Other Systems on MicroCosm™

A "LUCASFILM UNIVERSE" DOCUMENT
some comments on putting third-party software into the **MicroCosm** *environment*
by

Chip Morningstar

Lucasfilm Ltd. Games Division

October 17, 1985

Introduction

This document discusses the issues associated with installing third-party software in the **MicroCosm** environment. This includes both adding independently created objects and interactions to enhance the fantasy of the **MicroCosm** universe and using **MicroCosm** as a gateway into other (relatively unrelated) processes running on the network host alongside it.

One of the problems in the construction of an elaborate system such as **MicroCosm** is the volume of creative material which much be produced to give the resulting world a feeling of depth and a sense of variety. The homogeneity of a single mind's particular creative rut, even if the creator is quite imaginative, can result in a sameness to all things that is quite unlike the real world. The real world has variety because the number of independent minds working to shape it is large. We can obtain variation in the same manner by having a variety of outside developers create world elements.

Third-Party Objects

The first and most obvious way of enriching the depth and texture of the world is to allow third-parties to add objects to the world object database. Broadly speaking, objects can be said to consist of two components: image and behavior. *Image* is the way the object *appears* on the screen. *Behavior* is what the object *does*.

An object's image is displayed on the home system's screen. There may be a number of different home systems in use and each of these will require images for each object. One possible strategy for achieving this is to create a separate image for each object for each machine. This is the "brute force" approach. Unfortunately, as the number of objects increases the process of adding a new machine to the system becomes more and more unwieldy (since all the images will have to be created anew for the new machine). Similarly, as the number of machines supported increases, the process of adding a new object to the system becomes more and more unwieldy (since an image will be required for each possible home system). Clearly some sort of device-independent image representation scheme is in order, and that is exactly what we are going to develop. The details of the device-independent image representation are outside the scope of this document. We merely wish to point out that such a representation will be required.

An object's behavior happens in both the home system and the host system at once. This is because the aspects of object behavior which relate to display control, decision making, and direct interaction with the player must take place in the home system because of the host/home communications bottleneck. Aspects of object behavior which involve alterations to the permanent state of the object or that of other objects or which involve access to "privileged" information must take place in the host system because of security considerations. An object's behavior is thus implemented by two separate (but interlocking) pieces of program code, one of which runs in the home system and the other of which runs in the host.

The portion of an object's behavior programming which must reside in the home system suffers from the same problem that the images do: a separate implementation is required for each machine on which it must run. There are two strategies for coping with this. The first is to program the object behavior routines in a higher-level language of some sort, so that only one version has to be written. For each target machine there is a compiler or interpreter which deals with the language. This is the conventional means of dealing with a proliferation of target systems for a piece of software. There is a problem however, and that is that a number of the target systems that we are contemplating are so primitive in their computational power that the overhead associated with a higher-level language would adversely effect performance to the point where the system would not be able to function successfully in the manner we require. The second approach is to realize that the range of potential target processors is quite a bit smaller than the range of target machines in general. For example, the Apple II, the Commodore 64, and the Atari 800 all have a 6502 CPU in them. As long as the behavior software does not have to deal with any system-dependent aspects of these machines, the same identical object software can run on all three. We will design the run-time environment of the home system so that this will be the case.

During system operation, objects may be kept stored one of two places: in the long-term storage of the home system or in the host database itself. The advantage of keeping the objects in the home system is that we do not have to wait through the delays while they are transferred (at 300 baud!) from the host to the home when actual use is required. The advantage of keeping the objects in the host database is that the set of objects can grow arbitrarily large, free of the restriction that it fit withing the few hundred kilobytes of storage available on the typical home system. Also, the object set can be expanded and modified at will without the active participation of the players.

Since both schemes have major advantages and disadvantages, we will employ a compromise hybrid. We will store as many objects as we can in the home system. Any additional objects that the home system needs it gets from the host system. The home systems use conventional cacheing techniques to select the particular set of objects that they will store locally. Objects will be chosen for local storage on the basis of their frequency of use. Each home system will have its own individually selected cache of objects chosen to be most appropriate for its player's situation in the universe. The host will keep a complete set of all objects and make them available on demand. The idea, then, is that most of the time the home system uses objects out of local storage and only has to wait to get an object from the host when it encounters a new object for the first time. It then downloads this new object and keeps it, throwing out some other, older object that hasn't been referenced in a while if it has to make room.

One pleasant consequence of this scheme is that it provides a built-in distribution system for the addition of new objects to the system. Adding an object to the universe consists of inserting a new entry in the host system database which describes the object and then putting instances of this object into the world where people will find them. Whenever people encounter these instances and try to interact with them, their home systems download a copy of the relevant parts of the object into their machines, and away they go.

Thus, to create a new object, the following pieces must be created: an image to display the object with (which should be expressed in our **MicroCosm** device-independent image format), an object behavior program module for the host, and a set of object behavior program modules for the home (one for each target processor). These components must then be entered into the host database so the players can get at them.

To allow the creation of objects by third-parties, we will need the following: a programmer's library and toolkit for programming objects on the host, a matching toolkit for the various home systems, a graphics tool for composing the object images, and a package of documentation describing the various programming environments and setting forth conventions, standards and procedures for the creation and installation of new objects.

Potentially, the interface for making additions to the host database could be provided through the **MicroCosm** universe itself, by means of a special privileged object which we give to developers. This object would be an upload tool that would enable the developer cum player to take objects defined inside his ‘‘home’’ machine and give them to the host for incorporation into the world.

Gateways

The second avenue to enriching the detail and texture of the **MicroCosm** universe is a feature we are calling a *gateway*. A gateway is a very special sort of object which is placed at a fixed location somewhere in the simulated world. The host resident end of this object behaves very differently from they way conventional objects do. What it does is transfer responsibility for communications with the player from the **MicroCosm** system software in the host to some other program that might be running in the host at the same time. As far as **MicroCosm** is concerned, the player temporarily disappears into a sort of limbo. The alternate software to which the player has been handed then interacts with him completely independent of **MicroCosm**. Presumably, at the same time this occurs, the piece of the gateway object which resides in the player's home system similarly hands control over to a matching piece of software which is loaded there.

The function of this independent exchange can be just about anything. It can be something entirely independent of the **MicroCosm** world. For example, it might be a completely conventional database access service for the stock exchange. It can also be something related to or synergistic with the **MicroCosm** universe. For example, two players entering a gateway together could be connected to each other via a two-player strategy game that simply uses the network host as a communications channel between the players. This game could be something that the players just play, with **MicroCosm** simply acting as an access mechanism. Alternatively, the game could in some way effect the players' situations in the fantasy. For example, the gateway might put the players into a mode where they can joust, using completely different software for the graphics and communications, but the outcome of their battle could alter their avatars' health property in the **MicroCosm** world. This is accomplished by allowing external processes in the host to make requests of the **MicroCosm** database processer just as players may. Since these processes are running in the host they may be presumed to be trustworthy and can therefore be allowed to request things that they players themselves would not be allowed to do directly (such as altering the health property of an avatar).

The installation of such software on the host is readily accomplished. All that is needed is an appropriate library of routines to enable the independent software to communicate with **MicroCosm** itself. The usual documentation describing methods and standards will be needed, of course, and some sort of organizational procedure will be required of the operators of the host system to ensure that third-party installed software does not violate the integrity or the spirit of the **MicroCosm** system.