Targeting Ada95/DSA for Distributed Simulation of Multi-protocol Communication Networks

Dhavy Gantsou
University of Valenciennes
Lamih/ROI

F - 59313 Valenciennes Cedex 9
Phone : +33 327 511 944

Dhavy.gantsou@univ-valenciennes.fr

ABSTRACT

This paper describes the use of Ada95/DSA as middleware for modeling network's entities into distributed objects which can be used for both distributed execution and simulation without transformation of the model. First, general issues of simulation of network's entities are briefly presented. Then, the paper shows how DSA features are used to abstract these entities in a manner as they can be executed or simulated on a cluster of workstations. The DSA based model of distributed simulation is built on a combination of two kinds of classes: the first one support general purpose of distributed objects computing and the second deal with the development of distributed network elements. To illustrate the feasibility of this approach, a partial implementation of classes that have been used to model routers running a subset of border gateway protocol (BGP) is presented.

Keywords

Distributed system annex, distributed objects engineering, distributed networking protocols, distributed simulation.

1. INTRODUCTION

Conceptually a communication network like Internet is a collection of interconnected autonomous systems (AS), each of them being a group of network elements such as routers, switches, hosts and transmission media under a single administration. Those elements share information and policies, in order to provide quality of service (QoS) expected by users. Achieving this goal is extremely difficult because no single network technology can suffice for all uses.

Part of these objective can be achieved by enhancements of standard data networks protocols [15]. Another part involves multiprotocol technology (e.g the use of existing protocols in conjunction with constraint-based routing protocols from the switch technology such as ATM PNNI [5] and MPLS [16]). In both cases, given the network topology and the complexity of constraint-based routing protocols, parallel simulation (running simulation on shared memory multiprocessor platforms) is used to best study issues related to Qos constraints.

It is acknowledged that distributed computing platforms can provide same computing performances like parallel platforms. As parallel architecture are not widely available, an alternative is to use distributed memory architectures for running existing simulation frameworks. These software DaSSF [4] and JSSF [4]) were originally dedicated to be executed on parallel environment. With respect to parallel programming style, they are sequential. Thus, issues of distributed processing which are inherent to functionalities being simulated such as routing protocols have not been considered and incorporated in the model. Consequently the execution of these models on distributed memory environment cannot provides the expected reliability and efficiency, due to a wide range of problems. How to solve most of them is beyond the scope of this paper whose goal is to show how Ada95 [6] and its distributed object oriented middleware (DSA [6], [9]) can be used to model network entities and then run these models on cluster of workstations.

To achieve the objective of distributed simulation, following constraints may be satisfied:

1) incorporating distributed processing issues in the description of the model .
2) Designing software such that it can be executed on distributed memory architectures. This one requires to choose the suitable middleware.

The rest of this paper is structured as follows. Section 2 synopsis general issues of distributed simulation of netwok applications. Section 3 shows how Ada95/DSA features can be used to describe models and to develop distributed software that can be either executed or simulated on cluster of workstations. Section 4 presents relationships to other works, while conclusions are presented in section 5.

2. OVERVIEW OF ISSUES

A communication network connects processing entities that may be distributed across disparate physical locations. These entities interact to realize both networking functions and application processing. Network functions are separated in layers, each of them serving specific function. Each layer uses its own layer protocol to communicate with its peer layer in the other system. These hiarchical structure is used to model network architecture such as the five-layer TCP/IP protocol stack[17].

Tasks performed by a protocol fall into two categories : those dealing with information flow and those having to maintain protocol-related state information. The protocol is responsible for moving data to and from end user (eg host, intermediary system or end user program), and for information flow and protocol state. It also adds information to outgoing data, and interpretes header from incoming data.

Let us use the simple network shown in Figure 1 for illustration.

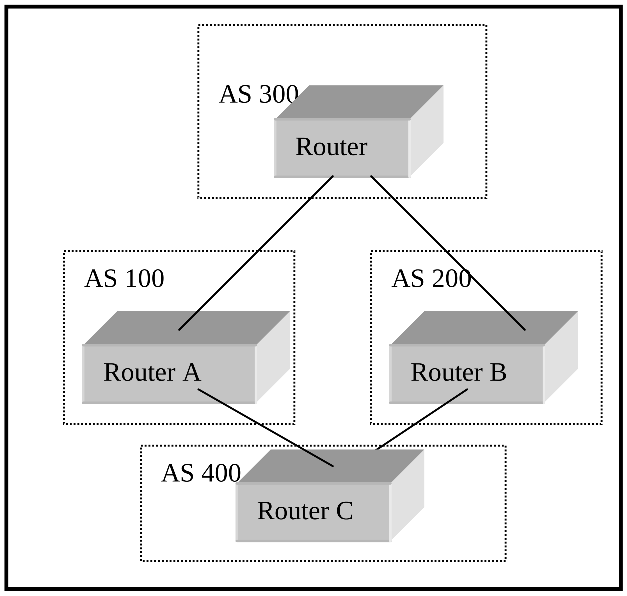

Figure 1. A simple exemple of network

In figure 1, the routers are the processing entities. A router has three fundamental functionalities. The first is to compute the best path that a packet should take for reaching its destination. The second role is to actually forward packets received on an input interface to the appropriate interface for transmission across the network. The third major router job is to temporarily store packets to absorb the bursts and temporary congestion.

The routers composing the network on Figure 1 run the BGP protocol (BGP speaker). BGP involves several functionalities. One of them is the decision algorithm [18].

When a BGP speaker receives updates from multiple ASs that describe different paths to the same destination, it must run the decision algorithm that enables to chose the single best path for reaching that destination. Once chosen, BGP propagates the best path to its neighbors.

A protocol maintains state information such as router identification, the topology data base or the link-state information in case of the open shortest path first (OSPF) [14]. One of the aspects inherent to state maintenance is the necessity for :

- controling access to global information when a protocol involves active entities accessing common data simultaneously.
- coordinating execution of distributed entities..

These requirements happen for example, when OSPF routers on a given network interact with the designated router (DR) acting as central point of contact for link-state information exchange figure 2.

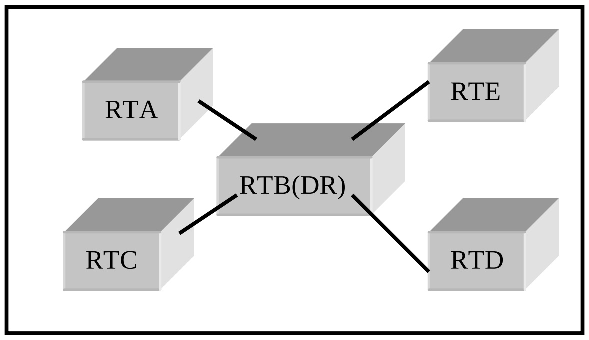

Figure2. Router acting as shared memory

In figure 2, routers are distributed across network, they do not share a common physical address space, but have to access to data located one of them (DR). Accessing the link-state information data base on the DR requires support that enable both to simulate shared memory (virtual shared memory) and to provide robust sharing of ressources. For ressource sharing, synchronization may be undertaken under more severe conditions [1] than in parallel computing . Beyond the need to manage the link-state synchronization, this situation requires to coordinate the execution of routers and to solve problems resulting on the inter-router communication..

Another aspect of state maintenance is to keep track of time (through timer) and to ensure that events happen in timely manner. This is the case with the hello and dead intervals in establishing adjacencies in OSPF protocol. The hello interval specifies the frequency, in seconds, that a router sends hellos. The dead interval is the time in seconds that a router waits to hear from a neighbor before declaring the neighbour router. These timers must be the same on neighboring routers. Timeliness is essential to correctness of such a protocols.

Network applications must operate under more-severe constraints than conventional software systems and yet perform reliably for long periods of time. Some of these constraints are schedulability, predictability, robustness, and deadlines. As such, they are considered as real-time systems

One of the requirements in the description of network entities is to have a modeling language which permits to abstract the structural and behavioral aspects of these entities. Real-time unified modeling language (UML) [2], [3] is an example of such languages. Although UML does not yet provide all tools for handling distributed components , it can be used to model the system.

A network application is a collection of objects that communicate via event exchange . Assuming these objects have been identified, the next step in the design may consist in:

- decomposing the system into components and describing their behavior.
- specifying all interactions between components, implicit or explicit.
- ensuring that components are consistent with each other. Each component of the model receives input from other interacting components, and provides them with outputs. Mode and type of sender’s parameters must match those of the receiver.
- providing for efficient execution of expressed models. This depends on how DSA will support issues that have been mentioned above.

3. MODELING APPROACH

3.1 Dsa features at a glance

DSA is part of the Ada95 ISO standard[4]. It aims at providing a framework for programming distributed systems within the Ada language, while preserving strong typing properties.

The distributed application model of DSA is more general. It provides RPC and remote execution of methods (facility and references to remote operations). It also allows the definition of a shared data space, through the abstraction of the shared passive package. DSA provides the Remote Call Interface (RCI) categorization pragma that makes operations (procedures and functions) of a package available for remote procedure calls. Remote Types (RT) package can define distributed objects and methods whose invocation may be synchronous or asynchronous. RT units allow non-remote access type, provided there are marshalling subprograms. DSA enables global data to be shared between active partitions, even they are declared in a Shared_Passive library unit.

For security and Qos purposes, DSA provides channels. A channel is connection between two communicating partitions. Having defined a channel, a designer may apply compression, and encryption of data exchanged through the channel.

DSA provides an integrated approach for application development and test: going from a non distributed application, which is easy to test and to debug, to a full distributed only requires the addition of one categorization pragma to each package that defines remote objects or subprograms.

3.2 Representing model with DSA features

To abstract active components, while ensuring their distributed execution, we use DSA remote_call_interface**,** remote_types and shared_passive categorized partition as building block. Each of these partitions communicates with its surrounding environment. Communication may be supplied by :

- RPC for RCI partitions.
- Remote objects invocation for RT partitions.

RPC and remote objects invocations may be synchronous or asynchronous. Communication involves transmission of events, modeled by parameters of mode :

- access when exchanging data of class-wide or complex type.

For illustration, the exemples belows are parts of classes that can be used to model BGP routers runing the decision algorithm Figure 1.

Computing the decision algorithm enables a router - for example Router C from AS400 - having two routes for reaching the router D from AS300 to decide which route to use. The decision is based on the values of the attributes (such as next hop, administrative weights, local preference, the origin of the route and the path length) that the update contains and other BGP-configurable factors.

```
package Router is
  pragma Pure;
  type Router_Class is abstract tagged limited private;
  type Community_Type is (no_export,no_advertise, internet);
  type Exchanged_Attributes is
    record
      As_Path      : Positive;
      Origin       : Common.String_Host;
      Next_Hop     : Common.String_Host;
      Local_Preference  : Natural;
      community        : Community_Type;
    end record;
  procedure send
           (Data    : access Router_Class;
           Update     : in Exchanged_Attributes;
        Src_Hostname : in Common.String_Host) is abstract;
  procedure receive
                   (Data : access Router_Class;
                 Update : in Exchanged_Attributes) is abstract;
Private
  type Router_Class is abstract tagged limited null record;
end Router;
```

Figure 3. A router abstraction

This abstraction can then be inherited by each distributed object performing the router basic functions as shown below

```
with Common,
     Object_Adapter,
     Interface;
package Bgp_Isp_Router is
 type Bgp_Isp_Router_Class is new Interface.Router_Class with
    record
          hostname    : Common.String_Host;
          neighboors  : Object_Adapter.Neighboor_Type;
    end record;

 procedure send (Data    : access Bgp_Isp_Router_Class;
                           Update : in Interface.Exchanged_Attributes;
                           Src_Hostname : in Common.String_Host);
 procedure receive (Data : access Bgp_Isp_Router_Class;
                            Update : in Interface.Exchanged_Attributes);
end Bgp_Isp_Router;
```

Figure 4. Basic Router functions

The operation Send models the interactions of one router with others. Inputs and outputs are naturally expressed using Ada95/DSA features. This is one of the salient differences to modeling approaches of abstraction based on TeD [3]. TeD is an object-oriented language for modeling telecommunication networks. Simulation frameworks using TeD model (eg DaSSF and JavaSSF) do not provide direct expression of real-world aspects such as inputs and/or outputs, simultaneous execution of active objects etc. To overcome these issues, the frameworks provide an interface defining classes enabling a more realistic abstraction. The exemples below show the interface enabling to abstract input in java and in C++.

```
public interface inChannel {
public Entity owner();
public Event[] activeEvents();
 public outChannel[]
mappedto();
                    }
```

Figure 5. public interface for modeling input in Java [1]

```
class inChannel {
public:
Entity* owner();
Event** activeEvents() // null-terminated
outChannel** mappedto();   // null-terminated
};
```

Figure 6.public interface for modeling input C++ [1]

Besides operations expressing interactions with the surrounding environment, an active object may include threads that model concurrent execution of arriving events or express event-driven or arrival-driven behavior. This goal is achieved through the use of the Ada.Real_Time annex features in conjunction with tasking and timing related statements such *delay t, delay until t, and the asynchronous transfer of control.*

Another issue to address in our approach is to model the sharing of resources. Sharing resources occurs at two levels of interactions :

- • among distributed objects
- • among concurrent objects executing simultaneously on a processor,

Enabling distributed objects to efficiently access common entities is a fundamental decision that has great impact on the performance of the system. The simplest, way to overcome all issues inherent to this aspect is to provide a high level abstraction based on the Shared_Passive partition which is DSA means to provide a virtual shared address space.

Controlling access to data inside an active object does not require an abstraction at the description level. Either for distributed or for concurrent interactions, data access control is handled at the implementation level. In both case the simplest, albeit robust way to achieve this goal is to use a protected type. A protected type is an Ada95 concept which enables the implementation of synchronization mechanisms that scales smoothly from a single processor to a multiprocessor. For distributed objects, shared ressources must be part of a partition Shared_Passive partition. Controlling access consists in implementing an entryless protected object in the shared_passive partition.

4. RELATED WORKS

This work is distinguished by its focus on using an object-oriented middleware built on top of a programming langage to deal with distributed modeling and execution of network protocols.

The question whether CORBA[12] or the implementation of the DSA (e.g Glade) for engineering distributed software has been considered in other contexts in [10], [11] and [19].

Despite its associated middleware (RMI [7]), Java has been used to implement sequential models of frameworks for parallel simulation of networks [4]. These models are based on the TeD [8] approach where the execution of generated models requires transformation of the software. This is contrary to our approach where, thanks the fact the simulation software is the implementation of the model, simulated software can be used as prototypes for testing and verifying the entities being implemented.

Unlike Corba and Java/RMI, DSA provides features such as the real-time annex, the shared_passive partition and the protected type that enable to efficiently address critical issues surrounding the development of real-time distributed software.

5. CONCLUSION

Like other middleware, DSA does not provide direct support for distributed simulation of network applications. Implementing distributed simulation software have shown the necessity to have a graphical user interface, and constructs that facilitate the engineering of distributed objects. From the point of view of language constructs, it is indispensable to have feature that support the IP multicast.

Although Glade-3.13p [9] does not provide all features yet, some results of our investigations have been already used for implementing distributed models simulating parts of OSPF and BGP4 protocols. We ran our models successfully on a solaris 2.7 running Sun Enterprise 5000, as well as on a 100M ethernet network of 15 Sun UltraSPARCs running solaris 2.6.

Using Glade to implement network applications is a great challenge that can help promoting Ada in this area and in education. The Ada community and particularly Glade practitioners can play a decisive role in the development of distributed applications.

6. REFERENCES

[1] Albert Y. H. Zomaya. Parallel & distributed computing handbook, McGraw-Hill Series on Computer Engineering, 1996.

[2] B. Powel Douglass. Doing Hard Time : Developing Real-Time Systems With UML, Objects, Frameworks, and Patterns'', Addison Wesley Longman, Inc, 1999.

[3] Grady Booch, James Rumbaugh, Ivar Jacobson. The Unified Modeling Language. Addison-Wesley ISBN 0-201-57168-4.

[4] **http://www.ssfnet.org**

[5] http://www-comm.itsi.disa.mil/atmf/pnni.html

[6] ISO Information Technology - Programming Language- Ada ISO /IEC/ANSI 8652 :1995.

[7] Java Remote Method Invocation Specification. JavaSoft, revision 1.50, JDK1.2

[8] Kalyan Perumalla, Richard Fujimoto, Andrew Ogielski MetaTeD: A Meta Language for Modeling Telecommunication Networks . GIT-CC-96-32, Technical Report, College of Computing, Georgia Institute of Technology, 1997.

[9] Laurent Pautet, Samuel Tardieu. Glade User Guide May 2000 ftp://ftp.cs.nyu.edu

[10] Laurent Pautet, Thomas Quinot, Samuel Tardieu. Corba & DSA: Divorce or Marriage? In proceedings of Ada Europe'99. Santander,Spain, June 1999.

[11] Laurent Pautet, Thomas Quinot, Samuel Tardieu. Corba and Corba Services for DSA. In Proceedings of ACM SIGAda'99 conference. Redondo Beach, CA, USA, October 1999.

[12] Object Management Group (1998c). The Common Object Request Broker: Architecture and specification revision 2.2 492 Old Connecticut Path, Framingham, MA 01701, USA

[13] RFC1776 Border Gateway Protocol version 4

[14] RFC 2328 OSPF2

[15] RFC2676 QoS Routing mechanisms and OSPF extensions.

[16] RFC 3031 Multiprotocol Label Switching Architectures.

[17] Richard Stevens TCP/IP Illustrated. Volume 1 Addison-Wesley 1994.

[18] Using the Border Gateway Protocol for Interdomain Routing http://www.cisco.com/univerced/cc/td/doc/cisintwk/ics/icsbgp4.htm

[19] Yvon Kermarrec. Corba vs. Ada95 DSA A programmer’s view. In Proceedings of ACM SIGAda’99 conference. Redondo Beach, CA, USA, October 1999.