실시간 객체 지향 모델의 스케쥴 가능성을 고려한 구현 자동화

김세화[0] 조석제 홍성수
서울대학교 전기컴퓨터공학부
{ksaehwa, sjcho, sshong}@redwood.snu.ac.kr

Automated Schedulability-Aware Implementation of Real-Time Object-Oriented Models

Saehwa Kim[0] Sukjae Cho Seongsoo Hong
School of Electrical Engineering and Computer Science

요 약

객체지향 디자인 방법론과 이에 대한 CASE 툴은 실시간 소프트웨어 개발 분야에서 널리 사용되고 있다. 객체 지향 CASE 툴은 객체 지향 모델로부터 테스크들을 생성하는 단계를 거쳐야 한다. 그러나 객체와 태스크는 근본적으로 차이가 있으며, 또한 실시간 특성을 반영하도록 태스크를 구성하는 것은 매우 어려운 일이므로, 현재의 CASE툴들은 개발자가 개체와 태스크의 매핑을 직접 하도록 하고 있다. 본 논문은 실시간 객체지향 모델을 스케줄 가능성을 고려해 멀티 스레드 태스크로 자동적으로 매핑하는 방법을 제안하고 본 방법의 스케줄 가능성 테스트 방법을 제시한다.본 논문에서 제안하는 방법은 실시간 객체 지향 모델에서 서로 다른 주기와 데드라인을 갖는 트랜젝션을 추출하고, 이를 스케줄 가능하도록 그룹화하여 스레드로 만드는 것이다.

1. 서론

실시간 객체 지향 모델과 객체 지향 CASE 툴은 실시간 시스템 소프트웨어 설계에 널리 사용되고 있다. 객체 지향 CASE 툴에서는 객체 중심으로 설계한 디자인 모델들을 실제 수행되는 주체인 태스크로 변환하는 과정이 필요하다. 그러나 이러한 과정은 객체 모델과 태스크의 근본적인 특성 차이와 스케줄 가능성 테스트에서 오는 어려움으로 실제 CASE 툴에서는 개발자가 개체와 태스크 간의 매핑을 직접 하도록 하고 있다. 그러나 개체와 태스크 간의 매핑은 시스템의 스케줄 가능성에 큰 영향을 주는 요소이다. 실시간 객체 지향 모델의 자동화된 구현과 스케줄 가능성 테스트에 관한 연구는 [1], [2], [3], [4], [5]와 같은 연구에서 이미 다루어 졌다. 그러나 이러한 연구들은 객체에서 태스크로의 매핑이 이루어진 상태에서만 스케줄 가능성을 테스트하며 스케줄 가능성을 고려한 상태에서 객체에서 태스크로의 매핑하는 방법을 제시하지는 못 했다. 본 논문에서는 실시간 객체 지향 모델에서 스케줄 가능성을 고려해서 태스크로 매핑하는 자동화된 방법을 제시한다. 또한 이 방법을 현재 널리 사용되고 있는 CASE툴에 어떻게 적용할 수 있는지 제시한다. 모델로부터 테스크를 추출하는 방법으로서 트랜잭션(transaction)의 개념을 사용하도록 한다. 트랜잭션은 각 노드의 외부 입력으로부터 하나의 외부 출력을 내기까지의 end-to-end 수행으로 정의한다. 실시간 객체 모델로부터 트랜잭션은 수행 과정의 이벤트의 연결로 표현할 수 있다. 본 논문에서는 실시간 객체 모델로부터 트랜잭션을 추출하고, 스케줄 가능성을 고려하여 트랜잭션의 그룹화하여 태스크로 매핑시키는 방법을 제안한다. 또한 이렇게 생성된 태스크의 수를 줄이기 위하여 [6]에서 제시한 preemption threshold 기법을 적용하였다.

이 논문의 구성은 다음과 같다. 2장에서는 본 논문에서 대상으로 하고 있는 객체 지향 언어인 UML-RT를 이용한 디자인과 구현에 관해 간략히 살펴보며, 3장에서는 우리의 접근 방법에 관해 자세히 다루도록 하겠다. 4장에서는 우리의 방법을 실제 CASE 툴에서 어떻게 구현될 수 있는지 다루도록 하며 5장에서는 논문의 결론을 맺도록 한다.

2. UML-RT와 CASE 툴의 개괄

이 장에서는 우리의 방법을 적용시킨 언어로 선택한 UML-RT와 UML-RT를 채용한 CASE 툴인 RoseRT에 관해 간략히 설명하고 현재 CASE 툴의 문제점을 제시하겠다.

2.1 UML-RT 모델링 언어

UML(Unified Modeling Language)는 소프트웨어 시스템을 규정, 시각화, 구조화, 문서화하며 또한 수 행까지 하는 그래픽한 언어이다. UML은 실제 대형 시스템에서도 성공적으로 적용되어 유용성을 입증 받았으며, 모델링 언어의 산업 표준으로 인정받고 있다. UML-RT(UML Real-Time)는 UML에 이벤트 중심이며 분산처리의 가능성이 있는 시스템에 필요한 새로운 개념을 추가한 것이다. UML-RT의 가장 기본

요소는 캡슐(capsule)이다. 캡슐은 특정 작업을 수행하는 수행가능 한 개체이다. 캡슐들은 기본적으로 서로 병렬적으로 동시 수행이 가능한 것으로 가정한다. 캡슐은 포트(port)라 불리는 인터페이스 오브젝트를 통해서만 메시지를 주고 받을 수 있다. 캡슐들 간의 포트들의 연결을 통해서 모델의 구조가 규정된다. 모델의 행동 양식은 FSM(finite state machine)로 정의된 각 캡슐에 따라 규정된다. 캡슐의 상태는 캡슐의 외부 포트 혹은 내부 포트로 전달된 메시지에 의해서만 변화가 일어난다. 본 논문에서는 여기서 설명한 UML-RT의 용어를 사용하도록 하겠다.

2.2 RoseRT CASE tool

RoseRT는 객체지향 실시간 시스템을 시각적인 디자인 할 수 있는 환경을 제공한다. 또한 RoseRT는 디자인된 모델로부터 어플리케이션 코드와 코드의 틀, 실시간 시스템 라이브러리를 제공하며, 이를 기반으로 프로그램을 생성할 수 있는 환경을 제공한다. RoseRT에 의해 생성된 프로그램의 핵심은 RoseRT run-time system(RTS)라 불리는 것으로 각 캡슐 개체에 전달되는 메시지들을 관리하는 역할을 한다. RTS는 메시지 큐에서 메시지를 받아서,메시지를 받는 캡슐의 FSM에 의해 정의된 캡슐의 행동을 수행하도록 한다.

UML-RT와 RoseRT는 복잡한 시스템을 시각적으로 모델링하고 실행코드를 만들어 내는 환경을 제공하지만, 프로그램 수행에 불필요한 블록킹을 발생시킨다. [3]에서 언급된 바와 같이 UML-RT에 의해 디자인된 모델은 (1) 메시지와 스레드의 이중 스케쥴링 (2) 메시지 패싱 (3) run-to-completion의 세 가지의 블록킹 원인을 갖게 된다.

3. 객체 지향 디자인 모델과 태스크 매핑 방법

이 장에서는 실시간 객체 지향 디자인 모델을 멀티스레드를 이용해 구현하는 방법을 제시하도록 하겠다. 우리의 접근 방법은 3단계로 이루어져 있다. (1) 디자인된 모델로부터 트랜잭션 추출한다. (2) 트랜잭션을 스케쥴링을 고려해 그룹화해 논리적 스레드(logical thread)를 만든다. (3) Preemption Threshold를 적용해 논리적 스레드 중 통합이 가능한 것은 병합해 물리적 스레드(physical thread)를 만든다. 논리적 스레드라는 용어는 최종 구현에 사용될 물리적 스레드와 구별짓기 위해서 사용하였다.

3.1 디자인 모델

이 절에서는 본 논문에서 제시한 디자인 모델에 사용할 부호와 가정을 정의하도록 한다. 각 디자인 모델은 입력 이벤트와 출력 이벤트의 집합을 가지고 있다. 각 외부 입력은 이벤트의 연속적인 발생, 즉 일련의 작업을 수행하도록 하는데 이것을 트랜잭션(tranaction)이라고 정의하자. UML-RT에서는 하나의 외부 입력 이벤트는 여러 개의 트랜잭션을 발생시킬 수 있다. 본고에서는 외부 입력 $E(\tau_i)$에 의해 발생한 일련의 액션들 $<A^i_1, A^i_2, \ldots, A^i_{ni}>$에 의한 트랜잭션을 τ_i로 나타내기로 한다.

액션 A^i_j에 의해 발생한 스테이트 트랜지션의 경우 트랜지션 이전 스테이트를 $S_s(A^i_j)$, 트랜지션 이후 스테이트를 $S_d(A^i_j)$로 나타내기로 한다. 또한 트랜지션에 의해 발생한 액션을 수행하는 데 최악 수행 시간은 $C(\tau_i)$로 나타내기로 한다. 또한 이 논문에서는 디자인 모델에 대해 다음과 같은 가정을 한다.

- 모든 외부 입력은 주기적이다.
- 트랙잭션의 데드라인은 주기와 같다.
- 메시지의 우선 순위는 디자인 시가 아니라 구현시에 결정된다.
- 같은 개체에서 시작한 트랜잭션들은 동시에 병렬적으로 수행될 수 없다.

3.2 1단계 : 트랜잭션의 추출

이 단계에서는 트랜잭션이 고정된 우선 순위를 갖지 않고 있다. 트랙잭션 τ_i의 우선순위를 $\pi(\tau_i)$로 하면, 액션 A^i_j과 이 액션에 의해 발생하는 이벤트들은 모두 τ_i의 우선순위를 상속 받게 된다. 액션 A^i_j의 수행이 낮은 우선순위의 액션에 의해 블록될 경우는 낮은 우선 순위의 액션이 같은 객체를 사용중일 경우, 이전의 상태 이전이 끝나기 전에는 다른 상태 이전이 대기해야 하는 run-to-completion 조건 때문에 생길 수 있다. 이 액션 A^i_j 의 블록킹 시간은 다음과 같은 식에 의해 계산할 수 있다.

$$b(A^i_j) = \max_{l::\pi(\tau_l)<\pi(\tau_i)} \{\max_k \{C(A^l_k) :: O(A^l_k) = O(A^i_j),\ S_d(A^l_k) = S_s(A^i_j)\}\}$$

3.3 2단계 : 논리적 스레드의 추출

논리적 스레드의 추출은 3단계로 이루어져 있다. 각 논리적 스레드에 대해서 스레드를 구성하는 트랜잭션을 찾아내고, 우선순위를 결정하며, preemption threshold를 찾는다.

트랜잭션의 그룹화 : 논리적 스레드의 구성원을 찾기위해서는 우선 동시에 실행이 불가능한 트랜잭션들을 찾아야 한다. 외부 이벤트를 처음으로 받는 객체를 공유하는 트랜잭션은 하나로 논리적 스레드로 묶을 수 있고 이 논리적 스레드를 L_i로 나타내도록 한다. 논리적 스레드는 여러 개의 트랜잭션의 집합이므로 여러 개의 주기, 최악 수행시간, run-to-completion 블록킹 시간을 가질 수 있다.

스케줄 가능한 우선 순위 할당 : 우리는 우선 순위 할당에는 Audsley의 알고리즘을[7] 스케줄 가능성 테스트에는 응답시간 분석[2][8]을 사용하였다. 논리적 스레드는 여러 개의 스케줄링 특성을 가지고 있으므로 기존의 분석 방법을 다음과 같이 확장하였다.

$$R(L_i) = \beta(L_i) + C^R(L_i) + \sum_{\forall j, \pi(L_j)>\pi(L_i)} \left(\left\lceil \frac{R(L_i)}{T(L_j)} \right\rceil \right) \cdot C^I(L_j)$$

$R(L_i)$는 $C^R(L_i)$와 $\beta(L_i)$가 증가함에 따라 단조 증가 하므로 $C^R(L_i)$와 $\beta(L_i)$의 값을 논리적 스레드의 트랜잭션 중 수행 시간과 run-to-completion 블록킹 시간을 최대로 하는 트랜잭션의 수행 시간과 블록킹 시간으로 한다. 반면에 L_i에 대한 높은 우선 순위 작업에 의한 간섭 시간을 $C^I(L_j)/T(L_j)$에 따라 단조 증가함으로 $C^I(L_i)$와 $T(L_j)$의 값을 주기 대 수행시간이 최대가 되는 트랙잭션의 수행 시간과 주기로 한다.

최대 preemption threshold 할당 : 높은 preemption threshold를 갖는 테스크일수록 더 적은 문맥 교환 횟수를 갖게 되므로 각 논리적 스레드에 가능한 최대의 preemption threshold를 할당하도록 한다. 여기서는 최대 preemption threshold 할당 알고리즘으로 [10]에 제시된 방법을 사용한다. 이 알고리즘은 preemption threshold를 고려하여 [2]와 [8]의 스케줄 테스트 알고리즘 변형하여 사용하며또한 본 논문에서 제시한 논리적 스레드의 특성을 고려하여 [10]에 제시된 분석 방법에 변경을 가하였다. Preemption threshold γ(Li)를 가진 논리적 스레드 Li의 응답시간 R(Li)는 다음과 같은 식에 의해 계산될 수 있다.

$$B(L_i) = \max_j \{C^I(L_i) :: \gamma(L_j) \geq \pi(L_i) > \pi(L_j)\}$$

$$S(L_i) = B(L_i) + b(A_1^i) + \sum_{\forall j, \pi(L_j) \geq \pi(L_i), i \neq j} \left(1 + \left\lfloor \frac{S_i}{T(L_j)} \right\rfloor\right) \cdot C^I(L_j)$$

$$\mathcal{R}(L_i) = \mathcal{F}(L_i) = S(L_i) + C^R(L_i) + (\beta(L_i) - b(A_1^i)) + \sum_{\forall j, \pi(L_j) > \gamma(L_i)} \left(\left\lceil \frac{\mathcal{F}(L_i)}{T(L_j)} \right\rceil - \left(1 + \left\lfloor \frac{S(L_i)}{T(L_j)} \right\rfloor\right)\right) \cdot C^I(L_j)$$

이 수식에서 B(Li)는 낮은 우선 순위를 가졌지만, 높은 preemption threshold를 가진 태스크에 의해 블록된 시간을 나타낸다. S(Li)와 F(Li)는 각각 논리적 스레드의 최악 시작 시간과 종료 시간을 나타낸다.

트랜잭션과 트랜잭션 내의 각 액션은 논리적 스레드의 우선 순위와 preemption threshold를 그대로 이어 받게 된다. 액션은 액션을 발생시키는 메시지와 연결되어 있으므로 메시지는 메시지에 의해 수행되는 논리적 스레드의 우선 순위와 preemption threshold를 할당받게 된다.

3.4 3단계 : 물리적 스레드 추출

논리적 스레드는 서로 preemption 할 수 없는 경우 하나의 그룹으로 합치는 것이 가능하다[9]. 이러한 그룹을 하나의 물리적 스레드로 만들 경우 스레드의 개수를 크게 줄일 수 있으며, 이에 따라 실행 시간의 문맥 교환 시간과 메모리 사용량을 줄일 수 있다.

4. 제안된 방법을 위한 툴 지원 방법

RoseRT에서는 구현 단계의 각 스레드는 controller라 불리는 메시지 처리기를 가지고 있다. 각 메시지는 메시지를 받는 캡슐의 정보를 가지고 있으며, 캡슐은 자신에 대한 콘트롤러의 정보를 가지고 있다. RoseRT의 구현과 달리 우리의 구현 방법에서는 각 메시지를 스레드와 매핑시킨다. 이를 위해서 본 연구에서는 LogicalThread라는 구조체를 정의하였다. 이 구조체는 논리적 스레드의 우선 순위와 preemption threshold, 연관된 물리적 스레드의 정보를 가지고 있다. 우리는 UML-RT 데이터 구조에 LogicalThread를 추가하고, 메시지 전송 함수와 target controller를 찾는 방법을 변경하였다. 우리는 콘트롤러를 변경하여 전송된 메시지의 우선 순위를 동적으로 처리할 수 있도록 변경하였다.

5. 결론

본 논문에서는 객체 지향 모델을 실시간 시스템의 특성을 고려하면서도 자동화된 방법으로 실행 가능한 멀티 스레드 태스크로 생성하는 방법을 제시하였다. 우리는 이 방법으로서 상호 동시 실행이 불가능한 트랜잭션들을 하나의 논리적 스레드로 만들었다. 이 논리적 스레드는 다양한 스케줄 특성을 가질 수 있으므로 이를 고려하여 스케줄 가능성을 태스트할 수 있는 방법을 제시하였다. 여기에 다시 preemption threshold 알고리즘을 적용하여 스레드의 개수를 줄였다. 결과적으로 본 논문에서는 객체 지향 모델에서 적은 수의 스레드 만으로 수행 가능한 코드를 생성해 내는 방법을 제시하였으며, 또한 이 과정이 완전히 자동화 될 수 있음을 보였다.

5. 참고 문헌

[1]. C.Liu and J. Layland. " Scheduling algorithms for multiprogramming in a hard real-time environment." Journal of the ACM, pages 46-61, January 1973

[2]. J.P. Lehoczky,L.Sha, and Y. Ding. "The Rate Monotonic Scheduling Algorithm: Exact Characterization And Average Case Behavior", In Proceedings of IEEE Real-Time Systems Symposium, pages 166-171, 1989

[3]. D. Gaudrean and P. Freedman, "Temporal Analysis And Object-Oriented Real-Time Software Development: A Case Study With ROOM/Objectime", In Proceedings of IEEE Real-Time Systems Symposium, pages 110-118, May 1996

[4]. M. Saksena and P. Freedman and P. Rodziewicz. "Guidelines For Automated Implementation Of Executable Object Oriented Models For Real-Time Embedded Control Systems", In Proceedings of IEEE Real-Time Systems Symposium, pages 240-251, Hune 1997

[5]. M. Saksena and A. Ptak and P. Freedman and P. Rodziewicz. "Schedulability Analysis For Automated Implementations Of Real-Time Object-Oriented Models" ,In Proceedings of IEEE Real-Time Systems Symposium, pages 92-102, December 1998

[6]. Y. Wang and M. Saksena, "Scheduling Fixed-Priority Tasks With Preemption Threshold", Proceedings of IEEE Real-Time Computing Systems and Applications Symposium, pages 328-335, 1999

[7]. N. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times", Technical Report YCS 164, Department of Computer Science, University of York, England, December 1991

[8]. K. Tindell and A. Burns and A. Wellings, "An Extendible Approach For Analyzing Fixed Priority Hard Real-Time Tasks", Real-Time Systems Journal, pages 133-151,1994

[9]. M. Saksena and Y. Wang, "Scalable Real-Time System Design using Preemption Thresholds", In Submission: Proceedings of IEEE Real-Time Systems Symposium, 2000