3

Functional requirement styles

Data requirements specify the data to be stored in the system. In contrast, functional requirements specify what data is to be used for, how it is recorded, computed, transformed, updated, transmitted, etc. The user interface is in most systems an important part of the functions because many data are recorded, updated and shown through it.

In this chapter we will discuss some of the many *styles* for stating functional requirements. These styles differ in several ways, in, for instance:

- their notation (diagrams, plain text, structured text),
- their ease of validation (customer's check),
- their ease of verification (developer's check),
- whether they specify something about the surroundings or the product functions,
- whether they simply identify the functions or give details of what they do.

Most of the styles in the chapter primarily identify the necessary functions. The last styles from section 3.15 and on use indirect ways of specifying the functions, for instance through standards or required development processes.

In this chapter we don't describe styles that can specify details of what the functions do. We have devoted Chapter 4 to this.

Real requirements specifications use a combination of styles. Most importantly, the different kinds of interfaces need different styles. For instance, the user interface needs a style that the expert user can validate, while the technical interfaces need more precise styles aimed at IT professionals.

In most systems the user interface is the most difficult to deal with, and in this chapter we primarily look at the styles from a user interface viewpoint. Chapter 5 discusses special interfaces, for instance printed reports or technical interfaces. In those sections we show how to combine the various styles to deal with the special interface.

The proper combination of styles also depends on the type of project (in-house, tender, COTS, etc.) and the way the project is carried out (up-front requirements, or detailed analysis leading to design-level requirements; see section 1.7).

3.1 Human/computer – who does what?

Highlights

Domain model: humans and computers united.
Physical model: what each of them do.
Product requirements divide the work.

A pervading issue is how the work is divided between human and computer. The division is not given in advance, but is decided more or less through the requirements.

Figure 3.1 illustrates the problem through a part of the hotel system: how to find a free room for the guest. At the top of the figure we see what human and computer must do together. Based on data about free rooms and the guest's wishes, human and computer must find a free room for the guest. We call this the *domain model* of the functionality.

We could have used various UML notations to illustrate the principle, but have chosen a simple *dataflow diagram*. A bubble means a function that uses some data fed to it through the arrows and computes a result leaving it through other arrows. Data is stored in 'files' shown as double lines, suggesting a pile of paper sheets.

In the lower part of the figure we see a possible division of labor between human and computer. The receptionist (the user) receives the guest's wishes and tells the computer about the dates in question and the kind of rooms needed. The computer scans the list of rooms and shows a list of those available. The receptionist chooses one of them. The computer saves the choice for later use when the guest has been recorded, etc. This is the *physical model* of functionality. In this model we have split the work between human and computer.

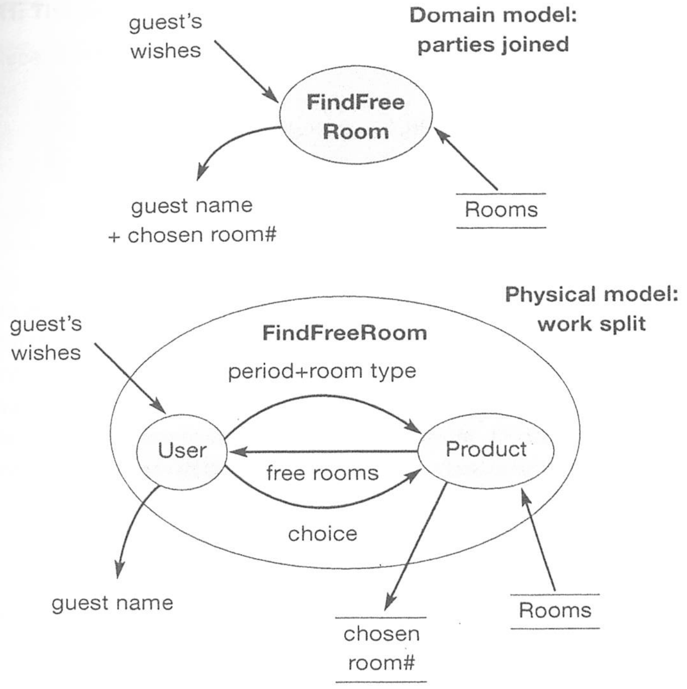
However, we may divide the work in other ways. We could for instance let the computer choose the first room on the list without asking the user. Or we could let the guest choose a room on the Web, seeing a floor map of the hotel. It is still the same domain task, but the division of labor is very different.

No matter how carefully we study the present user tasks, there is no automatic way of identifying the 'correct' functions in the product. Some creativity is necessary, particularly if we want a new system that is markedly better than the present one.

With data requirements it is easier. If we study the information used in the present tasks, we can almost automatically specify the data to be stored in the new system.

What should we specify as functional requirements to the product? The functions the product should have, for example, the screens we want? If we do this, then we

Fig 3.1 Human/computer – who does what?



have chosen the division of labor, and we have decided much of the future human work and the business processes. That is a huge responsibility to the requirements engineer. We should either do it very carefully at *requirements time*, or we should avoid specifying product functions until *design* time.

Furthermore, if we want to base the future system wholly or partly on existing commercial products, they already have their built-in functions and we shouldn't specify our own.

The rest of the chapter looks at ways of dealing with these fundamental issues.