3.3 Event list and function list

What is this?

Events to be handled by the product (or a list of product functions).
Events to be handled by human + computer (or a list of user tasks).
Product events are design-level issues.

An *event* is something that requests a system to perform some function. Usually an event arrives with some data telling the system what to do.

An event list shows the types of events that can arrive at the system. Since an event causes the system to perform a function, we can make a similar list of the functions that the system can perform.

The terminology and the details vary depending on whether we talk about events arriving at the domain or at the product. Figure 3.3 shows two lists of events for the hotel system: domain events and product events.

Domain events

Domain events arrive to the domain from the surroundings. Sometimes domain events are called *business events* or *triggers*.

In the hotel case, the domain has to respond to events such as a guest phoning to book a room, a guest arriving to check in, a service note arriving from the waiter who has served breakfast to room 14, etc.

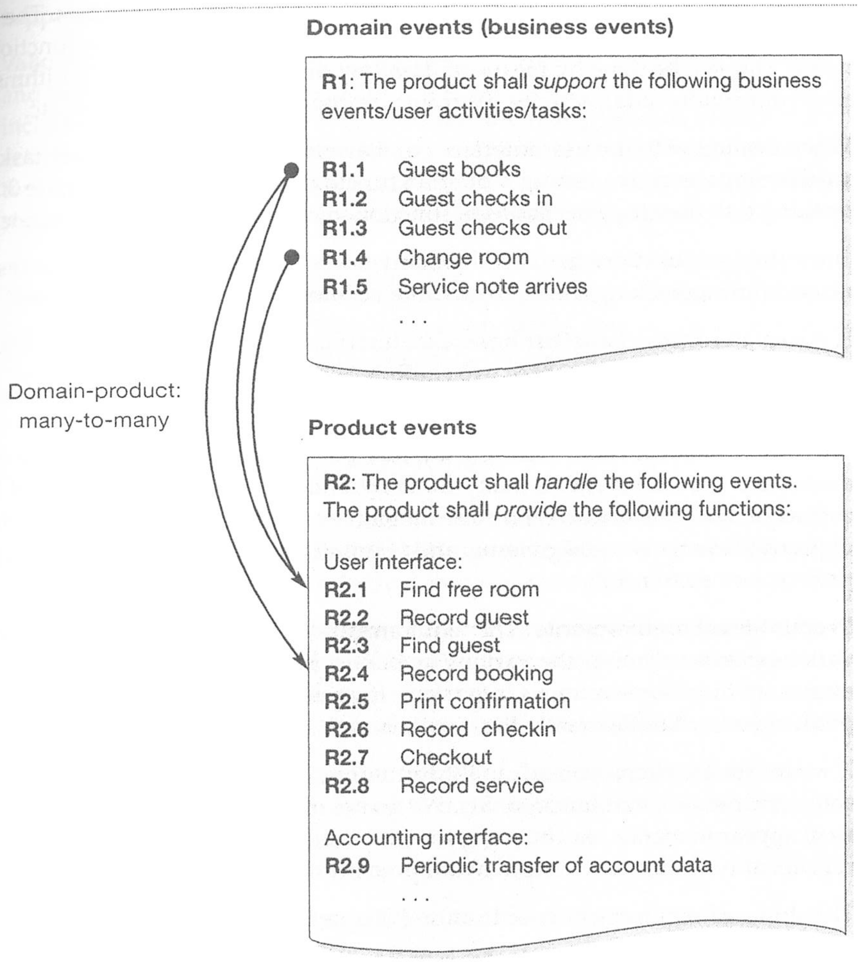
All these events have data attached to them, for example the name and address of the guest for a booking event, or the room number and type of breakfast for a service note event.

Each domain event will cause the receptionist to perform an activity, often called a *task* or a *use case*. Thus we can also look at the list of events as a list of tasks.

The more complex tasks on the list need more explanation than simply the task name. This can be done by means of task descriptions, use cases, dataflow diagrams, activity diagrams, etc. As part of the task, the receptionist will ask the hotel system to perform various functions, i.e. he will send events to the hotel system.

Domain-level requirements. The requirements are that the product shall *support* the various domain events or tasks. It is left to the developer to design the necessary product events and product functions.

Fig 3.3 Event list and function list



Domain events (business events)

R1: The product shall *support* the following business events/user activities/tasks:

R1.1	Guest books
R1.2	Guest checks in
R1.3	Guest checks out
R1.4	Change room
R1.5	Service note arrives
	. . .

Domain-product: many-to-many

Product events

R2: The product shall *handle* the following events. The product shall *provide* the following functions:

User interface:

R2.1	Find free room
R2.2	Record guest
R2.3	Find guest
R2.4	Record booking
R2.5	Print confirmation
R2.6	Record checkin
R2.7	Checkout
R2.8	Record service

Accounting interface:

R2.9	Periodic transfer of account data
	. . .

Product events

Product events arrive at the product from the domain. In the hotel case, the product has to respond to events such as the receptionist asking the system to find a free room, asking the system to record a guest, asking the system to record a service note, etc. The hotel system would also have to respond to a clock signal every night asking it to transfer accounting data to the accounting system.

It is a good idea to organize the event list according to the product interfaces, as outlined on Figure 3.3. The events are often called *messages* to indicate that they also carry data.

Each product event will cause the system to perform an activity, usually called a *function*. Thus we can also look at the list of events as a list of functions. The more complex functions on the list need more explanation than simply the function name. This can be done by means of dataflow diagrams, informal algorithms (mini-specs or pseudo-code), activity diagrams, mathematical specifications, etc.

When dealing with the user interface, we have to break down each user task into smaller steps, each requesting a product function. As illustrated on Figure 3.3, the booking task uses the *Find free room* function, the *Record booking* function, etc.

Some product functions are used by many tasks. Finding a free room, for instance, is used during booking, check in, and if a customer wants to change room.

It is easy to construct another hotel user interface that is based on a different set of product functions. For instance, we could let *Record booking* allocate a room automatically, thus avoiding the need for a *Find free room* function.

There are two conclusions from this discussion: (1) The relation between domain events and product events is many-to-many, and the product events cannot be defined in a simple manner from the domain events. (2) Deciding on a precise list of product events is a design issue, and it should only be done for certain types of projects (see section 1.7).

Product-level requirements. The requirements are that the product shall *handle* the various events or *provide* the various functions. Here the subtle difference between events and functions becomes important. If we list the events as requirements, the product has to handle exactly these events.

If we list the functions instead, analysts interpret it in the way that the product shall somehow provide that functionality. We do not insist that functions named that way shall appear in menus, etc. but only that the user is somehow able to invoke that functionality. In this way we can avoid premature design – at least to some extent.

The shipyard requirement specification (Chapter 11) uses function lists extensively.

User-interface versus technical interface. For human–computer interfaces, the list of functions is reasonable as a product-level requirement, while a detailed list of product events is a design-level requirement. For technical interfaces to other products, the detailed list of product events may be highly important. For example, the telephone system sends events to the hotel system. It is important to list these events in detail. The reason is that in this case, the product events are determined by an existing system, whereas for the user interface, they have to be determined during system design.

Advantages of event and function lists

Event and function lists are primarily checklists for what to develop. The lists should be supplemented with more detailed descriptions of the non-trivial items on the list. Often these detailed descriptions form the major part of the functional requirements.

Verification. Developers can easily check that each event/function on the list is supported or implemented.

Validation. Customers can to some extent validate the domain-level lists of events and tasks. However, they may find it difficult to check whether all events are included. One of the problems is that there are many variants of each event or activity. For instance, is arrival after having booked and without having booked two different events? Section 3.6 explains how to deal with event variants, and section 3.10 explains how to identify the important events.

Analysts can make a consistency check to see that the lists are logically complete. They compare the lists against the data model: Is there an event/function that can Create, Read, Update, and Delete each entity in the data model? If not, some event or function is probably missing. The check can be made on the domain level as well as the product level. Section 9.2.3 explains this.

isadvantages of event and function lists

The event list for the user interface is a design issue. Make it only if you need design-level requirements. If you want a commercial product, the list will be useless since the product has defined its own events already.

The lists may give you a false sense of security because they appear to contain the full truth. However, they are only close to the truth if you have elicited the requirements carefully, for instance identified all the user tasks in the inner domain as well as the outer domain. See section 3.10 for techniques for doing this.

Customers cannot usually validate the list of *product* events and functions. Unfortunately, they are often asked to do so. Customers can better validate the list of domain events and tasks.