



An improved approach on the model checking for an agent-based simulation system

Yinling Liu, Tao Wang, Haiqing Zhang, Vincent Cheutet

► To cite this version:

Yinling Liu, Tao Wang, Haiqing Zhang, Vincent Cheutet. An improved approach on the model checking for an agent-based simulation system. Software and Systems Modeling, 2020, 10.1007/s10270-020-00807-4 . hal-02946293

HAL Id: hal-02946293

<https://hal.science/hal-02946293>

Submitted on 25 Sep 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

An improved approach on the model checking for an agent-based simulation system

Yinling LIU^{a,*}, Tao WANG^b, Haiqing ZHANG^c, Vincent CHEUTET^a

^aUniversité de Lyon, INSA Lyon, Laboratoire DISP (EA4570), France;

^bUniversité de Lyon, Université Jean Monnet, Laboratoire DISP (EA4570), France;

^cSchool of Software Engineering, Chengdu University Information Technology, China;

Abstract

Model checking is an effective way to verify behaviours of an agent-based simulation system. Three behaviours are analysed: operational, control, and global behaviours. Global behaviours of a system emerge from operational behaviours of local components regulated by control behaviours of the system. The previous works principally focus on verifying the system from the operational point of view (operational behaviour). The satisfaction of the global behaviour of the system conforming to the control behaviour has not been investigated. Thus, in this paper, we propose a more complete approach for verifying global and operational behaviours of systems. To do so, these three behaviours are firstly formalized by automata-based techniques. The meta-transformation between automata theories and Kripke structure is then provided, in order to illustrate the feasibility for the model transformation between the agent-based simulation model and Kripke structure based model. Then, a mapping between the models is proposed. Subsequently, the global behaviour of the system is verified by the properties extracted from the control behaviour and the operational behaviour is checked by general system performance properties (e.g. safety, deadlock freedom). Finally, a case study on the simulation system for aircraft maintenance has been carried out. A counter-example of signals sending between Flight agent and Plane agent has been produced by NuSMV model checker. Modifications for the NuSMV model and agent-based simulation model have been performed. The experiment results show that 9% out of 19% of flights have been changed to be serviceable.

Keywords: Model checking; Agent-based simulation system; Global behaviours and operational behaviours; Model transformation.

*Corresponding author

Email addresses: yinling.liu@insa-lyon.fr (Yinling LIU), tao.wang@univ-st-etienne.fr (Tao WANG), haiqing.zhang.zhq@gmail.com (Haiqing ZHANG), vincent.cheutet@insa-lyon.fr (Vincent CHEUTET)

1. Introduction

The popularity of the Agent-Based Simulation System (ABSS) has increased dramatically over the past decade (Macal 2016; Abar et al. 2017). The present applications of the ABSS span a wide range of areas: project management (Song et al. 2018), logistics and transportation (Serrano-Hernandez et al. 2018), emergency evacuation (Joo et al. 2013), aircraft maintenance (Liu et al. 2018), etc. The basic elements of the ABSS consist of agents, agent relationships, and the agents' environment. The modeling of the ABSS is not that difficult. The modeling process involves the building of individual agents, the definition of the interactions between agents, and the design of the agents' environment. Agents are built individually and the interactions with neighboring agents are usually clear. However, patterns, structures, and behaviours emerging from the system are not explicitly programmed into the model but arise through the agent interactions (Macal and North 2010). Lacking the ability to represent and control agents not only increases the difficulty for the practical development of the ABSS but leads to the distrust for using agents in critical domains (Fisher 2005).

Formal verification is a powerful tool to demonstrate the correctness of system behaviours. In this paper, system behaviours are divided into global and operational behaviours. Many efforts have been made in this area. However, most of researches focus on how to extend the formal logic to support the description of the more complex system (Al-Saqqar et al. 2015; Raimondi 2006). Some other works contribute to the subject of model transformation (El Menshawwy et al. 2018; Keshanchi, Soury, and Navimipour 2017). Few studies have given attention to the system itself. The control behaviour, operational behaviour and global behaviour are often involved when we analyze a system. Understanding the conformity of global behaviours to control behaviours is another effective way to verify whether the system design is satisfactory (Bentahar et al. 2013). Therefore, the challenge of our research focuses on how to better verify the behaviours of the ABSS with only the existing formal logics.

In this paper, we address the issue of verifying both the global behaviour and the operational behaviour of the ABSS. The global behaviour is extracted from the ABSS, which describes the behaviour of the object of interest instead of concerning any details. The operational behaviour of the ABSS is based on how the ABSS functions from the operation level, where any reachable details will not be ignored. The differentiation of these two behaviours allows us to verify the behaviour of the ABSS from the abstract and detailed levels. The behaviour of the detailed level is verified by the general system performance properties like safety, liveness, etc. On the other hand, the verification for the behaviour of the abstract level is performed with a control behaviour. Control behaviours are application-independent, which regulate the operational behaviour (application-dependent) from the abstract level. In Bentahar et al. (2013) work, the control behaviour was proposed to verify the conformity of the operational behaviour of the composite web services. We verify the global behaviour of the ABSS based on the idea of Bentahar's work. The control be-

behaviour helps guarantee that the model of the ABSS considers design requirements and constraints. At the design time, it permits assessing the impacts of any changes in the global behaviour on the conformity of the global behaviour to the control behaviour. For instance, a new agent has to be integrated into the ABSS to complete its functionality. In this case, the control behaviour remains the same and it can be utilized to verify the conformity of the global behaviour of the new ABSS.

The symbolic model checking technique is chosen to verify global and operational behaviours of the ABSS. The problem of model checking is formally expressed by $M \models \varphi$, where M represents the system model, φ is a property, and \models is the satisfaction symbol to check whether the model M satisfies the property φ . If the property is not satisfied by the system, a counterexample is produced. The efficiency of this technique for verifying the multi-agent systems has been proven (Al-Saqqar et al. 2015), as it uses less memory than automata-based approaches and it alleviates the “state explosion” problem. NuSVM is a symbolic model checker designed to allow for the description of FSM (Finite State Machine) which ranges from completely synchronous to completely asynchronous, and from the detailed to the abstract (Cimatti et al. 2002). The primary purpose of NuSMV input language is to describe the transition relation of the FSM, which is quite suitable for describing the state transition of statecharts in agents. Therefore, in this paper, we choose NuSMV as the model checker. The approach proposed is employed to verify the behaviours of the ABSS proposed by Liu et al. (2019). As shown in Fig. 1, a number of agents such as Flight, Plane, etc., are involved and the general information between agents is provided. The ABSS has simulated the whole process of aircraft maintenance, including strategies, scheduling plans, the cost analysis and the service level analysis, etc. A significant exploration of maintenance information has been also performed in this system. As a result, we choose the design of this ABSS as our case study.

A framework is proposed to illustrate the overall approach to verifying the behaviours of the ABSS via symbolic model checking, which is shown in Fig. 2. This framework addresses the difference between global and operational behaviors, which implies the importance of the abstract and detailed design respectively. Global and operational behaviours are verified in different ways. The former is verified by the control behaviour. The latter is checked by the general system performance properties. Finally, the verification result can improve the system design by means of counterexamples. The counterexample of the global behavior can improve the design of the interactions between agents. The operational behaviour counterexample is able to correct the agent behaviour directly.

To conclude, the symbolic model checking technique is used to check the satisfaction of global and operational behaviours of the ABSS. The main contributions of this research work are illustrated as follows:

- proposing an improved model checking approach to verifying the ABSS from both the abstract and detailed points of view;

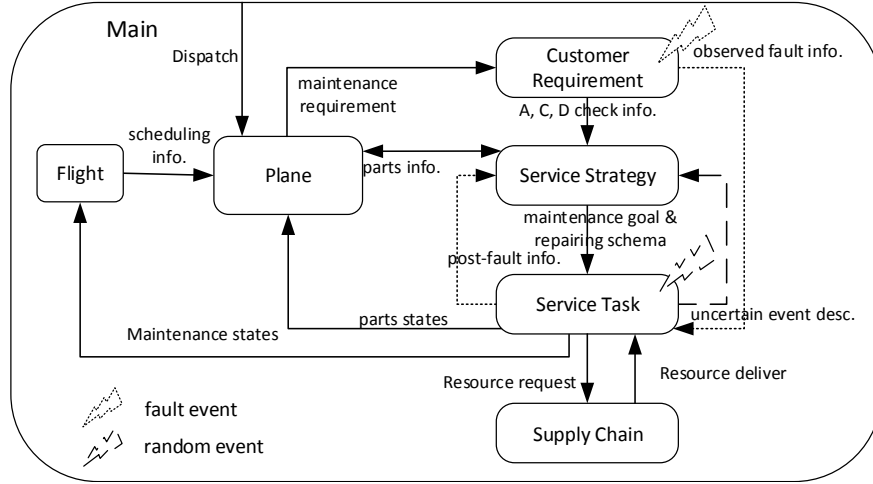


Figure 1: The architecture model of the ABSS, adapted from (Liu et al. 2019)

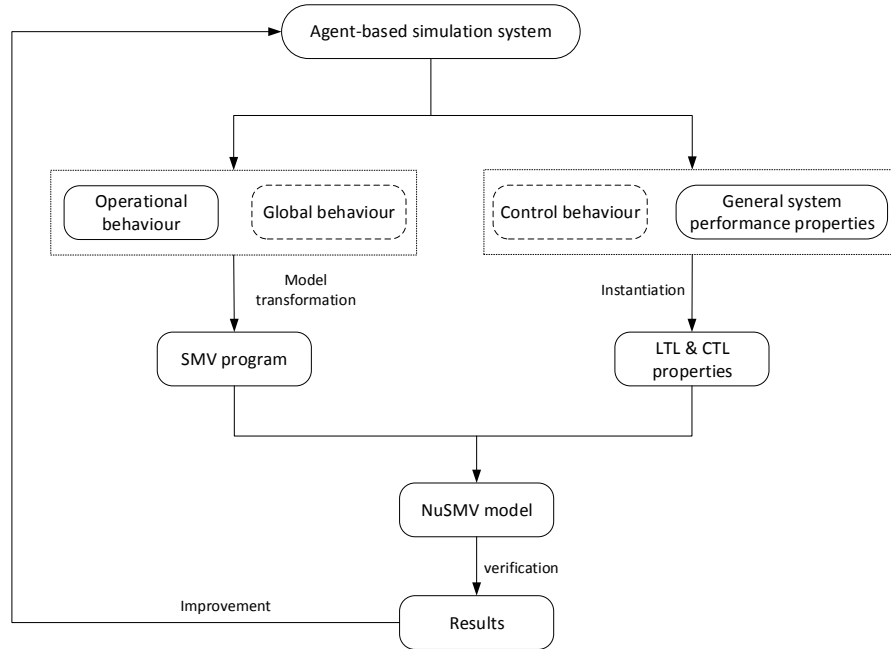


Figure 2: The framework on model checking for the ABSS

- providing a complete model transformation process from the ABSS to the formal model for verification;
- giving a concrete case study of formal methods in practice and demon-

strating the power of formal methods in improving designs and in providing new insights.

Based on the verification results, the problem of signals transferring between Flight agent and Plane agent has been discovered, which influences the system operation for a relatively long period of the simulation execution. The simulation results show that the efficiency of the aircraft maintenance has been improved after having modified the behaviour of Flight agent.

The remainder of this paper is structured as follows. Section 2 reviews the main related works. Section 3 formalizes behaviours of the ABSS. Section 4 provides the meta-model and model transformations as a basis for model checking and proposes properties to be checked and designs the properties to be checked. Section 5 gives the modification for agents based on model checking verification results and conducts a comparative simulation experiment to demonstrate the feasibility of the modification. Section 6 concludes the paper with future perspectives.

2. Related Researches and Limitations

Operational and control behaviours have been studied in Web services (Sheng et al. 2010; Bentahar et al. 2013; Souri and Navimipour 2014; Navimipour et al. 2015). The control behaviour was proposed by Bentahar et al. (2013) to verify the operational behaviour of composite web services. In their work, the control behaviour defines a general design of web services and guides and monitors the execution progress of the operational behaviour. The operational behaviour implies the business logic of a web service operation. The detailed formal definition of behaviours and the transformation process were provided. However, their verification approach is limited. The major problem may be that it can not check the operational behaviour where states transitions are defined with conditions. For example, the operation like $PlanePrepared \xrightarrow{\text{"invoke"}} PlaneScheduled$ can not be checked. Because the LTL (Linear Temporal Logic) or CTL (Computational Tree Logic) (Pnueli 1977) specification representing this kind of execution sequences can not pass by NuSMV model checker. Accordingly, we think that their approach is suitable to verify the global behaviour where fewer details are involved and no condition exists.

Many efforts have been made to verify the logic of multi-agent systems. Logic has been extended to support the model checking for complex systems. Al-Saqqar et al. (2015) extended the CTL logic with knowledge and commitments to verify the multi-agent systems with respect to knowledge and social commitments. Meski et al. (2014) extended LTL with the epistemic component for multi-agent systems. Raimondi (2006) extended the temporal logic to multi-modal logics for time, knowledge, correct behaviour, and strategies, in order to develop techniques and tools for the formal verification of the multi-agent system. These approaches principally contribute to improving the capability of model checking. Researchers tend to focus on the logic itself in model checking. However, few of them try to concentrate on better understanding

the system, in order to improve the efficiency of model checking. In this paper, we have not extended any logics. We have decided to check the ABSS by analyzing global and operational behaviours of the ABSS.

Model transformation is usually employed in model checking. Model checking often automatically verifies systems by model checkers. Many powerful model checkers such as SPIN¹, PRISM², UPPAAL³ and NuSMV⁴ are widely used. The modeling languages supported by model checkers are not usually appropriate to model the systems that need to be checked. Thus, model transformation techniques are involved in model checking. For example, Bordini et al. (2006) automatically transformed multi-agent systems into Promela or Java, in order to use the associated Spin and JPF⁵ model checker to verify system. Keshanchi, Souri, and Navimipour (2017) transformed the labeled transition system into NuSMV code to verify the genetic algorithm for task scheduling in the cloud environment. El Menshawry et al. (2018) transformed RTCTL^{cc} into RTCTL (real-time CTL) where RTCTL^{cc} expresses qualitative and quantitative commitment requirements. Another example is provided by Bentahar et al. (2013), they presented the process of transforming FSM into Kripke model. Similar to our work, however, their transformation process was not involving the messages.

Model checking is capable of verifying and demonstrating system behaviours. However, industrial applications of model checking are very limited. Researchers have a tendency to use less concrete examples. For example, Bentahar et al. (2013) used a ticket reservation system to illustrate their approach. This system just described the operation behaviour of the ticket reservation from the global point of view. No details about reservation processes were involved. Keshanchi, Souri, and Navimipour (2017) adopted the genetic algorithm to explain the feasibility of model checking. Only a single module was referred to. Al-Saqqar et al. (2015) modelled NetBill protocol to demonstrate the efficiency of their method. A real case was analyzed by El Menshawry et al. (2018). They chose the landing gear system in Boniol and Wiels (2014) as their case study, which was a real and industrial case. They corrected their NuSMV model after having found a counter-example by specifications. However, they did not discuss the reason why the original model was not correct and the comparison experiments were not provided regarding original and corrected models.

As a conclusion, we are motivated to propose a more complete approach of model checking and our method is explained based on a real simulation system for aircraft maintenance. So, the improvement of the system and comparison experiments are performed, which provides an example of an application of formal methods.

¹<http://spinroot.com/spin/whatispin.html> (accessed in March 2019)

²<http://www.prismmodelchecker.org/> (accessed in March 2019)

³<http://www.uppaal.org/> (accessed in March 2019)

⁴<http://nusmv.fbk.eu/> (accessed in March 2019)

⁵<https://github.com/javapathfinder/> (accessed in March 2019)

3. The Formalization of ABSS Behaviours

In this section, the specification of the ABSS is formalized as the global behaviour, operational behaviour and control behaviour. Control behaviour expresses the general behaviour of any process related to aircraft maintenance. However, global behaviour and operational behaviour are dependent on applications. These behaviours have been investigated for isolated or composite web services (Maamar et al. 2009; Yahyaoui, Maamar, and Boukadi 2010; Bentahar et al. 2013). Different methods like Büchi automaton and labeled transition system, were used to formalize the behaviours. Büchi automaton extends a finite automaton to infinite inputs, which meets the requirements for the continuous invocation of plane's service. Hence, we use Büchi automaton to describe the behaviours of the ABSS.

3.1. Global Behaviour and Operational Behaviour

The global behaviour (Fig. 3) describes the behaviours happening in the life cycle of planes from the global point of view. The operational behaviour (Fig. 4) is employed to describe the inner behaviour of the component. The definitions of behaviours are illustrated as follows.

Definition 3.1. (Global Behaviour) The global behaviour of an agent-based system is a 5-tuple $GB = \langle \Sigma_{gb}, Q_{gb}, Q^0, F_{gb}, \Delta_{gb} \rangle$, where:

- Σ_{gb} is a finite set of messages sending between agents;
- Q_{gb} is a finite set of states of agents;
- $Q^0 \subseteq Q_{gb}$ is a set of initial states;
- $F_{gb} \subseteq Q_{gb}$ is a set of final states;
- $\Delta_{gb} \subseteq Q_{gb} \times \Sigma_{gb} \times Q_{gb}$ is a transition relation.

Definition 3.2. (Agent Operational Behaviour) The operational behaviour of an agent is a 7-tuple $AOB = \langle \Sigma_{aob}, \Sigma^{in}, \Sigma^{out}, Q_{aob}, Q^0, F_{aob}, \Delta_{aob} \rangle$, where:

- Σ_{aob} is a finite set of messages of the agent, including the empty message \emptyset ;
- $\Sigma^{in} \subseteq \Sigma_{aob}$ is a set of messages inside the agent;
- Σ^{out} is a set of messages outside the agent;
- Q_{aob} is a finite set of states of the agent;
- $Q^0 \subseteq Q_{aob}$ is a set of initial states;
- $F_{aob} \subseteq Q_{aob}$ is a set of final states;
- $\Delta_{aob} \subseteq Q_{aob} \times \Sigma_{aob} \cup \Sigma^{out} \times Q_{aob}$ is a transition relation.

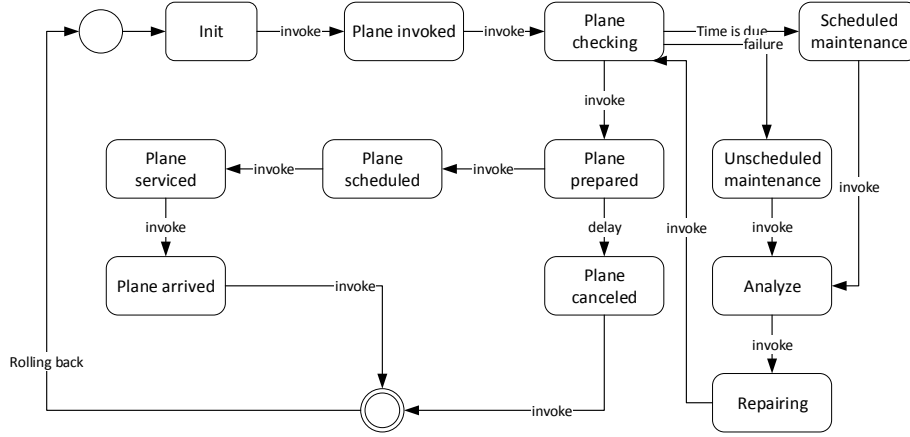


Figure 3: The global behaviour of the ABSS for aircraft maintenance

Definition 3.3. (Valid Conversation) A conversation in AOB over a sequence of messages $\Sigma_0, \Sigma_1, \dots, \Sigma_n$ from Σ_{aob} is a sequence $Q^0 \xrightarrow{\Sigma_0} Q^1 \xrightarrow{\Sigma_1} Q^2 \dots Q^{n-1} \xrightarrow{\Sigma_{n-1}} Q^n$ such that $\forall i \geq 0, Q^i, Q^{i+1} \in Q_{aob}, (Q^i, \Sigma_i, Q^{i+1}) \in \Delta_{aob}, Q^0 \in Q_0, Q^n \in F_{aob}$. The conversation is said to be valid iff it visits infinitely an element of F_{aob} .

As the forms of global and operational behaviours are similar, we just provide an example of the operational behaviour. Fig. 4 represents the operational behaviour of Flight agent. The elements of AOB over this agent are explained as follows:

- $\Sigma_{aob} = \{\text{STA.delay, Plane.ready, Plane.receive, "null", ("schedule", plane)}\};$
- $\Sigma^{in} = \{\text{STA.delay, Plane.ready, Plane.receive}\};$
- $\Sigma^{out} = \{("schedule", plane)\};$
- $\Delta_{aob} = \{(\text{Init, STA.delay, Checking}), (\text{Checking, Plane.ready, Ready}), (\text{Ready, ("schedule", plane), Scheduling}), (\text{Init, ("schedule", plane), Scheduling}), (\text{Scheduling, Plane.receive, End}), (\text{End, "null", Init})\}.$

One conversation in the AOB of Flight agent can be:

$\text{Init} \xrightarrow{("schedule", plane)} \text{Scheduling} \xrightarrow{\text{Plane.receive}} \text{End}.$

It should be noted that dashed, dotted and solid arrows imply the messages of receiving from other agents, sending to other agents, and sending to the interior respectively. The messages of sending or receiving are always associated with the destination agents. For the message of receiving from other agents like "STA.delay", it is composed of the name of agent (e.g. STA) and the signal (e.g. "delay"). In terms of the message of sending to other agents like ("schedule", plane), it consists of the signal ("schedule") and the agent (Plane).

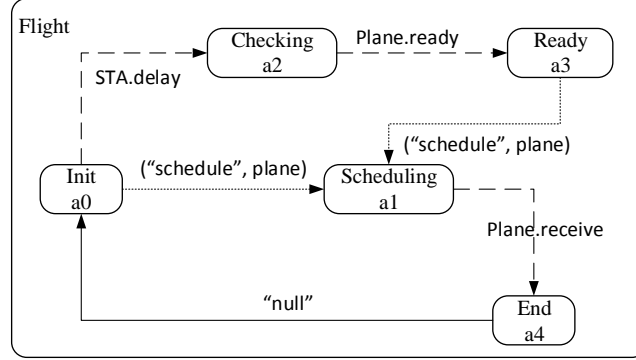


Figure 4: The operational behaviour of Flight agent

3.2. Control Behaviour

Control behaviours restrict the operation sequence of system behaviours. The conformity to the control behaviour reflects the correctness of the system behaviours. Since the control behaviour is formalized by Büchi automaton as well, we do not repeat its definition. Here we just introduce the content of the control behaviour.

Fig. 5 illustrates the control behaviour of the ABSS, which is adapted from Bentahar et al. (2013). They provided a control behaviour of composite web services. Since the control behaviour is the application independent specification of a business logic, it is possible to adapt control behaviours of one domain to another domain. In terms of web services, the concept of “compensate” implies the original execution prices will be refunded when the service provider can not provide the committed service or deliver the ordered commodity (Liu, Ngu, and Zeng 2004). However, as for aircraft maintenance, we have not involved this issue. Thus, we delete this behaviour. In addition, “Aborted” refers to one flight has to be canceled because of a long delay.

To illustrate the efficient design of the global behaviour of the system, we check the conformity of the global behaviour of the system to the control behaviour. The conformity is defined as the relationship between the conversations of the global behaviour and the control behaviour executions. According to Bentahar et al. (2013), the type of the conformity can be: weak and strong. The former implies each conversation c in the global behaviour is simulated by an execution e in the control behaviour, which can be denoted as $c \rightsquigarrow e$. The latter highlights each valid execution e in the control behaviour is simulated represented by a possible conversation c in the global behaviour ($e \rightsquigarrow c$).

Based on the two types of the conformity, all the conversations in Fig. 3 should be mapped onto valid executions in Fig. 5. On the other hand, all the valid executions in Fig. 5 should be mapped onto the conversations in Fig. 3. Since the control behaviour is application-independent and more abstract, the mapping is not necessarily state-to-state and transition-to-transition. For

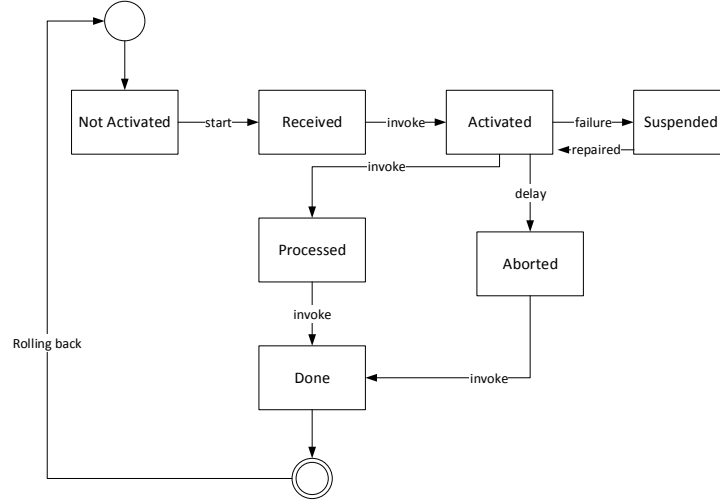


Figure 5: The control behaviour of the ABSS

instance, Conversation 1 in the global behaviour and Execution 1 in the control behaviour (* means repetition or loop) are shown as follows:

$$\begin{aligned}
 \text{Conversation1} = & (\text{Init} \xrightarrow{\text{"invoke"}} \text{PlaneInvoked} \xrightarrow{\text{"invoke"}} \text{PlaneChecking} \\
 & \xrightarrow{\text{"TimeIsDue"}} \text{ScheduledMaintenance} \xrightarrow{\text{"invoke"}} \text{Analyze} \xrightarrow{\text{"invoke"}} \text{Repairing} \\
 & \xrightarrow{\text{"invoke"}} \text{PlaneChecking} \xrightarrow{\text{"invoke"}} \text{PlanePrepared} \xrightarrow{\text{"delay"}} \text{PlaneCanceled})^* \\
 \text{Execution1} = & (\text{NotActivated} \xrightarrow{\text{"start"}} \text{Received} \xrightarrow{\text{"invoke"}} \text{Activated} \xrightarrow{\text{"failure"}} \text{Suspended} \\
 & \xrightarrow{\text{"repaired"}} \text{Activated} \xrightarrow{\text{"delay"}} \text{Aborted} \xrightarrow{\text{"invoke"}} \text{Done})^*
 \end{aligned}$$

The mappings between state-to-state (S-S) and transition-to-transition (T-T) are explained in Table 1. Verifying this conformity one by one manually is quite tedious even if a small number of states are involved. The number of possible conversations grows dramatically with the increase in the number of states. So, an automatic verification approach is proposed in Section 4.

3.3. The Adaptability of Büchi Automata

A typical agent-based model has three elements (Macal and North 2010):

- A set of agents, their attributes and behaviours;
- A set of agent *relationship* and methods of interaction: an underlying topology of connectedness defines how and with whom agents interact;
- The agents' *environment*: agents interact with their environment in addition to other agents.

Table 1: The mapping between Conversation1 and Execution1

	Conversation1	Execution1
S-S	<i>Init</i>	<i>NotActivated</i>
	<i>PlaneInvoked</i>	<i>Received</i>
	<i>PlaneChecking, PlanePrepared</i>	<i>Activated</i>
	<i>ScheduledMaintenance, Analyze, Repairing</i>	<i>Suspended</i>
	<i>PlaneCanceled</i>	<i>Aborted, Done</i>
T-T	<i>Init</i> \rightarrow <i>PlaneInvoked</i>	<i>NotActivated</i> \rightarrow <i>Received</i>
	<i>PlaneInvoked</i> \rightarrow <i>PlaneChecking</i>	<i>Received</i> \rightarrow <i>Activated</i>
	<i>PlaneChecking</i> $\rightarrow \dots \rightarrow$ <i>Prepared</i>	<i>Activated</i> \rightarrow <i>Suspended</i> \rightarrow <i>Activated</i>
	<i>PlanePrepared</i> \rightarrow <i>PlaneCanceled</i>	<i>Activate</i> \rightarrow <i>Aborted</i> \rightarrow <i>Done</i>

From this definition, the agent-based system not only concentrates on the behaviour of individual agents but addresses the interactive behaviour between agents.

The agent-based system is usually formalized by IS (Interpreted System) to reason about knowledge and temporal properties (Fagin et al. 2004). The formalism is illustrated as follows (Al-Saqqar et al. 2015) :

- A set of n agents $\mathcal{A} = \{1, 2, \dots, n\}$ such that each agent i is described by:
 - A non-empty set of local states L_i . The local state of agent i is represented by $l_i \in L_i$. Each local state of an agent represents the complete information about the system that the agent has at a given moment;
 - A set of local actions Act_i to account for the temporal evolution of the system;
 - A local protocol function $P_i : L_i \rightarrow 2^{Act_i}$ to identify the set of enabled actions that could be performed in a given local state;
 - A local evolution function τ_i that determines the transitions for an individual agent i between its local states and it is defined as follows: $\tau_i : L_i \times Act_i \rightarrow L_i$.
- The set of all global states in the system $G \subseteq L_1 \times \dots \times L_n$: a subset of the Cartesian product of all local states of n agents.
 - A global state $g \in G$ is a tuple $g = (l_1 \dots l_n)$ that represents a “snapshot” of the system;
 - The local state of agent i in the global state g is represented by the notation $l_i(g)$.
- $I \subseteq G$: the set of initial global states for the system;
- The global evolution (transition) function: it can be defined as follows: $\tau : G \times ACT \rightarrow G$, where $ACT = Act_1 \times \dots \times Act_n$ and each component $a \in ACT$ is a joint action, which is a tuple of actions (one for each agent);

- A set Φ of atomic propositions and a valuation function \mathcal{V} for those propositions $\mathcal{V} : G \rightarrow 2^{\Phi_p}$.

This definition implies that the consideration of the agent level (detailed level) and the global level (abstract level) is necessary for the description of the agent-based system. The formalization of states and transitions from both level are required. As for our definitions, Definitions 3.1 and 3.2 are corresponding to global level and agent level respectively. The core idea of the agent-based system from these two definitions is captured by our definitions. As a result, our definitions are appropriate for the description of the ABSS as well. In addition, the idea of providing a new definition based on automaton theory is to take advantage of Büchi automaton. For example, infinite runs can be captured by finite structures, which is compatible with continuous invocation of aircraft maintenance.

4. Model Checking for the ABSS

In this section, the meta-model transformation from Büchi automaton to Kripke structure is firstly proposed. Then, the model transformation from the ABSS to the NuSMV model is provided. Finally, the specifications of LTL and CTL are designed.

4.1. Meta-model Transformation

To illustrate the feasibility of the transformation from the ABSS to the NuSMV model, the meta-model transformation is accomplished. The meta-model transformation implies the transformation from Büchi automaton to Kripke structure. This transformation process lays the foundations for the next model transformation. The definition of Kripke structure is shown in Definition 4.1. The transformation between them has been provided by (Bentahar et al. 2013). They discussed the transformation of states and transitions in detail. However, the messages of Büchi automaton has not been involved. The information of messages should be transformed into the labels of Kripke structure.

Definition 4.1. (Kripke structure) A Kripke structure is a 4-tuple $\mathcal{K} = \langle S, I, \delta, L \rangle$, where:

- S is a finite set of states;
- $I \subseteq S$ is the subset of initial states;
- $\delta \subseteq S \times S$ is the transition relation;
- $L : S \rightarrow 2^{AP}$ (elements of 2^{AP} are called labels) is a labelling function. AP is a finite set of atomic propositions.

A Büchi automaton is a 5-tuple $\mathcal{A} = \langle \Sigma, Q, Q^0, F, \Delta \rangle$, which is generalized by Definition 3.1. The transformation process from Büchi automaton to Kripke structure is illustrated as follows:

- $S = Q$: the states of Kripke structure are the same as those of Büchi automaton;
- $I = Q^0$: the initial states of Kripke structure are the same as those of Büchi automaton;
- $\forall s, s' \in S, (s, s') \in \delta$ and $L(s') = a$, iff $\exists s, s' \in Q, a \in \Sigma$ and $(s, a, s') \in \Delta$.

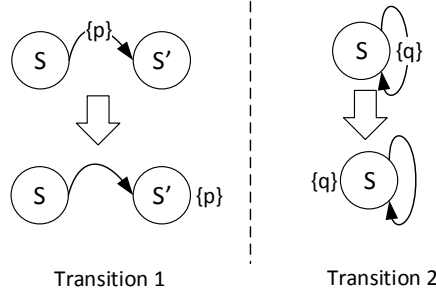


Figure 6: The meta-model transformation from Büchi automata to Kripke structure

Fig. 6 provides an example of the transitions in the transformation process. For Transition 1, $s, s' \in S, (s, s') \in \delta$ and $L(s') = \{p\}$, because $\exists s, s' \in Q, \{p\} \in \Sigma$ and $(s, \{p\}, s') \in \Delta$. Transition 2 shows the transition process when s, s' are the same state.

4.2. Model Transformation

The model transformation involves global and operational behaviours of the system. The former behaviour is verified by the control behaviour. So, the properties to be checked are extracted from the control behaviour. In order to make the verification clearer, all the states in the state chart of global behaviours will be corresponding to the state names of the control behaviour. As a result, the final state chart of the global behaviour is illustrated in Fig. 7 where the names of states are simplified as: NA (Not Activated), Re (Received), Ac (Activated), Su (Suspended), Ab (Aborted), Pr (Processed) and Do (Done). The mapping between global and control behaviours can be seen in Table 1. In addition, the states of *PlaneScheduled*, *PlanceServiced*, *PlaneArrived* of the global behaviour correspond to state *Processed* of the control behaviour. The transformation process will be analyzed in the following.

It should be noted that the verification of the global behaviour is not involved with labels. Because the verification focuses on the execution sequences of the states in the control behaviour. If labels are added to the transitions of NuSMV model, the properties about the execution sequence will never be satisfied. In general, transition processes are described as *TRANS* constraints. The constraints do not support that one state has more than one following state without labels. While the global behaviour of the system does not have any labels and one state may have more than one following state. Hence, *TRANS*

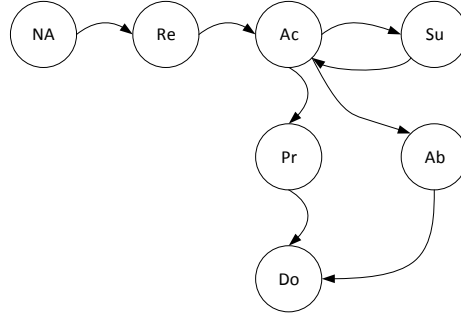


Figure 7: The state chart of the global behaviour of the system

constraints is not appropriate. We have chosen another semantically equivalent *ASSIGN* constraints. The transformation process of the global behaviour of the system is rather easy. The NuSMV code (Fig. 8) is given just by capturing the transitions in Fig. 7.

```

MODULE main

VAR state:{NA, Re, Ac, Su, Pr, Ab, Do};
ASSIGN
init (state) := NA;
next(state):=
  case
    (state = NA) : {Re};
    (state = Re) : {Ac};
    (state = Ac) : {Su, Pr, Ab};
    (state = Su) : {Ac};
    (state = Pr) : {Do};
    (state = Ab) : {Do};
    TRUE: state;
  esac;

```

Figure 8: The NuSMV model of the global behaviour of the system

The other transformation is dedicated to verifying the general system properties of the ABSS. This system has been implemented by simulation tool *Anylogic*⁶. The process principally includes the structure transformation and the agent transformation. The mapping between the ABSS and the NuSMV model is illustrated in Table 2.

The structure of the ABSS (Fig. 1) is transformed into the NuSMV code shown in Fig. 9. Here, we differentiate between module *main* in the NuSMV code and the instance of Main agent. The latter is named as *mainn*.

In terms of the agent operational behaviour, since *Anylogic* is used to create

⁶<https://www.anylogic.com/> (accessed in April 2019)

Table 2: The mapping between the ABSS and the NuSMV model

	ABSS	NuSMV model
Agent-based system structure	agents	processes
Agent operational behaviour	connections	parameters
	initial states	initial states
	transitions	transitions
	messages	actions
	timeout/ rate	Null (action)
	agent arrival condition	Reach (action) condition

the simulation system, the transition conditions of *Anylogic* should be transformed. The triggering types of a transition can be *messages*, *timeout*, *rate*, and *condition*. The *condition* of the ABSS implies either the logic combination of agent states or agent messages. Fig. 10 shows a complete transformation of Plane agent into the corresponding NuSMV code. The NuSMV code of lines five to the last provides an example of the condition transformation. The condition of the transformation from *ReadyToFly* to *Scheduled* in the state chart is that Plane agent receives *Flight.Schedule*. This condition is then transformed into

$(arg1.state = b7 \ \& \ arg2.action = Schedule) : b8.$

```

MODULE main
VAR
  flight : process Flight(flight, plane, sta);
  plane : process Plane(plane, flight, sta, ssa, mainn);
  cra : process CRA(cra, plane);
  sta : process STA(sta, cra, ssa, sca);
  ssa : process SSA(ssa, cra, sta);
  sca : process SCA(sca, sta);
  mainn : process Main(mainn, plane);

```

Figure 9: The transformation for the system architecture

4.3. Properties to Check

NuSMV allows to verify both CTL and LTL formulas (Cavada et al. 2019). CTL formulas specify properties over the computation tree of the FSM (branching-time approach). LTL formulas verify each linear path induced by the FSM (linear-time approach). Here, we employ the logics of CTL and LTL to express the properties. As global and operational behaviours should be checked, the properties are defined as two parts: global behaviour properties and operational behaviour properties.

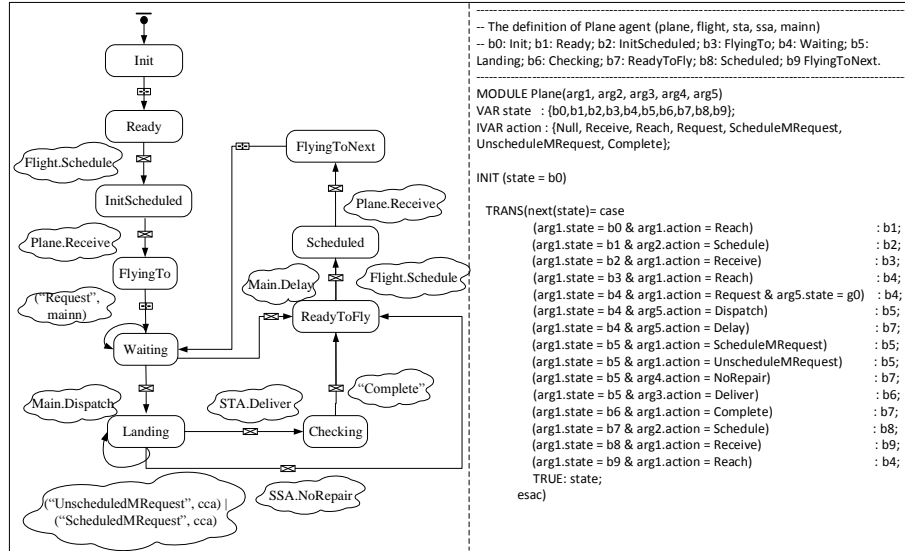


Figure 10: The transformation for Plane agent

The global behaviour properties are extracted from the control behaviour of the ABSS, which are illustrated as follows.

CTL specifications:

- $C1 - \varphi = AG(state = NA \rightarrow AX state = Re);$
- $C2 - \varphi = AG(state = Ac \rightarrow AX (state = Su \mid state = Pr \mid state = Ab));$
- $C3 - \varphi = AG(state = Ac \rightarrow EX (state = Ab));$
- $C4 - \varphi = AG((state = Pr \mid state = Ab) \rightarrow AX state = Do));$
- $C5 - \varphi = AG (state = NA \rightarrow EF state = Ab);$
- $C6 - \varphi = AG(state = Su \rightarrow EF (state = Pr \mid state = Ab));$
- $C7 - \varphi = AG(state = Su \rightarrow AX state = Ac);$
- $C8 - \varphi = AG(state = NA \rightarrow AF state = Ac);$

The formulas of C1, C2 and C7 follow the same form $\varphi = AG(state = A \rightarrow AX state = B)$. This form addresses that the next states of state A are always B. For example, property C1 illustrates that state *NotActivated* is followed by *Received*. C3 implies that the next state of state *Activated* is existentially *Aborted*. The properties of C5 and C6 share the form $\varphi = AG(state = A \rightarrow EF state = B)$, which highlights that state B is reachable from state A. The formula of C4 means that the process can reach *Aborted* state from *NotActivated*. The last formula specifies that the process will eventually in the future reach

state *Activated* from *NotActivated*. Therefore, property C8 shows that starting from state *NotActivated*, the process will eventually reach state *Activated*.

LTL specifications:

- $L1 - \varphi = G(\text{state} = NA \rightarrow X \text{state} = Re);$
- $L2 - \varphi = G(\text{state} = Ac \rightarrow X F(\text{state} = Su \mid \text{state} = Pr \mid \text{state} = Ab));$
- $L3 - \varphi = G((\text{state} = Pr \mid \text{state} = Ab) \rightarrow X F \text{state} = Do).$

Property L1 represents the form $\varphi = G(\text{state} = A \rightarrow \text{state} = B)$. This form implies in all possible computations, state *A* is followed by state *B*. Thus, L1 explains that the next state of state *A* is always *B*. The last two properties share the same form $\varphi = G(\text{state} = A \rightarrow X F \text{state} = B)$. It denotes that one of the future next states of state *A* will be state *B*. So, property L2 shows that the future next states of *Activated* will be *Suspended* or *Processed* or *Aborted*. Property L3 denotes that the system is at either *Processed* or *Aborted* state, it will eventually reach *Done* state.

Similar forms of LTL and CTL formulas have been proposed to verify the behaviour of the composite web services (Bentahar et al. 2013). However, after testing all the possibilities, some properties defined in their work such as property L1, C1 and C2 are recognized as incorrect properties. Because $p \ \& \ q$ in the formulas means conditions p and q hold at the same time and it can not express the sequence of states. However, the operator of X, EX, AX implies the satisfaction of the next state. Accordingly, there is no point in combining $\&$ and X, EX, AX . For example, $L1 - \varphi = G(! (Su \ \& \ X \ Do))$ always passes. On the other hand, $L1' - \varphi = G(Su \ \& \ X \ Do)$ is impossible to be satisfied. It is because $X \ Do$ is equal to *False* with no declaration of the current state. Su is treated as *True* because it is only a simple state declaration. Thus, the counter-example is as simple as this: "State :1.1: state = NA".

Properties from general system properties (Al-Saqqar et al. 2015) like safety, reachability, deadlock freedom, etc. are employed to check the operational behaviour of the system. We will define each property in the following.

Safety property

The safety property is often expressed by formula $AG \ ! \ \varphi$ (CTL property), where φ implies something bad. This formula means that the property φ is never satisfied in all possible computations. In the ABSS, bad situations like scheduling signal have been sent to the corresponding plane by Flight agent, whereas the plane has not received it. This situation can be expressed as follows:

CTL - S1 - $\varphi = AG(! (flight.state = Scheduled \ \& \ plane.state = ReadyToFly)).$

Liveness property

The liveness property means something good will be always possible to happen. The formulas of the liveness property are shown as follow. For example, the plane is always possible to fly (CTL - S2).

CTL - S2 - $\varphi = SPEC \ AG \ EF(plane.state = FlyingToNext);$

CTL - S3 - $\varphi = SPEC \ AG \ EF(plane.state = Waiting \rightarrow plane.state = ReadyToFly);$

CTL - S4 - $\varphi = \text{SPEC AG EF}(sca.state = \text{Buying});$
 CTL - S5 - $\varphi = \text{SPEC AG EF}(mainn.state = \text{Delaying}).$

Reachability property

The reachability property represents that some particular situation can be reached. The corresponding formulas are shown as follows. For example, in SSA agent, state *End* can be existentially reached from *StatesCheck*.

CTL - S6 - $\varphi = \text{SPEC EF}(ssa.state = \text{StatesCheck} \rightarrow ssa.state = \text{End});$
 CTL - S7 - $\varphi = \text{SPEC EF}(sca.state = \text{Buying}).$

Deadlock property

The dead-lock property can be checked by using the commands under the interactive mode. The commands are shown as follows:

1. *NuSMV -int ass.smv;*
2. *go;*
3. *check_fsm.*

The first two commands allow the NuSMV to enter the interactive mode. Command 3 checks the dead-lock situation for all the transition relations.

5. Simulation Experiment Results

In this section, we firstly illustrate the verification results from NuSMV model checker. The modification of the agent-based simulation model is then implemented according to counterexamples. Finally, a detailed discussion on our method is provided.

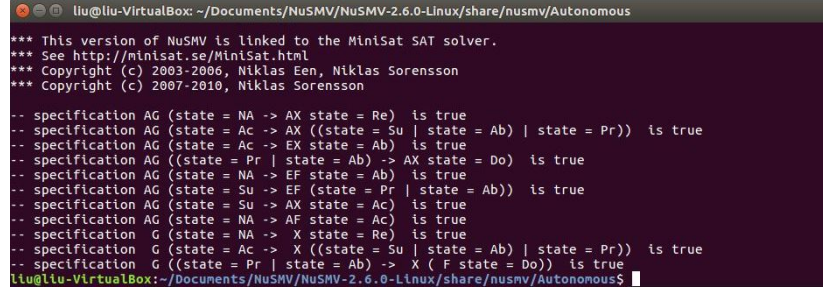
5.1. Verification Results

After combining the LTL, CTL specifications and the transformed NuSMV model, a verifiable NuSMV program has been obtained. The verification results are separated into two types: global behaviour (Fig. 11) and operational behaviour (Figs. 12 and 14). A counterexample of property CTL-S1 is proposed by NuSMV model checker, which is shown in Fig. 12. In this figure, the states transitions for Flight agent and Plane agent are concluded in Table 3. The meaning of codes like b0 can be seen in Fig. 10. In order to make it easier to follow the action in system with a large number of variables, only the values of variables that have changed in the last step are printed in the states of the trace (Cimatti et al. 2002). Thus, there is no value of state 1.3 for Flight agent. The transition between states 1.5 and 1.6 implies Plane agent missed the signal from Flight agent, because Plane agent was at state b4. However, state b4 is not the appropriate state to receive the signal from Flight agent. As a result, some constraints should be imposed on signals sending between these two agents.

The condition of “plane.state = Ready | plane.state = ReadyToFly” is added to transition $b0 \rightarrow b1$, which implies the scheduling signal can be sent to Plane agent, iff Plane agent reaches at state “Ready” or “ReadyToFly”. Similarly, the condition of “plane.state = ReadyToFly” is added to transition $b2 \rightarrow b1$. The former state b3 is deleted. It should be noted that “Ready” means the plane

Table 3: The states transitions for Flight and Plane agents of the counter-example

	state 1.1	state 1.2	state 1.3	state 1.4	state 1.5	state 1.6
flight	a0	a1		a4	a0	a1
plane	b0	b1	b2	b3	b4	b7



```

liu@liu-VirtualBox: ~/Documents/NuSMV/NuSMV-2.6.0-Linux/share/nusmv/Autonomous
*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

-- specification AG (state = NA -> AX state = Re) is true
-- specification AG (state = Ac -> AX ((state = Su | state = Ab) | state = Pr)) is true
-- specification AG (state = Ac -> EX state = Ab) is true
-- specification AG ((state = Pr | state = Ab) -> AX state = Do) is true
-- specification AG (state = NA -> EF state = Ab) is true
-- specification AG (state = Su -> EF (state = Pr | state = Ab)) is true
-- specification AG (state = Su -> AX state = Ac) is true
-- specification AG (state = NA -> AF state = Ac) is true
-- specification G (state = NA -> X state = Re) is true
-- specification G (state = Ac -> X ((state = Su | state = Ab) | state = Pr)) is true
-- specification G ((state = Pr | state = Ab) -> X ( F state = Do)) is true
liu@liu-VirtualBox:~/Documents/NuSMV/NuSMV-2.6.0-Linux/share/nusmv/Autonomous$

```

Figure 11: The results on checking properties of global behaviour of the system

is ready for the first scheduling, since then each ready state of the plane is expressed by “ReadyToFly”. Specification CTL-S1 has passed over NuSMV model checker, which is shown in Fig. 13.

The other properties are all satisfied by the checker. Deadlock property is checked by using commands and interactive mode of the model checker is necessary. So, this property is verified separately.

Therefore, verification results show that the global behaviour satisfies the constraints of the control behaviour. All the properties on liveness, reachability and deadlock are satisfied. Only property CTL -S1 does not pass. The problem is identified as an issue about signals sending between Flight agent and Plane agent.

5.2. Simulation Model Modification

Flight agent has been modified with respect to the correction of the NuSMV model. The original and modified processes are shown in Figs. 15 and 16 respectively. The difference between these two processes is highlighted by red lines. Some basic concepts are introduced, which aims to better understand the process. For example, *downtime* is the difference between *totalRepairTime* and *scheduledInterval*. *scheduledInterval* implies the interval between the time of the arrival and departure for a plane. *totalRepairTime* is only involved all the time used to repair faulty parts, excluding the time of waiting for the existing resources and scheduling. Additionally, *pollingTime* means the time used to check the state of a plane. In terms of service level, *onTimeFlights*, *delayFlights* and *canceledFlights* are defined. One flight is supposed to be canceled if the delayed time from the departure of a plane is more than 4 hours (Liu et al. 2019).

Comparison experiments have been conducted in order to investigate the impacts of the modified process of Flight agent on service level. The run length

```

liu@liu-VirtualBox: ~/Documents/NuSMV/NuSMV-2.6.0-Linux/share/nusmv/A
WARNING *** The model contains PROCESSES or ISAs. ***
WARNING *** The HRC hierarchy will not be usable. ***
-- specification AG !(flight.state = a1 & plane.state = b7) is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  flight.state = a0
  plane.state = b0
  cra.state = c0
  sta.state = d0
  ssa.state = e0
  sca.state = f0
  mainn.state = g0
-> Input: 1.2 <-
  flight.action = Schedule
  plane.action = Reach
  cra.action = UnscheduleMaintenance
  sta.action = ResourceEstimate
  ssa.action = Deliver
  sca.action = InventorySearch
  mainn.action = Process
  _process_selector_ = main
  running = TRUE
  mainn.running = FALSE
  sca.running = FALSE
  ssa.running = FALSE
  sta.running = FALSE
  cra.running = FALSE
  plane.running = FALSE
  flight.running = FALSE
-> State: 1.2 <-
  flight.state = a1
  plane.state = b1
  sta.state = d3
  sca.state = f1
-> Input: 1.3 <-
  plane.action = Request
  sta.action = Deliver
  ssa.action = StrategyAnalyze
-> State: 1.3 <-
  plane.state = b2
  sca.state = f2
  mainn.state = g1
-> Input: 1.4 <-
  plane.action = Receive
  sca.action = Found
-> State: 1.4 <-
  flight.state = a4
  plane.state = b3
  sca.state = f4
  mainn.state = g2
-> Input: 1.5 <-
  flight.action = Null
  plane.action = Reach
  sca.action = Deliver
  mainn.action = Dispatch
-> State: 1.5 <-
  flight.state = a0
  plane.state = b4
  sca.state = f5
  mainn.state = g3
-> Input: 1.6 <-
  flight.action = Schedule
  sca.action = Null
  mainn.action = Delay
-> State: 1.6 <-
  flight.state = a1
  plane.state = b7
  sca.state = f0
-- specification AG (EF plane.state = b9) is true
-- specification AG (EF (plane.state = b4 -> plane.state = b7)) is true
-- specification AG (EF sca.state = f3) is true
-- specification AG (EF mainn.state = g4) is true
-- specification EF sca.state = f3 is true
-- specification EF (ssa.state = e1 -> ssa.state = e4) is true

```

Figure 12: The result on checking general system properties (except for deadlock)

is set to twelve months, which takes around five hours. Fig. 17 shows the simulation results. The data on service level is the average of 10 repetitions'

```

liu@liu-VirtualBox: ~/Documents/NuSMV/NuSMV-2.6.0-Linux/share/nusmv/Autonomous
*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

WARNING *** The model contains PROCESSES or ISAs. ***
WARNING *** The HRC hierarchy will not be usable. ***
-- specification AG !(flight.state = a1 & plane.state = b7) is true
-- specification AG (EF plane.state = b9) is true
-- specification AG (EF (plane.state = b4 -> plane.state = b7)) is true
-- specification AG (EF sca.state = f3) is true
-- specification AG (EF mainn.state = g4) is true
-- specification EF sca.state = f3 is true
-- specification EF (ssa.state = e1 -> ssa.state = e4) is true
liu@liu-VirtualBox:~/Documents/NuSMV/NuSMV-2.6.0-Linux/share/nusmv/Autonomous$

```

Figure 13: The result on the corrected NuSMV model

```

liu@liu-VirtualBox: ~/Documents/NuSMV/NuSMV-2.6.0-Linux/share/nusmv/Autonomous
liu@liu-VirtualBox:~/Documents/NuSMV/NuSMV-2.6.0-Linux/share/nusmv/Autonomous$ NuSMV -int ass.smv
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:36:56 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler
*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

NuSMV > go
WARNING *** Processes are still supported, but deprecated. ***
WARNING *** In the future processes may be no longer supported. ***

WARNING *** The model contains PROCESSES or ISAs. ***
WARNING *** The HRC hierarchy will not be usable. ***
NuSMV > check_fsm

#####
The transition relation is total: No deadlock state exists
#####
NuSMV >

```

Figure 14: The result on checking the deadlock property

data. Warm-up period is set to five days to avoid initialization bias since the simulation system starts with new planes.

From this figure, we can observe that the modified process outperforms the original process in the number of cancelled flights. However, the modified process has failed to surpass the original process in the numbers of delayed and on-time flights. More precisely, the number of delayed flights has increased from 12% to 34% and the number of on-time flights has decreased from 69% to 56%. The disadvantages of the modified process on delayed and on-time flights cannot illustrate the inefficiency of the modification. In fact, it is quite reasonable to obtain this result. As less flights are cancelled in the modified model, it leads to more delayed flights. Delayed flights may result in producing more flights of delay. Therefore, the number of delayed flights has grown. On-time flights could be less for the modified model as well, because delayed flights may cause the “chain reaction” (i.e. one delayed flight maybe causes the next flight delayed). However, if one flight has to be cancelled, it just leads to the increase in the number of cancelled flights but it will start to serve on time in the next time. Therefore, less number of on-time flights is appropriate.

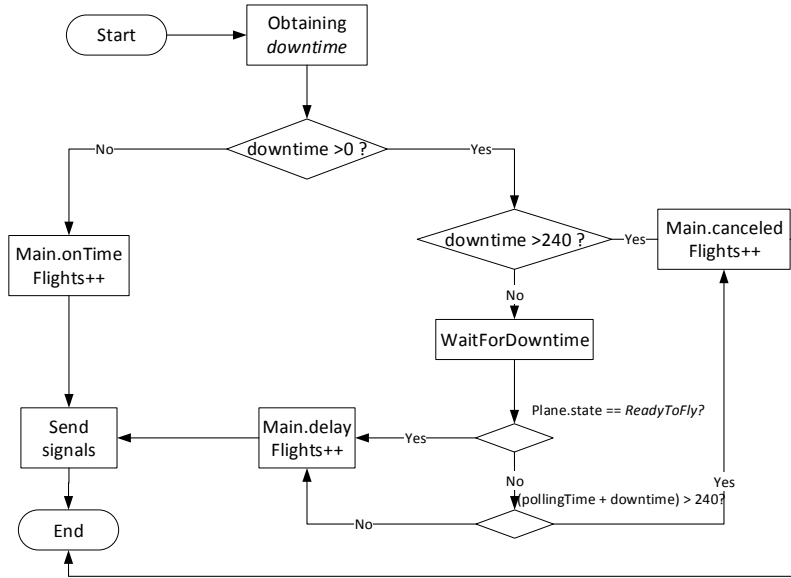


Figure 15: The original process for Flight agent

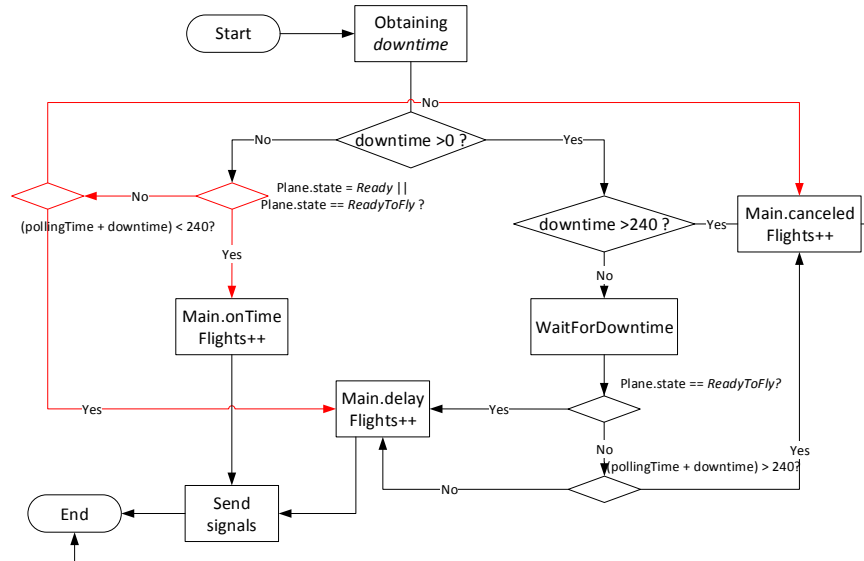


Figure 16: The modified process for Flight agent

Additionally, the efficiency of the modified process can be explained more clearly if the opposite of cancelled flights is considered. Nine percent of cancelled flights have turned to be serviceable flights. Since more flights are

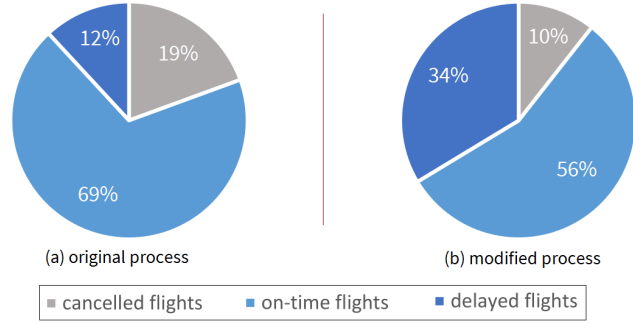


Figure 17: The simulation results on the original and modified models of the ABSS

serviceable, the resource will become more limited. Therefore, it will definitely influence the numbers of on-time and delayed flights.

To conclude, the modification for Flight agent has assured that Plane agent will not miss signals from Flight agent. It has successfully improved the performance for the ABSS in terms of cancelled flights, which illustrates the feasibility of the correction for the NuSMV model of the simulation system.

5.3. Discussion

The results on the simulation experiments demonstrate the efficiency of our method. The approach allows us to not only verify global behaviours but check operational behaviours. It focuses on how to verify the system behaviour from different points of view rather than extend the formal logic, which aims to improve the efficiency of model checking.

The original design of Flight agent (Fig. 4) seems to be rational. There are two ways to send signals to Plane agent. One is to send signals when there is no delay, the other is to receive the delay signal from STA agent, then continuing checking the state of Plane agent until it is ready, finally sending the scheduling signals. We have not noticed that the delay of one flight may have an influence on the departure of next flights that have no maintenance requests. For example, the downtime of one plane has exceeded the time of the interval of its next flight. It means this plane will be demanded to serve for the flight after next even if it has not arrived at the airport of the next flight. In this case, the downtime of this plane for the next flight is equal to zero. If sending scheduling signals only depends on delay signals, this plane will miss the signal for the scheduling for the flight after next. We can not ensure that one plane will depart on time even if it does not need repairing. Therefore, the original design of Flight agent should be improved. Checking the state of Plane agent is definitely necessary before sending scheduling signals.

Our way of verifying the global behaviour is quite similar to the work of Bentahar et al. (2013). However, we have identified the limits of their work. Firstly, their approach is not suitable to verify the system behaviour where transitions are associated with conditions. The detailed discussion can be seen

in the first paragraph of Section 2. Subsequently, alternating Büchi automaton was chosen to express the control behaviour, since it considered both universal choices and existential choices. The core concept of alternating Büchi automaton is the partial transition function (Vardi 1995). The partial function implies that one state can be transited to two states concurrently with one message in the system. However, the verified system has not been involved with messages. If their system is modified into the system of transitions with conditions, the LTL and CTL specification discussed in Section 4.3 for verifying the global behaviour will never pass by NuSMV. So, in our opinion, there is no point in choosing alternating Büchi automaton. Finally, the model transformation was provided in their work, but the messages-related transformation was not discussed. Three fundamental differences between our work and their work are identified. The first one is that we clarify the limit of the way we verify the global behaviour where the transitions with messages are not considered. The second one is that we propose the approach on the verification for the operational behaviour of the system as a complement. The last one is that we provide a complete process for transforming Büchi automaton into Kripke model.

Another advantage of our approach is that it enables us to check the satisfaction of the global behaviour of the ABSS with respect to standards. ABSSs of different domains address different key properties. The difference of properties can be represented by different control behaviours. Thus, a global sense of the satisfaction of the ABSS conforming to standards can be analyzed.

As a result, we propose an approach to verifying the behaviours of the ABSS from the point of view of the system itself. It permits system designers who know systems well but are not experts in analyzing formal logics to better verify the systems of interest.

6. Conclusion and Future Work

In this paper, we discussed about how to verify the behaviours of the ABSS from the point of view of the system itself. System behaviours are composed of global and operational behaviours. The division of system behaviours allows us to investigate whether the system design is satisfactory from both the abstract and the detailed points of view.

The Büchi automaton described system behaviours and the control behaviour, which satisfied the continuous invocation of the plane's service. The meta-transformation between Büchi automaton and Kripke structure (the theory of NuSMV model) was provided, aiming at giving the theoretic support between system behaviours and the corresponding NuSMV model. A mapping between the ABSS and the NuSMV model was then created. LTL and CTL were used to formulate the specifications, in order to verify the conformity of system behaviours. Simulation experiments were carried out to compare the performances of the original and modified processes, which illustrates the feasibility of the proposed approach.

The proposed approach have attempted to check the global behaviour of the ABSS, which lays the foundation for verifying the system behaviour from

the abstract level without concerning any details. It can be recognized as a general way of verifying behaviours of the ABSS implemented by *Anylogic*. This paper gave also a concrete case study of formal methods in practice and demonstrated the efficiency of formal methods in improving system designs.

However, the verification for the global behaviour can not support transitions between states with messages. This is the limit of NuSMV model checker. Hence, in the future, other model checking techniques like SPIN, PRISM and UPPAAL should be investigated in order to provide a more flexible approach to verifying system behaviours. On the other hand, few researchers have concentrated on extracting the global behaviour of the system. the automatic approach of extracting the system global behaviour from the ABSS will be our future work.

References

- Abar, Sameera, Georgios K Theodoropoulos, Pierre Lemarinier, and Gregory MP O'Hare. 2017. "Agent based modelling and simulation tools: a review of the state-of-art software." *Computer Science Review* 24: 13–33.
- Al-Saqqar, Faisal, Jamal Bentahar, Khalid Sultan, Wei Wan, and Ehsan Khosrowshahi Asl. 2015. "Model checking temporal knowledge and commitments in multi-agent systems using reduction." *Simulation Modelling Practice and Theory* 51: 45–68.
- Bentahar, Jamal, Hamdi Yahyaoui, Melissa Kova, and Zakaria Maamar. 2013. "Symbolic model checking composite Web services using operational and control behaviors." *Expert Systems with Applications* 40 (2): 508–522.
- Boniol, Frédéric, and Virginie Wiels. 2014. "The landing gear system case study." In *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, 1–18. Springer.
- Bordini, Rafael H, Michael Fisher, Willem Visser, and Michael Wooldridge. 2006. "Verifying multi-agent programs by model checking." *Autonomous agents and multi-agent systems* 12 (2): 239–256.
- Cavada, Roberto, Alessandro Cimatti, Gavin Keighren, Emanuele Olivetti, Marco Pistor, and Marco Roveri. 2019. "NuSMV 2.5 Tutorial." Accessed 2019-03-1. <http://nusmv.fbk.eu/NuSMV/tutorial/v25/tutorial.pdf>.
- Cimatti, Alessandro, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. 2002. "NuSMV 2: An opensource tool for symbolic model checking." In *International Conference on Computer Aided Verification*, 359–364. Springer. <http://nusmv.fbk.eu/>.
- El Menshawy, Mohamed, Jamal Bentahar, Warda El Kholy, and Amine Laarej. 2018. "Model checking real-time conditional commitment logic using transformation." *Journal of Systems and Software* 138: 189–205.
- Fagin, Ronald, Joseph Y Halpern, Yoram Moses, and Moshe Vardi. 2004. *Reasoning about knowledge*. MIT press.
- Fisher, Michael. 2005. "Temporal development methods for agent-based." *Autonomous Agents and Multi-Agent Systems* 10 (1): 41–66.
- Joo, Jaekoo, Namhun Kim, Richard A Wysk, Ling Rothrock, Young-Jun Son, Yeong-gwang Oh, and Seungho Lee. 2013. "Agent-based simulation of affordance-based human behaviors in emergency evacuation." *Simulation Modelling Practice and Theory* 32: 99–115.

- Keshanchi, Bahman, Alireza Souri, and Nima Jafari Navimipour. 2017. "An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: formal verification, simulation, and statistical testing." *Journal of Systems and Software* 124: 1–21.
- Liu, Yinling, Tao Wang, Haiqing Zhang, and Vincent Cheutet. 2018. "Information Systems Simulation for Performance Evaluation-Application in Aircraft Maintenance." In *IFIP International Conference on Product Lifecycle Management*, 789–799. Springer.
- Liu, Yinling, Tao Wang, Haiqing Zhang, Vincent Cheutet, and Guohua Shen. 2019. "The design and simulation of an autonomous system for aircraft maintenance scheduling." *Computers & Industrial Engineering* 137: 106041.
- Liu, Yutu, Anne H Ngu, and Liang Z Zeng. 2004. "QoS computation and policing in dynamic web service selection." In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, 66–73. ACM.
- Maamar, Zakaria, Quan Z Sheng, Hamdi Yahyaoui, Jamal Bentahar, and Khouloud Boukadi. 2009. "A new approach to model web services' behaviors based on synchronization." In *Advanced Information Networking and Applications Workshops, 2009. WAINA'09. International Conference on*, 43–49. IEEE.
- Macal, Charles M. 2016. "Everything you need to know about agent-based modelling and simulation." *Journal of Simulation* 10 (2): 144–156.
- Macal, Charles M, and Michael J North. 2010. "Tutorial on agent-based modelling and simulation." *Journal of simulation* 4 (3): 151–162.
- Meski, Artur, Wojciech Penczek, Maciej Szreter, Bozena Wozna-Szczesniak, and Andrzej Zbrzezny. 2014. "BDD-versus SAT-based bounded model checking for the existential fragment of linear temporal logic with knowledge: algorithms and their performance." *Autonomous Agents and Multi-Agent Systems* 28 (4): 558–604.
- Navimipour, Nima Jafari, Ahmad Habibizad Navin, Amir Masoud Rahmani, and Mehdi Hosseinzadeh. 2015. "Behavioral modeling and automated verification of a Cloud-based framework to share the knowledge and skills of human resources." *Computers in Industry* 68: 65–77.
- Pnueli, Amir. 1977. "The temporal logic of programs." In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, 46–57. IEEE.
- Raimondi, Franco. 2006. "Model checking multi-agent systems." PhD diss., University of London.
- Serrano-Hernandez, Adrian, Javier Faulin, Patrick Hirsch, and Christian Fikar. 2018. "Agent-based simulation for horizontal cooperation in logistics and transportation: From the individual to the grand coalition." *Simulation Modelling Practice and Theory* 85: 47–59.

- Sheng, Quan Z, Zakaria Maamar, Hamdi Yahyaoui, Jamal Bentahar, and Khouloud Boukadi. 2010. "Separating operational and control behaviors: A new approach to Web services modeling." *IEEE Internet Computing* 14 (3): 68–76.
- Song, Wen, Hui Xi, Donghun Kang, and Jie Zhang. 2018. "An agent-based simulation system for multi-project scheduling under uncertainty." *Simulation Modelling Practice and Theory* 86: 187–203.
- Souri, Alireza, and Nima Jafari Navimipour. 2014. "Behavioral modeling and formal verification of a resource discovery approach in Grid computing." *Expert Systems with Applications* 41 (8): 3831–3849.
- Vardi, Moshe Y. 1995. "Alternating automata and program verification." In *Computer Science Today*, 471–485. Springer.
- Yahyaoui, Hamdi, Zakaria Maamar, and Khouloud Boukadi. 2010. "A framework to coordinate web services in composition scenarios." *International Journal of Web and Grid Services* 6 (2): 95–123.