실시간 시스템 개발환경 확장을 위한 명시적 스케쥴링 제어

(Explicit Scheduling Controls for Extending Real-Time System Development Environments)

이 승 룡 † Tzilla Elard†† 전 태 웅 †††
(Sungyoung Lee) (Taewoong Jeon)

요 약 실시간 또는 반응(reactive) 시스템에서 서로 경쟁하는 반응(reactions)들 간에 하나를 선택해야 하는 경우가 종종 있으며 그러한 때에 동시적(simultaneously)으로 발생하는 사건(events)들에 대한 명시적 선택은 중요하다. 왜냐하면 이같은 선택의 제어 수단 결여는 불공정하고 예측(predictable)하기 힘든 시스템을 만들 수 있기 때문이다. 본 논문은 비결정적(nondeterministic)인 시스템의 성질(behavior)을 명시적(explicit) 으로 제어 할 수 있는 포괄적 스케쥴링제어(comprehensive scheduling control) 개념을 실시간 시스템 구현언어인 Ada에 적용시킬 뿐만 아니라, 시각적 명세화 언어인 Statechart에 경주제어(race control)를 도입함으로써 모델링시 실시간 시스템이 지녀야 할 예측성을 높일 수 있는 방법을 제시한다. 이와같이 명세화와 구현언어에 일관성있는 명시적 스케쥴링제어 개념의 적용은 모델링과 실행코드 간의 격차를 줄임으로써 결과적으로 실시간 시스템 개발 환경을 확장할 수 있다.

Abstract It is common for real-time and/or reactive systems to face a situation where the system has to make a choice among a set of possible contending reactions. Several events may occur simultaneously and an explicit choice is crucial. The lack of an explicit control to the nondeterministic behavior may result in unpredictable and unfair systems. In this paper, we propose a mechanism to guarantee a predictability of real-time systems by means of imposing the comprehensive scheduling controls explicitly on the real-time concurrent programming languages such as Ada or Concurrent-C as implementation languages. We also introduce an expression scheme of the race control which is one of the comprehensive scheduling controls, to the visual specification language, Statechart. This consistent application of the scheduling control mechanism to the specification and implementation languages enables us to reduce the gap between them which will consequently extend the development working environments of real-time systems.

1. 서 론

실시간 또는 반응시스템은 사건 구동형(event-driven) 으로 외부및 내부의 자극(stimuli)에 끊임없이 반응하며 물리적 구조는 대개 분산 형태이기 때문에 병행제어(concurrency control)가 중요한 요소 중의 하나이다. 어떤 상황에서 이러한 시스템들은 오류가 없는 가용한 반응들 중에서 하나를 선택해야 할 때가 있다 결정을 내려야 할 시점에서 충돌이 발생하는 경우 명시적인 요소를 결합한 제어 메카니즘이 선택의 문제를 해결하는 역할을 할 수 있다. 그러나, 실시간 시스템 프로그램 언어인 Ada[1], [2]나 Concurrent-C[12]등은 시스템의 비결정적인 상황에서 발생하는 경주(racing) 요인을 명시적으로 제어할 수 있는 기능이 결여되어 있거나 언어 의미(language semantic)의 암시적(implicit)인 기능에 의존하므로써 실시간 시스템이 자

· 이 논문은 1995년도 한국과학재단 중점과제연구회 지원에 의해서 연구 되었음.
† 정 회 원 : 경희대학교 전자계산공학과 교수
†† 비 회 원 : Illinois Institute of Technology 전산과학과 교수
††† 종신회원 : 고려대학교 전산학과 교수
논문접수 : 1994년 8월 30일
심사완료 : 1995년 11월 14일

녀야 할 예측성을 저하시키는 요인으로 작용한다 이때, 시스템과 프로그램의 비결정적인 성질을 명시적으로 제어할 수 있는 메카니즘이 경주제어[9], [10]이다. 경주제어는 "선택 가능한 것들 중에서 무엇을 선택해야만 하는가?" 혹은 "다음에 무엇이 진행되는가?" 라는 질문에 대한 해답을 제시하며, 그것의 정의는 최초 [8]에 나타난다

비결정성의 제어는 효과적인 병행 시스템[3], [6]을 구축하는데 중요하며, 시스템의 대상이 되는 응용 프로그램과 잘 연결되어야 한다. 그렇지 않으면 일관성이 결여된 병행 언어를 사용하는 결과를 초래할 뿐만 아니라 디버깅하기가 힘든 오류를 지닌 시스템이 될 수 있다. 한편, 실시간 또는 반응 시스템의 성질을 시각적으로 표현하기 위해 고안된 Statechart는 명세화 언어로 실시간 또는 반응 시스템을 정의하는데 사용되며 내장(embedded) 시스템의 응용[20]에 적합하다. 그러나, Statechart는 경주제어를 명세화할 수 있는 기능을 가지고 있지 않다. 명세화 언어에 스케쥴링 제어 메카니즘이 부족하면 시스템의 제어 수단을 충분히 제공하지 못하여 결과적으로 불공정하고 예측하기 힘들며 잘못된 판단을 할 수 있는 시스템을 개발할 수 있게된다.

본 논문은 병행 시스템의 비결정성을 제어할 수 있는 명시적 스케쥴링제어 개념을 실시간 시스템 구현언어인 Ada 또는 Concurrent-C등에 적용시킴으로써 시스템 구현 시 이같은 개념을 구체적으로 실현시킬 수 있는 방법을 제안하며 또한, 실시간 시스템을 위한 시각적 명세화 도구인 Statechart에 포괄적 스케쥴링제어의 하나인 경주제어 개념을 도입함으로써 시스템 개발 초기 단계에서 명세화와 구현언어간의 일관성을 유지하고, 예측성과 신뢰도를 저하시키는 요인을 제거할 수 있게 한다.

본 논문의 구성은 다음과 같다 2장에서는 경주제어의 기본개념이 포함되어 있는 포괄적 스케쥴링제어의 종류를 소개하고, Ada 또는 이와 유사한 병행언어로 이러한 개념의 구현을 보여준다. 3장에서는 Statechart의 상태(state)와 전이(transition)에 대하여 기술하며, 4장에서는 명시적 경주제어 개념을 Statechart로 확장하고, 그에 대응하는 Ada 코드의 생성을 보여준다. 5장에서는 명시적 스케쥴링 제어의 예를 Ada-9X의 방위레코드(Protected Record) [4]를 사용하여 보여주며, 마지막으로 6장에서 결론을 기술한다.

2. 포괄적 스케쥴링제어의 분류

2.1 배경

과학분야에서 분류(classification)란 지식에 대한 바탕을 제공할 뿐만 아니라 새로운 학문의 개념을 일관성 있게 개발할 때 중요한 역할을 한다. 예를 들면 화학 주기율표나, 생물학의 동식물 분류, 전산학에서 알고리즘의 시간 복잡도를 다항시간(polynomial time), 의사다항시간(pseudo polynomial time), NP-hard, NP-complete, 또는 풀수 없는 문제등으로 분류하는 것이 그것이다.

이런 관점에서 볼때 병행언어에서 스케쥴링 제어의 분류는 각기 다른 동기화 전략(예를 들면 세마포어(semaphore), 모니터(monitor), 랑데브(rendez-vous))을 구사하는 병행언어들을 그룹화하고 평가할 수 있는 기능을 제공하여 줄 뿐만 아니라, 응용 시스템 개발 시 시스템이 요구하는 스케쥴링 정책에 부합하는 명세화 대 구현 대응(specification-to-implement-ation mapping)의 수단을 제공할 수 있다.

한편, 병행프로그램 언어인 Ada의 태스킹(tasking) 모델은 프로그램 내에서 서로 다른 태스크들과 통신을 하면서 병행적으로 수행되는 태스크들의 집합으로 구성되어 있다. 랑데브 메카니즘은 태스크간의 통신을 동기화하는데 사용되는데, 한 태스크는 다른 태스크의 엔트리(entry)를 호출하고 호출된 태스크가 엔트리 지점에 도착하여 그 호출을 받아들일(accept)때까지 블럭킹 된다. 랑데브가 일어나는 시점에서 호출된 태스크는 호출 태스크를 위해 자료교환을 포함한 몇가지 기능을 수행한다. 이때 하나의 태스크는 다른 많은 태스크들을 위해 서비스할 수가 있다. 그러나 어느 특정 태스크의 서비스를 받기 원하는 다른 여러 태스크들의 호출이 이루어 졌을때 이들에 대한 서비스 순서가 사전에 결정되어 있지 않으므로 비결정성(nondeterminism)의 문제가 발생한다.

병행언어에서 이같은 비결정성의 문제는 공평성(fairness)의 문제에 직결된다. 공평성을 보장하기 위한 제어기능이 부족하면 어느 특정한 태스크 만을 선택하게 되어 기아(starvation) 문제가 발생한다 또한, 시간의 제약이 엄격한 실시간 시스템에서 명시적 스케쥴링 제어수단이 없다면 비결정적인 상황에서 우선순위가 낮은 태스크가 계속하여 선택됨으로써 임무가 중요한 태스크의 마감시간을 보장할 수 없는 경우가 발생할 수 있기 때문에 예측성이 결여된 시스템이 될 수 있다

한편 스케쥴링 제어는 크게 명시적 방법과 암시적(시뮬레이션) 방법으로 나눌 수 있다. 암시적 방법은 구현언어가 명시적이고 충분하게 시스템 명세화의 요구조건을 지원하지 못함으로써 시스템 개발자는 단지 구현 언어가 제공하는 기능만을 사용하게 된다 따라서 시스템

을 시험하기가 어렵고, 검증이 복잡하며, 수정하는데 많은 노력이 필요하게 된다. 예를 들자면 어떤 언어의 기능이 의미상 FIFO 스케쥴링을 지원하는 경우, 이와 다른 스케쥴링 정책을 구현 할 때는 언어의 의미가 지원하는 정책과 실제 구현하려는 정책간의 충돌을 야기할 수가 있다. 이에 반하여 명시적 방법은 시험, 검증, 수정을 컴파일시 수행할 수 있기 때문에 사용자에게 융통성과 이식성 그리고 효과적인 구현을 제공해 준다.

그러나 이 두가지 접근 방법에는 서로 상충관계(trade-off)가 존재한다. 명시적 방법은 프로그램 언어의 설계 구조가 복잡해지게 되며, 추가적으로 컴파일러나 런타임 시스템(run-time system)의 복잡성이 증대되기 때문에 언어의 구조 설계가 까다롭다. 반면 암시적 방법은 신뢰도와, 이식성, 재사용성을 보장하기가 힘들뿐만 아니라, 실시간 시스템 개발시 예측성을 유지하기가 어렵다. 따라서 실시간 시스템의 요구 사항을 만족하기 위하여서는 명시적 스케쥴링 제어 정책이 실시간 시스템 개발시 예측성과 신뢰도를 확보 할 수 있다는 측면에서 보다 효과적인 방법이라 할 수 있다.

병행 프로그램언어 에서 사용되는 모든 가능한 스케쥴링 제어의 집합을 포괄적 스케쥴링 제어라고 한다. 포괄적 스케쥴링 제어는 가용 제어(availability control)와 경주제어로 나누어진다(그림 1). 포괄적 스케쥴링 제어의 개념은 Elrad[8]에 의해 최초로 소개되었으며, 그후 Olsson[22]에 의해 더욱 발전되었다.

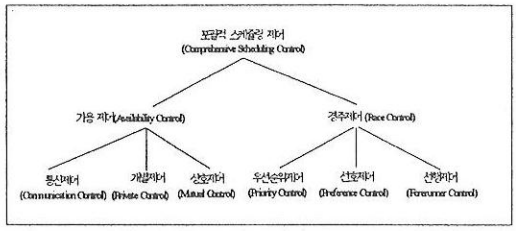

그림 1 포괄적 스케쥴링의 분류

2.2 가용제어 (Availability Controls)

가용제어는 선발(selection)을 위해서 비결정적인 선택(nondeterministic choice)을 가용(enable)하게 하거나 불가(disable)하게 하는 스케쥴링 제어이다. 이러한 제어는 시스템 관찰에 근거한 의사 결정 능력을 말하는데 그것은 일반적으로 가드(guard)를 사용함으로써 구현할 수 있으며, 경주 조건에서 기인되는 잘못된 반응(erroneous reactions)을 방지해야만 한다. 가용제어의 명세화는 현재 상태로 부터 가용한 상태로의 전이조건을 정의하는데, 그것은 올바른 수행경로(execution path)를 보장하고 무엇이 다음에 수행되어야 하는지를 결정한다.

가용제어는 얼터네티브 행동(alternative action)에 적용되는 서로 다른 제약(constraint)을 반영하기 위해서 통신제어(Communication Control), 개별제어(Private Control), 그리고 상호제어(Mutual Control)로 분류된다.

● 통신제어(Communication Control)

통신제어는 태스크의 외부 지역 환경(outside local environment)으로 부터 생성되어 대기중인 통신요청(pending communication requests)에 바탕을 두고 선택을 위하여 비결정성 얼터네티브를 가용하게 하거나 불가하게 하는 능력을 말한다. 통신제어는 특히 응답시 특정한 채널링(channelling)을 가용하게 한다. 태스크는 통신제어를 사용함으로써 다른 태스크에 의해서 기인된 사건에 응답한다. 대칭형(symmetric) 통신제어는 제어 부분에서 보내거나 받는 것을 허용하며(그림 2참조), 비대칭형(asymmetric) 통신제어는 받는것만을 위해서 가드를 허용한다.

```
[
        P1?X     ==>     X:= X+1;
    □
        P2!X     ==>     X:= X -1;
]
```

그림 2 CSP 언어에서 대칭적 통신제어의 예

그림 2에서 CSP(Communicating Sequential Processes) 코드는 대칭형 통신제어를 사용한다. P1?X (input or receive)와 P2!X (output or send)는 가드에 나타나는 문법에 의해서 통신제어를 수행할 수 있다.

● 개별제어(Private Control)

개별제어는 태스크 내부 지역상태(local state)에 따라 값을 갖는 부울 표현(boolean expression)에 근거하여 선택을 가용하게 하거나 불가하게 하는 능력이다. 이 제어는 특정한 상태전이가 발생하기 위한 지역조건을 검증하는데 도움이 된다. 그림 3 은 Ada와 유사한 가상 언어를 이용하여 명시적 개별제어의 예를 보여준다. 그림 3 에서 선호도 값은 명시적으로 초기화 지역에 주

어졌으나, 그후 실행시 선호도 값은 함수 몸체(function body)내에서 변할 수 있다.

```
Prefer1 := HIGH;
Prefer2 := MEDIUM;
Prefer3 := LOW;
loop
   select
      prefer Prefer1 when B1 ==> accept Service_1 ...;
   or
      prefer Prefer2 when B2 ==> accept Service_2 ...;
   or
      prefer Prefer3 when B3 ==> Internal_job ...;
   end select;
end loop;
```

그림 3 Ada 와 유사한 가상 언어에서 명시적 개별제어의 예

● 상호제어(Mutual Control)

상호제어는 태스크의 내부 지역상태와 호출 태스크가 전달한 파라메터에 따라 그 값이 결정되는 부울 표현에 의하여 비결정적 얼터네티브를 가용하게 하거나 불가하게 하는 능력이다. 상호제어는 호출 태스크에 의해서 전달된 파라메타의 값을 검사하거나, 검사하여 얻어진 정보를 사용하여 호출을 받아들일 것인지 또는 연기해야 할 것인가를 정하기 위하여 태스크를 가용화 시킨다. 그림 4 는 상호제어의 예를 보여주는데, Concurrent-C 는 "**suchthat**" 이라는 구조를 사용함으로써 지연시간 의미(delayed time semantics)를 허용한다. 이때, 파라메타 Distance 는 ActionRequest 엔트리를 호출한 태스크에 의해서 전달되며, 이러한 호출은 파라메타값이 1000 보다 적은 경우에 선택을 위해 가용하게 된다.

```
accept ActionRequest(Distance) suchthat (Distance<1000) {...}
```

그림 4 Concurrent-C 에서 상호제어의 예

2.3 경주제어(Race Controls)

병행프로그램이 실행되는 동안에 자원이 가용하게 되면 태스크는 프로그램 수준에서 스케쥴되기 위해 경주를 하며, 하나의 얼터네티브 선택구조(alternative selection construct)가 수행될 때 태스크 수준에서 열린 얼터네티브(open alternative)들은 선택되기 위해서 경주하며, 또한 태스크간의 통신을 하기 위해서 엔트리 수준에서 엔트리 호출들은 경주한다. 이러한 세가지 서로 다른 종류의 경주를 통제하며, 명시적 선택권을 부여하는 언어의 능력을 경주제어라고 한다.

가용제어와 경주제어의 차이점을 비교한다면 ***가용제어는 올바른 수행 경로를 보장하고 다음에 무엇이 수행될수 있는가를 결정하는 반면, 경주제어는 유용한 선택, 반응, 그리고 시스템 목표 중에서 가능한 충돌(possible conflicts)을 어떻게 해결할 것인가를 결정한다.*** 경주제어의 중요한 점은 가용한 선택 대상들을 우선순위화 할 수 있다는 데 있다.

프로그램, 태스크, 그리고 엔트리 수준에서의 경주를 반영하기 위하여 경주제어는 우선순위제어(Priority Control), 선호제어(Preference Control), 그리고 선행제어(Forerunner Control)로 분류할 수 있다. 각 수준에서 경주제어를 분류하는것은 시스템의 병행성에 대한 동적인 면을 이해할 수 있는 툴을 제공해 준다. 이러한 분류 방법(scheme)은 전체 시스템을 각기 다른 성격을 지닌 개체로 간주할 수 있다.

● 우선순위제어(Priority Control : Program-Race Control)

우선순위제어는 프로그램 수준에서 실행될 자격이 있는 태스크들 간의 경주를 해결할 수 있는 언어의 능력을 제공한다. 프로그램언어의 관점에서 본다면 우선순위제어는 컴파일시 제어할 수 있으나 런타임 시는 제어할 수가 없다. 예를 들자면 온도의 변화를 민감하게 측정하는 시스템에서 정상작업 모드에서는 "cool the system, do service1, do service2", 등의 얼터네티브들은 공평하게 선택된다. 그러나 센서로부터 얻은 온도에 대한 정보가 어떤 범위를 넘을 경우 시스템은 모듈을 변경하여 가장 높은 우선순위인 "cool the system"을 수행할 수 있는 (프로그램 수준에서 태스크간의 경주) 우선순위 제어기능이 제공되어야 한다.

● 선호제어(Preference Control : Task-Race Control)

선호제어는 태스크 수준에서 자격이 있는 얼터네티브 간의 경주를 해결하는 언어 능력을 제공한다. 각 태스크는 달성하기 위한 하위목표(subgoals)를 갖고 있다고 생각할 수 있으며 경쟁하는 하위목표중에 단지 하나의 하위목표가 어떠한 시간에 성취될수 있다. 얼터네티브 하위목표 중에서 상대적 선호 관계(relative preference relations)들이 충돌하는 경우, 시스템이 선택을 하기 위해서는 그것들은 명시적이 되어야 한다. 하위목표는 선호순서열(preference ordering)의 순서에 의해서 달성될 수 있다. 명시적인 제어 시스템은 다양한 시스템 조건에 따라 변화하는 복잡한 반응 형태를

표현하기 위해서 선호 관계를 사용할 수 있다.

Ada와 유사한 가상적인 언어로서 선호제어를 표시한다면 그림 5와 같다. 이 그림에서 선호도 값(preference value)들은 선택구조(select construct)가 런타임으로 실행되기 전에 얼터네티브에 명시적으로 할당이 된다. 런타임시 시스템은 ChangeSystemStrategy 엔트리에 새로운 선호도 값을 줌으로써 얼터네이티브 선택을 달리 할 수 있다.

```
Record. PREFERENCE := LOW.
Evaluate : PREFERENCE := MEDIUM.
React    : PREFERENCE := HIGH;
Change. PREFERENCE := URGENT;

loop
  select EXPLICIT
         prefer Record: accept NewValues(...) do ... end.
  or
         prefer Evaluate. accept UpdateStates(...) do    end;
  or
         prefer React: accept Response( ) do . . end;
  or
         prefer Change. accept ChangeSystemStrategy(—new
                preference values...) do ... end;
         — update the other three preference variables..
  end select;
end loop;
```

그림 5 Ada와 유사한 가상언어에서 선호제어의 예

• 선행제어 (Forerunner Control : Entry-Race Control)

선행제어는 엔트리 수준에서 엔트리 큐에 대기하고있는 통신 요청들 간의 경주를 해결하는 언어 능력을 제공한다. 선행제어는 보통 FIFO 의미를 사용함으로써 암시적으로 구현된다. 일부 언어는 어떠한 기준에 따라서 그들을 순차열(sorting)화 함으로써 입력되는 호출을 우선순위화하기 위한 메카니즘을 제공한다. 참고로 Concurrent-C와 같은 언어에서는 지연, 통신, 요청을 우선순위화 할 수 있는 "**by**"라는 구조가 선행제어의 한 예이다. 그림 6은 Concurrent-C에서 명시적 선행제어의 예를 보여준다. 여기서 호출(calls)들은 -Distance라는 파라미터의 값에 의해서 서열(ordering)화 할 수 있다.

```
accept ApproachingMisstle(Distance) by (-Distance) {...}
```

그림 6 Concurrent-C에서 명시적 선행제어의 예

3. Statechart의 기능

앞 장에서는 병행언어에서 사용할 수 있는 포괄적 스케줄링 제어의 기본개념을 간단히 소개하였다 본 장에서는 이들중 *경주제어의 개념*을 실시간 개발 작업 도구인 Statechart에 적용시키기에 앞서 Statechart의 기능과 거기에서 나타나는 비결정적인 상황의 예를 살펴보기로 한다.

실시간 또는 반응 시스템의 명세화나, 분석 설계 또는 개발할 때 작업 환경이 존재한다. STATEMATE[18]는 i-Logix에서 개발한 실시간 시스템처럼 크고 복잡한 시스템을 명세화, 분석, 설계, 문서화를 위하여 시각적으로 표현할 수 있는 도구이다. 이것은 구조적(structural), 기능적(functional), 성질적(behavioral) 기능을 가지는데 Statechart는 이들 중 시스템의 시간적 성질을 시각적 정형화(visual formalism) 할 수 있는 도구로써, 상태전이도(state transition diagram)를 여러가지 면에서 확장, 강화시켰다. Statechart [14], [15], [16], [17], [20]는 특히 계층적 기술(hierarchical descriptions), 상위수준 및 하위수준 사건(high-level and low-level events)의 표현이 용이하고, 또한 연속 반응(chain-reactions) 효과를 생성할 수 있는 방송통신 기능(broadcast communication machanism)을 가지고 다중 수준 병행성(concurrency or orthogonality)의 기능을 표현할 수 있다.

참고로, 기존의 많은 실시간 시스템 모델링 도구들[5], [11], [13], [19], [21], [23], [24], [25]은 정형화(formalism), 표현력(expression power), 강력한 의미(*strong semantic*) 등을 강조한 반면 구현 언어에 대한 지원 능력이 미비함으로써 명세화와 구현간의 격차가 심한 편이다. 그러나, Statechart는 명세화로 부터 Ada나 C언어의 실행코드 생성이 용이하기 때문에 명세화와 수행코드 생성간의 격차가 비교적 적다.

Statechart의 의미는 [14], [15]에 잘 기술되어 있다. 이 장에서 우리는 Statechart 의미의 가장 중요한 부분인 상태와 전이의 개념을 설명하고 그것에 근거하여 명시적 경주제어 메카니즘의 적용을 시도하고자 한다.

3.1 Statechart에 관한 일반 정보

Statechart의 기본적인 구성요소는 상태와 전이다. 상태는 시스템이 존재하고 있는 특정한 상황에 관련된 시스템의 정보를 저장한다. 전이와 관련된 3가지 요소는 사건 , 조건, 그리고 행동(action)으로 E[C]/Act 의 형태로 표현된다. 이는 사건 E가 발생하였을때 조건 C가 참이면 해당 전이가 발생하며 행동(Act)이 실행됨을 의미한다. 이때 행동은 또 다른 사건을 유발(trigger)시킬 수 있다. 전이와 연관된 각 요소들은 선택적

(optional)이다. 시간의 요구를 표현하는 기능으로는 지연(***delay***), ***timeout*** 사건, 그리고 어떤 상태에서 제한된 시간 범위를 가지고 작동하는 ***scheduled action*** 등이 있다.

또한 Statechart는 기하급수적으로 증가하는 상태 전이를 표현하는데 따르는 어려움을 극복하기 위해서 깊이(depth)라고 불리우는 계층적 추상화 개념을 시각적 정형화에 도입하고 있다.

3.2 추상화

추상화는 상태 복잡도의 기하급수적인 증가를 줄이는데 사용되며, 크게 XOR decomposition과 AND decomposition으로 나눌 수 있다.

- **XOR Decomposition**

XOR decomposition은 그래프식 추상화인데 이것은 상호 배제(mutual exclusion) 형태의 명세화를 표현한다 동일 XOR decomposition내에서 시스템은 단지 하나의 상태만이 될 수 있다. XOR decomposition은 동일 사건의 발생하에서 다음 상태로 전이하는 상태들을 함께 묶음으로써 깊이를 나타내는데 사용된다. 그림 7에서 시스템이 상태 U에 있을때, 그것은 상태 A 또는 B에 있을 수 있다.

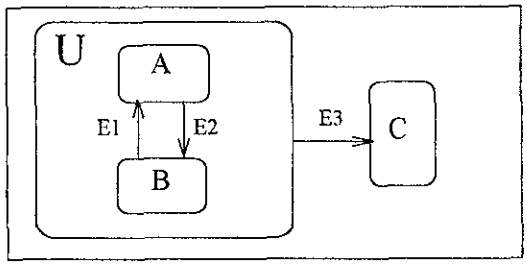

그림 7 XOR decomposition

- **AND Decomposition**

AND decomposition은 Statechart에 의해서 제공되는 또 다른 추상화인데, 이것은 동시적으로 존재하는 시스템 상태의 명세화를 표현한다. AND decomposition은 상태 중에서 orthogonality를 나타내는데 사용하며 병행성을 나타낸다

그림 8에서 시스템은 상태 A와 B에 있어야 한다 상태 A에 있다는 것은 상태 A1 또는 A2에 있다는 것을 의미하며, 상태 B에 있다는 것은 상태 B1 또는 B2에 있다는 것을 의미한다. 그러므로 시스템의 가능한 상태는 종속 상태 A와 B의 cartesian product이다.

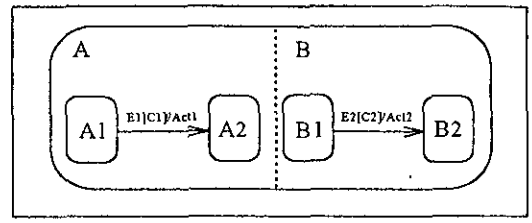

그림 8 AND decomposition

AND와 XOR decomposition은 태스킹 모델과 매우 유사한 추상화를 제공한다 상호 배제 는 XOR 그래프에 의해서 표현될 수 있으며, 병행적으로 실행되는 활동은 AND decomposition에 의해서 표현될 수 있다.

3.3 비결정성

Statechart에서 비결정적인 상황은 크게 다음과 같은 두가지의 경우에서 나타난다.

- **Orthogonal 상태로 부터의 전이에 기인한 비결정성 (얼터네티브 orthogonal 전이 사이의 경주)**

Orthogonal 상태에서 사건이 발생하였을때 전이의 비결정성을 말한다. 그러나 이러한 형태의 비결정성 전이들에 대한 명시적 선택을 표현할 수 있는 방법이 Statechart에는 없다

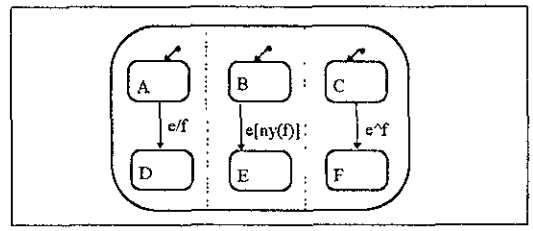

그림 9 Orthogonal 전이 사이에서 경주

그림 9는 orthogonal상태에서 전이의 비결정성을 보여준다. 시스템이 초기에 (A,B,C)에 있고 사건 e가 발생하였을때, 전이 순서가 A→D, B→E, C→F인 경우 시스템은 (D,B,F)에 도달한다 그러나 전이 순서가 B→E, A→D, C→F이면 시스템은 (D,E,F) 상태에 도달할 수도 있다 여기서 ny(f)는 not yet(f)의 약자이다

- **한 상태로 부터 여러전이에 기인한 비결정성 (한**

상태로 부터 얼터네티브한 전이 사이의 경주)

관련된 조건이 참값일 때 하나의 상태로 부터 발생하는 여러 전이간의 경쟁을 말한다. 이경우에서도 마찬가지로 경주하는 전이 중에서 명시적인 선택을 표현할 수 있는 방법이 Statechart에는 없다 그림 10에서 조건 C1과 C2가 모두 참일때 사건 e가 발생하면 A0→A1과 A0→A2의 전이 사이에 비결정성이 발생한다.

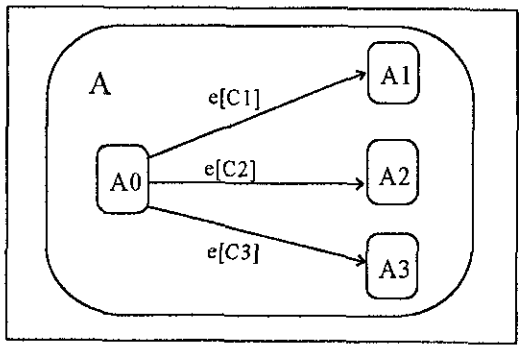

그림 10 한 상태로 부터 얼터네티브한 전이 사이의 경주 상황

4. Statechart에서 명시적 제어

4.1 Statechart에서 잘못된 전이 (Transition)를 피하는 방법

Statechart에서 사건이 발생하면 조건이 만족되는 상태로 전이가 일어나게 된다. 각 전이는 선택적 명세화인 E[C]로 나타낼 수 있는데, 이러한 표현 방법은 잘못된 실행을 막기 위하여 가용제어를 사용하는 것과 마찬가지로 잘못된 전이를 막기 위해 사용한다[9]. E[C] 명세화 방법은 혼합 통신제어(hybrid communication control) 부분 (엔트리 E라 불린다)과 개별제어 부분의 배리어조건(barrier condition) C을 합성한 것에 대응한다. E[C]에 대응되는 Ada 코드는 그림 11과 같다.

```
when C => accept E(...) do ... end;
```

그림 11 Statechart 전이 E[C]에 대응하는 Ada 코드

여기서 엔트리 E는 조건 C가 참 일때만 받아 드려지며, 얼터네티브는 다음에 가능한 경로 상태로써 가용하게 된다.

4.2. Statechart에서 경주제어

본 논문에서는 이미 가용제어와 혼사한 전이 명세화(transition specification)를 가지고 잘못된 전이를 제거하는 방법을 4.1장에서 제시하였다. 모든 전이 형태는 가용제어를 사용하여 결정하며, 그런 다음에야 가용한 상태집합에서 명시적인 선택을 할 수 있는 경주제어 메카니즘이 가능하다. 이 절에서는 3.3장에서 언급한 Statechart 의 비결정적인 동작을 제어할 수 있는 우선순위제어와 선호제어 기능을 포함하도록 Statechart를 수정하고자 한다. 이러한 시도의 이유는 시각적모델 도구에 명시적 스케쥴링 제어기능을 부여함으로써 시스템 개발초기인 명세화단계에서 구현될 실시간소프트웨어의 비결정적 특성을 미리 예측하여 제어할수있고, 명세화와 소스코드간의 일관성을 유지하여 시스템 개발시 이들간의 격차를 줄일 수 있기 때문이다.

4 2.1 Statechart에서 암시적인 경주제어

Statechart는 암시적 우선순위 제어기능을 가지고 있다. 만약 하나 이상의 전이가 동시에 발생했다면 발생 모체(source) 상태에서 전이의 우선순위는 상태의 계층 수준에 달려 있다. 예를 들면, 그림 12에서 시스템이 상태 S1에 있다고 가정하자. 만약, E1과 E2 사건이 동시에 발생하면 상위 수준에 있는 S2→Q2, 전이가 하위수준의 전이인 S1→Q1 보다 암시적 우선순위가 더 높기 때문에 S2→Q2 전이가 먼저 발생한다.

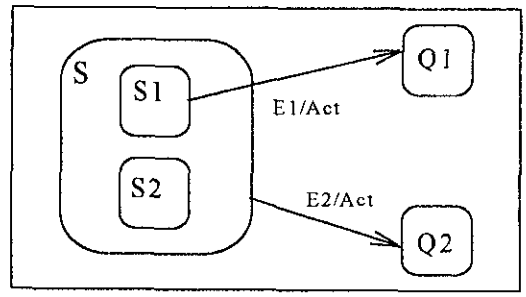

그림 12 Statechart에서의 암시적 우선순위제어

4.2.2 Statechart와 Ada에서 명시적 우선순위제어

우선순위제어는 모든 조건이 참값을 가지며, 가용한 사건(enabling events)들이 동시에 발생할 때 orthogonal 전이 사이에서 경주를 해결하기 위한 메카니즘이다 Statechart에서의 우선순위제어는 Ada에서 마치 태스크에 우선순위를 주는 것처럼, 어떤 orthogonal 컴포넌트에 우선순위를 주는 메카니즘을 사용한다(그림 13참조). 우선순위의 결정은 얼터네티브

전이가 충돌했을 때 내려진다. 우선순위 정보는 Statechart 상의 orthogonal 컴포넌트에 특별한 정의 지역(definition field)을 사용하여 주어지며, 이는 비결정성을 제어하는 역할을 하게 된다.

그림 13은 orthogonal 컴포넌트간의 경주로 인한 전이의 비결정성을 보여 준다. 시스템이 초기에 A0와 B0에 있고, E1[C1]/E4가 발생하였을 때 조건 C1과 C2가 참이고, 전이순서가 B0→B1, A0→A1 인 경우 다음 상태는 (A1, B1)이다. 그러나, 전이 순서가 A0→A1, B0→B1 이면 행동 /E4가 사전 E4를 유발하여 다음 상태는 (A1, B3)이 된다. 그러나, A 컴포넌트에 명시적으로 우선권을 줄 경우(Priority_A의 우선순위가 Priority_B 우선순위보다 높다고 가정한다) 전이의 다음 상태는 (A1, B3)로 쉽게 예측할 수 있다. 따라서, Statechart의 orthogonal 상태에서 전이의 비결정성이 나타나는 경우 이와 같은 명시적 제어가 실시간 시스템의 예측성을 보장해주기 위해서 중요한 요소가 될 수 있다.

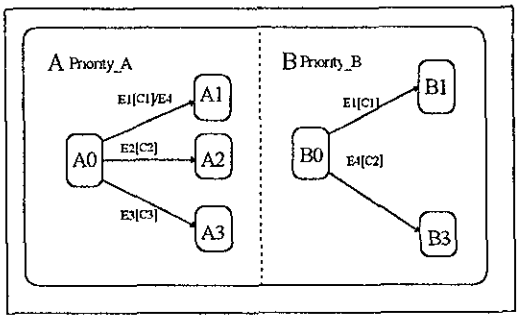

그림 13 Statechart의 Orthogonal 컴포넌트 사이의 명시적 우선순위제어

```
task A is
   pragma priority Priority_A:=2;
      -- 유동적으로 수정될 수 있다.
   entry E1();
   entry E2();
   entry E3();
end A;

task B is
   pragma priority Priority_B:=1;
      -- 유동적으로 수정될 수 있다.
   entry E1();
   entry E4();
end B;
```

그림 14 Statechart에서 명시적 우선순위 제어에 대응되는 Ada와 유사한 가상언어의 코드

이와 대응되는 Ada 태스크 모델에서는 우선순위제어의 정의는 실행 가능한 태스크들 간의 경주를 해결하는 메카니즘이다. 우선순위를 갖도록 확장된 Statechart의 표현에 대응하는 Ada와 유사한 가상언어의 코드는 그림 14와 같다.

4.2.3 Statechart와 Ada에서 명시적인 선호제어

선호경주는 한 상태로부터 발생하는 다중 전이에 의한 비결정성 상태에서 나타난다. Statechart에서 선호제어는 어느 특정한 상태에서 하나 이상의 전이가 동시에 발생할 때 경주를 해결하기 위한 메카니즘이다. 그러한 경우에 생기는 비결정성(그림 15 참조)을 명시적으로 제어하기 위해서는, 각각의 전이에 대하여 상대적인 선호도를 연관시켜 볼 수 있다. 만약 충돌이 발생하게 되면 선호도가 높은 전이 중에서 선택을 하게 된다.

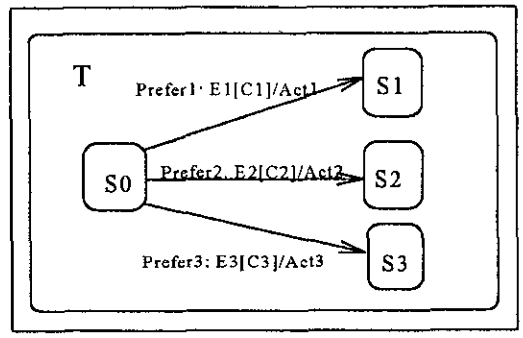

그림 15 Statechart에서 선호제어의 예

```
task T is
  entry E1();
  entry E2();
  entry E3();
end T;

task body T is
  ...
  loop
  select
        prefer1: when C1 ==> accept E1()
           do transition_to_S1 end;
     or
        prefer2: when C2 ==> accept E2()
           do transition_to_S2 end;
     or
        prefer3: when C3 ==> accept E3()
           do transition_to_S3 end;
     end select;
  end loop;
     -- S1, S2, S3의 값이 참조값으로 수정될 것이다.
  ...
end T;
```

그림 16 Statechart의 명시적 선호제어에 대응되는 Ada 코드

이와 대응하는 Ada 태스킹 모델에서 선호제어의 정의는, 테스크 수준의 가용한 선택 중에서 "**select**"문 내부에 있는 경주를 해결하는 메카니즘이다.

확장된 Statechart의 명시적 선호제어 부분을 그림 16과 같이 Ada 코드로 바꿀 수 있다. Ada-83[1] 에서는 엔트리를 위한 프로시듀어의 암시적인 선호제어 메카니즘을 가지고 있으며 이 코드는 count attribute를 이용하여 시뮬레이션 할 수 있다.

4.2.4. Ada에서 명시적인 선행제어

명시적인 선행제어는 엔트리의 큐에서 쌓여진 대기 호출 사이의 경주를 해결한다. Ada-83은 암시적인 선행 경주제어 메커니즘을 가지고 있다. 암시적인 전략은 호출이 FIFO로 순서화되어 있을 때, 우선순위 역전(priority inversion) 등 여러 가지 문제가 있음이 밝혀졌다[4]. Ada-9X는 선행제어를 위한 전략을 명시할 수 있는 융통성이 있다. 즉, 호출자(caller)의 우선순위에 의해 호출을 순서화하는 것이 그 예이다. 그러나, 아직도 사용자가 정의하는 전략을 충분히 지원하기에는 언어의 능력이 미흡하다. 현재의 Statechart 문법을 가지고 명시적 선행제어를 표현할 수 있는 방법은 없으나 2장에서 언급한바와 같이 Concurrent-C의 지연, 통신 및 호출을 우선순위화할 수 있는 "**by**"를 사용하여 Ada와 유사한 가상언어로 구현할 수 있으며, 그것의 예는 그림17 과 같다.

```
task T is
   entry E1(Urgency: urgency_type);
   entry E2(Distance: distance_type);
end T;

task body T is
   ...
   loop
      select
         when C1 ==> accept E1(Urgency: urgency_type)
            by (Urgency) do ... end;
      or
         when C2 ==> accept E2(Distance: urgency_type)
            by (-Distance) do ... end;
      end select;
   end loop;
   ...
end T;
```

그림 17 Ada와 유사한 가상언어를 사용한 명시적 선행제어 구현의 예

5. Ada-9X 방위 레코드를 이용한 명시적 스케쥴링제어의 예

본장에서는 Ada-9X의 방위레코드(protected record)를 사용하여 자원관리 문제를 해결하는 방법으로 포괄적 스케쥴링 제어의 사용 예 를 보여준다.

```
package resource_control is
   subtype resource is integer range 1..MAX;

   protected manager is
      procedure free(I: resources);
         -- 호출자가 소유한 자원의 방출
      entry allocate_high(I: resources);
         -- 상위 우선순위 태스크에 의해 호출
      entry allocate_medium(I: resources);
         -- 중위 우선순위 태스크에 의해 호출
      entry allocate_low(I: resources);
         -- 하위 우선순위 태스크에 의해 호출
   record
         resources_available : resources := MAX;
   end manager;
end resource_control;

package body resource_control is
   protected body manager is

      procedure free(I: resources) is
      begin
         resources_available := resources_available + I; ...
      end;

      entry allocate_high(I: resources)
         onlywhen (I <= resources_available) is
      begin
         resources_available := resources_available - I; ...
      end allocate_high;

      entry allocate_medium(I: resources)
         when allocate_high'count = 0
         onlywhen (I <= resources_available) is
      begin
         resources_available := resources_available - I; ...
      end allocate_medium;

      entry allocate_low(I: resources)
         when (allocate_high'count = 0
         and allocate_medium'count = 0)
      onlywhen (I <= resources_available) is
      begin
         resources_available := resources_available - I; ...
      end allocate_low;
   end manager;
end resource_control;
```

그림 18 방위레코드에 상호제어와 선행제어를 적용시킨 자원 관리자 Ada-9X 프로그램의 예

Ada-9X의 주요 요구사항은 스케쥴링에 대한 사용자의 제어 및 융통성에 있다. 따라서 경주제어는 Ada의 방위 레코드와 태스킹 모델에서 중요한 안건이 된다. 방위 레코드는 Ada-9X에서 제안한 경량 (lightweight) 동기화 기능이다[7]. 그러나, 현재 방위 레코드에 대한 Ada-9X 스케쥴링 제어는 암시적으로 규정된 언어의 의미로 인하여 만족할 만한 융통성이 확보되어 있지 않다.

그림 18은 포괄적 스케쥴링 제어 개념을 사용하여 자원관리자 (resource manager)의 문제를 해결 하는 한 예다. 자원관리자는 각기 다른 자원들의 풀(pool)로부터 자원을 할당하며 할당을 기다리는 높은 우선순위의 태스크를 처리하기 위하여 충분한 자원을 수집 할 때까지 할당을 원하는 낮은 우선순위의 태스크를 블럭킹 할 수 있다. 이때, 자원관리자 문제는 다음과 같은 두 가지의 제어가 필요하다.

(1)상호제어(mutual control): 관리자만이 충분한 자원을 가졌다면 그는 특정한 요청을 허락할 수 있지만, 얼마나 많은 자원이 요청 되었는지를 알 때까지는 그 요청이 처리될지 안될지의 여부를 모른다.

(2)선행제어(forerunner control): 우선순위 전략에 따라 엔트리에 대한 호출을 순서화 한다.

여기서 우리는 베리어로서 "**onlywhen**"을 사용하는데 그것은 호출자에 의해서 전달해 준 "**in**"모드의 파라메터에 대한 참조(reference)이다. 이와 같은 제어는 상호제어로 분류되며, 현재의 방위 레코드는 리큐(requeue) 라는 새로운 키워드와 오퍼레이션을 도입함으로써 상호제어를 시뮬레이션할 수 있다[26]. 한편 Concurrent-C언어의 경우 선행 제어를 위해서는 "**by**"를 사용한다.

"**onlywhen**"구조는 Concurrent-C "**such- that**"과 유사하다. 이 두 가지의 의미상의 차이는 베리어 조건을 검사해봄 으로서 확실해진다. 한편 "**when**" 과 "**onlywhen**" 과의 차이점은 다음과 같다. "**when**"은 엔트리 큐에 있는 모든 요청들에 적용되는 조건을 검사하는데 사용되며, 엔트리 큐와 관계가 있다. 만약 "**when**" 조건이 참이라면, 엔트리는 큐 밖에 있는 태스크에 접근 가능해진다. 반면, "**onlywhen**"은 큐에 있으면서 기다리고 있는 각각의 호출에 적용되는 조건을 검사한다. 다시 말하면 "**onlywhen**"은 엔트리에 도착하는 각각의 호출에만 관련이 있다.

5.1 명시적 경주제어가 실시간 시스템 응용에 제공하는 융통성

각기 다른 경주 결정 정책(race decision policies)에 대한 사용자의 제어와, 스케쥴에 대한 융통성의 수준을 높이는 것은 반응 시스템에 사용되는 언어의 중요한 목표이다. 실시간 응용에 대한 융통성을 주기 위하여 Ada에 명시적 제어 개념을 도입하여 위에서 언급한 자원관리자 문제를 그림 19 과 같이 수정하였다. 이때, 모든 태스크의 우선순위는 같다고 가정하였다.

```
Modification A :
    태스크의 기아현상을 피하기 위해, 많은 자원을 요청하는 태스크
    를 먼저  처리한다;

    entry allocate(I: resources)
        onlywhen (I <= resources_available)  by (I) is ...

Modification  B :
    평균적인 성능을 향상하기 위해 적은 자원을 요청하는 태스크를
    먼저 처리한다;

    entry allocate(I: resources)
        onlywhen (I <= resources_available)  by (MAX - I)
        is ...

Modification  C :
    태스크에 대한 반응시간을 빨리 하기 위해 긴급한 태스크부터 먼
    저 처리한다;

    entry allocate(I: resources, URGENT: urgency_level)
        onlywhen (I <= resources_available)  by (URGENT)
        is ...

Modification  D :
    성능 향상을 위해 주기가 짧은 자원을 요청하는 태스크부터 먼저
    처리한다;
    entry allocate(I: resources, PERIOD: period_type)
        onlywhen (I <= resources_available)  by (PERIOD) is...
```

그림 19 실시간 시스템 응용을 위한 수정된 관리자 프로그램의 예

6. 결 론

실시간 또는 반응 시스템의 개발은 문법적으로 (syntactically) 이해할 수 있는 구성 요소에 대하여 기능적, 구조적, 성질적 면이 잘 조화 되어야 한다. 명시적인 제어는 이러한 문법 구성 단위로 명세화 되었을 때 시스템의 성질적 면에 대한 요구사항들에 대하여 잘 작동하고, 특히 실제 런타임 시 발생하는 충돌을 해결할 수 있다.

본 논문에서는 실시간 또는 반응 시스템의 구현 언어인 Ada에 명시적 스케쥴링제어 메카니즘을 도입함으로써 실시간 시스템의 비결정적 성질을 효과적으로 제어하여 결과적으로 시스템의 예측성과 신뢰도를 높일 수 있

는 가능성을 제시하였다. 또한, 시각적 실시간 명세화 언어인 Statechart 에 경주제어의 개념을 표현 할 수 있는 기법을 제공하므로써 시스템 개발초기인 명세화단계에서 구현될 실시간 소프트웨어의 비결정적 특성을 예측하여 제어할수있고, 명세화와 소스코드간의 일관성을 유지하므로써 이들간의 격차를 줄여 결과적으로 시스템 개발시 작업환경을 확장할 수 있게 하였다. 이때, 경주제어의 기능은 시각적인 정형화 기법으로 캡슐화되어 있으며 최종적인 Ada 코드를 시스템의 동작에 반영하게 된다.

그리고, 우리는 Ada-9X의 방위 레코드를 이용하여 명시적 스케줄링제어 메카니즘의 적용예를 보여주므로써, 어떻게 그것이 실시간시스템 개발에 실제로 응용되는 지도 보여 주었다.

참 고 문 헌

[1] Reference Manual for the AdaProgramming Language, U S Department of Defence, ANSI/MIL-STD-1815A, 1983.

[2] Ada 9X Mapping Document, Draft, Volume II, Office of the Under Secretary of Defense for Acquisition, Department of Defense, Washington D.C , 1993

[3] G.R. Andrews and R.A Olsson, The SR Programming Language, Concurrency in Practice, The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1993.

[4] A.Burns and J Wellings, "In Support of the Ada 9X Real-Time Facilities", Ada Letters, Vol 2, No.1, January/February 1992, pp. 53-64

[5] J Coolahan and N Roussopoulos, "Timing Requirements for Time-Driven Systems Using Augmented Petri Nets," in IEEE Transactions on Sofware Engineering, Vol. 9, No. 5, September 1983, pp. 603-616.

[6] K.M. Chandy and C. Kesselman, "Parallel Progamming in 2001", IEEE Software, Nov., 1991, pp. 11-20.

[7] R.A. Duff, O. Pazy, and W.A White, "Lightweight Task Synchronization' The Protected Record Mechanism in Ada 9X", Proceedings of TRI-Ada 91, San Jose, CA, October 21-25, 1991.

[8] T. Elrad and F Maymir-Ducharme, "Distributed Language Design' Constructs for Controlling Preferences," Proceedings of the 1986 International Conference on Parallel Processing, St Charles, IL, August 19-22, 1986.

[9] T.Elrad, "Comprehensive Race Controls: A Versatile Scheduling Mechanism for Real-Time Applications", Proceedings of the Ada Europe Conference, ADA The Design Choice, Ed. Angel Alvarez, Cambridge University Press, June 1989

[10] T.Elrad, Final Report on Comprehensive Race Controls, Prepared for U.S. Army HQCECOM, Center for Software Engineering Advanced Software Technology, CIN:C08092KU 000100, February 1990

[11] A. Gabrielian and M. Franklin, "Multilevel Specification of Real-Time Systems", in Communications of the ACM, Vol. 34, No 5, May 1991,pp 50-60

[12] N. Gehani and W.D. Roome, The Concurrent-C Programming Language, Silicon Press, Summit, NJ, 1989.

[13] R Gerber and Insup Lee, "Communicating Shared Resources: A Model for Distributed Real-Time Systems", Proceedings of IEEE Real-Time System Simposium, December 1989, pp.68-78

[14] D. Harel, "On Visual Formalisms" Communication of ACM, Vol. 31, No 5, 1980, pp. 514-530.

[15] D Harel and A. Pnueli, "On the Development of Reactive Systems," in Logics and Models of Concurrent Systems, K R Apt, Ed. New York Springer-Verlag, 1985, pp. 477-498.

[16] D. Harel, A. Pnueli, J P. Schmidt, and R. Sherman, "On the Formal Semantics of Statechart," in Proc. 2nd IEEE Symp. Logic in Computer Science, New York, IEEE Press, 1987, pp. 54-64.

[17] D Harel, "Statechart: A Visual Formalism for Complex Systems," Science Computer Program., 1987, Vol 8, pp. 231-274.

[18] D Harel, et. al., "STATEMATE: a Working Environment for the Development of Complex Reactive Systems," in IEEE Transactions on Software Engineering, Vol. 16, No. 4, April 1990, pp. 403-414.

[19] F.Jahanian and A.K Mok, "Modechart' A specification Language for real-Time Systems," IEEE Transactions on Software Engineering, Vol. 20, No. 12, December, 1994, pp. 933-947

[20] S Landherr, M. Klein, and K. Fowler, "Inertial Navigation Systems Simulator: Behavioral Specification," in Technical Report CMU/SEI-89-TR-35, ESD-TR-89-46, October 1989.

[21] Z. Manna and A. Pnueli, The Temporal Logic of Reactive and Concurrent Systems: Specification Springer-Verlag, New York, NY, 1992.

[22] R.A. Olsson and C.M. McNamee, "Inter-Entry

Selection: Nondeterminism and Explicit Control Mechanisms," Computer Languages, Vol. 17, No. 4, 1992, pp. 269-282.
[23] J.S. Ostroff and W.M. Wonham, "Modeling, Specifying and Verifying Real-Time Embedded Computer Systems", Proceedings of IEEE Real-Time System Simposium, December 1987, pp.124-132.
[24] J. Peterson, Petri Net Theory and the Modeling of Systems, Prentice-Hall, Englewood Cliffs NJ, 1981.
[25] A. Shaw, "Communicating Real-Time State Machines," in IEEE Transactions on Software Engineering, Vol. 18, No. 9, September 1992, pp. 805-816.
[26] U. Verun and T. Elrad, "A Critique of the Ada 9X Mutual Control Mechanism (Requeue) and an Alternative Mapping (Onlywhen)," ACM Ada Letters, Vol. 12, No. 6, Nov.-Dec., 1992, pp. 75-80.

이 승 룡

1978년 고려대학교 공과대학 재료공학과 졸업. 1987년 미국 Illinois Institute of Technology 전산학과 석사. 1991년 미국 Illinois Institute of Technology 전산학과 박사. 1990년~1992년 미국 Governors State University 전임강사. 1992년~1993년 미국 Governors State University 조교수. 1993년~현재 경희대학교 전자계산공학과 조교수. 관심분야는 운영체제, 실시간 컴퓨팅, 실시간 스케줄링, 멀티미디어 시스템

Tzilla Elrad

1970년 이스라엘 Hebrew대학교 수학, 물리, 교육학 학사. 1978년 미국 뉴욕주 Syracuse 대학교 전산학 석사. 1981년 이스라엘 Technion Israel대학교 전산학 박사. 1981~현재 미국 Illinois Institute of Technology 전산학과 부교수. 관심분야는 Object-Oriented Concurrent- Programming, Real-time Parallel Programming, Reactive/Adaptive Systems, Reflective Systems.

전 태 웅

1981년 서울대학교 계산통계학과 학사. 1983년 서울대학교 계산통계학과 석사. 1992년 Illinois Institute of Technology 전산과학 박사. 1981~1983 금성통신 (현 LG전자) 연구소 촉탁연구원. 1983~1987 금성통신 (현 LG전자) 연구소 주임연구원. 1992~1995 LG산전 연구소 책임연구원. 1993 한국과학기술원 정보 및 통신공학과 대우교수. 1995~현재 고려대학교 전산학과 교수. 관심분야는 실시간 소프트웨어 공학, 지식기반 소프트웨어 공학, 소프트웨어 테스트, 소프트웨어 재사용