컴포넌트 기반 임베디드 소프트웨어를 위한 모델 중심 성능 예측 기술

김희진, 이선우, 김지민, 유민수
한양대학교 전자컴퓨터통신공학과
e-mail: {hjkim, swlee, jmkim, msryu}@rtcc.hanyang.ac.kr

Model-Centric Performance Estimation for Component-Based Embedded Software

Heejin Kim, Sunwoo Lee, Jimin Kim and Minsoo Ryu
College of Information and Communications, Hanyang University

요 약

현재까지의 컴포넌트 기술은 컴포넌트의 구조 및 인터페이스 그리고 컴포넌트 간의 조립 및 상호작용과 같은 컴포넌트의 기능적인 측면을 중심으로 연구가 이루어져왔다. 이러한 기능적인 측면의 컴포넌트 연구는 컴포넌트의 재사용성을 중심으로 한 소프트웨어의 생산성 향상과 품질의 최적화 등을 가능하게 하였다. 그러나 기존의 연구들은 컴포넌트의 성능적인 측면을 고려하지 않아 임베디드 소프트웨어에서 요구되는 성능을 만족시키거나 분석하는 것이 어렵다. 본 논문에서는 소프트웨어를 구현하기 전에 컴포넌트 모델을 이용하여 소프트웨어의 성능을 미리 예측하는 방법을 제시한다. 제안하는 방법은 성능 예측을 가능하게 하는 컴포넌트 및 태스크 모델을 정의한 후, 테스크 레벨의 응답시간을 예측하는 기법을 소개한다. 아울러 캐시나 파이프라인과 같은 하드웨어가 성능에 미치는 영향도 함께 고려한다.

1. 서론

지금까지의 컴포넌트 기술은 컴포넌트의 재사용성이 강조되어서 컴포넌트의 구조나 인터페이스와 같은 기능적인 측면의 연구와 개발에 치중되어 왔다. 그러나 컴포넌트의 성능적인 측면은 간과하여 임베디드 소프트웨어의 성능 분석 및 예측이 어렵다는 문제점이 있다.

일반적인 소프트웨어의 개발과정에서는 개발 과정 중에 소프트웨어 요구사항이 변경되거나 구현된 소프트웨어의 성능 측정 결과가 요구사항을 만족시키지 못할 경우, 대부분의 개발 과정을 다시 반복하게 된다. 하지만 컴포넌트의 성능 분석 및 예측이 가능하면 소프트웨어 개발 초기 단계에서 성능 요구사항의 만족 여부를 미리 확인 할 수 있을 것이고 적은 비용으로 다양한 설계 옵션을 시도하는 것이 가능해 질 것이다.

본 논문에서는 임베디드 소프트웨어의 성능을 예측할 수 있는 방법을 제안한다. 이 방법은 크게 두 가지로 구성된다: 성능 예측을 고려한 컴포넌트 모델 정의, 이를 기반으로 컴포넌트와 태스크의 성능 속성 값을 도출하는 기법. 컴포넌트의 정적 성능 분석을 위해서 WCET(Worst Case Execution Time)분석 방법이 사용되었다.

성능 예측을 위한 새로운 모델로는 WCET 분석을 위한 WCET 모델, 성능 분석 및 예측을 고려한 새로운 컴포넌트 모델 그리고 결합된 컴포넌트의 성능 분석을 위한 태스크 모델을 제안한다. 또한 이러한 모델을 이용하여 다수의 컴포넌트로 조립된 태스크의 성능을 예측하는 기법을 제시한다. 이 기법은 캐시와 파이프라인이 성능 속성에 미치는 영향도 고려한다. 태스크 성능의 예측은 멀티태스킹 환경에서 실시간 스케줄링 이론을 적용하여 시스템 관점의 실행 시간을 예측하는 방법도 소개한다.

본 논문은 다음과 같이 구성된다: 2장에서는 성능 예측을 위한 모델들을 제시하고, 3장에서는 모델을 이용한 컴포넌트와 태스크의 성능 예측 방법을 설명한다. 4장에서는 논문의 요약과 앞으로 계획을 제시하고 논문을 결론짓는다.

1.1 관련연구

기존에 연구되었던 소프트웨어 컴포넌트들 중에 대표적인 컴포넌트들로는 Koala[1], KobrA[2], Pin[3] 그리고 UML[4] 컴포넌트 등이 있다. 이러한 컴포넌트들은 컴포넌트를 구문(Syntax), 의미(Semantics) 그리고 조합(Composition)의 관점에서 정의한다. 다시 말해, 컴포넌트 기반 개발(Component-Based Development)에 사용될 컴포넌트들의 구조와 상호작용만을 컴포넌트 모델이나 프로그래밍 언어 등의 형식을 빌어서 기술한다.

UML 컴포넌트는 블랙박스 형태로 소프트웨어를 구성하는 컴포넌트들 사이의 연관 관계를 기술할 뿐 컴포넌트 내부 구조에 관한 정보는 표현 할 수 없다. 본 논문에서는 컴포넌트의 내부 구조를 표현하기 위해서 활동(activity)

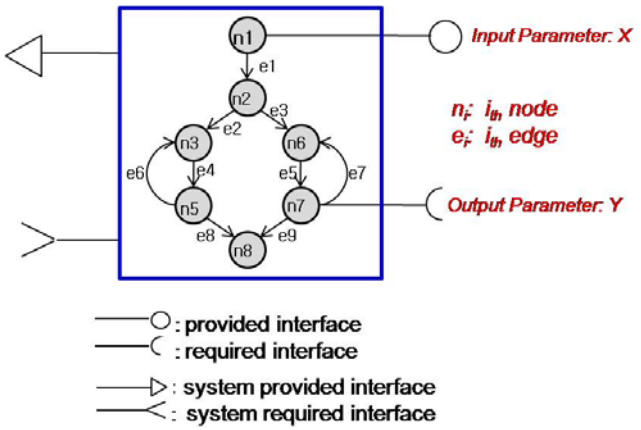
다이어그램을 혼용하여 사용하고 있다. 활동 다이어그램의 순차성과 병렬성은 행동 로직을 표현하는데 적합하여 C 언어의 순차적 행동 로직을 표현할 수 있다.

CMU(Carnegie Mellon Univ.)와 ITEA에서는 각각 PECT[5]와 space4U[6]라는 본 논문과 유사한 목적의 연구가 진행 중이다. PECT는 Reasoning Framework와 Construction Framework라는 컨셉을 제안하고 컴포넌트 단위의 성능 속성으로부터 어셈블리의 성능 속성을 예측하는 추상적인 방법을 제안하고 있다. 또한 space4U 프로젝트는 Robocop 컴포넌트라는 새로운 컴포넌트를 통해 응용 프로그램의 실시간 속성과 자원소비를 예측할 수 있는 Framework를 개발하려는 연구가 진행되고 있다.

2. 성능 예측을 위한 모델

2.1 WCET 분석을 위한 모델

WCET 분석을 위한 모델의 기본적인 형태는 (그림 1)과 같다. 이 모델은 프로그램의 코드를 바탕으로 작성되며 컴포넌트 내부의 행위를 기술하기 위해서 CFG(Control Flow Graph)를 사용한다. CFG의 Node(n_i)는 Atomic block을 나타내고 Edge(e_i)는 control flow를 나타낸다. Atomic block은 내부에 분기가 없고 명령어 구문들의 시간을 더함으로써 전체 block의 시간을 구할 수 있기 때문에 더 이상 작게 나눌 필요가 없는 코드 block을 의미한다. (WCET 분석 과정에서 발생할 수 있는 컴파일러 최적화에 의한 문제들을 최소화하기 위한 단위)



(그림 1) WCET 분석을 위한 모델

컴포넌트의 인터페이스는 기본적으로 4가지 종류로 분류되고 적어도 하나 이상의 인터페이스를 가져야 한다. 인터페이스의 종류는 다음과 같다.

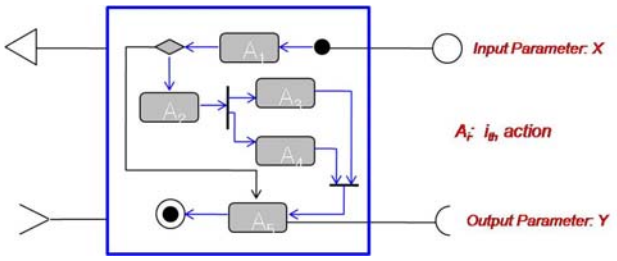
- Provided interface: 컴포넌트가 다른 컴포넌트에게 제공하는 서비스
- Required interface: 컴포넌트가 다른 컴포넌트에서 제공받는 서비스
- System provided interface (system output): 컴포넌트가 시스템 외부로 제공하는 서비스
- System required interface (system input): 컴포넌트가 시스템 외부로부터 제공받는 서비스

이 모델은 각각의 Node와 Edge 마다 성능 속성들을 기술한다.

- Per-node annotation property: Atomic block(n_i)의 순수한 처리 속도, 접근되는 메모리 블록 주소.
- Per-edge annotation property: Atomic block들 사이에서 파이프라인의 영향에 의해 줄어드는 싸이클(cycle) 수, 반복 횟수, 분기 조건.

2.2 컴포넌트 모델

컴포넌트 모델의 기본적인 형태는 (그림 2)과 같다. 컴포넌트의 인터페이스 종류는 Chapter 2.1에서 설명한 것과 동일하다. 또한 각각의 인터페이스는 태스크 모델에서 필요한 속성(asynchronous, iteration number, event number, data value 등)을 추가하여 표시할 수 있다.



(그림 2) 컴포넌트 모델

컴포넌트 내부의 행위를 기술하기 위해서 컴포넌트를 구성하는 함수 별로 UML 활동 다이어그램을 작성한다. 활동 다이어그램은 WCET 분석을 위한 모델을 통해 도출할 수 있다. 액션을 나누는 기준은 임계영역 진입과 탈출, 외부 컴포넌트와의 통신, 메모리 할당과 반환과 같은 컴포넌트의 중요 오퍼레이션으로 한다. 또한 액션을 단위로 성능 속성을 기술한다.

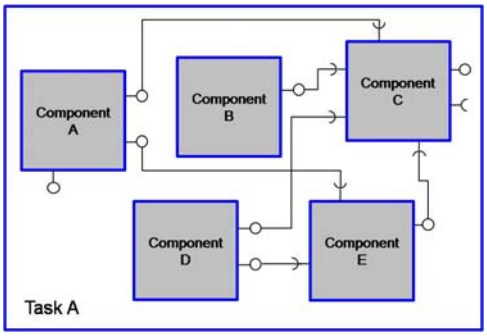
컴포넌트 모델에서 기술되는 성능 속성은 3가지 종류로 분류한다.

- Per-action annotation property: malloc() 함수, free() 함수 등 액션 안에서 불리는 heap 사용량과 관련된 함수들과 그 parameter 값, 액션(Ai)의 처리시간(Latency), 접근되는 메모리 블록 주소.
- Per-interface annotation property: Provide interface의 input parameter type과 개수, 지역 변수의 type과 개수.
- Per-component annotation property: 컴포넌트 전체의 코드 크기, data value의 크기 (read only), 전역 변수의 type과 개수.

2.3 태스크 모델

태스크는 의존 관계가 있는 컴포넌트들의 모음이다. 태스크들 사이에도 태스크들 간의 precedence와 데이터나 이벤트의 흐름과 같은 의존 관계를 가지고 있다면 그것 또한 태스크 모델에서 표현해 준다(periodicity, 선행 태스크 집합(태스크 이름, 수, periodicity, priority) 등). 태스크

의 표현법은 (그림 3)과 같다.



(그림 3) 태스크 모델

3. 모델을 이용한 성능 예측 방법

3.1 컴포넌트 레벨의 처리시간 예측 방법

컴포넌트 레벨의 처리시간은 각각의 내부 함수가 시스템 요소의 영향 없이 실행되는 순수한 시간을 의미한다. 내부 함수의 처리시간을 구하기 위해서는 함수 별로 표현된 활동 다이어그램의 액션의 처리시간을 구하는 과정이 선행되어야 한다.

액션의 처리시간 구하기: 액션의 처리 속도를 구하기 위해서는 먼저 WCET 모델에서 액션으로 묶을 Node와 Edge를 추출한다. 추출된 정보를 바탕으로 본 논문에서는 변형된 타이밍 스키마[8]를 이용하여 처리시간을 계산한다. 변형된 타이밍 스키마는 파이프라인이 처리시간에 미치는 영향을 고려하여 최초의 타이밍 스키마[9]를 변형시킨 형태이다.

변형된 타이밍 스키마는 Atomic block을 Reservation table[10] 형태로 표현하여 Atomic block 사이의 파이프라인 오버랩을 도출한다. 그러나 본 논문에서는 Reservation table은 생략하고 Atomic block 사이의 파이프라인 오버랩에 해당하는 cycle 수를 Edge에 표현해 주었다. 따라서 <표 1>의 식의 계산법은 해석을 달리하여 사용한다.

<표 1> 변형된 타이밍 스키마의 계산식

구분	계산식
S: $S_1;S_2$	$W(S) = W(S_1) \oplus W(S_2)$
S: if(exp) then S_1 else S_2	$W(S) = (W(exp) \oplus W(S_1)) \cup (W(exp) \oplus W(S_2))$
S: while(exp) S_1	$W(S) = (\oplus^{N}_{i=1} (W(exp) \oplus W(S_1))) \oplus W(exp)$

<표 1>의 첫 번째 계산식을 예로 들면, $W(S_1) \oplus W(S_2)$는 Atomic block1의 WCET + (Atomic block2의 WCET - block1과 2 사이의 파이프라인 오버랩 시간) 으로 해석한다. ∪ 기호는 ∪로 연결된 수식 값들 중 최대값을 선택하는 것으로 해석한다. 파이프라인의 오버랩 시간은 Edge의 cycle 수를 바탕으로 실행환경에 맞게 계산하여 사용한다.

활동 다이어그램의 처리시간 구하기: 활동 다이어그램의 처리시간은 다이어그램에 나타난 유효경로를 바탕으로 계산된다. 유효경로는 활동 다이어그램의 경로 중 프로그래머 혹은 모델러가 실행 시 유효하다고 판단한 경로이다. 활동 다이어그램의 시작 노드부터 유효경로를 따라 종료 노드까지 액션의 처리시간들을 합하면 활동 다이어그램의 처리시간을 구할 수 있다. 이 처리시간의 표현식은 다음과 같다:

$$Latency = Latency^{Operation} + Latency^{CS}_{j} + Latency^{FC}$$

- $Latency^{Operation}$: 임계영역과 인터페이스 호출을 제외한 interface의 처리시간
- $Latency^{CS}_{j}$: 임계영역 j의 처리시간
- $Latency^{FC}$: 액션에서 호출되는 다른 함수의 처리시간.

합산된 처리시간은 Provides interface마다 상수식(100, 250) 혹은 함수식(A*F(X) + B)으로 명세된다.

3.2 태스크 레벨의 응답시간 예측 방법

시스템 레벨의 응답시간(Response time)을 간단히 식으로 표현하면 다음과 같다: Response time(R_i) = Execution time(C_i) + Preemption time(I_i) + Blocking time(B_i). 여기에 본 논문에서는 [11]의 방법을 활용하여 문맥교환 시에 캐시 재적재(reload)의 영향으로 발생하는 오버헤드도 고려해준다.

실행시간 예측하기: 태스크 레벨의 실행시간(Execution time)은 각 컴포넌트의 실행 순서에 따른 활동 다이어그램의 처리시간을 합산한 것이 태스크의 실행시간이 된다. 태스크의 시스템 레벨 분석을 위해서 각 태스크의 임계영역 실행시간은 임계영역 별로 각각 명세된다. 또한 캐시 재적재 오버헤드를 구할 때 사용될 메모리 블록 주소들도 태스크와 임계영역 별로 각각 정리한다.

캐시 재적재 오버헤드 구하기: 캐시 재적재 오버헤드는 preempt 당했던 태스크가 다시 실행을 시작할 때 발생하게 된다.

(그림 4) 캐시 재적재 오버헤드

오버헤드를 구하는 단계를 살펴보면, 먼저 각 태스크에서 사용되는 메모리 블록 주소들을 구한다. 그 다음, 메모리 블록이 할당될 캐시 인덱스(최대 인덱스 N)를 고려하여 각 캐시 인덱스 당 메모리 블록 주소의 집합을 구성한다. 예를 들어, 태스크 τ_1의 memory block addresses = {0x000, 0x100, 0x010, 0x110, 0x210}일 때 캐시 인덱스를 고려한 집합(M_1)은 {{0x000, 0x100}, {0x010, 0x110, 0x210}}과 같이 구성된다. 그 후에 선점당하는 태스크와 선점하는 태스크의 메모리 블록 주소의 집합을 비교하여 경쟁하는 최대의 캐시라인 수를 구한다. 구해진 캐시라인 수에 cache miss penalty를 곱한 값이 캐시 재적재 오버

헤드이다. 정리한 식은 아래와 같다:

$$C_{cache}(M_i, M_k) = \sum_{r=0}^{N-1} (\min\{|M_{i,r}|, |M_{k,r}|, L\} \times C_{miss})$$

· |Mi,r|: 태스크 i에서 캐시 인덱스가 r인 메모리 블록 주소 집합의 element 개수

· L: Set associative cache에서 way 수

응답시간 예측하기: 응답시간을 예측하기 위해서 우리는 몇 가지 사항을 가정한다. 첫째, 성능 예측 환경은 싱글 프로세서이다. 둘째, 태스크의 스케줄링 정책은 DM(Deadline monotonic)방식을 사용한다. 따라서 태스크(τ_i)의 우선순위, 주기(T_i)와 데드라인(D_i)은 주어진다. 셋째, 문맥교환 시간, 스케줄링 오버헤드는 미리 알려진 값을 사용하거나 충분히 작다고 가정하여 무시한다. 넷째, 임계영역의 처리를 위해서 PCP(Priority Ceiling Protocol)[7]을 이용한다. 다섯째, 태스크들의 arrival time(AT_i)은 알려져 있다고 가정한다.

태스크의 응답시간을 계산하는 기본식은 다음과 같다:

$$R_i = C_i + B_i + \sum_{\forall k \in hp_i} \left\lceil \frac{R_i}{T_k} \right\rceil \times (C_k + C_{cache}(M_i, M_k))$$

$$FT_i = R_i + AT_i$$

· hp_i: i보다 높은 우선순위

· FTi: 태스크 i의 완료시간

위의 수식에 따르면 R값은 recursive value이기 때문에 반복적인 계산을 통해서 최종 값을 구할 수 있다. 반복적인 계산이 끝나는 시점은 반복 계산을 통해 나온 Response time의 값이 고정되거나 R>D가 되는 시점이다.

기본식만을 가지고는 태스크 사이의 precedence를 고려한 응답시간을 구하기 힘들다. (그림 5)는 태스크의 응답시간을 구할 때, 태스크 사이의 precedence를 고려하는 방법이다. 선행 태스크들에 대한 정보는 태스크 다이어그램에 명시된 정보를 이용한다. 이 방법을 고려하여 스케줄링 정책, 자원 할당 그리고 임계영역 동기화 알고리즘을 구체화 시키면 응답시간 예측 알고리즘이 완성된다.

```
while (true)
    for each task in a arrival time order
        if τiP_FTiP > τi_ ATi then
            if τi preempts τiP then
                Ri = Ci + Bi + Ii
            else
                Ri = (Ci + Bi + Ii)+ (τiP_FTiP - τi_ ATi)
        else
            Ri = Ci + Bi + Ii
        FTi = Ri + ATi
    endfor
    if( the target task's finish time has been determined) then
      BREAK;
endwhile          ※τiP는 τi 의 선행 태스크 set의 어떤 element
```

(그림 5) 태스크 간의 precedence 고려

4. 결론

본 논문에서 우리는 성능 속성을 고려한 모델들과 모델을 이용한 성능 예측 방법을 제시하였다. 하지만 성능 예측 방법에서 메모리 사용량 예측은 제외되었다. 또한 예측 환경을 싱글 프로세서로 한정하였다.

앞으로 우리는 메모리 성능 예측 방법을 제안할 것이다. 또한 멀티프로세서 환경의 성능 예측 방법도 제안 할 예정이다.

참고문헌

[1] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee, "The Koala component model for consumer electronics software," in the proceeding of IEEE Computer, pp. 78--85, March 2000.

[2] Kung-Kiu Lau and Zheng Wang, "A Survey of Software Component Models", (second edition) Pre-print CSPP-38, School of Computer Science, The University of Manchester, May 2006

[3] J. Ivers, N. Shinha and K.C Wallnau, "A Basis for Composition Language CL", Technical Report CMU/SEI -2002-TN-026,CMU SEI, 2002.

[4] OMG UML homepage: http://www.uml.org/

[5] K. Wallnau, "Volume III: A Technology for Predictable Assembly From Certifiable Components.", CMU/SEI-2003-TR-009. <www.sei.cmu.edu/publications /documents/ 03.reports/03tr009.html>

[6] Space4U public homepage: http://www..extra.research.philips.com/euprojects/space4u

[7] Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," in the proceeding of IEEE Transactions on Computers, September 1990.

[8] S. Lim, Y. H. Bae, G. T and Jang, B .et al., "An Accurate Worst Case Timing Analysis Technique for RISC Processors," in the proceeding of IEEE Real-Time Systems Symposium, pp. 97--108, December 1994.

[9] A. Shaw, "Reasoning about time in higher-level language software," in the proceeding of IEEE Transactions on Software Engineering, July 1989.

[10] P.M. Kogge, "The Architecture of Pipelined Computers", Hemisphere Publishing Corp., 1981.

[11] Yudong Tan, Vincent Mooney, "Timing analysis for preemptive multitasking real-time systems with caches," in the proceeding of ACM Transactions on Embedded Computing Systems (TECS), February 2007.