□기술해설□

시간이 긴요한 실시간 시스템의 요구 명세 및 분석

숭실대학교 이광용* · 정기원**

목 차

1. 소 개

실시간 시스템은 그 시스템에서 요구된 기능을 주어진 시간제약 범위 안에서 모두 수행하되 시스템의 외부 환경과의 상호작용이 올바르고 믿을만하게 이루어지도록 제어하는 시스템을 말한다[1-4]. 따라서, 외부세계와의 상당히 밀접한 관계를 맺고 있는 실시간 시스템은 하드웨어 장치들과 밀접하게 관련하며, 하드웨어 장치들의 수행능력을 최적화 시키고 정해진 요구기능을 만족해야 하는 등 실시간 시스템을 설계하고 구현하는 문제는 기존의 순차적 시스템과 비교하여 상대적으로 어렵고 복잡하다.

이러한 시스템을 구축하기에 앞서 시스템의 요구기능과 시간제약 사항에 대한 분석이 정확히 이루어지지 않으면 후에 이를 수정하는데는 개발 이상의 비용과 돌이킬 수 없는 결과를 발생시킬 수 있다. 따라서, 실시간 시스템을 모형화하는 방법이 필요하며 그 첫번째 단계로 구축하려는 시스템을 특정 모형화 방법을 이용하여 명세화하고, 설계하고, 그것으로 부터 정밀하게 분석하는 과정이 필요하다. 다시 말하면, 실시간 시스템을 개발하기 위해서는 효율적이면서 정밀한 분석방법 즉, 정형적인 방법이 필요하다. 이 정형적인 방법을 사용하게 됨으로써 얻을 수 있는 이점은 요구사항이나 설계사항을 정형화하게 됨으로써 모호함이나 상반된 것을 발견할 수 있고, 자동 또는 반자동으로 시스템을 개발할 수 있으며, 수학적인 방법에 의해 시스템을 검증하게 된다는 것이다.

현재, 소프트웨어 개발 단계의 초기에 요구사항을 검증·확인 하기 위한 효율적이면서, 정밀한 형식명세(formal specification) 방법은 소수에 불과하며, 대부분의 소프트웨어 명세 검증·확인 기술은 프로토타입이 개발된 후, 혹은 실행가능한 명세서를 작성한 후, 또는 개발이 완료되어서 시험단계에 도달해서야만 사용될 수 있는 수준에 불과하다. 실제로 실시간 소프트웨

*학생회원
**종신회원

어를 개발하면서 각 단계별로 검증·확인하기 위한 형식명세 방법들을 활용하고 있지 못한 이유는 현재까지 연구된 이 방법들은 형식언어(formal language) 수준으로 다루어졌기 때문에 분석가가 이해해서 사용하기가 어렵고, 분석가가 이해했더라도 실제 사용자와의 요구사항 확인은 사용자가 이해하기 어려운 언어로 쓰여져 있기 때문에 불가능하다는데 있다.

본 고에서는 실시간 소프트웨어를 개발하기 위해 이용되고 있는 형식명세 기법들의 특징 및 장 단점 측면에서 조사하고, 소프트웨어 개발방법론으로의 형식명세기법의 접목에 대해 살펴보기로 하겠다.

2. 구조적 기법의 요구명세

2.1 구조적 방법(Structured Method)

실시간 시스템을 위한 구조적 방법[3-7]은 10여년전에 산업에서 이용된 시스템 분석 방법들에서 유래된다. 이 방법들에서는 시스템 요구사항의 구조적인 면들을 표현할 수 있도록 하고 있다. 특정 시스템 요구사항에 대해 어떤 문제(what)를 해결해야 하는지, 그리고 그 문제는 어떻게(how) 구조화 되있어야 하는지를 명세화 한다. 자료흐름도는 이러한 것을 명세화하는 구조적 기법의 가장 대표적인 도구이다. 이 도표로 시스템의 기능을 계층적으로 분해하여 이들 사이의 자료 및 제어흐름을 표현한다. 또한, 상태도를 이용해서 시스템의 동적인 행위를 묘사하고, 자료사전에 모든 입출력 사건 및 자료·제어 흐름 등을 기록하고 사건들의 흐름을 시간 및 출현 빈도수에 따라 테이블 형태로 표현하고 있다. 그러나, 시간 요구사항에 대해서는 특별히 잘 표현하고 있다고는 보여지지 않는다.

현재, 이 구조적 방법들은 실제 산업계에서 성공적으로 활용되고 있다. 그러나, 이 방법에는 어떤 정형적인 시맨틱스(semantics)가 없기 때문에 결과 산출물을 이용해서 시뮬레이션(simulation)을 할 수가 없으며, 또한 어떤 코드 형태(Ada 혹은 C)로 자동으로 변환할 수가 없다. 그렇기 때문에 비결정적인 시스템들의 행위에 대한 적절한 모형화 방법은 아니다. 따라서, 실시간 시스템의 설계에 사용하기 위해서는 기능의 확장과 정형성을 더 추가할 필요가 있다. 이러한 점에 유의하여 1980년대 후반에 Gomaa 방법[6] 및 Nielsen & Shumate 방법[7]에서는 이러한 점에 유의하여 기능 확장을 시도하였으나, 아직 이들도 정형적인 명세 수준은 아니다.

2.2 STATECHART

STATECHART[8, 20]는 구조적 방법을 개선시켰는데, 구조적 방법의 상태도를 정형적으로 모형화할 수 있도록 하였다. 시스템의 복잡성이 증가함에 따라 그 시스템에 해당하는 유한상태기계(finite state machine)에 상태수는 지수 함수적으로 증가하게 된다. 이것은 시스템을 이해하기 곤란한 크고, 비구조화된 다이어그램을 초래하게 된다. STATECHART는 모듈성, 계층성, 및 구조적인 기술이 가능하도록 유한상태기계 표기법을 계층성(hierarchy), 병행성(concurrency), 브로드캐스트(broadcast) 통신의 세 가지 측면에서 확장하였으며, 다른 특징으로 임의상태(default states), 전역 타이밍(global timing), 시간 전이(timed transition)를 추가하였다.

먼저, STATECHART에서는 상태들을 묶어서(clustering) 공통된 상태전이가 가능하도록 표현하여 상태들의 계층성을 표현한다. STATECHART에서 상태는 둥근 사각형으로 표현하고 상태전이는 상태 혹은 상태집합의 둥근 사각형 경계로 부터 시작하여 나가는 화살표로 그린다.

예를 들면, 그림 1에서 보는 것처럼, 사건 b는 상태 A 혹은 C로 부터 공통적으로 발생되므로 A와 C 상태를 묶어서(clustering) 공통의 상태 D를 생성하고, 두개의 b 화살표 대신에 하나의 b 화살표로 D로 부터 B로의 상태전이를 표시한다. 여기서, D의 상태는 어느 한순간에 A와 C 상태 중 어느 하나로(XOR) 결정된다. 즉, D 상태에 시스템이 존재한다는 것은 어느 한순간에는 A 혹은 C 상태 중 어느 하나에만 있다는 것이지, A와 C 두 상태에 동시에 존재할 수 없다는 것이다. 그러므로, D는 A와 C의 추상화(abstraction)이고, 거꾸로 말하면, D는 A와

C로 세분화(refinement)된 상태임을 의미하는 것이다.

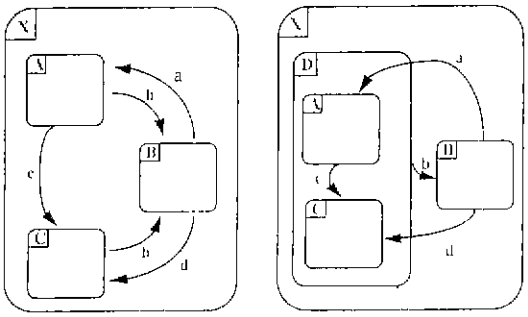


그림 1 Or-분해

그림 2에서 보는 것과 같이 병행성은 상태들의 And-분해에 의해 표현된다. 즉, X는 And 구성품인 A와 P로 구성되어 있으며, X의 상태는 A의 상태인 B와 C 그리고, P의 상태인 Q, R, S들의 And 조합에 의해 표시된다. STATECHART의 And 분해를 이용해서 상태전이가 서로 완전히 독립된 형태로 수행되는 것을 표시할 수도 있으며, 서로 동기를 맞추어서(synchronously) 수행되는 경우도 표현할 수 있다. 예를 들어서, 그림 2에서 X가 (B, Q) 상태에 있을 때 사건 c가 발생하게 되면 동시에 B는 C 상태로 Q는 R상태로 상태전이가 이루어진다. 그렇지만, X의 상태가 (B, S) 상태에 있을 경우 사건 c의 발생은 A 구성품의 상태만 B에서 C로 변경이 되고, P 구성품의 상태는 아무런 영향을 받지 않는다. 이와 같이 STATECHART에서 병행성은 상태들의 And-분해에 의해 표현된다.

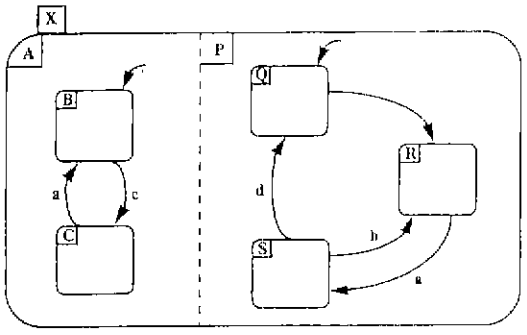


그림 2 And-분해

한편, STATECHART는 실시간 시스템을 명세화하기 위해 이용되므로 동기화나 타임아웃(timeout)과 같은 시간정보(timing information)를 제공하기 위한 전역시계(global clock)가 필요하다. 그림 3에서 보는 바와 같이 STATECHART에서는 시간전이 조건을 붙여서 시간정보를 명세화한다. 그림 3의 의미는 상태 B에서 정확히 10초 내에 사건 a가 수행되어야 하는 반면, 사건 b의 경우에는 5에서 10초 사이에 상태 C에서 상태 B로 상태전이가 발생해야 하는 사건을 표시한 것이다.

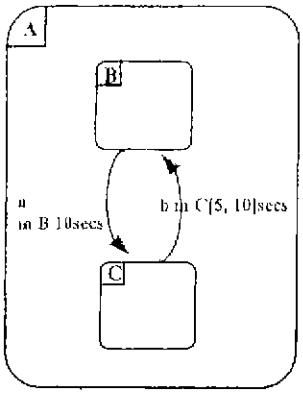


그림 3 시간표현

또한, 모든 유한상태기계(finite-state machine)는 임의상태(default state)로 시작상태를 갖고 있는데, STATECHART에서는 그 시작상태를 구별하기 위해 소스 상태(source state)가 없는 화살표로써 표시한다. 그림 2를 보면 (B, Q) 상태는 임의상태이다.

이러한 STATECHART는 엄격히 정의되어 있는 정형적인 시맨틱스(formal semantics)에 기초한다. 이것의 중요한 결과로서 자동화된 시뮬레이션 도구를 이용해서 사용자가 직접 모델을 실행해 볼 수 있도록 허용한다. 또한, 모든 가능한 상태들에 대한 소모적인 검사(exhaustive checking)를 지원한다. 그리고, 완전하지는 않지만 생성된 모델들에 대한 목표 환경에 맞도록 컴파일 되기도 한다.

그러나, 아직은 정형적인 검증을 완전히 지원하지는 못하고 있으며, 충분한 수준까지의 시간

제약에 대해서도 지원이 되지 않고 있다. 실시간 특성을 완벽하게 지원하기 위해서 아직도 STATECHART의 시맨틱스에 대한 연구가 필요하다. 예를 들면, 주기적 타이밍 함수 및 추가적인 상태들의 도입이 없이도 타이밍 예외상황의 명세화 등의 연구가 필요하다.

2.3 페트리네트(Petri Net)

1962년 C. A. Petri에 의해 개발된 페트리네트 이론[9]은 병행성, 비결정성, 사건들 사이의 인과관계를 처리하기 위한 첫번째 정형적인 표현 방법 중의 하나이다. 지금까지 페트리네트 이론은 많은 연구자들에 의해 여러 가지 목적을 가지고서 연구되어 왔으며 다양한 수정 및 확장이 있었다.

페트리네트 구조는 네 가지의 요소로 구성되어 있다. 플레이스(Place)의 집합 P, 트랜지션(Transition)의 집합 T, 입력함수 I, 출력함수 O로 구성되어 있는데, 입력함수와 출력함수는 트랜지션과 플레이스 사이에 발생하는 입력과 출력 관계를 규정하고 있다. 입력함수 I는 주어진 트랜지션 t_j에 입력으로 연결된 플레이스들의 집합을 규정하며 $I(t_j) = \{P_i\}$로 표기한다.

페트리네트는 시스템의 비결정적인 혹은 병행적인 프로세스들을 표현하는 모형화 도구로 개발되었는데, 시스템 내의 프로세스를 페트리네트 그래프로 나타낸다. 만약, 시스템 내의 하나의 프로세스가 조건(condition), 사건(event) 및 이들 사이의 관계를 서술하는 규칙(rule)으로 구성된다고 가정한다면, 페트리네트의 도식적 표현인 페트리네트 그래프(Petri Net graph)에서 이들 조건과 사건은 각각 플레이스(원으로 표현)와 트랜지션(선분으로 표시)으로 나타내고, 이들 사이의 규칙은 플레이스와 트랜지션의 네트워크 형태로 표현한다.

어떤 조건은 각 플레이스에 토큰(token)을 위치시킴으로써 표현하며 플레이스와 트랜지션, 트랜지션과 플레이스의 연결은 방향성 아크(directed arc)로 표현한다. 트랜지션(즉, 사건의 발생)은 자신에게로 입력되는 모든 플레이스가 토큰을 보유하고 있어야 점화(fire)될 수 있다. 트랜지션이 점화되면 자신의 각 입력 플레이스로 부터 토큰을 하나씩 제거하고 각 출력 플레이스에 토큰을 하나씩 첨가한다. 페트리네트 그래프에서 플레이스들에 분산된 토큰의 분포를 마킹(marking)이라 한다.

토큰의 위치와 갯수는 페트리네트가 실행되는 동안에 초기 마킹(initial marking)으로 부터 변하게 되는데 이것은 페트리네트 모델에서의 상태(state)를 변화시키는 것이다. 즉, 트랜지션의 점화에 의해 초기 마킹을 바꾸게 하며 페트리네트 모델의 상태를 변경시키는 것이다. 만일 임의의 시점에서 마킹이 μ인 페트리네트에서 트랜지션 $t_i \in T$가 점화됨으로써 마킹이 μ'로 변경되었을 때 μ'는 μ로 부터 도달가능(reachable)하다고 한다. 따라서, 페트리네트에서 초기 마킹 μ_0로 부터 도달가능한 모든 마킹들의 집합을 생각할 수 있는데 이것을 초기 마킹 μ_0에 대한 도달성 집합(reachability set)이라고 한다. 이 도달성 분석을 통하여 시스템의 도달성(reachability), 생존성(liveness), 제한성(boundness), 보존성(conservation), 안전성(safety), 교착상태(deadlock) 등의 특성을 파악한다.

그러나, 이러한 페트리네트는 프로그램의 자료구조를 표현하지 못하고 있는 점과 합성 오퍼레이터(composite operator)와 같은 것을 본래의 페트리네트 이론으로는 표현할 수 없는 것에 대해 여러 가지 비평이 있으며, 유한상태기계(finite state machine)와 같이 쉽게 이 모델을 프로그램 코드로 변환하기가 쉽지 않은 문제점을 갖고 있다. 또한, 이 보다 더 커다란 문제는 시스템이 커짐에 따라 도달성 그래프의 상태폭주(state explosion)가 발생하는 것으로 시스템을 분석하기에 상당한 어려움을 주고 있다. 그리고, 페트리네트 모델에는 근본적으로 시간개념이 도입되어 있지 않아 시간을 고려해야하는 경우에는 새로운 페트리네트 모델을 만들어야 하는 어려움도 있다. 현재, 시간 페트리네트(Timed Petri Net) 모델이 개발되어 있으며 그 종류도 여러 가지가 있다.

3. 논리 및 대수를 기반으로 한 요구명세

3.1 ESM과 RTTL(Extended State Machine and Real-Time Temporal

Logic)

이 절에서는 상태기계(Sate Machine)와 논리(Logic)를 결합한 요구명세 기법을 소개한다[2]. 이 두 가지 기법의 결합은 실시간 시스템을 효율적으로 명세하고 검증할 수 있도록 도와준다. ESM은 기본 상태기계에 시간(time bound) 개념을 더 추가한 것이다. ESM은 장치 프로세스들과 제어 프로세스들 사이를 기술하기 위해 적용되고, RTTL은 ESM에 묘사된 시스템을 검증하는데 적용한다. ESM에서는 하나의 시스템 M을 다음과 같은 형태로 묘사한다.

$$M = M_0 \parallel M_1 \parallel \ldots \parallel M_k \parallel M_{k+1} \parallel \ldots \parallel M_{k+n}$$

여기서, M_0는 전역시계(global clock)이고, M_1 ... M_k는 장치 프로세스들이며, M_{k+1} ... M_{k+n}는 제어기 프로세스들이다.

① ESM 구성품들

ESM은 다음과 같은 요소들로 구성되어 있다.

- 활동변수(activity variable)
- 자료변수(data variable)
- 통신채널의 집합(a set of communication channels)
- 사건의 집합(a set of event labels)

ESM에서 사건은 (A_e, guard, operation, A_s) 형태로 표현되는데, A_e는 exit 활동을, A_s는 source활동을 나타내고, 오퍼레이션(operation)은 assign, send, receive 오퍼레이션으로 구성되고, 가드(guard)는 사건에 대한 부울식을 나타낸다.

그리고, ESM 모델 즉, M_m(m = 1, 2, ..., k + n)에서 제어 프로세스들과 장치 프로세스들은 병행적으로 수행되면서 서로 통신채널을 통해 정보를 교환하게 된다. 예를 들어, M_i 프로세스에서의 사건을 E_i라 하고, M_j 프로세스에서의 사건을 E_j라 할 때, 두 사건 사이의 협력하는 행위(action) 즉, 그 행위에 의한 정보교환은 간단하게 두 사건사이의 할당(assign) 오퍼레이션으로 표시하면 된다. 또한, 두 사건 사이의 send와 receive 오퍼레이션 관계를 이용하여 상태 전이 집합(transition set)을 구성할 수 있다. 상태 전이 집합 중 하나의 전이 는 다음과 같이 정의한다.

$$\tau = (\alpha, e, h, l, u)$$

여기서,

- α는 Ei 사건의 이름,
- e는 전이조건,
- h는 상태변환함수,
- u와 l은 각각 상한시간(upper time)과 하한시간(lower time)

이다.

ESM 모델에서는 실시간 특성(real-time property)을 이 u와 l에 의해 표시한다. 이상과 같은, 활동변수, 자료변수, 사건들, 전이들을 가지고서 ESM 모델 M에서는 상태와 사건의 순서로 구성된 상태공간의 어떤 경로(path) 즉, trajectory를 형성한다. RTTL에서는 ESM 모델들의 이러한 trajectory들을 RTTL 시맨틱스로 표시하여 ESM 모델의 의미를 분석하고 검증한다.

② RTTL

RTTL은 과거의 linear-time temporal logic을 확장한 것이다. RTTL의 기본 오퍼레이터는 until([ ])과 next(⊙)이다. 이 두 오퍼레이터를 합성하여 다음과 같은 여러 가지 다양한 오퍼레이터를 만들어 낼 수 있다.

- eventually(◇) : ◇w는 (true [ ]w)에 대한 생략형으로, "언제라도 w는 어떤 상태에서는 true이다." 임을 의미한다.
- henceforth(□) : □w는](◇(]w))에 대한 생략형으로, "이 후 계속해서 w는 모든 상태에서 true이다." 임을 의미한다.
- unless(∪)
- preceeds(P) : $w_1 P w_2$는](( ]w_1)[ ]w_2)의 생략형으로 "만약 w_2가 발생한다면, w_1은 w_2

를 선행해서 수행되었어야 한다." 임을 의미한다.

- previous
- since,

등등이 있다.

RTTL에서는 활동변수(activity variable), 자료변수(data variable), 및 다음사건변수(next event variable) n을 이용해서 한 상태에서 다음 상태로의 시스템 상태 변경을 표현한다. 이것은 1차 논리 상태공식(first order state formula)이 된다. 그러나, 이 상태공식에는 시간에 관련된 오퍼레이터가 존재하지 않기 때문에 이 RTTL의 상태공식에다 ESM의 전이 τ를 추가하여 다음과 같은 시간표현 공식을 생성한다.

$$(n = \tau \wedge \psi_h) \rightarrow (e \wedge \odot_\psi)$$

여기서, τ는 ESM의 전이(transition)를 의미하고, ψ는 RTTL의 상태공식이고, e는 enabling 조건, h는 변환함수를 의미한다.

RTTL에서는 invariant 규칙, delay 규칙, real-time liveness 규칙과 같은 중요한 증명규칙들을 갖고 있다. RTTL에서는 이러한 증명규칙들을 이용하여 시스템이 만족해야하는 스케줄링 제약사항, 사건들의 순서제약들을 검증한다.

다음은 RTTL로 표현된 몇 가지 예들이다.

① $(w_1 \wedge (t = T)) \rightarrow \Diamond(w_2 \wedge (t \leq T+5))$.

의미 : 현재, w_1이 true이고, 현재시각(T)을 t로 읽었다면, T+5 시간 내에 w_2가 true이어야 한다.

② $w_1 \rightarrow w_2 P w_3$.

의미 : 현재, w_1이 true이면, w_3는 w_2 다음에 발생해야 한다.

③ $\Box \Diamond(n = tick)$.

의미 : 앞으로 시계는 tick을 자주 발생시킨다.

이와 같이, RTTL은 실시간 시스템의 시간에 관련한 요구사항을 잘 표현할 수 있는 표현법을 제시하고 있다. 그러나, 이 방법의 결정적인 문제는 모델점검(Model-checking)시 병행 프로세스들의 증가에 따른 상태폭주 문제로 시스템 전체를 대상으로 검증하기가 곤란하다는 것이다. 그래서, 이 방법은 실시간 시스템의 전체가 아닌 핵심부분 즉, 시간이 중요한 부분의 검증에 이용되고 있는 형편이다.

3.2 CSP(Communicating Sequential Process)

Process Algebra를 이용한 방법이다. CSP [10]는 병렬 및 분산화된 계산 환경을 위해 장치 혹은 프로세스로부터의 입/출력(Input/Output) 및 병행성(concurrency)등을 표현할 수 있는 언어이다. 병렬 및 분산 환경을 잘 묘사하기 위해서는 하나의 machine내에 존재하는 병행 프로세스들의 통신 및 동기화를 효과적으로 표현할 수 있어야 한다.

CSP에서 제시하는 기본적인 표현은 다음과 같은 6가지 사항들로부터 근거한다.

① CSP 방법에서는 Dijkstra의 "guarded command"와 "parbegin"에 근거한 병행성을 이용한다.

② 병행 프로세스들 사이의 통신을 표현하기 위해 간단한 형태의 I/O command를 소개하고 있다.

③ 프로세스들 사이의 통신은 첫번째 프로세스에서는 출력(output)을 위한 목적지로 다른 프로세스를 명명하고, 동시에 다른 프로세스에는 입력을 위한 소스(source)로 그 첫번째 프로세스를 명명하는 방식으로 이루어진다.

④ 입력 명령어는 가드 명령어(guarded command)로 나타날 수도 있다.

⑤ 반복 명령어(repetive command)는 입력 가드(input guard)를 가질 수 있다.

⑥ 단순한 패턴 매칭 기법을 이용하여 입력 메시지를 구분하고 프로세스 구성품(component)들은 액세스(access)하기 위해 이용된다.

CSP에서 위와 같은 기본사항들이 다음에서 보는 것과 같은 표기법으로 나타나고 있다.

① 병행성 및 입/출력의 표현

병행성은 ‖로 표현하는데 예를 들어 [cardreader?cardimage ‖ lineprinter!lineimage]는 두개의 프로세스 즉, cardreader?cardimage와

lineprinter!lineimage가 병행적으로 수행됨은 ‖로 표현한다. 여기서 ?는 cardreader장치로 cardimage를 입력하라는 표현이고, !는 lineimage를 lineprinter로 출력하라는 표현이다. 또한, 동일 기능을 가진 n개의 프로세스를 표현하기 위해서 X(::1..n)::CL로 표현하는데 이것은 $X(1)::CL_1 \| X(2)::CL2 \| \cdots \| X(n)::CL_n$을 의미하는 것이다.

② Guarded 명령어 대안(alternative) 및 반복(repetive)의 표현

CSP에서 guarded 명령어는 G→CL 형태로 표현하며 그 의미는 G가 만족(true)하게 되면 CL을 수행할 수 있음을 의미한다. 이러한 guarded command들 중에 가드(guard) 부분을 만족하는 것은 수행시키는 대안의 표현 예는 $G_1 \rightarrow CL_1 \square G_2 \rightarrow CL_2 \square \cdots \square G_n \rightarrow CL_n$이며, (i:1..n)G→CL로도 표현할 수 있다. 예를 들어, 만일 x가 y보다 크거나 같으면 m에 x를 할당하고 y가 x보다 크거나 같으면 m에 y를 할당하는 과정은 [x≥y→m:=x □ y≥x→m:=y]와 같이 표현한다. 한편, 반복의 표현은 예로 ⁎[c:character; west?c→east!c]에서처럼 ⁎를 붙여서 문자 c를 west로 읽어서 east로 내보내는 것을 west혹은 east 프로세스가 끝날 때까지 반복함을 표현한다. 다른 예로, i:=0; ⁎[i<size; context(i)≠n→i:=i+1]는 i=0, 1, …에 대한 i≥size 혹은 n의 값을 찾을 때까지 반복됨을 표현한 것이다.

CSP기법에서는 병렬 및 병행 처리가 가능하나 주어진 시간 내에 수행을 완료하기 위해서는 시작해야 하는 가장 늦은 시간을 계산하기가 어렵고, 공통 변수를 통해서만 통신을 해야하며, 독립 행위(action)들로 서로 방해를 받지 않고 수행되도록 제거해야하는 문제점이 있다. 구체적으로 실시간 시스템을 개발하기 위해 적합한 명세기법이 되기 위해서는 실행시간 산정, 표현 및 행위(action)들간의 직접적인 통신을 표현할 수 있어야 한다.

3.3 RTL(Real-Time Logic)

RTL[2, 11, 12]은 사건과 사건의 발생 시간을 표현하고 추론할 수 있는 형식언어(formal language)이다. 사건의 발생 시간(timing)과 사건들의 발생순서를 명세화 할 수 있도록 하기 위해 고안되었다. RTL에서는 사건의 발생 출현(occurrence)에 대해서 추론한다. 사건은 다음과 같이 4가지 형태로 구분할 수 있다.

① 외부 사건
② 시작 사건
③ 정지 사건
④ 전이 사건

RTL에서 이러한 사건들의 이름은 대문자 알파벳으로 명명되며, 외부사건에 대해서는 오메가(Ω) 문자 뒤에 사건 이름을 표시하고, 어떤 행위의 시작은 ↑A와 같은 형태로, 행위의 종말은 ↓A와 같은 형태로 표현하여 나타낸다. 예를 들어 ΩBUTTON1은 "BUTTON1 누름"이라는 외부 사건을 표현한 것이며, ↑SAMPLE과 ↓SAMPLE는 각각 SAMPLE 사건의 시작(start)과 종결(stop)을 의미하는 것이다. 한편 특정한 기능이 발생한 시점(timing)을 표현하기 위해 RTL에서는 @문자를 다음의 정의와 같은 형태로 표현한다.

@(e, i) ≡ 사건 e가 i번째 발생했을 때의 시점을 표시하며 e는 시작, 정지, 외부 혹은 전이 사건이며 i는 정수 상수이다.

예를 들어, @(↑SAMPLE, 1)은 SAMPLE 사건이 첫번째 시작될 때의 시점을 의미한다. 이 사건 발생함수 @(e, i)는 RTL에서 가장 중요한 표기법이다. 이 @함수를 가지고서 RTL에서 시스템의 타이밍 명세와 완전성 검사를 수행한다. 또한 RTL 공식에서는 equality/inequality 서술식과 universal 혹은 existential 한정자 및 1차 논리 연결자들(∧, ∨, …)을 위의 사건 표시법과 함께 사용하며 시스템의 시간을 표현하고 추론한다. 구체적으로 외부자극으로부터 정보를 검출(SAMPLE)하여 화면에 표시(DISPLAY)하는 간단한 시스템의 비정형적인 명세를 RTL로 정형화한 예는 다음과 같다.

명세 : 버튼1(BUTTON1)이 눌려 졌을 때,

30시간단위(time unit)내에 SAMPLE 행위가 수행되어야 하고, 이 행위가 수행되는 동안 그 정보는 검출(SAMPLE)되어야 하고, 연속적으로 화면(DISPLAY)으로 전달되어야 한다. 정보를 검출하는 계산 시간은 20시간단위(time unit)이다.

위와 같은 명세를 RTL로 기술하면 다음과 같다.

$\forall x\ @(\Omega BUTTON1, x) \langle = @(\uparrow SAMPLE, x) \wedge$
$@(\downarrow SAMPLE, x) \langle = @(\Omega BUTTON1, x) + 30$
$\forall y\ @(\uparrow SAMPLE, y) + 20 \langle = @(\downarrow SAMPLE, y)$

안정성 명세 : 만약 전달된 정보가 SAMPLE 행위가 끝난 후 10시간단위 내에 화면에 표시된다면, BUTTON1을 눌러 40시간 단위 내에 그 요청된 정보가 화면에 표시됨을 알 수 있다.

이러한 시간 정보를 RTL로 명세화 하면 다음과 같이 쓸 수 있다.

$\forall u\ \forall t\ @(\downarrow SAMPLE, u) \langle = @(\Omega DISPLAY, t) \wedge$
$@(\Omega DISPLAY, t) \langle = @(\downarrow SAMPLE, u) + 10$
$\rightarrow @(\Omega BUTTON1, u) \langle = @(\Omega DISPLAY, t) \wedge$
$@(\Omega DISPLAY, t) \langle = @(\Omega BUTTON, u) + 40$

RTL의 이용에서 중요한 것은 시스템의 안정성 혹은 명세의 정당성을 검증해 보는 것이다. [12]의 방법을 이용해서 정당성을 검증해 보면은 다음과 같다. 먼저, [12]의 방법에서는 RTL을 그래프형태로 표시하는데 그래프의 노드(node)는 어떤 사건의 발생(@(e, i))을 표시하고 아크(arc)는 RTL의 inequality(〈=)를 나타내며 여기에는 방향성 아크(→)로 가중치를 표시하여 나타낸다.

예를 들어 $\forall y : @(\uparrow SAMPLE, y) + 20 \langle = @(\downarrow SAMPLE, y)$는 그림 4와 같이 그래프로 표시할 수 있다.

이 그림 4에서 시작 사건으로부터 정지사건으로 방향성 화살표가 그려지며, 그 화살표에는

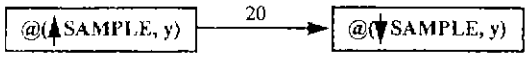


그림 4 RTL 제약 그래프 표현 예

+20의 가중치 값이 주어진다. 이러한 원리로 위의 예제를 그래프 형태로 표시하면 다음 그림 5와 같이 나타낼 수 있다.

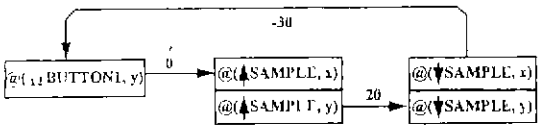
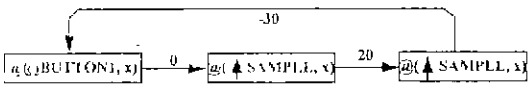
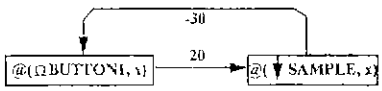


그림 5 제약성 그래프 예

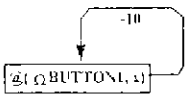
이 그래프를 이용해서 명세의 정당성을 검증하기 위해 그래프 중에 양수 사이클(positive cycle)이 존재하는지를 조사해 보아야 한다. 그림 5예제의 경우에는 그림 6에서 보는 것처럼 순차적으로 노드를 줄여보면 최종적으로 남은 노드의 사이클은 음수 사이클을 보이고 있다. 따라서 이 예제는 그 정당성이 입증이 된 것으로 볼 수 있다.



(a) 예제



(b) @(↑SAMPLE, x) 제거



(c) 최종 상태 · 음수 사이클 검출

그림 6 양수 사이클 검출

그러나, 위의 예제 명세서에서 "button#1을 눌렀을 때 SAMPLE 행위가 15시간 단위 내에 수행되어야 한다."라는 규칙으로 바뀌었다면 그래프의 최종 노드는 +5의 사이클을 보이기

때문에 그 정당성을 입증할 수 없을 것으로 예측 가능하다.

RTL의 사전 발생함수(예 : @(e, i))를 이용하는 모든 비주기적/주기적인 실시간 특성을 잘 표현할 수 있다. 그러나, RTL에서도 마찬가지로 엄격히 제한한 RTL이 아니라면 비결정적인 면이 많다. 그래서 크고 복잡하고 비구조화된 시스템인 경우에는 그 RTL을 이용해서는 검증·확인하기는 힘들다. RTL공식에는 사건들의 반순서 관계(partial corder)를 표시할 수 있기 때문에 상위 수준의 시간 표현에는 유용하게 쓰일 수 있을 것이다.

3.4 기타 명세언어들[13]

Z와 VDM은 집합이론(set theory)과 서술논리(predicate logic)를 기반으로한 명세언어이다. 이 방법을 이용해서 일반적인 대형 시스템을 명세화 하는데 유용하게 쓸 수 있지만 Z와 VDM 모두는 병행성 및 실시간성을 다루기에는 아직 미흡한 면이 많다.

Real-Time Hoare Logic은 단정 스타일(assertional style)의 실시간 시스템 명세 방법이다. 이 기법은 {g}P{r}의 세 가지 구성 요소로써 시스템을 명세하는데 여기서 P는 하나의 프로그램이며, g와 r은 각각 1차 서술논리이다. 이러한 명세에 의해서는 프로그램이 끝나야만 부분적인 정확성(correctness)을 검증 받게 된다. 그러나, 이와 같은 명세 기법으로는 무한히 수행해야 하는 프로세스들과 외부환경과 번번이 통신해야 하는 프로그램들에 대해서는 적용하기가 힘들 것이다. 그래서 이 Hoare triple에다가 세번째 단정(assertion)인 C를 추가시켰다. 즉, C : {g}P{r}형태의 명세 기법을 제시한다. 이 C는 commitment로 프로그램 P(전체 시스템의 부분 프로그램 모듈들)와 관련 있는 외부환경과의 통신을 표현한다. 이것은 종결 혹은 비종결 계산을 만족시키도록 강요하는 단정(assert)인 것이다.

또한, C에는 특별 시간 변수인 time을 허용하고 있다. 예를 들면, 프로그램 P가 통신 채널 C에 대해 어떤 통신도 수행해서는 되지 않아야 함을 이 기법으로 명세화 하면 (∀t :] comm via c at t) : {time = 0} p {true}와 같다. 또 다른 예를 12시간단위 후에 5만큼 증가시키면서 P가 끝나야 하는 경우라면 time < 12 : {x = u ∧ time = 0} P {x = v + 5}와 같이 나타나면 된다.

지금까지 시간이 긴요한 실시간 시스템을 위한 여러 가지 형식명세 기법들에 대해서 논하였다. 각각의 방법들을 나름대로의 정밀성과 표현력을 가지고서 유용하게 쓰일 수 있을 것을 판단되며, 또한 그 한계도 분명히 드러나고 있다. 최근의 이 분야의 연구는 이들 기법들의 장점을 혼합한 혼합형 모델을 만들려는 시도들도 있다. 예를 들면, STATECHART에 RTTL을 접목하려는 시도는 그 한 예이다.

4. 기존 시스템 명세방법의 문제점 및 향후 명세기법

4.1 기존 명세방법의 문제점

지금까지 시스템 명세 및 분석을 위한 여러 가지 형식명세기법들로 구조적기법과 논리와 대수를 근거로한 기법들을 살펴보았다. 이들 기법들의 문제점을 살펴보고 이를 근거로 하여 추후 보완해야될 사항들을 생각해 보면 형식명세기법 연구에 도움이 되리라 여겨진다. 우선, 앞서 살펴본 기존 실시간 시스템 명세 기법들의 문제점은 다섯 가지로 요약할 수 있다[14].

첫번째 문제는 과거의 형식명세에 대한 연구는 대부분 기호적인 표기법 혹은 추론 규칙에 대해서만 관심이 많았지, 방법론적 지원 및 자동화된 도구 지원 차원의 개발에 대해서는 결여되고 있다는 것이다. 현재의 정형적인 방법들은 정교한 표기법적인 구조 및 개념적인 구조를 제시하고 있을 뿐 정형적인 표기법으로 표기하기에 앞서 요구사항을 추출해내고 구조화하기 위한 가이드라인(guideline)은 없다. 따라서, 소프트웨어 개발자는 소프트웨어 요구사항을 발견해내고, 그것을 구조화해내는 방법을 스스로 고안해야만 했다. 즉, 현재의 형식명세기법을 사용하기 위해서는 개발자 스스로 front-end 활동들을 해야만 한다. 또한 도구 지원 부족으로 많은 명세를 가진 비교적 커다란 시스템을 개발, 분석하기에는 어려움이 상당히 많이 있다.

두번째 문제는 현재의 형식명세 표기법 및 개념적 문법은 이산수학 및 기호논리에 익숙해져 있기를 요구하고 있으나 대부분의 실제 소프트웨어 엔지니어, 설계자, 및 구현자는 현재 그러한 것에 익숙해져 있지 못하고 있다는 것이다. 아무리 좋은 명세방법을 개발하더라도 지금 같은 형태의 형식명세방법들은 실제로 적용되기에 힘들다.

세번째 문제는 요구 명세화 마지막 단계에 바라는 데로 만들어진 정형적인 명세라도 사용자가 이해하기가 상당히 어려운 명세를 만들어 낸다는 것이다. 형식명세에 대한 비전문가인 사용자는 개발자들에 비해 더더욱 이해할 수 없는 기호들로 가득차 있을 것이다. 이 상태에서 어떻게 요구사항을 검증·확인해 볼 수 있겠는가?

네번째 문제는 문제해결의 초기단계에 문제정의가 제대로 정의되어 있지 않은 상태에서 형식명세기법의 이용은 산출물의 품질을 보장할 수 없다는 것이며 더 혼란스럽게 한다는 것이다. 그렇기 때문에, 문제를 정의 혹은 정제(refinement)하고 있는 동안에 형식명세기법을 이용하는 것은 이상적인 방법은 아니다.

다섯번째 문제는 이익이 확실히 보장되지 않는 새로운 기술의 이용에는 보수적이며 비자발적이다라는 것에 있다. 즉, 상당히 어렵고 복잡한 형식명세 방법을 사용하면서 어떤 이득을 보장할 수 없는 방법이라면 사용되기가 힘들 것이다.

이상과 같은 현재의 형식명세 방법들의 문제점들을 해결해 나가는 한 가지 방법은 소프트웨어 개발방법론 기술의 장점과 형식명세 기술의 정형성을 통합하는 것이다. 다음 4.2절에서는 소프트웨어 개발방법론의 어떤 특징이 현재의 위와 같은 명세기술의 문제들을 해결할 수 있는지에 대해 살펴보고, 향후의 실시간 시스템의 형식명세 기법이 갖추어야할 조건을 제시하기로 한다.

4.2 향후 명세기법

소프트웨어 개발방법론 기술은 소프트웨어 개발 생산 시에 적용하는 구체적인 기법, 모델, 및 절차를 다루는 기술로서, 소프트웨어 개발 생산성을 향상시키고, 소프트웨어 품질을 높여 소프트웨어 개발을 경제적 및 질적으로 향상시키기 위한 것이다[3]. 앞절에서 제시한 형식명세기법의 대부분의 문제들은 소프트웨어 개발방법론 기술 분야에서는 일찍이 그에 대한 해결에 주안점을 두고 방법론들을 개발하여 해결하고 있다. 예를 들어, OMT[15]나 SA/SD[5] 방법들은 크고, 복잡한 시스템을 효율적으로 개발하기 위해 소프트웨어 생명주기(software life-cycle)의 전 단계를 지원하는 기법, 모델, 및 절차들로 구성된 방법론(methodology)들이다. 따라서, 이러한 방법들을 이용하게 되면 누구나 쉽게 이해할 수 있는 품질이 좋은 소프트웨어를 쉽게 개발할 수 있으며, 또한 유지보수가 용이한 시스템을 구축할 수 있게 된다.

그러나, 소프트웨어 개발 방법론 기술의 한계 중의 하나는 비정형성에 있다. 현재까지 제시된 방법론들에서 제공하고 있는 모델들은 모두 그나름대로의 어느 정도 정형화된 그래픽 모델을 제시하고 있으나 아직까지는 완전한 형태의 정형적인 명세수준으로 보기 어렵다[16]. 그리고, 대부분의 방법론에서는 그래픽 표현력을 이용하여 시각적으로 이해하기 쉬운 시스템의 모형을 그리는데 그 주안점을 두고 있는 형편이다. 이 결과, 보다 정형화된 표현에 의해 모호함이나 상반된 것을 발견하는 기능이나, 자동 혹은 반자동으로 시스템을 개발할 수 있는 기능, 혹은 수학적인 방법에 의한 시스템을 검증할 수 있는 기능 등을 제공하고 있지 못한 실정이다.

이에 비해, 형식명세기법은 소프트웨어 개발방법론의 위와 같은 문제점들 즉, 비정형성을 해결해 준다. 그러나, 앞절에서 살펴본 데로 현재의 형식명세기법은 결정적으로 비효율적이고 비실용적이다라는 문제가 있다[4]. 따라서, 서로의 장점 즉, 개발방법론의 실용성 및 효율성과 형식명세 기법의 정형성을 상호보완하는 형태의 새로운 기법이 연구되어야 하겠다. 또한, 현재 객체지향 기법의 장점 때문에 방법론 분야 뿐만 아니라, 형식명세기법에서도 이를 수용하고 있다. 현재 객체지향 기법을 수용한 형식명세 기법들[17-20]이 많이 제시되고 있으나 이들 모두는 개발방법론 형태가 아닌 형식언어(formal language) 수준의 정형성에 그 초점을

두고 있기 때문에 앞서 생각해 본 데로 문제점은 남아 있는 것으로 여겨진다. 이러한 사항들을 토대로 하여 향후 명세기법들이 갖추어야 할 요구사항을 생각해 보면 다음과 같다.

① 규모가 큰 실세계 개발 프로젝트를 위해 시스템 개발 방법론(methodology)과 형식명세기법의 결합이 필요하다.

② 조작가능한 단위들로 시스템을 분해하고 정형적으로 분석하기 위해 객체지향기법과 형식명세기법이 통합되어야 한다.

③ 개발자 상호간 및 사용자의 이해를 돕기 위해 그래픽 표현과 텍스트 표현이 통합되어 있어야 한다.

④ 비알고리즘적인 명세를 제공하여야 한다. 즉, 알고리즘적인 명세서는 세부설계를 수행하는 기분을 느끼도록 하기 때문이다.

⑤ 분석가가 명세를 작성하는 과정이 마치 실제 장치나 기계를 조작하는 것처럼 느끼도록 기계 같은 시뮬레이션 모델(machine-like simulation model)들을 제공해야 한다.

5. 결 론

실시간 시스템은 시스템의 정확성이 논리적인 정확성 뿐만 아니라 시간적인 정확성에도 의존하는 시스템이다. 그리고, 그 시간적인 제약이 정확히 분석이 되지 않은 상태에서 시스템이 개발되어졌을 때 돌이키기 힘든 결과를 초래할 수도 있는 위험한 시스템이 많다. 그러므로, 이러한 시스템을 효율적으로 개발하기 위해서는 시스템의 논리적인 면만 아니라 시간적인 행위를 정확히 명세하고 검증·확인하는 기술도 이 분야의 핵심연구 부분이다.

본 고에서는 이러한 실시간 시스템의 논리적(logical) 및 시간적(temporal) 행위를 기술하기 위한 구조적기법의 시스템 명세기법들과 논리 및 대수를 기반으로한 시스템 명세기법들에 대해 간략히 살펴보았다. 이 명세기법들에서는 실시간시스템 속성(real-time property)을 대부분 정해진 시간한도 내에 시스템이 유지해야 할 상태들로 표현하고 있다.

구조적 기법의 요구명세기법들과 같은 방법론(methodology) 기반 요구명세기법들은 어느 정도 정형화된 그레픽 모델들을 통해 시각적으로 이해하기 쉬운 시스템 모형을 그리는 방법(method), 도구(tool), 기술(technique)에 대하여 제시하고 있다. 이와 같이 함으로써 품질이 우수한 소프트웨어를 누구나 쉽게 개발할 수 있도록 하며, 또한 유지보수가 용이한 시스템을 구축할 수 있도록 하는 이점이 있다. 그러나, 이 명세기법들의 문제는 비정형성에 있다. 완전한 형태로 정형화되어 있지 못하기 때문에 정형화 기법들에서 취할 수 있는 이점을 취할 수 없다. 즉, 자동 혹은 반자동으로 시스템을 개발할 수 있다든가, 수학적 방법에 의해 시스템의 특성을 검증한다든가, 시스템에 존재하는 모호함이나 상반된 것을 찾는 등등의 것을 할 수가 없다.

2, 3장에서 살펴본데로, STATECHART와 페트리네트는 과거의 다른 구조적 기법들에 비해 정형적이라고 할 수 있지만, RTTL, CSP, RTL, Z, VDM 등과 같은 논리 및 대수를 기반으로한 요구명세기법들에 비해 아직도 부족한 면이 많음을 알 수 있다. STATECHART의 경우 아직 정형적인 검증을 완전히 지원하고 있지 못하며, 충분한 수준까지의 시간제약에 대해서도 지원이 되지 않는다. 그리고, 페트리네트의 경우도 마찬가지로 아직 정밀한 수준의 정형성은 지원하고 있지 못하며, 시스템이 커짐에 따라 도달성 그래프의 상태폭주 문제가 있다.

한편, RTTL, CSP, RTL 등과 같은 논리 및 대수를 기반으로한 요구명세기법들은 실시간 시스템 속성을 정형적으로 검증할 수 있도록 하고 있다. 이 정형적인 검증 기법의 제공은 실시간 시스템과 같은 위험한 시스템을 개발을 할 때 어떤 확신(confidence)을 보장(guarantee)해 주는 강점을 갖고 있다. 그러나, 이 기법들의 문제는 크고, 복잡한 시스템의 실시간 시스템 속성을 정형적으로 표현하고 검증하기가 그리 간단하지 않으며, 이 분야의 전문가가 아닌 개발자 혹은 사용자가 이 명세기법들로 명세화 된 경우에는 이해하기가 곤란한 형태로 표현되는 것이다.

이 두 가지 명세기법으로 부터 생각해 보아야할 사항은 보다 완벽한 시스템 명세기법이 되기 위해서는 방법론들에서 제공하는 비정형

성에 가까운 쪽의 기술의 강점과 정형성 부분에 가까운 형식명세기법들의 강점을 어떻게 상호보완해야 할것인가이다. 일반적으로, 시스템 명세기술 연구는 정형화 스펙트럼(formality spectrum)의 한 부분의 연구로 볼 수 있다.

이 정형화 스펙트럼 중 비정형 부분에 가까운 기술은 소프트웨어 요구사항을 기술하고 모형화하는 다이어그램, 텍스트, 테이블 및 간단한 표기법에 관련된 것으로 볼 수 있다. 또한 정형화 쪽에 보다 가까운 것은 정형화된 문법 및 의미를 이용해서 시스템 기능 및 행위를 기술하는 것이다. 이 쪽 부분의 연구에서는 수학적 형태의 형식명세기법 예를 들면, 서술논리(predicate logic)와 같은 것을 많이 이용된다.

구조적 기법 및 객체지향 기법의 요구명세 기법들은 비정형화된 부분에 가깝고, 논리 및 대수를 기반으로 한 요구명세 기법들은 수학적 형태의 형식명세기법들을 사용하므로 정형화된 부분에 더 가깝다. 그러나, 시스템 명세기술의 완성은 이 정형화 스펙트럼 전체를 완성하는데 있음을 유의해야 하겠다. 앞으로 현재의 방법론적인 명세기법들과 수학적인 기반을 가진 형식명세기법들 사이의 의미적 차이(semantic gap)를 어떻게 해결할 것인가를 더 생각해 보아야 하겠다.

참 고 문 헌

[1] Alan Burns and Andv Wellings, Real-Time Systems and Their Programming Languages, Addison Wesley, 1990.

[2] Shem-Tov Levi and Ashok K. Agrawala, Real-Tıme System Design, McGraw-Hill Publishing Company, 1990.

[3] Roger S. Pressman, Software Engineering : A Practitioner's Approach, Third Ed., McGraw-Hill, Inc., 1992.

[4] Sommerville, I. Software Engineering, 4th Ed., Addison-Wesley, Reading. Mass., 1992.

[5] Paul T. Ward and Stephen J. Mellor, Structured Development for Real-Time Systems Ⅰ, Ⅱ, Ⅲ, Yourdon Press, 1985.

[6] Hassen Gomaa, Software Design Methods for Concurrent and Real-Time Systems, Addison Wesley, 1993.

[7] Kjell Nielsen and Ken Shumate, Designing Large Real-Time Systems with Ada, McGraw-Hill, 1988.

[8] D Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, and M. Trachtenbrot, "Statemate : a working environment for the development of complex reactive systems," IEEE Transaction on Software Engineering, Vol. 16, No. 4, Apr., 1990, pp. 403-414.

[9] James L. Peterson, Petrı Net Theory and the Modelıng of Systems, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1981.

[10] C. A. R. Hoare, "Communicating Sequential Processes," CACM, Vol. 21, No. 8, Aug., 1978, pp. 666-677.

[11] Farnam Jahanian and Aloysius Ka-Lau Mok, "Safety Analysis of Timing Properties in Real-Time Systems," IEEE Transactions on Software Engineering, Vol. 12, No. 9, Sep., 1986.

[12] Jahanian F., and Mok, A., "A Graphic-Theoretic Approach for Timing Analysis in Real-Tıme Logic," Proceedings of Real-Time Systems Symposiam(IEEE), pp. 98-108, New Orleans, LA, Dec., 1986.

[13] Jonathan S. Ostroff, "Formal Methods for the Specification and Design of Real-Time Safety Critical Systems," in Real-Time Systems . Abstractions, Languages, and Design Methodologies, IEEE Computer Society Press, 1992.

[14] Marlin D. Fraser, Kuldeep Kumar, and Vijay K. Vaishnavi, "Strategies for Incorporating Formal Specifications in Softeware Development," CACM, Vol. 37, No. 10, Oct., 1994, pp. 74-86.

[15] James Rumbaugh, Mıchael Blaha, William Premerlani, Frederick Eddy, and William Lorensen, Object-Oriented Modeling and Design, Prentice-Hall, 1991.

[16] 이광용, 류성열, 정기원, "객체지향 실시간 시스템 개발 방법론," 한국정보과학회지, 제11권 2호(통권 제52호), 1993년, 4월, pp. 59-74.

[17] Stuart Faulk, John Brackett, Paul Ward,

and James Kirby, JR., "The Core Method for Real-Time Requirements," *IEEE Software*, Sep., 1992, pp. 22-33.

[18] Angelo Morzenti and Pierluigi San Pietro, "Object-Oriented Logical Specification of Time-Critical Systems," acm Transactions on Software Engineering and Methodology, Vol. 3, No. 1, Jan., 1994, pp. 56-98.

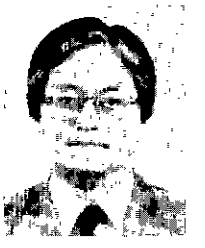
[19] Francesco Parisi-Presicce and Alfonso Pierantonio, "An Algebraic Theory of Class Specification," acm Transactions on Software Engineering and Methodology, Vol. 3, No. 2, Apr., 1994, pp. 166-199.

[20] Derek Coleman, Fiona Hayes, and Stephen Bear, "Introducing Objectcharts or How to Use Statecharts in Object-Oriented Design," IEEE Transaction on Software Engineering, Vol. 18, No. 1, Jan., 1992, pp. 9-18.

이 광 용

1991 숭실대학교 전자계산학과 (학사)
1993 숭실대학교 대학원 전자계산학과(석사)
1993~현재 숭실대학교 대학원 전자계산학과 박사과정
관심분야 : 소프트웨어공학, 실시간시스템, 인공지능, 모델링/시뮬레이션

정 기 원



1967 서울대학교 전기공학과 졸업
1982 미국 알라바마 주립대학(헌츠빌) 전산학 석사
1983 미국 텍사스 주립대학(알링턴) 전산학 박사
1966~1968 미필군 전기기사
1969~1971 대한전자공업 주식회사(자료처리과장)
1971~1975 한국과학기술연구소 전자계산실
1975~1990 국방과학연구소 책임연구원
1990~현재 숭실대학교 컴퓨터학부 교수
관심분야 : 소프트웨어공학, 모델링/시뮬레이션, 실시간시스템, 분산처리, 인공지능