HIGH LEVEL SPECIFICATION OF A GEOGRAPHIC ROUTING PROTOCOL FOR MOBILE AD HOC NETWORKS

by

Xianghua Jiang
B.Sc., University of New Brunswick, 2002

PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

In the School
of
Computing Science

SIMON FRASER UNIVERSITY

Fall 2004

APPROVAL

Name: **Xianghua Jiang**

Degree: **Master of Science**

Title of Project: **High Level Specification of a Geographic Routing Protocol for Mobile Ad Hoc Networks**

Examining Committee:

Chair: **Petra Berenbrink**
Assistant Professor, Department of Computer Science

Uwe Gleasser
Senior Supervisor
Associate Professor, Department of Computer Science

Martin Ester
Supervisor
Associate Professor, Department of Computer Science

Qianping Gu
Internal Examiner
Associate Professor of/Department of Computer Science

Date Defended/Approved: September 13th, 2004

SIMON FRASER UNIVERSITY

PARTIAL COPYRIGHT LICENCE

ABSTRACT

Mobile ad hoc networks (MANETs) are designed for wireless communication and require no fixed infrastructure as the mobile hosts also perform routing tasks. The high dynamics of such self-organizing networks require routing strategies substantially different from those employed in static communication networks. Because of the high mobility and limited resources in MANETs, designing an efficient and reliable routing strategy becomes a very challenging problem and received a lot of attention in recent years.

Although there are various protocols of MANETs, they are all described in a theoretical basis. From a software engineering point of view, the theoretical aspects are often very hard to understand by a software development team. Also, the cost of implementing such protocols in order to compare the performance of various protocols is too high. Therefore, the idea of trying to use high-level executable specification languages came into our mind.

We define a *distributed abstract state machine* (DASM) model of the network layer protocol for mobile ad hoc networks. We represent a DASM model and a high-level SDL specification of a layered communication architecture for an efficient geographic routing protocol for *mobile ad hoc networks* (MANETs). Our goal of modelling the protocols with ASMs is to support the definition and validation of wireless communication protocols and implementations based thereon. Our objective to define high-level specification with SDL for the protocols is to sharpen loosely defined system requirements into an architectural specification serving as a formal basis for analysing key system properties by analytical means and experimental validation using commercial SDL tools.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Uwe Glässer, for his supervision and guidance in the preparation of this project. Also, thank committee members, Dr. Petra Berenbrink, Dr. Martin Ester and Dr. Qian-Ping Gu's advice for me to finish this project report.

I would like to thank my parents and my sister for their support and encouragement.

TABLE OF CONTENTS

Approval ii

Abstract iii

Acknowledgements iv

Table of Contents v

List of Figures vi

1 Introduction 1

2 An Overview Routing Protocol in Mobile Ad Hoc Networks 3

3 An Efficient Geographic Routing Protocol 5

3.1 Distributed Location Service 6

3.2 Position Based Routing 7

4 Abstract Network Model With Dasm 9

4.1 Communication Infrastructure 9

4.2 Abstract Communication Model 10

4.2.1 Position Based Routing 12

4.2.2 Detection of Nearest Neighbours 12

4.2.3 Packet Routing 16

4.2.4 Distributed Location Service 17

5 SDL Specification of a Layered Communication Architecture 21

5.1 Overview of the Specification and Description Language (SDL) 21

5.1.1 System Behaviour 22

5.1.2 System Structure 22

5.1.3 Abstract Data 24

5.1.4 Diagram of System, Block, Process, Procedure and Substructure 24

5.2 SDL Specification of a Layered Communication Architecture 28

5.2.1 Position Based Routing 32

5.2.2 Detection of Nearest Neighbors 32

5.2.3 Packet Routing 33

5.2.4 Distributed Location Service 36

6 Discussion and Conclusion 39

Reference List 41

LIST OF FIGURES

Figure 1 Layered communication architecture 6
Figure 2 Sector si (p) with nearest neighbor ni(p) 7
Figure 3 Logical structure of packets for the communication between protocol entities 11
Figure 4 Interaction between system and environment 21
Figure 5 System behaviour [8] 22
Figure 6 Structuring a system into blocks and processes 23
Figure 7 Structure of SDL system 23
Figure 8 Structuring a block into process types 24
Figure 9 Newtype Node 24
Figure 10 System diagram 25
Figure 11 Block Diagram 26
Figure 12 Process diagram 27
Figure 13 Procedure diagram 27
Figure 14 Layered communication model 28
Figure 15 Networking service architecture 30
Figure 16 Position based routing 31
Figure 17 Neighbour detection cycle and packet forwarding 33
Figure 18 Neighbour detection cycle and packet forwarding (ctd). 35
Figure 19 Neighbour detection cycle and packet forwarding (ctd). 35
Figure 20 Distributed location service 36
Figure 21 Distributed location service (ctd.) 37
Figure 22 HandleLocationReply procedure 38

1 INTRODUCTION

Formal models and methods can be very useful in practical software development. They are used to reveal ambiguities, incompleteness, and inconsistencies in system requirements specifications. "For a method to be formal, it must have a well-defined mathematical basis, typically given by a formal specification language." [9] The group on Foundations of Software Engineering at Microsoft Research has developed the high-level executable specification language Asml based on the concept of *abstract state machine* (ASM). Asml supports specification and rapid prototyping of object oriented and component oriented software. [6] It is a successful practical instrument for systems design. ASMs are also used in a current ITU-T standard to formally define the semantics of the *Specification and Description Language* (SDL), a widely used specification language in the telecommunications area. In our project, we define a *distributed abstract state machine* (DASM) of an efficient ad hoc geographic routing protocol for *mobile ad hoc networks* (MANETs) proposed by A. Benczur and T. Lukovszki [1] and present a high-level specification of a layered communication architecture for this routing protocol based on SDL. Our goal of modelling the network layer protocol with ASML is to support the definition and validation of wireless communication protocols and implementations based thereon. The resulting behaviour model forms a formal basis for developing executable specifications and can serve as a platform for experimental validation of key system attributes and exploration of alternative design choice. Our objective in defining high-level protocol specifications with SDL is to sharpen loosely defined system requirements into an architectural specification serving as a precise basis for analysing key system properties by analytical means and experimental validation using commercial SDL tools.

In Section 2 of our report, we give an overview of routing protocols in MANETs. In Section 3, we describe an efficient geographic routing protocol for MANETs which was proposed by A. Benczur and T. Lukovszki [1]. In Section 4 and 5, we respectively define a DASM model and an SDL model for the geographic routing protocol. In Section 6, we give a discussion and some conclusions.

2 AN OVERVIEW ROUTING PROTOCOL IN MOBILE AD HOC NETWORKS

MANETs are designed for wireless communication and require no fixed infrastructure as the mobile hosts also perform routing tasks. Typical applications, for instance, are the establishment of connectivity among handheld devices or between vehicles. The high dynamics of such self-organizing networks require routing strategies substantially different from those employed in static communication networks. Storing and updating large routing tables at mobile hosts would congest the network with administration packets very fast. Because of the mobility and limited resources such as transmission power and bandwidth in MANETs, designing an efficient and reliable routing strategy becomes a very challenging problem.

Routing protocols in MANETs can be classified into three different groups: table-driven/proactive, demand-driven/reactive and hybrid. In the proactive protocols, each node maintains routing information to every other node in the network. The routing information is usually kept in a number of different tables who are periodically updated if the network topology changes. It attempts to keep an up-to-date topological map of the entire network. With this map, the route is determined immediately when a packet needs to be sent. In reactive protocols, a route determination procedure is invoked on demand when a packet needs to be forwarded. These routing strategies were designed to reduce the overheads in proactive protocols by maintaining information for active routes only. [10]

Both proactive and reactive routing has specific advantages and disadvantages. The proactive routing maintains information up-to-date, so that the delay before sending a packet is minimal. On the contrary, the reactive routing has to determine the route first, which may result in considerable delay if the information is not available from caches. It also inefficiently floods

the entire network with administrative packets for route determination. However, the proactive routing uses a large portion of bandwidth to maintain routing information up-to-date. [10]

The hybrid routing strategy is designed to increase the scalability by combining the advantages of local proactive and global reactive routing. In hybrid routing protocols, nodes with close proximity work together to form some sort of a backbone to reduce the route discovery overheads. This is mostly achieved by proactively maintaining routes to nearby nodes and reactively determining routes to far away nodes. [10] Even a combination of both strategies still needs high communication overhead to maintain routing fabrics in the MANET. This is the reason why the topological-based routing protocols, including proactive protocols (DSDV, OLSR, FSR and TBRPF), reactive protocols (DSR and AODV), and hybrid protocols (ZRP) have not been put into deployment. [7]

Geographical-based routing protocols eliminate some of the limitations of topology-based routing by using additional information, which are the physical positions of the participating nodes. With the growing popularity of locating devices (e.g., GPS), it is economically and technically feasible for the location device to determine its own position. The task of the geographic location service is to determine the position of the destination which needs to be included in packet header in addition to its destination address. [7] Then, the packet can be efficiently forwarded by means of the location information of intermediate neighbours.

3 AN EFFICIENT GEOGRAPHIC ROUTING PROTOCOL

In our project, we present an efficient geographic routing protocol proposed by A. Benczur and T. Lukovszki in [1]. This protocol combines the advantages of proactive and reactive routing. In this protocol, the *hypercubic location service* (HLS) which considers a very strong definition of fault-tolerance is used as the location service to find the position of the destination. Each mobile node consists of a set of distributed nodes that act as its location servers. Based on the assumption that every node can determine its current geographic position by using GPS or another type of positioning service, the nodes periodically resend position information to the location servers by the same routing mechanism used for normal data transmission. To initiate communication, one finds a location server of the destination node and obtains its geographic position. A location server can be determined by visiting a well-defined sequence of nodes such that each node acts as a location server of the next node in the sequence.

Building on the purely theoretical model in [1], basic algorithmic aspects of the protocol are modelled in the form of an abstract state machine [13]. Since the ASM is the formalism underlying the formal definition of SDL [12], we further elaborate on the fairly abstract algorithmic model of ASM constructing an architectural design revealing the core functionality of this routing protocol for mobile ad hoc networks. It turns out that the SDL paradigm of modelling complex protocols is particular appropriate for practical purposes.

In [1], the network layer is divided into two separate sublayers, one for a *distributed location service* (DLS), and one for a *position based routing* (PBR) between known locations, as illustrated in Figure 1. These two sublayers cooperatively implement the network layer protocol of a mobile ad hoc network, where the location service is based on HLS as part of a highly fault

tolerant, self-scaling architecture. For resolving interference problems, we assume the existence of a Media Access or MAC layer, e.g. IEEE 802.11, a commonly used standard.

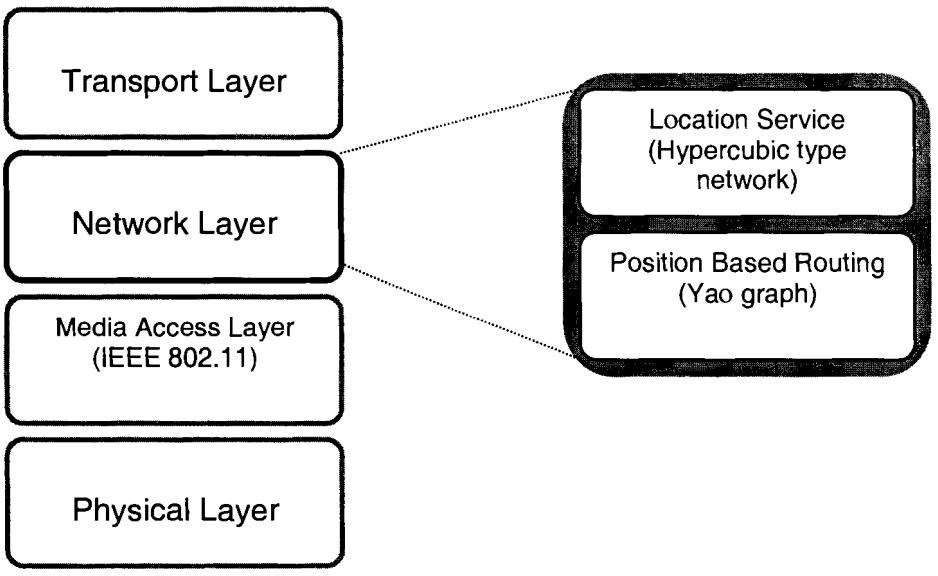

Figure 1 Layered communication architecture

3.1 Distributed Location Service

Conceptually, the DLS sublayer consists of dynamic hypercubic networks [11] with random IDs of nodes identifying the mobile devices. Random distribution of node Ids can be easily achieved by hashing the node's IP or other physical address into the device. Each node in the network maintains location information of its hypercubic neighbours by periodically sending its own location to those neighbours. When the location of a node is known, a PBR strategy is then used to pass messages to that node as outlined below.

To send data from a node u to another node v, first u has to detect the location of the destination node v as follows. The ID of v can be obtained from the ID of u through performing a certain sequence of elementary bit operations. This sequence defines a path in the hypercubic network from u to v. On this path node u then sends a location request to v carrying the ID of both u and v as well as the location of u.

Since the hypercubic neighbours are storing up-to-date location information from each other, the location request will be sent to the next node on the hypercube path by position based routing. When the location request reaches v, then v replies by sending its own location to u using position based routing. After u has received the position of v it sends the data to v using position based routing as well.

3.2 Position Based Routing

Based on the assumption that every node can determine its current geographic position at any time - e.g., by using a global positioning system or GPS - nodes periodically exchange position information by the same routing mechanism used for normal data transmission. Basically, any position based routing can be used in combination with the DLS. In [1], the so-called Yao-graph [3] is proposed as network topology

The Yao-graph $Gy(V) = (V,Ey)$, for a set V of n points in the plane, defines for each p in V a fixed number k of sectors $si(p)$, $i = 0,\ldots,k-1$, that have their origin in p. The orientation of $si(p)$ is given by the two edges $hi(p)$, $hi+1(p) \in Ey$ as illustrated in Figure 2. With each sector $si(p)$ we associate some nearest neighbour $ni(p) \in V$ located within $si(p)$. A nearest neighbour $ni(p)$ is determined based on the Euclidean distance from p to $ni(p)$.

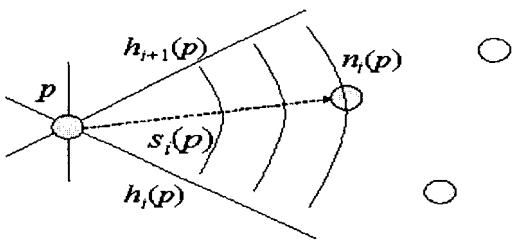

Figure 2 Sector si (p) with nearest neighbor ni(p)

The Yao-graph neighbours can be determined efficiently in a distributed and power efficient manner, which is to find a route whose energy consumption is within a small constant factor of the optimal route [15]. This distributed construction assumes that the mobile nodes use directed radio and that they can increase the transmission power within the sectors until at least one node replies y sending an acknowledge signal containing the geographic position of this node. Here we assume for simplicity that each node is within the maximum transmission range of each other node.

4 ABSTRACT NETWORK MODEL WITH DASM

We consider an asynchronous communication architecture consisting of some finite number of mobile hosts interconnected through a wireless communication network. The global topology is formed by viewing the mobile hosts as nodes of the network. Each node has the ability to act as communication endpoint and as router at the same time. The network is ad hoc and as such changes its topology frequently and without prior notice. Nodes interact with each other by asynchronously sending and receiving packets of variable length.

We construct our DASM network model serves as a basis for further development and experimental validation. For a rigorous mathematical definition of the underlying computation model, we refer to the original literature on the theory of ASMs [16]. Also see a tutorial introduction to the DASM paradigm of systems modelling [13].

4.1 Communication Infrastructure

We define the total number N of nodes as a parameter of the communication network. Because of the mobility aspect in MANETs, each node determines its current position through the use of locating device, for instance, such as a GPS receiver. In a fixed global space, each node has a unique ID/address by mapping some physical addresses, e.g. such as Internet host names, IP addresses, or MAC addresses to the node ids/addresses by some hashing function. We can mapping from nodes to some abstract domain of position and addresses via the *pos* and *address* function respectively. We define domains and functions with AsmL as follows [2]:

```
Domain NODE = {n_1, ... , n_N }
Domain POSITION, monitored pos : NODE -> POSITION
Domain ADDRESS, address : NODE -> ADDRESS
```

Behaviours of network protocol entities are represented by an asynchronous computation model in the form of the DASM agents. As described in Section 3, the network layer consists two sublayers: DLS and PBR. Then, each node has two autonomously operating agents to execute these two programs as stated in Section 3 by a unary dynamic function program defined on agents. Thus, the domain AGENT includes two sets: DLS and PBR. [2]

```
Domain AGENT = DLS ∪ PBR
node : AGENT -> NODE
```

There are three different kinds of operating status of a node in any given state of the network: switched on (active), switched off (passive), or undefined (crashed). We introduce monitored function *status* and *now* to represent the current status and the time as measured by some local clock on the node respectively. [2]

```
Domain STATUS = {switched_on, switched_off, undefined}
Domain TIME
Monitored status : NODE -> STATUS
now : NODE -> TIME
```

4.2 Abstract Communication Model

Nodes send and receive packets consisting of two basically distinct parts, a packet header and the actual payload, as illustrated in Figure 3. The header has a fixed structure consisting of components identifying: the final destination node Dest, the next receiver node Rcvr (next hop on the way to Dest), the sender node Sndr, and the packet type Type. Each of these node references

in a packet header specifies a node address and its geographic location. For representing those node references, we introduce the abstract data type NREF. [2] "The distinction between node references and nodes is crucial since nodes can continuously update their location information, whereas node references do not change while a packet is in transmit." [14]

```
Domain PACKET

Domain NREF,

sndr, rcvr : PACKET - > NREF
```

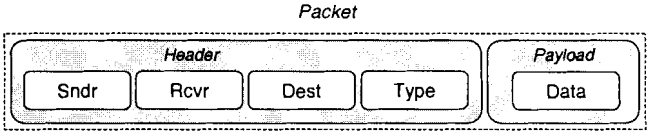

Figure 3 Logical structure of packets for the communication between protocol entities

"There are three basically different types of packets, namely: (1) detection packets, which are meaningful for the position based routing only; (2) discovery packets, which are meaningful for the distributed location service only; and (3) data packets. For detection packets, we further distinguish between neighbour requests and neighbour replies. Similarly, for discovery packets, we distinguish between location requests and location replies." [2]

```
Domain DETECTION = {neighborRequest, neighborReply}

Domain DISCOVERY = {locationRequest, locationReply}

Domain DATA

type : PACKET -> PACKETTYPE = DATAPACKET ∪ DETECTION ∪ DISCOVERY
```

According to the vertical view of communication, adjoining layers directly interact with each other through *service access points*. The network layer interacts with the MAC layer and the transport layer. Similarly, the position based routing, or PBR, interacts with the distributed location service, or DLS. Such interactions are restricted to the operations as provided by the *service primitives* of a *service access point*. We use here two kinds of *service primitives*: (1)

forwarding a single packet to the next high/lower layer; (2) receiving a single packet from the next high/lower layer. [2]

```
PACKET_to_<Layer>(<Packet>[{,<Parameter>}])

PACKET_from_<Layer>(<Packet>)
```

4.2.1 Position Based Routing

As described in Section 3, we associate with every node a fixed number k of sectors. The *neighbour* function is used to identify the nearest neighbour through a dynamic mapping from node sectors to corresponding node references. We define Euclidian distances on the plan as the DISTANCE domain and introduce a binary operation *distance* to calculate the distance between 2 nodes. [2]

```
domain SECTOR = {  … k-1}

neighbour : NODE -> (SECTOR -> NREF)

Domain DISTANCE

distance : POSITION x POSITION -> DISTANCE
```

4.2.2 Detection of Nearest Neighbours

The search for the nearest neighbours of a given node splits into k independent search operations, one for each of the k node sectors. In order to prevent the interference between different sectors, the sector information is encoded into the data part of the detection packets by means of *sector* function. [2]

sector: PACKET - > SECTOR

Nodes periodically update the information about their neighbours as the position of the nodes change dynamically. The frequency of updates depends on various parameters. We use the *updateNeighborEvent* to trigger the start of the detection cycles. Also, we have to store the intermediate new neighbour results until the completion of a detection cycle. [2]

```
monitored updateNeighborEvent : AGENT x SECTOR -> BOOL

newNeighbor : NODE -> (SECTOR -> NREF)
```

Each sector of a node has its own timer operating under control of the PBR agent of that node. A *timer* operation is used to specify the expiration time for a detection cycle. The *time* function is used to access the time stamp of the packet. A timer with time value t, now <= t < ∞, is considered to be active. The actual time form for detection cycles is represented by a distinguished element *duration* from a static domain of finite time intervals. [2]

```
time: PACKET -> TIME

timer: AGENT x SECTOR -> TIME  ∪ {∞}

Domain DURATION,

duration : DURATION
```

We can now define the rules for nearest neighbour detection that are executed by PBR agents as part of the *PositionBasedRouting* program. In the below rules, *ag* refers to a DASM agent from PBR. [2]

```
nearestNeighborDetection =
    let node = ag.node, position = ag.node.pos, time = ag.node.now
        forall s in SECTOR
            //start new detectioncycle
            if UpdateNeighborEvent(ag,s) then
                IssueNeighborRequest(s, position,time)
                NewNeighbor (node)(s): = undef
                timer(ag,s): = now(node) + duration //set timer
            if now(node) >= timer(ag,s) then //detection cycle timeout
                if newNeigbor(node)(s) ∈ NREF then
                    // assign finalresult
                    neighbor(node)(s) := newNeighbor(node)(s)
                timer(ag,s) := ∞ //reset timer
```

On the position based routing layer, the detection packet including neighbourRequest and neighbourReply are handled by two different operations, namely: *handleNeighbourRequest* and *handleNeighbourReply*. In the *handleNeighbourRequest*, new neighbour request packet is created by PBR containing the relevant information, namely: the node address and geographical position, the sector ID, and a time stamp. This packet then is sent to nodes within the specified sector by means of the MAC layer. The request and the reply are logically linked by copying information from request packet into the reply packet. In the *handleNeighbourReply*, the selection of neighbour is decided by comparing the distance to a new responding neighbour and the distance to the nearest neighbour detected so far. [2]

```
HandleNeighborRequest(p:PACKET) =
     extend PACKET with q
        extend NREF with r, s
            q.rcvr := r
            q.sndr := s
            r.address := p.sndr.address,
            r.position := p.sndr.position
            s.address := ag.node.address, s.position := ag.node.pos
            q.type := neighborReply
            q.time := p.time //copy time stamp from request packet
            q.sector := p.sector
            //copy sector information from request packet
            Packet_to_MAC(q, p.sndr.position)
```

```
HandleNeighborReply(p: PACKET) =
   let s = p.sector, node = ag.node, pos = ag.node.pos
       if p.time >= ag.node.now - durartion then
       // check distance against intermediately stored nearest neighbour
       let a = pos, b = p.sndr.positon, c = position(newNeigbor(node)(s))
        if distance (a,b) < distance (a,c) then
            newNeigbor (node)(s) := p.sndr
```

4.2.3 Packet Routing

The packet can be delivered locally by handing it over the distributed location service running on the local node or forwarded to the nearest neighbour with sector that matches with the position of the final destination node. It is decided by the position based routing by comparing the coming packet's receive node address with the local node address. And the respective nearest neighbour is computed through the position information of the local node and the destination. [2]

```
computerNeighbor: NODE x POSITION -> NREF
```

As described, the position based routing performs the actual routing task as well as the control of nearest neighbour detection cycles and the handling of detection requests and detection replies. Below is the model of the position based routing layer defined through the DASM program PositionBasedRouting. In the program, *ag* refers to a DASM agent from PBR. [2]

```
PositionBasedRouting =
   if status(ag.node) = switched_on
     NearestNeighborDetection // run detection cycle in parallel
     if Packet_from_MAC (P: PACKET) then
        if p.type = neighborRequest then
            HandleNeighborRequest(p)
        else
            if p.type = neighborReply then
               HandleNeighborReply(p)
            else // packets other than detection packets
               if ag.node.address = p.rcvr.address then
                   Packet_to_DLS(p) // deliver packet locally
               else // forward packet to remote destination
                   if computeNeighbor (ag.node, p.rcvr.position) ∈ NREF then
                      let v = computeNeighbor (ag.node, p.rcvr.position)
                          Packet_to_MAC(p, v.position) // Perform next hop
   if Packet_from_DLS(p:PACKET) then
     Packet_to_MAC (p, p.sndr.position)
else // node is passive
   Skip
```

4.2.4 Distributed Location Service

As described in Section 3, we associate with each node a set of direct neighbours in the dynamic hypercube as the distributed location services. The hypercubic neighbour on the way to the final destination is computed by a given node and a given destination address. Two functions are defined as following [2]:

```
hypercubicNeighbors : NODE -> NREF-Set

nextHyercubicNeighbor (u,a) = v ∈ NREF: v ∈
hypercubicNeighbors(u) ∩ v.address = computeNextID(u.address, a)
```

The function *computeNextID* performs the actual address calculation on the dynamic hypercube. For brevity, we refer to [1] for the definition of this address calculation.

```
computeNextID : ADDRESS x ADDRESS -> ADDRESS
```

Consider some sequence of consecutive packets received from the transport layer. We assume that the first packet can be recognized by the Boolean-values function on packets and needs to be stored locally to first determine the position of the destination node before sending it. [2]

```
monitored Firstpacket : PACKET -> BOOL

firstpacket :DLS ->PACKET
```

A location reply to the location service always matches a pending location request. Thus, the contained position information of the sender is added to a waiting first packet, which then eventually can be sent via the position based routing. [2]

When receiving a location request, the location service checks the address of the final destination. This address is encoded into the data part of the discovery packet. We therefore introduce a partial dynamic function *address* defined on packets. [2]

```
address : PACKET -> ADDRESS
```

If the final destination address does not match with the local node, we calculate the next hypercubic neighbor on the way to the final destination and forward the request.

```
HandleDiscoveryPacket(p:PACKET) =
   if p.type = locationReply then // extract position information
     ag.firstpacket.rcvr.position := p.sndr.position
     Packet_to_PBR(ag.firstpacket)
   if p.type = locationRequest then
     if p.address = ag.node.address then // generate location reply
        HandleLocationReply(p)
     else // compute next hypercubic neighbor and forward packet
        let v = nextHypercubicNeighbor(ag.node, p.address)
            p.rcvr := v // specify next hop in the hypercube
            Packet_to_PBR(p)
```

We can now define the model of the distributed location service through the below DASM program *DistributedfLocationService.*

```
DistributedfLocationService =
   if  Packet_from_Transport(p:PACKET) then
     if Firstpacket(p) then // discover destination position first
        DiscoverDestinationPosition(p.rcvr.address)
        Ag.firstpacket := p // intermediately store first packet
     else // destination position is already known
        Packet_to_PBR(p),
        p.rcvr.position := ag.firstpacket.rcvr.position
   if Packet_from_PBR(P:PACKET) then
     if p.type ∈ DISCOVERY then
        HandleDiscoveryPacket(p)
     else Packet_to_Transport(p)
```

```
DiscoverDestinationPosition(a:ADDRESS) =
   let v = nextHpercubicNeighbor (ag.node, a)
     Extend PACKET with q // create location request packet
        Extend NREF with r, s
           q.rcvr := r, r.address := v.address,
r.position := v.positiuon
           q.sndr := s, s.address := ag.node.address,
s.position := ag.node.pos
           q.type := locationRequest
           q.address := a
        Packet_to_PBR(q)
```

5 SDL SPECIFICATION OF A LAYERED COMMUNICATION ARCHITECTURE

5.1 Overview of the Specification and Description Language (SDL)

SDL is a standardized language for the specification and description of systems. It has been developed by the ITU-T, the telecommunication standardization body of the International Telecommunication Union (formerly known as CCITT). In contrast to a program, a formal specification is not intended to be run on a computer. A specification is a basis for driving implementations, it abstract from implementation details in order to give overview of a complex system. In addition to serving as a basis for driving implementations, it can be used for precise and unambiguous communication between human being. Also, the use of a specification language makes it possible to analyse and simulate alternative system solutions, which is not feasible for programming language due to the cost and the time delay. [8]

"SDL sees the world as divided into two parts: the system and its environment" [8]. The specification defines how the system reacts to events in the environment by signals, see Figure 4.

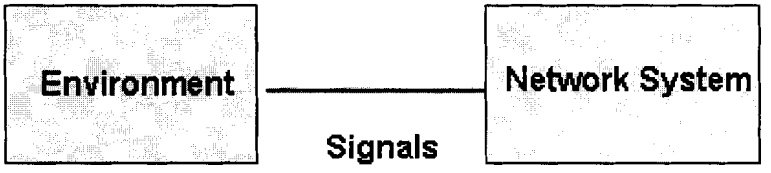

Figure 4 Interaction between system and environment

5.1.1 System Behaviour

The behaviour of a system is constituted by the combined behaviour of its parts – as represented by a number of processes in the system, see Figure 5. [8] SDL process instances are extended finite-state machines that work autonomously and concurrently with each other and interact through exchanging signals.

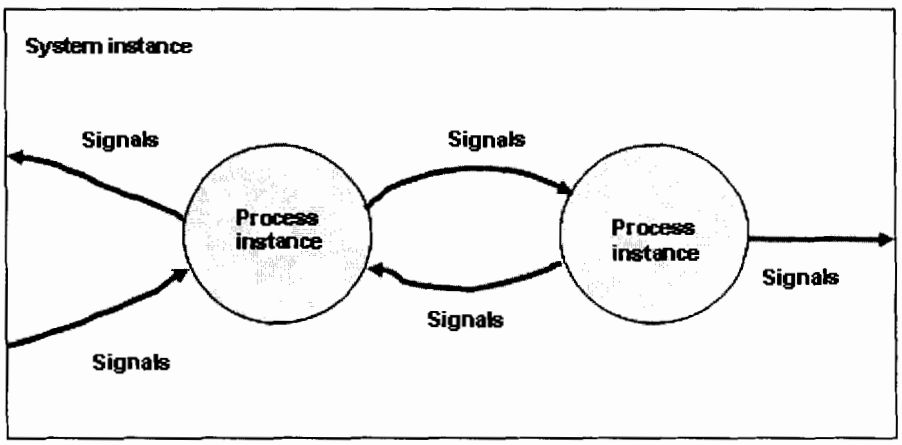

Figure 5 System behaviour [8]

5.1.2 System Structure

SDL provides mechanisms to structure systems to cope with complexity. In basic SDL, a system description is structured into block descriptions and process descriptions, see Figure 6. A system description contains one or more blocks interconnected with each other and with the boundary of the system by channels. A channel is a means of conveying signals, see Figure 7. A block contains one or more processes that communicate with each other via signal routes, see Figure 8. A procedure description can appear in a process description or in another procedure description.

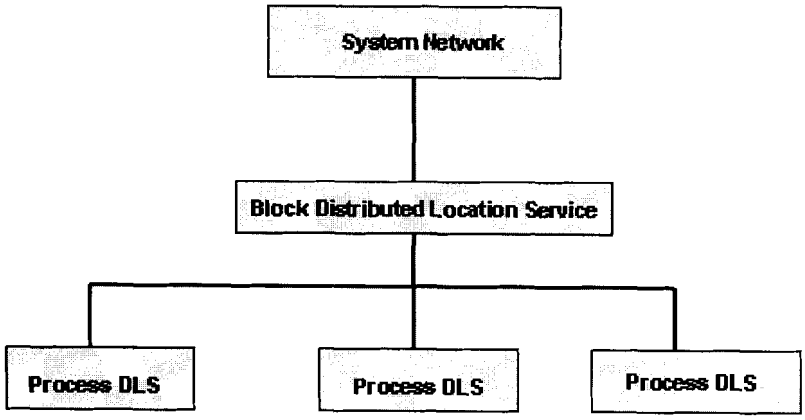

Figure 6 Structuring a system into blocks and processes

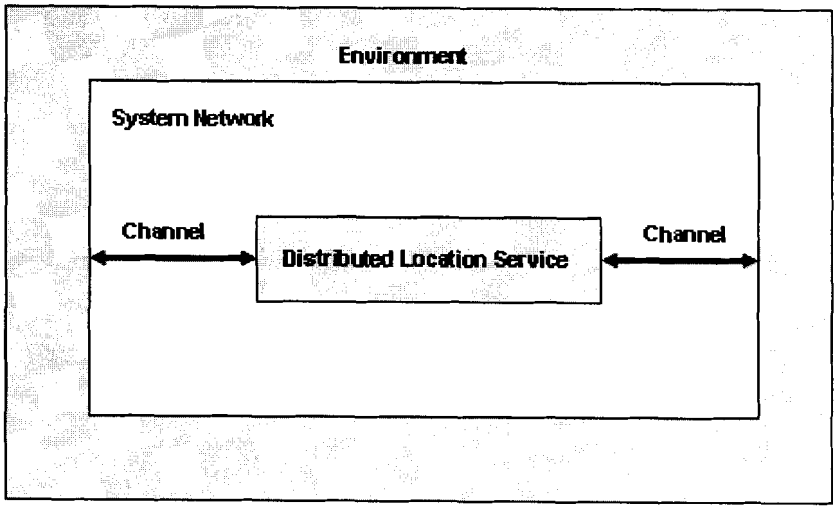

Figure 7 Structure of SDL system

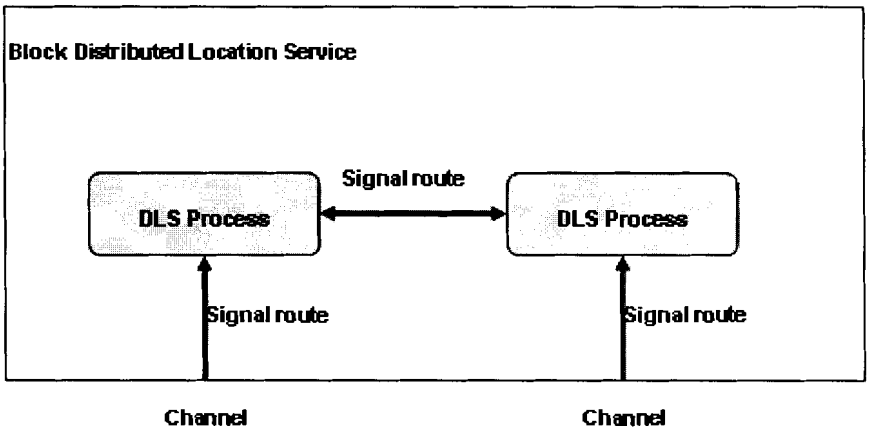

Figure 8 Structuring a block into process types

5.1.3 Abstract Data

SDL uses a concept of abstract data type which is quite different from the data types used in programming languages. For example an integer defined in SDL corresponds to the mathematical concept of integer and is not limited by the computer to 16 or 32 bits. Also, the user is able to define new operator which is not possible for most programming languages. For example, we define new type node in our project as follows:

```
newtype Node
    struct
        address Address;
        position Position;
    operators
        nref: Node -> NREF;
End newtype Node
```

Figure 9 Newtype Node

5.1.4 Diagram of System, Block, Process, Procedure and Substructure

A system diagram usually contains system name, signal descriptions, channel descriptions, data type descriptions and block descriptions, see Figure 9. A signal description

contains a signal name and the types of values conveyed by the signal. A channel description contains a channel name, a list of signal names for signals that can be transported by the channel.

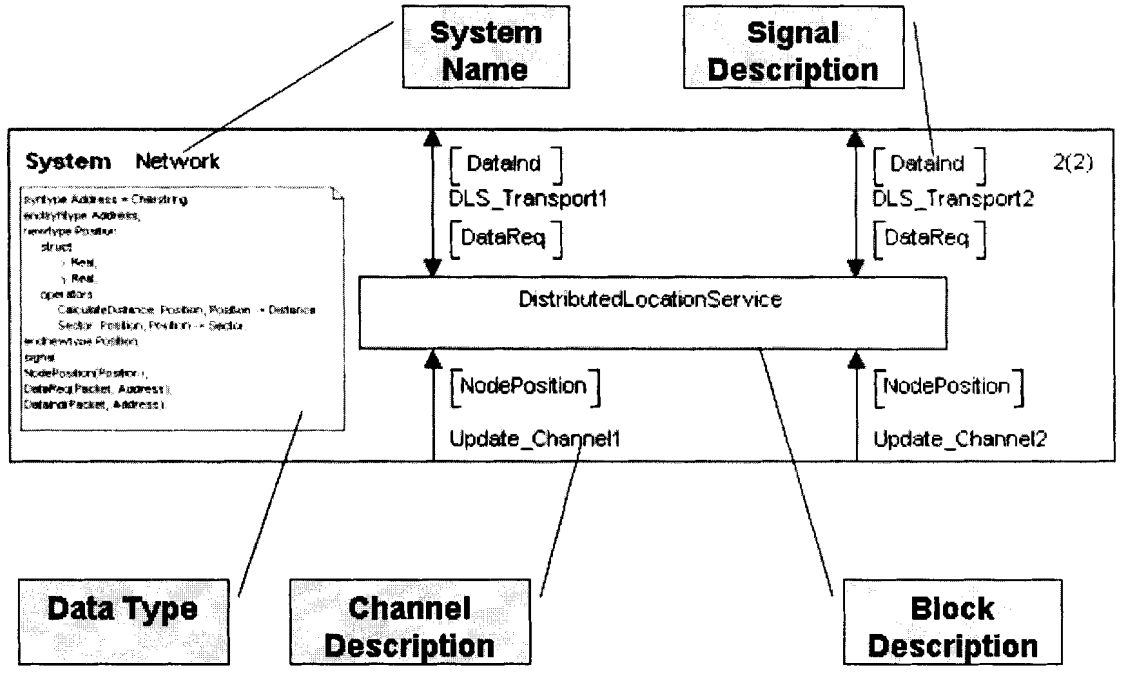

Figure 10 System diagram

A block diagram usually contains the block name, signal descriptions, signal rout descriptions, channel-to-route connections and process descriptions, see Figure 10.

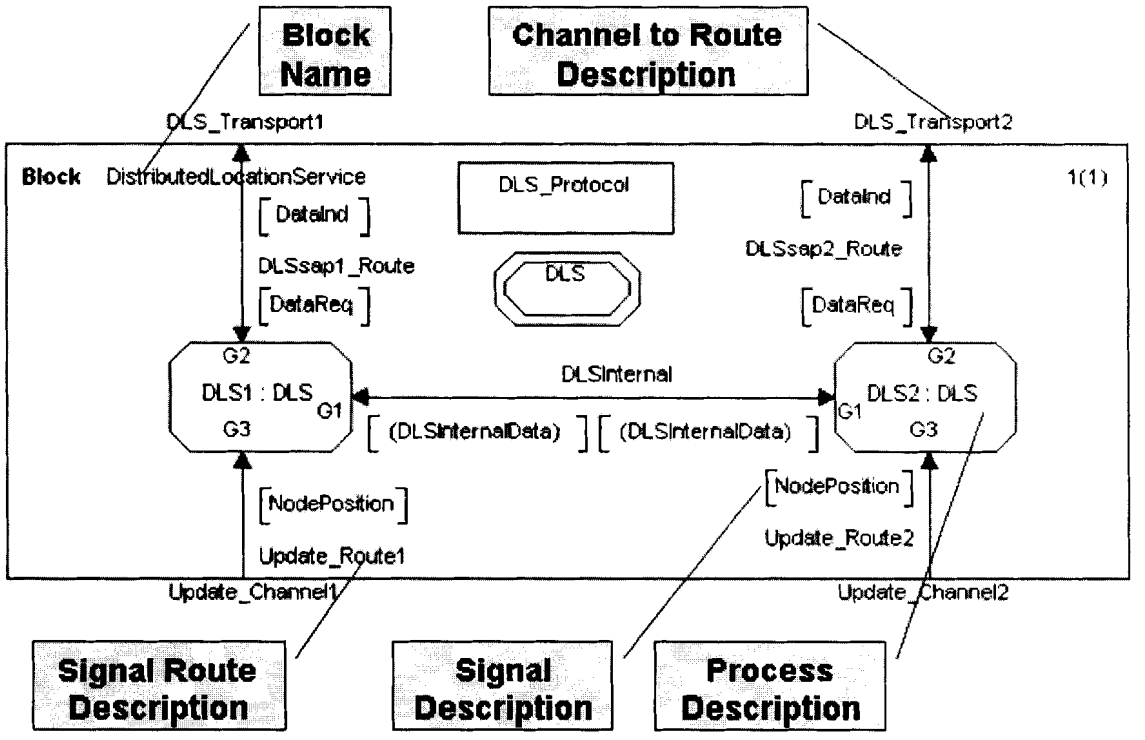

Figure 11 Block Diagram

A process diagram usually contains a process a name, formal parameters, variables descriptions, timer descriptions, procedure description and process graph, see figure 11. In SDL there are 6 basic constructs for the description of a process: start, input, current state, output, next state and return as illustrated in Figure 11.

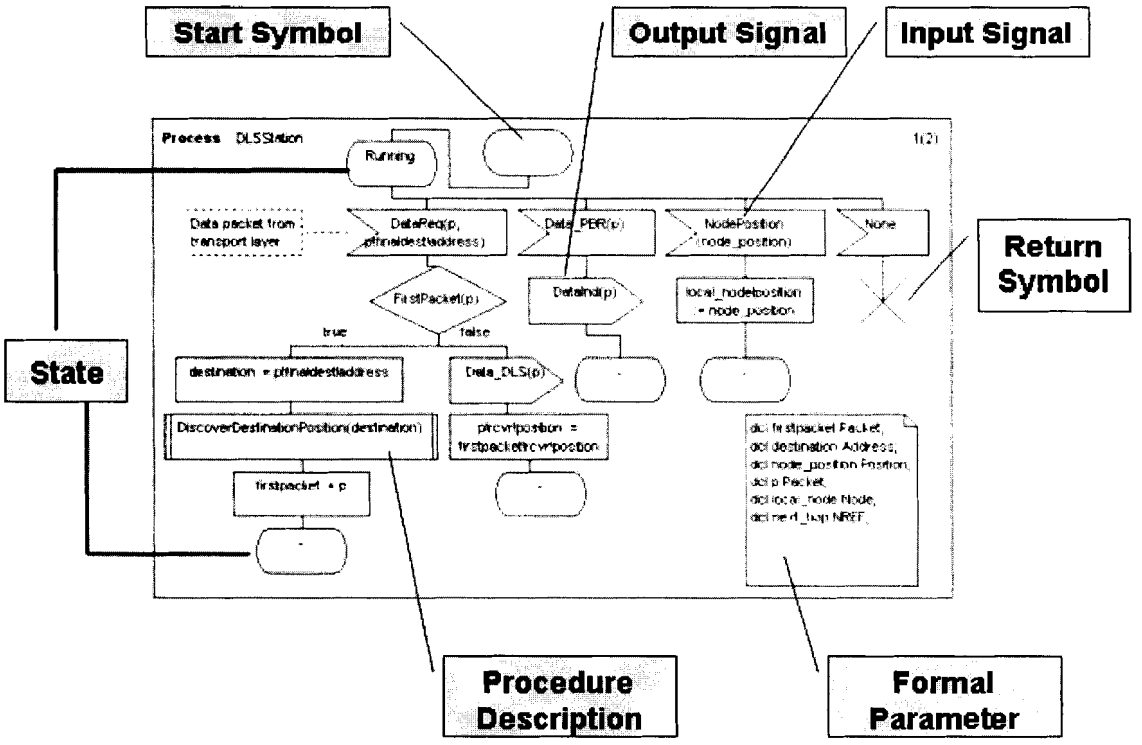

Figure 12 Process diagram

A procedure description is similar to the process description except for the start and return symbol (see Figure 12).

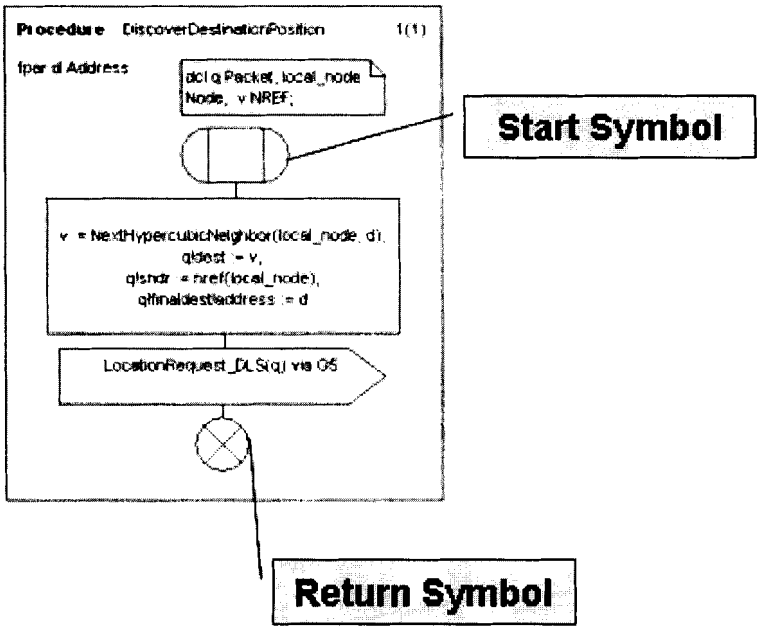

Figure 13 Procedure diagram

5.2 SDL Specification of a Layered Communication Architecture

Intuitively, we deal with two different views of communication, commonly referred to as *horizontal view* and *vertical view*. Direct communication between peer protocol entities (i.e., entities residing within the same layer) at different nodes is called horizontal communication. This form of communication is only virtual and effectively realized by the lower level network layers, as illustrated in Figure 13. For this purpose, the MAC layer renders a corresponding service to the network layer. This service is accessible through well-defined interfaces, called *service access points*. Similarly, the network layer renders its service to the next higher layer, the transport layer. Communication between different layers of the same node is called vertical communication. This form of the communication is restricted to adjoining layers.

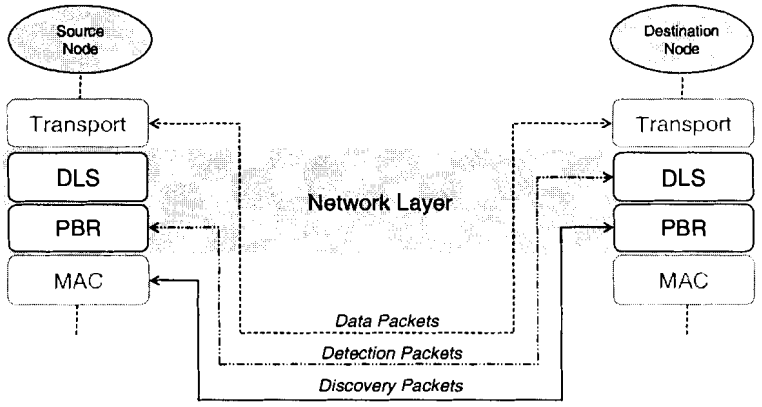

Figure 14 Layered communication model

In SDL, we model the network protocol through a system *Network* describing its behaviour on several different levels of abstraction in the form of a multi-layer architecture. The principles of layering require a strict hierarchical organization. Any interactions between layers are restricted to the operations as provided by the *service primitives* for the *service access points* of a service. To establish the service that a given layer n renders to the next higher layer n+1, the

underlying protocol employs the functionality encapsulated within the service at the next lower layer n-1. As such, the principles of layering resemble those of information hiding [5].

The SDL language supports the hierarchical definition of communication architectures through combined block specifications containing a *block substructure* in addition to *process* specification [8]. Combined *block* specifications express alternative views when looking at a *block* from different perspectives. Abstractly, a *block* specification may be viewed as containing only processes when dealing with behaviour. A contained *block substructure* then specifies an implementation of this behaviour through a collection of interconnected blocks, which in turn may contain *block substructures*, thus allowing for stepwise refinements. Structurally, both views of a combined block specification share the same external interfaces.

Our *Network* architecture consists of a single block *DistributedLocationService* that interacts with the external environment through the *service primitives* of the *service access points* as illustrated in Figure 14. The combined block specification of this block provides an abstract behaviour specification as well as an implementation of this behaviour in terms of a block substructure *DLS_Protocol.* The latter is defined by the block type *DLSunit* based on the PBR service encapsulated in the block *PositionBasedRouting.* The specification of the block type *DLSunit* introduces a process *DLSunit* using two procedures for discovering destination positions and handling of location replies respectively.

We illustrate the further structural decomposition of the block *PositionBasedRouting* into protocol entities whose behaviour is defined by the underlying SDL processes. Again, this calls for a combined block specification. The behaviour of this block is abstractly defined in terms of two PBR processes interacting by exchanging protocol data units (horizontal communication). The block substructure *PBR_Protocol* specifies an implementation effectively mapping those protocol data units to service data units of the underlying MAC service (vertical communication). This is illustrated in Figure 15.

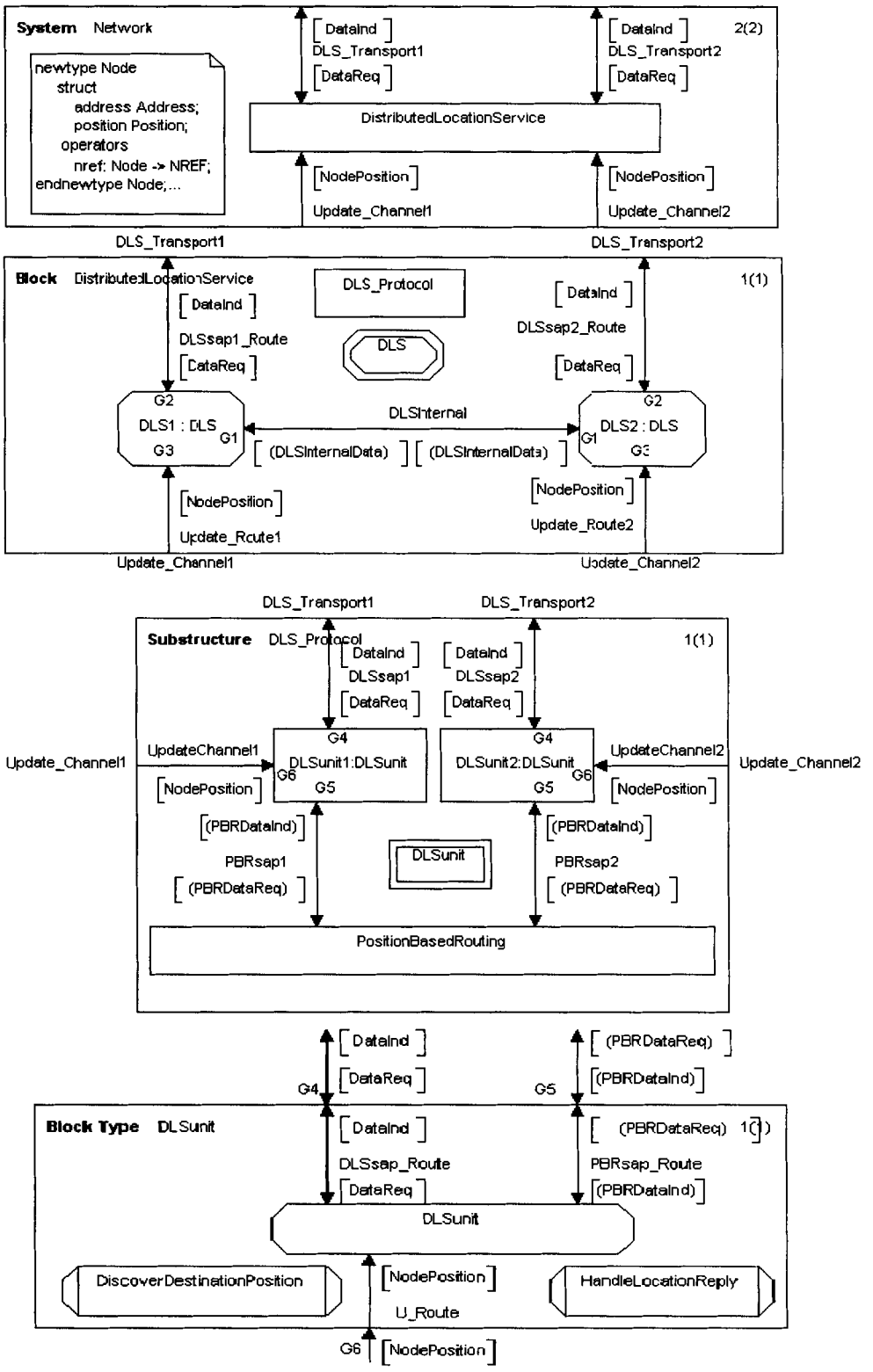

Figure 15 Networking service architecture

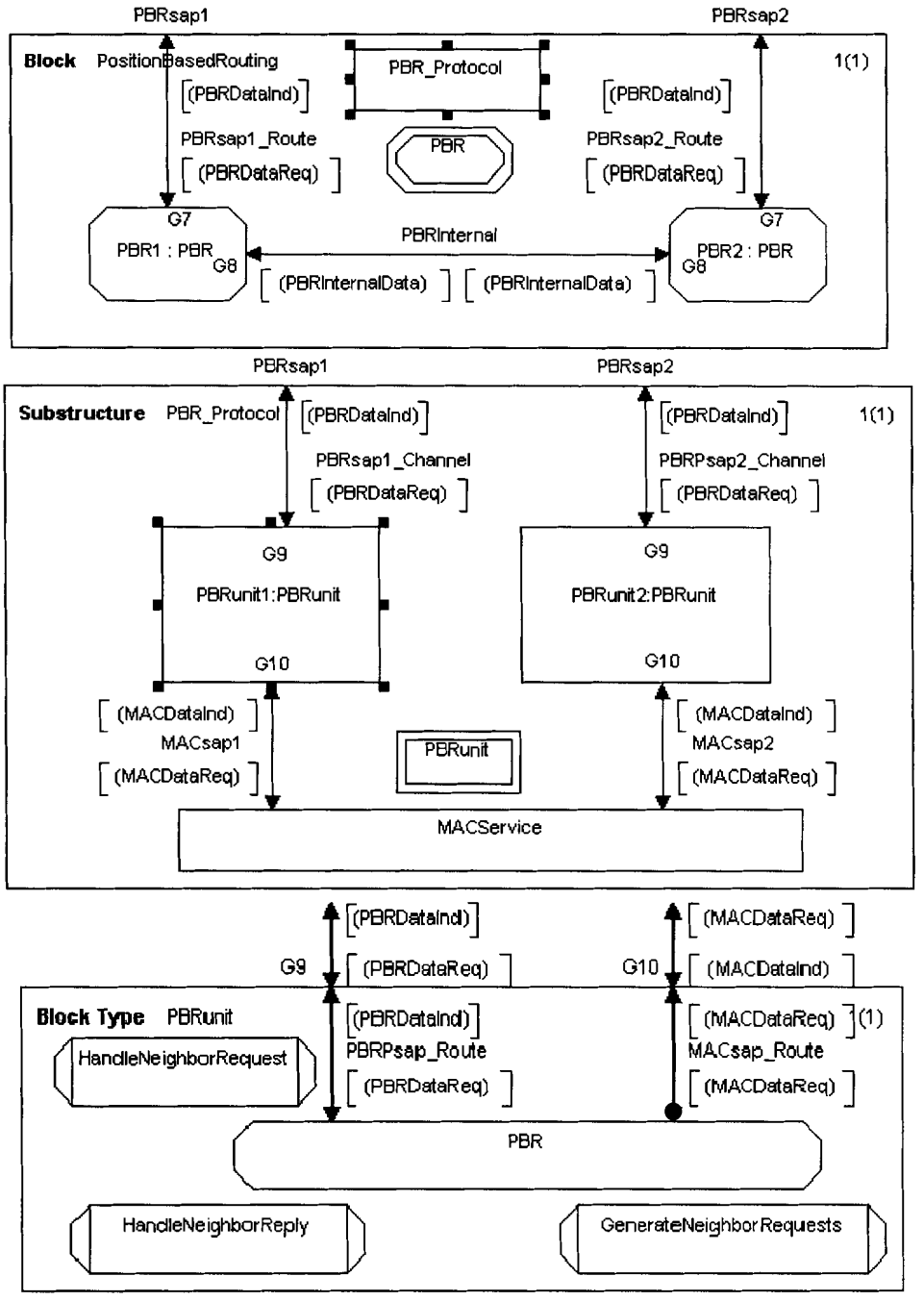

Figure 16 Position based routing

5.2.1 Position Based Routing

The PBR protocol logically splits into two separate parts according to two basically different tasks performed by this protocol. Section 5.2.2 specifies the protocol for the detection of nearest neighbors. Section 5.2.3 briefly outlines the routing of packets. For brevity, certain details of the SDL specification, e.g., signal and variable declarations and type definitions, are omitted here. However, this should not affect the conceptual understanding of the system architecture.

5.2.2 Detection of Nearest Neighbors

Nodes periodically update the information about their neighbours as the position of the nodes change dynamically and also because neighbours may become unreachable. The frequency of updates depends on various parameters, for instance, such as the relative speed of the neighbours and the node itself, and on the probability of node crashes. We therefore introduce a timer, called *update_timer*, that triggers the start of detection cycles. Similarly, a node periodically updates its position information as specified through signals of type *NodePosition* generated by some location system the node is connected to through an external interface (see Figure 16).

The detection of neighbours is a time critical operation as node positions may change over time and responses from neighbours may be subject to arbitrary delays (e.g., caused by retransmissions at lower layers and the fact that a neighbour may not always react instantaneously). A timeout event *cycle_timer* controls the termination of detection cycles so that late responses can be ignored. Every neighbour request thus carries a *time stamp* encoded into the data part.

Intermediate results from processing neighbour replies need to be stored temporarily until a detection cycle is fully completed. A local array variable *new_neighbour* is used for storing intermediate nearest neighbours. On completion of a detection cycle the result becomes effective by copying the node references of *new_neighbour* to *neighbour*, which then is used to refer to the neighbours of a node. However, it may happen that for some sector all the neighbours are temporarily unreachable. In this situation the nearest neighbour then remains undefined until a neighbour reply is received in a subsequent detection cycle.

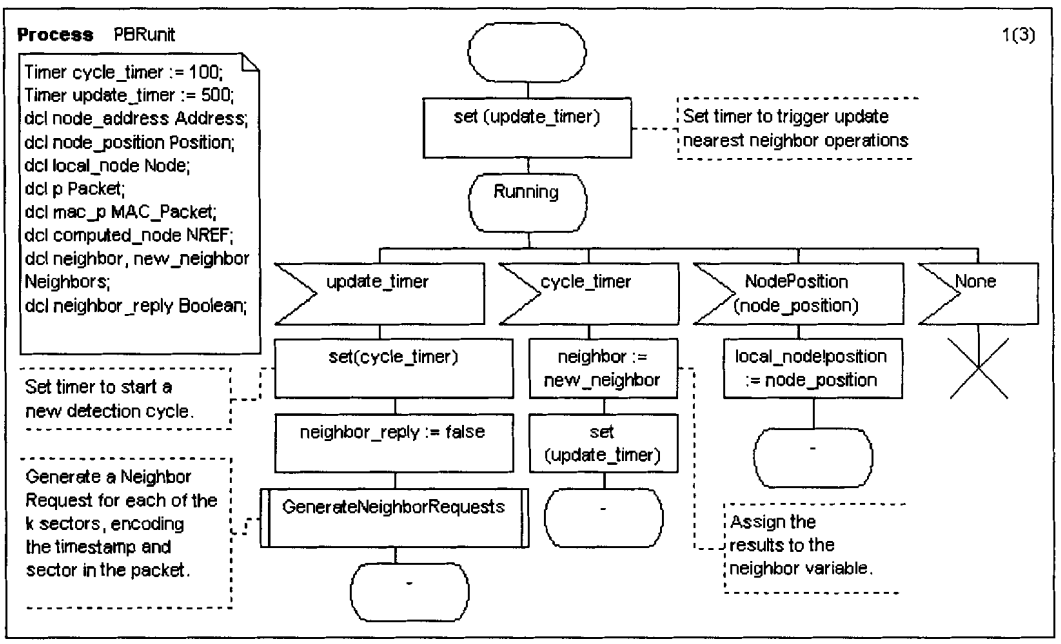

Figure 17 Neighbour detection cycle and packet forwarding

5.2.3 Packet Routing

When receiving a packet from the MAC layer, the position based routing decides whether the packet can be delivered locally, or it needs to be forwarded to a remote destination. Delivering a packet locally means that the packet is handed over to the DLS

running on the local node. Forwarding a packet means to send it to the nearest neighbour within the sector that matches with the position of the final destination node.

Depending on the position of the packet destination relative to the position of the local node, a nearest neighbour for the next hop of this packet is computed by means of a function *ComputeNeighbour* operating on position coordinates. The result yields the node to which the packet is then forwarded after updating the receiver information in the packet header (see Figure 17).

Data packets, location requests, and location replies which are received from the local DLS will be forwarded to the specified receiver node using the service rendered by the MAC layer (see Figure 18).

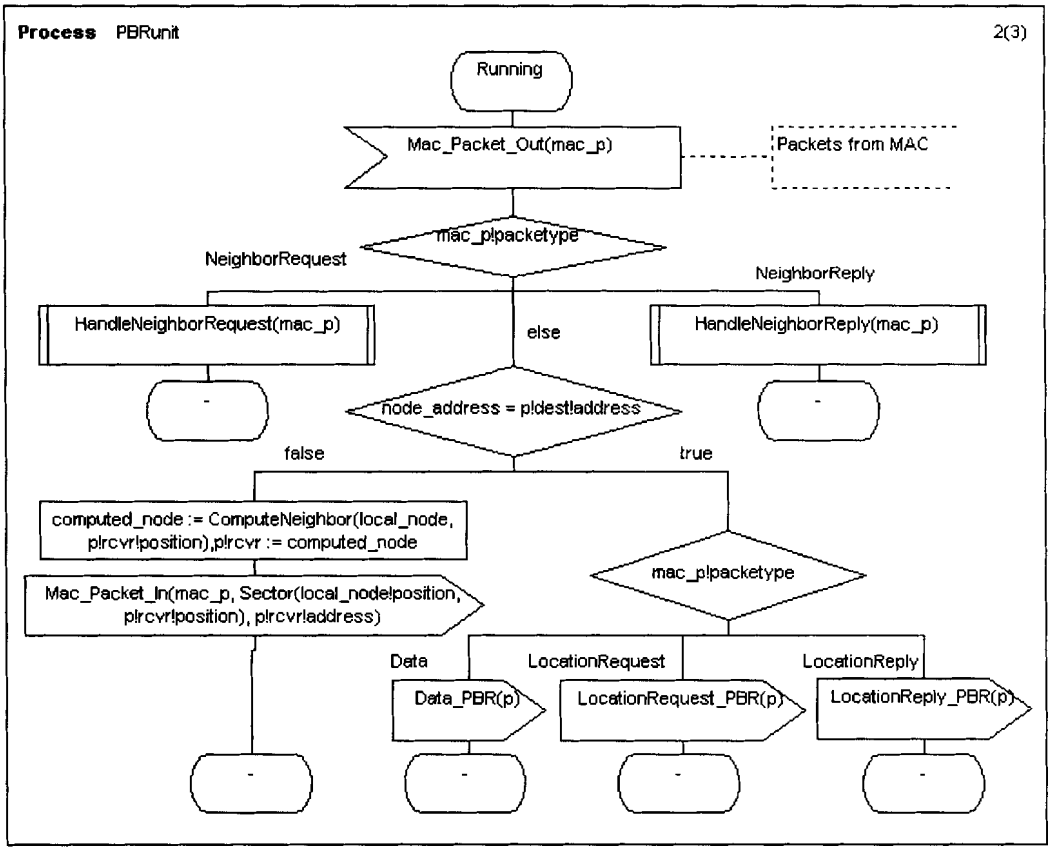

Figure 18 Neighbour detection cycle and packet forwarding (ctd).

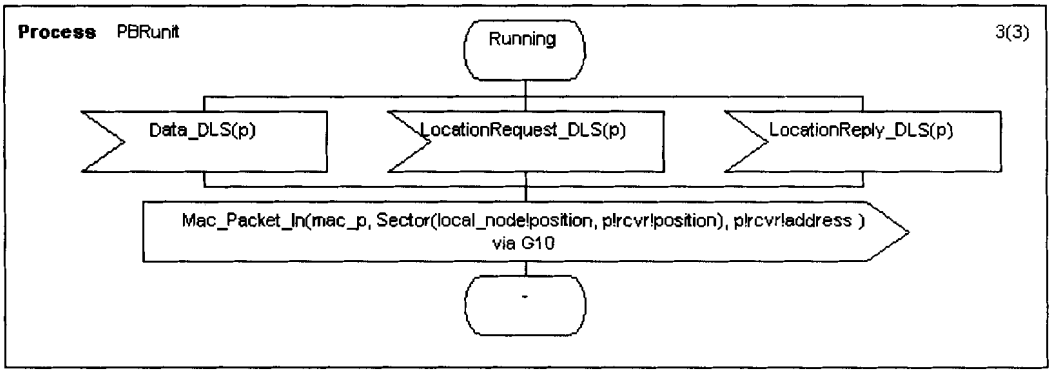

Figure 19 Neighbour detection cycle and packet forwarding (ctd).

5.2.4 Distributed Location Service

Consider some sequence of consecutive packets that the DLS receives, one by one, from the transport layer. The first packet of every such sequence requires special treatment. This packet needs to be stored locally at the DLS to first determine the position of the destination node by means of a location request. Accordingly, we assume that the first packet always can be recognized as such (see Figure 19).

A location reply to the DLS always matches a pending location request (related to a waiting first packet). The position information of the sender of the location reply is added to this first packet, which then eventually will be sent via the PBR service (see Figure 20).

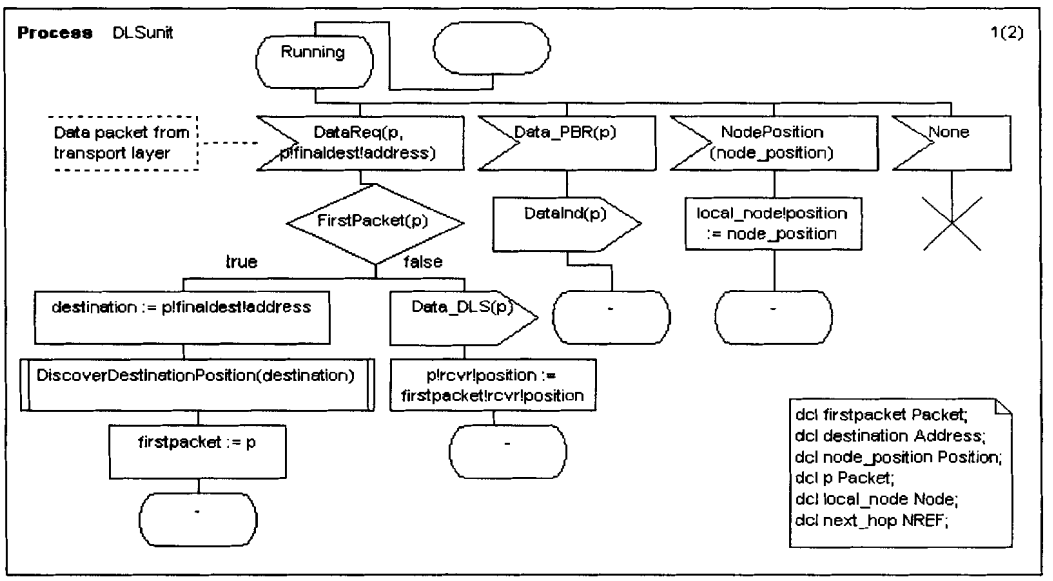

Figure 20 Distributed location service

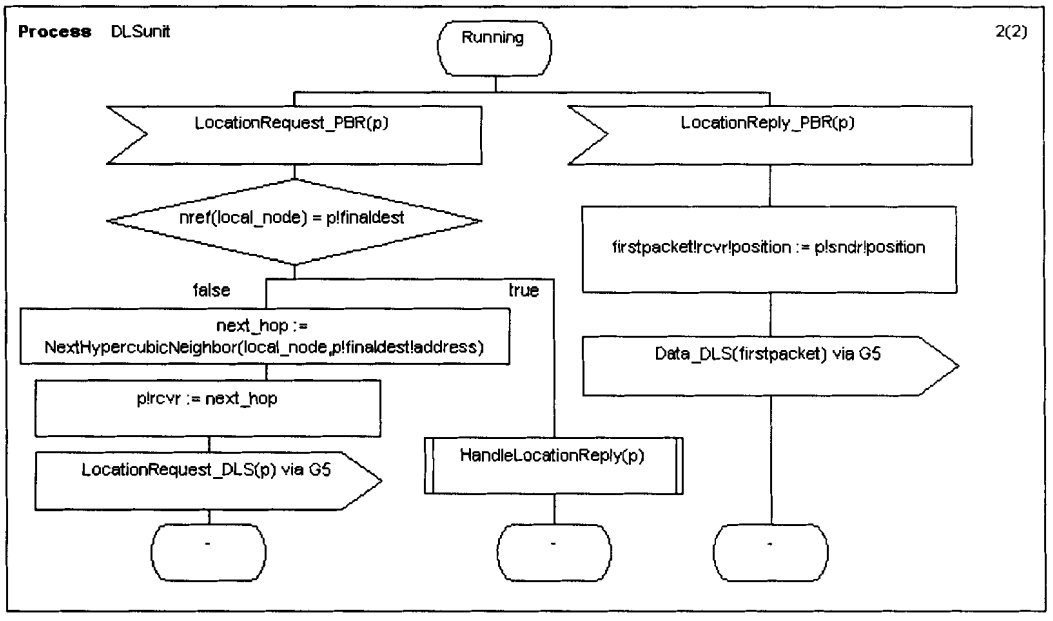

Figure 21 Distributed location service (ctd.)

When receiving a location request (see Figure 20), the DLS checks the address of the final destination, which is encoded in the discovery packet as *finaldest*.[1] If this address does not match with the local address, the DLS calculates the next hypercubic neighbor by means of a function *NextHypercubicNeighbor* and then forwards the request accordingly. Otherwise, if the local node matches with the final destination, the DLS generates a location reply that contains the requested position information (procedure *HandleLocationReply* in Figure 21).

[1] Note that the *final* destination may be different from the destination specified in the packet header, since the latter always refers to the next hop on the hypercube path.

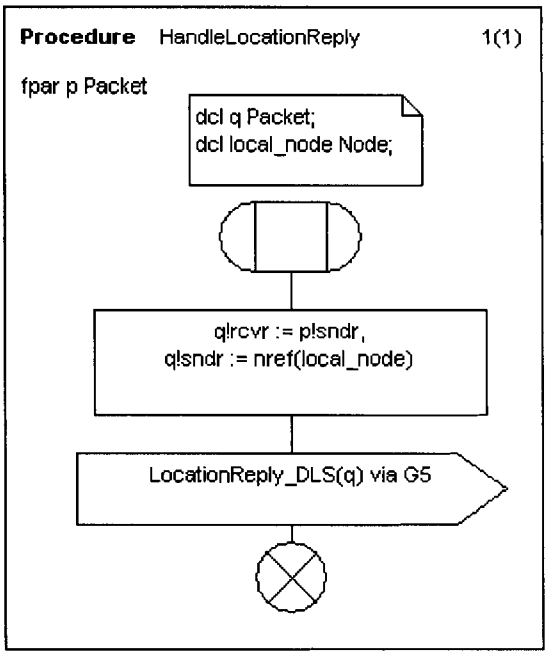

Figure 22 HandleLocationReply procedure

Not included is the reorganization of the dynamic hypercube as required for the elimination and insertion of nodes in order to ensure fault tolerance of the distributed location service. Finally, we do not describe here the operations performed when a node is switched on or switched off.

6 DISCUSSION AND CONCLUSION

A superior alternative to the routing strategy proposed by A. Benczur and T. Lukovski [1] is the distributed logical topology based location service (LTLS) protocol using the k-dim ALT data structure proposed by U. Glässer and Q-P. Gu in [14]. "With the k-dim ALT data structure, the LTLS protocol reaches the same fault tolerance property as that of the hypercube location service but is more efficient with smaller time delay, routing table, and number of administrative packets." [14]

We present here a DASM model of the network layer protocol for geographic ad hoc routing based on the hypercube location service defined in [1]. Conceptually, the network layer splits into two separate sublayers, one for the location service and one for the position based routing. The DASM model specifies the relevant algorithmic aspects with a degree of detail and precision that goes far beyond the informal, pseudocode-like description of fundamental operational aspects presented in Section 4. Based on this model, we have developed a multilayer communication architecture for geographic routing in mobile ad hoc networks in terms of an SDL system. This SDL system serves as a formal requirements specification and as a high-level executable model for analysis and experimental validation of the key system properties of HLS using commercial SDL tool environments like Cinderella SDL [4].

The abstract operational view of SDL allows us to represent a relatively complex protocol in a concise and coherent way. Our experiences with SDL are quite promising for designing complex communications software. Beyond basic event-handling and interface mechanisms, the language effectively supports hierarchical design and layering as required to cope with complexity when engineering distributed communication protocols with timing constrains.

Building on the purely theoretical model in [1], basic algorithmic aspects of the protocol are modelled in the form of an abstract state machine. The SDL paradigm of modelling complex protocols is particular appropriate for practical purposes, and even helped us to identify design flaws that remained hidden in the other formalisms.

REFERENCE LIST

[1] A. Benczúr and T. Lukovszki. A Degree *O(log log n)* Fault Tolerant Distributed Location Service for Geographic Ad-Hoc Routing. Technical Report, TR-RI-02-231, Heinz Nixdorf Institute, Paderborn, Germany, June 2002

[2] A. Benczur, U. Glässer and T. Lukovszki. Formal Description of a Distributed Location Service for Ad Hoc Mobile Networks. In E. Börger, A. Gargantini, E. Riccobene (Eds.): Abstract State Machines - Advances in Theory and Applications, vol. 2589 of LNCS, Springer Verlag, 2003

[3] A.C. Yao. On Constructing Minimum Spanning Trees in k-Dimensional Spaces and Related Problems. SIAM Journal on Comp., 11(4): 721-736, 1982

[4] Cinderella home page: *www.cinderella.dk*

[5] D. E. Comer. Internetworking with TCP/IP, *Principles, Protocols, and Architectures*. Prentice Hall, 2000

[6] Foundations of Software Engineering Group at Microsoft, the website, *http://research.microsoft.com/fse*

[7] H. Zhou, A Survey on Routing Protocols in MANETs, Department of Computer Science and Engineering, Michigan State University, Technical Report: MSU-CSE-03-08.

[8] J. Ellsberger, D. Hogrefe and A, Sarma. SDL – Formal Object-oriented Language for Communicating Systems. Prentice Hall Europe, 1997

[9] Jeannette M. Wing, A Specifier's Introduction to Formal Methods. Carnegie Mellon University

[10] M. Abolhasan, T. Wysocki, E. Dutkiewicz. A Review of Routing Protocols for Mobile Ad Hoc Networks.

[11] T. Leighton. Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes. Morgan Kaufmann, 1992.

[12] U. Glässer, R. Gotzhein, and A. Prinz. The Formal Semantics of SDL-2000: Status and Perspectives. Computer Networks, to appear.

[13] U. Glässer, Y.Gurevich and M. Veanes. An Abstract Communication Model. Foundations of Software Engineering, Microsoft Research, Microsoft Corporation, Technical Report, MSR-TR-2002-55, May 2002.

[14] U. Glässer, Qian-Ping Gu. Formal Description and Analysis of a Distributed Location Service for Mobile Ad Hoc Networks. SFU-CMPT-TR 2003-12

[15] X.-Y. Li and P.-J. Wan and Y. Wang. Power Efficient and Sparse Spanner for Wireless Ad Hoc Networks. IEEE International Conference on Computer Communications and Networks (ICCCN'01), 2001.

[16] Y. Gurevich. Evolving Algebras 1993: Lipari Guide. In E. Borger (Ed.), Specification and Validation Methods, Oxford University Press, 1995, 9-36