MicroCosm^{TM} Host System Architecture

the design of the host-based portion of the MicroCosm system
by
Chip Morningstar

Lucasfilm Ltd. Games Division
October 23, 1985

Introduction

This document attempts to describe the preliminary **MicroCosm** host system architecture. The purpose of this document is to try to get all the design straight before getting bogged down in implementation.

The names of system components referred to in this document match those used in the attached host system block diagram. Please refer to this illustration to minimize confusion.

Overview

The host system consists of 6 primary components: the *Communications Channel Controller*, the *Message Switcher*, the *Transaction Monitor*, the *Asynchronous Notifier*, *Looi* (the **L**ow **O**verhead **O**bject **I**nterpreter) and the *Database* itself. Each of these components is discussed in detail below. Note however, that the Communications Channel Controller and Message Switcher designs are affected by the host system architecture. Depending on the host, their functions may be subsumed by various preexisting host components rather than being implemented as parts of the **MicroCosm** software.

Communications Channel Controller

The Communications Channel Controller is responsible for the physical communications link between the host and the players at home. It encapsulates all knowledge about the packet switching network protocols, modem control, message formats, and so on. Depending on the host system underlying it, it may be a single process that mediates between **N** players and the *Message Switcher* or it may be a collection of (identical) processes each of which talks to a single player. In either case, its function is to handle the receipt and buffering of messages from the player to the host and the buffering and then transmission of messages back to the player. Other than those portions of messages that relate to the transmission medium itself it does not deal in any way with the actual message contents.

Message Switcher

The Message Switcher is responsible for routing messages between the various players and the host and between one player and another.

One of the Message Switcher's duties is handling messages from the players to the host itself. The barrage of messages coming from many sources is processed into a single serial input stream directed at the *Transaction Monitor*. The Message Switcher tags these incoming messages with an indication of their player of origin before passing them along. The Transaction Monitor in turn generates a single serial output stream of similarly tagged messages back to the players. The Message Switcher must route these messages to the appropriate output handler(s) (e.g., the Communications Channel Controller process assigned to a particular player).

Aside from handling the player identification tags, the Message Switcher ordinarily does not meddle with nor examine the contents of messages. However, it does have the capacity to recognize messages that are directed from one player to another. Rather than burdening the Transaction Monitor with the chore of taking these messages apart only to have to put them back together again and send them back out, it simply passes them directly to their destinations without further processing by the host. Such messages are, of course, checked for conformance with the player-to-player message protocols, but the contents of the messages themselves are not dealt with in any way.

Transaction Monitor

The Transaction Monitor unpacks messages from the players into *transactions*. Each transaction is a request from a player's home system for the host to perform some action or provide some information. Each such request boils down to a call to *Looi*.

The Transaction Monitor decodes incoming messages, generates the appropriate calls to Looi, assembles the result into a response message (if one is required) and then ships this response back out to the player via the Message Switcher. It will undoubtedly have to perform some error checking on the incoming request stream, but this must be minimal because of the time-critical nature of transaction processing.

The Transaction Monitor also receives requests from the *Asynchronous Notifier*. These are similar to ordinary requests from players, except that they originate within the system and therefore do not have to be unpacked as messages nor checked for certain types of errors. The responses to requests generated by the Asynchronous Notifier are generally directed to particular players rather than back to the points of origin. The Transaction Monitor must keep this straight.

Asynchronous Notifier

The Asynchronous Notifier is the means by which the actions triggered by a request from a player can in turn propagate further actions that result in messages being sent out to other players. The idea is that the code handling the behavior of some object can issue requests on the behalf of individual players. These players then receive the responses to these requests even thought they never issued anything themselves. This is how players are informed of asynchronous events; that is, events that effect them but which they did not initiate.

The Asynchronous Notifier is called by routines invoked under Looi. It simply queues up the requests and holds them until the Transaction Monitor is ready to process them. The requests are fed to the Transaction Monitor in first-in, first-out order in between its handling of actual messages from players.

Looi

Looi is the **MicroCosm** **L**ow **O**verhead **O**bject **I**nterpreter. It is described in gory detail in the companion document **Looi**. In essence, Looi is a remote-procedure-call interface to an object oriented database. Each incoming request contains the identifier of the object to which the request is directed, a number indicating the action or information being requested, and additional request dependent information handled as ‘‘arguments’’ that are internally significant to the code Looi invokes to actually handle the request. Each call to Looi yields a result that the Transaction Monitor deals with appropriately. Some times this result is interpreted as ‘‘all is well but don't bother to respond’’, in which case the Transaction Monitor does nothing but go on to the next request.

Database

The **MicroCosm** world model is object oriented. This means that everything in the universe is represented as an object of one sort or another. The Database itself is the collection of information that describes these objects. Each object consists of four components: the *object state*, the *host behavior code*, the *home behavior code*, and the *imagery*.

The imagery and the home behavior code are significant only to the home systems. The imagery is the set of pictures used to display the object on the player's computer screen. The home behavior code is the software that runs in the player's home computer to simulate the object's actions. The host simply acts as a central repository for these, providing them for home systems' use on demand.

The host behavior code is the software that runs in the host to execute the object's actions. It is coded in Noodl and invoked by Looi. Please refer to the documents **Noodl** and **Looi** for details.

An object's imagery, home behavior code and host behavior code are collectively called its *object description*. Each object in the world belongs to a *class* of objects that are functionally equivalent. A group of objects of the same class can share a common object description, since it is the same for each object in the group.

In contrast to the object description, an object's state is unique to the individual object. The object state is simply a series of bytes (how many depends on the class of object) that describes the particulars of an object as distinguished from other objects of the same type. This includes such things as location and owner as well as things that are entirely dependent on the nature of the object. The interpretation of the various bytes of the object state information is internal to the host behavior code.