論文96-33A-4-6

ATM 전달망 호제어 소프트웨어의 도메인 분석과 객체지향 설계

(A Domain Analysis and Objected-oriented Design of Call Control Software in ATM Transport Network)

金 漢 慶 *, 具 然 卨 **

(Han-kyoung Kim and Yeon-seol Koo)

요 약

ATM교환기 호제어 소프트웨어의 실현을 위하여 호제어 문제영역을 도메인으로 분류하고 각 도메인을 규격화하기 위하여 객체지향 분석기법을 도입한다. 객체의 집단으로 도메인을 규격화하는 방법을 ATM교환기 호제어 소프트웨어 실현을 통하여 구체적으로 검토한다. 이벤트와 조건을 시스템 invariant로 활용함으로서 도메인 사이의 관계를 규정하고 소프트웨어 설계 및 실현에 필요한 요구사항을 규격화한다. 또한 이러한 도메인들 사이에 존재하는 관계성을 도출하여 정형화시킴으로써 검증을 가능케 하는 소프트웨어 요구사항에 대한 설계기법을 제시한다.

Abstract

For the implementation of call control software, it will be shown how to divide problems into domain and also presents methodologies to specify domains according to the object oriented analysis techniques. Domains is specified by the set of related objects as shown in call control examples. Events and conditions are able to be transformed into system invariant so that is possible to figure out the relationship between domains and those concludes that the software requirements specification can be established for the design and implementation. Relationships existing between domains can be formalized so as to verify software requirements.

I. 서 론

문제영역을 설명하기 위해서, 그것을 보다 조그만 단위로 분해하여 논리적인 집단을 형성하는데, 그와 같은 집단은 공통의 파라다임(paradigm)을 갖는다. 파라다임은 집단의 활동의 범위를 결정하는데 여기서 범위라 함은 어떤 성질이 이 집단에 있는가, 어떤 활동이 포함되는가를 뜻한다[1]. 즉, 문제영역은 도메인으로 분해되며 각 도메인은 외적인 특징에 의해 구분된다. 이것은 도메인 속에 존재하는 각 부분들 사이의 응집력을 도메인으로 나타내거나 도메인으로 결정하지 않는 것을 의미한다. 바꾸어 말하면, 개체의 집합은 도메인인데 이는 어느 한 집단이 실세계의 어떤 모습을 모델링하는 데 유용한 것으로 받아들이는 어떤 범주까지이다[2].

교환기의 호제어라는 문제 영역을 규격화하기 위하여 그 규격의 범위를 어떻게 분류하고, 또 그것을 어떻게 명세화할 것인가 하는 문제를 본 논문에서 검토해 보고자 한다. 각 도메인이 자신의 파라다임을 갖듯이, 주어진 전체의 문제 영역인 시스템의 파라다임 또한 제시됨으로써 도메인 상호간에 존재하는 관계성이 정

* 正會員, 韓國電子通信研究所
(Electronics and Telecommunications Research Institute)
** 正會員, 忠北大學校 컴퓨터科學科
(Chungbuk National University, Depart of Computer Science)

接受日字: 1995年11月9日, 수정완료일: 1996年3月21日

리되어야 한다.

. 객체지향의 분석과 설계를 위하여 가장 많이 적용하는 OMT 방법론에 있어서, 그 절차는 객체 모델을 먼저 설정하고 이를 반복적으로 요구사항에 맞추어 봄으로서 보완하도록 제시하고 있으며, 객체 모델이 일단 제시되면 이 객체 모델에 동적 모델을 적용하여 외부로 부터의 자극과 반응에 대한 이벤트를 설정하고 상태 다이아그램을 구축한 다음 이벤트와 객체 사이의 적합성을 점검한다[3]. 객체 모델과 동적 모델이 구축되면 기능적인 흐름을 점검함으로서 기능적인 모델이 설정된다[3]. 이러한 분석과정은 단일의 응용 분야에서는 적절한 것으로 생각되지만 교환기 소프트웨어에서의 호제어 기능을 분석하고 설계함에 있어서, 요구사항 사이에 불일치가 존재하는 불완전성과 요구사항을 지속적으로 바꾸고 새로운 것이 추가되는 경향을 수용하고, 여러 분야를 여러 관점으로 기술하는 복잡다양한 환경적인 요건에서는 적용하기가 어렵다.

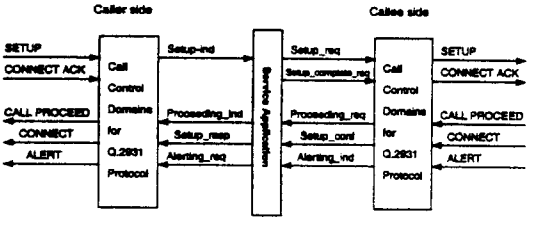
전화교환에 대한 객체지향의 설계는 교환이라는 단순한 기능을 객체 사이의 관계성 개념으로 설계함으로서 교환 소프트웨어의 객체지향적 설계를 제시하고 있다. 이러한 설계는 객체간의 관계성 설정을 고착화함으로서 향후 기능을 추가해야할 경우 또는 사용자 규격이 변경되어 소프트웨어를 보완해야할 경우, 객체지향 소프트웨어의 장점인 유연성과 재사용성의 잇점을 살리지 못하는 문제점을 드러낸다. 본 논문에서는 기존의 객체의 개념에 상태와 이벤트 그리고 시나리오 를 새롭게 추상화함으로서 이러한 새로운 개념을 수용하는 교환 소프트웨어 구조를 제시하고, 이를 지원하는 객체지향의 교환 소프트웨어 설계방법을 제시하고자 한다.

2장에서는 호제어라는 문제영역에 대하여 도메인으로 분류하는 방법과 각 도메인의 명세화 기법을 고찰한다. 3장에서 호제어 요구사항을 도메인으로 분류하여 상태기반의 정형화 방법을 이용하여 규격화를 시도하고 시스템의 설계 및 구현 방향을 구체화한다. 이와 같은 시스템의 구현에 따라 시스템의 명세화를 위한 모달 논리와 그 적용 내용에 대하여 4장에서 언급하였다. 5장에서는 이와 같은 분석과정과 시스템 설계과정을 방법론으로 정리 제시하고 마지막으로 결론을 내린다.

Ⅱ. 호제어 문제영역의 분석

1. 문제영역과 도메인

ATM교환기의 호제어 소프트웨어를 실현하기 위한 문제영역을 본 논문에서는 2가지로 정한다. 하나는 Q.2931에 근거한 프로토콜과 그 상위의 서비스에 대한 것이며 두 번째는 전용선 서비스 기능을 담당하는 것이다. Q.2931에 대한 설명은 그림 1로 표현되어 있으며, 두 번째 문제에 대하여는 그림 2에 나타나 있다[4]. 이 2가지 문제영역을 자세히 고찰하면 각기의 문제는 2가지의 계층으로 분리가 됨을 알 수 있다. 즉, Q.2931의 프로토콜과 HMI(Human Man Interface)의 통신 계층과 통신 계층 위에서 요구되는 서비스를 처리하는 서비스 계층이 그것이며, 서비스 계층 상에서 제공되는 각종 기능은 2가지의 문제 영역에 속하는 공통의 자원을 적절히 배분하고 조정되어야 함을 유추할 수 있다. 이와같은 서로 다른 문제영역을 동시에 고려해야하는 것은 이들 두 문제영역이 동일한 시스템 자원을 사용하게 되고 이들 시스템 자원은 객체로 쉽게 정의가 되기 때문에 하나의 문제 영역에서 파생되는 다수의 도메인에 대한 분석이 상호 의존적임을 보여준다. 이러한 사례는 우리로 하여금 쉽게 문제 영역을 어떻게 도메인으로 분리를 할 수 없을까라는 의문을 일으키게 한다. 또 문제 영역을 개발의 초기단계에서 세분하여 도메인으로 분리가 이루어진다면 시스템 분석가로 하여금 분석을 더욱 정확하게 할 수 있다[5].

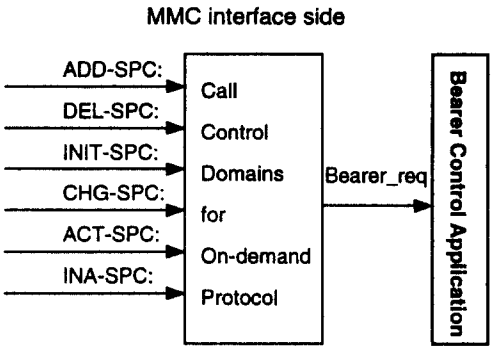


* RELEASE, STATUS, NOTIFY 등 프로토콜 설계에 영향이 적은 것은 제외함.

그림 1. Q.2931을 기반으로 하는 호제어 문제영역
Fig. 1. Q.2931-based call control problem domain.

그림 1과 그림 2에는 목적 시스템의 외부로 부터 입력되고 외부로 출력되는 입출력의 종류를 위에서 설명한 2가지의 계층 구조 엔티티에 맞추어 표시하였다. 입출력 종류로는 프로토콜 프리미티브, 프로토콜 메시지, 또는 운영자에 의한 입출력 명령어 등이 있다. Q.2931은 ITU-TS SG11에서 권고하는 광대역 점대점 호 서비스 제공을 위한 프로토콜 계층의 표준화 내용이며 이에 따라 광대역 통신망의 서비스가 제공된다.

Q.2931에 대한 입력은 시스템 외부로 부터 입출력되는 메시지와 시스템 내부의 서비스 계층과 프로토콜 계층 사이에 주고 받는 프리미티브로 구분된다. 외부 메시지 이름은 대문자로 표시되고 프리미티브는 대문자로 시작하나 소문자로 구성되어 있다. 이들 메시지는 정보요소를 포함하고 있어 제어정보의 교환이 가능하게 되어 있으며, 프리미티브는 시스템 내부에서 구현하는 방법에 따라 틀리게 나타난다. 그 실현 방법으로는 IPC, RPC, function call 등이 있을 수 있다.



* Bearer_req 는 internal protocol message를 총칭하여 표시함

그림 2. 전용선 서비스를 위한 호제어 문제영역
Fig. 2. Call control problem domain for semi-permanent connection service.

Q.2931을 살펴보면 프로토콜 기능에 대하여 규격화되어 있으며 이 프로토콜 위에 실현될 사용자 서비스는 언급하지 않음으로서 시스템 개발자에 의하여 다양한 서비스를 개발하여 적용케 함으로서 서비스 제공자에게 다양한 서비스를 제공할 수 있도록 하였다. 이에 따라 자연스럽게 교환 소프트웨어를 프로토콜 계층과 서비스 계층의 소프트웨어 분야로 나누도록 하며 이것은 다른 AFFIRM 프로젝트에서의 분석과 설계를 참조하였다[6].

그림 2의 내용은 동일한 광대역 망에서 제공되는 전용선 서비스 기능에 대한 입출력 관계를 BNF 형태로 표현한 것이다. 이러한 입출력은 Q.2931 메시지가 시그날의 형태를 취하는 것과는 달리 입출력 명령어의 형태를 취하지만 이를 받아 처리하는 프로토콜 계층에서 상위의 서비스 계층과의 인터페이스에서는 통일된 내부 프리미티브의 형태를 취하게 된다. 전용선 서비스에 대한 이벤트는 운영자로 부터 입력되는 Man Machine Communication 명령어인데 이에 대한 문법은 그림 3과 같다[7].

```
Verb-Objective [ -Complement ] :
                    [ parameter, ] + { parameter ] ;
  Verb ::= { ADD | DEL | INIT | CHG | ACT | INA }
  Objective ::= { SPC }
  Complement ::= { [FIX | RES ] }
  parameter ::= {PCR_value | VPI_value |
              Originating_address | Destinating_address
               | [Start_time ] | [End_time ] }
```

그림 3. 전용선 기능을 위한 MMC 명령어의 구성
Fig. 3. Format of MMC commands related to the leased line service.

Q.2931 처리 기능과 전용선 서비스 기능은 입력이 서로 틀리며 서비스의 대상도 서로 틀린다. 그럼에도 이들이 하나의 문제로 분석되어야 하는 것은 이러한 입력에 대하여 서비스 계층에서 이루어지는 각종의 행위들 사이에 시스템 내부적으로 관계성이 존재하기 때문이다. 관계성의 표현에 대하여 계속 검토한다.

2. 도메인 규격

도메인 분석의 목적은 Arango에 의하여 제시된 바와 같이 문제영역에 대한 모델을 만들기 위한 것이며 이를 위하여 모델에는 최소한 문제영역에 대한 다음의 3가지 정보를 포함하고 있어야 한다. 즉, (1) 그 영역에 있어서 시스템의 규격화를 가능하게 하는 개념, (2) 규격화된 것을 코드로 변환(map)하기 위한 계획, (3) 규격 개념에 관계있는 것들, 그들간의 관계, 실현계획과의 관계 등이 그것이다[8]. 여기에서 가장 기본이 되는 것은 시스템의 규격화를 가능케 하는 개념임을 알 수 있으며 이것에서 부터 시작되어 구축되는 정보들은 도메인 모델의 중요한 개념이며 도메인 규격의 기초가 된다. 도메인 규격은 도메인 모델을 이용하여 문제 분석을 하거나 또는 나중에 재사용 가능한 부품을 적용하고자 할 때, 애매모호한 사항이 발생하면 이에 대한 통일되고 결정적인 참고자료로 활용하기 위한 것이며, 교육 또는 전달을 위한 공유의 지식을 모아두는 저장고로 활용하여 재사용 부품을 적용하는 사람들에게 규격서로 활용을 할 수 있다. 이것은 시스템의 재사용 기반구조를 구축하는 것과 각 객체의 규격화가 한꺼번에 정리되어야 한다. 이를 규격화하는 방법론으로 대수적 명세기법을 이용한다.

대수적 명세기법을 선호하는 객체의 추상화 시각은 특정 언어의 특성을 피하고 모듈화 시키는 명세에 집

착한다[9]. 반면 도메인 명세의 중요한 착안점은 문제 영역에 대한 도메인의 구조적인 기술이다. 어떻게 구조적으로 구축할 것인가 하는 문제는 사용하는 언어와는 무관하게 정의되므로, 동일한 signature를 갖는 두개의 명세를 통합할 때 오퍼레이션을 고려하는 것과 같다. 이것은 도메인이 동일한 어드레스 공간상에서 각기의 객체를 통합하는 것을 상정한 것이며, 이와 같은 경우에는 도메인의 통합이 다음과 같이 이루어진다. 두개의 명세를 $S_1 = (\Sigma, \Phi_1)$, $S_2 = (\Sigma, \Phi_2)$라고 하고 각각은 하나의 signature와 axioms들로 구성되었다고 하자. 또 사용하는 model을 *Models* [S_1], *Models* [S_2]라고 하면 그들의 union인 $S_1 + S_2$는 그림 4와 같이 정의된다[9].

Signature [$S_1 + S_2$] = Σz
Axioms [$S_1 + S_2$] = $\Phi_1 \cup \Phi_2$
Models [$S_1 + S_2$] = *Models* [S_1] $\cap$ *Models* [S_2]

그림 4. 객체와 객체의 통합
Fig. 4. Combining of objects.

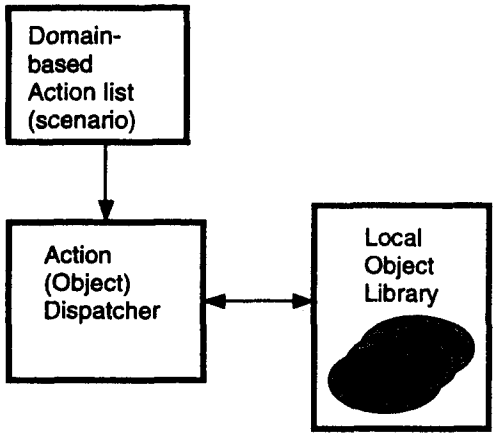


그림 5. 도메인 기반의 소프트웨어 구조
Fig. 5. Software struecure based on domains.

상기와 같은 새로운 시스템의 기반구조를 정리하고 필요한 객체를 규격화하여 재사용성을 향상시키기 위한 일련의 작업을 위하여 ATM교환기의 소프트웨어 시스템 구조를 본 논문에서는 지식공학적인 접근으로 해결하였으며, 이를 위해 그림 5에서 보여주는 바와 같이 객체 dispatcher, 객체 sequence scenario와 객체 라이브러리를 구축하였다. 객체 사이의 관계성을 시나리오를 통하여 구축하고 시나리오에 나타나는 각각의 객체를 객체 라이브러리에 실장시킴으로서 객체 처리기를 통하여 재사용이 가능케 하였다.

도메인을 규격화하기 위하여 주어진 이벤트마다 시스템이 동작하여야 할 시나리오를 BNF 표기법으로 나타내면 그림 6과 같다. 시나리오에 의해 정형화된 Action들은 Action dispatcher에 의하여 정해진 순서에 따라 객체가 invoke되고 수행되면서 도메인의 기능이 처리된다. 이때 도메인 내의 객체가 공유하는 signature는 local object library의 main에 위치시켜서 공유한다[10].

```
Terminals = {Object_name, Scenario_name, Null,
             End_of_sequence, P, T, J}
Scenario ::= {Condition, [Object_name |
             Scenario_name | Null ]}+
Condition ::= {Object_ind, Scenario_ind,
             End_of_sequence_ind}
Object_ind ::= {P_type | T_type}
             /* type of correspondent object */
Scenario_ind ::= {J_type}
End_of_sequence_ind ::= {End_of_sequence}
             /* end of scenario */
P_type ::= {P}     /* procedural object */
T_type ::= {T}     /* test stub object */
J_type ::= {J}     /* link of new scenario */
```

그림 6. 시나리오의 구성
Fig. 6. Scenario structure in BNF.

3. 복수 도메인 시스템

서로 다른 논리를 갖는 도메인들과 또 서로 다른 어드레스 공간에서 동작하는 도메인들 사이에 자원과 오퍼레이션을 공유하고 이들 사이에 관계성을 정의함으로써 상호 연결하여야 할 필요가 있다. 교환기 호제어 문제 영역에서 Q.2931 프로토콜에 의한 호 서비스 영역과 On-demand에 의한 전용선 서비스 제어 영역은 별개의 도메인으로 구분된다. 이것은 서비스의 적용 분야와 사용자가 서로 다르며 시스템에 실장 시키는 요구도 서로 틀리기 때문이다. 그럼에도 이들 사이에는 자원의 공유가 불가피한데 이것은 그림 7에서 보는 바와 같이 이 두 가지 도메인이 가상경로라는 물리적인 매체를 공유하고 이에 대한 정보가 각기 필요하기 때문이다.

두개의 서로 다른 도메인이 공유하는 정보들로서 CallRegister, CallReferenceNumber, Bandwidth를 그림 7의 예에서 보여 주고 있으며, 이외에도 자원객체 정보 즉, 가입자 정합모듈과 가입자의 상태, 스위치의

상태, 중계선 장치와 중계선로의 상태, 시스템의 In-Service 여부, 최대 셀 전달율, 가상경로 및 가상 채널 번호 등이 있다. 이들에 대한 정의를 내리고 또 이를 참조하기 위해서는 공통의 엔티티가 필요하다. 이러한 엔티티는 매개변수들을 추상화하여 객체 클라스로 정의하고, 시스템의 현재 조건을 객체 클라스로 정의하게 되는데 이러한 객체는 그 상태에 따라 시스템의 오퍼레이션에 영향을 준다. 즉, 이러한 객체는 그 상태가 시스템의 조건을 나타내는데 이러한 조건이 시스템의 invariant로서 작용한다. 이 invariant는 이벤트의 발생시 그 처리 조건을 형성하게 되고 이를 통하여 각각의 도메인이 상호 관계를 정립해 간다. 이것은 이벤트와 시스템 invariant에 의하여 시스템의 문제영역을 일정한 기능으로 분류가 가능하고 이 기능에 대하여 규격화가 이루어질 수 있다. 이 기능 규격을 실현할 대상으로 하여 처리 절차를 명세화하는 것이 시나리오의 작성이다. 바꾸어 말하면, 시나리오를 통하여 객체의 실행 순서, 추상화된 매개변수를 이용한 객체 사이의 관계성을 나타내었다. 이것을 지원하기 위한 소프트웨어의 기반구조를 그림 8과 같이 구축하였다.

```
DO WHILE (System_Status = InService);
    CASE event OF
        (SETUP) : IF CallRegister is not
                      available
                  THEN Call Object_dispatcher
                      (Scenario_CallRegisterError);
                  ELSE Call Object_dispatcher
                      (Scenario_Setup);
                  FI;
        (CONNECT) : IF CallReference is not
                        correct
                  THEN Call Object_dispatcher
                      (Scenario_CallReferenceError);
                  ELSE Call Object_dispatcher
                      (Scenario_Connect);
                  FI;
        (ADD-SPC) : IF Bandwidth is not
                        available
                  THEN Call Object_dispatcher
                      (Scenario_BWNotAvailable);
                  ELSE Call Object_dispatcher
                      (Scenario_AddSpc);
                  FI;
        (defaults) : /* given for other events */
    ESAC;
OD;
```

그림 7. Event & Condition Check 알고리즘
Fig. 7. Algorithm of Event & Condition Check.

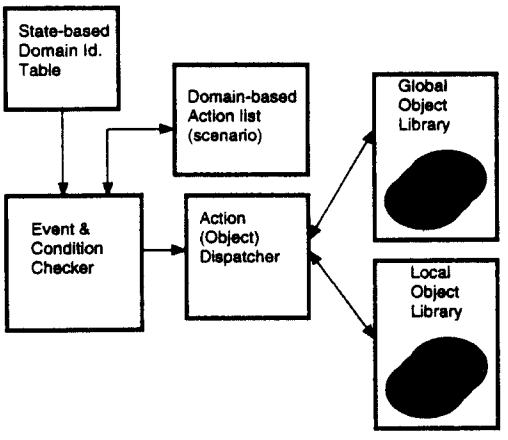


그림 8. 도메인 기반의 소프트웨어 구조
Fig. 8. Domain-based software infra-structure.

도메인의 분류를 위하여 Event & Condition Checker(ECC) 객체를 두고 여기에서 시스템의 상태에 따른 도메인의 분류를 Domain Identification Table(DIT)을 통하여 이루어진다. 이것은 도메인의 증가에 따른 로직의 증가를 최소화하고 객체의 상태 증가만으로 수용이 된다. 본 소프트웨어 구조는 지식을 공학화하는데 도메인 내에서는 객체의 통합을 오퍼레이션의 통합과 signature의 공유를 기본 개념으로 하고 있으며 도메인의 통합은 본질적으로는 시스템 signature를 정의하고 이를 도메인 상호간에 공유하게 한 것이다. 이것은 이러한 공유 signature들이 시스템 invariant로서 기능을 담당하며 나아가 시스템 검증의 조건이 된다.

Ⅲ. 시스템 규격

지금까지 문제 영역에 대한 도메인으로 분리하는 방법론과 이를 지원하는 소프트웨어의 기반구조를 제시함으로서 교환기에서의 호제어 기능에 관련된 객체의 수용이 가능하도록한 그 기초를 검토하였다. 즉, 문제영역이 도메인으로 구분이 되고 구분된 도메인 별로 외부로 부터 입력되는 자극을 이벤트로 구분하고 이들에 따라 시나리오를 형성하여 규격화를 시도함으로서 시나리오의 기본 구성품인 객체를 정의하고 수용할 수 있는 소프트웨어의 기반구조를 설정하였다. 여기에서는 외부로 부터 입력되는 이벤트와 이를 명세화하는 기본 요소들에 대해 검토하고 이들을 통하여 규격화하는 방

법론에 대하여 검토한다.

1. 이벤트와 조건

시스템으로의 입력은 환경의 변화 즉 버튼을 누르는 것과 같은 일련의 외적인 조건을 규정한 것이다[5]. 이러한 조건은 비록 몇 가지의 boolean 변수의 조합으로 표현되는 수도 있지만 대부분은 단일 boolean으로 표현된다. 시스템의 동작은 이벤트와 조건 값의 변화에 의해 정의되고 제어된다. 이벤트는 외부로 부터 주어진 자극이며, 이 자극이 시스템에 의미를 갖기 위해서는 조건 값이 만족될 경우이다. 환경의 변화는 이벤트이며, 시스템 내부에서 환경을 수용할 조건이 만족되면 이벤트는 그것이 발생된 그 시간, 장소에서만 검출되고 제어가 가능하다[11]. 예를 들면, 이벤트 @T은 이벤트 값이 false에서 true로 변경되는 것을 규정한다. 같은 의미로 이벤트 @F는 이벤트 값이 false로 변경되는 것을 규격화한다. 이러한 이벤트에서 @T를 이벤트의 기동조건(triggered condition)이라고 부른다. 이벤트의 발생이 다른 조건들의 값에 영향을 받을 수도 있다. 즉, @T(*Cond*)에 대한 표현은 주어진 조건 *Cond* 이 true일 때만 @T가 기동조건으로서 의미가 있다는 것이다. 복잡한 이벤트는 boolean 연산자를 사용하여 좀 더 단순한 이벤트와 조건으로 표현될 수 있다.

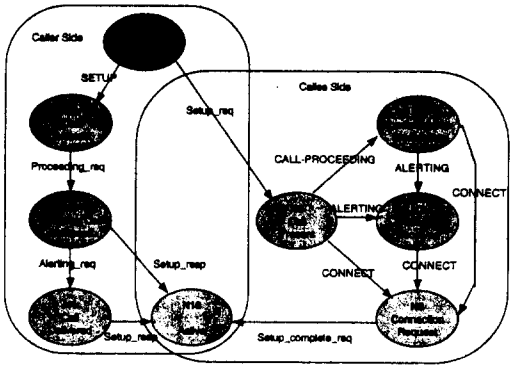


그림 9. Q.2931호 상태 천이도
Fig. 9. Call state transition diagram of Q.2931.

그림 9는 Q.2931의 이벤트에 대한 교환기 호에 대한 상태의 전이를 상태 다이아그램으로 표현한 것이며 이것을 형식화시켜 표현한 것이 표 1인데, 어느 한 상태에서 2개의 전이가 유발되는 경우와 하나의 이벤트에 의해 2개의 상태로 전이를 유발하는 상태의 비결정성

표 1. 호 상태 전이표
Table 1. Call state transition table.

현재상태	M1	M2	M3	M4	P1	P2	P3	P4	P5	다음 상태
Null	@T(a)	-	-	-	-	-	-	-	-	Call initiated
	-	-	-	-	@T(b)	-	-	-	-	Cakk Oresebt
Call Initiated	-	-	-	-	-	@T(c)	-	-	-	Outgo. Call Proceed.
Outgoing Call Proceeding	-	-	-	-	-	-	@T(c)	-	-	Call Delivered
	-	-	-	-	-	-	-	@T(c)	-	Active
Call Delivered	-	-	-	-	-	-	-	@T(c)	-	Active
Call Present	-	@T	-	-	-	-	-	-	-	Incom. Call Proceed.
	-	-	@T(c)	-	-	-	-	-	-	Call Received
	-	-	-	@T(d)	-	-	-	-	-	Connection Request
Incoming Call Proceeding	-	-	@T(c)	-	-	-	-	-	-	Call Received
	-	-	-	@T(d)	-	-	-	-	-	Connection Request
Call Received	-	-	-	@T(d)	-	-	-	-	-	Connection Request
Conn. Request	-	-	-	-	-	-	-	-	@T	Active
Active	-	-	-	-	-	-	-	-	-	Release Request

a : index(CallRegister) > 0
b : state(callee)=Null
c : sig_path(caller)
d : CallRegister.CallReference=CONNECT. CallReference

M1 : SETUP
M2 : CALL PROCEEDING
M3 : ALERTING
M4 : CONNECT

P1 : Setup_req
P2 : Proceeding_req
P3 : Alerting_req
P4 : Setup_resp
P5 : Setup_complete_req

을 배제할 수 있다. 이는 하나의 상태 변화가 종결되지 않은 상황에서 새로운 전이의 연속적인 변화를 허용하지 않도록 구성할 수 있음을 보여준다. 또한 표 1에서는 각 이벤트에 대한 조건을 표시하여 명세화함으로서 구현을 위한 요구사항으로 사용이 가능하게 되어있다. 조건 a는 주어진 CallRegister의 index 값이 0보다 큰 것을 요구하는데 이는 이벤트 SETUP이 true가 되기 위해서는 그 이벤트를 처리할 정보를 담아둘 CallRegister가 존재해야 된다는 것을 의미한다.

2. 모드, 상태와 전이

조건이 주어진 상태에서 추가적으로 필요한 자원 또는 정보를 표시하는 것인데 비하여, 상태는 미리 정의된 일련의 규칙, 법칙, 정책들이 적용되는 동안의 시스템 상황을 표시한다[11]. 상태는 도메인에서의 공통적인 특성에 속하며 경우에 따라서는 시스템의 공통적인 속성으로서 공유가 되어야 한다. 상태는 도메인 또는 시스템 조건들의 현재 값으로 정의되는데 이에 따라 상태 공간은 조건들의 모든 가능한 조합으로 만들어진다. 상태 공간의 크기를 축소하기 위해서는 일련의 계열화가 가능한 상태를 모드(mode)로 모아서 상태 공간과 모드 공간을 계층적으로 구축할 수도 있다. 모드를 사용할 경우 시스템 동작(behavior)에 중요치 않은 것을 생략할 수도 있다. 모든 시스템 조건 값은 발생되는 이벤트가 그 조건 값에 의해 상태 전이에 영향을 미칠 때에만 의미를 갖게 되고 따라서 시스템의 검증에 중요하다.

전이란 시스템의 상태가 변경되는 것으로서 동일한 모드 내에 있는 상태 사이에서 발생할 경우는 상태전이, 모드 사이에서 발생할 경우는 모드전이라고 한다. 상태전이는 조건과 이벤트에 의해 정의되고 기술된다[11]. 하나의 모드로는 시스템의 한 면(aspect)만을 나타내고, 시스템의 전체적인 동작을 나타내기 위해서는 시스템의 모든 모드들을 동원하여야 한다. 전이는 그 전이를 유발시킨 이벤트의 검출과 동시에 발생한다. 만일 2개의 이벤트가 동시에 발생하면, 동일한 상태에서 2개의 전이가 연속적으로 가능해진다. 이와 같은 경우의 상태는 비결정적(non-deterministic)이며, 2개 전이중 어느 하나는 요구사항을 만족시키나 둘 다는 만족시키지 못한다. 이러한 경우에 시스템의 invariant로 주어지는 조건에 따라 비결정성의 이벤트 발생을 결정성으로 규격화가 가능하다.

3. Action과 시나리오

각 상태에서 발생하는 이벤트에 따라 시스템 동작이 이루어진다. 시스템 동작은 일련의 action들로 구성되는데 이러한 action sequence를 시나리오라고 하며 이 시나리오는 이벤트의 발생 즉시 수행되고, 다른 이벤트가 수행되기 전에 완료된다[10]. Action은 타이머를 set할 수 있으며, 타이머는 종료될 때 새로운 이벤트를 발생시킨다. 시나리오를 구성하는 action은 객체로 규격화되고 기술이 된다[10].

```
Procedure : pd_TrafficDescriptor(x BYTE3, y BYTE3)
            return(INT)
operation
        pd_ TrafficDescriptor(x, y) - if x > y then
                                        return(1)
                                    else y-x -> y, return(2)
```

그림 10. 트래픽 기술자를 처리하는 Action의 정의
Fig. 10. Action definition of traffic descriptor.

시스템 invariant 어트리뷰트들은 시스템 라이브러리를 만들어 동일한 도메인 내에서 객체간의 정보 교류뿐만 아니라 도메인 사이의 정보 교류를 가능하게 한다. 이는 데이타를 작성할 때 적용하는 캡슐화 개념에 위배된다고 볼 수도 있으나 두개의 비동기 객체가 공통의 어트리뷰트를 기록할 경우 발생하는 정보의 손실을 방지한다.

```
Newmode CellRate = struct(
                   /* mode 객체에 대한 정의 */
    ext                     BIT(1),
    ID                      BIT(7),
    CASE OF
    : Val                   BYTE3;
    : Val_arr  array(1:3) BYTE1
    ESAC);
```

그림 11. 셀 전송속도를 나타내는 어트리뷰트의 정의
Fig. 11. Attribute definition of cell transmission rate.

Action은 시스템 라이브러리 객체들로의 제한된 접근 통로를 통해 query를 만들 수 있다. 이것은 다른 객체의 정보를 사용하는 것을 가정하기 때문에 캡슐화의 원리에 위배된다. 그러나 분석 시에 다른 객체가 데이타를 읽을 수 있도록 나타내기 위해서는, 그 데이타를 억세스해야 하는 필요성을 선언해야 한다. 이것은 설계와 실현 단계에서, 이러한 필요성은 대상 객체로

필요한 데이타를 요청하는 메시지를 전송함으로써 충족되기도 하고, 또는 가장 빈번하게 참조하는 객체에 해당 데이타를 캡슐화시킴으로서 만족되기도 한다. 이러한 방법은 시스템 invariant를 실현 또는 설계 단계에서 선언해야 하는 입장에서는 매우 적용이 어렵다. 본 논문에서는 이러한 시스템 invariant들을 그로벌 객체로 정의하고 모든 객체가 참조할 수 있도록 구조화시켰다.

Ⅳ. 규격서의 모달성

f와 g가 기본적인 모달 공식이라고 할 때, *~f, f&g, f|g* : *~(not)*, *&(and)*, *|(or)*는 논리적인 연결자들로서 통상적인 의미를 갖는다. 정형화된 요구사항으로부터 구해진 그림 12의 공식들은 역으로 추적하면 현재의 상태를 얻을 수 있다. 이것은 공식이 true일 때 시스템 요구사항이 충족되는 것을 뜻하며 공식에서 주어진 safety assertion도 만족된다. 이렇게 주어진 공식에서 safety assertion의 true 값에 따라 전이의 발생 순서를 추적할 수 있고 그에 따른 상태의 변화를 검증할 수 있게 된다. 이를 위해 모달 논리를 이용하는데 □*f* 는 현재 상태에서 도달할 수 있는 모든 상태에서 공식*f* 가 true로 유지됨을 나타낸다. *f* 는 현재 상태에서 도달 가능한 상태중 한 상태에서 공식*f* 가 true로 유지될 수 있는 가능성을 나타낸다. *f* 의 는 nextstate를 나타내는 연산자이며 여기에서는 *f*가 다음의 시스템 상태에서 true가 됨을 나타낸다[12].

상태 기계는 시스템의 modal logic model로 활용 가능하고, 이 모델에서 충족되는 safety properties를 modal formula와 같은 방법으로 시험할 수 있다. 이 상태기계에 적용하는 조건들은 모달 공식의 기본적인 명제의 형태를 갖게 되며 이러한 기본적인 명제들로 이루어지는 모달 공식 즉, 만일 f와 g가 모달 공식이라면, *~f, f&g, f|g, f*, □*f*도 모달 공식이 된다.

그림 1, 그림 7, 그림 9, 표 1의 Active 상태에서 Modality를 검증하는 방법을 검토한다. Outgoing Call Delivered 상태에서 Setup_resp 이벤트로 인하여 Active 상태로 변경된 경우와 Call Delivered 상태에서 Setup_resp 이벤트로 인하여 Active 상태로 변경된 경우와 Connection Request 상태에서 Setup_complete_req 이벤트에 의하여 Active 상태로 변경된 경우에서 각각에 대한 조건을 모달 공식으로 표현한다. 이 모달 공식에 포함되는 이벤트와 조건은 시스템의 invariant가 되고 이들을 역으로 추적할 수 있다면 시스템의 검증이 가능해진다.

Call Present 상태에서 CONNECT 이벤트가 발생하면 Call Register에 있는 Call Reference 번호와 CONNECT 메시지에 의해 전달된 Call Reference 번호를 점검하여 동일한 경우에 관련 시나리오를 추출한다.

□(CallInitiated => M1 & index(CallRegister) > 0)
□(CallPresent => P1 & state(callee)=Null)
□(OutgoingCallProceeding => P2 & sig_path(caller))
□(CallDelivered => P3 & sig_path(caller))
□(Active => P4 & sig_path(caller) | P5)
□(IncomingCallProceeding => M2)
□(CallReceived => M3 & sig_path(caller))
□(ConnectionRequest => M4 &
CallRegister.CallReference=CONNECT.CallReference)

그림 12. 시스템 invariant로 나타나는 safety assertion의 공식화
Fig. 12. Formulation of safety assertions derived from system invariants.

Ⅴ. 호제어 시스템 설계

시스템 설계에 있어 기존의 객체지향적인 설계 방법론은 시스템에 필요한 객체를 미리 구축한 다음 시스템의 요구사항 또는 규격에 맞추어 객체의 메소드를 부여하고 메시지를 정하는 즉, 반복적인 모델링 작업을 수행함으로서 설계가 완료되는 방법이다. 그러나 대형 실시간 시스템에서의 이와 같은 설계 방법은 적절하지 않다. 왜냐하면 일부 도메인은 실현이 되고 또 일부 도메인은 규격화 단계에 있을 수 있으며 또 다른 도메인은 구체적인 설계 단계의 작업이 진행되는, 말하자면 다수의 개발자에 의해 동시적이고 병행적인 개발이 이루어지기 때문에 시스템 차원에서 객체 모델링을 반복적으로 수행하는 것은 기대하기 어렵다.

모든 객체들이 동일한 주소공간에서 수행이 되도록 컴퓨터 환경을 구축하는 것보다, 교환기 시스템은 병행적이고 동시적인 객체의 수행을 지원하는 것이 시스템의 개발 효율과 다양한 교환기능을 단순하게 분리할 수 있으므로 분산구조와 아울러 동시 및 병행 수행 환경을 구축하는 것이 필요하다. 이를 지원하기 위하여 동일한 주소 공간내에 있는 객체 집단을 블럭이라고 정리한다. 블럭과 블럭 사이의 인터페이스는 IPC 또는

라이브러리를 통한 Procedure Call로서 정보의 전달이 가능하도록 2장에서 제시한 바와 같이 기반구조를 정리하였다. 이러한 블럭은 객체 클라스로서 존재하는데 여러 블럭이 병존하게 되고 각 블럭은 동시처리 되는 객체를 생성한다.

교환기 호제어 시스템 소프트웨어의 기반구조를 정립하고 이에 맞추어 요구되는 기능을 설계하는 것이 특징인 본 논문에서 제시하는 설계의 순서는 다음과 같다.

동적 모델의 설계

(1) 요구사항에서 사용자의 기능에 따라 문제영역을 분류하고 각 문제영역에서 도메인을 추출한다. 일반적으로 하나의 도메인이 정의되면 그 도메인을 시스템 내에 구현할 때 하나의 주소공간에서 실현이 된다. 이를 블럭으로 정의하고 능동객체라고도 하는데 병렬처리, 동시처리의 기본 단위가 된다.

(2) 도메인에서 외부로 부터 입력되는 자극을 추출하여 이것을 이벤트로 정의한다. 대체적으로 이벤트는 외부로 부터 시스템에 주어지는 자극을 기준으로 정리되지만 외부로 부터 자극이 기대됨에도 불구하고 자극이 없는 경우, 예를 들면 타이머에 의해 시스템이 반응하는 경우에도 이것을 time-out 이벤트로 정의한다.

(3) 이벤트 사이의 관계를 정리하고 한 이벤트에서 다음 이벤트가 발생할 사이의 시간적인 간격을 상태로 정의하여 상태 다이아그램을 정리한다. 이 상태 다이아그램이 완성되면 이를 상태표로 형식화시킨다. 형식화된 상태표는 단계 (5)의 설계가 진행되는 동안 구체화된 정보를 추가한 다음 프로그램 로직으로 변환되어 state-based domain id. table에 실장된다.

(4) 한 상태에서 발생 가능한 각 이벤트에 대하여 이를 처리할 시스템의 행위를 시나리오로 기술한다.

(5) 시스템의 행위를 기술한 시나리오를 시스템이 수행할 프로그램 단위인 추상화된 Action Sequence로 변환한다. 여기에서 기사용중인 Action이 기술되면 그것을 재사용하기 위하여 해당 Action을 공동으로 사용 가능한 라이브러리에 위치시킨다. 재사용을 위하여 local library와 global library가 정의되어 동일 주소공간에서 재사용할 것인가 또는 서로 다른 주소공간에서 재사용할 것인가에 따라 전자와 후자의 라이브러리에 실제 Action의 위치를 결정한다.

(6) 시스템 기반구조에 맞추어 Action 상호간의 인터페이스를 형식화시킨다. 이것은 객체 사이의 인터페이스를 단순화시킴으로서 객체 사이의 이벤트를 줄여 시스템 차원의 복잡성을 축소하는 효과를 낸다. 인터페이스를 통한 정보의 전달은 Call-Register의 인덱스에 국한되며 이 인덱스로서 각 Action은 공통영역에 있는 CallRegister의 내용을 참조함으로서 관련된 호의 상태, 부가된 정보, 점유된 자원 정보, 요구된 트래픽 정보 등을 추출한다..

객체 모델의 설계

(7) 각 Action은 구체적인 객체로 정리된다. Action은 수동객체라고도 하는데 추상화된 인터페이스를 통하여 수행이 되며, 전달되는 매개변수에 의해 오퍼레이션이 정해진다. 재사용을 최대화시키는 기준을 Action에 두고 시스템의 기반구조가 설계되었다.

(8) 공용 객체의 경우 라이브러리에 배치가 되도록 형상관리 차원에서 조정한다. 공용 객체로 정리가 되면 공용으로 사용하게 될 데이타 사전을 정리하여 공지한다. 인터페이스는 파라메타의 추상화를 통하여 이루어지므로 단순한 구조를 갖게 되는데, 이에 따라 불필요한 관계성이 제거된다.

(9) 상기 과정을 반복하여 정제시켜 나간다.

VI. 결 론

ATM교환기의 호제어 서비스에 대한 요구사항을 문제영역으로 보고 이것을 도메인으로 분류하여 분석하고 이들 도메인에 대한 이벤트와 조건을 표 형태로 표시하는 정형화 방법을 제시하였으며 이에 따라 safety assertion으로 나타나는 조건과 이벤트를 시스템 invariant화하고 이들을 통하여 모달 논리를 구축함으로서 이들에 대한 관계성의 검증이 가능함을 제시하였다.

문제영역을 도메인으로 분류하고 그 도메인에 대한 분석 업무와 시스템 설계 업무를 수행하는 과정을 방법론으로 제시하였다. 표로 작성된 요구사항 명세서에 따라 이벤트가 정해지고, 각 이벤트에 대하여 현재의 상태에 따라 수행해야할 시나리오를 만들고 이를

Action으로 변환하여 객체화시켜 나가는 방법론의 일관성을 점검하였다. 이 방법론에 의하면 교환기 시스템 내에서의 객체는 미리 존재하는 것이 아니라 필요성에 따라 기존의 객체를 확장하거나 또는 새로 생성하여야 하는데, 필요성이 발생할 때마다 즉, 새로운 요구사항이 발생하거나 또는 기존의 기능이 변경되는 경우, ATM교환기의 호제어 소프트웨어의 기반구조를 바탕으로 이러한 요구사항을 수용할 수 있음을 보여준다. 본 방법론은 동적 모델을 구축하는 과정에서 자연스럽게 객체 모델링 과정으로 옮겨갈 수 있는 특징이 있다. 객체가 많아질 경우 객체 모델을 그림으로 표현하기 어려운데 본 방법론을 적용할 경우 추상화된 객체가 구체적인 객체로 형상화되므로 복잡한 요구사항을 설계하는 교환기와 같은 시스템의 분석 및 설계에 효과적인 것으로 판단된다.

참 고 문 헌

[1] A. Spencer Peterson, Coming to Terms with Software Reuse Terminology: a Model-Based Approach, *ACM SIGSOFT Software Engineering Notes* Vol 16 No 2, pp45-51, Apr. 1991.

[2] Guillermo Arango, Domain Analysis - From Art Form to Engineering Displine, *Proceeding of 5th International Workshop on Software Specifications and Design*, pp152 159, 1989.

[3] J. Rumbaugh et. al., *Object-Oriented Modeling and Design*, Prentice-Hall International, Inc., 1991.

[4] Draft Recommendation Q.2931, ITU-TS, 1995.

[5] Sally Shlaer and Stephen J.Mellor, An Object Oriented Approach to Domain Analysis, *ACM SIGSOFT Software Engineering Notes* Vol 14 No 5, PP66-77, Jul. 1989.

[6] C.A. Sunshine, D.H. Thomson, R.W. Erickson, S.L. Gerhart, D. Schwabe, Specification and Verification of Communication Protocols in AFFIRM Using State Transition Models, *IEEE Transactions Software Engineering*, SE-8(5) (Sep.), pp460-489.

[7] 정지훈, et. al., ATM VP 교환 시스템에서의 전용선 서비스 제어에 관한 연구, 대한전자공학회 하계학술대회 논문집 제17권 1호, 1994

[8] G. Arango, R. Prieto-Diaz, Domain Analysis Concepts and Research Directions, *Domain Analysis and Software Systems Modeling*, edited by R. Prieto-Diaz and A. Arango, IEEE Computer Society Press, pp9-26, 1991.

[9] Y.V. Srinivas, Algebraic Specification for Domains, *Domain Analysis and Software System Modeling*, IEEE Computer Society Press, PP90-124.

[10] 김한경, 차세대 교환 소프트웨어 전망, 대한전자공학회 텔레콤, Vol 10 No 1, 1994

[11] Joanne M. Atlee and John Gannon, State-Based Model Checking of Event-Driven System Requirements, *IEEE Trans. on SW Eng.*, Vol 19, No 1, pp24-40, Jan. 1993.

[12] R. Frost, *Introduction to Knowledge Base System*, Collins Professional and Technical Books, 1986.

[13] Guttag, J.V., Notes on Type Abstraction, *Proceedings of Conference on Specifications of Reliable Software*, pp36-46, 1979.

[14] B. Liskov and J. Guttag, *Abstraction and Specification in Program Development*, The MIT Press, 1986.

[15] B. Liskov and V. Berzins, An Appraisal of Program Specifications, *Research Directions in Software Technology*, edited by P. Wagner, MIT Press, pp276-301, 1979.

[16] J. Goguen, More Thoughts on Specification and Verification, *ACM SIGSOFT* Vol 6 No 3, pp36-46, 1981.

저 자 소 개

金 漢 慶(正會員)

1995년 충북대학교 전자계산학과 박사과정 수료

1983년 ~ 현재 한국전자통신연구소 책임연구원

具 然 卨(正會員)

1979년 ~ 현재 충북대학교 컴퓨터과학과 교수