ENTITIES

William Bricken
June 1995

Entities

An *entity* is a collection of resources which exhibits behavior within an environment. The entity-based model of programming has a long history, growing from formal modeling of complex systems, object-oriented programming, concurrent autonomous processing and artificial life.

Entities act as autonomous systems, providing a natural metaphor for responsive, situational computation. When a single entity resides on a single node, the entity is a stand-alone executable program that is equipped with the VEOS functionalities of data management, process management, and inter-entity communication. In a virtual environment composed of entities, any single entity can cease to function (if, for example, the node supporting that entity crashes) without effecting the rest of the environment.

Entities provide a uniform, singular metaphor and design philosophy for the organization of both physical (hardware) and virtual (software) resources in VEOS. Uniformity means that we can use the same editing, debugging, and interaction tools for modifying each entity.

The biological/environmental metaphor for programming entities provides functions that define perception, action and motivation within a dynamic environment. *Perceive* functions determine which environmental transactions an entity has access to. *React* functions determine how an entity responds to environmental changes. *Persist* functions determine an entity's repetitive or goal-directed behavior.

Synchronization of entity processes (particularly for display) is achieved through *frames*. A frame is a cycle of computation for an entity. Updates to the environment are propagated by an entity as discrete actions. Each behavioral output takes a local tick in local time. Since different entities will have different workloads, each usually has a different frame rate. As well, the frame rate of processes internal to an entity is decoupled from the rate of activity an entity exhibits within an environment. Thus, entities can respond to environmental perturbation (reacting) while carrying out more complex internal calculations (persisting).

Systems-Oriented Programming

In object-oriented programming, an object consists of static data and responsive functions, called methods or behaviors. Objects encapsulate functionality and can be organized hierarchically, so that programming and bookkeeping effort is minimized. In contrast, entities are objects which include interface and computational resources, extending the object metaphor to a systems metaphor. The basic prototype entity includes VEOS itself, so that every entity is running VEOS and can be treated as if it were an independent operating environment. VEOS could thus be considered to be an implementation of *systems-oriented programming*.

Entities differ from objects in these ways:

- *Environment:* Each entity functions concurrently as both object and environment. The environmental component of an entity coordinates process sharing, control and communication between entities contained in the environment. The root or global entity is the virtual universe, since it contains all other entities.

- *System:* Each entity can be autonomous, managing its own resources and supporting its own operation without dependence on other entities or systems. Entities can be mutually independent and organizationally closed.

- *Participation:* Entities can serve as virtual bodies. The attributes and behaviors of an inhabited entity can be determined dynamically by the physical activity of the human participant at runtime.

In object-oriented systems, object attributes and inheritance hierarchies commonly must be constructed by the programmer in advance. Efficiency in object-oriented systems usually requires compiling objects. This means that the programmer must know in advance all the objects in the environment and all their potential interactions. In effect, the programmer must be omniscient. Virtual worlds are simply too complex for such monolithic programming. Although object-oriented approaches provide modularity and conceptual organization, in large scale applications they can result in complex property and method variants, generating hundreds of object classes and forming a complex inheritance web. For many applications, a principled inheritance hierarchy is not available, forcing the programmer to limit the conceptualization of the world. In other cases, the computational interaction between objects is context dependent, requiring attribute structures which have not been preprogrammed.

Since entities are interactive, their attributes, attribute values, relationships, inheritances and functionality can all be generated dynamically at runtime. Structures across entities can be identified in real-time based on arbitrary patterns, such as partial matches, unbound attribute values (i.e. abstract objects), ranges of attribute values, similarities, and analogies. Dynamic programming of entity behavior can be used by programmers for debugging, by participants for construction and interaction, and by entities for autonomous self-modification. Since the representation of data, function, and message is uniform, entities can pass functional code into the processes of other entities, providing the possibility of genetic and self-adaptive programming styles.

Entity Organization

Each entity has the following components:

- A *unique name*. Entities use unique names to communicate with each other. Naming is location transparent, so that names act as paths to an entity’s database partition.

- A *private partition* of the global database. The entity database consists of three sub-partitions. The *external* partition contains the entity’s environmental observations. The *boundary* partition contains an entity’s attributes and its observable form. The *internal* partition contains recorded transactions and internal structure.

- Any number of *processes*. Conceptually, these processes operate in parallel within the context of the entity, as the entity's internal activities. Collectively, they define the entity’s autonomous behavior.

- Any number of *interactions*. Entities call upon each others' relational data structures to perform communication and joint tasks. Interactions are expressed as perceptions accompanied potentially by both external reactions and internal model building.