# Elevator Control as a Planning Problem

**Jana Koehler** **Kilian Schuster**
Schindler Lifts Ltd.
CH-6031 Ebikon
Switzerland
jana_koehler|kilian_schuster@ch.schindler.com

## Abstract

The synthesis of elevator control commands is a difficult problem when new service requirements such as VIP service, access restrictions, nonstop travel etc. have to be individually tailored to each passenger. AI planning technology offers a very elegant and flexible solution because the possible actions of a control system can be made explicit and their preconditions and effects can be specified using expressive representation formalisms. Based on the specification, a planner can flexibly synthesize the required control and changes in the specification do not require any reimplementation of the control software. In this paper, we describe the application and investigate how currently available *domain-independent* planning formalisms can cope with it.

## Introduction

The design of modern elevator systems requires new flexibility in meeting the demands of customers. An elevator is not only a mechanic device to implement the vertical transport within a building, but becomes an integrative part of the logistics infrastructure a building offers to its inhabitants. This means to provide a variety of transport modes to passengers and to integrate the elevator into the security concept of the building.

In order to meet the needs of passengers, their desires and requirements have to be identified by the elevator system. For this purpose, Schindler Lifts Ltd. has developed Miconic-10™ an elevator system based on so-called *Destination Control* where passengers input their destination before they enter the elevator, cf. Figure 1. Miconic-10™ has been introduced into the market in 1996. More than 100 installations have been sold worldwide, in particular for upper range buildings with thousands of inhabitants and large elevator groups, e.g., the Rockefeller Center, New York, or Coeur Defense, Paris, where its transportation quality and passenger interface are unrivaled.

A 10-digit keypad is installed in front of the elevator group where passengers enter the floor they want to travel to, e.g., 22. After input of the destination, the elevator control system selects an elevator for the transport of the passenger, which meets all requirements of

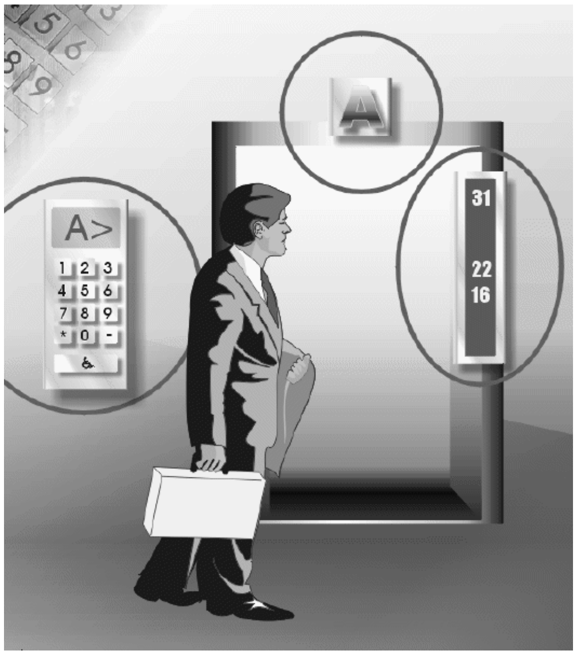

Figure 1: A Miconic-10™ keypad allows passengers to enter their destination before they enter the elevator. A display informs the passenger about the elevator that will offer the most suitable transport.

this passenger and offers the fastest and most comfortable ride. The identification of the elevator, usually a capital letter such as A,B,C, ... is displayed on the input terminal indicating to the passenger which elevator to take. An indicator in the door frame of the car confirms the destination this elevator will serve. This way, crowding of passengers in front of and within elevator cars is avoided. The performance of the elevator system is improved, because passengers with identical destinations can be grouped together and can thus reach their destination faster and more comfortable.

Miconic-10™ can identify each passenger as an individual being and obtain the following information about her personal needs or service requirements, before she has even entered the elevator:

1. The *origin* floor of the passenger is known from the position of the Miconic-10$^{TM}$ input terminal.
2. The *destination* floor is known from the number the passenger has input over the keypad.
3. If the wheelchair button is pressed on the terminal, the system knows that *special support for the disabled* is requested. For example, it activates spoken indication of floors within the elevator in order to facilitate the orientation of blind passengers. It informs the elevator that longer door opening and closing times should be observed and assigns more free space in the elevator cabin in order to accommodate a wheelchair.

Additionally, when identification devices are used, more individual requirements can be associated to each passenger:

- The *capacity requirement* is taken into account when selecting the elevator the passenger will be assigned to. For example, a passenger requires large space because she arrives with a shopping trolley.
- *Conflicting Passenger Groups*: Some groups of passengers should not meet within the elevator. For example, the room service delivering the breakfast and a room maid emptying trash bins should not meet in the elevator of a hotel for hygienic reasons. This means, if a passenger who belongs to a particular prespecified group requests transportation, the control must choose an elevator such that no encounters between conflicting passengers can occur inside a car.
- *Attended Travel*: Some passengers are not allowed to travel alone in the elevator, e.g., children. This means, at any moment of their travel, an accompanying person must be present in the car.
- *Nonstop Travel*: For some passengers one might offer a nonstop trip to their destination, i.e., the elevator does not execute intermediate stops to serve other passengers, but immediately reaches the destination of the passenger. For example, blind passengers know this way that the next stop is performed at their destination.
- *VIP service*: Another passenger might be identified as a VIP who is served by the elevator system with highest priority. For example, firefighters or medical emergency personal should be subject to VIP service. Other passengers may be delayed, i.e., they have to wait longer for transportation or might be required to travel a detour when boarding together with the VIP. VIP service always comprises nonstop travel.
- *Direct travel*: An annoying situation for a passenger in an elevator (except from the total brake-down of the system) occurs when she is transported opposite to the desired direction. For example, instead of going up from the lobby to the sky deck after the passenger has boarded in the lobby, the elevator travels down in order to pick up other passengers who have requested a trip from the parking garage to the sky deck. To avoid such confusing situations, we sometimes want to specify the requirement of *direct travel*, which informs the elevator that detours are not allowed (while intermediate stops are still possible) as long as a certain passenger is on board.
- *Access restrictions*: A visitor in a building with a certain security standard might be subject to access restrictions that the elevator system should observe, e.g., some floors are never served when certain passengers are present in the elevator.

Given this information, the elevator control software is required to assign the passenger to an elevator such that all requirements are satisfied. The presence of new functionalities such as space requirements, access restrictions, or VIP service makes this a tricky computational problem. Today, elevator systems can offer these services only in a very limited way by permanently or temporarily restricting the use of elevators. For example, today's VIP service is implemented by taking an elevator out of standard service, then sending this elevator to the VIP passenger, and—after the passenger has arrived at her destination—turning the elevator back to standard operating mode. It is impossible to integrate and embed these functionalities directly into the usual normal operation of a group of elevators.

At Schindler, we are currently developing new control software, which is able to guarantee these new functionalities and at the same time further improves on the performance of elevator systems. One part of this project has been devoted to the evaluation of new technologies that can handle the above mentioned requirements and that are open to future changes of elevator systems. AI planning technology is one of the promising candidates who has been in our focus. We are particularly interested in *domain-independent* planning approaches, because only these seem to offer the flexibility to integrate future changes without requiring a major change of algorithms and software. The basis of our investigation is the PDDL planning formalism (McDermott & others 1998), which emerged as the common platform for domain independent planners during the last years. In the following, we will describe our application in more detail and use PDDL to present one possible way of formalizing it.

On a first glance, the elevator control problem appears like a typical scheduling problem where the transportation jobs of the individual elevator cars have to be scheduled and optimized. But we obtain a very difficult instance of a scheduling problem where several jobs (the transportation requests) have to be scheduled in parallel on the same machine (the elevator). In this *cumulative* scheduling problem we are also faced with variable duration times of the individual jobs, because how long a passenger will need to reach her destination is influenced by the parallel scheduling. Consequently, none "off-the-shelf" algorithms are available to solve our problem.

When analyzing the problem from a planning perspective, it also seems to bear much of a typical planning task:

The *initial state* specifies where the elevator and the passenger are located. The *goal* is to serve all passengers. Possible actions in this domain model the operations of the elevator system. Interestingly, elevator control appears to be a typical representative of a number of *discrete* control problems, which can be successfully addressed using a planning approach, cf. the control of space crafts described in (Weld 1999). In this ideal technical environment, almost complete and reliable information is available. Besides this, state changes are discrete and the number of possible states and actions is rather limited. The problem of embedding planning (or until today, the computation of situation-specific control) and execution has been solved long ago, cf. (Closs 1970).

## Modeling of a Passenger

A passenger can be in three different states:

1. The passenger is *waiting* in front of the elevator door. The elevator has to stop first at the passenger's origin floor and subsequently at the passenger's destination floor.
2. The passenger has *boarded* the elevator. She is inside the elevator car and traveling to her destination floor that remains to be served.
3. The passenger has left the elevator at her destination floor. She has been successfully *served* by the elevator.

We introduce four predicates that characterize a passenger:

- *boarded(?p)* is true if passenger ?*p* has boarded the elevator, false otherwise.
- *served(?p)* is true if passenger ?*p* has left the elevator at the desired destination floor, false otherwise.
- *origin(?p,?f)* and *destin(?p,?f)* are static predicates that specify the origin and destination floor of a passenger. They are obtained from the input terminal and cannot be changed by the planning system.

Note that a passenger is waiting if she is neither *boarded* nor *served*. The three possible states of a passenger—*waiting, boarded, served* are characterized by the following truth value distribution of the predicates *boarded* and *served*:

| | waiting | boarded | served |
|---|---|---|---|
| served(?p) | FALSE | FALSE | TRUE |
| boarded(?p) | FALSE | TRUE | FALSE |

We also introduce an additional predicate that will be used later on to model the access restrictions that apply to a passenger within a building:

- *no-access(?p,?f)* is a static predicate and says that passenger ?*p* has no access to floor ?*f*.

In order to model the individual service requirements that we introduced in the previous section, we exploit the type hierarchy that is available in PDDL.

Every passenger is first of all of type *passenger*. If she is subject to further services, she belongs to one or *several* of the following subtypes of *passenger*:

- *going_up*: This and the subsequent type characterization are used to make the travel direction of a passenger explicit. A passenger of the type *going_up* travels upwards.

  *going_down*: A passenger of the type *going_down* travels downwards.

  Both subtypes will be used to guarantee that the requirement of *direct travel* can be fulfilled.
- *going_nonstop*: As the name suggests, a passenger of this subtype is subject to *nonstop travel*.
- *vip*: A passenger of this subtype is subject to VIP service and travels with highest priority and nonstop to her destination, i.e., she is served before any other passengers are served.
- *never_alone*: A passenger of this subtype is subject to the requirement of *attended travel*.
- *attendant*: A passenger of this subtype is a possible candidate to accompany a passenger of type *never_alone*.
- *conflict_A*, *conflict_B*: We introduce two conflicting groups of passengers, which should never meet in the car. More groups can be added easily, but this will render the planning tasks more and more difficult.

## Modeling of a Planning Task

A specific planning task can now be specified as follows:

The declaration of objects together with their types informs the system about the passengers who have requested the elevator service and specifies the floors that exist within the building.

The initial state description contains an *origin* and a *destin* fact for each passenger. It also contains *boarded* facts for all passengers who have boarded the elevator. If access restrictions apply, the corresponding *no-access* facts also get specified. Furthermore, we have to make explicit the topological structure of the building by introducing an *above*($?f_1, ?f_2$) predicate, which specifies for each pair of floors that above floor $?f_1$ lies floor $?f_2$. This representation is not very elegant, but here we are limited by the pure logical representation language. We cannot specify a transitive *above* relation, but need to explicitly enumerate its transitive closure. Finally, the position of the elevator is specified by introducing a *lift-at(?f)* fact.

The goal for the planning system is to achieve *served(?p)* for all passengers ?*p*. Figure 2 shows an example in PDDL syntax.

In this example, we are given a building with 7 floors. We only consider the planning problem of a single elevator. This elevator is currently located at floor 1 and

```
(define (problem example)
(:domain miconic)
(:objects p4 - passenger
          p3 - vip
          p6 - going_up
          p2 - going_nonstop
          p1 - conflict_A
          p5 - conflict_B
          f1 f2 f3 f4 f5 f6 f7 - floor)
(:init
(above f1 f2) (above f1 f3) (above f1 f4)
(above f1 f5) (above f1 f6) (above f1 f7)
(above f2 f3) (above f2 f4) (above f2 f5)
(above f2 f6) (above f2 f7) (above f3 f4)
(above f3 f5) (above f3 f6) (above f3 f7)
(above f4 f5) (above f4 f6) (above f4 f7)
(above f5 f6) (above f5 f7) (above f6 f7)
(origin p1 f2) (origin p2 f2) (origin p3 f1)
(origin p4 f7) (origin p5 f5) (origin p6 f6)
(destin p1 f4) (destin p2 f6) (destin p3 f4)
(destin p4 f2) (destin p5 f3) (destin p6 f7)
(no-access p4 f3) (no-access p4 f4) (no-access p4 f6)
(lift-at f1))
(:goal
(forall (?p - passenger) (served ?p))))
```

Figure 2: Specification of an elevator planning problem in PDDL.

6 passengers are waiting for transportation.[1] The type declaration informs the planner about the specific service modes that apply to each passenger. Passengers 1 and 5 should not meet each other because they belong to different conflict sets. Passenger 3 is a VIP who must be served first, while passenger 2 must travel without intermediate stops. Passenger 6 travels upwards and is thus subject to *direct travel*, while other passengers can possibly travel detours. Passenger 4 is not allowed to reach floors 3, 4, and 6 as specified in the initial facts.

## The Actions in the Elevator Domain

An elevator can perform three basic tasks: it can travel upwards, it can travel downwards, and it can stop at a specific floor.[2]

Two main features need to be modeled: First, we need to say when an elevator is allowed to stop at a given floor and second, we need to specify what effects this stop will have. The domain model is grounded on basic assumptions about the rational behavior of passengers:

[1]Note that no *boarded* facts are contained in the initial state of the example, i.e., the elevator is initially empty.

[2]At a more fine grained level, an elevator can also open and close doors, but we do not need to go into such a detailed description because a stop can be seen as comprising the opening and closing of the doors and it would also make the planning problem for a planning system much harder.

- If an elevator stops at a floor, then all passengers who are waiting at this floor for transportation, will usually enter the car unless no space is available.
- All passengers who are inside the car will leave it when the car stops at their destination floor.

We observe that the behavior of passengers cannot be planned -an elevator cannot force a passenger to travel beyond the desired destination once it has stopped at this floor. Furthermore, only a stop of the elevator (including the opening and closing of the doors) can impose a change of state of a passenger.

We introduce now a first basic variant of the **stop** operator, which reflects our observations. In this variant, no specific precondition is required to stop and the conditional effects of the operator model the typical behavior of passengers:

**stop**(*?f:floor*)
:precondition
:effect

$\forall ?p{:}passenger \quad origin(?p,?f) \wedge \neg\, served(?p)$
$\qquad \Rightarrow boarded(?p)$
$\forall ?p{:}passenger \quad destin(?p,?f) \wedge boarded(?p)$
$\qquad \Rightarrow \neg\, boarded(?p) \wedge served(?p).$

We want to emphasize that the state transitions of passengers from $\neg served(?p)$ to $boarded(?p)$ and from $boarded(?p)$ to $\neg boarded(?p) \wedge served(?p)$ are side-effects of the **stop** operator and are—to the best of our knowledge– most adequately modeled using quantified conditional effects. We also remark that two stops are necessary in order to achieve the two state transitions each passenger must pass through and that passengers with coinciding origin or destination floors can be served with the same stop. A planning system is now able to plan a sequence of **stop** actions such that all passengers reach their destination. It remains to model the individual service requirements that belong to each passenger. This can be achieved by adding additional preconditions to the **stop** operator that limit its applicability to only legal situations.

## Modeling of Service Requirements

In order to model the individual services, the planner needs to supervise additional conditions when passengers change their state:

1. A passenger is waiting at her origin. If the elevator stops, the passenger will board the elevator and can cause a possible conflict with other passengers who are also boarding or who are traveling in the elevator and will not leave the car at this stop.
2. A passenger is traveling in the elevator car and has not yet reached her destination. If the elevator performs an intermediate stop in order to serve other passengers, the entry of these passengers can cause a conflict with the traveling passenger.

Consequently, the precondition of the **stop** operator needs to be augmented such that when a stop is planned at a floor $?f$, the planner checks if any passengers are still waiting or if traveling passengers will remain in the elevator. The following disjunction formalizes both conditions:

$$\exists ?p : passenger \left[\overbrace{\neg served(?p) \wedge origin(?p, ?f)}^{waiting\ for\ the\ elevator}\right] \vee \left[\underbrace{boarded(?p) \wedge \neg destin(?p, ?f)}_{traveling\ in\ the\ elevator}\right]$$

It is important to observe that the predicates *origin(?p,?f)* and *destination(?p,?f)* are static and can be evaluated already during instantiation.

The implementation of the service requirements requires to supervise passengers at the different stops (origin, intermediate, destination) they have to transit through, cf. Figure 3.

| service | origin | interm. stop | destin. |
|---|---|---|---|
| conflicting passengers | ● | ● | |
| attended travel | ● | ● | ● |
| access restriction | | ● | ● |
| vip | | ● | |
| nonstop travel | | ● | |
| capacity requirement | ● | | |
| direct travel | | ● | |

Figure 3: Supervision of services at the origin, intermediate, or destination stop of a passenger who is subject to the service.

## Conflicting Passenger Groups

At each stop, the elevator has to make sure that a joint travel of two passengers, who belong to conflicting groups is impossible. This means, if a passenger $?p$ of the conflict set $A$ is waiting at floor $?f$ or if $?p$ is traveling in the car and the floor $?f$ is not her destination, then all passengers $?q$ of the conflict set $B$ must satisfy the following conditions:

1. $?q$ is not on board of the elevator
2. $?q$ has already reached her destination, i.e., she is in the *served* state, or her origin is another floor than $?f$.

If $?q$ is not on board of the elevator, then no conflict can be caused if $?p$ enters the same car. If $?q$ has already reached her destination, then no conflict can be caused by a simultaneous boarding of both passengers and also the situation that $?p$ is in the elevator (and will remain there!) and $?q$ is boarding cannot occur. On the other hand, a passenger leaving an elevator can resolve a potential conflict when a conflicting passenger wants to board at the same floor. Conflicting passengers at floors different to the currently planned stop cannot cause a conflict. We add therefore the following precondition to the **stop** operator:

$$\begin{aligned}&\exists ?p : conflict\_A \left[\neg served(?p) \wedge origin(?p, ?f)\right] \vee \\ &\quad \left[boarded(?p) \wedge \neg destin(?p, ?f)\right] \\ &\quad \Rightarrow \forall ?q : conflict\_B\ \neg boarded(?q) \wedge \\ &\qquad \left[served(?q) \vee \neg origin(?q, ?f)\right]\end{aligned}$$

The same condition must be formulated for all passengers of the conflict set $B$ in order to model the symmetry of this requirement:

$$\begin{aligned}&\exists ?p : conflict\_B \left[\neg served(?p) \wedge origin(?p, ?f)\right] \vee \\ &\quad \left[boarded(?p) \wedge \neg destin(?p, ?f)\right] \\ &\quad \Rightarrow \forall ?q : conflict\_A\ \neg boarded(?q) \wedge \\ &\qquad \left[served(?q) \vee \neg origin(?q, ?f)\right]\end{aligned}$$

## Attended Travel

This service requires that a passenger is never alone inside the elevator car. A simple solution is to choose an attendant who has to travel together with this passenger. A more elegant solution is to make sure that other passengers board and travel a part of the route together with the passenger. This means, the attendant can even change during the travel of the passenger who is subject to this service.

Thus, when stopping at a floor $?f$ the following conditions must be satisfied: If $?f$ is the origin of a "*never_alone*" passenger $?p$ and $?p$ is still waiting at $?f$ or if $?p$ is in the car and $?f$ is not her destination, then there must be another passenger $?q$ who is of type *attendant* and who satisfies one of the following conditions:

1. $?q$ is in the elevator and $?f$ is not her destination or
2. $?q$ is still waiting and her origin is $?f$.

In the first situation, the attendant will remain in the car, while in the second case, the attendant will board the car according to our model of behavior. We obtain the following precondition for the **stop** operator:

$$\begin{aligned}&\Big[\exists ?p : never\_alone \left[\neg served(?p) \wedge origin(?p, ?f)\right] \\ &\qquad \vee \left[boarded(?p) \wedge \neg destin(?p, ?f)\right]\Big] \\ &\quad \Rightarrow \\ &\Big[\exists ?q : attendant \left[\neg served(?q) \wedge origin(?q, ?f)\right] \\ &\qquad \vee \left[boarded(?q) \wedge \neg destin(?q, ?f)\right]\Big]\end{aligned}$$

## Access Restrictions

The multi-functional use of modern buildings requires to integrate security elements into the elevator control software, e.g., we want to make sure that passengers cannot access floors they are not allowed to reach by using the elevator. This means, an elevator can only stop at a floor if no access restriction is known for all passengers who are on board. When assuming that passengers can never wait in restricted areas, then only boarded passengers have to be supervised:

$$\forall ?p : passenger \left[boarded(?p) \Rightarrow \neg no\text{-}access(?p, ?f)\right]$$

An alternative representation of access restrictions is possible when using types, e.g., we introduce a subtype *visitor* and then modify the above precondition to

$$\forall ?p : visitor \quad boarded(?p) \Rightarrow ?f \neq c_1 \wedge \ldots \wedge ?f \neq c_k$$

where $c_1$ to $c_k$ are those floors, which visitors are not allowed to reach.

## Nonstop Travel

If a passenger has boarded the elevator, who has to be transported directly to her destination floor, i.e., without intermediate stops, then the next possible stop of the elevator must be the destination of this passenger. This is achieved by adding the following precondition to the **stop** operator:

$$\forall\ ?p : going\_nonstop \quad \left[boarded(?p) \Rightarrow destin(?p, ?f)\right]$$

Two passengers requesting nonstop travel at the same time are treated equally, i.e., one of them has to accept an intermediate stop if they travel in the same elevator.

## VIP Service

A VIP service can be offered in different ways. In the following variant, we require that a VIP request overrides all other requests and that the elevator must immediately serve the VIP origin and destination floors. This means that other passengers are delayed and may even travel a detour if their type classification does not exclude detours. After all VIPs have reached their destination, the remaining transportation requests can be served. This implies that either only empty elevators can serve VIP requests or an elevator is currently serving only passengers, for which no direct travel is required. The additional precondition that we add to the **stop** operator expresses that a stop is possible at a floor $?f$ if either all VIPs have already reached their destination or if $?f$ is the origin or destination of a VIP who is known to the planner.

$$\begin{array}{l} \forall\ ?p : vip \quad served(?p) \\ \vee \\ \exists\ ?p : vip \quad origin(?p, ?f) \vee destin(?p, ?f) \end{array}$$

So far, we have modeled the following services: *access restriction*, *nonstop travel*, *vip service*, *attended travel*, and *conflicting passengers*. These features can be handled by extending the **stop** operator with additional preconditions, which we introduced above. The formalization is based on the commonly agreed features of PDDL, which are also supported by various planners, e.g., IPP, PRODIGY, SGP, or UCPOP. Figure 4 summarizes the extended operator and displays it in PDDL syntax.

```
(:action stop
:parameters (?f - floor)
:precondition
(and (lift-at ?f)
(imply (exists (?p - conflict_A)
        (or (and (not (served ?p)) (origin ?p ?f))
            (and (boarded ?p) (not (destin ?p ?f)))))
        (forall (?q - conflict_B)
            (and (not (boarded ?q))
            (or (served ?q) (not (origin ?q ?f))))))
(imply (exists (?p - conflict_B)
        (or (and (not (served ?p)) (origin ?p ?f))
            (and (boarded ?p) (not (destin ?p ?f)))))
        (forall (?q - conflict_A)
            (and (not (boarded ?q))
            (or (served ?q) (not (origin ?q ?f))))))
(imply
        (exists (?p - never_alone)
        (or (and (origin ?p ?f) (not (served ?p)))
            (and (boarded ?p) (not (destin ?p ?f)))))
        (exists (?q - attendant)
        (or (and (boarded ?q)
            (not (destin ?q ?f)))
        (and (not (served ?q)) (origin ?q ?f)))))
(forall (?p - going_nonstop)
        (imply (boarded ?p) (destin ?p ?f)))
(or (forall (?p - vip) (served ?p))
        (exists (?p - vip)
            (or (origin ?p ?f) (destin ?p ?f))))
(forall (?p - passenger)
        (imply (no-access ?p ?f) (not (boarded ?p)))))
:effect (and
(forall (?p - passenger)
        (when (and (boarded ?p) (destin ?p ?f))
            (and (not (boarded ?p)) (served ?p))))
(forall (?p - passenger)
        (when (and (origin ?p ?f) (not (served ?p)))
            (boarded ?p)))))
```

Figure 4: STOP Operator in PDDL Syntax.

## Beyond the Expressivity of PDDL

Three main aspects remain to be modeled, which will yield us beyond what current domain-independent planning systems can handle:

First, we introduced the service that passengers should travel directly without detours, which has not been respected so far. This requirement implements a kind of fairness strategy, which guarantees that passengers— once they have boarded the elevator— will approach their destination and never visit floors that are not located between their origin and their destination. In certain situations where direct travel renders a planning problem unsolvable, we would also like to relax this restriction and allow a passenger to travel small detours, with "small" being situation- and/or passenger-dependent.

Secondly, the capacity of the elevator cars needs to be respected. So far, our domain model assumes that *all* passengers who are waiting in front of the elevator will board when the car stops and doors open. But this is only true, if the car has enough space available in order to accommodate all passengers. Otherwise, some of them will be left behind and even worse, the planner has no information about which of the passengers are inside the car and which of the passengers are still waiting. It has to assume that all designated passengers are on board and thus must stop at all their destinations. But this is in fact the wrong behavior as the elevator should only serve destinations of passengers who are indeed on board, while destinations of passengers who are left behind can be canceled, but their origin has to be served again by this or another car. It is therefore necessary to make sure that there is enough space to accommodate all waiting passengers.

Third, we would like to generate plans that satisfy additional criteria, i.e., they must not only be correct, but also be *optimal* solutions given an optimization function that is externally defined.

## Direct Travel without Detours

The requirement of *direct travel* can be modeled in several ways. When restricting ourselves to PDDL, we need to introduce two additional operators that model the upward and downward movements of the elevator and for each directly traveling passenger we need to add the information whether she is traveling downwards or upwards by typing her as *going_up* or *going_down*.

The location of the elevator is represented using the *lift-at(?f)* predicate and the **stop** operator requires as an explicit precondition that the elevator must be at the designated floor. The *above*$(?f_1, ?f_2)$ predicate, which we introduced in the beginning, specifies that floor $?f_2$ lies above floor $?f_1$.

The additional **up** and **down** operators guarantee that the elevator never travels opposite to the direction of a passenger whose traveling direction has been made known to the system.

```
up(?f₁,?f₂:floor)
:precondition $lift\text{-}at(?f_1) \wedge above(?f_1,?f_2) \wedge$
              $\forall\ ?p{:}going\_down\ \ \neg\ boarded(?p)$
:effect $lift\text{-}at(?f_2) \wedge \neg\ lift\text{-}at(?f_1)$.

down(?f₁,?f₂:floor)
:precondition $lift\text{-}at(?f_1) \wedge above(?f_2,?f_1) \wedge$
              $\forall\ ?p{:}going\_up\ \ \neg\ boarded(?p)$
:effect $lift\text{-}at(?f_2) \wedge \neg\ lift\text{-}at(?f_1)$.
```

Alternatively, one can define a total ordering over the floor names by mapping names to ordinal numbers

$$ord:\ ?f \longrightarrow N$$

and then augment the precondition of the **stop** operator by a test, which verifies that a passenger with direct travel never visits floors that are not located (ordered) between her origin and destination, i.e., the number of each floor has to lie in the interval having as bounds the numbers associated with the origin and the destination of this passenger. Such a representation would avoid the introduction of additional operators and keeps plans shorter and only composed out of **stop** actions. But unfortunately, we found no representation formalisms neither planning algorithms which support this kind of reasoning.

## Space Requirements and Elevator Capacity

So far, the boarding and exiting of passengers occurs as a side effect of a stop of the elevator. This means, the capacity of the elevator is a resource, which is affected by a universally quantified conditional effect. There are no reasonable ways of modeling the capacity in a purely logical formalism. As a work around one could identify space areas in the elevator and model them with a logical constant. Each passenger would then be allocated to a particular space in the elevator. But many planning systems would then start to permute passengers over spaces during search. Although, symmetry analysis such as it is available in STAN (Fox & Long 1999) can prevent this, such a domain representation appears rather questionable.

There are a few investigations in the recent planning literature (Wolfman & Weld 1999; Kautz & Walser 1999; Rintanen & Jungholt 1999; Walser, Iyer, & Venkatasubramanyan 1999; Vossen *et al.* 1999; Koehler 1998) which investigate resource-optimal or resource-constrained planning, but none of them can handle conditional resource effects. The elevator domain is an interesting example of a real-world resource-constrained planning problem, where the planner needs to respect the available space in the elevator, but no optimization of the space usage is required. To model the space constraint one could introduce a resource variable $\$C$ which records the availability of free space in the elevator. It obtains an initial value, which equals the elevator capacity if the car is empty. For each passenger, we know her individual capacity requirements $\$C(?p)$ or assume a standard value. Boarding passengers decrease the value of $\$C$, while exiting passengers increase it relatively to the capacity that is available before the elevator stops. To make sure that the value of $\$C$ never exceeds the predefined limit, one could either use a representation language that declares predefined intervals for resources and an algorithm that checks that no world state violates these declarations. Alternatively, the planner has to verify the available capacity in the antecedent of the conditional effect.

## Generation of Optimal Plans

The lack of optimization capabilities is an important missing feature in today's planning systems. For example in this domain, one would like to minimize the traveling or waiting times for all passengers. Given a plan as an ordered sequence of stops, one can easily determine its traveling costs because we know how long a ride between two given floors will take (the planner could look up the corresponding values in an externally maintained database), how long each stop needs to last such that all passengers can enter and exit, and which of the passengers will enter and exit according to their modeled behavior. It is not very difficult to develop a domain-specific search algorithm which is combined with a branch-and-bound approach to achieve any desired optimization. But domain-independent planners totally lack this feature and are—if at all—able to generate shortest plans, e.g., if they are based on Graphplan.

## Conclusions

The advantage of using a general purpose planner lies in the availability of an automated reasoner, which is able to solve the desired planning tasks within some reasonable time. No implementational work is necessary and all effort can be concentrated on the domain representation. The formal semantics of plan representation languages allowed in our case to study different interpretations of the various service requirements and to develop a formal specification for them. One can now formally prove that the domain representation possesses certain properties. For example, it meets the requirement that all passengers will reach their destination if their behavior corresponds to the underlying model, which is normally the case. A formal proof is possible by formulating and proving invariants of domain properties that are valid over all states. Unfortunately, this has to be done by hand because no domain modeling tools are available. There are a few attempts described in the literature, e.g., (Biundo & Stephan 1996; 1997) but this aspect of planning has not been in the main focus of planning research. Besides this, we do not know if the above representation is the best possible one. Even further, it is also not clear which properties define the quality of a domain model and how they can be evaluated.

The strengths and weaknesses of today's *domain-independent* planning systems are the following:

+ experimental tool to explore domain specifications,
+ general-purpose problem solver,
+ representation languages allow to model quite complex behaviors,
+ formal semantics of representation languages supports specification of behaviors and proof of domain properties,
- no support of metric/resource constraints,
- no consideration of cost functions during planning,
- lack of optimization capabilities. requirements.

We hope the domain provides an interesting testbed for other researchers. The operators have complex preconditions and effects, and we have identified open issues PDDL cannot handle. The PDDL domain is available at http://www.informatik.uni-freiburg.de/~koehler/elev.tar.gz. The domain can be scaled across several dimensions and modifications and extensions should be easy to incorporate:

- Increasing numbers of passengers and floors can be specified.
- The topological structure of the building can be changed. Instead of simple vertical hoist ways one can also describe a branching structure in which a potentially unlimited number of elevator cars can move.
- Different subsets of service requirements can be used (together with the corresponding typing of passengers) and also new services can be added.

The current development at Schindler is not based on a domain-independent planning algorithm, but uses a problem-specific optimization algorithm. It is based on the domain model we introduced here, but translates the declarative specification of the service requirements into a piece of code. We consider the loss of declarativity and the dependency on a specific implementation to be the main disadvantage of domain-specific solutions. Each slight change in the specification requires modifications of the software, which also renders the experimentation with inhouse simulation tools much more labor-intensive.

## References

Biundo, S., and Stephan, W. 1996. Modeling planning domains systematically. In *ECAI-96*, 599–603.

Biundo, S., and Stephan, W. 1997. System assistance in structured domain model development. In *IJCAI-97*, 1240–1245.

Closs, G. 1970. *The Computer Control of Passenger Traffic in Large Lift Systems*. Ph.D. Dissertation, University of Manchester.

Fox, M., and Long, D. 1999. The detection and exploitation of symmetry in planning problems. In *IJCAI-99*, 956–961.

Kautz, H., and Walser, J. 1999. State-space planning by integer optimization. In *AAAI-99*.

Koehler, J. 1998. Planning under resource constraints. In *ECAI-98*, 489–493.

McDermott, D., et al. 1998. *The PDDL Planning Domain Definition Language*. The AIPS-98 Planning Competition Comitee.

Rintanen, J., and Jungholt, H. 1999. Numeric state variables in constraint-based planning. In *ECP-99*.

Vossen, T.; Ball, M.; Lotem, A.; and Nau, D. 1999. On the use of integer programming models in ai planning. In *IJCAI-99*, 304–309.

Walser, J.; Iyer, R.; and Venkatasubramanyan, N. 1999. An integer local search method with application to capacitated production planning. In *AAAI-99*.

Weld, D. 1999. Recent trends in planning. *AI Magazine* 20(2):93–123.

Wolfman, S., and Weld, D. 1999. The LPSAT engine and its application to resource planning. In *IJCAI-99*, 310–316.