Transparent Ada Rendezvous in a Fault Tolerant Distributed System

Roger Racine
C.S. Draper Laboratory
555 Technology Sq.
Cambridge, MA 02139
(617) 258-2489

Abstract

There are many problems associated with distributing an Ada program over a loosely coupled communication network. Some of these problems involve the various aspects of the distributed rendezvous. The problems addressed in this paper involve supporting the "delay" statement in a selective call and supporting the "else" clause in a selective call. Most of these difficulties are compounded by the need for an efficient communication system. The difficulties are compounded even more by considering the possibility of hardware faults occurring while the program is running. With a hardware fault tolerant computer system, it is possible to design a distribution scheme and communication software which is efficient and allows Ada semantics to be preserved. An Ada design for the communications software of one such system will be presented, including a description of the services provided in the seven layers of an International Standards Organization (ISO) Open System Interconnect (OSI) model communications system. The system capabilities (hardware and software) that allow this communication system will also be described.

Background

There are many reasons for using distributed computer systems. Key among these is the ability to recover when a fault occurs in one of the computing sites. Other reasons include increased throughput and separate subsystem development by different contractors (or the ability to buy off-the-shelf subsystems).

The Ada programming language has the concept of parallelism built in (in the form of tasks). To expand this concept to include running one Ada program on multiple computers, with communication taking place over some network, creates a number of problems. One must consider how to specify the location of processes, the distributed elaboration of the program, whether the various software engineers involved are able to tell where various components will be located, what should happen in the case of hardware faults, and how to implement the various communication mechanisms available in Ada.

In the interest of space, the focus of this paper will be in the area of the distributed Ada rendezvous. In a rendezvous, one task calls an "entry" in another task. The first task then waits for the server task to "accept" the call. Conversely, if the server task attempts to accept the call before it is made, it will wait. When the caller and server both have arrived, the rendezvous occurs, with parameters passed to the entry, a block of code executed, and any output parameters passed back to the caller. The two tasks are then free to execute again in parallel.

That is the simple rendezvous. Ada also provides selective calls and selective accepts, timed calls and timed accepts, and guarded accepts.

A selective call is a call which must be accepted immediately. If any other task is being served, or the server task is anywhere in its execution except waiting at the accept statement, the call is cancelled. The language requires that the server task be checked to determine if the entry is available. It is necessary, therefore, for two messages to be sent over the network to obtain the information. The first message will ask for the rendezvous; the second will either be a message saying the rendezvous could not be accepted or else the second will contain the result of the rendezvous.

A timed call is one which must be accepted within a given amount of time. The call will be cancelled if it is not accepted within that time. The semantics of a timed call are different depending on the value of the delay. If the delay is zero or is negative, the semantics of a selective call will be followed. At least two communication messages must be sent over the network. However, if the delay is positive, and the rendezvous is known, by the caller, not to be able to occur within the delay period, it is not necessary to even attempt the rendezvous. All that is necessary is to wait the delay period before giving control back to the calling task. No communications over the network will be required in this case.

A selective accept allows a server task to accept a call to one entry arbitrarily from among a list.

A timed accept allows a server to wait only a finite time for a task to call one of its entries. At the end of the time period, if no task has called, the server task will regain control, and will execute alternate code.

A guarded accept allows a server to accept, in a selective accept, one of a list of entries based on conditions. The conditions on accepting the various entries will be checked at run-time, and one of the "open" entries will be picked.

The selective accept, the timed accept and the guarded accept can all be managed on the server task's processor, without any network communication.

To a designer of a distributed computer system, these built-in constructs raise a number of issues:

- Should one even allow the use of the Ada constructs when communicating between two tasks on different computers? Specific communication packages could be provided instead, with pragmas used to make the Ada constructs "erroneous". The assumption used here is that the Ada constructs should be used so that the application does not need to know where various tasks are.

- What happens if a task does a timed entry call, but the computer of the called task fails at some time before the rendezvous occurs? It is possible to send enough messages to ensure that the Ada semantics are followed, even in the case of failures, but the time involved in transferring the messages is large. If the rendezvous is extremely inefficient, it is not usable.

- The rendezvous semantics specify that once the rendezvous has started, it must complete before the calling task can continue. What should happen if the processor running the server task fails during the rendezvous?

The AIPS Project

For the Advanced Information Processing System (AIPS), reliability is the most important issue, with efficiency also being a priority issue. The NASA sponsored AIPS project will produce a flexible, fault tolerant, distributed, real-time computer system. It has been designed in terms of "building blocks", such that different applications, such as deep space probe or a manned space station, could use the components.

The building blocks include the following (this is not an exhaustive list. It only includes those blocks pertaining to intertask communication):

- Fault Tolerant Processors (FTPs). One FTP consists of two (duplex) or three (triplex) microprocessors, each executing identical instructions. A triplex FTP has the ability to mask a single fault from the rest of the system. A duplex FTP can determine that a fault exists.

- A fault tolerant Intercomputer (IC) network. This network is a triplicated circuit-switched nodal network with sufficient links in each network to be able to reach all FTPs on the network after experiencing a single fault in the network. Because the network is triplicated, it is possible to have reliable communication with multiple faults.

- Systems software flexible enough to handle an arbitrary number of FTPs connected to the network. The network management process must be able to recognize faults in the network, and reconfigure

it automatically (and invisibly to applications processes). The communications software must allow for any number of FTPs to communicate. Systems management software must be able to reconfigure the system (functionally "move" a group of tasks from one FTP to another) if extreme failures occur. Local FTP management software must be able to reconfigure the FTP in the presence of processor failures (downmode to a duplex from a triplex, for example).

- Local Operating System (OS) software capable of working alone (simplex), in duplex or in triplex. The local OS is concerned with the tasks on one FTP (the local scheduler, local rendezvous software, etc.).

Fault Tolerant Distributed Ada

In the absence of fault tolerance, it is difficult to design a rendezvous scheme between tasks on different computers without multiple transmissions over the network to ensure processors remain active throughout the wait for the rendezvous. A message would need to be sent to request a rendezvous. An acknowledgement would be necessary within some time limit in the case of a timed or selective call, to make sure the call has been received and put on the queue. Another message would need to be sent stating that the entry is accepting the call. If the caller is making a timed call, and the delay runs out before this message is received, a message could be sent to take the call off the queue. Finally, the results of the rendezvous can be sent back to the caller.

These messages make up a minimal set of transmissions over the network at the highest level. There might be other transmissions at a lower level to make certain that each complete message is received correctly.

In the fault tolerant AIPS system, the problem of unknown processor failures does not exist. If one of the processors in an FTP fails, the fault is detected. If possible (in a triplex FTP, for example), processing continues normally. If it is not possible to isolate the fault, the System manager will reconfigure such that functions on the failed FTP are run on a different FTP.

For this type of system, it is possible to design an efficient communication service to implement the Ada rendezvous. Because the tasks involved are virtually assured of continuing execution throughout the rendezvous, little error detection needs to be done in the communications software of the processor containing the calling task.

For the case of the timed rendezvous, with a positive delay value, the design calls for the operating system on the called processor to time the wait, if the delay value is larger than the minimum necessary to transmit the rendezvous request and receive a response back. If the entry is not accepted within the given amount of time, a message will be

sent back to the calling processor, and the calling task can execute alternate code. The only messages that need to be sent would be the initial message that the caller wants to communicate, and the final message that the server is finished (for whatever reason). If the delay amount is smaller than the minimum needed to transfer messages, no communication is needed. The calling task can be given control back after the specified delay.

If the delay amount is zero or negative, or if it is a conditional call, the messages still must be sent, and the rendezvous might occur.

The IC network services will keep track of whether the called task is moved from one FTP to another. It is also possible that the network will be reconfigured while the tasks are waiting to communicate. All of this will be transparent to the application program.

The design for the intertask communication has been subdivided into two parts, the local communication and the interprocessor communication. The local communication consists of the "normal" rendezvous between two colocated tasks. The interprocessor communication consists of doing the same thing across an IC network.

The "glue" between these two services is called the "context manager". Its function is to determine, for each attempted rendezvous, whether the called entry is on the same processor as the calling task. If it is, the local communication service is invoked. If the called entry is on some other processor, the IC network service is invoked.

The design of the context manager includes a table of locations of what are known as "migratables". As was mentioned above, when a fault is detected, tasks can be transferred to another FTP. The tasks will be grouped into large units. All the tasks within a migratable unit will always be colocated; if they are moved, they will move as a block. Therefore, the table of locations can be organized hierarchically. This will allow a fast algorithm to be designed to determine in which FTP a called task is being run.

The network services are organized into layers, as in the ISO Open Systems Interconnect model. The highest layer, the Application layer, will provide the interface between the context managers on the FTPs, and the IC network.

The interface between the context manager and the IC network has been designed to be as similar as possible to the interface between the context manager and the local communication service. This is not a necessity, but since the context manager is a potential bottleneck, there should be no translation of data to support different interfaces.

The Application layer is responsible for the Ada rendezvous semantics. When a rendezvous is with a task on another FTP, this layer must make sure the semantics are followed. With a fault tolerant system, this layer is fairly simple. At system initialization, a table of task to task communications is used to create logical connections between each pair. When the rendezvous is actually requested by the caller,

this layer sends the input parameters, along with the timeout value, to the Application layer on the server's FTP. The server's Application layer calls the server with the appropriate delay (adjusted to take into account communication delays). When the rendezvous is complete, the Application layer returns output parameters. If an exception is raised or the call times out, a message is sent back to the caller specifying the problem. The Application layer on the caller's FTP then either gives control back to the caller at the appropriate point or raises the specified exception to the caller.

The other layers, except the lowest software layer (the Network layer), are designed to support general network services (not just Ada communication), and are not affected by the fault tolerance of the system.

The Presentation layer is responsible for translating data when the format on the receiver is different from that on the sender. The system being built (the proof of concept, or POC, system) has all processing sites identical; therefore no transformation routines will be coded.

The Session layer is responsible for verifying the legitimacy of the communication. It is possible for users (in some anticipated applications) to attempt to communicate with tasks to which they should not be allowed access. A table of allowed communications will be checked for all connections.

The Transport layer is responsible for determining the hardware destination of the communication. It will have a table of locations for the various tasks. If a communication destination is changed (if a task is moved to another processor), this layer will be notified so that communication can continue.

The Network layer is responsible for detecting and masking hardware faults. On a triplex FTP, each processor is connected to one of the three IC networks for transmission. Each processor has receivers all three networks. Masking faults is not trivial when receiving messages from processors which: are not fault tolerant, are duplex FTPs or are triplex FTPs. It is, however, still much faster than detecting faults through multiple acknowledgments at the Application layer. In fact, in the usual case of no faults, a triplex FTP's Network layer needs to do very little processing to obtain (reliable) data for each of the three processors. It is only in the presence of faults that extra processing needs to be done.

The Datalink layer is responsible for sending packets across the network. It contends with the other FTPs for the network using a modified Laning poll which allows one triplex FTP to win the triplicated network in the presence of a single fault. This protocol is somewhat more complex than is necessary for a single network. This added complexity, on the POC, adds a 10% overhead on each transmission. The Datalink layer uses the HDLC protocol to transmit data over each network.

The Hardware layer has two bit-protocols: the data bit and the poll bit.

Conclusion

In the presence of faults on a system which is not fault tolerant, it is difficult to design an efficient communication system to support the Ada rendezvous. For the fault tolerant AIPS computer system, however, it is much easier to design the upper layers of the ISO OSI communications model. The Network layer and the Datalink layer each have more processing to do for each communication, but the amount of processing is small when there are no errors occurring, and the number of communications can be reduced to two at the Application layer.

The result is an extremely reliable, efficient communication system allowing Ada tasks to communicate as if they were on the same FTP.

Distributed systems have many benefits. The distribution allows the system to run in parallel, giving more throughput than in a nondistributed system. The distribution allows the system to be reconfigured in the presence of faults. The distribution allows the system to be able to continue in the presence of damage, by putting the various computers in different parts of the vehicle. Adding hardware fault tolerance complements the distribution by allowing the software to isolate faults and in many cases to mask the fault. This allows software systems such as the communication system to be much simpler than in systems which are not fault tolerant.