

An Analysis Pattern for the Request and Allocation of Limited Resources

Fei Dai and Eduardo B. Fernandez
Department of Computer Science and Engineering
Florida Atlantic University
777 Glades Road, Boca Raton, FL 33432
E-mail: {fdai, ed}@cse.fau.edu

1. Intent

How to allocate a limited number of nonreusable resources between several requests, both fairly and efficiently?

2. Context

In many situations the number of available resources is smaller than the requests. For example, tickets for a popular sports game are sold out far before the date of play; a well-paid position in a company may have many applicants; in China, stocks are always sold out when first offered because of their lower price. These are nonreusable resources in that, from the system point of view, there is no provision for the return of the resource and its future allocation. This is in contrast with reusable resources such as vehicles, hotel rooms, and airplane seats, which are returned after use and scheduled to be used again in the future.

Allocation of limited nonreusable resources is complicated because of its many possible combinations of operations. Usually, pre-requesting (sending the request and waiting for later allocation) is required, but in some rare cases, a client can acquire resources directly without reservation or wait, e.g., when the resource validity is near expiration. For example, ticket requests for important soccer games are fulfilled after the request is sent; however, one can go to a ticket counter when the game is ready to start and get a ticket if any is left (direct acquisition). Usually the allocation is deferred until a specific date or event; for example people in the standby line of a flight get seats shortly before departure, when it is clear that no more passengers with reservations will arrive. In other cases, allocation is done as soon as a request is issued (as in the soccer example above). There are also different ways to distribute the resources: they can be

selected in a first-come-first-served manner, by comparing their priorities, by drawing a lottery, by negotiation, or by other policies.

We assume also that the resources are indistinguishable from each other, they just belong to a specific type. For example, a request specifies a number of tickets of a given price, an amount of stock of a given type.

3. Problem

The analysis pattern must satisfy the following basic use cases:

- *Request resource*: A client requests a given number of resources of a certain type. The resources are indistinguishable from each other. The request is put into a queue waiting for resource allocation. The client can be a person, an institution, or a computer entity (a process).
- *Allocate resource*: After a specific time, or whenever there are enough resources available (not yet allocated), a request is retrieved from the waiting queue and resources are allocated to it.
- *Acquire resource*: After its request is granted, the client acquires the allocated resources immediately or after a specific time. Clients may be satisfied with partial allocations. For example, somebody who asked for ten tickets may settle for five.
- *Cancel request*: A client cancels its request and releases any allocated resources not yet acquired.

The following forces will affect a possible solution:

- The handling of requests should be fair (they are all handled in the same way), although the specific allocation strategies may not be fair, e.g. priority allocation.
- The solution must embody just fundamental semantics, it must represent a minimal application, consisting of a few use cases. This is a requirement to make this pattern a Semantic analysis pattern [Fer00]. This type of pattern is useful to build conceptual models.
- Effective use of entities. No redundant or irrelevant entities.

- Many applications require paper documents. Standard documents should be explicit in the model. For example, the delivery of a set of reserved tickets may come with a delivery document indicating price, date, etc. This makes the model more intuitive and facilitates the generation of these documents.

4. Solution

We use an abstract queue¹ as the core of the analysis model. Requests are put into the queue according to the allocation strategy. For example, a First-Come-First Served strategy would put a new request at the tail of the queue, a priority strategy places new requests in places according to their priority.

Every request must be put into the queue. Resources can only be allocated to the request in the front of the queue. A direct allocation is viewed as a special case where the request is immediately in the front of the queue.

Resources become available at some specific points of time and when an allocated request is cancelled. Clients can choose whether a partial allocation for their requests is acceptable.

4.1 Class Diagram

Figure 1 shows the necessary classes. Clients make requests for resources of some type. We have a **Queue** for each type of resource. Requests are enqueued in this queue. Depending on their availability, resources are allocated, the allocation is described by the **AcquisitionRecord**. An **AllocationStrategy** is used to allocate resources. Note that requests ask for types of resources but the actual allocation is for specific resources. An attribute *acceptPartialAssignment* indicates if the client is willing to accept a smaller allocation.

¹ Notice that this is an abstract queue or list, not a data structure. It can be implemented in a variety of ways.

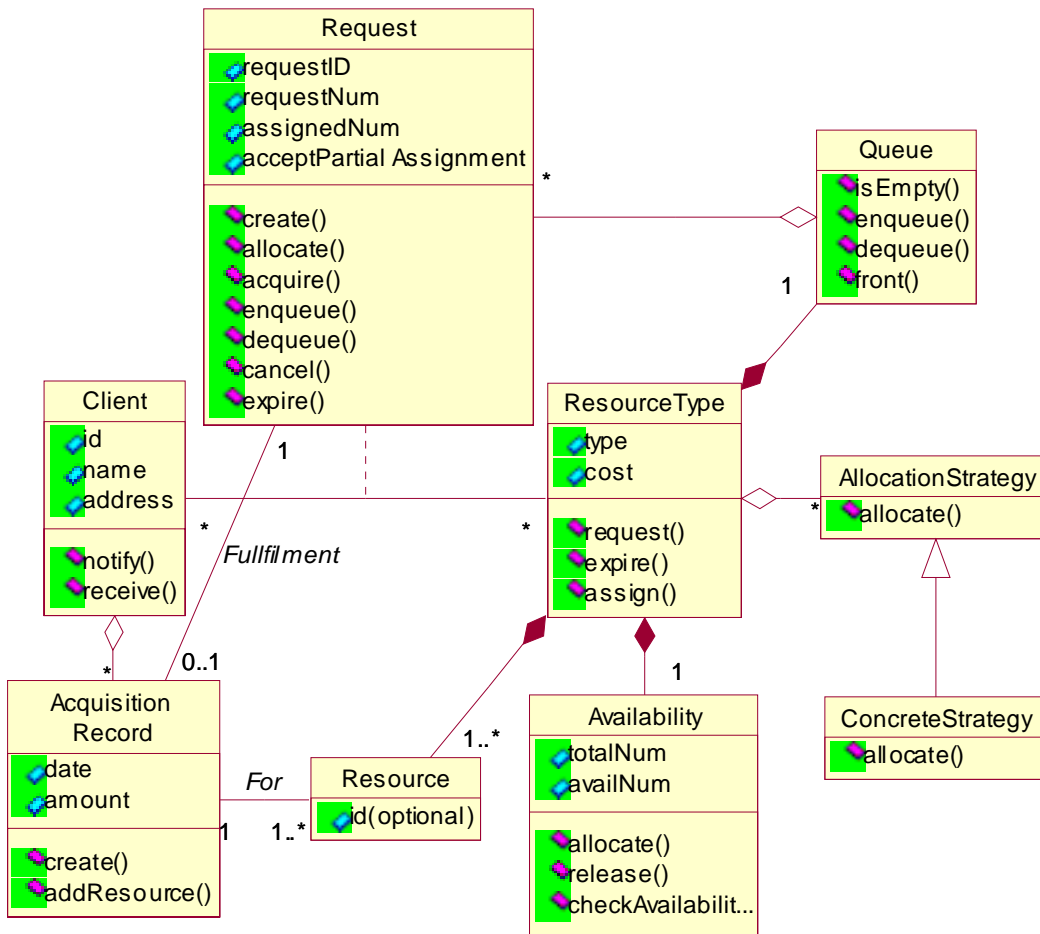


Figure 1. Class diagram for the pattern.

4.2 Dynamics

Figure 2 shows a state diagram for a request object. A request is first created, then enqueued, then resources are allocated to it. The request can be cancelled at any time or it may expire after a timeout.

Figure 3 shows a sequence diagram for resource request, allocation, and acquisition. First, a request object is created and enqueued. The front of the queue is checked and the request at the front is allocated resources and notified. The client acquires the resource and an AcquisitionRecord is created.

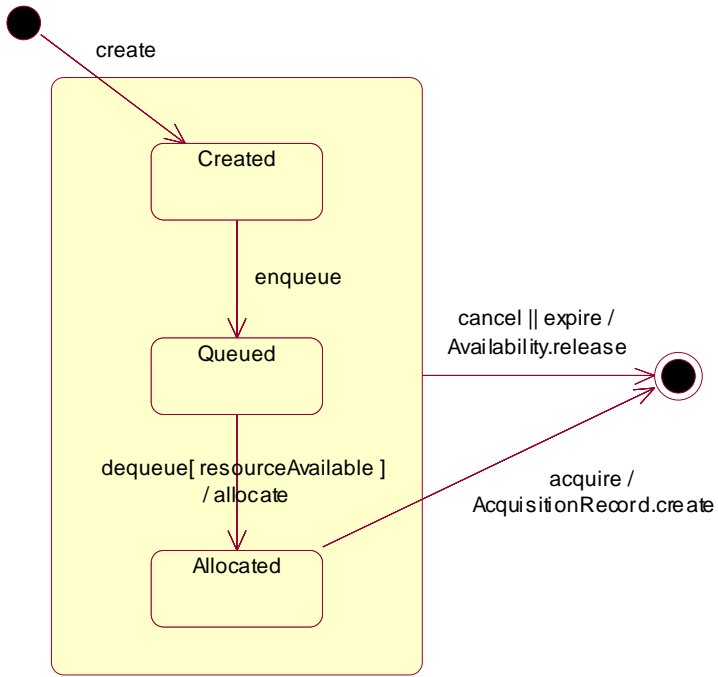


Figure 2. State diagram for a request object.

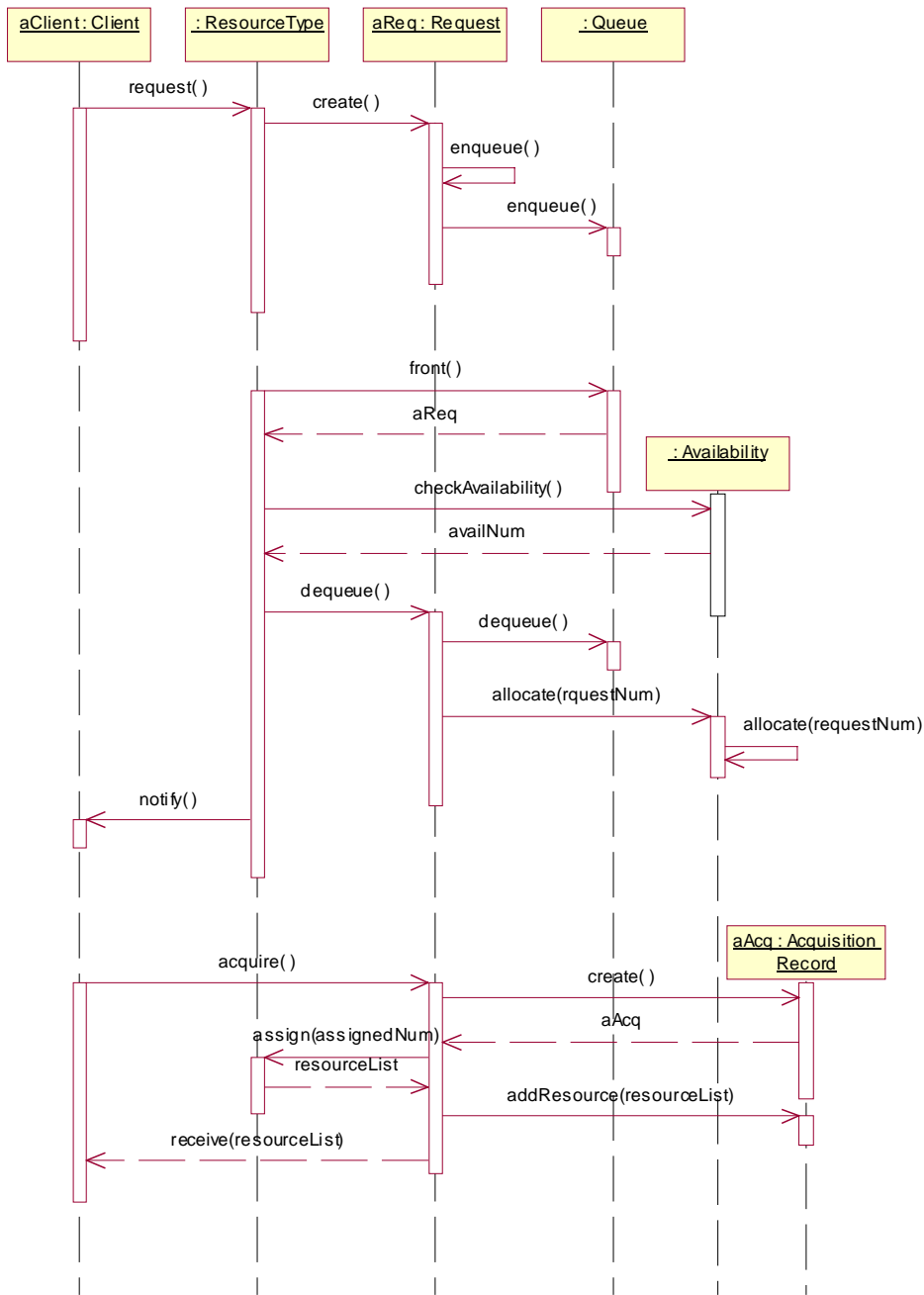


Figure 3. Sequence diagram for requesting, allocating, and acquiring a resource.

5. Example resolved: Stock issuing in the Chinese stock market

In China, when a company issues new stocks to the market, they are sold directly to all participants of the market, including both big investors (foundations, investment banks, etc.) and numerous small investors (individuals). Because the issuing price of a stock is usually lower than the market price, the purchase requests always exceed by far the number of available stocks. In order to provide a fair opportunity to each investor, the administration adopts a lottery system to distribute purchase options. A typical stock issuing cycle has the following stages:

- The company advertises the code, name, and price of the newly issued stock. During the following pre-request period (normally 1-3 days), each investor can place her request containing the request number. The requesting investor should have enough cash in her account to buy the requested number, and the corresponding amount will be frozen until the end of the second stage. The interest revenue produced in the freeze period is to the benefit of the company issuing the stock.
- Each request is given a virtual lottery ticket with continuous sequence numbers. The number of tickets is proportional to the request number. Then, several special ending digits are randomly generated to determine the winning tickets. The frequency of the winning tickets is based on the total number of requests and the number of newly-issued stocks. The virtual winning tickets, or purchase options, are sent to the corresponding investors. This stage is usually done in one week.
- In the third stage (usually one day), the winning investors can use the purchase options to buy stocks. However, they can also give up the option without buying anything. If there are stocks left at the end of the stage, which is not likely, the rest of the stocks are sold to a predetermined investment bank.

Figure 4 shows the class diagram for this use of the pattern, tailored as:

- The Resource class has been removed because stocks of the same company are not distinguishable. We only need to know the number of stocks in each transaction.
- There is only one LotteryStrategy corresponding to all stocks. The distribution policy is unique and fair to all investors.

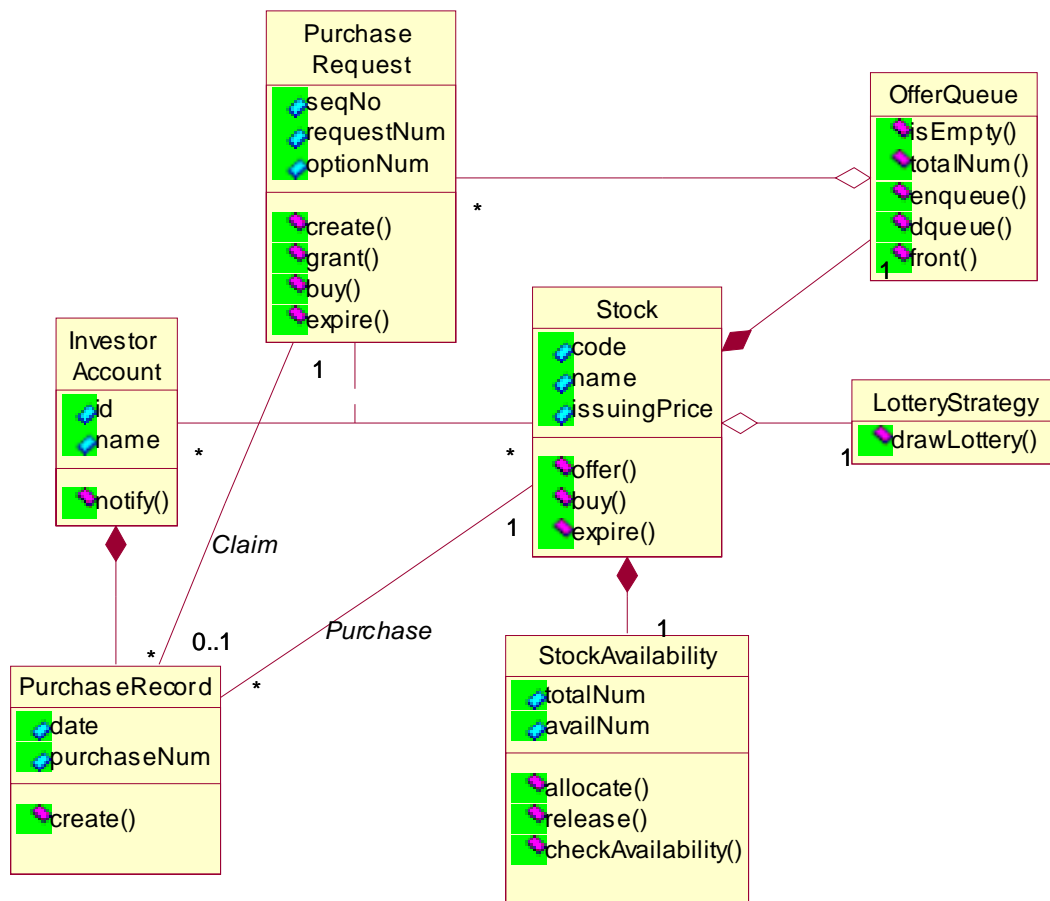


Figure 4. Class diagram for stock purchase application.

6. Known Uses

Some examples of uses of this pattern include:

- Ticket reservation and purchase for popular sports games (e.g., World Cup 2002).
- Job application. Many applicants for a few jobs.
- Public issue of stock shares in China, where investors first apply for buy-options based on a lottery system, and then buy shares within the limit of their buy-options (see Section 5).

- Sale of affordable apartments provided by the city government (e.g., in Shenzhen, China), where applications are put in a priority queue.
- Allocation of manufacturing components to a product.

7. Consequences

The pattern satisfies the forces:

- The queue mechanism provides a fair handling of requests, every request is added to the queue for subsequent resource allocation.
- As shown in the known uses section, this pattern has a wide applicability: tickets, stocks, job positions, etc.
- Every aspect of the requirements is represented and there are no redundant classes.
- The pattern has explicit classes for documents Request and AcquisitionRecord.
- The pattern can be tailored easily to define variations. In the stock example we left out the Resource class because there was no need to identify the purchased stocks.

To make the pattern more general we have left out some details:

- There may be different types of clients that need to be treated differently.
- There may be interactive negotiation during the allocation. This would require additional classes in the design stage to define user interfaces for negotiation.
- Some systems may use several queues for each type of resource. For example, in the airline counters at the airport first class passengers have a separate queue.

8. Related Patterns

This pattern extends some aspects of the Reservation and Use pattern [Fer99]. The main extension is the application queue used to store temporally unsatisfied requests. Also, this pattern does not consider the use of the resources, it ends after the resources have been acquired. Other differences include the fact that the resources are treated as not reusable, and requests may expire after a given duration of time. The Order and Shipment pattern [Fer00b] complements this pattern,

providing the details of resource occupation; that is, the shipment and payment of the allocated resources.

Two other patterns can be found as sub-patterns of this pattern:

- The Strategy pattern [Gam95]. This is used here to model the possibility of using different strategies to allocate resources (classes **AllocationStrategy** and **ConcreteStrategy**).
- The Reservation pattern. This is a sub-pattern of the Reservation and Use pattern [Fer99]. This corresponds to classes **Client**, **Request**, and **ResourceType**. The request is similar to a reservation, waiting to be fulfilled, with the difference that a real reservation includes a confirmation, not present here.

Acknowledgements

We thank our shepherd Pascal Costanza for his detailed suggestions that considerably improved the quality of this paper.

References

- [Fern99] E.B.Fernandez and X.Yuan, "An analysis pattern for reservation and use of entities", *Procs. of Pattern Languages of Programs Conf. (PLoP'99)*, <http://st-www.cs.uiuc.edu/~plop/plop99>
- [Fer00a] E.B. Fernandez and X. Yuan. "Semantic analysis patterns", *Procs. of 19th Int. Conf. on Conceptual Modeling, ER2000*, 183-195. Available from: <http://www.cse.fau.edu/~ed>
- [Fer00b] E. B. Fernandez and X. Yuan. "Analysis patterns for the order and shipment of a product", *Procs. of Pattern Languages of Programs Conf. (PLoP 2000)*. <http://jerry.cs.uiuc.edu/~plop/plop2k>
- [Gam95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns –Elements of reusable object-oriented software*, Addison-Wesley 1995.