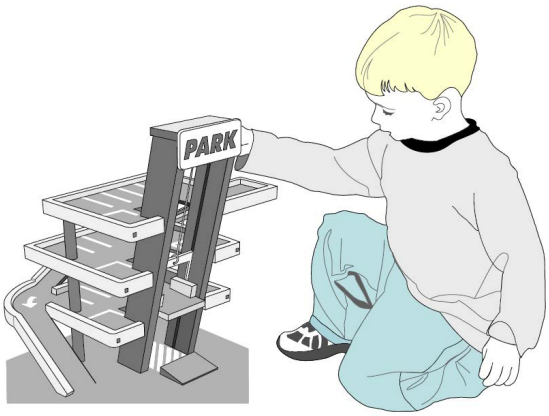
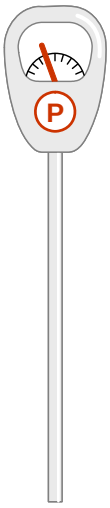
Software Engineering Course Project

Parking Garage/Lot

This project develops a computerized system to manage parking usage and online reservations in a parking garage. It is intended to be done by a team of 4-6 undergraduate students during an academic semester, in conjunction with lectures and other class activities.



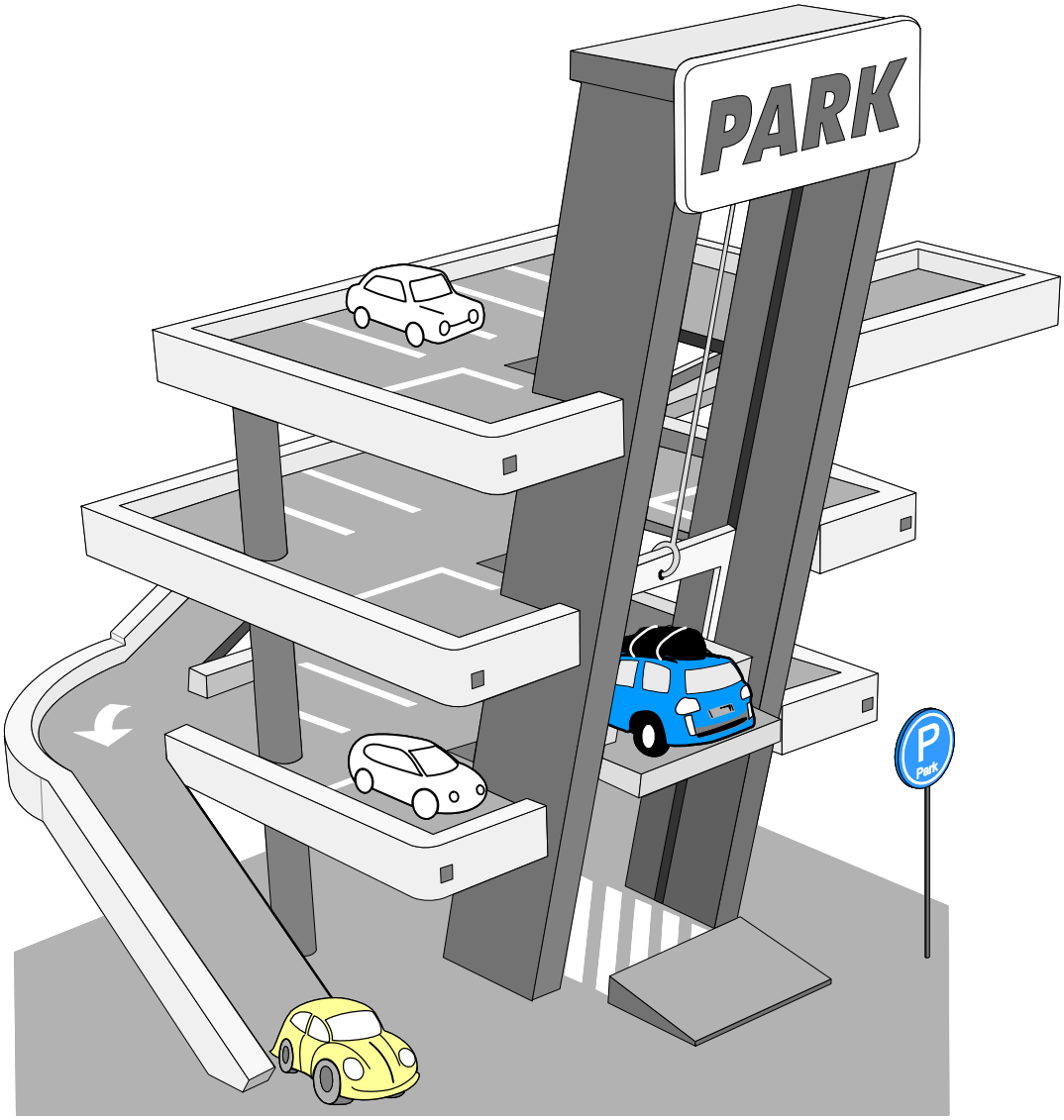


1. Project Description

The purpose of this project is to track and manage occupancy of a parking garage and allow customers to find and reserve available parking places.

The system-as-is can be described as follows. The parking garage currently operates without any computerized system. The management has concerns about inefficiencies of sub-optimal usage of parking space (lost opportunity/profit). Congestion inside the garage is often caused by drivers searching for vacant spots. In addition, it is well known that a great deal of traffic congestion in cities generally is caused by drivers looking for a parking space. Currently, the management monitors the garage occupancy by having employees walk around the decks to inspect the occupancy of individual spots. Some parking garages already monitor the occupancy using a

sensor system which keeps tally of the vehicle entrance and exit events that occur in the parking structures. Our customer garage decided to develop a more advanced system, called “Park-a-lot,” as described next.



The parking garage will be remodeled so that the parking decks above the ground level will be accessible only using an elevator that will lift the vehicles to different decks. There will be no other way to reach the upper decks. All vehicles will depart the garage by descending down the designated exit pathway to the ground level. Only passenger vehicles can be parked in this parking garage. That is, large trucks, busses, etc., cannot enter this parking garage. The ground level will be reserved for walk-in customers. All other levels will be reserved for registered customers that made advance reservation. The parking garage will *not* distribute special electronic tags or markings for customer vehicle identification. Instead, the garage will use camera-based license-plate recognition systems.

Customers will *register* at the company website in advance of using the parking garage. At the *registration time*, the customer will provide demographic information and a valid email and his or

her credit card number. The customer may provide the license plate numbers for his or her vehicle(s), but this is not required to allow registration of customers who do not own vehicles, but will use a borrowed or rented vehicle. The same vehicle may appear under several customers, for example different family members, or vehicle borrowed from a friend, or rented from a rental agency. In case of a borrowed or rented vehicle, the customer will specify the registration plate number at the time of making a parking reservation. If the specified registration number is different from the number in the customer's profile, the software-to-be will create a temporary association of the number to this customer, and the association will be deleted after the parking garage is used during the requested interval.

The registration software may also support **guaranteed reservations**, which allow customers to make a (monthly) *contract* with the parking garage for a parking spot. For example, commuters going regularly to work need parking on a daily basis during a predetermined period. Another example is corporate customers who wish to keep permanently reserved parking space for their personnel and visitors. Such customers are desirable because they can provide predictable and steady income.

The following **devices** will be installed inside the parking garage (see Figure 1):

S1 & S2. The garage will have installed two *license-plate readers*: one at the lift platform and the other at the end of the exit pathway. The reader will use a digital camera and a license-plate recognition system widely used in toll stations in road-tolling systems. When a vehicle drives up on to the lift platform, the license-plate reader will read the vehicle registration number. The other reader will record the registration number of the departing vehicles.

S3. Every parking spot has installed a sensor that senses the occupancy of the spot by a vehicle. The sensor could be based on visible or infra-red light, ultrasound, or a similar sensing technique.

D1. There will be a digital display installed on the ground floor that will indicate available vacancies for the walk-in customers without reservations. This display will also indicate if the ground-level parking area is full.

D2. There will be a digital display installed in the vehicle elevator to display various messages. Other messages will include information for non-registered customers of a denied access to upper decks, or information for registered customers of changes in their reservation.

D3. An entrance console will be installed in the vehicle elevator. If the vehicle registration number is not recognized by the license-plate reader (e.g., because this is a borrowed or rented vehicle and the customer did not know the license plate number at the time he or she made the parking reservation), then the system will display the message for and offer the driver option to enter the *reservation confirmation number* on the console installed in the vehicle elevator.

To prevent the drivers from entering the upper decks via the exit driveway, there will be a one-way barrier installed at the endpoint of the exit driveway.

1.1 Business Policies for the System-to-be

The company has decided to adopt the following **business policies** for the system-to-be:

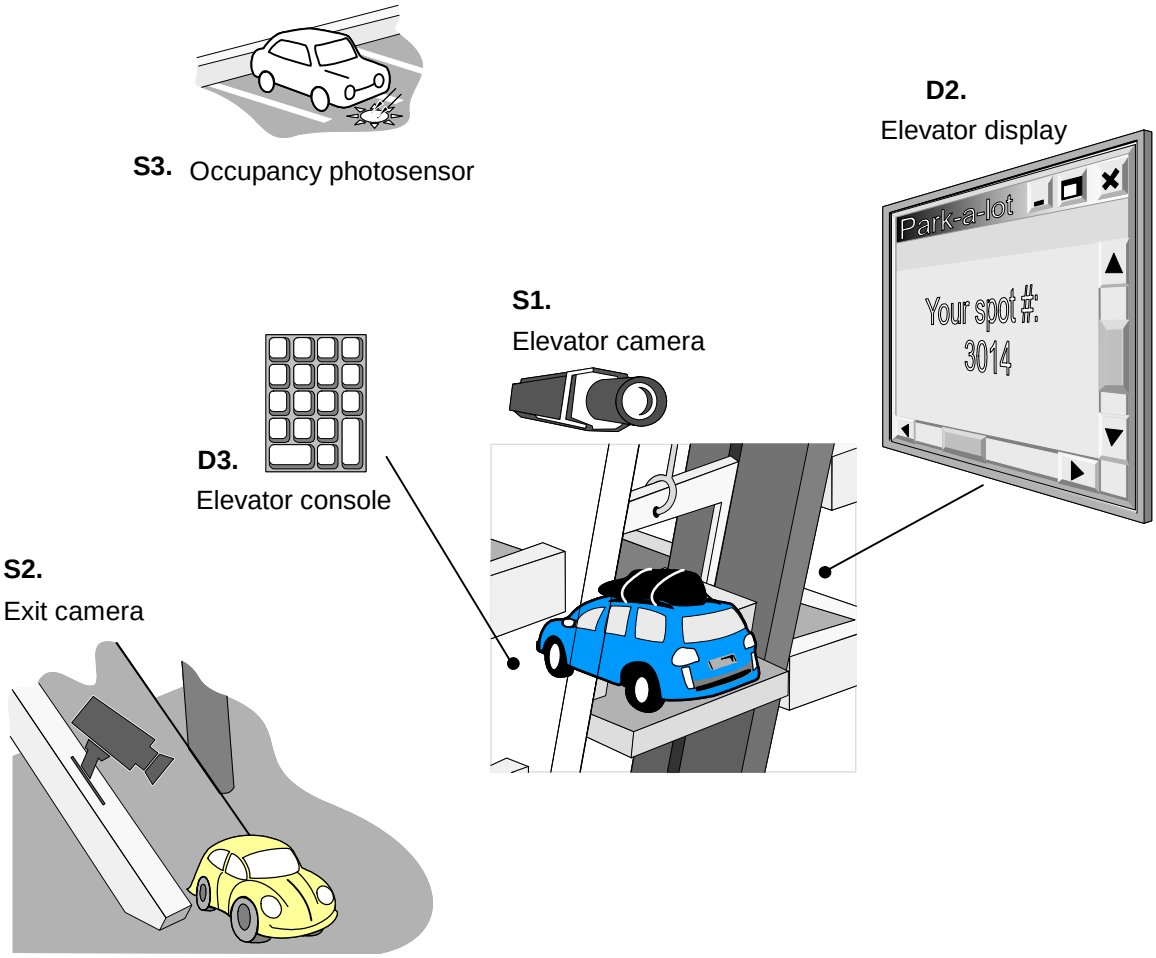


Figure 1: Parking garage system-to-be: sensors, displays, and other devices.

P1. If the license-plate recognition system does not recognize the vehicle registration number as associated with a registered customer (e.g., a walk-in customer drives on to the lift platform by mistake), the platform will remain motionless and the display board will notify the driver that the registration number is not recognized, and request to enter their reservation confirmation number or membership number (as per the business policy (**P2**) described next) on the console installed in the vehicle elevator, or back away from the platform and park on the ground level. Vehicles with a missing license plate will be treated the same as those with an unrecognized registration number.

P2. A registered customer may be allowed to *walk-in without a reservation* if there are currently available parking spots. If the vehicle registration number is recognized, but the system cannot find an existing reservation associated with the customer who owns this vehicle, then the customer will be offered to specify the expected duration of parking or departure time using the console installed in the vehicle elevator. If the vehicle registration number is *not* recognized, the customer will be offered to type in their *membership number* and the expected parking duration. If the membership number is not recognized, the customer will be asked to back away.

P3. If a customer does not show up at the start of their reserved interval, the parking spot will be held reserved for a given “grace period” (e.g., one-half hour) after the start of the reserved

interval. If the customer arrives within the holding period, he or she shall park on their reserved spot and will be billed for the full reserved period. The customer will be offered to pay an additional fee to hold the reservation beyond the regular grace period.

P4. If the customer arrives any time after the grace period after the start of the reserved interval, he or she will be asked for how long he or she plans to stay. If there are vacant and unreserved spots during the desired interval, the customer will be offered to park. The customer will be billed from the start of their original reservation until the end of their newly reserved interval.

P5. If the customer does not arrive during their reserved interval, he or she will be billed for the entire duration of their reserved interval.

P6. If a customer departs before their reserved period expires ("understay"), he or she will be billed for the full reserved period. The spot's status in the database will be changed to "vacant" immediately as the sensor detects that the vehicle has left, and the spot will be made available to other customers for use.

P7. The customer is allowed to extend an existing reservation one-half hour prior to the scheduled expiration if only if there are available (not reserved or occupied spots) for the desired period. The reservation may be extended unlimited number of times if the extension is requested one-half hour prior to the scheduled expiration if only if there are available for the desired period.

P8. If a customer fails to depart as scheduled after their reserved period expires, he or she will be billed for the duration of their reserved period at a regular rate and at a higher rate for the duration of their overstay. The rate will be increased progressively with the duration of overstay. A notification will be sent to the customer about these actions.

P9. Each customer is allowed to have multiple standing reservations on his or her names, but these reservations cannot be contiguous. A minimum of one hour gap is imposed between any consecutive reservations, and a maximum of three outstanding reservations is allowed. If a customer tries to make contiguous reservations, he or she should be offered to merge the contiguous reservations into a single one, or to cancel or modify some of the contiguous reservations.

P10. If a customer arrives and his or her reserved spot is still occupied by a previous customer who failed to depart as scheduled, but there are other available spots, the arriving customer will be offered to park on an available spot. The message will be displayed on the display inside the vehicle elevator.

P11. The system may overbook the parking space reservations. The overbooking mechanism will be described below.

P12. If a customer arrives on a *full parking garage*, because of overbooking or some customers failed to depart as scheduled, the customer will be asked to leave without parking, and will be given a rain check.

P13. The registered customer is billed once a month by emailing a **monthly statement**, which includes parking fees or penalty fees, if any.

P14. If the recognized vehicle registration number is associated with a single registered customer, the garage usage will be billed to this customer. If the vehicle is currently associated with more than one registered

customer, the vehicle will be billed to the customer with the current temporal association with this vehicle number, which is created at the time of the parking reservation request.

The practice of **overbooking** (policy (**P11**))—accepting reservations for more spots than are available by forecasting the number of no-show reservations, overstays, understays, and walk-ins, with the goal of attaining 100 percent occupancy—is common in many service industries, such as airlines and hotels. For example, the book *Hotel Front Office Management* (by James A. Bardi, 4th Edition, John Wiley & Sons, Inc., Hoboken, NJ, 2007) describes how overbooking works in the hotel industry.

The operator always wants to minimize the financial loss due to no-shows and other issues. The occupancy management policy is based on management of the occupancy categories into which customers are placed: those with confirmed reservations, those with guaranteed reservations, overstays, understays, and walk-ins. The reservation is guaranteed with a credit card number on record in the customer's profile, to ensure the customer's intent of arrival and thus guarantee payment for parking services. Here is a (partial) glossary of terms used in overbooking decisions:

Confirmed reservations represent registered customers who make reservations as the need arises. These reservations are honored until a specified time (including the grace period after the start of the reserved interval). Such customers represent the critical element in no-shows.

Guaranteed reservations represent the registered customers who made a *contract* with the parking garage for a parking spot, such as commuters going to work who need parking on a daily basis during a predetermined period. Such customers represent a less volatile group because they need to show up for their work.

Overstays are currently parked customers who wish to extend their stay beyond the time for which they made reservations.

Understays are customers who arrive on time but decide to leave before their predicted time of departure.

Walk-ins are registered customers who arrive to the parking garage without a contract or reservation. They are welcome because they can enhance the garage occupancy percentages (if there are available parking spaces).

The following occupancy management formula considers confirmed reservations, guaranteed reservations, no-show factors for these two types of reservations, predicted overstays, predicted understays, and predicted walk-ins to determine the number of additional parking reservations needed to achieve 100 percent occupancy. No-show factors are based on prior experience with customers with confirmed or guaranteed reservations who did not show up.

$$
\begin{aligned}
&\text{total number of parking spots available} \\
&- \text{confirmed reservations} \times \text{no-show factor based on historical data} \\
&- \text{guaranteed reservations} \times \text{no-show factor based on historical data} \\
&- \text{predicted overstays} \\
&+ \text{predicted understays} \\
&- \text{predicted walk-ins} \\
&\hline \\
&= \text{number of additional parking spots available to achieve 100 \% occupancy}
\end{aligned}
\qquad (1)
$$

1.2 Assumptions about the System-to-be

We identified the following **assumptions** about the parking garage system-to-be:

A1. The license-plate recognition system works correctly 100 % of time, with no errors because of dirty or scratched license plates. If the registration number cannot be recognized, the software-to-be will assume that this vehicle does not belong to a registered customer.

A2. If the license-plate recognition system cannot recognize the vehicle registration number and the customer does not provide a valid confirmation number or membership number, the customer will be asked (display message) to back away from the vehicle elevator (business policies (**P1**) and (**P2**)). We assume that the customer will always obey and depart. If we suspect otherwise, we may manage this risk by having the system to notify the garage security personnel. We also need an additional sensor to sense a vehicle presence in the elevator and assumptions about its correct operation. (Perhaps the license-plate recognition camera can serve this purpose?)

A3. The spot-occupancy recognition system works correctly 100 % of time, with no errors because of sensor malfunctioning or incorrect sensing. If the sensor positively detects occupancy, we assume that this result is only due to a vehicle, and no other object occupying the spot. For example, if the sensor is based on visible light, we assume that "dark" is sensed is only because a vehicle is present above the sensor, rather than because of lighting outage or some other condition; similarly, "light" is sensed is only because of a vacant spot and not because of another accidental light source.

A4. The vehicle elevator will lift the vehicle to the appropriate deck, and will never stop on a wrong deck by mistake.

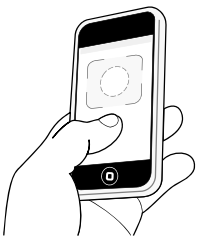
A5. In the case of the business policy (**P10**), we assume that the customer will always accept the offered parking spot and will never wish to opt out and leave the parking garage without parking his or her vehicle.

A6. We assume that the customer will always park at their assigned spot, and will never park at an arbitrary vacant spot. We assume that the garage currently does not have installed sensors for continuous tracking of customers from the vehicle elevator to their assigned spot. This is a strong assumption (and we know that people are unpredictable!), but the accuracy of the parking reservations table depends on this assumption. If a customer parks on a wrong spot (e.g., spot *B*), and the system thinks that the customer is parked correctly on spot *A*, then the system will direct future customers to an already occupied spot *B* (or accept reservations for *B*), and meanwhile the system will consider spot *A* as occupied, while it is actually available.

A7. If the license-plate reader successfully recognizes the vehicle registration number, the system assumes that the driver is a registered customer. The system-to-be will not consider separately scenarios where the driver is a non-registered customer who borrowed the vehicle from a friend who is a registered customer, or if the vehicle was stolen from a registered customer. The customer to be billed for the parking garage use will be decided as per the business policy (**P14**).

A8. It is possible that the customer is an organization with multiple individuals (rather than an individual person).

A9. We assume that the customer has access to email and a mobile phone with SMS texting capabilities. We do *not* assume that the customer will regularly check his or her email or will always be able to receive instantaneously SMS messages. For example, the customer may be in a meeting with the phone turned off. Also, we do *not* assume that the customer has access to a computer or smartphone while driving.



The above description of the system is only preliminary, provided as a reference example. The student developers team may add, drop, or modify any of the statements as deemed appropriate. Also, the team should consider how will the system functioning be affected by scenarios in which the above assumptions are not satisfied.

1.3 Statement of Requirements for User Interaction

The architecture of the envisioned parking garage computer system is shown in Figure 2. A relational database is maintained at the server. The database contains various information, including:

- Information about the registered customers
- The occupancy state of each parking spot: “available,” “reserved,” or “occupied”
- Current parking reservations
- The record of transactions for each customer, such as past reservations, garage usages, whether the customer showed-up late, or failed to show up during their reserved period, etc.
- Various statistics about the garage usage

The garage operator should be able to view but not edit the profiles of registered customers. The operator should also be able to set the prices for different services, such as parking fee within the reserved period, parking fee during overstays, and the fee for no-shows.

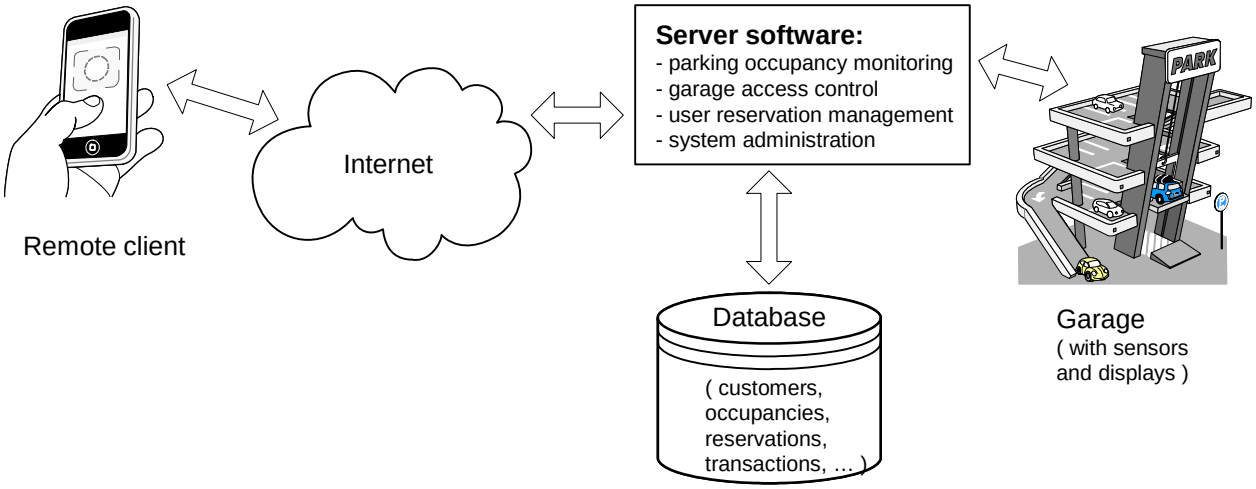


Figure 2: The architecture of the parking garage computer system.

The customer should be able to check the parking space availability by specifying the desired date and time interval, using a client device such as Web browser or a smart phone app. If the system responds stating that there are available spots, the customer should be able to make the parking *reservation*. Upon successful reservation, the customer is issued a *reservation confirmation number*.

The customer should be able to *modify* their existing reservation(s) before the starting time of a particular reservation.

The customer should be able to *extend* their current occupancy of a parking space (in case they realized they cannot depart as scheduled).

The customer should be notified about the *identifier* (e.g., number) of their specific parking spot. Because the parking does not have installed a driver-guidance system, the customer will use this identifier to locate their parking spot in the garage.

If the license-plate recognition system in the vehicle elevator does not recognize the car's registration number, the customer should be able to type in their reservation confirmation number and enter the parking garage.

User Interface Design Issues

If the remote client is run on a smartphone or another small device, the interface should be simple for quick and easy interaction. This is particularly true because the user may be engaged in some other activity, such as driving. Therefore, the number of data entries required by the user should be kept at a minimum that is required to support the given functionality.

Notice that the business policy (**P10**) is ambiguous about whether the customer can reserve a specific spot, and how the assigned spot can be changed under certain conditions. We have two options to consider. One option during the reservation process is to show the the availability map of the entire parking garage, and allow the user to see the state of each parking spot: "available," "reserved," or "occupied." The user would be able to select and book a specific available spot. Allowing the customer to select the desired spot at the reservation time may appear as a useful feature. However, if unanticipated events happen (e.g., previous customer overstayed), then the system needs to modify the reservation and notify the customer about the changes in the spot assignment. We cannot do this notification before the customer arrives, so unexpected changes at the arrival may annoy the customer. Given that it is possible that the customer may need to be relocated because the previous customer overstayed, or to optimize the parking space utilization, the relocation may occur frequently enough to make it annoying for customers who made specific reservation. The downside of allowing manual spot selection is that the system may appear to be flaky and unreliable. Finally, the benefits of giving the customer the choice of the parking spot are not clear. It is not clear that allowing the user the choice would somehow increase user convenience or user satisfaction, or contribute in some other way.

Another option is *not* to allow specific choices. The customer would not know his or her assigned spot until they arrive to the garage, and only at the arrival time the system would inform the customer about his or her assigned spot number. Automatic spot selection has an advantage of simplifying the user interface, because there is no need to show the availability map and support the spot selection. This simplification makes the interface easy to develop as well as easy to use,

which is important for customers who may try making parking reservation while driving. A simple user interface may even support voice-based reservations, making it easy to make reservation while driving.

The developer team should consider the above options and explain their arguments for the design that they will eventually choose.

The developer(s) should count the number of clicks/keystrokes that are necessary to accomplish individual tasks. This is particularly important for the remote client interface that allows making the reservations. The customer may need to make a reservation in hurry, perhaps even while driving. Make every effort to reduce the number of clicks/keystrokes in the system interaction, while not compromising functionality and security.

2. Parking Garage Simulator

This project relies on a parking garage simulator instead of working with a real parking garage. The simulator program will simulate the physical garage with its sensors and displays. It will allow the user to pretend entering the garage and parking the vehicle. The server-side software shown in Figure 2 will include a provisional user interface that allows the user to enter data that in a real system will be captured by physical sensors. The interface should look as shown in Figure 3. The left-hand side (Figure 3(a)) shows how the future system-to-be would look like once it is connected to an actual garage. The right-hand side (Figure 3(b)) shows a current improvised system.

Our system shown in Figure 1 envisions two types of sensors: occupancy photosensors on each individual spot and a camera for license-plate recognition. Instead of entering the occupancy state for all spots, we will temporarily assume that customers will enter the garage one-at-a-time. As shown in Figure 3(b), it is enough to have a single text field to input the spot ID and two radio-buttons to specify the occupancy state of that spot. In addition, the user will enter the license-plate that will normally be obtained from the camera-based recognition system.

The provisional solution in Figure 3(b) allows us to develop the software without having access to an actual garage and its sensors. However, as illustrated in Figure 3, the rest of **the software-to-be should be developed so that it is easy to unplug the current provisional interface and plug actual sensors, and deploy the system in a real garage**.

The simulator does *not* consider “legacy” customers who use the garage in the old-fashioned way, independently of the computerized reservation system. Recall that separate decks are reserved for legacy customers and we assume that they will not interfere with computer-based customers.

The *subsystems* or *modules* of the software-to-be are illustrated in Figure 4 and described next. (Module numbers are labeled in Figure 4.)

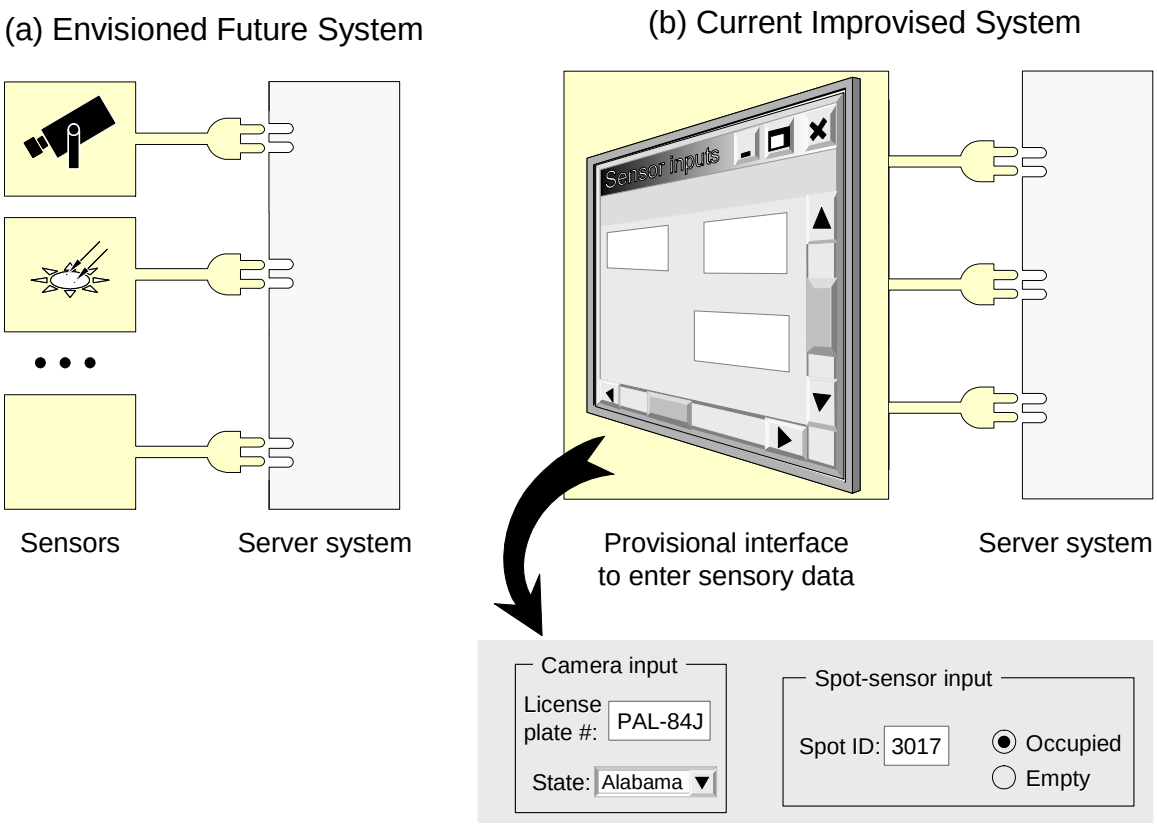


Figure 3: Improvised solution for the server side of the parking garage system.

Module-1: User Interaction

This module supports (a) registration of new customers, (b) requests for parking-space reservation, and (c) general account management, such as allowing the user to see the list of recent transactions with the parking garage. This module can be a server-side application, such as PHP script, that accepts client connections over the Web and interacts with the relational database to process the client requests.

This module implements the business policy (**P11**) and also enforces the policy (**P9**). To support *overbooking* (policy (**P11**)), customer occupancy categories (defined in Section 1.1) need to be tracked so that the parking operator can more accurately predict occupancy. The operator can obtain the data for equation (1) by reviewing the statistics collected by Module-5. Also, the operator should check local business events, sports events, and other special events.

There are several important issues to consider in the design of this module:

- We already mentioned that the number of data entries required by the user should be kept at a minimum, because the remote client may be run on a smartphone or another small device (vehicle dashboard?). We need to carefully determine what is required to support the given functionality. Should we support only requests for parking-space reservation from small devices, and require that the user uses a regular computer for all other activities, such as registration of new customers and general account management?

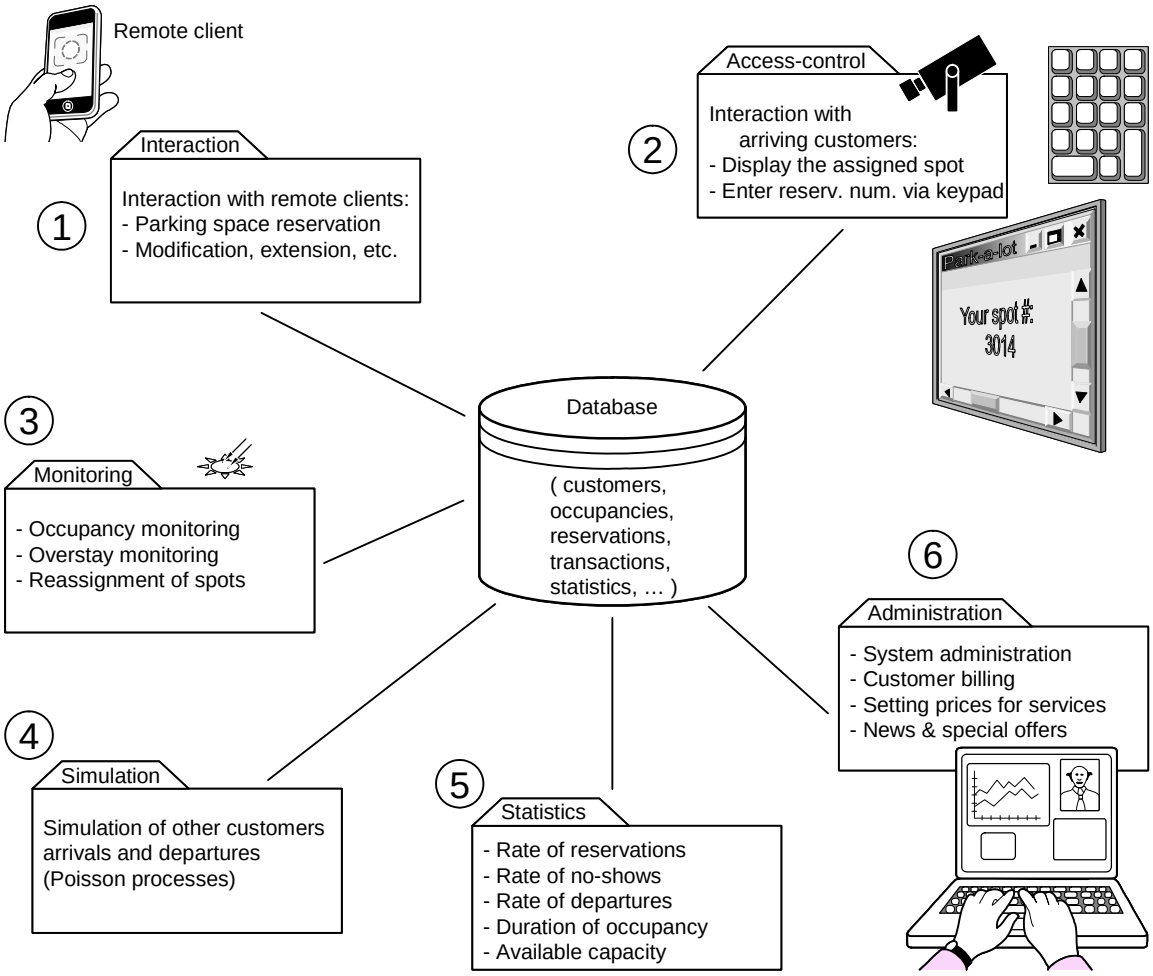


Figure 4: Functional organization of the server side of the parking garage software-to-be.

- To support requests for parking-space reservation, this module will query the relational database and search for the available spots during the interval that the customer specified. It is realistic to expect that the garage capacity will be less than 1,000 spots. However, retrieving each record from the database and comparing it to the specified interval may be too slow for a hurried customer. We need to look for an efficient algorithm for finding available spots. This problem is further discussed in the Appendix of this document.
- Requests may be originated by multiple customers simultaneously, so the server needs to support concurrent access. Web servers have a built-in capability to handle simultaneous requests. If the developers will develop their own server, they will need to implement a thread pool, so if simultaneous requests arrive, a thread from the pool is assigned to handle each request.

Module-2: Garage Access Control

This module controls the access to the garage parking space. The functionality includes processing the inputs from cameras for license-plate recognition, presenting information on

displays, and supporting entrance-console interaction for entering reservation confirmation number (in case the vehicle registration number cannot be recognized).

This module implements the business policies (**P1**), (**P10**), and (**P12**).

Because this is a simulation of the physical process, we need to develop a separate interface to support actual parking behavior. After making a reservation using a remote client, the customer will use this new interface (different from the reservation interface) to simulate the parking activity—to pretend entering the garage and parking his or her car at the reserved spot.

An example scenario for entering the garage could be as follows:

1. System shows two choices: “Arrive” and “Depart.”
2. User selects “Arrive.”
3. System asks the user to type in their vehicle registration number.
4. User types in their vehicle registration number. [Assume that the system does not recognize the entered number.]
5. System informs the user that the number is not recognized and offers the user to try with their reservation confirmation number.
6. User types in their reservation confirmation number.
7. System recognizes the reservation confirmation number as correct and informs the user about the identifier of their parking spot.
8. User confirms that his or her vehicle is parked correctly. System changes the spot status to “occupied” and starts billing the customer (see details below of how exactly this is done).

Similar scenarios can be imagined to simulate the departure activities.

In the Step 8 above, this module (Module-2) should not directly change the spot status in the database. Instead, this module should invoke an operation in Module-3 (described next) to record the occupancy, because Module-3 implements additional checks.

Module-3: Monitoring of Occupancy and Space Reassignment

Each parking spot has a record of its current state in the database: “available,” “reserved,” or “occupied.” The state transition diagram for individual spot occupancy status is shown in Figure 5. This module should ensure that only the allowed state transitions will occur. However, the system should continuously track the customer from the entry point to his or her assigned spot in order to know whether the customer parked to the correct or wrong spot. Also see the discussion in assumption (**A6**) and the Appendix.

Our current system will not be connected to a real garage and cannot track vehicles in real-time. However, the interface in Figure 3(b)) allows the user to simulate what will happen if the customer parks on a wrong spot. For example, assume that a customer arrives and is assigned the spot number 3014 and the spot ID is shown on the elevator display in Figure 1. However, the customer notices another spot, say number 3017, and parks there. The interface in Figure 3(b) allows us to simulate this scenario and test that the rest of the software-to-be works correctly even

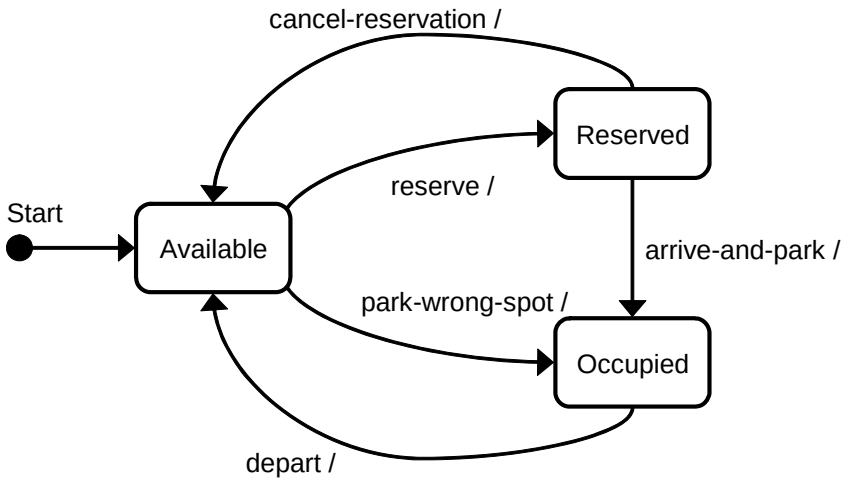


Figure 5: State diagram for an individual parking spot.

if a customer parks on a wrong spot. See more discussion about the actions that need to be taken for such scenarios in the Appendix.

In a real-world implementation, this module would receive the occupancy change event notification from the occupancy sensor installed in each parking spot. Because we are implementing a simulation, the occupancy event is detected by Module-2, as described above in Step 8 of the garage-entering activity, when the user confirms that his or her vehicle is parked correctly. For this reason, Module-3 should have a function for changing the occupancy state that will be called by Module-2 when customers enter or depart the parking garage.

This module periodically queries the database for reservations and determines if some customers did not arrive as scheduled by their reservation. Reservations are held for a limited "grace period," as per the business policy (**P3**). The reservation is released after the grace period expires by changing the database status of the reserved spot to "Available," unless the customer has paid an additional fee to hold the reservation beyond the regular grace period. The system also applies the business policies (**P4**) and (**P5**) for late-arriving or no-show customers. Each policy is applied by recording the event, such as "arrived late," or "no-show," in the record of customer's transactions.

The module periodically queries the database for current occupancies and determines if some customers failed to depart the garage as scheduled. The system applies the business policies (**P6**), (**P7**), and (**P8**) accordingly, and records the customer's transactions with the system, such as extension of the reservation, overstay, etc.

This module uses the business policy (**P14**) when deciding how to attribute the transaction to a particular customer.

Module-4: Simulation of Arrivals and Departures

Having a single customer at a time to park in the garage would not exhibit interesting behaviors. On the other hand, it would be difficult to allow many users to simultaneously simulate the parking activity. We would need to develop a server that can handle many simultaneous

interactions and recruit many people. Instead, in addition to the *real* customers who will interact with the system and pretend to park using Module-2 in Figure 4, we will simulate many *artificial* customers by using two *Poisson processes*. One process will simulate artificial customer arrivals: customers will arrive one at a time and their arrivals will be modeled as a Poisson process. The other process will simulate how artificial customers depart the garage, also one at a time.

For a Poisson process with average arrival rate λ, the probability of seeing n arrivals in the time interval Δt equals:

$$\Pr(n) = \frac{e^{-\lambda \cdot \Delta t}(\lambda \cdot \Delta t)^n}{n!} \text{ and } E\{n\} = \lambda \cdot \Delta t \tag{2}$$

Inter-arrival time t (time between successive arrivals) in a Poisson process follows *exponential distribution* with parameter λ:

$$\Pr(t) = \begin{cases} \lambda \cdot e^{-\lambda t} & t \geq 0 \\ 0 & t < 0 \end{cases} \text{ and } E\{t\} = \frac{1}{\lambda} \tag{3}$$

To generate exponentially distributed random numbers, generate a *uniformly* distributed random number u on the unit interval [0, 1]. Then apply the following function to obtain an *exponentially* distributed random number rx:

$$rx(u) = \frac{-\ln(u)}{\lambda} \tag{4}$$

where $ln(\cdot)$ is the natural logarithm (using basis e). Let us assume that the unit interval is one hour, so the parameter λ specifies the average number of arrivals per hour.

This module runs two threads in infinite loops as follows. The first thread simulates *arrivals*:

1. Query the database if there are currently any "Available" spots. If yes, select one randomly and change its state to "Occupied." If there are no available parking spaces, record this attempt as an "Overbooked" event in the statistics table, maintained by Module-5 in Figure 4.
2. Generate an exponentially-distributed random number rx using equation (4). Convert the number to the time scale, e.g., if $rx = 0.3$, then $t(rx) = 0.3 \times 60$ minutes $= 18$ minutes. This number represents the time of the next arrival.
3. Suspend this thread to sleep for $t(rx)$ time. When the thread wakes up, go to Step 1.

A similar thread runs the departures process. The departures thread selects a random occupant/customer from the database for departure. We must be careful to allow dislodging of only artificially generated customers, and *not* the real customers who checked in using Module-2 in Figure 4. For this purpose, each database record of spot occupancy should also have a tag for a "real" or "artificial" occupant. Artificial customers are those generated by Poisson process simulation and only artificial customers can be selected for a random departure.

A more realistic simulation would also simulate reservations and another Poisson process. However, two processes for arrival and departures are sufficient as a starting point.

The developers should experiment using different values for the average arrival rate λ_a and the average departure rate λ_d. Report any interesting behaviors or phenomena that you observe.


Visit http://en.wikipedia.org/wiki/Exponential_distribution for information about exponential probability distribution

An important issue is how to generate customer identity for the artificial customers. This is important for correct functioning of other modules of the system in Figure 4. One option is to generate a pool of artificial customers with randomly generated names, vehicle registration numbers, etc. We must ensure that the newly generated values are not already in use by another (artificial) customer and that the values are such that can never interfere with the values for real customers. Notice that the values for real customers cannot be known in advance, because the system should allow registration of new customers at any time.

Module-5: Statistical Data Collection

This module periodically queries the database and collects the statistics about garage occupancy over different periods (day, week, month, etc.), number of overbooked reservations, number of customers who were turned away because of overbooking, number of customers who do not show up, depart earlier than booked, or overstay, average duration of overstays, etc.

In addition to the statistics collected by this module, the operator should check local business events, sports events, and other special events.

Module-6: System Administration

The parking operator should be able to configure the simulator with parameters such as:

- Total capacity of the parking space as well as configuration of the garage (number of decks, number of spots per deck, etc.)
- Rates for parking usage as reserved
- Special fees for overstays
- Average arrival rate λ_a and average departure rate λ_d, that are used in Poisson processes in Module-4 in Figure 4 (see the description above)

Implement a system for billing reserved occupancy time, extensions, overstays, and walk-ins. Periodically (e.g., once a month), the system examines the list of transactions for all customers and generates the monthly statement. Also, the operator may explicitly request the list of all transactions for a given customer, in case of termination of the membership or similar scenarios.

The parking operator should be able to view various statistical charts about garage occupancy over different periods (day, week, month, etc.), number of overbooked reservations, number of customers who were turned away because of overbooking, number of customers who do not show up, depart earlier than booked, or overstay, average duration of overstays, etc.

There is an **important design issue** to consider, that spans several modules in Figure 4. When a customer makes reservation, we assume that Module-1 will select an available spot from the database and change its status to “Reserved” during the reservation period requested by the customer. Notice that the same spot may be in use by another customer before this customer’s reserved period. When the customer arrives to park, Module-2 informs the customer about his or her assigned spot. Module-3 monitors spot occupancies and changes “Reserved” to “Occupied” when the customer parks. If at the time of the new customer arrival the spot state is still “Occupied,” meaning that the previous customer did *not* depart as scheduled, one module needs to automatically reassign another parking spot for the new customer (if any is available). The

design issue to consider is: should this reassignment be done by Module-2 or Module-3. The advantage of Module-2 is that it is already interacting with the customer and will display the assigned spot. The advantage of Module-3 is that it must contain the logic for spot reassignment in any case, because it performs monitoring of no-shows and releasing their reservations, as well as the detection of overstays. Module-3 could simply perform the reassignment when it detects an overstay. The developer team should carefully consider this issue before proposing a solution.

2.2 Domain Fieldwork

Visit local parking garage(s) and interview personnel to understand the current practice, what problems they are facing, and identify the opportunities where introducing automation can help increase customer satisfaction and operator's profits. Discuss whether the business policies stated above are realistic in terms of their business and economic merits. Discuss the policies for compensating the customers who make advance reservation but find a full garage at the time of their arrival, and policies for charging the customers who overstay their reservation or arrive late. You may also find that more policies need to be formulated, in addition to those listed in Section 1.1. For example, how to handle the case where a customer with a valid reservation enters the garage but then immediately decides to leave (e.g., because of receiving an urgent call)? Should the customer be charged any fee, because the system may have rejected other reservation requests and this customer's spot will be left unused for some time?

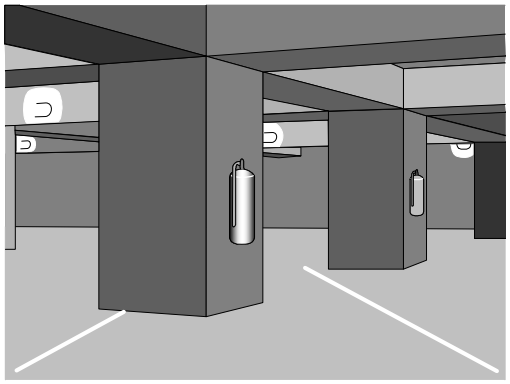
The *tradeoff* is usually between the desire to *maximize the profit* and the *degree of the customer inconvenience*. A high degree of customer inconvenience may negatively affect profits. For example, the business policy (**P8**) states that the customer will be charged at a higher rate for staying longer than booked in their reservation. This may be reasonable from the garage-operator's viewpoint, to encourage customers to keep their promises and to cause less inconvenience for other customers. However, the current operator practice (system-as-is, without any automation) allows the customers to park as long as they wish at the same rate, without advance specifying their departure time. The customers may perceive the new policy (**P8**) as a significant departure from the existing practices and complain. This is particularly problematic in scenarios when the garage space is booked to the full capacity. If the operator tolerates overstays, this policy will inconvenience other customers who will find the parking garage unavailable although they previously successfully made a reservation. Should the operator charge higher rates for overstays only if the garage space is booked to the full capacity? A key question for the operator is, which kind of customer inconveniencing will have more negative effect on the profits?

Customer-friendly policies should be preferred if they have minimal or no impact on the profit. For example, allowing a grace period for late arrivals or departures may be a good policy if no other customer will be turned away for the lack of parking space. Simulations using Module-4 in Section 2 can quantify the impact on the profit (profit loss) by quantifying how many customers (on average) will be turned away, multiplied by the lost income that could have been collected from these lost customers. Perform an economic study to find optimal scheduling algorithms for overbooking, and forecasting of vacancies.

Another unresolved issue is to specify how far ahead of time or into the future to accept reservations. For example, should a customer be able make reservation one year ahead of actual

use of the parking garage? Should the garage operator set the time limit and accept reservations only for, say, the next ten days starting from the present moment?

Additionally, how long before the booked interval is the customer allowed to modify his or her reservation. For example, the customer has booked a spot from 9 to 11 o'clock. However, at 8:50 they find out that they have another urgent engagement from 9 to 10 o'clock, and decide to modify their existing parking reservation and specify a new interval from 10 to 12 o'clock. Should this be allowed? Should the operator impose a limit and allow no modifications, say, one half hour before the start of the booked interval? For example, hotels usually do not allow reservation modifications within the last 24 hours.



The issue of modifying the existing reservations is more complex than implied by the previous paragraph. If the customer is currently using the parking garage and for some reason cannot depart as scheduled by their reservation, the customer may wish to *extend* their reservation. The semantic difference between "modifying" and "extending" needs to be clarified. The time constraints on the ability to extend the existing occupancy need to be specified. Also, how to deal with cases when the garage space is booked to the full capacity and extensions will cause overbooking and inconvenience for other customers?

Our assumption (**A6**) states that we assume that the customer will always park at their assigned spot, and will never park at an arbitrary vacant spot. Of course, in reality this may not be true. If the information in the database does not reflect the reality, this may create major problems. For example, if a customer parks on a wrong spot, the system may direct future customers to an already-occupied spot (or accept reservations for this spot), and meanwhile the system will consider another spot as occupied, while it is actually available. How will the sensor check that the right user is parking on a particular spot? A camera-based license-plate recognition system may be too expensive to install on every spot. Perhaps the spot should have a "reserved" signal turned on to deter other customers from trying to occupy a reserved spot? Or, perhaps the system may perform some form of "virtual tracking" of vehicles from the vehicle elevator to the assigned parking spot. We know that customers enter the garage one at a time, because the elevator can carry only a single vehicle at a time. The system can use the time elapsed from the moment when the vehicle leaves the elevator until one spot-occupancy sensor reports spot occupancy and some average driving speed, to infer whether the parked spot is the one associated with this customer. If the system suspects that the customer parked on a wrong spot, it needs to notify the customer and request relocation. This requires a display or an audio announcement system, which may be too expensive to install. Another option is to do nothing and charge the customer a penalty fee in a monthly statement. However, the customer may dispute such charges and they may be difficult to prove long time after the fact.

If the garage will have sensors installed that will be able to detect when a customer parks on a wrong spot, then the system should rearrange the table of parking reservations. See more discussion in the Appendix, Figure 7.

3. Extensions

The above project description should be considered only as a reference. The student team should customize it to accommodate for their knowledge of software development and ambition.

A more ambitious team could consider a scenario where the same operator operates several garages at different locations in the city, or partners with other garage operators who will be using the same system-to-be. The customer may request the nearest available garage, or the one closest to his or her destination point.

The server may support priority-based reservations, e.g., based on organizational affiliation, monthly membership fees, "guaranteed reservations" as defined earlier, etc.

The database-centered design in Figure 4 (Section 2) represents one possible software architecture for the parking system (known as "Repository Architectural Style"). An extension is to consider the merits of other architectural designs, which are not necessarily centered around a relational database. For example, modules may be communicating directly (known as "Peer-to-Peer Architectural Style"), instead of indirectly via the database.

Consider different *pricing schemes*, so prices during the peak-demand periods are significantly higher than during the trough periods. Another option is to support auctions and allow customers to bid for slots. The duration of the auction must be relatively short to make sense.

Design a touch-screen-based interface for an in-dashboard screen that the user can use with minimum number of required inputs. Alternatively, design a reservation system based on **voice recognition**.

The current version of Module-1 in Figure 4 will search for an *exact match* for the customer's reservation period. A more sophisticated the scheduling system would find a *nearest match* if it cannot find and exact match. For example, the customer may request a reservation from 9:00 – 12:00 o'clock, but there may be no available spot during this period, and the nearest match could be between 9:30 – 12:00. Design a nearest-matching algorithm and consider the issues of user interface design to make reservation for nearest-match scenarios. Another option is for the system to *reshuffle the existing reservations* to explore if it is possible to find an available period for the current customer. For example, if spot X is available during period T_1 and spot Y is available during period T_2, the system may be able to reassign the reservations for one of the spots and create an available period of $T_1 + T_2$ long. This problem is further considered in the Appendix (see Figure 7).

An "intelligent" version of the parking system may offer a forecast of how long the customer might need to wait until a spot will become vacant (for a walk-in customer), or what is the likelihood that a spot will become available within the next, say, half-hour, for a desired reservation period. Check the project website (given below) for links to the existing literature that describes such forecasting systems.

Consider the assumptions listed in Section 1.2. Describe the **risks** to customer safety, operator profits, etc., if some of the assumptions are not met in reality. Propose tactics for risk reduction.

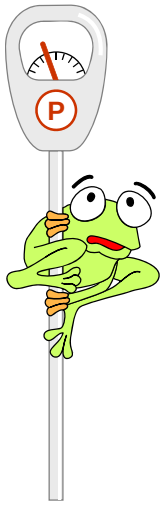
We may consider imposing a "buffer time" between the successive reservations to account for overstays. If departures are distributed according to a Poisson distribution, what is the probability


Visit http://en.wikipedia.org/wiki/Software_architecture for examples of architectural styles and patterns

that a spot will be vacated if we introduce a “buffer time” between the successive reservations? How does this intervention affect the profit?

Develop a system that automatically searches the Web for local business events, sports events, and other special events. This information can be correlated with the parking occupancy statistics collected by Module-5 in Figure 4 and used by the parking operator when making overbooking decisions.

The current design of the garage building has some advantages and disadvantages. The advantage of using the vehicle elevator is that it streamlines the access to the upper decks for the reserved customers. It simplifies the design of the display board, the license-plate recognition system, and the entrance console. However, it may also be a **traffic bottleneck** and expensive to maintain. An alternative is to build instead an ascending driveway. The developer team may need to consider what issues will be introduced if one or more driveways are built (with barriers to control the access). Similarly, what issues will be introduced if multiple vehicle elevators are built? For example, the module for assigning parking spots to the newly arriving customers must be careful not to assign the same spot to two different customers that are using different vehicle elevators or different entry driveways.



5. Appendix: Efficient Finding of Free Spots

This appendix continues the discussion about the problem of efficient finding of free spots for the period specified by the customer, mentioned in Section 2 for Module-1. This problem is similar to the "Free space bitmap" problem (http://en.wikipedia.org/wiki/Free_space_bitmap). The way to approach this problem would be to keep the complete information for all reservations in the database and create a *bitmap* (binary table) of all parking spots where each cell indicates whether the corresponding spot is reserved or available. This bitmap is equivalent to the "Free space bitmap" and will allow quick finding of a free parking spot when a customer requests reservation.

Notice that our assumption (**A6**) states that we assume that the customer will always park at their assigned spot, and will never park at an arbitrary vacant spot. If a customer parks on a wrong spot and the system cannot detect this event (i.e., there are no physical sensors installed in the garage that allow the system to detect if customer parked on a wrong spot), then the system will be working with a reservations table that does not reflect the reality.

Given that reservations are restricted to be in increments of 15 (or 30) minutes and must be aligned to one of the few points through the hour. With increments of 30 minutes, the reservations must start either at the top of an hour or halfway through the hour; with 15-min increments, there will be three 15-minute points within the hour. Then construct a bitmap for each day that has N rows for time and M columns for parking spot identifiers. Assuming 15-minute increments, there will be $N = 4 \times 24 = 96$ rows (for 15 min increments during a 24-hour period). A garage with 2000 spots can be considered as large, so $M = 2000$.

Then the bitmap has $N \times M = 96 \times 2000 = 192{,}000$ cells. Because we need only 1 bit per cell (reserved/available), the bitmap size for each day is $192{,}000 \div 8 = 24{,}000$ bytes. This is manageable for contemporary desktop computers or servers. A single copy of the whole bitmap can be held in the working memory and updated by different threads that process simultaneous reservations from multiple customers. If the bitmap is considered too large, it can be split into several smaller ones by partitioning the garage by floors, and only one small bitmap will be maintained in the working memory at any time.

There are two key issues:

1. How to keep the bitmap(s) consistent with the database information?

2. Where to store the bitmap?

For issue #1, the tread that processes the reservation should both update the bitmap and the database record. In addition, the system could run a "demon" process during idle periods to check that the bitmap is synchronized with the database.

For issue #2, for a large garage that allows reservations over a period of several days, it may not be feasible to keep this bitmap in the working memory. Again, a solution may be to maintain different bitmaps for different days as well as different floors of the garage. In addition, perhaps some kind of smart caching could be applied?

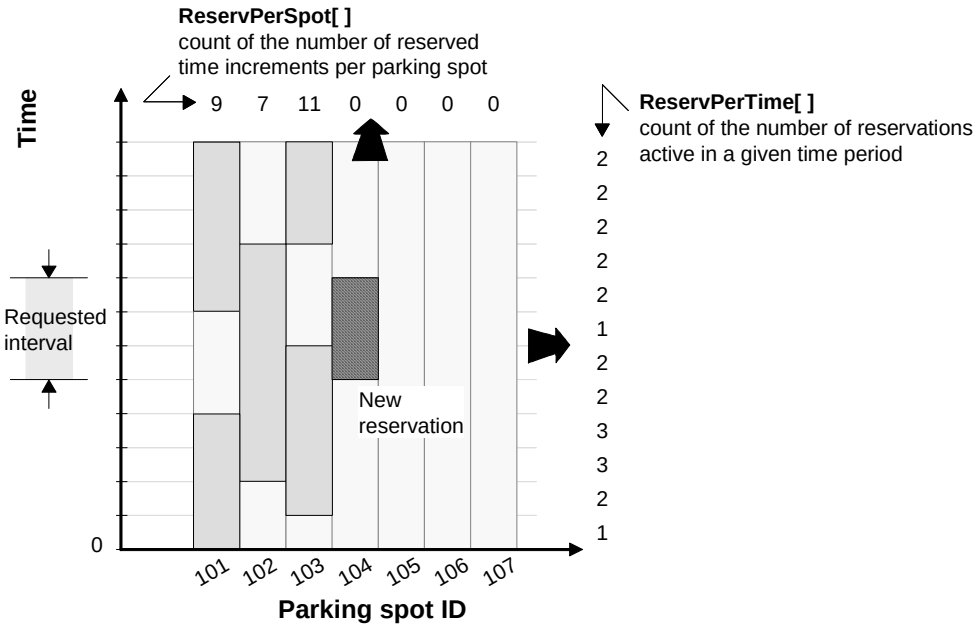


Figure 6: A simple compression of the parking-reservations bitmap to two vectors.

If the bitmap is too large for the working memory (in case of a large garage), we may try compressing it. For example, we could maintain only two vectors for the sums of rows and columns (Figure 6). The vector ReservPerTime[] contains the total number of reservations for each time increment (15-minute or 30-minute). This vector would have *N* elements, e.g., 96 for 15-minute periods in the day. Each element would contain one value, the count of the number of reservations active in that time period. The other vector in Figure 6, ReservPerSpot[], contains the total number of reserved time increments per given parking spot.

When a new reservation is requested, the system first checks if any count in ReservPerTime[] during the requested interval is equal to the maximum number of parking spaces in the garage. If yes, the system declines the reservation request. If no, the system next checks ReservPerSpot[] and considers only the spots for which the total count of reserved time increments is less than or equal to the maximum number of time increments minus the requested number of increments. For example, the system uses 15-minute increments and the customer requests the interval from 9:00 – 11:30 AM. Then the total number of 15-min increments for 24 hours is 96, and the number of requested increments is 10 (for two and half hours). The system would then consider as candidate spots only those for which the total count of reserved time increments is less than or equal 96 – 10 = 86. As the last step, the database records for all candidate spots need to be examined in detail to complete the reservation request. This approach may be useful to reduce the number of candidate spots that need to be examined in detail. However, there may be many candidate spots that are not are suitable. For the example in Figure 6, spots 101, 102, and 103, would all be declared as candidate spots. However, a detailed examination would find that none of them is available during the requested period. The first available spot in this example is 104. The problem is that this simple approach with two vectors performs loses too much information in the process of compressing the reservations bitmap.

Another approach is to use a lossless compression algorithm, such as LZW (http://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch) to make the bitmap small

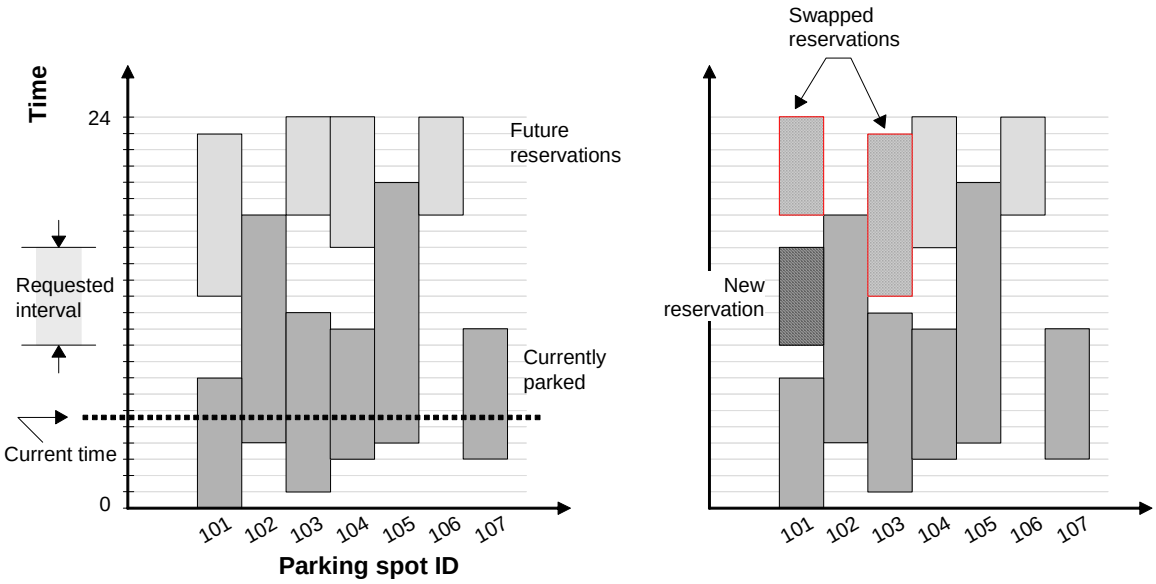


Figure 7: Illustration for the problem of creating a free parking spot by rearranging the existing reservations.

enough so it could be kept in the working memory. Then the key questions is, can the compressed bitmap be used *directly* (as is, without decompression) for finding free spots? If not, it will need to be decompressed and compressed frequently, so this approach would not be feasible.

We also need to consider the problem of inefficient use of parking spaces. This problem may arise particularly if some of the existing reservations are canceled and random gaps are left in the reservations bitmap. Another example scenario involves a customer who parks at an arbitrary vacant spot, rather than at their assigned spot. If the garage has built-in sensors to detect this event, then the system needs to rearrange the reservations table to reflect the reality. For example, current customer *X* has a reservation from 9 – 11 o'clock and is assigned spot *A*, but parks at a wrong spot *B*. The spot *B* is currently (at 9 o'clock) available but is reserved from 10 o'clock for another customer *Y*. Then the system must mark the spot *B* ac "occupied" from 9 – 11 o'clock move the reserved customer *Y* to another spot. It could be spot *A*, if it is available for the duration of customer-*Y* reservation, or any other available spot.

Figure 7 illustrates how a free parking spot could be found by rearranging the existing reservations. We assume that the customer is told his or her parking spot identifier only when they arrive to the garage. Therefore, any rearrangements of existing reservations (illustrated in Figure 7) are transparent to customers. This approach has two advantages: (1) for customer, it is possible to find a free spot that otherwise could not be found; (2) for the operator, the parking spaces are used more efficiently.

The problem of rearranging the reservations for efficient use resembles *disk defragmentation* (http://en.wikipedia.org/wiki/Disk_defragmenter), but disk defragmentation algorithms probably cannot be directly applied without modifications. The "reservation table defragmentation" could be run as a demon process during idle periods or periods of low activity.