# AdaptiveFlow: An Actor-based Eulerian Framework for Track-based Flow Management

No author info because of double-blind review

## ABSTRACT

In this paper, we present AdaptiveFlow as a framework for modeling and analysis of track-based flow management systems. Typical examples for track-based systems are train systems, bus transportation systems, air traffic control systems and automatic transportation systems in warehouses. We use Hewitt actors as the model of computation and an Eulerian view for flow management. In AdaptiveFlow, we model tracks as actors, and moving objects as messages. The framework is equipped with adaptation policies to react to dynamic changes in the system. Timed Rebeca is used for system modeling, and Rebeca Model Checker is used for safety and performance analyses.

In order to investigate the applicability of the developed framework, we considered the Electric Site Research Project of Volvo Construction Equipment as a case study. In this project, a fleet of autonomous haulers is utilized to tra nsport materials in a quarry site. We performed two sets of experiments varying input parameters and analyzing their effects on safety and performance of the fleet.

## CCS CONCEPTS

• **Computing methodologies → Model verification and validation**; *Modeling and simulation*; *Modeling methodologies*; • **Software and its engineering → Model checking**;

## KEYWORDS

Actor model, Track-based flow management, Model checking, Adaptive model, Performance evaluation

**ACM Reference Format:**
No author info because of double-blind review. 2019. AdaptiveFlow: An Actor-based Eulerian Framework for Track-based Flow Management. In *Proceedings of ACM SAC Conference (SAC'19).* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/xx.xxx/xxx_x

## 1 INTRODUCTION

Nowadays, flow management is an essential part of a wide variety of systems. In many of these systems, objects are moving on predefined tracks. For example, we have trains on rails, cars on roads, automated vehicles in aisles of warehouses, airplanes in the airspace, or even packets in wired networks. We need to assure safety and optimal performance of these systems.


*SAC'19, April 8-12, 2019, Limassol, Cyprus*

ACM ISBN 978-1-4503-5933-7/19/04.
https://doi.org/xx.xxx/xxx_x

In Fluid Mechanics, two specifications are widely used for flow analysis: Eulerian and Lagrangian. Considering a river, and a boat floating on the river, in a Lagrangian model the observer is sitting on the boat and drifting down the river. However, in an Eulerian model, the observer is sitting on the bank of the river and watching the boat passing through different locations in the river.

In this paper, we propose AdaptiveFlow, as a framework that allows to model and analyze track-based flow management systems based on an Eulerian model. In AdaptiveFlow, the system specification consists of the following three elements:

- **environment** which provides a basis for object movement (e.g. urban roads for the public transportation system, or railway network for the train system),
- **topology** of the network of the points of interest (PoIs). Fuel stations for the airport system, or the loading/unloading points in a quarry, are examples of PoIs, and
- **configuration** of each moving object (including features like loading capacity, fuel consumption rate, and moving velocity).

This specification is translated to a Timed Rebeca (TRebeca) [12] model. TRebeca is an actor-based modeling language supported by Rebeca Model Checker (RMC) [6–8]. RMC can be used for both verifying the correctness of TRebeca models, and their performance evaluation (Section 2). RMC automatically checks deadlock freedom and deadline misses for any TRebeca model. Moreover, using assertions in the model, we can check several properties e.g. if the moving objects are moving correctly, if they collide with each other or obstacles, and if the current configuration may lead to a starving situation.

In AdaptiveFlow, particular attention is paid to the unpredictability of changes to the system environment. Bad weather conditions or unexpected obstacles may affect the efficiency and safety of the analyzed system. To adapt with these changes, the framework is equipped with three different adaptation policies (Section 4), which allow the moving objects to avoid collisions against all possible types of obstacles, whether they are static or dynamic.

We investigate the efficiency of our approach and the applicability of the AdaptiveFlow framework by doing a set of experiments. As a case study, we consider the Electric Site Research Project that Volvo Construction Equipment (VCE) is working on [3]. In this project, a fleet of autonomous machines (haulers) are utilized to transport materials in the quarry site. The detailed description of this scenario is presented in Section 3 and the corresponding TRebeca model is presented in Section 5. To illustrate the applicability of AdaptiveFlow for the Volvo Construction Equipment use case, in Section 6, we analyze different configurations for an outdoor site. Finally, in Section 7 we present the outcomes of the experiments, showing that performing many comparative experiments with different configurations, will lead users to select the one that is more in line with their expectations.

## 2 REBECA AND TREBECA

**Rebeca** [16, 17] is an actor-based modeling language with formal semantics and a rich tool support and an active community. A Rebeca model consists of two parts: a set of *reactive class* declarations and a *main* method. Each reactive class defines the properties and behavior of an actor (or *rebec*), and comprises three parts:

- **knownrebecs** which declares the set of rebecs that this rebec knows and may communicate with,
- **statevars** which defines the state variables of this rebec, and
- a set of **msgsrv** definitions each acting as a message server, and processing the messages received from the known rebecs.

In Rebeca, each rebec takes a message from its message queue and executes the corresponding message server. Execution of a message server takes place atomically and asynchronously, which is non-blocking for both sender and receiver.

The internal state of a rebec is represented by the valuation of its state variables, and the messages in its queue. The behavior of a Rebeca model is defined in terms of the concurrent processing of the messages passed between the rebecs. In **Timed Rebeca (TRebeca)** [12] some timing primitives are added to the Rebeca syntax to cover timing features that a modeler might need to address in a message-based, asynchronous and distributed setting. These features are:

- **Computation time:** the time needed for a computation.
- **Message delivery time:** the time needed for a message to travel between two objects (network delay).
- **Message expiration:** the time within which a message is still valid.
- **Periods of occurrences of events:** elapsed time between two consecutive occurrences of periodic events.

In TRebeca, each rebec has its own local clock, but there is also a notion of global time based on synchronizing the clocks of all the rebecs. Messages that are sent to a rebec are put in its message queue together with their arrival time, and their deadline. Message servers are executed atomically, but the duration of executing each message server is modeled. In addition, communication delay and deadline for execution of messages can be defined in the model. The timing primitives that are added to the Rebeca syntax to support these features are:

- **Delay**: *delay(t)*, increases the value of the local clock of the respective rebec by the amount *t*.
- **Deadline**: *r.m() deadline(t)*, after *t* units of time, the message is not valid any more and is purged from the queue (timeout).
- **After**: *r.m() after(t)*, the message cannot be taken from the queue before *t* time units have passed.

## 3 THE ELECTRIC QUARRY SITE CASE

In this section we describe a use case from the construction equipment industry. Construction equipment machines are multi-purpose machines which can be applied for different tasks and in different environments. One such an environment is a quarry site which is a typical open site, where rocks are extracted from the ground and processed to meet customer needs. Depending on the type of rocks (e.g. marble or granite), different products are created as an output.

In the context of this research, we consider a quarry site, where gravel of different granularity is produced for building construction, road work or railway beds. The rocks in such an application scenario are blasted in one area of the quarry and the big blocks are crushed into smaller transportable 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 targeted granularity. In today's applications, human operated machines are used to transport the material from the movable crusher to the stationary crusher.

In the electric site research project at Volvo Construction Equipment, the material transportation from the primary crusher to the secondary crusher is realized by a fleet of autonomous haulers (called HX).

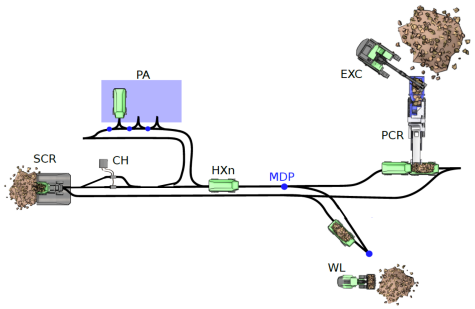


**Figure 1: Schema of the VCE Electric Site**

Figure 1 shows the tracks for the HX. They 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 HX are loaded, they travel to the secondary crusher (SCR) where they dump the load. Since the HX are electrified and equipped with batteries, they need to be charged at the chargers (CH). Since the missions are set by a central site control unit, which is supervising all activities, different queuing points are necessary where the HX receive their next mission. In order to make decisions for optimal production, the HX are queuing at the main decision point (MDP) and once a loading mission is assigned, the HX will move to the assigned loading position. The fleet of HX 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 scenario are exposed to harsh environmental conditions, which can change rapidly. Based on the changed conditions, the site control system must be able to adjust the fleet of HX by e.g. adding or removing loading spots, changing missions, routes or the number of HX. In summary, a new optimal setup of the fleet needs to be identified and rolled out on the site.

## 4 ADAPTIVEFLOW DESCRIPTION

In this section, we provide the reader with the detailed description of AdaptiveFlow, and its usage. The architecture of AdaptiveFlow is shown in Figure 2. It is composed of 3 modules which, in sequence, starting from the initial input files, process the outputs produced

by the preceding module and generate the final outcomes. The modules are: *Pre-Processing: Model Generation*, *Model-Run: State-Space Generation*, and *Post-Processing: State-Space Analysis*. The detailed description of these modules is provided in the following section.

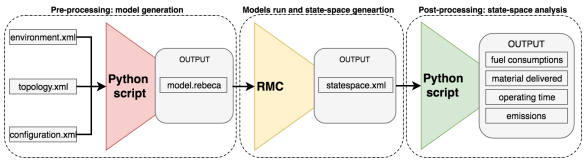


**Figure 2: AdaptiveFlow architecture**

## 4.1 The Pre-Processing Module

In the pre-processing module, AdaptiveFlow generates the TRebeca model based on the input specifications provided by the user. In detail, the user provides three input files in the Extensible Markup Language (xml) format, namely *environment*, *topology*, and *configuration*. A Python script processes these files and generates a TRebeca model and sends it as an input to the next module.

The **environment** input file contains the description of the base layer in which the movements of the transporting machines take place (Listing 1). Abstracting from many details, the environment can be seen as a collection of squared areas (segments) each characterized by a unique identifier (i.e. id) and coordinates (i.e. x and y). Each segment knows its neighboring segments and may differ from other segments in terms of length, traversing speed, and capacity. At any moment of time, each segment may be either available (i.e, it can be traversed), or unavailable (i.e. there is a static impediment e.g. a hole, a building, etc. in the segment). The maximum number of each segment's neighbors is equal to 8, one for each cardinal position (i.e. north, north-east, east, east-south, south, south-west, west, west-north).

```
<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 environment specification**

By means of the **topology** input file, the user specifies the locations within the environment in which the machines can perform their tasks (e.g. pick up passengers at a bus station, loading stones in a quarry, charging fuel, etc.), namely Points of Interest (PoI). Each PoI is characterized by its unique identifier (i.e. id), its position on the map (i.e. x and y), its type (i.e. the tasks it can process) and eventually its operating time. This latter represents the time needed for performing the specific task at the PoI. An example of the structure of the *topology.xml* file is provided in Listing 2. Currently, AdaptiveFlow supports only the most significant PoIs: *ParkingStation*, where machines are parked, *ChargingStation*, where machines can recharge their fuel, and *LoadUnloadingPoint*, where machines can load/unload materials.

```
<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 topology specification**

The **configuration** input file includes information about the configuration of the whole system. This information includes the following:

- *resendingPeriod*: the time for re-sending a segment request in case of a negative response (i.e. unavailability of the target segment),
- *numberMachines*: the number of operating machines,
- *safeDistance*: the safe distance between two machines to avoid collision,
- *fuelReserve*: the amount of fuel reserved by each machine,
- *policy*: the adaptation policy used to bypass dynamic obstacles,
- *maxAttempts*: number of attempts to send request to an unavailable segment before re-planning the path.

Furthermore, AdaptiveFlow allows users to specify the frequency of obstacle injection in the configuration file. In detail, the Python script that generates the TRebeca model, will inject random segment disruptions based on the values in the configuration file (see Listing 3). The aim is to emulate the concept of dynamic obstacles or bad weather conditions.

```
<system>
    <obstacleOccurences value="5"/>
  <obstacleNumber value="4"/>
  <obstacleMaxTime value="1000"/>
  <obstacleMaxDuration value="100"/>
</system>
```

**Listing 3: Example of random obstacle generation**

For what concerns the configuration of the machines (Listing 4), the user can specify:

- *id*: the unique identifier of the machine,
- *type*: the type of the simulated machine,
- *leavingTime*: the time in which the machine leaves the parking station and gets operational,
- *fuelCapacity*: the capacity of the machine's fuel tank,
- *fuelConsumption*: the fuel consumption rate of the machine,
- *speed*: the average speed of the machine,
- *emission*: the $CO_2$ emissions,
- *capacity*: the loading capacity of the machine,
- *unloadTime*: the time for unloading the materials.

Moreover, each machine comes with a sequence of tasks to perform in order to complete its daily operating cycle e.g., moving from the parking slot to the loading station.

```
<machines>
  <machine 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 >
  </machine> <\machines>
```

**Listing 4: Example of machine configuration**

## 4.2 The Model-Run Module

The so-generated TRebeca model is run with RMC for direct model checking the generated model. In detail, RMC converts the input (T)Rebeca model to a set of C++ files. These files are then compiled to an executable file. Running the executable file applies the model checking algorithm to the input model, and generates the verification results.

In addition to typical properties (e.g. safety, deadlock-freedom, etc.), AdaptiveFlow checks the following properties on the TRebeca model:

- if the machines are out of fuel,
- if the machines are moving correctly,
- if the machines collide with an obstacle or another machine,
- if the current configuration may lead to a starving situation.

Moreover, running RMC results in the generation of the whole state-space of the model, which can be used to evaluate different quantitative measures.

## 4.3 The Post-Processing Module

This module includes a Python script, which analyzes the state-space of the TRebeca model. In particular, each state of the system is analyzed and those measures that are meaningful for the analysis of the system are evaluated. These measures include the amount of consumed fuel, moved material, operating time and emitted $CO_2$.

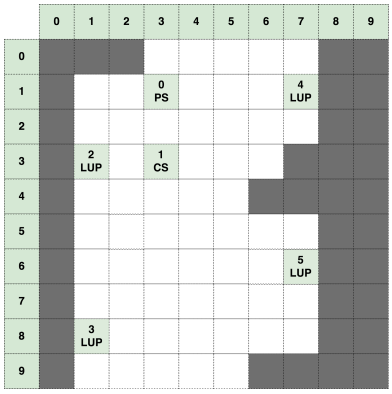


**Figure 3: The VCE electric quarry site environment**

## 4.4 Adaptation Policies

One of the most valuable features of the AdaptiveFlow framework is its support for the adaptation of the path plans to environment changes. As mentioned above, the framework is equipped with three policies to handle temporary unavailability of segments. The policies currently supported by AdaptiveFlow are:

- *postpone*, which allows the machine to postpone its planned movement by an amount of time that is equal to the *resendingPeriod* value specified in the configuration file. In case, the segment is unavailable for a number of attempts greater than the *maxAttempts* value, the re-route policy is applied.
- *passby*, which lets the machine to pass by the segment through its surrounding segments.
- *re-route* which exploits the Dijkstra shortest path algorithm [2] to find an alternative path between the current position and the destination PoI.

It is worth noting that, the aim of developing multiple policies is not to prove the effectiveness of a policy against the others, but to show that in AdaptiveFlow, it is possible to design and implement different adaptation policies based on the simulated context or user needs.

# 5 THE TREBECA MODEL

In this section, we describe the TRebeca model generated by AdaptiveFlow for the VCE quarry site case. As shown in Figure 3, the operational environment can be considered as a collection of segments. accordingly, the model consists of one reactive class named *Segment* representing the minimal portion of the environment in which machines can move. The behavior of each segment may differ from others depending on the input specifications. In particular, the topology of the PoIs within the environment affects how each *Segment* will behave when an event occurs. In case a *Segment* position corresponds to a PoI position, a machine that requires to perform a task at that PoI, needs to reach the corresponding *Segment*. For instance, considering the parking station located at (1, 3) in Figure 3, the *Segment* at this location is in charge of managing the machines at the parking station (i.e. machines waiting for task assignment). For the sake of completeness, the *Segment* in which the charging station is located (i.e. coordinate (3, 3) in Figure 3) represents the refueling station, and the load/unload PoI (e.g. coordinate (3, 1) in Figure 3) corresponds to the *Segment* in which machines can either load or unload materials.

In the TRebeca model, the state of each *Segment* is defined through the following state variables:

- *neighborSegments*, a list containing the 8 neighbor segments,
- *coord*, the coordinate of the segment within the environment,
- *currCapacity*, number of machines that can be in the segment simultaneously,
- *isParkingStation*, a Boolean variable indicating if the PoI is a parking station,
- *isChargingStation*, a Boolean variable indicating if the PoI is a charging station,
- *isLoadUnloadPoint*, a Boolean variable indicating if the PoI is a load/unload point.

The *Segment* class includes the following message servers:

- *initRoute*, finds the initial route (i.e. sequence of segments to traverse) from a machine position to a PoI by means of the Dijkstra shortest path algorithm [2],
- *moveToNext*, asks the next segment in the route for a permission to enter,
- *allowEntrance*,the *Segment* is informed that the machine is allowed to enter the next *Segment*,
- *inhibitEntrance*, the *Segment* is informed that the next position is currently out of capacity or temporally unavailable,
- *doMove*, moves the machine to the next segment in the route,
- *changeRoute*, applies an adaptation policy to change the current route,

Figure 4 illustrates the flow of events during the model execution using a flowchart. At start, the initial route is determined for each machine by the *initRoute* message server of the corresponding segment. Then, each machine follows its route by moving to the next segment in each time step. In case, the next segment was not available, the machine may change its route using one of the adaptation policies.

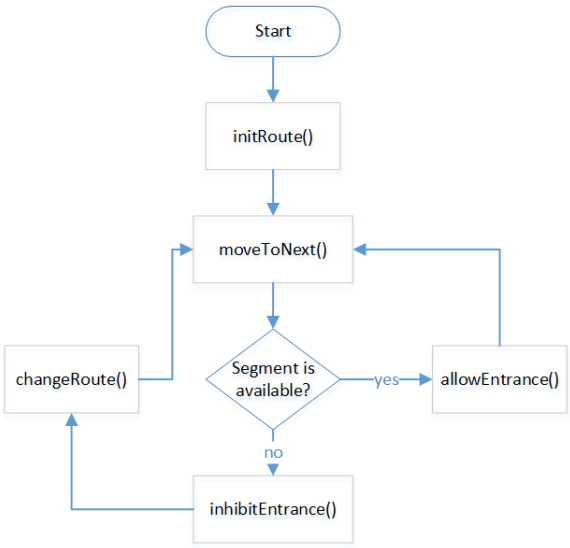


**Figure 4: Event flow in the TRebeca model.**

## 6 EXPERIMENTAL SETUP

In order to demonstrate the applicability of AdaptiveFlow, we describe the experimental setups for the VCE quarry site case in the following. Figure 3 is the graphical representation of the scenario we are interested to model and analyze. The environment is composed of 100 segments, 10 rows (i.e. x coordinate), and 10 columns (i.e. y coordinate). 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).

We performed two sets of experiments. In the first set, the goal was to evaluate how the position of a PoI, as well as the adaptation policies may affect the performance of the machines. We varied the positions of the charging and parking stations, while all the other parameters remained unchanged. Dynamic obstacles were generated randomly during the pre-processing phase, and the model was executed 45 times. Certainly, the output measures evaluated by the post-processing module would help users to easily select the configuration that best suits their needs (e.g. minimizing operational times, reducing fuel consumption, etc.).

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

**Table 1: Characteristics of machines A and B**

| Type | Fuel Capacity | Fuel Consumption Rate | CO2 Emission Rate | 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. |

## 7 ANALYSIS RESULTS

In this section, we discuss the results of the two sets of experiments presented in the previous section. For clarity, values shown in the figures refer to the fuel consumed, the $CO_2$ emitted and the time needed for executing all the given tasks. Moreover, the configurations are compared with respect to the adaptation policies.

For what concerns model checking, in all the experiments the properties mentioned in Section 4 were 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. no starvation), we noticed that the current design of the model did not support configurations in which two PoIs were adjacent, unless they could provide service to multiple machines at once.

An example of this situation is shown in Figure 5. The red machine has just finished charging fuel at (0, 0) and it is approaching the loading point at (1, 1). Vice versa, the blue machine needs to reach the gasoline charging station, since it almost ran out of fuel after having loaded materials at (1, 1). These two machines want to move to the same PoI simultaneously, and in this case, the first adaptation policy (i.e. postpone) is applied by both of them. Therefore, they will wait until the PoI becomes available again, which will never happen since both of them are waiting for each other.

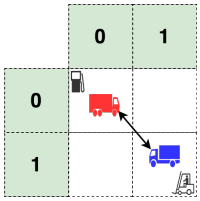


**Figure 5: Example of a configuration which leads to starvation**

***Changing PoI positions.*** Figures 6, 7, and 8 show the outcomes of the first set of experiments. Accordingly, the positions that optimize 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 adaptation policies, the one that minimizes the operating time is the third policy. With this policy, a machine'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,

avoiding the obstacle. As expected, the first policy (i.e. postpone) imposes the highest operating time. However, the fuel consumption is the lowest, since machines do not consume fuel when they are waiting. This is not the case for $CO_2$ emissions, since we assume that waiting machines 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.

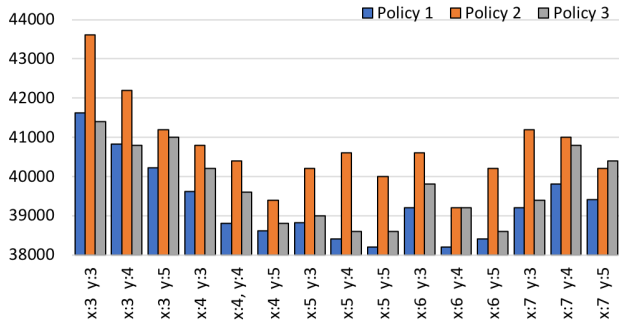


**Figure 6: Exp.1-Fuel consumption comparison**

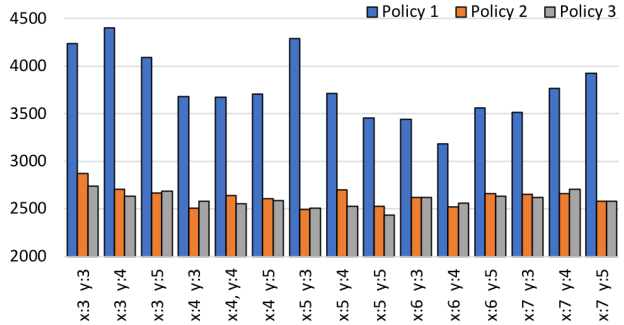


**Figure 7: Exp.1-CO2 emissions comparison**

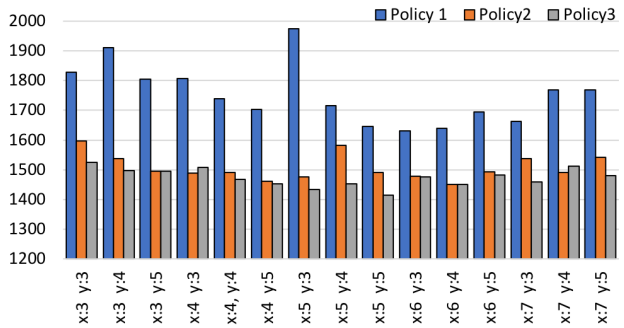


**Figure 8: Exp.1-Operating time comparison**

***Using different types of machines.*** In the second set of experiments, considering the results shown in Figures 9, 10, and 11, we can notice that replacing machines of type A with type B would increase both fuel consumption and $CO_2$ emissions. On the other hand, the operating time decreases with the increase in the number of machines of type B. This is true only when the maximum speed for each segment is greater than 6 m/s allowing machines of type B to exploit their greater velocity. It is also worth remarking that, differently from the first set of experiments, in which all the runs ended with a total amount of transported material that is equal to 1500 quintals, employing machines with higher transportation capacity led the system to be more productive. Indeed, the more is the number of type B machines, the higher is the amount of moved material, i.e., 1500, 1650, 1800, 1950, 2100, and 2250 quintals for configurations with 0, 1, 2, 3, 4, and 5 machines of type B, respectively.

From the adaptation policy point of view, the results indicate that fuel consumption is almost the same for all the three adaptation policies, and using either policy 2 or 3 instead of policy 1 would significantly reduce both $CO_2$ emissions and operating time.

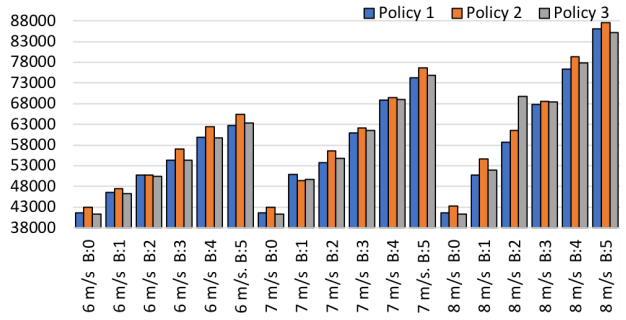


**Figure 9: Exp.2-Consumption comparison**

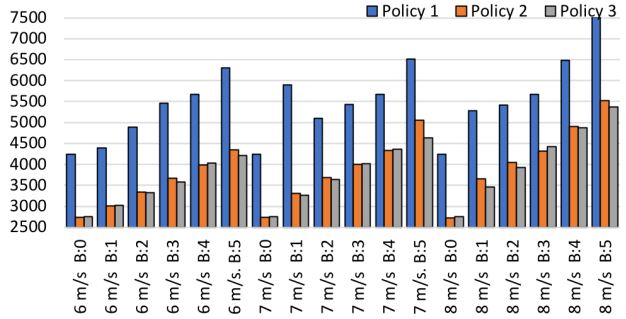


**Figure 10: Exp.2-CO2 Emissions comparison**

## 8 RELATED WORK

The closest work to this paper is presented by Bagheri et al. in [1]. They have targeted track-based traffic control systems in which the traffic flows through pre-specified sub-tracks and is coordinated by a traffic controller. They introduced a coordinated actor

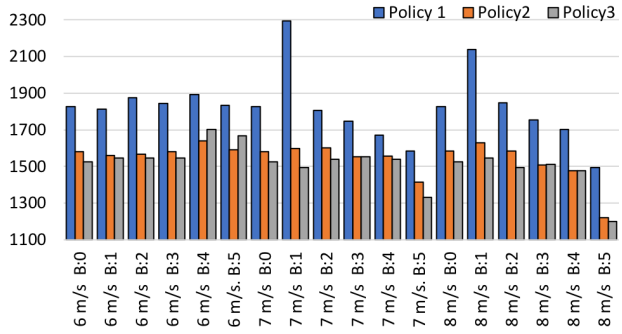


**Figure 11: Exp.2-Operating time comparison**

model and showed how to use it for simulating and analysis of self-adaptive systems. In comparison to this work, our work supports the decentralized implementation of control systems and there is no need for a centralized coordinator. In addition, we use model checking instead of simulation, which would lead to more accurate evaluations.

In [14], the authors proposed a model checking approach to path finding for mobile robots. They modeled the environment of a robot as a grid of cells. Starting from an initial cell, the robot was supposed to find a path to its destination performing a specific task at each cell on its way. The authors used timed automata to model the robot's behavior and analyzed reachability and safety properties using UppAal. Smith et al. [18] addressed the same problem using weighted transition systems and generalized LTL. They demonstrate that how in every environment model, and for every formula, a robot trajectory which minimizes the cost function is computed. It is clear that modeling in this approach is harder than that of [14] as it is in the lower level of abstraction. Authors in [11] did the same for the analysis of A* algorithm using Z modeling language and its corresponding tool-set. In comparison to these works, AdaptiveFlow is capable of modelling and analyzing the concurrent behaviors of multiple agents. This way, not only it is easier to model complex behaviors with TRebeca than automata, but also we can address the interference between different activities.

Authors in [13] addressed the safe path planning problem in a multi-robot configuration. They used mCRL2 to specify robots and the environment and examined that the collective behavior of a group of robots satisfied certain desired properties or not. They illustrated 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. In comparison to our work, the authors in [13] use the Lagrangian model instead of Eulerian which may require more message passing between robots. More message passing may increase the probability of state-space explosion. Similarly, the authors of [9, 10] used the Lagrangian model to address multi-robot path planing. However, they utilized timed and hybrid automata for model checking.

There are also some other contributions that use formal methods to synthesize safe paths for moving objects (e.g. [4, 15]). However, the main concern of the authors is only ensuring the safety of movements rather than choosing the optimal path in case of dynamic obstacles.

## 9 CONCLUSION AND FUTURE WORKS

In this paper, we presented AdaptiveFlow, as a general framework for modeling and analyzing track-based flow management. The framework is completely decentralized and generalized to support any track-based system that has the following characteristics: moving objects that transport an asset (e.g. passenger, material, etc.) among a number of dedicated locations (e.g. train station, airports, loading stations, etc.), and refuel at some charging stations.

In AdaptiveFlow TRebeca is used for modelling the moving objects and the Rebeca model checker is used to analyze the desired properties. AdaptiveFlow allows users to easily personalize the system by means of user-friendly input files, and to evaluate how their decisions can affect the throughput of the simulated system. Furthermore, the framework is designed in such a way that the objects can adapt their path plans to the unexpected changes in the environment. Currently, several cost functions such as fuel consumption and $CO_2$ emissions are supported by the framework.

As a future work, we will enrich AdaptiveFlow with more adaptation policies to handle unexpected changes in the environment. As an example, we are interested to implement the adaptation policy named *Dipole flow field* [5, 19] in our framework.

## REFERENCES

[1] M. Bagheri, M. Sirjani, E. Khamespanah, N. Khakpour, I. Akkaya, A. Movaghar, and E.A. Lee. 2018. Coordinated actor model of self-adaptive track-based traffic control systems. *Journal of Systems and Software* 143 (2018), 116–139. https://doi.org/10.1016/j.jss.2018.05.034

[2] Edsger W Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1 (1959), 269–271.

[3] Volvo Construction Equipment. 2018. Innovation at Volvo construction equipment. Retrieved 2018-09-20 from https://www.volvoce.com/global/en/this-is-volvo-ce/what-we-believe-in/innovation/

[4] Georgios E. Fainekos, Hadas Kress-Gazit, and George J. Pappas. 2005. Temporal Logic Motion Planning for Mobile Robots. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, ICRA 2005, April 18-22, 2005, Barcelona, Spain*. IEEE, 2020–2025. https://doi.org/10.1109/ROBOT.2005.1570410

[5] R. Gu, R. Marinescu, C. Seceleanu, and K. Lundqvist. 2018. Formal verification of an autonomous wheel loader by model checking. In *Proceedings of the 6th Conference on Formal Methods in Software Engineering, FormaliSE 2018, collocated with ICSE 2018, Gothenburg, Sweden, June 2, 2018*. 74–83.

[6] A. Jafari, E. Khamespanah, M. Sirjani, H. Hermanns, and M. Cimini. 2016. PTRebeca: Modeling and analysis of distributed and asynchronous systems. *Sci. Comput. Program.* 128 (2016), 22–50. https://doi.org/10.1016/j.scico.2016.03.004

[7] E. Khamespanah, R. Khosravi, and M. Sirjani. 2018. An efficient TCTL model checking algorithm and a reduction technique for verification of timed actor models. *SCP* 153 (2018), 1–29. https://doi.org/10.1016/j.scico.2017.11.004

[8] E. Khamespanah, M. Sirjani, M. Viswanathan, and R. Khosravi. 2016. Floating Time Transition System: More Efficient Analysis of Timed Actors. In *Formal Aspects of Component Software - 12th International Symposium, FACS 2015, Rio de Janeiro, Brazil, October 14-16, 2015*.

[9] T. John Koo, Rongqing Li, Michael Melholt Quottrup, Charles A. Clifton, Roozbeh Izadi-Zamanabadi, and Thomas Bak. 2012. A framework for multi-robot motion planning from temporal logic specifications. *SCIENCE CHINA Information Sciences* 55, 7 (2012), 1675–1692. https://doi.org/10.1007/s11432-012-4605-8

[10] M. Melholt Quottrup, T. Bak, and R. Izadi-Zamanabadi. 2004. Multi-robot Planning: a Timed Automata Approach. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation, ICRA 2004, April 26 - May 1, 2004, New Orleans, LA, USA*. IEEE, 4417–4422. https://doi.org/10.1109/ROBOT.2004.1302413

[11] E. Rabiah and B. Belkhouche. 2016. Formal Specification, Refinement, and Implementation of Path Planning. In *12th International Conference on Innovations in Information Technology, IIT 2016, Al Ain, UAE, November 28-30, Proceeding*.

[12] A.H. Reynisson, M. Sirjani, L. Aceto, M. Cimini, A. Jafari, A. Ingólfsdóttir, and S.H. Sigurdarson. 2014. Modelling and Simulation of Asynchronous Real-Time Systems using Timed Rebeca. *SCP* 89 (2014), 41–68.

[13] Arash Khabbaz Saberi, Jan Friso Groote, and Sarmen Keshishzadeh. 2013. Analysis of Path Planning Algorithms: a Formal Verification-based Approach. In *Proceedings of the Twelfth European Conference on the Synthesis and Simulation of Living Systems: Advances in Artificial Life, ECAL 2013, Sicily, Italy, September 2-6, 2013*, Pietro Liò, Orazio Miglino, Giuseppe Nicosia, Stefano Nolfi, and Mario Pavone

(Eds.). MIT Press, 232–239. https://doi.org/10.7551/978-0-262-31709-2-ch035

[14] Rim Saddem, Olivier Naud, Karen Godary-Dejean, and Didier Crestani. 2017. Decomposing the Model-Checking of Mobile Robotics Actions on a Grid. In *20th World Congress of the International Federation of Automatic Control, IFAC WC 2017, Toulouse, France, July 9-14, Proceeding*.

[15] Ali Narenji Sheshkalani and Ramtin Khosravi. 2018. Verification of visibility-based properties on multiple moving robots in an environment with obstacles. *International Journal of Advanced Robotic Systems* 15, 4 (2018), 1729881418786657. https://doi.org/10.1177/1729881418786657 arXiv:https://doi.org/10.1177/1729881418786657

[16] M. Sirjani and A. Movaghar. 2001. *An Actor-Based Model for Formal Modelling of Reactive Systems: Rebeca.* Technical Report CS-TR-80-01. Tehran, Iran.

[17] M. Sirjani, A. Movaghar, A. Shali, and F.S. de Boer. Dec. 2004. Modeling and Verification of Reactive Systems using Rebeca. *Fundamenta Informatica* 63, 4 (Dec. 2004), 385–410.

[18] Stephen L. Smith, Jana Tumova, Calin Belta, and Daniela Rus. 2010. Optimal path planning under temporal logic constraints. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 18-22, 2010, Taipei, Taiwan*. IEEE, 3288–3293. https://doi.org/10.1109/IROS.2010.5650896

[19] Lan Anh Trinh, Mikael Ekström, and Baran Cürüklü. 2017. Dipole Flow Field for Dependable Path Planning of Multiple Agents. In *IEEE/RSJ International Conference on Intelligent Robots and Systems IROS, 24 Sep 2017, Vancouver, Canada.*