ForTIA: LOTOS 기반의 정형기법 지원도구
(ForTIA : A Tool Supporting Formal Method based on LOTOS)

조 수 선 † 천 윤 식 †† 오 영 배 ††† 정 연 대 ††
(Soosun Cho) (Yoonsik Cheon) (Youngbae Oh) (Yun Dae Chung)

요 약 본 논문에서는 ForTIA라 불리는 LOTOS 지원도구의 개발을 소개한다. ISO 표준 정형 명세 언어인 LOTOS는 사용자의 요구사항이나 시스템 모형을 추상화하여 정형적으로 작성할 수 있도록 함으로써 구현에 이르기 전에 명세 상에서 시스템을 확인 및 검증할 수 있게 한다. ForTIA는 LOTOS 정형기법이 산업계에 적용될 수 있도록 확인 위주의 경량 정형기법 기능을 제공한다. ForTIA의 핵심적인 기능은 명세 시뮬레이션과 C++ 코드 생성이다. 시뮬레이션은 편리하고 직관적인 상호작용을 위한 트리 기반의 시각적 명세확인 메카니즘을 제공하고 C++코드생성은 LOTOS로부터 완전한 C++ 코드를 생성하여 시스템의 실제 구현에 이용될 수 있도록 한다.

Abstract In this paper, we introduce the development of a LOTOS-based tool, supporting formal methods, called ForTIA (A Formalism for Telecommunication and Information Systems). By using LOTOS, an ISO standard formal specification language, the user requirements and system models can be abstracted and represented formally. Therefore, the system can be validated and verified on the specifications, before implementations. ForTIA supports light-weight formal methods based on validation to be used in real industry. Key functions of ForTIA are simulation and C++ code generation. In simulation, tree based visual validation mechanism is provided and in code generation, the full C++ source code is generated to be used for system implementations.

1. 서 론

최근 소프트웨어 시스템은 산업의 전 분야로 사용 영역이 확산되고 있고 고 안전성 및 고 신뢰성을 요하는 시스템의 개발에 정형 기법(formal methods)을 적용하려는 시도가 늘고있다. 일반적으로 정형 기법의 적용이라 함은 수학적으로 잘 정의된 정형 명세 언어(formal specification language)를 사용하여 명세를 작성하고 이것의 검증(verification) 및 확인(validation) 작업을 통하여 올바른 명세를 획득하며, 또한 명세를 더 구체적인 형태로 변환(transform) 또는 정제(refine)함으로써 구현에 이르게 하는 일련의 과정을 말한다. 엄격한 수학적 바탕을 가진 정형 명세 언어를 사용하여 작성된 요구 명세는 애매 모호함이 없는 간결하고 정확한 표현으로 이루어져 있으므로 명세의 분석 단계에서 미리 심각한 오류들을 제거할 수 있고 이 오류들이 설계 및 구현으로 파급되는 것을 막을 수 있기 때문에 보다 효과적으로 고품질의 소프트웨어 시스템을 개발할 수 있게 한다[1].

본 논문에서는 이러한 정형기법의 실질적인 적용을 위한 LOTOS 명세 지원 도구인 ForTIA(A Formalism for Telecommunication and Information Systems)를 소개하고자 한다. LOTOS(Language Of Temporal Ordering Specification)는 개방형 분산 시스템 - 특히, OSI의 서비스와 프로토콜 - 명세를 위하여 ISO에서 개발한 정형 명세 언어이다[2]. LOTOS에서는 시스템을 하나의 프로세스로 나타내며 프로세스는 관측되지 않는 내부 동작(internal action)을 취할 수도 있고, 외부(environment)로부터 관측 가능한 동작을 취할 수도 있다. 프로세스들 간의 상호작용은 관측 가능한 동작

† 정 회 원 : 한국전자통신연구원 컴·소·연 S/W공학연구부 연구원
scho@etri.re.kr
††† 비 회 원 : 한국전자통신연구원 컴·소·연 S/W공학연구부 연구원
cheon@etri.re.kr
ydchung@computer.etri.re.kr
††† 정 회 원 : 수원여대 컴퓨터응용학부 교수
yboh@suwon-c.ac.kr
논문접수 : 1999년 3월 11일
심사완료 : 2000년 2월 21일

(action)들이 프로세스들 사이에서 동시에 발생함으로써 동기화(synchronization)될 때 일어난다. 따라서 LOTOS에서 시스템의 행위는 외부에서 관측되는 동작들 간에 순차적인 관계를 정의함으로써 표현되며 이를 위하여 Milner의 CCS(Calculus of Communicating Systems)와 Hoare의 CSP(Communicating Sequential Processes)와 같은 프로세스 대수학이 도입되었다. 또한 상호작용시 교환되는 자료를 표현하기 위하여 대수방식의 정형 명세 언어인 ACT ONE을 사용한다. 자료 부분의 표현 없이 CCS/CSP 형태로만 표현된 것을 기본(basic) LOTOS, 자료 부분까지 포함하는 것을 완전(full) LOTOS라고 부르기도 한다. 따라서 자료와 제어의 상호보완적인 명세를 위해 ACT ONE과 CCS/CSP라는 두 정형기법을 결합한 것이 LOTOS이다.

정형 기법이 효과적으로 적용되기 위해서는 소프트웨어 개발을 지원하는 지원 도구와 개발 환경이 필수적이다. 특히, 정형 기법이 산업계에 적용되기 위해서는 정형 증명과 같은 이론 지향적인 도구보다는 실용적인 - 예를 들면, 사용자의 요구사항이 명세에 제대로 반영되었는지 파악할 수 있는 확인 위주의 경량 도구가 더욱 필요하다. ForTIA는 명세 시뮬레이션과 코드 생성을 대표적인 기능으로 제공함으로써 LOTOS 정형 명세를 이용한 요구 사항 분석에서부터 구현에 이용되는 C++ 소스 코드의 생성까지 실질적인 개발 과정을 지원한다.

본 논문은 다음과 같이 구성되었다. 이어지는 제 2절에서는 전체 시스템의 구성 및 기능을 소개하고, 제 3절과 4절에서는 대표적인 단위도구인 명세 시뮬레이터와 C++코드생성기에 대하여 각각 자세히 설명한다. 5절에서는 ForTIA의 구현 및 사용 예에 대하여 소개하고 마지막으로 6절에서 결론을 맺는다.

2. 시스템 구성 및 기능

2.1 ForTIA의 구성

ForTIA는 그림 1과 같이 구문지향 편집기, 시뮬레이터, 코드생성기, 자료평가기 등 대표적인 4가지 단위도구로 구성된다. 이 단위도구들 간의 자료 교환을 위한 공동의 내부 포맷으로는 LOTOS 구문트리가 사용된다. LOTOS 구문트리는 LOTOS명세로부터 구문/의미 검사를 거친 후 생성되는 추상구문트리(Abstract Syntax Tree, AST)의 일종으로서 구문 및 정적 의미 상에 오류가 없음이 보장되어 타 단위도구의 입력으로 사용된다. 4가지 단위도구와 구문/의미 검사는 Win95/98 환경의 GUI를 통해 사용자와 인터페이스할 수 있다. 각 단위도구에 대한 간단한 설명은 다음과 같다.

- 구문지향 편집기 : LOTOS 문법에 기반한 구조적인 입력을 위한 편집기로서, 사용자가 원하는 LOTOS 구문을 문법에 맞게 채워나가는 템플릿 입력 방식의 편집 기능을 제공한다. LOTOS 구문에 익숙하지 않은 사용자라 할지라도 구문 오류 없이 손쉽게 명세를 작성할 수 있다.
- 시뮬레이터 : LOTOS 명세가 사용자의 요구 사항을 만족하는지 확인하기 위하여 명세를 모의 실행을 할 수 있도록 한다. 명세의 실행 과정을 단계적으로 검사하는 단계 시뮬레이션 및 가능한 모든 실행 경로를 한꺼번에 검사하는 명세 확장 기능을 제공한다.
- 코드생성기 : 명세로부터 실행 가능한 완전한 C++ 프로그램을 자동 생성한다. 생성된 C++ 프로그램은 명세의 동작 모형으로 이용될 수 있고 시스템의 실제 구현에 사용될 수도 있다.
- 자료평가기 : LOTOS 명세의 자료 부분에 대한 검사를 위해 자료 표현식을 정규형으로 개서한 결과를 보여 줌으로써 자료 타입의 정확한 명세를 돕는다.

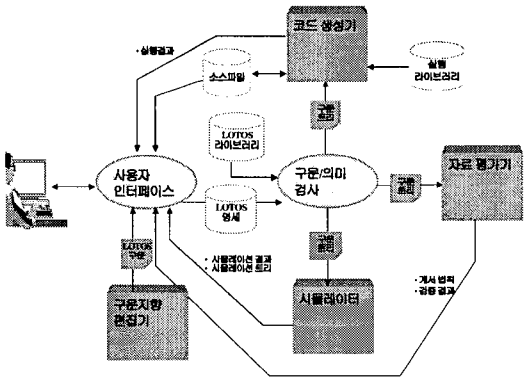


그림 1 ForTIA 구성

2.2 대표 기능

ForTIA의 기능은 앞에서 소개한 4가지 대표적인 단위도구의 기능으로 정의될 수 있다. 하지만 그 중에서도 특히 시뮬레이션과 코드생성은 명세의 확인 작업과 구현에서의 재사용을 위한 대표적인 기능이므로 본 논문에서는 이 두 가지 기능을 중심으로 ForTIA를 소개한다.

2.2.1 명세 시뮬레이션 기능

ForTIA 시뮬레이션은 크게 행위이동, 단계시뮬레이션, 명세확장의 3가지 기능을 제공한다. 행위이동은 전

체 명세 중에서 관심 있는 부분 행위로 내부 커서를 옮겨주는 기능이다. 내부 커서가 가리키는 부분 행위를 루트로 하여 명세확장과 단계시뮬레이션이 이루어진다.

LOTOS 명세의 의미론은 라벨형 전이시스템(Labeled Transition System, LTS)으로 정의되기 때문에 명세의 단계적 확인은 명세로부터 전이시스템을 생성함으로써 가능해진다. 따라서 주어진 명세를 동작의 선택에 의한 상태 전이(sate transition)의 연속으로써 실행시킬 수가 있다. 이것을 명세의 시뮬레이션이라 하고 이 작업은 사용자와 상호 작용하는 시뮬레이터를 이용하여 수행된다. 일반적인 의미의 시뮬레이션이 step-by-step으로 이루어지기 때문에 ForTIA에서는 명세 확장과 구분하기 위해 단계 시뮬레이터라는 이름으로 개발되었고 시뮬레이션 절차에 따른 시각적인 사용자 인터페이스를 주요 특징으로 제공한다.

ForTIA에서 구현된 명세 확장(Expansion)도 크게 보면 시뮬레이션에 속한다. 단계 시뮬레이션이 사용자와의 상호 작용에 의해 선택된 동작으로 상태 전이를 계속해 나가는 것이라면 명세 확장은 가능한 모든 종류의 전이를 자동으로 발생시키면서 상태결합에 도달하면 멈추는 방법으로 유한 상태기(Finite State Machine, FSM)를 자동 생성한다. ForTIA의 명세확장에서는 확장의 결과를 그래픽 트리로 표현하여 한눈에 결과를 파악하는데 도움을 준다.

2.2.2 C++ 코드생성 기능

ForTIA의 코드생성 기능은 LOTOS 명세로부터 C++ 소스 코드를 자동 생성한다. LOTOS 명세는 사건이 일어나는 시간적 순서를 묘사하는 행위 정의 부분과 자료값의 특성을 묘사하는 자료 타입 정의 부분으로 구성되고 자료 값 표현식의 의미 해석을 위하여 행위 부분이 자료 부분을 참조한다는 것을 제외하면 둘은 완전히 독립적으로 동작한다. 따라서 ForTIA는 자료 부분과 행위 부분에 대하여 독립적인 코드 생성을 지원한다. 자동 생성된 C++ 코드는 명세의 동작 모형을 제공하거나 실제 구현에 사용될 수 있다. 명세의 동작 모형은 프로토타입의 생성 및 실행을 통하여 사용되며 명세에 내포된 시스템 요구 사항 오류를 개발 초기에 검출할 수 있도록 한다. 생성된 코드는 조율하거나 외부 프로그램과 연계하여 실제 구현에 직접 사용될 수도 있다. 이를 위하여 ForTIA 코드생성기에서는 주석, 수작업 코드와 연계, API 등을 제공한다.

ForTIA는 위에서 나열한 대표적인 기능 이외에도 명세의 구문 및 정적 의미 검사와 같은 부분적인 검증 기능을 제공한다. 따라서 사용자의 목적과 취향에 따라 다양한 형태로 사용될 수 있다. 이와 같은 ForTIA의 기능을 이용함으로써 시스템 개발자는 PC Win95/98 환경의 편리한 인터페이스하에서 LOTOS기반의 정형기법으로 시스템을 점진적으로 개발할 수 있다. 가시화된 명세의 시뮬레이션은 사용자 요구 명세의 오류를 신속하고 손쉽게 검출할 수 있게 하며, 자동화된 코드 생성을 통하여 소프트웨어의 품질을 향상시키고 신속한 프로토타이핑 및 재사용 가능한 C++ 코드를 생성한다.

2.3 기존 도구와의 비교

프랑스 INRIA에서 개발한 CADP(CAESAR/ALDEBARAN Development Package)는 LOTOS 지원 통합도구이다[3]. 이것은 기존의 단위 도구 CAESAR와 ALDEBARAN을 중심으로 하고, 몇몇 보조적인 단위 도구를 추가하여 하나의 패키지로 만든 것이다. CAESAR와 CAESAR.ADT는 각각 LOTOS의 행위 부분과 자료 부분으로부터 C코드를 생성하는 컴파일러이고, ALDEBARAN은 LTS로 표현된 통신 시스템의 검증을 위한 도구이다. 검증을 위해 bisimulation 관계를 이용하여 LTS들을 최소화하고 비교한다. CADP는 최근까지 업데이트되고 있으며 가장 많은 적용 사례를 가지고 있다. CADP가 ALDEBARAN이라는 검증 위주의 도구와 C 코드 생성기를 중심으로 한 통합 도구인 반면, ForTIA는 확인 위주의 시뮬레이션 도구와 C++ 코드 생성기를 대표적인 단위도구로 포함하고 있다.

대표적인 LOTOS 시뮬레이션 도구는 네덜란드의 트웬테 대학에서 개발한 SMILE[4]과 캐나다 오타와 대학에서 개발한 ISLA[5]이다. 이들은 모두 단계 시뮬레이션을 주 기능으로 한다. SMILE은 ISLA에 비해 심볼형 전이(symbolic transition)를 지원한다는 점에서 한 차원 높은 기능을 제공한다. 일정한 깊이(depth)만큼 자동적으로 전이 트리를 생성하는 명세 확장 도구로는 마드리드 대학의 LOLA[6]와 오타와 대학의 SELA[5]가 있다. 이 두 도구는 모두 LOTOS의 확장 정리(Expansion Theorem)를 이용하여 유한 상태기 형태의 전이 트리를 생성한다. LOLA는 또 다른 LOTOS 명세로의 변환 방법을 취하는 반면 SELA는 LOTOS 명세를 Prolog로 바꾼 후에 전이 트리를 자동 생성한다.

대표적인 LOTOS 코드 생성 도구는 마드리드 대학에서 개발한 TOPO이다[7, 9.3절]. TOPO는 코드 생성을 두 단계로 구분한다. 먼저 명세로부터 가상 머신(Virtual Machine) 코드라 불리는 중간 코드를 생성하고 이를 C 또는 ADA 코드로 변환한다[8]. 이러한 접근 방법은 역할을 분담하고 기능을 모듈화 하여 도구의 확장 및 유지 보수를 용이하게 한다. 자료부분은 개서

시스템(rewrite system)을 바탕으로 하며 행위 부분은 유한 오토마타를 바탕으로 한다. 또 다른 코드 생성 도구인 COLOS는 독일의 베를린 대학에서 개발되었는데 TOPO와 달리 행위부분만을 지원한다[7, 9.5절]. LOTOS 자료부분을 C 코드로 변환하는 CAESAR. ADT에서는 패턴 매칭 컴파일러 알고리즘을 사용하여 대수 명세를 C 언어의 자료구조와 프로시져로 정적 컴파일하고 결정 코드를 생성한다[9].

소개된 도구들 중 TOPO는 C와 ADA 코드를 생성하고 있으나 요즘 산업계에서 널리 사용되고 있는 C++ 언어를 지원하지는 않는다. 이론적으로는 C가 C++의 부분집합으로 C++가 사용되는 곳에서 C를 사용할 수 있으나, 실제적으로는 통합이 쉽지 않으므로 별개의 C++ 코드 생성기를 개발하는 것은 매우 의미 있는 일이다. 또한 생성되는 코드는 효율적인 실행을 지원해야 하고, 코드를 재사용할 경우에 쉽게 이해하고 수정할 수 있는 구조를 가져야한다. C++ 언어를 사용하면 위의 두 요건을 만족할 수 있다. 또한 기존의 도구들은 대부분 UNIX 환경을 기반으로 하고 있으며 LOLA와 CADP는 각각 PC의 DOS와 Linux를 지원하고 있다. 요즘의 일반적인 OS 환경인 Win95/98 환경을 지원하는 것은 찾아보기 힘들고, 대학의 실험실에서 개발된 것이 대부분이므로 사용자와의 상호작용이 빈번하게 발생하는 시뮬레이션에서 이를 위한 직관적인 그래픽 인터페이스를 지원하는 것도 드물다. 본 논문에서 소개하고자 하는 ForTIA는 실제로 산업계에서 쓰이는 개발 언어인 C++로의 코드 생성을 지원하고, Win95/98 등 범용 OS를 바탕으로 하며 시뮬레이션을 위한 효과적인 상호작용 메카니즘을 제공하는 보다 진보된 형태의 LOTOS기반 정형 기법 도구이다.

3. 명세 시뮬레이터

3.1 단계 시뮬레이터

3.1.1 전이 시스템과 시뮬레이션 트리

라벨형 전이시스템에서 LOTOS의 행위 표현(behavior expression)은 상태에 해당하고 동작은 전이에 해당한다. 주어진 상태에서 가능한 모든 전이들의 집합은 공리와 추론 규칙으로 구성되는 추론 시스템(inference system)으로 정의된다. 자료부분을 포함하지 않는 기본 LOTOS에서 동작 접두어(action prefix)와 성공적 종결(successful)은 아래와 같은 공리로 정의된다

action prefix	$g;B \xrightarrow{g} B$
successful termination	$\mathbf{exit} \xrightarrow{\delta} \mathbf{stop}$

다른 모든 행위 구성자들은 하나 이상의 추론규칙으로 정의된다. 아래 표는 선택(choice), 병렬(interleaving), 동기화(synchronization) 연산자에 대한 추론 규칙을 나타낸다.

표 1 LOTOS 연산자의 추론규칙

choice	$\frac{B_1 \xrightarrow{g} B_1' \quad B_2 \xrightarrow{g} B_2'}{B_1[]B_2 \xrightarrow{g} B_1' \; B_1[]B_2 \xrightarrow{g} B_2'}$
interleaving	$\frac{B_1 \xrightarrow{g} B_1', g \neq \delta \quad B_2 \xrightarrow{g} B_2', g \neq \delta \quad B_1 \xrightarrow{\delta} B_1', B_2 \xrightarrow{\delta} B_2'}{B_1 \vert\vert\vert B_2 \xrightarrow{g} B_1' \vert\vert\vert B_2 \; B_1 \vert\vert\vert B_2 \xrightarrow{g} B_1 \vert\vert\vert B_2' \quad B_1 \vert\vert\vert B_2 \xrightarrow{\delta} B_1' \vert\vert\vert B_2'}$
synchronization	$\frac{B_1 \xrightarrow{i} B_1' \quad B_2 \xrightarrow{i} B_2' \quad B_1 \xrightarrow{g_1} B_1', B_2 \xrightarrow{g_2} B_2', g_1 = g_2, g_1 \neq i}{B_1 \vert\vert B_2 \xrightarrow{i} B_1' \vert\vert B_2 \; B_1 \vert\vert B_2 \xrightarrow{i} B_1 \vert\vert B_2' \quad B_1 \vert\vert B_2 \xrightarrow{g_1} B_1' \vert\vert B_2'}$ where i denotes the unobservable, or *internal* event
recursion	$\frac{B \xrightarrow{g} B'}{p \xrightarrow{g} B'}$ where $p:=B$ is a process definition

한 상태에서 두개 이상의 전이가 가능할 때, 선택된 이벤트에 의해 각각 다른 상태로 전이되며 그 결과는 경로(trace)로 나타난다. 예를 들어, 아래와 같은 기본 LOTOS 행위 표현식이 있다고 하자.

g; (h; exit ||| g; exit)

시뮬레이터는 동작 접두어 공리를 적용하여 g로 라벨화된 전이를 생성하며 다음 상태는 아래와 같이 된다.

h; exit ||| g; exit

이 상태에서는 사용자의 선택에 따라 g와 h 두 가지의 전이가 일어날 수 있으며 선택된 이벤트들의 나열이 경로가 된다. 대개 시뮬레이터는 backtrack과 또 다른 이벤트를 실행시키는 것을 허용하며, 이러한 방법으로 완전한 트리가 생성되면 이것을 시뮬레이션 트리라 한다. 이 트리에서 두개의 상태가 서로 일치하는 것이라면 하나의 상태로 표시하여 트리 대신 그래프로 나타내기도 한다. 이것을 상태결합(state matching)이라 한다[10, 2.1절]. 그림 2는 위의 예에서 만들어진 시뮬레이션 트리와 그래프이다.

3.1.2 단계 시뮬레이션 인터페이스

LOTOS 단계 시뮬레이션에서는 다음과 같은 절차를 적용하여 명세에 표현된 시스템의 행위가 명세자의 의도대로 올바르게 실행되는지 손쉽게 분석할 수 있다[11].

- 시뮬레이터는 현재 상태에서 동기화 가능한 모든 동작을 계산하여 메뉴로 제시한다.
- 사용자는 동작 메뉴에서 관심 있는 하나의 동작을

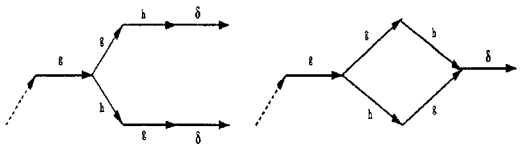


그림 2 시뮬레이션 트리와 시뮬레이션 그래프

선택한다. 이 때 사용자는 명세된 시스템의 환경으로 작용하여 선택한 동작으로 동기화를 이루는 역할을 한다.

- 시뮬레이터는 현재 상태와 선택된 동작으로 다음 상태를 계산하여 그 상태로 전이한 후 다시 동작 메뉴를 제시한다.
- 사용자의 동작 선택에 의해 상태 전이가 반복된다.

ForTIA에서는 이러한 단계 시뮬레이션 절차를 앞 절에서 소개한 시뮬레이션 트리를 이용하여 시각적으로 지원한다. 시뮬레이션 트리를 그래픽으로 제시하고 사용자 인터페이스가 모두 이 그래픽 트리 상에서 이루어지므로 효율적인 명세 분석이 가능하게 된다. 구체적인 인터페이스의 내용은 다음과 같다.

- 개념적인 시뮬레이션 트리를 그래픽으로 구성하여 화면상에 표시해 준다.
- 사용자는 그래픽 시뮬레이션 트리상에서 도구와 상호작용할 수 있다. 현재 상태의 노드들 중 하나를 마우스 클릭으로 선택하여 다음 상태로 전이를 이룰 수 있다. 선택 가능한 노드들이 동작 메뉴에 해당한다.
- 사용자의 선택에 의해 새로 구성된 그래픽 시뮬레이션 트리를 다시 표시해 준다. 단계 시뮬레이션의 절차상 상태 전이에 해당하고 새로 생긴 노드들이 새 동작 메뉴가 된다.
- 사용자는 그래픽 시뮬레이션 트리에서 마지막 부모 노드를 클릭함으로써 최근의 상태 전이를 취소하고 이전 상태로 되돌아 갈 수 있다.
- 그래픽 시뮬레이션 트리에서 현재까지의 경로를 나타내는 노드들의 연결선들을 구분되는 색으로 표시하여 한눈에 알아 볼 수 있게 한다.

아래의 명세는 단계 시뮬레이션의 예로서 사용되는 LOTOS 행위 명세이다. 간단한 자동판매기를 나타내는 이 LOTOS 명세는 두 종류의 동전을 받아들여서 그 값이 기준치 이상이 되면 두 종류의 음료를 제공하거나 또는 동전을 반환하는 기능을 가진다. 자료형의 명세는 생략되었다.

```
behaviour
  VM[coin, cofB, cokB, retB] (0)
where
process VM[coin, cofB, cokB, retB] (n:Nat) : noexit :=
  coin !1; VM[coin, cofB, cokB, retB] (n+1)
  [ ]
  coin !2; VM[coin, cofB, cokB, retB] (n+2)
  [ ]
  cofB !coffee [n ge 1]; VM[coin, cofB, cokB, retB] (n-1)
  [ ]
  cokB !coke[n ge 2]; VM[coin, cofB, cokB, retB] (n-2)
  [ ]
  retB !n [n gt 0]; VM[coin, cofB, cokB, retB] (0)
endproc
```

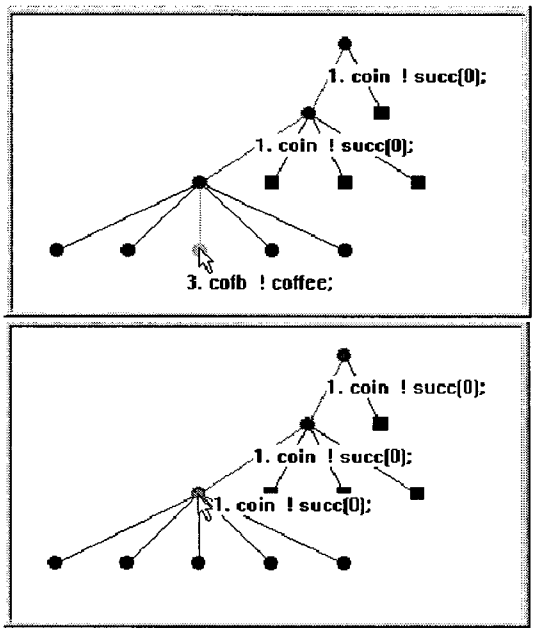


그림 3 ForTIA의 그래픽 시뮬레이션 트리

이 명세를 ForTIA의 단계 시뮬레이션을 통해 그래픽 트리로 표현해 보면 그림 3과 같다. 현재 상태는 coin !1을 연속으로 두 번 발생시켰을 때이다. 위쪽 그림은 사용자가 현재 상태에서 동작 cofb !coffee를 선택하는 순간이며 아래쪽 그림은 전이 취소를 위해 부모 노드를 클릭하는 순간이다. 이와 같은 향상된 인터페이스를 통해 시뮬레이션 중에 동작 메뉴 및 경로를 포함한 현재

상태를 한눈에 파악할 수 있고 동작 선택과 취소를 쉽고 빠르게 수행할 수 있어 효율적인 명세 분석 작업이 가능하게 된다.

3.2 명세 확장 시뮬레이터

3.2.1 FSM과 확장 트리

LOTOS는 Temporal Ordering 이란 이름에서 나타나듯이 이벤트의 동시성을 모든 가능한 종류의 이벤트의 순차적인 발생으로 해석한다. 따라서 LOTOS 명세는 유한 상태기(FSM)로 변환될 수 있고 이것은 LOTOS 확장정리[2]를 일반화하여 적용함으로써 가능해진다. FSM 변환을 위하여 Petri-Net, Prolog 등이 이용되기도 하고[5] 또 다른 LOTOS 명세를 생성하기도 한다. LOLA에서는 병렬 연산자를 포함한 LOTOS 명세를 동작접두어, 행위선택(behaviour choice), 가드(guards) 및 프로세스만을 가진 확장된 LOTOS 명세로 바꾸어줌으로써 FSM을 생성한다[6]. 이 때 변환 대상이 되는 명세는 병렬 연산자나 불가능/가능 연산자와 함께 재귀 용법이 사용되지 않아야 하는 제약이 따른다.

ForTIA에서는 LOLA의 LOTOS 명세의 확장 모듈을 이용하여 FSM을 생성하며 이것을 트리로 재구성하여 표현해 줌으로써 사용자로 하여금 시스템의 행위를 보다 쉽고 빠르게 분석할 수 있게 해 준다. 사용자는 현재 상태를 나타내는 LOTOS 명세를 확장해 봄으로써 가능한 모든 전이의 종류와 개수를 알 수 있고 교착점(deadlock)이 발생하는지 한눈에 파악할 수 있다. 명세 확장은 상태 탐색뿐만 아니라 효과적인 구현 유도나 모델 확인을 위한 입력 그래프 등에 사용할 수 있다. 일반적인 확장 규칙에서 확장을 멈추는 경우는 상태결합이나 교착점에 도달했을 때와 주어진 확장 깊이에 도달했을 때이다.

3.2.2 명세 확장 인터페이스

확장 트리의 그래픽 표현은 사용자의 이해를 돕고 상태 분석에 대한 전체적인 파악에 큰 도움을 준다. 더구나 확장 결과에 의해 생성된 상태들의 종류를 그래픽으로 한눈에 구분할 수 있다면 더욱 유용할 것이다. ForTIA에서는 LOTOS 명세 확장을 위해 다음과 같은 인터페이스를 제공한다.

- 명세 확장에 의해 생성된 모든 상태를 노드로 표시하고 전이를 에지로 표시하는 그래픽 확장 트리를 생성하여 디스플레이한다.
- 그래픽 확장 트리에서 노드들은 분석된 상태, 결합 상태 등 상태의 종류에 따라 다른 색으로 구분해 준다.
- 그래픽 확장 트리에서 마우스로 노드를 선택하면 자세한 상태 정보를 제공해 준다.

그림 4는 ForTIA의 명세 확장 결과 화면이다. 확장 깊이를 3으로 준 경우이며 위쪽, 왼쪽부터 일련의 노드 번호가 부여된다. ForTIA의 그래픽 확장 트리에서는 녹색으로 표시되는 분석 노드 수가 총 3개이며 청색의 결합 노드 수가 1임을 알 수 있고 이들 중 하나를 선택했을 때 그 상태를 나타내는 확장된 명세를 확인 할 수 있다. 그림에서는 노드 4의 상태를 나타내고 있다. 또한, 확장 깊이에 도달하여 분석되지 못한 상태는 백색의 노드로 표시하고 있다. ForTIA는 LOTOS 명세 확장을 위해 이와 같은 기능을 제공함으로써 상태들간의 연결관계, 즉 전이의 전후 관계를 파악할 수 있게 하고 관심있는 상태가 어떤 경로를 통해 도달되는지 쉽게 알아볼 수 있게 한다. 이 것은 다른 도구들이 단순히 결과 통계치만 제시하는데 비해 보다 명확한 명세의 분석 작업이 이루어질 수 있도록 지원한다.

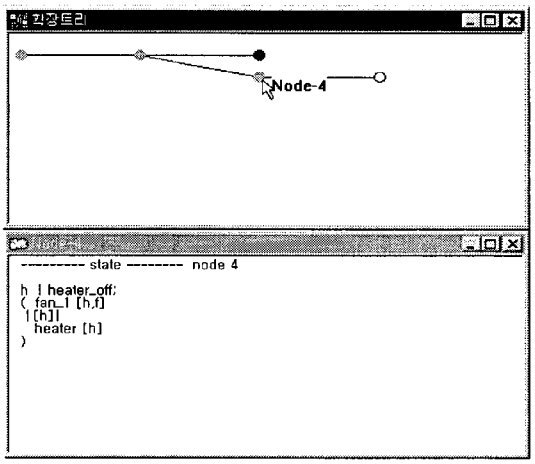


그림 4 ForTIA의 그래픽 확장 트리

4. C++ 코드 생성기

4.1 자료코드 생성기

4.1.1 개서 시스템 기반의 코드 생성

일반적으로 자료 명세는 소트 정의, 연산자 정의, 자료 등식의 세 부분으로 구성된다. 먼저 추상 값의 집합을 나타내는 소트를 정의하고 연산자라 불리는 소트 상에 정의된 전칭 함수를 정의한다. 연산자의 의미는 연산자 상호 관계를 등식으로 서술한다. 아래 명세는 소트 정의, 연산자 정의, 자료 등식이 포함된 간단한 LOTOS

자료 명세이다.

```
type Stack is Boolean, Natural
sorts Stack
opns
  empty: -> Stack
  push: Stack, Nat->Stack
  pop: Stack->Stack
  top: Stack->Nat
  isempty: Stack->Bool
eqns
  forall s:Stack, n, m:Nat
  ofsort Nat
  top(push(s,n)) = n;
  ofsort Stack
  pop(push(s,n)) = s;
  pop(empty) = empty;
  ofsort Bool
  isempty(empty) = true;
  isempty(push(s,n)) = false;
endtype
```

LOTOS 자료명세에 사용된 ACT ONE의 수학 모형은 다종 대수(many-sorted algebra)이다. 특히, 초기 의미론(initial semantic)을 갖는데[12], 이는 수학적으로 잘 정립되었으나 명세의 실행 모형으로 사용될 수 없다. 따라서 텀 개서 시스템(term rewrite system, TRS)을 사용한다. 개서 시스템은 등식에 종속 혹은 독립적인 동적 개서 엔진이나 패턴 매칭 기반의 정적 컴파일 방법으로 구현할 수 있다. 가장 일반적인 방법은 등식에 방향성을 부여하여 개서 규칙으로 간주하는 것이다. Caesar.ADT와 TOPO가 이 방법을 채택하고 있다. 명세의 등식으로부터 개서 규칙을 도출하고 개서 규칙에 따라 텀을 개서한다. 예를 들면, 연산자 pop을 제약하는 두 등식은 다음과 같은 개서 규칙으로 변환할 수 있다.

```
pop(push(s,n)) → s;
pop(empty) → empty;
```

이렇게 생성된 개서 시스템은 완전성을 가지지 않을 수도 있으며 완전성 보장을 위해서는 Knuth-Bendix 알고리즘이나 이의 확장을 사용하여야 한다.

4.1.2 ForTIA의 자료코드 생성 방법

LOTOS 자료명세로부터 C++ 프로그램을 생성하기 위해 소트는 C++ 클래스로 구현하고 연산자는 기본적으로 해당 클래스의 멤버 함수로 구현한다[13]. 예를 들면, 앞 절의 Stack 명세는 다음과 같은 C++ 헤더 파일을 생성한다. (CVRef는 동적 메모리 관리를 위한 템플릿 클래스이다.)

```
CVRef<Stack> empty();
class Stack: public KSort {
public:
  Stack(int oid = 0, KLink *p = (KLink*) NULL) :
KSort(oid,p) {}
CVRef<Bool> isempty();
CVRef<Nat> top();
CVRef<Stack> pop();
  CVRef<Stack> push(const CVRef<Nat>&);
};
```

따라서 LOTOS 값 표현식 pop(push(empty, 0))에 해당하는 C++ 문장은 다음과 같다.

```
CVRef<Stack> s = empty()->push(zero())->pop();
```

프로그램 모듈화를 위해 생성된 모든 클래스는 커널 클래스라 불리는 시스템 클래스 KSort의 서브 클래스로 정의된다. 따라서 실행 가능한 완전한 C++ 프로그램은 자동 생성 코드와 커널 클래스가 결합된 형태이다. 커널에서는 자동 생성 클래스의 공통된 특성이 정의되어 값을 구현하기 위한 자료 구조나 이를 조작하기 위한 함수를 제공한다. 특히 내부 자료구조의 초기화, 대수명세의 초기 모형 구현, 추상 값의 스트링 변환, 동적 메모리 할당과 같은 서비스를 제공하며, 자동 생성 클래스의 조율과 수작업 코드와 연계를 위한 메카니즘도 제공한다.

코드 생성은 추상 명세를 실행 가능한 형태로 구체화하는 과정으로서 명세의 추상값이 C++ 자료구조로 표현되어야 하고 연산자가 C++ 함수로 변환되어야 한다. 대수 명세에서 추상값은 값 표현식 즉, 해당 소트를 결과 소트로 갖는 연산자의 적용으로 표현된다. 예를 들면, push(push(empty, 0),0)은 소트 Stack이라는 집합에 속하는 추상값이다. 추상 자료값을 구현하는 방법은 다양하나 가장 일반적인 방법은 추상 구문 트리를 사용하는 것이다. 연산자는 C++의 멤버 함수로 변환된다. 자료 명세의 등식으로부터 개서 규칙만 도출되면, 멤버 함수로의 변환은 기계적으로 처리될 수 있다. 예를 들면, 소트 Stack을 C++ 클래스 Stack으로 구현한다고 가정할 때, 연산자 pop은 다음과 같은 형태의 C++ 멤버 함수로 구현할 수 있다.

```
Stack Stack::pop()
{
  switch (this) {
```

```
    case "of the form push(s,n)";
    return s;
    break;
    case "of the form empty";
    return empty();
    break;
    }
}
```

4.2 행위코드 생성기

4.2.1 TOPO의 행위 컴파일러

ForTIA 행위 코드 생성기의 개발에서는 TOPO에서 생성한 중간 코드(가상 머신 코드)를 입력으로 하는 방법이 채택되었다. 이 방법은 TOPO의 문법/의미 검사 모듈을 재사용할 수 있고, 잘 정의된 가상 머신 코드에서 목표 코드를 생성할 수 있으므로 안정적인 시스템을 단기간 내에 개발할 수 있는 효과적인 방법이다. TOPO의 행위 컴파일러는 C 또는 Ada 코드를 생성하기 전 단계인 중간 코드 형태로 확장 오토마타와 유사한 BUT(Behaviour UniT)의 집합을 생성한다. 이 오토마타는 프로세스 생성, 관리, 통신 기능을 지원하는 전통적인 운영체계와 유사한 커널과 협력하여 작동한다. TOPO에서는 주로 프로세스들간의 통신에 관심을 두고 있다. 오토마타는 각 상태에서 통신 요구사항 - LOTOS 용어로 동기화 환경 - 을 커널에 알려 준다. 오토마타는 커널을 통하여 다른 오토마타에 동작을 전달하고 동기화가 되면 다음 상태로 전이한다.

커널은 동기화 요구와 행위 정보를 저장하고, 이 정보에 따라 동기화가 일어날 부분을 결정한다. 그리고, 해당 오토마타가 다음 상태로 이동하기 위해서 활성화된다[14]. 여러 가지 LOTOS 컴파일러가 있으나 완전한 LOTOS 문법을 지원하는 것은 TOPO의 방법뿐이다[15]. 커널 모듈은 미리 만들어져 있어서 사용자가 작성한 LOTOS 명세로부터 생성된 코드와 연결되어 완전한 실행 가능한 프로그램을 만든다.

4.2.2 ForTIA의 행위코드 생성 방법

행위코드 생성에서는 목표코드가 TOPO의 중간코드를 기반으로 정의되어야 하므로 TOPO 가상머신 구조에 따라 C++ 코드 형식을 정의한다. 중간코드를 C++로 표현할 때, 가장 중심이 되는 문제는 BUT을 어떻게 표현할 것인가 하는 것이다. 본 연구에서는 TOPO의 BUT을 C++의 클래스로, BUT의 행위는 그 클래스의 멤버 함수로, BUT의 자료는 private data로 표현한다. TOPO에서 함수, 프레임 등으로 흩어져 있는 구조를 클래스 구조를 사용하여 한 곳에 모으고, 간결한 코드 생성이 가능하도록 하였다. 실제 코드에서는 BUT 클래스에 BUT이 가질 수 있는 기본 기능을 묘사한다. 아래의 BUT 클래스에서 offer 멤버 함수는 오토마타의 행위를 커널에 전달하는 역할을, values 변수는 BUT에서 사용되는 변수의 값을 저장하는 역할을 담당한다. LOTOS 명세의 프로세스로부터 생성되는 BUT은 BUT 클래스를 상속하고 자신만의 특성 즉, 이름, 변수, 변수의 소트, 오토마타 등을 추가로 묘사한다.

```
class BUT {
privates:
  int next;  /* next entry point */
protected:
  Offer* offer;
  CVRef<XSort> values[MAX_NUM_VAL];
public:
  BUT(int n = 0);
  BUT(BUT *obj);
  Cmd* run() {
  wipeOutOffer();
  offer = new Offer;
  return _run();
  };
virtual Cmd* _run();
virtual char* name() { return ""; };
  void setNext(int n) { next = n; };
  int getEntryPoint() { return next; };
  ...
  ~BUT();
};
```

4.2.3 커널 코드

커널은 BUT의 정보를 받아서 동기화될 동작을 선정하여 실행시키는 기능을 수행한다. 이 코드는 미리 만들어져 있고 사용자가 작성한 LOTOS 명세로부터 생성된 코드의 실행을 지원한다. 커널의 가장 중요한 기능은 행위정보를 커널 트리로 관리하면서 동기화를 실현하는 것이다. ForTIA 행위코드 생성기 커널의 특징은 생성 코드의 형식에서 드러난다. TOPO가 제시한 코드 형식은 생성된 코드가 직접 커널 함수를 호출하여 커널 트리 연산을 수행하도록 설계되어 복잡하고 읽기 어려운 반면, ForTIA에서 설계된 코드는 오토마타의 각 상태 정보만 나열하도록 되어 있어 간결하고 읽기 쉽다. 커널 트리에 관련된 연산은 커널이 담당하도록 한 결과이다. 이렇게 함으로써 생성 코드를 재사용 할 경우 코드를 이해하기 쉽고, 수정이 용이하다.

커널이 게이트의 동기화를 검사하기 위해서는 게이트에 제공된 값의 동치성을 검사해야 한다. 커널 코드는 미리 만들어져 있어야 하므로 커널이 작성될 때는 어떤 소트가 있는지 알 수 없고 모든 소트의 포괄 소트

(generic sort)만을 알 수 있다. 자료 컴파일러에서 정의한 포괄 소트는 XSort이다[13]. XSort로 저장된 두 값을 x, y라고 가정하고, 두 값이 동일한지 검사하려면 equal이라는 독립함수를 사용하여 "x->equal(y)"를 실행하면 된다. equal은 자료 컴파일러에서 generic class의 가상함수로 정의되어 있으므로 x는 실제 타입으로 실행되지만 y는 여전히 XSort 타입으로 인식된다. generic 타입으로 저장된 값이 동치 검사를 위해서 실제 타입으로 변환된다. 이 방법은 자료 코드 생성기를 수정하지 않고 해결하는 최선의 방법이다.

5. ForTIA 구현 및 사용 예

5.1 구현

ForTIA의 구조를 자료흐름도로 표현해 보면 그림 5와 같다. 전체 ForTIA 도구는 관리 프로세스를 중심으로 단위 도구별 처리를 위한 4개의 프로세스 (구문지향 편집, 시뮬레이션, 코드생성, 자료평가)와 전처리를 위한 타입검사 프로세스, 단순편집 프로세스 및 사용자 인터페이스를 위한 GUI 프로세스로 구성된다. 관리 프로세스는 자료흐름도에 나타난 것처럼 GUI를 포함한 7개의 프로세스를 서로 연결해주는 역할을 하며 통합도구로서 전체적인 관리 및 프로세스 제어를 위해 필요한 모든 기능들을 수행한다. 특히, 단위 프로세스에서 GUI를 통해 사용자와 상호 작용할 때에는 논리적으로 반드시 관리 프로세스를 거치도록 하였다. 관리 프로세스는 그 역할 수행을 위해 ForTIA 환경정보를 저장 및 참조한다.

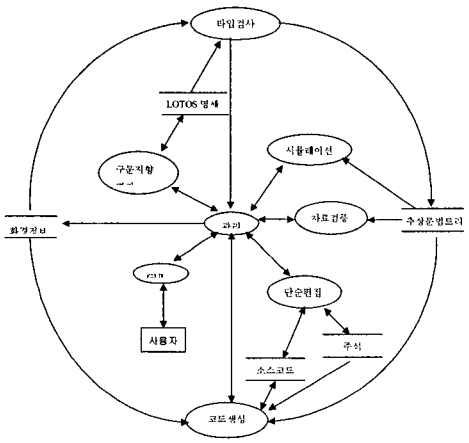


그림 5 자료흐름도를 이용한 ForTIA설계

ForTIA의 설계 및 구현에서 가장 중요한 특징은 소프트웨어 아키텍처(software architecture)라 불리는 시스템 구조화 기법을 사용한 점이다. 그림 6은 이에 따른 ForTIA의 기본 구조를 보여 주고 있다. 도구의 핵심 기능은 GUI로부터 분리되어 층 구조를 이루는데, GUI 층, 관리 층, 핵심 층으로 구성된다. 가장 하위 층인 핵심 층은 코드 생성과 시뮬레이션을 위한 기본적인 기능을 제공한다. 대표적으로 명세의 구문 및 타입 검사, 중간 코드 생성, C++ 변환, 시뮬레이션 엔진 등의 기능을 제공한다. 대부분 외부 실행 파일 형태의 독립 모듈로 구현되며 TOPO와 LOLA의 소스 코드가 부분적으로 재사용되었다. 관리 층은 도구의 조정자 역할을 하며 핵심 층의 모듈간 혹은 상위 층과의 통신 채널을 제공한다. 상위 층에서 요구하는 서비스를 제공하기 위하여 핵심 층의 모듈들을 동적으로 배치하고 연결한다. 대표적인 기능은 자료 및 행위 부분 C++ 코드 생성, 프로토타입 생성과 실행, 명세 시뮬레이션이다. 가장 상위 층인 GUI 층은 편리하고 사용자 지향적인 GUI를 제공하며 시각적 시뮬레이션 환경을 지원한다. 또한 파일 검색, 소스 코드 보기, 환경 설정 등과 같은 부가적인 기능도 수행한다. 각 층은 파이프라인 구조(pipeline architecture), 객체지향 구조(object-oriented organizational style), 사건기반 암시적 호출방식(event-based implicit invocation style), 저장소 방식(repository style) 등과 같은 다양한 아키텍쳐 스타일이 사용되었다.

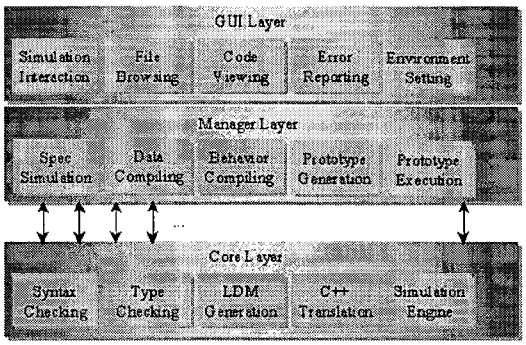


그림 6 구조화 기법을 이용한 ForTIA설계

ForTIA는 C++로 구현되었으며, LEX/YACC과 같은 도구가 사용되었다. 현재 마이크로소프트 Win95/98에서 운용되며, 기존의 LOTOS 도구와 달리 사용자 지향적인 편리한 GUI를 제공한다[16]. 앞에서 언급한 것과 같이 TOPO와 LOLA의 코드가 부분적으로 재사용 되었다. 예를 들면, 명세의 타입 검사와 중간 코드 생성과 같은 전처리 모듈은 TOPO의 소스 코드가 사용되었으며, 시뮬레이션 엔진의 구현을 위하여 LOLA의 C 코드

로부터 C++ 래퍼(wrapper) 클래스를 작성하였다.

개발의 많은 비중을 전체 도구의 구조 설계와 단위 모듈의 통합이 차지하였는데, 적절한 소프트웨어 아키텍쳐의 사용과 정형 방법의 적용으로 많은 시간과 노력을 절약할 수 있었다. 정형화의 가장 큰 목적은 코딩 작업에 앞서 설계한 시스템이 제대로 동작한다는 것과 이에 대한 확신을 높이기 위해서였다. TOPO와 LOLA의 기존 코드에 대한 이해를 높이고 재사용의 편의를 위하여 프로그램 모듈의 외부 인터페이스와 행위에 관한 정형화가 부분적으로 이루어졌다. 정형 방법의 효과적인 적용을 위하여 LOTOS, Z, Larch와 같은 여러 정형 명세 언어가 복합적으로 사용되었다.

5.2 ForTIA 사용 예

5.2.1 개발 절차

그림 7은 ForTIA를 사용한 시스템 개발 절차도이다. 먼저 명세를 편집하고 타입을 검사한 후 시뮬레이션을 통하여 명세의 동적 의미를 확인한다. 자료평가를 통해 올바른 자료형이 정의되었는지 별도로 확인할 수도 있다. 확인 결과를 바탕으로 명세를 수정 및 편집하고 다시 확인하는 작업을 반복하여 명세를 정제시킨다. 어느 정도 정제된 명세로부터 C++ 프로그램을 생성하고 프로토타입으로 실행시키며, 필요에 따라 생성된 프로그램을 조율하고 수정하여 테스트한다. 이와 같은 과정을 거쳐 점진적으로 완전한 시스템을 개발한다.

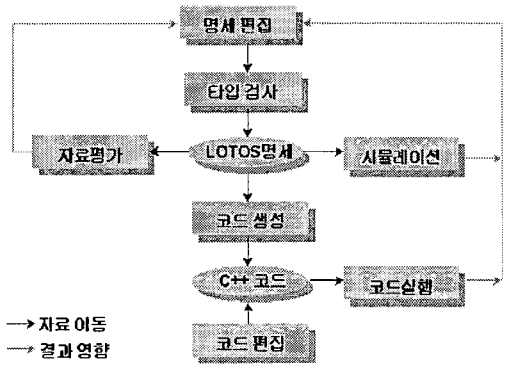
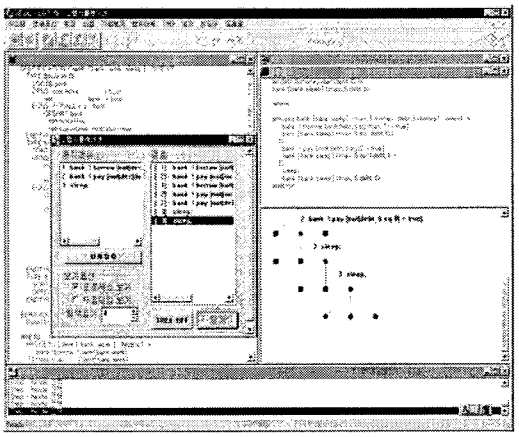


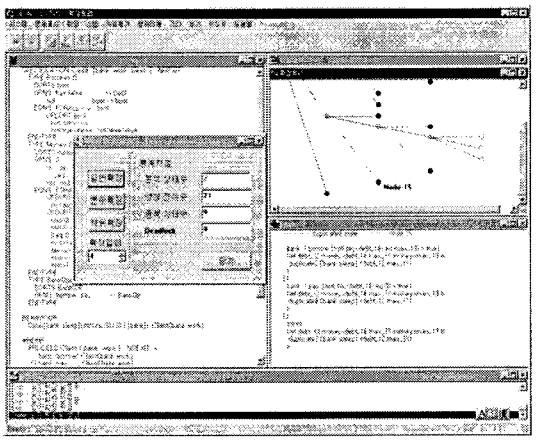
그림 7 ForTIA 사용도

5.2.2 대표 화면

본 논문에서는 앞 절에서 소개한 개발 절차 중 ForTIA의 핵심 기능인 시뮬레이션과 코드생성을 화면 중심으로 소개한다. 자세한 사용 예는 ForTIA 튜토리얼[17]을 참고하기 바란다. 다음의 그림 8은 ForTIA의 시뮬레이션이 진행되는 화면이다.



(a) 단계 시뮬레이션 실행 화면



(b) 명세 확장 실행 화면

그림 8 ForTIA 시뮬레이션 화면

그림 8의 (a)는 단계 시뮬레이션의 진행 화면으로서 선택된 경로에 의한 시뮬레이션 트리가 화면의 오른쪽 아래에 나타나 있고 이 트리 상에서 바로 입력 처리가 가능하게 되어있다. 즉, 트리의 잎 노드들 중 하나를 더블 클릭함으로써 다음 상태로의 전이를 유발시키고, 전이가 일어난 후 바로 상위의 노드를 더블 클릭함으로써 전이를 취소시킬 수도 있다. 한편, 왼쪽 중앙에는 선택 가능한 동작의 종류와 지금까지의 경로를 텍스트 형식으로 보여주는 다이얼로그가 위치한다.

그림 8의 (b)는 명세 확장의 진행 화면인데 화면 왼쪽의 바탕에 나타나있는 전체 명세 중에서 관심 있는 부분을 먼저 선택하여 부분적인 명세를 확장할 수 있고, 그 결과가 오른쪽 상단의 확장 트리로 표현된다. 한편, 왼쪽 중앙에는 확장의 종류와 깊이를 받아들이고 확장의 결과 계산된 분석 상태 수, 생성 전이 수, 교착점 등이 표현된 다이얼로그가 위치한다.

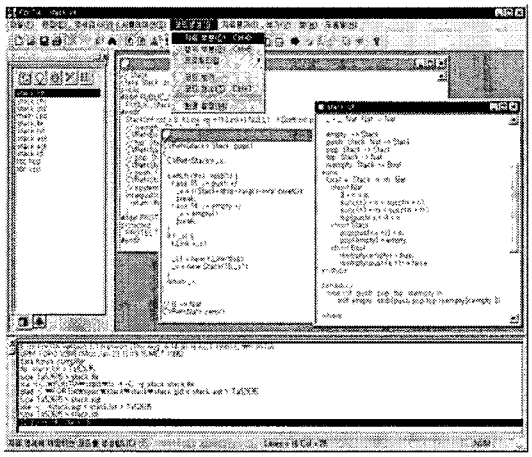


그림 9 자료 코드 생성 화면

그림 9는 ForTIA의 자료 코드 생성 화면이다. 코드 생성은 여러 단계를 거쳐서 일어난다. 일반적으로 구문 검사, 타입 검사, 주석 처리, 중간코드 생성, C++ 변환의 과정을 통하여 C++ 프로그램이 생성된다. 경우에 따라서는 몇몇 단계가 생략될 수 있다. 예를 들면, 타입 검사 메뉴에 의해 이미 해당 명세의 구문 검사 및 타입 검사가 완료되었으면 이 단계가 생략된다. 코드 생성 과정은 ForTIA의 메시지 창에 표시된다. 만약 코드 생성 과정에서 오류가 발생하면 이 또한 메시지 창에 표시된다. 오류 발생 부분은 해당 메시지를 더블 클릭하여 찾을 수 있다. 코드 생성이 완료되면 생성된 C++ 파일과 각종 중간 파일들이 ForTIA 작업창에 나타난다.

6. 결 론

대용량, 고신뢰도, 복잡성, 분산 환경이라는 특징을 지닌 정보 통신용 소프트웨어 개발에서 최소의 비용으로 최고의 품질을 지닌 제품을 생산하기 위해서는 향상된 정형 명세 기법과 이러한 기법을 지원하는 도구가 필수적이다. ForTIA는 이를 위해 산업계의 실사용을 목적으로 개발된 LOTOS지원 도구이다[18].

ForTIA는 명세 시뮬레이션[19][11]과 코드 생성[20]에 중점을 두고 개발되었으며, 주요 특징은 시스템의 점진적 개발을 지원하며 윈도우 기반의 편리한 사용자 인터페이스를 제공한다는 것이다. ForTIA의 시뮬레이션 기능은 기존의 LOTOS 시뮬레이션 도구에서 찾아보기 힘든 시각적 시뮬레이션 트리를 이용한 사용자 인터페이스를 정의하고 구현하였다. 사용자는 향상된 상호작용 방법을 통해 보다 직관적이고 편리하게 명세의 시뮬레이션 작업을 수행할 수 있으며 사용상의 어려움을 가장 큰 난제로 삼는 정형 기법 분야에서 쉽고 빠른 명세 확인 기법이라는 새로운 가능성을 제시한다.

ForTIA의 코드 생성 기능은 기존 도구가 C, 함수 언어, 논리 언어를 지원하는 것에 반하여 C++코드의 생성을 지원한다. 산업계에서 표준으로 자리 잡아 가고 있는 객체지향 언어를 채택함으로써 여러 가지 이점이 있는데 특히, 수작업 코드와의 연계 및 자동 생성 코드 대치를 위해 상속과 같은 객체지향 개념을 활용할 수 있다. C++코드에 적합한 목표 코드 형식을 정의했으며, 또한 다른 모듈에서 재사용되거나 필요에 따라 수정되어야 하므로 생성된 코드의 간결함을 위해서 노력하였다.

ForTIA는 시스템 구조화 기법을 이용하여 3계층(핵심 층, 관리 층, GUI층) 구조로 개발되었으며 기존 도구인 LOLA 및 TOPO의 소스 코드가 부분적으로 재사용되었다. 또한 ForTIA의 설계 및 구현에서는 적절한 소프트웨어 아키텍처가 사용되었고 객체지향 기법과 정형기법이 단계별로 적용되었다.

참 고 문 헌

[1] S. L. Pfleeger, "Investigating the Influence of Formal Methods," Computer, pp. 33-43, Feb. 1997.

[2] ISO. Information Processing Systems - Open Systems Interconnection - LOTOS: A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, International Standard 8807. ISO, 1988.

[3] H. Garavel, M. Jorgensen, R. Mateescu, C. Pecheur, M. Sighireanu, and B. Vivien, "CADP'97 - Status, Applications, and Perspectives," In I. Lovrek, editor, Proceedings of the 2nd COST 247 International Workshop on Applied Formal Methods in System Design, Zagreb, Croatia, June 1997.

[4] E. H. Eertink, SMILE user manual (release 4.0), Tele-Informaics and Open Systems Group, 1993.

[5] B. Ghribi and L. Logrippo, A Validation Environment for LOTOS, Protocol Specification, Testing and Verification, XIII, Elsevier Science publishers B. V., pp. 93-108, 1993.

[6] D. Larrabeiti, S. Pavon and G. Rayvay, LOLA user manual (version 3R6), Department De Ingenieria Telematica, 1995.
[7] M. Caneve and E. Salvatori, editors, LITE User Manual, LOTOSPhere Consortium, 1992.
[8] J. A. Manas and T. de Miguel, "From LOTOS to C," In K.J. Turner, editor, Proceedings of the 1st International Conference on Formal Description Techniques (FORTE '88), University of Stirling, Scotland, pp. 247-261, North-Holland, September 1988.
[9] H. Garavel, "Compilation of LOTOS abstract data types," In S.T. Vuong, editor, Proceedings of the 2nd International Conference on Formal Description Technique (FORTE '89), Vancouver, Canada, pp. 147-162, North-Holland, December 1989.
[10] E. H. Eertink, Simulation Techniques for the Validation of LOTOS Specifications, University of Twente, The Netherlands, 1994.
[11] S. Cho, K. Lee, Y. Oh and H. Kim, "An Interactive LOTOS Toolset for Users in Industry," Proceedings of International Computer Symposium '98, Tainan, Taiwan, December 17-19, 1998.
[12] J. de Meer, R. Roth, and S. Vuong. "Introduction to Algebraic Specification Based on the Language ACT ONE". Computer Networks and ISDN Systems, 23(5), pp. 363-392, February 1992.
[13] Y. Cheon, K. Kim and Y. Oh, "Automatic Generation of C++ Programs from LOTOS Data Type Specifications," In Roger Lee, editor, Proceedings of the IASTED Conference on Software Engineering, Las Vegas, Nevada, USA, October 28-31, 1998, pp. 103-106. ACTA Press, 1998.
[14] J. A. Manas, J. Salvachua, and T. de Miguel, "The TOPO implementation of the LOTOS multiway Rendezvous". Department Ingenieria Telematica Technical University of Madrid, 15 Jan. 1991.
[15] E. E. Dubuis, "Compiling the Behaviour Part of LOTOS," Ph.D thesis, Swiss federal institute of technology ZURICH, 1993.
[16] 조수선 외 4인, ForTIA 사용자 설명서, 기술문서 8MG4420-TDP-1017, 한국전자통신연구원, 1998년 11월.
[17] 천윤식 외 3인, ForTIA 튜토리얼-LOTOS 기반의 정형기법 지원 도구, 기술문서 TM98-3400-36, 한국전자통신연구원, 1998년 6월.
[18] 전진옥 외, 정보통신용 시스템 개발방법론 구축에 관한 연구(III), 연구보고서, 정보통신부, 1998.
[19] 조수선, 이광용, 오영배, "LOTOS 정형명세의 시각적 분석 지원 도구", 정보처리논문지, 제5권, 제 12호, pp. 3117-3126, 1998.
[20] 김철홍, 천윤식, 김강호, "LOTOS명세로부터 C++소스코드의 자동 생성", 정보처리논문지, 제5권, 제 12호, pp. 3138-3150, 1998.

조 수 선
1987년 서울대학교 계산통계학과 졸업(학사). 1989년 서울대학교 대학원 계산통계학과 졸업(이학석사). 1989년 ~ 1994 (주)웅진미디어 CBE개발부 연구원. 1994년 ~ 현재 한국전자통신연구원 컴퓨터·소프트웨어기술연구소 선임연구원. 관심분야는 정형기법, 소프트웨어공학, 실시간OS

천 윤 식
1989년 고려대학교 전산과학과 졸업(학사). 1991년 Iowa 주립대학교 전산과학과 졸업(공학석사). 1992년 ~ 현재 Iowa 주립대학교 전산과학과(박사과정). 1995년 ~ 현재 한국전자통신연구원 컴퓨터·소프트웨어기술연구소 선임연구원. 관심분야는 객체지향 프로그래밍, 정형기법, 소프트웨어공학

오 영 배
1982년 고려대학교 기계공학과 졸업(학사). 1995년 인하대학교 전자계산공학과(공학석사). 1999년 정보처리기술사. 1995년 ~ 1997년 미국캘리포니아대학(UCI) 객원연구원. 1983년 ~ 2000년한국전자통신연구원 컴퓨터·소프트웨어기술연구소 책임연구원, 영역기반소프트웨어재사용기술 과제 책임자. 2000년 ~ 현재 수원여대 컴퓨터응용학부 교수. 관심분야는 컴포넌트기반소프트웨어공학, 소프트웨어재사용기술, 정형기법, 전자상거래

정 연 대
1978년 서강대학교 수학과 졸업(학사). 1988년 서강대학교 경영대학원 졸업(석사). 1998년 ~ 1999년 USC(University of Southern California) 초빙연구원. 1978년 ~ 1983년 한국과학기술연구소 부설 전산개발센타 연구원. 1983년 ~ 1990년 한국과학기술연구원 부설 시스템공학연구소 선임연구원. 1990년 ~ 1998년 한국전자통신연구원 컴퓨터·소프트웨어기술연구소 개발자동화 연구실장. 1998년 ~ 현재 한국전자통신연구원 컴퓨터·소프트웨어기술연구소 책임연구원. 관심분야는 객체지향 데이타베이스, 객체지향 방법론, 정형기법, CBSD(Component-Based Software Development)