# OSI Testing기법

元裕憲·印素蘭·郭鎬榮
(홍익대학교 교수, 박사과정 재학)

■ 차 례 ■



## 1 개 요

컴퓨터망 및 각종 다양한 통신망은 그의 규모와 복잡성이 사용자의 요구에 따라 크게 증대되고 이고 이들에 소요되는 각종 시스템들은 여러 회사에서 각자 독특한 방법으로 구현 개발 생산하고 있다.

이러한 상황에서 통신망간의 상호접속 및 다양한 통신을 제공하려면 별도로 다른 시스템이 도입 연결되어야 한다. 즉, Gateway시스템, Interworking Unit등 중간 역할(코드변환, 속도변환, 미디어변환, 프로토콜변환) 할 수 있는 시설이 재요구되므로 시간적, 경제적 낭비가 있다. 통신 및 관련부분에 대해 국제기구인 CCITT와 ISO에서는 이의 필요성을 인지하여 분야별로 활발한 활용을 함으로서 이를 해결하려고 하고 있다.

특히 ISO에서는 OSI Standards형태로 layer Independent Standards, 응용계층표준, 표현계층표준, Session계층표준, transport 계층표준, 네트워크계층표준, data-link 계층표준 및 물리계층표준 등 각계층별 표준을 작성 제시하고 있다.

이들 중에서 계층독립표준(layer Independen Standards)은 각 계층의 특성과는 별도로 모든 계층에 독립적으로 사용될 수 있는 표준으로서 Formal Description Technique(이하 FDT라 칭함), Conformance testing 그리고 Registration Authorities 등으로 구성되어 있다.

현재 OSI testing기법 중의 하나인 FDT 는 ISO에서는 Extended State Transition Model (이하 ESTELLE라 칭함, ISO 9074), Temporal Ordering of Observational behavior(이하 LOTOS라 칭함, ISO8807)가 있고 CCITT에서는 SDL이 있다.

또한 OSI Testing종류 중 프로토콜테스트의 가장 대표적인 Conformance testing 부분도

OSI Conformance testing methodology & framework에 관한 일반적인 개념(ISO DP 9646-1), Abstract test suite Specification(ISO DP9646-2), Executable Test Derivation(ISO DP9646-3), Requirements for Suppliers and clients laboratories(ISO DP 9646-4), 그리고 Test laboratory operation (ISO DP 9646-5)로 되어 있다.

## 2 용어해설

IUT
SUT
RI
EG
ATS
Test Driver
Test Reponsder

## 3 Test 종류

Test는 Test제품의 어느 분야를 test 하느냐에 따라서 크게 Conformance test, Performance test, robustness test, 그리고 reliability test 등 4가지로 분류가 된다.

Conformance test는 통신제품이 해당 통신 프로토콜 규격에 어느 정도까지 일치되게 구현시켰는지를 검증하는 test이고, robustness test는 통신제품이 착오가 발생된 경우에 대해 어느 정도까지 재회복을 시켜줄 수 있는지를 검증하며, Performance test는 통신제품이 동일한 통신 프로토콜 규격으로 구현된 다른 타 통신제품들에 비해 어느 정도의 성능을 지니도록 했는지 검증하는 테스트이다.

본고에서는 FDT ESTELLE이 사용된 통신제품에서 그 적용된 계층별 프로토콜들이 프로토콜 규격에 어느 정도 적합하게 구현했는지를 테스트하는 Conformance test에서 사용되는 Test 범주로 정하였다.

## 4 Test 기법

프로토콜 테스트 기법은 테스트 시스템을 구성하는 분야, 테스트방법, 그리고 테스트 시나리오를 작성하는 기법측면 등 3분야로 다시 분류해 볼 수 있다. 이들 각각에 대해서는 다음과 같이 설명할 수 있다.

### 가. 테스트 시스템 구성하는 측면

이는 테스트를 실시하는 측과 테스트를 당할 측이 분리되어 있느냐, 합쳐져 있느냐, 테스트 제품 내에 함께 들어 있느냐에 따라서 크게 local test, remote test, distributed test로 분류된다.

이들은 테스트방법과 integrate되어서 하나의 테스트 환경을 구축한다.

### 나. 테스트 방법

테스트 방법은 테스트 대상 프로토콜이 단일 계층만을 선정하여 테스트하는가, 다중계층(Multiple layer)을 테스트하는가, 테스트를 주관하는 시스템과 테스트를 당할 프로토콜을 지닌 시스템을 한 장소, 한 시스템에 두느냐, 별도로 두느냐 중의 테스트 환경조건에 따라서 크게 6가지로 분류가 되고 있다.

즉, Local Single Layer(LS)방법, Local Multi Layer(LM)방법, Remote Single Layer(RS)방법, Remote Multi Layer(RM) 방법, Distributed Multi Layer(DM)방법 등 6가지가 있다.

이들은 테스트를 할 통신제품의 규격과 테스트를 실시할 환경을 기준으로 하여 선택한 후 실시한다.

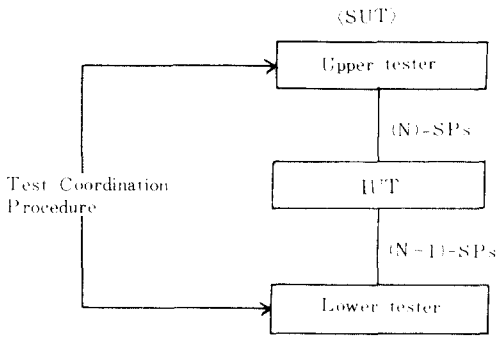



그림 1 Local Test 방법

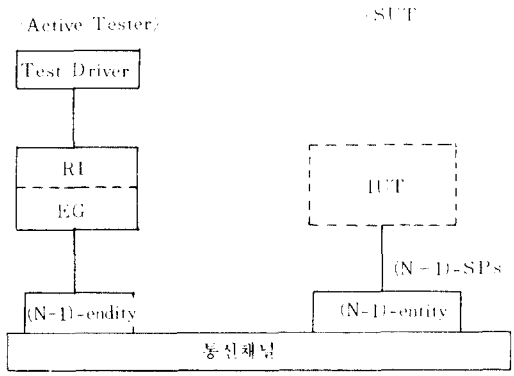



그림 2 Remote Test 방법

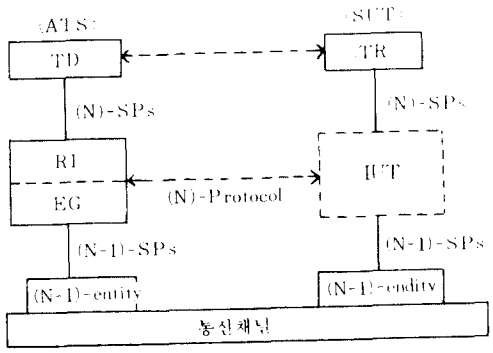



그림 3 Distributed Test 방법.

다. 작성기법측면.

이는 ISO에서 1980년부터 연구 및 규격화하고 있는 프로그래밍언어와 관련된 분야로서 통신 프로토콜규격을 정의하고 작성 및 검증하는데 사용되는 FDT이다.

FDT는 프로토콜의 정의적 규격과 분석적 목적에 따라 프로토콜 모델에 제공되도록 통신망 분야의 망시스템에서 사용되고 있으며, 자연어로 표현하는 추세와 수학적으로 표현하는 경향으로 분류되어 연구개발되고 있으나 표기적인 장치 및 규격의 기본 개념상에 있어서 약간의 차이와 장단점이 있다.

FDT의 대표적인 예는 ISO에서 제시한 ESTELLE과 LOTOS, 그리고 CCITT에서 제시한 SDL이 있으며 그외에도 SAN(State Architecture Notation), RSPL(Reliable Software Production Language) 등 다양하다.

이들 FDT는 Signal, 구조, 표기법, 수행방법 등 기본적인 속성을 지니고 있으므로 이들을 토대로 기존 FDT들을 표 1과 같이 요약할 수 있다.

데이타통신 프로토콜의 규격은 IS(International Standards)화 하고자 1980년경부터 ISO에서 시작됐고 1986년경에는 SDL의 특성인 EFSM(Extended Finite State Machine)을 지닌 ISO-PASCAL(Levelo: Sequential)의 확장형인 Concurrent개념이 추가된 ESTELLE를 제안하였다.

ESTELLE은 기존FDT에 비해 프로그래머가 사용하기 쉽고, 읽기 쉬운 형태로 되어 있으며, 여러 측면에서 통신제품의 검증을 적용하기에 용이한 언어이다. (표 2, 표 3, 표 4, 표 5, 표 6 참조)

표 1 주요한 FDT의 기본 속성 요약표.

| 기본속성 / 명칭 | orientation | structure | signal | component | notation | specification philosophy | Execution |
|---|---|---|---|---|---|---|---|
| ESTELLE | State | Multi component | limited Static, Pulsed | DSM* CF* UQ* Delay PC* | 문자 | Meta-implementation Monitor : 가능 | Concurrent |
| Petri-net | State | Multi component | Static | CF* DSM* | 도형 | Meta-implementation | Concurrent |
| RSPL | Sequence | Monolithic | Pulsed | PC* | 문자 | Monitor | Sequential |
| SAN | State | Multi component | Continuous, Static, Pulsed | DSM* CF* UQ* PC* Delay Derivative | 문자, 도형 | Meta-implementation Monitor | Concurrent |

*주) SAN(State Architecture Notation)
DSM(Discrete State Machine)
CF(Combinational Function)
UQ(Unbounded Queue)
PC(Programmable Clock)
RSPL(Reliable Software Production Language)

표 2 Functional Coverage.

| area / issues 종류 | | SDL | ESTELLE | LOTOS |
|---|---|---|---|---|
| Concurrent | Parallism | 없 음 | 제약성있음 | 강 력 함 |
| | Communication | Asynchronous | Asynchronous | Synchronous |
| | Synchronization | 안 됨 | 안 됨 | 된 다 |
| Data Description | data 규격 | formalism | pascal-like 기법 | formalism |
| | data 보호 | module & block 기법 | 제공지기법 | 사용치않음 |
| | data안정성 | ADT로 제공된 기능을 통해서만 사용 | 방법이 없음 | ADT로 제공된 기능을 통해서만 사용 |
| Sequential | control flow | 제한된 강력한 primitive set | 강력한 primitive set | 제한된 강력한 primitive set |
| | exception handling | 없 음 | 없 음 | 없 음 |
| Real-time 제약사항 | T초내에 수행되어지는 행위 | | 강 력 함 | |
| | T초이내/후에 반응하는 환경 | | 강 력 함 | |
| | 시간종속적 대기상태 | delay | delay | normal data |
| System Testing | 규격테스트 facility | 없 음 | "observer" process | 없 음 |

표 3 Formal Definition.

| area \ 종류 / Issues | | SDL | ESTELLE | LOTOS |
|---|---|---|---|---|
| Syntax & Semantic | formalilies | • PR & GR 문법<br>• Semantic not recommand | 현재 진행중 | • 문법<br>• interpretation 모델<br>• mapping |
| | Different paradigm | • ADT<br>• FSM | • FSM<br>• general imperative 언어 | • TRUE<br>• FALSE |
| Analyzabilily | consistency, completeness, coherence, deadlock 결손 | Formal Semantic | • Static Checking<br>• formal semantic 이 필요 | Formal Semantic |
| | equivalence | 없음 | 유용치 못함 | 유용함 |
| extendability | | • macro<br>• process/ procedure<br>• ADT<br>• renaming<br>• derivation<br>• parameterization | • named procedure<br>• ADT, MACRO없음 | • Procedure-like Processes<br>• ACT ONE |
| Compatability & implementability | 규격의 interpretation 모델, target 의 interpretation 모델 | queued 통신 | | dynamic 부분 |
| 표준화 | | CCITT | ISO | ISO |
| Machine independence 구현모델에 따른 interpretation 모델의 순위 | | 2 | 1 | 3 |

표 4 Human Orientation.

| area \ issue \ 종류 | SDL | ESTELLE | LOTOS |
|---|---|---|---|
| usability | • double 표현허용<br>• 복잡게 사용<br>• 이해의 용이성<br>• ADT 사용<br>높은 수준 | • 프로그래밍 언어와 유사 (Module 지 (Channel) 정<br>중간 수준 | • 수학적인 개념에 준함<br>• 상당히 강력함<br>• ADT 사용<br>낮은 수준 |
| declarative & procedure facility | 높은 수준 | 중간 수준 | 높은 수준 |

표 5 표현력

| area \ issues \ 종류 | | SDL | ESTELLE | LOTOS |
|---|---|---|---|---|
| 규격의 정확성 | non-determinism imprecision | trick | 특수명령어 | 특수 symbol |
| constraints spec. | | 승인안됨 | 승인안됨 | 승인안됨 |
| concurrency & synchronization | concurrency | implicit | implicit | explicit |
| | synchronization | Waiting queue | Delay 문장 (interleaving 표현이 어렵다) | Asynchronous(interleaving 표현이 용이함) |
| | communication | Asynchronous | Asynchronous | Synchronous |
| Data 규격 | strong data type | 강 조 | 강 조 | 강 조 |
| | ADT | include | not include | include |
| | 매개변수화 및 data의 증가 가능범위 정의 | 가 능 | data 규격보다 프로토콜 규격에 더 적합 | 가 능 |

표 6 Structuring & Reusability.

| area \ 종류 | SDL | ESTELLE | LOTOS |
|---|---|---|---|
| structuring | • tree<br>• refinement 관계 | • tree<br>• refinement 관계 | • dynamic |
| reusability | • ADT 로서 formalization<br>• GR 형태<br>• procedure/ process 의 매개변수화 능력 ( 설계 단계 ) | • PASCAL 선언능력<br>• PASCAL능력<br>• procedure/ process 의 매개변수화 능력(Coding 단계 ) | • ADT 로서 formalization<br>• procedure/ process 의 매개변수화 능력 ( 설계 단계 ) |

## 5 ESTELLE 기법

### 가. 출현 배경

컴퓨터 통신은 설계 및 분석을 통해 규격화된 시스템이 완전하고 일치성이 있어야 하며, 모호한 표현이 배제되어야 국제적으로 인정받을 수 있고 기술적으로도 안정도가 높은 시스템이 될 수 있다. 현재 일반 통신 장비들 간의 통신을 수행하기 위해서는, 우선 protocol 을 설계하고

표 7 Tools

| 종류 / area | SDL | ESTELLE | LOTOS |
|---|---|---|---|
| Syntax Checking | • Graphic editor<br>• SDL-PR | • Syntax-directed editor | • Syntax-directed editor |
| Module interface Checking | Static semantic | Static semantic | • Static Checker<br>• interaction report generator |
| Prototyping & Simulation | • Simulator<br>• Symbolic evaluator<br>• Semi-automatic translation<br>• Prolog executor | 상당히 효율적인 Simulator | • Prolog executor<br>• LOTOS에서 CHILL로 변환을 진행중 |
| Dynamic state tracking & transition stepping | 제 공 | 제 공 | 쉽게 구성 |
| Control & Data flow Analysis | • Petri-net<br>• execution Graph개발 | • Petri-net<br>• execution Graph개발 | 연구중 |
| deadlocks & starvation발견 | | deadlock 발견 package 있음 | |
| Test Generation | 있 음 | 없 음 | 없 음 |
| Refinment Process Verification | 없 음<br>ADT에 따라 제공가능 | 없 음 | 없 음<br>ADT에 따라 제공가능 |
| 성능 분석 | queued petri-net로 더 쉽게 Mapping함 | queued petri-net로 mapping함 | |

그 설계된 protocol을 수행하는 소프트웨어나 하드웨어를 실제로 구성하는 방법을 택하였다.

그러나, 최근의 경향은 protocol의 하드웨어 검증을 배제하고 소프트웨어 시뮬레이션에 의한 검증의 추세로 진행되고 있다.

이에 따라 1980년부터 ISO (International Organization for Standardization)에서 통신상의 protocol 규격을 형식적으로 정의 기술하기 위해 FDT (Formal Description Technique) 기법을 사용하기 시작하였다. 또한 이 개념은 OSI(Open System Interconnection)의 목적을 성취하기 위해서도 반드시 필요한 기법이 된다. 통신 system을 구현해야하는 구현자들에게는 시스템이 목적에 맞게 정확히 구현되었는가, 또는 다른 구현 방법과 호환성이 있는가를 이FDT에 의해 정의할 수 있도록 하는 것이다.

이 FDT는 다음과 같은 용도로 이용된다.

- 서어비스, 프로토콜, 인터페이스 및 reference model에 대한 애매하지 않고 명확하며 간결한 규격의 제공.
- 규격의 완전성을 결정함.
- 규격이 모든 조건을 만족하는 가를 검증함.
- 구현이 표준 규격에 일치하는 가를 결정함.
- 표준들이 서로간에 일관성을 유지하는 가 를 결정함.
- 구현을 지원할 수 있는 도구를 제공함

현재 이 FDT가 널리 이용되고 있으며, 그대표적인 FDT의 종류는 ESTELLE, LOTOS, SDL 및 Petri-Net와 같은 것들이 있으며, 이들 FDT를 정의하는 활동은 ISO및 CCITT 를 중심으로 강력히 추진되고 있다.

현재 ISO에서 연구하고 있는 FDT에는ESTELLE과 LOTOS가 있는데, ESTELLE 은 프로토콜 entity의 내부 state가 존재한다는 것에 기초를 두고 있다. 프로토콜 entity는 signal이나 외부적인 자극에 의해 그 해당 자극에 대응하는 적절한 동작을 수행한 이후에 내부 state를 변경시킨다.

LOTOS는 이와 반대로 프로토콜 entity 와 그 환경(environment)간의 경계에서 발생되는 envent들에 대한 temporal ordering에 기초를 두고 있다.

또한, SDL은 CCITT를 중심으로 하여 전자교환기의 소프트웨어를 기술할 목적으로개발되었으며, 이후 SDL은 통신망 프로토콜에 적합하도록 확장되었다. SDL 역시 state에 기초를 두고 있어 ESTELLE과 표현 능력에 있어서는 거의 동일하다.

Petri-Net는 병렬 프로세스를 표현할 수 있는 graphical method이다. 프로토콜은 병렬프로세스들의 집합이므로 Petri-Net을 프로토콜

의 표현에 적용시키는 것은 매우 자연스러우며 Petri-Net에 대한 여러가지 변형(Timed Petri-Net, Numerical Petri-Net 등)들도 이용할 수 있다.

이상의 여러 FDT중 컴퓨터 상에서 소프트웨어 시뮬레이션이 가능하도록 프로그래밍 언어로 정의되어 있는 ESTELLE이 현재로서는 SDL이나 LOTOS보다 주목의 대상이 되고 있으며, ISO의 NBS 산하에서는 UNIX system 5 version에서 수행할 수 있는 prototype compiler를 개발하고 있으나, 아직 완성의 단계는 아니며, ESTELLE의 subset으로 구현한 것이 발표되었다.

## 나. 특 성

ISO 서브그룹 B에서 연구되고 있는 ESTELLE은 대표적인 구조화 프로그래밍 언어인 PASCAL에 기반을 둔 syntax와 semantic 을 지니고 있고 EFSM (Extended Finite State Machine)의 정의를 수월하게 할 수 있는 여러 기능을 지닌 언어이다.

특히, ESTELLE의 syntax는 프로토콜의 기능을 표시하는 모듈이라는 개념과 모듈들 간의 통신을 제공해 주는 채널이라는 개념을 이용하여 구성하게 되어 있으므로 다른 FDT 보다 융통성, 용이성 및 효율성이 높다.

### (1)구조적 모델(architectural model)

ESTELLE의 model은 표현을 위한 직관적인 방법으로 의미(semantic)를 제공하는데 사용되는 수단으로서, specification의 모델은 여러 모듈들의 집합으로 구성되는데, 각 모듈은 interaction points에서 관찰이 가능한 서브모듈로 나누어질 수 있는 데 이를 sub-structuring이라고 한다. 그리고 더 이상 sub-structuring이 불가능한 최소 단위의 모듈은 input interaction, output interaction, state와 transition으로 구성된다.

각 모듈들은 서로 상호 연관을 갖고 interaction들을 교환하게 되어 있는데 이들이 서로 interact하는 모듈의 지점을 interaction points라고 하며, 이 interaction들이 교환되는 모듈간의 통로를 채널이라고 한다. 즉, 채널이란 모듈들이 통신을 할 수 있는 매개체이다. 각 모듈은 interaction points들을 연결함으로써 상호 연결될 수 있다. 하나의 모듈은 그 기능에 따라 여러 개의 세부 모듈로 나뉘어질 수 있는데 ESTELLE에서는 더 이상 나뉘어지지 않는 모듈을 activity class로 정의하고 나머지 모듈은 process class로 정의한다.

언어의 Syntax 구조는 "Specification" 이라는 하나의 protocol module이 정의되고, 그 내부에서 통신에 필요한 채널의 개념과 실제 행동양식을 기술하기 위한 nested module의 header와 body가 존재한다.

내부 모듈에서는 상태 전이(state transition)를 발생시키는 "trans"의 부분이 있어 조건에 따라서 통신의 원리에 해당하는 행동을 하도록 한다. 이 상태 전이는 현재의 state에서 다음 state로의 이동을 표현하며, 그 이동은 channel을 통한 acknowledge에 의하여 대부분이 결정되어진다.

ESTELLE의 전체적인 구조를 그림으로 표시하면 다음과 같다.

### (2)기능적 모델(functional model)

ESTELLE언어의 기능적 모델은 Specification, Module, Channel, IP 등으로 분류할 수 있다.

가. Specification

하나의 protocol 규격을 정의할 때 그 규격자체를 표현하는 가장 상위의 module을 의미하며, 모든 Specification을 기술함에 있어서의 모든 내용을 포함한다.

나. Module

Specification내의 module은 여러 개가 존재할 수 있으며, module-header-definition과 module-body-definition으로 구분되어지며, 동일한 module-header를 가지는 여러 개의 module-body-definition들이 있을 수 있다. Module-

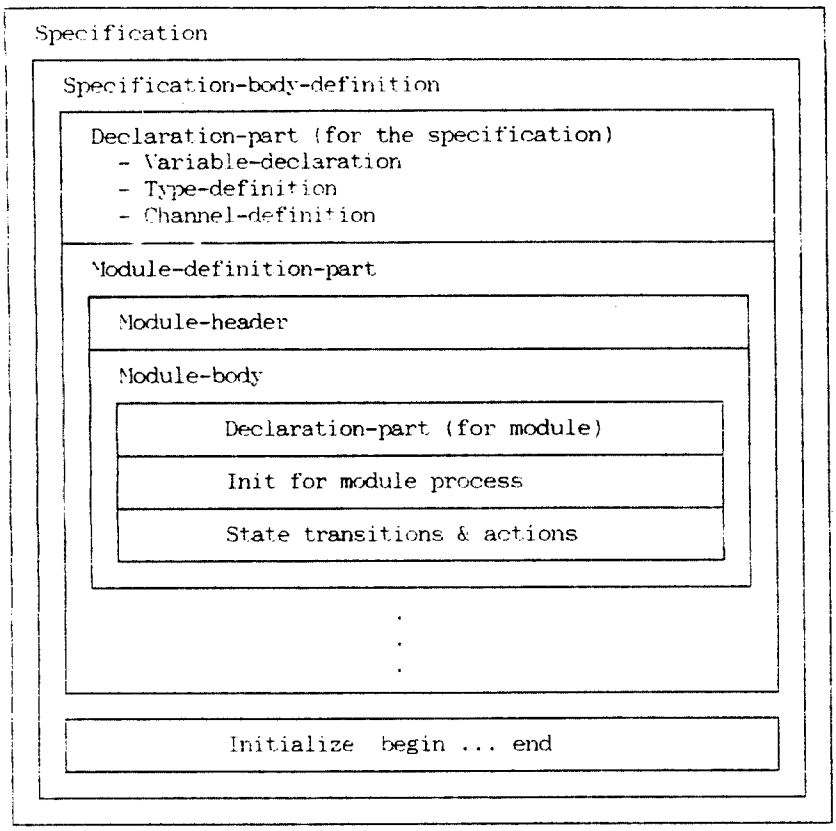



그림 4 ESTELLE의 Sys Syntax 구조.

header는 모듈의 전체적인 역할(process 또는 activity)과 이름을 나타내며 모듈 body는 모듈의 동작을 나타낸다.

header는 초기화되는 시점에서 현재의 모듈이 어느 Channel을 통해 통신이 이루어 지는가 하는 IP(Interaction Point)와 Formal parameter를 정의한다.

또한, module은 개념적으로 concurrent 한 수행을 하므로 acknowledge passing을 위한 queue의 의미를 정의하며, system process class, system activity class, process class, activity class 등의 4가지 class 중 하나로 정의된다.

Body부분은 자료를 선언하는 선언부, module을 수행하기 전에 초기화 작업을 하고 start state 상태로 전환하는 초기화부, 각 signal 에 의해 정의된 protocol specification대로 상태 전이를 발생하는 부분으로 나뉜다.

다) Channel

Channel-type-definition은 interaction-points를 통하여 모듈들 간에 교환될 수 있는 interaction의 집합을 정의한다. 채널 형태의 specification은 채널을 통하여 일어날 수 있는 가능한 interaction primitives(예를 들면, TCON-req, TCON-rsp등)를 나열함으로써 이루어진다. Interacting 모듈들의 구별을 위하여 각각의 역할에 따라 롤(Role) 개념이 도입되었는데 이 임무들은 interaction points를 통해 연결되는 각 모듈의 instances에 의해 역할을 행하게 된다.

channel-definition의 syntax는 다음과 같다.

channel-type-definition: = channel-heading
channel-block
channel-heading: = "channel" channel-type-identifier"("role-list")""
role-list: = role-identifier{", "role-identifier}
channel-type-identifier: = IDENTIFIER
role-identifier: = IDENTIFIER
channel-block: = + {interaction-group}
interaction-group: = "by" role-list " : "
+ {interaction-definition}
interaction-definition: = interaction-identifier ["("VALUE-PARAMETER-SPECIFICATION{";"VALUE-PARAMETER-SPECIFICATION}")"] ";"
interaction-identifier: = IDENTIFIER

주어진 interaction point에 대한 모듈 instance는 interaction-group에 포함된 역할을 행한다. 따라서, interaction-group에서 interaction-definition에 의해 정의된 interaction을 시작하면 반대편의 역할을 수행하는 모듈 instance는 이 interaction을 받을 수 있다.

이러한 채널은 프로토콜 규격의 각 layer 간의 통신을 담당하는 임무를 지니고 있으며, 한 layer를 중심으로 할 때 그 상위 layer와 하위 layer 간의 IP로서 acknowledge를 주고 받는다. 일반적으로 사용자(user) 측면과 제공자(provider) 측면의 두 측면으로 정의하며, 모듈 내에 포함되어 선언된 module과의 통신도 module내에서 선언된 channel을 통해 이루어진다.

라) IP (Interaction Points)

자신의 IP와 함께 수행되는 다른 module 과 interaction을 상호 교환하는데 사용되는 것으로서 하나의 module에 대한 abstract interface이다.

IP는 queuing의 원리와 channel identifier, 그리고 module의 임무 등 3가지 속성을 가지고 있으며, 다음과 같이 2가지로 분류한다.

- EIP (External IP) - module header에서 정의한 IP로서 children EIP 들은 그의 parent에게 visible하다.
- IIP (Internal IP) - module body내의 declaration part에서 정의된 IP로서 parent의 I-IP들은 그의 children과 함께 Interaction을 교환하는데 사용된다.

IP에서 Queue의 사용은 각각의 IP용 individual queue를 사용하거나 한 module이 다른 IP들과 공통으로 사용하는 common queue 를 사용한다.

마) ESTELLE Program의 구성

ESTELLE 프로그램의 주요 구성 요소는 앞의 모듈 구성에서 언급한 바와 같이 body-definition이 된다. Body-definition을 구성하는 declaration part, initialization part, transition declaration part의 각각에 대한 구성 형태 및 내용은 다음과 같다.

A. Declaration Part

Declaration part는 다음과 같은 part 들로 구성된다.

- CONSTANT-DEFINITION PART
- TYPE-DEFINITION PART
- channel-type-definition
- module-type-definition
- VARIABLE-DECLARATION-PART
- state-set-definition-part
- use clause
- PROCEDURE-AND-FUNCTION-DECLARATION-PART

이들 중 CONSTANT-DEFINITION, TYPE-DEFINITION, VARIABLE-DECLARATION, PROCEDURE-AND-FUNCTION-DECLARATION part들은 PASCAL의 경우와 동일 하다.

Channel-type-definition은 interaction point들을 통한 모듈들 간에 교환 가능한 interaction들이나 한 모듈내에서 내부 interaction point를 통하여 교환 가능한 interaction 들을 정의한다.

Module-type-definition은 process나 activity 같은 정해진 class module type을 정의한

다.

State-set-definition은 한번에 여러 state들을 참조할 수 있다. State-set은 transition 내의 from절이나 to절, 또한 init-procedure 내의 to절에만 이용할 수 있다.

User-clause는 현재의 body가 access 하는 header type(Module type)들을 정의한다.

B. Initialization Part

Initialization-part는 하나의 모듈 instance가 생성될 때에 수행되는 procedure를 정의하며, 이의 syntax는 다음과 같다.

```
initialization-part: ={"initialize"init-procedure}
init-procedure: = + {"with" RECORD-VARI
  ABLE-LIST "do" init-procedure}
  + {"providgd" (BOOLEAN-EXPRESSION
  "otherwise") init-procedure}
  transition-block ";"
to-list: = to-element {", " to-element}
to-element : = "same" | state-identifier |
  state-set-identifier
state-list: = state-element {", "state-element}
state-element : = state-identifier
```

to-clause내에서 하나의 state-identifier를 포함하는 to-list는 major state-variable이 state값을 가질 수 있게 한다. 하나 이상의 state identifier를 갖는 to-list는 protocol machine이 하나 이상의 state-identifier를 포함하면, 이 state-identifier는 procedure header 내에서 포함되어 있는 provided 절에 의해 선택되거나, 수행될 nextstate명령을 결정하는 transition-block내의 logic에 의해 선택된다.

Major-state-variable은 선택된 to 절에 대한 state-identifier값이나 수행된 nextstate statement에 대한 next-state-selector값을 갖는다.

모듈이 초기화될 때는 하나 이상의 transition-block이 enable되어 있더라도 하나의 transition-block만이 수행된다.

C. Transition Declaration Part

한 모듈내에 있는 각 protocol machine은 EFSM에 대한 instance이며, 이의 transition들은 해당 모듈에 대한 transition type들에 의해 정의된다. 어떤 protocol machine이 간단한 finite state machine을 특징짓는 single state variable이 아닌 여러 variable들을 포함할 경우에 그 protocol machine은 "extended"라고 부른다. 규약에 따라 이 variable은 "major state variable"이라 부른다.

Protocol machine에 대한 "complete state"는 major state variable과 다른 variable들을 사용하므로, major state variable을 제외한 다른 variable들은 state variable 또는 context variable이라 불리운다.

한 모듈에 대한 protocol machine의 transition들은 수많은 transition type들에 대한 specification에 따라 정의된다.

Transition type을 특징짓는 요소는 다음 두 가지가 있다.

(1) Enable condition

이의 구성은 다음과 같다.

- 현재의 major state (from 절)
- 입력 interaction (when 절)
- 가능한 predicate (provided 절)
- delay 기능 (delay 절)
- priority (priority 절)

(2) Transition의 운용

이의 구성은 다음과 같다.

- 다음의 major state(to 절)
- transition의 statement part에 포함된 출력 interaction을 만드는 것과 수행되는 action(transition-block)

## 다. 구 현

### (1)구현방법

프로그래밍 언어의 구현은 일반적으로 compiler방법, interpreter방법, 그리고 preprocessor 방법으로 나누어지는데, 본고에서는 시스템의 의존성이 가장 적은 preprocessor 방식으로 구현하였다. 이 preprocessor 기법은 원시(source) 프로그램을 번역하여 생성되는 tar-

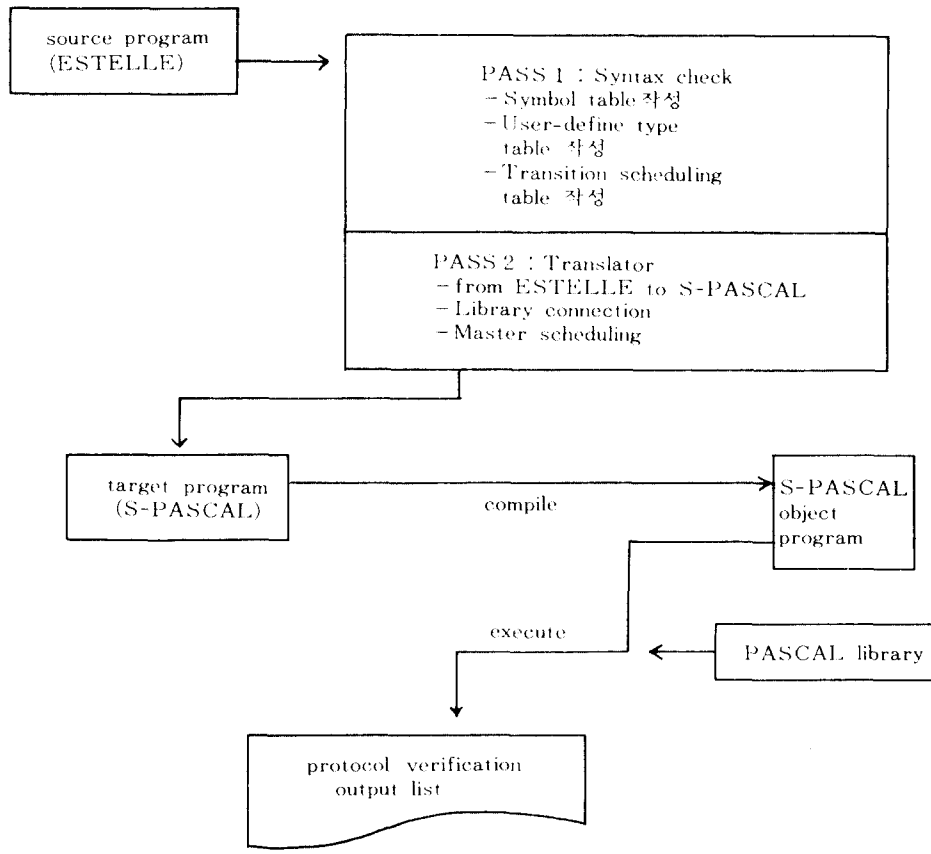



그림 5　ESTELLE의 처리과정.

get program이 기계어나 중간 코드 형태로 생성되는 것이 아니라 현재의 시스템에서 사용 가능한 다른 고급 언어로 된 프로그램을 생성하는 것이다.

여기서는 ESTELLE언어로 작성된 source 프로그램을 sequential PASCAL로 target 언어가 번역, 생성되도록 하였다. 그 전체적인 과정을 그림 5에 표시하였다.

(2) Implementation

ESTELLE 언어의 각 문장과 module은 다음과 같이 번역된다.

(1) procedure & function

PASCAL의 procedure와 function의 개념과 동일하며 1 : 1 대응으로 변환되며 body 부분이 없는 primitive나 pure로 선언된 형태의 고정적으로 사용되는 library로 하였다.

(2) Data declaration

ESTELLE언어의 data 선언 방법은 PASCAL과 동일하므로 그대로 옮겨지도록 하였고, system dependent한 부분은 (예제의 "…") 은 User가 정확한 값을 배정하도록 되어 있는 것이므로 문제가 없다. 그러나, PASCAL은 변수의 선언을 program 서두에서 하도록 되어 있는데 반하여 ESTELLE은 "Specification" body의 initialize 부분에서 "modvar"이라는 선언부가 존재한다. 이는 번역시 program의 서두 부분에 놓이도록 처리하였다.

(3) Channel

Channel은 By User와 By provider가 구분되어 acknowledge의 주고 받음을 나타내므로 Queue로 정의된 file에서의 signal 입출력으로 처리하였다. 그 이유는 상태전이 부분에서는 입력되어진 acknowledge에 따라 state의 변환이 이루어지기 때문이다.

(4) Module header

Module header에서 선언되는 내용은 현재의 module에서 사용할 channel과 queue의 형태, 매개변수들이다. 그러므로 필요한 channel procedure를 연결해주고, 사용할 queue의 형태가 individual인지 common인지에 따라서 acknowledge를 보관할 장소를 배정하도록 한다.

(5) Module body

모든 행동 양식이 기술된 부분으로서 PASCAL구조의 표현과 동일하며 다음과 같은 protocol specification을 정의하기 위한 문장들을 갖는다.

가) State와 stateset

상태전이의 한 단계씩을 표현하며 실행 시에는 transition scheduler table의 scheduling에 따라 수행되어진다.

나) Initialize

PASCAL로 전환된 부분으로서의 의미는 module 실행의 시작점을 나타내고, 이 action을 취한 후 transition scheduler table의 state를 가리키게 된다.

다) Trans

acknowledge에 따라 상태전이가 일어나도록 하는 부분으로서 각각의 transition을 procedure block으로 구성하여 처리하고, scheduling에 의해 각 state의 transition이 필요할 때마다 호출되도록 처리하였다.

라) When, Provide

When절과 Provide절의 의미는 상위 또는 하위 layer에서 queue에 입력된 acknowledge의 종류와 변수의 값의 조건에 따라 필요한 transition을 발생하도록 하는 것으로 scheduling의 기본항목이 된다.

마) Output

action부분의 한 명령으로서 현재의 layer에서 상위 또는 하위 layer로 acknowledee를 전송하고 동시에 상위 하위 layer에서 오는 결과 acknowledge를 queue에 제공받도록 한다.

### (3) ESTELLE 프로그램 예

ESTELLE syntax에 따른 ESTELLE 프로그램의 작성 예로서 가장 간단한 프로토콜인 A B (Alternating Bit) 프로토콜을 예로 들어 살펴보기로 한다.

A_B프로토콜이란 송신측에서 하나의 데이터를 전송한 다음에 수신측으로부터 해당 데이터에 대한 ACK (Acknowledge)를 받아 데이터의 전송 여부를 확인한 후에 그 다음 데이터를 송신하는 프로토콜이다. 이를 ESTELLE기법으로 프로그램하기 전에 시스템의 전체적인 구조와 A_B프로토콜 내부 구조를 모델화 하면 그림 6과 같다. 여기서 시스템이란 A_B프로토콜의 상하위 계층을 모두 포함한 하나의 개체를 말한다.

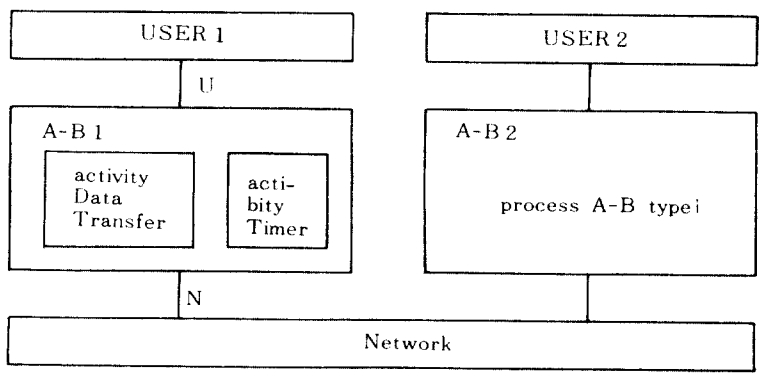



그림 6 A_B 프로토콜의 구조.

<ESTELLE SOURCE PROGRAM>

```
Specification Example systemprocess; timescale seconds;

const
    low          = any integer;
    high         = any integer;
    Retran_time  = any integer;

type
    Cep_type     = low..high;
    U_Data_type  = ...;
    Seq_type     = 0..1;
    Id_type      = (DATA,ACK);
    Ndata_type   =
        record
            Id   : Id_type;
            Conn : Cep_type;
            Data : U_Data_type;
            Seq  : Seq_type
        end;

channel U_access_point(User,Provider);

    by User:
       SEND_request(Udata: U_Data_type);
       RECEIVE_request;
    by Provider:
       RECEIVE_response(Udata: U_Data_type);

channel N_access_point(User,Provider);

    by User:
       DATA_request(Ndata: Ndata_type);
    by Provider:
       DATA_response(Ndata: Ndata_type);

module User_type process(Conn_end_pt_id: Cep_type);
    ip
       U: U_access_point(User) common queue;
end;

module Alternating_bit_type process(Conn_end_pt_id: Cep_type);
    ip
       U: U_access_point(Provider) common queue;
       N: N_access_point(User) individual queue;
end;
```

```
module Network_type process;
    ip
       N: array[Cep_type] of
          N_access_piont(Provider) individual queue;
end;

body Network_body for Network_type; external;

body User_body    for User_type;    external;

body Alternating_bit_body for Alternating_bit_type;

type
    Msg_type       =
        record
            Msgdata: U_Data_type;
            Msgseq : Seq_type
        end;
    Buffer_type    = ...;

channel S_access_point(User,Provider);
by User;
       TIMER_request;

by Provider;
       TIMER_response;

module Data_transfer_type activity
       (U: U_access_point(Provider) common queue;
        N: N_access_point(User)     common queue;
        S: S_access_point(User) individual queue;
                           param number: Cep_type);

module Timer_type activity
       (S: S_access_point(Provider) individual queue;
                           param Time: integer);

body Data_transfer_body for Data_transfer_type;
var
    Send_buffer, Recv_buffer: Buffer_type;
    Send_seq,    Recv_seq   : Seq_type;
    P,Q                     : Msg_type;
    B                       : Ndata_type;

state ACK_WAIT, ESTAB;

stateset
    EITHER = [ACK_WAIT,ESTAB];
```

```
function Ack_ok(Nd: Ndata_type): boolean;

begin
    Ack_ok := (Nd.Id = ACK) and (Nd.Seq = Send_seq);
end;

procedure Copy(var To_Data: U_Data_type; From_data: U_Data_type);
primitive;

procedure Empty(var Data: U_Data_type);
primitive;

procedure Format_data(Msg: Msg_type; var B: Ndata_type);

begin
    B.Id   := DATA;
    B.Conn := Conn_end_pt_id;
    copy(B.Data, Msg.Msgdata);
    B.Seq  := Msg.Msgseq;
end;

procedure Format_ack (Msg: Msg_type; var B: Ndata_type);

begin
    B.Id   := ACK;
    B.Conn := Conn_end_pt_id;
    empty(B.data);
    B.Seq  := Msg.Msgseq;
end;

procedure Empty_buf(var Buf: Buffer_type);
primitive;

procedure Store(var Buf: Buffer_type; Msg: Msg_type);
primitive;

procedure Remove(var Buf: Buffer_type);
primitive;

function Retrieve(Buf: Buffer_type): Msg_type;
primitive;

function buffer_empty(Buf: Buffer_type): boolean;
primitive;

procedure Inc_send_seq;

begin
```

```
   Send_seq := (Send_seq + 1) mod 2
end;

procedure Inc_recv_seq;

begin
   Recv_seq := (Recv_seq + 1) mod 2
end;

initialize

to ESTAB
   begin
      Send_seq := 0;
      Recv_seq := 0;
      Empty_buf(Send_buffer);
      Empty_buf(Recv_buffer);
   end;

trans

from ESTAB
  to ACK_WAIT
     when U.SEND_request
     begin
        copy(P.Msgdata,Udata);
        P.Msgseq := Send_seq;
        Store(Send_buffer,P);
        Format_data(P,B);
        output N.DATA_request(B);
        output S.TIMER_request;
     end;

from EITHER
  to same
     when U_RECEIVE_request
        provided not buffer_empty(Recv_buffer)
        begin
           Q := Retrieve(Recv_buffer);
           output U.RECEIVE_response(Q.Msgdata);
           Remove(Recv_buffer);
        end;

from ACK_WAIT
  to ACK_WAIT
     delay(Retran_time)
     begin
```

```
            P := Retrive(Send_buffer);
            Format_data(P,B);
            output N.DATA_request(B);
        end;

from ACK_WAIT
  to ESTAB
        when N.DATA_response
            provided Ack_ok(Ndata)
            begin
                Remove(Send_buffer);
                Inc_send_seq;
            end;

from ESTAB
  to ESTAB
        when S.TIMER_response
             begin
              (* Do nothing *)
             end;

from EITHER
  to same
        when N.DATA_response
            provided Ndata.Id = DATA
            begin
                copy (Q.Msgdata, Ndata.Data);
                Q.Msgseq := Ndata.Seq;
                Format_ack(Q,B);
                output N.DATA_request(B);
                if Ndata.Seq = Recv_seq then
                begin
                    Store(Recv_buffer,Q);
                    Inc_recv_seq
                end
             end;
end;

body Timer_body for Timer_type;
var
       Stop, Stop_bis : boolean;

initialize
       begin
          Stop := true;
          Stop_bis := true;
       end;
```

```
trans
when S.TIMER_request;
       begin
          Stop := true;
          Stop_bis := false;
       end;

trans
provided not Stop_bis
       begin
          Stop_bis := true;
          Stop := false;
       end;

provided not Stop
    delay(Time,Time)
       begin
          Stop := true;
          output S.TIMER_response;
       end;

var
   Data_transfer : Data_transfer_type;
   Timer : Timer_type;

   initialize
   begin
       init Data_transfer with Data_transfer_body(Name);
       init Timer with Timer_body(Retran_time);
       connect Data_transfer.S to Timer.S;
       attach U to Data_transfer.U;
       attach N to Data_transfer.N;
   end;
end;

modvar

   User:array[Cep_type] of User_type;
   Alternating_bit:array[Cep_type] of Alternating_bit_type;
   Network: Network_type;

initialize

begin
   init Network with Network_body;
   all Cep: Cep_type do
      begin
         init User[Cep] with User_body(Cep);
         init Alternating_bit[Cep] with
```

```
                Alternating_bit_body(Cep);
            connect User[Cep].U to Alternating_bit[Cep].U;
            connect Alternating_bit[Cep].N to Network.N[Cep];
        end;
    end;
    end.
```

### (4) 프리프로세서를 통한 번역의 예

```
channel U_access_point(User, Provider);
    by User :
       SEND_request (Udata:Data_type);
       RECEIVE_request;
    by Provider :
       RECEIVE_response(Udata:Data_type);

channel N_access_point(User, Provider);
    by User :
       DATA_request (Ndata : Data_type);
    by Provider :
       DATA_response(Ndata : Data_type);
```

전환⇨

```
procedure SEND_request(var SEND_request_ok:boolean; var Udata:Data_type);
Begin
      WRITE(' SEND_request(T/F)?');
      READLN( SEND_request_ok);
 END;
procedure RECEIVE_request(var RECEIVE_request_ok:boolean);
Begin
      WRITE(' RECEIVE_request(T/F)?');
      READLN( RECEIVE_request_ok);
 END;
procedure RECEIVE_response(var RECEIVE_response_ok:boolean; var Udata:Data_type);
Begin
      WRITE(' RECEIVE_response(T/F)?');
      READLN( RECEIVE_response_ok);
 END;

procedure DATA_request(var DATA_request_ok:boolean; var Ndata : Data_type);
Begin
      WRITE(' DATA_request(T/F)?');
      READLN( DATA_request_ok);
 END;
procedure DATA_response(var DATA_response_ok:boolean; var Ndata : Data_type);
Begin
      WRITE(' DATA_response(T/F)?');
      READLN( DATA_response_ok);
 END;(*--  *)
```

```
trans
  from ESTAB
       to ACK_WAIT
          when U.SEND_request
               begin
               P.Msgdata := Udata;
               P.Msgseq  := Send_seq;
               Store(Send_buffer,P);
               Format_data(p,b);
               output N.Data_request(B);
               output S.TIMER_request
               end;
  from EITHER
       to same
          when U.RECEIVE_request
               provided not buffer_empty(Recv_buffer)
          begin
               Q := Retrieve(Send_buffer);
               output U.RECEIVE_response(Q.Msgdata);
               Remove(Recv_buffer,Q)
          end;
  from ACK_WAIT
       to ACK_WAIT
          when S.TIMER_response
          begin
               P := Retrieve(Send_buffer);
               Format_data(P,B);
               output N.DATA_request(B);
               output S.TIMER_request
          end;
```

전환⇨

```
ESTAB:

SEND_request(SEND_request_ok);
if SEND_request_ok
   THEN
BEGIN
P.Msgdata   :=   Udata;
P.Msgseq    :=   Send_seq;
Store(Send_buffer,P);
Format_data(p,b);
Data_request(Data_request_ok ,B);
TIMER_request(TIMER_request_ok );
      GOTO ACK_WAIT
END;

EITHER:

RECEIVE_request(RECEIVE_request_ok);
if RECEIVE_request_ok

   AND  NOT     buffer_empty(Recv_buffer)
   THEN
BEGIN
Q      :=     Retrieve(Send_buffer);
RECEIVE_response(RECEIVE_response_ok ,Q.Msgdata);
Remove(Recv_buffer,Q);
      GOTO ACK_WAIT
END;

ACK_WAIT:

TIMER_response(TIMER_response_ok);
if TIMER_response_ok
   THEN
BEGIN
P      :=     Retrieve(Send_buffer);
Format_data(P,B);
DATA_request(DATA_request_ok ,B);
TIMER_request(TIMER_request_ok );
      GOTO ACK_WAIT
END;
```

## 6 결 론

본 고에서는 부분 ESTELLE 언어를 개인용 컴퓨터에 개발함으로써 통신 관련 연구소와 학계는 물론, 국내 관련 기업체에서 이를 사용함으로써 다음과 같은 효과를 얻으리라 예상된다.

### 가. Verification tool 로서 FDL

어떤 protocol통신 제품에 대해서 실제의 표준 protocol 규격과 동일한지, 또는 동일한 성질을 가지고 있는지를 검증하는 showing equivalence와 property proving을 하는데 도움을 얻을 수 있다. 이는 대상 protocol 설계에 있어서 종래에는 하드웨어에 의존하여 검증을 하였으나, 이제까지 기술한 FDL-ESTELLE 로서 Specification을 작성하고 simulation 해봄으로써 cost나 시간적인 면, 아울러 document 의 측면에서 커다란 가치가 있다고 본다.

### 나. User efficiency 측면에서의 FDL

ESTELLE 언어로 protocol spec. 을 기술하려면 우선 실제의 protocol 과 통신및 그 관련된 구조에 대해서 확실한 이해를 가지고 있어야 하고, ESTELLE 자체의 Syntax도 알아야 한다는 이중부담이 있다. 그러나, 통신 protocol을 설계하는 사람의 입장에서는 이미 protocol spec. 정의를 위하여 기존의 FDT를 사용하고 있으므로 protocol 정의의 문제는 없다고 본다. 다만 ESTELLE의 Syntax를 숙지하여야 하는 문제가 남는데 이 언어는 PASCAL 과 구조가 동일함으로 기존 언어의 범주를 벗어나지 않는다.

그러므로, FDL을 이용한 protocol specification의 작성은 readability를 높일 수 있고, 검증까지 행함으로써 reliability를 높일 수 있다는 장점이 있다.

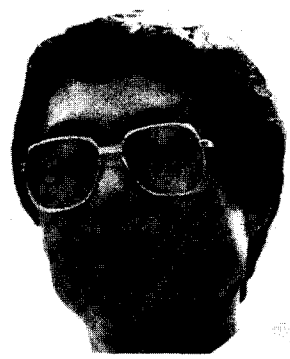
### 3. Formal Description용으로서의 Language

현재 이미 발표되어 사용중이거나 개발-구현중인 통신프로토콜 제품들은 그의 규격서를 설계 또는 개발-구현하는 측의 특성이나 기법에 따라 Informal 한 방법을 사용하고 있다. 그러나 CCITT나 ISO의 표준규격안과는 일치되지 않는 부분을 찾아볼 수 있다. 따라서 각종 제품의 규격 작성에 있어서 ESTELLE고 같은 E-DL로서 작성을 하면 작성한 protocol specification program 자체가 document가 되어 어디서나 동일한 규격서로 이용될 수 있다는 것이다.

## 참 고 문 헌


1. T. Blumer & R.L. Tenny, "An automated formal specification technique for protocols", Computer Networks, Vol.6, pp.201-217, 1982.
2. C.A. Vissers, R.L. Tenny, G.V. Bochmann, "Formal Description Techniques", IEEE, Vol.71, No.12, Dec. 1983.
3. R.J. Linn, "The Features and Facilities of ESTELLE: a formal Description Technique based on an Extended Finite State Machine Model", proceedings of the IFIP WG6.1 Fifth International Workshop On Protocol Specification, testing and Verification, V, June 1985.
4. [ISO/TC97/SC21/N4221, "Information Processing Systems-Open system Interconnection-ESTELLE-A Formal Description Technique Based on An Extended State Transition Model", June 1985.
5. [ISO/TC97/SC21/319], "A First Draft of A Session Formal Description in the Language ESTELLE", Jan. 1985.
6. [ISO/TC97/SC21/N933], "Guidelines for The Application of Formal Description Techniques to OSI", Dec. 1985.
7. [ISO/TC97/SC21/N9371], "Provisional ESTELLE tutorial", Dec. 1985.
8. [ISO/TC97/SC21/N2125], "Revised Text of ISO/DP9074, Information Processing System-Open System Interconnection-ESTELLE-A Formal Description Technique Based on An Extended State Transition Model", Mar. 1987.

9. 원유헌, 이종창, "RPG컴파일러디자인"홍대 논총 제12집 pp. 375-385, 1980.
10. Ghezzi, C. & Jazayeri, M., "Programming Language Concepts", New York: J. Wiley, 1983.
11. Terrence W. Pratt, "Programming Languages Design and Implementation", Prentice Hall, 1984.
12. Aho Seth Ullman, "Compilers, Principles, Techniques, and Tools", Addison Wesley, 1986.
13. N. Wirth, "Data Structure + Algorithm = Programs", Prentice Hall, 1976.
14. Roland C. Backhous, "Syntax of Programming Languages", Prentice Hall, 1979.
15. Barron, D.W., "Pascal: Language and it's implementations", Prentice Hall, 1980.
16. 원유헌, "프로그래밍 언어 개념", 정익사, 1983.
17. IBM, "MS-DOS 3.30 Handbook"
18. Al Kelley & Ira Pohl, "A BOOK ON C", Benjamin/Cummings Publishing Co., 1984.
19. Martin D. Davis & Elaine J. Weyuker, "Computability, Complexity, and Languages", Academi Press, INC. 1983.
20. Bjarne, "The C++ Programming Language", Addison Wesley, 1986.



元 裕 憲

저자약력

- 1985 : 고려대학교 대학원 이학박사(프로그래밍 언어론 전공)
- 1981~1982 : 충남대학교 교환교수
- 1986~1987 : 미국 R.P.I 대학 POST DOC.
- 1988~현재 : 홍익대학교 공과대학 전자계산학과 부교수

印 素 蘭

저자약력

- 1978 : 홍익대학교 전자계산학과 이학사
- 1982 : 홍익대학교 대학원 전자계산학과 이학석사
- 1988~현재 : 홍익대학교 대학원 전자계산학과 박사과정 재학
- 1988~현재 : 한국전자통신연구소 접속처리연구실 선임연구원

郭 鎬 榮

저자약력

- 1983 : 홍익대학교 전자계산학과 이학사
- 1985 : 홍익대학교 대학원 전자계산학과 이학석사
- 1988~현재 : 홍익대학교 대학원 전자계산학과 박사과정 재학

**♧용 어 해 설♧**

●**인덱스 레지스터**(index register) : 프로그램 수행중 다른 데이터나 명령어를 추출하고자 할 때에 오퍼랜드의 내용을 변경하기 위한 수치를 기억하는 레지스터를 가리킨다.

●**인덱스 마아커**(index marker) : 자기드럼, 자기 디스크 장치에서 트랙의 시작을 나타내는 물리적인 표시로서 프로그램으로는 판독 및 기록을 할 수가 없다. 예를 들면 자기 디스크 위의 전트랙은 이 인덱스 마아커를 검출함으로써 트랙의 시작을 알 수 있다.

●**인라인 프로세싱**(in-line processing) : 통신 제어 장치나 온라인 제어 프로그램을 거치지 않고 단말에서 데이터를 넣어 문의를 행하는 처리방식이다. 인 하우스(inhouse)처리라고도 한다.

●**인버어트 회로**(inverter circuit) : 전력 변환 장치의 일종으로 직류 전력을 교류 전력으로 변환하는 역변환 회로이다. 논리 회로에서는 1개의 입력 단자와 1개의 출력 단자를 가지며 2진 신호중 어느 한쪽(0또는 1)을 입력 단자에 첨가하면 출력 단자에 부정적인 신호(1또는 0)를 출력하는 회로이다.

●**인쇄 컨덴서**(printed capacitor) : 인쇄 배선을 이용하여 만든 컨덴서. 베이클라이트 기판 상에 빗 모양의 전극을 마주 보게 하여 맞물리게 배치한 소용량의 것과 기판에 티탄산 바륨 자기를 써서 그 양면에 전극을 인쇄한 대용량의 컨덴서 등이 있다.