This chapter focuses on a number of issues that have come up in recent years in the design, development, and implementation of scheduling systems. The first section discusses issues concerning uncertainty, robustness and reactive decision making. In practice, schedules often have to be changed because of random events. The more robust the original schedule is, the easier the rescheduling is. This section focuses on the generation of robust schedules as well as on the measurement of their robustness. The second section considers machine learning mechanisms. No system can consistently generate good solutions that are to the liking of the user. The decision-maker often has to tweak the schedule generated by the system in order to make it usable. A well-designed system can learn from past adjustments made by the user; the mechanism that enables the system to do this is called a learning mechanism. The third section focuses on the design of scheduling engines. An engine often contains an entire library of algorithms. One procedure may be more appropriate for one type of instance or data set, while another procedure may be more appropriate for another type of instance. The user should be able to select, for each instance, which procedure to apply. It may even be the case that a user would like to tackle an instance using a combination of various procedures. This third section discusses how a scheduling engine should be designed in order to enable the user to adapt and combine algorithms in order to achieve maximum effectiveness. The fourth section goes
into reconfigurable systems. Experience has shown that system development and implementation is very time consuming and costly. In order to reduce the costs, efforts have to be made to maintain a high level of modularity in the design of the system. If the modules are well designed and sufficiently flexible, they can be used over and over again without any major changes. The fifth section focuses on design aspects of web-based scheduling systems. This section discusses the effects of networking on the design of such systems. The sixth and last section discusses a number of other issues and presents a view of how scheduling systems may look like in the future.

18.1 Robustness and Reactive Decision Making

In practice, it often happens that soon after a schedule has been generated, an unexpected event happens that forces the decision-maker to make changes. Such an event may, for example, be a machine breakdown or a rush job that suddenly has to be inserted. Many schedulers believe that in practice, most of the time, the decision making process is a reactive process. In a reactive process, the scheduler tries to accomplish a number of objectives. He tries to accommodate the original objectives, and also tries to make the new schedule look, as much as possible, like the original one in order to minimize confusion.

The remaining part of this section focuses primarily on reactive decision making in short term scheduling processes. The number of random events that can occur in a short term may, in certain environments, be very high. Rescheduling is in many environments a way of life. One way of doing the rescheduling is to put all the operations not yet started back in the hopper, and generate a new schedule from scratch while taking into account the disruptions that just occurred. The danger is that the new schedule may be completely different from the original schedule, and a big difference may cause confusion.

If the disruption is minor, e.g., the arrival of just one unexpected job, then a simple change may suffice. For example, the scheduler may insert the unexpected arrival in the current schedule in such a way that the total additional setup is minimized and no other high priority job is delayed. A major disruption, like the breakdown of an important machine, often requires substantial changes in the schedule. If a machine goes down for an extended period of time, then the entire workload allocated to that machine for that period has to be transferred to other machines. This may cause extensive delays.

Another way of dealing with the rescheduling process is to somehow anticipate the random events. In order to do so, it is necessary for the original schedule to be robust so that the changes after a disruption are minimal.

Schedule robustness is a concept that is not easy to measure or even define. Suppose the completion time of a job is delayed by $\delta$ (because of a machine breakdown or the insertion of a rush job). Let $C'_j(\delta)$ denote the new completion time of job $j$ (i.e., the new time when job $j$ leaves the system), assuming the sequences of all the operations on all the machines remain the same. Of course,