Verification of Group Address Registration Protocol using PROMELA and SPIN

Tadashi Nakatani*
Fuchu Works, Toshiba Corporation, Japan

February 16, 1997

Abstract

This paper demonstrates robustness and effectiveness of Group Address Registration Protocol (GARP) by successful use of SPIN(3). GARP is a datalink-level protocol for dynamically joining and leaving multicast groups on a bridged LAN. It is currently in the process of standardization as a part of IEEE P802.1p(1). SPIN is used to verify a model of multicast-base LAN described in PROMELA(3). The protocol has been proved to be free from incorrectness such as unspecified receptions and deadlocks. It has also been proved that there is no violation of a temporal claim for membership consistency. SPIN is found to be very powerful and easy to use without much knowledge of the formal verification. GARP is well modeled in PROMELA though it depends heavily on timers and multicasting which are not directly supported by PROMELA.

1 Introduction

The multicast protocol has attracted much attention with recent advent of an interactive multi-media communication via a internet, such as web TV. The Group Addresss Registrattion Protocol(GARP) has been proposed to realize multicast interactive multimmedia communications across bridged LANs. The GARP is included in a proposed draft of IEEE P802.1p, 'Standard for Local and Metropolitan Area Networks–Supplement to Media Access Control (MAC) Bridge: Traffic Class Expediting and Dynamic Multicast Filtering'.

By ssuccessful use of multicast group membership information carried by GARP, bridges can expedite delivery of time critical traffic and limit the extent

*This work was performed while the autor was a visitor at Stanford University, USA

of high bandwidth multicast traffic within a bridged LAN.

This paper applies SPIN verication tool to GARP protocol on a multicast-base LAN such as Ethernet and demonstrates robustnesss and effectiveness of the protocol in dynamically joining and leaving multicast groups on a bridged LAN. The protocol correctness and a temporal claim are proved with several modification of the model in PROMELA.

This paper is organized as follows. In section 2, an overview of GARP specification is presented. In section 3, a model of the protocol which is described in PROMELA is introduced. In section 4, the verification process and the result is shown. And section 5 concludes this paper.

2 GARP Specification

2.1 Purpose

GARP allows MAC service users to dynamically register (and subsequently, de-register) Multicast group membership information with the bridges attached to the same LAN segment, and for that information to be disseminated across all bridges in the bridged LAN. multicast group membership information indicates that one or more members of a group exist, along with the MAC address associated with the group.

2.2 Protocol Entity Components

The GARP protocol entity involves the following component functions.
a)**Applicant** function: associated with each end station in a bridged LAN, which allows MAC service user(s) in the end station to dynamically register/de-register multicast group membership.
b)**Regsistrar** function: associated with each port of a MAC bridge, which respond to requests for registration/de-registration.
c)**Proxy Applicant** function: associated with each port of a MAC bridge, which acts on behalf of all Applicants reachable via the other ports of the bridge in order to propagate their registration/de-registration requirements across the Spanning Tree.
d)**Leave All** function: associated with each port of a MAC bridge, which performs a garbage collection function that ensures that all registered information that is no longer in use is removed from the filtering database.

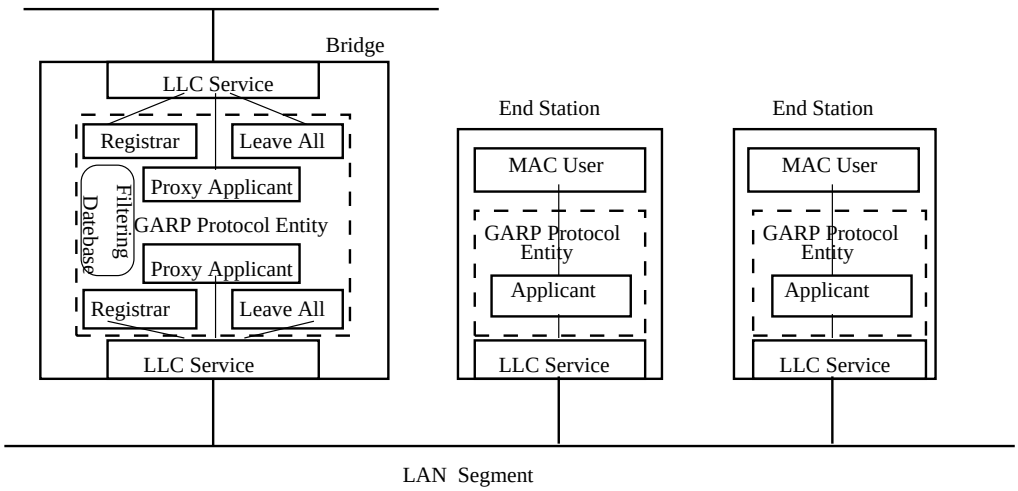

Figure 1: GARP Protocol Entity Components

2.3 Overview of Protocol Operation

GARP protocol exchanges take place between GARP protocol entities which comunicate by means of LLC type 1 services.

2.3.1 Initial membership registration

An Applicant state machine participates in GARP protocol exchanges if a MAC service user in that Applicant station signals the intent to join a Group. Registration as a Group member is achieved by the Applicant state machine sending a GARP Join PDU to the GARP address. The Applicant will then wait for a time period, JoinTime, to see whether any other Applicant on the segment registers membership of the same Group(s); if so, the Applicant takes no further action, if not, the Applicant sends a further Join PDU. In either case, the Applicant considers itself to have joined the Group(s) after issuing the first Join PDU.

2.3.2 Registrar response to registration

Any Registrar in receipt of a GARP Join PDU responds to it by updating the contents of its Filtering Database in a manner consistent with the information received. If the Join PDU indicates specific Group memberships which are currently not represented in the Filtering Database, the necessary entries are

created. If the information received represents an updating of existing entries in the Filtering Database, the entries are updated appropriately.

2.3.3 Applicant initiated de-registration

De-registration as a Group member occurs when the MAC service user in an Applicant station signals the intent to leave a Group. De-registration is achieved by the Applicant state machine sending a GARP Leave PDU to the GARP address. At that point, the Applicant state machine considers that, from its own point of view, registration has terminated. Any Applicant that see a Leave PDU that specifies a Group of which they wish to remain a member, respond by sending a Join PDU for that Grouup. Their subsequent behavior is as for initial registration.

2.3.4 Registrar response to de-registration

Any Registrar that receives a GARP Leave PDU for a Group that it considers to be registered on its Port, will wait for a time period, LeaveTime, to see whether any Applicant re-registers membership of the Group. If a Join PDU is seen, the Registrar considers the Group to be registered, and takes no further action. If another Leave PDU for the same Group is seen during LeaveTime, the timer is re-started with its initial value. On timer expiry, the Registrar issues a Leave PDU for the group concerned, and waits for a further LeaveTime. If the group is again not re-registered, the Registrar considers the Group to have been de-registered, and removes the relevant Port information from the Filtering Database entry.

2.3.5 Bridge initiated de-registration

At regular time intervals, the LeaveAllTime, the Leave All state machine attempts to force de-registration for all MAC addresses currently registered in its filtering database for a given Port, by issuing a GARP Leave PDU. Any Applicant stations that do not wish particular de-registrations to occur respond in the manner indicated in Applicant initiated de-registration. This procedure has the effect of confirming registrations that are still alive, and garbage-collecting any registrations for which there are no longer any active participants on the LAN. The subsequent behavior of the Registrar state machine is as for Applicant initiated de-registration.

3 Modeling

3.1 Model Systems

The system configured with 1 Bridge, 2 End Stations, and 1 LAN Segment was selected as a simple but enough general Basic Model System.

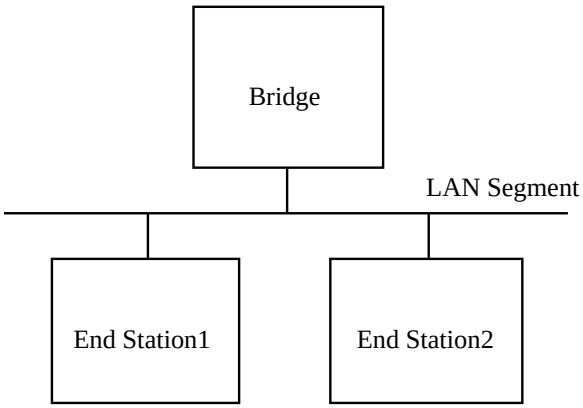

Figure 2: Basic Model System

3.2 Modeling of Functions

The following functions were modeled one function by one process.

(End Station) Applicant*, LLC Service, MAC Service User
(Bridge) Proxy Applicant*, Registrar*, Leave All*, LLC Service

Functions marked with '*' are the components of GARP protocol entity, the others are a tester(MAC Service User) and environment(LLC Service).

The following points were especially considered when processes were designed.

1)Message Queues
GARP PDUs are multicasted via a shared access LAN like Ethernet. It seems reasonable to model packet inputs to LLC Service and packet outputs from LLC Service with synchronous message queues(6). Output queues from LLC Service Process is designed to have size(1) to model receiver station's buffer while input

queues to LLC Service Process are rendezvous.

2)Lossy LLC Service Process
To model packet losses 'on the wire', LLC Servic Process forwards a message from a sender process to 0-2 receiver process(es) undeterminently.

3)Timeouts
(Proxy) Applicant, Registrar, and Leave All state machines have timers. As PROMELA does not have a way to describe quantitative timer, 'empty' of message queues is used to represent timeouts(7). And timer-flags which represents timer is running or stopping are introduced.

4)End labels
To identify deadlocks correctly, end labels are attached to the message waiting states of (Proxy) Applicant, Registrar, and LLC Service processes.

5)Unspecified receptions
To identify unsigned receptions correctly, 'assertion(0)'s are placed where unspecified messages may be received.

6)Unbalanced MAC Service User Processes
To suppress the number of states as well as to maintain enough generality, two MAC Service User Processes don't have to be symmetrical.

All processes described in PROMELA are provided in Appendix A.

4 Verification

4.1 Running Environment

H/W: Pentiam-120, Mem > 100MB
S/W: Linux 2.0.24, SPIN 2.9.3, XSPIN 2.9.1

4.2 State Properties

First, the state properties, that is, the absenses of unspecified receptions, unreached codes and deadlocks was proved.

The first run was performed at supertrace mode modeling non-message-overflow, that is, message send operation is executable only if the target channel is non-full.
Soon SPIN detected a deadlock and showed the senario to the deadlock and the final states of each process.

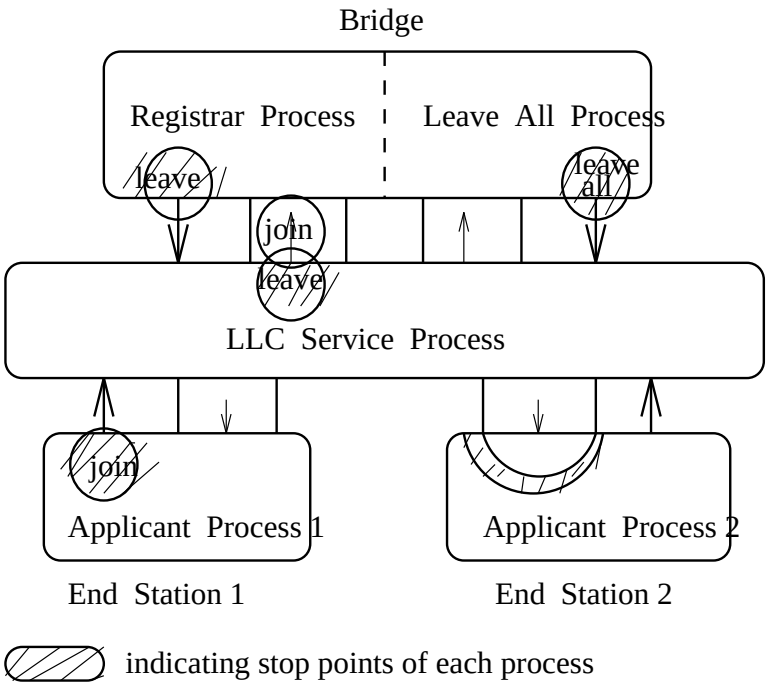

Figure 3: Deadlock in Non-Message-Overflow-Model

The reason of this deadlock is LLC Service Process and Registrar Process are trying to send messages simultaneously when both channels between the processes are full. In a real system, bridges and end stations have enough buffers and packets are rarely dropped. But it can happen theoretically. So the model was modified to Message-Overflow-Model.
In Message-Overflow-Model, messages can be discarded only when messages try to get into full queues from LLC Service Process. (The other queues are still synchronous because they are configured as rendezvous queues.)

In this time, SPIN found no assertion violations, no deadlocks, and no unreached codes. But the state coverage was not enough(hash factor = 5.25).

To reduce complexity without losing generality, the verification was performed for the following two models seperately.

1)1 Bridge(1 Registrar), 2 End Stations(2 Mac Service Users, 2 Applicants),

1 LAN Segment(1 LLC Service)
2)1 Bridge(1 Registrar, 1 Leave All), 1 End Stations(1 Mac Service User, 1 Applicant), 1 LAN Segment(1 LLC Service)

In both cases, SPIN proved that **the protocol has no assertion violations(i.e. unspecified receptions) and no deadlocks** with acceptable state coverages (hash factors: 14.4, 213) In the case 1, **all codes were reached.**

4.3 Temporal Claim

The most important requirement of GARP is that there are no membership inconsistencies between Bridges and End Stations.
The membership consistency means
If at least one member exists in a LAN, the Bridge must recognize it as soon as possible (at least eventually)

The claim was expressed as follows using the temporal logic.

[](p→<>([]q))
p: (MAC User Process1 ends)
q: (Registrar Process state is not OUT)
where Process1 was modified to only send ReqJoin message once.

Then this claim was transformed to 'Never Claim' of PROMELA using XSPIN. Before verifying this claim, Leave All Process which performs garbage collection periodically was removed from the model. Considering the above requirement 'as soon as possible', we should not depend on it at first. Also LLC Service was modified so as not to lose packets 'on the wire' because it is obvious there are cases that violates the claim without the modification. (Queue overflows modeling buffer overflows of receivers are still possible)

SPIN found a violation of the claim but it's in an unrealistic senario.
The senario: Applicant1 retrieves messages from the queue so late and causes message losses.(The senario is presented in Appendix B of this paper as a exapmle of Sequence Message Chart.)

Increase the number of messages which the queues from LLC Service Process can buffer. (1→3)

SPIN proved **there is no violation of the temporal claim** with high coverage (hash factor = 64.2).

5 Conclusion

It has been proved with SPIN that GARP is free from incorrectness such as unspecified reception, deadlocks, and unreached codes. Besides it was also proved there is no violation of a temporal claim for membership consistency.

SPIN was found to be very powerful and easy to use without much knowledge of the formal verification. GARP was well modeled in PROMELA though it depends heavily on timers and multicasting which are not directly supported by PROMELA.

GARP is designed to be used in a multi-segment bridged LAN though this paper verified the single-segment LAN model. Therefore the future work is to verify a multi-segment LAN model. Moreover the protocol is being changed significantly. So it is also necessary to verify the new version of GARP.

Acknowledgments

I wish to thank Professor David Cheriton of Stanford University who let me work on the protocol. I am grateful to G.J.Holzmann and other contributers to SPIN who made my first experience of the formal verification a pleasant one.

References

(1) IEEE P802.1p Standard for Local and Metropolitan Area Networks - Supplement to Media Access Control (MAC) Bridges: Traffic Class Expediting and Dynamic Multicast Filtering (Draft4), September 1996

(2) D.R.Cheriton, S.E.Deering, and K.J.Duda, Ethernet Group Membership Protocol (EGMP) Draft RFC, October 1995

(3) G.J.Holzmann, Design and Validation of Computer Protocols, Prentice Hall, 1991

(4) G.J.Holzmann, Basic Spin Manual, Technical Report, AT&T, Bell Laboratories, 1994

(5) G.J.Holzmann, What's new in SPIN Version 2.0, Technical Report, AT&T, Bell Laboratories, August 1996

(6) H.E.Jensen, K.G.Larsen, and A.Skou, Modeling and Analysis of a Collision

Avoidance Protocol using SPIN and UPPAL, In Proceedings of SPIN96

(7) P.Kars, Formal Methods group, Experience using Spin and Promela in the Design of a Storm Surge Barrier control System, In Proceedings of SPIN95

A GARP Model in PROMELA

```
------------ for the Basic Model -----------------------

/*
 * PROMELA Validation Model
 * GARP(Global Definitions)
 */

#define N_APL 2 /* number of applicants */
#define N_RGS 1 /* number of registrar */
#define N_LVAL 1 /* number of leaveall */

#define true 1
#define false 0

#define out 1
#define lanx 2
#define in 3
#define vanx 4

#define out_reg 1
#define awt_rjin 2
#define lv_imm 3
#define in_reg 4


mtype = {
reqjoin, reqleave,
join, leave, leaveall
}

chan user_to_appl[N_APL] = [0] of { byte };
chan appl_to_llc[N_APL] = [0] of { byte };
chan llc_to_appl[N_APL] = [1] of { byte };
chan regist_to_llc[N_RGS] = [0] of { byte };
chan llc_to_regist[N_RGS] = [1] of { byte };
```

```
chan leaveall_to_llc[N_LVAL] = [0] of { byte };
chan llc_to_leaveall[N_LVAL] = [1] of { byte };

byte r_state;


/*
 * PROMELA Validation Model
 * GARP(main)
 */

#include "defines"
#include "macuser"
#include "macuser1"
#include "llc"
#include "applicant"
#include "registrar"
#include "leaveall"

init
{ atomic {
  run macuser(0); run macuser1(1);
  run llc();
  run applicant(0); run applicant(1);
  run registrar(0);
  run leaveallpro(0)
}
}


/*
 * PROMELA Validation Model
 * GARP(MAC Service User)
 */

proctype macuser(byte n)
{
do
:: user_to_appl[n]!reqjoin
:: user_to_appl[n]!reqleave
:: break
od
}
```

```
/*
 * PROMELA Validation Model
 * GARP(MAC Service User 1)
 */

proctype macuser1(byte n)
{
if
:: user_to_appl[n]!reqjoin
:: user_to_appl[n]!reqleave
:: skip
fi
}


/*
 * PROMELA Validation Model
 * GARP(LLC Service)
 */

proctype llc()
{ byte type;

endIDLE:
do
:: atomic
   { appl_to_llc[0]?type ->
if
:: llc_to_appl[1]!type; llc_to_regist[0]!type
:: llc_to_appl[1]!type
:: llc_to_regist[0]!type
:: skip /* lose message */
fi
   }
:: atomic
   { appl_to_llc[1]?type ->
if
:: llc_to_appl[0]!type; llc_to_regist[0]!type
:: llc_to_appl[0]!type
:: llc_to_regist[0]!type
:: skip /* lose message */
fi
   }
```

```
:: atomic
   { regist_to_llc[0]?type ->
if
:: llc_to_appl[0]!type; llc_to_appl[1]!type
:: llc_to_appl[0]!type
:: llc_to_appl[1]!type
:: skip /* lose message */
fi
   }
:: atomic
   { leaveall_to_llc[0]?type ->
if
:: llc_to_appl[0]!type; llc_to_appl[1]!type
:: llc_to_appl[0]!type
:: llc_to_appl[1]!type
:: skip /* lose message */
fi
   }
od
}


/*
 * PROMELLA Validation Model
 * GARP(Applicant)
 */

proctype applicant(byte n)
{ bool jointimer;
byte type, state;

state = out;
endIDLE:
do
:: user_to_appl[n]?type -> /* event from macuser */
if
:: (type == reqjoin) ->
if
:: (state == out) ->
jointimer = true;
appl_to_llc[n]!join;
state = lanx
:: (state == lanx) /* ignore */
:: (state == in) /* ignore */
```

```
:: (state == vanx) /* ignore */
:: else -> assert(0) /* protocol violation */
fi
:: (type == reqleave) ->
if
:: (state == out) /* ignore */
:: (state == lanx) ->
jointimer = false;
appl_to_llc[n]!leave;
state = out
:: (state == in) ->
appl_to_llc[n]!leave;
state = out
:: (state == vanx) ->
jointimer = false;
appl_to_llc[n]!leave;
state = out
:: else -> assert(0) /* protocol violation */
fi
:: else /* ignore */
fi

:: llc_to_appl[n]?type-> /* event from llc */
if
:: (type == join) ->
if
:: (state == out) /* ignore */
:: (state == lanx) ->
jointimer = false;
state = in
:: (state == in) /* ignore */
:: (state == vanx) ->
jointimer = true;
state = lanx
:: else -> assert(0) /* protocol violation */
fi
:: (type == leave) || (type == leaveall) ->
if
:: (state == out) /* ignore */
:: (state == lanx) ->
jointimer = true;
state = vanx
:: (state == in) ->
jointimer =true;
```

```
state = vanx
:: (state == vanx) ->
jointimer = true;
:: else -> assert(0) /* protocol violation */
fi
:: else /* ignore */
fi

:: empty(user_to_appl[n]) && empty(llc_to_appl[n]) &&
(jointimer == true) -> /* jointimer expired */
if
:: (state == lanx) ->
jointimer = false;
appl_to_llc[n]!join;
state = in
:: (state == vanx) ->
jointimer = false;
appl_to_llc[n]!join;
state = in
:: else -> assert(0) /* protocol violation */
fi

od
}


/*
 * PROMELLA Validation Model
 * GARP(Registrar)
 */

proctype registrar(byte n)
{ bool leavetimer, member_exist;
byte type;

r_state = out_reg;
endIDLE:
do
:: llc_to_regist[n]?type -> /* event from llc */
if
:: (type == join) ->
if
:: (r_state == out_reg) ->
member_exist = true;
```

```
r_state = in_reg
:: (r_state == awt_rjin) ->
leavetimer = false;
r_state = in_reg
:: (r_state == lv_imm) ->
leavetimer = false;
r_state = in_reg
:: (r_state == in_reg) /* ignore */
:: else -> assert(0) /* protocol violation */
fi
:: (type == leave) || (type == leaveall) ->
if
:: (r_state == out_reg) /* ignore */
:: (r_state == awt_rjin) ->
leavetimer = true
:: (r_state == lv_imm) ->
leavetimer = true
:: (r_state == in_reg) ->
leavetimer = true;
r_state = awt_rjin
:: else -> assert(0) /* protocol violation */
fi
:: else /* ignore */
fi

:: empty(llc_to_regist[n]) &&
(leavetimer == true) -> /* leavetimer expired */
if
:: (r_state == awt_rjin) ->
regist_to_llc[n]!leave;
r_state = lv_imm
:: (r_state == lv_imm) ->
leavetimer = false;
member_exist = false;
r_state = out_reg
:: else -> assert(0) /* protocol violation */
fi
od
}


/*
 * PROMELLA Validation Model
 * GARP(Leave All)
```

```
 */

proctype leaveallpro(byte n)
{ bool leavealltimer;
byte type, state;

leavealltimer = true; /* leavealltimer on */

do
:: llc_to_leaveall[n]?type /* ignore */
:: empty(llc_to_leaveall[n]) &&
(leavealltimer == true) ->/* leavealltimer expired */
leaveall_to_llc[n]!leaveall
od
}



------------ for the temporal claim -----------------------
/*
 * PROMELA Validation Model
 * GARP(main)
 */
#include "defines"
#include "macuser1"
#include "macuser2"
#include "llcnoloss"
#include "applicant"
#include "registrar"
#include "leaveall"
byte pid;

init
{ atomic {

  pid =(run macuser1(0));
  run macuser2(1);
  run applicant(0);
  run applicant(1);
  run llcnoloss();
  run registrar(0);
}
}
```

```
/*
 * Formula As Typed: [](p-><>([]q))
 * The Never Claim Below Corresponds
 * To The Negated Formula !([](p-><>([]q)))
 * (formalizing violations of the original)
 */
#define p (macuser1[pid]@user1_end)
#define q (r_state != out_reg)

never {    /* !([](p-><>([]q))) */
T0_init:
if
:: (1) -> goto T0_init
:: ((p)) -> goto T0_S3
:: (! ((q)) && (p)) -> goto accept_S3
fi;
accept_S3:
if
:: (1) -> goto T0_S3
fi;
T0_S3:
if
:: (1) -> goto T0_S3
:: (! ((q))) -> goto accept_S3
fi;
accept_all:
skip
}

/*
 * PROMELA Validation Model
 * GARP(MAC Service User1)
 */
proctype macuser1(byte n)
{
atomic
{
user_to_appl[n]!reqjoin;
user1_end:
skip
}
}
```

```
/*
 * PROMELA Validation Model
 * GARP(MAC Service User2)
 */
proctype macuser2(byte n)
{
if
:: user_to_appl[n]!reqjoin
:: user_to_appl[n]!reqleave
:: skip
fi;
if
:: user_to_appl[n]!reqjoin
:: user_to_appl[n]!reqleave
:: skip
fi
}


/*
 * PROMELA Validation Model
 * GARP(LLC Service no loss)
 */
proctype llcnoloss()
{ byte type;

endIDLE:
do
:: appl_to_llc[0]?type ->
llc_to_appl[1]!type; llc_to_regist[0]!type
:: appl_to_llc[1]?type ->
llc_to_appl[0]!type; llc_to_regist[0]!type
:: regist_to_llc[0]?type ->
llc_to_appl[0]!type; llc_to_appl[1]!type
od
}
```

B Exapmle of Message Sequence Chart