Magnifier: A Compositional Analysis Approach for Autonomous Traffic Control

Maryam Bagheri, Marjan Sirjani, Ehsan Khamespanah, Christel Baier, and Ali Movaghar

Abstract—Autonomous traffic control systems are large-scale systems with critical goals. To satisfy expected properties, these systems adapt themselves to possible changes in their environment and in the system itself. The adaptation may result in further changes propagated throughout the system. For each change and its consequent adaptation, assuring the satisfaction of properties of the system at runtime is important. A prominent approach to assure the correct behavior of these systems is verification at runtime, which has strict time and memory limitations. To tackle these limitations, we propose Magnifier, an iterative, incremental, and compositional verification approach that operates on an actor-based model where actors are grouped in components, and components are augmented with a coordinator. The Magnifier idea is zooming on the area (component) affected by a change and verifying the correctness of properties of interest of the system after adapting the component to the change. Magnifier checks if the change is propagating, and if that is the case, then it zooms out to perform adaptation on a larger area to contain the change. The process is iterative and incremental, and considers areas affected by the change one by one. In Magnifier, we use the Coordinated Adaptive Actor model (CoodAA) for traffic control systems. We present a formal semantics for CoodAA as a network of Timed Input-Output Automata (TIOAs), and prove the correctness of our compositional reasoning. We implement our approach in Ptolemy II. The results of our experiments indicate that the proposed approach improves the verification time and the memory consumption compared to the non-compositional approach.

Index Terms—Self-adaptive Systems, Model@Runtime, Compositional Verification, Track-based Traffic Control Systems, Ptolemy II

1 Introduction

Many activities of the modern society are entirely managed by traffic control systems. These systems are large-scale, time and safety-critical systems that consist of numerous moving objects whose movements are adjusted and coordinated by controllers. The application domain of these systems is not only limited to air traffic control systems or rail traffic control systems but also includes robotic systems, maritime transportation, smart hubs, intelligent factory lines, etc. The traffic in such systems can pass through pre-specified tracks, that based on the minimum safe distance between the moving objects, are partitioned into a set of sub-tracks. A system with this structural design is called a Track-based Traffic Control System (TTCS) [1].

Due to the dynamic nature of a TTCS and its surrounding world, a TTCS is vulnerable to failures, threatening human lives or causing intolerable costs. Autonomous response to context changes is a mechanism to prevent a failure in self-adaptive systems that are able to adjust their structures and behaviors in response to changes. The controller in an autonomous TTCS uses the track-based design to safely and efficiently manage the traffic whenever an unpredicted change happens. For each change and its consequent adaptation, verifying the system's safety and quality is necessary, which should be performed at runtime. For performing the analysis and verification at runtime, an abstract model of the system and its environment, the so-called *model@runtime* [2], is generated, updated, and verified during the system execution.

In [3], we introduced the Coordinated Adaptive Actor model (CoodAA) for constructing and analyzing self-adaptive track-based traffic control systems. CoodAA is an actor-based model [4], [5], where actors are grouped in multiple components, and each component has its own coordination policy. In CoodAA, we model each sub-track as an actor, moving objects as messages passed by the actors, and the controller as a coordinator. A TTCS is a large-scale system partitioned into a set of control areas where each area has its own controller, so, a model of a TTCS can be intrinsically built as a set of components and is matched to CoodAA. The moving objects are sent and received at specified times through specified routes.

In this paper, our focus is on the analysis that is performed for adaptation at runtime, and we propose the *Magnifier* idea. Magnifier uses an iterative and incremental process on CoodAA. When a change occurs, Magnifier zooms-in on the affected component and checks if properties of interest still hold. If not, it adapts the component affected by the change by finding a new plan. Then, Magnifier checks if, because of the new plan, the change is *propagated* to other components. If not, the properties of interest are satisfied. Otherwise, Magnifier zooms out and runs another adaptation plan for a larger area. The same process is repeated iteratively and the area under consideration is enlarged incrementally until the change is contained. The idea is to instead of analyzing the whole system for each change, check the effects of the change on the smallest possible area, i.e. the least number of neighborhood components, and try to contain it by adaptation. The general idea of Magnifier is not specific for TTCSs and can be applied

M. Bagheri and A. Movaghar are with the Department of Computer Engineering, Sharif University of Technology, Iran. M. Sirjani is with the School of IDT, Mälardalen University, Sweden, and the School of Computer Science, Reykjavik University, Iceland. E. Khamespanah is with the School of Electrical and Computer Engineering, University of Tehran, Iran, and the School of Computer Science, Reykjavik University. C. Baier is with the department of Computer Science, Technical University of Dresden, Germany.
Manuscript received XX, 2020; revised XX, XXXX.

for any autonomous control system. But in our work, we focus on CoodAA and TTCSs, provide formal semantics and necessary theorems for compositional verification of TTCSs, and illustrate the results by implementing the approach.

In Magnifier, we use a compositional approach, we focus on the interface of each component, which in CoodAA means the inputs and outputs of the component at a specified time from/to a specified source/destination. According to the new plan, if the adapted component generates new outputs or generates outputs by making new assumptions on its inputs, the effects of the change may propagate to the connected components. So, the connected components (or the so-called environment components) are adapted considering the new interface of the component. Then, Magnifier zooms-out and creates a new component by composing all components adapted to the change. The propagation of the change stops if the interface of the new (composite) component remains unchanged.

To prove the correctness of our incremental compositional approach, we present a formal compositional semantics for CoodAA as a network of Timed Input-Output Automata (TIOAs) [6], and adopt the compositional verification theorem of Clark et al. [7]. Each component is represented by TIOAs of its constituent actors and its coordinator. We check the propagation of a change by checking the *compatibility* of TIOAs of the adapted component and TIOAs of its environment components. We call two (or more) TIOAs compatible if they do not reach a deadlock state in their parallel product.

In [7], each component of the model is supplied with a correctness property. By composing a component with an abstraction of its environment components and verifying a property over the composition, the satisfaction of the property over the whole system is proved. Similar to [7], we use abstractions of the environment components. To reduce the state space, instead of TIOAs of the environment components, we only consider TIOAs of border actors that directly communicate with the adapted component. In contrast to [7], we do not use any logical formula to express the properties, since it is enough to check whether the adapted component interacts with its environment as expected (i.e. their compatibility).

Note that the verification of the *propagation* of a change is checking whether the interface of a component remains unchanged after adapting to a new plan. The verification is performed on a static snapshot of the system (the actor-based model@runtime in CoodAA) after each adaptation.

To illustrate the applicability of our approach, we implement it in Ptolemy II [8]. Ptolemy II is an actor-oriented open-source modeling and simulation framework. A Ptolemy model consists of actors that communicate via message passing. Actors are grouped together and coordinated by directors. The semantics of communications of the actors in Ptolemy is defined by different models of computation, implemented in directors. Here, to perform verification in Ptolemy II, we develop a Magnifier director. Our director generates the state space of the affected component, automatically extends its domain to include other components, and performs the reachability analysis over this extended domain. Comparing the compositional and non-compositional approaches, the results of our experiments for an example in the domain of air traffic control systems indicate a significant improvement in the verification time and the memory consumption for Magnifier.

Novelty, importance, and contribution. Our contribution is proposing a compositional, iterative and incremental approach for verification of a component-based actor model for traffic control. We propose an interface theorem for our abstraction and composition technique. Proposing a compositional approach and proving its correctness is not at all trivial. Components may be tightly coupled and dependent on each other, and the problem becomes more serious when we encounter circular dependency. In Magnifier, we take advantage of the structure of TTCSs and the encapsulation and decoupling of actors [9] to build our compositional approach and prove its correctness.

Although compositional verification can be a successful approach for alleviating the state-space explosion, like any other divide-and-conquer techniques, its usage in practice is limited. According to Clarke et al. in [7] and Bensalem et al. in [10], the dependency and specially circular dependency of components can be problematic, and a correct abstraction of interactions is crucial. We also need to be careful with the time alignments in timed systems [11].

An advantage of Magnifier is that it can be seen as a decentralized adaptation mechanism. Adaptation in a decentralized setting is a well-known challenge [12], [13]. It significantly improves scalability and is a suitable option in hard real-time settings, when the reaction to a change should be performed in a negligible amount of time [13]. On the other hand, preserving global goals in a decentralized setting is difficult [13], as several components may need to reach a consensus about an adaptation policy to satisfy a global goal. Magnifier meets the global goals by first applying local adaptation to the component affected by a change. If it is not successful, it dynamically extends its adaptation (and verification) domain to consider more components. The Magnifier approach relies upon the assumption that the environment components of a component are recognisable at the analysis time.

CoodAA is introduced in [3] and its applicability in modeling TTCSs is shown by implementing a case study. In [1], the coordinator is augmented with different rerouting/rescheduling policies, a look-ahead prediction is done for each policy at the time of the change using simulation, and the best policy is selected. We briefly presented the Magnifier idea as a future work in a short work-in-progress paper [14]. In the current paper, we present the formal foundation of CoodAA and Magnifier, and support the idea of effectiveness of Magnifier by an implementation of Magnifier in Ptolemy II and experimental results. The summary of the main contributions is:

- Proposing an approach for compositional, iterative and incremental verification of model@runtime in CoodAA using Magnifier,
- Proposing an abstraction technique for environment components in Magnifier to reduce the state space while preserving the effects of interactions between the components, and proof of correctness of the compositional approach using the interface theorem.

The rest of the paper is organized as follows. We provide

a general overview of TTCSs in Section 2. We recall the definition of a TIOA in Section 3. In Section 4, the formal compositional semantics of CoodAA is described in terms of TIOAs. Section 5 describes the details of the Magnifier approach. The implementation of Magnifier in Ptolemy II and the results of our experiments are shown in Section 6. We describe the related work in Section 7, and conclude the paper in Section 8.

2 Problem Definition and an Example

Track-based Traffic Control Systems (TTCSs) are safety-critical systems. A TTCS works based on the track-based design of the traveling space. To reduce the risk of collision between moving objects, they move on certain tracks instead of moving around freely. Based on the safe distance between two moving objects, each track is divided into a set of sub-tracks. Each sub-track is a critical section that accommodates only one moving object in-transit. A large-scale TTCS is divided into a set of areas, while the traffic of each area is controlled by a centralized controller. Considering congestion and environmental changes, the controller uses the track-based infrastructure of the area to navigate the moving objects safely. As explained in [1], the application domain of TTCSs ranges from Air Traffic Control Systems (ATCs), rail traffic control systems, maritime transportation, to centralized robotic systems and intelligent factory lines. For instance, ATC in the North Atlantic follows a track-based structure that is called an organized track system [15]. The North Atlantic organized track system consists of a set of nearly parallel tracks positioned in light of the prevailing winds to suit the traffic between Europe and North America.

In the real-world applications of TTCSs, each moving object has an initial traveling plan that is generated prior to the departure of the moving object from its source. A traveling plan consists of a route, time schedule decisions, and depending on the application, fuel, etc. The route is a sequence of sub-tracks traveled by the moving object from its source to its destination. The time schedule decisions consist of the departure time of the moving object from its source, assumed arrival time at each sub-track in its route, and assumed arrival time at its destination. TTCSs are sensitive to unforeseen changes in their context. It may need to modify the traveling plans of moving objects when a dynamic environmental change happens. Therefore, following a change in the context, a sequence of changes might happen. For instance, in an ATC, the aircraft flight plans are changed if a storm happens in a part of their flight routes. While changing traveling plans, several safety issues should be considered, i.e., loss of the separation between two moving objects should be avoided and the remaining fuel should be checked. To avoid conflicts, changing the traveling plan of a moving object may result in changing the traveling plans of other moving objects. These changes can be propagated to the whole system. Besides the safety concerns, performance metrics such as arrival times of the moving objects at their destinations or sub-tracks in their routes are important. In a TTCS, the controller is in charge of coordinating the moving objects by rerouting/rescheduling them.

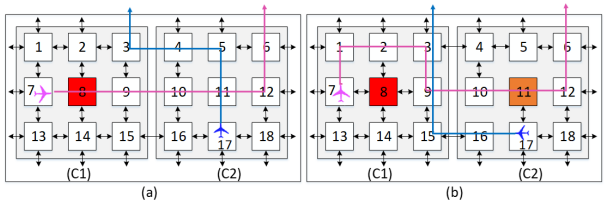


Fig. 1: A TTCS with 18 sub-tracks. The effect of the change in sub-track 8 is propagated to the component c_2. To avoid the collision in sub-track 11, the blue moving object is rerouted.

Example. An example of the change propagation is described for a TTCS as follows. Assume Fig. 1(a) and Fig. 1(b) show a TTCS with two control areas ($C1, C2$), where each area has nine sub-tracks. The traffic flows from the west to the east and vice versa. Each moving object of the eastbound traffic is able to travel towards a sub-track in the north, south, and east. The initial routes of the moving objects are shown in Fig. 1(a). The moving object with an unavailable sub-track in its route is rerouted and its new route is shown in Fig. 1(b). The red sub-track is an unavailable sub-track through which no moving object can travel. For instance, if a storm happens in a part of the airspace in an ATC, the aircraft cannot cross over the sub-tracks affected by the storm and are rerouted. Suppose that the traveling times of the moving objects through each sub-track are the same and are equal to one. The initial traveling planes of the purple and blue moving objects in Fig. 1(a) are $\{(0,7),(1,8),(2,9),(3,10),(4,11),(5,12),(6,6)\}$ and $\{(5,17),(6,11),(7,5),(8,4),(9,3)\}$, respectively. The first entry of each tuple shows the arrival time of the moving object at the sub-track mentioned in the second entry. For instance, two subsequent tuples $(0,7),(1,8)$ mean that the purple moving object arrives at sub-track 7 at time zero and arrives at sub-track 8 at time 1 (which is the same time that it exits sub-track 7).

Suppose that a change happens to sub-track 8 and it becomes unavailable. As a consequence, the traveling plan of the purple moving object is changed to $\{(0,7),(1,1),(2,2),(3,3),(4,9),(5,10),(6,11),(7,12),(8,6)\}$, shown in Fig. 1(b). With the new plan, the purple moving object enters into sub-track 10 (next area) at time 5 instead of 3, and this way the change propagates from $C1$ to $C2$. Now, the purple moving object arrives at sub-track 11 at time 6. At this time, the blue moving object has to enter into sub-track 11 based on its initial traveling plan. To prevent the collision between two moving objects, the controller employs a rerouting algorithm (adaptation policy) and changes the plan of the blue moving object to $\{(5,17),(6,16),(7,15),(8,9),(9,3)\}$. As can be seen, by the occurrence of a change, e.g. a storm, a sequence of changes happens, e.g. rerouting a set of moving objects. This example also shows that the change circulates between two areas. Based on the new traveling plan obtained for the blue moving object, it enters into $C1$ at time 7 instead of 9, and this way the change propagates back to $C1$.

As a change in the context of a TTCS and its consequent adaptations in the system happen at runtime, the satisfaction of properties of interest should be checked at runtime. The properties include: the moving objects have to arrive

at their destinations at the pre-specified times, the collision of the moving objects should be avoided, the fuel of the moving objects should not be less than a threshold, and the system should be deadlock-free. These properties are checked by verification.

3 BACKGROUND

In this section, we present some background of our research. We provide an overview of a TIOA and recall the definition of a deadlock state in a TIOA that is used to define the compatibility of two TIOAs in Section 5. Furthermore, we introduce the coordinated adaptive actor model.

3.1 Timed Input-Output Automata

A timed automaton with a set of input actions and a set of output actions is called a TIOA. Let $\# \in \{\leq, <, =, \geq, >\}$ and $c \in \mathbb{N}$. For a set A, $\mathcal{B}(A)$ denotes the set of conjunctions of constraints of the form $x \# c$ or $x - y \# c$ for $x, y \in A$. A TIOA with integer variables [16] is defined as follows.

Definition 3.1. *(TIOA) A Timed Input-Output Automaton is a tuple* $TA = (Q, q_0, Var, Clk, Act_{in}, Act_{out}, T, I)$ *where* Q *is a finite set of locations,* $q_0 \in Q$ *is the initial location,* Var *is the set of integer variables,* Clk *is a finite set of clocks,* Act_{in} *is a set of input actions,* Act_{out} *is a set of output actions,* $T \in Q \times (\mathcal{B}(Clk) \cup \mathcal{B}(Var)) \times (Act_{in} \cup Act_{out} \cup \{\tau\}) \times 2^{Clk} \times 2^{Ass} \times Q$ *is a set of edges, and* I *is an invariant-assignment function. The set of all variable assignments is denoted by* Ass*. The function* $I : Q \rightarrow \mathcal{B}(Clk)$ *assigns invariants to locations.* □

Based on the above definition, the edge $e = (q, \psi, l, r, u, q') \in T$, besides action l, is labeled with a guard ψ, a sequence u of assignments, and a set r of clocks. Let $v_C, v'_C : Clk \rightarrow \mathbb{R}_{\geq 0}$ and $v_V, v'_V : Var \rightarrow \mathbb{Z}$ be clock and variable valuations, respectively. A state of the system modeled by a TIOA is in the form of (q, v_C, v_V). There is a discrete transition $(q, v_C, v_V) \xrightarrow{l} (q', v'_C, v'_V)$ for an edge $e = (q, \psi, l, r, u, q')$ such that v_C and v_V satisfy ψ, v'_C is reached by resetting the clocks in the set r to zero, and v'_V is obtained as a subset of variables are set to their new values in the assignment set u. The clocks and variables not mentioned in r and u remain unchanged. Furthermore, v'_C satisfies $I(q')$. The TIOA can stay in the location q as long as the invariant $I(q)$ is valid. Let for $x \in Clk$ and $d \in \mathbb{R}_{\geq 0}$, $(v_C + d)(x) = y + d$ iff $v_C(x) = y$. For each delay $d \in \mathbb{R}_{\geq 0}$ there is a timed transition $(q, v_C, v_V) \xrightarrow{d} (q, v_C + d, v_V)$ such that $v_C + d$ satisfies $I(q)$. A state of the system can be a deadlock state that, based on [17], is a state from which no outgoing discrete transition is enabled, even after letting time progress.

Definition 3.2. *(Deadlock State) A state* s *is a deadlock state if there is no delay* $d \in \mathbb{R}_{\geq 0}$ *and action* $l \in (Act_{in} \cup Act_{out} \cup \{\tau\})$ *such that* $s \xrightarrow{d} s' \xrightarrow{l} s''$. □

Let $\mathcal{N} = \{TA_i | i = 1, \cdots, n\}$ denotes a network of TIOAs, where they run in parallel and communicate through global variables. TIOAs synchronize over time and common input and output actions in their parallel composition (product). The detailed definition of the parallel product of TIOAs $TA_1, \cdots, TA_n$, denoted as $TA_1 \otimes \cdots \otimes TA_n$, is presented in [16]. We also recall this definition in our technical report [18]. Based on [16], when two edges of two TIOAs synchronize over an action, their variables are updated by first executing the variable assignments of the output transition, and then by executing the variable assignments of the input transition. Furthermore, the input transitions do not update the shared variables. Note that the state of the system modeled by a network of TIOAs is obtained by clock values, values of all variables, and the locations of all TIOAs in the network.

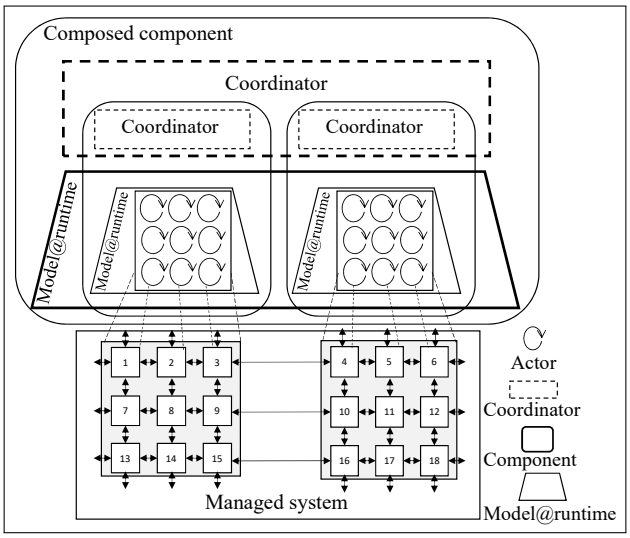


Fig. 2: The CoodAA model. Areas and sub-tracks in the managed system are represented as components and actors, respectively. Magnifier uses the actor model (the model@runtime) for analysis and replanning.

In the rest of the paper, we benefit from the syntax of the UPPAAL modeling language [19] and use functions as macros for expressions in guards and updates in TIOAs.

3.2 Coordinated Adaptive Actor Model

We introduced the coordinated adaptive actor model in [3]. In CoodAA, actors are units of computation, communicating via message passing, and grouped into components. As shown in Fig. 2, each component has its own coordinator with a set of coordination policies. Components themselves can be grouped into composite components. For a precise formal definition of composition refer to [18].

CoodAA is designed to model self-adaptive control systems for track-based systems and conforms to the MAPE-K feedback control loop [20]. The managed system in a self-adaptive system is controlled by the MAPE-K loop consisting of *Monitor*, *Analyze*, *Plan*, and *Execute* activities together with a *Knowledge base*. The model@runtime is an abstract projection of the system and is kept in the *Knowledge base*. The *Monitor* activity monitors the system and updates the model@runtime. In the case of detecting a change, the *Analyze* activity analyzes the model@runtime and passes the analysis results to the *Plan* activity to make an adaptation plan. The adaptation plan is applied to the model@runtime and consequently, the model@runtime is analyzed again. If the requirements of the system are satisfied, the adaptation

plan is sent to the system through the *Execute* activity. Otherwise, the *Plan* activity makes another adaptation plan.

As shown in Fig. 2, areas and sub-tracks in the managed system are represented as components and actors in CoodAA, respectively. Actors of each component construct its model@runtime, and each coordinator consists of the *Analyze* and *Plan* activities of the MAPE-K loop. CoodAA does not contain the *Monitor* and *Execute* activities, and they can be added to the model using the implementation platform [1]. In CoodAA, actors of the components construct the model@runtime of the composite component, and Magnifier acts as the top-most coordinator; it is notified of a change and analyzes the model@runtime to find a safe adaptation plan. Magnifier analyzes iteratively and incrementally using a compositional verification approach.

4 Compositional Semantics of Simplified CoodAA for Magnifier

In this section, we present a formal specification and compositional semantics for a simplified version of CoodAA, where the coordinator and adaptation of actors are ignored. In Magnifier, the analysis is performed on the model@runtime after applying the adaptation policy. So, we only consider parts of the model that have a role in the analysis, i.e., actors (without the adaptation state) grouped into components. The complete semantics of CoodAA, considering the coordinator, its operation, and all states of actors, including the adaptation state, is presented in [18]. In the rest of the paper, we use the terms CoodAA and simplified CoodAA in the context of Magnifier interchangeably.

4.1 Summary of Definitions

In this section, the basic elements of simplified CoodAA, including actors and components are formally described, and the most used notations are presented in TABLE 1. An actor has a variable *status* that shows whether the actor is free or occupied. It also has a variable *plan* to keep the plan that specifies the direction and the time to send out the message. An actor has several input and output ports (modeling several directions that a moving object can arrive at or depart from a sub-track). Input and output ports are communication interfaces of the actor with other actors. An actor also has a message handler.

Definition 4.1. *(Actor) An actor, a_i, with the unique identifier i, is defined as $(status_i, plan_i, interact_i(j, transferredP), P_{I_i}, P_{O_i})$, where $status_i$ has a value of $\{Free, Occupied\}$, $plan_i$ stores the traveling plan of the moving object, $P_{I_i} = \{p_{I_{i,j}} | j = 1, \cdots, inDirections_i\}$ and $P_{O_i} = \{p_{O_{i,j}} | j = 1, \cdots, outDirections_i\}$ are respectively the sets of input and output ports, and $interact_i(j, transferredP)$ is the message handler where $j = 1, \cdots, inDirections_i$ and $transferredP = (objectId, travelPlan)$. The constants $inDirections_i$ and $outDirections_i$ are respectively the numbers of input and output ports.* □

The main computation of the actor is performed in its message handler, which receives a message, reads and updates the *status* and *plan* variables, introduces a delay to model the passage of time, and sends a message over an output port. Two actors are connected if an output port of one actor is bound to an input port of another one. Each output port is connected to at most one input port. The bindings between the ports are defined through the binding set of the component, so, this set defines the topology of the model. Each component has boundary input and output ports through which it communicates with other components. The boundary input and output ports of a component are respectively input and output ports of the constituent actors where these ports are not connected to any ports of the constituent actors and stay lose to be connected to other components. A component is defined as follows.

Definition 4.2. *(Component) A component, C_i, with the unique identifier i, is defined as $C_i = (A_i, B_i, P_{I_{C_i}}, P_{O_{C_i}})$, where A_i is the set of internal actors of C_i, $B_i = \{(p_1, p_2) | \ f : P_O \mapsto P_I \wedge f(p_1) = p_2\}$ is the binding set of C_i, and $P_{I_{C_i}} = \{p \mid p \in P_I \wedge \nexists (p_1, p_2) \in B_i \ \cdot \ p_2 = p\}$ and $P_{O_{C_i}} = \{p \mid p \in P_O \wedge \nexists (p_1, p_2) \in B_i \ \cdot \ p_1 = p\}$ are respectively the sets of boundary input and output ports of the component. The function f is a partial function, $P_O = \bigcup_{a_j \in A_i} P_{O_j}$, and $P_I = \bigcup_{a_j \in A_i} P_{I_j}$.* □

The composition of two (or more) components forms a composite component that is a component itself and is defined as follows.

Definition 4.3. *(Composite Component) A composite component $C_k = (A_k, B_k, P_{I_{C_k}}, P_{O_{C_k}})$ is a component built from composition of other components, where if C_k be the composition of C_i and C_j, denoted by $C_k = C_i \parallel C_j$, and $C_i = (A_i, B_i, P_{I_{C_i}}, P_{O_{C_i}})$ and $C_j = (A_j, B_j, P_{I_{C_j}}, P_{O_{C_j}})$, then $A_k = A_i \cup A_j$ and $B_k = B_i \cup B_j \cup NewB$. The links between boundary output ports of a component and boundary input ports of another component are defined using the binding set $NewB$. So, $P_{I_{C_k}}$ is the set of boundary input ports of C_i and C_j and $P_{O_{C_k}}$ is the set of boundary output ports of C_i and C_j where these boundary ports are not bound to any ports of actors of A_k.* □

The coordinated adaptive actor model is a component composed of all components of the model.

4.2 Compositional Semantics of the Simplified CoodAA Based on TIOAs

In this section, we present the compositional semantics of CoodAA as TIOAs. Each actor is specified by a separate TIOA, and hence, a component is presented in the form of a network of TIOAs of actors. The TIOA of an actor a_j is shown in Fig. 3. We use different colors such as purple, green, light blue, and dark blue to respectively distinguish invariants, guards, synchronization actions, and clock reset and variable assignments in the TIOA. The notations ! and ? mark the output and input actions, respectively.

The automaton of Fig. 3 has two locations, corresponding to the values of the *status* variable of the actor, and a *plan* variable that corresponds to the *plan* variable of the actor in Definition 4.1. Let $b = (p_{out}, p_{in})$ denotes the binding between a port of a_j and a port of another actor. We call b an input binding of a_j if $p_{in} \in P_{I_j}$ and an output binding of a_j if $p_{out} \in P_{O_j}$. The automaton has a set of input actions $interact_b$, where b is an input binding of the actor. These actions correspond to the interact handler of the actor for each input port. Similarly, for each output binding b of a_j,

Notation	Definition
a_i	An actor with the unique identifier i
$status_i$	A variable with a value of $\{Free, Occupied\}$, denoting the status of an actor a_i
$plan_i$	The variable of an actor a_i storing the id and the travel plan (including the route, schedule, fuel, and speed) of the moving object
$interact_i$	The message handler of an actor a_i with the input arguments j (denoting the port) and $transferredP = (objectId, travelPlan)$ (including the id and the travel plan of the moving object)
P_{I_i}	The set of input ports of an actor a_i
P_{O_i}	The set of output ports of an actor a_i
C_i	A component with the unique identifier i
A_i	The set of actors of C_i
B_i	The binding set of C_i
$P_{I_{C_i}}$	The set of boundary input ports of C_i
$P_{O_{C_i}}$	The set of boundary output ports of C_i

TABLE 1: The most used notations for the simplified version of the CoodAA model used for the analysis by Magnifier

an output action $interact_b$ is defined. The automaton has access to the global variable *transferredP* that corresponds to the input argument of the interact handler and is used to transfer a plan between TIOAs of two actors. The TIOA of an actor is defined as follows.

Definition 4.4. *(TIOA of an Actor) The TIOA of an actor a_j is $TA = (Q, q_0, Var, Clk, Act_{in}, Act_{out}, T, I)$, where $Q = \{Free, Occupied\}$, $q_0 = Free$, $Var = \{plan_j\}$, $Clk = \{clock\}$, $Act_{in} = \{interact_b | b \in allInBind(j)\}$, $Act_{out} = \{interact_b | b \in allOutBind(j)\}$, $I(Occupied) = clock \leq travT(plan_j)$, and T is defined as follows.*

$$\forall b \in allInBind(j) : (Free, true, interact_b, \{clock\}, \{plan_j = update(transferredP)\}, Occupied) \quad \text{(Receive)}$$

$$(Occupied, clock = travT(plan_j), interact_{outBind(j,plan_j)}, \emptyset, \{transferredP = plan_j\}, Free) \quad \square \quad \text{(Send)}$$

In the following, we describe edges of TIOA of a_j. An actor a_j is always ready to receive a message over an input port if it is in the state *Free*. If a message is present over the input port $p_{j,l}$, the actor receives the message and the interact handler $interact_j(l, transferredP)$ is triggered. The *Receive* edge (the top edge of Fig. 3) represents this operation. This edge is defined for every input binding of the actor. When the message is received, the *status* variable of the actor is set to *Occupied*. Accordingly, the automaton moves from *Free* to *Occupied*, showing that a moving object enters into the sub-track. The message *transferredP* includes the object id of the message (*objectId*) and the travel plan (*travelPlan*) which consists of the traveling route of the moving object, its schedule, the amount of fuel, and its speed. This information is stored in the *plan* variable of the actor. The actor updates the traveling plan (the route of the moving object) before storing it in *plan*. The route of a moving object is a sequence of sub-tracks traveled by the moving object. When a moving object passes from a sub-track, the first entry in the route, referring to the current sub-track, is removed.

The auxiliary functions used over the *Receive* edge are described as follows. Let AId be the set of all actor identifiers and Msg be the set of all messages in the form

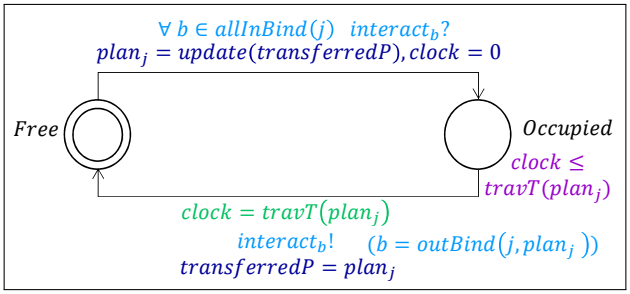


Fig. 3: The TIOA of an actor a_j. The actor receives a message on the top edge. It stays at the location *Occupied* until the time progresses up to a time calculated by *travT* and sends a message on the lower edge.

of (*objectId*, *travelPlan*). The function $allInBind(j)$, where $allInBind : AId \rightarrow 2^{P_O \times P_I}$, returns the set of all input bindings of the actor a_j (P_I and P_O are defined in Definition 4.2). The function $update : Msg \rightarrow Msg$ receives a message and returns a new message in which the traveling plan is updated.

The actor stays in the *Occupied* state for an amount of time derived from the traveling plan stored in the *plan* variable and showing the traveling time of the moving object across the sub-track. This behavior is formulated as the invariant $clock \leq travT(plan_j)$, where $travT : Msg \rightarrow \mathbb{R}_{\geq 0}$ gets an input and outputs the amount of the delay.

After the time passes for the *delay* amount of time, the actor sends out the message $plan_j$ over an output port derived from the traveling plan. The *Send* edge (the lower edge of Fig. 3) represents this operation. The function $outBind(j, plan_j)$, where $outBind : AId \times Msg \rightarrow P_O \times P_I$, returns the binding b including the output port over which $plan_j$ is sent. After sending the message, the *status* variable of the actor is set to *Free*. Accordingly, TIOA moves from *Occupied* to *Free*, showing that the moving object leaves the sub-track. The message is transferred between two actors whenever TIOA of the sender is synchronized with TIOA of the receiver over the $interact_b$ action. The message is delivered to the receiver actor using the *transferredP* variable. The function $allOutBind(j)$ in Definition 4.4, where $allOutBind : AId \rightarrow 2^{P_O \times P_I}$, returns the set of all output

bindings of the actor a_j.

As we mentioned earlier, a component is presented as a network of TIOAs of actors. In the next section, we define the notion of compatibility for the networks of TIOAs of the components to detect the change propagation in the Magnifier approach.

5 Verification of Model@runtime Using Magnifier

In this section, we explain our compositional approach to verify the system in the case of a change occurring and applying adaptation to components. Magnifier is a compositional and iterative approach for adaptation. We focus on the area where the change happens, adapt to the change, and then check whether the change is propagated. If the change is propagated, we enhance the area by composing the areas affected by the change and building a larger area. We explain this in the following section (Section 5.1) in more detail. We use TIOAs of components (representing areas) to perform the analysis of whether the change is propagated or not. If the changed component and its environment components can interact without any problem, the change is contained. In this case, TIOAs of the components are compatible which means TIOAs can be composed without reaching a deadlock state. In Section 5.2, we explain how we abstract the environment components to reduce the cost of compatibility analysis of TIOAs. Section 5.3 presents the formal definition of compatibility of TIOAs and specifies TIOAs of the abstracted environment. Section 5.4 includes the proof of our compositional verification technique based on abstract interface components.

5.1 Compositional and Iterative Approach

When a track-based system is designed, initial traveling plans of the moving objects are selected in a way that no conflict happens between the moving objects, and the moving objects arrive at their destinations at the pre-specified times. In fact, the initial traveling plan of a moving object imposes constraints on its arrival at each area of its route. When a change happens to an area, the moving objects traveling across the area are rerouted if there is an unavailable sub-track in their routes. This way, the plan for the area is adapted. Note that the presence of a change (or its effect) in an area may last for a while, so any change in the plan must consider the possible future effects. When an area is adapted to the change, the validity of properties has to be checked again.

The main required properties of track-based systems include collision avoidance and on-time arrival of moving objects at their destinations. The collision of moving objects is avoided by design; a sub-track can only contain one moving object at a time. On-time arrival at destinations is checked by Magnifier. We also consider a certain amount of fuel for each moving object, and Magnifier checks if the amount of fuel goes under a certain threshold. A special condition is a deadlock condition, and it happens whenever moving objects are stuck in a traffic blockage in an area and cannot find available routes towards their destinations, and hence do not depart from the area. Specific functions are used to detect deadlock and running out of fuel, and the analysis stops if one of those is detected. Under the following conditions, the main property of *on-time arrival and departure* holds for an adapted area, and the change does not propagate:

- Cond. 1. The *departure* of moving objects from the area at the pre-specified *times* and over the pre-specified *ports* is not changed. This condition applies to both moving objects traveling across the area and entering into the area at a future time.
- Cond. 2. The *arrival* of moving objects to the area at the pre-specified *times* in future and over the pre-specified *ports* is not changed.

In the case of violating any of the above conditions, the change is propagated to the adjacent areas. The adaptation for an adjacent area is triggered whenever the change propagates into it. Therefore, all areas affected by the change are composed to form a new composed area. The traveling plans of the moving objects traveling across the new composed area are adapted. If the moving objects arrive and depart at/from this area based on their initial traveling plans, the change propagation stops. This way, Magnifier uses an iterative algorithm to involve the least number of components in the analysis of the correctness of the system.

One can argue that in a compositional approach, we can check the change in one component and then check its propagation to the neighborhood components one by one. But a change may propagate back to the component which was the source of the change and develop a circular dependency. This situation is shown in the example of Section 2. In Magnifier, by composing the components and forming a new larger component, all changes circulating between two components happen inside of the new component and their effects are considered.

5.2 Abstraction and Interface Components

In this section, we explain the abstraction technique and present the definitions which our approach relies on. The notations and the summary of definitions are given in TABLE 2. We model a track-based system as a coordinated adaptive actor model $C_{\mathcal{M}}$ that is a composed component. The component C_i of $C_{\mathcal{M}}$ models an area of the system and interacts with a set of components called **environment components** of C_i. Environment components of a component C_i are those components whose boundary ports are connected to ports of C_i. We use $Env(C_i)$ to denote the set of all environment components of C_i. In Fig. 4(a), a model consisting of five components is shown. The connections between the actors are denoted by arrows. The components C_2, C_3, and C_4 are environment components of C_1, i.e. $Env(C_1) = \{C_2, C_3, C_4\}$. An **environment actor** of a component is an actor whose input and output ports are bound to the input and output ports of the component (and hence directly sends or receives messages to/from the component). The red actors shown in Fig. 4(a) are environment actors of C_1, i.e. a_1 and a_2 are environment actors of C_1 included in C_2.

Abstraction of the environment. In order to abstract the environment of a component, we define **interface components**. The interface components of a component C_i (or

Notation	Definition
$Env(C_i)$	The set of all environment components of C_i
$C_j \downarrow_{C_i}$	The set of augmented environment actors of C_i, where each actor of this set corresponds to an environment actor of C_i in C_j ($C_j \in Env(C_i)$)
$\mathcal{N}_{C_i}$	The network of TIOAs of the actors of the component C_i
$TA_1 \otimes \cdots \otimes TA_n$	The parallel product of TIOAs $TA_1, \cdots, TA_n$. TIOAs are *compatible* if their parallel product does not reach a deadlock state.

TABLE 2: The most used notations for the abstraction of the environment and the compatibility definition in Magnifier

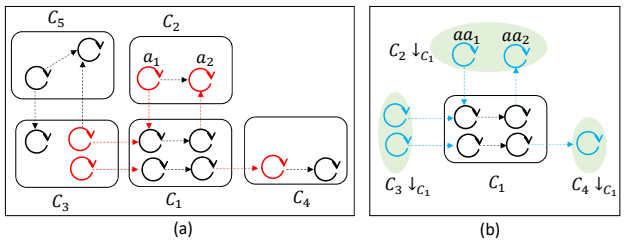


Fig. 4: A model consisting of 5 components is shown in (a). The connections between actors are shown with dashed arrows. The red actors are environment actors of the component C_1. The interface components of C_1, i.e. $C_j \downarrow_{C_1}$, $j = 2, 3, 4$, are shown in (b). The augmented environment actors of C_1 along with their ports are shown in blue.

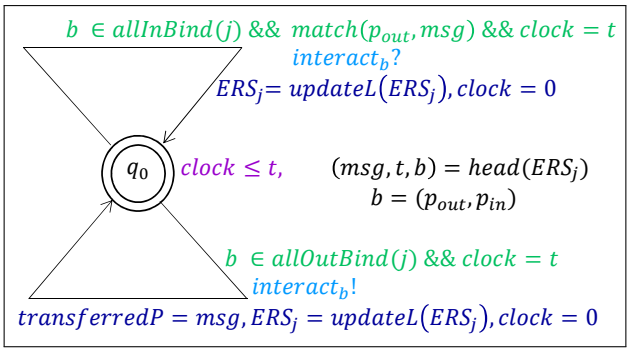


Fig. 5: The TIOA of an augmented environment actor aa_j, which has a list ERS of messages that have to be sent or received by the actor at the pre-specified times and over the pre-specified ports. aa_j receives an expected message on the top edge and sends a message on the lower edge.

visible parts of its environment) are defined based on the sets of environment actors of C_i. For each environment actor, we define an **augmented environment actor**. Let C_j be an environment component of C_i. We use $C_j \downarrow_{C_i}$ to denote the set of augmented environment actors of C_i where each actor of this set corresponds to an environment actor of C_i included in C_j. The augmented environment actors in $C_j \downarrow_{C_i}$ are augmented with all the significant information of the environment component C_j which can affect C_i.

An augmented environment actor has a list, called *ERS*, containing the crucial data related to conditions Cond. 1 and Cond. 2 in Section 5.1. Each entry of *ERS* contains a message, a delay value, and a pair of output and input ports (a binding). Besides, this actor has an init method, and similar to Definition 4.1 for actors, has a set of input ports, a set of output ports, and an interact handler. The input and output ports of the augmented environment actor are only those ports of its corresponding environment actor that are connected to the component C_i. The augmented environment actors of C_1 along with their ports are shown in blue in Fig. 4(b), i.e. $C_2 \downarrow_{C_1} = \{aa_1, aa_2\}$ where aa_1 and aa_2 correspond to the actors a_1 and a_2 in Fig. 4(a), respectively. The detailed formal definition of augmented environment actors is available in our technical report [18].

Definition 5.1. *(Interface Component) For each $C_j \in Env(C_i)$, $C_j \downarrow_{C_i}$ is called an interface component of the component C_i.* □

The definition of the interface component is inspired from the approach of Clarke et al. in [7], where interface *processes* are defined. For two processes P_1 and P_2, $P_1 \downarrow \Sigma_{P_2}$ is an interface process of P_2, where Σ_{P_2} is the set of symbols (i.e. atomic propositions) associated with P_2. The interface process $P_1 \downarrow \Sigma_{P_2}$ is the process P_1 in which all symbols that do not belong to Σ_{P_2} are hidden.

5.3 Semantics of Interface Components

Here, we present the semantics of an interface component. Each augmented environment actor is specified by a separate TIOA, and hence, an interface component is presented as a network of TIOAs of the augmented environment actors. To check the change propagation in Magnifier, we check whether the network of TIOAs of the changed component and the networks of TIOAs of its interface components are compatible. We call two networks of TIOAs compatible if all TIOAs in these networks are compatible. A set of networks of TIOAs are compatible if they are pairwise compatible. Here we define compatible TIOAs.

Definition 5.2. *(Compatible TIOAs) Two or more TIOAs are compatible if the parallel product of them does not reach a deadlock state.* □

Our definition of compatibility is inspired from the approach of [21], in which two components (timed interfaces) are compatible if there is an environment to avoid the parallel product of the components from reaching an error state (the environment makes the components work together). In our approach, we do not consider any helpful environment to check the compatibility. Note that a deadlock state in the product of two TIOAs is different from a deadlock in a track-based system.

The TIOA of an augmented environment actor, which is used in the model@runtime, is shown in Fig. 5. This automaton has an *ERS* variable that corresponds to the *ERS* list of the actor. *ERS* is an ordered list. Each entry of this

list contains a message, a delay value, and a binding, and specifies that the augmented environment actor expects to receive a message over an input port or intends to send a message over an output port after a certain amount of time. Let t and t' be the delay values kept in the first and the second entry of *ERS*, respectively. The message of the first entry is sent or received at time t. The augmented environment actor will send or receive the message of the second entry at time $t + t'$. The same argument is valid for the rest of the entries, and the delay value in none of the entries is zero. As *ERS* in a model of a track-based system is calculated from the initial traveling plans of the moving objects, the schedules of the moving objects in *ERS* do not lead to any conflicts between the moving objects.

Besides *ERS*, the automaton has a set of actions $interact_b$. These actions correspond to the interact handler of the actors and can be an input action (for the input binding b) or an output action (for the output binding b). The automaton has access to the global variable *transferredP* that corresponds to the input argument of the interact handler and is used to transfer a value between TIOAs of an actor of CoodAA and the augmented environment actor.

Definition 5.3. *(TIOA of an Augmented Environment Actor) The TIOA of an augmented environment actor aa_j is $TA = (\{q_0\}, q_0, Var, \{clock\}, Act_{in}, Act_{out}, T, I)$, where $Var = \{ERS_j\}$, $Act_{in} = \{interact_b | b \in allInBind(j)\}$, and $Act_{out} = \{interact_b | b \in allOutBind(j)\}$. Let (msg, t, b), where $b = (p_{out}, p_{in})$, denotes the first entry of ERS_j in the location q_0. Then, $I(q_0) = clock \leq t$, and T is defined as follows.*

$$(q_0, b \in allInBind(j) \wedge match(p_{out}, msg) \wedge clock = t, \\ interact_b, \{clock\}, \{ERS_j = updateL(ERS_j)\}, q_0) \quad \text{(Receive)}$$

$$(q_0, b \in allOutBind(j) \wedge clock = t, interact_b, \{clock\}, \\ \{transferredP = msg, ERS_j = updateL(ERS_j)\}, q_0) \quad \square \quad \text{(Send)}$$

Herein, we describe edges of TIOA of aa_j. To have simple expressions in Fig. 5, we access the first entry of ERS_j using $head(ERS_j)$. The actor aa_j looks at $(msg, t, b) = head(ERS_j)$ in the location q_0, where $b = (p_{out}, p_{in})$, and stays at this location until the time progresses up to t at which aa_j sends or receives the message *msg*. The automaton synchronizes on the action $interact_b$ to send *msg* over the output port p_{out} or receive it over the input port p_{in}.

As shown on the top edge of Fig. 5 (the *Receive* edge), if b is an input binding, the clock equals t, and the message *msg* derived from the head of ERS_j matches the expected message (checked by the function $match(p_{out}, msg)$), then the actor aa_j pops the current entry of ERS_j by calling the function *updateL*, resets the clock to zero and goes back to q_0. The function *allInBind(j)*, defined in Section 4.2, returns all input bindings of aa_j. The function $match : P_O \times Msg \rightarrow Boolean$ checks whether a given message (*msg*) matches the message ready to be sent from the sender actor over a given output port (p_{out}), where P_O is the set of all output ports of all actors. The function $updateL : 2^{En} \rightarrow 2^{En}$ removes the first entry of a given list and returns the rest of the list, where *En* is the set of all entries of all *ERS* lists.

As shown on the lower edge of Fig. 5 (the *Send* edge), if b is an output binding and the clock equals t, then the actor sends out the message *msg* by putting the message in the global variable *transferredP*, pops the current entry of ERS_j by calling the function $updateL$, resets the clock to zero, and goes back to q_0. The function *allOutBind(j)*, defined in Section 4.2, returns all output bindings of aa_j.

In Magnifier, if there is a deadlock in the product of TIOAs of the component C_i and the interface component $C_j \downarrow_{C_i}$, the change propagates from the component C_i to the component $C_j \in Env(C_i)$. This means that there is an augmented environment actor in $C_j \downarrow_{C_i}$ that is not able to either send a message or receive a message over a pre-determined port at a pre-specified time, and hence no transition is performed in the location q_0 of the actor (see Definition 5.3).

In the rest of the section, we use $\mathcal{N}_{C_i}$ and $\mathcal{N}_{C_j \downarrow_{C_i}}$ to respectively denote the network of TIOAs of the component C_i and the network of TIOAs of the interface component $C_j \downarrow_{C_i}$, such that all TIOAs in each network are compatible.

For a better understanding of the Magnifier approach, consider the following example.

Example. Suppose that a change in the component C_1 of Fig. 4(a) is detected and this component is adapted. If after adaptation, $\mathcal{N}_{C_1}$ and one or more of the networks $\mathcal{N}_{C_2 \downarrow_{C_1}}$, $\mathcal{N}_{C_3 \downarrow_{C_1}}$, and $\mathcal{N}_{C_4 \downarrow_{C_1}}$ are not compatible, the change propagates into one or more of the components C_2, C_3, and C_4. Let's assume that the change propagates to the components C_2 and C_3. It means that C_1 with its current adaptation is not able to either receive messages from C_2 and C_3 or send messages to those components at the pre-specified times and over the pre-specified ports. Consequently, the adaptation for C_2 and C_3 is triggered. The components C_1, C_2, and C_3 are adapted and are composed to provide the new component $C_{1,2,3}$. If $\mathcal{N}_{C_5 \downarrow_{C_3}}$, $\mathcal{N}_{C_4 \downarrow_{C_1}}$, and $\mathcal{N}_{C_{1,2,3}}$ are compatible, the change propagation stops, and the change is not propagated further than C_1, C_2, and C_3.

5.4 Correctness of the Compositional Reasoning

In the previous section, we defined the interface components of a component. An interface component is an abstraction of an environment component. In the absence of a change, the correctness properties of the system are preserved as each component of the system preserves a set of local correctness properties, i.e. each component receives and sends messages at the pre-specified times and over the pre-specified ports. In the presence of a change, the correctness properties are satisfied if the adapted component can work with its environment, i.e. the adapted component and its environment satisfy their input and output assumptions. This is where the networks of TIOAs of the adapted component and its interface components are compatible. In this section, the correctness of the proposed approach is proved in Theorem 5.1, explaining that reducing the environment components to the interface components is correct.

Theorem 5.1. *(Interface Theorem) The networks of TIOAs of the adapted component and its interface components are compatible if and only if the networks of TIOAs of the adapted component and its environment components are compatible.*

Proof. "if": By contradiction. Let the networks of TIOAs of the adapted component C_i and its environment components be compatible, but the networks of TIOAs of C_i and its interface components are not compatible, i.e. $\mathcal{N}_{C_i}, \mathcal{N}_{C_{j_1}\downarrow_{C_i}}, \cdots, \mathcal{N}_{C_{j_n}\downarrow_{C_i}}$, where $C_{j_k} \in Env(C_i)$, $k = 1, \cdots, n$, and $n = |Env(C_i)|$, are not compatible. It means that there exists an interface component $C_{j_l} \downarrow_{C_i}, C_{j_l} \in Env(C_i)$, and an augmented environment actor $aa_j \in C_{j_l}\downarrow_{C_i}$ such that this actor is not able to either receive an expected message from an expected port at a pre-specified time or send a message over a pre-specified port at a pre-specified time. Let aa_j corresponds to the environment actor a_j. However, the actor a_j belongs to C_{j_l}, e.g. $a_j \in A_{j_l}$. This means that $\mathcal{N}_{C_i}, \mathcal{N}_{C_{j_1}}, \cdots, \mathcal{N}_{C_{j_n}}$ are not compatible, which contradicts the assumption.

"only if": By contradiction. Let the networks of TIOAs of the adapted component C_i and its interface components be compatible, but the networks of TIOAs of C_i and its environment components are not compatible, i.e. $\mathcal{N}_{C_i}, \mathcal{N}_{C_{j_1}}, \cdots, \mathcal{N}_{C_{j_n}}$, where $C_{j_k} \in Env(C_i)$, $k = 1, \cdots, n$, and $n = |Env(C_i)|$, are not compatible. We assumed that the adaptation results in a new network of compatible TIOAs for the component C_i. Furthermore, as each component $C_j \in Env(C_i)$ is not yet affected by a change, all TIOAs in $\mathcal{N}_{C_j}$ are compatible. Therefore, there exists $C_j \in Env(C_i)$ and an environment actor $a_j \in A_j$ such that this actor is not able to either receive an expected message from an expected port at a pre-specified time or send a message over a pre-specified port at a pre-specified time. However, this actor corresponds to an actor of $C_j \downarrow_{C_i}$. This means that $\mathcal{N}_{C_i}, \mathcal{N}_{C_{j_1}\downarrow_{C_i}}, \cdots, \mathcal{N}_{C_{j_n}\downarrow_{C_i}}$ are not compatible, which contradicts the assumption. □

Focusing on the TIOA model of CoodAA, note that each TIOA interacts with a fixed set of TIOAs and over a fixed set of input and output actions. An specific feature of the TIOA modeling an actor in CoodAA is that the actions shared between two TIOAs are not shared with a third TIOA. Moreover, there are no shared variables among TIOAs except for the one used to represent the characteristics of communication, e.g., *transferredP* in Definition 4.4.

6 Evaluating Magnifier

We compared CoodAA and our approach to perform the analysis at runtime with similar approaches in [1], [3]. We include a subsection on this comparison in Section 7. The focus of this paper is on comparing the non-compositional verification approach and our compositional approach. To this end, we compare the time and memory consumption of both approaches in this section. We implement an ATC case study with several control areas in Ptolemy II [8] as a proof of concept for effectiveness and efficiency of the compositional approach. We decided to use Ptolemy II because the framework enables us to automate the iterative and incremental process of Magnifier using its so-called *director*. The change propagated through the system can be automatically traced, and the verification scope can be extended to bring more components into the analysis incrementally. We used TIOA for defining the compositional semantics of CoodAA, as compositionality can be easily represented using automata, but we did not find UPPAAL the best tool for implementing the rather complicated iterative and incremental approach of Magnifier whereas Ptolemy II is a Java-based tool giving us the necessary programming power.

In [1], we developed a Ptolemy template for CoodAA to model and analyze self-adaptive TTCSs. In this template, we modeled each sub-track as a Ptolemy actor and the moving objects as messages passed by the actors. The pathways between the sub-tracks are modeled by interconnections between the actors. Furthermore, we modeled the controller (coordinator) as a Ptolemy director. In this paper, we use this template and extend its director to develop the Magnifier director that supports formal verification.

In each iteration, after replanning, Magnifier checks the compatibility of components. This is done by generating the state space of a given component, and checking if everything is performed according to the plan (satisfying Cond. 1 and Cond. 2 in Section 5.1) using reachability analysis. For the sake of simplicity, we assume that all coordinators of all components (the ATC controllers of all areas) have the same adaptation policy (rerouting algorithm). This way, we have only one coordinator (instead of a nested model and multiple coordinators). The Magnifier director generates the state space of the model of an ATC example with several components, where the components are composed to create a new component. The rerouting algorithm and the algorithm to generate the state space are implemented in the director. It is notable that designing the rerouting algorithm is not the concern of this paper. The details of the implementation of Magnifier in Ptolemy II and the pseudo-code of the algorithm to generate the state space are available in [18]. The Ptolemy model and implementations of the provided algorithms in the next section are also available[1].

We perform our experiments for different settings including different sizes of the traffic network, different times for occurring the change, different parameters of the exponential distribution to generate the departure times of aircraft from their sources (different traffic volumes), and different numbers of aircraft. The results denote that Magnifier significantly decreases the usage of time and memory.

6.1 Experimental Setting

To compare the compositional and non-compositional approaches, we focused on an ATC example with a $n \times n$ mesh map. We also considered $2n - 1$ source airports and $2n - 1$ destination airports (each one of the source and destination airports is connected to a sub-track). We developed an algorithm to generate the initial flight plans of m aircraft and an algorithm (an adaptation policy) to reroute the aircraft as follows. We described the algorithms in detail in our technical report [18].

ALG1: Generating the initial plans. This algorithm randomly generates the source, the destination, and a departure time from the source airport for each aircraft. Similar to the XY routing algorithm [22], ALG1 finds a route from the source to the destination by first traversing the X dimension and then traversing the Y dimension of the mesh. It finds time conflict-free routes, where two or more aircraft are not allowed to travel across a sub-track at the same time. ALG1

1. https://github.com/maryambagheri1989/Magnifier

does not guarantee to find the most efficient (e.g., shortest) route.

ALG2: Rerouting algorithm. This algorithm gradually substitutes a part of the initial route with a new sub-route, such that the resulting route has the same length as the initial route. If no such route is found, ALG2 finds a route ignoring the length of the initial route. If using both above approaches no route is found, the aircraft will stay one more unit of time in its current location. It then will fly based on its initial route if the first sub-track in its route becomes available. Otherwise, the procedure of ALG2 repeats. ALG2 uses the same procedure as ALG1 to find a route. It avoids the stormy track but does not check the time conflict with other aircraft in the future. If a potential conflict is detected, we will take care of it by rerouting the aircraft upon detecting the conflict.

Scenarios. Different parameters such as the rerouting algorithm, the time of the storm, the place of the storm, the network traffic volume, the amount of concurrency arisen from flight plans of the aircraft, and the network dimension change the results of experiments. We perform three sets of experiments; (ES1) that is to compare the time and memory consumptions between the compositional and non-compositional approaches, (ES2) that is to depict the variation of the time consumption in a set of experiments for each approach, and (ES3) that is to compare the scalability of the approaches. The scenarios are described in the following. In our experiments, we assume that the traveling time of an aircraft across a sub-track is one. We also assume that the aircraft consumes one unit of fuel per one unit of the traveling time. Furthermore, the fuel of each aircraft is more than the length of the longest path in the traveling network. For the place of the storm, we select the middlemost sub-track of the network.

(ES1). We consider a 15×15 mesh structure, divided into 9 regions of 5×5, as the traffic networks in (ES1). The fuel of each aircraft is set to 325. We use ALG1 to generate 150 batches of flight plans per each λ in $\{0.5, 0.25, 0.125\}$, where λ is the parameter of the exponential distribution to generate departure times of the aircraft from source airports. By increasing the value of λ, the mean interval time between two departures decreases. As a result, the network traffic volume and subsequently the concurrency contained in the model might increase. We expect that the compositional approach performs better than the non-compositional approach even in a low concurrency model. Each generated batch contains flight plans of 2000 aircraft. Per each batch $P_i, 1 \leq i \leq 150$, we generate 4 batches $P_{ij}, 1 \leq j \leq 4$, such that P_{i1} contains the first 500 flight plans of P_i, P_{i2} contains the first 1000 flight plans of P_i, and so on. We use both approaches to analyze each batch P_{ij} per each time of the storm in $\{100, 200, 400, 600, 800\}$. Obviously, whenever the storm occurs late, most of the moving objects have arrived at their destinations. We remove the batch P_i from the experiments of both approaches if for a batch P_{ij} and a time of the storm, the model in one of the approaches is not deadlock-free, or its verification time passes the threshold (the results of experiments in which the models are not deadlock-free are investigated in (ES2)). Table. 3 shows the number of experiments in which the model in both approaches runs to completion within the time limit (Complete), faces a deadlock (Deadlock), and verification does not reach a result within the time limit (TimeOver). The threshold of the analysis time is set to an hour. In our experiments, per each j, we calculate the averages of the analysis time and the number of states of the batches P_{ij}.

TABLE 3: The number of experiments in which the model in both approaches runs to completion within the time limit (Complete), faces a deadlock (Deadlock), does not reach a result within the time limit (TimeOver). The traffic network has a $n \times n$ mesh structure. λ is the parameter of the exponential distribution to generate the departure times of the aircraft.

n	λ	Complete	Deadlock	TimeOver
15	0.5	120	27	3
15	0.25	110	37	3
15	0.125	94	56	0

(ES2). The traffic network in (ES2) has the same configuration as the traffic network in (ES1). In (ES2), we use the batches of flight plans generated in (ES1) for $\lambda = 0.5$. The reason for considering $\lambda = 0.5$ is that the network might have the highest traffic volume for $\lambda = 0.5$ compared to $\lambda \in \{0.25, 0.125\}$. We use both approaches to analyze each batch P_i, containing the flight plans of 2000 aircraft. Since the possibility of propagating the change increases when the storm happens early, we suppose that the storm happens at time 100. As shown in Table. 3, the model in 120 experiments is deadlock-free and is analyzed in less than the predefined threshold. Compared to (ES1) that calculates the average of the analysis time for this set of experiments, (ES2) illustrates the variation of the analysis time in this set for each approach. Furthermore, (ES2) depicts the variation of the time consumption to detect a deadlock in 27 experiments that are not deadlock-free.

(ES3). As the aim in (ES3) is to compare the scalability of those two approaches, we consider a larger traffic network that is a 18×18 mesh structure with 9 regions of 6×6 in our experiments. The fuel of each aircraft is set to 425. We assume that the change happens at time 100. We use ALG1 to generate a batch P of 5500 flight planes with $\lambda = 0.5$. In (ES3), we start with the first 100 flight plans of P, and gradually increase the number of flight plans to compare the scalability of the two approaches. The scalability of the approaches is measured by the number of the aircraft. We define a threshold for the verification time and set this threshold to 45 minutes. The approach that can analyze a model with more number of the aircraft in less than the defined threshold is more scalable.

6.2 Comparison of the Magnifier Approach and the Non-compositional Approach

We run our experiments on an ubuntu 18.04 LTS amd64 machine with 67G memory and Intel (R) Xeon (R) CPU E5-2690 v2 @ 3.00GHZ. A part of our experimental results is shown in Figures 6, 7, 8 and 9. In these figures, "C" and "NC" refer to the compositional and non-compositional approaches, respectively. The legend entry $C - i, i \in \{100, 200, 400, 600, 800\}$ depicts the experimental results of the compositional approach for the time i at which

the storm happens. The legend entry $NC - i$ depicts the results for the non-compositional approach. As shown in Fig. 6 and Fig. 7, using the compositional approach results in decreasing the verification time and the number of states. As expected, by increasing the number of aircraft, the number of states, and accordingly, the verification time increase. The same results are valid for the smaller value of the time at which the storm occurs, since fewer aircraft have arrived at their destinations when the storm happens. By increasing the time at which the storm occurs, the differences between the results of the compositional and non-compositional approaches decrease. It is because most of the aircraft have arrived at their destinations when the storm happens late. To have a better representation of the verification time difference between the compositional and non-compositional approaches, we depict the results of the verification time for $\lambda = 0.5$ in two diagrams with two different time scales, shown in Fig. 7. As can be seen, the compositional approach is able to verify a model with 2000 aircraft in a few seconds for the smallest value of the time at which the storm happens. By increasing the time interval between two departures from a source airport, the number of aircraft entering into the traffic network after the storm happens increases. Therefore, as shown in Fig. 6, the number of states in the compositional approach increases whenever the value of λ decreases.

The results of our experiments in (ES2) are shown in Fig. 8. For the compositional approach, the variation of the verification time in a set of experiments with no deadlock is shown in Fig. 8(a). The results of the same set of experiments for the case in which the non-compositional approach is used are depicted in Fig. 8(b). We also depict the variation of the time needed to detect a deadlock in a set of experiments using the compositional and non-compositional approaches in Fig. 8(c). As shown in Fig. 8(a), excluding the outliers, the model in our experiments is analyzed in less than 22 seconds using the compositional approach, while this time is around 2190 seconds in the non-compositional approach. Also, in our experiments, the average time for detecting a deadlock in the compositional approach is around 11 minutes, while this value in the non-compositional approach is around 20 minutes.

To detect a deadlock situation, we set up a timeout mechanism. In ALG2, the aircraft may circulate between a few sub-tracks trying to find a way out. We can only detect this situation using a timeout. This is not the best way to detect the deadlock; we set a large timeout value, and it causes an increase in the deadlock detection time. So, we exclude these scenarios from reported experiments in (ES1).

We can define a time limit for running the adaptation algorithm, and when we reach the limit, human intervention will take place. For example, if we consider a time limit of three minutes, from 150 experiments, only 21 experiments need human intervention in the compositional approach. These 21 experiments include the three experiments in which the verification time passes the threshold, the outlier experiment in Fig. 8(a), and 17 experiments (of 27 experiments) that face a deadlock. All experiments in the non-compositional approach need more than three minutes analysis time.

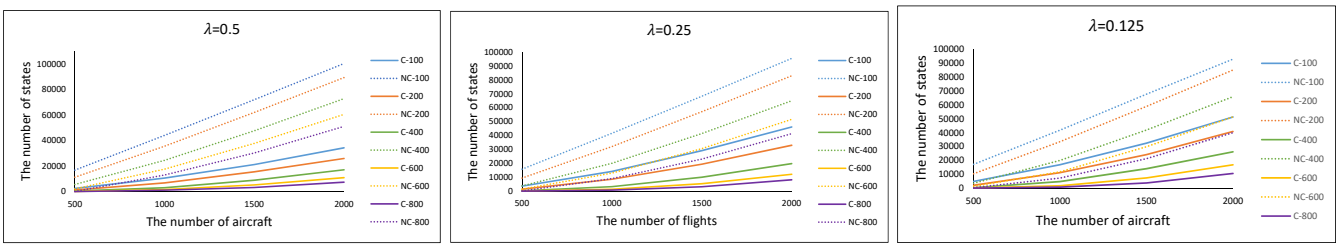
The results of our experiment in (ES3) are shown in Fig. 9. To compare the scalability of both approaches, we run both approaches for the same scenario, and we define a threshold of 45 minutes for the verification time. The non-compositional approach does not scale for more than 2800 aircraft. The results of the compositional approach in Fig. 9 have fluctuations appeared between 4600 to 5000 aircraft. By adding new aircraft to the traffic network, some areas are congested, and consequently, the concurrency of the model increases. This results in some fluctuations and the fast growth of the "C" plot between 4600 to 5000 aircraft. Except for this range, this plot has a normal growth, since by adding the new aircraft, the behaviors of the congested areas have not sensibly changed.

6.3 Discussion and Threats to Validity

We discuss an observation regarding a comparison between the non-compositional approach and the worst case for Magnifier when the change propagates all the way to include the whole system. We argue that even in the worst case, Magnifier performs better than the non-compositional approach. Our experiments testify this argument. This observation can be justified as follows. Suppose that a change happens at time t. The non-compositional approach involves all actors of the model that have a message at time t into the analysis, and starts to generate the state space. In contrast, Magnifier focuses on the component affected by the change, and starts to generate the state space by involving those actors of the component having a message at time t. Let the branching factor for a state be the number of outgoing transitions of the state or the number of actors that can be triggered at the state. At the beginning, Magnifier has a lower branching factor on all states of the state space compared to the non-compositional approach. At some point in future, e.g. at time t', when all components are affected by the change, both approaches involve the same number of actors into the analysis, and both approaches generates the same number of states and transitions. However, between t and t', the graph of the state space in Magnifier is smaller than the graph of the state space in the non-compositional approach. Therefore, even in the worst case, Magnifier performs better than the non-compositional approach in terms of the verification time and the memory consumption.

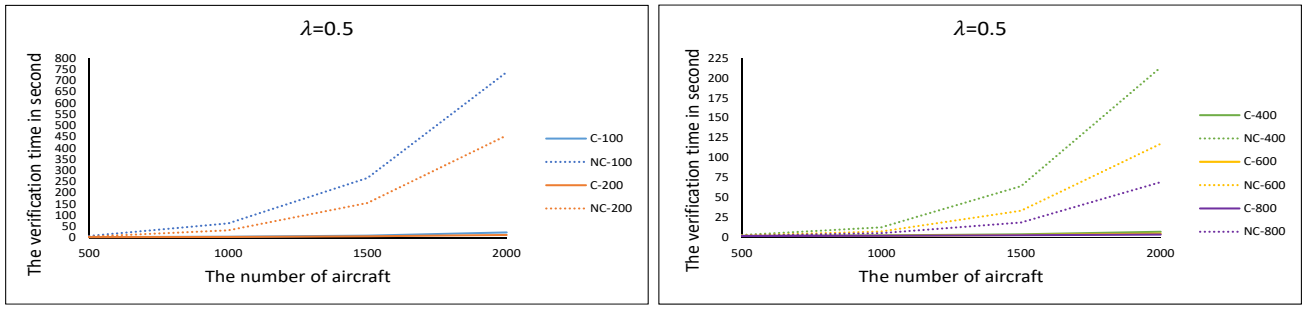
It is also possible that the rerouting algorithm contains the change in a small area of the network. Regarding this case, we considered the same termination condition for both approaches to have a fair comparison. Same as Magnifier, the non-compositional approach follows the change propagation and terminates whenever the propagation stops. As discussed, Magnifier incrementally adds components into the analysis and hence performs better.

The main threat to validity lies in possible implementation errors. To reduce this threat, we checked the propagation of the change and reviewed the state space of various small to large batches of flight plans to validate the operation of both approaches. Another threat concerns the traffic volume and the size of the traffic network that affect the results. To reduce this threat, we considered different λ to randomly generate departure times from source airports, different times for happening the storm, and different size for the traffic network.



(a) (b) (c)

Fig. 6: The number of states in (ES1) for each value of λ in $\{0.5, 0.25, 0.125\}$, where λ is the parameter of the exponential distribution to generate the departure times of the aircraft. The notations C and NC refer to the compositional and non-compositional approaches, respectively. The time at which the storm happens varies in the set $\{100, 200, 400, 600, 800\}$. As an instance, $C - 100$ depicts the results of the compsitional approach when a storm occurs at time 100.



(a) (b)

Fig. 7: The verification time in (ES1) for $\lambda = 0.5$. The left side depicts the verification time in the compositional (C) and non-compositional (NC) approaches when a storm occurs at a time in $\{100, 200\}$. The right side depicts the verification time of each approach when a storm occurs at a time in $\{400, 600, 800\}$. The right and the left side figures show the verification time with different scales.

7 Related Work

In this section, we concentrate on four classes of most related studies, modeling and verifying traffic control systems, formal analysis of self-adaptive systems at runtime, compositional methods for verification, and interface theory.

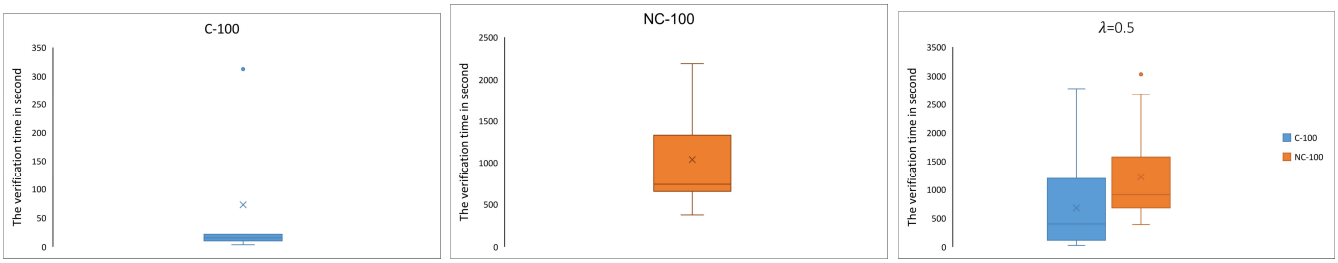
Modeling and Verifying Traffic Control Systems (TCSs). TCSs such as ATC and train control systems, due to the tight interconnection of the physical plant and the controller software, are mostly categorized as hybrid systems. There is a vast literature on verifying dynamic models of TCSs to detect the future conflicts among the moving objects [23], [24], [25], to resolve the potential conflicts through the trajectory planning [26], [27], and to evaluate the correctness of the communication protocols among different entities of the system [28], [29], [30]. These approaches use the *Lagrangian* models in which the moving objects along with their operational details are the concern of modeling [31], [32]. Modeling the dynamic behaviors of each moving object in these approaches needs a set of differential equations, which due to a large number of the moving objects, makes the analysis of TCSs difficult [31]. This approach of modeling is only necessary when we need to have a microscopic view of the traffic for our analysis purposes.

Our approach is based on *Eulerian* models in which the regions of the traveling space, e.g. sub-tracks, are the concern of modeling [31], [32]. This kind of modeling is more appropriate for modeling rerouting/rescheduling of the moving objects [32]. By modeling each sub-track as an actor, we develop a one-dimensional model of the traveling space instead of a complex multi-dimensional model of the moving objects. This approach of modeling not only provides an acceptable fidelity for the problem [1], but also relieves the analysis difficulties. It is notable that a few of the mentioned approaches such as [23] verify the system at runtime. The approach of [23] is not compositional. The approach of [24] uses simulation, and [26] and [25], [28], [29] respectively analyze one aircraft and one to four trains.

Based on the related work, there is an increased interest towards the scheduling and path planning of moving objects in TCSs, i.e. [33], [34] use priced timed automata for the resource scheduling and the aircraft landing problem, [35], [36] use timed automata for the path planning in robotic systems, and [37] use the P programming language and attempts to compute an optimal collision-free motion plan for a robot. The scheduling and the path planning are not concerns of this paper.

Formal Analysis of Self-adaptive Systems at Runtime. PobSAM [38], [39] is an integration of algebraic formalism and actor-based Rebeca [40] models for modeling and verification of self-adaptive systems. In PobSAM there is no notion of timing constraint, and no focus on the verification at runtime.

The approach of [41] uses an incremental verification



(a) (b) (c)

Fig. 8: The verification time in (ES2) for $\lambda = 0.5$. The storm occurs at time 100. The variations of the time needed to verify the experiments with no deadlock using the compositional (C) and non-compositional approaches (NC) are depicted in parts (a) and (b), respectively. The variations of the time needed to detect a deadlock using both approaches are depicted in part (c).

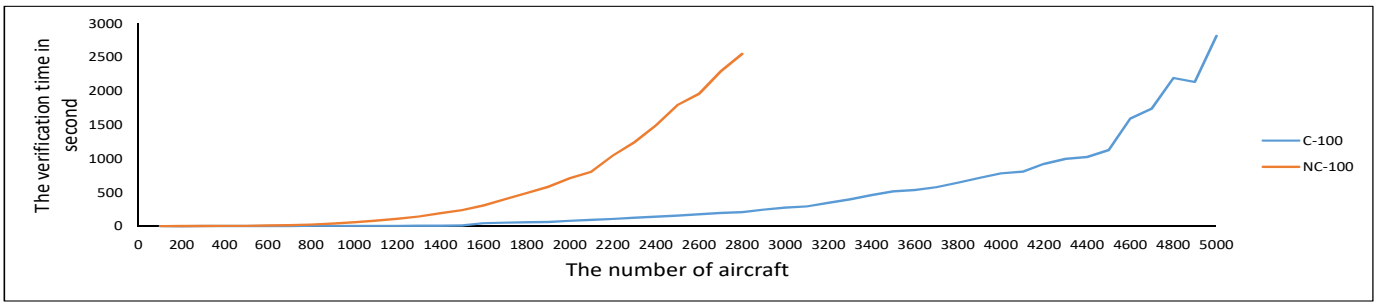


Fig. 9: The scalability of compositional (C) and non-compositional (NC) approaches in (ES3). Both approaches are run for the same scenario with $\lambda = 0.5$. The storm occurs at time 100. The scalability is measured in the number of aircraft, while the verification time is set to a threshold. The non-compositional approach is not able to verify a model with more than 2800 aircraft in a time less than the defined threshold.

technique to verify a Markov Decision Process (MDP) when parameters of the model are changed at runtime. The MDP is constructed incrementally by inferring a set of states needed to be rebuilt. In [42], a parametric Discrete Time Markov Chain (DTMC) is analyzed and a set of symbolic expressions is reported as the result. This way, the runtime verification of a DTMC is reduced to calculating the expressions' values when parameters get values at runtime. The work of [43] designs a self-adaptive software as a dynamic software product line, and uses parametric DTMCs to model common behaviors and variation points of the products separately. Therefore, there is no need to verify each configuration separately. RINGA [44] develops a design-time model of a system using Finite State Machines (FSM), where transitions are assigned equations parameterized by environmental variables and trigger the adaptations encoded in the states. RINGA abstracts the model for using at runtime. Lotus@runtime [45] monitors execution traces of the system and updates probabilities of a model designed by a transition system. The desirable properties in [45] are explained through a source state, a target state, a condition to be satisfied, and the probability of satisfying the condition. In comparison to [41], [42], [43], [44], [45] which use state-based models, an actor model is in a higher level of abstraction. Our actor-based approach besides decreasing the semantic gap between the model@runtime and applications, facilitates the modular analysis of the system.

The failure propagation is studied in [46] that checks whether the structural adaptation of the system is fast enough to prevent a hazard. Our approach, besides detecting a hazard, checks timing properties over a model. The latency-aware adaptation is studied in [47], where a probabilistic model checker proactively selects an adaptation strategy to maximize the utility of the system. Unlike [47], our focus is on effectively verifying the system behavior. The work of [48] investigates which state of the system is a safe state to update the implementation of the system whenever an environment assumption is changed. It also automatically synthesizes a new controller for the system. The approach of [48] is applied on a RailCab system where an accident should be avoided before the RailCabs enter into a crossing. [48] does not verify the system after adapting it.

Compositional Methods. The idea of compositional verification of actor-based models are first introduced for Rebeca in [40], [49], and providing a compositional semantics for Rebeca using automata is presented in [50]. Compositional methods for Timed Rebeca [51], [52] models are not yet investigated in depth. The approaches of [53], [54] use an assume-guarantee reasoning to respectively verify self-adaptive systems at design time and check the satisfaction of a property over a real-time system. [53] focuses on safety properties and uses a backward reasoning to generate new assumptions on the context of an adapted component. If it reaches a null assumption on the context of the system, the adaptation is incorrect. A property in [54] is divided into a set of subspecifications for which an assumption and a

guarantee are defined. The property is satisfied if its sub-specifications refine a combination of their corresponding assumptions and guarantees. This approach is not proposed for self-adaptive systems. The approach of [55] partitions the state space of a MDP into regions, magnifies on each region, and calculates the maximal probabilities by obtaining the upper and lower bounds of probabilities on each region. Unlike [55], we focus on timed systems.

In [56], a variant of UML diagrams is used to define components of a system, their interactions, and their timing and hybrid behaviors. Besides separately checking the safety of each component, [56] checks whether interfaces of components are well-defined. Compared to our approach, [56] models hybrid behaviors. The verification at runtime is not a concern in [56]. The approach of [57] groups components of the system in a way that a group is verified separately and its adaptation affects the satisfaction of one requirement. In contrast to [57], instead of considering a fixed number of components per each requirement, we increase the verification domain whenever it is needed.
None of the above studies consider the change propagation.

Interface Theory. The theory of interfaces is a widely studied topic, describing the main features that each component-based design should obey, such as refinement, structural composition, and conjunction. The approach of composition in the studies is either optimistic [21], [58], [59], meaning that two components are compatible if there is a helpful environment to avoid an error state in their parallel product, or pessimistic [60], [61], meaning that two components should work together in all environments. In [21], a theory of timed interfaces is proposed, and the interfaces in [58] are specified by Timed Input Output Transition Systems. Compared to [21], the system in [58] is input-enabled that is also an assumption in [58], where [58] uses TIOAs to specify interfaces. In [59], an interface theory for Modal Input Output Automata is proposed, and the interfaces in [61] are specified by Modal Input Output transition systems in which timing constraints are not specified. Compared to the related work, we follow a pessimistic approach of composition. Also the same as [21], we are able to express the input assumptions and there is no need for the input-enabledness assumption.

8 Conclusion and Future Work

We proposed Magnifier, a compositional approach that iteratively detects the propagation of a change and incrementally involves the components affected by a change into the analysis. An adaptation policy may contain the change and prevent the change to be propagated. In the worse case, the change propagates to the whole system, and Magnifier needs to compose all components of the model. We compared the compositional approach of Magnifier and the non-compositional approach in Section 6. We argued that even in the worst case, Magnifier performs better than the non-compositional approach. Looking more carefully into this comparison and building a formal proof is a part of our future work. The comparisons between our model, CoodAA, and other similar models on self-adaptive systems are presented in [1], [3]. In Section 7, we included a comparison of Magnifier with other analysis approaches for TTCS and other compositional methods.

Our Ptolemy II implementation of Magnifier is specialised for ATC. In [32], Lee and Sirjani show how CoodAA can capture TTCS applications in general. Here we consider a constant number of ports for all actors, and the topology formed by connecting the ports is a mesh. The extension to a dynamic number of ports and further than that to dynamic bindings, seem like natural future work. The general idea of Magnifier is not limited to TTCSs. It can be generalised for any control system with a modular design. We need to extend our model to include more general actors with different behaviors and different bindings among the ports. To investigate the details of such extension is another future direction. The possibility of analyzing actors in a compositional way is a consequence of their isolation discussed in [9] by Sirjani et al. Hence, we believe that CoodAA and Magnifier can be further extended and used in different areas and applications based on the foundations provided in this paper.

Acknowledgments

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. The work of the second author is supported in part by KKS SACSys Synergy project (Safe and Secure Adaptive Collaborative Systems), KKS DPAC Project (Dependable Platforms for Autonomous Systems and Control), and SSF Serendipity project at Malardalen University, and MACMa Project (Modeling and Analyzing Event-based Autonomous Systems) at Software Center, Sweden. The authors would like to thank Professor Edward A. Lee for his valuable comments.

References

[1] M. Bagheri, M. Sirjani, E. Khamespanah, N. Khakpour, I. Akkaya, A. Movaghar, and E. A. Lee, "Coordinated actor model of self-adaptive track-based traffic control systems," *Journal of Systems and Software*, vol. 143, pp. 116 – 139, 2018.

[2] B. H. C. Cheng, K. I. Eder, M. Gogolla, L. Grunske, M. Litoiu, H. A. Müller, P. Pelliccione, A. Perini, N. A. Qureshi, B. Rumpe, D. Schneider, F. Trollmann, and N. M. Villegas, *Using Models at Runtime to Address Assurance for Self-Adaptive Systems*. Cham: Springer International Publishing, 2014, pp. 101–136.

[3] M. Bagheri, I. Akkaya, E. Khamespanah, N. Khakpour, M. Sirjani, A. Movaghar, and E. A. Lee, "Coordinated actors for reliable self-adaptive systems," in *Formal Aspects of Component Software: FACS 2016*, O. Kouchnarenko and R. Khosravi, Eds., 2017, pp. 241–259.

[4] C. Hewitt, "Description and theoretical analysis (using schemata) of planner: A language for proving theorems and manipulating models in a robot," MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTELLIGENCE LAB, Tech. Rep., 1972.

[5] G. Agha, *Actors: A Model of Concurrent Computation in Distributed Systems*. Cambridge, MA, USA: MIT Press, 1986.

[6] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager, "Timed i/o automata: A mathematical framework for modeling and analyzing real-time systems," in *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003.* IEEE, 2003, pp. 166–177.

[7] E. M. Clarke, D. E. Long, and K. L. McMillan, "Compositional model checking," in *Logic in Computer Science. LICS'89, Proceedings., Fourth Annual Symposium on*. IEEE, 1989, pp. 353–362.

[8] C. Ptolemaeus, *System Design, Modeling, and Simulation: Using Ptolemy II*. Ptolemy. org Berkeley, CA, USA, 2014.

[9] M. Sirjani, E. Khamespanah, and F. Ghassemi, "Reactive actors: Isolation for efficient analysis of distributed systems," in *2019 IEEE/ACM 23rd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, 2019, pp. 1–10.

[10] S. Bensalem, M. Bozga, J. Sifakis, and T.-H. Nguyen, "Compositional verification for component-based systems and application," in *Automated Technology for Verification and Analysis*, S. S. Cha, J.-Y. Choi, M. Kim, I. Lee, and M. Viswanathan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 64–79.

[11] S. Bensalem, "Compositional verification of timed systems." in *VECoS*. Citeseer, 2014, pp. 5–11.

[12] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker, "A survey on engineering approaches for self-adaptive systems," *Pervasive and Mobile Computing*, vol. 17, no. Part B, pp. 184 – 206, 2015, 10 years of Pervasive Computing' In Honor of Chatschik Bisdikian.

[13] R. de Lemos, H. Giese, H. Müller, M. Shaw, J. Andersson *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*, ser. LNCS, R. de Lemos, H. Giese, H. Müller, and M. Shaw, Eds. Springer Berlin Heidelberg, 2013, vol. 7475, pp. 1–32.

[14] M. Bagheri, E. Khamespanah, M. Sirjani, A. Movaghar, and E. A. Lee, "Runtime compositional analysis of track-based traffic control systems," *SIGBED Rev.*, vol. 14, no. 3, pp. 38–39, Nov. 2017.

[15] *North atlantic operations and airspace manual*, International Civil Aviation Organization (ICAO), 2016.

[16] A. Zbrzezny and A. Półrola, "Sat-based reachability checking for timed automata with discrete data," *Fundamenta Informaticae*, vol. 79, no. 3-4, pp. 579–593, 2007.

[17] S. Tripakis, "Verifying progress in timed systems," in *Formal Methods for Real-Time and Probabilistic Systems*, J.-P. Katoen, Ed. Springer Berlin Heidelberg, 1999, pp. 299–314.

[18] M. Bagheri, M. Sirjani, E. Khamespanah, C. Baier, and A. Movaghar, "Magnifier: A compositional analysis approach for autonomous traffic control," *CoRR*, vol. abs/1905.06732, 2020. [Online]. Available: http://arxiv.org/abs/1905.06732

[19] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on uppaal 4.0," *Department of computer science, Aalborg university*, 2006.

[20] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan 2003.

[21] L. de Alfaro, T. A. Henzinger, and M. Stoelinga, "Timed interfaces," in *Embedded Software*, A. Sangiovanni-Vincentelli and J. Sifakis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 108–122.

[22] S. D. Chawade, M. A. Gaikwad, and R. M. Patrikar, "Review of xy routing algorithm for network-on-chip architecture," *International Journal of Computer Applications*, vol. 43, no. 21, pp. 975–8887, 2012.

[23] L. Bu, Q. Wang, X. Chen, L. Wang, T. Zhang, J. Zhao, and X. Li, "Toward online hybrid systems model checking of cyber-physical systems' time-bounded short-run behavior," *SIGBED Rev.*, vol. 8, no. 2, pp. 7–10, Jun. 2011.

[24] H. A. P. Blom, J. Krystul, and G. J. Bakker, "A particle system for safety verification of free flight in air traffic," in *Proceedings of the 45th IEEE Conference on Decision and Control*, 2006, pp. 1574–1579.

[25] R. Cheng, J. Zhou, D. Chen, and Y. Song, "Model-based verification method for solving the parameter uncertainty in the train control system," *Reliability Engineering and System Safety*, vol. 145, pp. 169 – 182, 2016.

[26] S. M. Loos, D. Renshaw, and A. Platzer, "Formal verification of distributed aircraft controllers," in *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '13. ACM, 2013, pp. 125–130.

[27] K. Margellos and J. Lygeros, "Toward 4-d trajectory management in air traffic control: A study based on monte carlo simulation and reachability analysis," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 5, pp. 1820–1833, 2013.

[28] W. Damm, A. Mikschl, J. Oehlerking, E.-R. Olderog, J. Pang, A. Platzer, M. Segelken, and B. Wirtz, *Automating Verification of Cooperation, Control, and Design in Traffic Applications*. Springer Berlin Heidelberg, 2007, pp. 115–169.

[29] Y. Zhao and K. Y. Rozier, "Formal specification and verification of a coordination protocol for an automated air traffic control system," *Science of Computer Programming*, vol. 96, pp. 337 – 353, 2014, special Issue on Automated Verification of Critical Systems (AVoCS 2012).

[30] "Performance analysis and verification of safety communication protocol in train control system," *Computer Standards and Interfaces*, vol. 33, no. 5, pp. 505 – 518, 2011.

[31] P. K. Menon, G. D. Sweriduk, and K. D. Bilimoria, "New approach for modeling, analysis, and control of air traffic flow," *Journal of guidance, control, and dynamics*, vol. 27, no. 5, pp. 737–744, 2004.

[32] E. A. Lee and M. Sirjani, "What good are models?" in *Formal Aspects of Component Software*, K. Bae and P. C. Ölveczky, Eds. Cham: Springer International Publishing, 2018, pp. 3–31.

[33] G. Behrmann, K. G. Larsen, and J. I. Rasmussen, "Optimal scheduling using priced timed automata," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 4, pp. 34–40, Mar. 2005.

[34] J. I. Rasmussen, K. G. Larsen, and K. Subramani, "On using priced timed automata to achieve optimal scheduling," *Formal Methods in System Design*, vol. 29, no. 1, pp. 97–114, Jul 2006.

[35] M. M. Quottrup, T. Bak, and R. I. Zamanabadi, "Multi-robot planning : a timed automata approach," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 5, April 2004, pp. 4417–4422 Vol.5.

[36] Y. Zhou, D. Maity, and J. S. Baras, "Timed automata approach for motion planning using metric interval temporal logic," in *2016 European Control Conference (ECC)*, June 2016, pp. 690–695.

[37] A. Desai, I. Saha, J. Yang, S. Qadeer, and S. A. Seshia, "Drona: A framework for safe distributed mobile robotics," in *2017 ACM/IEEE 8th International Conference on Cyber-Physical Systems (ICCPS)*, April 2017, pp. 239–248.

[38] N. Khakpour, S. Jalili, M. Sirjani, U. Goltz, and B. Abolhasanzadeh, "Hpobsam for modeling and analyzing it ecosystems – through a case study," *Journal of Systems and Software*, vol. 85, no. 12, pp. 2770 – 2784, 2012, self-Adaptive Systems.

[39] N. Khakpour, J. Kleijn, and M. Sirjani, "A formal model to integrate behavioral and structural adaptations in self-adaptive systems," in *Fundamentals of Software Engineering - 8th International Conference, FSEN 2019, Tehran, Iran, May 1-3, 2019, Revised Selected Papers*, 2019, pp. 3–19.

[40] M. Sirjani, A. Movaghar, A. Shali, and F. S. de Boer, "Modeling and verification of reactive systems using rebeca," *Fundam. Inform.*, vol. 63, no. 4, pp. 385–410, 2004.

[41] V. Forejt, M. Kwiatkowska, D. Parker, H. Qu, and M. Ujma, "Incremental runtime verification of probabilistic systems," in *Runtime Verification*, ser. LNCS, S. Qadeer and S. Tasiran, Eds., 2013, vol. 7687, pp. 314–319.

[42] A. Filieri and G. Tamburrelli, "Probabilistic verification at runtime for self-adaptive systems," in *Assurances for Self-Adaptive Systems*, ser. LNCS, J. Cámara, R. de Lemos, C. Ghezzi, and A. Lopes, Eds., 2013, vol. 7740, pp. 30–59.

[43] C. Ghezzi and A. Molzam Sharifloo, "Dealing with non-functional requirements for adaptive systems via dynamic software product-lines," in *Software Engineering for Self-Adaptive Systems II*, ser. LNCS, R. de Lemos, H. Giese, H. Müller, and M. Shaw, Eds., 2013, vol. 7475, pp. 191–213.

[44] E. Lee, Y.-G. Kim, Y.-D. Seo, K. Seol, and D.-K. Baik, "Ringa: Design and verification of finite state machine for self-adaptive software at runtime," *Information and Software Technology*, vol. 93, no. Supplement C, pp. 200 – 222, 2018.

[45] D. M. Barbosa, R. G. D. M. Lima, P. H. M. Maia, and E. Costa, "Lotus@runtime: A tool for runtime monitoring and verification of self-adaptive systems," in *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, May 2017, pp. 24–30.

[46] C. Priesterjahn, D. Steenken, and M. Tichy, "Timed hazard analysis of self-healing systems," in *Assurances for Self-Adaptive Systems: Principles, Models, and Techniques*, J. Cámara, R. de Lemos, C. Ghezzi, and A. Lopes, Eds., 2013, pp. 112–151.

[47] G. A. Moreno, J. Cámara, D. Garlan, and B. R. Schmerl, "Proactive self-adaptation under uncertainty: a probabilistic model checking approach," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE*, 2015, pp. 1–12.

[48] C. Ghezzi, J. Greenyer, and V. P. L. Manna, "Synthesizing dynamically updating controllers from changes in scenario-based specifications," in *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, June 2012, pp. 145–154.

[49] M. Sirjani, "Rebeca: Theory, applications, and tools," in *Formal Methods for Components and Objects, 5th International Symposium, FMCO 2006, Amsterdam, The Netherlands, November 7-10, 2006, Revised Lectures*, 2006, pp. 102–126.

[50] M. Sirjani, M. M. Jaghoori, C. Baier, and F. Arbab, "Compositional semantics of an actor-based language using constraint automata," in *Coordination Models and Languages, 8th International Conference, COORDINATION 2006, Bologna, Italy, June 14-16, 2006, Proceedings*, 2006, pp. 281–297.

[51] 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.
[52] M. Sirjani and E. Khamespanah, "On time actors," in *Theory and Practice of Formal Methods - Essays Dedicated to Frank de Boer on the Occasion of His 60th Birthday*, 2016, pp. 373–392.
[53] P. Inverardi, P. Pelliccione, and M. Tivoli, "Towards an assume-guarantee theory for adaptable systems," in *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, 2009, pp. 106–115.
[54] A. David, K. G. Larsen, A. Legay, M. H. Møller, U. Nyman, A. P. Ravn, A. Skou, and A. Wksowski, "Compositional verification of real-time systems using ecdar," *International Journal on Software Tools for Technology Transfer*, vol. 14, no. 6, pp. 703–720, Nov 2012.
[55] L. de Alfaro and P. Roy, "Magnifying-lens abstraction for markov decision processes," in *Computer Aided Verification*, W. Damm and H. Hermanns, Eds., 2007, pp. 325–338.
[56] H. Giese and W. Schäfer, *Model-Driven Development of Safe Self-optimizing Mechatronic Systems with MechatronicUML*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 152–186.
[57] A. Borda, L. Pasquale, V. Koutavas, and B. Nuseibeh, "Compositional verification of self-adaptive cyber-physical systems," in *2018 IEEE/ACM 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, May 2018, pp. 1–11.
[58] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski, "Timed i/o automata: a complete specification theory for real-time systems," in *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*. ACM, 2010, pp. 91–100.
[59] F. Bujtor, "Modal interface automata: A theory for heterogeneous specification of parallel systems," Ph.D. dissertation, University of Augsburg, Germany, 2018.
[60] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager, "Timed i/o automata: a mathematical framework for modeling and analyzing real-time systems," in *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*, Dec 2003, pp. 166–177.
[61] S. S. Bauer, R. Hennicker, and S. Janisch, "Interface theories for (a)synchronously communicating modal i/o-transition systems," in *Proceedings Foundations for Interface Technologies, FIT 2010, Paris, France, 30th August 2010.*, 2010, pp. 1–8.

Ehsan Khamespanah is an Assistant Professor at University of Tehran. He is graduated from a double-degree Ph.D. program in the School of ECE at University of Tehran and the Department of Computer Science at Reykjavik University. His research interests include formal methods, software testing, cyber-physical systems, and software architecture. Ehsan has a BE in computer engineering form University of Tehran.

Christel Baier is a full professor and head of the chair for Algebraic and Logic Foundations of Computer Science at the Faculty of Computer Science of the Technische Universität Dresden since 2006. From the University of Mannheim she received her Diploma in Mathematics in 1990, her Ph.D. in Computer Science in 1994, and her Habilitation in 1999. She was an associate professor for Theoretical Computer Science at the University of Bonn from 1999 till 2006.

Maryam Bagheri received the M.S. degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2013. She is currently working toward the Ph.D. degree in computer engineering at the Department of Computer Engineering at Sharif University of Technology.

Ali Movaghar received the B.S. degree in electrical engineering from the University of Tehran, in 1977, and the M.S. and Ph.D. degrees in computer, information, and control engineering from the University of Michigan, Ann Arbor, in 1979 and 1985, respectively. He is a professor in the CE Department, Sharif University of Technology, in Tehran, Iran. His research interests include performance/dependability modeling and formal verification of distributed real-time systems. He is a senior member of the IEEE and the ACM.

Marjan Sirjani is a Professor and chair of Software Engineering at Mälardalen University, and the leader of Cyber-Physical Systems Analysis research group. Her main research interest is applying formal methods in Software Engineering. She works on modeling and verification of concurrent, distributed, and self-adaptive systems. Marjan and her research group are pioneers in building model checking tools for actor models, and designed and built the Rebeca modeling language.