for it to mature and become mainstream. Today, it is used in major applications such as KOffice and KDevelop. It is typically applied in larger-scale, more complex software systems, where the need for concurrency and out-of-band processing becomes more pressing.

ThreadWeaver is a job scheduler for concurrency. Its purpose is to manage and arbitrate resource usage in multi-threaded software systems. Its second goal is to provide application developers with a tool to implement parallelism that is similar in its approach to the way they develop their GUI applications. These goals are high-level, and there are secondary ones at a smaller scale: to avoid brute-force synchronization and offer means for cooperative serialization of access to data; to make use of the features of modern C++ libraries, such as thread-safe, implicit sharing, and signal-slot-connections across threads; to integrate with the application's graphical user interface by separating the processing elements from the delegates that represent them in the UI; to allow it to dynamically throttle the work queue at runtime to adapt to the current system load; to be simplistic; and many more.

The ThreadWeaver library was developed to satisfy the needs of developers of event-driven GUI programs, but it turned out to be more generic. Because GUI programs are driven by a central event loop, they cannot process time-consuming operations in their main thread. Doing so would freeze the user interface until the operation is finished. In some windowing environments, the user interface can be drawn only from the main thread, or the windowing system itself is single-threaded. So a natural way of implementing responsive cross-platform GUI applications is to perform all processing in worker threads and update the user interface from the main thread when necessary. Surprisingly, the need for concurrency in user interfaces is rarely ever as obvious as it should be, although it has been emphasized for OS/2, Windows NT, and Solaris eons ago. Multithreaded programming is more complicated and requires a better understanding of how the written code actually functions. Multithreadings also seems to be a topic well understood by software architects and designers, and badly disseminated to software maintainers and less-experienced programmers. Also, some developers seem to think that most operations are fast enough to be executed synchronously, even reading from mounted filesystems, which is a couple of orders of magnitude slower than anything processed in the CPU. Such mistakes surface only under extraordinary circumstances, such as when the system is under heavy I/O load or, more commonly, a mounted filesystem has been put to sleep to save power or—heavens!—when, all of a sudden, the filesystem happens to be on the network.

The following section will describe the architecture of the library along with its underlying concepts. At the end of the chapter, we will explore how it found its way into KDE 4.

Introduction to ThreadWeaver: Or, How Complicated Can It Be to Load a File?

To convince programmers to make use of concurrency, it needs to be conveniently available. Here is a typical example for an operation performed in a GUI program—loading a file into a