형식적 방법론을 이용한 통신 소프트웨어 개발 지원 환경의 설계와 구현

이 은 석†

요 약

금후의 고도의 정보화 사회에서 발생할 다양한 형태의 시스템 요구와 기대에 능동적이고도, 효과적으로 대처해 나가기 위해서는 보다 체계적인 컴퓨터 통신 소프트웨어 개발 방법론의 정비와 확립이 절실히 요구되고 있다.

본 논문에서는 (1)종래의 형식적 기술 기법보다 사용자 친화성(user-friendliness)이 높은 표현 기법으로 통합형 에디터의 제공, (2)(1)에서의 새로운 표현 기법으로부터 국제 표준의 형식적 기술 기법인 LOTOS로의 자동변환법의 제공, (3) 형식사양 수준에서의 소프트웨어 재사용의 방법론을 제공한다. 그 결과, 사용자는 특정FDT(Formal Description Technique)에 대한 깊은 지식이나 충분한 사용 경험이 없어도 시스템에 대한 구조적 요구와 동작요구, 데이터요구만 가지고 최종적으로 원하는 형식사양을 생성해 낼 수 있다. 이는 특정 FDT를 직접적으로 이용하는 것 보다 상대적으로 초기 학습의 부담이 적고 기술과 이해가 용이하다.

Design and Implementation of a Support Environment for Communication Software Development based on Formal Methods

Eun-Seok Lee†

ABSTRACT

Systematic paradigms and methodologies for computer communication software development are essentially required to cope with diverse system requirements and expectations, to be appeared in the future advanced information society actively and effectively

This paper describes (1) an integrated editor which has relatively high user friendliness than other conventional structured editors for formal description such as G-LOTOS editors, (2) an automatic translation function to LOTOS from the expression made by the proposed editor, and (3) a software reuse methodology in formal specification level. As a result, a user can generate a formal specification based on LOTOS even if he/she does not have sufficient knowledges and experiences on a specific FDT. This can also reduce the effort required for the initial learning than using a specific FDT directly.

1. 서 론

컴퓨터 통신 시스템의 대규모화, 복잡화가 점점 가속화되어짐에 따라 그러한 시스템을 개발하는 데는 많은 시간과 비용, 그리고 전문인력을 필요로 하게 된다. 이는 인적, 물적 자원의 제약을 받는 현재의 많은 시스템 개발자 및 개발조직에게는 커다란 부담이 되고 있고 이러한 상황은 갈수록 심화될 전망이다. 따라서 이러한 시스템의 개발부담을 줄이고 앞으로의 고도의 정보화 사회에서 발생할 다양한 형태의 시스템 요구와 기대에

※ 본 논문은 1996년도 한국 학술진흥 재단의 공모과제 연구비에 의하여 연구되었음.

† 종신회원 : 성균관대학교 정보공학과 조교수

논문접수 : 1998년 1월 15일, 심사완료 : 1998년 4월 29일

능동적이고도, 효과적으로 대처해 나가기 위해서는 보다 체계적인 컴퓨터 통신 소프트웨어 개발 방법론의 정비와 확립이 절실히 요구되고 있다.

이러한 요구에 대응하기 위한 효과적인 방법으로서, 특히 다음의 두가지 접근 방법이 주목되고 있다.

1) 형식적 기술 기법(Formal Description Technique, FDT)을 이용한 형식 사양(Formal Specification)의 구축과, 구축된 형식 사양으로부터의 자동/반자동 프로그래밍 방법을 이용한 소프트웨어 개발법[5],[6]

2) 기존의 소프트웨어의 재사용에 의한 조립형 소프트웨어 개발법[8],[9]

1)의 경우, LOTOS[1], SDL[2], Estelle[3]와 같은 형식적 기술 기법은 통신 소프트웨어 개발 공정 과정의 상위 과정에서의 사양의 형식화를 위한 유효한 표현 수단을 제공한다. 또한 검증, 구현등 하위 과정에 있어서의 자동/반자동화 처리를 가능하게 하기도 한다. 이러한 형식적 기술 기법의 유용성과 폭 넓은 이용 가능성에도 불구하고 그것의 사용범위 및 실제의 사용자는 크게 제한되는 실정이다. 이는 형식적 기술 기법에 관련한 지금까지의 연구가 어떻게 하면 형식성을 더욱 향상시킬 수 있을 것인가에 초점을 맞추어 행해져 왔고, 기술용이성(Describability)과 이해성(Understandability)등 형식적 기술기법의 사용자 친화성을 향상시키기 위한 노력의 부족이 그 원인으로 판단된다.

한편, 2)의 경우 기존의 방대한 소프트웨어 자원의 재사용이란 측면에서 개발 공정 기간의 단축과, 소프트웨어 품질의 향상, 생산성 향상, 개발 비용의 절감 등 많은 효과를 기대 가능하게 한다. 그러나 실제에 있어서는, 각 개발 과정에서 장차 예상되는 재사용을 강하게 의식한 소프트웨어를 설계하는 방법론(Software Development for Reuse)과, 그러한 자원을 이용하여 목적하는 시스템을 만드는 방법론(Software Development with Reuse)의 어느 쪽도 확립되어 있지 않아 실질적인 적용은 어려운 상황이다.

따라서 본 연구의 목적은, 먼저 위의 두 접근방법의 각각의 문제점을 해소해 나가는 방안을 고안하고, 다음으로 두 방법을 하나의 시스템으로 통합 운영하는 지원 시스템을 구축하여 최종적으로 통신 소프트웨어의 생산성과 품질의 향상, 그리고 개발 비용 절감의 가능성을 제시하는데 있다.

구체적으로 본 논문에서는 상기의 문제점들에 대해서 각각 아래와 같은 접근 방법을 제안, 구현한다.

a) 종래의 형식적 기술 기법보다 친화성이 높은 표현 기법의 제공,

b) a)에서의 새로운 표현 기법으로부터 특정의 국제 표준의 형식적 기술 기법, 구체적으로 LOTOS로의 자동 변환법의 제공,

c) 형식 사양 수준에서의 소프트웨어 재사용의 방법론을 고안하여, 소프트웨어 재사용의 수준을 프로그램(코드) 수준이 아닌 사양 수준으로 고급화하여 개발에 참여하는 사람들의 지적 부하의 경감은 물론 사용 시스템(프로그래밍 언어, 운영체제)에 대한 종속도를 낮출 수 있어 재사용의 보다 큰 효율화를 추구한다.

본 논문의 구성은 다음과 같다. 2장에서는 형식적 기술기법의 개요와 본 논문에서 특히 다루게 되는 LOTOS에 대한 개요에 대해 설명한다. 3장에서는 제안시스템의 구성과 적용 알고리즘에 대해 기술한다. 4장에서는 시스템 구현과 정성적 평가 결과에 대해서 정리한다. 마지막으로, 5장에서는 결론 및 향후 연구 방향에 대해서 기술한다.

2. 형식적 기술 기법(FDT)과 LOTOS

형식적 기술 기법은 일반적으로 1)대상이 되는 시스템을 엄밀하고 정확하게 기술할 수 있을 것, 2)작성된 사양의 적절한 해석을 가능하게 하는 형식적 수학 모델을 제공할 것, 3)사양의 구조화, 추상화를 위한 수단을 제공할 것 등의 조건을 갖추어야 한다. 형식적 기술 기법은 크게 확장형 유한 상태 기계(Extended Finite State Machine)를 기초로 한 기법과 CCS (Calculus of Communicating Systems)[14], CSP(Communicating Sequential Processes)[15] 등 프로세스의 동작을 주된 기술 대상으로 하는 프로세스 대수 이론(Algebraic Theory)에 기초 한 것으로 나눌 수 있다. 전자에 해당하는 대표적인 것으로 SDL (Specification and Description Language)[2]과 Estelle[3]가 있다. 본 논문에서 대상으로 하는 LOTOS(Language of Temporal Ordering Specification)[1]는 후자에 속하며 1981년 부터 1988년 사이에 ISO에 의해 개발되어 OSI의 각종 서비스의 정의 및 프로토콜 사양을 형식적으로 기술하는 것을 목적으로 하고 있으나, 분산형 정보 처리 시스템 등 여

더 가지 복잡한 시스템으로의 적용도 크게 기대되어지고 있다. LOTOS는 "시스템은 게이트(gate)라 불리는 외부와의 접점을 통해 외부와 주고 받는 관측 가능한 사건간의 시간 순서를 기술하는 것으로 사양화 가능하다" 라는 기본 개념을 갖고 있으며 이는 시스템의 내부 상태의 상태 천이를 기술하는 SDL이나 Estelle와는 달리 시스템의 내부는 블랙 박스로 간주하기 때문에 보다 추상도가 높은 사양을 작성할 수 있다는 강점을 가지고 있다. LOTOS사양은 기술되는 시스템의 동작을 기술하는 동작부(Behavior Section)와 시스템과 외부 환경과 주고 받는 데이터를 정의하는 데이터부(Data Section)로 구성되며, 동작부는 상기의 프로세스 대수에, 데이터 부분은 추상 데이터형의 기술 언어인 ActOne[12]에 근거하는 등 수학적 모델에 강하게 기초를 두고 있기 때문에 엄밀한 기술은 물론 작성된 사양의 해석에 유효한 이론적 틀을 제공할 수 있다.

이하, 본 논문의 이해를 돕기 위해 LOTOS의 구문에 대하여 기본 LOTOS를 중심으로 간단히 요약한다.

(1) Stop (**stop**)

LOTOS에서 가장 기본이 되는 프로세스로 어떠한 액션도 발생하지 않는 비활성화 프로세스이다.

(2) Action Prefix (;)

액션의 실행 순서를 표현하는 것으로, 어떤 액션 a가 발생한 후에 어떤 동작식(Behavioral Expression) B를 수행한다는 것을, "a; B"로 표현한다.

(3) Choice ([])

선택적인 동작을 기술할 때 사용되며, 두 개의 동작식 B1, B2의 어느 한쪽이 동작하도록 기술할 경우, "B1 [] B2"로 표현한다.

(4) Interleaving (|||)

두 개의 동작식 B1, B2가 서로 완전히 독립하여 동작하도록 기술할 경우 사용되며, 시스템의 액션의 발생 순서는 두 개의 동작식의 액션이 혼재한 형태로 발생한다. 예를 들어, B1 = a; b; **stop**, B2 = c; d; **stop** 이라 하면, "B1 ||| B2"의 외부로부터 관측 가능한 동작의 한 예는 "a, c, b, d"가 된다.

(5) Synchronization (||)

두 개의 동작식이 완전히 동기를 취하면서 동작하도록 기술할 경우 사용된다. 여기서, 완전한 동기라 함은, 한쪽의 동작식에서 발생 가능한 액션이 다른 한쪽의 동작식에 있어서도 발생가능 할 때에 한해서 그 액션은 시스템 전체로서 발생 가능함을 의미한다. 예를 들어, B1 = a; (b; **stop** [] d; **stop**), B2 = a; (c; **stop** [] d; **stop**)이라 하면, "B1 || B2"의 외부로부터 관측 가능한 유일한 동작은 "a, d"가 된다.

(6) Selective Parallel (|[g]|)

두 개의 동작식이 액션리스트 g에서 주어진 액션에 대해서 동기를 취하면서 동작하도록 기술할 경우 사용된다. 예를 들어, B1 = a; b; c; **stop**, B2 = d; a; c; **stop** 라 하면, "B1 |[a]| B2"의 외부로부터 관측 가능한 동작의 한 예는 "d; a; (b; c; **stop** ||| c; **stop**)"가 된다.

(7) Hiding (**hide** *g* **in**)

동작식의 어떤 액션을 외부로부터 보이지 않게 기술할 때 사용된다. 예를 들어, B = a; b; c; **stop** 이라 하면, "**hide** b **in** B"의 외부로부터 관측 가능한 동작은 "a, *i*, c"가 된다. 여기서 *i* 는 외부로부터 관측 불가능한 내부액션을 나타낸다.

(8) Enable (>>)

두 개의 동작식이 연속적으로 행해지도록 기술할 때 사용된다. 예를 들어, B1 = a; b; **exit** , B2 = c; d; **stop** 이라 하면, "B1 >> B2"의 외부로부터 관측 가능한 동작의 한 예는 "a; b; *i*; c; d"가 된다. 여기서 **exit**는 정상적으로 수료한 것을 나타내는 프로세스로 **exit** $-\delta\rightarrow$ **stop** 가 된다. δ는 정상 수료한 것을 나타내는 특별한 액션으로 외부로부터는 관측 불가능하다.

(9) Disable ([>)

두 개의 동작식에서 하나의 동작의 도중에 끼어들기를 하여 다른 하나의 동작을 수행하도록 기술할 때 사용된다. 예를 들어, B1 = a; b; **stop** , B2 = c; d; **stop** 이라 하면, "B1 [> B2"의 외부로부터 관측 가능한 동작의 한 예는 "a; c; d"가 된다.

3. 제안 시스템

3.1 시스템 설계를 위한 기본 개념 및 사상

1.에서 언급한 바와 같은 이유로, 본 시스템을 설계하는데 있어서 특히 다음의 2가지에 중점을 두었다.

(1) 종래의 형식적 기술 기법(FDT)보다 친화성이 높은 표현 기법의 제공

(2) 형식사양 수준에서의 소프트웨어 재사용의 방법론의 고안

(1)에 관해서는, 종래의 FDT를 기반으로 한 통신

소프트웨어 개발 환경은 단순히 그 FDT를 위한 텍스트 에디터의 제공이나 그래픽 표현을 가질 경우 그것에 대응하기 위한 그래픽 에디터의 제공이 주 였었다. 그러나 그 어느쪽도 사용하는 FDT의 문법이나 특유의 기술 스타일을 충분히 이해하고 경험하여야 만이 사용 가능한 것으로서 사용자의 초기학습 부담을 줄이는 데는 한계가 있었다.

따라서 본 논문에서는, 특정 FDT에 의존하지 않아 범용적이며 상대적으로 기술 용이성과 이해성이 뛰어난 기술 기법으로 계층적 구조 표현 기법과 LTS (Labelled Transition Diagram)에 기반한 동작 표현 기법을 제공한다. 그리고 사용자가 최종적인 목표로 하는 특정 FDT(본 논문에서는 LOTOS)기반의 사양을 생성하기 위해서 상기의 표현 기법에 기반한 사양으로부터의 자동변환 기능을 제공한다.

(2)에 관해서는, 종래의 소프트웨어 재사용이 주로 코드 수준에서 이루어 졌던 것에 비교하면 시스템 의존도를 낮춤으로서 재사용의 효율화를 기할 수 있고, 보다 상류 과정에서의 개발의 효율화와 형식성을 통한 이후 과정(구현, 검증)의 자동/반자동화로 전체 소프트웨어 개발 비용의 절감을 기대할 수 있다. 이러한 장점을 갖는 (형식)사양 수준에서의 소프트웨어 재사용을 위한 몇몇 연구 예가 있다[8],[13],[21],[22]. 이들 시스템에서 재사용을 위한 부품으로서의 관련 사양을 검색하는 공통적인 방법은, 목표로 하는 시스템에 있어서의 내부 구조나 시스템이 갖는 기능등에 관한 키워드의 조회에 의한 것이었다. 즉, 재사용 부품마다에 구별할 수 있는 이름을 부쳐 놓고 찾는 사람은 그 이름을 가지고 검색해 나가는 것이다. 그러나 이러한 방법이 갖는 가장 큰 문제는, 설계자 또는 분야마다의 어휘의 정의가 서로 다른 점이다. 즉, 온토로지(Ontology)의 문제가 큰 걸림돌이 된다. 일반적으로 키워드라고 하는 것은 각 개인의 주관에 의해 정의되어지는 것으로 어떤 하나의 대상을 가리키는 경우에 있어서 복수의 사람이 항상 같은 키워드를 사용한다고는 기대하기 어렵다. 이는 분야의 차이에 있어서도 마찬가지로 나타난다. 그러므로 특히, 여러 분야에 걸쳐 많은 사람이 이용하는 재사용 시스템의 경우를 상정할 때 키워드 기반의 검색은 시스템 이용 효율에 한계가 있다.

따라서 본 논문에서는, 사양 수준에서의 소프트웨어 재사용을 위한 방법으로 사양간의 유사성(Similarity)의 계산에 근거한 사양 검색 및 재사용법을 도입한다. 여기서, 사양간의 유사성을 계산하는 알고리즘은 저자들에 의해서 제안되어져 이미 그 유효성을 인정 받은 상태이며[7],[11],[12] 본 논문에서는 그 알고리즘을 사용하여 관련 사양을 검색하고 재사용하는 기능을 추가하여 본 제안 시스템에 포함시키고 있다.

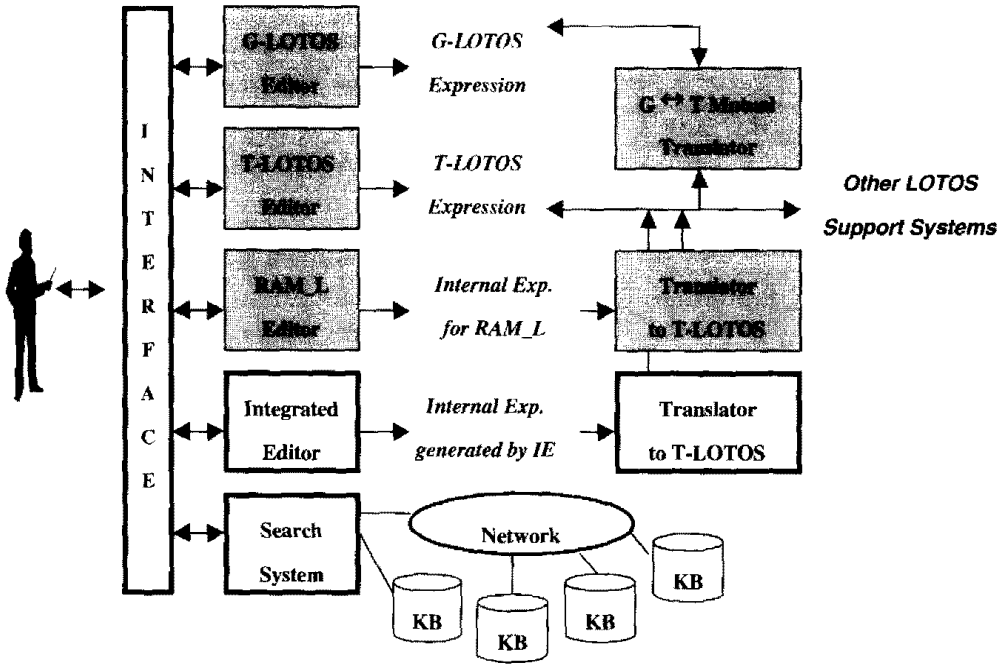
(1)과 (2)를 통해, 1.에서 지적했던 장차 예상되는 재사용을 강하게 의식한 소프트웨어를 설계하는 방법론과, 그러한 자원을 이용하여 목적하는 시스템을 만드는 방법론을 양립 시킬 수 있는 하나의 구체적인 제안으로서의 역할을 기대할 수 있다.

3.2 시스템 구성

본 시스템은 통신 소프트웨어 개발을 위한 통합형 시스템으로서 (그림1)에서 보이는 바와 같이, (1)G-LOTOS Editor, (2)T-LOTOS Editor, (3)RAM_L Editor, (4)Integrated Editor, (5)Search System, (6)GT Mutual Translator, (7)Translator to T-LOTOS from RAM_L, (8) Translator to T-LOTOS from IE, (9)Knowledge Base등으로 구성된다.

(1)~(4)는 본 시스템에서 제공하는 에디터들로서 크게 LOTOS의존형 에디터와 LOTOS에 의존하지 않는 에디터로 나눌 수 있다. 전자에 해당하는 것으로 (1)G-LOTOS(Graphical expression for LOTOS) 에디터[5], (2)T-LOTOS(Textual expression for LOTOS) 에디터[5], (3)RAM_L(Requirement Acquisition Mechanism for LOTOS) 에디터[10]가 있고, 후자에 해당하는 것이 본 논문에서 주로 다루게 되는, (4)통합형(Integrated) 에디터이다. (1), (2), (3)은 다양한 수준의 LOTOS사용자를 대상으로 자신의 지식이나 경험 수준에 맞는 에디터를 선택하여 사용할 수 있도록 지원하는 목적으로 저자들에 의해서 이미 만들어 졌던 것으로 현재 LOTOS사양 작성용으로 이용되고 있다.

한편 (4)의 경우, 세부적으로 다시 (4-1)구조적 기술을 위한 에디터(Editor for Structural Description)와 (4-2)동작 기술용 에디터(Editor for Behavioral Description), (4-3)데이터 타입 기술 에디터(Editor for Data Type Description)로 구성되어지며, (1),(2),(3)에 비교해 상대적으로 특정 FDT에 대한 의존도가 낮은 기술이 가능하며, 각각을 통해 만들어진 사양을 통합하여 특정 FDT로 자동 변환하는 자



(그림 1) 시스템 구조
(Fig. 1) System Architecture

동 변환기(Translator)에 대한 입력데이터를 제공한다. (4)의 상세한 내용에 관해서는 3.2.1에서 기술한다.

(5)의 검색 시스템(Search System)은, 본 시스템에 연결되어있는 지식베이스로부터 기존의 관련된 사양을 검색하게 하여줌으로써 사양레벨에서의 재사용을 지원하여 준다.

(6)~(8)은 서로 다른 표현간의 변환 기능들로서, (6)은 T-LOTOS표현과 G-LOTOS표현간의 상호 변환을 지원하며[5], (7)은 (3)의 RAM_L에 의한 기술내용을 T-LOTOS표현으로 변환하는 기능을 제공한다[10]. 본 논문에서 주로 다루는 (8)은 (4)의 통합형 에디터에 의해 작성되고 통합된 파일을 T-LOTOS로 변환하는 기능을 제공하고 있다. 이러한 변환 기능들을 제공함으로써 사용자가 T-LOTOS에디터 외의 어떤 에디터를 사용하건 기존의 LOTOS 관련 지원 시스템, 예를 들어, 검증(Verification)시스템[17], 적합성 시험(Conformance Test)시스템[18], C로의 자동 코딩(Automatic Implementation to C)시스템[19] 등 과의 접속과 이용을 가능하게 한다. (9)에는 재사용 가능한 사양들이 지식이라는 형태로 축적되어 있다.

본 논문에서는, 본 시스템에 새로이 추가된 (4), (5), (8), (9)에 관해서 주로 기술한다.

3.2.1 통합형 에디터(Integrated Editor)

(가) 구조적 기술을 위한 에디터(Editor for Structural Description: ESD)

ESD는 대상으로 하는 시스템을 구조적 관점에서 표현하는 것을 지원하는 것을 목적으로 하고 있으며, 그 시스템을 구성하는 프로세스간의 수직적 관계(vertical relationship)와 수평적 관계(horizontal relationship)를 표현할 수 있게 한다. 여기서 수직적 관계란, 프로세스간의 부모-자식관계를 계층적으로 표현하는 것으로 일반적인 프로세스 구조를 표현하는데 효과적이다. 반면에 수평적 관계란, 하나의 프로세스를 부모로 두는 여러 서브프로세스간의 상호관계를 가리키며 특히 서브프로세스간의 동기(synchronization)나 동작 순서 등을 표현 하는데 효과적이다. 따라서 사용자는 필요에 따라 상호보완적으로 선택하여 사용할 수 있다. (그림2)에서 후면의 좌측이 수직적 관계를 기술하기위한 공간이고, 우측이 수평적 관계를 기술하기 위한 것이다. 본 ESD를 이용하여 프로세스간의 관계를 시각적으로 표현할 뿐만이 아니라, 각 프로세스의

속성 예를 들어, Name, Access Point, Data type, Sub-process Name, Relationship between sub-processes등을 템플리트를 이용하여 기술할 수 있다.

ESD를 이용함으로써, 사용자는 대상 시스템에 대한 단계적 해석 및 분해가 가능하여 목적하는 사양의 골격을 용이하게 작성 가능하다.

(나) 동작 기술용 에디터(Editor for Behavioral Description: EBD)

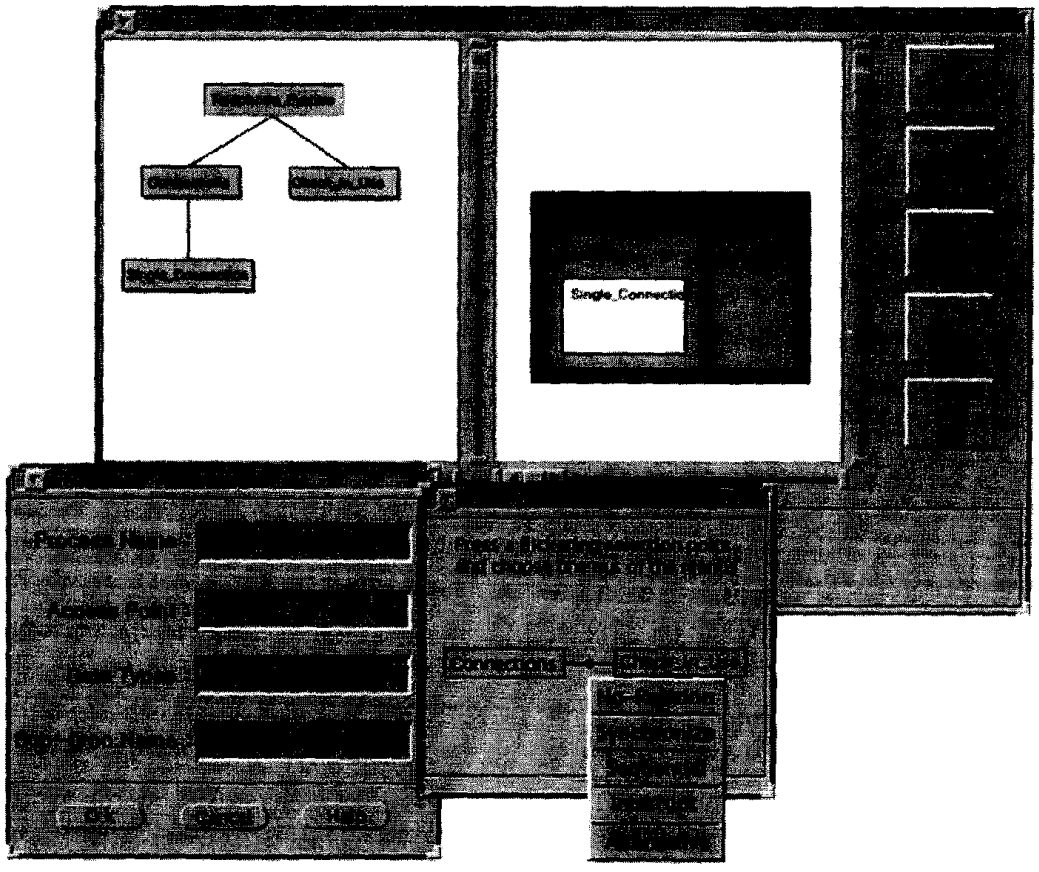
EBD는 대상으로 하는 시스템을 동작 관점에서 기술하는 것을 지원하는 것을 목적으로 하고 있으며, (가)에서의 각 프로세스에서 발생하는 행위(action)를 발생순서 관계(temporal ordering relation)를 이용하여 기술할 수 있게 한다. 이를 위해, EBD에서는 사용자에게 친화성이 높은 LTS(Labeled Transition System)[7]를 기술 도구로 제공한다. LTS는, 상태천이그래프의 일종으로, 각 노드(node)는 상태를 가리키고, 노드간을 연결하는 선에 첨기 되어있는 레이블(label)은 상태변화를 일으키는 행위를 가리킨다.

사용자는 (그림3)에서 보이는 바와 같이 GUI를 직접 조작하는 감각으로 프로세스의 동작을 기술할 수 있다. 이는 종래의 SDL/GR의 에디터[20]나 G-LOTOS 에디터[5]와 같이 특정 FDT의 문법이나 기술 형태에 의존하는 것과는 달리 이해하기 쉽고 기술하기 쉬운 등 사용자 친화성이 높다는 특징을 가지고 있다.

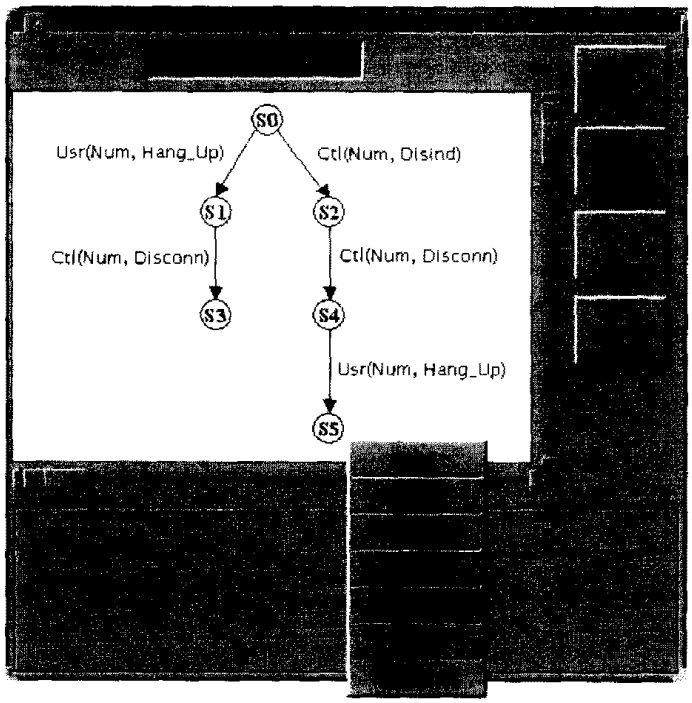
(다) 데이터 타입 기술 에디터(Editor for Data Type Description: EDD)

EDD는 추상 데이터형(abstract data type) 기술언어인 ActOne[16]에 근거한 데이터 타입 기술을 지원하는 것으로 크게 두 종류의 템플리트를 제공한다.

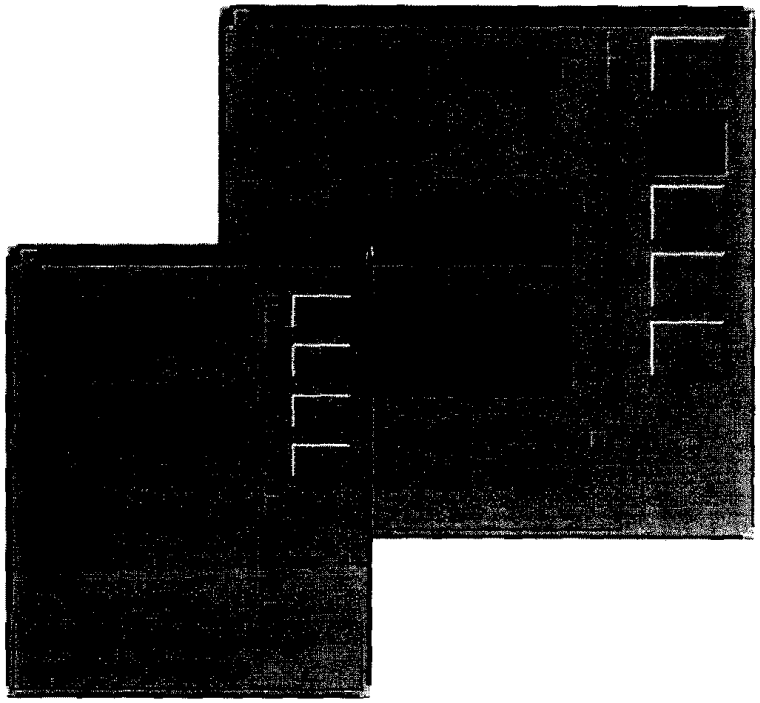
하나는, 일반적인 데이터 타입 정의 과정을 지원하는 것으로, (그림4)의 후면에 보이는 것과 같이 정의하고자 하는 데이터의 이름(type), 참조(imported) 데이터 타입 등을 순서적으로 기술한다.



(그림 2) 구조적 기술용 에디터의 예
(Fig. 2) An Example of the Editor for Structural Description



(그림 3) 동작 기술용 에디터의 예
(Fig. 3) An Example of the Editor for Behavioral Description



(그림 4) 데이터 타입 기술용 에디터의 예
(Fig. 4) An Example of the Editor for Data Type Description

다른 하나는 (그림4)의 전면에 보이는 것과 같이 이미 표준으로 정의되어져 있는 데이터의 라이브러리(library)와 사용자가 이전에 정의해 놓은 데이터를 사례(case)로서 제공하여 사용자는 이를 이용하여 재명명(renaming), 구현화(actualizing) 등 을 통하여 자신의 정의 과정에서 참고로 사용할 수 있게 한다.

사용자는 (가)~(다)를 이용함으로써, 대상으로 하는 시스템의 구조와 동작의 기술은 물론 그 시스템에서 사용되어지는 데이터의 정의까지 통합적으로 수행해낼 수 있다.

3.2.2 검색 시스템(Search System) 및 지식베이스 (Knowledge Base)

사양의 재사용도를 높이기 위해서는 필요한 사양을 지식베이스에 축적하고, 필요할 때 그 지식베이스로의 접근을 용이하게 지원해줄 기능이 요구된다.

본 시스템의 지식베이스에 축적되어지는 요소와 각각의 검색 방법은 다음과 같다.

(가) FDT로 기술된 형식사양 ($\{S_i\}$, $i \geq 0$)

직접적으로 재사용되어지는 대상으로서 3.2에서 언급한 (1)G-LOTOS Editor, (2)T-LOTOS Editor, (3)RAM_L Editor, (4)Integrated Editor등에 의해서 작성된 후 T-LOTOS표현으로 변환되어진 형식사양들이 저장되어져 있다. 이는 새로이 사양이 작성될 때마다 추가로 저장되어지며, 저장하는 과정에서는 그 사양의 내용을 설명할 수 있는 주요어를 사양의 이름으로 하여 저장하며, 이는 종래의 검색어에 의한 검색 방법론을 지원하기 위한 것이다.

(나) (가)의 형식사양에 대응하는 LTS($\{L_i\}$, $i \geq 0$)

LTS는 (가)에 저장되어 있는 형식사양의 의미모델로서도 사용되어지고 있으며, 따라서 (가)의 각 형식사양에 대응하는 LTS가 저장되어있다. 관련하는 LTS를 검색하는 방법으로는, 검색어에 의한 방법과 LTS간의 유사성(Similarity) 계산에 의한 방법[7]이 사용되고 있다.

(다) 데이터 타입 라이브러리(Data Type Library)

(그림4)에서 보이는 바와 같이, 새로이 데이터 타입을 정의하는데 재사용될 수 있는 데이터 타입을, 특히 표준으로 정의되어져 있는 데이터 타입을 라이브러리 형식으로 저장하여 제공하고, 그 외 표준의 데이터 정의(라이브러리)를 이용하여 만들어진 데이터 타입을 케이스(Case)로 저장하여 제공하게 한다.

3.2.3 변환 시스템(Translator to T-LOTOS)

본 변환 시스템은 3.2.1에서의 통합형 에디터에 의해 작성된 사양을 T-LOTOS로 변환시키는 기능을 갖는다. 변환 과정은 내부적으로, 1)통합형 에디터에 의한 3종류의 사용자 요구(구조, 동작, 데이터 타입)를 하나로 통합하는 단계와, 2)통합한 사용자 요구를 T-LOTOS로 변환시키는 단계를 거치게 된다. 이러한 변환 과정에서의 사용자 요구의 내부 표현 형식은 LOTOS 사양의 구조를 반영하고 있다.

(가) LOTOS 사양의 구조

LOTOS 사양의 구성 요소를 크게 나누면, 기술하고자 하는 시스템의 동작을 기술하는 a)동작부(Behavioral expression part)와 그 시스템을 구성하는 프로세스간에 주고 받는 데이터의 구조와 값을 기술하는 b)데이터부(Data definition part)로 나눌 수 있다. (그림5)는 전형적인 LOTOS 사양의 구조를 보이고 있다.

동작부는 다시 a-1)기술대상의 프로세스를 구성하는 서브프로세스(Sub-process)간의 관계를 기술하는 부분과, a-2)각 서브프로세스의 각각에 대한 상세 기술을 하는 부분으로 나눌 수 있다. (그림5)를 예로 설명하면, 1행(#1)에서의 프로세스 P는 기술 대상의 시스템이고, 게이트(gate) a, b에서 외부와의 데이터 전달이 이루어진다. 2행에서 4행까지가 상기의 데이터부에 해당된다. 5행 이하가 동작부에 해당되는데 특히, 6행에서는 프로세스 P가 서브프로세스 P1, P2, P3로 구성되며 그들 간에는 (P1 ||| P2) |[c]| P3의 관계(*프로세스 P1과 P2는 서로 완전 비동기 병렬 관계이고, 게이트 c를 통해서 각각 프로세스 P3와 동기한다*)가 있음을 나타내고 있다. 각 서브프로세스 P1, P2, P3의 상세 기술은 8~10행, 11~13행, 14~16행에서 이루어지는데 상위 프로세스 P와 같은 방법으로 상세화 되어진다. 그러나 이때, 어떤 프로세스가 더 이상 서브프로세스로 분할되지 않을 때 그 프로세스는 자신과 환경(자기 외의 모든 외부 프로세스)사이에 주고받는 사건(event)을 시간 순서적으로 기술하는 형태로 자신의 동작을 표현하게 된다.

이와 같이 LOTOS사양에서의 동작부의 기술은 프로세스간의 계층구조를 이용한 구조화 사양(Structured Specification)으로 작성되어 진다.

```
〈#1〉  process P[a,b]: noexit:=
〈#2〉  type
〈#3〉     /*data type definition of the process
          P*/
〈#4〉  endtype
〈#5〉  behavior
〈#6〉     (P1 ||| P2) |[c]| P3
〈#7〉  where
〈#8〉     process P1[a,c]: noexit:=
〈#9〉        /* behavioral description of
             process P1*/
〈#10〉    endproc
〈#11〉    process P2[b,c]: noexit:=
〈#12〉       /* behavioral description of
             process P2*/
〈#13〉    endproc
〈#14〉    process P3[c]: noexit:=
〈#15〉       /* behavioral description of
             process P3*/
〈#16〉    endproc
〈#17〉 endproc
```

(그림 5) LOTOS 사양의 전형적인 구성
(Fig. 5) A Typical Structure of LOTOS Specification

(나) 사용자 요구의 통합

통합형 에디터에 의한 사용자 요구는 ESD에 의한 구조적 요구(sr)와 EBD에 의한 동작요구(br), EDD에 의한 데이터 타입요구(dr)의 3종류가 있으며 각각 아래와 같이 정의되어진다.

[Structural Requirement, *sr*]

- *sr* = ⟨*pid*, *AP*, *DI*, *bi*, *sp*⟩, *structural requirement of a process*
  - *pid* : a process identifier,
  - *AP* : a set of access points in the process,
  - *DI* : a set of data identifier defined in the process,
  - *bi* : behavior identifier defined in the process,
  - *sp* : information about sub-processes of the process,

- *sp* = ⟨*PI*, *PR*⟩
  - *PI* : a set of sub-process identifier,
  - *PR* : a set of relationship among sub-processes

[Behavioral Requirement, *br*]

- *br* = ⟨*bid*, *bd*⟩, *behavioral requirement of the process*
  - *bid* : behavior identifier,
  - *bd* : behavior description

[Data type Requirement, *dr*]

- *dr* = ⟨*did*, *dd*⟩, *data type requirement of the process*
  - *did* : data identifier,
  - *dd* : data description

상기와 같은 정의를 바탕으로, 각각의 에디터에 의해 3종류의 요구가 작성되고 저장된다. 그러나 이들 요구간에는 밀접한 관련이 있어, 결국 기술하고자 하는 시스템(프로세스)을 구성하는 서브프로세스들의 계층구조와 그 계층구조에서 표현되는 각 프로세스의 동작, 프로세스간의 주고받는 데이터를 정의하는 것이므로 최종적으로 하나로 골격으로 통합할 필요가 있다. 이러한 통합작업을 용이하게 하기 위하여, 본 시스템에서는 ESD에서 특정 프로세스를 선택하여 EBD나 EDD를 가동하게 함으로서, 프로세스-동작-데이터 요구기술의 연결을 용이하게 하였다.

구체적인 통합 알고리즘은 아래와 같다.

【Algorithm for Requirements Integration】

- Input : a set of structural requirements, $SR = \{ sr_i \}$,
  a set of behavioral requirements, $BR = \{ br_j \}$,
  a set of data requirements, $DR = \{ dr_k \}$
- Output : Integrated Requirement, IR

〈step 1〉 Start from the initial structural

requirement sr_1. For sr_1,
$process_{01} = pid_1$

〈step 2〉 For sr_i,
$access\text{-}point_i = AP_i$,
$functionality_i = fc_i$

〈step 3〉 For all $di_n \in DI_i$, import data description dd_k of dr_k such that
$di_n = did_k$,
$datai_n = dd_k$

〈step 4〉 Import behavior description bd_j of br_j such that .
$bi_i = bid_j$
$behavior_i = bd_j$

〈step 5〉 If a set of relations among sub-processes $PR_i \neq 0$,
$relation_i = PR_i$,
and for all $pi_m \in PI_i$, import structural requirement sr_h such that
$pi_m = pid_h$,
$process_{ih} = pid_h$

and repeats from 〈step 2〉

(다) T-LOTOS로의 변환

(나)에서 통합되어진 사용자 요구를 T-LOTOS로 변환하기 위한 단계이다.

LOTOS로 변환하기위해서는, LOTOS기반의 사양에서 자주 사용되는 표현을 반영할 수 있게 사용자 요구 정의의 확장이 필요하다. 구체적으로 LOTOS기반의 사양에서는 서브프로세스 간의 관계 기술과 동작 기술을 명확하게 구별하기 어려울 때가 있다. 예를 들어, a ; b ; *proc1*[c, d] ||| e ; f ; *proc2*[g, h] 와 같이 동작기술 내에 서브프로세스간의 관계가 정의되어지는 경우다. 이러한 LOTOS특유의 기술 상황에 대처하기 위해서 아래와 같이 (나)에서의 동작요구, br 을 LOTOS의존형 brl(Behavioral Requirement for LOTOS)으로 확장, 추가할 필요가 생긴다.

〔Extended Behavioral Requirement for LOTOS, *brl*〕

- *brl* = { *info*$_j$ }, *behavioral requirement of the process*

  info = 〈*bd*, s, *pr*〉
  - *bd* : behavior description,
  - *s* : state
  - *pr* : process or processes with a relationship

사용자의 통합요구 IR과 brl을 사용하여 실제로 LOTOS기반 사양을 생성하는 알고리즘은 아래와 같다. 알고리즘 내의 고딕으로 표현된 부분(**specification**, **hide**, **in**, **behavior**, **where**등)은 LOTOS 사양에서 사용되는 키워드에 해당된다.

【Algorithm for Translation from IR to LOTOS based Specification】
- Input : an integrated requirement, *IR*, a set of behavioral requirements for LOTOS, *BR* _L= { *brl*$_i$ }.
- Output : formal specification-based on LOTOS

〈step 1〉 Generate
specification *process*$_{01}$ [*access-point1*] : **noexit**

〈step 2〉 For all *n*, add data$_{1n}$.

〈step 3〉 If sub-processes have a set of gates *G* which *process*$_{01}$ does not have, then add
hide *G* **in**

〈step 4〉 Add
behavior

〈step 5〉 If there exists *behavior*$_1$, then translate an LTS to LOTOS,
Translate(*behavior*$_1$),
else
TranswithRel(*brl*$_1$, *relation*$_1$)
If there exists *process*$_{1k}$, then add
Where
and for all *process*$_{1k}$, let $j = 1$ and go 〈step6〉

〈step 6〉 1)For all *m*, add data$_{km}$.
2)Add
process *process*$_{jk}$

{ *access-pointk*} :

3)If *processjk* is a post-executed process for sequential relationship, then add value parameters.

4)If *processjk* is a pre-executed process for sequential relationship, then add **exit**(sort of values):= else add **noexit** :=.

5)If sub-processes have a set of gates *G* which *processjk* does not have, then add **hide** *G* **in**

6)If there exists *behaviork*, then translate an LTS to a behavior description in LOTOS, Translate(*behaviork*), else TranswithRel(*brlk*, *relationk*)

〈step 7〉 If there exists *processki*, then add **Where** and for all *processki*, let $j = k$ and $k = i$, and go 〈step 6〉. Then add **endproc**

〈step 8〉 Finally, add **endspec**

Translate(*behavior*)

〈step 1〉 If *behavior* has loops whose start is not the initial state, then give the name of process to the start state.

〈step 2〉 Start from the initial state as the following way.

1)If there is a transition from the state, then add the action and ;. Repeat 〈step 2〉 to the next state.

2)If there are at least two transitions from the state, then combine the actions with [ ] and add ; after each action. Repeat 〈step 2〉 to each next state.

3)If there is no transition from the state, then if this process is a pre-executed process for sequential relationship, then add **exit**(values) else add **stop**

4)If the next state is the initial state or the start state of loop, then add the name of the appropriate process and gate.

TranswithRel(*brl*, *relation*)

〈step 1〉 For all *infoi* = $\langle bd_i, s_i, pr_i \rangle \in brl$, translate bd_i to a behavior description in LOTOS as well as Translate(bd_i), except for adding pr_i to the state s_i.

〈step 2〉 Combine the behavior descriptions translated in 〈step 1〉, using *relation*.

4. 시스템 구현 및 평가

본 시스템의 구성 요소 중 본 논문에서 주로 다루었던 (1)통합형 에디터, (2)검색 시스템, (3)변환 시스템 등은 Unix환경상에서 C를 이용하여 구현하였다. 이는 기존의 시스템과의 연계와 통합을 위한 것이다. 통합형 에디터의 화면 이미지는 이미 (그림2)~(그림4)에서 부분적으로 설명한 바 있지만, 이하 상세 설명한다.

(그림2)는 구조적 에디터(ESD)의 실제 사용 예를 보이고 있다. 기술과정은 다음과 같다. 빈 화면상에서 클릭하면 프로세스 속성 기술용 템플리트(Process Attributes)가 표시되고, 여기서 프로세스 이름(Process Name), 외부와 데이터를 주고 받는 상호 접속점(Access Point), 데이터 타입(Data Types), 서브프로세스 이름(Sub-proc. Name) 등을 입력하게 된다. 서브프로세스 이름을 입력하고 Ok 버튼을 누르면,

서브프로세스간의 관계를 정의하는 화면이 표시되고 pop-up 메뉴를 통해 서브프로세스간의 관계를 정의하게 된다. 이때 서브프로세스의 이름을 입력하지 않고 Ok 버튼을 선택하면 (그림3)의 동작 기술용 에디터(EBD)가 자동 기동 된다. 이는 동작 기술은 일반적으로 i)서브프로세스간의 관계의 정의 또는 ii)LTS (Labeled Transition Diagram)에 의한 액션의 시간 순서적 기술, 또는 i)과 ii)의 조합에 의해 이루어 지므로 위에서 서브프로세스의 이름을 입력하지 않고 Ok 버튼을 선택했다고 하는 것은 ii)를 하고자 한 것으로 판단하여 행해지는 지원 전략이다.

한편, ESD의 우측 아이콘들의 기능은 다음과 같다. Reuse 아이콘은 지식베이스 속에 저장되어 있는 재사용 가능한 사양 중 현재까지의 기술 내용과 가장 근접한 사양을 검색하여 출력한다. 이때 검색 방법은 사양의 이름에 의한 검색과 사양간의 유사도 계산에 의한 방법이 제공되며 어느쪽인가 선택할 수 있게 한다. Translate는 지금까지의 기술내용을 3.2.3에서의 변환 알고리즘을 이용하여 LOTOS로 변환하는 기능을 제공한다. File은 3.2.3의 요구통합 알고리즘의 입력 데이터로서 구조적 요구에 관한 데이터를 파일로 생성한다. Help는 ESD의 튜토리얼을 제공한다. Exit는 지금까지의 기술내용을 저장하고 작업을 종료할 것인지 새로운 ESD를 기동할 것인지 선택할 수 있게 한다. ESD의 화면상의 특정 프로세스를 선택하면, Process Attributes, EBD, EDD중 어느것을 기동할 것인지 선택할 수 있게 한다.

(그림3)은 (그림2)의 ESD에서 User_Disconnect라는 프로세스를 선택하고 동작기술 에디터(EBD)를 기동한 한 예이다. 여기서는 LTS표현에 의한 프로세스의 동작기술을 지원한다.

(그림4)는 (그림2)의 ESD에서 최상위의 Telephone_System이라는 프로세스를 선택하고 데이터기술 에디터(EDD)를 기동한 한 예이다. 후면의 그림이 새로이 정의하는 것을 지원하기 위한 템플리트이고, 정의과정에서 기존의 정의된 데이터 기술을 참조 또는 재사용하기 위하여 Reuse 아이콘을 선택하면, Library와 Case의 형태로 제공된다. 전자는 ISO에서 표준으로 제공하는 데이터를, 후자는 사용자가 새로이 정의, 추가, 변경하여 등록시킨 데이터 기술의 예들이다.

(그림6)은 Telephone_System[23]을 본 시스템의 통합 에디터를 통해 기술하고 이를 LOTOS로 자동 변환 시킨 실제 예이다. 결국 내부적으로는 ESD를 통해 EBD와 EDD가 통합되어져 있고, ESD에서 Translate 명령을 통해 최종적인 LOTOS사양을 생성하게 된다. 이렇게 생성된 LOTOS사양이 사용자가 통합에디터를 이용하여 작성한 것과 일치하는 지의 여부는 LOTOS로 작성된 사양의 의미는 LTS로 나타낼 수 있고[7] 이것이 사용자가 동작기술용 에디터로 작성한 내용과 일치하는 지를 통해 확인할 수 있다. 이 LOTOS사양은 3.2에서 언급하였던 다른 LOTOS기반 시스템, 예를 들어, 검증, 적합성 테스트, 자동 구현 등의 입력으로 사용되어진다.

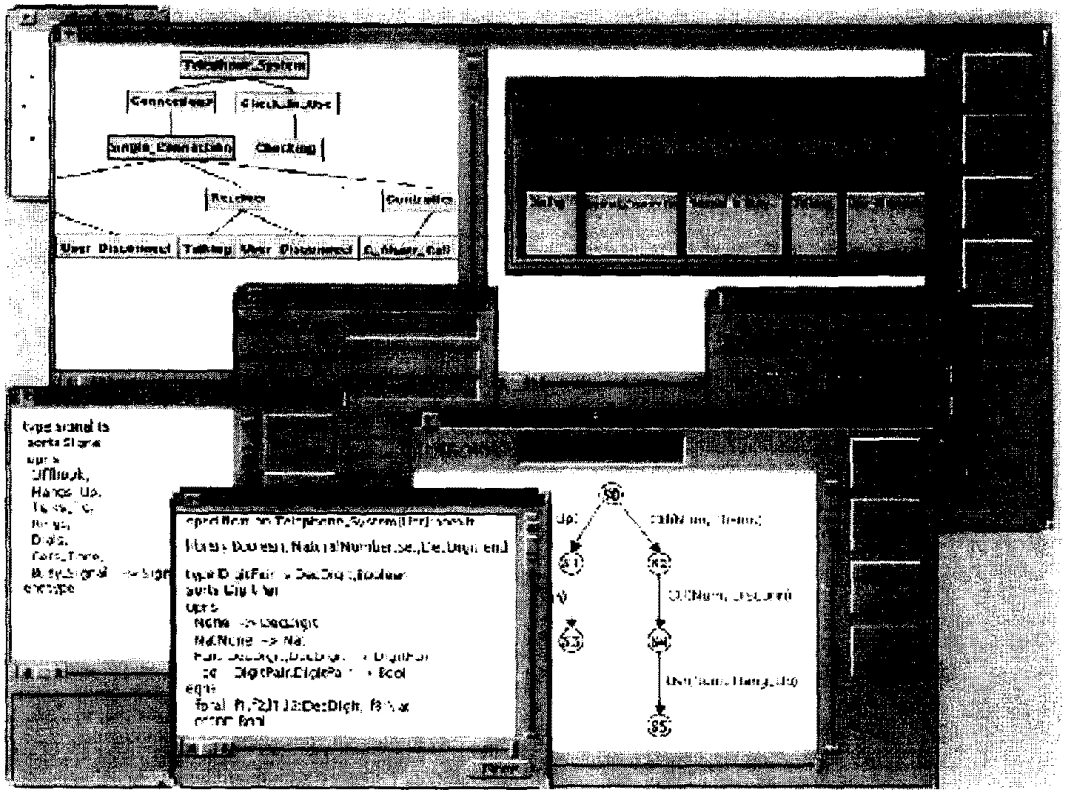
상기와 같은 기능을 갖고 동작하는 본 시스템의 유효성을 정량적으로 평가하는 것은 용이하지 않으나 다음과 같은 몇 가지 측면을 통해 그 유효성에 관해 언급하고자 한다.

1) 사용 용이성
2) 재사용의 효율성
3) 시스템 개발 효율성

1)에 관해서는, 위에서 설명한 통합형 에디터를 이용함으로써 사용자는 특정FDT에 대한 깊은 지식이나 충분한 사용 경험이 없어도 시스템에 대한 구조적 요구와 동작요구, 데이터요구만 가지고 최종적으로 원하는 형식 사양을 생성해 낼 수 있다. 이는 특정 FDT를 직접적으로 이용하는 것 보다 상대적으로 초기 학습의 부담이 적고 기술과 이해가 용이하다. 통합형 에디터에 대해서는 또한 보다 범용적인 사양 기술 도구로서의 역할도 기대할 수 있다.

2)에 관해서는, 각각의 요구 형태별 재사용을 가능하게 하였고, 검색을 위한 방법으로 종래의 키워드에 의한 방법 외에 사양간의 내용상의 유사성을 계산하여 검색하는 방법을 추가 구현하였다. 이로서 키워드 만에 의존할 경우에 생길 수 있는 문제 즉, 영역이나 분야별 온토로지의 부재로 인해 재사용 부품의 저장과 검색이 지나치게 주관적인 용어로 이루어져 저장도 차후의 재사용을 위한 검색도 어려울 수 밖에 없는 문제를 부분적으로 해결할 수 있다.

3)에 대해서는, 소프트웨어 공학 전반에 걸친 공통된 중요한 문제로서, 본 논문에서 도입한 형식사양의 사용과 형식사양 수준에서의 소프트웨어의 재사용으로, 전체 개발 공정의 상위과정에서의 검증 및 검사를 통한 빠른 피드백이 가능해져, 종래의 하위과정에서 그것이 이루어졌던 것과 비교하면 대대적인 수정, 보



(그림 6) 통합형 에디터와 자동변환기능을 이용한 Telephone_System의 기술예
(Fig. 6) Description Example of Telephone_System by using the Integrated Editor and Automatic Translation Function

수를 피할 수 있어 그만큼 개발 전체의 효율화를 기할 수 있다.

한편 다음과 같은 사항들이 문제점 또는 앞으로 해결해야 할 과제로 평가된다.

a) LOTOS에 대한 의존도 경감

b) 다양한 기술 스타일 지원

a)에 관해서는, 현재의 통합형 에디터를 이용하여 사용자의 요구를 표현하고자 할 때, 최소한의 LOTOS 의존의 표현, 예를 들어 Access-Point 나 서브프로세스간의 관계 등 을 사용하게 된다. 이는 본 시스템의 목표가 LOTOS를 크게 의식하지 않고 LOTOS기반의 형식 사양을 생성하는 것인데 대한 당면한 제약으로 작용하며, 차후 개선해야 할 과제이다.

b)에 관해서는, 현재의 통합형 에디터를 이용할 경우, [24]에서 언급하는 네 종류의 일반적인 사양기술 스타일, i)resource-oriented, ii)constraint-oriented, iii)state-oriented, iv)monolithic style 중 i), ii), iv)에 의한 기술을 지원할 수 있으나, iii)을 지원하고자 하면 EBD의 수정이 요구된다. 이는 현재의 통합 에디터가 SDL과 같이 FSM(Finite State Machine)을 기반한 것을 대상으로 하지는 않는 다는 것을 뜻하기도 한다.

본 시스템은 Top-down의 구조적인 분석 및 설계가 가능하여 시스템의 크기나 복잡성에 그다지 영향을 받지 않고 사용될 수 있다. 그러나 규모가 큰 사양을 기술하고자 할 때 화면에 다수의 윈도우가 표시되어 그들의 효율적인 정리를 위한 고려가 추가되어져야 한다. 또한 본 시스템의 최종적인 목표인 범용적인 에디터로서의 역할을 위해서는 다양한 분야에 적용하여 그 가능성을 확인하는 작업이 추후 요구된다.

5. 결 론

본 논문에서는 (1)종래의 형식적 기술기법보다 친화성이 높은 표현 기법으로 통합형 에디터의 제공, (2)(1)에서의 새로운 표현기법으로부터 국제표준의 형식적 기술 기법인 LOTOS로의 자동변환법의 제공, (3)형식사양 수준에서의 소프트웨어 재사용의 방법론을

제공하였다. 이로서 얻어질 수 있는 효과와 앞으로 더욱 검토되어 져야 할 과제는 4.에서 언급하였다.

특히, 본 논문에서 주 대상으로 하였던 LOTOS의 경우, 객체지향을 잇는 차세대 소프트웨어 패러다임으로 주목 받고있는 AOP(Agent Oriented Paradigm)의 사양기술 언어로 크게 주목 받고 있으나[25], 그 언어적 어려움으로 인하여 실질적 이용에 한계가 있는 것이 사실이다. 본 논문에서 제안하는 기능들을 종합적으로 사용함으로써, LOTOS를 크게 의식하지 않고도 최종적으로 LOTOS기반의 사양을 자동 생성할 수 있고 기존의 사양을 효율적으로 재사용할 수 있다는 측면에서 LOTOS의 보급과 이용에 도움이 될 것으로 판단된다.

향후 과제로서, 본 시스템을 인터넷을 기반으로 한 분산 소프트웨어 개발에 이용하기 위해 Java에 의한 재구현을 현재 검토 중이다.

참 고 문 헌

[1] ISO 8807, Information Processing Systems Open Systems Interconnection LOTOS A Formal Description Technique based on the Temporal Ordering of Observational Behavior, 1989.

[2] CCITT Recommendation Z.100, Specification and Description Language, 1989.

[3] ISO 9074, Information Processing Systems Open Systems Interconnection Estelle A Formal Description Technique based on an Extended State Transition Model, 1989.

[4] ISO : G-LOTOS : A Graphical Syntax for LOTOS, ISO/IEC JTC 1/SC 21N 4228, 1989.

[5] E. S. Lee, N. Shiratori and S. Noguchi, Construction and Implementation Environment SEGL based on G-LOTOS, Transactions of IPSJ, Vol.32, No.3, pp.314-323, 1991.

[6] N. Shiratori and E. S. Lee, An Integrated User Friendly Specification Environment for LOTOS, IEICE Transactions on Communications, Vol.E75-B, No.10, pp.931-941, 1992.

[7] U. Yamamoto, E. S. Lee and N. Shiratori, Reuse based Specification Support Method using Mathematical Similarity, IEICE Transactions on Fundamentals, Vol.E79-A, No.11, pp.1752-1759, 1996.

[8] C. F. Huang, T. Karahashi, E. S. Lee and N. Shiratori, A Flexible Service Development Support System for Communication Systems by Reuse Methodology, International Conference on Parallel and Distributed Systems (ICPADS'94), pp.426- 431, 1994.

[9] E. S. Lee, U. Yamamoto and N. Shiratori, Requirement Acquisition and Cases Based Specification Environment, SERA, International Conference on Information Networking (ICOIN'94), pp.279-284, 1994.

[10] E. S. Lee and N. Shiratori, ISEL : Integrated Specification Environment for LOTOS - Directed towards Multi-Level Users, International Conference on Information Networking(ICOIN'94), pp.285-290, 1994.

[11] B. H. Park, S. Kimura, E. S. Lee and N. Shiratori, 『Error Detection of Education Support Environment for LOTOS, IEICE Transactions on Fundamentals, D-II, Vol.J 80-D-II, No.4, pp.961-971, 1997.

[12] B. H. Park, S. Kimura, E. S. Lee and N. Shiratori, 『An Equivalence Algorithm to Print out Errors for Basic LOTOS in Distributed System and Its Prototype, International Conference on Parallel and Distributed Systems(ICPAD'97), pp.230-235, 1997.

[13] A. Mili, R. Mili and R. Mittermeir, Storing and Retrieving Software Components: a Refinement based System, Proc. of 16th International Conference on Software Engineering, pp.91-100, 1994.

[14] R. Milner, Communication and Concurrency, Prentice-Hall International, 1989.

[15] C.A.R. Hoare, Communicating Sequential Processes, Prentice-Hall International, 1985.

[16] H. Ehrig et al, Algebra of Communication Processes with Abstraction, TCS37, pp.77-121, North-Holland, 1985.

[17] W. H. P. v. Hulzen, LOTTE- A LOTOS Tool

Environment, Proc. of FORTE88, pp.61-66, 1988.

[18] K. Bogards, LOTOS Support System Development, Proc. of FORTE88, pp.279-294, 1988.

[19] J. A. Manas and T.d. Miguel-More, From LOTOS to C, Proc. of FORTE88, pp.79-84, 1988.

[20] K. E. Cheng and L. N. Jackson, MELBA+: An Software Engineering Environment, SDL 89: The Language at Work, Elsevier Science Publishers(North-Holland), pp.95-103, 1989.

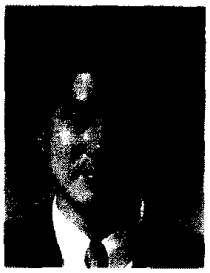
[21] C. Wild and D. Rosca, Evolution and Reuse of Formal Specification using Decision Structure, Proc. of 9th Knowledge-based Software Engineering Conference, pp.108-115, 1994.

[22] D. Jokanovic and M. Ohta, Library of Reusable Service Specification, IEICE Technical Report, SS92-25, 1993.

[23] M. Faci and B. Stepien, Formal Specification of Telephone Systems in LOTOS, Proc. of 9-th IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification, 1989.

[24] C. A. Vissers, G. Scollo and M. V. Sinderen, Architecture and Specification Style in Formal Descriptions of Distributed Systems, Protocol Specification, Testing and Verification VIII, pp.189-204, 1988.

[25] J. Williams, Bots and Other Internet Beasties, Sams Net Publishing, 1996.



이 은 석

1985년 성균관대학교 전자공학과 학사

1988년 일본 도호쿠(동북)대학교 정보공학과 석사

1991년 일본 도호쿠(동북)대학교 정보공학과 박사

1992년~1993년 일본 미쯔비씨 정보전자연구소 특별연구원

1994년 일본 도호쿠(동북)대학교 전기통신연구소 Assistant Prof.

1995년~현재 성균관대학교 정보공학과 조교수

관심분야 : 소프트웨어 공학, HCI, 에이전트지향 지능형 시스템, 인공지능응용 등임