멀티프로세서 임베디드 소프트웨어를 위한 컴포넌트 기반 설계 및 성능분석 방법

이선우*, 정병관*, 유민수**
*한양대학교 전자컴퓨터통신공학과
**한양대학교 정보통신학과
e-mail : {swlee, bkjung, msryu}@rtcc.hanyang.ac.kr

Component Based Design and Performance Analysis for Multiprocessor Embedded Real-Time Software

Sunwoo Lee, Byung Kwan Jung, Minsoo Ryu
Dept. of Electonic, Computer and Communication Science, Hanyang University

요 약

현재까지 소프트웨어 개발을 위한 많은 컴포넌트 기술들이 연구되어 왔다. 하지만 기존의 기술들은 멀티프로세서 환경에서 사용하기에 적합하지 못하다. 멀티프로세서 임베디드 소프트웨어를 구성하는 다수의 쓰레드들이 병렬적으로 수행될 수 있고, 그로 인한 성능 문제 또한 고려되어야 하기 때문이다. 본 논문에서는 새로운 컴포넌트 모델과 태스크 모델, 그리고 소프트웨어 설계 과정을 제안한다. 그리고 컴포넌트 모델과 태스크 모델 사이의 변환 과정 및 병렬성 극대화를 위한 태스크 분할 과정을 소개하여, 최종적으로 성능분석이 가능한 멀티프로세서 임베디드 소프트웨어 개발 과정을 제시한다.

1. 서론

최근 높은 계산 능력을 요구하는 임베디드 시스템이 증가하면서 멀티프로세서가 많이 사용되고 있다. 이러한 추세는 소형 임베디드 장치에서 산업용 자동화 시스템과 같은 대형 장치까지 폭넓게 진행되고 있다. 하지만 멀티프로세서에서 소프트웨어가 효율적으로 실행되기 위해서는 다양한 성능 문제가 고려되어야 한다.

멀티프로세서 임베디드 소프트웨어는 복잡도가 높고, 그 성능이 병렬성에 의해 크게 영향을 받는다. 임베디드 소프트웨어의 높은 복잡도는 멀티프로세서를 사용하기 이전부터 대두되었던 문제로서, 이미 많은 연구가 진행되어 왔다. 그 중에서 CBSD (component based software development)가 적절한 해결책으로 사용되고 있다. CBSD 는 기본적으로 “divide and conquer”와 “systematic reuse”방식의 컴포넌트 기술로 볼 수 있는데, 이는 복잡도가 높은 소프트웨어를 좀 더 빠르고 쉽게 개발 할 수 있도록 한다. 대표적인 CBSD 기술로 CCM (CORBA component model), EJB, DCOM, Koala [8, 14, 15, 1] 등이 있다.

최근에는 CBSD 에서 성능에 관련된 측면이 부각되기 시작했다. Bondarev 는 [3, 2]에서 컴포넌트 기반 소프트웨어의 성능 예측 문제를 소개한다. 이들은 어플리케이션 시나리오에서 태스크를 찾아내고, Robocup 컴포넌트 모델을 사용하여 런타임의 응답시간을 예측하였다. SEI [10, 6]의 PACC (predictable assembly from certifiable components)팀 역시 유사한 방법을 제시한다. 이들은 성능 예측을 위한 컴포넌트 모델을 고안하고, 이를 사용한 소프트웨어 성능 예측 과정을 제안한다. Raghavan 은 use case 를 사용하여 문제를 접근하였으며, 각 use case 를 job 이라는 컴포넌트 단위로 분할한다. 그리고 Queueing Model Simulation [12]을 통해서 그 성능을 예측한다.

하지만 이러한 기존의 컴포넌트 기술들은 멀티프로세서 임베디드 소프트웨어가 가지는 특성을 고려하지 못한다. 멀티프로세서 소프트웨어는 병렬성의 형태와 그 정도에 따라 성능이 크게 좌우된다. 또한 병렬수행되는 태스크들은 서로 영향을 끼치게 되며 성능 분석이 매우 어려워진다.

본 논문에서는 새로운 컴포넌트 모델과 태스크 모델, 그리고 모델 기반의 멀티프로세서 임베디드 소프트웨어 디자인 방법을 제시한다. 제안하는 컴포넌트 모델은 RT-Component (real time component)라고 부르며 이는 UML 의 activity diagram 을 사용하여 소프트웨어의 구조와 성능정보를 표현한다. RT-Component 들은 함수 호출관계에 따라 연결되어 하나의 어셈블리로 결합되고, 이는 행위정보에 의해 RT-Task (real time task)들로 다시 분할된다. 본 논문은 이러한 설계 모델 및 변환 과정을 통해서 소프트웨어의 병렬성을 극대화하는 방법과 실시간 스케쥴링 정책에 따른 성능

분석 방법을 제시한다. 본 논문의 구성은 다음과 같다. 2 장에서 RT-Component 모델과 컴포넌트 어셈블리에 대해 소개한다. 3 장에서는 RT-Task 모델로의 변환 과정을, 그리고 병렬성을 극대화하기 위한 분할 과정을 언급하며 4 장에서는 성능 분석을 위한 수식을 설명한다. 마지막으로 5 장에서는 전체적인 요약과 함께 논문을 결론짓는다.

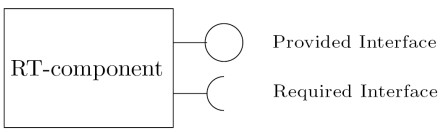
2. RT-Component 모델과 컴포넌트 어셈블리

소프트웨어의 구조는 Philippe Kruchten[7]의 "4+1"관점과 같이 다양한 측면에서 묘사될 수 있다. "4+1"관점중에 하나인 logical and development views 는 주로 기능적인 측면에서 소프트웨어의 요구사항 및 디자인, 그리고 그 구조에 초점을 맞춘다. 그에 비해 process view 에서는 프로세스나 태스크의 수행에 중점을 두며, 이는 소프트웨어의 성능이나 유효성과 같이 기능과 관련되지 않은 측면을 설명한다. 기존의 CBSD 기술들은 주로 logical and development views 에 기반하고 있으며, 컴포넌트를 구조적인 단위로 정의한다.

이러한 정의는 멀티프로세서 임베디드 소프트웨어를 설계하기에 적합하지 못하다. 때문에, 기존의 컴포넌트 모델을 process view 로 확장시켜 정의하는 것이 필요하다. 본 절에서는 새로운 컴포넌트 모델인 RT-Component 를 설명하며, 기존의 구조적인 측면 이외에도 행위적인 측면까지 두 가지로 정의된다.

2.1 RT-Component 의 구조 모델

구조적 측면에서의 RT-Component 는 소프트웨어에 구현된 함수들과 그 구조적 특성들을 통해서 정의된다. 그리고 외부 환경과의 상호작용을 위한 두 가지 인터페이스를 제공하는데, 그림 1 과 같이 표현한다. 각 Provided interface 는 컴포넌트 내부의 함수를 의미하고 Required interface 는 외부 컴포넌트나 소프트웨어 플랫폼에 함수 호출을 요청할 때 사용된다.



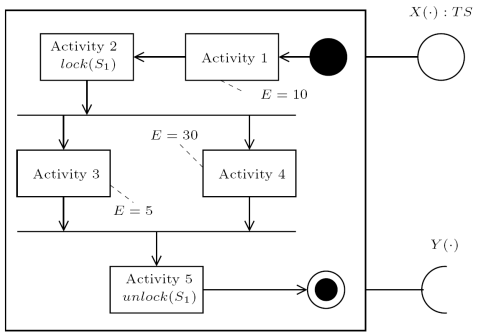
(그림 1) RT-Component 의 구조 모델

2.2 RT-Component 의 행위 모델

RT-Component 의 목적은 소프트웨어의 병렬성 및 성능에 영향을 줄 수 있는 행위정보를 찾아내어 모델링 하는 것이다. RT-Component 의 행위 모델은 UML 의 activity diagram 을 사용하여 행위정보를 표현한다. UML 2.0 은 state diagram 이나 sequence diagram 과 같은 여러 행위 모델을 제공하지만, activity diagram 이 본 논문의 컴포넌트 모델에서 사용되기에 가장 적합하다.

Activity diagram 은 다른 행위 모델에 비해 다음과 같은 장점들을 가진다. 우선 activity diagram 은 소프트웨어 내부의 수행절차를 표현할 수 있다. 이는 대부분의 임베디드 소프트웨어가 C 와 같은 절차적 언어로 작성되고 있음을 감안했을 때 큰 장점이 될 수 있다. 또 다른 장점은 activity diagram 의 fork 와 join 노드를 사용하여 소프트웨어의 병렬성을 표현할 수 있다는 것이다. 이는 병렬성을 극대화 시키기 위한, 이후의 목적에 부합되는, 매우 중요한 장점이다.

RT-Component 의 행위모델은 그림 2 와 같이 표현된다. 컴포넌트 내부의 함수 하나는 activity diagram 하나로 그려지며, fork, join 노드와 decision 노드, merge 노드 등을 사용하여 병렬성 및 조건 수행등을 표현할 수 있다.



(그림 2) RT-Component 의 행위 모델

RT-Component 의 activity diagram 은 UML 의 activity diagram 에 존재하지 않는 몇 가지 성능 속성을 표현한다. 성능 속성은 도착시간, 수행시간, 주기, 종료시한으로 구성되며 다음과 같이 정의한다.

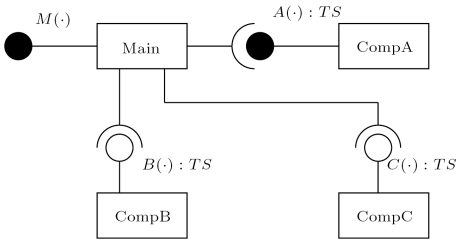
- 도착시간 R 은 함수의 수행이 시작되는 가장 이른 시간으로 정의한다.
- 수행시간 E 는 함수가 선점당하지 않고 수행되는데 요구되는 CPU 시간으로 정의한다.
- 주기 P 는 주기적인 함수의 연속한 두 번의 수행 사이에 걸리는 시간간격으로 정의한다.
- 종료시한 D 는 함수의 수행이 완료되어야 하는 한계 시간으로 정의한다.

또한 각 함수들은 호출 타입을 가진다. 만약 함수가 쓰레드 호출을 허용한다면 해당 인터페이스에 TS (thread safe)와 같이 표시하고, TS 표시가 없는 함수는 쓰레드 호출이 허용되지 않는 것으로 간주한다. 쓰레드 호출이 허용되기 위해서는 공유자원을 접근하기위한 동기화 메커니즘이 제공되어야 한다.

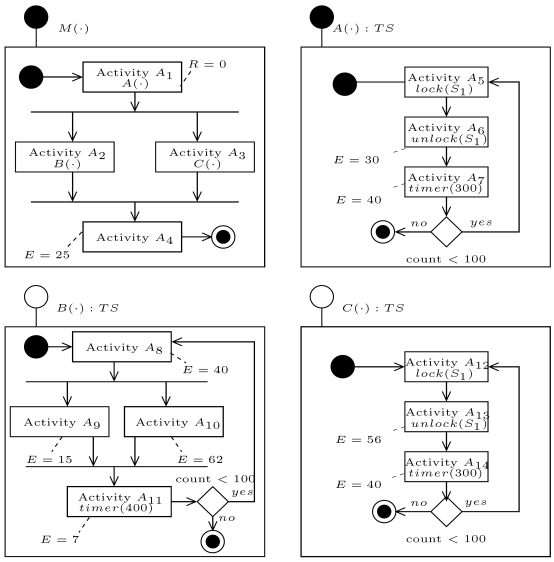
2.3 컴포넌트 어셈블리

컴포넌트 어셈블리는 RT-Component 들의 호출관계를 표현한 연결도이다. 모든 required interface 와 해당 provided interface 를 연결시키면 소프트웨어의 호출관계에 따른 하나의 큰 컴포넌트 어셈블리를 구성하게 된다. 어셈블리 모델을 작성할 때는 임의의 함수 호출을 쓰레드 호출로 결정할 수 있다. 이 때는 검정색 원을 사용하여 인터페이스를 표현하며, TS 타입의 함수만을 쓰레드로 호출할 수 있다. 그리고 동일한 함수를 다수의 쓰레드로 호출할 수 있으므로 해당 인터페이스에 쓰레드 숫자도 표기한다. 그림 3 은 컴포넌

트 어셈블리의 외형적인 구조를 표현한 그림이며, 그림 4 는 컴포넌트 내부의 activity diagram 들을 각각 표현한 예이다.



(그림 3) 컴포넌트 어셈블리



(그림 4) 컴포넌트 어셈블리의 내부 activity diagram

3. 병렬성과 동적인 행위정보를 표현하는 소프트웨어 디자인 방법

본 절에서는 정적인 구조정보를 가지는 컴포넌트 어셈블리를 통적인 행위정보를 가지는 RT-Task 모델로 변환하는 방법을 소개한다. 변환 과정은 두 단계에 걸쳐서 진행된다. 우선 컴포넌트 어셈블리는 RT-Task (real time task) 들로 연결된 간단한 그래프로 변환된다. 그리고 각 태스크들은 내부의 잠재적인 병렬성을 극대화 시키기 위해 쓰레드 수준에서 분할 또는 결합된다.

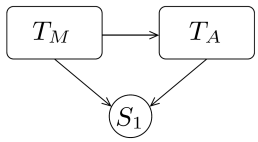
3.1 RT-Component 에서 RT-Task 로의 매핑

컴포넌트는 정적인 소프트웨어 구조로 정의되는 반면, 태스크는 동적인 실행 단위로 정의된다. 실행 단위는 컴포넌트 내부의 쓰레드 호출을 의미하며, 검정색 원으로 표현된 인터페이스를 기준으로 컴포넌트 어셈블리를 분할하면 태스크들을 얻을 수 있다. 이 과정은 정적인 구조 정보를 포함하던 컴포넌트 모델에서 동적인 실행 정보를 표현하는 태스크 모델로의 변환을 의미한다.

위처럼 컴포넌트 어셈블리를 분할한 태스크를 RT-Task 라고 하며 각 RT-Task 들은 RT-Component 의 성능 속성을 그대로 유지한다. RT-Task 의 정의는 다음과 같다.

- T = {T_1,...,T_n} 는 RT-task들의 집합으로 정의한다.
- S = {S_1,...,S_m} 는 공유자원들의 집합으로 정의한다.
- E ⊆ (T x T) ∪ (T x S) 는 RT-task 들의 연결 또는 RT-task 와 공유자원 사이의 연결을 의미한다. 예를 들어 T_i → T_j 는 T_i 가 T_j 와의 통신을 초기화 한다는 것을 나타내고, T_i → S_k 는 T_i 가 공유자원 S_k 를 접근한다는 것을 의미한다

그림 5 는 그림 3 의 컴포넌트 어셈블리가 RT-Task 들로 분할되어 TCG (task communication graph)를 구성한 모습이다.



(그림 5) TCG (task communication graph)

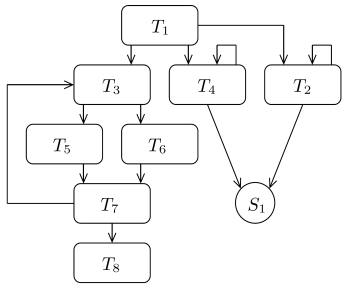
그림 5 의 T_M 과 T_A 는 각각 함수 M(.)과 A(.)의 쓰레드 호출을 의미한다.

3.2 쓰레드 수준의 병렬성 탐색 및 극대화

컴포넌트 어셈블리에서 바로 변환된 TCG 에는 잠재적인 병렬성이 존재한다. Thread-safe 한 함수들, 그리고 fork, join 노드를 사용한 activity 구조는 잠재적인 병렬성을 가지며, 그림 4 를 예로 설명할 수 있다. Main 컴포넌트는 내부의 thread-safe 한 함수 A(.), B(.), C(.)들을 호출한다. 이 중 A(.)는 이미 쓰레드로 호출되지만, 나머지 B(.)와 C(.)는 단순한 함수호출이므로 추가적인 쓰레드 호출이 가능하다. 또한 B(.) 내부의 fork, join 노드로 구성된 activity 들은 세 개의 추가적인 쓰레드를 호출하여, Activity 9, Activity 10, 그리고 join 후의 Activity 11 를 수행하는 것이 가능하다. 이렇게 병렬수행이 가능한 부분을 탐색하여 극대화 시키면 그림 6 과 같이 확장된 TCG 를 얻을 수 있다.

4. 실시간 스케쥴링 이론에 기반한 성능 분석

본 절에서는 응답시간 (response time)과 종료시간 (completion time) 두 가지 성능을 정의하여 그 분석 방법을 설명한다. 응답시간은 주기적인 태스크의 job 이 한 번 수행하는데 걸리는 시간으로 정의하며, 종료시간은 해당 job 의 수행이 완료되는 시간으로 정의한다.



(그림 6) 확장된 TCG

또한 본 논문의 성능 분석은, 태스크 사이의 선후관계 및 동기화를 고려한다. 성능분석을 위해서 다음과 같은 사항들을 가정한다.

- 쓰레드는 주기적이거나 비주기적인 구조를 가진다.
- 쓰레드는 partitioned 스케쥴링 정책 및 고정 우선순위를 사용한다.
- 프로세서의 개수는 고정된 숫자로 가정한다.
- 쓰레드는 MPCP (multiprocessor priority ceiling protocol) [13]을 사용하여 동기화 한다.

위의 가정하에, [11]의 연구를 확장하여 다음과 같은 수식들을 얻을 수 있으다. TCG 의 성능정보를 수식 (1-3)에 대입하여 응답시간을 계산할 수 있다.

$$R_{i,j} = \max\{F_{k,j} \mid T_k \in S_i^{pred}\} \quad (1)$$

$$F_{i,j} = R_{i,j} + E_i + I_i + B_i \quad (2)$$

$$RT_{i,j} = \max\{F_{i,j} - R_{i,j}\} \quad (3)$$

$R_{i,j}$ 는 $T_{i,j}$ 의 도착시간을, S_i^{pred} 는 T_i 의 predecessor 들의 집합을 가리킨다. 그리고 E_i 는 $T_{i,j}$ 의 수행시간을, I_i 는 같은 프로세서에서 수행되면서 $T_{i,j}$ 보다 우선순위가 높은 태스크들에 의해 선점 당하는 시간을, B_i 는 $T_{i,j}$ 보다 낮은 우선순위를 가지는 태스크들의 critical section 중 가장 긴 것을 가리킨다. $RT_{i,j}$ 는 $T_{i,j}$ 의 응답시간을 의미한다.

본 논문은 멀티프로세서 기반의 MPCore 및 EBboard 환경에서 소프트웨어의 성능을 분석하여 제안하는 방법을 검증하였다. 3D ray tracer 프로그램인 Tachyon[5]을 모델링하여 성능을 분석한 결과 표 1 과 같은 결과를 도출하였다. 모델 분석과정 및 모델링 결과는 생략하도록 한다.

(표 1) Tachyon 성능 분석

	Min (usec)	Ave (usec)	Max (usec)
예측치	3547336	3564092	3573455
측정치	2799167	3031483	3377152

5. 결론

본 논문은 새로운 컴포넌트 모델과 태스크 모델을 제안하여, 멀티프로세서 임베디드 소프트웨어를 위한 모델 기반의 디자인 방법을 소개했다. 이는 소프트웨어의 병렬성을 극대화 하고 그 성능을 분석할 수 있는 새로운 디자인 방법으로서, 실시간 스케쥴링 정책에 의존적인 성능분석 문제를 수식으로 정형화 하고 그 해결책을 제시하였다.

참고문헌

[1] C. Atkinson, B. Paech, J. Reinhold, and T. Sander. Developing and applying component-based model-driven architectures in kobra. In *Proceedings of International Symposium on Component-based Software Engineering*, 2004.

[2] E. Bondarev, P. de With, and M. Chaudron. Towards predicting real-time properties of a component assembly. In *Proceeding of the 30th EUROMICRO Conference*, 2004.

[3] E. Bondarev, J. Muskens, P. D. With, M. Chaudron, and J. Lukkien. Predicting real-time properties of component assemblies: a scenario-simulation approach. In *Proceedings of the 30th EUROMICRO Conference*, 2004.

[4] R. Eshuis and R. Wieringa. Tool support for verifying uml activity diagrams. *IEEE Transactions on Software Engineering*, 30(7):437–447, 2004.

[5] John Edward Stone, An Efficient Library for Parallel Ray Tracing And Animation. University of Missouri, 1998

[6] J. Ivers and G. A. Moreno. PACC Starter Kit: Developing software with predictable behavior. In *Proceeding of International Conference on Software Engineering*, 2008.

[7] P. Kruchten. Architectural blueprints–the 4+1 view model of architecture. *IEEE Software*, 12(6):42–50, 1995.

[8] K.-K. Lau and Z. Wang. A taxonomy of software component models. In *Proceedings of EUROMICRO Conference on Software Engineering and Advanced Applications*, 2005.

[9] G. Liao, E. R. Altman, V. K. Agarwal, and G. Gao. A comparative study of multiprocessor list scheduling heuristics. In *Proceedings of the Twenty-Seventh Hawaii Internation Conference on System Sciences*, 1994.

[10] P. Merson and S. Hissam. Predictability by construction. In *Proceeding of Conference on Object Oriented Programming Systems Languages and Applications*, 2005.

[11] G. Raghavan, A. Salomaki, and R. Lencevicius. Model based estimation and verification of mobile device performance. In *Proceedings of the 4th ACM international conference on Embedded software*, 2004.

[12] R. Rajkumar and J. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proceedings of the IEEE Real-Time Systems Symposium*, 1988.

[13] J. Sun and J. W. Liu. Bounding the end-to-end response time in multiprocessor real-time systems. In *Proceedings of Workshop on Parallel and Distributed Real Time Systems*, 1995.

[14] C. Szyperski. Component technology - what, where, and how? In *Proceedings of International Conference on Software Engineering*, 2003.

[15] R. C. van Ommering. Koala, a component model for consumer electronics product software. In *Proceedings of the Second International ESPRIT ARES Workshop on Development and Evolution of Software Architectures for Product Families*, 1998.