control, steering control, and brake control. Unless we are dealing with automatic transmission, the gearshift also has to be automated. These actuators are based on two control tasks, longitudinal and lateral. The longitudinal control refers to an AGV's speed regulation and involves throttle and brake. The lateral control includes an AGV's steering to follow a track reference, and turning lights to warn other vehicles. Autonomous city vehicles that are detailed in section 2.3 are used in this paper as SUT. These AGVs continuously communicate and synchronize information about their surrounding world through V2V and V2I. They also interact with a set of world actors considered to be necessary for autonomous driving. Our case study uses a structured world (U.S. Highways) to model instead of using an unstructured (Urban) one as described in (Furda and Vlacic, 2010).

3.2 U.S. Highway

A highway can be divided into multiple snippets (entrance ramp, divided highway, and exit ramp). Each snippet contains relevant instances of traffic control devices (TCDs). For instance, the entrance ramp includes a Speed-limit sign and a Red-Green traffic light. The express highway is composed of only a group of Speed-limit signs. Moreover, the same TCDs in different snippets may send different messages. For example, a Speed-limit sign on an entrance ramp sends a different message than a Speed-limit sign on the express highway. For example, the entrance ramp may contain many actors such as a red traffic light, a stopped vehicle, a single solid white line showing the lane's boundary, and a single dotted white line guiding road users to the highway junction. The U.S. Department of Transportation, Federal Highway Administration (FHWA), has developed the design details of TCDs on all public streets, highways, bikeways, and private roads open to public traffic as a standard (The Manual on Uniform Traffic Control Devices (MUTCD)) (U.S. Department of Transportation, 2013). We use MUTCD as an information resource to build the highway world model. In general, TCDs are classified into three groups, *signal*, *sign*, and *marking* control devices. *Signal* control devices send light signals such as red, yellow, and green

light to control traffic. *Sign* control devices use text and numbers, such as speed limit and stop signs, to regulate/alert road users. *Marking* control devices are usually painted on the ground, such as dotted, solid, single, double, white, and yellow lines that guide road users to promote highway safety and efficiency. All TCD groups are also designed to send four types of messages (regulatory, guidance, option, and support). Regulatory messages have a statement of required, mandatory, or specifically prohibitive practice. Guidance messages convey a statement of recommended, but not mandatory, practice. A statement of practice is a permissive condition and carries no requirement or recommendation. Support messages express informational statements that do not convey any degree of mandate, recommendation, authorization, prohibition, or enforceable condition. For detail of United States TCDs, see (U.S. Department of Transportation, 2013).

4 APPROACH

Our objective is to provide a systematic model-based test generation approach to generate test cases from autonomous systems' world model. Because of some scalability and complexity issues of the dynamic worlds, especially when actors act independently and unpredictably, we concentrate on actors that autonomous systems are dealing with, behaviors of these actors, and stimuli (messages) that autonomous systems can read (perceive) from actors. The locations where actors interact are also considered. A group of actions that a set of actors can perform can happen over a particular snippet. For instance, when an AGV travels on the highway, it performs certain functions (e.g., speed up, slow down, change lane, or emergency stop) depending on messages perceived from highway actors such as traffic control signs and other road users. Therefore, we build the world behavioral model in two steps. First,

we construct a *structural model* of actors to represent their attributes, functions and relations. Second, we construct the *behavioral model* to describe actors' states and transitions and their interactions. Each actor is presented by one behavioral model showing messages they can send/receive. The communication between these actors represents the world model. These communications need to be modeled by a communicating behavioral semantic model such as a CEFSM that handles the communication between the actors. As such in our application (AGV and highway), actors are interacting simultaneously, the communicating behavioral model should cover not only the internal transitions of actors, but also the communicating between them. The communicating behavioral model can then be leveraged to generate world behavioral test cases. Once we build the world behavioral model, any member of the graph-based testing criteria from (Ammann and Offutt, 2008) can be used to generate behavioral test paths, which are abstract world behavioral test cases (AWBTCs). The test generation process is illustrated in Figure 1. The world model-based test generation process has the following three phases:

- Model the world by constructing structural and behavioral models.

- Select proper graph-based test criteria to cover the communicating behavioral model.

- Generate AWBTCs which are test paths extracted from the communicating behavioral model based on selected criteria.

4.1 Phase 1: World Modeling

4.1.1 Structural Model

The structural model is constructed using a UML class diagram, where classes represent actors including their important characteristics, messages, and relationships. Each group of actors is aggregated into a single snippet. The number of involved actors in the snippet is determined by their multiplicity relationship and similar actors that send common mes-
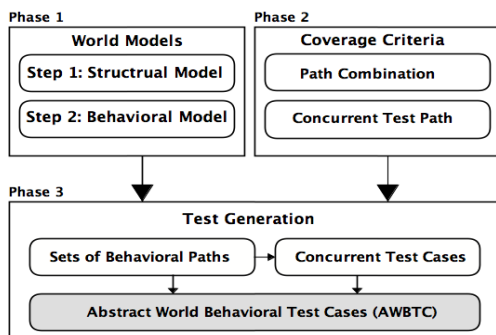

Figure 1: Behavioral Test Generation Process.

Table 1: Highway snippets instances.

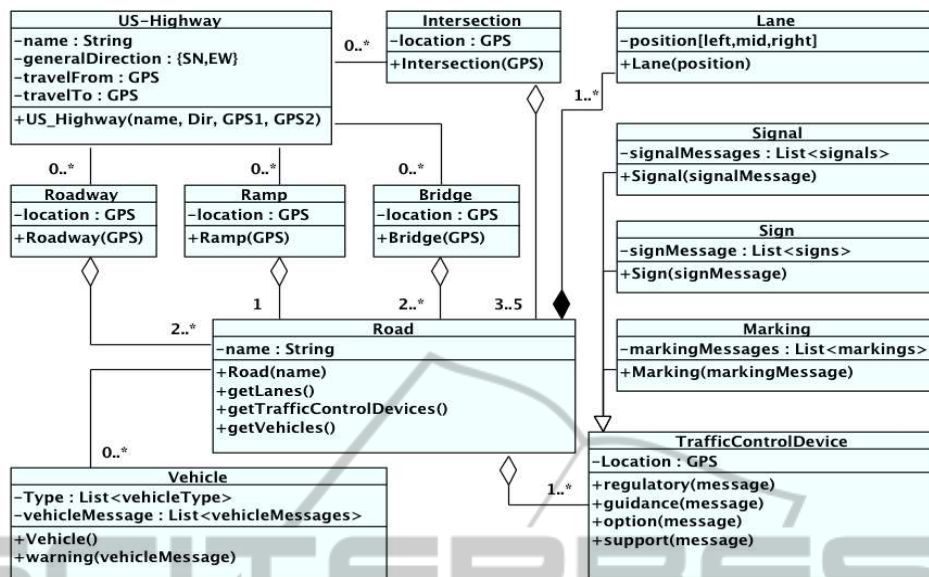| Class vs-pace0.08cm	Snippet Instances
Roadway	Freeway, expressway, non-toll highway, toll highway, and toll plaza.
Ramp	Entrance, exit, and switch ramp.
Bridge	Movable/unmoveable bridges and tunnels.
Intersection	Three, four, and five way of intersections.

Figure 2: Structural Model for Highway Actors.

sages are generalized into a single class. The association relation is also used to describe that one actor can support or be part of another actor or snippet. In our application, the highway can be represented by four types of snippets, roadways, ramps, bridges, and intersections. Table 1 shows examples of highway snippet instances. Actors that are considered with a highway are of four types: vehicles, signals, signs, and markings traffic devices. For simplicity, at this point, we assume that pedestrians do not exist in the highway world. Actors' instances and messages they can send are illustrated in Table 2. The UML class diagram that represents the structural model of the highway world is shown in Figure 2.
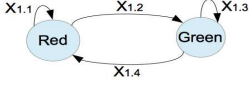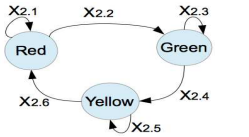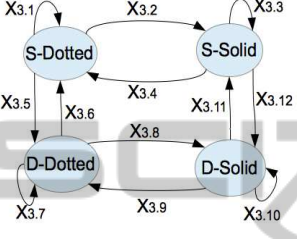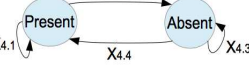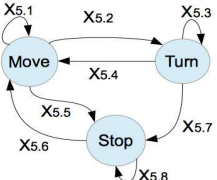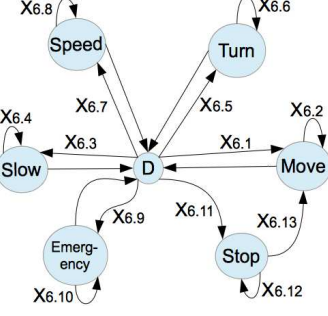
Table 2: Highway actors instances.

Class	Actor Instances	Messages
Vehicle	Passenger vehicles, fire vehicles, law enforcement vehicles, ambulances, and other official emergency vehicles.	ambulance.warning('turn flash light'). fireTruck.warning('siren released').
Signal Traffic Device	(Red and green), (Red, yellow, and green) traffic light, steady, flashing arrows, No U-turn movement flashing.	redGreenTLD.reglutory('redlight'). greenFlashing Arrow. guidance ('turn is allowed').
Sign Traffic Device	Stop, yield, speed limit, fines double, do not pass, emergency stopping only.	speedLimit.regulatory ('speed limit 45'). doNotPass.regulatory('Do Not Pass').
Marking Traffic Device	White left-turn arrows, single dotted and solid white line, double solid white and yellow lines, lanes reduction.	laneReduction.regulatory('lanes reduction single white line').

4.1.2 Behavioral Model

Although a wide range of behavioral models exists, we illustrate the behavioral model using communicating extend finite state machines (CEFSMs). The strength of CEFSM is that it can model orthogonal states of a system in a flat manner and does not need to compose the whole system in one state as in state charts, which would make it more complicated and harder to analyze and/or test (Brand and Zafiropulo, 1983; Li and Wong, 2002). *CEFSM = (S, s_0, E, P, T, A, M, V, C), such that:* S is a finite set of states, s_0 is the initial state, E is a set of events, P is a set of boolean predicates, T is a set of transition functions such that T: $S \times P \times E \rightarrow S \times A \times M$, A is a set of actions, M is a set of communicating messages, V is a set of variables, and C is the set of input/output communication channels used in the CEFSM. State changes (action language): The function T returns a next state, a set of output signals, and an action list for each combination of a current state, an input signal, and a predicate. It is defined as: $T(s_i, p_i, get(m_i))/(s_j, A, send(m_{j_1}, ..., m_{j_k}))$ where, s_i is the current state, s_j is the next state, p_i is the predicate that must be true in order to execute the transition, e_i is the event that when combined with a predicate triggers the transition function, $m_{i_1}, ..., m_{i_k}$ are the messages. CEFSM is a generalization of an EFSM (Cheng and Krishnakumar, 1993) (i.e., adding communication channels between EFSMs). We model individual actors as EFSMs and their interaction as CEFSM. Table 3 shows a set of EFSMs that represent a group of highway actors. Predicates that control transitions are also ex-

Table 3: Highway Actors' Behavioral Models.

X1. Red & Green traffic light	
	$X_{1.1}$:(Red,[timer<30sec],_)/(Red,timer++,actor.regulatory('Red light'))
	$X_{1.2}$:(Red,[timer=30sec],_)/(Green,timer=0,actor.regulatory('Green light'))
	$X_{1.3}$:(Green,[timer<30sec],_)/(Green,timer++,actor.regulatory('Green light'))
	$X_{1.4}$:(Green,[timer=30sec],_)/(Red,timer=0,actor.regulatory('Red light'))
X2. Red, Green & Yellow traffic light	
	$X_{2.1}$:(Red,[timer<60sec],_)/(Red,timer++,actor.regulatory('Red light'))
	$X_{2.2}$:(Red,[timer=60sec],_)/(Green,timer=0,actor.regulatory('Green light'))
	$X_{2.3}$:(Green,[timer<60sec],_)/(Green,timer++,actor.regulatory('Green light'))
	$X_{2.4}$:(Green,[timer=60sec],_)/(Yellow,timer=0,actor.regulatory('Yellow light'))
	$X_{2.5}$:(Yellow,[timer<30sec],_)/(Yellow,timer++,actor.regulatory('Yellow light'))
	$X_{2.6}$:(Yellow,[timer=30sec],_)/(Red,timer=0,actor.regulatory('Red light'))
X3. Dotted and Solids Marking Devices	
	$X_{3.1}$:(S-Dotted,[!Change],_)/(S-Dotted,_,actor.regulatory('Single dotted white line'))
	$X_{3.2}$:(S-Dotted,[Change],_)/(S-Solid,_,actor.regulatory('Single solid white line'))
	$X_{3.3}$:(S-Solid,[!Change],_)/(S-Solid,_,actor.regulatory('Single solid white line'))
	$X_{3.4}$:(S-Solid,[Change],_)/(S-Dotted,_,actor.regulatory('Single dotted white line'))
	$X_{3.5}$:(S-Dotted,[Change],_)/(D-Dotted,_,actor.regulatory('Double dotted white lines'))
	$X_{3.6}$:(D-Dotted,[Change],_)/(S-Solid,_,actor.regulatory('Single solid white line'))
	$X_{3.7}$:(D-Dotted,[!Change],_)/(D-Dotted,_,actor.regulatory('Double dotted white lines'))
	$X_{3.8}$:(D-Dotted,[Change],_)/(D-Solid,_,actor.regulatory('Double solid white lines'))
	$X_{3.9}$:(D-Solid,[Change],_)/(D-Dotted,_,actor.regulatory('Double dotted white lines'))
	$X_{3.10}$:(D-Solid,[!Change],_)/(D-Solid,_,actor.regulatory('Double solid white lines'))
	$X_{3.10}$:(D-Solid,[Change],_)/(S-Solid,_,actor.regulatory('Single solid white line'))
	$X_{3.12}$:(S-Solid,[Change],_)/(D-Solid,_,actor.regulatory('Double solid white lines'))
X4. Traffic sign	
	$X_{4.1}$:(Present,[Seen],_)/(Present,_,actor.regulatory('Traffic sign's rule'))
	$X_{4.2}$:(Present,[!Seen],_)/(Absent,_,actor.regulatory('Start traffic sign's rule'))
	$X_{4.3}$:(Absent,[!Seen],_)/(Absent,_,actor.regulatory('Keep following rule'))
	$X_{4.4}$:(Absent,[Seen],_)/(Present,_,actor.regulatory('New traffic sign's rule'))
X5. Passenger Car	
	$X_{5.1}$:(Move,_,_)/(Move,_,_)
	$X_{5.2}$:(Move,_,$get(m_i)$)/(Turn,_,actor.regulatory('Turning flash light on'))
	$X_{5.3}$:(Turn,_,_)/(Turn,_,actor.regulatory('Turning flash light on'))
	$X_{5.4}$:(Turn,_,_)/(Move,_,actor.regulatory('Turning flash light off'))
	$X_{5.5}$:(Move,_,$get(m_i)$)/(Stop,_,actor.regulatory('Stop light on'))
	$X_{5.6}$:(Stop,_,$get(m_i)$)/(Move,_,actor.regulatory('Stop light off'))
	$X_{5.7}$:(Turn,_,$get(m_i)$)/(Stop,_,actor.regulatory('Stop light on'))
	$X_{5.8}$:(Stop,_,_)/(Stop,_,actor.regulatory('Stop light on'))
X6. Car In an Emergency Situation	
	$X_{6.1}$:(D,_,_)/(Move,_,_)
	$X_{6.2}$:(Move,_,_)/(Move,_,_)
	$X_{6.3}$:(D,_,_)/(Slow,_,actor.regulatory('Stop light on'))
	$X_{6.4}$:(Slow,_,$get(m_i)$)/(Slow,_,actor.regulatory('Stop light on'))
	$X_{6.5}$:(D,_,$get(m_i)$)/(Turn,_,actor.regulatory('Turning flash light on'))
	$X_{6.6}$:(Turn,_,$get(m_i)$)/(Turn,_,actor.regulatory('Turning flash light'))
	$X_{6.7}$:(D,_,_)/(Speed,_,_)
	$X_{6.8}$:(Speed,_,_)/(Speed,_,_)
	$X_{6.9}$:(D,_,_)/(Emergency,_,actor.regulatory('Siren/Emergency-flashing'))
	$X_{6.10}$:(Emergency,_,_)/(Emergency,_,actor.regulatory('Siren/Emergency-flashing'))
	$X_{6.11}$:(D,_,$get(m_i)$)/(Stop,_,actor.regulatory('Stop light'))
	$X_{6.12}$:(Stop,_,$get(m_i)$)/(Stop,_,actor.regulatory('Stop light'))
	$X_{6.13}$:(Stop,_,$get(m_i)$)/(Move,_,actor.regulatory('Stop light off'))

pressed in Table 3. Highway actors are a red-yellow-green traffic light, a dotted-line marking device, a speed-limit sign device, an ambulance,..., a passenger car,..., etc. Note that X6 (Car in an emergency situation) uses a dynamic node D to model communication between the various states. This is done to reduce the number of transitions. Without it the graph would be fully connected. This approach also uses

dummy edges without transition annotations as input to node D. In our application, we selected the entrance ramp snippet as an example. We also selected five actors (red-green traffic light, dotted-solid lines, speed limit traffic sign, passenger vehicle, and emergent vehicle) to participate in this snippet. Three of them, the red-green traffic light, the dotted-solid lines, and the traffic sign, do not receive messages from outside. They only send messages periodically. The other two actors, passenger and emergency vehicles, however, do react to the messages that other actors send. They also send and react to each other. For instance, when the red-green traffic light actor turns to red, it sends a message to all vehicles to stop but this traffic light actor does not react to any message that comes from the world. On the other hand, a vehicle actor must stop when it receives a red light message from the traffic light actor. Figure 3 shows the CEFSM model that represents the concurrent interactions between these five actors over the entrance ramp snippet.

4.2 Phase 2: Coverage Criteria

Since actors interact concurrently, world behavioral model can be defined as a collection of concurrent processes. Each process is modeled as a $CEFSM_i$ that can be represented as a directed graph $G_i = (N_i, E_i)$ where N_i is a set of nodes and E_i is a set of edges and is considered as a conventional graph where it is treated sequentially (Yang and Chung, 1990). Test criteria such as edge-coverage, prime-path coverage etc. (Ammann and Offutt, 2008), can be applied. Using any of a number of test path generation techniques, test paths then can be generated that fulfill these coverage criteria. Let $P_i = (p_{i1}, p_{i2}, ..., p_{ik})$ be a set of such paths that cover G_i, where $1 \leq i \leq |CEFSMs|$ and $|CEFSMs|$ is the number of CEFSMs, which also corresponds to the number of actors that share a snippet, and k is the number of paths that internally cover G_i. We used edge-coverage to generate internal paths that cover actors' behavioral models $(X1, X2, ..., X6)$ from Table 3. The generated internal path sets, $P_1, P_2, ..., P_6$, are shown in the left column of Table 4. These actor CEFSMs communicate via the exchange of messages. There are two different types of paths, internal and global paths, that represent the concurrent execution behavior of CEFSMs. The internal paths describe the internal execution of the processes that can be characterized by the input and the sequence of the states involved in the execution. The global paths represent the communications between the CEFSMs. The behavior varies depending on changes in synchronization conditions among the concurrent processes. Therefore, the global paths

are considered concurrent paths. The concurrent interaction between different paths that represent multiple CEFSMs produces an arbitrary combination of internal paths of CEFSMs. As a result, we have two types of coverage criteria, path combination and concurrent test path coverage criteria.

4.2.1 Path Combination Coverage Criteria

As mentioned earlier, each actor is represented as a CEFSM. These CEFSMs communicate concurrently via global messages which are seen as global paths, as shown in the behavioral world model (High-level), in Figure 3. Therefore, we first have to generate global test paths that cover the global messages. We used simple-path coverage, see (Ammann and Offutt, 2008), to generate global test paths that cover the top level of the world behavioral model. Figure 3 shows six simple paths, $SP_1, SP_2, ..., SP_6$, that cover the top level of the world model of the entrance ramp snippet. Let $\mathbb{WBM}$ be defined as the set of simple paths $\{SP_1, SP_2, ..., SP_m\}$. Each simple path SP_i is composed of multiple sets of $\{P_i | 1 \leq i \leq n, n$ is the number of actors that participate in $SP_i\}$ of internal paths, which actually represent a behavioral scenario that can occur over the snippet. As a result, we have to combine each internal path p_{1i} (a path in P_1) with other internal paths, p_{5j} (a path in P_5) and p_{6i} (a path in P_6), in order to cover all possible combinations of internal paths that SP_1 can handle. We use the same process for the other simple paths. Therefore, path combination coverage criteria should determine what combinations are required. Let $P_1, P_2, ..., P_n$ be a path through the behavioral world model $\mathbb{WBM}$ and the sets of internal paths (nodes in SP_1), $P_1 = \{p_{11}, p_{12}, ..., p_{1i}\}$ and $P_2 = \{p_{21}, p_{22}, ..., p_{2j}\}$. Then, the selection of p_{1i} from P_1 and a p_{2j} from P_2 is called a path combination. Let $len(p)$ be the number of nodes in path p, the length of p. The path combination set of a simple path SP_i, $Comb_{SP_i} = \{(p_{jk}, ..., p_{mn}) | p_{mn} \in P_m, m = len(SP_i), n = |P_i|, 1 \leq j \leq m, 1 \leq k \leq n\}$. The number of all path combinations of SP_i will be the product of the number of internal paths of each P_i. Then, $|Comb_{SP_i}| = \prod_{j=1}^{len(SP_i)} |P_j|$.

At this point, we cover all possible combinations of internal paths that a simple path SP_i is composed of, although not all of them are feasible due to lack of reachability (Lei and Carver, 2006). Nevertheless, the resulting set of path combinations are not yet test paths, only path combinations that represent the references to internal paths of various CEFSMs. Therefore, we need test path coverage criteria that process each path combination individually in order to generate concurrent test paths.
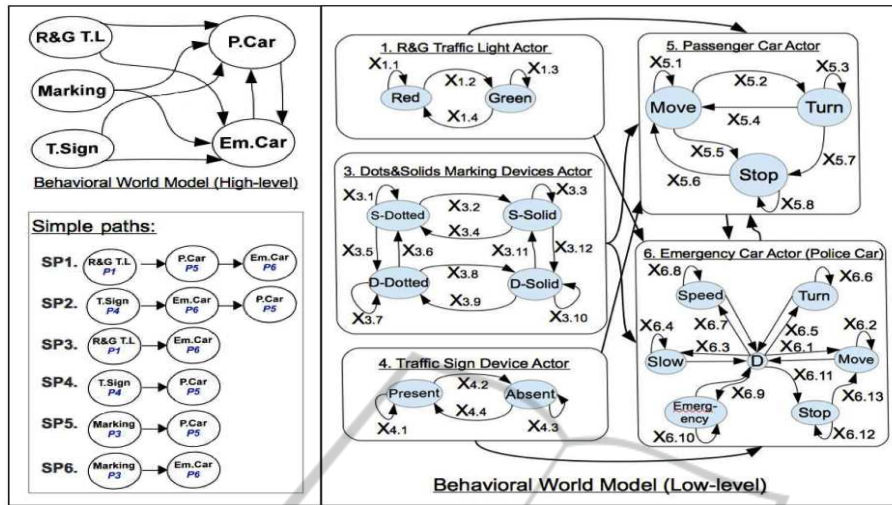
Figure 3: World Behavioral Model for Highway Entrance Ramp Snippet.

Table 4: Internal Paths of CEFSMs (using Edge Coverage) and Examples of AWBTCs.

P_1 is set of paths that covers X1: p_{11}: Red $\xrightarrow{X_{1.1}}$ Red $\xrightarrow{X_{1.2}}$ Green p_{12}: Green $\xrightarrow{X_{1.3}}$ Green $\xrightarrow{X_{1.4}}$ Red P_3 is set of paths that covers X3: p_{31}: S-Dotted $\xrightarrow{X_{3.2}}$ S-Solid $\xrightarrow{X_{3.4}}$ S-Dotted p_{32}: S-Dotted $\xrightarrow{X_{3.5}}$ D-Dotted $\xrightarrow{X_{3.6}}$ S-Dotted p_{33}: D-Solid $\xrightarrow{X_{3.9}}$ D-Dotted $\xrightarrow{X_{3.8}}$ D-Solid p_{34}: S-Solid $\xrightarrow{X_{3.12}}$ D-Solid $\xrightarrow{X_{3.11}}$ S-Solid P_4 is set of paths that covers X4: p_{41}: Present $\xrightarrow{X_{4.2}}$ Absent p_{42}: Absent $\xrightarrow{X_{4.4}}$ Present P_5 is set of paths that covers X5: p_{51}: Move $\xrightarrow{X_{5.2}}$ Turn p_{52}: Turn $\xrightarrow{X_{5.3}}$ Turn $\xrightarrow{X_{5.4}}$ Move p_{53}: Move $\xrightarrow{X_{5.5}}$ Stop p_{54}: Turn $\xrightarrow{X_{5.7}}$ Stop p_{55}: Stop $\xrightarrow{X_{5.8}}$ Stop $\xrightarrow{X_{5.6}}$ Move P_6 is set of paths that covers X6: p_{61}: Slow $\xrightarrow{X_{6.4}}$ Slow $\xrightarrow{X_{6.11}}$ Stop p_{62}: Stop $\xrightarrow{X_{6.12}}$ Stop $\xrightarrow{X_{6.13}}$ Move p_{63}: Move $\xrightarrow{X_{6.5}}$ Turn $\xrightarrow{X_{6.6}}$ Turn p_{64}: Move $\xrightarrow{X_{6.7}}$ Speed $\xrightarrow{X_{6.8}}$ Speed p_{65}: Move $\xrightarrow{X_{6.9}}$ Emergency $\xrightarrow{X_{6.10}}$ Emergency p_{66}: Speed $\xrightarrow{X_{6.3}}$ Slow $\xrightarrow{X_{6.1}}$ Move	**Example of path combinations and concurrent test paths that represent simple paths, shown in Figure 3:** • Simple path *SP1*, where P_1, P_5 and P_6 are synchronized: - Combination $c_{11} = (p_{11}, p_{51}, p_{61})$ p_{11} [Red $\xrightarrow{X_{1.1}}$ Red $\xrightarrow{X_{1.2}}$ Green] $\xrightarrow{\|}$ p_{51} [Move $\xrightarrow{X_{5.2}}$ Turn] $\xrightarrow{\|}$ p_{61} [Slow $\xrightarrow{X_{6.4}}$ Slow $\xrightarrow{X_{6.11}}$ Stop] - Combination $c_{12} = (p_{11}, p_{51}, p_{62})$ p_{11} [Red $\xrightarrow{X_{1.1}}$ Red $\xrightarrow{X_{1.2}}$ Green] $\xrightarrow{\|}$ p_{51} [Move $\xrightarrow{X_{5.2}}$ Turn] $\xrightarrow{\|}$ p_{62} [Stop $\xrightarrow{X_{6.12}}$ Stop $\xrightarrow{X_{6.13}}$ Move] • Simple path *SP2*, where P_4, P_6 and P_5 are synchronized: - Combination $c_{21} = (p_{41}, p_{61}, p_{51})$ p_{41} [Present $\xrightarrow{X_{4.2}}$ Absent] $\xrightarrow{\|}$ p_{61} [Slow $\xrightarrow{X_{6.4}}$ Slow $\xrightarrow{X_{6.11}}$ Stop] $\xrightarrow{\|}$ p_{51} [Move $\xrightarrow{X_{5.2}}$ Turn] - Combination $c_{22} = (p_{41}, p_{61}, p_{52})$ p_{41} [Present $\xrightarrow{X_{4.2}}$ Absent] $\xrightarrow{\|}$ p_{61} [Slow $\xrightarrow{X_{6.4}}$ Slow $\xrightarrow{X_{6.11}}$ Stop] $\xrightarrow{\|}$ p_{52} [Turn $\xrightarrow{X_{5.3}}$ Turn $\xrightarrow{X_{5.4}}$ Move] • Simple path *SP3*, where P_1 and P_6 are synchronized: - Combination $c_{31} = (p_{11}, p_{61})$ p_{11} [Red $\xrightarrow{X_{1.1}}$ Red $\xrightarrow{X_{1.2}}$ Green] $\xrightarrow{\|}$ p_{61} [Slow $\xrightarrow{X_{6.4}}$ Slow $\xrightarrow{X_{6.11}}$ Stop] - Combination $c_{32} = (p_{11}, p_{62})$ p_{11} [Red $\xrightarrow{X_{1.1}}$ Red $\xrightarrow{X_{1.2}}$ Green] $\xrightarrow{\|}$ p_{62} [Stop $\xrightarrow{X_{6.12}}$ Stop $\xrightarrow{X_{6.13}}$ Move] • Simple path *SP4*, where P_4 and P_5 are synchronized: - Combination $c_{41} = (p_{41}, p_{51})$ p_{41} [Present $\xrightarrow{X_{4.2}}$ Absent] $\xrightarrow{\|}$ p_{51} [Move $\xrightarrow{X_{5.2}}$ Turn] - Combination $c_{42} = (p_{41}, p_{52})$ p_{41} [Present $\xrightarrow{X_{4.2}}$ Absent] $\xrightarrow{\|}$ p_{52} [Turn $\xrightarrow{X_{5.3}}$ Turn $\xrightarrow{X_{5.4}}$ Move]

4.2.2 Concurrent Test Path Coverage Criteria

The produced path combination sets do not show how these paths interact concurrently. Consequently, we

have to have concurrent test path coverage criteria. In this paper, we investigate all possible serialized execution sequences that a combination of paths can produce although we expect that not all of these seri-

alizations are feasible. We also use the *Rendezvous* technique, as in (Yang and Chung, 1990). These concurrent test path coverage criteria are defined as follows:

- All Possible Serialized Execution Sequences Coverage Criterion (APSESCC): Test requirements contains a set of all possible serialized nodes of the paths that are included in each path combination, i.e. each node in path p_{ij} can be triggered by each node in path p_{kl} and vice versa. For example, let p_{ij} be $a \rightarrow b$ and p_{kl} be $x \rightarrow y$, where p_{ij} and p_{kl} are in the same path combination, and $a \rightarrow b$ means a sends a global message to b. Then, all serialized execution sequences of path combinations (p_{ij}, p_{kl}) will be: $((a \rightarrow b \rightarrow x \rightarrow y), (a \rightarrow x \rightarrow b \rightarrow y), (a \rightarrow x \rightarrow y \rightarrow b), (x \rightarrow y \rightarrow a \rightarrow b), (x \rightarrow a \rightarrow y \rightarrow b), (x \rightarrow a \rightarrow b \rightarrow y))$.

  If a path combination includes two paths and each one contains three nodes, there are 20 possible serializations. In general, all possible number of serializations of nodes is $\sum_{i=1}^{|Comb_{SP_i}|} \left(\frac{(\sum_{j=1}^{len(c_{ij})} |p_{mn}|)!}{\prod_{j=1}^{len(c_{ij})} (|p_{mn}|)!} \right)$.

  We do not consider this a practicable criterion, rather we can use it as an upper bound.

- Rendezvous Coverage Criterion (RCC): The test requirements contain a set of all paths that have rendezvous nodes. Then the possible number of rendezvous-paths $\mathbb{RZV}$ of the simple path SP_i is $\prod_{j=1}^{n} (P_j + 1) - 1$.

4.3 Phase 3: Test Generation

Common algorithms that satisfy conventional test coverage criteria stated in (Ammann and Offutt, 2008) are used. Our case study uses *edge* coverage to generate the internal test paths for each CEFSM. These test paths are then combined according to the *simple path* coverage criterion to cover the high level behavioral model. An example of internal test paths is shown in the left column of Table 4. The test paths in the right column of the same table are the combinations of the internal test paths that represent the

concurrent processes. We used the serialization algorithm in (Yang and Chung, 1990) to generate these concurrent paths. The concurrent paths illustrated in the right column are serialized nodes of the internal paths. We expressed the concurrency of the paths using double-bar "||" as used in LOTOS for defining concurrent functions, see (Sighireanu et al., 2000). However, the generated test paths are still abstract. To make these test paths executable, *test-data* coverage criteria, i.e. input-space partitioning (Ammann and Offutt, 2008), should be defined. One can also use the input selection method defined in (Ran et al., 2009). The number of concurrent behavioral test paths depends on the coverage criteria chosen to combine and generate test paths. The criteria used in this example produced a huge number of AWBTCs, which is expensive and requires reachability analysis (Carver and Lei, 2013; Hwang et al., 1994; Yang and Chung, 1990) of the test paths. Table 5 shows the number of combinations for each simple path $SP_i(i = 1 \ldots 6)$.

When we impose the All Serialized Execution Sequences Coverage Criterion (APSECC) on the path combination set $Comb_{SP_i}$, the number of produced concurrent test paths increases exponentially. For instance, the first path combination of $Comb_{SP_1}$, as seen in Table 4, $c_{11} = (p_{11}, p_{51}, p_{61})$, produced $\frac{(3+2+3)!}{3! \times 2! \times 3!} = 560$ test paths. The total number of AWBTCs generated from $Comb_{SP_1}$ is 60480 test paths, and the whole number of AWBTCs generated from all path combination sets, $Comb_{SP_1}, Comb_{SP_2}, \ldots, Comb_{SP_6}$, is 241920 test paths. This is clearly not scaleable. The Rendezvous Coverage Criterion (RCC) on SP_1 results in 125 test paths as $(((2+1) \times (5+1) \times (6+1)) - 1)$. Table 6 shows the number of test paths for each simple path $SP_i(i = 1 \ldots 6)$. The total number of test paths generated from simple paths, $SP_1, SP_1, \ldots, SP_6$ is 350 test paths. This number of test paths is reasonable due to the complexity of AGV systems.

5 CONCLUSION

This paper presented a novel model-based test generation approach that allows testing of autonomous

Table 5: Number of Path Combinations.

Simple Path	#Path Combination Sets
$SP_1 \rightarrow$	$(P_1, P_5, P_6) = 2 \times 5 \times 6 = 60$
$SP_2 \rightarrow$	$(P_4, P_6, P_5) = 2 \times 6 \times 5 = 60$
$SP_3 \rightarrow$	$(P_1, P_6) = 2 \times 6 = 12$
$SP_4 \rightarrow$	$(P_4, P_5) = 2 \times 5 = 10$
$SP_5 \rightarrow$	$(P_3, P_5) = 4 \times 5 = 20$
$SP_6 \rightarrow$	$(P_3, P_6) = 4 \times 6 = 24$

Table 6: Number of Test Paths for RCC.

Simple Path	#Test Paths
$SP_1 \rightarrow$	$(P_1, P_5, P_6) = (((2+1) \times (5+1) \times (6+1)) - 1) = 125$
$SP_2 \rightarrow$	$(P_4, P_6, P_5) = (((2+1) \times (6+1) \times (5+1)) - 1) = 125$
$SP_3 \rightarrow$	$(P_1, P_6) = (((2+1) \times (6+1)) - 1) = 20$
$SP_4 \rightarrow$	$(P_4, P_5) = (((2+1) \times (5+1)) - 1) = 17$
$SP_5 \rightarrow$	$(P_3, P_5) = (((4+1) \times (5+1)) - 1) = 29$
$SP_6 \rightarrow$	$(P_3, P_6) = (((4+1) \times (6+1)) - 1) = 34$

systems in their dynamic world. We modeled an active world of an autonomous system. A test generation process is defined. In addition, a path combination technique is introduced and formalized in order to generate concurrent test paths (AWBTCs). Two concurrent test criteria (APSESCC and RCC) are investigated. The findings indicate that RCC is practically feasible. Reachability analysis (Carver and Lei, 2013; Hwang et al., 1994; Yang and Chung, 1990) is required for both criteria (APSESCC and RCC) to increase their feasibility and efficiency. Future work will explore other path combination and concurrent test path coverage criteria. Future work will also experiment using the CADP (Construction and Analysis of Distributed Processes) toolbox (Garavel et al., 2013). Reachability analysis investigations and scalability of the world behavioral model will also be provided. We also plan on extending the approach to other robotic systems such as robots interacting with humans in manufacturing.

ACKNOWLEDGEMENTS

REFERENCES

Ammann, P. and Offutt, J. (2008). *Introduction to Software Testing*. Cambridge University Press, 32 Avenue of the Americas, New York, NY 10013, USA, first edition.

Andrews, A., Offutt, J., Dyreson, C., Mallery, C. J., Jerath, K., and Alexander, R. (2010). Scalability issues with using fsmweb to test web applications. *Inf. Softw. Technol.*, 52(1):52–66.

Arnold, J. and Alexander, R. (2013). Testing autonomous robot control software using procedural content generation. In Bitsch, F., Guiochet, J., and Kaniche, M., editors, *Computer Safety, Reliability, and Security*, volume 8153 of *Lecture Notes in Computer Science*, pages 33–44. Springer Berlin Heidelberg.

Brand, D. and Zafiropulo, P. (1983). On communicating finite-state machines. *J. ACM*, 30(2):323–342.

Carver, R. and Lei, Y. (2013). A modular approach to model-based testing of concurrent programs. In Lourenco, J. and Farchi, E., editors, *Multicore Software Engineering, Performance, and Tools*, volume 8063 of *Lecture Notes in Computer Science*, pages 85–96. Springer Berlin Heidelberg.

Cheng, H. (2011). *Autonomous Intelligent Vehicles: Theory, Algorithms, and Implementation*. Springer-Verlag, Springer London Dordrecht Heidelberg, New York, first edition.

Cheng, K. T. and Krishnakumar, A. (1993). Automatic functional test generation using the extended finite state machine model. In *Proceedings of the 30th International Design Automation Conference*, DAC '93, pages 86–91, New York, NY, USA. ACM.

Couceiro, M., Vargas, P., Rocha, R., and Ferreira, N. (2014). Benchmark of swarm robotics distributed techniques in a search task. *Robotics and Autonomous Systems*, 62(2):200–213.

Dias-Neto, A., Subramanyan, R., Vieira, M., and Travassos, G. H. (2007). A survey on model-based testing approaches: A systematic review. In *Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies: Held in Conjunction with the 22Nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*, WEASELTech '07, pages 31–36. ACM.

Fisher, M., Dennis, L., and Webster, M. (2013). Verifying autonomous systems. *Commun. ACM*, 56(9):84–93.

Furda, A. and Vlacic, L. (2010). An object-oriented design of a world model for autonomous city vehicles. In *Intelligent Vehicles Symposium (IV), IEEE*, pages 1054–1059.

Garavel, H., Lang, F., Mateescu, R., and Serwe, W. (2013). Cadp 2011: a toolbox for the construction and analysis of distributed processes. *International Journal on Software Tools for Technology Transfer*, 15(2):89–107.

Ghete, I., Heizmann, M., Belkin, A., and Beyerer, J. (2010). World modeling for autonomous systems. In Dillmann, R., Beyerer, J., Hanebeck, U., and Schultz, T., editors, *KI 2010: Advances in Artificial Intelligence*, volume 6359 of *Lecture Notes in Computer Science*, pages 176–183. Springer Berlin Heidelberg.

Hessel, A., Larsen, K., Mikucionis, M., Nielsen, B., Pettersson, P., and Skou, A. (2008). Formal methods and testing. chapter Testing real-time systems using UPPAAL, pages 77–117. Springer-Verlag, Berlin, Heidelberg.

Hwang, G.-H., Tai, K.-C., and Huang, T. (1994). Reachability testing: an approach to testing concurrent software. In *Proceedings of the Software Engineering Conference, 1994 First Asia-Pacific*, pages 246–255.

Iqbal, M., Arcuri, A., and Briand, L. (2012). Empirical investigation of search algorithms for environment model-based testing of real-time embedded software. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, ISSTA 2012, pages 199–209, New York, NY, USA. ACM.

Jacoff, A., Messina, E., Weiss, B., Tadokoro, S., and Nakagawa, Y. (2003). Test arenas and performance metrics for urban search and rescue robots. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, volume 4, pages 3396–3403 vol.3.

Lei, Y. and Carver, R. H. (2006). Reachability testing of concurrent programs. *IEEE Trans. Softw. Eng.*, 32(6):382–403.

Li, J. and Wong, W. (2002). Automatic test generation from communicating extended finite state machine

(CEFSM)-based models. In *Proceedings of 5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. (ISORC 2002)*, pages 181–185.

Lill, R. and Saglietti, F. (2012). Model-based testing of autonomous systems based on coloured petri nets. In *ARCS Workshops (ARCS)*, pages 1–5.

Nguyen, C., Perini, A., Bernon, C., Pavon, J., and Thangarajah, J. (2011). Testing in multi-agent systems. In *Proceedings of the 10th International Conference on Agent-oriented Software Engineering*, AOSE'10, pages 180–190, Berlin, Heidelberg. Springer-Verlag.

Ran, L., Dyreson, C., Andrews, A., Bryce, R., and Mallery, C. (2009). Building test cases and oracles to automate the testing of web database applications. *Inf. Softw. Technol.*, 51(2):460–477.

Rehman, S. and Nadeem, A. (2013). Testing of autonomous agents: A critical analysis. In *Proceedings of the 2013 Saudi International Electronics, Communications and Photonics Conference (SIECPC)*, pages 1–5.

Shafique, M. and Labiche, Y. (2013). A systematic review of state-based test tools. *International Journal on Software Tools for Technology Transfer*, pages 1–18.

Shirole, M. and Kumar, R. (2013). Uml behavioral model based test case generation: A survey. *Softw. Eng. Notes, SIGSOFT*, 38(4):1–13.

Sighireanu, M., Chaudet, C., Garavel, H., Herbert, M., Mateescu, R., and Vivien, B. (2000). Lotos nt user manual.

Tretmans, J. (2008). Model based testing with labelled transition systems. In Hierons, R., Bowen, J., and Harman, M., editors, *Formal Methods and Testing*, volume 4949 of *Lecture Notes in Computer Science*, pages 1–38. Springer Berlin Heidelberg.

U.S. Department of Transportation, F. H. A. F. (2013). Manual on uniform traffic control devices MUTCD.

Utting, M., Pretschner, A., and Legeard, B. (2012). A taxonomy of model-based testing approaches. *Softw. Test. Verif. Reliab.*, 22(5):297–312.

Yang, R. and Chung, C.-G. (1990). A path analysis approach to concurrent program testing. In *Proceedings of the 9th Annual International Phoenix Conference on Computers and Communications*, pages 425–432.

Zander, J., Schieferdecker, I., and Mosterman, P. J. (2012). *Model-based testing for embedded systems*. CRC Press, 6000 Broken Sound Parkway NW,Boca Raton, FL 3348, USA, first edition.