An Actor-based Design Platform for System of Systems

Marjan Sirjani
School of Innovation, Design and Engineering
Mälardalen University
Västerås, Sweden
marjan.sirjani@mdh.se

Giorgio Forcina
School of Innovation, Design and Engineering
Mälardalen University
Västerås, Sweden
giorgio.forcina@mdh.se

Ali Jafari
School of Computer Science
Reykjavik University
Reykjavik, Iceland
ali@ru.is

Stephan Baumgart
Embedded System and Hardware Department
Volvo Construction Equipment AB
Eskilstuna, Sweden
stephan.baumgart@volvo.com

Ehsan Khamespanah
School of Computer Science
Reykjavik University
Reykjavik, Iceland
ehsank@ru.is

Ali Sedaghatbaf
School of Innovation, Design and Engineering
Mälardalen University
Västerås, Sweden
ali.sedaghatbaf@mdh.se

***Abstract*—In this paper we present AdaptiveFlow as a platform for designing system of systems. A model-based development approach is proposed and tools are provided for formal verification and performance evaluation. The actor-based language, Timed Rebeca, is used for modelling, and the model checking tool Afra is used for checking the safety properties and also for performance evaluation. We investigate the efficiency of our approach and the applicability of the developed platform by conducting experiments on a case study based on the Electric Site Research Project of Volvo Construction Equipment. In this project, a fleet of autonomous haulers is utilised to transport materials in a quarry site. We used three adaptive policies as plugins to our platform and examined these policies in different scenarios.**

***Index Terms*—System-of-systems, Actor model, Track-based flow management, Model checking, Performance evaluation**

I. Introduction

System-of-systems (SoS), or collaborating systems, are comprised of interdependent systems that collaborate to realise a common goal. According to Fitzgerald, systems-of-systems are formed by network-enabled synergistic collaborations between systems that are distributed, evolve dynamically and exhibit emergence [1]. Maier emphasizes that a system-of-systems, or alternatively collaborative system, should posses two characteristics regardless of the complexity or geographic distribution of its components: the components must have operational and managerial independence [2]. One of Maier's conclusions is that a collaborative system is defined by its interfaces because the interfaces are the primary points at which the designer can exert control, and the architecture should be largely defined by the communication model.

Some examples of collaborating systems are integrated manufacturing systems, enterprise information systems, emergency response collaborations and collaborative transportation systems. Intelligent Transportation Systems (ITS) is a typical example of collaborative systems [3]. The goals for a traffic management system include improving safety, increasing transportation system efficiency, enhance mobility, reduce fuel consumption and environmental cost, and increase economic productivity. Advanced Traffic Management Systems (ATMS) are a group of services in ITS with the long-term goal of coupling traffic management with route selection in individual vehicles. Other traffic management systems, like air traffic control or rail road scheduling, have similar goals.

In this work, we consider a generalised view to the traffic management system as an example of collaborating systems. Our focus includes a wide range of applications consisting of collaborating systems that are distributed, operate independently, and move around to accomplish a mission. We look at these applications as flow management systems and focus on flow management where the mobile systems move on tracks. Air traffic control, railroad scheduling, unmanned aerial vehicles (UAV) traffic management, smart transport hubs in cities, automated warehouses, and autonomous transport vehicles (ATVs) are examples where we have track-based traffic and transportation [4].

In many different applications, flow management is needed for a system of mobile systems that are traveling on pre-specified tracks. We see similar patterns where we have trains on rails, cars on roads, automated vehicles in aisles of a warehouse, airplanes in predefined airspace-tracks. These systems are mostly mission critical and we need to guarantee safety. Moreover, we need optimisation in different angles. According to Maier, in the field of collaborating systems, optimisation work must be refocused to get away from the point optimisation typical of conventional research and embrace the search for invariants and robust strategies that typify evolving systems [3].

Based on Lee's viewpoint [5], science and engineering are both all about models. Model-based development together with using formal verification and analysis help in building dependable systems. In [6], it is argued that faithful models generally increase the ease of use and flexibility, and often improve analysability. Faithfulness is about the similarity of the model and the system, and decreasing the semantic gap, where the structures and features of the model matches the ones of the system.

In this paper, we propose AdaptiveFlow, a model-based design platform that provides formal verification and performance analysis for track-based flow management systems. Using AdaptiveFlow, a designer can build a model of the collaborating system by defining the necessary properties of the network infrastructure, and the features and missions of the mobile systems. AdaptiveFlow is an actor-based platform, based on encapsulated actors with clear interfaces and asynchronous communication. So, the design naturally reflects the features of the collaborating system and provides a faithful model.

The actor-based modelling language, Timed Rebeca [7], that is used to build AdaptiveFlow is provided by formal semantics and is supported by model checking tools [8], [9]. Using model checking, we are able to verify the correctness of models and check the safety and progress properties [10]. The model checking tool of Timed Rebeca automatically checks deadlock freedom and deadline misses for a given Timed Rebeca model. Moreover, using assertions in the model, we can check more model-specific properties. In AdaptiveFlow we are for example interested in checking properties like collision avoidance, being on the correct track for the mission, and running out of resources or fuel.

In AdaptiveFlow, we also use Timed Rebeca Model Checking tool for performance evaluation and optimisation. We provide a usable interface for changing the parameters and explore the design space by checking different configurations for arrangement of collaborating systems. Possibility of dynamic changes both in the behaviour of each moving system and in the configuration of the network infrastructure are considered in the design platform. The moving systems can use different policies for adapting 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.

We use the case study of the Electric Site Research Project of Volvo Construction Equipment (VCE) [11] to show the applicability of AdaptiveFlow for designing a system of collaborating systems. 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 deign 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 guarantee the safety of each vehicle and the overall system safety.

In Section II we introduce Timed Rebeca, and in Section III we explain the VCE Electric Site example. In Section IV the AdaptiveFlow platform is described. In Section V, we present outcomes of experiments, showing comparative experiments with different configurations that helps the designer to make a decision. In Section VII, we conclude the paper and discuss the future work.

II. Rebeca and Timed Rebeca Languages

Rebeca (Reactive Object Language) [12]–[14] is an actor-based language. Actors are introduced by Hewitt [15] and promoted as a concurrent object-based functional language by Agha [16]. 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 top of its message queue, and executes the corresponding method (called *message server*) non-preemptively.

In Timed Rebeca (the real-time extension of Rebeca) [7], [17], [18], 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 actor is defined by the values of its state variables and the contents of its message buffer. Each actor has a set of *known rebecs* to which it can send messages. Reactive classes have constructors, with the same name as their reactive class. They are responsible for initializing the actor's state variables and putting initially needed messages in the message buffer of that actor. See Figure 1 for an abstract syntax of Timed Rebeca.

The way an actor responds to a message is specified in a *message server*. The state of an actor can change during the executing of its message servers through assignment statements. An actor makes decisions through conditional statements, communicates with other actors by sending messages, and performs periodic behavior by sending messages to itself. Since communication is asynchronous, each actor has a *message buffer* from which it takes the next incoming message. An actor 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 *nondeterministic assignment* statement which is used to model the nondeterminism in the behavior of a message server.

Finally, the `main` block is used to instantiate the actors of the model.

$$
\begin{aligned}
Model &::= Class^{*}\ Main \\
Main &::= \textbf{main}\ \{\ InstanceDcl^{*}\ \} \\
InstanceDcl &::= className\ rebecName\ (\langle rebecName\rangle^{*}) \\
&\quad : (\langle literal\rangle^{*}); \\
Class &::= \textbf{reactiveclass}\ className\ \{ \\
&\quad KnownRebecs\ Vars\ MsgSrv^{*}\ \} \\
KnownRebecs &::= \textbf{knownrebecs}\ \{\ VarDcl^{*}\ \} \\
Vars &::= \textbf{statevars}\ \{\ VarDcl^{*}\ \} \\
VarDcl &::= type\ \langle v\rangle^{+}; \\
MsgSrv &::= \textbf{msgsrv}\ methodName(\langle type\ v\rangle^{*}) \\
&\quad \{\ Stmt^{*}\ \} \\
Stmt &::= v = e;\ \mid v = ?(e\langle, e\rangle^{+});\ \mid Call; \mid \\
&\quad if\ (e)\ \{\ Stmt^{*}\ \}\ [else\ \{\ Stmt^{*}\ \}]; \mid \\
&\quad \textbf{delay}(t); \\
Call &::= rebecName.methodName(\langle e\rangle^{*}) \\
&\quad [\textbf{after}(t)][\textbf{deadline}(t)]
\end{aligned}
$$

Figure 1. Abstract syntax of Timed Rebeca (from [19]). Angled brackets ⟨...⟩ are used as meta parenthesis, superscript + for repetition at least once, superscript $*$ for repetition zero or more times, whereas using ⟨...⟩ with repetition denotes a comma separated list. Brackets [...] indicates that the text within the brackets is optional. Identifiers $className$, $rebecName$, $methodName$, v, $literal$, and $type$ denote class name, rebec name, method name, variable, literal, and type, respectively; and e denotes an (arithmetic, boolean or nondetermistic choice) expression.

Timed Rebeca adds three primitives to Rebeca to address timing issues: *delay*, *deadline* and *after* [7]. Each primitive is used as follows:

- **Delay**: *delay(t)* models the passage of time for an actor 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 keywords *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 keywords *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*).

III. 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, typical used for building construction, road work or railway beds, see Figure 2 . The rocks 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 the electric site research project at Volvo Construction Equipment the material transportation from the primary crusher to the secondary crusher is done by a fleet of autonomous haulers (called HX).

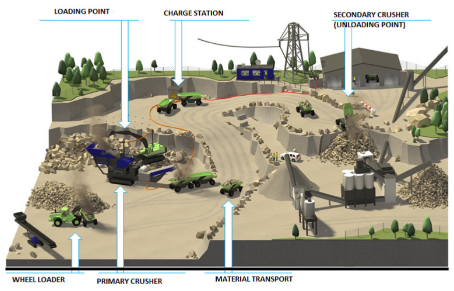

Figure 2. Volvo Construction Equipment Electric Site

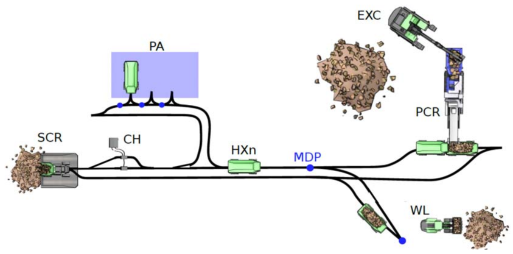

Figure 3. Schema of the Volvo Construction Equipment Electric Site

In Figure 3 tracks for the autonomous haulers are shown. The HX 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 are traveling 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). 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 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. Therefore, the site control system must be able to adjust the fleet of HX based on the new changed conditions.

IV. AdaptiveFlow Design Platform

AdaptiveFlow is built based on the actor-based modelling language Timed Rebeca and its model checking tool, Afra (see [20]). In Timed Rebeca, each independent system can be represented as an actor. The model has to be designed using encapsulated actors with no shared variables among them. Actors have a clear interface with the surrounding systems, and their model of communication is based on asynchronous messages. A collaborative system is a network of independent systems, and Timed Rebeca model captures the communication pattern and protocol among these systems. This is align with the principle mentioned by Maier [2], [3] that in a collaborative system the intersystem communications is the architecture. Afra is used to formally verify the properties specified as assertions or temporal logic formulas. While performing model checking, Afra generates the state space. In AdaptiveFlow, we explore the generated state space and provide the designer with performance metrics that can be used for design space exploration and optimisation purposes. We developed a set of tools surrounding Afra, we provide a user-friendly interface for the designer to input the configuration, then we use Afra to check the properties and generate the state space, and finally we explore the state space for performance analysis.

A. Timed Rebeca Models in AdaptiveFlow

The Timed Rebeca Model of the collaborating 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 as a collection of segments 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 obstacles 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 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 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 machines are parked when they are not operating, the *Charging Station* where machines can recharge, and the *Loading-Unloading Point* where machines can either load or unload materials.

In addition to the environment and the topology we need information regarding the *machines*, and the *configuration* of the system. For the Electric Site case study, each machine has its own identifier, its own mission, and its own type. The mission specifies the initial location of the machine, and the path it has to take. The type of the machine 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 machines, the safety distance between machines, and the level of fuel that should be reserved. As the model is representing the asynchrony of the system and communication among different systems, we model the request to enter to the adjacent cell by sending a message. This message can represent a query from the central control unit which is supervising all activities like in the VCE site, or represent a query from the adjacent cell if the cells are considered as smart units, or alternatively it can model a periodic checking of the surroundings by the machine itself. The request may get rejected because the adjacent cell may be occupied by another machine, or blocked due to some problem like an obstacle or a hole created by rain. So, the request need to be resent periodically. The period for resending the request if the first request gets rejected is an important parameter that can be set. During the analysis phase this period can be changed to find a safe and also more efficient configuration.

Another important feature is configuring the adaptation policies, we can set the parameters to show which adaptation 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 collaborating systems. In our Timed Rebeca model we can have event handlers (or message servers) that handle adaptation. In the current version, the model has been equipped with three algorithms (i.e. policies) to manage the situation where some segments are temporary unavailable. The policies are different based on the different decisions: wait for the blocked segment to be available again, by-pass the blocked segment and continue on the predefined route, run a routing algorithm and continue based on a complete new route. More policies can be added to the model. More detailed explanation is the following.

Policy 1 (postpone) allows the machine to postpone its planned movement by an amount of time that is equal to the *resending period* value given in the configuration input. 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 machine to bypass the segment that is occupied, allowing it to overpass that position according with the current environment and obstacles, and then continue on the pre-specified route in the mission plan.

Policy 3 (re-route) uses the Dijkstra shortest path algorithm [21] 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 V we compare the above policies from different viewpoints, but note that the main purpose here is not to prove the effectiveness of an algorithm against the others, 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. Based on the properties of interest, we may use two design patterns in building the Timed Rebeca model in AdaptiveFlow. One pattern is modelling each mobile system as an actor, and the other is modelling each track of the infrastructure as an actor. These two patterns reflect the two general views for analysing a flow, inspired from fluid mechanics [4]. In fluid mechanics, two well-known alternative ways to model fluid flow are Lagrangian and Eulerian models [22]. In a 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 infrastructure is modelled as an actor, seems less faithful to the collaborating 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 resending period may represent the same phenomena in both patterns. So, we still may model the main features of the independent machines in the Eulerian approach. More in-depth discussion on this topic is outside the scope of this paper.

B. AdaptiveFlow Modules and Analysis Process

AdaptiveFlow consists of three modules for (1) pre-processing and building the Timed Rebeca model from a more friendly input, (2) running Afra and do the formal verification and generating the state space, and (3) post-processing and perform the analysis on the state space. The architecture used for AdaptiveFlow is based on [23] and can be seen in Figure 4.

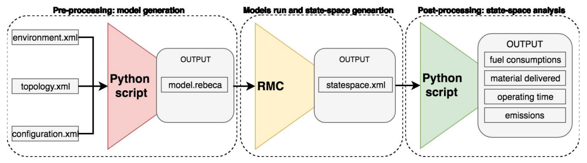

Figure 4. AdaptiveFlow Modules and Processes

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. The user provides three input files into Extensible Markup Language (XML) format, namely the *environment*, the *topology*, and the *configuration*. A Python script processes their content and generates the input for the next module (i.e. the TRebeca model).

Listing 1 illustrates a sketch of the Timed rebeca code generated for the VCE Electric Site (the full code can be accessed from AdaptiveFlow Web page [24]). This model includes 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 1 includes the declaration of the most important message servers of the segment actor. However, their internal behavior is eliminated due to space limitation.

The Timed Rebeca model generated from the pre-processing phase, is model checked using Afra (the Eclipse integrated tool) or Rebeca Model Checker (RMC), a tool for direct model checking of Rebeca models (see [20]). RMC is used to convert the Timed 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 and generates the verification result. RMC automatically verify if the model is deadlock free.

In addition, we also check the following properties:

- if the machines are out of fuel,
- if the machines are moving correctly on the predefined path,
- if the machines crash into each other or obstacles,
- if the current configuration may led 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.

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 machines and the system, for example: fuel consumed, material moved, operating time and CO_2 emitted.

V. 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 5 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 [24].

We perform two sets of experiments in this scenario. In the first set, the goal is to evaluate how the PoI positions, as well as the adaptive policies, affect the performance of the machines. In the second set of experiments, we consider two types of machines working in the quarry with different speed, fuel consumption, load capacity, fuel capacity, and CO_2

```
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 1. A sketch of Timed Rebeca model in AdaptiveFlow

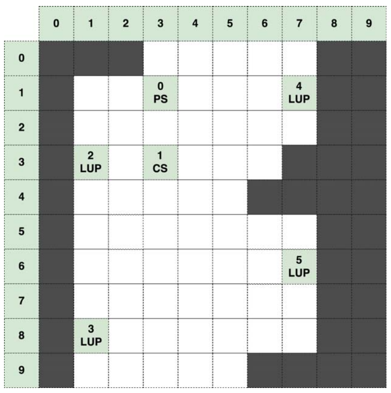

Figure 5. The environment used in experiments

emissions. The aim is to evaluate how replacing machines of type A with machines of type B affect the throughput.

Here, we discuss the results of the two sets of experiments. 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 the model does not support configurations in which two PoIs are adjacent, unless each PoI can provide service to more than one machine simultaneously.

An example of this situation is shown in Figure 6. The red machine has just finished charging fuel at (0, 0) and it is approaching the loading point at (1, 1). The blue machine needs to reach the charging station, since it almost ran out of fuel after having loaded materials at (1, 1). These two machines want to move to the other PoI simultaneously, and 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 and we will have deadlock.

Experiment 1: Changing the position of different Points

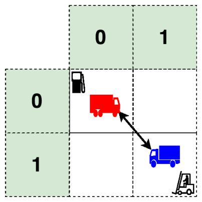

Figure 6. Example of a configuration which leads to starvation

of Interest. Here, the goal is to evaluate how the PoI positions, as well as the adaptive policies, affect the performance of the machines. 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 7, 8, and 9 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 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, 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 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.

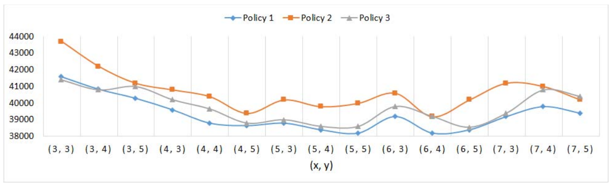

Figure 7. Exp.1-Fuel consumption comparison

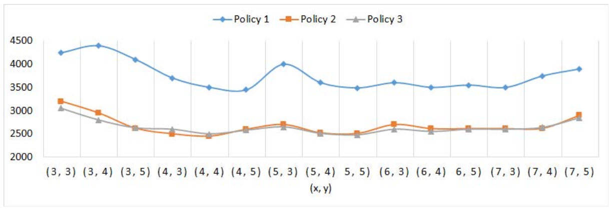

Figure 8. Exp.1-CO2 emission comparison

Experiment 2: Using different types of machines. In the second set of experiments, we consider two types of machines working in the quarry. They differ in terms of speed, fuel consumption rate, load capacity, fuel capacity, and CO_2 emissions. Table I shows the characteristics of these types of machines. The aim is to evaluate how replacing machines of type A with machines of type B affect 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 machines 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 10, 11, and 12, we notice that replacing machines 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 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 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 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 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.

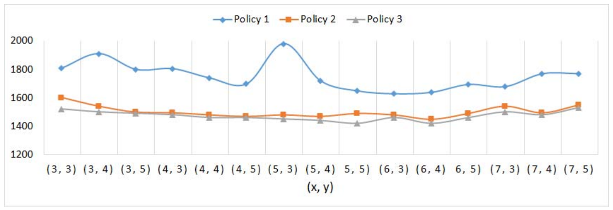

Figure 9. Exp.1-Operating time comparison

Table I
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.

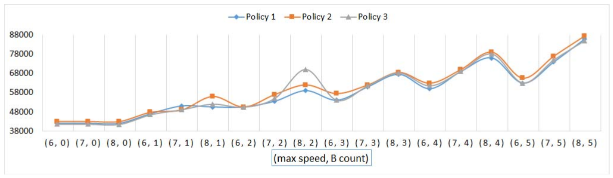

Figure 10. Exp.2-Fuel consumption comparison

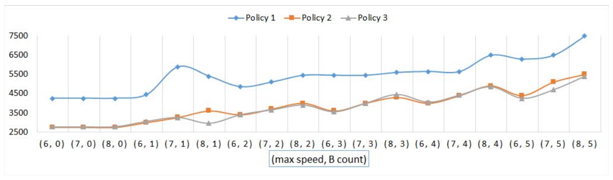

Figure 11. Exp.2-CO2 emission comparison

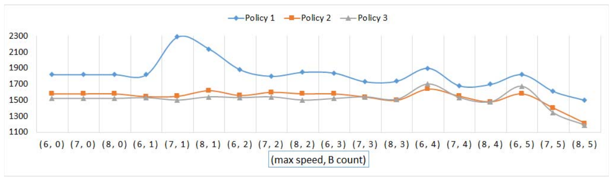

Figure 12. Exp.2-Operating time comparison

VI. Related Work

Bagheri et al. proposed a coordinated actor model in [25] 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 [25], 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 [26] simulator.

There are some work on using formal methods for analysis of path-finding in AGVs (Automated Guided Vehicles). Authors in [27] propose an approach for AGVs to explore an unknown grid-based environment and find a path to a destination. They modelled the problem using timed automata and analysed it using UPPAAL. To make the analysis possible, they showed that 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 [28] addressed the motion of the robot in the environment using weighted transition systems for the model specification and generalised LTL formula for the goal specification. Their approach shows that 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 comparing to the work of [27]. Authors in [29] have a similar approach for analysing of A* algorithm, using Z modelling language and its corresponding toolset. Authors in [30] modelled a vehicle and consider different features, such as position localisation, human and obstacle detection, collision avoidance and analyse the model to avoid fatal accidents. They provide a timed automata description of the vehicle's control system, including the abstracted path planning and collision avoidance algorithms used to navigate a vehicle, and model checked it using UPPAAL. All the above mentioned works are focused on path-finding in one machine while we consider a a set of collaborating machines and address the interference of activities of different machines.

Authors in [31] addressed the safe path planning problem in the multi-robot configuration. They used 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 [31] look into their neighbouring cells in each step and proceed one step and they do not have specific missions or plans to reach their destinations. In addition, based on what they mentioned 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 [32] and with hybrid automata in [33]. For the latter, they showed how the generated hybrid automata can be embedded into automata which can be model checked using SMV. They discussed that such an embedding does not change the result of verification for reachability properties.

VII. Conclusion and Future Works

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 customise the system by means of user-friendly input files, and to evaluate how their decisions can affect the throughput of the simulated 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 consumptions and CO_2 emissions. This is done automatically by AdaptiveFlow thanks to the chain of modules explained in Section IV.

As future work, we plan to enrich AdaptiveFlow with more adaptation policies to handle other unexpected changes in the environment. In particular, we will add a policy that avoid machines to stuck in situations like the one explained in Section V. Moreover, we will implement the adaptive algorithm named *Dipole flow field* described in [34] and used by [30]. We are also working on generating ROS (Robot Operating System) code from the Timed Rebeca models of AdaptiveFlow.

Acknowledgment

The work of the first and second authors are supported in part by DPAC Project (Dependable Platforms for Autonomous Systems and Control) at Malardalen University, Sweden, and the work of the third and fifth authors are supported by Self-Adaptive Actors: SEADA (nr 163205-051) project from the Icelandic Research Fund.

References

[1] J. Fitzgerald, J. Bryans, and R. Payne, "A formal model-based approach to engineering systems-of-systems," in *Collaborative Networks in the Internet of Services*, L. M. Camarinha-Matos, L. Xu, and H. Afsarmanesh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 53–62.

[2] M. Maier, "Architecting principles for systems-of-systems," *Systems Engineering*, vol. 1, p. 267–284, 1998.

[3] ——, "Research challenges for systems-of-systems," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Waikoloa, Hawaii, USA*, 2005, pp. 3149–3154.

[4] E. A. Lee and M. Sirjani, "What good are models?" in *Formal Aspects of Component Software - 15th International Conference, FACS 2018, Pohang, South Korea, October 10-12, 2018, Proceedings*, ser. Lecture Notes in Computer Science, K. Bae and P. C. Ölveczky, Eds., vol. 11222. Springer, 2018, pp. 3–31. [Online]. Available: https://doi.org/10.1007/978-3-030-02146-7_1

[5] E. A. Lee, "Modeling in engineering and science," *Commun. ACM*, vol. 62, no. 1, pp. 35–36, 2019. [Online]. Available: https://dl.acm.org/citation.cfm?id=3231590

[6] M. Sirjani, "Power is overrated, go for friendliness! expressiveness, faithfulness, and usability in modeling: The actor experience," in *Principles of Modeling - Essays Dedicated to Edward A. Lee on the Occasion of His 60th Birthday*, 2018, pp. 423–448. [Online]. Available: https://doi.org/10.1007/978-3-319-95246-8_25

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

[8] E. Khamespanah, M. Sirjani, M. Viswanathan, and R. Khosravi, "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*, 2016.

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

[10] A. Jafari, E. Khamespanah, M. Sirjani, H. Hermanns, and M. Cimini, "Ptrebeca: Modeling and analysis of distributed and asynchronous systems," *Sci. Comput. Program.*, vol. 128, pp. 22–50, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.scico.2016.03.004

[11] V. C. Equipment, "Innovation at volvo construction equipment," 2018. [Online]. Available: https://www.volvoce.com/global/en/this-is-volvo-ce/what-we-believe-in/innovation/

[12] M. Sirjani and A. Movaghar, "An actor-based model for formal modelling of reactive systems: Rebeca," Tehran, Iran, Tech. Rep. CS-TR-80-01, 2001.

[13] M. Sirjani, A. Movaghar, A. Shali, and F. de Boer, "Modeling and Verification of Reactive Systems using Rebeca," *Fundamenta Informatica*, vol. 63, no. 4, pp. 385–410, Dec. 2004.

[14] M. Sirjani and M. M. Jaghoori, "Ten years of analyzing actors: Rebeca experience," in *Formal Modeling: Actors, Open Systems, Biological Systems - Essays Dedicated to Carolyn Talcott on the Occasion of Her 70th Birthday*, 2011, pp. 20–56.

[15] C. Hewitt, "Description and theoretical analysis (using schemata) of PLANNER: A language for proving theorems and manipulating models in a robot," MIT Artificial Intelligence Technical Report, Tech. Rep., 1972.

[16] G. Agha, *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, USA, 1990.

[17] L. Aceto, M. Cimini, A. Ingólfsdóttir, A. H. Reynisson, S. H. Sigurdarson, and M. Sirjani, "Modelling and simulation of asynchronous real-time systems using Timed Rebeca," in *FOCLASA*, 2011, pp. 1–19.

[18] M. Sirjani and E. Khamespanah, "On time actors," in *Essays Dedicated to Frank De Boer on Theory and Practice of Formal Methods - LNCS 9660*. Springer, 2016, pp. 373–392.

[19] E. Khamespanah, M. Sirjani, Z. Sabahi-Kaviani, R. Khosravi, and M. Izadi, "Timed rebeca schedulability and deadlock freedom analysis using bounded floating time transition system," *Sci. Comput. Program.*, vol. 98, pp. 184–204, 2015.

[20] Rebeca, "Rebeca Homepage: http://www.rebeca-lang.org/."

[21] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[22] G. K. Batchelor, *An introduction to fluid dynamics*. Cambridge, UK: Cambridge University Press, 1973.

[23] C. Castagnari, J. de Berardinis, G. Forcina, A. Jafari, and M. Sirjani, "Lightweight preprocessing for agent-based simulation of smart mobility initiatives," in *International Conference on Software Engineering and Formal Methods*. Springer, 2017.

[24] AdaptiveFlow Project, "Rebeca Homepage - AdaptiveFlow Project: http://www.rebeca-lang.org/allprojects/AdaptiveFlow."

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

[26] C. Ptolemaeus, *System Design, Modeling, and Simulation using Ptolemy II*. Berkeley, CA: Ptolemy.org, 2014. [Online]. Available: http://ptolemy.org/books/Systems

[27] R. Saddem, O. Naud, K. Godary-Dejean, and D. Crestani, "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*, 2017.

[28] S. L. Smith, J. Tumova, C. Belta, and D. Rus, "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, 2010, pp. 3288–3293. [Online]. Available: https://doi.org/10.1109/IROS.2010.5650896

[29] E. Rabiah and B. Belkhouche, "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*, 2016.

[30] R. Gu, R. Marinescu, C. Seceleanu, and K. Lundqvist, "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*, 2018, pp. 74–83.

[31] A. K. Saberi, J. F. Groote, and S. Keshishzadeh, "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*, P. Liò, O. Miglino, G. Nicosia, S. Nolfi, and M. Pavone, Eds. MIT Press, 2013, pp. 232–239. [Online]. Available: https://doi.org/10.7551/978-0-262-31709-2-ch035

[32] M. M. Quottrup, T. Bak, and R. Izadi-Zamanabadi, "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, 2004, pp. 4417–4422. [Online]. Available: https://doi.org/10.1109/ROBOT.2004.1302413

[33] T. J. Koo, R. Li, M. M. Quottrup, C. A. Clifton, R. Izadi-Zamanabadi, and T. Bak, "A framework for multi-robot motion planning from temporal logic specifications," *SCIENCE CHINA Information Sciences*, vol. 55, no. 7, pp. 1675–1692, 2012. [Online]. Available: https://doi.org/10.1007/s11432-012-4605-8

[34] L. A. Trinh, M. Ekström, and B. Cürüklü, "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*, 2017.