멀티프로세서용 임베디드 소프트웨어의 MDA 기반 개발†

충북대학교 홍장의
한국과학기술원 배두환

1. 서 론

임베디드 소프트웨어는 매우 다양한 영역에서 사용자 서비스를 제공하기 위하여 사용되고 있으며, 그 기능 또한 점점 복잡해지고 다양화되는 추세이다. 또한 정부 차원에서 임베디드 소프트웨어에 대한 기술 개발 및 산업 육성에 많은 노력을 기울이고 있으며, 이러한 상황은 임베디드 소프트웨어를 개발하고 활용하는 측면에서 여러 가지 환경 요소들에 대한 변화를 유도하고 있다.

이러한 변화중의 하나가 임베디드 소프트웨어를 탑재할 하드웨어 플랫폼에 대한 변화인데, 기존에는 초소형의 단일 프로세서만을 사용하여 사용자 서비스를 제공해 왔으나, 이제는 응용 영역이 더 복잡해지고 다양해지면서 신속하고 병렬처리가 가능한 높은 하드웨어 성능을 요구하는 다중 프로세서 환경으로 변화하고 있다. 또한 소프트웨어 개발에 있어서도 이러한 환경 변화를 수용하기 위한 새로운 기술적 접근을 필요하게 되었다.

MPSoC(Multi-Processor SoC)는 2개 이상의 프로세서 코어를 탑재하는 SoC로써, 향후 2~3년 이내에 그 소요가 급증할 것으로 보고 있다. 그런데 MPSoC는 설계 및 제작이 어렵고, 그 비용 또한 매우 높아서 한번 제작된 MPSoC를 변경하는 것은 지극히 어려운 일이다[1]. 따라서 이러한 플랫폼에 탑재될 임베디드 소프트웨어의 개발 또한 매우 신중하게 이루어져야 하는데, 이는 탑재될 임베디드 소프트웨어가 플랫폼 변경을 유발하는 원인을 제공하지 않아야 하기 때문이다. **본 연구에서는 이러한 MPSoC 환경, 또는 멀티프로세서 환경에서의 임베디드 소프트웨어를 개발하기 위한 모델 기반의 개발 방법을 제시한다.** 이를 위해서는 분석 및 설계 과정에서 고려되어야 하는 중요한 사항들이 있는데, 첫 번째는 분석 모델을 어떻게 다수의 프로세서로 할당할 것인가 하는 것이고, 두 번째는 할당으로 발생할 수 있는 프로세서간의 통신을 어떻게 최적화하여 성능 요구사항을 만족시킬 것인가 하는 것이다. 또한 MPSoC 플랫폼에 적합한 소스 코드, 특히 C 코드를 생성하는 과정도 매우 중요한 고려 요소라 할 수 있으며, 모델에 대한 검증 및 시뮬레이션을 통해 기능 성능 요구사항을 충분히 만족하고 있는지 확인하는 것 또한 필요하다.

본 연구에서는 특히 UML 2.0을 사용하여 MPSoC형 임베디드 소프트웨어의 모델링을 지원하고 모델링 과정에서 고려되어야하는 주요 이슈들을 어떻게 반영할 것인지에 대하여 살펴본다.

2절에서는 모델 기반의 임베디드 소프트웨어 개발을 위한 전체적인 개발 프로세스를 제시하고, 3절과 4절에서는 개발 프로세스에 따른 PIM 모델링 단계와 PSM 모델링 단계를 간단한 예제를 통해 설명한다. 그리고 5절에서는 결론을 제시한다.

2. 모델 기반 개발 프로세스

UML 2.0을 사용하여 임베디드 소프트웨어의 모델링을 수행하는 시도가 더욱 많아지고 있다[2,3,4]. 이는 2005년 후반에 OMG에서 발표한 UML 2.0의 정의에서 임베디드 소프트웨어의 특성을 표현하기 위한 다양한 모델링 특성(features)들이 추가되었고[5], 특히 임베디드 시스템의 구성요소들이 객체의 개념과 잘 매핑될 수 있다는 직관성 때문이다.

본 연구에서는 특히 임베디드 소프트웨어의 모델링 과정으로 발생할 수 있는 엔지니어의 부담을 줄이고, 효과적이며 직관적인 소프트웨어 개발을 진행할 수 있도록 지원하는 ESUML(Embedded Software modeling with UML 2.0) 방법을 제시하였다[6,7,8]. 제시하는 ESUML 방법의 특징은 다음과 같다.

- 초기 시나리오부터 분석 및 설계 활동이 자연스럽고 직관적으로 수행될 수 있도록 상호작용 기반

† 본 연구는 정보통신부 정보통신연구진흥원이 지원하는 선도기반기술사업의 지원을 받아 수행되었습니다.

의 모델링 방법을 제공한다. 이는 추후 모델의 변경으로 발생할 수 있는 일관성의 손실을 최소화할 수 있다.

- ○PIM 모델과 PSM 모델의 분리를 통해 소프트웨어를 개발할 수 있도록 모델 기반 개발 접근방법[9]을 제공한다.
- ○MPSoC/멀티프로세서 환경을 고려하여 소프트웨어가 개발될 수 있도록 모델링 활동을 포함하고 있다.
- ○모델에 대한 분석 및 검증을 위한 시뮬레이션 기능을 제공하며, 이는 지원도구에 의해 이루어진다.
- ○마이크로 스텝에서는 계층적 모델링을 통해 소프트웨어의 복잡함을 핸들링할 수 있도록 하였으며, 매크로 스텝에서는 반복적 개발을 통한 모델 정제 및 품질 개선을 할 수 있도록 하였다.

이상의 특징을 제공하는 ESUML 방법에 대한 소프트웨어 개발 절차는 그림 1과 같다.

그림 1에서 ESUML 기반 임베디드 소프트웨어 개발 절차는 초기 요구사항을 확보하는 단계를 수행한 후, 소프트웨어의 구조 및 행위 모델링 활동, 즉 PIM 모델링 단계와 PIM 모델을 분할하여 MPSoC 환경으로 매핑하는 활동, 즉 PSM 모델링 단계로 정의하였다. 후속 단계로써 모델 검증 단계와 코드 생성 단계를 두고 있다. 모델 검증 단계에서는 규칙 기반의 모델 정적 분석 및 시나리오 기반의 동적 시뮬레이션 기능을 제공하며, 코드 생성 단계에서는 C 언어 또는 자바 언어 등의 코드 생성을 가능하도록 제공한다. 물론 이러한 ESUML 프로세스는 자동화된 도구에 의해 지원되고 있다.

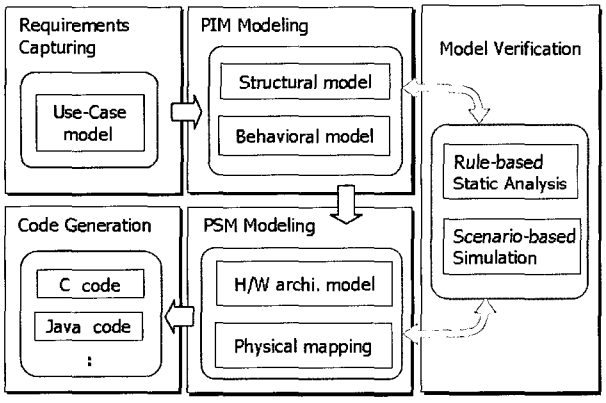

그림 1 ESUML에 의한 소프트웨어 개발 절차

3. ESUML: PIM 모델링

본 절에서는 그림 1에서 제시된 ESUML 개발 절차에서의 각 단계별 모델링 방법과 주요 산출물에 대하여 설명한다. 이해의 용이성을 위하여 이동통신 단말장치(cellular phone)의 심비안 OS 버전 9.1 스펙의 2.2.2절에 기술되어 있는 메시지 처리 시스템(MHS)에 대한 요구사항[10]을 기반으로 설명한다.

3.1 요구사항 확보 단계

요구사항의 확보는 UML 2.0의 Use Case Diagram (UCD)에 의해 이루어진다. UCD는 사용자의 요구사항을 직관적으로 표현할 수 있는 모델링의 유용성을 제공한다[11]. 특히 ESUML 방법에서는 액터와 시스템간의 상호작용을 갖는 이벤트 중심의 Use Case가 정의되며, 또한 복잡도의 핸들링을 위하여 계층적인 UCD의 작성이 가능하다.

그림 2는 예제 시스템에 대한 UCD의 예를 보여준다. 이는 상위 UCD의 상세화된 하위 UCD의 하나로써, 상위 UCD는 크게 3개의 Use Cases, 즉 "Send Message," "Manage Message," 그리고 "Receive Message"를 갖는다. 이중에서 "Send Message"에 대한 상세 모델을 그림 2에 나타내었다.

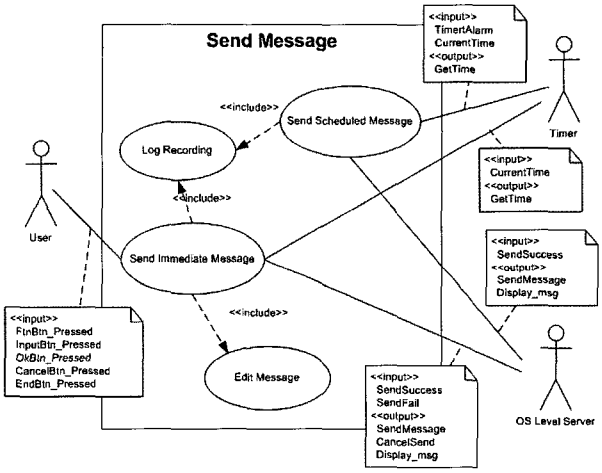

그림 2 MHS의 Use Case 다이어그램 예제

그림 2에서 보는 바와 같이 각 액터와 Use Case간에는 입출력 이벤트가 정의되며, 이들은 Use Case Description에서 보다 구체적으로 시나리오의 각 단계(step)와 대응된다. 오른쪽 하단의 액터 "OS Level Server"는 MHS와 연동하는 운영체제(또는 미들웨어)에 해당된다.

3.2 소프트웨어 구조 모델링

임베디드 소프트웨어의 구조 모델링은 시스템을 구성하는 독립적인 요소들을 추출하여 객체로 정의하는 활동으로 UML의 클래스 다이어그램을 작성한다. 본 모델링에 있어서 각 클래스의 정의는 그림 3과 같은 요소로 정의된다.

```
Class Structure :: =
    [{ + {abstract | concrete } + }] +
    Class_Name +
    [<< + Stereotype + >>] +
    {Attribute}* + {Operation}*;
Stereotype ::=
    Process_Type + Device_Type;
Device_Type ::=
    { input | output | memory | processor };
Process_Type ::=
    { entity | periodic | asynchronous |
    synchronous | control | null };
```

그림 3 클래스 다이어그램의 구성요소

그림 3의 정의에서 PIM 단계에서 확정되지 않는 디바이스 클래스의 경우는 {abstract} 형태를 가지며, 이는 PSM 단계에서 {concrete} 형태의 스테레오타입을 갖는 클래스로 표현된다. 또한 각 클래스가 Active 클래스인지, Passive 클래스 인지를 나타내기 위해 각 클래스들의 프로세스 타입을 정의한다.

3.3 소프트웨어 행위 모델링

행위 모델링은 ESUML 방법에서의 가장 핵심적인 부분으로써, UML의 Interaction Overview Diagram (IOD)과 Sequence Diagram(SD)에 의해 표현된다. 행위 모델링을 위한 세부 절차는 다음과 같다.

(1) UCD 계층 구조상의 최하위에 나타나는 Use Case별로 하나의 IOD를 작성한다.
  (1.1) 앞 단계에서 작성한 UC Description을 검토한다.
  (1.2) UC Description의 성공시나리오에 나타나는 각 스텝을 중심으로 Interaction-Use 노드를 식별하고 이들 간의 제어 흐름을 정의 한다.
  (1.3) 입출력 이벤트가 매핑된 시나리오의 스텝에 대하여 적절한 분기타입을 선택하여 모델에 반영한다.
  (1.4) 분기 타입은 조건 분기, 단순 분기, 인터럽트에 의한 분기로 구분된다.
  (1.5) UC Description의 대안(alternative) 시나리오에 정의된 각 스텝에 대하여 해당 IOD에 통합 작성한다. 이러한 통합은 (1.4)의 분기타입에 의해 결정된다.
(2) 작성된 IOD 다이어그램은 복잡도를 감소하기 위해 계층적으로 모델링 가능하다.
(3) IOD에 나타나는 각각의 InteractionUse에 대하여 SD를 작성한다.
(4) 작성되는 SD에 대해서도 "ref" 프레임을 사용하여 계층화할 수 있다.

그림 4는 위의 주어진 절차에 따라 작성된 IOD의 간단한 예를 보여준다. 그림 4의 예제는 그림 2에서의 Use Case "Send Immediate Message"에 대한 IOD의 일부분으로써, 사용자가 임의의 기능 버튼을 누르면 시스템은 해당 동작을 수행하고 외부로 시그널을 보내는 성공 시나리오에 대한 동작 부분과 시스템 동작 수행 중에 임의의 버튼, 예를 들면 종료나 취소 버튼이 눌려질 수 있는 대안 시나리오(회색 부분)를 모두 포함하고 있다. 특히 그림 4의 예제에서 대안 시나리오는 조건(if-clause)에 의해 이루어지는 행위가 아닌 인터럽트에 의해 이루어지는 분기 동작이기 때문에 Interruptable Region으로 모델링 되었다.

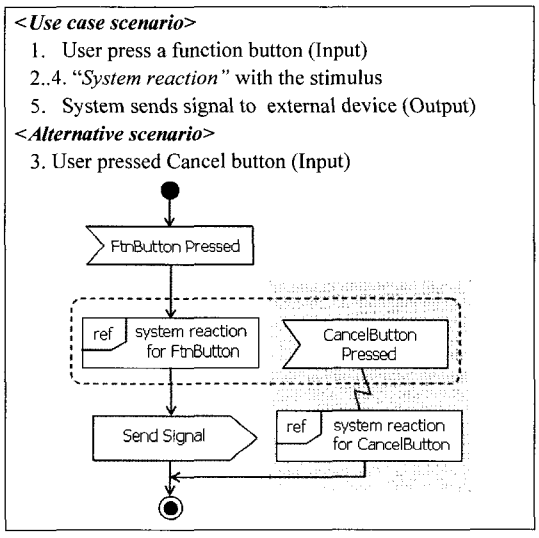

그림 4 IOD 다이어그램의 예제

그림 4에 나타난 InteractionUse, 즉 "ref" 프레임에 대해서는 대응하는 SD로 상세화 된다. 이러한 모든 상세화 과정이 진행되면, 앞서 작성되었던 클래스 다이어그램에 대한 정제가 이루어지는데, 이는 클래스의 세부 속성 및 연산에 대한 추가적인 사항을 정의하기 위함이다.

4. ESUML: PSM 모델링

4.1 PSM 모델링 개요

PIM 단계가 완료되면, PSM 단계가 진행된다. PSM 단계의 주요한 활동들은 하드웨어 아키텍처 다이어그램 작성, PIM 모델의 분할, 태스크 할당 등으로 구성된다. 그림 5는 PSM 단계의 수행 절차를 도식화 한 것이다.

그림 5에서 하드웨어 아키텍처 모델링은 요구사항을 기반으로 작성하게 되며, 모델 분할은 Active 클래스

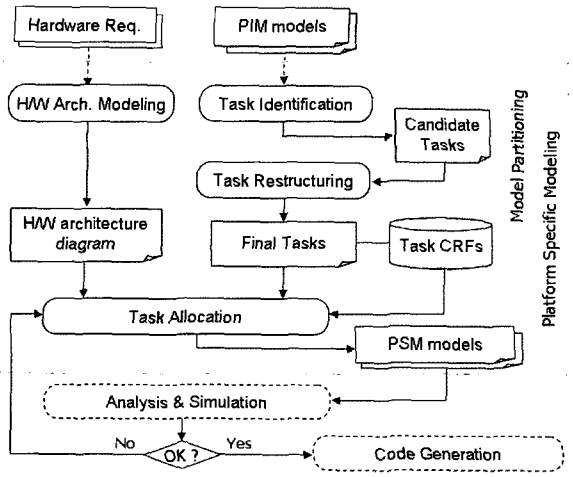

그림 5 ESUML: PSM 모델링 절차

를 중심으로 후보 태스크를 식별하고, 이들로부터 최종 태스크를 도출한다. 태스크의 도출과정에서 태스크간의 상관성을 정의하는 CRFs(Correlation Factors)가 산출된다. CRFs 값은 최종 태스크를 하드웨어 컴포넌트에 할당하는 과정에서 사용된다.

4.2 하드웨어 아키텍처 모델링

PSM 단계의 첫 번째 작업은 하드웨어에 대한 아키텍처 다이어그램을 작성하는 것이다. 초기 사용자 요구사항에서 주어진 하드웨어 스펙을 기준으로, 또는 PIM 모델링 과정을 고려한 하드웨어 컴포넌트를 결정하여 모델을 작성한다. 하드웨어 아키텍처 다이어그램을 생성하기 위한 고려사항은 다음과 같다.

- ○하드웨어 아키텍처는 다음과 같은 구성요소들로 이루어질 수 있다[12].
  - Bridge, BUS, CPU, DSP, HW Intellectual Properties, IO, Memory, Peripheral
- ○하드웨어 구성요소들은 상용의 컴포넌트에 대한 세부 사양을 가지고 있으며, 이들은 사전에 정의된 라이브러리 형태로 유지 관리된다.
- ○모델러는 라이브러리로부터 원하는 컴포넌트를 Drag & Drop함으로써, 하드웨어 아키텍처 다이어그램을 작성 한다.

그림 6은 하드웨어 아키텍처 다이어그램의 예를 보여준다. 참고로 그림 6의 모델은 통신 단말용 하드웨어 아키텍처 다이어그램이 아닌 시뮬레이션용 보드의 아키텍처임을 밝힌다.

그림 6에 나타난 하드웨어 아키텍처 다이어그램의 XML에 의해 표현될 수 있는데, 이중에서 "ARM1"에 대한 부분적인 XML 정의는 그림 7과 같다.

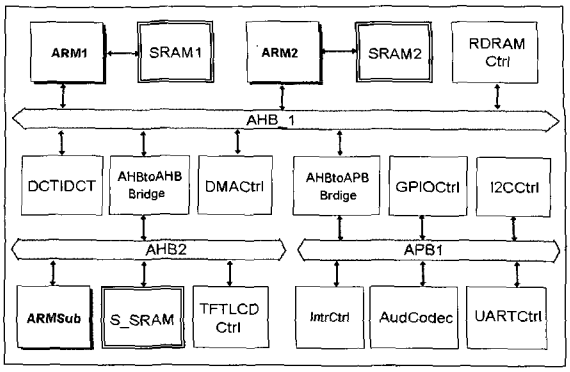

그림 6 하드웨어 아키텍처 다이어그램 예제

```
-<architecture_diagram archi_diagramID="_ARCH" isModified="true">
 <archi_diagram_name>MPSoC_ARCH</archi_diagram_name>
 -<archi_diagram_description>
  -<![CDATA[this diagram could be sample MPSoC Architecture
      which could be used as the demo architecture]]>
 </archi_diagram_description>
 -<component_list>
  -<cpu componentID="ARM926EJS_1" nameType="ARM926EJS"
    name="ARM1" isMaster="false">
    -<description>
      <![CDATA[First Main CPU ]]> </description>
    <informationname="ARM926EJS" vendor="UMC" version="1.0" />
  -<configuration>
     <definedType="ARM" Value="ARM926EJS" />
     <definedType="Cache" Value="IDCache" />
     <definedType="MaxSpeed" Value="600" />
     <configType="OS" Value="Velos" />
     <configType="DcacheSize" Value="0x10000" />
     <configType="IcacheSize" Value="0x10000" />
     <configType="Speed" Value="400" />
     <configType="Compiler" Value="armcc" />
     <configType="Assembler" Value="armasm" />
     <configType="Linker" Value="armlink" />
     <configType="IcacheAssociation" Value="2" />
     <configType="DcacheAssociation" Value="2" />
  </configuration>
```

그림 7 하드웨어 아키텍처의 XML 표현

4.3 모델 분할 및 태스크 식별

하드웨어 아키텍처가 결정되면, 이에 탑재될 소프트웨어에 대한 분할 작업이 진행된다. 모델의 분할 작업은 각 프로세서에 할당할 단위 태스크들을 식별하는 작업[2]이다.

4.3.1 후보태스크 도출

후보 태스크를 도출하기 위해서는 먼저 클래스의 스테레오타입이 (1) Asynchronous, (2) Periodic, 그리고 (3) Control인 클래스들을 후보 태스크로 정의하는 것이다. 이들이 후보 태스크가 되는 이유는 프로세스 특성상 독립적으로 수행할 가능성이 높기 때문이다. 또한 후보 태스크이외에 공유 메모리(그림 6의 S_SRAM 컴포넌트)로 할당되는 클래스는 영속하는 데이터를 갖는 〈〈entity〉〉 스테레오타입들이다. 이들도 초기 매핑을 위한 대상으로 사전 정의된다. MHS 예제 시스템의 모델링에 있어서 전체 클래스는 17개가

식별되었고, 이중에서 4개의 클래스가 후보 태스크로, 6개 클래스가 엔티티 태스크로 선정되었다.

- ○〈〈asynchronous device input〉〉 ButtonClass
- ○〈〈control〉〉 MHScontroller
- ○〈〈asynchronous timer input〉〉 TimerInterfaceClass
- ○〈〈asynchronous device interface〉〉 OSInterfaceClass
- ○〈〈entity〉〉 MultimediaFileClass
- ○〈〈entity〉〉 ReveivedMsgBoxClass
- ○〈〈entity〉〉 SentMsgBoxClass
- ○〈〈entity〉〉 SavedMsgBoxClass
- ○〈〈entity〉〉 ScheduledMsgBoxClass
- ○〈〈entity〉〉 ReservedMsgBoxClass

4.3.2 CRFs 값의 산출

후보 클래스가 식별되면 나머지 클래스들에 대하여 후보 클래스로 통합하여 최종의 태스크를 도출하게 되는데, 이는 각 클래스에 대하여 CRFs 값을 산출하고, 이 값을 기준으로 최종 태스크를 도출하는 것이다. CRFs 값을 산출하기 위한 공식은 다음과 같다.

$$F(T_i,\ O_j) = k(\sum_l \omega_l C_l) + \sum_m I_m \qquad \text{(식 1)}$$

여기서 ω_l는 C_l에 대한 가중치의 값이며, k는 C_l를 I_m보다 더 비중있게 고려하기 위한 중요도 상수이다. C_l과 I_m에 대한 정의는 표 1과 같다.

표 1 CRFs 산출식의 측정요소에 대한 정의

$C_{l=1}$	후보 태스크 T_i와 객체 O_j 간의 Temporal Cohesion 여부 (ω = 3)
$C_{l=2}$	후보 태스크 T_i와 객체 O_j 간의 Sequential Cohesion 여부 (ω = 2)
$C_{l=3}$	후보 태스크 T_i와 객체 O_j 간의 상호 배타적 관계 여부 (ω = 1)
$I_{m=1}$	태스크 T_i와 객체 O_j 간의 상호작용(관계) 수
$I_{m=2}$	태스크 T_i와 객체 O_j 간에 발생하는 서로 다른 타입의 상호작용 수
$I_{m=3}$	태스크 T_i와 객체 O_j 간에 전달되는 메시지의 크기

표 1에서 $C_{l,\ l=1..3}$의 값은 0 또는 1의 값을 갖는 불리언 척도이며 후보 태스크 객체와 잔여 객체간의 응집력을 산출한다. $I_{m,\ m=1..2}$척도는 정수 값을, $I_{m=3}$은 기본 데이터 타입을 1로 가정했을 때의 상대적인 메시지의 크기로 정의하였다. (식 1)을 이용하여 산출된 예제 시스템에 대한 CRFs 산출 결과는 표 2와 같다.

표 2 예제 시스템에 대한 CRFs의 값

후보태스크	클래스	$F(T_i, O_j)$, k=4
ButtonClass	*FtnButton*	2*k* +3 =11
	InputButton	2*k* +12 =20
	CancelButton	2*k* +3 =11
	EndButton	2*k* +3 =11
	OKButton	2*k* +3 =11
	MessageList..	2*k* +3 =11
	Message	2*k* +3 =11
	MHS_contro..	2*k* +7 =15
MHScontroller	*ButtonClass*	0*k* +7 =7
	TimerInter..	2*k* +3 =11
	OSInterfac..	2*k* +8 =16
	FtnButton	0*k* +0 =0
	InputButton	0*k* +0 =0
	CancelButton	0*k* +0 =0
	EndButton	0*k* +0 =0
	OKButton	0*k* +0 =0
	MessageList..	1*k* +16 =20
	Message	2*k* +12 =20
TimerIn..	*MHScontro..*	2*k* +4 =12
	FtnButton	0*k* +0 =0
	InputButton	0*k* +0 =0
	CancelButton	0*k* +0 =0
	EndButton	0*k* +0 =0
	OKButton	0*k* +0 =0
	MessageList..	0*k* +0 =0
	Message	0*k* +0 =0
OSInterf..	*MHScontro..*	2*k* +8 =16
	FtnButton	0*k* +0 =0
	InputButton	0*k* +0 =0
	CancelButton	0*k* +0 =0
	EndButton	0*k* +0 =0
	OKButton	0*k* +0 =0
	MessageList..	0*k* +0 =0
	Message	0*k* +0 =0

표 2로부터 각 후보 태스크에 대한 CRFs 값을 산출할 때, 후보 태스크간의 CRFs 값도 함께 산출하였다. 이들 값은 다음 단계의 태스크 할당 과정(4.4절)에서 유용한 정보로 사용될 것이며, 최종 태스크를 선정하기 위한 과정에서는 사용되지 않는다.

4.3.3 최종 태스크 결정

4개의 후보 태스크와 6개의 엔티티 태스크를 제외한 나머지 7개의 클래스들은 각각 산출된 CRFs 값에 의해 최종의 태스크들로 그룹핑된다. 표 2의 결과에서 보듯이 태스크의 그룹핑에 대한 결과는 매우 명확하다. 그룹핑은 상호 응집력이 있고, 상호작용이 많은 경우에 진행된다. 예제 시스템에 대해서는 초기 설정된 4개의 후보 태스크가 최종 태스크로 확정되는 경우이다.

만약 임의의 한 클래스가 모든 후보 태스크들에 대하여 일정한 임계치 이하의 CRFs 값을 갖는 경우 이는 별도의 후보 태스크로 도출될 수 있다.

4.4 태스크 할당

최종적으로 결정된 태스크에 대하여 하드웨어 아키텍처 다이어그램의 구성요소인 프로세서와 메모리로의 할당 작업이 진행된다. 프로세서에는 일반적인 처리 태스크가 할당되고, 메모리에는 영속하는 데이터가 할당된다. 태스크 할당에서는 다음과 같은 경우를 고려할 수 있다.

(1) 태스크의 수와 프로세서의 개수가 일치하는 경우 : 해당 프로세서로 일대일 매핑한다.
(2) 프로세서의 개수가 많은 경우 : 하나의 프로세서에 대하여 하나의 태스크를 할당하고, 나머지 프로세서는 무시한다.
(3) 태스크의 수가 프로세서 개수보다 많은 경우 : 태스크간의 CRFs 값을 이용하여 태스크를 서로 병합한 후에 할당한다.

위의 (3)의 경우를 처리하기 위하여 그림 8과 같이 최종 태스크간의 CRFs 값을 고려한다. 두 태스크, Ti와 Tj간의 CRFs 값은 어느 태스크를 기준으로 보는 간에 따라 서로 다른 CRFs 값을 갖게 된다. 이는 태스크에서 다른 태스크로 전달되는 이벤트의 특성에 따라 서로 다른 처리 형태를 보이기 때문이다.

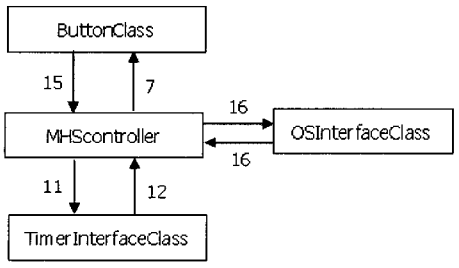

그림 8 태스크간의 CRFs 산출치

그림 8에서 보여주는 CRFs 값에 대하여 두 태스크 간의 값을 합하여 비교해 보면 "ButtonClass"와 "MHScontroller" 간의 CRFs 값이 22로 가장 작음을 알 수 있다. 따라서 프로세스의 수가 3개인 경우는 "ButtonClass"와 "MHScontroller"를 하나의 프로세서로 할당하게 되며, 프로세서의 수가 2개인 경우는 "ButtonClass", "MHScontroller" 그리고 "TimerInterfaceClass"가 하나의 프로세서로 할당되게 된다.

5. 결 론

본 연구에서는 객체지향 기반으로 임베디드 소프트웨어를 개발하고자 하는 경우, 시스템의 상호작용을 중심으로 하는 UML 2.0 기반의 모델링 방법론을 소개하였다. 상호작용 중심의 모델링 방법론은 초기에 식별되는 Use Case를 이용하여 자연스럽게 분석 및 설계 활동을 수행할 수 있기 때문에 직관적이며 용이한 모델링이 가능하다는 장점을 제공한다.

특히 본 연구에서는 멀티프로세서 환경에서의 임베디드 소프트웨어를 개발하는데 있어서 PIM 모델과 PSM 모델을 상호 어떻게 매핑 할 것인가에 대한 연구 결과를 제시하였다. 객체지향 모델로부터 태스크를 도출하고 이들 간의 상관성 인자(CRFs)를 산출하여 최종 태스크를 정의하는 방법에 대하여 간단한 통신단말장치의 예제를 통해 설명하였다.

점점 더 복잡해지고 여러 가능을 하나로 통합하려는 유비쿼터스 시스템 환경에서 MPSoC 플랫폼에 대한 요구가 증대될 것이고, 이로 인하여 MPSoC용 임베디드 소프트웨어의 개발을 지원하기 위한 방법이 필요할 것이다. 본 연구는 이러한 필요성을 충족시키는 하나의 해법이 될 수 있을 것으로 판단한다.

참고문헌

[1] Giovanni De Micheli, "MPSoC HW Challenges - Reliability and Reiable Design," Tutorial of MPSoC'05, July 2005. France.
[2] H. Gomaa, Designing Concurrent, Distributedm and Real-Time Applications with UML, Addison-Wesley, 2000.
[3] B. P. Douglass, Real Time UML, 3rd ed., Addison-Wesley, 2004.
[4] M. Awad, et. al., Object-Oriented Technology for Real Time Systems, Prentice-Hall, 1996.
[5] OMG, UML 2.0 Superstructure Specification, Doc., ptc/04-10-02, Oct., 2004.
[6] 전상욱, 이희진, 홍장의, 배두환, ESUML: UML 기반 임베디드 소프트웨어 모델링 방법론, 한국정보과학회 추계학술대회, 2005년, pp.343-345.
[7] 전상욱, 홍장의, 배두환, Operational Architecture에 기반한 임베디드 소프트웨어 계층적 모델링, 한국 소프트웨어공학 학술대회, 2006년 2월, pp.49-56.
[8] S-U Jeon, J-E Hong, and D-H Bae, "Interaction-Based Behavior Modeling of Embedded Software using UML 2.0," ISORC 2006, April 2006, Korea, pp.351-355.
[9] A. Kleppe, et. al., MDA Explained: The Model Driven Architecture, Addison-Wesley, April 2003.

[10] Symbian OS Ver9.1 Product description, Revision 1.1, Feb. 2005.
[11] A. Dennis, et. al., "System Analysis and Design with UML Version 2.0, Wisley, 2005.
[12] Steve Furber, ARM System-on-Chip Architecture, 2nd Ed., Addison-Wesley, 2000.

홍 장 의

1988 충북대학교 전산학과(이학사)
1990 중앙대학교 전산학과(석사)
2001 한국과학기술원 전산학과(박사)
2002 국방과학연구소 선임연구원
2004 (주)솔루션링크 기술연구소장
2004~현재 충북대학교 전기전자컴퓨터공학부 조교수
관심분야 : 임베디드 소프트웨어 공학, 소프트웨어 개발 방법론, 소프트웨어 품질, 소프트웨어 프로세스 개선
E-mail : jehong@chungbuk.ac.kr

배 두 환

1980 서울대학교 조선공학(학사)
1987 미국 Univ. of Wisconsin - Milwaukee 전산학(석사)
1992 미국 Univ. of Florida 전산학(박사)
1995~현재 한국과학기술원 전산학과 교수
관심분야 : 소프트웨어 프로세스, 객체지향 프로그래밍, 컴포넌트기반 프로그래밍, 임베디드 소프트웨어 설계, 관점지향 프로그래밍
E-mail : bae@se.kaist.ac.kr