a change in clock speed, a chip with four cores ought to be able to do four times as much as a chip with a single core. In fact, the speed-up will not be quite linear, as there are other parts of the system that are not made concurrent in the same way. But increases in the overall performance of the system can be obtained by the use of concurrency, and building chips for such concurrent use is far simpler than building chips in which the clock speed is increased.

On the face of it, MMOs and virtual worlds ought to be reasonable candidates for multicore chips and distributed systems. Most of what goes on in an MMO or virtual world, like most of what goes on in the real world, is independent of the other things that are happening in that world. Players go on their own quests or decorate their own rooms. They battle monsters or design clothes. Even when they are engaged with another player or occupant of the world, they are interacting with only a very small percentage of the occupants of the world. This is the characterization of an embarrassingly parallel computational task, and that is just the sort of thing that multiple cores and multiple machines ought to be good at doing.

Although the tasks in these systems may be embarrassingly parallel, the programmers who work on such systems are not trained or experienced in the techniques of either distributed computing or concurrent programming. These are exceptionally subtle fields, difficult even for those who have been trained in them and who have considerable experience in using these techniques. To ask most game programmers to develop a highly concurrent, distributed game server would be asking them to go well outside of their area of expertise or experience.

The First Goal

This context gave us our first goal for the architecture. The requirements for scaling dictated that the system be distributed and concurrent, but we needed to present a much simpler programming model to the game developer. The simple statement of the goal is that the game developer should see the system as a single machine running a single thread, and all of the mechanisms that would allow deployment on multiple threads and multiple machines should be taken care of by the Project Darkstar infrastructure.

In the general case, hiding either distribution or concurrency from the application is not possible. But MMOs and virtual worlds are not the general case. The kind of hiding that we are trying to accomplish comes at the price of requiring a very specific and restricted programming model. Fortunately, it is just the sort of model that lends itself to the kind of programming already used in the server-side components of games and virtual worlds.

The general programming model that Project Darkstar requires is a reactive one, in which the server side of the game is written as a listener for events generated by the clients (that is, the machines being used by the game players, generally either a PC or a game console). When an event is detected, the game server should generate a task, which is a short-lived sequence of computations that includes manipulation of information in the virtual world and communication with the client that generated the original event and possibly with other clients. Tasks can also be generated by the game server itself, either as a response to some