Overview of Entities in MOSES

The MOSES Project:

Meta Operating System And Entity Shell

Daniel J. Pezely

20 March 1991

---

1 Introduction

The MOSES Project goals are to develop a system suitable for virtual environments. Developing for virtual environments means handling arbitrary data efficiently within a distributed system while provide system independence when implemented.

The fundamental design element which MOSES uses is the notion of an *entity*. Here, we define exactly what we mean by an entity and leave other papers to describe specific uses.

2 Entity-Based Systems

Today's systems designs whether for operating systems, CAD systems, computation systems, etc, require distributed and parallel processing for today's needs. Our system design is no different.

Unlike modern operating systems (OS's) such as Mach [1] or Amoeba [2], our design uses the notion of *entities* which function similarly to tasks and objects. However, there is more than just functional value to entities.

3 Notion of an Entity

The term *entity* means: that which exists. And in this context, entities are information; therefore, everything which can be represented symbolically can be an entity. This symbolic information may be executable functions or may be data. As data, entities may be lists of named functions to be executed or lists of access rights or lists of coordinates and attributes to be passed to a graphic rendering service. As functions, entities work just as more traditional function forms in software. Services, which are specific functions, are also entities.

3.1 Recursion in the Design

Everything is an entity. Entities may be nested forming hierarchies or recursive chains. Looking at these forms as programming language objects, the entity would be the root or base class from which all other data class (data types) derive from. [1] Continuing the analogy, entities have three main components: memory, function, and communication. Grouping constructs and renderable data objects can be seen as types of entities which simply contain more or different information.

3.2 The Three Components of Entities

In general, we say that everything is an entity, and an entity can be anything. There a many components to the notion of an entity. The first is that the definition allows recursion, thus all levels of derived entities are entities also. But to give some basic attributes to an entity, we associate three elements needed for useful existence: memory, function, and communication.

[1] Although we compare entities to programming objects, this is just to give the reader a frame of reference. Some view root-objects, as is done in SmallTalk, to be bad. We are not limiting our selves to that paradigm.

The boundaries between each of the three components which all entities have are very vague.

Memory refers to all storage. No matter where data is stored, a uniform access is provided. At the user [2] level, no distinction is made between immediate memory, such as RAM, and mass storage, such as disks, tapes, and database servers.

All memory appears unified and under the exclusive ownership and control of the user. The unification bridges remote and local memories as well as devices and other system resources. Various bridging techniques allow for the user's illusion to work. [4]

Function refers to all processing power. Having memory access is not very useful without being able to process it. The system has built-in routines which are accessible to all entities as primitives, and external routines may be used as services in a client-server relationship. Both types of routines provide function to the user. Additional functionality may come from interprocess communication and remote procedure call paradigms.

Communication refers to the most commonly used set of functions. A simple entity which does anything and is not just data must perform some communication task. Communication can be with any input/output channel which include storage device access.

I/O channels connect external servers and clients executing in parallel, local processes executing concurrently, local data files, and local virtual devices provided by the native operating system. (See [3] for the respective descriptions of sockets, pipes, file descriptors, and devices.) Communication via storage device access is achieved by one user writing to the memory thereby allowing another user to access that memory.

4 Types of Entities

Every device of the underlying operating system should be represented as an entity. Although the three main elements of an entity are memory, function, and communication, there also exists a RAM-entity, disk-entity, database-entity, execute-function-entity, execute-program-entity, execute-remote-service-

[2]Throughout this paper, we intend *user* to mean a person using the system or an application.

entity, send-message-entity, receive-message-entity, etc. These specific entities relate to (and may access) device drivers on operating systems.

Such entities provide a system-independent interface to resources by enforcing their own communication protocol. This allows applications to avoid the idiosyncrasies of the native OS.

A grouping construct is needed since entities may nested. Grouping abstracts a collection of entities appearing to be a single entity. Again, entities are recursive by definition, so each hierarchical level or chain link is an entity in itself. An example would be an operating-system-entity containing other entities such as the entities for the kernel, the memory manager, the input/output driver, the file system, etc. We establish grouping by naming or referencing the grouped entities.

To render specific information, some entities might contain additional information which a render-entity could use. Here, rendering refers to any output, not just graphic output.

The distinction between the different types of entities (data types) has absolutely nothing to do with the internal system data structures; instead, distinctions are made at the user level. Information is data—period. The same information may be used as data by multiple entities (specifically, the function elements of entities) and the use is determined by the context. Different *contexts* for entities may use different elements or different interpretations, but the *content* of entities remains the same.

5 Entities as Building-Blocks

In most of today's operating systems designs, objects are used similarly to the object-oriented programming model. Designers intended kernels to be small, making use of external objects which provide various services for accessing main memory, controlling the file system, scheduling processes, etc. The service objects may all inter-communicate with the system kernel and possibly with each other. (See [2], [5], and [1] for example uses of this paradigm.)

Also, UNIX [3] systems provide users with small discrete programs and utilities which may be linked together in a *pipeline* serving a task which each utility's author might not have imagined. [6] This task could translate high-level commands into small packets for communication, as one example.

[3]UNIX is a Registered Trademark of AT&T Bell Labs.

With the resources which all entities have, entities are ideal for use as both distributed processing objects and pipeline utilities. Since each entity can access memory, initiate a task, and communicate, we have all the requirements necessary for using entities as objects and utilities.

6 Summary

Entities are information—any information—which can be represented symbolically. Information may be functions as well as data, but we do not distinguish between different entities. Different entities come about by containing more or different information. Three elements, or resources, are inherent to entities: memory, function, and communication. The usefulness of these resources may be compared to the objects in the message-passing operating systems / object-oriented programming paradigm and to the pipelining ability of UNIX utility programs. We leave the possibilities of application to the reader for the moment, but articles demonstrating some uses follow.

References

[1] Forin, A., Barrera, J., Young, M., Rashid, R. "Design, Implementation and Performance Evaluation of a Distributed Shared Memory Server for Mach," *Proceedings of the Winter USENIX Conference*, January 1989.

[2] Tanenbaum, A.S.; Renese, R. van; Stavern, H. van; Sharp, G.J.; Mullender, S.J.; Jansen, J.; Rossum, G. van. "Experiences with the Amoeba Distributed Operating System," *Communications of the ACM*, vol. 33, no. 12, December 1990, 46-63.

[3] Tanenbaum, A.S., *Operating Systems: Design and Implementation*, Prentice Hall, Englewood Cliffs, NJ, 1987.

[4] Pezely, D.J., *The DataSpace Function Specifications*, Human Interface Technology Laboratory, Washington Technology Center, University of Washington, Seattle, WA, 1991.

[5] Pike, R., Presotto, D., Thompson, K., Trickey, H., "Plan 9 from Bell Labs," Expected to be published in *1990 UKUUG Conference*, 1990.

[6] Pike, R., *Software Tools*, Prentice Hall, Englewood Cliffs, NJ, 1976.