실시간 시스템의 동시성을 지원하기위한 재진입 상태챠트

김종혁°, 유민수
한양대학교 정보통신대학원
ntkjh@ihanyang.ac.kr°, msryu@hanyang.ac.kr

Reentrant Statecharts for Concurrent Real-Time Systems

Jong-Hyuk Kim°, Minsoo Ryu
The Graduate School of Information and Communications, Hanyang University

요 약

Harel과 UML에서 제공하는 기존 개념의 상태챠트는 실시간 작업들의 동시성(concurrency)을 모델링하기 어려운 단점을 가진다. 본 논문에서는 이러한 문제점을 해결하기 위해 재진입 상태머신과 재진입 상태챠트라는 새로운 개념을 제안한다. 재진입 상태머신은 병렬적으로 실행할 작업들을 교차수행(interleave)함으로서, 상태챠트로 표현되는 실시간 작업들의 동시성(concurrency)을 효과적으로 지원할 수 있다. 이러한 재진입 상태머신을 기반으로, 재진입 상태챠트는 간결하고 풍부한 표현 의미를 제공하며, 아울러 동시적으로 처리되는 행동 모델들을 구체적으로 표현할 수 있다. 재진입 상태머신과 재진입 상태챠트를 이용하여 실시간 시스템을 구현하면 프로세스간 혹은 쓰레드간의 스위칭이 불필요해짐에 따라 그에 따른 실행 오버헤드를 최소화할 수 있다.

1. 서론

상태챠트는 복잡하고 동시성을 요구하는 실시간 소프트웨어들의 동적인 행동과 동시적으로 실행되는 컴포넌트간의 동적 교류를 모델화 하는 방법으로 폭넓게 사용되고 있다. [1,2,3,5,9] 하지만, Harel과 UML에서의 상태챠트를 포함한, 전통적인 상태챠트는 동시적으로 처리되는 실시간 작업들에 대해, 미약한 동시성 개념을 지원한다.[3,5] 하지만, 기존 상태머신의 문제점은 한순간에 하나의 작업만을 나타낼 수 있으며, 이벤트 단위로 처리하지 않고, 작업 단위의 순차적인 방법으로 현재 실행되고 있는 동시성 작업들을 표현한다는 점이다.[3,5] 결과적으로 낮은 실시간 성능을 보이며, CPU와 네트워크 대역폭과 같은 시스템 자원의 낮은 이용률을 가져온다.

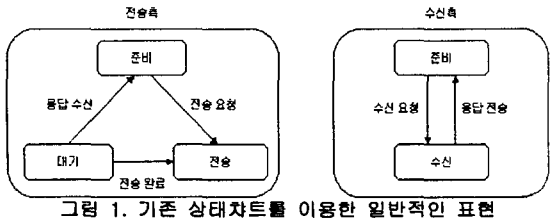

그림 1. 기존 상태챠트를 이용한 일반적인 표현

그림 1의 상태챠트는 신뢰성을 보장하는 프로토콜 구조를 따르고 있는 송신자와 수신자간의 동적인 교류를 보여준다. 송신자 컴포넌트가 수신자 컴포넌트에 메시지를 전송할 때, 전송자는 수신자로부터 응답을 받을 때까지 대기 상태에 머물러 있어야한다. 여기서 우리는 전송자 입장에서 {준비->전송->대기->준비}라는 순서화된 상태 전이 흐름에 관한 시나리오를 정의할 수 있다. 또한 메시지를 보내기위한 요청은 작업으로 모델화될 수 있다. 전송자는 이미 "대기"상태에서 작업을 수행중이고, 다른 메시지를 전송해야하는 새로운 작업이 요구된다고 가정해보자. 전송자가 휴지 상태에 있다하더라도, 새로운 작업을 즉시 처리할 수 없으며, 수신측으로부터 응답을 받음으로서 앞선 작업(task)을 완료시킬 때까지 대기상태에 머물러야만 한다.

본 논문에서는 동시적으로 처리되는 실시간 작업들을 지원하기 위해, 동시성 개념을 담고 있는 재진입 상태머신과 재진입 상태챠트라는 새로운 개념을 보이고자한다. 이 개념은 기존 상태머신과 상태챠트에, 동시성을 표현하기위한 문법과 의미, 우선순위에 입각한 이벤트 핸들링을 추가시킨 것이다.

2. 배경

유한 상태머신은 <I, O, S, S0, Δ, Γ,>의 6개 터플로 정의되며, I는 유한개의 입력 이벤트 집합을, O는 유한개의 출력 액션 집합을, S는 유한개의 상태 집합을, S0 ∈ S는 초기 상태를, Δ:S * I -> S는 상태 전이 함수를, 그리고 Γ:S * I -> O는 출력 함수를 지칭한다. 유한 상태머신은 상태 전이 다이어그램으로 표현되며, 머신의 상태에 대응되는 정점을 가진 방향성 있는 그래프와 상태 전이를 위한 교신 선을 가진다. 각 선은 전이와 관련된 입력 이벤트, 발생 조건, 그리고 입력에 따른 결과물인 행동을 포함한다.

David Harel은 상태 전이 다이어그램의 확장격인 상태챠트를 제시하였다.[5,4] 이 상태챠트에서 가장 주목할만한 개념은 '계층성'과 '직교성'이다. 계층성 개념은 상태 안에 내포된 상태와 같은 계층적 상태 분할을 허용한다. 바깥쪽에서 감싸고 있는 상태는 상위 상태, 안쪽 상태를 하위 상태라 부른다. 직교성이라 불리는, 또 다른 중요한 개념은 하나의 상태챠트 안에서 분할된 두개 이상의 상태가 동시적으로 처리되는 동시에, 독립적인 '직교적 지역들'의 상태로 존재한다는 것을 표현한다.

UML 상태챠트의 표기와 의미는 Harel의 초기 버전에 'pseudostate'와 객체지향 특성 등, 몇 가지 내용을 덧붙여 수정되었다.[3,7] UML 상태챠트는 상태머신 안에 객체를 포함하는 것을 요구한다. 또한 UML 상태챠트의 의미는 실행-완료 규칙(run-to-completion rule)에 기반을 둔다.[3,9] 입력된 이벤트들은 한 번에 처리되며, 앞선 이벤트의 처리가 완료되었을 때, 다음 이벤트가 처리된다. 그러나, 비록 Harel의 상태챠트와 UML 상태챠트를 포함하는 기존 모델들이 소프트웨어 시스템들의 행동 제어를 상대적으로 잘 표현할 수 있지만, 동시성을 요구하는 응용 프로그램들의 정확한 행동을 나타내기는 어렵다.

3. 재진입 상태머신으로 표현된 동시성 실시간 행동 모델링

이 단락에서는 재진입 상태머신과 재진입 상태챠트의 표기방

식(syntax)와 그 의미(semantics)를 정의한다. 제시된 정의는 Harel의 상태머신 모델을 기반으로 하며, 또한 의미와 구문사이의 불일치를 피하기 위해, UML 상태머신 모델을 따르고자 한다.

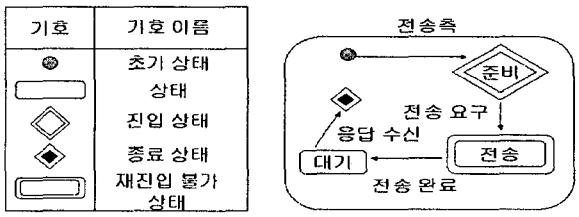

그림 2. 재진입 상태차트 기호와 전송측 수정의 예

3.1 재진입 상태차트의 의미와 그래픽 표기

재진입 상태머신 안에서 동시적으로 처리되는 실시간 작업들을 다루기 위해, 각 허가된 작업들에게 자신만의 "현재 상태"를 가지게 하고, 같은 재진입 상태머신 안에서 다른 허가된 작업들과는 독립된 상태 전이를 부여하였다. 기존의 상태머신 안에서는 유일한 현재 상태만이 허용되지만, 이러한 방식에 따라, 다수의 현재 상태는 많은 수의 허가된 작업들을 기반으로 재진입 상태머신 안에서 존재한다. 먼저 '진입 상태'의 의미를 정의하고자한다. 이 상태는 작업에 대해 허가를 내릴 수 있다는 점을 제외하고는 보통의 경우에 의미하는 상태와 같으며, 재진입 상태머신 안에서 새로운 작업들을 시작한다. 재진입 상태머신이 진입 상태에 들어왔을 때, 진입 상태는 활성화되고, 재진입 상태머신의 종료 시점까지 존재한다. 또한, 재진입 상태머신은 언제나 진입 상태와 연관된 적절한 이벤트를 다룸으로서 새로운 작업을 허가할 수 있다. 새로운 작업의 시작은 설계에 따른 상태 전이에 따라 수반되며, 이것은 허가된 작업의 현재 상태가 된다. 진입 상태들과 대비하여, '종료 상태'는 재진입 상태머신에서 허가된 작업들이 완료되었다는 의미를 갖는다. 만약 허가된 작업이 종료 상태에 들어간다면, 해당 작업은 완료된 것으로 간주된다. 또한, 우리는 한 시점에 하나의 작업만이 진입할 수 있는 '재진입 불가 상태'를 정의하고자 한다. 재진입 상태는 비선점형 I/O 디바이스 장치와 같은 공유 자원에 대해 동기화된 접근을 필요로 하는 허가된 작업들을 나타내는 경우에 유용하게 사용된다. 요약하자면, 재진입 상태차트는 <I, O, S, S0, Δ, Γ, Snr, Sentry, Sexit>의 9개의 터플로 정의되며, Snr ⊂ S, Sentry ⊂ S, 그리고 Sexit ⊂ S는 각각 재진입 상태의 집합, 진입 상태의 집합, 그리고 종료 상태의 집합을 의미한다. 재진입 상태차트의 비주얼한 표기법과 재진입 상태차트의 예는 그림 2에서 보이고 있다.

3.2 재진입 상태차트의 의미

재진입 상태머신의 핵심은 교차수행(Interleave) 방식으로, 허가된 작업들을 스위칭하여, 동시적으로 처리한다는 점에 있다. 각 작업들에 대해, 상태 전이 순서의 정확성을 보장하기위해, 재진입 상태머신은 '우선순위 이벤트 큐'와 '작업 문맥 테이블'을 유지한다. 어떤 진입 상태도 활성화되지 않은 시간 동안, 재진입 상태머신은 기존 상태머신과 동일한 행동을 한다.

먼저 '시나리오'와 '작업'에 용어에 대해 설명하고자한다. 시나리오 T_i는 진입 상태에서 종료 상태에 이르는 상태 전이의 순서화된 흐름과 진입 상태에 연계된 초기 이벤트를 포함하는 정적 속성 값들에 의해 정의되며, {초기 이벤트 : 진입 상태 -> ... -> 종료 상태}로 나타낸다. 중간에 나올 수 있는 상태 전이를 나타내지 않았는데, 이는 주어진 시나리오에 대해 진입과 종료 상태 사이에 여러 갈래가 존재할 수 있다는 점에 기인한다. 작업은 실행시간에 시나리오의 특정한 사건에 의해 정의되며, 시나리오 'T_i'의 'J번째' 사건은 '$T_{i,j}$'로 나타낸다. 각각의 작업 '$T_{i,j}$'는 그 시나리오의 정적 속성값과 {작업 식별자, 현재 상태}로 표현되는 동적 속성값에 의해 정의된다.

우선순위 이벤트 큐: 재진입 상태머신은 우선순위 이벤트 큐를 유지한다. 이벤트는 {이벤트 식별자, 작업 식별자, 우선순위}와 연관되어있다. 작업 식별자는 어떤 허가된 작업에 해당 이벤트를 전달해야하는지를 알기위해, 재진입 상태머신에서 사용된다. 발생한 이벤트는 이벤트 큐에 저장되고, 타겟이 되는 작업이 어떤 상태 전이를 일으킬 수 있는지를 검사한다. 그 후, 이벤트는 이벤트 큐로부터 타겟 작업에 전송되며, 상태머신에 의해 처리된다. 만약 이벤트 큐에 하나 이상의 적절한 이벤트가 있다면, 재진입 상태머신은 가장 높은 우선순위의 이벤트를 선택한다.

작업 문맥 테이블: 허가된 작업에 대한 모든 중요한 정보는 작업 문맥 테이블에서 유지된다. 다음 사항들을 포함한다.

- 작업 식별자 : 허가된 작업들을 구별하기위해 사용함
- 현재 상태 : 해당 작업이 마지막으로 진입한 상태
- 작업 문맥 : 작업 문맥은 재진입 상태머신이 여러 허가된 작업 사이에서 스위칭을 할 시에 사용되는 중요 정보들을 포함함. 재진입 상태머신은 다른 작업으로 스위칭될 때 문맥 정보를 저장해야만하며, 그 저장된 정보는 차후 해당 작업이 정확한 연속 흐름을 갖도록 해줌

작업 문맥 테이블은 재진입 상태머신에게 허가된 작업에 대한 중요 정보를 유지하도록 하는 중요 저장소이다. 재진입 상태머신이 다른 작업으로 스위칭되었을 때, 적절한 이벤트가 해당 작업에 전달되어, 후에 각 작업을 재시작하는데 필요한 최소의 정보만을 저장하기위해 작업 문맥 테이블이 사용된다. 작업 문맥에 어떤 내용이 포함되는지는 응용 프로그램과 구현에 따른 문제임으로 여기서 정의하지 않겠다.

작업 스위칭은 해당 작업에 따른 정보를 저장하고 불러들이는 일이 필요하다. 성능 측면에서, 작업 스위칭에 사용되는 시간은 오버헤드로 나타나지만, 분할된 프로세스 혹은 쓰레드의 형태로 동작중인 같은 상태머신에 대해, 다수의 실례를 갖고 있는 운영체제 레벨에서 프로세스나 혹은 쓰레드간의 스위칭에서 발생되는 오버헤드에 비해 매우 낮다.

3.3 재진입 상태차트와 직교성

재진입 상태차트를 통해, 우리는 직교성 개념을 포함하는 Harel의 상태차트의 장점을 이용할 수 있다. 직교적 지역들은 다른 상위 상태와 같은 방법으로 표현되기 때문에[5,4], 재진입 상태차트는 직교적 지역에 대한 개념을 손쉽게 포함시킨다. 하지만, 이럴 경우, 작업 문맥 테이블을 어떠한 방식으로 관리해야하는가에 대한 의문점이 나타난다. 이 경우 모든 직교적 지역들에 대해 단일 작업 문맥 테이블을 사용하거나, 혹은 각 직교적 지역에 대해 따로 관리하는 두 가지의 방식 모두 가능하며. 선택은 시스템의 물리적 환경에 달려있다. 멀티 쓰레드 환경에서, 각기 다른 쓰레드들은 같은 컴포넌트내의, 다른 직교적 지역 안에서 일제히 실행될 경우엔, 작업 문맥 테이블을 나눠서 사용하는 것이 바람직하며, 만약 이러한 경우 단일 작업 문맥 테이블을 사용한다면, 그 테이블로의 모든 접근은 정

보의 일치성을 유지하기위해 동기화를 해야만 한다. 따라서 동기화에 따른 오버헤드를 피하기 위해, 각 직교적 지역마다 작업 문맥 테이블을 나눠서 유지하는 것이 바람직하겠다.

4. 엘리베이터 관리 시스템을 이용한 예시

두 대의 엘리베이터와 컨트롤러가 배치되어있다고 가정한다. 엘리베이터 컨트롤러는 각 층에서 발생하는 요구들과 엘리베이터에서 나오는 요청을 포함한 사용자들의 요구들에 대해 서비스를 제공해야하고, 엘리베이터에게 이동과 멈춤에 대한 명령을 내려야한다. 또한 컨트롤러는 각 엘리베이터가 방문해야하는 층들에 대한 정보를 유지하기 위한 리스트를 갖는다.

4.2 엘리베이터 컨트롤러에 대한 재진입 상태차트

재진입 상태차트를 설계하기 위해, 우리는 엘리베이터 컨트롤러가 수행해야하는 다섯 가지 시나리오를 정의하고자 한다.

- 시나리오 T_1 : {이동 층 요구 : 준비->...->종료}
- 시나리오 T_2 : {엘리베이터 요구 : 준비->...->종료}
- 시나리오 T_3 : {엘리베이터 정지 : 준비->...->종료}
- 시나리오 T_4 : {엘리베이터 이동 : 준비->...->종료}
- 시나리오 T_5 : {문 닫기 : 준비->...->종료}

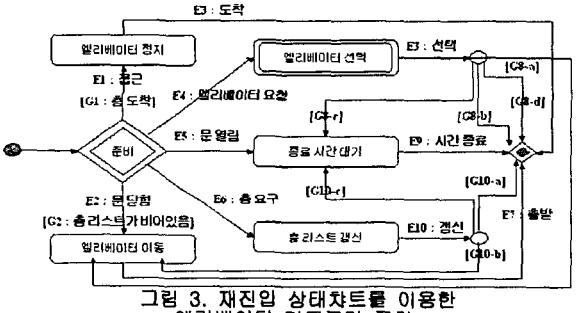

그림 3. 재진입 상태차트를 이용한 엘리베이터 컨트롤러 표현

그림 3은 엘리베이터 컨트롤러에 대한 재진입 상태차트를 보여준다. "준비" 상태는 엘리베이터 컨트롤러가 각 동시성 작업들에 대해 실행을 허용하는 진입 상태로 정의된다. 첫 번째 엘리베이터의 문이 열렸을 시 시나리오 T_5를 가정해보자. 엘리베이터 컨트롤러는 1초안에 문을 닫기 위해, 작업 $T_{5,1}$을 시작한다. 재진입 상태머신은 시간을 설정하기위해 동작을 수행하고, "대기 타이머" 상태로 들어가며, 이것은 작업 $T_{5,1}$의 현재 상태가 된다. 이때 두 번째 엘리베이터가 목적 층에 접근하고, 도착 센서가 시나리오 T_3에서와 같이 엘리베이터 컨트롤러에게 알린다. 그 후, 엘리베이터 컨트롤러는 엘리베이터에게 정지 명령을 보내기위해 새로운 작업 $T_{3,1}$을 시작한다. 재진입 상태머신은 행동을 수행하고, "엘리베이터 정지" 상태에 들어가며, 작업 $T_{3,1}$의 현재 상태가 된다. 타이머의 시간이 종료되었을 때, 작업 $T_{5,1}$로 스위치며, $T_{5,1}$은 종료 상태로 들어간다.

스케줄링 알고리즘은 비선점적이기 때문에, 대부분의 스케줄링 알고리즘은 한순간에 하나의 요청만을 다룰 수 있다. 따라서, 우리는 "엘리베이터 선택" 상태를 재진입 불가 상태로 정의하고자한다. 작업 $T_{2,j}$가 이미 "엘리베이터 선택" 상태에 있다고 가정해보자, 이 순간, 다른 작업들은 $T_{2,j}$가 이 상태를 벗어날 때까지, 재진입 불가 상태인 "엘리베이터 선택" 상태에 진입하지 못한다. 이러한 경우, 재진입 상태머신은 이벤트 큐에 이벤트들에 대한 정보를 유지하여 해당 작업의 실행을 연기하며, 후에 이벤트 우선순위에 기반하여 연기된 작업들을 시작한다.

5. 결론

본 논문에서의 결론은 매우 유망하다. 추후 동시성을 필요로 하는 시스템 개발과 연계될 연구는, 재진입 상태머신을 사용함으로서 무엇을 달성할 수 있는지에 대해 완벽한 이해를 요구하며, 이외의 외부 연구 영역에 대한 탐구 기반이 될 것이다. 특히, 두 가지 흥미 있는 주제가 있다. 첫째, 우선순위에 기반하고 있는 이벤트 관리는 설계자에게 각 작업에 관해 정확성을 요구하는 시간적 행동 분석을 수반하게 한다는 점이다. 이것은 어떤 시점의 오류들은 분석단계에서 발견되는 것이 시스템이 구현된 이후에 발견되는 것보다 보수 측면에서 비용이 적게 든다는 이유에 기인한다. 두 번째, 재진입 상태차트는 실행중인 모델들을 사용한다는 점이다. 이는 재진입 상태차트가 형식에 맞춘 구문과 엄격한 의미를 가지고 정의되었기 때문이다.

6. 참고 문헌

[1] Grady Booch, James Rumbaugh, and Ivar Jacobson. "The Unified Modeling Language User Guide" Addison Wesley, 1999.

[2] Hassan Gomaa. "Designing Concurrent, Distrubuted, and Real-Time Applications with UML" Addison Wesley, 2000.

[3] The Object Management Group. "OMG Unified Modeling Language Specification" Version 1.5, 2003.

[4] D. Harel and A. Naamad. "The Statemate semantics of satecharts" ACM Transaction on Software Engineering and Methodology, 5(4):293-333, October 1996.

[5] D. Harel. "Statecharts: A visual formalism for complex systems" Science of Computer Programming, 8(3):231-274, June 1987.

[6] D. Harel and Eran Gery. "Executable object modeling with statecharts" IEEE Computer, 30(7):31-42, 1997.

[7] D. Latella, I. Majzik, and M. Massink. "TOWARDS A FORMAL OPERATIONAL SEMANTICS OF UML STATECHART DIAGRAMS" IFIP TC6/WG6.1 Third International Conference on Formal Methods for Open Object-Based Distributed Systems, 1999.

[8] Daniel Nikovski and Matthew Brand. "Decision-theoretic group elevator scheduling" Proceedings of the 13th International Conference on Automated Planning and Scheduling, 2003.

[9] Bran Selic, Garth Gullekson, and Paul Ward. "Real-Time Object Oriented Modeling" Wiley, 1994.

[10] F. Wagner. "VFSM executable specification" IEEE International Conference on Computer System and Software Engineering, page 226-231, 1992.

[11] Klaus Weidenhaupt, Klaus Pohl, Matthias Jarke, and Peter Haumer. "Scenarios in System Development: Current practice" IEEE Software, 15(2):33-45, March/April 1998.

[12] Jon Whittle and Johann Schumann. "Generating statecharts designs from scenarios" International Conference on Software Engineering, page 314-323, 2000