EasyChair Preprint

№ 4943

Verifying UML-RT Model of a Conveyor Belt System with nuXmv Model Checker

Duy Hieu Vo, Minh Gia Huy Cao, Nhat Nam Pham,
Tong Ngoc Dang, Sneha Sahu and Ruth Schorr

January 30, 2021

Verifying UML-RT Model of a Conveyor Belt System with nuXmv model checker

Hieu Vo, Huy Cao, Nam Pham, Ngoc Dang, Sneha Sahu[0000−0002−6143−6153], and Ruth Schorr

Frankfurt University of Applied Sciences, 60318 Frankfurt am Main, Germany

Abstract. Conveyor belt system is a convenient and reliable system used in automated distribution of goods and merchandise. The effectiveness of merging items from multiple induction lines onto one main line is one of the important aspects in implementing such a labour-saving system. One way of ensuring correctness and reliability for such a system is by introducing model-checking at design level. In this paper, a simple conveyor belt system is first modelled using Papyrus-RT, which is an implementation of the UML-RT modelling language. Then, based on a set of translation rules, this model is mechanically translated into the nuXmv symbolic model checker for property verification. Counter examples are presented in case of incorrect properties or failures inside the model, along with a log trace comparison indicating time-dependent inconsistencies between nuXmv and Papyrus-RT models.

Keywords: Conveyor Belt System · UML-RT · Papyrus-RT · Model Translation · Symbolic Model-Checking · nuXmv

1 Introduction

Conveyor belts have been playing a huge role in the industry since they first appeared, with the capability of greatly saving manpower. Modelling of the conveyor belt with real-world requirements is necessary for understanding its operation and help improvise performance and efficiency. In this research we have considered the one with merge-configuration that helps to serve multiple check-in counters at an airport. An industrial-grade modelling tool called Papyrus-RT [5], based on the idea of Model Driven Engineering (MDE) [4], is used for the system design. UML-RT [9] design models that describe the model behaviour in this tool, serve as the key artifact for auto generation of an executable CDT project [5]. However, the semantic correctness of the model completely relies on the system designer and testing at later stages. Hence, the UML-RT system model is translated into a suitable model checker so that a set of pre-defined formal properties could be used to verify the semantics of the system. The remainder of this paper is organized as follows. In the next chapter we briefly explain the translation approach, followed by the description of the conveyor belt system, its corresponding translation and verification. The paper is concluded with a short discussion on further research.

2 Methodology

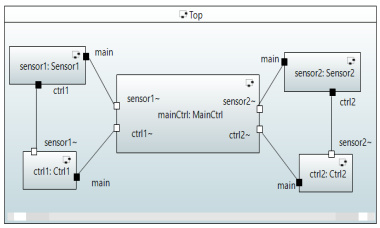
The UML-RT system model [5] consists of multiple components called *capsules* connected through *ports* for communication. Communications are trigger-based and the system semantics are described with the help of state diagrams for each capsule. Papyrus-RT provides default protocols called 'log' and 'timer' and also supports user-defined custom protocols. These protocols are basically a set of rules defining the types of incoming and outgoing messages for individual ports. In [2], the researchers present a translation for a subset of the UML-RT state diagrams into Promela(SPIN) [6]. The work however excludes model constructs such as hierarchical states, signal payloads, multiplicity of model elements, pseudo-states and guard conditions, which are considered in [10]. The paper presents an approach for translation into an equivalent nuXmv [3] model in two steps – *First*, State Diagrams to Finite State Machines(FSMs) [8] and *Second*, UML-RT components and FSMs into SMV model. There are 2 kinds of states allowed in the UML-RT State Diagrams, which in the FSM are treated as the same kind, with different kinds of guard conditions. Also, some new wait states are introduced in the FSMs to clearly distinguish the receipt and send of messages which otherwise appear to occur at the same instant in the UML-RT State Diagram. Any multi-level state hierarchy is removed to achieve the same level for all states. As for the non-trivial language dependent code snippets, abstraction needs to be used to achieve similar behaviours in nuXmv. In the second phase, UML-RT capsules and protocols along with the FSM are translated into corresponding nuXmv Modules; the 'Top' capsule is translated as the 'Main Module' in nuXmv; Asynchronous communication is handled by introducing a specific controller logic in the 'Main Module'; Capsule Ports are translated into Module VARiables; Connections are fed as Module input parameters; Transitions, guards and actions from FSMs are translated under the NEXT and TRANS sections of the corresponding Modules.

3 Case Study: Conveyor Belt System

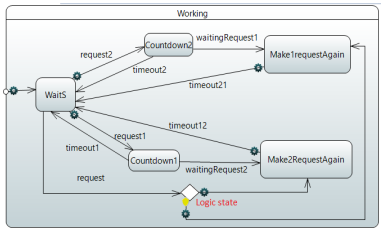
3.1 Model Description

The system contains one main line and two induction lines as in this research [7]. For requirements, the system must ensure that no induction line is eventually blocked. Packages are sequentially dropped onto main line only after receiving confirmation signals. A minimum distance must be maintained between packages and there must be no collision of baggage on the main line. For simplicity, the times for the reserved slot to move from the top of main line to loading positions of two induction lines are strictly defined as 5 and 10 seconds.

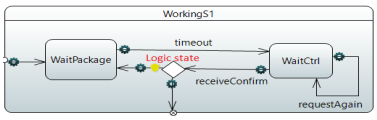
Figure 1 shows the Papyrus-RT model of the system. The Top capsule (Fig. 1a) synchronizes the work of overall system through two protocols. The individual capsules work in following order:



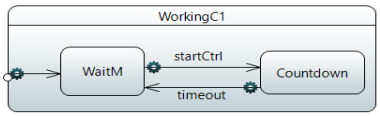
(a) Top Capsule representing overall system



(b) Main controller state diagram



(c) State diagram for each sensor



(d) State diagram for each controller

Fig. 1: Papyrus-RT models.

1. **Sensor**(Fig. 1c): It simulates the arrival of baggage. In *WaitPackage*, after a random time, it sends request to the Main Controller (MC). In *WaitCtrl* state, if it receives rejection from MC, it requests again. If it receives confirmation from Controller (Ctrl), sensor proceeds to *logic state* to check the end condition of the simulation.
2. **Main Controller**(Fig. 1b): It orchestrates the dropping of baggage. In *WaitS*, if MC receives a single request, it issues command to controller and transfers to *Countdown1/2*. If 2 signals come simultaneously, *logic state* is the next state which chooses one request to process in round robin fashion and goes to *MakeRequestAgain* to reject the other request. In *Countdown1/2*, MC waits 5 seconds for the next reserved slot to come and goes back to *WaitS*. If another signal comes before timeout, it also comes to *MakeRequestAgain*.
3. **Controller**(Fig. 1d): It sends confirmation to sensor. In *WaitM*, if it receives signal from MC, it goes to *Countdown* and starts counting down 5 or 10 seconds (controller 1 or 2, respectively). After timeout, it confirms with sensor and goes back to *WaitM* .

3.2 Model Translation

After a runnable Papyrus-RT model is constructed, NUXMV symbolic model checker [3] is utilized to verify this model against its requirements. The Papyrus-RT model must first be translated into NUXMV language. According to the model translation rules [10], each Papyrus-RT protocol and capsule along with the respective state diagrams is mechanically converted into a corresponding NUXMV module.

The NUXMV translations for the capsules - **top, sensor and controller,** did not require any special handling. Whereas, for the **main controller** capsule, two new waiting states – *wait_*Down1* and *wait_*Down2* are inserted in order to split up the receive and send events [10]. This is further supported by creating

copies of the remaining outgoing transitions of the actual state into this newly added states, as shown in Fig. 2.

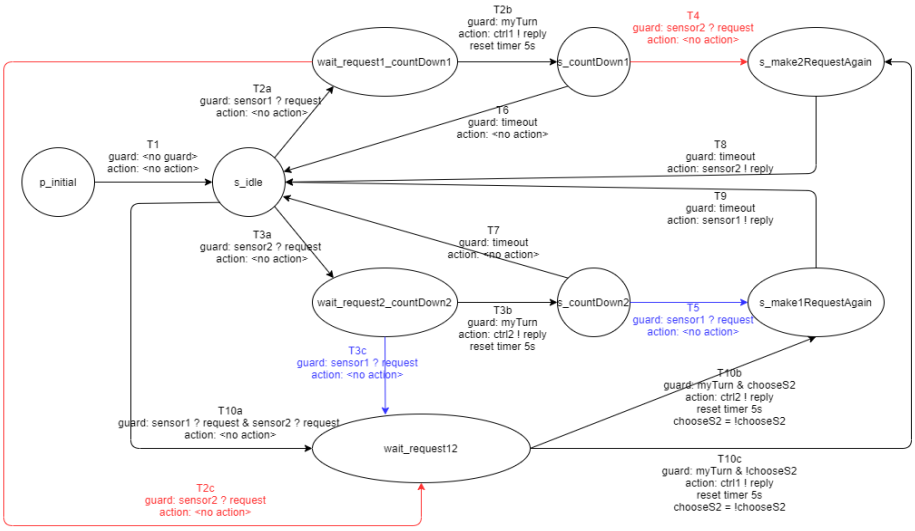


Fig. 2: Replicated outgoing transitions for the new wait states.

Although, in the simulation trace, the sequence of events were observed to be the same in case of both Papyrus-RT (Fig. 3a) and NUXMV (Fig. 3b), there was some time-dependent lagging in the NUXMV log trace. In the Papyrus-RT

```
Starting sensor 1 at 0
Starting sensor 2 at 0
Line 2: Package 1 at: 1
Line 2 waits for loading slot
Line 1: Package 1 at: 3
Line 1 waits for loading slot
Line 2 load successfully at 11
Line 2 load successfully after 10
Line 1 load successfully at 11
Line 1 load successfully after 8
```

(a) Papyrus-RT

```
-> State: 1.2 <-
sensor1 . timerResetValue = 3
sensor2 . timerResetValue = 1
...
-> State: 1.4 <-
sensor1 . timer . time = 2
sensor2 . timer . time = 0
sensor2 . timer . timeout = TRUE
-> State: 1.5 <-
sensor1 . timer . time = 1
sensor2 . port . mainController . out
. request = TRUE
-> State: 1.6 <-
sensor1 . timer . time = 0
sensor1 . timer . timeout = TRUE
mainCtrl . state =
wait_request2_countdown2
-> State: 1.7 <-
sensor1 . port . mainController . out .
request = TRUE
mainCtrl . state =
wait_request2_countdown2
-> State: 1.8 <-
mainCtrl . state = wait_request12
...
-> State: 1.10 <-
mainCtrl . state =
s_make1RequestAgain
mainCtrl . port_controller2 . in_reply
= TRUE
...
-> State: 1.17 <-
mainCtrl . port_sensor1 . in_reply =
TRUE
...
-> State: 1.20 <-
sensor1 . port_mainController .
out_request = TRUE
...
-> State: 1.23 <-
ctrl2 . port_sensor . in_confirm =
TRUE
mainCtrl . port_controller1 . in_reply
= TRUE
...
-> State: 1.31 <-
ctrl1 . port_sensor . in_confirm =
TRUE
```

(b) NUXMV

Fig. 3: Log traces

model, the time in the log trace is in seconds. However, the actual time step of the simulation is as small as the processing speed of the computer. Therefore, when a request comes, it is processed almost immediately. The NUXMV model, on the other hand, proceeds in discrete simulation steps. Numerous NUXMV steps can be done in Papyrus-RT in term of nanoseconds and considered as instantly.

3.3 Property Verification

To verify the model, informal properties are derived from general requirements and then formalized with Linear Temporal Logics (LTL) [1]. For instance, based on the requirement that no induction line is eventually blocked and no collision should occur on main line, the property in Fig. 4 is created.

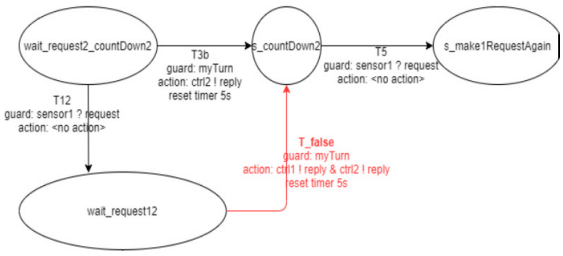
Informal properties	Formalized properties
Whenever main controller receives requests from 2 sensors at the same time, it will eventually send a message back to one sensor and send command to the controller of the other sensor.	LTLSPEC G(mainCtrl.state=wait_request12 -> F((mainCtrl.port_controller2.in_reply & F mainCtrl.port_sensor1.in_reply) | (mainCtrl.port_controller1.in_reply & F mainCtrl.port_sensor2.in_reply))) LTLSPEC G(!(mainCtrl.port_controller2.in_reply & mainCtrl.port_controller1.in_reply

Fig. 4: Informal property and its formalization

The formalized properties are then verified against the translated NUXMV model and can be used to detect flaws in the original Papyrus-RT model. For instance, in the main controller, when two requests come at the same time, both are accepted instead of only one (Fig. 5b). This design flaw will result in collision of packages from two induction lines since one free loading slot on the main line is assigned to two different induction lines.

```
...
-> State: 1.7 <-
mainCtrl.state = wait_request12
-> State: 1.8 <-
mainCtrl.port_controller1.in_reply = TRUE
mainCtrl.port_controller2.in_reply = TRUE
-> State: 1.9 <-
ctrl1.state = s_countDown
ctrl1.timerResetValue = 5
ctrl1.timerResetFlag = TRUE
ctrl2.state = s_countDown
ctrl2.timerResetValue = 10
ctrl2.timerResetFlag = TRUE
...
-> State: 1.15 <-
ctrl1.portTimer.time = 0
ctrl2.portTimer.time = 5
ctrl1.portTimer.timeout = TRUE
-> State: 1.16 <-
ctrl1.port_sensor.in_confirm = TRUE
...
```

(a) Log trace of counter example



(b) Incorrect model where collision of packages occurs

Fig. 5: Flaw introduced into the model and how NUXMV detects it

Running verification again against the formalized properties in Fig. 4, the model checker is able to detect the violation. The counter example in Fig. 5a shows that main controller sends commands to both controllers at the same state 1.8. Subsequent states in the trace also evince the collision of packages on the main line. At state 1.9, both controllers start their timers at the same time. This brings about the situation where controller 2 sends confirmation to sensor 2 exactly 5 time steps, after controller 1 confirms to its sensor the success of dropping the package on the main line. According to the system's requirements, package from the induction line 1 passes by the dropping point of line 2 in 5

time steps and hence will collide with the newly dropped package by controller 2.

4 Discussion

In this case study of a conveyor belt, the system is first modelled using Papyrus-RT. Then using a given set of rules, the model is mechanically translated into nuXmv language for verification of the formalized system properties. This not only helped to validate the system requirements, but also detect hidden system flaws through counter examples.

As further work, the model of the system could be enhanced to include more induction lines and more sophisticated merging algorithm to increase the efficiency and throughput of the system. The time lapse in case of the UML-RT model is real-time in terms of seconds, whereas in the translated nuXmv model, this is simply a step-counter. This could be improvised by exploring the use of clock and Timed-Automata in nuXmv. The translation could also be further enhanced to include a larger domain of UML-RT components including non-integral data types and multiple payloads along the communication channels. An automation of the improvised translation approach, thereby integrating model driven engineering with model-checking, could be a very good contribution towards formal verification of Real-Time Embedded Systems, where interaction between the various components plays a significant role.

References

1. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)
2. Carlsson, M.G., Johansson, L.G.: Formal verification of UML-RT capsules using model checking. Master's thesis, Chalmers University of Technology (2009)
3. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuXmv Symbolic Model Checker. In: International Conference on Computer Aided Verification (CAV). pp. 334–342 (2014)
4. Favre, J.M.: Towards a basic theory to model model driven engineering. In: 3rd Workshop in Software Model Engineering, WiSME. pp. 262–271. Citeseer (2004)
5. Hili, N., Dingel, J., Beaulieu, A.: Modelling and code generation for real-time embedded systems with UML-RT and papyrus-RT. In: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C). pp. 509–510. IEEE (2017)
6. Holzmann, G.J.: The SPIN model checker: Primer and reference manual, vol. 1003. Addison-Wesley Reading (2004)
7. Johnstone, M., Creighton, D., Nahavandi, S.: Simulation-based baggage handling system merge analysis. Simulation Modelling Practice and Theory **53**, 45–59 (2015)
8. Koshy, T.: Finite-State-Machines. In: Discrete mathematics with applications. pp. 771–802. Elsevier (2004)
9. Posse, E., Dingel, J.: An executable formal semantics for UML-RT. Software & Systems Modeling **15**(1), 179–217 (2016)
10. Sahu, S., Schorr, R., Medina-Bulo, I., Wagner, M.: Model Translation from Papyrus-RT into the nuXmv Model Checker. In: Software Engg. and Formal Methods. SEFM 2020 Collocated Workshops. pp. 3–20. Springer Intl. Publishing (2021)