■ 特輯 · 技術解說

Protocol Engineering

崔 陽 熙
韓國電子通信研究所 데이타通信 研究室長(工博)

요 약

컴퓨터 통신망과 정보처리기기의 급속한 발전에 따라 다양한 통신 프로토콜이 출현하였다. 이에 따라 다른 종류의 프로토콜 사이의 호환성이 심각한 문제로 대두하였으며, 그 해결 방안으로 OSI(Open Systems Interconnection)의 개념이 정립되었다. OSI는 시스템간의 통신 체계를 개념적으로 정립하고 관련된 프로토콜의 사양을 포함하고 있으나 실제적인 프로토콜 설계, 시험, 구현등의 중요한 부분이 제외되어 있다. 본고에서는 프로토콜 설계로부터 마지막 시험에 이르기까지 한 통신 프로토콜이 거쳐야할 단계를 체계적으로 정리 분석 하였다. 이 분야는 프로토콜 엔지니어링 이라는 새로운 학문 분야로 컴퓨터 통신의 발전에 큰 기여를 할 것으로 보인다.

Ⅰ. 프로토콜 엔지니어링의 대두

컴퓨터가 출현한 이래 전자, 통신 및 정보처리 분야는 눈부신 발전을 거듭해 왔으며 이는 하드웨어의 향상, 반도체 기술의 발전, 운영체제의 혁신, 프로그래밍 언어의 다양화, 컴퓨터 구조의 꾸준한 개선으로 집약될 수 있다. 최근 십년간을 돌이켜보면 컴퓨터 통신이라는 새로운 분야가 보편화되고 이에 따라 분산처리, 분산시스템에 관한 연구, 개발, 보급에 많은 노력이 쏟아져왔다. 특히 컴퓨터 통신에서 중추적인 역할을 하는 통신 아키텍쳐와 통신 프로토콜은 생산업자마다 또는 나라마다 제각기의 방식을 고안하여 운용중에 있어서 통신에서 가장 중요한 다른 시스템간의 호환성의 문제가 크게 대두되고 있다.

ISO(International Standard Organization) 및 CCITT(The International Telegraph and Telephone Consultative Committee)를 주축으로 하여 정립된 OSI 개념은[1] 앞서 언급한 호환성의 문제를 해결하기 위한 것으로 앞으로 수십년간 정보통신 시스템에 적용되리라 본다. OSI 접근 방법은 간단히 말하면, 이종 시스템간의 통신은 첫째 각 시스템이 채택하는 개념적인 통신 아키텍쳐 모델이 같을 때 가능하며, 둘째 수직계층적 구조인 이 모델(그림 1)에서 각 계층이 상위계층에 제공하는 서비스가 같을 때 가능하며, 마지막으로 각 계층내에 존재하는 프로토콜이 동일할 때 가능하다는 것이다.

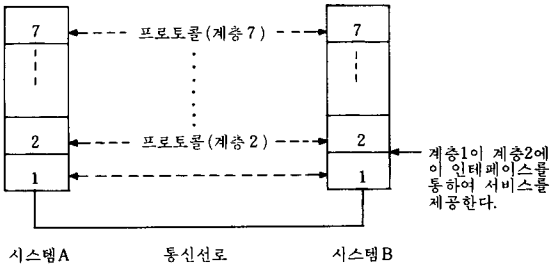


그림 1. OSI의 수직적인 계층구조 모델

따라서 정보통신에 참여하는 시스템들은 위의 세가지 기준을 지키기만 하면 시스템 고유의 특성(운영체제, 아키텍쳐등)을 그대로 유지하면서 서로 원활한 통신이 보장된다. 특수한 경우, 두 시스템이 같은 통신 아키텍쳐 모델, 같은 서비스 규격을 가지나 이것을 구현하는 통신 프로토콜이 서로 달라서 접속이 불가능한 경우가 있는데, 이때는 두 시스템 사이에 프로토콜 변환장치를 두어서 해결한다.

OSI 모델에서 한 계층은 바로 아래의 계층으로부터 하위 서비스를 제공받으며, 여기에 자기계층의 프로토콜을 이용하여 생긴 서비스를 추가하여 바로 윗 계층에 제공하게 된다. 역으로 한계층에서 바로 아래 계층으로부터 제공받은 서비스를 알고, 또한 바로 윗 계층에 제공해야 될 서비스가 잘 정의되어 있다면 이로부터 이 계층에서 해야될 일을 찾아낼 수 있고, 이를 가

능하게 하는 프로토콜을 설계할 수 있다.

이렇게 설계된 프로토콜은 대부분 자연언어를 사용하여 표현되어 왔다. 그러나 자연언어로 쓰여진 프로토콜 규격은

1) 정확하지 않고 여러가지로 해석할 수 있는 부분을 포함하거나

2) 규격 자체에 설계오류가 포함되어 있는 수가 많아서

똑같은 프로토콜 규격을 토대로 구현된 제품들 사이에 통신이 안되는 예가 많았다. 따라서 프로토콜 규격을 정확한 방식으로 표현하는 formal description techniques가 근래에 많이 소개되었다. SDL(Specification and Description Language)[2], LOTOS[3], ESTELLE[4], Petri Net[5] 등이 그것이며 다음장에서 자세히 설명하기로 한다.

통신 프로토콜은 분산된 시스템간에 일정하지 않은 전송지연시간, 에러발생율등을 극복할 수 있도록 설계되어야 하며 일반적으로 여러개의 하드웨어 및 소프트웨어 모듈로 구성되므로 모든 상황 아래에서 안정된 동작을 규격작성 단계에서 확인하는 것이 꼭 필요하게 된다. 따라서 formal description technique으로 쓰여진 프로토콜 규격상의 에러를 검출하고 이를 수정하는 프로토콜 Validation이 중요한 분야로 인식되어 왔으며 해석적 또는 시뮬레이션을 통한 여러 방법이 소개되었다.

잘 설계되고, 잘 쓰여지고, 오류가 없는 프로토콜이라도 실제의 시스템위에 구현하기가 어렵거나 경제성이 없으면 그 가치를 상실하게 된다. 또한 같은 프로토콜도 여러가지 다른 방법으로 구현할 수 있는데 방법에 따라, 구현의 용의성, 미래에 확장성, 동작의 효율성 및 경제성이 모두 차이가 나게 된다. 프로토콜 규격으로부터 어떻게 손쉽고 효율적인 방법으로 구현시키는가도 중요한 이슈로 등장하고 있다. 프로토콜 규격이 formal method로 쓰여지고, 이것이 computer에 의하여 자동으로 compile이 된다면 target system에 그대로 porting시킬 수 있는 object code를 기계적인 수단으로 얻어낼 수도 있게 된다. 이러한 자동구현이 근래에 많이 연구되었으며 이는 프로토콜 규격이 어떠한 formal technique으로 쓰여졌는가도 밀접한 관련을 갖는다.

한 통신 프로토콜은 여러 다른 생산업자들에 의해 다양한 종류의 기기내에 구현되는데 이들간의 통신은 각 기기들이 본래의 규격을 충실히 지켰을때만 가능하다. 하드웨어와 소프트웨어로 구성된 기기들을 대상으로 프로토콜 규격을 얼마나 지켰는가 검증하는 protocol testing의 역할은 대단히 중요하며 검증 시스템의 완벽도 및 성능에 모든 기기들의 성능 또한 비례한다고 볼 수 있다. 최근에 들어서 새로운 프로토콜을 설계, 규격 제정할 때는 이와 병행하여 검증 방법 및 검증 시스템에 관한것을 동시에 제정하는 추세에 있다.

앞서 소개한 프로토콜의 design-specification-validation-implemention-test는 어느 프로토콜에서나 거쳐야 할 일대기가 된다. 이 모든 분야를 체계적으로 정리, 분석한 학문 분야를 새로이 protocol engineering이라고 부르며[6][7] 복잡한 통신망, 통신 프로토콜의 증가와 더불어 그 중요성이 날로 인식되고 있다.

II. Formal Description Techniques

프로토콜 규격(specification)이 지녀야 할 성질을 먼저 살펴보기로 하자.

- 프로토콜 규격은 적어도 다음 것들을 포함해야 한다. 제어순서, 메세지 코팅방법, 성능기준, 에러검출 및 처리하는 방법, 상위 및 하위 계층과의 인터페이스
- 프로토콜 규격은 모호한 부분이 있어서는 안된다.
- 프로토콜 규격은 가능한 간결하게 묘사되어야 하며 중복성이나 over-specification을 피해야 한다.
- 기계적인 방법으로 규격자체의 검증, 실제 시스템으로의 구현 및 테스트에 도움을 줄 수 있는 표현 방식이 권고된다.

Formal Description Techniques는 위의 요구사항을 모두 충족시키고 있는데 이를 소개하기에 앞서 일반적인 protocol machine의 구조를 살펴보기로 한다. 그림 2에 보인바와 같이 한 시스템은 서로 interact하는 module의 집합으로 나타내어진다. module 사이는 channel이 존재하며, module과 channel은 interaction point를 통하여 연결한다.

Channel을 통한 interaction은 interaction primitive

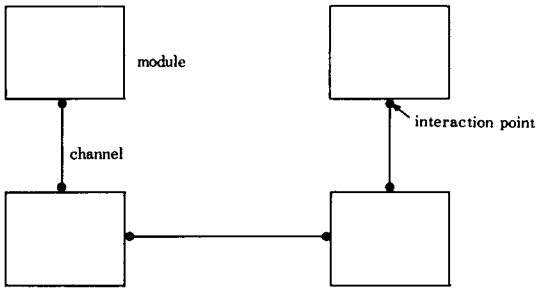


그림 2. Protocl machine의 구성

를 사용하는데, 예를들면 connect, disconnect, data 등이 이에 속한다. 한 프로토콜은 일반적으로 하나 또는 여러개의 module로 표현된다. 또한 한 module이 너무 abstract 한 경우 여러개의 작은 submodule로 쪼개는 것도 가능하다.

Formal Description Technique는 state 개념에 기초를 둔 부류와 event에 기초를 둔 부류로 크게 나눌 수 있다. SDL, Petri Net, ESTELLE등이 state를 바탕으로 하였고 LOTOS, Temporal logic등은 event에 기초를 두었다. 어느 것이나 한 module이 동작하는 내용을 묘사하는데 방법상의 차이일 뿐 내용의 차이는 없다.

State에 기초를 둔 것중 가장 간단한 것이 FSM (finite state machine)이다. 한 module은 내부적으로 정해진 갯수의 state들 사이를 이동하게 되는데, 한 state에서 다른 state로 옮기는 것은 외부의 자극(interaction primitive)을 받아 일어나고 이때 module의 외부로의 행동(interaction primitive의 생성)을 수반할 수 있다.

그러나 FSM은 복잡한 프로토콜을 묘사하고자 할때 state의 갯수가 너무 많아져 실제 이용이 힘들다. 따라서 extended finite state automaton을 이용하는데, 이 경우 한 module의 state space는 여러개의 variable에 의해 결정된다. 이 variable중 하나가 "STATE"라 불리우며 FSM에서 다루는 state에 대응한다. "STATE" 사이의 transition은 다른 variables의 값에 영향을 받으며, action도 variables의 값을 바꾸는 것을 포함하게 된다. ISO에서 다루는 ESTELLE이 extended FSM중 대표적이며 Pascal 언어와 유사한 언어를 묘사하고 있다. 예를 들면 그림3에 나타낸 transition은 ESTAB이라는 major state에서 N. DATA response라는 primitive를 받으면 그 type이 DATA인가 확인해보고 맞는 경우 begin과 end사이의 action을 취하고 다시 ESTAB이라는 state로 돌아오는 것을 표시한다. action중에는 module 외부로 interaction primitive를 보내는 것도 있고, 내부의 variable의 값을 변환시키는 것도 있다.

```
FROM ESTAB TO ESTAB
WHEN N.DATA_response
PROVIDED N.DATA.id=DATA
begin
  q.msgdata := NData.data;
  q.magseq := NData.seq;
  send.ack(q)
  if NData.seq = recv.seq then
  begin
    store(recv.buffer, q);
    incr_recv_seq
  end
end;
```

그림 3. ESTELLE로 표현한 예

Petri net은 protocol이 parallel process의 집합이라는데 착안하여 도입된 기법이다. condition은 circle로 표시되고 transition은 bar로 표시되며, condition이 만족되는 경우(circle안에 •으로 표시된 token이 있는 경우) bar를 통하여 action을 한 다음, 다음 circle로 token이 이동하게 된다. 그림4에서 한 message가 생성되어 외부로 송신되고 module은 acknowledge 수신대기 상태로 움직이는 단계를 표시하고 있다.

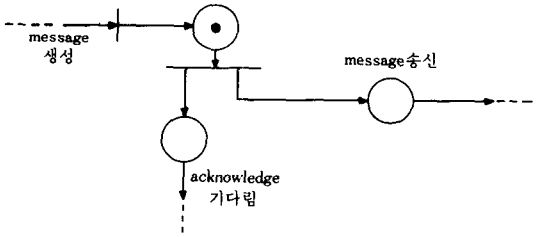


그림 4. Petri net의 예

CCITT에서 제공하는 SDL은 ESTELLE과 기능상 유사하나 다른 notation을 사용하고 있고 graphic notation은 널리 쓰이고 있다. 그림5에 간단한 예를 보였다.

ESTELLE과 함께 CCITT에서 제공한 LOTOS는 event에 바탕을 두었으며 module의 성격을 module의 각 interaction point에서 일어나는 event들의 시간적 인과관계를 정의해 줌으로서 완벽히 표현해 주고 있다. conc, seq, muterm, select등의 temporal ordering primitive들을 쓰며 지금까지 소개한 formal description technique들중 가장 abstract하며 축소된 분량으로 규격 작성이 가능하다.

Protocol specification technique들은 각기 장 단점을 지니고 있으며 어느 것이 특별히 우수하다고 하기 어렵다. 세계적으로 SDL, Petri Net, ESTELLE, LOTOS가 모두 활발히 연구되고 있는 실정이다. 사용

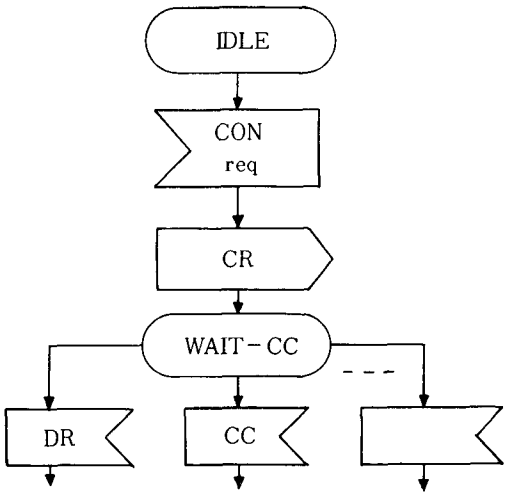


그림 5. SDL을 사용한 예

자가 쓰기 편리한점, implementation에 주는 도움의 정도는 현 수준에서 보면 SDL이나 ESTELLE이 우수하나, 다른 방법의 빠른 발전에 따라 곧 비슷해질 것으로 예상된다.

Ⅲ. Protocol Validation

Protocol validation의 목적은 한 계층의 프로토콜 규격이 밑의 계층에서 주어진 서비스에 얹어졌을 때, 원하는 대로 윗계층에 제대로 서비스를 공급하는가를 확인하는 것이다. 프로토콜 검증은 두가지의 상호 독립적인 검증분야로 다시 나뉘어진다. 첫번째는 한 프로토콜 고유의 기능(메세지 전달, 에러복구 등)에 관계없이 어느 프로토콜 규격에서나 지켜야 될 일반적인 성질을 검사하는 것으로 deadlock의 유무검출, 모든 state로 부터 reinitialization의 가능여부 조사 infinite loop검출, 중복성등이 그 예가 된다.

두번째 분야는 한 프로토콜이 행해야 되는 고유기능을 검증하는 것으로 예를 들면 에러검출, flow control, connection management등이 있다.

프로토콜 검증에 사용되는 대표적인 기법으로는 reachability analysis[8]와 assertion[9]을 들 수 있다.

Reachability analysis는 state machine으로 표현된 프로토콜 규격에 적용이 가능하며 일반적인 기능을 검증하는데 쓰인다. 프로토콜은 두 시스템간의 통신을 정의하는 규약이므로 두개의 state machine간의 interaction으로 항상 나타나게 된다. 각각의 state machine은 내부 state, input, output 그리고 transition 으로 정의되며, 한 state machine의 output은 다른 state machine의 input으로 동작하게 된다. Reachability analysis는 두 state machine의 internal state의 조합으로 표시되는 global state들 사이의 transition을 표시하는 tree를 작성하고 이 tree위에서 error를 찾아내고 있다.

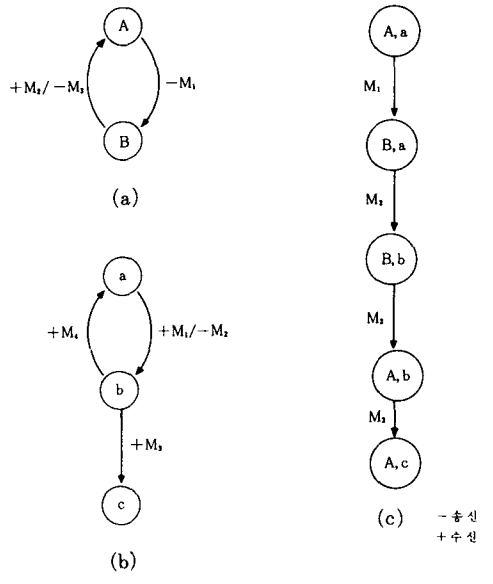


그림 6. Reachability Analysis의 예

예를 들어 그림 6을 보자. (a)와 (b)는 서로 통신하는 두 state machine을 나타낸다. 초기 상태는(A, a)이다. M 1 이라는 message를 송신하고 B state로 옮기고, 수신측에서는 이에 대하여 M 2 를 보낸다. 송신측에서는 M 2에 M 3 로 답하는 데 수신측에서는 M 3 를 받으면 C state라는 deadlock state로 들어가 버린다. 이는 그림 6 (c)에서 마지막 global state(A, c)로부터 출구가 없는 것으로 표시되어 이 state가 deadlock state임 을 말해주고 있다.

Reachability analysis를 이용하여 state machine으로 쓰여진 프로토콜을 검증하는 소프트웨어들이 외국에서 개발되었으며, 수작업으로 발견하지 못했던 에러를 검출하는데 많은 도움을 주었다.

Assertion이란 어떤 프로토콜 module의 state에서 긍정의 값을 가질 predicate를 나열하고, 이것을 프로토콜 규격으로부터 검증해내는 방법을 말한다. 예를 들면, HDLC 프로토콜에서 다음의 assertion이 성립한다.

"module sender의 state=established
module receiver의 state=established에서
수신된 I frame의 N(S)값=V(R) 값 이면
(receiver에 수신된 데이타)=(sender가 보낸 데이타) 이다."

Assertion을 이용한 방법은 정확하게 assertion을 묘사하기 어려운 점, 그리고 assertion을 실제로 증명하기 어려운 점등의 단점을 갖고 있다.

Ⅳ. 구현 및 실험

프로토콜의 구현은 OSI 모델에서 계층1~3 간은 하드웨어에 밀접한 영향을 받으며 일반적으로 소프트웨어는 성능을 고려하여 assembly 언어로 쓰여진다. 대표적인 프로토콜은 VLSI화 되어서 실제구현에 손쉽다. 그러나 계층 4 이상이므로 올라가면 소프트웨어로 구현시키고 있으므로 software module 단위의 decomposition, 운영체제 및 I/O와의 인터페이스 등이 매우 중요해진다. 또한 프로토콜이 real-time으로 동작하며 성능이 중요한 system parameter이고, 또 갖가지 error나 malfunction에 대해 스스로 보호할 수 있도록 구현되어야 하므로 소프트웨어 코드의 효율성이 강조된다. 나아가서 한 프로토콜의 규격은 기술의 발전, 사용자의 요구에 따라 자주 그 내용이 조금씩 바뀌므로 소프트웨어 코드의 수정 또한 용이 해야 한다.

Formal description technique으로 쓰여진 프로토콜 규격을 입력으로 하고 실제 시스템에 porting 될 수 있는 소프트웨어 코드를 출력으로 하는 컴파일러의 개발이 근래에 활발한데, 이러한 automatic implemention은 출력되는 소프트웨어 코드상에 에러가 없고, 만족할 만한 효율성, 그리고 프로토콜 규격의 변경시 빠른 소프트웨어의 수정이 가능하다는 점에서 그 이용 가치가 높다.

한 프로토콜은 소프트웨어로 구현하고자 할때 procedure에 해당하는 부분은 이와 같이 자동처리 할 수 있으나, local environment와의 matching은 수작업으로 해결하게 된다. 즉 운영체제와의 인터페이스, I/O와의 인테페이스 및 timer, counter 등의 이용은 target system마다 다르므로 자동화가 어렵다. 대개 자동화가 가능하며 target system에 관계없이 portable한 소프트웨어의 구성비율은 50%로 보고있다. 이 구성 비율은 UNIX나 standard I/O 를 쓰는 경우 더 높일 수 있을 것으로 기대된다.

프로토콜 엔지니어링의 마지막 단계인 프로토콜 시험(testing)[10]의 중요성은 formal specification 기법의 발달과 그 맥락을 같이 하고 있다. 즉 어떤 프로토콜을 구현한 시스템이 있을 때, 이것이 과연 원래의 프로토콜 규격에 부합하는 가를 살피려면, 원래의 규격에 대한 정확한 해석이 먼저 요구되기 때문이다.

시험대상의 구현된 프로토콜(IUT : implementation under test)을 시험하기 위하여 IUT와 상위계층의 인터페이스, IUT와 하위계층의 인터페이스, IUT가 송신할 메세지(PDU : protocol data unit), 그리고 IUT와 local system environment 사이의 interaction을 모두 제어할 수 있어야 완벽을 기할 수 있다. 그러나 실제로 IUT는 target system내에 한 부분으로 integration 되어 있으므로 위의 네가지 인터페이스를 직접 컨트롤 하기는 불가능하다.

프로토콜 시험은 OSI 모델의 하위계층(계층1-3)과 상위계층(계층4-7)이 각각 다른 방법을 쓰고 있다. 하위계층을 시험하고자 할때는 시험대상의 시스템(SUT : system under test)과 시험기기(tester)를 직접 연결한다. (그림 7)

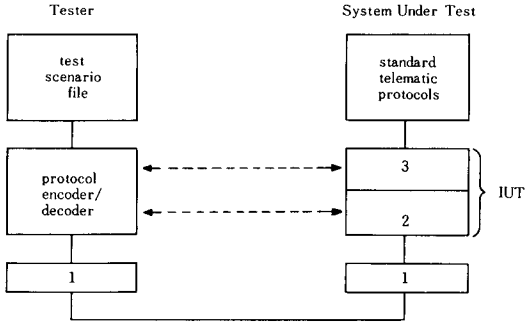
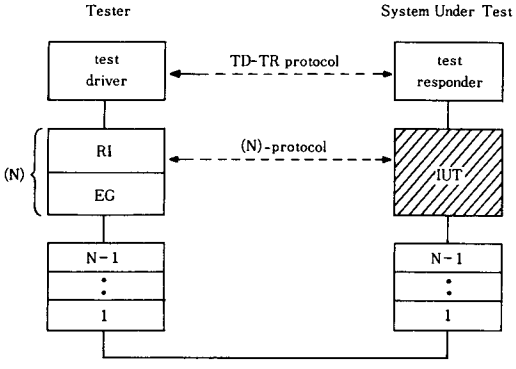


그림 7. 프로토콜 시험방법 - 하위계층

Tester에는 시험할 procedure를 sequence별로 묶어놓은 test scenario file이 있고, 이 file로부터 한 test case가 선택되었을 때 주어진 대로 PDU를 생성, IUT로 전송시키는 protocol encoder/decoder(PED)가 있다. PED는 시험대상 프로토콜을 완전히 구현하고 있지 않고 test case마다 그에 필요한 부분만을 가지고 있는 partial implementation이다. 예를들어 IUT의 connection 처리기능을 test 할 경우 PED에는 CONNECT-Request↔CONNECT-Confirm의 PDU 교환만을 처리하는 루틴이 담겨있게 된다.

이러한 하위계층 시험기는 protocol analyzer/simulator 라는 이름으로 많은 종류가 판매되고 있으며 X.25, X.21, SDLC등을 처리하고 있다.

상위계층의 경우, SUT와 tester는 직접 연결, 또는 통신망을 통하여 연결할 수 있다. 상위계층의 경우도 하위계층과 같이 PED를 사용하는 것이 가능하나 대부분은 그림 8에 보인바와 같이 reference implementation (RI) 를 사용한다.



RI : reference implementation
EG : exceptron generator
IUT : implementation under test

그림 8. 프로토콜 시험방법 – 상위계층

RI 위에 테스트 순서의 조정, 동기유지 등을 담당하는 test driver (TD), RI 아래에 RI에서 다룰 수 없는 error나 exception case를 생성, 처리하는 exception generator (EG)를 두어서 TD, RI, EG가 합쳐서 PED를 대신하게 된다.

STU내의 IUT 위에 test responder (TR)을 두어서 IUT와 상위계층 사이의 인터페이스 컨트롤을 하며 TR이 할 일은 특수한 TD-TR프로토콜을 통하여 TD로부터 지시받게 된다.

프로토콜 시험에 관한 연구는 영국, 프랑스, 미국, 독일, 일본 등에서 활발하며 ISO 및 CCITT에서도 프로토콜 시험에 관한 표준모델, 표준 test cases 등을 제정하고 있다. 우리나라도 컴퓨터망의 보급, 정보통신망의 다양화, 새로운 서비스의 출현에 따라 한국 전자통신 연구소에서 프로토콜 시험기 (Protocol Test Facility)의 연구, 개발을 추진하고 있다. 이 시험기는 앞으로 LAN, ISDN 및 새로운 서비스 (Teletex, MHS) 등에 관한 프로토콜 시험을 주요 목표로하고 있다.

V. 앞으로의 방향

프로토콜 엔지니어링의 개념이 소개된지 불과 수년에 불과하나, 그동안 눈부시게 발전해 왔다. 통신망, 정보통신서비스가 복잡해짐에 따라 프로토콜 엔지니어링의 중요성이 더 강조되고 있다.

프로토콜 엔지니어링에서 가장 중요한 부분은 formal description techniques와 protocol testing으로 보아진다. 앞으로 프로토콜에 관한 개발이나, 표준화에는 이 두분야가 꼭 포함되어야 궁극적인 목표의 달성, 즉 동일한 프로토콜을 구현한 갖가지 시스템간의 원활한 소통이 손쉬울 것이다. 따라서 우리나라의 프로토콜 엔지니어링 연구개발 방향도 이 두분야에 비중을 두어야 할 줄로 믿는다.

프로토콜 시험은 책임있는 기관에서 맡아 서비스를 제공해주는 것이 바람직하며 생산업체, 학교, 연구소, 사용자등은 이 시험기에 통신망을 통하여 연결되어야 한다. 또한 한 시험기에 다룰 수 없는 특수 프로토콜인 경우 국내의 다른 곳, 또는 외국기관의 시험기와도 연결을 시켜서 시험을 행하는 프로토콜 테스트 망(test net)이 국내 및 세계적으로 구축되어야 할 것이다.

끝으로 우리나라의 정보산업, 통신산업의 도약에 큰 힘이 될 주요한 프로젝트인 국가 기간 전산망구축, 종합정보통신망, 새로운 서비스등에 프로토콜 엔지니어링 개념이 도입되어 좋은 결실을 맺기를 희망하는 바이다.

參 考 文 獻

[1] ISO, Information Processing Systems - Open Systems Interconnection - Basic Reference Model, ISO 7498.

[2] CCITT, Recommendations Z.101-Z.104, Yellow Book, Geneva, 1981.

[3] ISO, Information Processing Systems - Open Systems Interconnection-LOTOS-A Formal Description Technique based on the temporal ordering of observational behavior, ISO TC97/SC21 N423, Feb., 1985.

[4] ISO, Estelle-A Formal Description Technique based on an Extended State Transition Model, ISO TC97/SC21 N422, Feb., 1985.

[5] M. Diaz, "Modeling and analysis of communication and cooperation protocols using Petri net based models, *"Computer Network*, no. 6, pp. 419-441, June, 1982.

[6] H. Rudin, "An informal overview of formal protocol specification," *IEEE Communications Magazine*, vol. 23, no. 3, pp. 46-52, Mar., 1985.

[7] H. Zimmermann, "On Protocol engineering," Proceedings of Information Processing 83, pp. 283-292, IFIP, Paris, 1983.
[8] P. Zafiropulo, "Protocol validation by duologue-matrix analysis," *IEEE Trans. Comm.*, vol. COM-26, no.8, pp. 1187-1194, Aug., 1978.
[9] G.V. Bochmann and J. Gecsei, "A unified method for the specification and verification of protocols," Proc. IFIP, pp. 229-234, 1977.
[10] ISO, working draft for OSI confermance testing methodology and framework, ISO TC97/SC21 N410, Feb., 1985. *