Organizational Structures of Running a Hotel
Information Management for Computer Science

Mara Spadon (2688689, m.c.spadon@student.vu.nl, BSc AI),
Caspar Grevelhoerster (2707848, c.m.s.grevelhorster@student.vu.nl, BSc AI),
Elisa Bermejo Casla (2665178, e.bermejocasla@student.vu.nl, BSc AI)

Vrije Universiteit Amsterdam, De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands

Table of Contents p.

Table of Figures **p.**

1 Introduction

Hotels have become a ubiquitous commodity, catering to travelers all around the world. They provide shelter and numerous other services to their guests, ranging from incredibly luxurious all-inclusive experiences to low-priced rooms. However, one thing that all hotels have in common is the multiplex process of internal organization. This includes room reservations, check-in, check-out, as well as cleaning, general inspections and services during the stay. This report focuses on an established hotel that is already running successfully. Therefore, the depicted processes are representing a section of time.

We will scrutinize the process of running a hotel and further analyze it by formalizing the process using state-transition diagrams and Petri nets. After describing the process and providing models accordingly, a reflection will be provided, including the advantages and limitations of the proposed models.

1.1 Overall Process Description

The organizational structure can be seen as one large process consisting of multiple smaller sub-processes that happen simultaneously and occasionally intercept. Any attempt to model processes is, necessarily, a simplification of reality. As such, modeling implies choosing what is important and leaving out aspects or variables -in our view- not directly relevant for the model. In this section, we attempt to offer an overall overview of the complete process of running a hotel by illustrating the most important events that occur from the moment a guest steps into a hotel until the moment they leave.

The process starts when a guest wishes to visit the hotel. The first step is to check whether there are any available rooms. Subsequently, if there is any room available the prospective guests can book it. After the reservation, the payment transaction will be performed as a prerequisite for the check-in to take place. Afterward, the guests will be able to enjoy their stay during which they can choose to utilize the services that the hotel has to offer. These services can include eating at the restaurant, visiting the pool or sauna as well as ordering room service.

As the guest leaves the hotel, there will be a check-out performed after which the registration system shows the formerly booked room as empty. Once the room is empty, the hotel will request their cleaning professionals to clean the room and prepare it for the next guests. Finally, front desk staff will ensure that the room is in a satisfactory state to welcome other guests (i.e. no cleaning or repairs are required). The process will repeat itself for the next guest.

Another aspect of the modeling is that there might be external guests that do not stay overnight in the hotel but merely enjoy some of the services such as the restaurant.

The last aspect taken into account is a periodic in-depth inspection of each room and hotel facility. The inspector ensures that every detail, such as lighting, electricity, or furniture, is up to standards. They report their findings and repairs are arranged accordingly.

1.2 Components

The components of running a hotel consist of the participating actors as well as their activities and their interactions. The subsequent sections will provide lists showing all of these components.

1.2.1 Actors

In the complex process of running a hotel, there are a number of actors involved in the process. These are listed below.

- Guest
- Receptionist
- Cleaning Professional
- Inspector
- Additional Services Agents

1.2.2 Interactions

Additionally, there are interactions present which represent all the activities that are involved when running a hotel. These interactions can be found below.

- Checking Room Availability
- Booking Room
- Payment Transaction
- Check-In
- Restaurant Booking
- Check-Out
- Cleaning Room
- Checking Room
- Yearly Inspection; optional: Repair / Reorganization

2 Modeling with Transition Systems

The description provided in Section 1.1 will be modeled using state-transition diagrams. Furthermore, we will divide the processes over the actors to show each individual process. These diagrams will give a clear overview of the processes as they consist of a straightforward graphical representation.

2.1 Overall Process View

First of all, we provided a state-transition diagram showing the complete process of running a hotel, see Figure 1. This diagram depicts a simplified version of the process without focusing on any actors individually. The total description of all the steps can be found in Section 1.1. The transition system, TS=(*S, TR, s0*), of the diagram depicted below consists of the following state, transition relation and initial state:

$S = \{$*booking_request, check_availability, book_room, payment, check_in, stay, utilize_services, check_out, clean_room, available_room*$\}$
$TR = \{$*(booking_request, check_availability), (check_availability, book_room), (book_room, payment), (payment, check_in), (check_in, stay), (stay, utilize_services), (utilize_services, stay), (stay, check_out), (check_out, clean_room), (clean_room, available_room), (available_room, booking_request)*$\}$
$s0 = \{$*booking_request*$\}$

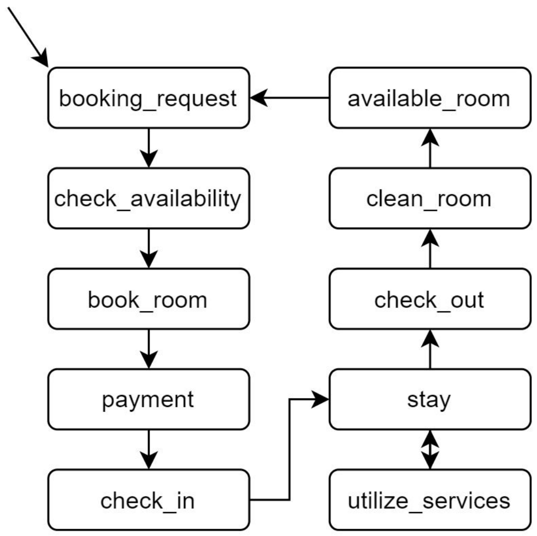

Fig. 1. General state-transition diagram of running a hotel

This state-transition diagram starts with a booking request after which the availability of the rooms will be checked. Afterwards, the room will be booked according to the request. Subsequently, the payment and check-in will take place. The guest will then stay for a certain amount of time and they have the option to utilize the services that the hotel has to offer. These services can include a restaurant, a sauna, room service etc. Once their stay is over, the guests will check out and the cleaning professionals will be notified that they can clean the room. Lastly, the room will be available to book again after it has been cleaned.

2.2 Process of Guests

The guests play a major role in keeping the hotel running. Without the guests, there would be no business to run. They follow a set of steps in order to stay at the hotel. These steps are as follows:

1. The first step in this process involves guests wishing to book either a room or a stay at the restaurant.
2. When staying at the restaurant, the guests check if there are any rooms available for their stay. This can either be done by checking the availability online, calling the hotel or asking at the service desk in person.
   – Contrarily, there are also outside guests who are only there to eat at the restaurant, so they will not follow the process of booking a room. Instead, they will book a table at the restaurant, eat there and eventually leave as a restaurant guest.
3. Once they have established that there is a room available for them, they can book said room. This could once again be done by booking online, calling the hotel or booking in person at the service desk.
4. Subsequently, the guests will have to pay for their room. For this report, we assumed that the payment happens before their stay.
5. After the payment is completed, the guests will be able to check in and receive the key to their room.
6. During their stay at the hotel, they have the option to utilize the services that are offered by the hotel. In this case, we included eating at the restaurant.
   – If they choose to stay at the restaurant, they will return to stay at the hotel afterwards.
7. Once their stay is over, they will have to go to the service desk to check out.
8. The very last step consists of the guests leaving the hotel after they have checked out.

Using these steps, we developed a state-transition diagram that shows the process of a guest staying at the hotel. However, we also took into consideration that there could be guests who are not staying at the hotel and will only visit the restaurant. This process can be seen in the state-transition diagram. The finished diagram can be seen in Figure 2. The transition system of this state-transition diagram consists of the following state, transition relation and initial state:

S = {*wish_to_book, room_available, booking, payment, check_in, stay, check_out, book_restaurant, eat_at_restaurant, hotel_guest, restaurant_guest, leave*}
TR = {*(wish_to_book, room_available), (wish_to_book, book_restaurant), (room_available, booking), (booking, payment), (payment, check_in), (check_in, stay), (stay, eat_at_restaurant), (eat_at_restaurant, stay), (book_restaurant, book_restaurant), (stay, check_out), (check_out, leave), (eat_at_restaurant, hotel_guest), (hotel_guest, stay), (eat_at_restaurant, restaurant_guest), (restaurant_guest, leave), (leave, wish_to_book)*}
s0 = {*wish_to_book*}

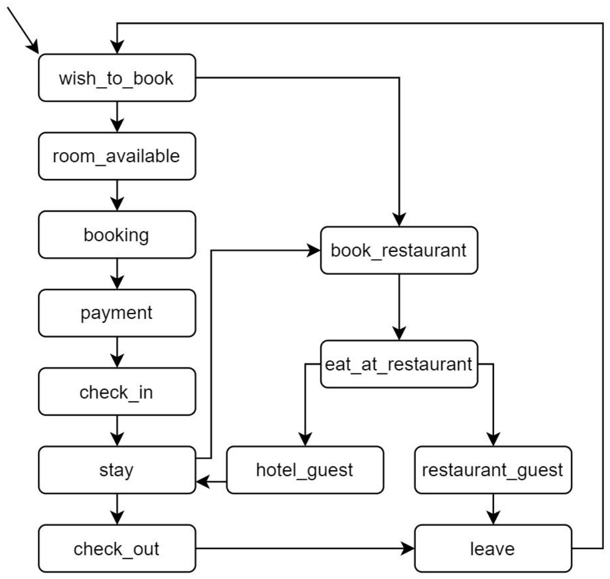

Fig. 2. State-transition diagram of the process of guests

2.3 Process of a Receptionist

Additionally, there is a group of receptionists present to make sure everything runs smoothly in the hotel. Their responsibilities mainly cover the booking of the rooms, check-in, check-out and making sure the rooms are ready for reservation. We established that they follow the subsequent steps:

1. The receptionist receives a booking request either in person, through the phone, or online.
2. They will check if there are any rooms available that fit the desires of the guest.
3. Afterwards, the receptionist will gather all the personal information of the guest to finish and confirm the booking of a room.
4. Once the room has been booked successfully, the guests needs to pay to confirm their booking and gain access to the room.
5. The guest will arrive at the hotel to check-in on the predetermined date. During the check-in, the receptionist will hand over the key to their room.
6. Once their stay has ended, the guest will return to the receptionist to check-out and the receptionist will take back the key.
7. The receptionist now knows that the guests have left the room and they will request the cleaning employees to clean the room.
8. After they are done cleaning, the receptionist will enter the room to make sure that everything is clean and ready for the next guests.

9. Optionally, they can request a reparation when something in the room is broken. Moreover, they could ask the cleaning employees to clean again if they are not satisfied.
10. Once the receptionist has established that the room is available for new guests, they enter that into the system so the room can be booked again.

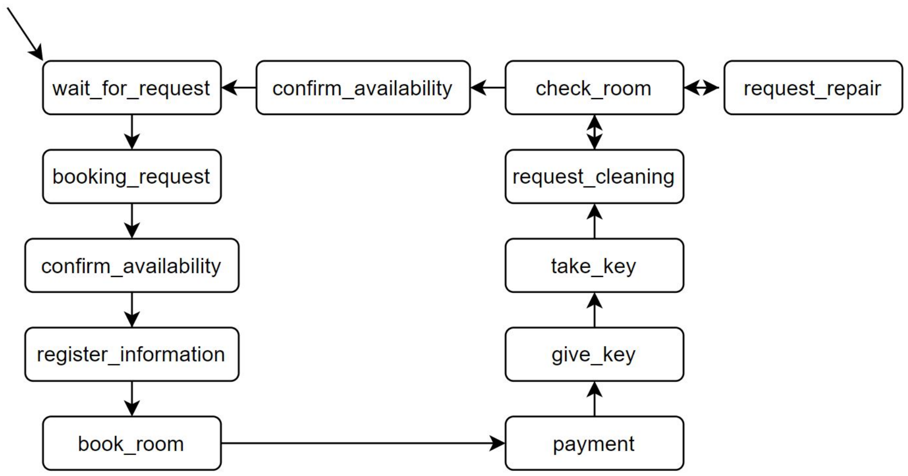

Fig. 3. State-transition diagram of the process of receptionists

The finished state-transition diagram of this process can be seen in Figure 3. The transition system of this state-transition diagram consists of the following state, transition relation and initial state:

S = {*wait_for_request, booking_request, check_availability, register_information, book_room, payment, give_key, take_key, request_cleaning, check_room, request_repair, confirm_availability*}
TR = {*(wait_for_request, booking_request), (booking_request, check_availability), (check_availability, register_information), (register_information, book_room), (book_room, payment), (payment, give_key), (give_key, take_key), (take_key, request_cleaning), (request_cleaning, check_room), (check_room, request_cleaning), (check_room, request_repair), (request_repair, check_room), (check_room, confirm_availability), (confirm_availability, wait_for_request)*}
$s0$ = {*wait_for_request*}

2.4 Process of a Cleaning Professional

The third group of actors are the cleaning professionals. Their role can be described in a small state-transition diagram. The individual steps happening in this process can be observed below.

1. The cleaning staff receives a cleaning request from the receptionists. This happens after guests have checked-out of their room.
2. Once they have received this request, they make their way to the appointed room. In the hotel room, they first check if anything is broken that needs to be fixed.
   (a) If so, they will request repairment and they will be asked to clean the room afterwards.
   (b) If nothing is broken or missing, they can start cleaning the room.
3. Finally, they will confirm that the cleaning has been finished to the receptionists so new guests will be able to book the room.

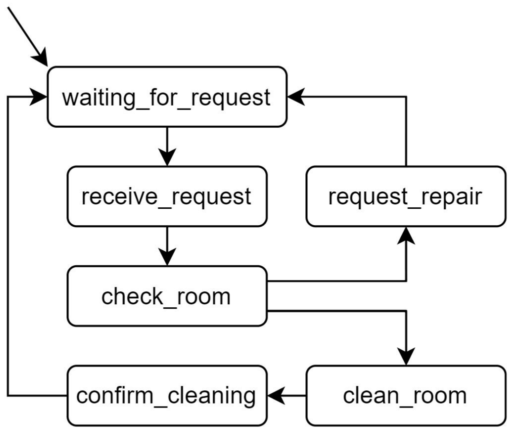

Fig. 4. State-transition diagram of the process of cleaning professionals

The transition system of this state-transition diagram, which can be found in Figure 4, consists of the following state, transition relation and initial state:

S = {*waiting_for_request, receive_request, check_room, clean_room, request_repair, confirm_cleaning*}
TR = {*(waiting_for_request, receive_request), (receive_request, check_room), (check_room, clean_room), (check_room, request_repair), (clean_room, confirm_cleaning), (confirm_cleaning, waiting_for_request), (request_repair, waiting_for_request)*}
s0 = {*waiting_for_request*}

2.5 Process of a yearly Inspector

The fourth group of actors is the inspector which is invited to the hotel on a yearly basis. Their role can be described in a small state-transition diagram. The individual steps happening in this process can be observed below.

1. During the initial state, the inspector waits for the request to come and inspect a hotel.
2. Next, the inspector receives an inspection request from the hotel management on a yearly basis in order to check the functionality of all private rooms as well as public areas.
3. The actual inspection takes place, which includes actions like checking light switches, opening windows or checking general tidiness.
   (a) If there is anything found to be missing functionality, a repair request is sent.
   (b) If everything works and is up to the set standard, the inspection is ended by a confirmation of satisfaction which is sent to the hotel management.

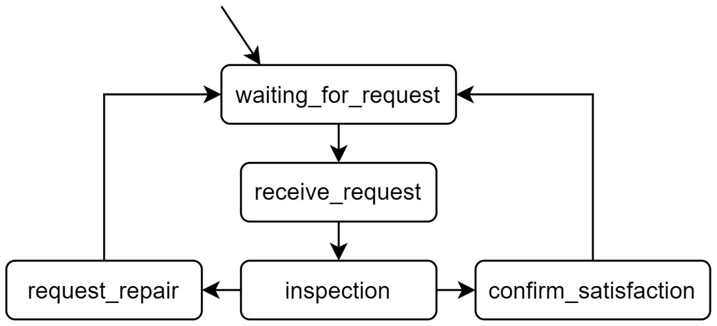

Fig. 5. State-transition diagram of the process of yearly inspection

The final state-transition diagram can be found in Figure 5. The transition system of this state-transition diagram consists of the following state, transition relation and initial state:

S = {*waiting_for_request, receive_request, inspection, confirm_satisfaction, request_repair*}
TR = {*(waiting_for_request, receive_request), (receive_request, inspection), (inspection, confirm_satisfaction), (inspection, request_repair), (confirm_satisfaction, waiting_for_request), (request_repair, waiting_for_request)*}
s0 = {*waiting_for_request*}

3 Modeling with Petri Nets

In this section, the process of running a hotel will be modeled using Petri nets. Petri nets are directed graphs with two kinds of nodes, namely places and transitions, represented by circles and squares respectively.

To design our Petri nets, we started with modeling the general overall process of running a hotel as described in section 1.1. Afterwards, we delved into certain sub-processes that are relevant when running a hotel, such as the functionality of the services that the hotel has to offer. These sub-processes will also be modelled using Petri nets. Because these are sub-processes, they were inevitably smaller and thus contain less places and transitions than the High-Level overview Petri net.

3.1 High-Level Overview

This section shows the main Petri net which includes the overall organization of the hotel. Though it does not go into every detail specifically, it does serve the purpose of showing the general process of the hotel.

Different design choices were taken when creating this Petri net. For instance, the payment is processed before the check in takes place and every other cost in the hotel is paid for separately (i.e. it is not possible to pay on tab). In addition, both internal (stay-in) and external guests can visit the restaurant, but the externals leave immediately after while the internals stay checked in. We chose not to model the specific extra services a guest could request since it would made the model overly complicated, difficulting the understanding of the general process.

Furthermore, there were different possibilities to consider when it came to the cleaning process. We concluded that it most suitable to model a process were after cleaning, a second person (e.g. front office worker) verifies that the room is indeed clean and ready to accept a new guest. Finally, we decided to limit the tokens that represent the guests, the number of rooms available and the the free tables at the restaurant to one. This decision was taken in order to simplify the reachability graph and obtain the subsequent analysis. The finished Petri net can be found in Figure 6.

The overall Petri net system, consists of the following places (P), transitions (T), flow relations (F) and initial markings ($m0$):

P = {*paid_restaurant_as_external, restaurant_reservation_as_external, dirty_table, paid_restaurant_as_internal, free_restaurant_tables, restaurant_reservation_for_internals, guest_arriving, room_is_available, need_to_pay, checked_in, paid_room, room_booked, enjoying_extra_services, dirty_room, cleaning_requested, cleaned, checked, rooms_available, room_inspected, room_out_of_order, repaired*}
T = {*pay_restaurant_as_external, cleaning_table, leaving_restaurant_as_external, leaving_restaurant_as_internal, book_restaurant_as_external,*

book_restaurant_as_internal, pay_restaurant_as_internal, check_availability, book_room, checking_in, checking_out, paying_room, request_extra_services, paying_extra_services, request_cleaning, cleaning, check_cleaning, not_clean, clean, inspect_room, positive, negative, request_repairs, becomes_available}
F = {*(guest_arriving, book_restaurant_as_external),(free_restaurant_tables, book_restaurant_as_external),(book_restaurant_reservation_as_external, restaurant_reservation_as_external),(restaurant_reservation_as_external, pay_restaurant_as_external),(pay_restaurant_as_external, paid_restaurant_as_external),(paid_restaurant_as_external, leaving_restaurant_as_external), (leaving_restaurant_as_external, guest_arriving), (leaving_restaurant_as_external, dirty_table), (dirty_table, cleaning_table), (cleaning_table, free_restaurant_tables), (guest_arriving, check_availability), (rooms_available, check_availability), (check_availability, room_is_available), (room_is_available, book_room), (book_room, need_to_pay), (need_to_pay, paying_room), (paying_room, paid_room), (paid_room, checking_out), (checking_out, guest_arriving), (checking_out, dirty_room), (dirty_room, request_cleaning), (request_cleaning, cleaning_requested), (cleaning_requested, cleaning), (cleaning, cleaned),(cleaned, check_cleaning), (check_cleaning, checked), (checked, not_clean), (not_clean, dirty_room), (checked, clean), (clean, rooms_available), (paying_room, room_booked),(room_booked, checking_in), (checking_in, checked_in), (checked_in, checking_out), (checked_in, request_extra_services), (request_extra_services, enjoying_extra_services), (enjoying_extra_services, paying_extra_services), (checked_in, book_restaurant_as_internal), (book_restaurant_as_internal, restaurant_reservation_for_internals), (restaurant_reservation_for_internals, pay_restaurant_as_internal), (pay_restaurant_as_internal, paid_restaurant_as_internal), (paid_restaurant_as_internal, leaving_restaurant_as_internal), (leaving_restaurant_as_internal, checked_in), (leaving_restaurant_as_internal, dirty_table), (free_rooms, inspect_room),(inspect_room, room_inspected), (room_inspected, positive), (positive, checked), (room_inspected, negative), (negative, room_out_of_order), (room_out_of_order, request_repairs), (request_repairs, repaired), (repaired, becomes_available), (becomes_available, checked)*}
m0 = [*guest_arriving, rooms_available, free_restaurant_tables*]

Additionally, the pre- and post-sets of this Petri net include:

•*leaving_restaurant_as_internal* = {paid_restaurant_as_internal}
leaving_restaurant_as_internal• = {dirty_table, checked_in}
•*pay_restaurant_as_external* = {restaurant_reservation_as_external}
pay_restaurant_as_external• = {paid_restaurant_as_external}
•*cleaning_table* = {dirty_table}
cleaning_table• = {free_restaurant_table}
•*leaving_restaurant_as_external* = {paid_restaurant_as_external}
leaving_restaurant_as_external• = {guest_arriving, dirty_table}

$\bullet$*book_restaurant_as_external* = {guest_arriving, free_restaurant_tables}
book_restaurant_as_external$\bullet$ = {restaurant_reservation_as_external}
$\bullet$*book_restaurant_as_internal* = {free_restaurant_table, checked_in}
book_restaurant_as_internal$\bullet$ = {restaurant_reservation_for_internal}
$\bullet$*pay_restaurant_as_internal* = {restaurant_reservation_for_internals}
pay_restaurant_as_internal$\bullet$ = {paid_restaurant_as_internal}
$\bullet$*check_availability* = {rooms_available, guest_arriving}
check_availability$\bullet$ = {room_is_available}
$\bullet$*book_room* = {room_is_available}
book_room$\bullet$ = {need_to_pay}
$\bullet$*checking_in* = {room_booked}
checking_in$\bullet$ = {checked_in}
$\bullet$*checking_out* = {checked_in, paid_room}
checking_out$\bullet$ = {guest_arriving, dirty_room}
$\bullet$*paying_room* = {need_to_pay}
paying_room$\bullet$ = {room_booked, room_paid}
$\bullet$*request_extra_services* = {checked_in}
request_extra_services$\bullet$ = {enjoying_extra_services}
$\bullet$*paying_extra_services* = {enjoying_extra_services}
paying_extra_services$\bullet$ = {checked_in}
$\bullet$*request_cleaning* = {dirty_room}
request_cleaning$\bullet$ = {cleaning_requested}
$\bullet$*cleaning* = {cleaning_requested}
cleaning$\bullet$ = {cleaned}
$\bullet$*check_cleaning* = {cleaned}
check_cleaning$\bullet$ = {checked}
$\bullet$*not_clean* = {checked}
not_clean$\bullet$ = {dirty_room}
$\bullet$*clean* = {checked}
clean$\bullet$ = {rooms_available}
$\bullet$*inspect_room* = {rooms_available}
inspect_room$\bullet$ = {room_inspected}
$\bullet$*positive* = {room_inspected}
positive$\bullet$ = {checked}
$\bullet$*negative* = {room_inspected}
negative$\bullet$ = {room_out_of_order}
$\bullet$*request_repairs* = {room_out_of_order}
request_repairs$\bullet$ = {repaired}
$\bullet$*becomes_available* = {repaired}
becomes_available$\bullet$ = {checked}

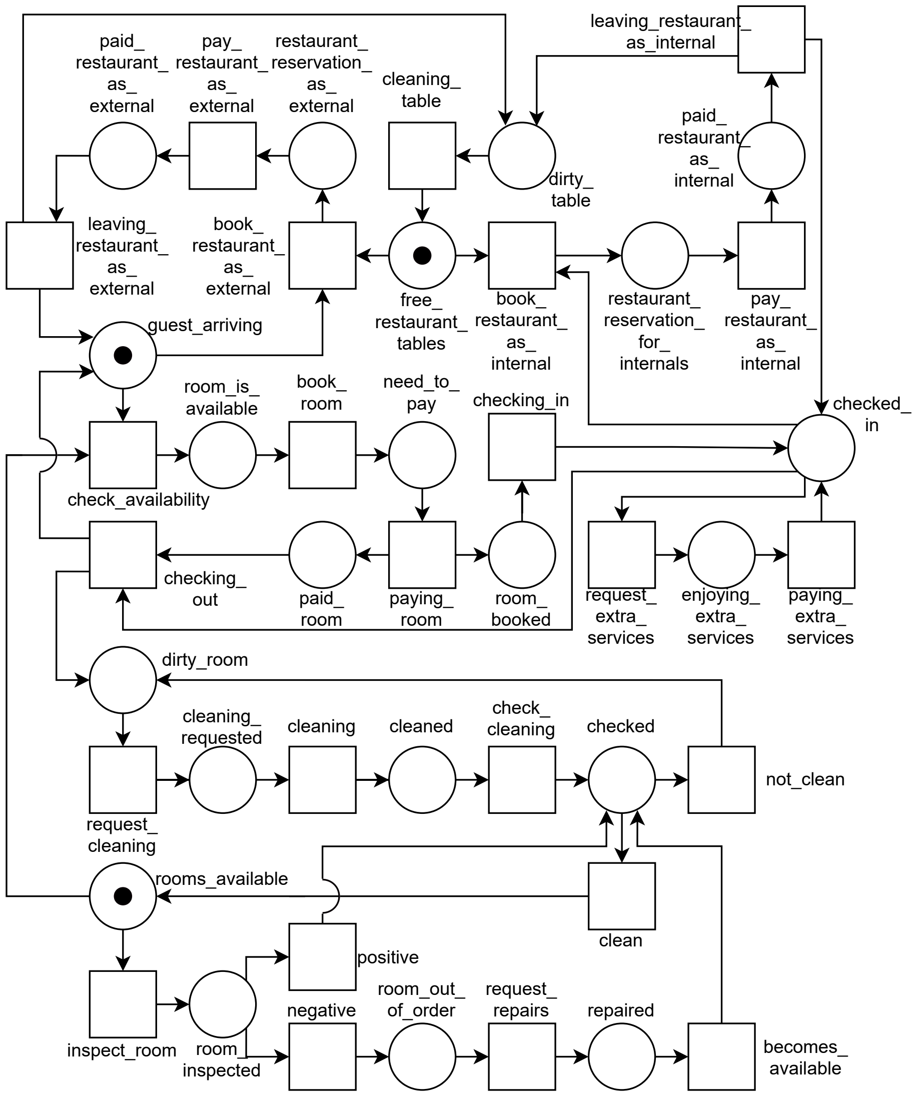

Fig. 6. High-Level Petri net of general processes

Reachability Graph and Petri Net Analysis

When analyzing Petri nets, there are several properties to be taken into account. In the following, each property will be applied to the Petri net and analyzed accordingly. The cited properties are from pages 257 and 258, section 8.2.2 *Standard Petri Net Properties*, from the book 'Evaluating Business Processes'. The reachability graph can be found in Figure 7.

1. **Dead Transitions**: "A transition t of a Petri net system is dead if and only if t is not enabled at any reachable marking."
   This Petri net has no dead transitions since all of them occur in at least one run, that is, all are enabled at some reachable marking.
2. **Liveness**: "A transition t is live if and only if from every reachable marking m there is a marking *m'* reachable such that t is enabled at *m'*. A Petri net system is live if and only if every transition $t \in T$ is live."
   This Petri net is live, since all of its transitions are live. That is, from any of its markings there is always another marking reachable such that a transition is enabled at the second marking. This ensures that all of the transitions can be enabled again.
3. **Terminating**: "A Petri net system is terminating if and only if every run is finite."
   This Petri net is not terminating because the runs are not finite. This can be seen in that the reachability graph (figure 7) is cyclical.
4. **Boundness**: "A Petri net system is *k-bounded* (or bounded for short) if and only if no place $p \in P$ contains more than k tokens in any reachable marking. Otherwise, the net is unbounded. If k is equal to one, the Petri net system is safe."
   This Petri net does not contain any place which can contain more than one token simultaneously. Therefore, it can be described as safe.
5. **Deadlock Freedom**: "A Petri net system is deadlock free if and only if at least one transition is enabled at every reachable marking."
   This Petri net does not have any deadlocks since at every reachable marking, at least one transitions is enabled. Therefore, it can be considered deadlock-free.
6. **Home-Marking**: "A marking m is home-marking if and only if from any reachable marking we can reach m."
   Since all the markings of this Petri net can be reached again, they are all home markings.
7. **Reversibility**: "A Petri net system is reversible if and only if its initial marking is a home-marking. A Petri net is reversible if its reachability graph is strongly connected."
   The reachability graph for this Petri net is strongly connected, and all reachable markings are home markings, which means that it is reversible.

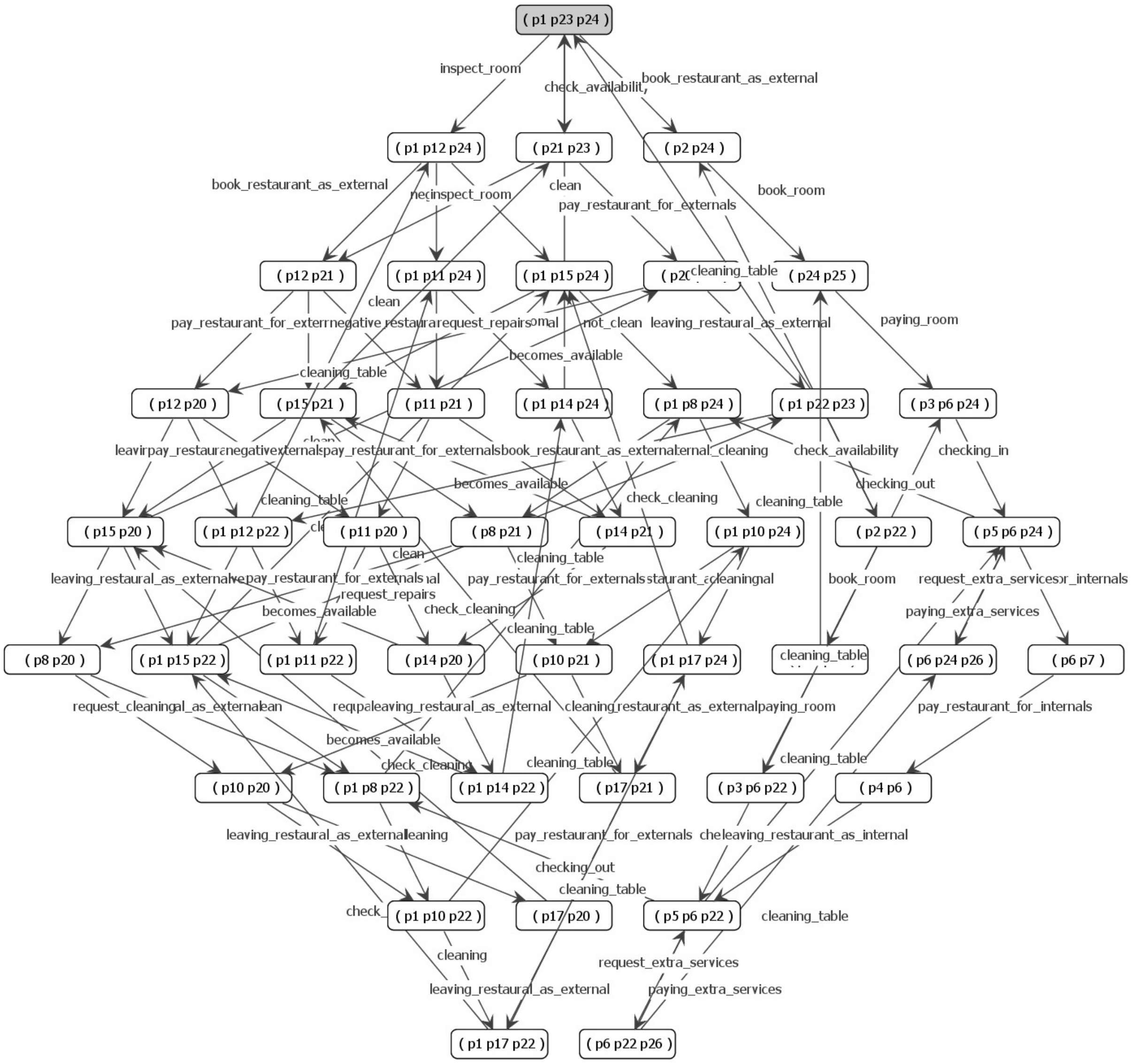

Fig. 7. Reachability graph of high-level Petri net

3.2 Chauffeur Service

The Petri net depicting the process of the chauffeur service starts with the place *guest* where a token representing a guest is placed. From there, a ride can be requested by using *request_ride*, which leads to a place named *ride_requested*. When the ride is requested and a chauffeur is available (in the case that a token exists in place *chauffeurs_available*), the transition *start_ride* can be accessed, leading both to a *ride_in_progress*- as well as a *need_to_pay*-place. From the latter, a *paying*-transition can be accessed leading to the place *paid*. While the ride is in progress and once the ride is paid, the *end_ride*-transition will finish the cycle by returning one token to the *chauffeur_available* and another one to the *guest*-place.

There were two central design choices taken during the design process. Firstly, it was decided that only one token would be used both for the available chauffeurs as well as the guests requesting the rides. Secondly, it would have been possible to extend the Petri net to different methods of transportation (pick-up, drop-off, shuttle-service), but a decision was taken to redefine the process to one type of riding only. The finished Petri net can be found in Figure 8.

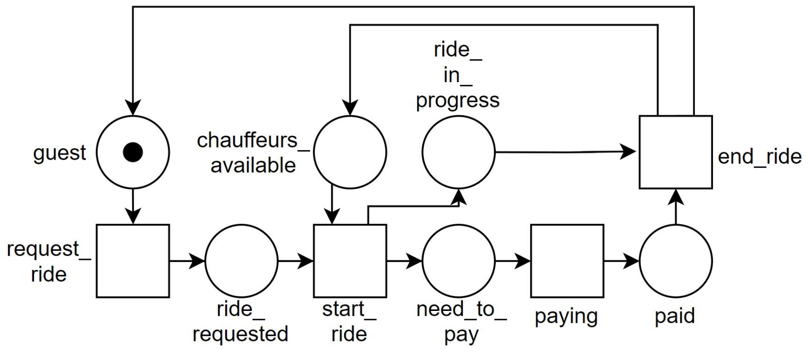

Fig. 8. Petri net of chauffeur service

The Petri net system, $(P,\ T,\ F,\ m0)$, consists of the following places, transitions, flow relations and initial markings:

P = {*guest, chauffeurs_available, ride_requested, ride_in_progress, need_to_pay, paid*}
T = {*request_ride, start_ride, paying, end_ride*}
F = {*(guest, request_ride), (request_ride, ride_requested), (ride_requested, start_ride), (start_ride, need_to_pay), (start_ride, ride_in_progress), (need_to_pay, paying), (paying, paid), (ride_in_progress, end_ride), (paid, end_ride), (end_ride, guest), (end_ride, chauffeurs_available),*

(chauffeurs_available, start_ride)}
m0 = [*guest, chauffeurs_available*]

Additionally, the pre- and post-sets of this Petri net include:

•*request_ride* = {guest}
request_ride• = {ride_requested}
•*start_ride* = {ride_requested, chauffeurs_available}
start_ride• = {ride_in_progress, need_to_pay}
•*paying* = {need_to_pay}
paying• = {paid}
•*end_ride* = {ride_in_progress, paid}
end_ride• = {guest, chauffeurs_available}

Reachability Graph and Petri Net Analysis
The reachability graph can be observed in Figure 9.

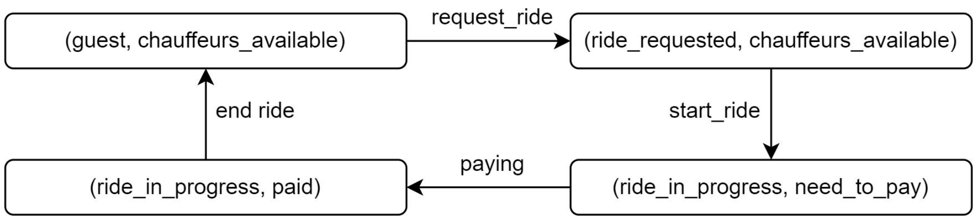
(guest, chauffeurs_available)
request_ride
(ride_requested, chauffeurs_available)
end ride
start_ride
paying
(ride_in_progress, paid)
(ride_in_progress, need_to_pay)

Fig. 9. Reachability graph of chauffeur service

Each state in the reachability graph represents a different marking of the Petri net (i.e. how the tokens are distributed among the places of the Petri net). These tokens can be either people, more concretely guests or chauffeurs; or information (ride_requested, need_to_pay, paid). The arrows on the graph show how the transitions lead from one marking to another.

For the analysis of this Petri net, properties will be considered that have already been described in section 3.1.

1. **Dead Transitions**: The Petri net for the chauffeur services has no dead transitions since all of them occur in at least one run, that is, all are enabled at some reachable marking.
2. **Liveness**: This Petri net is live, since all of its transitions are live. That is, from any of its markings there is always another marking reachable such that a transition is enabled at the second marking. This ensures that all of the transitions can be enabled again.

3. **Terminating**: Our Petri net is not terminating because the runs are not finite. This can be seen in that the reachability graph (figure 9) is cyclical.
4. **Boundness**: This Petri net does not contain any place which can contain more than one token simultaneously. Therefore, it can be described as safe.
5. **Deadlock Freedom**: This Petri net does not have any deadlocks since at every reachable marking, at least one transition is enabled.
6. **Home-Marking**: Since all the markings can be reached again, they are all home markings.
7. **Reversibility**: The reachability graph for this Petri net is strongly connected, and all reachable markings are home markings, which means that the net is reversible.

3.3 Bike Rental Service

The Petri net that depicts the process of the bike rental service starts with the place *guest* where a token representing a guest is placed. The available bikes are depicted using tokens in the place *bikes_available*. Once the bike has been requested, the service employees will check the availability of the bikes at the transition *check_availability*. Once they have established that the desired bikes are available, the guest will need to pay the rental price and a deposit to complete the payment and reach the place *payment_completed*. The guest will return the bike after they are done renting it. This is shown in the Petri net with the transition *return_bike*.

Subsequently, the bike rental service will check the condition of the bike after the guest has returned it during the transition *check_bike_state*. If the bike is still in good conditions, the token will place itself in place *good_state*. Contrarily, it will move to *bad_state* if the bike has been damaged in any way. If the bike is in a good state, they will return the deposit. If not, they will keep the deposit and use it to proceed with the repair.

While designing this Petri net, some assumptions were made. For instance, if a client requests a bike but there are no available bikes, they will wait until one is available. Furthermore, it is assumed that it is always possible to repair a bike. Finally, it was assumed that a guest will always return a bike (either in good or bad state). The number of tokens was set to one for both the guests and the bikes available to facilitate the understanding of the reachability graph and the posterior analysis of the properties of the Petri net. The finished Petri net can be observed in Figure 10.

The Petri net system for the bike rental service, $(P,\ T,\ F,\ m0)$, consists of the following places, transitions, flow relations and initial markings:

$P = \{$*guest, bikes_available, need_pay_rental, need_pay_deposit, paid_rental, paid_deposit, bike_rented, bike_returned, checked, good_state, bad_state, repairs_needed*$\}$
$T = \{$*renting, paying_rental,paying_deposit, completing_payment, return_bike, check_bike, positive, negative, return_deposit, keep_deposit, repair*$\}$

$F = \{$*(guest, renting), (bikes_available, renting), (check_availability, bike_available), (renting, need_pay_rental), (renting, need_pay_deposit), (need_pay_rental, paying_rental), (need_pay_deposit, paying_deposit), (paying_rental, paid_rental), (paying_deposit, paid_deposit), (paid_rental, completing_payment), (paid_deposit, completing_payment), (completing_payment, bike_rented), (bike_rented, return_bike), (returned_bike, bike_returned), (bike_returned, check_bike), (checked_bike, checked), (checked, positive), (checked, negative), (positive, good_state), (negative, bad_state), (good_state, return_deposit), (bad_state, keep_deposit), (keep_deposit, repairs_needed), (repairs_needed, repair), (repair, bikes_available)*$\}$
m0 = [*guest, bikes_available*]

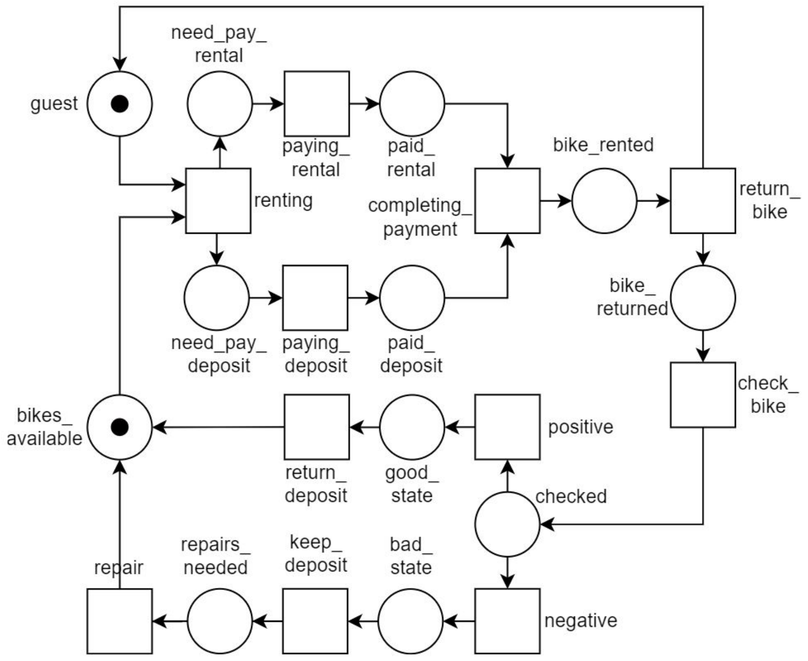

Fig. 10. Petri net of the bike rental service

Additionally, the pre- and post-sets of this Petri net include:

$\bullet$*renting* = {guest, bikes_available}
renting$\bullet$ = {need_pay_deposit, need_pay_rental}
$\bullet$*paying_rental* = {need_pay_rental}
paying_rental$\bullet$ = {paid_rental}
$\bullet$*paying_deposit* = {need_pay_deposit}

paying_deposit• = {paid_deposit}
•*completing_payment* = {paid_deposit, paid_rental}
completing_payment• = {bike_rented}
•*return_bike* = {bike_rented}
return_bike• = {guest, bike_returned}
•*check_bike* = {bike_returned}
check_bike• = {checked}
•*positive* = {checked}
positive• = {good_state}
•*negative* = {checked}
negative• = {bad_state}
•*return_deposit* = {good_stat}
return_deposit• = {bikes_available}
•*keep_deposit* = {bad_state}
keep_deposit• = {repairs_needed}
•*repair* = {repairs_needed}
repair• = {bikes_available}

Reachability Graph and Petri Net Analysis

The reachability graph of the bike rental service can be seen in Figure 11. For the analysis of this Petri net, properties will be considered that have already been described in section 3.1.

1. **Dead Transitions**: The Petri net for the bike rental service has no dead transitions since all of them occur in at least one run. From this we can conclude that all transitions are enabled at some reachable marking.
2. **Liveness**: This Petri net is live, since all of its transitions are live. That is, from any of its markings there is always another marking reachable such that a transition is enabled at the second marking. This ensures that all of the transitions can be enabled again.
3. **Terminating**: Our Petri net is not terminating because the runs are not finite. This can be seen in that the reachability graph (figure 11 is cyclical.
4. **Boundness**: This Petri net also does not contain any place which can contain more than one token simultaneously. Therefore, it can be described as safe.
5. **Deadlock Freedom**: This Petri net does not have any deadlocks since at every reachable marking, at least one transitions is enabled.
6. **Home-Marking**: Since all the markings can be reached again, they are all home markings.
7. **Reversibility**: The reachability graph for this Petri net is strongly connected, and all reachable markings are home markings. From this, we can conclude that this Petri net is reversible.

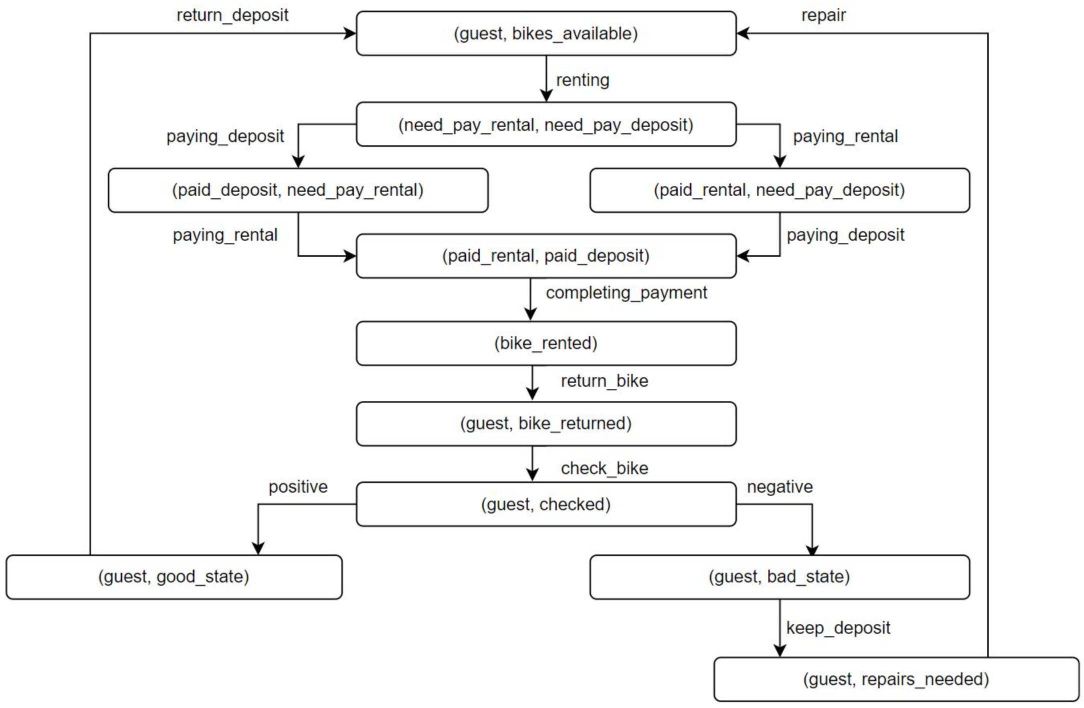

Fig. 11. Reachability graph of bike rental service

3.4 In-Depth Restaurant Service

Another relevant process to take into account when running a hotel, is running the hotel's restaurant. Although this process was roughly described as part of the overall process (see Section 1), this section will delve into the problem more in-depth. When a guest arrives at the restaurant, they are free to choose a table, given that there are tables available. These steps are shown in the Petri net through the place *guest_arriving*, the place *tables_available*, and the on both dependent transition *choose_table*. After the guest has taken a seat (place *guest_seated*), they are free to order, which is modeled through the transition *ordering*. By ordering, they need to pay (place *need_to_pay*) and the dish is being prepared (place *dish_being_prepared*). Next, the dish will be served and they can enjoy their food and drink. These last steps are modeled though the transition *serving* and the place *food_or_drink_consumption* respectively. Following the payment of the bill - that is, the transition *paying_bill* and the place *bill_paid* - the guests are free to leave the restaurant, indicated in the Petri nets by the transition *leaving_restaurant*. Once the guest leaves (place *guest_left*), the table must be cleaned to be ready to welcome the next guest, which is modeled through the place *dirty_table* and the transition *cleaning_table* which then lead to the place *tables_available*.

The most obvious choice made during setup of this Petri net was the decision not to make it cyclic like it was done for the previous problems. Instead, this Petri net is terminating and depicts the process the restaurant goes through per individual guest. Another option which was decided upon was the payment and serving processes to be happening in parallel. By ordering, two places are activated at the same time, representing the dish being prepared as well as the created need to pay. A modeling choice would have been to instead make the payment come after the food or drink consumption, which would follow the most common procedure of restaurant visits. However, guests are theoretically already able to pay while or even before eating if they choose to do so - which is the thought this choice was based on. The final Petri net can be found in Figure 12.

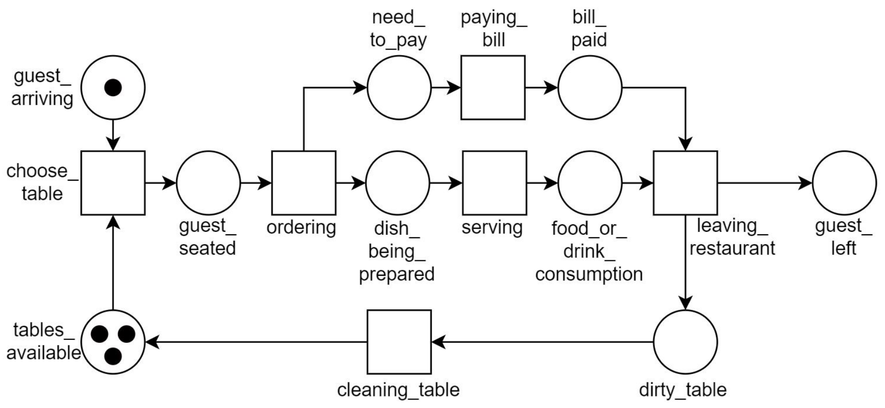

Fig. 12. Petri net of in-depth restaurant service

The Petri net system for the restaurant service, $(P,\ T,\ F,\ m0)$, consists of the following places, transitions, flow relations and initial markings:

P = {*guest_arriving, tables_available, guest_seated, dish_being_prepared, need_to_pay, bill_paid, food_or_drink_consumption, dirty_table, guest_left*}
T = {*choose_table, ordering, paying_bill, serving, leaving_restaurant, cleaning_table*}
F = {*(guest_arriving, choose_table), (tables_available, choose_table), (choose_table, guest_seated), (guest_seated, ordering), (ordering, need_to_pay), (ordering, dish_being_prepared), (need_to_pay, paying_bill), (paying_bill, bill_paid), (dish_being_prepared, serving), (serving, food_or_drink_consumption), (bill_paid, leaving_restaurant), (food_or_drink_consumption, leaving_restaurant), (leaving_restaurant, guest_left), (leaving_restaurant, dirty_table), (dirty_table, cleaning_table), (cleaning_table, tables_available)*}
m0 = [*guest_arriving, 3*tables_available*]

The pre- and post-sets of the restaurant Petri net are the following:

$\bullet$*choose_table* = {guest_arriving, tables_available}
choose_table$\bullet$ = {guest_seated}
$\bullet$*ordering* = {guest_seated}
ordering$\bullet$ = {need_to_pay, dish_being_prepared}
$\bullet$*paying_bill* = {need_to_pay}
paying_bill$\bullet$ = {bill_paid}
$\bullet$*serving* = {dish_being_prepared}
serving$\bullet$ = {food_or_drink_consumption}
$\bullet$*leaving_restaurant* = {bill_paid, food_or_drink_consumption}
leaving_restaurant$\bullet$ = {guest_left, dirty_table}
$\bullet$*cleaning_table* = {dirty_table}
cleaning_table$\bullet$ = {tables_available}

Reachability Graph and Petri Net Analysis
The reachability graph of this restaurant service can be found in Figure 13. For the analysis of this Petri net, properties will be considered that have already been described in section 3.1.

1. **Dead Transitions**: The Petri net for the restaurant service has no dead transitions since all of them occur in at least one run. From this we can conclude that all transitions are enabled at some reachable marking.
2. **Liveness**: This Petri net is live, since all of its transitions are live. That is, from any of its markings there is always another marking reachable such that a transition is enabled at the second marking. This ensures that all of the transitions can be enabled again.
3. **Terminating**: Our Petri net is terminating because the runs are finite. This can also be seen in that the reachability graph (Figure 13) is non-cyclical. The reachability graph is linear and does not loop back to any of the previous states.
4. **Boundness**: This Petri net does contain any place which can contain more than three tokens simultaneously, namely the place *tables_available*. Therefore, it can be described as 3-bounded and it is not safe.
5. **Deadlock Freedom**: This Petri net does not have any deadlocks since at every reachable marking, at least one transitions is enabled. From this, we can conclude that the Petri net is deadlock free.
6. **Home-Marking**: There are no home-marking, since there are no markings that can always be reached again.
7. **Reversibility**: This Petri net is not reversible since the reachability graph (Figure 13) is not strongly connected.

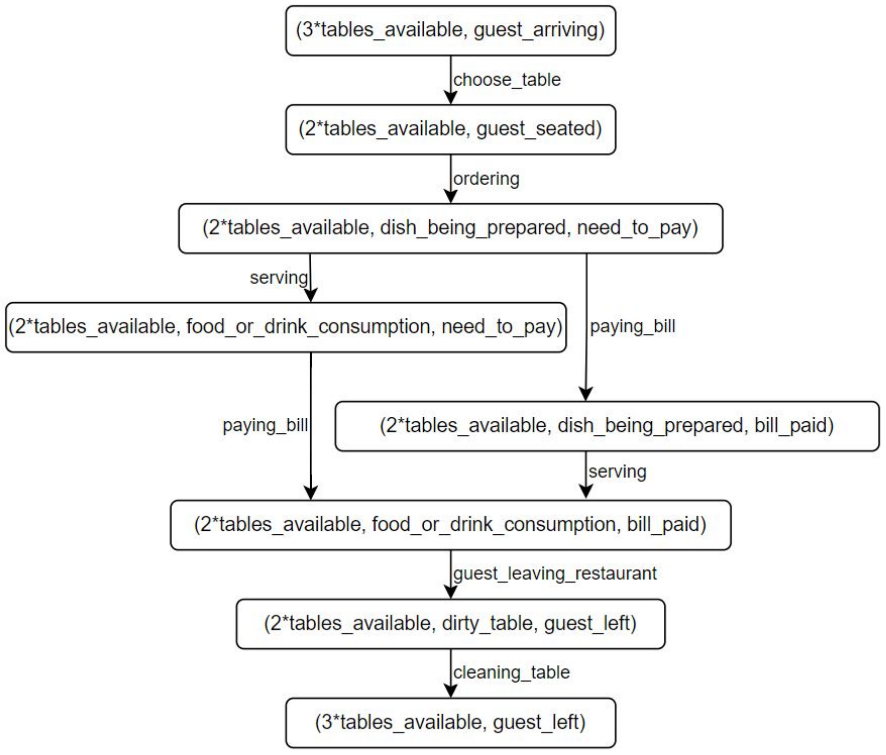
(3*tables_available, guest_arriving)
choose_table
(2*tables_available, guest_seated)
ordering
(2*tables_available, dish_being_prepared, need_to_pay)
serving
(2*tables_available, food_or_drink_consumption, need_to_pay)
paying_bill
paying_bill
(2*tables_available, dish_being_prepared, bill_paid)
serving
(2*tables_available, food_or_drink_consumption, bill_paid)
guest_leaving_restaurant
(2*tables_available, dirty_table, guest_left)
cleaning_table
(3*tables_available, guest_left)

Fig. 13. Reachability graph of in-depth restaurant service

4 Reflection

In this section, we will reflect on our provided work. We will discuss the advantages, discoveries, limitations and task division of this project.

Advantages
We provided a big state-transition system and Petri net that described the overall process of running a hotel. To go more in depth, we also decided to model several sub-processes that would provide further insight into the process. These sub-processes included state-transition diagrams from the perspective of the different actors and Petri nets that described several external services like a bike rental service, a chauffeur service and an in-depth overview of the restaurant service.

Furthermore, we think that our high-level Petri net was able to capture the entire process that we envisioned. We took into account that there could be 'external' guests who would only utilize the restaurant. Additionally, the 'internal' guests are able to go to the restaurant and book a room.

Discoveries
The project helped the group members to understand the concept of Petri nets better. We realized that they are useful for visualizing and conceptualizing workflows as well as their analysis. In the future - especially during setup of new process and information flows - Petri nets will serve as a useful tool. During their construction, they trigger a conversation about the actual setup which makes the information-flow architecture more mature right from the beginning. Through their simple and easy-to-understand graphical structure, they help show limitations of the processes. Generally, they can be seen as a promising tool to rationally analyze processes already during setup.

We have also noticed that Petri nets are extraordinarily versatile, places in the diagrams can represent numerous things, which makes them useful for many different applications. Furthermore, they provide a useful tool of communication between theoretical and practical structures of a company through which inefficiencies can be handled more easily by providing an easy mean of communication.

Limitations
When reflecting on our work, we came across a few possibilities for further improvement or limitations. The fact that our models are general, while having the advantage of making them generalizable, also means that they are somehow different from the way any concrete real-world hotel runs.

During the modeling process, many assumptions and choices were taken. Although we tried to make these choices explicit, other choices could have been made and might have fit better. One example of such choices is the payment system that we implemented for the overall process. To simplify the model, we made it so each service was paid for at the moment it was enjoyed. However, many hotels provide a tab system for their guests, which is not accounted for in our model.

Furthermore, we modeled some sub-processes that are, in our view, relevant for a hotel. However, different hotels provide or focus on different services, which means that they might find other sub-processes more relevant.

Another choice taken was to limit the number of tokens in the Petri net by having only one simultaneous guest at the time. While this made the analysis easier, it is further away from the real world situation as only one guest at a time can be tracked. Additionally, we found that we had to simplify some of our Petri nets in order to be able to make clear and understandable reachability graphs. This decreased the complexity of some of our Petri nets.

Lastly, we found that the WoPed program was unstable for some of our larger Petri nets and frequently crashed during the modeling process. Especially when we wanted to auto-generate a reachability graph. It worked in the end, but we also decided to design the Petri nets in draw.io after finishing them to be able to make a clear version of the Petri nets to include in our report.

Task Division

Overall, we think that the work was divided equally and we do not think that one's work exceeded way above the work of the other group members. It was a very natural process, we did not make a strict division. We decided to meet up at the university buildings of the VU almost every day to discuss and divide the tasks accordingly. We also set deadlines for one another to make sure we finished everything in time. As we were able to meet up in real life, we were able to reflect on each others work and adapt certain things if needed. The subsequent paragraph provides a clear overview of who contributed what to the report.

Mara mainly focused on the state-transition diagrams, the overall layout/ text of the report and the (analysis of) reachability graphs. Caspar mainly focused on optimizing the graphical representations and design of each diagram. This also allowed him to check them and adapt them if needed. Additionally, he evaluated the Petri nets and reachability graphs to write the analysis of each Petri net accordingly. Elisa mainly focused on developing the Petri nets after discussing the processes with the other group members. She also contributed to creating the reachability graphs. We all collectively contributed to the writing of the report and the mathematical notations of the Petri net systems (including the pre-sets and post-sets) and the transition systems.

Appendix

In this appendix, we will show work that was not included in the final paper. Figure 14 shows a Petri net that was built for the initial guest registration handled by the receptionists. Because of the many parallel pathways during the information registration for unregistered guests, the correspondent reachability graph exceeds any boundaries, making it impossible to show and analyze. Nevertheless, we think it is important to show what we thought the registration process might look like.

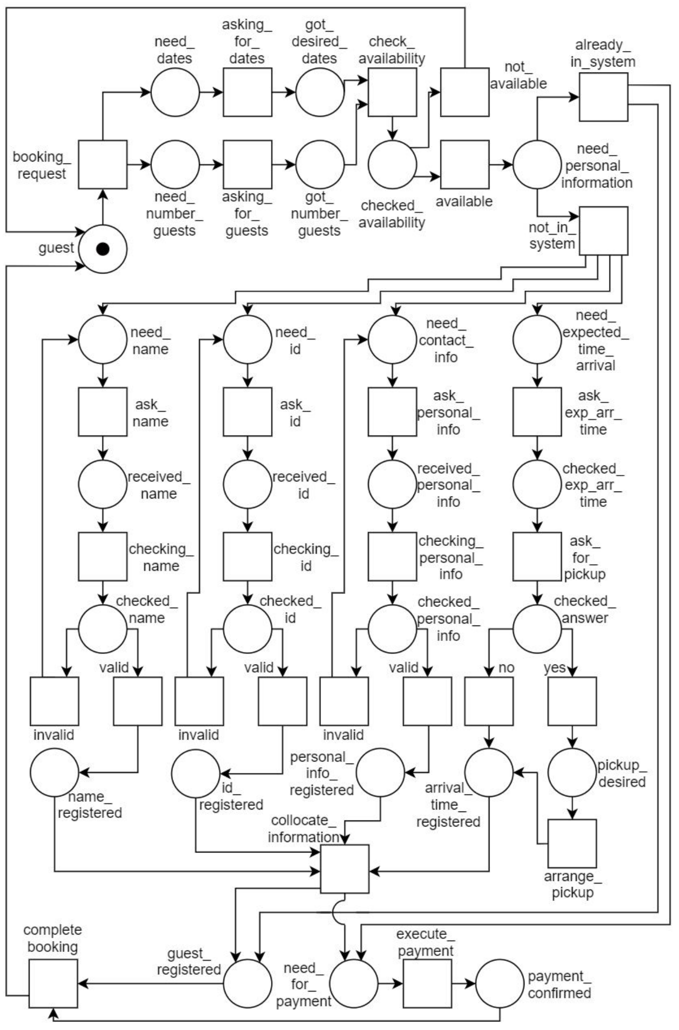

Fig. 14. Petri net of in-depth reservation system