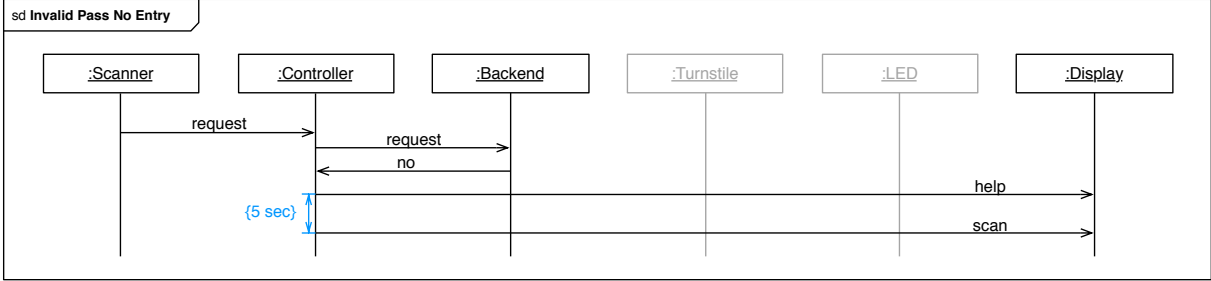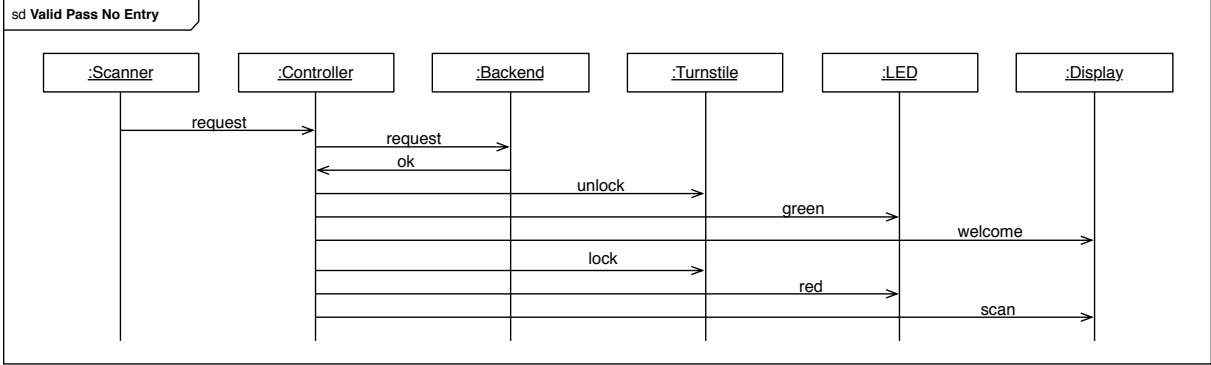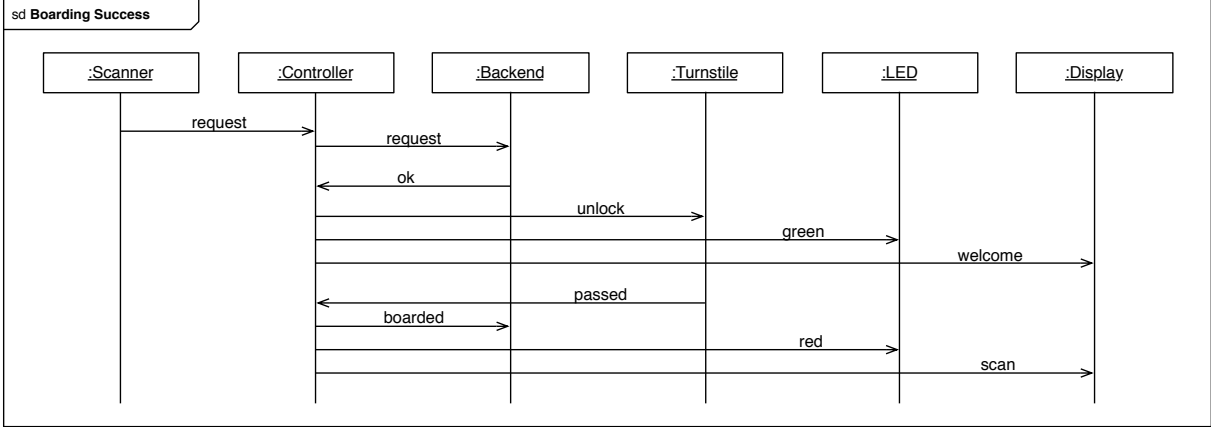The three diagrams each show a specific scenario. They don't show all solutions, only a selected course of events.

sd Boarding Success

:Scanner	:Controller	:Backend	:Turnstile	:LED	:Display

- request (Scanner → Controller)
- request (Controller → Backend)
- ok (Backend → Controller)
- unlock (Controller → Turnstile)
- green (Controller → LED)
- welcome (Controller → Display)
- passed (Turnstile → Controller)
- boarded (Controller → Backend)
- red (Controller → LED)
- scan (Controller → Display)

sd Valid Pass No Entry

:Scanner	:Controller	:Backend	:Turnstile	:LED	:Display

- request (Scanner → Controller)
- request (Controller → Backend)
- ok (Backend → Controller)
- unlock (Controller → Turnstile)
- green (Controller → LED)
- welcome (Controller → Display)
- lock (Controller → Turnstile)
- red (Controller → LED)
- scan (Controller → Display)

sd Invalid Pass No Entry

:Scanner	:Controller	:Backend	:Turnstile	:LED	:Display

- request (Scanner → Controller)
- request (Controller → Backend)
- no (Backend → Controller)
- help (Controller → Display)
- {5 sec}
- scan (Controller → Display)

The last diagram also specifies a timing constraint (in blue) to separate the help message from the scan message for the next passenger.

State Machine

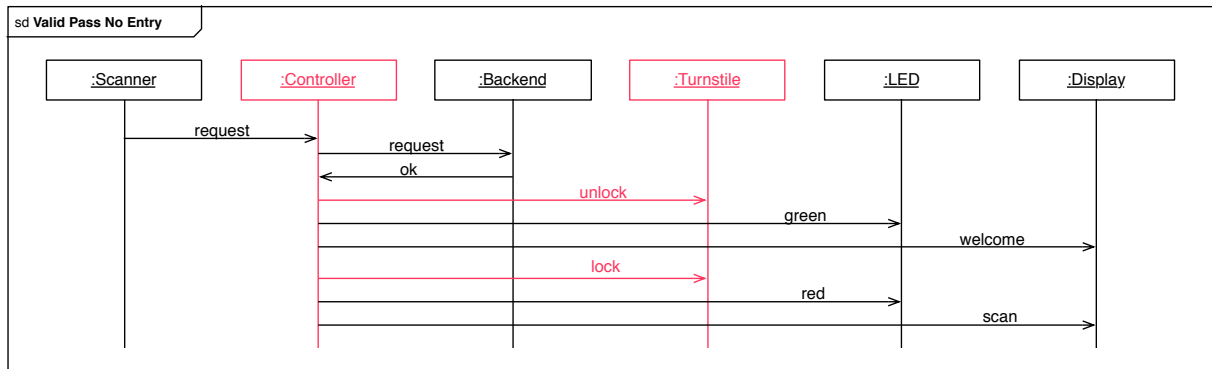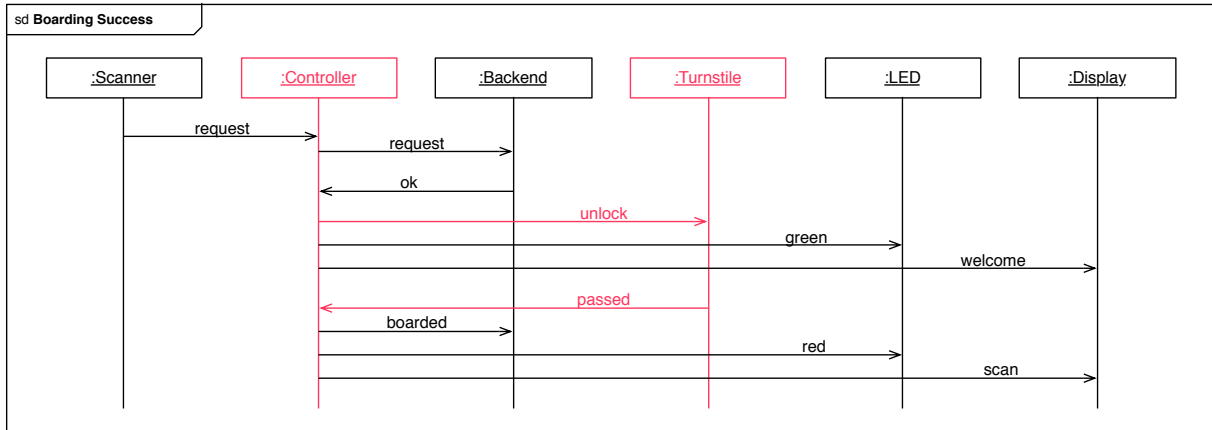We first look at the lifeline for the controller and highlight it in all of the three diagrams:
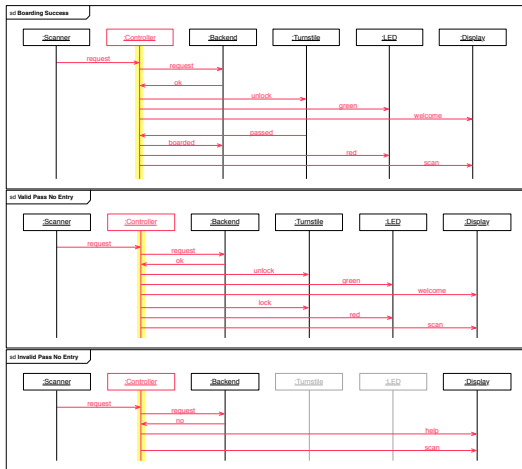
We can simply start with state idle in the beginning, at the top of the first lifeline, and then add incoming and outgoing messages as transitions, step by step.
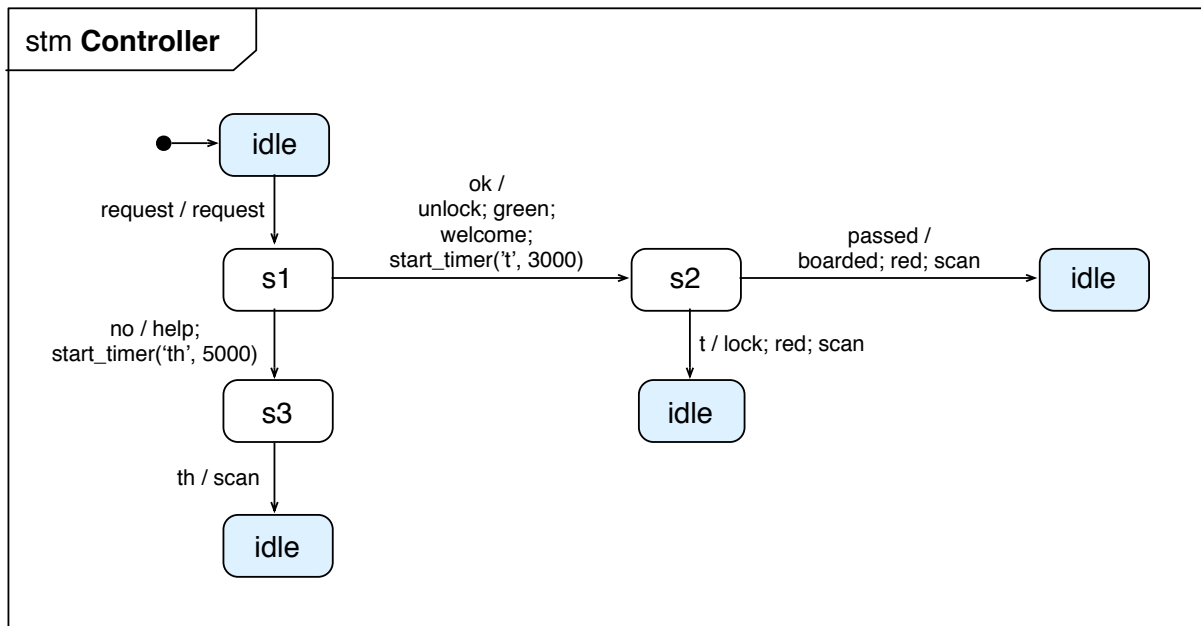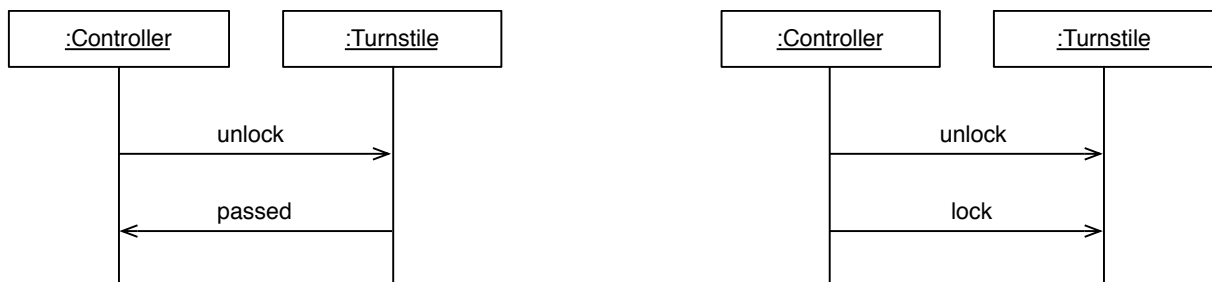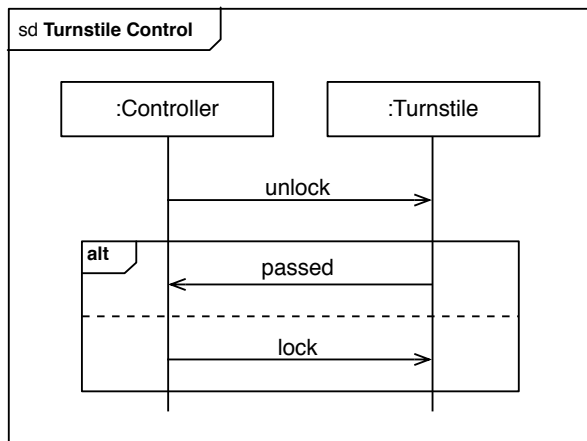
Turnstile — Controller

For the detailed look at the interaction between turnstile and controller, we mark the relevant lifelines and messages in both diagrams that are relevant for the turnstile:

sd Boarding Success

:Scanner	:Controller	:Backend	:Turnstile	:LED	:Display

- request
- request
- ok
- unlock
- green
- welcome
- passed
- boarded
- red
- scan

sd Valid Pass No Entry

:Scanner	:Controller	:Backend	:Turnstile	:LED	:Display

- request
- request
- ok
- unlock
- green
- welcome
- lock
- red
- scan

From these we can extract two simple diagrams:

:Controller	:Turnstile

- unlock
- passed

:Controller	:Turnstile

- unlock
- lock

With an alt fragment, we can combine these two into a single one:

This interaction contains a mixed initiative. We can illustrate this more clearly when we exaggerate the runtime of the messages by showing them with a slope:
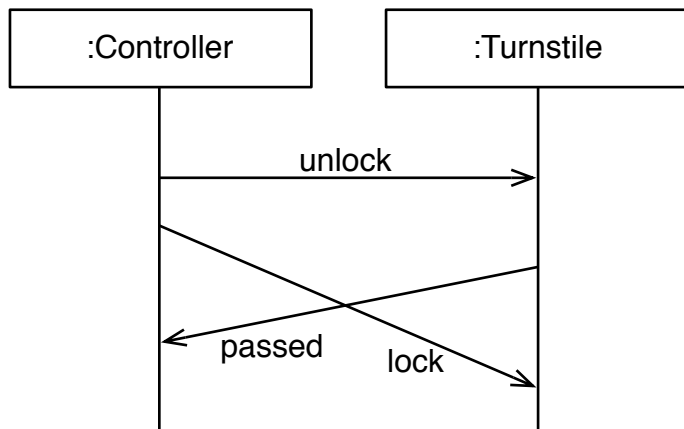

This means that the controller times out and wants to lock the turnstile again, but just after it sent the message lock, the passenger goes through and boards the airplane and the turnstile sends passed.

In the state machine we did not take care of this! After the timer t expires and we send lock to the turnstile, we are again in state idle. If now the message passed arrives, it is discarded since idle does not have any outgoing transition triggered by a message passed. **We would have missed a boarded passenger because we did not send it to the backend!**
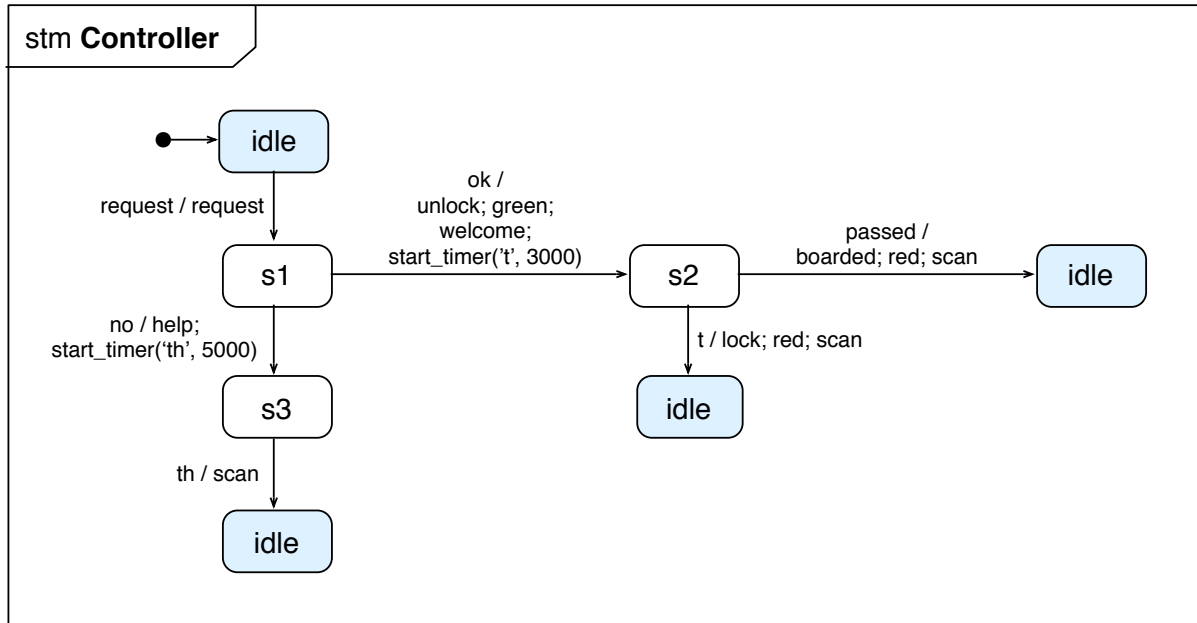
We can solve the problem by extending the state machine with a new transition:
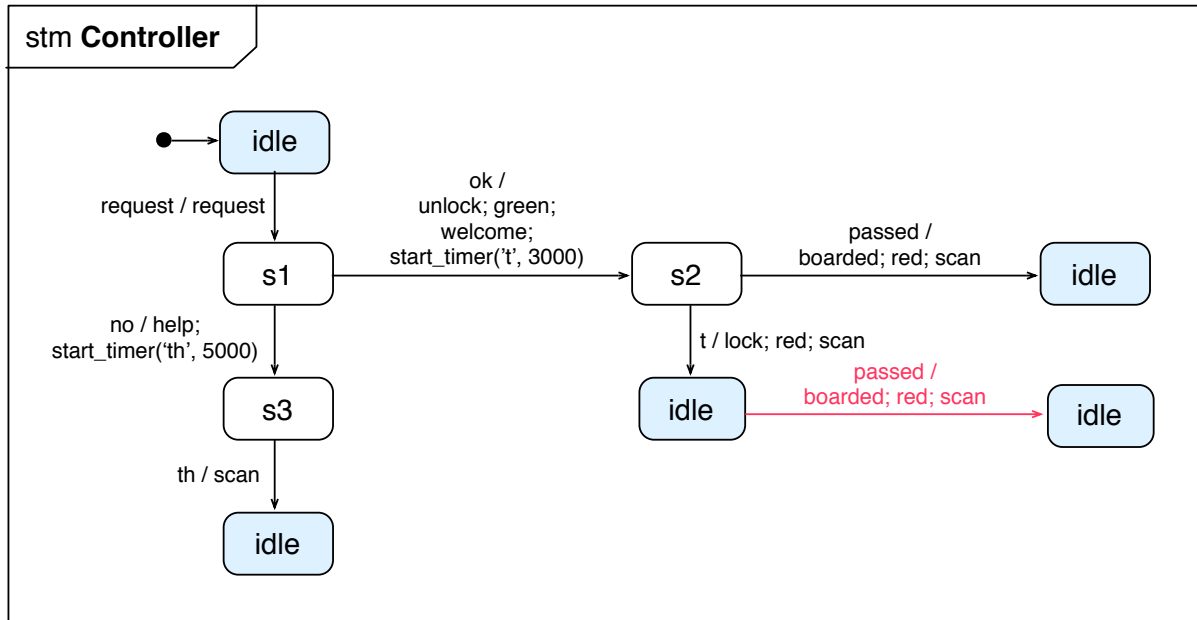
The transition (here marked in red) goes from state idle to idle (hence a self-transition) and registers the passenger with the backend. It could also be specified as an internal transition of state idle.