The CLOS Project: Towards an Object-Oriented Environment for Application Development

Igor Burdonov Victor Ivannikov German Kopytov
Alexander Kosachev Sergei Kuznetsov

Institute for Problems of Cybernetics, USSR Academy of Sciences, Moscow

Abstract

The CLuster Operating System (CLOS) is based on the asynchronous cluster concept. A very simple and small CLOS kernel with a fully machine independent interface provides CLOS with openness and portability. CLOS is intended as an object-oriented environment for the design and development of complex applications with internal parallelism particularly in advanced information systems.

1 Introduction

The object-oriented approach in programming [TDK89] is widely recognized and considered as a promising basis for new software techniques. This approach is relevant to programming languages and systems [AOC+88,Str86], database management systems (DBMS) [KTT89,HK89] and certainly operating systems [ABLN85,YTR+87].

While many features of the object-oriented approach are peculiar to the CLuster Operating System (CLOS) outlined in this paper, it was not our intent to analyze and compare the CLOS underlying ideas with those of "canonical" object-oriented approaches peculiar, for example, to Smalltalk systems [GR83].

This paper presents an outline of the basic CLOS concepts. Implementation details of CLOS are not described here. Our goal is to show how the CLOS environment may be used for complex applications design and development.

The most important CLOS features are:

- system flexibility and maximum openness
- reduced set of basic machine-independent concepts
- simple and small kernel with full machine independency
- unified approach for system and application developments via object orientation.

The rest of the paper is organized as follows. The second section describes the main CLOS concept - asynchronous cluster - elaborated on through a simple example. The third section discusses the notion of a subsystem - the basis for application design and development. The set of tools supporting such design and development is also briefly discussed. The paper concludes by reporting on the status of the CLOS project.

2 Asynchronous Cluster - The Basic CLOS Concept

The main CLOS concept is that of an asynchronous cluster which is very close to a CLU cluster [LSAS77]. A CLOS cluster provides encapsulation and is, therefore, an indissoluble aggregate of a processed object and the operation set on it. The concrete data structure representing the object proves "hidden" by the set of operations defined on the object.

The main CLOS extension of this traditional notion consists in the cluster having the property of asynchronous execution. The notion of asynchronous cluster integrates not only the notions of data and operation set but the notion of process too. Operation inside a cluster is executed independently of the invoking cluster, which continues its execution after the invocation; receiving return parameters requires a response invocation and corresponding mutual synchronization. The CLOS kernel buffers messages containing parameters which one cluster wants to pass to another. We will refer to invocation of one cluster (client) to another cluster (object) as message sending and refer to receiving corresponding parameters by cluster-objects as message receiving.

The second extension of cluster semantics deals with the problem of synchronization. We claim that any synchronization is the synchronization of operations on objects, therefore, it is one of the object management functions and should be performed by the cluster-object itself. To perform such synchronization it is sufficient for the cluster-object (according to its state) to select or not to select messages that cluster-clients send into the various entries corresponding to operations.

To protect an access of cluster-client to any (or some) entries of a cluster-object we use the concept of connection of one cluster to an entry group of another cluster.

A cluster includes one or more entries. The entry corresponds to a semantically separate operation on an object. At the end a message is sent to one of the entries of some cluster.

All cluster entries join up in one or more entry groups; each entry may be included in several different groups. The capability of access to a cluster is defined at the entry group level: a cluster able to access the entry group of another cluster has the right to send messages to any entry of this group. Such access capability is described by the object called connection. A connection contains a pointer to the cluster-object entry group as well as a pointer to the cluster-client owning this connection.

Some entry groups are defined as static. Immediately after cluster creation access will be enabled only to such entry groups. "Who is allowed to access?" is another question discussed below when describing the process of subsystem assembling. In the execution process a cluster-object is able to permit or forbid access dynamically to any of its entry

groups which might be non-static. To enable an access to any of its entry groups a cluster-object calls up the kernel primitive which creates a connection to the given entry group; immediately after this creation the owner of the connection will be the cluster-object itself. Further, it may pass this connection as a message parameter to another cluster-client, which will be the new connection owner. The cluster-object may create several connections to its entry group and pass them to different clusters-clients.

To complete our discussion of the concept of asynchronous cluster we will consider briefly the structure of cluster storage. Cluster logical storage is segmented. There are two statically defined memory segments in each cluster: command segment and local data segment. At cluster executing both of these segments are permanently mapped into its virtual memory (i.e. opened); their location in virtual memory is fixed and remains unchanged. A cluster may have an arbitrary number of additional (dynamic) memory segments. These segments may be mapped into virtual memory by special kernel primitives.

A segment may be passed to another cluster as a message parameter: in this case a segment is automatically removed from virtual memory of the cluster-sender (if it was opened) and changes its owner. Further, it may be opened only in the new owner virtual memory while the previous owner has no capability of access to the segment.

The main objective of the CLOS kernel is to provide inter-cluster communications. It provides a very small set of machine-independent primitives that allow the user to send and receive messages, to create and destroy connections and to map and unmap segments. Certainly specific communication protocols may be built below this underlying level, particularly the remote procedure call protocol.

Let us consider a simplified example of a hypothetical cluster-file. It has five entries and three entry groups.

The entry group (OPEN-FOR-READ, OPEN-FOR-WRITE) is static. A connection to the entry group (CLOSE, READ) is supplied as the response to a message to the entry OPEN-FOR-READ, and a connection to the entry group (CLOSE, READ, WRITE) as the response to a message to the entry OPEN-FOR-WRITE. These connections are destroyed at the execution of the CLOSE operation. Each entry possesses an interface of a "remote procedure's" type. The response message from CLOSE remote procedure serves as the signal for the completion of "close file" operation.

The cluster may be in one of three states: CLOSED, OPENED-FOR-WRITE and OPENED-FOR-READ. When the cluster is in the last state it supports an opening number counter. Since operations in the cluster-object are executed concurrently with the cluster-client it is possible to send a series of messages without the necessity of waiting for response messages. One of the READ (WRITE) entry parameters is a segment served as a buffer for file reading (writing). The cluster-file returns this segment as a response message parameter.

3 Subsystems and Complex Applications Design and Development

All other functions of the operating system itself and application systems are implemented with the basic CLOS objects - clusters. There is, however, one important programming limitation with clusters - each cluster has only single control flow. If the application requires several control flows then it should be based on another CLOS concept - the concept of subsystem.

A subsystem is a statically defined set of components with statically established relationships (i.e. connections) between them. A subsystem component is a subsystem or a cluster. The process of subsystem construction from components takes place before its usage and is called subsystem assembling.

From the standpoint of its usage a subsystem is analogous to the cluster. It also corresponds to some object (but in contrast to the clusters, subsystems are complex composite objects) that implements cluster operations and calls operations in other objects. An external operational medium may be considered as an additional fictitious subsystem component - an environment.

To define a subsystem's outer interface the subsystem description should contain the subsection of subsystem entry groups description and the subsection of environment entry groups description. A subsystem entry group is identified with some subsystem component entry group and finally is implemented by a static entry group of some cluster. An environment entry group is a group of operations which are used by the subsystem but are defined in some object implemented outside this subsystem. Finally the environment entry group is implemented by a static entry group of some cluster placed outside the subsystem.

The internal structure of the subsystem consists of subsystem components and the relationships between them. Since the subsystem component is a subsystem itself (in particular - a cluster) its outer interface also contains a description of component entry groups and component environment entry groups. From the standpoint of a component its environment is "a rest" of a subsystem, i.e. the aggregate of other subsystem components including a fictitious component - "subsystem environment".

To assemble a subsystem it is necessary for each component X to identify each environment entry group Xi described in the outer interface of X with some entry group Yj of a component Y described in the outer interface of Y. This identification operation is called relationship establishment or linking. Y may be the same as X (a component uses operations implemented by itself) or may be another subsystem component. If Y is a subsystem environment then the entry group Yj is implemented outside the subsystem and X is just that part of the subsystem (not necessarily one) that uses such an outer implemented entry group.

There may exist one or more connections to a cluster environment entry group that are statically defined in the cluster. Usually there exist several connections to a subsystem component entry group (in particular to the subsystem environment entry group). These connections may be defined in several subsystem components (finally in clusters) since

the environment entry groups of those components may be linked with just the same subsystem component entry group.

The subsystem assembling language is a constituent of the CLOS system to design program systems. The construction of a subsystem from earlier prepared components corresponds to the "top-down" design. For such a design the possibility of using "non-elaborated" components is provided. Only the outer interface is defined for the "non-elaborated" component. The main assembling action - linking - is based only on the outer interfaces of components and the whole subsystem.

A component that only has the outer interface defined and the internal structure left undefined is called a frame. With top-down design a subsystem at first consists of frames only. At the next stage the elaboration of those frames is performed: they are described as subsystems which may contain frames too. It is possible to have several such stages. Sooner or later the frames will be refined by already available clusters or fully defined subsystems.

A frame does not define uniquely an outer interface of the subsystem by which it may be refined. Actually a frame corresponds to a family of subsystems. This family is defined by a frame outer interface. Each subsystem of the given family must have all static entry groups which are defined for the family, but it may have other static entry groups. Furthermore, a set of subsystem environment entry groups must be a subset of the family environment entry groups set.

A completely defined subsystem (without frames) is the unit of CLOS dynamic loading. The loaded subsystem may be changed in the process of its execution: the new components may appear (by dynamic loading), the existing components may be destroyed, new relationships between the components may appear and previous relationships may vanish. However, a subsystem notion is kept when it executes and a subsystem structure is used in processor and memory management.

4 Project Status

The first experimental version of CLOS was implemented in 1988 on the Soviet workstation "Electronika-85" (fully compatible with a DEC Professional 350). The first implementation was based on a specially developed programming language system including slightly extended C and two small specialized languages. The first CLOS version was bootstrapped in a UNIX[1] environment.

Currently CLOS is ported onto 32-bits workstation "Besta-88". Its more powerful hardware based on Motorola 68020/68030 allows implementation of the CLOS project in full. The CLOS file system has been implemented and is presently under experimental operation. A new, more powerful and convenient language system that contains the cluster programming language Cclos and a set of tools for subsystem design and assembling is being developed. A distributed variant of CLOS is in the design phase.

[1]UNIX is the trademark of AT&T in the US and other countries.

5 Acknowledgements

U.P. Smirnov took active part in the design and implementation of CLOS at an early stage.

References

[ABLN85] G.T. Almes, A.P. Black, E.D. Lazowska, J.D. Noe. The Eden System: A Technical Review. IEEE Trans. Software Eng.,SE-11 (1), pp. 43-58, Jan. 1985.

[AOC+88] G.R. Andrews, R.A. Olsson, M. Coffin, I.J.P. Elshof, K. Nilsen, T. Purdin. An Overview of the SR Language and Implementation. ACM Trans. Prog. Lang. and Syst., 10(1), pp. 51-86, 1988.

[GR83] A. Goldberg, D. Robson. Smalltalk-80: The Language and its Implementation. Addison-Wesley, Reading, MA, 1983.

[HK89] S.E. Hudson, R. King. Cactis: A Self-Adaptive, Concurrent Implementation of an Object-Oriented Database Management System. ACM Trans. Database Syst., 14(3), pp. 291-321, Sept. 1989.

[KTT89] T.F. Keefe, W.T. Tsai, M.B. Thuraisingham. SODA: A Secure Object-Oriented Database System. Computers and Security,8(6), pp. 517-533, 1989.

[LSAS77] B. Liskov, A. Snyder, R. Atkinson, C. Schaffert. Abstraction Mechanisms in CLU. Comm. ACM, 20(8), pp. 564-576, 1977.

[Str86] B.Stroustrup. The C++ programming language. Addison-Wesley, Reading, MA, 328 pp., 1986.

[TDK89] R.P.Ten Dyke, J.C. Kunz. Object-Oriented Programming. IBM Syst. J., 1989, 28(3), pp. 465-478, 1989.

[YTR+87] M. Young, A. Tevanian, R. Rashid, D. Golub, J. Eppinger, J. Chew, W. Bolosky, D. Black,R. Barom. The Duality of Memory and Communication in the Implementation of Multiprocessor Operating System. Operating Syst. Review, 21(5), pp. 63-76, Nov. 1987.