A DSA Model For Data Access in Self-Organizing Systems

Dhavy Gantsou
University of Valenciennes
F - 59313 Valenciennes Cedex 9
Phone : +33 327 511 944
Dhavy.gantsou@univ-valenciennes.fr

ABSTRACT

Data availability is an important issue for self-organizing systems, which include both Peer-to-Peer (P2P) systems and mobile ad hoc networks (MANETs). In P2P systems, the problem of data availability is solved by replicating data across the network. However, this approach wastes resources, and so is not appropriate for MANETs where resource frugality is essential. Unlike P2P systems, MANETs routing protocols require real-time features to cope with a highly dynamic environment, and efficient synchronization mechanisms to guarantee consistent updates of routing information. Unfortunately, due to the semantics of the languages that are most used in network programming, such mechanisms are extremely hard or even impossible to implement in efficient way. To build an application capable of meeting almost all of these requirements, while providing a framework capable of evolving over time, we propose an approach that is based on DSA.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]. C.2.4 [**Distributed Systems**]. D.1 [**Programming Techniques**]. D.3.3 [**Language Constructs and Features**]

General Terms

Languages, Design .

Keywords

Distributed System Annex , Distributed Synchronization, Real-time Data Access, Mobile Data Access, Self-organizing Systems.

1. INTRODUCTION

Self-organizing systems offer an alternative to traditional distributed systems in cases where central coordination or central databases, pre-existing fixed network infrastructure and centralized services are precluded. In self-organized systems, every node (peer) of the system acts as both client and server, and no node has a global view of the system. Given that nodes connect and disconnect from the network frequently, efficient data access mechanisms must be provided.

The need to provide efficient data access mechanisms in self-organizing systems is highly dependent on the targeted application domain. For instance, in peer-to-peer systems like Gnutella [1], Freenet [2], and Chord [3], which are the most widely used implementations of self-organizing systems, this issue is not of primary importance. The primary goal is to guarantee both efficient searches and permanent data availability. To this end, the required data are replicated across the network. Replication, however, wastes resources that are critically needed by mobile ad hoc networks[4], and so is not suitable in MANETs environments. Moreover, MANETs involve protocols whose behavior is highly dependent both on real-time access to data and the data's coherency. For instance, updating the link state databases (LSD) in optimized link state routing (OLSR) protocols entails updating the LSD of each individual node, which means immediately propagating the update to each of the node's neighbors in order to assure consistency. Situations such as these require real-time supports and synchronization mechanisms that are better able to cope with the highly dynamic MANETs environment.

From the design perspective, this implies representing the overall system, its component parts, the behavior of these components, and their interactions. Such challenges are the reality of critical self-organizing systems like MANETs. The details of how we created a DSA based prototype that allowed us to test our approach are described in this paper

The rest of the paper is organized as follows: section 2 presents the conceptual foundations of the system, the issues raised by each of these concepts, and the way these issues were addressed. Section 3 describes the use of the proposed strategy for prototyping a subset of the OLSR protocol that involves considerable interaction and, consequently, much related data access. Section 4 presents the Ada implementation of our approach. Section 5 concludes the paper.

2. CONCEPTUAL FOUNDATIONS

As stated above, the architecture of the self-organizing system we propose aims to favor real-time access and reliable data access in constantly evolving environments.

To achieve this goal, we must consider these two characteristics within the several dimensions that generally must be taken into account when designing distributed systems:

- Distribution
- Communication
- Centralized or decentralized control

Distribution is crucial for attaining reliability. It consists of splitting a software component into sub- components, and subsequently spreading it over the system. Our architecture must, then, be implemented as a network of distributed objects performing the functions of the nodes.

Having decided to organize the system as a network of distributed objects, the next issue in designing our system involves ensuring real-time data access and consistent data management. Real-time access to data requires both low communication latencies and efficient synchronization mechanisms. The first goal can be achieved by implementing the system in such a way that interactions rely on asynchronous communications. However, there are cases that cannot make effective use of asynchronous communication. In such cases, third-party involvement must be eliminated in the communication between two nodes by using a fully decentralized design approach.

3. THE ARCHITECTURE MODEL

Our design methodology was guided by the need for software that would allow data access and distribution to be represented under accurate conditions in self-organizing systems, such as MANETs routing protocols. We needed a realistic model that would be able to express the behavior of system components and the relationships among them.

We focused on the distributed access to mobile data involved in the computation of a multi-point relay (MPR) [5]. MPR is a key concept in the MANETs optimized link state routing protocol (OLSR)[6] designed at INRIA [7]. MPRs are nodes selected by their counterparts to forward the information needed for route calculation; this process is called link state advertisement (LSA). An important element of OLSR is the absence of a dedicated node for the LSA database because this database is truly distributed over the nodes. Given that these nodes are mobile, a fairly complex system of self-organizing supports must be provided to maintain the MPR neighborhood and to allow mobile nodes to access the LSA databases maintained by other nodes.

Since nodes are highly mobile, an MPR can leave the network without warning, making it necessary for all neighboring nodes to quickly select another MPR. To this end, nodes must communicate periodically with their neighbors in order to detecting topology changes and begin the timely selection of a new MPR.

4. SOFTWARE IMPLEMENTATION

This section provides a brief description of the various kinds of software components used in our software model. The architecture is composed of two major kinds of components. The first one implements classes that are used to create objects that will perform application-specific functions. The salient classes include :

- the class ***All_Node***: ***All_Node*** is a virtual class acting as base class that specifies the interface of the distributed objects implementing a MANET node. The code fragment below shows the specification of the package that implements the class.

```
      …..
 with Shared_Entities ;
 Package All_Node is
     Pragma Remote_Types;
    Type All_Node is abstract tagged limited private ;
                    ...
 end All_Node ;
```

All_Node defines both a tagged type and the series of abstract methods that will be inherited by the objects (***Node_Obj***) called on to implement the functions determining the node's behavior. Examples of these functions include, but are not limited to, neighbor address retrieval and sending/receiving messages. The remote type categorization was chosen to ensure that derived objects act as distributed objects, which amounts to implementing the distribution concept described in section 2.

- the class ***Node_Obj*: *Node_Obj*** is designed so that in addition to methods inherited from ***All_Node***, it can also be used to implement several application-specific functions. Like ***All_Node*** from which it is derived, ***Node_Obj*** has the remote type categorization, as shown by the code fragment below :

```
       .. .
with All_Node ;
Package Node_Obj is
       Pragma Remote_Types;
        Type LS_Data_Base is new All_Node.All_Node
        with private ;
                      ...
        private
          Type LS_Data_Base is new All_Node.All_Node
           with
                Record
                         -- OLSR-specific database
                end record ;
  end Node_Obj ;
```

Besides methods inherited from the base class, each node performs several protocol-specific functions, some of which require time management and concurrent data access:

- time management: Like other link state protocols, OLSR uses timers extensively for a variety of reasons. These include, but are not limited to, ensuring the reliable delivery of packets and LSAs to neighbors via periodic transmissions, the continued retransmission of database description packets to neighbors pending acknowledgement of reception, the rate-limiting of certain functions in order to bound resource demands, and the transition of neighbor states in the absence of a response. Timers require the implementation of fairly complex supports. C and C++, the *de facto* programming languages for network designers, do not provide time services. To overcome this lack, several time services have been proposed [8], [9]. However, these implementations have two major drawbacks in addition to their complexity and their inefficiency. They are operating-system dependent, and as such, are not portable. More importantly, they don't provide monotonic time. Ada, on the other hand, provides constructs that easily meet all these requirements.

→ concurrent data access: Many functions performed in OLSR require the synchronization of data access in a fully distributed environment. As shown in the following code fragment, we used the protected object feature to implement the necessary operations.

```
….
Protected Synchro_Neighborhood_Update ;
Procedure Write_LS_Table(Nodes : Shared_Entities.LS_List) ;
Function  Retrieve_LS_Table return  Shared_Entities.LS_List ;
Private
     ...
end Synchro_Neighborhood_Update ;
```

For information exchanges with its neighbors, an instance of ***Node_Obj*** uses the packet created by another object, called ***Packet_Processor.*** ***Packet_Processor*** uses the data provided by other objects to create these packets. The following code shows how the abstract tagged type ***Message,*** declared in ***Shared_Entities,*** is derived by ***Packet_Processor*** to create a protocol specific packet:

```
...
With Shared_Entities ;
Package Packet_Processor is
Pragma remote types ;
     Type OLSR_Packet is new Shared_Entities.Root_Message
     with private ;
                    ...
     private
        Type OLSR_Packet is new Shared_Entities.Root_Message
        with
              Record
                ... -- Application specific data
              end record ;
end Packet_Processor ;
```

Like ***Node_Obj,*** ***Packet_Processor*** is also a remote types categorized package. It implements functions that can be used to implement packet creation in a given MANETs routing protocol. This is accomplished through derivation of the tagged type ***Root_Message***, provided in the ***Shared_Entities*** package, which encapsulates the entity declarations common to MANETs protocol software.

Our application also includes software components that are not directly concerned with routing. One of them is the proxy. The proxy's responsibilities include registering joining nodes, giving them the IP address of already-connected nodes, as well as creating and maintaining the object references.

A node joins the network by contacting the proxy, which gives that node the IP address of one or more already-connected nodes. Each object joining the network is thus registered. Once registered, the proxy creates that object's reference and extracts the required data about the object from its neighboring nodes. These data are then stored in the remote reference list used for tracking registered objects and for maintaining the neighborhood database.

```
…
with Shared_Entities, All_Node ;
Package Proxy is
Pragma Remote_Call_Interface;
Type Node_Ref is access all All_Node.All_Node'Class ;
Type Remote_Node_Data is ….
Type Node_Ref_List is ….
…
end Proxy ;
```

A joining node must establish a connection with some other node in the network. Each node in the network knows only about those neighboring nodes with which it is directly connected. Node interaction is completely ad hoc and reliable, in that each node may decide when and how to talk to neighboring nodes. Such communication can be synchronous or asynchronous, depending on the operation involved. Providing efficient asynchronous communication is easier said than done, however. For instance, Sun's remote method invocation (RMI)[9], the most prominent example of object-oriented middleware that extends the Java programming language (Java), does not provide asynchronous object requests. To build asynchronous communications, Java/RMI programmers must use threads and synchronous method invocations. We did not have to worry about this, however, since asynchronous communication is a built-in feature of GLADE [10], the distributed object-based middleware we used as part of our framework's implementation. Experienced software designers will agree that built-in constructs are easier to use and execute more efficiently than those superimposed on a middleware.

GLADE is an open source implementation of a distributed system annex (DSA), which extends the Ada95 programming language [11]. It is based on GNAT [12], which is also an open source implementation of Ada95. As such, GLADE inherits Ada95's principal properties, including strong typing, native multithreading, and real-time distributed object-based supports. These properties permit our implementation to easily meet the design challenges of reliability and performance in self-organizing systems like MANETs and the diverse applications that run on these infrastructures. We tested our model successfully on a network of PCs running under Linux.

5. CONCLUSION

This paper presented a new approach to the way that self-organizing systems are implemented and applied, by taking the important characteristics of these systems into consideration. This paper focused on the use of Ada features for implementing a framework capable of evolving over time and able to provide efficient data access and distribution in highly dynamic self-organizing systems. A subset of a MANET routing protocol was used as a test example. We are currently seeking to extend the framework's capabilities in order to investigate Ada's impact both on the design and implementation phases and on the performance evaluation of highly dynamic self-organizing systems, particularly MANETs.

6. REFERENCES

[1] http://gnutella.wego.com

[2] Ian Clark, Oskar Landberg, Brandon Wiley, Theodore W. Hong : Freenet : A Distributed Information Storage and Retrieval System. Proceedings of International Workshop on Design Issues in Anonymity and Unobservability. Lecture Notes in Computer science 2009, Springer Verlag Berlin 2001.

[3] Franck Dabek, Emma Brunskill & all. : Building Peer-to-Peer Systems With Chord, A distributed Lookup Service. In Proc. of the 8th Workshop on Hot Topics in Operating Systems (HOTOS_VIII), 2001

[4] Zygmunt J.Hass, Jing Deng, Ben Liang, Panogiotis Papadimitraros, and S. Sajama : Wireless Ad Hoc Networks.In Encyclopedia of Telecommunications, John Proakis editor, John Wiley 2002.

[5] A.Qayyum, L. Viennot, A. Laouti : Multipoint Relaying: An Efficient Technique For Flooding in Mobile Wireless Networks. 35th Annual Hawaii Conference on System Sciences (HICSS'2001)

[6] OLSR Optimized Link State Routing Protocol http://hipercom.inria.fr/olsr

[7] P. Jacquet, T. Clausen, & all. Internet-draft, draft-ietf-manet-olsr-11.txt, work in progress, July 2003

[8] Douglas E. Comer , David L. Stevens : Internetworking with TCP/IP, Vol. II: Design, Implementation, and Internals. Prentice Hall, 1999

[9] John T. Moy : OSPF Complete Implementation. Addison-Wesley, 2001

[10] Java/RMI : http://java.sun.com/j2se/1.4.1/docs/api

[11] Laurent Pautet, Samuel Tardieu : Glade User's Guide. Glade version 3.15p, September 2002. http://libre.act-europe.fr/GNAT

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

[13] GNAT http://www.act.com

[14] Laurent Pautet, Samuel Tardieu: GLADE : a Framework for Building Large Object-Oriented Real-time Distributed System. Proc. Of the 3rd IEEE International symposium on Object-Oriented Real-time Distributed Computing (ISORC'00), Newport Beach, California, USA, June 2000.