The second stack-level service is the Task Service, which is used to schedule and perform the tasks that are generated either in response to some event received from the clients or by the internal logic of the game or world server itself. Most tasks are one-time affairs, generated because of some action on the client, that read some data from the Data Service, manipulate that data, perhaps perform some communication, and then end. Tasks can also generate other tasks, or they can be generated as periodic tasks that will be run at particular times or intervals. All tasks must be short-lived; the maximal time for a task is a configured value, but the default is 100 milliseconds.

The game or world programmer sees a single task being generated either by an event or by the server logic itself, but under the covers the Darkstar infrastructure is scheduling as many simultaneous tasks as it can. In particular, tasks generated by the server logic will run in parallel with tasks generated in response to a client-initiated event, as will events generated in response to different clients.

Such concurrent execution leads to the possibility of data contention. To deal with such contention requires that the Task Service and the Data Service conspire. Under the covers and invisible to the server programmer, each task scheduled by the Task Service is wrapped in a transaction. This transaction ensures that either all of the operations in the task complete or none of them do. In addition, any attempts to alter values of objects held in the Data Service are mediated by that service. If more than one task attempts to alter the same data object, all but one of those tasks will be aborted and rescheduled to be performed later. The remaining task will run to completion. Once the running task has been completed, the other tasks can be run. Although it is possible for the server programmer to indicate that the data being accessed will be modified, this is not required. If a data object is simply read and then later modified, the modification will be detected by the Data Service before the task is committed. Indicating that modification is intended at the time of read is an optimization that allows early detection of conflicts, but the failure to indicate the intent to modify does not affect the correctness of a program.

Wrapping the tasks in a transaction means that the communication mechanisms must also be transactional, with messages sent only when the transaction wrapping the task that sends the messages commits. This is accomplished through the two remaining core services of the Darkstar stack.

Communication Services

The first of these is the Session Service, which mediates communication between a client and the game or world server. Upon login and authentication, a session is established between the client and the server. Servers listen for messages sent by the client on the session, parsing the contents of the message to determine what task to generate in response to the message. Clients listen on the channel to receive any responses from the server. These sessions mask the actual endpoints to both the client and the server, a factor that is important in the multimachine