Glossary

Compiled by the VEOS/MOSES Project Teams *

1990, 1991

Standard distributed processing and operating systems terms apply to this project, as do the tools and techniques used for artificial intelligence.

- *connection-less communication*
  Network messages of this type are sent but not necessarilly verified, thus may arrive out of order or mangled, if at all. Without additional transport control, this model is suitable for non-wide area applications such as small metropolitan and local area netowrks. Also known as a *Datagram* transmission. See *connection-oriented communication* also.

- *connection-oriented communication*
  All network messages of this type are verified for corruption and passed on to the user in order of transmission. Also known as a *virtual circuit.* See *connection-less communication* also.

- *daemon*
  A service program which the user does not directly control at run-time is considered to be a daemon. See *kernel service* also.

- *datagram*
  See *connection-less communication.*

- *DataSpace*
  The unity of all memory of all *entities* which exist within the environment provided by MOSES forms the DataSpace. It's an associative

*VEOS/MOSES Project people: William Bricken, Daniel Pezely, Dav Lion, David Doll, Geoff Coco, Mark Evenson, Mike Almquist

shared memory space with address access attempts are permitted (but may be denied due to stale addresses from clients) and forms a virtual environment. All memory for the kernel, kernel services, and applications using the operating system which includes all global, local, and temporary variables, should be stored within the DataSpace. Each user has the illusion of ownership, and while may actually possess one subspace of the total DataSpace, all subspaces are accessible as a whole. See [6] and [7].

- *entity*
  Any information which exists and represented symbolically is an entity. An entity may have various forms, descriptive or executable, and represented by a *grouple*. Within the kernel, entities possess three elements: memory, function, and communication. See [8].

- *flat grouple*
  In the context of programming and implementation, a flat grouple is an allocated `GroupleHeadClass` data structure which has no terms (`GroupleTermClass` structures) referenced within it. See also *grouple*, *null grouple*, and *term*.

- *grouple*
  This is an *entity* represented by lists of information (tuples) which may be nested via internal addressing. See also *messages*, `GroupleHeadClass`, *flat grouple* and *null grouple*. See [6].

- `GroupleHeadClass`
  This is the data structure used in implementations and called a *grouple*. All allocated grouples are referenced from the *DataSpace* at run-time. The structure is similar to file system i-nodes. See *grouple*.

- `GroupleSymbolClass`
  This is a type-definition used in implementations also called *symbols*. It provides a generic data type for storing atomic data of any kind. The interpretation of the data type is done by the *user*. See *grouple*.

- `GroupleTermClass`
  This is the data structure used in implementations, referenced by `GroupleHeadClass` structures, and called *terms*. Terms can reference only three types of

data: symbols, sublists, and functions. All non-sublists will either be functions or atomic data. All atomic data are stored as `GroupleSymbolClass` types, and functions are referenced indirectly. See *grouple*.

- *inode*
  File-system information node. Usually contains information about a file pertaining to location of data blocks, creation time, update time, access time, file size, etc. Filenames are not associated with inodes; names are kept in directories, thus allowing multiple filenames pointing to the same file (inode). See *grouple* for VEOS/MOSES related item.

- *IPC*
  Inter-process communication. In the context of MOSES, the intercommunication may be across the network to another entity. Such network traversal should be transparent to the user.

- *kernel service*
  A program or entity which provides functionality to the meta operating system is considered a kernel service. This is a *daemon* in that although it may be invoked by a *user*, the user does not interact with it directly. The scheduler is an example of a common kernel service. See [9].

- *Linda*
  This is one of the early shared memory models with a decent implementation in both performance and simplicity. There are five operations usually associated with the system: *in* (read-in and remove from the memory), *out* (write-out something to the memory), *eval* (evaluate something from the memory), *rd* (read-in and copy from the memory), and two predicates, *inp* and *rdp*, which do not wait for data. See [1].

- *message*
  The structure of messages for message-passing and distributed processing are *grouples*. The actual syntax will be the Common Lisp[2] programming language syntax since grouples are a hierarchical data structure and do not lend easily to transmission in that form.

- *MOSES*
  Meta Operating System and Entity Shell: a project initiated by Daniel

Pezely to develop an implementation of the VEOS design at HITL. See [5].

- *native operating system*
  This is the operating system which MOSES will run as an application of in the early versions while the unique features of our system are being developed. The ideal native operating system (ie, the ones used during development) are the POSIX compliant systems such as SunOS and BSD.

- *null grouple*
  In the context of programming and implementation, a null grouple is an unallocated `GroupleHeadClass` structure. See also *grouple* and *empty grouple*.

- *participation*
  Usually given in the context of human participation, this means human-computer interaction. In a true virtual reality system, this is taken to the ultimate level: interactive graphics, three dimensional sound, and real-time everything including photo-realistic rendering. On a more plausible frame for the Nineteen Nineties, the participant is the user, but has much more control than feeding parameters and digesting output. See [3].

- *property list*
  Since we compare our data structures to Lisp s-expressions, it is important to state how Lisp property lists fit into the design. Within the DataSpace, a protoerty list is nothing more than a grouple which lists the properties of another grouple. The property list will typically be a *sublist* of the grouple which it describes. This descript, however, may be anything which the user wants it to be and is not restricted to Lisp properties.

- *protocol hierarchy*
  A protocol hierarchy refers to the interaction of individual protocols within a *protocol suite*. This hierarchy is not necessarilly a *protocol stack*, since layers may be by-passed. The DoD ARPANET Internet Protocol (IP) suite is an example of a hierarchy. See [4] for a more complete description.

- *protocol stack*
  A protocol stack is most commonly associated with the International Standards Organization's (ISO) Open Systems Interconnections (OSI) protocol stack reference model. This is a rigid stack of layers in which each layer can only communicate with its immediate neighbors and cannot by-pass any layers. See [4] for a more complete description.

- *protocol suite*
  A protocol suite refers to the collection of individual protocols used within a *protocol stack* or *protocol hierarchy*.

- *RPC*
  Remote procedure call. This is a paradigm for a *connection-less* (ie, *datagram*) transmission between network entities.

- *s-expression*
  Internal storage data structure from the Lisp programming language which allows nesting of lists. See [2].

- *space*
  Also called a subspace of the *DataSpace*: a category for a *grouple* which names or addresses other grouples as its only data permitted by convention. This convention is used for classifications purposes only, thus not enforced. See [10].

- *subspace*
  See space.

- *sublist*
  A nested *grouple*. Referenced from within a *term*.

- *symbol*
  In the context of programming or implementation, a symbol is the `GroupleSymbolClass` type-definition. It's generic data type for storing any class of data, and interpretation of symbol values is up to the *user*— floating point real numbers, signed integers, etc— are stored as unsigned long integers.

- *term*
  In the context of programming and implementation, a term is the

`GroupleTermClass` data structure. This is the element which an allocated *grouple* references which, in turn, references symbols, sublists, and functions.

- *terminal*
  This is the end-point of communication where the *user* is located. Virtual terminals (in the traditional sense) may be controlled by applications or networked terminals. Actual terminals (hardware: chips, metal, plastic, and the whole lot) will typically be controlled by a human participant. The hardware will resemble the graphics workstation class of computer systems, not necessarily only dumb terminals.

- *tuple space*
  In the context of VEOS or MOSES, the *DataSpace* is used. Traditionally, a tuple space is used for storing arbitrary length lists of information. However, the DataSpace permits nesting which breaks the definition of a tuple. See *grouple* also.

- *user*
  User refers to the client program code or a person accessing the system. The user is also an *entity*.

- *VEOS*
  Virtual Environment Operating Shell: a project initiated by William Bricken, Principal Scientist, at the Human Interface Technology Laboratory, Seattle. The goals of the VEOS Project are to develop a system platform which will make for a suitable virtual environments while not being tied down to today's technology. The golden rule of design is that *everything* will change.

- *virtual circuit*
  See *connection-oriented communication*.

References

[1] Carriero, N., Gelernter, D., "Applications Experience with Linda," *Symposium on Principles and Practice of Parallel Programming, Proceedings of the ACM/SIGPLAN*, Volume 23, Issue 9, September 1988, pp. 173-187.

[2] Steele, G.L.,Jr. *Common Lisp*, Second Edition, Digital Press, Cambridge, MA, 1990, pp. 12-13, 238-246, 247-287.

[3] Bricken, M., "No Interface To Design," ed: Benedikt, M., *Cyberspace: The First Steps* MIT Press, Cambridge, MA, 1991.

[4] Spragins, J.D., *Telecommunications: Protocols and Design*, Addison-Wesley Publishing Company, New York, NY, 1991, pp. 118-149.

[5] Pezely, D.J., *An Introduction To MOSES*, Human Interface Technology Laboratory, Washington Technology Center, University of Washington, Seattle, WA, 1991.

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

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

[8] Pezely, D.J., *Overview of Entities in MOSES*, Human Interface Technology Laboratory, Washington Technology Center, University of Washington, Seattle, WA, 1991.

[9] Pezely, D.J., *MOSES Kernel Implementation Design*, Human Interface Technology Laboratory, Washington Technology Center, University of Washington, Seattle, WA, 1991.

[10] Pezely, D.J., *Building Multiple, Distributed, Shared, Virtual Environments with Entity Projection and Transportation Facilities*, Human Interface Technology Laboratory, Washington Technology Center, University of Washington, Seattle, WA, 1991.