Targeting Ada95/DSA for Distributed Simulation of Multiprotocol Communication Networks

Dhavy Gantsou
University of Valenciennes
ISTV 2
59313 Valenciennes Cedex 09
France
E-mail : Dhavy.gantsou@univ-valenciennes.fr

Phone (+33) 327 511 944 , Fax (+33) 327 511 800

Abstract
The last years have seen an increasing, albeit restricted simulation of large-scale networks on shared memory parallel platforms. As the complexity of communication protocols and the network topology increase, so does the need for high performance simulation techniques. An impediment to the widespread use of these simulations is the high coast of parallel platform. To satisfy these requirements, network designers or researchers have to use cost-effective networks of commodity processors (workstations or Pc) as computing environment. Towards this end, the solution may be to transfert implementations of parallel simulation techniques to distributed architectures ; this leads inevitably to unreliable and inefficient software. One reason, though certainly not the only one, that such risk exists is that real-time and distributed issues inherent to networks have not been incorporated into model describing actual network and its components. This paper is concerned with the use of Ada95/DSA as building block for abstracting network architectures and protocols into distributed simulation models. Besides the fact that we use DSA features to naturally expresses the distributed characteristics of networks architectures and protocols, we show that generated models can be efficiently executed on distributed environments, while consistency is preserved by strong typing proprties of Ada.

1 Introduction

Conceptualy a communication network like Internet is a collection of interconnected autonomous systems (AS), each of them being a group of network elements - routers, switches, hosts and transmission media - under a single administration. These elements share information and policies, in order to provide users expected quality of service (QoS). Achieving this goal is becoming extremely difficult with the emergence of new paradigms, new users and new applications. Many of these new paradigms and applications require network services which are not ideally matched by those provided by traditional protocols. To aim this target, it is necessary either to design and implement new protocols, or to extend existing ones. Designing, implementing, and deploying new protocols generally require to modify millions of end systems. This can lead to problems such as routing protocols scalability and stability and packet loss which in turn requires to understand and to predict network behaviours. Parallel simulation – performed on shared memory machines – is the suitable tool for this end.

Implementations of parallel simulators are not commonplace. DaSSF[1] and JSSF [1] are the few examples. It is widely acknowledged that distributed simulation can provide same performances as parallel simulation. As coast-effective distributed computing platforms are widely available, the tentation to porting software intended for simulation on parallel architectures to distributed environments is intense.

However, as efficient as parallel simulators are, they do not execute efficiently on distributed shared memory architectures due to fundamental problems of distributed computing. In a distributed system architectures, interconnected processors do not share a common physical address space,

consequently shared data cannot be accessed through simple load and store instructions. On the other hand, the time needed to access data depends on their location. Accessing data on remote node (remote data) needs more time than accessing local data. Moreover, in a distributed environment, synchronization may be undertaken under more severe conditions [2] than in parallel computing . Other problems arise from the nature of simulation.

Distributed simulation can be defined as the process of using networked workstations or PC simultaneously for executing a single simulation, with the goal of reducing the total execution time. There are two major questions that network simulator designers have to answer :

- how to abstract network components into simulation models, and,
- how to express the model for providing distributed execution?

This paper describes our Ada95/DSA-based approach for answering these questions.

2 Modeling approach

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

We can achieve our goal through a modeling language which permits to abstract the structural and behavioural aspects of network architectures. Real-time unified modeling language (UML) [5], [6] is an example of such language that we will use. Assuming all functional objects have been identified, the focus must be on following aspects :

- naturally expressing the model. Since targeting distributed simulation, this implies to use abstraction permiting both:

1 – to decompose the system – into components (active or not) - and control its complexity ;

2 - a realistic description of the behaviour of active components.

This concerns two aspects : first one, the simultaneous execution of active components given the fact that they are to be distributed across interconnected processors on wich they have to be executed (distributed execution). The second is the fact that an active component may include subcomponents being executed concurrently.

- 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 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 issues such as access to shared ressources, synchronization, communication and coordination of executions are addressed inside and between active components.

2-1 Identifying objects

A communication network includes functional objects (physical and logical) wich interact to realize both networking functions and application processing. Networking applications 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. Each layer's protocol exchanges information, called protocol data units(PDU), between peer layers. A given layer uses a more specific name for its PDU.

The tasks performed by a protocol fall into two categories : those dealing with information flow and and those having to maintain protocal-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 header to outgoing data, and removes and interpretes header from incoming data . For example in the five-layer TCP/IP protocol stack[7], the transport layer communicates with the peer transport layer using segments.

The peer-layer protocol communication is achieved by using the services of the layer below it. The layer below any current layer provides its services to the current layer. Each lower-layer service takes upper-layer information as part of the lower-layer PDU it exchanges with its layer peer. Thus, the TCP segment becomes part of the network-layer packet (also called datagram) exchanged between IP peers. In turn, the IP packets must become part of the data-link PDU - frames- exchanged between directly connected devices. Ultimately these frames must become bits as the data is finally transmitted by physical-layer protocol using hardware.

A protocol maintains two kinds of state information : information specific to a particular session and global information. Session specific state includes current sequence numbers, data sent by a peer entity but not yet acknwoledged etc. Examples of global information include router identification - used to identify an router to the open shortest path first (OSPF) [8] - , topology data base, link-state information etc.

One of the aspects inherent to state maintenance is the necessity for :

- controling access to global information when a protocol involves active components executing simultaneously.
- coordinating execution of distributed active.

These requirements happen for example, when OSPF routers on a given network interact with the designated router (DR) [8] acting as central point of contact for link-state information exchange. Beyonds the need to manage the link-state synchronization, this situation requires coordinating the execution of routers.

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. This timers must be the same on neighboring routers. Timeliness is essential to correctness of such a protocol.

2–2 Model Abstraction

2-2-1 DSA features at glance [9]

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 (subprograms). 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 purpose, DSA provides channel. 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.

2-2-2 Objects abstractions using Ada95/DSA

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 synchrone or asynchrone. Communication involves transmission of event, modeled by parameters of mode :

- in, out, in out when exchanging data of elementary type.
- access when exchanging data of class-wide or complexe type.

To illustrate, the example below is a snippet of a model of a router.

```
with Common,  Interface;
package Router is
        ……..
type Current_Data is new Interface.Root_Class with
            record
                        ……..
            end record;
procedure Update(Source : access Current_Data ;
                    Update : Interface.Exchanged_Attributes);
            ……..
end Router ;
```

The operation Update models an interaction of a router with others. Inputs and outputs are naturally expressed using Ada95/DSA features. This is one of the salient differences to modeling approaches of TeD-based abstraction model[3]. TeD is an object-oriented language for modeling telecommunication networks. Simulation framaworks 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 issue these frameworks provide an interface defining classes enabling a more realistic abstraction. The example below shows the interface enabling to abstract input in java and in C++.

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

public interface for modeling input in Java [1]

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

public interface for modeling input C++ [1]

Besides operations expressing interactions with the surrounding environment, an active object may include threads modeling concurrent execution of arriving events or for expressing event-driven or arrival-driven behavior. This goal is achieved through the combined use of the Ada.Real_Time annex features, tasking timie 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 objects executing on interconnected processors (eg distributed objects).
- among objects executing simultaneously on a processor (eg concurrent objects),

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

```
with Annex_Data ;
use Annex_Data ;
package Shared_Entities is
pragma shared_passive;
     ……
type Class_Root  is abstract tagged limited private;
 type List_Template is array(Key_Typ range <>) of
                                        Link_State_Info;

type List_Template_Ref is access all List_Template ;
 function Retrieve (Link_State_Data  : access Class_Root ;
         My_ID :Msg_Typ) return Link_State_Info is abstract;
procedure Deposit(Target  : access Class_Root ;
                        Source  : List_Template) is abstract;
                  ……
private
       type Class_Root is abstract tagged limited null record;
end  Shared_Entities;
```

This abstraction can then be inherited by each distributed object having to access shared entities as shown below

```
with Shared_Entities, Annex_Data ;
use Shared_Entities, Annex_Data ;
 package Object_Modeled_Router is
  pragma Remote_Types;
type Current_List_Obj is new The_Class_Root with
        record
            Tab:  List_Template_Ref;
        end record;
    ……
function Retreive(Link_State_Data : access Current_List_Obj ;
                My_ID  : Msg_Typ) return Link_State_Info ;
procedure Deposit(Target  : access  Current_List_Obj ;
                                Source  : List_Template) ;
   ……
end Object_Modeled_Router;
```

Controling access to data inside an active object does not require an abstraction. Either for distributed interactions or for concurrent interactions, it 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 mechanism that scales smoothly from a single processor to a multiprocessor. For distributed objects, shared resources must be part of a partition Shared_Passive partition which is an DSA mean to provide a virtual shared address space. Controlling access consists in implementing an entryless protected object in the shared_passive partition.

3- Related works

DaSSF and JSSF are large-scale network modeling software. They are based on the TeD [3] approach making the description independent of simulation. Thus, the execution of expressed models is only possible through translation. Although they can run on distributed architectures, models from which they are constructed are targeted for simulation on parallel architectures. Their description does not naturally express the distributed environment, moreover fundamental issues inherent to distributed execution such as synchronization are not integrated in the model. The TeD model only supports transmitting events modeled by elementary types, certainly due to the semantics of the C++ used for its implementation.

4- Conclusion

No programming language provides direct support for abstracting and executing models of network applications, Ada95 does. However, it provides a distributed object-oriented middleware and a rich set of fundamental building blocks which allows to design, test, end efficiently execute the generated distributed network models.

Implementing network simulation software raises a wide range of issues. Some results of our investigations have been already used for implementing distributed models simulating parts of DHCP, OSPF and BGP4 protocols. Although Glade-3.13p [9] (the implementation of DSA) does not provide all features yet, we ran our models successfully on a solaris 2.7 running SunEnterprise

5000, as well as on a 100M ethernet network of 15 SunUltraSPARC 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 particulary Glade practitioners can play a decisive role in the development of network software, and particulary in distributed network simulation techniques. Contrary to C++ and Java which are commonly use, not for their efficiency , Ada95 /DSA has a rich set of built-in features which permit to tackle a wide range of issues surrounding the developpment of network applications.

References

[1] http ://www.ssfnet.org

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

[3] 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

[4] ISO Information Technology – Programming Language- Ada ISO /IEC/ANSI 8652 :1995

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

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

[7] TCP/IP W. Richard Stevens ''TCP/IP Illustred'', Volume 1
Addison-Wesley 1994

[8] OSPF2 RFC 2328

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