A Study for Inter-Node Communication in Functionally Distributed Systems

A. Fantechi, S. Gnesi, N. Lijtmaer

Istituto di Elaborazione dell'Informazione
Via S. Maria 46
Pisa - Italy

ABSTRACT

This work has been developed on the basis of a previous work made inside a project for a functionally distributed system (Cnet), in which an inter-node communication mechanisms was defined in terms of Ada™ packages. This paper gives a more general framework of properties for desired communication schemes.

Some design issues such as the use of formal verification techniques and the integration of the communication mechanism in an high level language are then discussed.

™ Ada is a registered trademark of the U. S. Government, Ada Joint Program Office.

This work has been supported by the Consiglio Nazionale delle Ricerche (Italian National Reserch Council).

1. INTRODUCTION

Functionally distributed systems can be viewed as a collection of heterogeneous computer nodes, providing several capabilities and sizes, that are connected by means of a communication network [Liskov 85].

Key aspects of this approach are node autonomy and inter node cooperation. The main consequence of the assumption of autonomy is that the programmer, not the system, must control where programs and data reside.

The system may not breach the autonomy of a node by moving processing to it for purposes of load sharing.

Although the system provides the maximum autonomy, this approach does not coincide with the protocol based network systems (layered) approach because coherence and cooperation among nodes give still a unique but distributed view of an operating system.

The operating system must provide to the user mechanisms for inter-node communication, configuration and fault-tolerance.

A natural way to provide these mechanisms to the user is to include them in a system language.

A project based on this approach is the Cnet project (inside the Computer Science Program of the CNR, the Italian National Research Council), aimed at implementing a distributed system on a high bandwidth local network along with its distributed operating system and software development environment [Cnet 85]. The hosts are heterogeneous computers, while logical homogeinity is obtained through the unique high level language acting as the system language as well as the application language. Ada has been chosen as the system language, and linguistic extensions have been proposed using the abstraction facilities of Ada to cope with inter-node communication, configuration and fault-tolerance.

In this paper, we want to extend the principles behind the definition of the Cnet Inter-Node Communication Mechanism to a wider framework in which the properties of more general mechanisms are given as an aid for their design and implementation, and other design issues are discussed.

The environment in which the communication mechanisms are defined remains adherent to the one of the Cnet Distributed System.

Hence, the following basic characteristics remain fixed:

1) the distributed system is seen at the high language level as a collection of (virtual) nodes. The following communication mechanisms abstract the physical network;

2) asynchronous communications with decentralized control, are used, in which no single entity can have complete knowledge of the entire state of the system, asynchronous communications are typical of a local network;

3) an indirect message passing scheme is adopted, which uses ports in order to achieve modularity and a reduced run-time overhead on the passing of a single value. The ports are passive objects of two kinds:
   i) **input ports**: a node can create an input port, through which it can receive messages. The visibility of the input port in the network is given by a unique port name. The establishment of such unique name must be made in agreement with the visibility naming rules of the high level language chosen;
   ii) **output ports**: an output port can be considered as an interface towards an input port belonging to another node; this interfacing is achieved by connecting an output port on a node to an input port on another node via the unique name of the latter;

4) the ports can have an associated type, so allowing the exchange of typed messages, and extending the type checking rules of the high level language chosen to the communication over the network.

The next section gives some formal properties of four communication schemes defined according to the previous characteristics. Such properties are shown to be useful as an aid to the design and implementation of the communication mechanisms.

The third section discusses some other important issues on the design of the communication mechanisms in an existing high level programming language.

2. COMMUNICATION PROPERTIES

In this chapter we give some basic communication schemes, characterizing them with formally defined properties. It is then shown that such properties are useful in the design of the communication mechanisms.

2.1 Communication Schemes

As we have said in the previous section the distributed system is seen as a collection of (virtual) nodes. We consider four possible primitive communication schemes for the communication between virtual nodes. communication mechanisms follow the basic characteristics sketched in the previous section.

i) point to point scheme allows the exchange of messages of the same type through typed ports. The scheme must be open to cover either the case of two cooperating nodes or the client-server case. Message types might exhibit a complex user defined structure;

ii) broadcast scheme allows the rapid dissemination of information and guarantees delivery of a message to all nodes faced on the network. The broadcast facility is an abstraction of the underlying network. The broadcast scheme is used for several reasons: control messages, initialization channels and messages, network fault messages, mailing messages etc. Messages are to be sent to all nodes and there must be minimum assumption about the type of the message;

iii) multicast scheme, sawn as point to multipoint communication, permits a subset of the virtual nodes to receive messages of the same type. The multicast service is an abstraction of a multi-channel communication. As in point to point scheme, message types might exhibit a complex user defined structure;

iv) typed broadcast scheme allows the exchange of messages of the same type but using a broadcast scheme: the message is spread to all virtual nodes accepting messages of that type. The sender needs not to have any knowledge about the receiver nodes.

2.2 Sets of destinations

In the following we give formally the properties of the communication schemes above using some of the concepts presented in [Schneider 82].

First, we give for each communication scheme the set of nodes that receive a certain message. The sets are denoted by M_x, where x is in {p,b,m,t} and denotes the communication scheme, with the parameters needed for the definition of the set.

Nodes are seen as sets of input ports; the nodes have two possible states: **active** and **failed**. A node is **active** when it has a normal activity in the network. A node is **failed** if this interrupts its execution and all belonging information to the node are lost. A predicate FAILED is defined to say whether a node is active or not.

Notice that in the following we are concerned only about the very communication, without boring of the preliminary actions needed as the connection between input and output ports (mentioned previously in the basic characteristics). Such connection merely introduces a level of indirectness in the denotation of an input port of another node.

i) Point to Point

$$M_p(i,m,p) = \{ j \mid p \in j \wedge \neg \text{FAILED}(j) \}$$

where:

i = name of the sender node
m = message
p = name of the input port to which the message is sent.

Note that the cardinality of this set is one.

ii) Broadcast

$$M_b(i,m) = \{ j \mid j \in RV - \{ i \} \wedge \neg \text{FAILED}(j) \}$$

and

i = name of the sender node
m = message
RV = set of all nodes in the virtual network.

iii) Multicast

A multicast message is received by the nodes on which the input ports specified in the message reside.
Now we can define M_m (i,m,P) as:

$$M_m(i,m,P) = \{ j \mid j \in RV - \{ i \} \wedge \neg FAILED(j) \wedge \exists p \in P, p \in j \}$$

where

i = name of the sender node
m = message
P = set of ports to which the message is sent.

iv) Typed Broadcast

To define this communication scheme we need a function T that returns the type associated to a port.
We can define M_t (i,m,t) as:

$$M_t(i,m,t) = \{ j \mid j \in RV - \{ i \} \wedge \neg FAILED(j) \wedge \exists p \in j, T(p) = t \}$$

where

i = name of the sender node
m = message
t = type of the message

2.3 Reliability Properties

In a distributed system for each communication scheme it is important to define which is the level of the reliability of the communication.

In general, we can observe three different levels of reliability in communication:

i) reliable communication;
ii) reliable communication with acknowledgement;
iii) unreliable communication.

i) Each active node can always communicate, directly or indirectly with every other active node;

ii) it is the same of i) but an acknowledgement message is also sent from the receiver to the sender, to guarantee the sender itself that the message has been received;

iii) there is no guarantee that a message is received by an active node.

The reliability properties can be given formally by temporal logic formulas [Lamport 83].

Such formulas use the predicates: sent, which is true if a node has sent a message; received, which is true if a node has received a message; ack which is true if a node has been acknowledged a message by another node.

The formulas are given in a generic form, good for any communication scheme; in the formulas we use m_x to denote the message of the communication scheme x (plus some possible information on the destination of the message), and M_x is one of the sets defined previously.

i) $\text{sent}(i,m_x) \supset \forall\, j \in M_x\,(i,m_x) : \Diamond\, \text{received}(j,m_x)$

$\text{received}(j,m_x) \supset \exists\, i: \text{sent}(i,m_x) \wedge j \in M_x\,(i,m)$

The first formula means that, when sent, a message is eventually received by the destinations; the second means that if a message has been received, it has been sent by a sender.

ii) $\text{sent}(i,m_x) \supset \forall\, j \in M_x\,(i,m_x) : \Diamond\, \text{received}(j,m_x)$

$\text{received}(j,m_x) \supset \exists\, i: \text{sent}(i,m_x) \wedge j \in M_x\,(i,m)$

$\forall\, j \in M_x\,(i,m_x) : \text{received}(j,m_x) \supset \Diamond\, (\text{ack}(i,j,m_x) \wedge \neg\, \text{FAILED}(i))$

The first and second formulas are as before; the third one means

that,if a message has been received, its sender will be eventually acknowledged, if it is not failed in the meanwhile.

iii) received $(j,m_x) \supset \exists\, i$: sent $(i,m_x) \wedge j \in M_x\,(i,m)$

In the case of unreliable communication, there is no guarantee that a message is received by the destination: so we can only say that, if a message has been received, it has been sent by some node.

Of the three properties listed before, any can apply to any of the communication schemes; however, we want to underline that some are more natural for some of the communication schemes. In particular, point-to-point scheme, being a direct communication between nodes, should have the reliable communication property, with or without acknowledge, while broadcast, being closer to the underlying phisic network, should have the reliable communication property (without acknowledge) or the unreliable communication property. The same is for typed broadcast: in fact, as we can see from the expression of the reliability properties, for the acknowledgement a knowledge of all the destination nodes is needed at the sender node, and this is not always achievable for broadcast and typed broadcast.

2.4 Use of the Properties

The definition of the properties given before can be used in several ways to aid the design and implementation of the communication mechanism. They can be useful in defining the primitives that provide the communication facilities, expecially for deriving their parameters, and to derive the information that need to be exchanged among nodes.

Moreover, they can be used to verify some implementation; the following is an example in which a multicast communication mechanism, with acknowledge, is implemented using the low-level communication facilities offered by an (Ethernet-like) broadcast mechanisms.

Suppose that the low-level mechanism obeys the following properties (broadcast, reliable communication without acknowledge):

$M_e(i,m) = \{ j \mid j \in RV - \{i\} \wedge \neg FAILED(j) \}$

$sent_e(i,m) \supset \Diamond (received_e(j,m) \wedge j \in M_e(i,m))$

$received_e(j,m) \supset \exists i: sent_e(i,m) \wedge j \in M_e(i,m)$

Now, to implement a multicast with acknowledge, we need to give a sending algorithm, a receiving algorithm and a message structure based on the low-level facilities.

The message structure needed at the low-level to transmit a multicast message m is the triple <P,m,i> where P is the list of destination ports and i is the sender node.

The receiving algorithm employs a low-level receiving, a test that a port in the destination list is existing on that node, and the sending of an acknowledge message; the algorithm is given in terms of relations among predicates:

$received_m(j,m) = \exists i \exists P: received_e(j,\langle P,m,i\rangle) \wedge \exists p \in P, p \in j$

$received_e(j,\langle P,m,i\rangle) \supset \Diamond sent_e(j,\langle ack,i,j\rangle)$

The sending algorithm is a simple sending, plus the possibility to be acknowledged by the destination nodes:

$sent_m(i,P,m) = sent_e(i,\langle P,m,i\rangle)$

$ack_m(i,j,P,m) = j \in M_m(i,P,m) \wedge received_e(i, \langle ack,i,j\rangle)$.

What is to show is that the predicates $received_m$, $sent_m$, and ack_m so defined give a correct implementation of a multicast with acknowledge, i.e. they respect the following properties:

(1) $sent_m(i,P,m) \supset \forall j \in M_m(i,P,m) : \Diamond received_m(j,m)$

(2) $received_m(j,m) \supset \exists i \exists P: sent_m(i,P,m) \wedge j \in M_m(i,P,m)$

(3) $\forall j \in M_m(i,P,m) : received_m(j,m) \supset \Diamond ack_m(i,j,P,m) \wedge \neg FAILED(i)$.

The proof is the following:

$sent_m\ (i,P,m) =$

$= sent_e\ (i,\langle P,m,i\rangle)$

-- for the sending algorithm

$\supset \forall\ j \in M_e\ (i,\langle P,m,i\rangle) : \Diamond\ received_e\ (j,\langle P,m,i\rangle)$

-- for the broadcast property

$\supset \forall\ j \in M_m\ (i,P,m) : \Diamond\ received_m\ (j,m)$

-- for the receiving algorithm and

-- since $M_m\ (i,P,m) \subset M_e\ (i,\langle P,m,i\rangle)$

Hence, (1) is proved.

$received_m\ (j,m)$

$= \exists\ i\ \exists\ P$: $received_e\ (j,\langle P,m,i\rangle) \wedge \exists\ p \in P, p \in j$

-- for the receiving algorithm

$\supset \exists\ i\ \exists\ P$: $sent_e\ (i,\langle P,m,i\rangle) \wedge \exists\ p \in P, p \in j$

-- for the broadcast property;

$= \exists\ i\ \exists\ P$: $sent_m\ (i,P,m) \wedge j \in M_m\ (i,P,m)$

-- for the sending algorithm and

-- for the definition of $M_m\ (i,P,m)$

Hence, (2) is proved.

$\forall\ j \in M_m\ (i,P,m) : \Diamond\ received_m\ (j,m)$

$\supset \forall\ j \in M_m\ (i,P,m) : \Diamond\ sent_e(j,\langle ack,i,j\rangle)$

-- for the receiving algorithm

$\supset \forall j \in M_m\ (i,P,m)\ \forall k \in M_e(j,\langle ack,i,j\rangle) : \Diamond\ received_e(k,\langle ack,i,j\rangle)$

-- for the broadcast property

$\supset \forall\ j \in M_m\ (i,P,m) : \Diamond\ received_e\ (i,\langle ack,i,j\rangle) \wedge \neg\ FAILED\ (i)$

-- since i, if not failed, belongs to $M_e(j,\langle ack,i,j\rangle)$

$= \Diamond\ ack_m\ (i,j,P,m) \wedge \neg\ FAILED\ (i).$

-- for the sending algorithm

Hence, (3) is proved.

The three properties are verified and the implementation scheme given previously can be said to be correct.

Summarising, the method adopted is constituted by the following definitions:

a) properties of the underlying communication mechanism;
b) properties of the desired communication mechanisms;
c) message structure;
d) sending algorithm;
e) receiving algorithm.

Then, a verification is made that the algorithms effectively implement a communication scheme respecting the properties defined in b).

Note that this method can be applied for any type of communication mechanism and any other kind of low-level communication facilities (e.g. a ring).

3. SPECIFICATION DESIGN IN HLL

As we have said, a natural way to provide the user of a functionally distributed system with a unique view of the distributed system is through the use of a system high level language; this language should express features typical of an operating system, including the communication mechanisms themselves.

To integrate the design of the communication mechanisms in the framework of existing programming languages, we have, as a first step, to individuate the main components that have to be expressed in language terms.

Whichever the linguistic constructs or linguistic abstraction mechanisms are, the following functionalities, grouped in components, can be identified:

1) components for communication primitives: there will be components for input and output ports for each of the communication schemes. In particular, these components will contain: i) send/receive primitives, ii) creation/deletion of ports (for the communication schemes for which is needed), iii) connection of output ports with

remote input ports (if needed). Due to the use of typed ports (referring to the basic characteristics given previously), it can be the case that some or all of the primitives will be parametrized in some way to the type of the message.

2) components for the creation of unique port identifiers: as we have said, port identifiers should be unique, hence a network-wide unique name service must be provided.

Moreover, rules for mapping virtual nodes to linguistic constructs should be settled: this could involve both the use of further primitives, and the use of particular structures, or linguistic constructs, in defining the distributed programs.

Obviously, the richer the language is from the point of view of abstraction mechanisms and modularization concepts, the easier will be to integrate in it the communication mechanisms.

Let us take as an example the definition of the Inter-Node communication mechanisms of the Cnet project [Fantechi 85]. The language chosen was Ada [Ada 83], and the communication mechanism had to be expressed at the Ada level by means of the Ada abstraction mechanism. The components mentioned before has been mapped in the following way in Ada terms:

1) the communication schemes provided are the point to point and the broadcast one: three Ada packages has been defined to group the broadcast primitives, the point to point input facilities and the point to point output facilities respectively. The broadcast package provides send and receive procedures for an unstructured message. The input port package provides procedures to create/delete input ports (making use of the name serving facilities), and to receive a message of any Ada type. The output port package provides procedures to create/delete ouput ports, to connect them to remote output ports (of the same type) and to send a message of any Ada type. The parametrization on the type of the message has been achieved by the use of generic packages.

2) the name serving facilities has been realized by a structured set of Ada packages exporting functions that :

i) give a "unique-on-the-node" name and
ii) compose this with a node name to give a "unique-on-the-net" name.

The mapping of the nodes in Ada entities has been achieved by means of a philosophy of configuration of the virtual network that uses constraints on the structure of the Ada program (in particular, the constraints are on the program library that constitute the Ada program itself).

Such philosophy, fully described in [Inverardi 85], has impact on both the two previous points: it imposes a further level of generic packages surrounding the communication packages and impose a more complex structure (here also with much use of generics) to the name serving packages.

4 CONCLUSIONS

Starting from the principles behind the definition of the Cnet inter-node communication mechanisms, more general properties have been developed in a formal style for four communication schemes: point to point, broadcast, multicast, typed broadcast.

It has been presented and discussed how these properties can constitute an aid to the design and implementation of the communication mechanisms.

Then other general issues on the use of an high level language in the definition of the communication mechanisms have been discussed.

The work can be considered as a first step in the direction of identifying a uniform framework in which the design of communication mechanisms in functionally distributed systems can be carried out. Future work will be developed in this direction.

REFERENCES

[Ada 83] "Reference Manual for the Ada Programming Language", Military Standard, United States Department of Defence, ANSI/MIL-STD-18-15A, January 1983.

[Cnet 85] " Distributed Systems on Local Networks", Cnet Project, ETS, Pisa June 1985.

[Fantechi 85] A. Fantechi, P. Inverardi, N. Lijtmaer, "Using High Level Languages for Computer Network Communication": a Case Study in Ada", to appear in Software Practice and Experience.

[Inverardi 85] P.Inverardi, F. Mazzanti, C. Montangero, " The Use of Ada in the Design of Distributed Systems", ADA in use - Proceedings of the Ada International Conference 1985, Cambridge University Press, 85-96, May 1985.

[Liskov 85] B. Liskov, "The Argus Language and System" in Distributed Systems - Methods and Tools for Specification - An Advanced Course, Lecture Notes on Computer Science, vol. 190, Springer-Verlag New York 1985.

[Lamport 83] L. Lamport, "What Good is Temporal Logic?", Information Processing 83, R. E. A. Mason. ed., North Holland, Amsterdam 1983.

[Schneider 82] F.B. Schneider, D. Gries, R.D.Schlichting, "Fast Reliable Broadcast", Cornell University Technical Report TR 82-519,September 1982.