Journal of Information Processing Vol.28 588–598 (Sep. 2020)

[DOI: 10.2197/ipsjjip.28.588]

Regular Paper

Safe Design of Flow Management Systems Using Rebeca

Giorgio Forcina[1,a)] Ali Sedaghatbaf[1,b)] Stephan Baumgart[2,c)] Ali Jafari[3,d)]
Ehsan Khamespanah[4,3,e)] Pavle Mrvaljevic[1,f)] Marjan Sirjani[1,g)]

Received: November 8, 2019, Accepted: June 1, 2020

Abstract: Track-based flow management systems like transportation systems and traffic control systems play a crucial role in our daily life. Safety and performance are among the most important quality requirements for these systems. This paper presents AdaptiveFlow as a framework for modeling, safety checking and performance analysis of track-based flow management systems. AdaptiveFlow is based on the Hewitt actors computation model. In particular, tracks are modeled as actors and moving objects as messages. Timed Rebeca is used for modeling, and the model checking tool Afra is used for safety verification and performance evaluation in AdaptiveFlow. To react to dynamic changes in the environment, AdaptiveFlow provides support for three adaptive policies, which can be examined and compared in different scenarios. To demonstrate the applicability of AdaptiveFlow, we consider the Electric Site Research Project of Volvo Construction Equipment as a case study. In this project, a fleet of autonomous haulers is utilized to transport materials in a quarry site. Furthermore, to show the reusability of the framework for other flow management scenarios, an experiment on an urban garbage collection system is presented.

Keywords: track-based flow management, actor model, model checking, safety verification, performance evaluation

1. Introduction

Track-based flow management systems such as warehouse management systems and public transportation systems play a crucial role in our daily life. All of these systems include a set of moving objects which travel on predefined tracks e.g., aisles, railways and roads. We see a common pattern among these systems. In fact, we have trains on rails, cars on roads, automated vehicles in aisles of a warehouse, and airplanes in predefined airspace-tracks. As the majority of these systems are mission critical, there is a need for assuring their safety, and optimizing their performance from different aspects e.g., resource consumption and operational time.

In this work, we consider a generalised view to the flow management systems. Our focus includes a wide range of applications consisting of flowing entities that are distributed, operate independently, and move around to accomplish a mission. In our view, the flowing entities are machines e.g., buses, trains or haulers that move around some environments e.g., streets, railways or quarry roads, transporting some assets e.g., passengers, packages or stones, between some points of interest (PoIs) e.g., bus/metro stations or loading/unloading stations. We present a formal framework which provides a common abstraction for movement scenarios in these systems, and utilizes model-checking to verify their safety and analyze their performance. This framework is called AdaptiveFlow and was first introduced in the conference paper [1]. This paper is an extension of Ref. [1],and provides more implementation details including the input format and the formal model, and adds a new case study to demonstrate the reusablity of the proposed framework for different kinds of flow management systems.

AdaptiveFlow is based on Timed Rebeca [2], which is an extension of Rebeca [3] with timing primitives. Rebeca is an actor-based modeling language well-suited for modeling and analysis of asynchronous communications and event-driven computations. The model checking tool Afra [4], [5] facilitates developing (Timed) Rebeca models, verifying their correctness and checking safety and progress properties. Deadlock freedom and deadline misses are examples of properties that can be verified using this tool. The idea of building AdaptiveFlow is rooted in different projects on flow management using actors and is explained in Ref. [6]. These projects include routing packets in network-on-chip [7], smart urban planning and managing the resources in transportation hubs [8], and rerouting and rescheduling flights in a track-based traffic control system [9].

In addition to typical safety properties, AdaptiveFlow enables checking correctness properties important for flow management systems. These properties include collision avoidance, being on the correct track for the mission, and running out of resources or fuel. Furthermore, AdaptiveFlow is capable of measuring the amount of consumed fuel, emitted pollution, operational time and transported assets. It provides an easy to use interface for specifying the input parameters which include system configuration

[1] Mälardalen University, Västerås, Sweden
[2] Volvo Construction Equipment AB, Eskilstuna, Sweden
[3] Reykjavik University, Reykjavik, Iceland
[4] University of Tehran, Tehran, Iran
[a)] giorgio.forcina@gmail.com
[b)] ali.sedaghatbaf@mdh.se
[c)] stephan.baumgart@volvo.com
[d)] ali@ru.is
[e)] e.khamespanah@ut.ac.ir
[f)] pmc19001@student.mdh.se
[g)] marjan.sirjani@mdh.se

and topology, and environment characteristics. Hence, the designer can explore the design space by changing the input parameters and analyze the effects of those changes on safety and performance. Additionally, AdaptiveFlow allows the designer to specify policies for adapting the system behavior with respect to possible changes in the environment. Sudden changes like blocking of a track, or change of a point of interest like a charging station being out of order can be modelled, too.

To show the applicability of AdaptiveFlow for designing flow management systems, we use the case study of the Electric Site Research Project of Volvo Construction Equipment (VCE) [10] In the Electric Site project, a fleet of self-driving autonomous electrified vehicles (haulers) transport materials in the quarry site. Since vehicles are electrified and equipped with batteries they need to be charged at chargers in the site. There are two loading points which are each independent systems, and an unloading point. The missions are currently supervised by a central unit. The plan is to move towards a more distributed control. We used AdaptiveFlow to explore the design space and check different configurations where we can have the optimum transportation paths for haulers which for example reduces power consumption, and at the same time guarantees the safety of each vehicle and the overall system safety.

In addition to the VCE project and in order to show the reusability of AdaptiveFlow for different kinds of flow management scenarios, we also present a study on an urban garbage collection system. The outcomes of this study indicate the usefulness of the proposed framework for designing a safe flow management system with optimal performance.

In Section 2 we introduce Timed Rebeca, and in Section 3 we explain the VCE Electric Site example. In Section 4 the AdaptiveFlow framework is described. In Section 5, we present comparative experiments with different configurations that help designers make proper decisions. Section 6 is dedicated to the study on a garbage collection system to demonstrate the reusability of the proposed framework. Section 7 discusses about the research contributions related to this work. Finally, in Section 8, we conclude the paper and outline the future directions.

2. Rebeca and Timed Rebeca Languages

Rebeca (Reactive Object Language) [11], [12] is an actor-based language. Actors are introduced by Hewitt [13] and promoted as a concurrent object-based functional language by Agha [14]. Rebeca is designed to be a bridge between the formal methods community and software engineers, it is designed as an imperative language, with Java-like syntax, and is supported by a model checking toolset.

Actors are units of concurrency, with no shared variables, communicating by asynchronous messages. There is no explicit receive statement, and send statements are non-blocking. There is only one single thread of execution in each actor and one message queue. The actor takes a message from the top of its message queue, and executes the corresponding method (called *message server*) non-preemptively.

In Timed Rebeca (the real-time extension of Rebeca) [2], [15], [16], instead of a message queue we have a message bag, where messages are tagged with their time-stamps. There is a concept of synchronized local clocks throughout the model for all the actors (which can be considered as a global time). The sender tags a message with its own local time, at the time of sending.

A Rebeca model consists of a number of *reactive classes*, each describing the type of a certain number of *actors* (called *rebecs*). Each reactive class declares the size of its message buffer, a set of *state variables*, and the messages to which it can respond. The local state of each rebec is defined by the values of its state variables and the contents of its message buffer. Each rebec has a set of *known rebecs* to which it can send messages. Each reactive class has a constructor with the same name. This constructor is responsible for initializing the state variables and putting initially needed messages in the message buffer.

The way a rebec responds to a message is specified in a *message server*. The state of a rebec can change during the execution of its message servers through assignment statements. A rebec makes decisions through conditional statements, communicates with other rebecs by sending messages, and performs periodic behavior by sending messages to itself. Since communication is asynchronous, each rebec has a *message buffer* from which it takes the next incoming message. A rebec takes the first message from its message buffer, executes its corresponding message server in an isolated environment, takes the next message (or waits for the next message to arrive) and so on. A message server may have a *non-deterministic assignment* statement which is used to model the non-determinism in the behavior of a message server. Finally, the `main` block is used to instantiate the rebecs of the model.

Timed Rebeca adds three primitives to Rebeca to address timing issues: *delay*, *deadline* and *after* [2]. Each primitive is used as follows:

- **Delay**: *delay(t)* models the passage of time for a rebec during execution of a message server, it increases the value of the local clock of the respective rebec by the amount *t*. Note that all other statements of Timed Rebeca are assumed to execute instantaneously.
- **After**: The keyword *after* is used in conjunction with a method call and indicates that it takes *n* units of time for a message to be delivered to its receiver.
- **Deadline**: The keyword *deadline* is used in conjunction with a method call and expresses that if the message is not taken in *n* units of time, it will be purged from the receiver's message bag automatically (timeout).

These primitives provide the syntax to cover timing features that a modeler might need to address in a message-based, asynchronous and distributed setting, including computation time (*delay*), message delivery time (*after*), periods of occurrences of events (*after*), and message expiration (*deadline*).

3. Volvo Construction Equipment Electric Site

As an industrial case study, we consider the Volvo CE electric quarry site, where gravel of different granularity is produced. This gravel is typically used for building construction, road work or railway beds (see **Fig. 1**). The rocks are blasted in one area of the quarry and the big blocks are crushed into smaller trans-

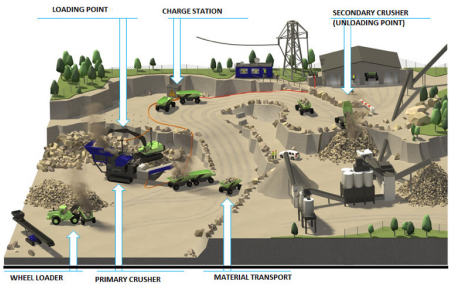

Fig. 1 Volvo construction equipment electric site.

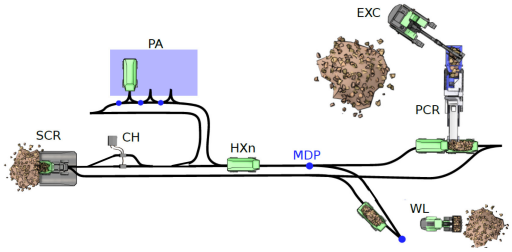

Fig. 2 Schema of the volvo construction equipment electric site.

portable rocks using a movable crusher (primary crusher). The crushed material is then transported to a stationary crusher (secondary crusher) where the material is crushed into the target granularity. In the electric site research project at Volvo Construction Equipment, material transportation from the primary crusher to the secondary crusher is done by a fleet of autonomous haulers (called HX).

In **Fig. 2** tracks for the autonomous haulers are shown. The haulers are loaded at the Primary Crusher (PCR) or by a human operated wheel loader (WL). The primary crusher is fed by a human operated excavator (EXC). Once the haulers are loaded, they travel to the secondary crusher (SCR) where they dump the load. Since the haulers are electrified and equipped with batteries, they need to be charged at the chargers (CH). The missions in this site are set by a central site control unit, which is supervising all activities. So, different queuing points are necessary where the haulers receive their next mission. In order to make decisions for optimal production, the haulers queue at the main decision point (MDP), and once a loading mission is assigned, the haulers will move to the assigned loading position. The fleet of haulers can be parked or maintained at the parking area (PA).

Compared to automated guided vehicle (AGV) applications in predefined environments like warehouses, the AGVs in the quarry site are exposed to harsh environmental conditions, which can change rapidly. Therefore, the site control system must be able to adjust the fleet of haulers based on the changed conditions.

4. AdaptiveFlow Framework

AdaptiveFlow is built based on Timed Rebeca and its model checking tool, Afra (see Ref. [17]). Each flow management system is a network of independent systems, and the Timed Rebeca model in AdaptiveFlow captures the communication pattern and protocol among these systems. Afra is used in AdaptiveFlow to formally verify the properties specified as assertions or temporal logic formulas. While performing model checking, Afra generates the state space of the Timed Rebeca model. In AdaptiveFlow the generated state space is explored and the designer is provided with performance metrics that can be used for design space exploration and optimisation purposes.

4.1 Timed Rebeca Model in AdaptiveFlow

The Timed Rebeca Model of the flow management system is built based on the features of interest in the system. Here we explain the problem domain which is the basis for building the model.

The *infrastructure* (or the environment) is modelled as a collection of segments each characterised by a unique identifier and a coordinate. Each segment (cell) is linked with its neighbour cells and may differ from others in terms of length, allowed speed, and capacity. The cells can be either available, in case they can be traversed, or unavailable, in case there is an obstacle blocking them. A segment knows its adjacent cells. The maximum number of neighbours is eight, one for each cardinal position (i.e. north, north-east, east, south-east, south, south-west, west, north-west). The *topology* shows the positions within the environment in which the vehicles can perform their tasks (e.g., picking up passengers at a bus station, loading stones in a quarry, charging fuel, etc.), namely Points of Interest (PoI). Each PoI is characterised by its unique identifier (i.e., id), its position on the map (i.e., x and y), its type and its operating time. The operating time represents the time needed for performing the specific task at the current PoI. For example, for the VCE Electric Site, we have the following PoIs: the *Parking Station* where the fleet of vehicles are parked when they are not operating, the *Charging Station* where vehicles can recharge, and the *Loading-Unloading Point* where vehicles can either load or unload materials.

In addition to the environment and the topology we need information regarding the *vehicles*, and the *configuration* of the system. For the Electric Site case study, each vehicle has its own identifier, its own mission, and its own type. The mission specifies the initial location of the vehicle, and the path it has to take. The type of the vehicle declares several features like capacity of the fuel tank, fuel consumption rate, average speed, CO_2 emission, and load capacity.

For the configuration we can set different parameters including the number of operating vehicles, the safety distance between vehicles, and the level of fuel that should be reserved.

Communications are modeled through asynchronous message passing in AdaptiveFlow. In particular, whenever a vehicle wants to move to an adjacent cell, the host cell sends a request message to the target cell. This message can represent a query from the central control unit, which is supervising all activities (as in the VCE site), or alternatively it can model a periodic checking of the surroundings by the vehicle itself.

The request may get rejected because the adjacent cell may be occupied by another vehicle, or blocked due to some problem like an obstacle or a hole created by rain. So, the request needs to be resent periodically. The period for re-sending the rejected request is an important parameter that can be set. During the analysis phase, this period can be adjusted to find a safe and also more efficient configuration.

Another important feature is configuring the adaptive policies.

We can configure AdaptiveFlow so that it knows which adaptive policy to use, and when to switch to another policy. More details follow.

Adaptive Policies. Dynamic behaviour and adaptability are among the main features of flow management systems. In our Timed Rebeca model we have event handlers (or message servers) that handle adaptation. In the current implementation, the model has been equipped with three adaptive policies to manage the situation in which some segments are temporarily unavailable. The policies are defined based on the different decisions available for the haulers. These decisions include the following: (1) wait for the blocked segment to be available again, (2) bypass the blocked segment and continue on the predefined route, and (3) run a routing algorithm and continue based on a complete new route. Accordingly, AdaptiveFlow supports the following three policies:

- *Policy 1* (*postpone*) allows the vehicle to postpone its planned movement by an amount of time that is equal to the *re-sending period* value specified in the input configuration. In case the Segment is unavailable for a number of attempts greater than the *max attempts* value, the re-route policy is applied.
- *Policy 2* (*overpass*), lets the vehicle bypass the segment that is occupied, allowing it to overpass that position according to the current environment and obstacles, and then continue on the route pre-specified in the mission plan.
- *Policy 3* (*re-route*) uses the Dijkstra shortest path algorithm [18] to calculate an alternative path from the current position to the destination PoI, taking into consideration the current situation of the environment including both static and dynamic obstacles.

In Section 5 we compare the above policies from different viewpoints, but note that the main purpose here is not to prove the effectiveness of a policy against other policies, but to show that in AdaptiveFlow it is possible to design, implement and analyse different adaptive policies according to the context or user's needs.

Lagrangian and Eulerian Rebeca Models.

AdaptiveFlow supports two design patterns to build the Timed Rebeca model. One pattern is modelling each mobile system as an actor, and the other is modelling each track of the environment as an actor. These two patterns reflect the two general views for flow analysis, inspired from fluid mechanics [6]. In fluid mechanics, two well-known alternative ways to model fluid flow are Lagrangian and Eulerian models [19]. In the Lagrangian model of a fluid, the observer follows an individual fluid parcel as it moves through space and time. In the Eulerian view, the observer fixes on a region of space and observes fluid mass passing through that space. For example, in studying the flow of a river, the Lagrangian view is like sitting in a boat and floating down the river, whereas the Eulerian view is like sitting on the bank and watching the boats float by.

The Eulerian model, where each track of the environment is modelled as an actor, seems less faithful to the flow management system we are modelling. In this model, the mobile systems are modelled as messages or packets passing through the tracks. For the Electric Site case study, each packet represents a hauler, and has its own identifier, its own mission, and its own type. The request re-sending period may represent the same phenomena in both patterns. So, we still can model the main features of the independent vehicles in the Eulerian approach. More in-depth discussion on this topic is outside the scope of this paper.

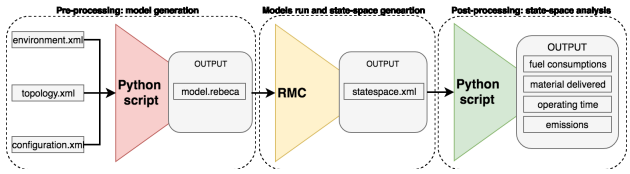

Fig. 3 AdaptiveFlow modules and processes.

4.2 AdaptiveFlow Modules and Analysis Process

The architecture of AdaptiveFlow is based on Ref. [20] and can be seen in **Fig. 3**. Accordingly, AdaptiveFlow consists of three modules for (1) pre-processing and building the Timed Rebeca model from a more friendly input, (2) running Afra, performing formal verification and generating the state space, and (3) post-processing through state-space analysis. These three modules are elaborated in the following.

4.2.1 Pre-processing

Pre-processing is for *model generation*. In the pre-processing phase, AdaptiveFlow generates the Timed Rebeca model that depicts the one provided by the user by means of the input files based on the input data provided by the user. The user is supposed to provide three input filesin Extensible Markup Language (XML) format. These files include *environment*, *topology*, and *configuration* data.

The file *environment.xml* includes data about the segments of the environment that vehicles are supposed to move in. An example environment specification is presented in Listing 1. This example includes two segments each characterized by a unique identifier, links to its surrounding segments, its coordinates, availability, length, capacity and speed limit. Accordingly, both segments can hold at most one hauler at a time, and only the second segment is available in the initial state.

The file *topology.xml* contains data about PoIs. An example of the structure of topology specification is provided in Listing 2.

This example includes one instance for each type of PoI supported by AdaptiveFlow. The parking and charging stations are located on the same row, and the time needed to (un)load materials is five times the time needed to charge fuel.

The third AdaptiveFlow input file (i.e., *configuration.xml*) includes two sections: (1) the configuration of the system, and (2) the properties of the transportation vehicles. The first section comprises the following:

- *resendingPeriod*: the time for re-sending a segment request in case of a negative response,
- *numbervehicles*: the number of operating vehicles,
- *safeDistance*: the safe distance between vehicles,
- *fuelReserve*: the level of running out of fuel,
- *policy*: the adaptive policy in case of re-planning,
- *maxAttempts*: number of attempts before re-planning.

For what concerns the specification of the vehicles (Listing 3),

```
<segment id="seg_0_0" N="null" NE="null" E="seg_0_1" ES="seg_1_1"
        S="seg_1_0" SW="null"      W="null"       WN="null" available="false"
        x="0" y="0" capacity="1" length="200" freespeed="6"/>
<segment id="seg_1_1" N="seg_0_1" NE="null" E="null" ES="null" S="null"
        SW="null" W="seg_1_0" WN="seg_0_0" available="true" x="1" y="1"
        capacity="1" length="200" freespeed="6"/>
```

Listing 1 Example of the environment specification.

```
<topology>
   <POIs>
      <poi id="0" x="1" y="3" type="ParkingStation"/>
      <poi id="1" x="3" y="3" type="ChargingStation" chargingTime="0.1"/>
      <poi id="5" x="6" y="7" type="LoadUnloadingPoint" loadTime="0.5"/>
   </POIs>
</topology>
```

Listing 2 Example of the topology specification.

```
   <vehicles>
      <vehicle id="0" type="hauler" leavingTime="10" fuelCapacity="7000"
            fuelConsumption="1" speed="6" emission="6" capacity="100"
            unloadTime="0.1">
         <tasks>5,3,2,4,5,3,0 </tasks>
      </vehicle> <\vehicles>
```

Listing 3 Example of vehicles' configuration.

the user can specify:

- *id*: the unique identifier of the vehicle,
- *type*: the typology of the simulated vehicle,
- *leavingTime*: the time in which the vehicle leaves the parking and gets operational,
- *fuelCapacity*: the capacity of the vehicle's fuel tank,
- *fuelConsumption*: the fuel consumption of the vehicle,
- *speed*: the average speed of the vehicle,
- *emission*: the CO_2 emissions,
- *capacity*: the load capacity of the vehicle,
- *unLoadTime*: the time needed to discharge the transported material.

Moreover, each vehicle comes with a sequence of tasks to perform in order to conclude its daily operating cycle. For instance, moving from the parking slot and reaching the loading station.

A Python script processes the content of the above files and generates the input for the next module (i.e. the Timed Rebeca model). Listing 4 illustrates a sketch of the Timed Rebeca model generated for the VCE Electric Site (the complete implementation can be accessed from AdaptiveFlow Web page [21]). This model includes a set of environment variables which declare important properties of the quarry, vehicles and stations. These properties are extracted from the input files. Number of segments, maximum speed and battery consumption rate are among these properties.

The Time Rebeca model consists of only one reactive class, which defines an environment segment. The definition of the *knownrebecs* section indicates that each segment knows and interacts with its eight surrounding segments. The *statevars* section includes declaration of the segment id, its position on the map, its capacity at each moment, and its type. Listing 4 includes the declaration of the most important message servers of the segment actor. The implementation details of these message servers are eliminated due to space limitation.

Message server *startMovingVehicles* starts moving the vehicles by checking the capacity of the segments and running the Dijkstra path-finding algorithm for each vehicle. This algorithm is implemented by another message server called *initRouteWithDijkstra*. After the path is determined, each segment hosting a vehicle will ask the next segment in the path for entrance permission. Permission acquisition is performed by message server *givePermitionForVehicle*. This message server checks the availability of the target segment, and if it is available, invokes *getPermission* to move the vehicle and informs the target segment by invoking *vehicleEntered*. Otherwise, it means that there is an obstacle, and message server *segmentNotFree* is invoked which uses the configured adaptive policy to identify the next segment to ask permission.

4.2.2 Formal Verification and State-space Generation

The Timed Rebeca model generated from the pre-processing phase is model checked using Rebeca Model Checker (RMC), which is the model checking engine of the Afra tool. RMC converts the Timed Rebeca model to a set of C++ files. These files are then compiled to an executable file. Running the executable file, RMC applies the model checking algorithm and generates the verification results. RMC automatically verifies if the model is deadlock free.

In addition,AdaptiveFlow is able to check the following properties:

- if the vehicles are out of fuel,
- if the vehicles are moving correctly on the predefined path,
- if the vehicles crash into each other or obstacles,
- if the current configuration may lead to a deadlock situation.

Moreover, running RMC results in the generation of the whole state-space of the model, as input for the next module.

4.2.3 Post-processing

In the post-processing phase, the state-space generated from the previous module is evaluated with a Python script. In particular each state of the system is analysed and those variables that are meaningful for the analysis of the system are extracted. Once these values are collected, they are organised such that the useful data can be extracted for vehicles and the system. These data include the amount of consumed fuel, moved material and emitted CO_2, and also the operational time of each vehicle.

5. Experimental Results

In order to demonstrate the applicability of AdaptiveFlow, we describe the experimental results for the VCE Electric Site case study in the following. **Figure 4** shows the graphical representation of the scenario we are interested to model and analyse. In this scenario, the environment is composed of 100 segments in 10 rows, and 10 columns. There are six PoIs, including one parking station (PS, id: 0), four loading/unloading points (LUP with id: 2, 3, 4 and 5), and one fuel charging station (CS, id: 1). The input files that define the environment, the topology, and the system configurations are available on the AdaptiveFlow web page [21].

In order to analyze the safety and performance of this scenario, we performed two sets of experiments. In the first set, the goal was to evaluate how the PoI positions and the adaptive policies would affect the safety and performance of the vehicles. In the

```
env int RESENDING_PERIOD = 15; // time to wait
    before ask again the availability of a segment
env int NUMBER_VEHICLES = 5; // number of simulated
    vehicles
env int SAFE_DISTANCE = 20; // meters
env int BATTERY_LIMIT = 1000; // battery reserve
env int MAX_ATTEMPTS = 2; // max attempts on the
    same segment
/* Used adaptive policy: 1 wait, 2: overpass, 3:
    change route, 4: rong */
env int POLICY = 1;
/* Fuel configuration: tank size and costs, wat/km
    for each vehicle */
env double[5] VEHICLES_BATTERIES = {7000, 7000,
    7000, 7000, 7000}; // watt
env int[5] VEHICLES_SPEED = {6, 6, 6, 6, 6}; // m/s
env double[5] BATTERY_CONSUMPTION = {1, 1, 1, 1, 1};
    // max battery consumption: watt/meter
/* CO2 emissions per vehicles grams per 100 meters */
env double[5] CO2_EMISSIONS = {6, 6, 6, 6, 6}; //
    grams/100 meters
env int OBSTACLE_OCCURRENCES = 5;
env int OBSTACLE_NUMBER = 4;
/* Location of PoIs */
env int[1][2] PARKING_STATION_POSITIONS = {{1,3}};
env int[1][2] CHARGING_STATION_POSITIONS = {{3,3}};
env int[4][2] LOAD_UNLOAD_POSITIONS =
    {{3,1},{8,1},...}; // Vehicles can load or
    unload at each point
env int[6][2] PoIs_LOCATION={{1,3},{3,3},...};
env int NUM_OF_PARKING_STATIONS = 1;
env int NUM_OF_CHARGING_STATIONS = 1;
env int NUM_OF_LOAD_UNLOAD_STATIONS = 4;
env int NUM_OF_SEGMENTS = 100;
env int NUM_OF_ROWS = 10;
env int NUM_OF_COLUMNS = 10;
...
reactiveclass Segment(6) {
    knownrebecs{
        Segment N, E, S, W, NE, ES, SW, WN; //the
            eight surrounding segments
    }
    statevars{
        int id;
        int currCapacity; // current capacity
        int[2] coord; //(x, y) coordinates
        boolean isParkingStation, isChargingStation,
            isLoadUnloadLocation;
    }
    Segment(int sid, int x, int y){
        id = sid;
        coord[0] = x;
        coord[1] = y;
    }
    msgsrv startMovingVehicles(...){...} //Starts
        moving the vehicles
    msgsrv initRouteWithDijkstra((...){...} //Creates
        the route from a PoI to the next one
    msgsrv givePermisionForVehicle(...){...} //The
        preceding segment asks this segment to allow
        the vehicle to enter it
    msgsrv getPermision (...){...} //the next segment
        grants permission to the vehicle
    msgsrv segmentNotFree(...){...} //The segment
        denies the vehicle to enter it
    msgsrv startSendingToNext(...){...} //The segment
        selects the next segment for the vehicle
    msgsrv vehicleEntered(...){...} //The preceding
        segment informs the segment that the vehicle
        has entered
    msgsrv changeRouteWithPolicy2(...){...}
        //overpasses the obstacle
    msgsrv changeRouteWithPolicy3(...){...} //finds a
        new route using Dijkstra
}
main{
    Segment sg11(..., sg12, ...):(1, 4, 5);
    Segment sg12(..., sg11, ...):(2, 6, 3);
    ...
}
```

Listing 4 A sketch of Timed Rebeca model in AdaptiveFlow.

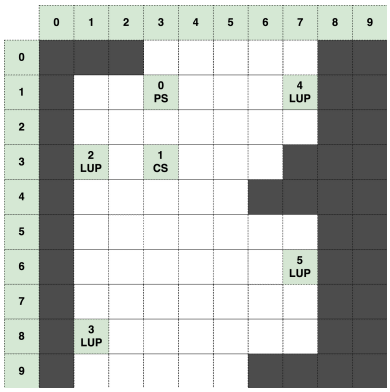

Fig. 4 The environment used in experiments.

second set of experiments, we considered two types of vehicles working in the quarry with different speed, fuel consumption, load capacity, fuel capacity, and CO_2 emissions. The aim was to evaluate how replacing vehicles of type A with vehicles of type B would affect the performance.

Here, we discuss the results of the two sets of experiments discussed above. Values shown in the figures refer to the consumed fuel, the emitted CO_2 and the time needed for executing all the given tasks. Moreover, the configurations are compared with respect to the adaptive policies. The results related to each policy are presented with a different colour.

For what concerns model checking, in all the experiments the properties mentioned in Section 4 are satisfied, confirming that the models with the given configurations did not violate the requirements. It is worth saying that the first three properties were satisfied by model design, i.e., the behavior of the *Segment* rebec was defined such that these crucial needs were respected. Regarding the last property, (i.e., deadlock freedom), the current design of AdaptiveFlow does not support configurations in which two PoIs are adjacent, unless each PoI can provide service to more than one vehicle simultaneously.

An example of this situation is shown in **Fig. 5**. The red vehicle has just finished charging fuel at (0, 0) and it is approaching the loading point at (1, 1). The blue vehicle needs to reach the charging station, since it almost ran out of fuel after having loaded materials at (1, 1). These two vehicles want to move to the other PoI simultaneously, and the first adaptive policy (i.e., postpone) is applied by both of them. Therefore, they will wait until the PoI becomes available again, which will never happen and we will have deadlock.

5.1 Experiment 1: Changing PoI Positions

Here, the goal is to evaluate how the PoI positions, as well as

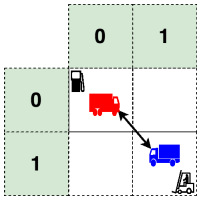

Fig. 5 Example of a configuration which leads to starvation.

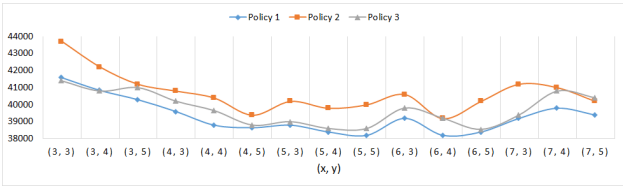

Fig. 6 Exp.1-Fuel consumption comparison.

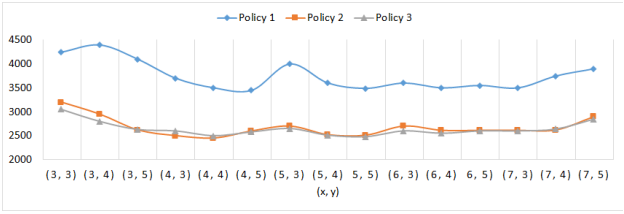

Fig. 7 Exp.1-CO2 emission comparison.

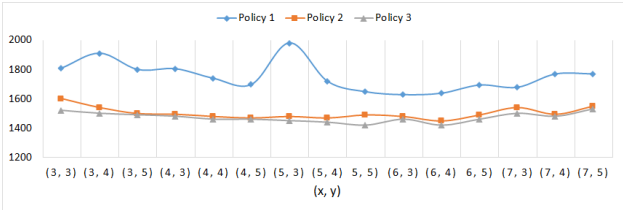

Fig. 8 Exp.1-Operating time comparison.

the adaptive policies, affect the performance of the vehicles. We change the positions of the charging and parking stations, while all the other parameters remain unchanged. Dynamic obstacles are generated randomly during the pre-processing phase, and the model is executed 45 times. The output of the post-processing module helps the designer to select the configuration most suitable for her needs (e.g. minimising operational times, reducing fuel consumption, etc.).

Figures 6, **7**, and **8** show the outcomes of the first set of experiments. Accordingly, the positions that optimise all the evaluated measures are those located in the center of the site (i.e., x and y are between 4 and 5). Considering the role played by the adaptive policies, the one that minimizes the operating time is the third policy. With this policy, a vehicle's route is re-computed whenever it reaches an obstacle. This means that it will follow the shortest path from the current position to the PoI, while avoiding the obstacle. As expected, the first policy (i.e., postpone) imposes the highest operating time. However, the fuel consumption is the lowest for this policy, since vehicles do not consume fuel when they are waiting. This is not the case for CO_2 emission, since we assume that waiting vehicles produce a little amount of pollution. It is worth saying that these assumptions can be changed without much effort and in accordance with the system to be simulated.

Table 1 Characteristics of haulers A and B.

Type	Fuel Capacity	Fuel Consumption	CO2 Emission	Speed	Load Capacity
A	7000 W	1 W/m	60 g/km	6 m/s	100 ql.
B	10000 W	2 W/m	120 g/km	8 m/s	150 ql.

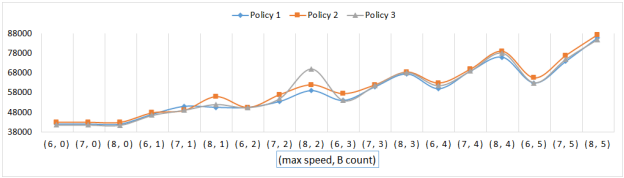

Fig. 9 Exp.2-Fuel consumption comparison.

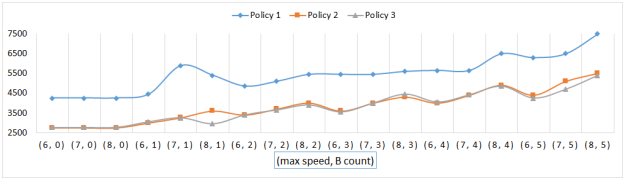

Fig. 10 Exp.2-CO2 emission comparison.

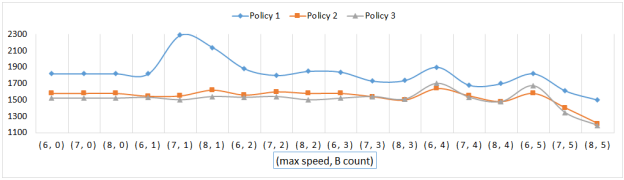

Fig. 11 Exp.2-Operating time comparison.

5.2 Experiment 2: Changing Vehicle Types

In the second set of experiments, we consider two types of vehicles working in the quarry. They differ in terms of speed, fuel consumption rate, load capacity, fuel capacity, and CO_2 emissions. **Table 1** shows the characteristics of these types of vehicles. The aim is to evaluate how replacing vehicles of type A with vehicles of type B affects the throughput of the VCE quarry site. Moreover, we gradually increased the permitted traversing speed on segments from 6m/s up to 8 m/s, so that vehicles of type B could exploit their higher velocity. All these configurations were evaluated with the three adaptive policies and the model was executed 54 times.

In the second set of experiments, considering the results shown in **Figures 9**, **10**, and **11**, we notice that replacing vehicles of type A with type B increases both fuel consumption and CO_2 emissions. On the other hand, the operating time decreases with the increase in the number of vehicles of type B. This is true only when the maximum speed for each segment is greater than 6 m/s allowing vehicles of type B to exploit their higher velocity. It is also worth remarking that in contrast to the first set of experiments in which all the runs ended with a total amount of transported material that is equal to 1,500 quintals, employing vehicles with higher transportation capacity led the system to be more productive. Indeed, the greater the number of type B vehicles, the higher the amount of moved material, i.e., 1,500, 1,650, 1,800, 1,950, 2,100, and 2,250 quintals for configurations with 0, 1, 2, 3, 4, and 5 vehicles of type B, respectively.

From the adaptive policy point of view, the results indicate that fuel consumption is almost the same for all the three policies, and using either policy 2 or 3 instead of policy 1 would significantly

reduce both CO_2 emissions and operating time.

5.3 Scalability

In all of the experiments discussed above, the size of the environment and the number of obstacles were kept constant. In fact, the environment included 100 segments in 10 rows and 10 columns and four obstacles were generated by the Python script in each experiment.

In order to analyze the scalability of AdaptiveFlow, we repeated the experiments by gradually increasing the environment size and the number of obstacles. We noticed that the amount of memory consumed by the model checker increased linearly with respect to the environment size. We also observed that policy 3 led to the least amount of memory consumption but the most execution time for model checking. We also observed that the system ran out of memory for an environment with 225 segments. This event is well known as the state-space explosion problem in the model checking domain. To tackle this problem, several techniques have been proposed in the past two decades. Partial order reduction and bounded model checking are among these techniques [22].

Our past experiences with compositional modeling and analysis [23], [24] indicate that it is an effective means to reduce the complexity and improve the scalability of actor-based approaches. Extending AdaptiveFlow with compositional analysis is considered as future work.

6. Reusability

In this section, we show the reusability of AdaptiveFlow by analyzing the safety and performance of a Garbage Collection (GC) system as another flow management system. In particular, the purpose of the experiments discussed in this section is to demonstrate the applicability of AdaptiveFlow to any kind of flow management scenario in which we are interested to assure the safety of some objects flowing around some environment, and analyze their performance.

Urban garbage collection is a public service that municipalities provide to citizens, and consists of collecting the garbage that we generate every day and transporting it to centralized treatment plants. Here, we consider a scenario in which two collectors are supposed to collect garbage from three loading stations and transport them to a treatment plant. Similar to the experiments in Section 5, the environment that the collectors operate in is divided into a set of segments. As depicted in **Fig. 12**, the environment consists of 72 segments in 6 rows and 12 columns. There are three loading stations, one parking station and one charging station. We performed a set of experiments to analyze the effects of the adaptive policies on the safety and performance of the collectors. The input files used in these experiments can also be found on the AdaptiveFlow web page [21].

From the safety point of view, the design of the Timed Rebeca model assures satisfaction of all of the four properties specified in Section 4. For performance analysis, we defined the goal as to find the optimal positions for the charging/parking stations.The places that we chose for the parking/charging station are highlighted with blue/yellow background in Fig. 12.

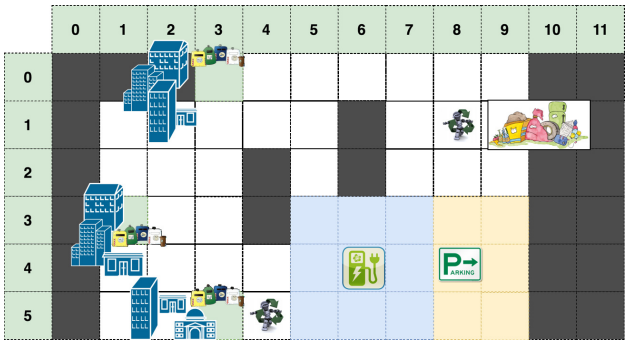

Fig. 12 The garbage collection environment.

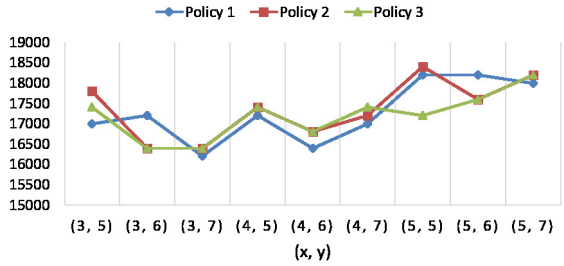

Fig. 13 GC-Fuel consumption comparison.

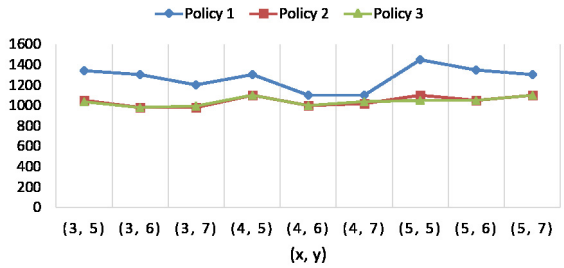

Fig. 14 GC-CO2 emission comparison.

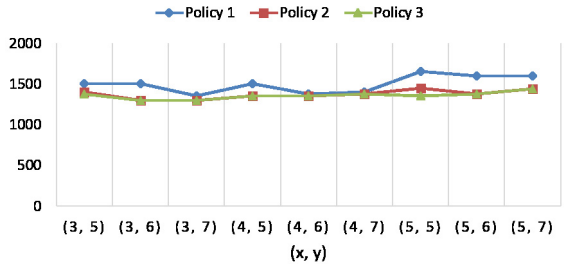

Fig. 15 GC-Operating time comparison.

Figures 13, **14** and **15** show the performance analysis results for all positions considered for the charging station with respect to each adaptive policy. These results indicate that (3, 7) is the optimal position for the charging station if the postpone policy is applied. However, for the re-route and overpass policies, both (3, 6) and (3, 7) are optimal. Furthermore, it is evident that the re-route policy leads to the minimum operating time in all cases, and the postpone policy leads to the maximum CO2 emissions.

These experiments indicate that the generality of AdaptiveFlow makes it usable for analyzing different kinds of flow management systems.

7. Related Work

Bagheri et al. proposed a coordinated actor model in Ref. [9] that can be used to model track-based traffic control systems in which the traffic flows through pre-specified sub-tracks and is coordinated by a traffic controller. In comparison to Ref. [9], AdaptiveFlow supports the decentralised implementation of control systems and there is no need for a centralised coordinator. Moreover, we provide model checking facilities while the coordinated actor model is supported by Ptolemy II [25] simulator.

Some researchers have contributed to using formal methods for the analysis of path-finding in AGVs (Automated Guided Vehicles). Authors in Ref. [26] propose an approach for AGVs to explore an unknown grid-based environment and find a path to a destination. They model the problem using timed automata and analyse it using UPPAAL. To make the analysis possible, they show how the grid-based environment can be decomposed to a smaller area while the analysis result is valid for the original model.

Smith et al. in Ref. [27] addressed robot path planning using weighted transition systems for the model specification and generalised LTL formula for goal specification. Their approach shows how in every environment model, and for every formula, a robot trajectory which minimises the cost function is computed. Modelling in this approach is at a lower level of abstraction compared to the work of Ref. [26]. Authors in Ref. [28] have a similar approach for analysing A* algorithm, using Z modelling language and its corresponding toolset. Authors in Ref. [29] model a vehicle and consider different features, such as position localisation, human and obstacle detection, collision avoidance. This model is then analyzed to avoid fatal accidents. In particular, they use timed automata to model the vehicle's control system, including the abstracted path planning and collision avoidance algorithms used to navigate a vehicle, and model check it using UPPAAL.

The implicit assumption in the above methods is that no time constraint is associated to plan specifications. To provide support for path planing under time constraints, Zhou et al. propose a method based on metric interval temporal logic and timed automata [30]. All the above mentioned contributions are focused on path-finding in one machine while we consider a set of collaborating machines and address the interference of activities of different machines.

Authors in Ref. [31] address the safe path planning problem in the multi-robot configuration. They use mCRL2 to specify robots and the environment, and check if the collective behaviour of a group of robots satisfies certain desired properties. They illustrate the applicability of their approach using a simple path planning algorithm which conducts a set of robots from their initial positions to their destinations on a planar surface. Moving objects in Ref. [31] look into their neighbouring cells in each step and proceed one step while they do not have specific missions or plans to reach their destinations. In addition, based on what the authors mention in the experimental results, they couldn't check models with many robots and big environments. This problem is tackled using the same approach and facing the same shortages with timed automata in Ref. [32] and with hybrid automata in Ref. [33]. For the latter, they show how the generated hybrid automata can be embedded into automata which can be model checked using SMV. They discuss that such an embedding does not change the result of verification for reachability properties.

Another category of research work related to the scope of this paper is safe path synthesis using formal methods. As an example, Fainekos et al. [34] propose a method to generate trajectories for mobile robots. These trajectories satisfy temporal logic formulas specifying goal configurations, synchronization and temporal ordering of different motions. This work was then extended in Ref. [35] to support sensor specifications. The path planning method presented in Ref. [36] is another work in this category, which specifically addresses dynamic obstacles. In comparison to our work, these contributions do not consider optimisation issues; i.e. although they provide safe paths, they do not consider other possible optimal paths.

8. Conclusion and Future Work

In this paper, we present AdaptiveFlow, a design platform for modelling and analysing collaborating systems, in the domain of traffic management systems and track-based flow management. The model can capture independently operating and autonomous mobile systems that transport assets (e.g., passenger, material, etc.) among a number of systems at dedicated locations (e.g., train stations, airports, loading stations, etc.). Each system may have different features, like capacity and speed limit, and mobile systems need to refuel at some charging stations. The models are written in Timed Rebeca, and the Rebeca model checking tools are used both for checking property violations and also for performance evaluation. AdaptiveFlow allows users to easily customize the system by means of user-friendly input files, and to evaluate how their decisions can affect the throughput of the modelled system. Moreover, the model is designed in such a way that the movements of machines can adapt to the unexpected changes of the environment. The platform also supports modelling and analysis of cost parameters, like fuel consumption and CO_2 emissions. This is done automatically by AdaptiveFlow thanks to the chain of modules explained in Section 4.

As future work, we plan to enrich AdaptiveFlow with more adaptive policies to handle other unexpected changes in the environment. In particular, we will add a policy that avoids machines getting stuck in situations like the one explained in Section 5. Moreover, we will implement the adaptive algorithm named *Dipole flow field* described in Ref. [37] and used by Ref. [29]. We are also working on generating ROS (Robot Operating System) code from the Timed Rebeca models of AdaptiveFlow.

Acknowledgments The work of a subset of authors is partly supported by KKS DPAC Project (Dependable Platforms for Autonomous Systems and Control), KKS SACSys Synergy project (Safe and Secure Adaptive Collaborative Systems), and Self-Adaptive Actors: SEADA (nr 163205-051) project from the Icelandic Research Fund.

References

[1] Sirjani, M., Forcina, G., Jafari, A., Baumgart, S., Khamespanah, E. and Sedaghatbaf, A.: An actor-based design platform for system

of systems, *43rd IEEE Annual Computer Software and Applications Conference, COMPSAC 2019*, Vol.1, pp.579–587 (2019).
[2] Reynisson, A., Sirjani, M., Aceto, L., Cimini, M., Jafari, A., Ingólfsdóttir, A. and Sigurdarson, S.: Modelling and Simulation of Asynchronous Real-Time Systems using Timed Rebeca, *SCP*, Vol.89, pp.41–68 (2014).
[3] Sirjani, M., Movaghar, A., Shali, A. and de Boer, F.S. : Modeling and verification of reactive systems using rebeca, *Fundam. Inform.*, Vol.63, No.4, pp.385–410 (2004).
[4] Khamespanah, E., Sirjani, M., Viswanathan, M. and Khosravi, R.: Floating Time Transition System: More Efficient Analysis of Timed Actors, *Formal Aspects of Component Software - 12th International Symposium, FACS 2015* (2016).
[5] Khamespanah, E., Khosravi, R. and Sirjani, M.: An efficient TCTL model checking algorithm and a reduction technique for verification of timed actor models, *SCP*, Vol.153, pp.1–29 (2018).
[6] Lee, E.A. and Sirjani, M.: What good are models?, *Proc. Formal Aspects of Component Software - 15th International Conference, FACS 2018*, Lecture Notes in Computer Science, Bae, K. and Ölveczky, P.C. (Eds.), Vol.11222, pp.3–31, Springer (online), DOI: 10.1007/978-3-030-02146-7 (2018).
[7] Sharifi, Z., Mosaffa, M., Mohammadi, S. and Sirjani, M.: Functional and performance analysis of network-on-chips using actor-based modeling and formal verification, *ECEASST*, Vol.66 (2013).
[8] de Berardinis, J., Forcina, G., Jafari, A. and Sirjani, M.: Actor-based macroscopic modeling and simulation for smart urban planning, *Sci. Comput. Program.*, Vol.168, pp.142–164 (2018).
[9] Bagheri, M., Sirjani, M., Khamespanah, E., Khakpour, N., Akkaya, I., Movaghar, A. and Lee, E.: Coordinated actor model of self-adaptive track-based traffic control systems, *Journal of Systems and Software*, Vol.143, pp.116–139 (2018).
[10] Volvo, Innovation at volvo construction equipment (2018) (online), available from ⟨https://www.volvoce.com/global/en/this-is-volvo-ce/what-we-believe-in/innovation/⟩.
[11] Sirjani, M., Movaghar, A., Shali, A. and de Boer, F.: Modeling and Verification of Reactive Systems using Rebeca, *Fundamenta Informatica*, Vol.63, No.4, pp.385–410 (Dec. 2004).
[12] Sirjani, M. and Jaghoori, M.M.: Ten years of analyzing actors: Rebeca experience, *Formal Modeling: Actors, Open Systems, Biological Systems - Essays Dedicated to Carolyn Talcott on the Occasion of Her 70th Birthday*, pp.20–56 (2011).
[13] Hewitt, C.: Description and theoretical analysis (using schemata) of PLANNER: A language for proving theorems and manipulating models in a robot, MIT Artificial Intelligence Technical Report (1972).
[14] Agha, G.: *Actors: A Model of Concurrent Computation in Distributed Systems*, MIT Press, USA (1990).
[15] Aceto, L., Cimini, M., Ingólfsdóttir, A., Reynisson, A.H., Sigurdarson, S.H. and Sirjani, M.: Modelling and simulation of asynchronous real-time systems using Timed Rebeca, *FOCLASA*, pp.1–19 (2011).
[16] Sirjani, M. and Khamespanah, E.: On time actors, *Essays Dedicated to Frank De Boer on Theory and Practice of Formal Methods - LNCS 9660*, pp.373–392, Springer (2016).
[17] Rebeca: Rebeca Homepage, available from ⟨http://www.rebeca-lang.org/⟩.
[18] Dijkstra, E.W.: A note on two problems in connexion with graphs, *Numerische mathematik*, Vol.1, No.1, pp.269–271 (1959).
[19] Batchelor, G.K.: *An introduction to fluid dynamics*, Cambridge University Press (1973).
[20] Castagnari, C., de Berardinis, J., Forcina, G., Jafari, A. and Sirjani, M.: Lightweight preprocessing for agent-based simulation of smart mobility initiatives, *International Conference on Software Engineering and Formal Methods*, Springer (2017).
[21] AdaptiveFlow Project: Rebeca Homepage - AdaptiveFlow Project, available from ⟨http://www.rebeca-lang.org/allprojects/AdaptiveFlow⟩.
[22] Clarke, E.M., Klieber, W., Novácek, M. and Zuliani, P.: Model checking and the state explosion problem, *Tools for Practical Software Verification, LASER, International Summer School 2011*, pp.1–30 (2011).
[23] Bagheri, M., Khamespanah, E., Sirjani, M., Movaghar, A. and Lee, E.A.: Runtime compositional analysis of track-based traffic control systems, *SIGBED Review*, Vol.14, No.3, pp.38–39 (2017).
[24] Sirjani, M., Khamespanah, E., Mechitov, K. and Agha, G.: A compositional approach for modeling and timing analysis of wireless sensor and actuator networks, *SIGBED Review*, Vol.14, No.3, pp.49–56 (2017).
[25] Ptolemaeus, C.: *System Design, Modeling, and Simulation using Ptolemy II* (2014) (online), available from ⟨http://ptolemy.org/books/Systems⟩.
[26] Saddem, R., Naud, O., Godary-Dejean, K. and Crestani, D.: Decomposing the model-checking of mobile robotics actions on a grid, *Proc. 20th World Congress of the International Federation of Automatic Control, IFAC WC 2017* (2017).
[27] Smith, S.L., Tumova, J., Belta, C. and Rus, D.: Optimal path planning under temporal logic constraints, *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp.3288–3293, IEEE (2010).
[28] Rabiah, E. and Belkhouche, B.: Formal specification, refinement, and implementation of path planning, *Proc. 12th International Conference on Innovations in Information Technology, IIT 2016* (2016).
[29] Gu, R., Marinescu, R., Seceleanu, C. and Lundqvist, K.: Formal verification of an autonomous wheel loader by model checking, *Proc. 6th Conference on Formal Methods in Software Engineering, FormaliSE 2018, collocated with ICSE 2018*, pp.74–83 (2018).
[30] Zhou, Y., Maity, D. and Baras, J.S.: Timed automata approach for motion planning using metric interval temporal logic, *2016 European Control Conference, ECC 2016*, pp.690–695 (2016).
[31] Saberi, A.K., Groote, J.F. and Keshishzadeh, S.: Analysis of path planning algorithms: a formal verification-based approach, *Proc. 12th European Conference on the Synthesis and Simulation of Living Systems: Advances in Artificial Life, ECAL 2013*, pp.232–239 (2013).
[32] Quottrup, M.M., Bak, T. and Izadi-Zamanabadi, R.: Multi-robot planning: a timed automata approach, *Proc. 2004 IEEE International Conference on Robotics and Automation, ICRA 2004*, pp.4417–4422, IEEE (2004).
[33] Koo, T.J., Li, R., Quottrup, M.M., Clifton, C.A., Izadi-Zamanabadi, R. and Bak, T.: A framework for multi-robot motion planning from temporal logic specifications, *SCIENCE CHINA Information Sciences*, Vol.55, No.7, pp.1675–1692 (2012).
[34] Fainekos, G.E., Kress-Gazit, H. and Pappas, G.J.: Temporal logic motion planning for mobile robots, *Proc. 2005 IEEE International Conference on Robotics and Automation, ICRA 2005*, pp.2020–2025, IEEE (2005).
[35] Kress-Gazit, H., Fainekos, G.E. and Pappas, G.J.: Temporal-logic-based reactive mission and motion planning, *IEEE Trans. Robotics*, Vol.25, No.6, pp.1370–1381 (2009).
[36] DeCastro, J.A., Alonso-Mora, J., Raman, V., Rus, D. and Kress-Gazit, H.: Collision-free reactive mission and motion planning for multi-robot systems, *Robotics Research, Proc. 17th International Symposium of Robotics Research, ISRR 2015*, pp.459–476 (2015).
[37] Trinh, L.A., Ekström, M. and Cürüklü, B.: Dipole flow field for dependable path planning of multiple agents, *IEEE/RSJ International Conference on Intelligent Robots and Systems IROS* (2017).

Giorgio Forcina is a Ph.D. student at Malardalen University and a member of Cyber-Physical System Analysis research group. He graduated in 2016 in Computer Science in a double-degree Master Program from the University of Camerino and the Reykjavik University. He got his bachelor degree in Industrial Computer Science in 2014 from the University of Camerino. His research interests include formal methods, cyber-physical systems, and artificial intelligence.

Ali Sedaghatbaf received his B.S., M.Sc. and Ph.D. degrees in computer engineering (software) (in 2009, 2011 and 2017, respectively) from School of Computer Engineering, Iran University of Science and Technology (IUST), Tehran, Iran. His research interests include software architecture, model-driven engineering, machine learning, and cyber security. He is currently a researcher at Mälardalen University, Västerås, Sweden.

Stephan Baumgart is working as a functional safety systems engineer at Volvo Construction AB in Sweden. At the same time he is a Ph.D. student at MÃd'lardalen University in VÃd'sterÃěs, Sweden. His research interests include functional safety in system-of-systems and autonomous vehicles. Stephan has a Licentiate degree from MÃd'lardalen University and a M.Sc. from Humboldt University in Berlin, Germany.

Ali Jafari is currently collaborating with the research group at Malardalen University in Sweden under the supervision of Prof. Marjan Sirjani. His research interests include formal methods, model checking, formal verification, applying formal methods in system design in different areas including cyber-physical systems, Internet of things, and self-adaptive systems. He received his B.S. and M.S. degrees in Computer Engineering from Ferdowsi University in Iran in 2006 and 2008 respectively, and his Ph.D. degree in Computer Science from the Reykjavik University in Iceland in 2016. From 2016 to 2018, he worked as a postdoctoral researcher at Reykjavik University under the supervision of Prof. Marjan Sirjani.

Ehsan Khamespanah is an assistant professor at University of Tehran. He is graduated from a double-degree Ph.D. program in the ECE Department at the University of Tehran and the department of computer science at Reykjavik University. His research interests include formal methods, software testing, cyber-physical systems, and software architecture. Ehsan has a B.E. in computer engineering from Tehran University and M.Sc. from the Amirkabir University of Technology.

Pavle Mrvaljevic received an ICM scholarship in 2019 for Master studies in Sweden and, this year, he is graduating from a Master's program (M.Sc.) in Software Engineering at MÃd'lardalen University, as well as a Specialization program (Spec. Sci) in Information Technologies at Mediterannean University. He received his B.Sc. degree in 2019 at Mediterannean University in Information Technologies with a specialization in Software Engineering.

Marjan Sirjani is a Professor and chair of Software Engineering at MÃd'lardalen University, and the leader of Cyber-Physical Systems Analysis research group. Her main research interest is applying formal methods in Software Engineering. She works on modeling and verification of concurrent, distributed, and self-adaptive systems. Marjan and her research group are pioneers in building model checking tools, compositional verification theories, and state-space reduction techniques for actor-based models. She has been working on analyzing actors since 2001 using the modeling language Rebeca (http://www.rebeca-lang.org). Her research is now focused on safety assurance and performance evaluation of cyber-physical and autonomous systems in which she is collaborating with Ptolemy group at UC Berkeley. Marjan has been the PC member and PC chair of several international conferences including SEFM, iFM, Coordination, FM, FMICS, SAC, FSEN. She is an editor of the journal of Science of Computer Programming.