02224

Modelling and Analysis of Real-Time Systems

Baggage Sorting Facility

Authors:
Hjalte Bøgehave

Student Numbers:
s214018

Contents

1 Introduction

In this report, we set out to implement the simple baggage sorting facility in UPPAAL shown in figure 1. The main components of the system consists of two feeding belts, two sensors and a distribution belt, with two colors of bags to sort: black and yellow. We sought out to implement this system in UPPAAL while keeping the various constraints on time, collisions in check, while ensuring bags were delivered to their desired location based on their color.

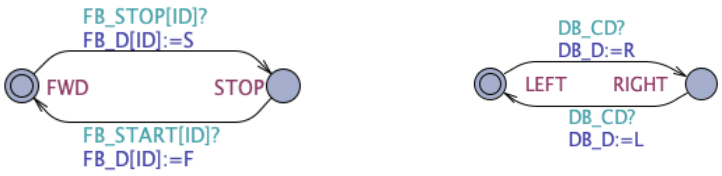
2 Implementation

The final implementation of the system in UPPAAL consists of six total templates. The templates representing the physical environment are the distribution belt, the feed belt and the bag. For handling the logic of the belt management, a controller template was implemented. Additionally, one synchronisation templates were created, acting as a clock template for handling the timings for the distribution belt. Finally, a user template was implemented for simulating the inputs.

2.1 Physical Environment

2.1.1 Feed belt and Distribution belt

The implementation for the feed belt can be seen in figure 1(a) and the distribution belt in figure 1(b).



(a) Feed Belt **(b)** Distribution Belt

Figure 1

The implementations of the feed belt and distribution belt are rather simple. The reason for separating them out into their own templates is partly due to keeping the UPPAAL system readable and partly due to easier debugging when running the simulator.

Both of the templates follows the same structure. Each are listening on a synchronisation channel in order to recieve any changes sent by the controller. The new direction is then stored in its respective variable such that it can be used as guards in other templates.

2.1.2 Bags

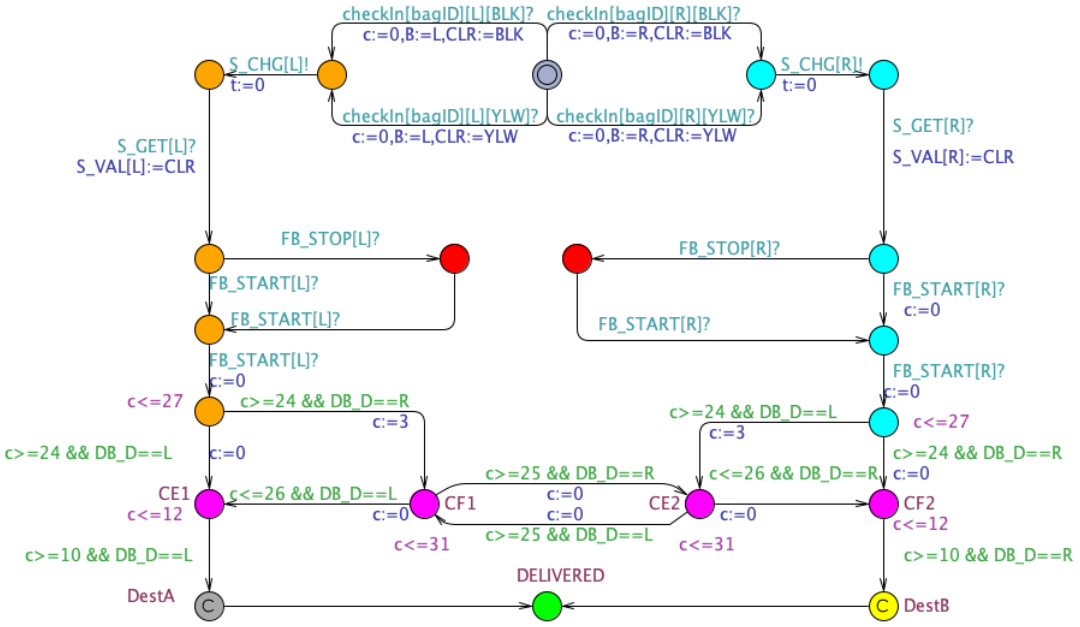


Figure 2: Bag Template

This template is the most comprehensive template in the system, as all the possible locations for the bag needs to be implemented. The orange locations represents locations on the left belt, cyan locations are on the right feed belt and magenta are on the distribution belt. Furthermore, goal 1 is represented in yellow, goal 2 is represented in dark grey, wait locations are represented as red and completed is represented in green.

It is in the bag template that the sensor readings are simulated. When the bag initially passes in front of the sensor, the controller is notified (S_CHG). The controller will then wait until a valid reading can be acquired and read the value (S_GET). The controller handles the timing for the bags when they are on the feed belt, so the bag locations only moves based on a STOP or START signal from the controller. When the bag leaves the feed belt, we instead use a local clock to ensure the timing between the locations hold.

2.2 Controller & User

2.2.1 Controller

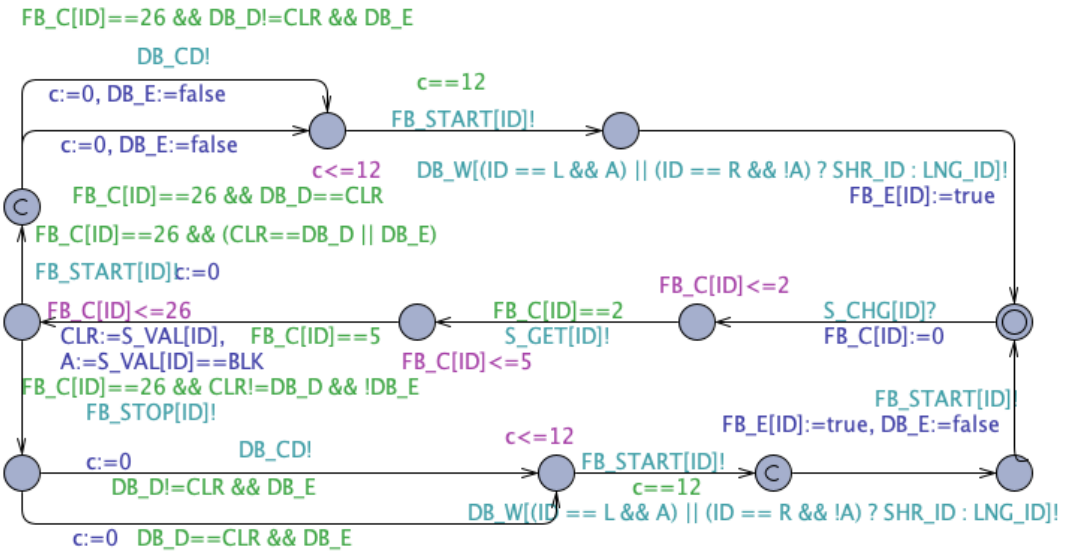


Figure 3: Controller Template

The controller represents the brains of the system, one for each feed belt. The main trigger for a new bag coming in is by detecting a change in the sensor value. From there, we wait until we can ensure a valid reading of the color, and then retrieve it. With FB_C[ID], we calculate the time until the bag is at the desired stop location, where it is then checked whether the distribution belt is blocked in the opposite of the desired direction, or if the distribution belt is free or already moving in the correct direction. Finally a channel is used to start the blocking time of the distribution belt.

2.2.2 User

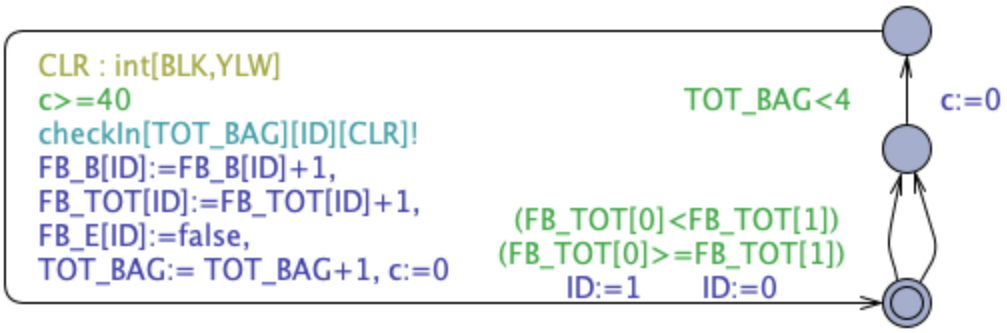


Figure 4: User Template

The user is responsible for handling the bags. The template represents a simple loop that first checks which feed belt has received the least bags in order to distribute the bags among the belts equally. Afterwards we check if the total number of bags that has been checked in is equal to our desired number of checked in bags. Finally we ensure a minimum of 40 clock ticks before the user can check in another bag.

2.3 Distribution Belt Clock

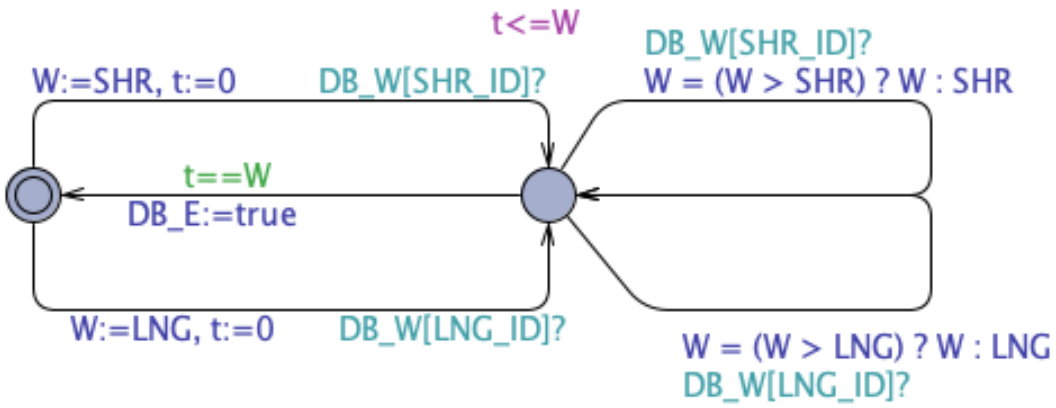


Figure 5: Distribution Belt Clock Template

This template handles the amount of time to block the distribution belt for, depending on the combination of the belt and the desired goal. Furthermore, to not add unneeded waiting time when a bag goes on the belt in the same direction the distribution belt is already locked in, we only update the waiting time if the new reserving time is longer than the current reserve time.

3 Assertions

OBS: I could not get the query exporting to work on my mac.

In this section, we use UPPAAL queries in order to verify that our desired constraints are satisfied for all states in our system.

3.1 Bags are delivered at the right destination.

```
A[] forall(i:t_id) (
  // Bags with color BLK should only reach destination A
  not (bag(i).CLR == BLK and bag(i).DestB) and
  // Bags with color YLW should only reach destination B
  not (bag(i).CLR == YLW and bag(i).DestA)
)
```

This query loops over all the bags and verifies that the destination reached is in compliance with the color of the bag.

3.2 No collisions take place (when the system is properly used).

```
A[] forall(i:t_id, j:t_id) (
  i != j imply
  // No two bags have the same position on the distribution belt
  not (bag(i).on_distribution_belt and bag(j).on_distribution_belt and pos(i
)
```

Here, we loop over all the permutations of the bags in order to verify that no bags are located in the same location at the same time.

3.3 While a bag is turning (in section c), neither the feed belt nor the distribution belt is stopped or reversed.

```
A[] forall(i:t_id) (!((bag(i).CE1 or bag(i).CF1 or bag(i).CE2 or bag(i).CF2)
```

Here, we check that for all bags, the DB_CD channel is not being synchronised while the bag is at one of the distribution belt locations.

3.4 Every bag is eventually delivered.

```
A[] forall(i:t_id) (
  bag(i).DELIVERED
)
```

Loop over the bags and ensure that all the bags will eventually reach the DELIVERED location.

4 Uncertainties

In order to check if the system fails if we go outside the certainty boundaries, we can modify the bag template to reflect timings on the distribution belt outside the given boundaries. After changing the values, we can rerun the queries from the last chapter in order to confirm or deny if any breaking changes was introduced.

5 Handling Time

In order to find the shortest and longest delivery time, a python script was used to search the space of queries

```
import subprocess

def run_uppaal_query(query):
    uppaal_command = ["./verifyta", "-q", "model.xml", "query.q"]
    with open("query.q", "w") as query_file:
        query_file.write(query)

    result = subprocess.run(uppaal_command, capture_output=True, text=True)
    return "true" in result.stdout
```

```
def binary_search(min_time, max_time, query_in):
    while min_time < max_time:
        mid_time = (min_time + max_time) // 2
        if run_uppaal_query(query):
            max_time = mid_time
        else:
            min_time = mid_time + 1

    return min_time

if __name__ == "__main__":
    min_time = 0
    max_time = 500

    delivery_time = binary_search(min_time, max_time)
    print(delivery_time)
```

5.1 Fastest Handling Time

For the fastest time the following query was used:

```
query = f"E[] exists(i:t_id) (bag(i).DestA or bag(i).DestB) and delivery_tim
```

Showing a Fastest handling at 62 time steps.

5.2 Longest Handling Time

For the fastest time the following query was used:

```
query = f"A[] forall(i:t\_id) (bag(i).Delivered imply delivery_time <= {mid_
```

Showing a longest handling at 336 time steps.

6 Conclusion

Modelling a system in UPPAAL, even in the case of a simple baggage sorting system, proved to be an iterative process in order to reach a properly represented model of the system. Debugging in UPPAAL is very limited, so even small corrections in the model, often led to unwanted consequences that required a vast amount of trial and error to resolve for someone who was previously unfamiliar with UPPAAL. Although after becomming more familiar with the UPPAAL syntax and getting the system to the desired state, one could gain a vast amount of insights into all the possible outcomes of the model, both desired and undesired.