Active World Model for Testing Autonomous Systems Using CEFSM

Anneliese Andrews
Department of Computer Science
University of Denver
Denver, CO 80208 USA
Email: andrews@cs.du.edu

Mahmoud Abdelgawad
Department of Computer Science
University of Denver
Denver, CO 80208 USA
Email: abdelgaw@cs.du.edu

Ahmed Gario
Department of Computer Science
University of Denver
Denver, CO 80208 USA
Email: agario@du.edu

***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 a dynamic world. We use Communicating Extended Finite State Machine (CEFSM) to illustrate an active world model that describes behaviors of environmental factors (i.e., actors). Abstract World Behavioral Test Cases (AWBTCs) are then generated by covering the active world model using graph coverage criteria. We also generate *test-data* by input-space partitioning to transform the generated AWBTCs into executable test cases. We apply the World Model-based Test Generation (WMBTG) technique to a case study from the Human-Robot Interaction domain (HRI) specifically a tour-guide robot. Reachability of the active world model and efficiency of coverage criteria are also discussed.**

I. Introduction

Autonomous systems are commonly defined as those systems that are able to accomplish entirely or in part certain tasks/goals without human intervention [1,2]. Autonomous systems exist in various applications such as driverless vehicles (so-called Unmanned Vehicles) [3], Search and Rescue robots (SaR) [4], and Human-Interaction Robots (HRI) [5]. As such, the robotic vacuum cleaner (Roomba) is a prime example of autonomous systems [6]. 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 uncertainty of their surroundings. In order to generate behavioral test cases in the form of simultaneous world stimuli, Model-based Testing (MBT) is able to leverage behavioral models, such as CEFSM [7], Coloured Petri Nets (CPN) [8], Labelled Transition Systems (LTS) [9], and sequence and communication diagrams in Unified Modeling Language (UML) [10], to describe the behavioral scenarios that can occur between the System Under Test (SUT) and its world. This requires testing solutions to deal with the large number of possibilities of the behavioral scenarios. Current MBT approaches for testing Real-time Embedded Systems (RTES) interacting with their worlds assume a static world model [11]. However, for autonomous systems, the world cannot be described only statically with attributes and values, the world should also be presented and used for testing dynamically.

To address these challenges, we propose a systematic MBT approach, World Model-based Test Generation (WMBTG), that identifies *what, where and how* to use worlds for testing autonomous systems inteacting with their surroundings. Test cases are generated by aggregating test paths in the individual models. These test paths are grouped as concurrent test paths which can be used with simulators or test-harnesses to validate autonomous systems. WMBTG has been introduced, in our previous work, and was applied to the unmanned vehicle application domain [12]. We extend the applicability of WMBTG to the Human-Robot Interaction (HRI) domain. We also evaluate the efficiency of test path coverage criteria used to generate Abstract World Behavioral Test Cases (AWBTs). We evaluate input-space partitioning coverage criteria [13] used to generate *test-data*. We use UML class diagrams to construct the structural model of actors and their relationships. We also use CEFSM [7,14] to represent landscapes in worlds. We call these landscapes *snippets*. *Snippets* are used to link the behavioral models of various actors that are involved in this world. We also explore the applicability of the MBT technique for testing autonomous systems behaviors in dynamic worlds in [12] alongside behavioral testing that is flexible, systematic, scalable, and shows that this technique is extendable to other domains of autonomous systems and to other types of behavioral models.

The remainder of this paper is organized as follows. Section II discuses the state of research related to MBT, testing autonomous systems, and World Model-based Testing. Section III defines the human-robot interaction domain, explores HRI classifications, and describes the case study. Section IV presents our approach and applies it to the case study. We analyze and discuss reachability and efficiency issues in section V. Section VI draws conclusions and future work.

II. State of Research

A. Model-based Testing (MBT)

According to Dias-Neto *et al.* [15], 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.* [16] define MBT as an algorithm that generates test cases

automatically from models instead of creating them manually. Utting *et al.* [16,17] 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 techniques as state, history, functional, operational, stochastic, and transition based. Shirole *et al.* [10] present a survey on model-based test generation from UML behavioral specification diagrams (sequence, state chart and activity diagrams). They classify the various research approaches based on formal specifications, graph-based, heuristics, and direct UML specification processing. In graph-based testing, a test case is a path that covers some specific system requirement. Test case generation from graphs includes the following steps: build a graph model, identify test requirements, select test paths to cover those requirements, and derive test data. Shafique and Labiche [18] present a systematic review to determine the current state of the art of MBT tool support. They scope their study to tools that use state-based models: FSM, Extended FSM, Abstract State Machine (ASM), State-chart, UML state machine, (timed, input/output)-automata, Harel Statechart, Petri Net, State Flow diagram and Markov chain. They grouped MBT tools based on test criteria similarity. Twelve MBT tools are selected for primary studies. A comparison enables tool selection based on project needs. The literature shows that CEFSM is practicable for representing dynamic behaviors including dependency, concurrency, and communication.

B. Testing Autonomous Systems

The techniques for testing autonomous systems are mostly computer-based simulation and test fields/arenas. Jacoff *et al.* [19], by the National Institute of Standards and Technology (NIST), introduce a standard for designing and evaluating test arenas *(Reference Test Arena for Autonomous Mobile Robots)*. The test arenas consist of collapsed structures that are designed from buildings in various stages of the collapse. Pepper *et al.* [20] illustrate a computer-based simulation technique for evaluating USAR robots using *USARSim*, a robot simulation tool. The computer-based simulation is flexible, repeatable, and accurate compared with physical test fields/arenas; however, it lacks physical fidelity. Both techniques, computer-based simulation and test fields/arenas, also limit possible behavioral scenarios that may occur in autonomous system worlds. Lill *et al.* [8] use a MBT technique for testing autonomous systems. The authors, first, 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 co-operating autonomous systems. The comparison is based on four evaluating criteria (understandability, well-definedness, scalability, and testability). They then select CPNs to model a factory robot that carries a load from one place to another. Four behavioral transitions *(look ahead, raise alarm, go ahead, mission completed)* were modeled. Obstacle passing is not considered. They also define coverage criteria tailored to the characteristics of CPNs, such as color-based and event-based coverage criteria. Color-based coverage criteria focus on the consumption of tokens that belong to pre-defined color sets. Event-based coverage criteria focus on the CPN events, where an event is defined as a transition together with enabling variables. This work does not provide validation, nor does it address dynamic worlds. The authors however found MBT is the most promising option for testing autonomous systems.

C. World Model-based Testing

Most approaches in the literature for 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. In our approach, the world is considered as independent actors interacting with the SUT instead of being part of the system. These approaches are mostly for the purpose of increasing the understandability of the decision-making module to the relevant surroundings in order to implement proper, efficient, and safe behaviors, but they are not aiming for model-based testing. Gheta *et al.* [21] contribute an intelligent information storage and management system approach for autonomous systems with the aim of modeling the world of an autonomous system. The world model is represented as instances of classes with class specific attributes and relations. Furda and Vlacic [22] also present an object-oriented world model approach for the road traffic environment of autonomous vehicles. The approach uses UML class diagrams to represent the structure of the world actors. The authors conducted an experiment, a field trial, using two autonomous vehicles. The experiment illustrates that the world model strongly supports the decision-making module for making appropriate driving decisions in real-time. Nevertheless, this work neither intends to be a testing technique for autonomous systems nor does it handle the dynamic aspect of world actors.

A closely related approach for world model-based testing is presented in [23]. It generates black-box test cases automatically based on a 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. They also use Object-Constraints Language (OCL)for specifying environmental constraints. 2) Test oracles are then generated automatically from the world model. They use a simulator to observe actual response. 3) To identify feasible test cases and maximize possibilities of fault detection, heuristic algorithms such as Random Testing (RT), Adaptive Random Testing (ART), and Search-based Testing (SBT) (specifically, Genetic algorithm and (1+1) Evolutionary algorithm) 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 these algorithms. Recently, this approach has been enhanced and applied to automotive sensor system [24]. This

approach generates use cases automatically by using Natural Language Processing (NLP). The approach (so called Use Case Modeling for System Tests Generation (UMTG)), is applied to BodySense system. BodySense system monitors a car seat to classify the occupants. It disables the airbag for children and unoccupied seats while it enables airbag for adults. It also includes a seat belt reminder function. The result indicates that test requirements generated by UMTG are entirely feasible. However, this approach limits the world model to a static world. It is also specified for testing real-time embedded systems (RTES). It is not applicable for autonomous systems because their worlds are dynamic.

III. Application description

In this paper, we consider tour guide-robot applications. According to [5,25,26], a tour guide-robot is classified as a highly autonomous, serviceable, anthropomorphic, navigational, and social HRI robot. Tour guide-robots usually perform in indoor sensory environments where sensors are placed ubiquitously for perceiving static/mobile objects. Tour guide-robots use the sensory environment to understand their surroundings and to localize themselves. Museum, campus, shopping arcade, train station, and hotel-lobby are examples of indoor sensory environments. MacDougall *et al.* [27] present a sensory environment. They built the sensory environment in the Electrical and Computer Engineering Department of Kettering University for the purpose of college tours. They then conducted a field trial on a tour guide-robot that gives tours to visitors. Although, experimenters use visitors' opinions (questionnaire), which is not enough for validation, these experiments illustrate that the tour guide-robots are capable to be proximate, conversational, serviceable, and sociable. Burgard *et al.* [28] conducted a field trial on a tour guide-robot, RHINO, in the Deutsche Museum Bonn. For six days, RHINO gave tours to more than 2000 visitors. Thrun *et al.* [29] experimented with a tour guide-robot, MINERVA, in the Smithsonian National Museum of American History. MINERVA successfully educated and entertained many thousand visitors. Socially, MINERVA is compared with RHINO. A key difference between both robots relates to their interactive capabilities. RHINO acts more rudimentarily and does not exhibit emotional states, while MINERVA behaves more effectively in attracting people and making progress. The Microsoft Research Team [30] also conducted a field trial on a tour guide-robot (a humanoid robot (NAO), from Aldebaran Robotics, France). The experiment focuses on conversational engagement, handling queries, and providing directions to visitors. The experimental results show successful conversational engagements with individuals but not multiparty conversational situations. The HRI literature with the most relevance to the guide-robot applications comes from a series of studies on *Robovie*, a humanoid robot invented by Advanced Telecommunications Research (ATR) Institute, Japan. Therefore, we select *Robovie* as a System Under Test (SUT) in our case study.

1) Tour-guide Robot (Robovie): *Robovie* is an interactive humanoid robot performing human-like physical expressions [31]. It has a head with two eyes, two arms, a body, and wheels for mobility. Its mobile platform includes two driving wheels and one free wheel. *Robovie* is equipped with 10 tactile sensors, an omnidirectional vision sensor, two microphones to listen, and 24 ultrasonic sensors for detecting obstacles. The eyes have a pan-tilt mechanism with direct-drive motors, and they are used for stereo vision and gaze control. It also has skin sensors for realizing interactive behaviors. *Robovie* communicates with its sensory environment via wireless LAN. Environmental sensors that *Robovie* communicates with usually are Laser Range Finders (LRFs). Many cognitive experiments were conducted on *Robovie* to increase its behavioral skills. For instance, *Robovie* can predict human behaviors. This skill was added recently to *Robovie* for the purpose of escaping from children's abuse [32]. *Robovie* can perform meaningful interactive-behaviors for a human. For example, it can *gaze, gesture, greet, converse, listen, assist, follow, accompany and guide* people. *Robovie* also can perform in various sensory environments. Kanda *et al.* [33] present a field trial conducted on *Robovie* at a shopping mall for five weeks. Each day, approximately 100 groups of customers signed up to interact with the robot. The experimenters observe *Robovie's* interaction with customers. They also collect feedback using a questionnaire. The findings show that the customers accepted the robot with positive impressions. In [34] and [35], the authors present the results of two field trials on *Robovie* at a train station. Its task includes greeting, providing directions, and advertising. The experimenters also consider this robot being capable to elicit spontaneous participation from pedestrians. This experiment investigates the robot's technical performance and its attitudes. The authors illustrate fine results in both considerations except speech recognition. Shiomi *et al.* [36] also present a field trial conducted on *Robovie* at Osaka Science Museum for two months. The robot is assigned to welcome, guide, and provide scientific information about the exhibits. In this field trial, the target visitors are children, therefore *Robovie* expresses childlike behaviors such as handshaking, hugging, and free-playing. The findings indicate that performing childlike interactions effectively attracts visitors' attention for scientific explanations. These field trials were conducted with a few interventions (so-called semi-autonomous trial) by operators (humans). Operators use ubiquitous cameras to watch participants and start the robot for greeting when a participant talks to it. They also intervene when the robot encounters a critical situation. In compliance with these field trials, one can image that *Robovie* generally deals with three types of environmental actors (people, obstacles, and environmental sensors). Also, to present an active world model, we assume that this robot is fully autonomous.

2) Crowds in a train station: As mentioned, *Robovie* can perform in various sensory environments where crowds can be formed. A crowd is defined as a large group of individuals in the same physical environment, sharing a common goal [37]. Each crowd has a set of behaviors resulting from world actors. For instance, in a museum, an attendee can attend, ask, film, and leave an exhibit. In a train station, crowds can be

Fig. 1: Shopping Arcade in a Train Station, [35]

formed over many snippets (entry gate, train doors, shopping arcade, restaurant, exit gate). Each snippet has a somewhat different set of behaviors from other snippets. At a restaurant, for example, people can walk, talk, order, sit, eat, etc. At the entry gate, people run, sometimes push and shove, follow, pass, etc. In both locations, people engage in common behaviors such as talking, listening, gesturing, gazing, etc. In addition, people naturally do these behaviors simultaneously. Therefore, crowd behaviors are infinite and simultaneous. [37] explores the crowd behaviors in detail.

In this paper, the shopping arcade in the train station presented in [35] is considered as *a world snippet* of the case study. As shown in Fig. 1, the shopping arcade in a train station consists of participants, obstacles, and four LRFs mounted around the trial area. In [35], the authors classify the participants as *addressees, side-participants, bystanders*, and *pedestrians*. *Addressee* is a person in front of the robot listening to and following. *Side-participants* are participants accompanying an addressee. *Addressee* and *Side-participants* are responsible for responding to the robot. *Bystanders* are participants encouraged by the robot, *addressee*, or *side-participants* but they are not responsible for interacting with the robot. *Pedestrians* are people who are not arranged in any of these classes. Obstacles are classified as *Mobile* obstacles (e.g., drivable cleaning-cart) and *Static* obstacles like "caution: wet floor" sign. The four LRFs are assigned to sense the obstacles and the participants' movements in the trial area and provide the sensory data to the robot. Fig. 2 visualizes a set of shopping arcade actors.

Fig. 2: A set of shopping arcade actors

The areas also are classified into two types, Area of Audience (AoA) and Area of Passing (AoP) [35]. AoAs are locations where pedestrians tend to become members of the audience like a restaurant. AoPs are locations where pedestrians tend to keep passing, for instance entry and exit gates, see [35]. The shopping arcade is considered as AoA.

IV. Approach

Our objective is to apply a systematic model-based test generation approach [12] to generate test cases from an active world model that represents world actors of an autonomous system. There are a multitude of world actors. They can be mobile or static. World actors also act independently,

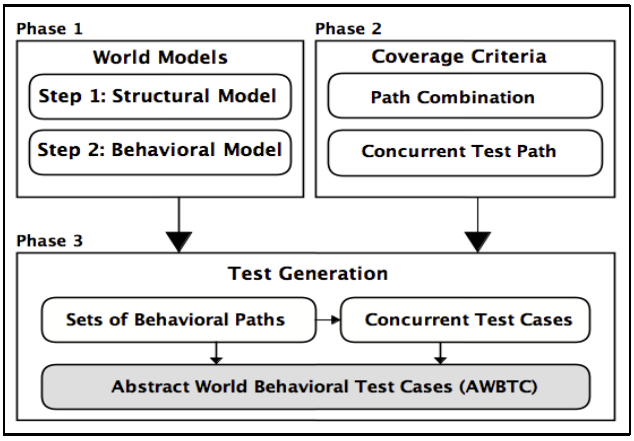

Fig. 3: World Behavioral Test Generation Process

simultaneously, and unpredictably. To avoid scalability and complexity issues of the dynamic worlds, we concentrate on actors that autonomous systems are dealing with and the behavioral messages that autonomous systems can perceive from these actors. The locations where actors interact are also considered. In other words, a set of actions that a group of actors can perform may occur over a particular snippet. For instance, when travelers stop by a shopping arcade in a train station, behaviors can be walking or standing.

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' possible states and transitions and their interactions. Each actor is presented by one behavioral model showing its behavioral messages. The interactions between these actors represent the active world model. These interactions need to be modeled by a communicating behavioral semantic model such as CEFSM. As such in our application, actors are interacting simultaneously, the active world model should cover not only the internal transitions of actors, but also the interactions between them. The active world model can then be leveraged to generate world behavioral test cases. Once we build the active world model, any member of the graph-based testing criteria from [8,13] can be used to generate abstract behavioral test paths, which are AWBTCs. Finally, we generate *test-data* by input-space partitioning to transform the generated AWBTCs into executable test cases. The test generation process is illustrated in Fig. 3. The world Model-based test generation process follows three phases:

- Model the active world by constructing structural and then behavioral models.

TABLE I: Instances of Shopping Arcade Actors

Class	Actor Instance	Behavioral message example
Participant	An elderly person, an adult, a teen, and a child	child.attend() child.behave(talk) elderly.behave(gaze)
Mobile obstacle	Cleaning cart and maintenance cart	cleaningCart.appear() cleaningCart.act(moveBW) cleaningCart.act(flashingLight)
Static obstacle	Caution signs, flash lights, siren, and fire alarms	caution.appear() caution.inform("Wet Floor") alram.inform("siren")
LRF	LRF1, LRF2, LRF3, and LRF4	LRF1.on() LRF1.detect(participant)

- Select proper graph-based coverage criteria for test-path generation and proper input-space partitioning coverage criteria for *test-data*.
- Generate AWTCs which are concurrent test paths and then generate *test-data* to transform these concurrent test paths into executable test cases.

A. Phase 1: World Models

1) Structural Model: The structural model is constructed using a UML class diagram, where classes represent actors including their important characteristics, behavioral messages, and relationships. In our application, the shopping arcade in a train station can be represented by a single snippet *"Crowd"*. World actors that are considered to perform in this snippet are of three types: *participants (humans), obstacles,* and *LRFs*. A participant can be addressee, side-participant, bystander, or pedestrian [35]. Obstacles also are classified as mobile or static. Pedestrians and mobile obstacles perform independently

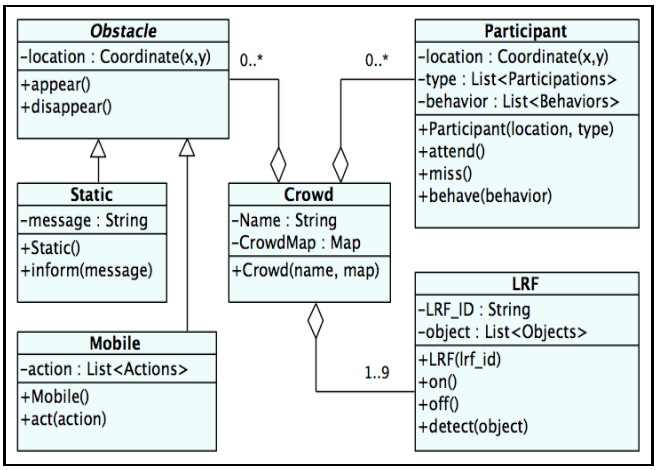

Fig. 4: Structural Model for Shopping Arcade

and concurrently several behaviors (communicated via messages); however, static obstacles inform messages only. LRFs detect objects that appear in the *Crowd* and describe these objects to the robot. However, these LRFs do not interact with other world actors except the robot. The UML class diagram that represents the shopping arcade in a train station and its world actors is shown in Fig. 4. The actors are aggregated into a *Crowd* snippet. Similar actors are generalized to a single class. For instance, *Static* and *Mobile* obstacle are generalized into the *Obstacle* class. The number of actors in the *Crowd* is determined by their multiplicity relationship. Instances of shopping arcade actors and examples of their behavioral messages are illustrated in Table I. We assume that only one robot performs in a snippet; this robot is not considered a world actor as it is the SUT.

2) Behavioral Model: Although a wide range of behavioral models exists, we illustrate the behavioral model using communicating extend finite state machine (CEFSM). 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 [7,38]. *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 trigger the transition function, $m_{i_1},..., m_{i_k}$ are the messages. CEFSM is a generalization of an EFSM [39] (i.e., adding communication channels between EFSMs). Modeling behavioral models follows two steps. First, we

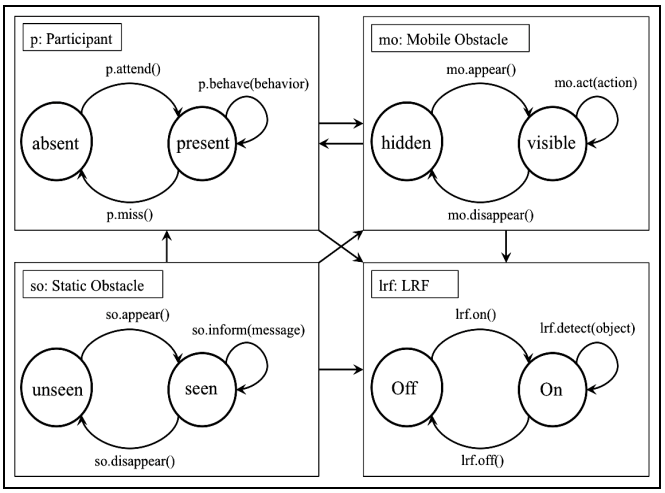

Fig. 5: Behavioral Models for Shopping Arcade Actors

model each individual actors as EFSMs. Then we model the interaction between these actors as CEFSM. Fig. 5 shows a set of CFSM that represents a group of shopping arcade actors interacting with each other. It is clear that a participant

actor can express multiple behavioral messages simultaneously. For instance, a participant can walk and talk at the same time. Similarly, mobile obstacles also can concurrently reveal several behavioral messages. For instance, a cleaning-cart moves forward/backward and flashes its lighting-alarm at the same time. However, in some cases, exhibiting different behavioral messages concurrently is infeasible. For example, a pedestrian cannot sit and walk at the same time. Therefore, proper input-space partitioning criteria can be used to exclude the infeasible combinations of behavioral messages. However, static obstacles inform by messages only. For example, a cautionary sign shows a *wet floor* message. As shown in Fig. 5, a participant initially is out of the crowd (*absent*). When this participant attends the crowd, the *attend()* transition fires and the participant moves to *present* place. This participant then starts a behavior. Whenever this participant behaves, the *behave()* transition fires, the behavioral message reveals, and the participant moves to *present* state again. This process can occur at will. It is similar to other actors' processes. The key difference is that the behavioral/information messages these processes reveal are dissimilar. Secondly, these behavioral models (EFSMs) that represent world actors are linked together into a higher level behavioral model which describes the interactions among these actors. Fig. 6 illustrates the high level behavioral model. This model shows that participants can interact with mobile obstacles and vice-versa. However,

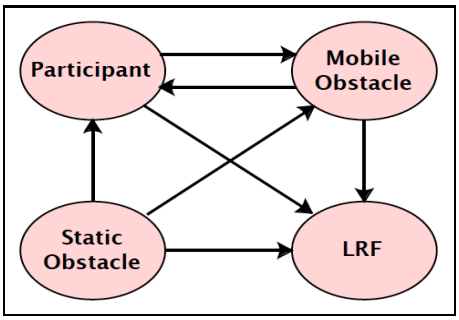

Fig. 6: Behavioral Model (High-Level)

static obstacles can only show messages to other actors (i.e., unidirectional interaction). A LRF also is a unidirectional actor due to the fact that they detect objects only.

B. Phase 2: Coverage Criteria

Since each actor is represented as a EFSM (a process), an active world behavioral model can be defined as a collection of concurrent processes, $AWM = \{P_1, P_2, \ldots, P_i\}$ where $1 \leq i \leq M$ and M is the number of actors that share a snippet. *Model-flow coverage* criteria such as node (state-based) coverage, edge (transition-based) coverage, etc. [8,13], can be applied. Using any of a number of test path generation techniques, test paths that fulfill these coverage criteria can then be generated. Let $TP_i = (tp_{i1}, tp_{i2}, ..., tp_{ik})$ be a set of such internal test paths that cover the process P_i and k is the number of these internal test paths. These internal paths describe the internal execution (possible behaviors) of the processes. We use *transition-based* coverage [7] to generate internal paths that cover the processes of shopping arcade actors. In the shopping arcade, as illustrated in Table II, each of these

TABLE II: Internal Test Path Sets

1. Participant process, $TP_1 = \{tp_{11}\}$ $tp_{11} : absent \xrightarrow{p.attend()} present \xrightarrow{p.behave(behavior)} present \xrightarrow{p.miss()} absent$
2. Mobile obstacle process, $TP_2 = \{tp_{21}\}$ $tp_{21} : hidden \xrightarrow{mo.appear()} visible \xrightarrow{mo.act(action)} visible \xrightarrow{mo.disappear()} hidden$
3. Static obstacle process, $TP_3 = \{tp_{31}\}$ $tp_{31} : unseen \xrightarrow{so.appear()} seen \xrightarrow{so.inform(message)} seen \xrightarrow{so.disappear()} unseen$
4. LRF process, $TP_4 = \{tp_{41}\}$ $tp_{41} : off \xrightarrow{lrf.on()} On \xrightarrow{lrf.detect(object)} On \xrightarrow{lrf.off()} Off$

processes TP_i is coincidentally covered by one internal test path tp_{i1} only because the size of these processes is small. The test path sets, (TP_1, TP_2, TP_3, TP_4), interact concurrently with each other via the exchange of behavioral/information messages (i.e., interaction messages). These interaction messages represent the high level of execution behavior of the active world model, as shown in the behavioral model (High-Level), in Fig. 6. The interaction among the processes can be covered by interaction test paths which represent the possibilities of execution behavior of the interaction messages. To avoid cyclic paths, the *Simple-path* coverage criterion [13], is used to generate the interaction test paths that cover the shopping arcade interaction messages. Table III shows six simple paths, Interaction Test Paths $(ITP_1, ITP_2, \ldots, ITP_6)$,

TABLE III: Interaction Test Paths

$ITP_1 : (participant \rightarrow mobile\ obstacle \rightarrow participant)$
$ITP_2 : (participant \rightarrow LRF)$
$ITP_3 : (mobile\ obstacle \rightarrow LRF)$
$ITP_4 : (static\ obstacle \rightarrow participant)$
$ITP_5 : (static\ obstacle \rightarrow mobile\ obstacle)$
$ITP_6 : (static\ obstacle \rightarrow LRF)$

that cover the high level of the active world model for shopping arcade actors. Each interaction test path combines the internal test paths of processes TP_i that are involved in the interaction scenario. For instance, ITP_1, shown in Table III, covers the interaction between the participant process and the mobile obstacle process. Thus, the internal paths of this interaction is $(tp_{11} \rightarrow tp_{21} \rightarrow tp_{11})$. However, the behaviors vary due to the non-deterministic interactions between the internal test paths. For instance, in the shopping arcade, when a participant moves ahead of a cleaning-cart, the cleaning-cart may stop to give the

participant free way, or it may keep moving and alerting the participant by a beep. The interaction test paths are considered concurrent paths. The concurrent interaction between internal test paths that represent multiple processes produces a number of possible combinations of internal paths. As a result, we have two types of coverage criteria, path combination and concurrent test path coverage criteria.

1) Path Combination: In order to cover all possible combinations of internal paths, path combination coverage criteria should determine what combinations are required. Let $(TP_1, TP_2, \ldots, TP_n)$ be sets of internal test paths for $(P_1, P_2, \ldots, P_n)$ where $TP_1 = \{tp_{11}, tp_{12}, \ldots, tp_{1k}\}$, $TP_2 = \{tp_{21}, tp_{22}, \ldots, tp_{2k}\}$, $\ldots$, and $TP_j = \{tp_{j1}, tp_{j2}, \ldots, tp_{jk}\}$. Then, the selection of a tp_{1i} from TP_1 and a tp_{2j} from TP_2 is called a path combination. Let len(tp_{ij}) be the number of nodes in tp_{ij}, the length of tp_{ij}. The combination set for interaction test path ITP_i, $Comb_{ITP_i} = \{(tp_{jk}, \ldots, tp_{mn}) | tp_{mn} \in TP_m,\ m = len(ITP_i),\ n = |TP_i|,\ 1 \leq j \leq m,\ 1 \leq k \leq n\}$. The number of all path combinations of ITP_1, for instance, will be the product of the number of internal paths of TP_1, TP_2, and TP_1. Each combination introduces a set of concurrent test paths.

2) Concurrent Test Path Coverage Criteria: The path combination sets do not show how these paths interact concurrently. Therefore, concurrent test path coverage criteria are required. In this work, we apply the all possible serialized execution sequences coverage criterion. We also use the *Rendezvous* coverage criterion, as in [14]. These concurrent test path coverage criteria are defined as follows:

- All Possible Serialized Execution Sequences Coverage Criterion (APSESCC): Test requirements contain a set of all possible serialized nodes of the test paths that are included in each path combination, i.e. each node in test path tp_{ij} can be triggered by each node in test path tp_{mn} and vice versa. For example, let tp_{ij} be $a \rightarrow b$ and tp_{mn} be $x \rightarrow y$, where tp_{ij} and tp_{mn} are in the same path combination. All serialized execution sequences of path combination $c_{xy} = (tp_{ij},\ tp_{mn})$ 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, the all serialized execution sequences will be 20 possible serializations. All possible number of serializations of nodes is

  $$\sum_{i=1}^{|Comb_{ITP_i}|} \left(\frac{(\sum_{j=1}^{len(c_{ij})} |tp_{ij}|)!}{\prod_{j=1}^{len(c_{ij})} (|tp_{ij}|)!}\right).$$

- 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 interaction test path ITP_i is $\prod_{j=1}^{n} (TP_j + 1) - 1$.

3) Input-space Partitioning Coverage Criteria: The generated concurrent test paths are still abstract. To transform these concurrent test paths into executable test cases, *test-data* coverage criteria, i.e. input-space partitioning [13], also are required. The input-space partitioning criteria can be considered as methods to divide a collection of values (*input-domain*) into *test-data* blocks that make the concurrent test paths executable. The *input-domain* is the set of possible values that input variables can take on. In the shopping arcade snippet, the behavior execution of actors is controlled by five input-domains: *participant type, participant behaviors, mobile obstacle actions, static obstacles* and *LRF detected objects*. There is one block for each. The Participant type block includes {*addressee, side participant, bystander, and pedestrian*}. The Participant behavior block consists of values {*sit, walk, listen, talk, gaze, gesture, eat, drink*}. The mobile obstacle actions block consists of {*move forward, move backward, turn right, turn lift, beep, flash lights*} while the static obstacles block includes the messages {*"Wet Floor", "Do Not Enter"*}. The LRF detected objects block contains {*participant, mobile obstacle, and static obstacle*}. In this work, we use All Combinations Coverage (ACoC) which exercises all possible combinations of *test-data*. The number of *test-data* sets that satisfy ACoC is $\prod_{i=1}^{Q} (B_i)$, where B_i is a block of values for a parameter and Q is the number of parameters. To compare with ACoC, we also use Each Choice Coverage (ECC) that selects one value from each block of values. The number of *test-data* that satisfy ECC is $MAX_{i=1}^{Q} (B_i)$ [13].

C. Phase 3: Test Generation

The path combinations are represented as ordered references to internal test paths of the processes involved in the execution. These combinations may result in a huge number of concurrent test paths, AWBTCs, although not all of these concurrent test paths are feasible. We used the serialization algorithm in [14] to generate these concurrent test paths. The concurrent test paths are serialized nodes of the internal paths. We expressed the concurrency of test paths using double-bar "||" as used in LOTOS for defining concurrency [40]. In the shopping arcade snippet, each process involved in the interaction is satisfied by one internal test path only. As a result, each interaction test path is composed of one combination which represents the concurrent test path, an AWBTC. Table IV shows the path combinations and the AWBTCs that satisfy the interaction test paths presented in Table III. Six path combinations are created for covering interaction test paths of the shopping arcade; consequently, six AWBTCs are generated. This number of AWBTCs is reasonable for this small number of actor processes. Nevertheless, when we impose the APSESCC and RCC to serialize these AWBTCs, the total number of test paths serialized by the APSESCC is 35000 serialized paths while the RCC produces 244 rendezvous paths. To transform these AWBTCs into executable test cases, we also apply ACoC and ECC coverage criteria to generate *test-data* that meet these AWBTCs. The five blocks of values described in section IV-B3 meet these criteria. For each interaction test path ITP_i, there

TABLE IV: Combinations of Concurrent Test Paths

1. Combination $Comb_{ITP1}(tp_{11}, tp_{21}, tp_{11}) = AWBTC_1$: $(tp_{11}[absent \xrightarrow{p.attend()} present \xrightarrow{p.behave(behavior)} present \xrightarrow{p.miss()} absent] \xrightarrow{\|} tp_{21}[hidden \xrightarrow{mo.appear()} visible \xrightarrow{mo.act(action)} visible \xrightarrow{mo.disappear()} hidden] \xrightarrow{\|} tp_{11}[absent \xrightarrow{p.attend()} present \xrightarrow{p.behave(behavior)} present \xrightarrow{p.miss()} absent])$
2. Combination $Comb_{ITP2}(tp_{11}, tp_{41}) = AWBTC_2$: $(tp_{11}[absent \xrightarrow{p.attend()} present \xrightarrow{p.behave(behavior)} present \xrightarrow{p.miss()} absent] \xrightarrow{\|} tp_{41}[off \xrightarrow{lrf.on()} On \xrightarrow{lrf.detect(object)} On \xrightarrow{lrf.off()} Off])$
3. Combination $Comb_{ITP3}(tp_{21}, tp_{41}) = AWBTC_3$: $(tp_{21}[hidden \xrightarrow{mo.appear()} visible \xrightarrow{mo.act(action)} visible \xrightarrow{mo.disappear()} hidden] \xrightarrow{\|} tp_{41}[off \xrightarrow{lrf.on()} On \xrightarrow{lrf.detect(object)} On \xrightarrow{lrf.off()} Off])$
4. Combination $Comb_{ITP4}(tp_{31}, tp_{11}) = AWBTC_4$: $(tp_{31}[unseen \xrightarrow{so.appear()} seen \xrightarrow{so.inform(message)} seen \xrightarrow{so.disappear()} unseen] \xrightarrow{\|} tp_{11}[absent \xrightarrow{p.attend()} present \xrightarrow{p.behave(behavior)} present])$
5. Combination $Comb_{ITP5}(tp_{31}, tp_{21}) = AWBTC_5$: $(tp_{31}[unseen \xrightarrow{so.appear()} seen \xrightarrow{so.inform(message)} seen \xrightarrow{so.disappear()} unseen] \xrightarrow{\|} tp_{21}[hidden \xrightarrow{mo.appear()} visible \xrightarrow{mo.act(action)} visible \xrightarrow{mo.disappear()} hidden])$
6. Combination $Comb_{ITP6}(tp_{31}, tp_{41}) = AWBTC_6$: $(tp_{31}[unseen \xrightarrow{so.appear()} seen \xrightarrow{so.inform(message)} seen \xrightarrow{so.disappear()} unseen] \xrightarrow{\|} tp_{41}[off \xrightarrow{LRF.ON()} On \xrightarrow{LRF.detect(object)} On])$

is a set of *test-data* that fulfills at least one concurrent test path that belongs to this ITP_i. This set of *test-data* is selected from blocks that only are related to the actor processes involved in the ITP_i. For instance, ITP_1 represents the interactions between participant and mobile obstacle; as a result, three blocks (participant types, participant behaviors, and mobile obstacle actions) are used to generate *test-data* for ITP_1. The ACoC results in 429 *test-data* while the ECC produces 39. When these 429 test-data are used with the 35000 serialized paths, this results in 6669486 executable test cases. On the other hand, the number of executable test cases generated by ECC with RCC is 1736. Using APSESCC with ACoC is clearly impractical; however, it presents the upper bound of test cases. We also consider using ECC with RCC as a lower bound.

V. Reachability & Criteria Efficiency

A. Reachability

To perform reachability analysis on the behavioral models, we use the Construction and Analysis of Distributed Processes (CADP) toolbox [41]. For generating all possible states that a system can reach, CADP transforms the LOTOS code that represents a CEFSM into a Labelled Transition System (LTS) graph. The reachability graph generated by CADP, for four concurrent processes with three nodes each, consists of 352 states connected by 1046 arcs. However, the number of reachable states grows exponentially as the number or the size of processes increase. For instance, for 8 processes, the number of reachable states expands to 1346 states with 3658 arcs. Although CADP is scalable up to 10^{13} nodes, a display in CADP is no longer easily readable.

B. Criteria Efficiency

It is clear that the number of generated concurrent test paths depends on several factors: the number of actor processes that are involved in the execution and the size of these processes, the combination criteria selected to combine the internal paths of these actor processes, the coverage criteria chosen to serialize these internal paths, and the coverage criteria chosen to generate *test-data*. We use APSESCC and RCC coverage criteria on six interaction test paths to serialize the generated AWBTCs. We then apply ACoC and ECC coverage criteria to generate test-data in oder to transform the serialized test paths into executable test cases. Table V illustrates the efficiency of these coverage criteria. As mentioned above, although exercising AWBTCs generated by APSESCC on *test-data* sets selected by ACoC is not practicable, it is considered as an upper bound. On the other hand, exploiting RCC on ECC is more feasible and efficient.

TABLE V: Test Criteria Efficiency

Interaction Test Path	***Test-paths* C.**		***Test-data* C.**		**APSESCC with ACoC**	**RCC with EEC**
	APSESCC	RCC	ACoC	ECC		
ITP_1	34650	124	192	8	6652800	992
ITP_2	70	24	96	8	6720	192
ITP_3	70	24	18	6	1260	144
ITP_4	70	24	96	8	6816	192
ITP_5	70	24	18	6	1260	144
ITP_6	70	24	9	3	630	72
Total	**35000**	**244**	**429**	**39**	**6669486**	**1736**

VI. Conclusion and Future work

This paper presented the applicability of a model-based test generation approach [12] that allows testing of autonomous systems in their active world. We modeled an active world of an autonomous system. A test generation process is applied. Path serialization techniques APSESCC and RCC are imposed. The APSESCC is also compared withx RCC. The findings indicate that RCC is practically feasible. To transform the generated AWBTCs into executable test cases, we also exploited ACoC and ECC coverage criteria to generate *test-data*. The findings also show that the number of executable test cases depends on the size of generated *test-data* and the size/number of actor processes that are involved in the execution. The CADP toolbox is used for reachability analysis. Future work will explore other testing techniques such as search-based testing techniques to handle path-selection and test-data generation of the concurrent processes. Future work will also investigate the effectiveness of this approach by executing the generated test cases.

VII. Acknowledgments

This work was supported, in part, by NSF IUCRC grant #0934413, 1127947, 1332078, and 1439693 to the University of Denver.

References

[1] L. Steels, "When are robots intelligent autonomous agents?" *Robotics and Autonomous Systems*, vol. 15, no. 12, pp. 3–9, 1995, the Biology and Technology of Intelligent Autonomous Agents.

[2] S. Franklin and A. Graesser, "Is it an agent, or just a program?: A taxonomy for autonomous agents," in *Intelligent Agents III Agent Theories, Architectures, and Languages*, 1997, pp. 21–35.

[3] H. Cheng, *Autonomous Intelligent Vehicles: Theory, Algorithms, and Implementation*, 1st ed. Springer London Dordrecht Heidelberg, New York: Springer-Verlag, 2011.

[4] Y. Liu and G. Nejat, "Robotic urban search and rescue: A survey from the control perspective," *Journal of Intelligent and Robotic Systems*, vol. 72, no. 2, pp. 147–165, 2013.

[5] M. A. Goodrich and A. C. Schultz, "Human-robot interaction: A survey," *Found. Trends Hum.-Comput. Interact.*, vol. 1, no. 3, pp. 203–275, Jan. 2007.

[6] Y.-W. Bai and M.-F. Hsueh, "Using an adaptive iterative learning algorithm for planning of the path of an autonomous robotic vacuum cleaner," in *Proceedings of the 1st IEEE Global Conference on Consumer Electronics (GCCE)*, 2012, pp. 401–405.

[7] J. Li and W. Wong, "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)*, 2002, pp. 181–185.

[8] R. Lill and F. Saglietti, "Model-based testing of autonomous systems based on coloured petri nets," in *ARCS Workshops (ARCS)*, 2012, pp. 1–5.

[9] J. Tretmans, "Model based testing with labelled transition systems," in *Formal Methods and Testing*, ser. Lecture Notes in Computer Science, R. Hierons, J. Bowen, and M. Harman, Eds. Springer Berlin Heidelberg, 2008, vol. 4949, pp. 1–38.

[10] M. Shirole and R. Kumar, "UML behavioral model based test case generation: A survey," *Softw. Eng. Notes, SIGSOFT*, vol. 38, no. 4, pp. 1–13, Jul. 2013.

[11] M. Iqbal, A. Arcuri, and L. Briand, "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*, ser. ISSTA 2012. New York, NY, USA: ACM, 2012, pp. 199–209.

[12] A. Andrews, M. Abdelgawad, and A. Gario, "Towards world model-based test generation in autonomous systems," in *Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD) 2015*. SCITEPRESS Digital Library, 2015, pp. 165–176.

[13] P. Ammann and J. Offutt, *Introduction to Software Testing*, 1st ed. 32 Avenue of the Americas, New York, NY 10013, USA: Cambridge University Press, 2008.

[14] R. Yang and C.-G. Chung, "A path analysis approach to concurrent program testing," in *Proceedings of the 9th Annual International Phoenix Conference on Computers and Communications*, Mar 1990, pp. 425–432.

[15] A. Dias-Neto, R. Subramanyan, M. Vieira, and G. H. Travassos, "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*. ACM, 2007, pp. 31–36.

[16] J. Zander, I. Schieferdecker, and P. J. Mosterman, *Model-based testing for embedded systems*, 1st ed. CRC Press, 2012.

[17] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," *Softw. Test. Verif. Reliab.*, vol. 22, no. 5, pp. 297–312, August 2012.

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

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

[20] C. Pepper, S. Balakirsky, and C. Scrapper, "Robot simulation physics validation," in *Proceedings of the 2007 Workshop on Performance Metrics for Intelligent Systems*, ser. PerMIS '07. New York, NY, USA: ACM, 2007, pp. 97–104.

[21] I. Ghete, M. Heizmann, A. Belkin, and J. Beyerer, "World modeling for autonomous systems," in *KI 2010: Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, R. Dillmann, J. Beyerer, U. Hanebeck, and T. Schultz, Eds. Springer Berlin Heidelberg, 2010, vol. 6359, pp. 176–183.

[22] A. Furda and L. Vlacic, "An object-oriented design of a world model for autonomous city vehicles," in *Intelligent Vehicles Symposium (IV), IEEE*, June 2010, pp. 1054–1059.

[23] M. Iqbal, A. Arcuri, and L. Briand, "Environment modeling with UML/MARTE to support black-box system testing for real-time embedded systems: Methodology and industrial case studies," in *Model Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, vol. 6394, pp. 286–300.

[24] C. Wang, F. Pastore, A. Goknil, L. Briand, and Z. Iqbal, "Automatic generation of system test cases from use case specifications," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ser. ISSTA 2015. New York, NY, USA: ACM, 2015, pp. 385–396. [Online]. Available: http://doi.acm.org/10.1145/2771783.2771812

[25] H. Yanco and J. Drury, "Classifying human-robot interaction: an updated taxonomy," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, Oct 2004, pp. 2841–2846 vol.3.

[26] A. Steinfeld, T. Fong, D. Kaber, M. Lewis, J. Scholtz, A. Schultz, and M. Goodrich, "Common metrics for human-robot interaction," in *Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human-robot Interaction*, ser. HRI '06. New York, NY, USA: ACM, 2006, pp. 33–40.

[27] J. MacDougall and G. Tewolde, "Tour guide robot using wireless based localization," in *Proceedings of the IEEE International Conference on Electro/Information Technology (EIT)*, May 2013, pp. 1–6.

[28] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "Experiences with an interactive museum tour-guide robot," *Artif. Intell.*, vol. 114, no. 1-2, pp. 3–55, Oct. 1999.

[29] S. Thrun, M. Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz, "Minerva: a second-generation museum tour-guide robot," in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 3, 1999, pp. 1999–2005 vol.3.

[30] D. Bohus, C. W. Saw, and E. Horvitz, "Directions robot: In-the-wild experiences and lessons learned," in *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, ser. AAMAS '14. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2014, pp. 637–644.

[31] T. Kanda, H. Ishiguro, T. Ono, M. Imai, T. Maeda, and R. Nakatsu, "Development of robovie as a platform for everyday-robot research," *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, vol. 87, no. 4, pp. 55–65, 2004.

[32] D. Brscić, H. Kidokoro, Y. Suehiro, and T. Kanda, "Escaping from children's abuse of social robots," in *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, ser. HRI '15. New York, NY, USA: ACM, 2015, pp. 59–66.

[33] T. Kanda, D. Glas, M. Shiomi, and N. Hagita, "Abstracting people's trajectories for social robots to proactively approach customers," *Robotics, IEEE Transactions on*, vol. 25, no. 6, pp. 1382–1396, Dec 2009.

[34] M. Shiomi, D. Sakamoto, T. Kanda, C. Ishi, H. Ishiguro, and N. Hagita, "Field trial of a networked robot at a train station," *International Journal of Social Robotics*, vol. 3, no. 1, pp. 27–40, 2011.

[35] M. Shiomi, T. Kanda, H. Ishiguro, and N. Hagita, "A larger audience, please!: Encouraging people to listen to a guide robot," in *Proceedings of the 5th ACM/IEEE International Conference on Human-robot Interaction*. IEEE Press, 2010, pp. 31–38.

[36] ——, "Interactive humanoid robots for a science museum," in *Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human-robot Interaction*. ACM, 2006, pp. 305–312.

[37] M. Bouchard, J. Haegele, and H. Hexmoor, "Crowd dynamics of behavioural intention: train station and museum case studies," *Connection Science*, pp. 1–24, 2014.

[38] D. Brand and P. Zafiropulo, "On communicating finite-state machines," *J. ACM*, vol. 30, no. 2, pp. 323–342, Apr. 1983.

[39] K. T. Cheng and A. Krishnakumar, "Automatic functional test generation using the extended finite state machine model," in *Proceedings of the 30th International Design Automation Conference*, ser. DAC '93. New York, NY, USA: ACM, 1993, pp. 86–91.

[40] M. Sighireanu, C. Chaudet, H. Garavel, M. Herbert, R. Mateescu, and B. Vivien, "LOTOS NT user manual," 2000.

[41] H. Garavel, F. Lang, R. Mateescu, and W. Serwe, "CADP 2011: a toolbox for the construction and analysis of distributed processes," *International Journal on Software Tools for Technology Transfer*, vol. 15, no. 2, pp. 89–107, 2013.