Coordinated Actors for Reliable Self-adaptive Systems

Maryam Bagheri[1(✉)], Ilge Akkaya[2], Ehsan Khamespanah[3,4], Narges Khakpour[5], Marjan Sirjani[4,6], Ali Movaghar[1], and Edward A. Lee[2]

[1] CE Department, Sharif University Of Technology, Tehran, Iran
mbagheri@ce.sharif.ir
[2] EECS Department, University of California at Berkeley, Berkeley, CA, USA
[3] School of ECE, University of Tehran, Tehran, Iran
[4] School of CS and CRESS, Reykjavik University, Reykjavik, Iceland
[5] CS Department, Linnaeus University, Växjö Campus, Växjö, Sweden
[6] School of IDT, Mälardalen University, Västeras, Sweden

Abstract. Self-adaptive systems are systems that automatically adapt in response to environmental and internal changes, such as possible failures and variations in resource availability. Such systems are often realized by a MAPE-K feedback loop, where Monitor, Analyze, Plan and Execute components have access to a runtime model of the system and environment which is kept in the Knowledge component. In order to provide guarantees on the correctness of a self-adaptive system at runtime, the MAPE-K feedback loop needs to be extended with assurance techniques. To address this issue, we propose a coordinated actor-based approach to build a reusable and scalable model@runtime for self-adaptive systems in the domain of track-based traffic control systems. We demonstrate the approach by implementing an automated Air Traffic Control system (ATC) using Ptolemy tool. We compare different adaptation policies on the ATC model based on performance metrics and analyze combination of policies in different configurations of the model. We enriched our framework with runtime performance analysis such that for any unexpected change, subsequent behavior of the model is predicted and results are used for adaptation at the change-point. Moreover, the developed framework enables checking safety properties at runtime.

Keywords: Self-adaptive system · Model@runtime · Performance analysis · Cyber physical system · Air Traffic Control System

1 Introduction

The ubiquitous presence of software systems in everyday life, specially in safety-critical domains like Healthcare and transportation, makes building *reliable* cyber-physical systems (CPS) crucial. Moreover, to guarantee desirable behavior, systems need to evolve in response to environmental and internal changes, such as possible failures and variations in resource availability. Therefore, building reliable self-adaptive systems that are able to adjust their behavior in accordance

O. Kouchnarenko and R. Khosravi (Eds.): FACS 2016, LNCS 10231, pp. 241–259, 2017.
DOI: 10.1007/978-3-319-57666-4_15

with their perception of the environment and the system itself is an important research topic [9,22].

A self-adaptive system is typically realized through one or a collection of feedback loops that control adaptation of the core system. The MAPE-K feedback loop introduced in [17], is a common approach for realizing control feedback loops where a loop consists of *Monitor*, *Analyze*, *Plan* and *Execute* components, together with a *Knowledge* part, which contains information about the system and its environment. To guarantee correctness and quality of the self-adaptive systems, providing quality assurance techniques - in form of verification, validation and performance analysis- is a complicated and important issue. These techniques not only have to be applied in off-line manner, but also need to be used at runtime [8]. For instance, an abstract model of the system and the environment can be kept as the *model@runtime* in the Knowledge component, be updated periodically, and be analyzed for safety assurance, and also be used for runtime optimization and (re-)planning.

In order to design reliable self-adaptive systems, a number of the state of the art approaches benefits from the formal methods at the design time to assure the correct behavior of systems [13,18,19,30,32] and the behavior of the MAPE-K feedback loops [4,15]. In contrary, some approaches employ formal methods at runtime [7,15] or attempt to provide efficient techniques for runtime verification of systems [10–12]. However, a few of approaches have developed an integrated framework for constructing the MAPE-K loops and formal models@runtime, and providing runtime analysis techniques to detect or predict violation of the system's goals.

Here, we propose an actor-based [2,14,26] approach augmented with coordination policies for constructing and analyzing self-adaptive track-based traffic control systems. We create a total solution and build the MAPE-K feedback loop together with the model@runtime, but the focus of this paper is on building and analyzing the model@runtime. To this aim, we encapsulate the Analyze and Plan components of the MAPE-K feedback loop as a coordinator, together with the Knowledge component which keeps an updated actor-based model of the system. Monitor and Execute components can smoothly be added using features provided by our proposed implementation platform. It is noteworthy that the presented model@runtime is executable benefiting from java-based definition of the actors in the proposed implementation platform. The actor-based approach is aligned with the structure of distributed self-adaptive systems with behavioral adaptation. Loosely coupled actors as the units of concurrency, with asynchronous message passing, and event-driven computation, are natural candidates for modeling highly dynamic distributed systems. Moreover, in our problem domain, track-based traffic control systems, we have a (multiple) centralized control which is mimicked here as a coordinator. This enhances the properties of our interest (fidelity), and it is easy to understand and build the model with the least needed effort (usability).

The problem domain of the proposed framework, track-based traffic control systems, is a class of traffic control systems in which the traffic is passed through

pre-specified tracks and is coordinated by a controller, like railways and some air traffic control systems. These systems follow a common structural and behavioral pattern founded on the track-based configuration. In our model, actors represent the tracks and the moving objects are modeled as messages passing among actors. Our model@runtime has to capture the current configuration which is mainly the placement of messages within the model. The coordinator in our model besides governing message passing between the actors, analyzes the current configuration of the model, predicts the future configuration and makes a plan for adaptation purpose. Modeling a coordinator with mentioned features, separate from functionalities of the actors results in reusability.

To illustrate the applicability of the proposed approach, we model an Air Traffic Control system (ATC), a large scale cyber-physical system, as our real-world case study. Our ATC model is a simplified version of North Atlantic Organized Track System (NAT-OTS) [16], consisting of a set of transatlantic flight routes, and can easily be generalized to other track-based traffic control systems (e.g., railway systems). ATCs are responsible for managing the flow of the air traffic and assuring the safe flight of the aircraft. ATC is a prototypical example where people in charge are constantly dealing with various kinds of changes in the environment and system, such as changes in weather, propagating delays, strikes of the support staff, or unforeseen problems in the aircraft.

Different adaptation policies are implemented to check how changes in environment can affect the performance of the system. Furthermore, we carry out a mechanism in which the performance of the model is predicted using different policies and the prediction results are used for switching between adaptation policies at runtime.

Our proposed framework is implemented in Ptolemy [23], which is an actor-oriented open source platform that provides an extensive actor library for modeling, simulation and analysis of the system feedback loop, provides support for connecting to the physical world and updating the model@runtime, and also offers a graphical design environment that aids in visualization of the system architecture.

Our contributions are summarized as follows.

1. *Coordinated Actor-based model@runtime*: Proposing a modular, reusable, and executable coordinated actor-based model@runtime for track-based traffic control systems,
2. *ATC model@runtime*: Developing executable model@runtime for automated ATC as a real-world large scale cyber-physical system,
3. *Performance Evaluation*: Providing a framework for comparing different adaptation policies and evaluating their impacts on the performance of the system,
4. *Runtime Prediction*: Looking ahead through the model@runtime to explore the future behavior of a model to predict property violations and select appropriate policy at change-points.

The rest of the paper is organized as follows: in Sect. 2, we provide a general overview of track-based traffic control systems and introduce ATCs as our

running example. Section 3 presents coordinated actor-based modeling approach and shows how it can be used for building and analyzing model@runtime of track-based traffic control systems, e.g. ATCs. In Sect. 4, Ptolemy platform is introduced briefly and the implementation of an ATC in Ptolemy is explained. Section 5 demonstrates analysis results of the implemented system under different adaptation policies and runtime performance prediction. We describe related work in Sect. 6, and compare the proposed approach with the previous studies. Section 7 concludes the paper and outlines the future work.

2 Track-Based Traffic Control Systems

Transportation systems can be managed in different ways. We focus on those systems where the traffic can only move on certain tracks, like in railways, subways, and even roads. Traffic control for such systems follow a common structural and behavioral pattern. Tracks are generally divided into sub-tracks and the traffic controller uses this structure to guarantee safety and improve performance.

Air traffic is not necessarily guided through predefined tracks, but the ATC system in North Atlantic is based on an Organized Track System (OTS) which follows the same pattern, and is the system we have studied [1]. This pattern can be further generalized, for example a network on chip (NoC) can be considered as a track-based traffic control system in which a packet (moving object) is transferred via a set of routers (sub-tracks).

To have a fully automated traffic control system, we need a control algorithm for managing the traffic, taking into account the environment and congestion conditions. This control algorithm has several decision making capabilities, including accepting or rejecting traffic into a sub-track, and routing and rerouting traffic. Based on the nature of a system, the control algorithm can be distributed or centralized. For example, the control algorithm in NoC is distributed among routers (sub-tracks). In [24], an actor model is used for modeling and analyzing NoCs where each router (sub-track) is mapped into an actor and packets are mapped into messages. Railway systems are examples of centralized traffic control systems. A controller knows the complete map of the traffic network with the details of the current traffic. This knowledge is updated upon any change in the system, and is used for online planning.

In our model, sub-tracks can be augmented with decision making abilities. For example, based on the capacity of a sub-track, it can decide to allow or reject a moving object to enter the sub-track. This way, the safety of a sub-track (mutual exclusion) is wired in at design time. But this property can cause deadlock and fatal problems that has to be avoided. In our framework we use a centralized coordinator to manage the overall traffic and avoid deadlocks. The framework can be customized based on different applications. In our current implementation, the required time for a moving object to pass a sub-track is computed by the controller based on the speed of the object and the length

[1] This system is studied in collaboration with Isavia, the air traffic control company in Iceland (http://www.isavia.is).

of the sub-track. Furthermore, at a certain time, a sub-track is reserved as the next sub-track of a moving object by the controller. This way collision is avoided among objects that may request to enter into a sub-track simultaneously. If we encounter a traffic blockage, the controller handles the situation by rerouting the traffic objects, asking for changes in traffic speed, etc. We will show in the next sections how coordinated actor model is used for modeling and analysis of these cases. To this aim, we use ATC as a running example for easier explanation of the modeling and the analysis approach.

Air Traffic Control System: ATC is a system equipped with supervision instruments on the ground and the links for communication, which monitors and controls flights along the airspace routes and operations in the airports [1]. In real-world, each aircraft has a predefined flight plan, which includes its flight route (an ordered sequence of the sub-tracks to be traversed from the source to the destination), speed of flight, and the initial fuel. The aircraft's flight plan is generated prior to its take off, but dynamic changes in the weather conditions, delays in landing and taxiing, etc., requires some modifications in flight plans. For safety concerns, while changing the original flight plan of an aircraft, several parameters are carefully considered, for example loss of separation between two aircraft has to be avoided, and the remaining fuel must be enough for safely arrival to destination. A number of researches in the ATC domain, such as [3,5,6], propose mathematical models and solutions for resolving congestion of air spaces, to achieve the minimum delay and safe movement of the aircraft by taking the rerouting decisions into account. According to [5], modeling rerouting decisions is one of the greatest challenges in this field. The framework presented in this paper (Fig. 3) addresses these challenges by modeling and runtime analysis of ATC. The structure of North Atlantic Organized Track System (NAT-OTS) [16] that we consider as our ATC model consists of a set of nearly parallel tracks, positioned in the light of the prevailing winds to suit the traffic between Europe and North America.

3 A Coordinated Actor Model for Self-adaptive Track-Based Traffic Control Systems

To develop a self-adaptive system using the MAPE-K loop, a model is kept in the Knowledge component, updated by the Monitor component, and analyzed by the Analyze component. Based on the analysis results, the Plan component makes decisions for adapting the system to the new configuration and the decisions are sent to the managed system using the Execute component. To ensure the correct behavior of self-adaptive systems despite the internal and environmental changes, keeping its knowledge model updated at runtime is crucial.

Figure 1 illustrates our overall approach for modeling self-adaptive track-based traffic control systems. As shown in Fig. 1, our proposed *coordinated actor model* realizes three components of the MAPE-K loops, Knowledge, Plan, and Analyze. Using this approach, the model of a self-adaptive system is divided into two different parts, actors and a coordinator. The model@runtime encloses

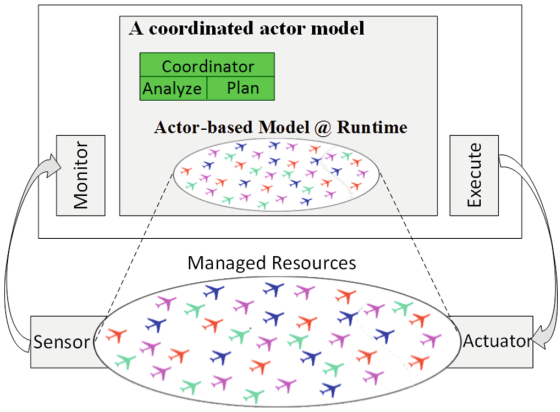

Fig. 1. Modeling self-adaptive track-based traffic control systems with coordinated actor model

the actors in addition to the part of the coordinator which handles the message delivery between the actors; and the Analyze and Plan components are the decision making parts of the coordinator. Actors communicate via asynchronous message passing, but unlike the original (Agha's) actor model, although the sent messages are immediately delivered to their target actors, the target actors do not pick messages. Instead, an event is triggered for the coordinator upon any message sending. The coordinator receives the event related to that message and puts it into its internal buffer. The internal buffer of the coordinator is sorted based on some predefined rules, e.g. an event with the least arrival time has the highest priority. Upon choosing an event with the highest priority, the coordinator can access the target actor, ask about its internal state, change the sent message, drop it, or execute the target actor to pick the message and process it. This way, the message passing among the actors is kept as the basic communication means in the target distributed system but message delivery policies are integrated in the coordinator. This mechanism gives the coordinator the necessary knowledge about the actors which is used in the decision making part. The Analyze part of the coordinator is activated when the updated knowledge shows some pre-specified changes. Analyze is capable of simply analyzing the acquired knowledge or using complicated methods such as executing the model@runtime to obtain new knowledge for predictive analysis (more details in Sect. 3.3). Then, the results of analysis are passed to the Plan part to choose appropriate actions for adapting the system (Managed Resources in Fig. 1).

3.1 Structure of Self-adaptive Track-Based Traffic Control Systems Model

In our proposed model to describe a track-based traffic control system, an actor is associated with each sub-track and the controller is modeled as a coordinator.

Furthermore, the moving objects are modeled as messages, passed by (through) the actors and messages carry the necessary information of moving objects, i.e. traveling route, fuel, and speed. In addition to the track actors and a coordinator, we distinguish the source and destination actors to model the source and the destination of moving objects. The coordinator has a complete map of the track network and current states of the sources and destinations of the traffic. The knowledge of the coordinator is updated by occurrence of events that change the internal state of the actors. These events can be environment changes in sub-tracks, and arrival or departure of moving objects into/from sub-tracks. In other words, upon accessing the actors by the coordinator, it polls their internal states and updates its knowledge.

Using the described structure, different adaptation policies are encapsulated in coordinators. In some traffic control systems, adaptation results in rerouting of traffic, like in ATCs; and in some others it may result in rescheduling, like in train control systems. In the following we will show how the rerouting of moving objects are implemented for our ATC example.

3.2 Adaptation in ATC: Dynamic Reroute Planning

In our ATC model design in Sect. 4, routing an aircraft in the network of track actors is modeled by a sequence of messages which are sent by adjacent track actors. To implement rerouting policies, different algorithms can be considered, affecting the performance of the system in different ways. In the following, three different policies for ATCs are presented. Assume $[T_1, T_2, T_3, \cdots, T_n]$ be a sequence of sub-tracks which shows the flight route associated with the aircraft A. In this sequence, T_1 is the sub-track started from the source airport and T_n is the destination airport. Now, when the aircraft A wants to leave T_1 based on this flight route and the subsequent sub-track on its flight route (sub-track T_2) is unavailable (i.e. it is stormy or is dedicated to another aircraft), the coordinator applies one of the following algorithms to reroute the aircraft A.

- **Safer Route Policy (SRP):** SRP finds the feasible shortest flight route from T_1 to T_n. A feasible flight route is a flight route which does not contain an unavailable sub-track. If there is no such route, the shortest flight route with the least number of unavailable sub-tracks is returned.
- **Blocked Area Avoidance Policy (BAAP):** Similar to the SRP, BAAP tries to find the feasible shortest flight route from T_1 to T_n. If there is no such route, it finds the shortest route which does not pass a blocked area. As shown in Fig. 2, a blocked area is a part of airspace in which all its sub-tracks have been occupied and surrounded by other aircraft at time of rerouting the aircraft A. In other words, an aircraft in a blocked area may hold its current sub-track due to the high traffic around it for a long time. Hence, the aircraft A probably would not be able to pass a sub-track in the blocked area if that sub-track is selected as a part of the aircraft's flight route. We consider three arrangements of the aircraft in the airspace which form the blocked areas. These arrangements are shown in Fig. 2. For instance, if four aircraft hold the

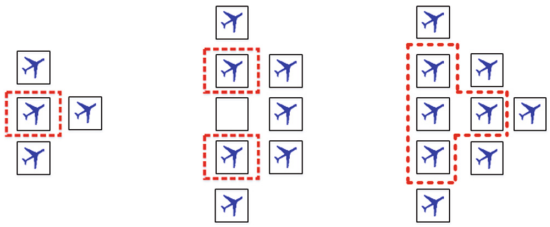

Fig. 2. Three different arrangements of the aircraft in the airspace which form blocked areas. The blocked areas containing unavailable sub-tracks are shown with the dotted-rectangles. The blocked areas are avoided in the BAAP rerouting policy.

sub-tracks such that the left shape of Fig. 2 appears in the airspace, a blocked area, shown by a dotted-rectangle, is formed. Since BAAP avoids areas with higher traffic, it is expected to impose less flight duration comparing to SRP.

– **Shrinking Search Policy (SSP):** This policy employs a multi step searching algorithm. At the first step, SSP tries to find a feasible shortest flight route from T_1 to T_n. If there is no such route, it tries to find a feasible shortest flight route from T_1 to T_{n-1} and so on. If a route is not found, aircraft A holds its position in T_1 for some amount of time, attempts to continue flying based on its initial flight route. Since SSP attempts to find the feasible shortest flight route for the largest possible part of the flight route, it is expected to result in less blocking probability for the aircraft in the airspace.

To improve the adaptation mechanism, one may enrich both planer and analyzer of the coordinator to select a rerouting policy from a pool of policies. For instance, assume that the analyzer is enriched with the ability of detecting blocked area in the whole airspace. This way, planer is able to use SRP as the default rerouting policy but upon increasing the number of blocked areas to a threshold, planer is informed to switch to BAAP to avoid passing the aircraft from the blocked areas.

3.3 Adaptation in ATC: Runtime Performance Prediction

The main purpose of runtime performance prediction is predicting the subsequent behavior of the system after occurrence of any unexpected changes and using the prediction results to adapt the system to the new configuration. To this aim, the required performance metrics and safety issues of the system are defined and a proper adaption takes place to fulfill them.

In the runtime performance prediction, the analyzer looks ahead through the executable model@runtime to explore the future behavior of the model to predict property violations. To this aim, upon detecting any safety violation or crossing the thresholds of the performance metrics, the planner is triggered and tries to adapt the system using its set of predefined prediction based adaptation policies. So, the models@runtime have to be augmented with techniques to record

the states of the models at the change points (taking snapshots of models), look ahead and backtrack as follows. In the coordinated actor model, the state of the model is defined as the local states of actors along with the state of the coordinator. The state of each actor is defined as the assigned values to its variables and the coordinator state is defined as the assigned values to its variables, communicated messages between the actors together with the triggered events, and the time. So, when a change happens (based on the coordinator knowledge), state of the model is stored by the analyzer, and model proceeds with its execution to the end. At the final step, the values of performance metrics which are gathered during the execution of the model are gathered by the analyzer. These values are used by the planner part of the coordinator to choose the most suitable adaptation policy for the change.

4 Coordinated Actor Model of ATC in Ptolemy

Ptolemy II is an actor-oriented open-source modeling framework. A Ptolemy model is made up of actors that communicate through ports via message passing. Actors are implemented in Java with a well-defined interface, which includes ports and parameters. Ports are communication points of actors and parameters are used to configure the internal behavior. Parameter values can be changed dynamically during the execution of model. In Ptolemy, the semantics of interactions and communications of actors is defined by a Model of Computation (MoC), implemented by a special component called a *Director*. Different MoCs are implemented in Ptolemy which can be composed to yield heterogeneous models. Ptolemy is open-source and new MoCs can be created by extending or modifying existing MoC implementations. Heterogeneous hierarchical actor-oriented design capabilities in Ptolemy provides a strong modeling and simulation toolkit for CPS design [23]. One of the most widely used MoCs in Ptolemy is Discrete-Event (DE). Actors governed by a DE director communicate via time-stamped events (i.e. time-stamped messages of actors), where events are processed by each actor in time-stamp order. Considering different MoCs in Ptolemy, we choose DE for ATC model, because time-stamped events are natural for representing the decision points in a track-based traffic control system.

Following the coordinated actor model presented in Sect. 3, we model coordinator of ATCs as a new Ptolemy DE director, we call ATCDirector. Here, airports and tracks are modeled using actors of Ptolemy and their interconnections showed by wiring of actors' ports. The model of a simplified ATC model is presented in Fig. 3. In this model traffic is passed from west to east. As shown in Fig. 3, there is a network of tracks where each track is shown by a white aircraft. To present a track which is associated to an aircraft, the distinct color of that aircraft is used for that track. Tracks affected by thunderstorms are highlighted with a red circle around them. As shown in Fig. 3, connections of a track to its neighbors are illustrated as the output ports of the track and ids of its neighbors are set as the parameter of the track. Arrival and departure airports are also modeled by Ptolmey actors, shown by blocks with captions of *Airport* and

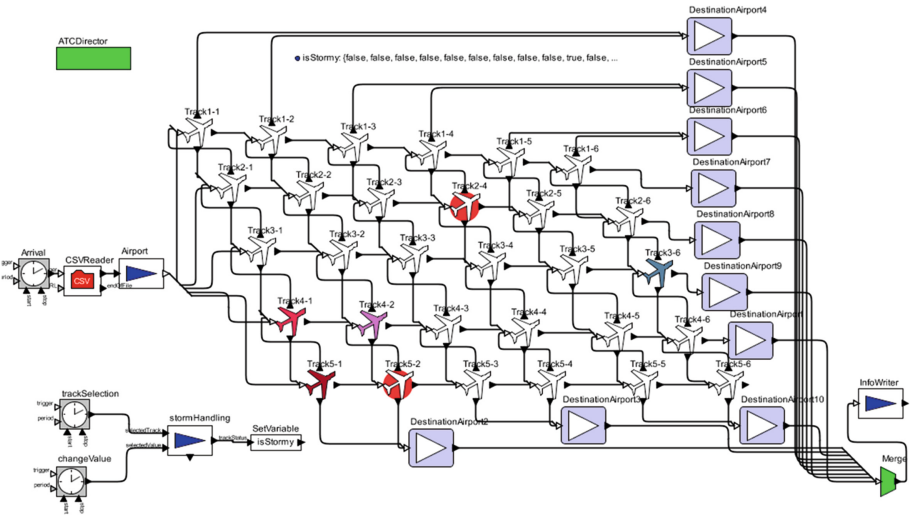

Fig. 3. ATC model in Ptolemy (Color figure online)

Destination Airport in Fig. 3. ATCDirector of this model is shown as a green rectangle with caption of the ATCDirector in Fig. 3.

To execute the ATC model, departure of the aircraft and weather changes are simulated as follows. In this model flights are initialized by Arrival and CSVReader actors. The Arrival actor is a discrete clock that determines the departure time of aircraft. CSVReader reads flight plans and other detailed information of the aircraft from a text file. In Fig. 3 three more actors are implemented for simulating the weather condition changes, called TrackSelection, ChangeValue, and StormHandling. TrackSelection selects a track (with a predetermined probability) and ChangeValue decides about how the weather condition of the selected track must be changed. ChangeValue has the role of both generating and removing storms. StormHandling applies the made decision of ChangeValue on its target track. The ATC model also can be fed by real values from real-world sensors using features of Ptolemy.

5 Safety Check and Performance Analysis

In this section, we study the effectiveness of our approach by evaluating and comparing various performance criteria for rerouting policies described in Sect. 3.2. Our results show that applying runtime performance prediction improves the performance metrics[2].

[2] The source code which is used for these experiments is uploaded in http://rebeca.cs.ru.is/files/ATC.zip.

5.1 The Experimental Settings

The performance metrics of the ATC are measured when an aircraft reaches its destination. In our experiments, the performance metrics are the number of aircraft arrived to their destinations, the number of missing aircraft (the aircraft that are stuck in a traffic blockage in the airspace and cannot find a route to their destinations using the rerouting policies), the number of aircraft blocked in their source airports, the average delay of the aircraft in their source airports, the average flight duration of the aircraft, and the throughput. Throughput is defined as the number of aircraft arrived in their destination airports in a specific period of time.

We assumed one unit of time for the takeoff/landing of an aircraft in addition to passing a sub-track. An aircraft sends a request to enter into a sub-track 0.2 unit of time before its planned arrival time. If the aircraft is asked to stay in its current sub-track, it waits for one more unit of time and sends a new request to enter into the same sub-track again. In addition, we assume that each airport has one runway and each sub-track can contain at most one aircraft at a time.

The ATC model in our experiments contains two source airports, eight destination airports and thirty tracks. There are fifty aircraft that intend to fly from a north airport to one of the three south destination airports and the same number of aircraft plan to fly from the west airport to the five destination airports in the east. The initial flight route of an aircraft is the shortest route from its source airport to its destination. For each source airport, twenty batches of flight plans are generated and each batch contains the flight plans for fifty aircraft. The departure times of the aircraft, and the times of the weather changes are produced using Poisson processes with parameters λ in $\{0.5, 2, 3.5\}$ (the mean interval time between two events) and 1 as μ, respectively.

5.2 Safety Checking

In the experiments, we check deadlock-freedom and a few safety issues. Deadlock is detected if none of the aircraft in the airspace change their current sub-tracks within a pre-determined time. This means that the aircraft are stuck in a traffic blockage and cannot find a route to their destinations. As safety issues, the separation between the aircraft, and safe rerouting in stormy weather are checked and guaranteed by the model. A sub-track in our model denotes the minimum separation between two aircraft, so, two aircraft must never be in a sub-track at the same time. The guarantee of this property is built-in in the behavior of the coordinator at design time. The coordinator never triggers a track actor to process a new message (aircraft) while the track actor is handling a message (another aircraft), neither when it is stormy. Also, the remaining fuel level of an aircraft is checked as another safety issue of the ATC. If the fuel level becomes less than a threshold, a notification is raised.

Table 1. Performance metrics in the SRP, BAAP, SSP rerouting policies, Combined Rerouting Policies (CRP), and the Runtime Performance Prediction (RPP)

Policy	λ	Throughput	Number of arrived aircraft	Avg. flight duration	Avg. delay in source airport	Number of missing aircraft	Aircraft blocked in the airport
SRP	0.5	0.65	39.60	10.43	14.39	16.94	43.45
	2	0.60	44.73	9.84	4.59	16.28	38.98
	3.5	0.45	65.52	8.95	1.91	10.26	24.21
BAAP	0.5	0.65	38.05	10.28	13.61	18.02	43.92
	2	0.59	43.54	9.84	4.39	17.21	39.23
	3.5	0.45	63.96	8.88	1.92	10.96	25.06
SSP	0.5	0.68	43.84	10.37	16.49	13.86	42.29
	2	0.62	48.38	9.97	5.04	13.04	38.57
	3.5	0.46	66.03	8.94	1.92	8.21	25.74
CRP	0.5	0.65	41.34	10.08	16.06	7.12	51.52
	2	0.60	45.32	9.81	5.39	6.98	47.69
	3.5	0.46	63.23	8.94	2.01	5.39	31.36
RPP	0.5	0.66	44.72	10.72	17.98	13.09	42.18
	2	0.61	47.67	9.99	5.17	13.04	39.28
	3.5	0.46	67.37	8.96	2.11	7.95	24.67

5.3 Performance Evaluation of Dynamic Reroute Planning

The model mentioned in Sect. 5.1 is executed 150 times per each value of λ and the average of the performance metrics are calculated. Table 1 illustrates the results of executing different rerouting policies under the above-mentioned conditions. Setting the number of departing aircraft to a constant value (100), by increasing the value of λ and increasing the total duration time of the simulation, we expect increasing in the number of arrived aircraft. This is because of the fact that increasing the value of λ results in making the traffic lighter, ends in decreasing delays in the source airports and the flight durations.

Table 1 shows the performance metrics for SSP, BAAP, SRP, CRP and RPP. In the CRP policy, the three basic rerouting policies are combined and used dynamically according to a set of predefined policies. The number of missing aircraft in all policies indicates that in our setting the deadlock is inevitable; the reason is that we increase the crossovers of aircraft intentionally to be able to see the differences among different policies. However, designing rerouting policies without a deadlock is not impossible, but it is beyond the scope of this paper. This table shows that the BAAP rerouting policy performs better than SSP and SRP policies based on the metrics of average delay in the source airports and the average flight duration. The SSP has a better performance in terms of the number of missing aircraft in the sub-tracks and the number of blocked

aircraft in the source airports. The cause is that SSP attempts to find the feasible shortest flight route for the longest part of the flight route. However, as SSP is a conservative policy in comparison with the other policies in finding a route, it increases the average delay in the source airport. The reason is that it holds an aircraft in its source airports until it finds a route for the rejected aircraft holding their position in the sub-tracks. The BAAP policy reduces the average flight duration, because if it cannot find a feasible shortest route for a rejected aircraft, it tries to find a route, avoiding to pass the aircraft from the blocked areas. When the aircraft are not allowed to pass from the blocked areas, they rarely need to change their route and also hold the sub-tracks for a fewer time. Consequently the average delay of the aircraft in the source airport decreases.

In our implementation of the CRP, the analyzer continuously monitors the configuration of the model and the planner chooses a suitable rerouting policy to be applied in that configuration. As an example, assume that the coordinator starts rerouting the aircraft using the SRP policy. Upon detecting a blocked area in the airspace, the coordinator switches to BAAP to reduce the flight durations of aircraft by avoiding the blocked area. Moreover, if the number of rejected arrivals to the sub-tracks passes a predefined threshold (that sets to 20 in our experiments), and at least a number of the sub-tracks are occupied (10 in our case), the coordinator will switch to SSP and decreases the airspace load by restricting the number of takeoffs from the source airports. When a predefined percentage of the aircraft arrive to their destinations (50% in our experiments), the coordinator switches to its former policy and the aircraft blocked in the source airport are permitted to takeoff. As shown in Table 1, CRP increases the aircraft delay in ground but decreases the average flight duration and the number of the missing aircraft.

The last row of Table 1, presents the performance evaluation results of rerouting with runtime performance prediction. In our experiments the SSP policy is the default rerouting policy of the coordinator. As shown in Table 1, SSP performs better than BAAP in terms of the number of the missing aircraft but has a higher average flight duration compared to BAAP. If a storm happens, the runtime performance prediction method is applied; if the predicted number of the missing aircraft using SSP exceeds 18 and its average delay in the airport exceeds 13.61, the planner switches its routing policy to BAAP. Furthermore, if the coordinator reroutes the aircraft using the BAAP rerouting policy, the predicted number of the missing aircraft becomes greater than 13.86, and the average delay in an airport becomes greater than 16.49, the coordinator switches to SSP. As a result, the number of the missing aircraft decreases and consequently, the number of the aircraft arrived at the destination airports increases.

6 Related Work

In this section we discuss the most related and recent state of the art in the modeling and analysis of self-adaptive systems. At first, different approaches in modeling feedback control loops are presented and then the applications of formal verification methods in analyzing model@runtime are addressed.

Modeling and analyzing feedback loop: Feedback loops are the most crucial elements of self-adaptive systems which have to be modeled explicitly [27]. The MAPE-K feedback loop is a widely used approach for realizing self-adaptive systems. In order to model the MAPE-K loops, different solutions have been proposed. ActiveFORM is introduced in [15] together with a tool for verifying adaptation goals at runtime as well as the design time. It also provides possibility of changing the system goals at runtime dynamically. ActiveFORM uses timed automata to model each component of the MAPE-K loop and TCTL for specifying properties of the adaptation behavior. To realize self-adaptation, provided models are transformed to codes which are executable on a virtual machine. According to [28], dependencies between the MAPE components and the Knowledge component in ActiveFORM are modeled by signal passing through timed automata; thus, they are only available in an implicit form. Unlike ActiveFORM, our approach does not focus on modeling and verifying MAPE-K loops and does not provide Analyze and Plan components separately. Instead, our focus is on analyzing the models@runtime. Furthermore, encapsulating Analyze and Plan components in a coordinator makes our approach more faithful to the real-world applications in our problem domain comparing to the timed automata.

Vogel et al. [27] presented EUREMA, a model-driven engineering approach to specify and execute multiple feedback loops, runtime models and the dependencies between the feedback loops and the adaptable software. The EUREMA modeling language provides behavioral feedback loop diagram (FLD) to model the feedback loop components and runtime models. It benefits from structural layer diagram (LD) to describe the wiring of FLDs and adaptable software. For executing feedback loops, EUREMA models are interpreted at runtime. Interpretation impacts on the efficiency of the approach but the overhead is negligible. Since our proposed model@runtime is executable, the Analyze activity of the MAPE-K feedback loop is capable of predicting a property violation, while EUREMA has not addressed this feature. EUREMA supports specifying multiple feedback loops which is a part of our future work.

Another approach proposed in [20], introduces Feedback Control Definition Language (FCDL) based on the actor model for modeling adaptable feedback control loops (FCL). In FCL, sensors, actuators, filters, and decision makers are represented by actors. Different FCLs can be organized hierarchically or coordinate with each other. Modeling, structural consistency checking, and code generation using this approach is facilitated by ACTRESS toolkit [21]. ACTRESS transforms FCDL models into Promela models and verifies connectivity and reachability properties of the FCLs using SPIN. As our approach uses the Ptolemy platform, there is no need for more effort to generate executable code, while in ACTRESS an executable application is provided through a set of model-to-model transformation. Furthermore, unlike our approach, there is not model@runtime in ACTRESS.

Multi-agent Abstract State Machine (ASM) is proposed in [4] for modeling and verification of decentralized MAPE-K feedback loops at design time. In this approach, computations of one loop can be distributed over several agents and

the behavior of each MAPE-K loop component is specified by ASM transition rules. In the proposed approach, a system exposes a number of MAPE-K loops, one per each adaptation concern. The model is verified using AsmetaSMV tool with the aim of detecting interferences between different feedback loops. This approach models one MAPE-K loop per each adaptation concern, while in our approach for each sub-system a MAPE-K loop is generated. Furthermore, unlike ASM, our approach does not focus on verifying the MAPE-K feedback loops. Instead it focuses on analyzing the model@runtime.

Model@runtime analysis: Forejt et al. [12] proposed an approach for incremental runtime verification of Markov Decision Process (MDP) models which are described in PRISM. In this approach, runtime changes are limited to vary parameters of PRISM models. The MDP of models are constructed incrementally by inferring a set of states, needed to be rebuilt. The constructed MDPs are verified using the incremental verification technique. Besides, QoSMOS is proposed in [7] for the development of adaptive service-based systems. QoSMOS integrates a set of pre-existing tools to model MAPE-K loop. It adapts system in a predictable way by the aim of optimizing quality of service requirements. Model@runtime is expressed in PRISM and is verified in the analysis step. Results of the verification are used to adapt the configuration of systems. Our coordinated actor model is in a higher level of abstract comparing to the state-based model of PRISM. This reduces the semantic gap between the model and the real-world applications in our problem domain and increases the fidelity.

A parametric approach is presented in [10,11] for the runtime verification of Discrete Time Markov Chains (DTMCs). In this method, probabilities of the transitions are given as variables instead of constant values. Then, the model is analyzed and a set of symbolic expressions is reported as the result. This way, the verification at runtime is reduced to calculating the values of the symbolic expressions by substituting real values of variables.

Designing a self-adaptive software as a dynamic software product line (DSPL) is proposed in [13]. In this approach an instance of DSPL is chosen at runtime considering the environmental changes. This approach separately models common behavior of the products and each variation point by the parametric DTMC. So, there is no need for the verification of each configuration separately. In comparison with both of the above approaches, our actor-based model is more reusable and easy to build comparing to the state-based models such as DTMCs.

Wuttke et al. in [31] worked on the effectiveness of developing frameworks to compare and evaluate different adaptation policies, proposed for self-adaptation. To this aim, the authors developed a Java-based simulator for comparing different adaptation policies in automated traffic routing. In addition to it, Weyns et al. in [29] provided an implementation of the Tele Assistance systems and compared the effect of different adaptation policies in this application.

7 Discussion, Conclusion and Future Work

We proposed coordinated actor models for modeling and analyzing self-adaptive track-based traffic control systems. The coordinator encompasses Analyze and Plan components of the MAPE-K loop as well as managing interactions between the actors. The proposed coordinated actor model is augmented with a runtime performance prediction mechanism. Track-based traffic control systems were targeted, as the widely used application domain in cyber-physical systems. We designed an automated adaptive model for these systems which specialized and implemented for ATCs. We used Ptolemy platform to design and analyze model@runtime. Ptolemy enabled our proposed coordinated actor model to be executed at runtime.

In a nutshell, we can describe the highlights of our approach as follows.

- Fidelity and usability of the model@runtime is built-in because of the actor-based modeling. Compared to the analytical state-based models, actor-based modeling decreases the semantic gap between the real-world and the model. This makes the model more understandable and usable and easy to build for software engineers. The level of abstraction can be tuned, and efficient analysis techniques can be used.
- Concurrency, modularity and reusability are provided as the model@runtime is based on actors.
- Scalability results from simplicity, usability and modularity of the approach. In other words, model@runtime can be the model of a large scale self-adaptive system. However, scalability suffers from the bottleneck produced by the coordinator as the message passing between the actors is governed through the coordinator in a centralized way. In our future work we will address this problem by having multiple coordinators.
- As actors in Ptolemy are implemented in Java, the step for transforming the MAPE-K loop and model@runtime to code is already covered.
- The suggested framework allows defining any behavioral adaptation policies which leads to high flexibility.
- The model leads to a correct by construction design because the coordinator performs safety checks (e.g. to avoid collision), and enforces a set of time constraints on handling the messages by the actors that prevent the aircraft to enter a sub-track (and hence lose the safe distance) simultaneously.

The whole air space is divided into multiple air space regions and is controlled by multiple air traffic controllers. As our future work, we will extend our proposed architecture and model@runtime to capture the decentralized nature of ATC when spread over different air space regions. We will develop multiple coordinated actor models (multiple MAPE-K feedback loops) interacting with each other. Moreover, we will define the formal semantics of the coordinated actor model and extend our previous results [25] to provide formal verification of the model@runtime.

Acknowledgment. The work on this paper has been supported in part by the project "Self-Adaptive Actors: SEADA" (nr. 163205-051) of the Icelandic Research Fund.

References

1. Airport and air traffic control system (1982). https://www.princeton.edu/~ota/disk3/1982/8202/8202.PDF
2. Agha, G.: Actors: A Model of Concurrent Computation in Distributed Systems. MIT Press, Cambridge (1986)
3. Agustn, A., Alonso-Ayuso, A., Escudero, L., Pizarro, C.: On air traffic flow management with rerouting. Part II: stochastic case. Eur. J. Oper. Res. **219**(1), 167–177 (2012)
4. Arcaini, P., Riccobene, E., Scandurra, P.: Modeling and analyzing MAPE-K feedback loops for self-adaptation. In: Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015, pp. 13–23. IEEE Press (2015)
5. Bertsimas, D., Lulli, G., Odoni, A.: The air traffic flow management problem: an integer optimization approach. In: Lodi, A., Panconesi, A., Rinaldi, G. (eds.) IPCO 2008. LNCS, vol. 5035, pp. 34–46. Springer, Heidelberg (2008). doi:10.1007/978-3-540-68891-4_3
6. Bertsimas, D., Patterson, S.S.: The traffic flow management rerouting problem in air traffic control: a dynamic network flow approach. Transp. Sci. **34**(3), 239–255 (2000)
7. Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., Tamburrelli, G.: Dynamic QoS management and optimization in service-based systems. IEEE Trans. Softw. Eng. **37**(3), 387–409 (2011)
8. Calinescu, R., Ghezzi, C., Kwiatkowska, M., Mirandola, R.: Self-adaptive software needs quantitative verification at runtime. Commun. ACM **55**(9), 69–77 (2012)
9. Cheng, B.H.C., et al.: Using models at runtime to address assurance for self-adaptive systems. In: Bencomo, N., France, R., Cheng, B.H.C., Aßmann, U. (eds.) Models@run.time. LNCS, vol. 8378, pp. 101–136. Springer, Cham (2014). doi:10.1007/978-3-319-08915-7_4
10. Filieri, A., Ghezzi, C., Tamburrelli, G.: Run-time efficient probabilistic model checking. In: Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, pp. 341–350. ACM (2011)
11. Filieri, A., Tamburrelli, G.: Probabilistic verification at runtime for self-adaptive systems. In: Cámara, J., de Lemos, R., Ghezzi, C., Lopes, A. (eds.) Assurances for Self-Adaptive Systems. LNCS, vol. 7740, pp. 30–59. Springer, Heidelberg (2013). doi:10.1007/978-3-642-36249-1_2
12. Forejt, V., Kwiatkowska, M., Parker, D., Qu, H., Ujma, M.: Incremental runtime verification of probabilistic systems. In: Qadeer, S., Tasiran, S. (eds.) RV 2012. LNCS, vol. 7687, pp. 314–319. Springer, Heidelberg (2013). doi:10.1007/978-3-642-35632-2_30
13. Ghezzi, C., Molzam Sharifloo, A.: Dealing with non-functional requirements for adaptive systems via dynamic software product-lines. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems II. LNCS, vol. 7475, pp. 191–213. Springer, Heidelberg (2013). doi:10.1007/978-3-642-35813-5_8

14. Hewitt, C.: Description and theoretical analysis (using schemata) of planner: A language for proving theorems and manipulating models in a robot. Technical report, DTIC Document (1972)
15. Iftikhar, M.U., Weyns, D.: ActivFORMS: active formal models for self-adaptation. In: Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2014, pp. 125–134 (2014)
16. International Civil Aviation Organization (ICAO): North atlantic operations and airspace manual (2016)
17. Kephart, J., Chess, D.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003)
18. Khakpour, N., Jalili, S., Talcott, C., Sirjani, M., Mousavi, M.: PobSAM: policy-based managing of actors in self-adaptive systems. Electron. Notes Theor. Comput. Sci. **263**, 129–143 (2010). Proceedings of the 6th International Workshop on Formal Aspects of Component Software (FACS 2009)
19. Khakpour, N., Jalili, S., Talcott, C., Sirjani, M., Mousavi, M.: Formal modeling of evolving self-adaptive systems. Sci. Comput. Program. **78**(1), 3–26 (2012). Special Section: Formal Aspects of Component Software (FACS 2009)
20. Křikava, F., Collet, P., France, R.B.: Actor-based runtime model of adaptable feedback control loops. In: Proceedings of the 7th Workshop on Models@Run.Time, MRT 2012, pp. 39–44. ACM (2012)
21. Křikava, F., Collet, P., France, R.B.: ACTRESS: domain-specific modeling of self-adaptive software architectures. In: Proceedings of the 29th Annual ACM Symposium on Applied Computing, SAC 2014, pp. 391–398. ACM (2014)
22. de Lemos, R., et al.: Software engineering for self-adaptive systems: a second research roadmap. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems II. LNCS, vol. 7475, pp. 1–32. Springer, Heidelberg (2013). doi:10.1007/978-3-642-35813-5_1
23. Ptolemaeus, C.: System Design, Modeling, and Simulation: Using Ptolemy II. Ptolemy. org, Berkeley (2014)
24. Sharifi, Z., Mosaffa, M., Mohammadi, S., Sirjani, M.: Functional and performance analysis of network-on-chips using actor-based modeling and formal verification. ECEASST 66 (2013)
25. Sirjani, M., Jaghoori, M.M.: Ten years of analyzing actors: Rebeca experience. In: Agha, G., Danvy, O., Meseguer, J. (eds.) Formal Modeling: Actors, Open Systems, Biological Systems. LNCS, vol. 7000, pp. 20–56. Springer, Heidelberg (2011). doi:10.1007/978-3-642-24933-4_3
26. Talcott, C.: Composable semantic models for actor theories. Higher-Order Symbolic Comput. **11**(3), 281–343 (1998)
27. Vogel, T., Giese, H.: Model-driven engineering of self-adaptive software with eurema. ACM Trans. Auton. Adapt. Syst. **8**(4), 18:1–18:33 (2014)
28. Wätzoldt, S., Giese, H.: Classifying distributed self-* systems based on runtime models and their coupling. In: Proceedings of the 9th Workshop on Models@ run. time Co-located with 17th International Conference on Model Driven Engineering Languages and Systems, Ceur-WS, pp. 11–20 (2014)
29. Weyns, D., Calinescu, R.: Tele assistance: a self-adaptive service-based system examplar. In: Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015, pp. 88–92. IEEE Press (2015)
30. Weyns, D., Malek, S., Andersson, J.: Forms: unifying reference model for formal specification of distributed self-adaptive systems. ACM Trans. Auton. Adapt. Syst. **7**(1), 8:1–8:61 (2012)

31. Wuttke, J., Brun, Y., Gorla, A., Ramaswamy, J.: Traffic routing for evaluating self-adaptation. In: Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2012, pp. 27–32. IEEE Press (2012)
32. Zhang, J., Goldsby, H.J., Cheng, B.H.: Modular verification of dynamically adaptive systems. In: Proceedings of the 8th ACM International Conference on Aspect-oriented Software Development, AOSD 2009, pp. 161–172 (2009)