저전력 임베디드 소프트웨어 개발을 위한 재사용 컴포넌트의 전력소모 특성 명세 방법†

(Energy Characteristic Specification Method of Reusable Component for Energy Efficient Embedded Software Development)

김두환‡ 이재욱§ 홍장의¶
(Doohwan Kim) (Jae-Wuk Lee) (Jang-Eui Hong)

요 약 컴포넌트 기반 개발 방법론은 개발 편의성, 개발에 소요되는 시간, 비용 등의 이점으로 인해 그 적용 영역이 확대되고 있다. 특히 일반적인 소프트웨어에 비해 플랫폼 의존성이 높고, 제품 군(product family) 개발이 가능한 임베디드 소프트웨어의 경우 컴포넌트 기반 개발은 그 유용성이 매우 높다고 할 수 있다. 그 중에서도 스마트 폰, 태블릿 PC 등과 같은 휴대형 임베디드 시스템의 경우, 전력 소모량이 적은 소프트웨어 개발의 중요성이 부각되고 있기 때문에 재사용 컴포넌트의 소모 전력에 대한 특성은 컴포넌트 기반 개발에서 고려되어야 할 중요한 품질요소 중의 하나가 되었다. 본 연구에서는 재사용 기반의 임베디드 소프트웨어 개발 시, 컴포넌트에 대한 소모전력 특성을 고려할 수 있도록 지원하기 위하여 소모전력 특성을 포함하는 컴포넌트 명세언어를 제안한다. 이는 추후 컴포넌트 저장소에서 소모전력 특성을 고려하는 컴포넌트 검색 및 선택 등과 같은 영역에서 활용할 수 있다.

키워드 컴포넌트, 컴포넌트 명세, 전력 분석

Abstract Component-based Software development(CBSD) is widely used in various area due to its efficiency of time, cost and effort. In the embedded software which has high dependency of platform and can be developed by product family, the efficiency of CBSD is maximized by reuse. These embedded software has various limitations of the resources. Specially, the effective energy consumption is very important in the portable embedded software such as smart phone and tablet PC, because they are operated with limited energy source like a battery. Therefore, energy efficient problem became very important issue in the CBSD. In this paper, we identified characteristics and environment that influence energy consumption of components. Afterward, we defined a component specification language which is consisted to describe energy characteristics of the components. This supposed specification language can be utilized to energy efficient component search and selection.

Key words Component. Component Specification, Energy analysis

1. 서 론

컴포넌트 기반 소프트웨어 개발 방법론은 기존에 개발된 재사용 컴포넌트들을 이용하여 소프트웨어를 개발하는 방법으로 개발 편의성 향상, 비용절감, 시간단축 등의 이점을 제공한다. 특히 임베디드 소프트웨어 개발에 있어서는 제품 간에 유사한 기능을 포함하는 경우가 많으며, 개발

† 이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임(2011 0010396)
‡ 학생회원 : 충북대학교 전자정보대학 컴퓨터과학과
dhkim@selab.cbnu.ac.kr
§ 학생회원 : 충북대학교 전자정보대학 컴퓨터과학과
jwlee@selab.cbnu.ac.kr
¶ 종신회원 : 충북대학교 전자정보대학 컴퓨터과학과 부교수
jehong@chungbuk.ac.kr
논문접수 : 2011년 06월 02일
심사완료 : 2004년 06월 24일

기간이 다소 짧다는 특성 등으로 인해 컴포넌트 기반 개발의 도입이 지속적으로 늘어나고 있다. 이러한 임베디드 소프트웨어는 일반적으로 가용 전력량, 메모리 등과 같은 자원에 대한 제약을 갖는다[1]. 이와 같은 특성들을 통해 임베디드 소프트웨어 개발 시에는 신속한 의사결정과 다양한 비기능적 요소들에 대한 고려가 필수적으로 요구된다. 특히 휴대형 임베디드 시스템은 배터리를 통해 제한된 전력을 공급받기 때문에 소모 전력량에 대한 비기능적 요구사항이 중요하다. 저전력 소모를 위한 소프트웨어 개발에서 우선적으로 수행해야 하는 것은 소프트웨어의 정량적 소모전력 분석이며, 일반적으로 소프트웨어의 소모 전력량 분석은 그것의 행위를 기술하는 모델을 이용하여 수행 가능하다. 이는 소프트웨어가 단독적으로 전력 소모를 발생시키는 것이 아니라, 그것이 실행 될 때 하드웨어의 동작을 유발시킴으로써 간접적인 전력 소모를 일으키기 때문이다[2]. 다시 말해 소프트웨어의 단위 행위는 그에 해당하는 일련의 하드웨어 동작을 나타내고, 이때 하드웨어의 동작을 통해 전력 소모가 발생된다.

소프트웨어 컴포넌트는 기능적 행위를 포함하는 구현된 코드이다. 컴포넌트 기반으로 임베디드 소프트웨어를 개발하기 위해서는 가급적 동일 기능을 갖지만 소모전력량이 적은 컴포넌트를 개발하거나 또는 재사용 컴포넌트를 선택할 필요성이 있다. 따라서 재사용 컴포넌트의 명세에 컴포넌트가 소모할 전력량을 포함하도록 명세하는 것이 필요하다.

컴포넌트 명세는 해당 컴포넌트의 정보를 제공하여 그것의 재사용을 지원하도록 하는 역할을 수행한다. 즉, 컴포넌트 명세는 컴포넌트의 이용, 검색 등에 있어서 아주 중요한 일종의 이력서와도 같다. 기존의 컴포넌트 명세에 있어서는 일반적으로 컴포넌트가 제공하는 기능과 이들 기능에 대한 정형적 인터페이스 명세만이 중요한 고려사항이 되었다[3]. 그러나 이제는 앞서 설명한 바와 같이 소모전력 특성을 컴포넌트 명세에 반영해야 한다. 소프트웨어의 소모 전력은 그것의 행위와 밀접한 관계를 갖는다. 따라서 컴포넌트 명세를 이용한 전력소모 특성 명세를 위해서는 각 컴포넌트에서 제공하는 인터페이스에 따라 관리하는 것이 합당하다.

본 연구는 컴포넌트 기반의 소모 전력량 분석을 지원하기 위해 컴포넌트 명세에서 각 컴포넌트에 대한 전력 소모 특성을 기술하기 위한 방법을 제시한다. 제안된 컴포넌트 명세 방법을 통해 얻을 수 있는 이점은 다음과 같다.

- 전력소모 특성을 고려한 컴포넌트의 체계적 관리 지원 가능
- 전력 효율적인 컴포넌트 선택을 지원하기 위한 가이드라인 제시
- 컴포넌트 기반 소프트웨어 아키텍처 상에서 전력 소모량 분석 지원 가능

본 논문은 다음과 같이 구성된다. 2장에서는 관련연구를 통해 기존의 컴포넌트 명세 기법들에 대해 소개하고, 3장에서는 컴포넌트 명세를 위해 요구되는 정보들을 살펴본 후 본 연구에서 제시하는 컴포넌트 명세 언어를 정의한다. 4장에서는 정의된 명세 언어의 적용을 살펴보고, 5장에서 이에 대한 활용 방안에 대해 논의한다. 마지막으로 6장에서는 결론 및 향후 연구에 대해 언급한다.

2. 관련연구

컴포넌트 기반 소프트웨어 개발을 수행하는 조직에서는 개발에 참여하는 모든 사람이 쉽게 컴포넌트에 접근할 수 있도록 전사적 차원에서 관리되는 컴포넌트 라이브러리가 필요하다.

이때, 라이브러리는 컴포넌트뿐만 아니라 그것의 사용을 지원할 수 있도록 해당 컴포넌트에 대한 정보들을 포함한다. 때문에 컴포넌트에 대한 정보를 효율적으로 관리할 수 있도록 하는 컴포넌트 명세의 구성에 대한 연구들이 진행되어 왔다[4-6]. 각 연구에서 제안하고 있는 컴포넌트 명세 기법은 일반적으로 요구되는 컴포넌트에 대한 정보들이 유사하게 포함되어 있는 반면 해당 개발 조직에서 주로 개발하는 소프트웨어의 도메인이나 특성, 정책 또는 가치에 따라 서로 상이한 정보를 포함하기도 한다. 이러한 연구들의 대표적인 사례는 RAS(Reusable Asset Specification)[4]와 X-ARM[5]이 있다.

RAS는 OMG에서 소프트웨어 컴포넌트뿐 아니라 개발주기 전반에서 발생하는 재사용 가능한 자산에 대한 명세를 지원하기 위해 발표하였다. RAS에서 정의한 컴포넌트 명세는 기본적으로 UML 기반의 소프트웨어 개발을 지원하기 위해 설계되어있으며, 이로 인해 일반적인 컴포넌트를 이용하기 위해 필수적으로 요구되는 정보 외에도 UML을 지원하기 위한 정보들이 다소 포함되어 있어, RAS를 이용한 컴포넌트 명세는 다소 번거로울 수 있다. 때문에 몇몇 컴포넌트 명세 기법들은 RAS를 참고하여 해당 조직에서 필요로 하지 않는 항목을 제거하고, 추가적으로 요구되는 정보들을 추가 정의하여 사용되기도 한다.

X-ARM은 RAS와 OSD(Open Software Description)[7]를 참조하여 컴포넌트와 그에 대한 정보를 효율적으로 저장소에 저장하기 위해 설계되었다. X-ARM에서도 RAS와 같이 MDD(Model Driven Development)를 지원하기 위해 컴포넌트 명세에 “ComponentModel”이라는 정보를 포함하고 있으나, 이는 굳이 UML을 사용하지 않더라도 기술할 수 있도록 하였다. 이와 같이 UML에 대한 사용을 배제하고, 유연하게 사용할 수 있도록 설계함으로써 X-ARM에서 나타내는 컴포넌트 명세는 RAS와 비교하여 확연히 단순한 구조를 갖는다. 다만, X-ARM은 컴포넌트를 저장소에 저장하고 배포하는 등의 이용을 지원하도록 설계하여, 이에 해당하는 정보들이 추가되어 있다.

위와 같은 명세기법들은 비기능적 품질 속성 명세 항목을 정의하고 있으나, 구체적으로 어떠한 속성을 어떻게 명세할 것인지에 대한 정의가 없어 실질적으로 이를 집적이용한 비기능적 송성 명세에는 어려움이 있다. 비기능적 품질 속성에 대한 기술을 수행하고, 컴포넌트 선택 등의 의사결정을 지원하기 위한 명세 기법으로는 L. Iribarne의 연구가 있다[6].

L. Iribarne의 연구는 컴포넌트 명세 기법들이 비기능적 요구사항에 대한 명세 및 이를 통한 의사결정 지원을 위해 XML기반의 비기능적 속성을 표기 하는 방법을 제안하였다. 이때, 비기능적 속성은 NFR[8]을 이용하여 기술하였으며, 컴포넌트에서 나타내는 다양한 비기능적 속성에 대해 기술하고 각각의 속성에 대한 상호 연관관계를 나타내어 컴포넌트 선택에 있어서 의사결정을 지원할 수 있도록 설계되어 있다. 하지만 제시된 기법에서 각각의 품질 속성이 절대치로 평가되기 때문에 컴포넌트간의 상대적 효율성 비교를 평가가 주로 이용되는 속성의 경우에는 제시된 방법으로 명세하기가 쉽지 않다는 단점을 갖는다.

이와 같이 기존의 컴포넌트 명세 기법들은 비기능적 품질 속성을 고려한 의사결정 지원에 적합하지 않거나, 그것을 지원하더라도 각 품질 속성에 대한 평가 등을 수행하기에 부족하다는 단점이 있다. 본 연구에서 제시하는 소모전력 특성 명세 방법은 컴포넌트의 전력소모량이 나타내는 특성에 따라, 컴포넌트가 제공하는 인터페이스별 전력소모량의 가변적 특성을 고려하여 해당 인터페이스에 대한 Macro-model의 형태로 기술한다. 이러한 방식의 명세는 호출된 인터페이스에 따라 달라질 수 있는 소모 전력량을 고려하여, 전력

효율성에 따른 컴포넌트 선택 시 이에 대한 가이드라인을 제시할 수 있다. 또한 컴포넌트 기반 아키텍처 상에서의 전력소모 분석에 대한 지원이 가능하다.

3. 전력소모 특성 명세 방법

3.1 컴포넌트 명세 정보

컴포넌트의 실질적 사용을 지원하기 위해서는 해당 컴포넌트에 대해 기술해야 하는 몇 가지 정보가 있다. 이러한 정보들은 컴포넌트를 식별하고, 기능을 파악하며, 이를 사용하기 위해서는 어떠한 인터페이스를 이용해야 하는지에 대한 정보를 참조할 수 있도록 한다. 이와 관련하여 Michael Gsterfer는 컴포넌트가 재사용되기 위해서는 다음과 같은 사항들이 명세 되어야 한다고 주장하였다[3].

- Operational specification
- Operation context
- Non-functional properties
- Required interface and resources

이러한 정보들은 해당 컴포넌트의 이름과 기능적 특성 등을 명세하고, 이를 이용하기 위해 요구되는 인터페이스 및 해당 컴포넌트에 대한 비기능적 속성을 기술함으로써, 개발자가 컴포넌트를 식별하고 사용할 수 있도록 지원한다. 여기서 비기능적 속성은 다양한 속성들을 포함할 수 있으며, 각각의 성격에 따라 표현 방법 등이 상이할 수 있다. 본 연구에서 명세하고자 하는 전력소모 특성 또한 이 범주에 속하며, 소프트웨어의 전력 소모량은 그것이 나타내고 있는 행위적 특성과 밀접한 관계가 있다는 점에서 일반적인 기술이 어려울 수 있다.

재사용 컴포넌트의 행위를 나타내는 구성요소는 오퍼레이션들이다. 즉, 컴포넌트의 전력소모 특성을 기술하기 위해서는 오퍼레이션들을 이용해야 한다. 이러한 오퍼레이션의 종류에는 해당 컴포넌트에 서비스 요청을 위해 외부로 노출된 오퍼레이션과 해당 서비스를 제공하기 위해 내부적으로 호출되는 오퍼레이션이 있다. 여기서 외부로 노출되는 오퍼레이션들이 인터페이스에 해당하며 각각의 인터페이스들은 서로 다른 행위를 수행한다. 내부적으로 호출되는 오퍼레이션에 대한 제어 흐름을 내포하고 있다. 이와 같은 이유로 컴포넌트의 소모전력 특성 명세는 각 인터페이스별로 수행되어야 한다.

3.2 전력 소모 요소

소프트웨어의 전력 소모는 그것의 운영 환경이 어떻게 구성되어 있는지, 또는 어떠한 컴포넌트가 사용되었는지에 따라 전력 소모량이 크게 달라질 수 있다. 이는 소프트웨어의 운영 환경에 따라 연산 수행에 소요되는 시간 등이 변하는 것과도 같은 이치이다. 그림 1은 컴포넌트 기반 소프트웨어의 전력 소모량에 영향을 미치는 요소들을 중심으로 도시화한 소프트웨어의 운영 환경을 나타낸다.

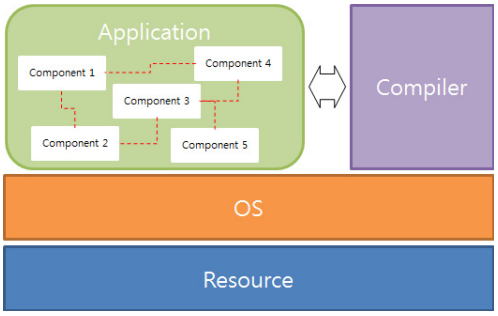

그림 1 컴포넌트 기반 소프트웨어의 운영 환경

3.2.1 Hardware Platform

소프트웨어의 실행을 통해 실질적인 전력소모의 주체는 하드웨어라는 점에서 그것의 구성에

따라서 소모전력 특성이 달라진다. 때문에 저전력 소모를 위한 초기 연구들은 하드웨어의 구성 소자 및 그들간의 구조를 전력 효율적으로 구성하는 것에 초점을 맞춘 연구들이 주로 진행되었다[9-12]. 그러므로 전력 분석의 대상이 하드웨어가 아닌 이를 제어하는 소프트웨어로 변경되었다 하더라도 하드웨어의 구성에 대한 정보가 없다면 이를 이용하기가 쉽지 않다. 특히 CPU의 구성 및 Clock speed에 대한 정보와 메모리의 크기가 중요한 역할을 한다[13, 14]. 따라서 재사용 컴포넌트가 추후 어떠한 하드웨어 플랫폼에서 실행할 것인가에 대한 정보가 요구된다.

3.2.2 운영체제

운영체제는 일반적인 응용 소프트웨어가 실행되기 위해 요구된다. 이는 응용 소프트웨어가 운영되기 위한 환경을 제공하고, 스케쥴링 또는 메모리 관리 등의 작업을 지원한다. 때문에 어떠한 운영체제를 사용하느냐에 따라 idle 상태의 소모 전력량 또는 운영되는 소프트웨어와의 상호작용에서 발생하는 전력량 등이 차이를 보일 수 있다[14]. 때문에 운영체제의 종류와 버전 또한 중요한 전력 소모 요소들 중 하나이다.

3.2.3 Compiler

소프트웨어가 같은 언어를 이용하여 구성되었다고 하더라도, 이에 사용된 Compiler의 종류나 버전에 따라 코드 최적화 정책이 서로 상이하게 적용될 수 있으며, 결과적으로는 동일한 코드를 컴파일 하더라도 컴파일러에 따라 해당 소프트웨어의 소모 전력량이 변화될 수 있다[15]. 때문에 소프트웨어 또는 컴포넌트의 소모 전력량을 명세하기 위해서는Compiler의 종류와 버전 등에 대한 정보가 요구된다.

3.2.4 Interface

그림 1을 살펴보면 “Component 5”를 제외한 모든 컴포넌트가 두 개 이상의 다른 컴포넌트와 상호작용하고 있다. 인터페이스는 이때 각 컴포넌트간의 상호작용이 가능하도록 외부로 노출된 오퍼레이션을 의미한다. 이때 컴포넌트는 최소한 하나 이상의 인터페이스를 제공해야만 한다. 만약 컴포넌트가 제공하는 인터페이스가 하나도 없다면 이는 소프트웨어에서 전혀 사용되지 않음을 의미하며, 필요 없는 코드가 포함되어 있는 것과도 같다. 3.1절에서도 설명한 바와 같이 컴포넌트가 제공하는 각 인터페이스는 서로 다른 행위를 나타낸다. 이는 곧 어떠한 인터페이스가 호출되었느냐에 따라 해당 컴포넌트에서 소모하는 전력량이 달라진다는 것을 의미하며, 이에 대한 정보는 컴포넌트 기반 소모 전력량 분석에서 매우 중요한 역할을 한다.

3.2.5 Input parameter

컴포넌트의 인터페이스는 별도의 파라미터 없이 호출될 수 있는 경우도 있지만, 입력으로 주어지는 파라미터가 요구되는 경우가 일반적이다. 파라미터를 요구하지 않는 인터페이스의 경우에는 항상 동일한 행위를 수행하므로 전력 소모량의 변화가 없지만, 파라미터가 요구되는 경우 주어진 파라미터의 특성에 따라 호출된 인터페이스의 분기, 반복 등의 제어 구조가 변경될 수 있다. 이러한 제어구조의 변화는 파라미터가 다음과 같은 특성을 나타낼 때 발생한다.

- 인터페이스의 분기를 결정하는 Flag로 사용되는 경우
- 파라미터가 가변적 길이를 갖고, 해당 인터페이스는 파라미터의 모든 요소에 어떠한 연산을 반복적으로 수행하는 경우

어떠한 컴포넌트에서 주어진 데이터의 정렬을 수행하는 인터페이스가 “List sort(List data, Boolean asc)” 로 정의되어 있다고 가정하자. 해당 인터페이스의 파라미터 중 “asc” 값이 “true”이면 오름차순 정렬을 수행하고, “false”이면 내림차순 정렬을 수행한다면, 해당 파라미터의 값에 따라 인터페이스의 분기가 어떻게 수행될 지 결정된다. 이러한 경우에는 하나의 인터페이스라도 “asc”와 같은 파라미터가 나타낼 수 있는 값의 종류에 따라 서로 독립된 인터페이스가 제공되는 것과 같이 전력소모 특성을 분류하는 것이 효율적이다. 또한 위의 인터페이스 정의는 “data”는 길이가 가변적이므로 이에 따라 수행해야 할 연산의 개수도 달라지게 된다. 결과적으로 “data”로 주어진 파라미터의 길이에 따라 소모되는 전력량 또한 변화된다.

이와 같이 입력으로 주어지는 파라미터가 해당 인터페이스의 분기, 반복 등의 제어 흐름을 결정하기 위한 요소가 될 때, 이것은 해당 인터페이스의 전력 소모량을 결정하는 중요한 요소가 된다. 그러므로 앞서 언급한 바와 같이 각 인터페이스의 소모 전력량은 일반적인 Scalar 값을 갖는 경우도 있지만(주어진 파라미터들이 단순한 연산에만 이용되며 제어 흐름에 영향을 미치지 않는 경우 또는 파라미터를 요구하지 않는 경우), 입력 파라미터에 따라 가변적인 전력소모량을 보일 경우에는 Macro-model을 이용하여 표현하는 것이 합당하다. 이와 같이 Macro-model을 통해 소프트웨어의 소모 전력량을 나타내는 방식은 T. K. Tan의 연구에서 이미 이용된바 있다[14].

3.3 전력소모 특성을 고려한 컴포넌트 명세 언어

본 절에서는 전력소모 특성을 고려하여 컴포넌트를 명세하기 위한 언어를 정의한다. 이를 위해서는 3.2절에서 식별한 소프트웨어의 소모 전력량에 영향을 미치는 요소들을 컴포넌트 명세 상에서 기술하는 방법에 대한 정의가 우선적으로 요구된다. 표 1은 하나의 컴포넌트를 기준으로 각 요소가 소프트웨어 컴포넌트의 전력 소모량에 대해 영향을 미치는 범위를 나타낸다.

표 1 전력 소모 요소의 영향 범위

구분	범위
Hardware	Component 전체
OS	Component 전체
Compiler	Component 전체
Interface	개별 Interface
Parameter	개별 Interface

3.3.1 전력 소모 요소에 따른 명세 방안

표 1을 살펴보면 전력 소모 요소들이 영향을 미치는 범위는 컴포넌트와 인터페이스로 나누어질 수 있다는 것을 알 수 있다. HW와 OS는 실제로 컴포넌트기반의 소프트웨어가 운영되기 위한 환경을 제공하는 역할을 하므로, 하나의 컴포넌트뿐 아니라 전체 소프트웨어에 동일하게 적용되는 사항이다. 물론 이러한 사항은 분산처리 환경에서 예외가 될 수 있으나, 컴포넌트 명세는 하나의 컴포넌트만을 명세하기 때문에 분산처리 환경에 대한 별도의 고려가 요구되지는 않는다. Compiler는 해당 소프트웨어 또는 Compiler가 실행될 수 있도록 기계어로 변환하는 역할을 수행한다. 여기서 Compile의 범위는 전체 소프트웨어가 될 수도 있으며, 컴포넌트 자체가 될 수 있다. 물론 두 가지 중 어떠한 경우라도, 하나의 컴포넌트가 둘 이상의 서로 다른 Complier로 나누어 Compile되지는 않으므로, 이 또한 전체 컴포넌트에 적용되는 요소이다.

이러한 세가지 항목들은 모두 컴포넌트 전체에 동일한 영향을 미치는 요소들 이였으나, 인터페이스와 입력 파라미터의 경우는 다수의 인터페이스 중 어떠한 인터페이스가 이용되었으며,

파라미터의 구성은 어떻게 되었는지에 따라 서로 다른 결과를 나타낸다. 때문에 이 두 가지 항목에 대해서는 3.2에서 식별한 것과 같이 각 인터페이스별로 독립적인 관리가 요구되거나, 파라미터의 구성에 따른 Macro-model을 이용해야만 한다.

3.3.2 CoSLanE

본 연구에서는 컴포넌트의 전력 소모 특성 정보를 고려한 컴포넌트 명세 언어인 CoSLanE (Component Specification Language for Energy)을 정의하였다. CoSLanE은 일반적으로 컴포넌트 명세에 이용되는 정보들을 바탕으로 하여, 전력 소모 특성을 명세하기 위한 정보들을 추가하여 정의되었다. 그림 2는 BNF[16]를 이용하여 정의한 CoSLanE의 구성을 나타낸다. CoSLanE은 컴포넌트를 명세하기 위해 <name>, <id>, <description>, <mngInf>, <commode>, <resource>, <interface>를 이용한다. 여기서는 전력소모 특성을 나타내기 위해 사용된 항목을 중심으로 설명한다.

(1) <resource>

<resource> 항목은 컴포넌트가 운용되는 환경을 명세하기 위해 사용된다. 본 연구에서는 이를 이용하여 해당 컴포넌트의 Hardware 구성과 OS 및 Compiler에 대한 정보를 나타내기 위해 사용하였다. <hardware>항목을 구성하는 <cpu>는 CPU의 제품 정보를 나타내며, <memory>는 운영환경 상에서 memory의 size를 나타내기 위해 사용된다. <OS>와 <compiler>는 각각 OS와 Compiler의 종류 및 버전을 나타내기 위해 사용된다. 이와 같이 <resource>를 통해 명세되는 정보들은 3.3.1에서도 정의한 바와 같이 해당 컴포넌트 전체에 적용되는 정보들로써 컴포넌트 명세 전체에서 한번만 정의함으로써 이용이 가능하다.

(2) <interface>

<interface>항목은 해당 컴포넌트에서 호출되는 Interface에 대한 전력 소모 특성을 실질적으로 명세한다. 이는 어떠한 인터페이스가 호출되었는지, 어떠한 파라미터를 통해 호출되었는지에 따라 가변적인 소모 전력량을 나타낼 수 있다. 때문에 컴포넌트가 제공하는 모든 인터페이스에 대해 독립적으로 정의해야 한다. <interface>항목은 <id>, <name>, <protoType>, <parameters>, <returnType>, <energyModel>로 구성되어 있다. <id>와 <name>, <protoType>의 항목은 각각 컴포넌트 라이브러리 상에서 관리 될 때를 고려하여 해당 인터페이스의 ID와 이름, 인터페이스의 Signature를 나타내기 위해 정의하였다.

```
<Component> ::= <name><id><description><mngtInf>
        <comModel><resource><interface>
<id> ::= <string>
<description> ::= <String>
<mngInf> ::= <location><version><author><status>
<location> ::= <string>
<version> ::= <version> | <number> | '.'<number>
<author> ::= <string>
<status> ::= <artifactType><ready>
<ready> ::= <bool>
<resource> ::= <hardware><OS><compiler>
<hardware> ::= <cpu><memory>
<cpu> ::= <string>
<memory> ::= <real_number>
<OS> ::= <name><version>
<compiler> ::= <name><version>
<interface> ::= <id><name><protoType><parameters>
            <returnType><energyModel>
<protoType> ::= <returnType>' '<string>
            '('<parameters>')'
<parameters> ::= <parameters> | <parameter>
<parameter> ::= <dtype>' '<string>
<returnType> ::= <void> | <dtype>
<energyModel> ::= <energyModel>
            | <opmode><expression>
            | <expression> |[<map>]
<opmode> ::= <string>
<expression > ::= <expression >+<term >
            | <expression >- <term > | <term >
<term > ::= <term >* <factor > | <term >/ <factor >
        | <factor >
<factor > ::= <real_number>| <variable>
<variable> ::= <name>
<map> ::= <parameter>'->'<factor>
<real_number> ::= <number>'.'<number>
<number> ::= <number> | [0|1|2| ... |9]
<name> ::= <string>
<string> ::= <string> |[a|b|c|...z|A|B|...Z]
        | [_| |-] | <number>
<bool> ::= [true|false]
<artifactType> ::= [model|code|binary]
<dtype> ::= [int|char|double|float|short] | <string>
```

그림 2 CoSLanE의 정의

<parameters>항목은 해당 인터페이스에 이용되는 파라미터들을 나타낸다. 본 명세 언어에서 가장 중요한 역할을 하는 <energyModel>은 주어진 파라미터 값에 따라 분기 구조가 변경되는 경우를 나타내기 위한 <opmode>와 해당 인터페이스의 Macro-Model을 나타내기 위해 사용되는 <expression>을 통해 명세 된다.

<expression>은 <term>을 통해 기술되거나, 재귀적으로 <expression>과 <term>의 산술적 합 또는 차를 통해 기술된다. 각 <term>은 <term>과 <factor>의 산술연산 곱셈 또는 나눗셈을 통한 조합 또는 <factor>만을 이용하여 구성된다. <factor>는 실수를 나타내는 <real_number>또는 해당 factor의 변수 명을 나타내기 위해 <name>을 이용하여 기술된다. <map>항목은 <parameters>에서 정의된 파라미터가 어떠한 <factor>에 해당하는지에 대한 매핑을 수행하기 위해 이용된다.

(3) <interface>를 이용한 전력소모 특성 명세

어떠한 interface A의 전력소모 특성을 표현할 때, 해당 인터페이스에 대한 전력소모 특성이 "2.4x + 314.5"와 같다면 앞서 정의한 CoSLanE의 <energyModel>을 통해 그림 3에서 나타내는 바와 같이 해석될 수 있다.

이와 같이 정의된 CoSLanE를 이용해 각 인터페이스에 대한 전력소모 특성을 기술함으로써 다음과 같은 효과를 얻을 수 있다.

- 해당 컴포넌트의 소모전력 특성을 효율적으로 표현할 수 있다.
- 전력 효율성을 고려한 컴포넌트 선택을 지원할 수 있다.
- 정의된 CoSLanE를 통해 소프트웨어 아키텍처 수준의 소모 전력량 분석을 수행할 수 있다.

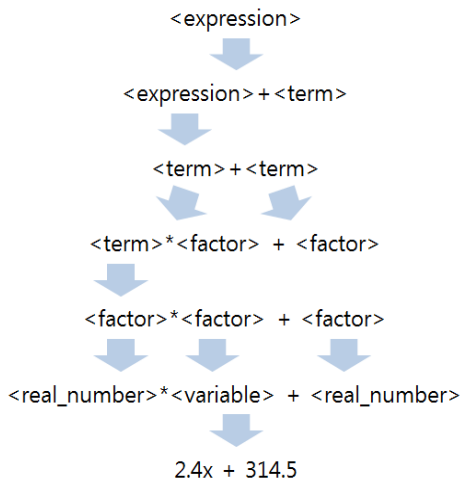

그림 3 CoSLanE을 이용한 전력소모 특성 표현

4. CoSLanE의 적용

3장에서는 컴포넌트의 소모 전력량에 영향을 미치는 요소들을 식별하고, 전력 소모 특성을 기술하기 위한 언어인 CoSLanE을 정의하였다. 본 장에서는 3장에서 정의한 소모전력 특성 명세 방법을 이용하여 전력소모 특성을 나타내는 컴포넌트 명세에 대해 논의하도록 한다. 그림 4와 그림 5는 이를 이용한 컴포넌트 명세의 예시이며 XML 형태로 표현하였다.

그림 4의 "Key"는 AES-SFT[17]의 구성 요소 중 암호화 키를 관리하는 컴포넌트로써, "expandKey()"와 "getRoundKey()"를 인터페이스로 갖는다. 주어진 예시에서 나타나는 두 개의 인터페이스가 모두 파라미터를 요구하지 않으므로, 그들의 전력 소모량은 Scalar 값으로써 각각이 "127681238.519417 (nJ)"과 "22659163.689320(nJ)"을 소모하게 된다. 그림 5는 Linux의 "ipc.h"를 통해 제공되는 인터페이스 중 Message send 등과 같은 함수들에 대한 명세 예시를 나타낸다. "msgctl()"의 경우 flag로 주어진 값에 따라서 어떠한 행위를 할 지에

대한 결정을 행하게 된다. 때문에 opmode 항목에 따라 달라지는 소모 전력량을 기술해준 것을 확인해 볼 수 있으다. “msgsnd()” 함수는 주어진 prarameter 중 “msgsz”의 크기에 따라 소모 전력량이 가변성을 나타내므로, 이에 대한 전력 소모 특성을 “4.0x + 4554.0”과 같이 macro- model로 표현해 준 후, “map”을 통해 어떠한 파라미터가 해당 macro-model에 영향을 주는지 정의 할 수 있다.

```
<Component ID="E001" Name="KEY" Description=" Key Managing Component of AES">
 < Interface ID="001" name="KeyExpansion">
  <Prototype type="void expandKey()"/>
  <EnergyModel model="127681238.519417"/>
 </Interface >
 <Interface ID="002" name="GetRoundKey">
   <Prototype type="void getRoundKey()"/>
   <EnergyModel model="22659163.689320"/>
 </Interface>
 <ManagementInformation Location="local library" Version="1.0" Author="D.H. Kim">
  <Status Ready="True" ArtifactType="Code"/>
</ManagementInformation>
 <Resource>
   <OS Type="Embedded Linux" Version="2.4"/>
   <Hardware CPU_Family="StrongARM(SA1100)"/>
   <Compiler name="gcc" Version="2.95.3"/>
 </Resource>
</Component>
```

그림 4 Key 컴포넌트의 명세 예시

5. 활용방안

5.1 전력소모 특성을 고려한 컴포넌트 관리

CoSLanE은 컴포넌트를 재사용하기 위한 정보들과 해당 컴포넌트의 전력 소모 특성을 기술하기 위한 정보들을 포함하여 설계되어 있다. 때문에 정의된 CoSLanE를 이용하여 컴포넌트 명세를 수행할 때, 컴포넌트의 전력소모 특성에 대한 정보를 포함하여 전사적 차원의 라이브러리를 통해 관리가 가능하다. 이러한 정보는 전력 소모량에 대한 요구사항의 중요도가 높은 소프트웨어 개발 시 유용하게 사용될 수 있다.

5.2 컴포넌트 선택 가이드라인

CoSLanE을 이용하여 컴포넌트를 명세할 경우 같은 기능을 제공하는 두 개 이상의 컴포넌트들 중 어떠한 컴포넌트가 전력 소모에 효율적인지를 판단할 수 있도록 가이드라인을 제시한다.

그림 6에서 Component A와 Component B는 각각 주어진 인터페이스를 통해 “2x+4854”와 “10x+4680”의 전력 소모를 한다. 이때 주어진 입력 값의 크기가 16byte 미만일 경우 Component B가 A보다 효율적으로 보이나, 이보다 클 경우에는 현저하게 많은 전력량을 소모한다는 사실을 알 수 있다. 이와 같이 컴포넌트에 대한 전력 소모량을 그래프 그렸을 때, 그래프간에 교차점이 발생한다면 어떠한 컴포넌트가 더 전력 소모 측면에서 효율적이라고 말 하기란 쉽지 않은 일이다. CoSLanE는 이러한 상황에서 소프트웨어 개발자가 해당 컴포넌트를 통해 처리되는 데이터양의 평균치를 분석하여 주어진 상황에서의 효율성을 고려한 선택을 수행할 수 있도록 지원한다.

```
<Component ID="Sys001" Name="ipc.h" Description="This is a library for IPC(Inter Process Communication ">
 < Interface ID="001" name="msgctl">
  <Prototype type="int msgctl(int msqid, int cmd, struct msqid_ds *buf)"/>
   <opmode=IPC_SET>
   <EnergyModel model="3334.7"/>
   </opmode>
   <opmode=IPC_STAT>
    <EnergyModel model="3835.7"/>
   </opmode>
   <opmode=IPC_RMID>
    <EnergyModel model="3603.6"/>
   </opmode>
 </Interface >
 <Interface ID="002" name="msgsnd">
  <Prototype type="int msgsnd(int msqid, struct msgbuf *msgp, int msgsz, int msgflg"/>
   <parameters>
    <parameter="int msqid"/>
    <parameter="struct msgbuf *msgp"/>
    <parameter="int msgsz"/>
    <parameter="int msgflg"/>
   </parameters>
   <returnType="int">
   <EnergyModel model="4.0x+4554.0"/>
   <map="int msgz->x">
 </Interface>
 <ManagementInformation Location="local library" Version="1.0" Author="GNU">
   <Status Ready="True" ArtifactType="Code"/>
 </ManagementInformation>
 <Resource>
   <OS Type="Embedded Linux" Version="2.4"/>
   <Hardware CPU_Family="StrongARM(SA1100)"/>
 </Resource>
</Component>
```

그림 5 ipc.h의 일부에 대한 명세

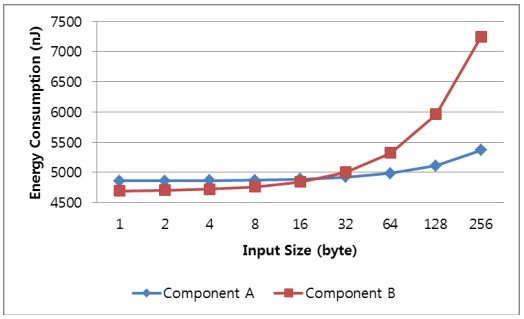

그림 6 같은 기능을 수행하는 컴포넌트간의 전력 소모량 비교 예시

5.3 Architecture를 이용한 전력 분석 지원

마지막으로, CoSLanE을 통해 명세한 컴포넌트를 이용하여 소프트웨어 아키텍처 수준에서의 전력 분석을 수행할 수 있다. 그림 7은 AES-SFT의 아키텍처를 나타낸다.

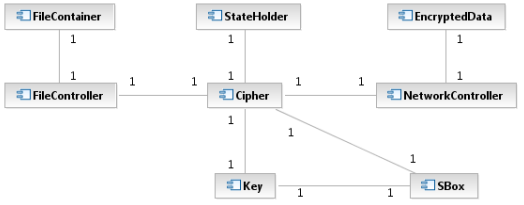

그림 7 AES-SFT의 Architecture

그림 4에서 명세한 "Key"는 그림 7에서 나타내는 아키텍처를 구성하는 하나의 컴포넌트다. 이때, 해당 아키텍처의 동적 구조를 통해 각 컴포넌트간의 상호작용을 도출하여 전력 분석을 수행한다면, 그림 4에 나타난 인터페이스의 소모 전력 특성을 통해 해당 컴포넌트의 소모 전력량을 도출 할 수 있다. 이와 같은 방법으로 소프트웨어를 구성하는 모든 컴포넌트들에 대한 명세가 존재 한다면, 이들을 합성하여 전체 소프트웨어가 소모하는 전력량을 도출 할 수 있다. 특히 이러한 경우 그림 5의 "ipc.h"와 같이 OS나 OS에 서비스를 요청하는 시스템 콜 등을 컴포넌트로 간주하여 이들에 대한 컴포넌트 명세를 수행한다면, 컴포넌트가 OS와 상호작용하는 경우에도 정확도가 높은 전력소모 분석을 수행할 수 있다.

6. 결론 및 향후 연구

본 연구에서는 소프트웨어 컴포넌트를 이용한 전력 소모 분석을 지원하기 위해 전력 소모 특성을 고려한 컴포넌트 명세 언어인 CoSLanE을 정의하였다. 정의된 CoSLanE을 통해 얻을 수 있는 효과는 다음과 같다.

- 전력 소모 특성을 고려한 컴포넌트 명세 및 이에 대한 효율적 관리
- 전력 효율적인 컴포넌트에 선택에 대한에 가이드라인 제시
- Software Architecture 수준에서의 소모 전력량 분석 지원

특히 위의 3)에서 나타내는 소프트웨어 아키텍처 수준에서의 컴포넌트 기반의 전력 분석은 소프트웨어 생명 주기의 앞 단계에서 분석을 수행할 수 있으며, 정확도 또한 높다는 장점을 기대해 볼 수 있다.

제안하는 명세언어를 이용하여 향후에는 CoSLanE에서 정의하는 내용을 기반으로 컴포넌트를 관리할 수 있는 컴포넌트 라이브러리 및 컴포넌트 기반 소프트웨어 아키텍처의 전력 분석 프레임워크를 구축할 예정이다.

참 고 문 헌

[1] Stephen S. Yau, "Embedded Software in Real-time Pervasive Computing Environments", in Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004.

[2] K. Naik, David S. L. Wei, "Software Implementation Strategies for Power-Conscious Systems", Mobile Networks and Applications, Vol. 6, Issue 3, pp.291-305, 2001.

[3] CJ Michael Geisterfer, Sudipto Chosh, "Software Component Specification: A Study in Perspective of Component Selection and Reuse", Proceedings of the Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Syatems (ICCBSS 2006), pp. 1-9, 2006

[4] OMG, "Reusable Asset Specification", version 2.2(formal/05-11-02), 2005 available from : http://www.omg.org

[5] G. Elias, et al., "X-ARM: An Asset Representation Model for Component Repository Systems", The 21st Annual ACM Symposium on Applied Computing Dijon, 2006

[6] L. Iribarne, et al., "A Non-Functional Approach for COTS Components Trading", in proc. Of the fourth workshop on requirements engineering (WER' 01), 2001

[7] A. V. Hoff, et al., "The Open Software Description Format (OSD)". Submission to the World Wide Web Consortium (W3C), 1997. available from : http://www.w3.org/TR/NOTE-OSD

[8] Chung, L., Nixon, B., Yu, E., and Mylopoulos, J., "Non-Functional Requirements in Software Engineering", Kluwer Academic Publisher, 2000.

[9] F. N. Najm, "Low-Power Design Methodology: Power Estimation and Optimization", Proceedings on the 40th Midwest Symposium on Circuits and Systems, Vol. 2, pp.1124-1129, 1997.

[10] A. Raghunathan, N. K. Jha, S. Dey, "High-level Power Analysis and Optimization", Kluwer Academic Publishers, Norwell, MA, 1998.

[11] J. Rabaey and M. Pedram, "Low Power Design Methodologies", Kluwer Academic Publishers, Norwell, MA, 1996.

[12] C. L. Su, C. Y. Tsui, and A. M. Despain, "Low Power architecture design and compilation techniques for high-performance processors", in Proceeding on IEEE COMPCON' 04, pp. 489-498, 1994.

[13] D. Sarta, D. Trifone, and G. Ascia, "A Data Dependent Approach to Instruction Level Power Estimation", IEEE Alessandro Volta Memorial Workshop on Low Power Design, pp. 182-190, 1999.

[14] T. K. Tan, A. Raghunathan, et al., "Energy Macromodeling of Embedded Operating Systems", ACM Transaction on Embedded Computing Systems, Vol. 4, Issue 1, pp. 231-254, 2005.

[15] J. W. Backus, M. Ibrahim, et al., "Compiler-based optimizations impact on embedded software power consumption", in Proceedings of the Jing Conference NEWCAS-TRRAISA, 2009.

[16] C. Backus, "The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference", Proceedings of the International Conference on Information Processing, UNESCO, pp.125-132, 1959.

[17] NIST, "FIPS-PUB-197: Advanced Encryption Standard", 2001. Available from : http://www.aescrypt.com

저 자 소 개

김 두 환

2007년 충북대학교 컴퓨터공학과 졸업(학사)
2009년 충북대학교 전자계산학과 졸업(석사)
2010년~현재 충북대학교 컴퓨터과학과 박사과정.

〈관심분야〉 소프트웨어 아키텍처, 임베디드 소프트웨어 모델링, 임베디드 소프트웨어 품질공학, 저전력 소프트웨어

이 재 욱

2010년 충북대학교 컴퓨터공학과 졸업(학사)
2011년 충북대학교 컴퓨터과학과 수료(석사)
2011년~현재 충북대학교 컴퓨터과학과 박사과정.

〈관심분야〉 소프트웨어 아키텍처, 소프트웨어 모델링, 임베디드 소프트웨어 품질공학

홍 장 의

1988년 충북대학교 전산학과(학사)
1990년 중앙대학교 컴퓨터공학과(석사)
2001년 한국과학기술원 전산학(공학박사)
2002년 국방과학연구소 선임연구원
2002년~2004년 (주)솔루션링크 기술연구소장
2004년~현재 충북대학교 컴퓨터공학 부교수

〈관심분야〉 소프트웨어공학, 임베디드 소프트웨어, 소프트웨어 품질공학, 소프트웨어 프로세스 개선, 저전력 소프트웨어