Politecnico di Milano
2020/2021

Formal Methods for Concurrent and Real-Time Systems

Model Checking of Warehouse Robotics

Homework Project Report

Authors
Giacomini Davide
Giarduz Andrea
Grosso Veronica

Instructors
Prof. San Pietro Pierluigi
Dr. Lestingi Livia

Contents

List of Figures **I**

List of Tables **I**

Abstract **1**

1 Introduction **1**

2 Component Description **2**
2.1 Environment and System Declarations 2
2.2 Task Queue 3
2.3 Robots 3
2.4 Human Operator 4
2.5 Design Choices 4

3 Properties **5**
3.1 Pods unavailable 5
3.2 Overflowing Task Queue FIFO 6
3.2.1 First Scenario 7
3.2.2 Second Scenario 8
3.2.3 Third Scenario 9

4 Conclusions **10**

References **10**

List of Figures

1 Example of a grid layout 2
2 12x8 grid layout 9

List of Tables

7 Statistical parameters of UPPAAL 5
8 Pods unavailable results 6
9 Scenario one 7
10 Scenario two 8
11 Scenario three 9

Abstract

The development of cutting-edge technologies in automated warehouses has taken outstanding steps in the last few years. This fast-paced environment requires a high level of reliability, in terms of speed and interaction between the many system components. Formal verification plays an essential role in ensuring an error-free environment, by analysing synthetic models, that capture the main features of the real-world problem.
The Model Checking of Warehouse Robotics project employs UPPAAL [2] to model a simplified automated warehouse environment. The goal is to verify some significant properties in different scenarios, in order to show the correctness of the model.

1 Introduction

The Model Checking of Warehouse Robotics project is divided in two main focus areas: first, the efforts are placed in modelling the environment; then, the target becomes the verification of significant properties, applied to the aforementioned model.

The goal is the handling of multiple tasks: each task contains the information about a specific item (placed in a pod inside the warehouse) that needs to be delivered. In particular, the high level model consists in a set of three different components: the human, the robots and the task queue generator. The environment where these components interact is a rectangular grid (i.e., the warehouse floor plan). It is possible to identify on the map an entry point, a delivery point and a set of pods.

Entry point	It is the initial slot where all the robots are placed at the beginning of the simulation.
Delivery point	It is the slot where the items need to be delivered.
Pod	It includes some items and it can be moved by the robots.

Specifically, these are the main features of the components:

Human	The job of the operator is to pick up an item, when a robot comes into the human working station with its item. The operator is kept by default at the edge of the system.
Robot	It is in charge of the delivery of a specific pod to a human, according to what is specified in the relative task.
Task queue	It periodically generates new tasks and places them into a First-In-First-Out (FIFO) queue.

Non-deterministic delays in the actions performed by the components have been included in order to create a model closer to a real-world situation, using UPPAAL SMC.

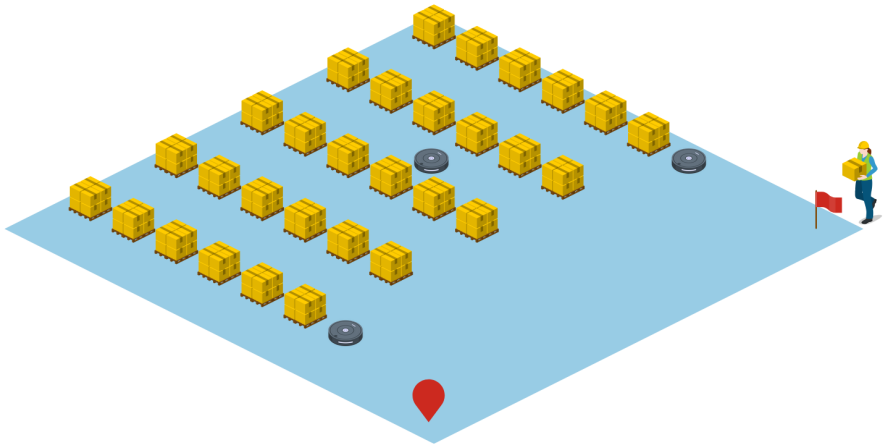

Figure 1: Example of a grid layout

2 Component Description

The detailed description of the system components is presented in this section. It is comprehensive of the mandatory features and behavioural aspects, the simplifying assumptions and the rationale behind the adopted choices.

2.1 Environment and System Declarations

The project declarations contain the auxiliary functions and variables to facilitate the implementation of the model. For instance, functions such as `stdNormal()` and `Normal(mean, stdDev)` are essential to produce delays that follow a Normal Distribution [1].

Moreover, this section features the constants and the data structures. The most important characteristics are:

Grid The warehouse floor plan has been represented by a parametric N × M matrix. The matrix choice easily allows to compute offsets and distances between grid slots. Furthermore it has an intuitive visual prospect.
The grid layout is flexible and can be changed by setting the N, M values, the pods positions, as well as the entry and delivery points. In Figure 1 it is possible to see an example of a grid layout.

Pods The pods id, position (x and y) and status are saved in a matrix `pods[PODS_N][3]`, where `PODS_N` is the total number of pods. The pods can either be free or assigned to a task. This data structure allows an easy access to both the position and status of pods by their id.

Queue It is a FIFO array, of size `MAX_T`, containing the pending tasks. The parameter `MAX_T` represents the maximum queue size. Together with the `current_length_queue` variable, it is possible to model a queue variable in length, despite UPPAAL not providing dynamic memory management.

2.2 Task Queue

The Task Queue template models the generation and dispatching of tasks. Since this is the first entity to start the computation, it initializes the grid, the pods and the queue.

Normal Execution

`task_generation`	The Task Queue generates a new task and places it in the FIFO queue. The generation happens with a variable delay `task_delay`, computed every time following a Normal Distribution.
`task_sent`	The timed automaton synchronizes with an idle robot on the first item of the task queue. It sends the information about the pod id, matched with the task, to the robot. After that, it removes the assigned task from the queue. This is useful because it avoids that multiple robots process the same task.

Error States

`Error_maxSizeReached`	This error state is reached when the task queue is full. The newly generated tasks would be lost, so the timed automaton stops its behaviour.
`Err_podsUnavailable`	The pods unavailable error state happens whenever the number of tasks overcomes the number of available pods. In particular, the new tasks cannot be associated with any pod, creating an undesired situation.

2.3 Robots

The Robot template is the most complex component in the system, since it handles the task retrieval, the robot movement algorithm, the pod delivery and the returning of the pod to its original position. In order to do so, it synchronizes both with the Task Queue and the Human Operator.

The key feature in this automaton is that the states represent ongoing actions (e.g., reaching a pod, going to the delivery point, etc.), that are gradually executed with the self-loop transitions. This way, the model is kept as simple as possible, by avoiding to insert intermediate states.
Transition between different states, on the other hand, are related to different phases of the robot activity (e.g., the pod has been delivered, so the destination of the robot changes). Thus, the model has a constant structure. The idle position, `start`, features an exponential delay, which could represent a non-deterministic robot behaviour in the processing before taking a new task.

2.4 Human Operator

The Human Operator is modelled as a two-state timed automaton, to make it as simple as possible. As soon as a robot arrives in the delivery point with a pod, it synchronizes with the operator. The human picks up the right item from the pod. It takes a delay `human_delay`, modelled by a Normal Distribution, for the human to finish the action. When the job is done, it notifies the robot that it is free to go.

2.5 Design Choices

The main design choices and assumptions for this project can be summarized into four main areas:

Robot movement In order to keep the verification times short, while using an efficient movement algorithm, we supposed the existence of a *robot highway* on the right side of the map, containing the entry and delivery points. The algorithm works by choosing, if possible, the tile with the minimum Manhattan distance to the destination. We implemented the Manhattan distance as the robot is not able to move diagonally. The preferred direction, at this point, is moving towards the *robot highway*, hence the assumption about its existence. This way, robots are less prone to routing congestion. If possible, the robot tries not to move to the tile it previously occupied. This avoids starving conditions where robots keep performing the same movements over and over again. Because of these design choices, the algorithm works best with horizontal alternated parallel lanes of pods.

States number The target of the modelling phase for each template was to keep the number of states low, thus allowing a faster verification and a better readability.

Channel policy The UPPAAL SMC extension requires all channels to be `broadcast`. In our model, this is not a problem, since communication is ensured to be one-to-one. The only relevant point where this could have been an issue is the `free_robot` channel, as the others have only one possible receiver (either the Task Queue or the Human Operator). In spite of this, only one robot can occupy the delivery point, hence only the rightful synchronization will be activated.

New task selection A reduction of complexity also took place in the policy of new task selection. A number $n \in [0, \texttt{PODS\_N}]$ is randomly chosen by the tool. In order to find the first available pod, we compute $m = n \mod \texttt{free\_pods\_number()}$, and then we search the m-th free pod in the pods status array. This ensures to always find a free pod, no matter the value of n chosen by UPPAAL.

3 Properties

In this section we are going to describe the properties that we have verified over the system to ensure its correctness and to analyse its behaviour. Unless otherwise specified, we ran our verifications with a time bounded to 5000 time units, in order to emulate a realistic environment when fully operational.

We have reported two verifications, as we considered them crucial for analysing the behaviour of our system. The first one, described at section 3.1, has been performed with the aim of investigating the probability of running out of pods available in the grid. The second one aims at analysing the probabilities of running out of space in the task queue, during an execution.

During the verification phase we have used the default statistical parameters of Uppaal. It is reasonable to consider a confidence of the 95% for verification purposes, as well as the other values already present by default inside the tool, since a simplified model would not need higher statistical accuracy. The parameters with the respective values are reported in the Table 7.

Parameter	Value
$\pm\delta$	0.01
β	0.05
α	0.05
ϵ	0.05
μ_0	0.9
μ_1	1.1
Trace resolution	4096
Discretization step	0.01

Table 7: Statistical parameters of Uppaal

3.1 Pods unavailable

This property has been generally analysed, without entering in the details of each single configuration.

We analysed two cases and observed the reaction of the system in each of them.

Considering that the pods could become unavailable when

$$\text{PODS\_NUMBER} < \text{QUEUE\_LENGTH} + \text{ROBOTS\_NUMBER} \tag{1}$$

we considered a case in which this formula holds, and another one in which it does not. By modifying the grid layout or the grid dimensions, the number of pods can vary. Hence, we changed the grid dimensions accordingly with our necessities.

To verify this property, we checked if the template `Task_Queue` eventually enters the state *task_queue.pods_unavailable*. The formula can be found in the tab *Verifier* of Uppaal. The equivalent CTL formula is:

$$\forall\Diamond \textit{task\_queue.pods\_unavailable} \tag{2}$$

In the first case, we used a `6x6` grid. This configuration respects the equation 1. As shown in Table 8, the results coincide with what was expected. Indeed, it can be considered almost sure that the state *task_queue.pods_unavailable* will eventually be reached.

In the second case, we used a `10x10`, thus violating the equation 1. In the Table 8 we can see that now, as we expected, the state will be reached with a very small probability. In fact, it should be impossible to reach the state *task_queue.pods_unavailable* when the equation is not respected.

	First configuration	Second configuration
Grid dimension	`6x6`	`10x10`
Number of runs	54	29
$prob \in$	$[90.11\%, 99.95\%]$	$[0\%, 0.098\%]$

Table 8: Pods unavailable results

3.2 Overflowing Task Queue FIFO

This is the most important property to verify, in order to understand if the system reacts in a proper way to several configurations. The system should sustain a high number of tasks generated during the execution.

In this section we will stress the system by changing the delays and the grid configuration. We will also try adding some robots into the system, so that we will be able to see the main differences, comparing them and understanding what are the trade-off to take into account. In particular, the parameters which I will refer to are listed here:

- μ_T: the average delay of the task generation
- σ_T: the standard deviation of the task generation
- μ_H: the average delay when a human picks up an item
- σ_H: the standard deviation when a human picks up an item
- K: the *deterministic* units of time that passes between each move of the robot
- λ: the delay with which the robot claims a task. Note that this parameter has not been changed in any configuration, as we considered $\lambda = 1$ a realistic situation: we want the robots to be reactive.

We analysed three different scenarios. In the first one, we analysed the behaviour of the system with a `10x10` grid and 3 robots. In the second one, we increased the number of robots to 6, maintaining the same grid dimension. Finally, we observed how the system reacts to a `12x8` grid. We also considered the length of the queue to be always equal to 20 maximum tasks.

For all the scenarios we used the same query, that can be found in the tab *Verifier* of UPPAAL. It is equivalent to the CTL formula:

$$\forall\Diamond task\_queue.max\_size\_reached \tag{3}$$

3.2.1 First Scenario

The first scenario is the simplest one. In this case, there are 3 robots and the dimension of the grid is `10x10`.

Given that three robots are not able to handle a large number of tasks, we firstly chose to give a high average time for the task generation delay (μ_T) while maintaining a general realistic behaviour between the robots and the human (μ_H). We assumed that the human is slower than the robots and that it has a significantly high standard deviation (σ_H). We also assumed the task generation to be quite constant, keeping a small standard deviation (σ_T), in order to avoid creating a too floating situation. All these parameters are collected in the scenario *Scenario 1.a* of the Table 9.

The aforementioned assumptions could bring to the conclusion that the robots should be able to handle the tasks generated quite easily. However, the verification results show that there is about the 30% of probability that the queue will reach the maximum size.

As a consequence, we decreased the task generation frequency. Our goal is to understand what is, in this configuration, the maximum frequency that the system can handle. In this scenario we focused on the average delay time μ_T and we changed the other parameters only in order to give a realistic behaviour to the overall system.

We used the values indicated at Table 9 under *Scenario 1.b* and we considered the resulting probabilistic interval sufficient for considering that the system can handle at least $\mu_T = 150$.

Finally, we tried stressing the system in order to understand when it can no more handle the situation. We did it in three different ways:

- Firstly, we increased K, with the aim of delaying the robots. It was enough to double the delay in order to reach the maximum probabilistic range as shown at Table 9 under *Scen 1.c*.

`10x10`	Scen 1.a	Scen 1.b	Scen 1.c	Scen 1.d	Scen 1.e
μ_T	40	150	40	30	30
σ_T	5	10	5	5	5
μ_H	3	6	6	3	6
σ_H	2	4	4	2	4
K	1	1	2	1	1
λ	1	1	1	1	1
$p[\%] \in$	$[24.8, 34.8]$	$[4.14, 14.1]$	$[90.2, 100]$	$[76.4, 86.4]$	$[90.2, 100]$

Table 9: Scenario one

- Subsequently, we decreased the task generation delay. At Table 9 under *Scen 1.d*, we can see that, just decreasing the generation delay by 10, the probabilistic range increased of about 50% with respect to the *Scen 1.a*.
- Finally, we modified the previous analysis. We kept the same values, except for delaying the human (Table 9, *Scen 1.e*). It is enough to delay the human few time units in order to reach the maximum possible range.

In conclusion, we would like to underline the main characteristics identified in the first scenario. We can notice that, in order to stress the system, decreasing the task generation delay by a only few time units is enough to have the system unable to handle the configuration. The same can be said for increasing the human delay.

On the other hand, the system begins to behave in a smooth way only when we set the task generation delay to higher values. We will better understand in section 3.2.2 that this behaviour is typical when there are few robots in the system.

3.2.2 Second Scenario

In this scenario, we will analyse a configurations with six robots and a `10x10` grid.

The intuitive reasoning could be that the more robots there are in the system, the smoother it will be in handling a non trivial number of tasks. Moreover, it should be improbable for the robots to find 'traffic' in the map, as they are a small proportion of the total number of available cells.

However, the results of the verification are worse than we expected. In order to compare the two scenarios, we used the same values of the *Scen 1.a* at Table 9. The resulting probability is shown at Table 10 at *Scen 2.a*. We can observe that the probability of filling up the queue is 35% higher than the one of *Scen 1.a*.

Subsequently, we applied the same reasoning of section 3.2.1, as presented in *Scen 2.b* and *Scen 2.c* of Table 10. Comparing *Scen 2.b* with *Scen 1.b* underlines that the resulting probability of the former is 15% higher than the probability of the latter. However, we can notice that the gap between the two configurations has decreased from 30% to 15%. As we mentioned at the end of section 3.2.1, increasing the number of robots helps the

`10x10`	Scen 2.a	Scen 2.b	Scen 2.c
μ_T	40	150	200
σ_T	5	10	10
μ_H	3	6	6
σ_H	2	4	4
K	1	1	1
λ	1	1	1
$p[\%] \in$	$[60.2, 70.2]$	$[20.0, 30.0]$	$[6.19, 16.2]$

Table 10: Scenario two

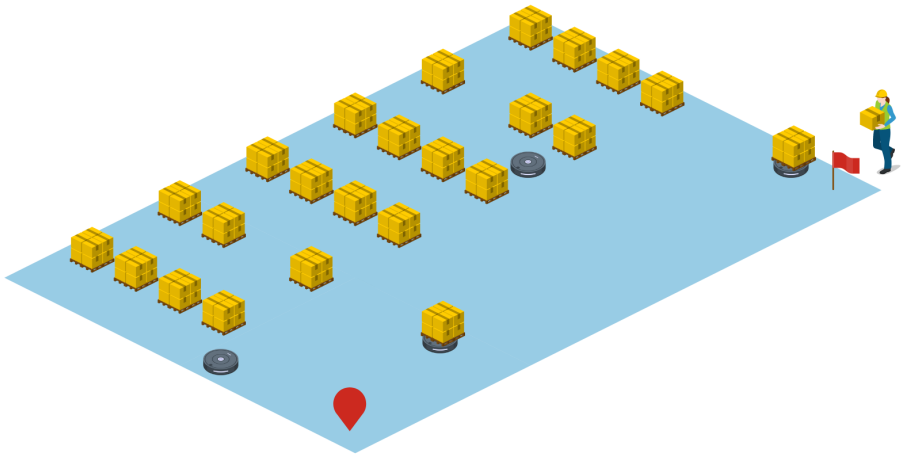

Figure 2: 12x8 grid layout

system in handling a greater number of tasks.

Finally, we tried finding the maximum frequency that allows us considering the system stable and smooth, as we did in section 3.2.1. *Scen 2.c* shows the results of a verification with $\mu_T = 200$. We can consider this result acceptable from the point of view of the smoothness of the system.

In conclusion, we can observe that there is a trade-off to be taken into account. More robots are able to handle more tasks in a more efficient way. However, depending on the grid configuration, they could obstruct each other and worsen the overall efficiency.

3.2.3 Third Scenario

The last scenario that we would like to present is less specific. We stressed an environment with a `12x8` grid and 4 robots, as shown in Figure 2, in order to analyse the behaviour of the system in a rectangular grid, rather than only square ones.

Here we focused more on the system reactions when changing the standard deviations (σ_H and σ_T) and the human average delay time (μ_H).

`12x8`	Scen 3.a	Scen 3.b	Scen 3.c	Scen 3.d
μ_T	40	40	40	40
σ_T	5	80	5	5
μ_H	3	3	6	6
σ_H	2	2	2	15
K	1	1	1	1
λ	1	1	1	1
$p[\%] \in$	$[12.7, 29.1]$	$[7.87, 22.4]$	$[16.9, 34.7]$	$[27.6, 47.2]$

Table 11: Scenario three

The relevant results are collected at Table 11. We began analysing the system in a realistic environment (*Scen 3.a*) and then we changed the relevant parameters. We isolated the analysis for each parameter, changing only one value at a time, in order to ease the behavioural evaluation.

In the first place, we analysed the system reaction in a standard and realistic situation, as it is shown at *Scen 3.a*. The resulting probabilistic interval is quite low, as it floats around the 20%, but it is not sufficient for considering the system smooth in this configuration.

We subsequently stressed the environment in three different ways:

- Firstly, we increased σ_T. We tried several values to see how the system reacts to them, and we understood that the system's probabilities are strongly affected when the standard deviation is a lot greater than the average (in this case, μ_T). *Scen 3.b* at Table 11 highlights this behaviour.
- Secondly, we increased μ_H, simulating a situation in which the human is much slower than the robot. As expected, it results in an increase of the resulting probability. The human appears to be a key bottleneck of the scenario. The result is detailed at Table 11, *Scen 3.c*.
- Finally, we increased $\sigma_H$while maintaining the *Scen 3.c* parameters (Table 11, *Scen 3.d*). As in the previous cases, the general expected behaviour is respected and the probability range increases.

In conclusion, we have noticed that the standard deviation has a lower impact on the system, compared to the average delay. This was expected, even though this verification has been very useful in order to understand the numerical differences between standard deviation and average.

4 Conclusions

In this report, we presented an analysis of the problem of the Model Checking of Warehouse Robotics. The scrutiny was focused on both realistic configurations and stressed configurations, in order to understand the validity of the representation of the system. To sum up, the model appears coherent with the specifications of the project and with abstract real-life situations.

References

[1] Alexandre David et al. "Uppaal SMC tutorial". In: *International Journal on Software Tools for Technology Transfer* 17.4 (Aug. 2015), pp. 397–415. ISSN: 1433-2787. DOI: 10.1007/s10009-014-0361-y. URL: https://doi.org/10.1007/s10009-014-0361-y.

[2] UPPAAL *tool*. URL: https://uppaal.org.