Report Number: WUCS-82-9

1982-02-01

Communicative Processes: A Model of Communication

Takayuki D. Kimura and Will D. Gillett

This paper introduces a conceptual model of communicative organization as a part of the formal semantic study of distributed computation. The model includes, as communication primitives, three independent modes of communication: mailing, posting, and broadcasting. Mailing models thin-wire communication, and posting models shared memory communication. While broadcasting is not prominent in today's parallel programming languages, it has an important role to play in distributed computation. Other fundamental notions in the model are process, symbol, site, process class, symbol class and site class.

... Read complete abstract on page 2.

Communicative Processes: A Model of Communication

Takayuki D. Kimura and Will D. Gillett

Complete Abstract:

This paper introduces a conceptual model of communicative organization as a part of the formal semantic study of distributed computation. The model includes, as communication primitives, three independent modes of communication: mailing, posting, and broadcasting. Mailing models thin-wire communication, and posting models shared memory communication. While broadcasting is not prominent in today's parallel programming languages, it has an important role to play in distributed computation. Other fundamental notions in the model are process, symbol, site, process class, symbol class and site class.

COMMUNICATIVE PROCESSES:
A MODEL OF COMMUNICATION

T. D. Kimura and W. D. Gillett

WUCS-82-9

Presented at the 10th IMACS World Congress
on Systems Simulation and Scientific Computation
August 8-13, 1982, Montreal, Canada

Department of Computer Science

Washington University

St. Louis, Missouri 63130

February 1982

Communicative Processes: A Model of Communication

T. D. Kimura and W. D. Gillett

Department of Computer Science
Washington University
St. Louis, MO 63130

Summary

This paper introduces a conceptual model of communicative organization as a part of the formal semantic study of distributed computation. The model includes, as communication primitives, three independent modes of communication: mailing, posting and broadcasting. Mailing models thin-wire communication, and posting models shared memory communication. While broadcasting is not prominent in today's parallel programming languages, it has an important role to play in distributed computation. Other fundamental notions in the model are process, symbol, site, process class, symbol class and site class.

I. Introduction

As VLSI technology becomes more available, distributed computation will become more prevalent, and communication resources will become more expensive than computational resources [1]. As a part of our efforts to develop a design methodology for distributed medical information systems [2, 3] and as a part of the formal semantic study of distributed computation, we have developed a conceptual model of distributed computation based on the notion of communicative organization. A communicative organization is a collection of active entities (processes), distributed over time and space, communicating with each other through a linguistic mechanism to achieve the common goal. We view computation as a social activity performed by a communicative organization, and a programming language as a specification language for such an organization. A program specifies the social structure (local or global) of a communicative organization as well as the behavior of individual communicating entity in the organization.

The objective of this paper is to introduce a conceptual model of communicative organization in a summary form without formal definitions. We call our model Mercury and its specification language the Mercury language. We will concentrate on a discussion of communication primitives. Due to the space limitations, many other aspects of the model must be omitted.

Mercury can be considered to be an extension of LISP with communication capabilities. Our model shares the general philosophy and some of the objectives of the Actor model [4]. By introducing process and symbol as two separate object categories, we eliminate the conceptual difficulty of differentiating between actor as a symbol and actor as a process.

A model of communicative organization, as a communication model, should contain a mechanism for symbol interpretation; this is lacking in the traditional communication models, such as the Shannon-Weaver model [5]. The notion of physical space also should be included in any model of distributed computation. Mercury includes the capability of symbol interpretation as an inate property of each individual process. It also includes the notion of site, which is an abstraction of both geographical space and an operating system environment with a set of language capabilities and intrinsic service processes. This is an initial step of our effort to develop a programming language for distributed computation which incorporates the notion of space. No metric is applied to the abstractions of either time or space.

This work was supported in part by the National Center for Health Services Research under Grant HS03792 and the National Institutes of Health under Grant RR00396.

II. Objects

An object is any entity that can be named in the Mercury language. There are six categories of objects.

Symbol: A passive element of communication (what is to be communicated). A symbol is a structured, static entity used for naming and describing objects when it is interpreted by a process. A message is a symbol, a datum is a symbol, and so is a program.

Process: An active element of communication (what communicates). A process is a dynamic entity that interprets symbols, executes communication actions. It can create and destroy other objects, and can communicate with other processes via mailing, posting and broadcasting.

Site: An environment of communication (where communication takes place). A site is a collection of language capabilities and intrinsic service processes. These are resources necessary to support communication activities of processes in a particular boundary of space. Each site has a finite number of neighboring sites. A process at a site can execute only programs that are written in a language which the site supports.

Symbol class: A set of symbols sharing some common structural property. A symbol class defines a symbol type by which a selective communication can be achieved. A process may choose to receive only certain types of messages based on the symbol class of the message. A symbol class is also used to represent a function from a symbol class to a symbol class.

Process class: A set of processes sharing the same program. Two processes belong to the same class if and only if their actions are controled by the same program, i.e., they execute the same program. A process class is specified by the program that every process in the class executes.

Site class: A set of sites sharing the same capabilities. Two sites belong to the same class if and only if they have the same set of intrinsic processes and the same set of language capabilities. A site class is determined by (1) specifying a set of intrinsic processes and (2) specifying the syntax and semantics of the languages that every site in the class is capable of supporting. The syntax of a language can be specified as a symbol class, and its semantics also can be specified as a symbol class representing a function from the syntax of the language to the syntax of the Mercury language.

III. Process Organization and Structure

A communicative organization is a dynamic collection of processes, distributed over different sites, communicating with each other through mailing, posting and broadcasting of symbols (Figure 1). Posting and broadcasting provide only intra-site communication capability, while mailing can be used for both inter-site and intra-site communication.

Processes can create and destroy symbols, other processes, and neighboring sites. They can change the topology of the sites and migrate from one site to another. When a process migrates, the new site must support the language in which the program of the process is written.

When a site is destroyed, all processes at the site are destroyed. When a process is destroyed, all of its descendants are destroyed. Therefore, the processes in a communicative organization, at any moment, constitute a set of hierarchies, each element of the set structured by pedigree. The visibility and community formation (i.e., who can communicate with whom) is controlled through both the genealogical location in the pedigree and the geographical location of each process.

A process can execute an action according to a prescription it carries within itself. We call this prescription the <u>gene</u> (program) of the process. It is a symbol that specifies the sequence of actions the process executes. During the lifetime of a process, the gene of the process never changes. A set of processes carrying the same gene forms a process class.

A process gene can be encoded in different languages. We say that two processes whose genes are encoded in the same language belong to the same <u>species</u>. The type of language interpreted by a process is inate to the process and identical for all processes of the same species. The Mercury language is intended for encoding a process gene. It is a universal language because of its capability of prescribing any process behavior within the framework of the Mercury model.

The structure of processes in Mercury is illustrated by Figure 2. The bulletin board, receiver, tuner, name and gene each contain at most one symbol. The mail box contains an arbitrarily long queue of symbols. The name and gene cannot be changed. The timer is an alarm clock which is reset whenever some action is executed by the process. An alarm goes off after a predefined time period. Every timer at the same site has the same duration, but there is no synchronization among the timers. This time-out capability is required for formulating fault-tolerant and error recovery procedures in a distributed communicative organization.

The set of actions executable by a process can be divided into three categories: (I) object creation and deletion, (II) communication, and (III) community structuring. Each action will be explained below in English with pseudocode for the Mercury language.

(I) Creation (and naming) and destruction of symbols, processes and sites.

(1) <u>create</u>(m,w) : create an object with specification w and name m.

When w specifies a symbol class, a new symbol will be created within the process and the new symbol's name will be m. Similarly, if w specifies a process class (i.e., a specification of a gene), then a new process will be created at the same site with w as its gene, and it will be named m. If w specifies a site class, a new site will be created as a neighbor of the site in which the creating process resides. The new site will have the capabilities designated by w.

(2) <u>destroy</u>(m) : destroy the object with name m.

Only the creater process can destroy an object by its name. When a process is destroyed, all of its descendants are destroyed, and when a site is destroyed, all processes currently at the site and their descendants (possibly at other sites) will be destroyed.

(II) Communication with other processes through mailing, posting, and broadcasting.

(3) <u>send</u>(m,p) : send the symbol m to the process p by mailing.

<u>Mailing</u> is a selective symbol dispatch which can reach processes at any site. A process may receive mail from more than one process. It is a private mode of communication in the sense that no process other than the receiver has access to the mailed symbol. The arrival time at the destination process is unknown; however, it is guaranteed that the sequence of arrival at a common destination will be the same as the sequence of dispatch from the same process. The symbols received by a process are queued in the <u>mailbox</u> of the process in the order of arrival. The process can select a symbol to retrieve by designating a symbol class by the retrieval command <u>mail(t)</u>, which designates the oldest mail of type t currently in the mailbox.

(4) <u>post</u>(m) : post the symbol m on the bulletin board.

<u>Posting</u> is a non-selective information dispatch which can be accessed by only processes at the same site; thus, posted information is semi-private. Each process has one and only one bulletin board which can post at most one symbol at a time. The posted bulletin remains available until it is overwritten by a new bulletin. A posted symbol can be <u>scanned</u>, by any process acquainted (in the genealogical or casual community, to be discussed below) with the posting process residing at the same site through the command <u>bulletin</u>(p), which designates the current symbol posted by the process p. The scanning action produces a copy of the posted symbol within the scanning process.

(5) <u>broadcast</u>(m) : broadcast the symbol m.

<u>Broadcasting</u> is a completely public information dispatch which reaches every process at the same site. Each process has one and only one <u>receiver</u> which registers the most recently broadcasted message (symbol) at the site; the content of the receiver is designated by <u>message</u>(). Each process is also equipped with a <u>tuner</u> by which the process can select only certain type of messages to be received. The command <u>tune</u>(t) will set the receiver to register only symbols of the symbol class t.

(III) Community structuring by process migration, reconfiguration of site network topology, introduction of one process to another, and orphaning of a son process.

For each process in the Mercury organization at a given moment, there exists a collection of processes with which the process can interact legally through mailing, posting, or broadcasting. We call this collection the <u>community</u> of that process at that moment. There are three kinds of communities for each process P: The <u>geographical</u> community of P is the set of all processes that reside at the same site as P. The <u>genealogical</u> community of P is the set of all ancestors and their immediate sons. The <u>casual</u> community of P is the set of all processes that are <u>introduced</u> to P. The process P can send mail to the members of its genealogical and casual communities. (P may not belong to the receiver's community.) It can scan the bulletin of members of its geographical community who also belong to its genealogical or casual community.

A process can change its own community and that of another by the following commands.

(6) <u>move</u>(p,s) : move the process p to the site s.

Only the father process can move a son process from one site to one of its neighbor sites.

(7) connect(s1,s2) : make the sites s1 and s2 immediate neighbors.

The site of the process executing this command must be the immediate neighbor of both s1 and s2.

(8) disconnect(s1,s2) : disconnect the neighboring sites s1 and s2.

The same condition as (7) applies here.

(9) introduce(p1,p2) : introduce the process p1 to the process p2.

The process executing this command must contain both p1 and p2 in its genealogical or casual community. After this action, p1 belongs to the casual community of p2, but p2 may not belong to p1's community.

(10) orphan(p) : orphan the son process p.

Only the father process can orphan a son process. After this command, all of the p's ancestors will be deleted from the genealogical community of p. Thus, the process p may not mail to or scan the bulletin of any ancestor unless p is introduced by some other process, but p's father can mail to p and scan p.

The control structure of the Mercury language is that of Markov's normal algorithms [6] with nondeterminism. It also can be viewed as an extension of Dijkstra's Guarded Commands [7] augmented with priority levels. A program (gene) consists of an ordered sets of guarded commands. If there is an executable command in the first group, it will be selected for execution. If there is more than one executable command in the group, then one is chosen nondeterministically to be executed. If there is no executable command in the first group, then the second group is searched for an executable command, etc. After each execution, always the first group will be searched. This effectively provides a priority control for command execution. There is no explicit sequencing control structure available, and no nesting of control structure is allowed.

IV. Conclusion

We have outlined a conceptual model of communicative organization by informally describing the fundamental notions included in the Mercury model. Space limitations restrict us from demonstrating the capability of the model to represent the semantics of common programming concepts, such as (1) block structure, (2) subroutine, (3) coroutine, (4) interrupt, (5) monitor, (6) abstract data type, (7) exception handling, and (8) package in Ada. Note that these conventional programming concepts do not require the notion of site, because no conventional programming language has the concept of space as a primitive. Similarly, the broadcasting mode of communication is seldom needed to represent conventional programming concepts. Yet, it is our belief that mailing, posting, and broadcasting serves distinctly different purposes in a genuinely distributive organization.

It may seem that broadcasting is "simulatable" by another communication mode, say mailing; however besides the different effects the two modes manifest in time and space (e.g., mailing takes longer than broadcasting), the simulation requires either that every process "knows" everyone else by name, or that there exists a process (post office) which is "known" to everyone and which "knows" everyone at the site. The first case is unrealistic when the community size becomes large because of the conceptual complexity of each process. The second case is unacceptable for a model intended for distributed computation because such a process introduces a centralized communication center which is a potential bottleneck in any communicative organization. A similar argument can be made against posting as a simulation of broadcasting.

Although broadcasting may have to be simulated by some other form of communication in a specific implementation, its concept should not be ignored. It is a powerful tool for expressing the communication of certain types of information, e.g., error recovery and triggering of constraint maintenance in a database [8]. The exclusion of this concept hinders both the designer and implementer.

V. References

[1] Mead, C., Conway, L., Introduction to VLSI Systems. Addison-Wesley, 1980.

[2] Cox, Jr., J. R., Kimura, T., Moore, P., Gillett, W., and Stucki, M. J., "Design Studies Suggested by an Abstract Model for Medical Information System," Proceedings of Fourth Annual Symposium on Computer Applications in Medical Care, Washington D. C., (November 1980) pp. 1485-1494.

[3] Kimura, T., Cox, Jr., J. R., Gillett, W., "An Abstract Model of an Unstratified Database System," Proceedings of Fourteenth Hawaii International Conference on System Sciences, (January 1981), pp. 115-126.

[4] Hewitt, C., Smith, B., "Towards a Programming Apprentice," IEEE Transactions of Software Engineering, SE-1,1 (March 1975) pp. 26-45.

[5] Cherry, C., On Human Communication. MIT Press, 1957.

[6] Galler, B. A., Perlis, A. J., A View of Programming Languages, Addison-Wesley, 1970.

[7] Dijkstra, E. W., "Guarded Commands, Nondeterminancy and Formal Derivation of Programs," CACM 18,8 (August 1975) pp. 453-457.

[8] Tsichritzis, D. C., Lochovsky, F. H., Data Models, Prentice-Hall, 1982.

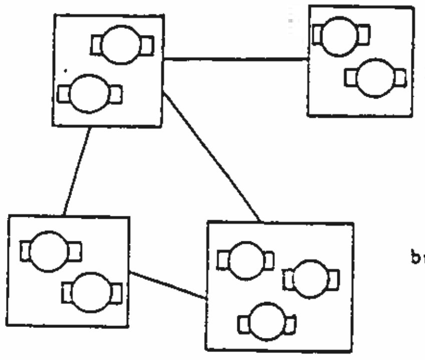

Figure 1: Communicative Organization

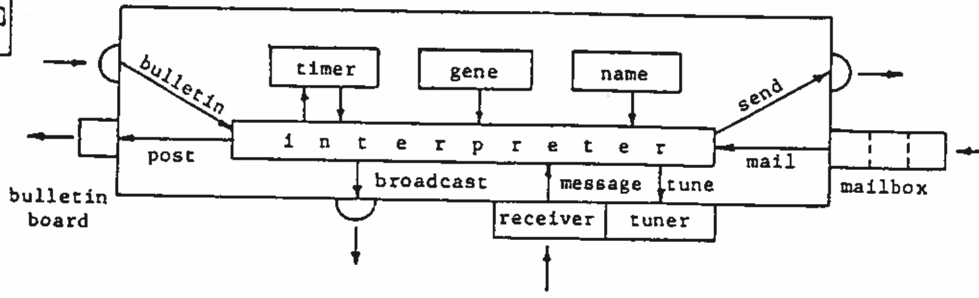

Figure 2: Process Structure