# SDL 개발환경을 이용한 IMT-2000 무선 프로토콜의 호스트 시뮬레이터 설계 및 구현

송 평 중[†]·노 철 우[††]·노 문 환[†††]·윤 영 배[††††]·이 종 필[†††††]


## 요 약

공인된 통신 언어인 SDL을 이용하여 시스템을 설계하고, SDL 개발 도구인 SDT를 이용하여 실행가능한 소프트웨어를 개발하고 실제 타겟 시스템과 동일한 환경에서 개발된 기능을 검증하는 개발 순기를 모두 수용하는 시뮬레이터의 개발이 대규모의 복잡한 통신 프로토콜에서는 필요하다. 본 논문에서는 IMT-2000 무선 프로토콜의 한 엔티티인 계층 2의 무선 링크제어 프로토콜 엔티티 RLC-AM(Acknowledged mode)의 개발 및 검증을 위하여, RLC-AM을 설계 및 구현한 후 이의 실행화일을 호스트 상에서 수행 시킬 수 있는 호스트 시뮬레이터를 개발하였다. 호스트 컴퓨터상에서 SDT 개발환경, 컴파일 환경 및 운영체제 환경을 연동하여 개발된 호스트 시뮬레이터는 대상 프로토콜 엔터티를 MAC등 타 엔터티로 바꾸어서도 같은 방식으로 수행할 수 있다.


# Design and Implementation of Host Simulator for IMT-2000 Wireless Protocol using SDL Development Environments

Pyung-Joong Song[†] · Cheul-Woo Ro[††] · Moon-Whan Nho[†††] · Yung-Bae Yoon[††††] · Jong-Phil Lee[†††††]


## ABSTRACT

A simulator which is well suited to be the core of full scale projects such as design of system by using SDL which is a well known language defined by ITU-T, development of the executable software using SDT (SDL Design Tool), and validation of the developed functions in the host environment is essential for the development of large and complex communication protocols. After design and implementation of RLC_AM which is an entity of IMT-2000 wireless protocol under SDL development environment, an executable file is generated. In this paper, a host simulator which can run this executable file on the host computer is developed. The host simulator is generated by integrating the SDT, compile, and operating system development environment. This simulator can run the another protocol entity like MAC instead of RLC_AM in the same way.


**키워드 : SDL, SDT, RLC, IMT-2000, 호스트 시뮬레이터(host simulator), 무선 프로토콜(wirless protocol)**

## 1. 서 론

검증 과정없이 프로토콜[1]을 개발하게 되면 통신호처리 자체가 불안할 수 있어 기본 통신기능뿐만 아니라 장치간 연동시험도 매우 어려워지며 프로토콜 오류가 규격에 내재하고 있는 경우는 전체 개발 및 시험 일정이 많이 소요되고 초기 시스템 개발보다는 소프트웨어의 유지 보수 및 관리에 더 많은 비용이 투자된다. 이와 같은 이유로 시스템의 개발 및 검증을 정형화된 수단을 통해 자동적으로 수행할 수 있는 노력이 세계 도처에서 활발히 진행되고 있다. 그 사례로써, 설계서와 구현된 프로그램 사이의 엄격한 일관성을 보장받을 수 있는 ITU-T의 SDL[2-5]을 이용하여 시스템을 설계하고, 국제적으로 공인된 개발 도구(SDT : SDL Design Tool)[6-8]를 이용하여 실행 가능한 소프트웨어를 개발하고 실제 타겟 시스템과 동일한 환경에서 개발된 기능을 검증하는 개발 순기를 모두 수용하는 시뮬레이터 개발을 들 수 있다. 이로써 시스템 규격, 설계 및 개발 단계에서 나타나는 여러 유형의 오류를 제거한 검증된 소스 프로그램의 생성은 물론, 각종의 시험 시나리오를 자동 생성하여 장치 내부의 계층간 정합시험 및 장치간의 연동시험에 신속하고 효과적으로 대처하는 수단을 제공한다. 이 같은 프로토콜의 설계, 구현 및 검증의 자동화 수단은 표준 규격의 잦은 갱신, 새로운 기능의 빈번한 추가 그리고 전문 인력의 이직 등에 따라 그 필요성이 날로 더해 가고 있으며 특히, 무선 프로토콜의 경우에는 더욱 그렇다.

† 정 회 원 : 한국전자통신연구원 이동 프로토콜 연구팀장
†† 정 회 원 : 신라대학교 컴퓨터정보공학부 교수
††† 준 회 원 : 한국전자통신연구원 이동프로토콜연구팀 연구원
†††† 정 회 원 : 델슨전자 IMT-2000 센타
††††† 준 회 원 : 충남대학교 대학원 컴퓨터공학과
논문접수 : 2000년 9월 15일, 심사완료 : 2001년 1월 8일

본 논문에서는 IMT-2000 무선 프로토콜[9]의 한 엔티티인 계층 2의 무선 링크제어 프로토콜 엔티티 확인응답형 모드(Radio Link Control-Acknowledged Mode : RLC-AM)[10]의 개발 및 검증을 위하여, RLC-AM을 설계 및 구현한 후 이의 실행 파일을 호스트 상에서 수행시킬 수 있는 호스트 시뮬레이터를 개발하였다. 호스트 컴퓨터 상에서 SDT 개발환경, 컴파일 환경 및 운영체제 환경을 연동하여 개발된 호스트 시뮬레이터는 대상 프로토콜 엔티티를 MAC등 타 엔티타로 바꾸어서도 같은 방식으로 수행할 수 있다. 2장에서는 SDL 개발환경 하에서의 RLC-AM 프로토콜 설계에 대하여 언급하고 3장에서는 호스트 시뮬레이터를 개발할 수 있는 통합(integration)의 개념 설명에 이어 호스트 시뮬레이터의 구성 및 소켓 통신을 이용한 설계 및 구현과 그 동작 절차에 대해 기술한다.

## 2. 무선 프로토콜의 설계

본 논문에서 언급되는 무선 프로토콜은 유럽방식의 IMT-2000인 UMTS [9]의 프로토콜을 의미한다. UMTS는 3GPP 그룹에서 현재 표준화가 진행중인 국제표준을 바탕으로 개발중인 비동기 시스템이며, GSM 기반의 네트웍 기술과 CDMA 기반의 액세스 기술로 구성된다. 무선 프로토콜 엔터티중 확인응답 모드 서비스를 갖는 계층 2 엔터티인 RLC-AM 블록을 설계 및 검증대상으로 하여 이를 위한 시뮬레이터의 설계에 대하여 언급한다. 프로토콜 설계는 3GPP/무선 프로토콜 설계 규격서[10, 11]를 기본으로 하여 수행하였다.

### 2.1 RLC-AM 프로토콜 분석

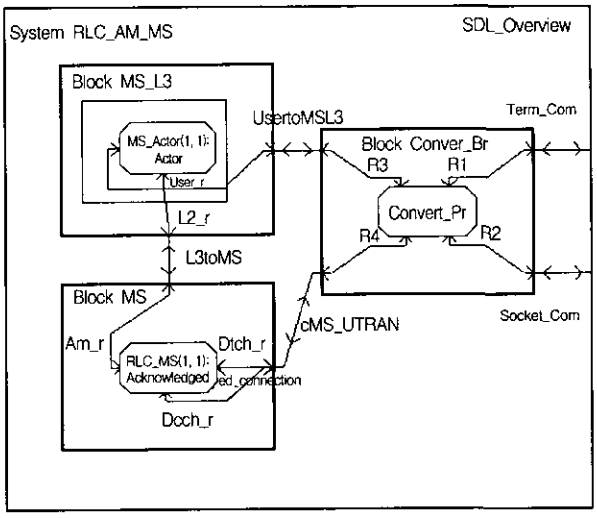
RLC는 정보의 분할 및 조합, 연결제어, 연쇄화, 오류 제어, 상위계층 PDU의 순서적인 전달, 중복검출, 흐름제어, 프로토콜 오류 검사 및 복구 등의 기능을 수행한다. 이중 RLC의 핵심 기능인 오류 제어는 정보전송단위인 프레임마다 고유의 일련번호를 추가하여 오류가 발생한 프레임만 재전송하는 선택적-ARQ 방식을 이용하며, 무선환경의 오류 레벨에 따라 프레임 길이를 가변시키는 세그멘테이션 기능이 이용된다. RLC-AM모드에서 일반 정보는 AMD PDU에 실려 가며, 이에 대한 확인응답 및 상태정보는 STATUS PDU에 실려간다. 이외에도 전송효율을 향상시키기 위하여 피기백 메커니즘과 패딩을 채택하고 있다[10].

### 2.2 RLC-AM 프로토콜 설계

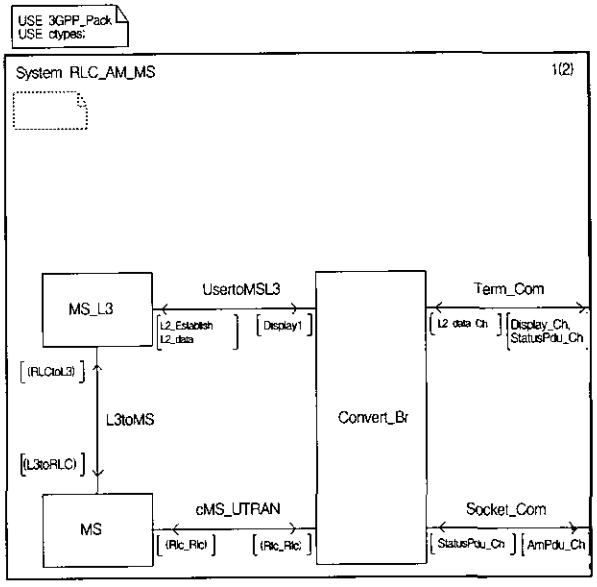
RLC-AM 프로토콜은 SDL 개발 방법론[6]에 따라 시스템, 블록 및 프로세스 레벨의 top-down 방식으로 설계하였다.

#### 2.2.1 시스템 설계

UMTS의 구성요소인 MS와 UTRAN사이의 레디오 인터페이스 개발기능 확인을 주 목적으로 하여 본 논문에서는 MS와 UTRAN 두 시스템간 메시지의 전송유무를 확인할 수 있는 호스트 시뮬레이터를 개발하고자 한다. RLC-AM 프로토콜 MS측의 시스템 구성은 (그림 1)과 같다. 시스템은 개발 및 검증대상인 RLC-AM 프로토콜이 탑재될 MS 블록과 상위계층인 MS_L3, 그리고 사용자 인터페이스를 위한 Convert_Br 블록으로 구성된다. 블록간의 통신경로인 채널과 프로세스사이의 통신경로인 라우터에 대하여 (그림 1)에서 명시하고 있다.




(a) 시스템 개요




(b) 시스템 레벨 설계구조

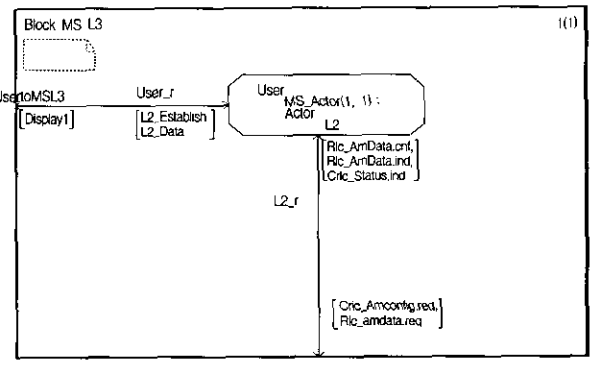
(그림 1) RLC-AM의 MS 개요와 시스템 레벨 설계구조

#### 2.2.2 블록 설계

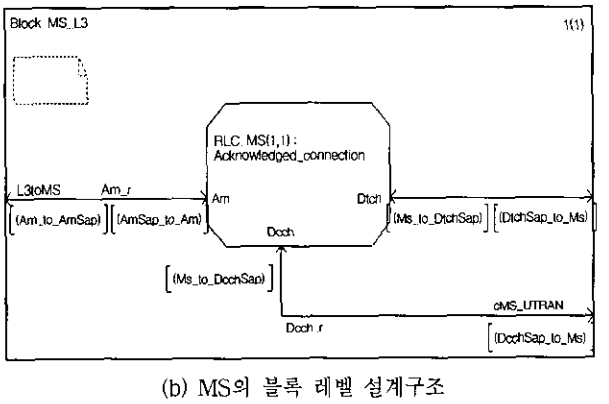
MS_L3와 MS 블록은 (그림 2)의 3GPP_Pack에서 프로세스 타입[5]으로 정의된 Actor와 Ack_Con(Acknowledged _connection)의 인스탄스인 MS_Actor(1,1)와 RLC_MS(1,1)

로 구성된다.

(그림 2-a)에서 MS_Actor(1,1)는 3GPP 계층 3의 기능을 단순화시켜 나타낸 것이며, User_r 신호 라우터를 통해 형상정보를 송수신하며, L2_r을 통해 MS 블록과 정보를 송수신한다. MS_Actor(1,1)에서 MS 블록으로 전송되는 정보는 Crlc_Amconfig.req와 Rlc_Amdata.req 시그널을 통해 전송된다. (그림 2-b)에서 RLC_MS(1, 1)은 RLC-AM 프로토콜의 기능을 나타낸 것이며, Am_r의 신호 라우터를 통해 상위 계층과 통신하며, Dtch_r, Dcch_r을 통해 피어 엔티티와 정보를 송수신한다. 프로세스 타입을 채택한 주된 이유는 SDL 명세의 편리함, 재사용성과 동적인 프로세스의 생성을 지원하기 위함이다.




(a) MS_L3의 블록 레벨 설계 구조

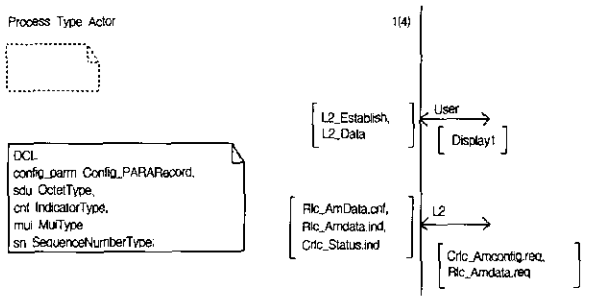



(b) MS의 블록 레벨 설계구조

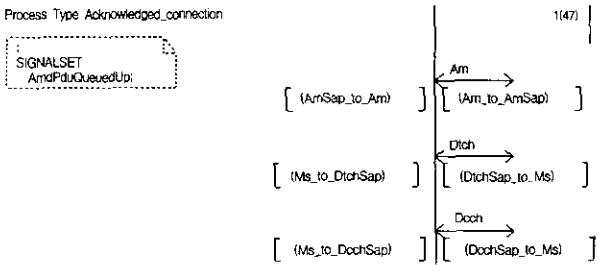
(그림 2) 프로세스 타입을 이용한 블록 레벨 설계구조

2.2.3 프로세스 설계

프로세스는 프로토콜의 실제 동작을 명세하는 부분이며, RLC의 확인응답 모드에서는 MS와 UTRAN측에서 동일한 동작이 수행되므로 객체-지향 설계 기법인 프로세스 타입을 이용하여 프로세스를 정의한다. (그림 3-a)는 프로세스 타입 Actor를 정의하기 위해 필요한 변수의 선언(DCL부분)과 게이트 선언(User, L2), 그리고 게이트를 통한 시그널 정의를 나타낸 것이고 (그림 3-b)는 프로세스 타입 Ack_Con의 정의 중 일부를 나타낸 것이다. Dtch, Dcch는 게이트를, Dtch_r, Dcch_r은 라우터를 표시한 것이다. UTRAN 측도 MS측과 동일한 방식으로 시스템, 블록, 프로세스를 정의하며 이들 두 시스템에 대한 상세한 설계는 [12]를 참조한다.




(a) Actor의 프로

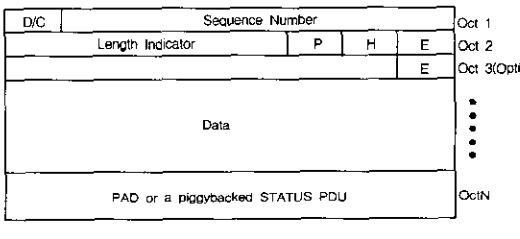



(b) Ack_Con의 프로세스 레벨 설계구조

(그림 3) 프로세스 타입을 이용한 프로세스 레벨 설계구조

2.2.4 PDU 자료구조

확인응답 모드 서비스를 위한 데이터 PDU의 명세서에 의한 정의는 (그림 4-a)이고, SDL 표기법을 사용한 구조체의 설계는 (그림 4-b)이다.




(a) PDU 정의

```
NEWTYPE AmPdu STRUCT
    snofsdu        SequenceNumber Type ;
    vt_dat         SequenceNumber Type ;
    dc             SequenceNumber Type ;
    p              Bit ;
    he             Bit_String ;
    li             Bit_String ;
    e              Bit ;
    sli            SequenceNumber Type ;
    se             Bit ;
    data           Octet_String ;
ENDNEWTYPE ;
```

(b) PDU의 SDL 설계

(그림 4) PDU 자료구조

## 3. 시뮬레이터 설계 및 구현

시뮬레이터는 호스트 시뮬레이터와 타겟 시뮬레이터를

light 통합과 tight 통합에 의해 각각 만들 수 있으며 타겟 시뮬레이터는 특정 타겟 시뮬레이터 라이브러리 및 타켓 O.S와의 통합에 의해서 생성되고 특정 대상 타겟에서 수행된다. 반면 호스트 시뮬레이터는 SDT에 의해서 생성된 C 코드와 시뮬레이터 라이브러리를 함께 컴파일 및 링크를 하여 light 통합에 의하여 생성된다.

### 3.1 통 합

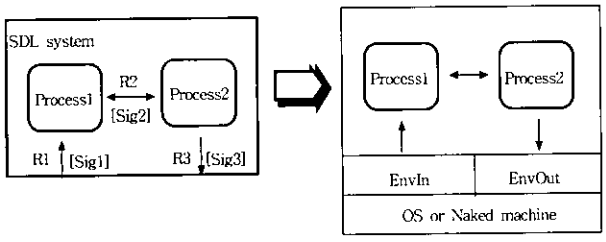
코드 생성기에 의하여 생성된 소스코드는 플랫폼에서 사용될 수 있는 라이브러리와 함께 컴파일 및 링크되는 통합에 의해 실행 가능한 파일이 생성된다. light 통합은 PC나 워크스테이션의 운영체제인 UNIX나 Windows NT에서 실행될 수 있는 응용 시스템을 실행 화일로 만들고 tight 통합은 특정 타겟에서 실행될 수 있는 응용 시스템을 RTOS와 통합에 의해서 실행 파일로 만들어 진다. C 코드생성기는 각 SDL 심벌에 대응되는 매크로를 포함하는 코드를 생성하며 이들 매크로는 light 통합시 SDT 커널의 함수 호출과 tight 통합시의 RTOS 함수 호출을 포함한다. 즉 light 통합은 SDT 커널의 프리미티브와 스케쥴러를 사용하는 반면 tight 통합은 타겟 RTOS가 갖고있는 프리미티브를 사용하게 된다[7, 8].

#### 3.1.1 Light 통합

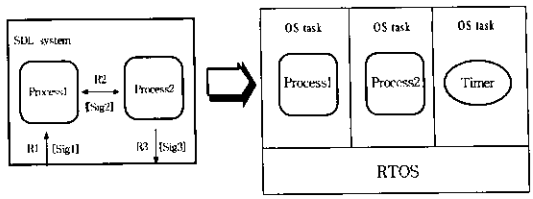
SDT 커널과 함께 컴파일 및 링크되는 통합에 의해 UNIX나 Windows NT에서 실행 가능한 SDL 응용 시스템은 SDL 프로세스가 한번에 한 천이만을 수행하게끔 해주는 스케쥴러에 의하여 실행된다. SDL 프로세스 사이에서 전송된 시그널은 수신 SDL 프로세스의 입력 큐에 바로 입력되며 SDL 시스템에서 외부로의 시그널 송수신은 사용자가 작성하게끔 되어있는 주변함수(environment function)에 의해서 처리된다. 한번에 하나의 SDL 프로세스만이 수행될 수 있다. 즉 선점이 허용되지 않으며 펜딩된 프로세스는 그 다음 천이에서 바로 수행된다(그림 5-a).

#### 3.1.2 Tight 통합

SDL 프로세스 인스턴스들은 RTOS의 타스크로 맵핑되며 SDL 시그널 전송은 RTOS 타스크 사이의 메시지 전송으로 맵핑된다. 스케쥴링은 RTOS에 의해 수행되며 environment는 하나의 RTOS 타스크로 주어진다. 하나의 SDL 프로세스 인스턴스는 하나의 RTOS 타스크로 맵핑되며 스




(a) Light 통합 모델




(b) Tight 통합 모델

(그림 5) 통합 모델

케쥴링은 RTOS 타스크 우선순위로 처리되고 선점과 time slicing이 사용된다. 즉 여러 개의 SDL 프로세스가 동시에 실행될 수 있다(그림 5-b)[6].

### 3.2 호스트 시뮬레이터 설계 및 구현

#### 3.2.1 주변함수(Environment function)

응용 대상별 호스트 시뮬레이터의 개발을 위해서는 SDT 커널 및 관련 인터페이스와의 컴파일 및 링크에 대한 light 통합뿐만 아니라 주변함수의 코딩이 필요하다. C 코드 생성기를 가지고 대상 시스템의 시뮬레이터를 만들기 위해서는 SDL 시스템, 시스템의 물리적인 주변(O.S, 파일 시스템, 하드웨어, 컴퓨터 망 등), 그리고 이들 둘의 연결에 필요한 주변함수를 고려해야 한다

주변함수를 이용해서 대상 시스템에 시그널을 보내는 것 외에 파일 읽기, 망으로의 메시지 송수신, 인터럽트 핸들링, 소켓이나 하드웨어의 포트 정보 가져오기 등의 일을 할 수 있다. 이렇게 SDL 시스템과 주변 함수를 분리함으로써 문제 복잡도의 감소와 테스트의 효율성 및 동시성의 장점을 가진다.

주변 함수를 얻을 수 있는 방법으로 SDT에서 제공하는 sctenv.c 파일을 복사하여 사용하는 것과 SDT의 organizer의 make 메뉴에서 'Generate environment function' 옵션을 선택하여 *시스템 파일명*_env.c 라는 주변함수 파일이 자동으로 생성되게 할 수 있다. 여기서 설계된 SDL 시스템과 실제 호스트 상에서 통신을 위해 소켓을 이용하였는데 GUI (Graphic User Interface)와 env.c 파일의 연동을 위해 소켓 통신에 필요한 프로그램을 개발하였다. 주변함수는 다음 네 가지 기본 기능함수로 구성되어 있다.

① xInitEnv는 SDL 시스템을 시작할 때마다 호출된다. 소켓 통신을 위해 소켓 초기화 및 호스트의 정보나 IP 주소를 가져오는 일을 한다.

② xCloseEnv는 SDL_Halt 함수를 호출한다. SDL 시스템 종료시 호출된다.

③ GUI에서 SDL 시스템으로의 시그널 전송은 xInEnv를 역으로의 전송은 xOutEnv를 이용한다. GUI에서 소켓을 이용해 입력된 시그널은 xGetSignal 함수에 의해 StatusPdu 시그널이 가지고 있는 메세지 값을 함수

xAss_SDL_ Charstring에 의해 SDT에서 인식할 수 있는 타입으로 할당하고 SDL_Output 함수에 의해 SDL 시스템으로 시그널이 가지고 있는 메세지의 값과 함께 입력된다. xGetSignal 함수는 처음 메세지에 의해 명세된 타입의 시그널 인스턴스를 나타내는 데이터 영역에 대한 포인터를 반환한다(그림 6-a).

④ xOutEnv는 추출된 AmPdu_Sig 시그널의 메세지 값을 send_data( ) 함수에 의해 MS에서UTRAN측으로 소켓 전송이 가능하게 되고, Control_ Send( ) 함수를 이용하여 MS측의 GUI로 Amd Pdu_Sig의 메세지 값을 전송한다(그림 6-b).

```
/*-------------------------------------------------------
xInEnv extern
-------------------------------------------------------*/
#ifndef XNOPROTO
extern void
xInEnv(SDL_Time Time_for_next_event)
#else
extern void
xInEnv(Time_for_next_event)
 SDL_Time Time_for_next_event ;
#endif

{
 char          *Instr ;
 int           ctlcount, msgcount, NrOfReadChars ;
 char          SignalName='₩0' ;
 xSignalNode   yOutputSignal, *S ;
 int           i=0 ;
 int           datacount ;

#ifndef XMONITOR
   if(SignalName=='S') {
    /*--------SEND THE DATAINTO THE SDL SYSTEM--------*/
    /* SDL-signal StatusPdu */
        sprintf(tmpbuffer, "Signal To System : StatusPdu('%s')₩n", Instr+1) ;
        Output(tmpbuffer) ;
    yOutputSignal=xGetSignal(StatusPdu, xNotDefPId, xEnv) ;
    scanf(Instr+1, "%n", &NrOfReadChars) ;
    xAss_SDL_Charstring(&((yPDP_StatusPdu)(OUTSIGNAL_DATA_PTR))->Param1,
                        (Instr+NrOfReadChars+1), XASS_MR_ASS_NF) ;
   SDL_Output(yOutoutSignal, (xIdNode *)0) ;
```

(a) xInEnv

```
*-------------------------------------------------------
   xOutEnv extern
-------------------------------------------------------*/
#ifndef XNOPROTO
extern void
xOutEnv(xSignalNode  *S)
#else
extern void
xOutEnv(S)
  xSignalNode  *S ;
#endif
{
  char Outstr[150] ;
  /* SDL-signal AmPdu_Sig */
  if((*S)->NameNode==AmPdu_Sig)
  {
   sprintf(Outstr, "A%s",((yPDP_AmPdu_Sig)((*S)))->Param1) ;
   sprintf(tmpbuffer, "Signal From System : AmPdu_Sig()(*S)))->Param1+1) ;
   output(tmpbuffer) ;

   /* Send the data out the message socket */
   send-data(Outstr) ;

#ifndef XMONITOR
        Control_Send((char *)(((yPDP_AmPdu_Sig)((*S)))->Param1)+i) ;
#endif
  xReleasesSignal(S) ;

  return ;
}
```

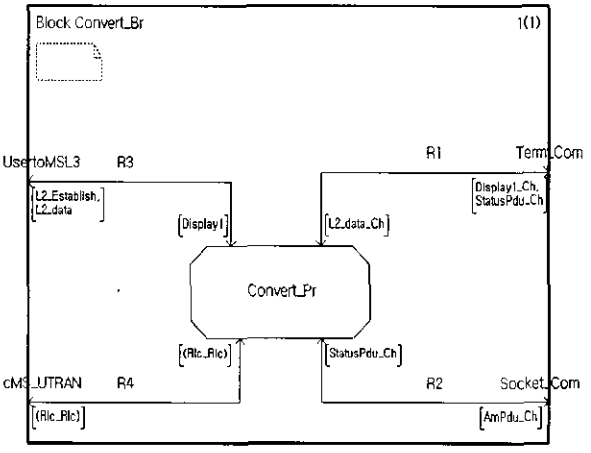
(b) xOutEnv

(그림 6) 주변함수

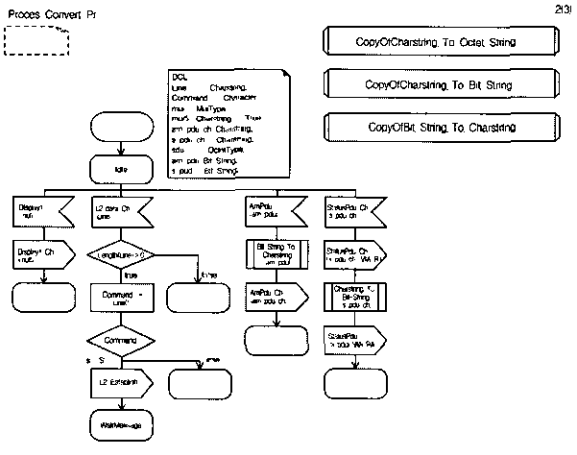
xInEnv와 xOutEnv에서는 사용자의 요구사항 정보를 추출할 수 있도록 시그널을 선정하고 그 시그널이 가지고 있는 메세지 값을 GUI나 다른 SDL 시스템으로 전송하는 역할을 한다.

3.2.2 타입변환 함수의 설계 및 구현

대상 시스템의 SDL 설계에 이어 사용자가 시뮬레이션 수행시 결과를 주변함수에서 쉽게 확인할 수 있도록 해주는 타입변환 함수의 설계 및 구현이 필요하다. 즉 비트 타입을 문자(character) 타입으로 변환하여 사용자가 쉽게 볼 수 있도록 하고 SDT 내에서는 비트 연산을 위하여 다시 문자 타입을 비트 타입으로 변환하는 기능이 필요하다. 이를 위하여 다음 Convert_Br 블록과 Convert_Pr 프로세스를 설계 및 구현하였다(그림 7-(a,b)).




(a) Convert_Br 블록 레벨 설계구조




(b) Convert_Pr 프로세스 레벨 설계구조

(그림 7) 타입변환 블록과 프로세스 레벨 설계구조

Convert_Br 블록의 Convert_Pr 프로세스에는 명세서에서 메시지의 입력 변수에 대한 형 타입을 SDT의 옥텟 타입으로 명세하고 있기 때문에 통합에 의해 생성되는 호스트 시뮬레이터의 메시지 입력 변수 타입인 스트링 타입과는 일

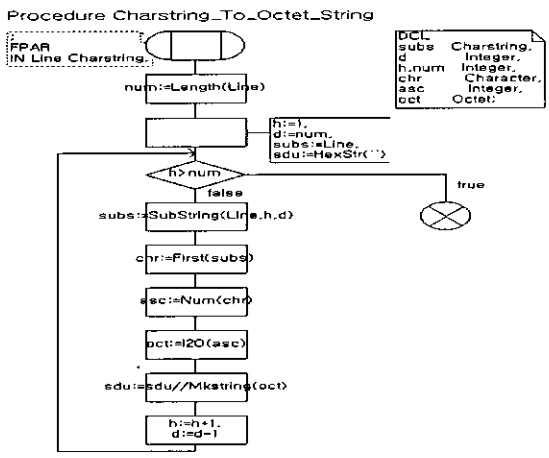
치하지 않는다. 따라서 매개 변수의 입력 및 전달을 위해 SDT가 내장하고 있는 연산자들을 사용하여 문자스트링 타입을 SDT의 옥텟스트링 타입으로 변환시켜 매개변수를 전달하게 되고 출력을 위해 다시 옥텟스트링 타입을 문자스트링 타입으로 변환하기 위한 프로시져들로 구성된다.

(그림 8-a)는 문자스트링 타입의 변수 값을 SDT의 옥텟스트링 타입으로 변환하기 위한 SDL 설계이다. 연산자 Length를 사용하여 입력받은 문자스트링의 길이를 반환하고 연산자 SubString과 First를 사용하여 첫 문자부터 하나씩 떼어내고 문자를 정수로 바꾸는 연산자 NUM을 사용하여 하나의 문자를 아스키코드 값의 정수로 변환하고 그 정수를 연산자 I2O를 사용하여 옥텟으로 변환한다. 연산자 옥텟을 옥텟스트링으로 변환하는 연산자 Mkstring을 사용하고 연산자 //(Concatenation)를 사용하여 옥텟스트링으로 선언된 임의의 변수에 변환 될 문자가 없을 때까지 변환된 값을 연결시킨다. 그래서 옥텟스트링 타입의 변수 sdu에 문자스트링 타입의 변수 값이 옥텟스트링으로 변환되어 반환된다.

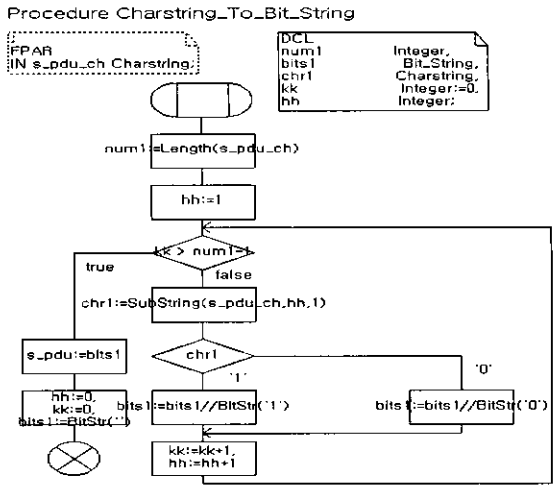
(그림 8-b)는 문자스트링 타입의 변수 값을 SDT의 비트스트링 타입으로 변환하기 위한 SDL 설계이다. 문자스트링의 입력 값을 '0'과 '1'의 비트스트링으로 변환하기 위해 연산자 BitStr를 사용한다.

(그림 8-c)는 비트스트링의 입력 값을 한 비트씩 읽어 '0'일 경우 문자스트링 변수에 '0'을, '1'인 경우엔 문자스트링의 변수에 '1'을 연결하여 반복한 후 '0'과 '1'로 구성된 문자스트링으로 변환된다.

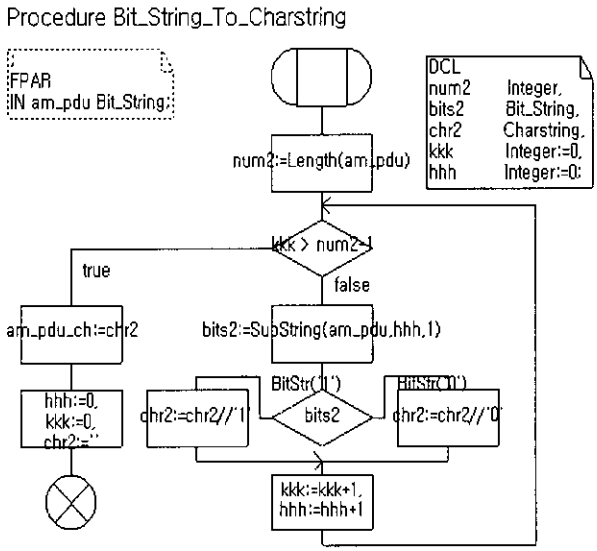
(그림 8-d)는 옥텟스트링을 문자스트링으로 변환하기 위함인데, 옥텟을 정수로 변환시켜 주는 연산자 O2I를 사용하기 위해 연산자 SubString과 First를 사용하여 옥텟스트링을 옥텟으로 한자씩 변환하고 이 옥텟으로 변환된 아스키코드 값의 정수를 문자로 변환하기 위해 연산자 Chr를 사용하였고 연산자 //를 사용해서 변환된 모든 문자를 연결시킨다.



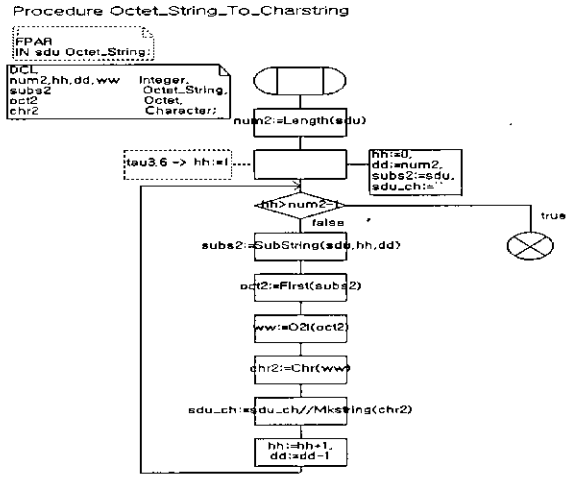

(a) Charstring_To_Octet_String




(b) Charstring_To_Bit_String




(c) Bit_String_To_Charstring

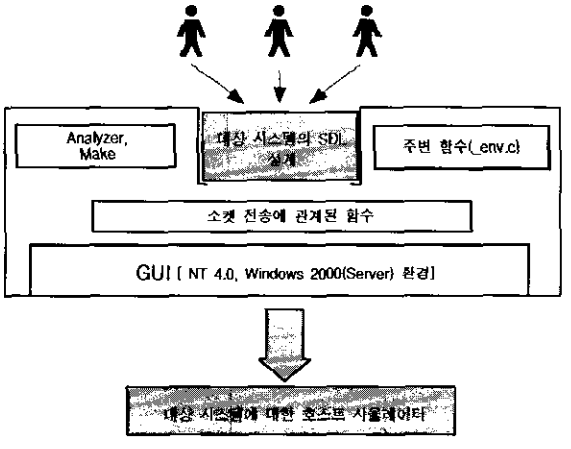



(d) Octet_String_To_CharString

(그림 8) 타입변환 프로시져 설계 구조

3.2.3 호스트 시뮬레이터 구성 및 구현

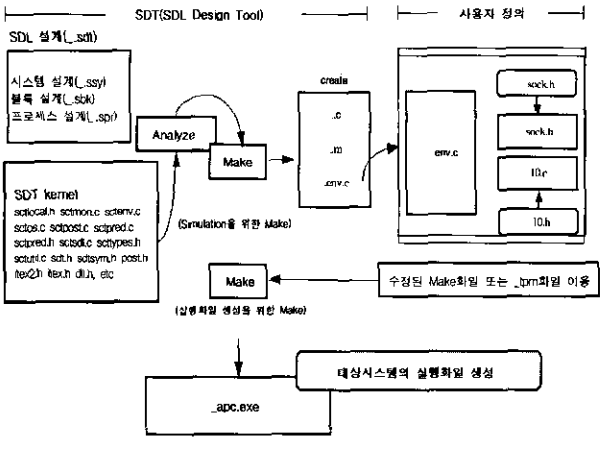
시뮬레이션 하고자 하는 대상 시스템은 SDL로 설계된 후 코드생성기에 의하여 C 소스 코드로 자동생성 된다. 여기에 SDT 도구에 의하여 C 소스코드와 함께 생성되는 주변함수의 기능을 가진 _env.c 파일과 소켓 및 GUI를 포함한 입출력 기능을 가진 파일들을 함께 링크 및 컴파일 하여 최종 실행파일인 호스트 시뮬레이터가 생성된다. (그림 9)에서는 시뮬레이터의 구성을, (그림 10)에서는 생성과정을 보여준다.




(그림 9) 호스트 시뮬레이터 개발 구성

① SDL 편집기를 이용한 SDL 설계를 바탕으로 SDT의 전체 개발 도구를 관리하는 Organizer의 분석기를 통한 오류검사와 make에 의한 컴파일을 수행한다. C 코드 생성기에 의하여 C 소스코드와 주변함수인 _env.c 파일을 생성한다.

② 사용자 정의부분인 소켓 및 GUI를 포함한 입출력 함수와 함께 수정된 make 파일을 이용하여 컴파일을 수행하고 최종 실행파일(_apc.exe)인 호스트 시뮬레이터를 생성한다.




(그림 10) 대상 시스템의 실행파일 생성 과정

3.2.4 동작 절차 및 실행

앞 절에서 언급한 실행 파일이 본 논문에서는 RLC-AM의 호스트 시뮬레이터가 되며 다음 동작 절차에 따라 시뮬레이터의 확인 동작을 거치게 된다. 호스트 시뮬레이터 상에서는 AmPdu에 의한 데이터 전송과 StatusPdu에 의한 응답으로 사용되는 send_ data( ) 함수를 위한 MSG_Sent_socket과 MSG_Incoming_Socket, 그리고 GUI와 클라이언트간, 클라이언트와 서버간 전송시 사용되는 Control_send() 함수를 위한 Control_Socket이 사용되며 이 들간의 구성이 (그림 11)에 나와있다. (그림 12)는 호스트를 Windows NT로 하여 개발한 시뮬레이터의 동작 절차를 나타낸다.

① MS Client GUI에서 'abcd'라는 메시지를 입력하고 Send Message 버튼을 누르면 입력한 메시지가 MS GUI의 Layer2에 출력되고 그 메시지는 함수 Control_send( )에 의해 AmPdu의 시그널로 MS SDL 시스템(rlc_am_apc.exe)으로 전송된다

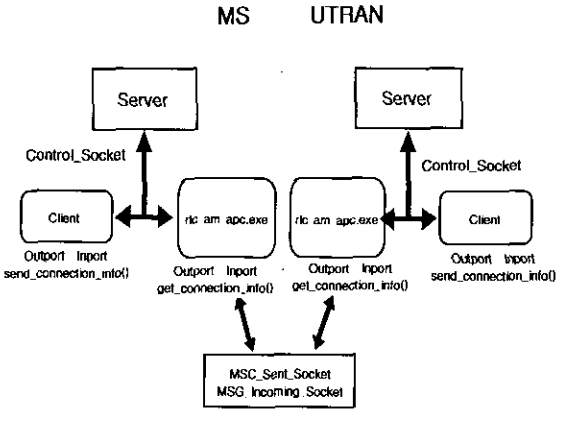
② MS SDL 시스템으로 수신된 입력 데이터인 'abcd'는 Control_send( )에 의해 다시 MS Client GUI로 전송하고 Layer3에 나타낸다.

③-④ MS는 UTRAN SDL 시스템으로 AmPdu를 함수 send_data( )에 의해 소켓 전송한다.

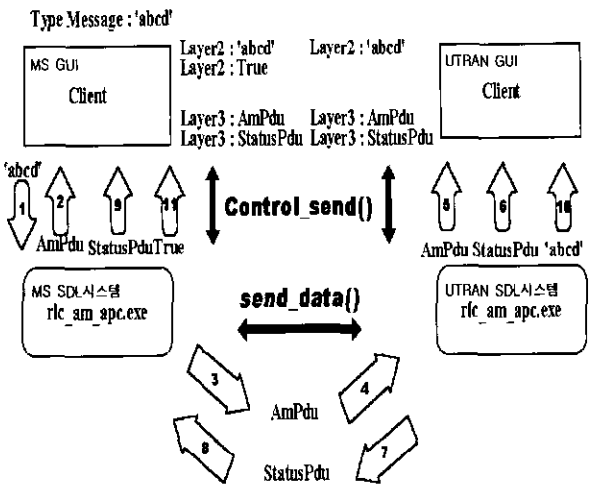
⑤-⑥ UTRAN SDL 시스템(rlc_am_apc.exe)은 MS에서 수신한 AmPdu와 StatusPdu를 함수 Control_send( )에 의해 UTRAN Client GUI로 전송하⑦-⑧ UTRAN SDL 시스템은 StatusPdu를 함수 send_data( )를 이용해 MS SDL 시스템으로 보낸다.

⑨ MS SDL 시스템으로 전송 된 StatusPdu는 함수 Control_send( )를 이용해 MS Client GUI Layer3에 나타낸다.

⑩ MS SDL 시스템에서 송신한 AmPdu는 UTRAN SDL 시스템에서 변환 과정을 거친 후 MS Client GUI에서 입력한 메시지와 동일한 메시지를 Control




(그림 11) MS와 UTRAN의 소켓 전송 함수




(그림 12) 호스트 시뮬레이터의 동작 절차

send( )를 이용해 UTRAN Client GUI Layer2에 나타낸다. 즉 MS에서 보낸 메시지가 UTRAN측으로 오류없이 전송됨을 확인한다.

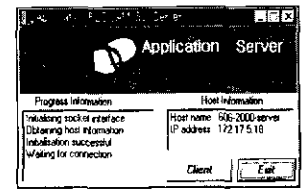
⑪ MS SDL 시스템은 올바른 StatusPdu를 받게 되면 MS Client GUI Layer2에 'True'라는 값을 나타낸다.

(그림 13-(a, b))와 같이 MS와 UTRAN측의 서버를 실행한다. (그림 13-a)의 MS Server의 Client 버튼을 누르면 (그림 13-c)와 같이 MS SDL 시스템인 rlc_am_apc.exe가 실행되고 MS Client GUI가 실행된다. UTRAN도 (그림 13-d)와 같이 동일하게 실행된다.

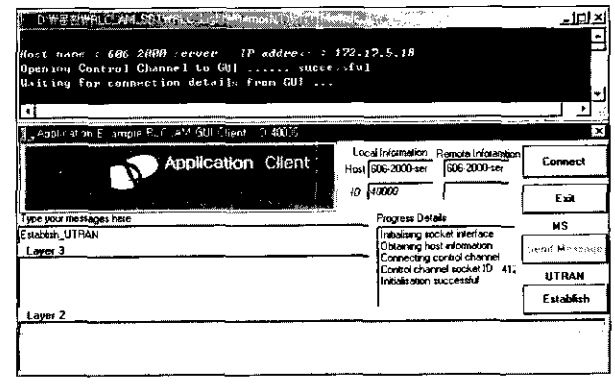
다음으로 MS Client GUI에서 Remote Information의 ID란에 UTRAN의 Local Information의 ID값(40001)을 적고 UTRAN Client GUI에서는 Remote Information의 ID란에 MS의 Local Information의 ID값(40000)을 적고 각각 Connect 버튼을 누르면 서로 소켓 연결된다. MS와 UTRAN이 성공적으로 소켓 연결이 되었다면 Send Message 버튼이 활성화된다. 그 다음엔 UTRAN Client GUI에서 Establish 버튼을 눌러 UTRAN측의 연결설정을 완료한다. MS측은 입력 값을 적고 Send Message 버튼을 누름으로써 연결설정이 되도록 SDL 설계를 하였다. (그림 13-e)에서는 MS Client GUI에서 'abcd'를 입력하고 Send Message 버튼을 누르면 사용자가 원하는 AmPdu와 StatusPdu의 값과 입력한 메시지가 MS에서 UTRAN측으로 정확히 전송됨을 보이고 있다.

## 4. 결 론

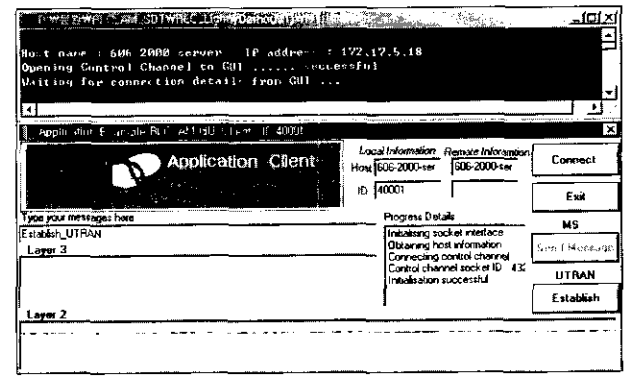
본 논문에서는 SDL 개발환경을 이용한 프로토콜의 시뮬레이터 개발 모델을 제안하고 IMT-2000 프로토콜인 RLC-AM 엔터티를 대상으로 호스트 시뮬레이터를 개발하였다. IMT-2000 프로토콜 규격을 표준 설계 언어인 SDL로 설계



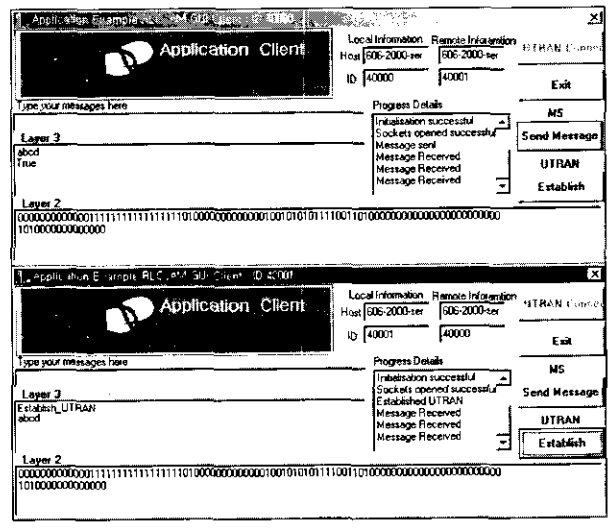

(a) MS Server의 실행 화면 (b) UTRAN Server의 실행 화면




(c) MS S이 시스템과 Client GUI의 실행 화면




(d) UTRAN SDL 시스템과 Client GUI의 실행 화면




(e) MS에서 UTRAN으로 메시지의 전송 결과 화면

(그림 13) 호스트 시뮬레이터 실행 화면

하고 SDL 개발 도구에 의하여 컴파일 및 코드 생성을 하여 각종 시뮬레이션을 수행한 후 검증된 설계서와 소스코드를 얻을 수 있었다. 또한, RLC 엔터티에 대한 통합 과정을 거쳐 실행 가능한 통신 엔진 모듈을 생성하고 소켓을 사용한 NT 환경의 일반 프로그래밍 언어로 개발된 GUI와 연동시켜 호스트 상에서 개발자가 직접 원하는 PDU에 대한 정보를 추출하였다. SDT의 시뮬레이션 도구인 Simulation UI를 이용한 MSC의 결과와 호스트 시뮬레이터의 수행 결과를 비교 분석하여 오류없는 메시지의 전송을 확인하였다. 표준 규격의 잦은 변경, 새로운 기능의 빈번한 추가 및 전문인력의 이동 등에 따라 소프트웨어의 개발 방법 및 유지보수 방안에 대한 대책이 절실한 현재의 상황에서 본 논문에서 제시한 호스트 시뮬레이터 개발 사례는 프로토콜의 설계, 구현, 시험 및 검증의 자동화는 물론 프로토콜 시험기 개발 수단의 좋은 방안이 될 것이다. 호스트 상에서의 시뮬레이터 개발에 이어 타겟 상에서의 시뮬레이터 개발에 대한 연구가 진행 중에 있다.

## 참 고 문 헌


[1] Holzmann, G.J.,"Design and Validation of Computer Protocols," Prentice-Hall, 1991.

[2] A. Olsen, O. færgemend, B. Møller Pedersen, R. Reed, and J. Smith,"Systems Engineering Using SDL-92," ELSEVIER SCIENCE B. V., 1995.

[3] Jan Ellsbetger, Dieter Hogrefe and Amardeo Sarma, 'SDL Formal Object-oriented Language for Communicating Systems,' Prentice Hall, 1997.

[4] Roberto Saracco, "Telecommunications Systems Engineering using SDL," North_Holland, 1989.

[5] Edel Sherratt, Chris Loftus, "Designing Distributed Services with SDL," IEEE, 2000.

[6] Telelogic Tau 3.4 SDT Methodology Guidelines, 1999.

[7] The SDT Simulator, 'Telelogic Tau 3.4 ORCA and SDT Started,' 1998.

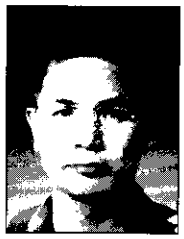
[8] Belina, F., Hogrefe, D. and Sarma, A. "SDT with Applications from Protocol Specification," Prentice Hall, 1991.

[9] Harri Holma and Antti Toskala, "WCDMA FOR UMTS," JOHN WILEY & SONS, LTD., 2000.

[10] 3G TS 25.322 v3.0.0(3rd Generation Partnership Project ; Technical Specification Group) Radio Access Network ; RLC Protocol Specification, 1999.

[11] TSGR2#8(99)F25, TSG-RAN Working Group 2 meeting #8, Cheju, Korea 2nd to 5th Nov. 1999.

[12] 노철우, 남영호, "IMT-2000/비동기 방식(3GPP) 무선 프로토콜의 SDL 설계검증 및 평가에 관한 연구", 전자통신 최종 연구보고서, 1999.


**송 평 중**

e-mail : pjsong@etri.re.kr

1980년 한양대학교 통신공학과 학사

1982년 한양대학교 통신공학과 석사

1995년 한양대학교 통신공학과 박사

1986년~1988년 벨지움 Bell telephone 파견근무(system 1240 교환기 sw 개발)

1998년 ITU-T SGX1/WP3/SWP2 Editor(IMT2000 LAC/MAC 분야)

1982년~현재 한국전자통신연구원 IMT-2000 개발본부 이동 프로토콜 연구팀장



**노 철 우**

e-mail : cwro@silla.ac.kr

1980년 서강대학교 물리학과 졸업(학사)

1982년 동국대학교 전자계산학과 졸업(석사)

1995년 서강대학교 전자계산학과 졸업(박사)

1982년~1991년 한국전자통신연구소 선임연구원

1996년~1997년 미국 Duke 대학교 컴퓨터 공학과 객원교수

1991년~현재 신라대학교 컴퓨터 정보공학부 정보통신공학전공 부교수

관심분야 : 프로토콜 공학, IMT-2000 무선 프로토콜 설계/검증/시험, 페트리 네트 모델링 및 분석



**노 문 환**

e-mail : mhro@lycos.co.kr

1999년~2000년 신라대학교 전자계산학과 대학원 졸업(석사)

2000년~현재 한국전자통신연구원 이동프로토콜 연구팀 위촉연구원

관심분야 : SDL 환경을 이용한 IMT-2000 무선 프로토콜 설계/검증/시험



**윤 영 배**

e-mail : justnow1@hanmail.net

2001년 신라대학교 정보통신공학과 졸업(학사)

2001년~현재 텔슨전자 IMT-2000 센타 근무

관심분야 : IMT-2000, 무선 프로토콜 설계, 검증

**이 종 필**

e-mail : jtyi@ce.cnu.ac.kr

2000년 충남대학교 컴퓨터공학과 학사

1999년~2000년 한국전자통신연구원 이동 프로토콜연구팀 위촉연구원

2000년~현재 충남대학교 컴퓨터공학과 석사과정