리눅스 기반 임베디드 시스템에서의 템플릿을 이용한 재사용 가능한 디바이스 드라이버 개발

최재현° 이우진 정기원
숭실대학교 대학원 컴퓨터학과
uniker80@empal.com°, bluewj@dreamwiz.com, chong@computing.ssu.ac.kr

Reusable Device Driver Development using Template for Embedded Systems with Linux

Jaehyun Choi°, Woojin Lee, Kiwon Chong
Department of Computing, Graduated School, Soongsil University

요 약

임베디드 시스템의 발전에 따라, 임베디드 소프트웨어의 활용 분야와 복잡도는 급격하게 증가하는 반면, 실제적인 임베디드 소프트웨어의 개발 생산성과 품질은 그 요구에 부합하지 못하고 있다. 이러한 문제의 근본적인 해결을 위해서는, 기존의 임베디드 시스템 의존적인 방식에서 벗어난 임베디드 시스템 독립적인 소프트웨어의 개발 기법이 필요하다. 이에 따라 본 논문에서는, 이종 플랫폼 통합 개발방식으로서 제시된 MDD의 개념을 바탕으로, 임베디드 시스템 소프트웨어인 디바이스 드라이버를 템플릿 형태로 정의하고, XML 문서로 표현된 임베디드 시스템 정보와 매핑을 통해 실제 디바이스 드라이버를 개발하는 방안을 제시한다. 이것은 하나의 디바이스를 템플릿을 통해 시스템 독립적으로 표현하고, XML 로 표현된 임베디드 시스템 정보와의 매핑을 통하여 여러 임베디드 시스템에서 동작가능한 디바이스 드라이버를 자동으로 생성함으로써, 임베디드 소프트웨어의 개발생산성과 유지보수성 향상을 보장한다. 또한, 동일 소프트웨어의 중복개발 방지 및 재사용성 향상을 통해 개발비용의 축소와 개발 기간의 단축을 보장한다.

1. 서론

임베디드 시스템에서 산업용 기기를 제어하기 위해 사용되던 임베디드 소프트웨어는 최근, 멀티미디어 처리, 다중작업 및 실시간 처리 능력이 강화되고 유무선통신 및 네트워크와의 접목으로 전통적인 제조, 유통, 금융, 서비스 산업뿐만 아니라 항공, 우주, 국방, 의료, 멀티미디어 통신 및 에너지 개발 등의 첨단 분야에 이르기까지 그 사용범위와 영향력이 점점 커지고 있다[1].

하지만, 이러한 임베디드 소프트웨어는 범용으로 개발되기 보다는 특정 임베디드 시스템을 대상으로 하여 개발되고, 자체 개발능력을 갖춘 수백 개의 소형 업체들에 의해 독자적으로 개발되기 때문에, 동일 기능을 수행하는 임베디드 소프트웨어라 할지라도 특정 임베디드 플랫폼에 따라 별도로 제작되어야 하는 어려움이 있다. 이러한 상황은, 임베디드 소프트웨어의 개발 생산성을 떨어뜨리고, 소프트웨어에 대한 유지보수성 및 품질 저하에 큰 영향을 끼치고 있다. 특히, 임베디드 시스템 소프트웨어인 디바이스 드라이버는 임베디스 시스템을 구성하는 장치의 운용을 위한 제어 소프트웨어로써, 동일 장치를 대상으로 동일 기능의 제공을 목적으로 개발되지만, 대상 임베디드 시스템에 따라 별도로 개발되고 있어 임베디드 시스템 개발 생산성 저하의 주요 원인이 되고 있다.

따라서 본 논문에서는, 리눅스 기반의 임베디드 상에서 동작하는 디바이스 드라이버 개발에 있어서, 생산성 및 유지 보수 비용, 품질 측면을 고려한 템플릿 기반의 임베디드 디바이스 드라이버 개발 방안을 제안한다.

2. 관련 연구

2.1 디바이스 드라이버(Device Driver)

장치 제어기 또는 구동 드라이버로서 정의 된다. 하드웨어와 운영체제 응용프로그램의 연결 고리가 되는 프로그램으로 하드웨어 구성 요소가 운영체제하에서 작동하기 위해 필요한 모듈이다. 이러한 모듈은 근본적으로 특권(Privilege)층에서 실행되고, 메모리에 상주하며, 저급 하드웨어 처리 루틴을 가진 공유 라이브러리이다[2]. 이러한 디바이스 드라이버는 하드웨어 디바이스의 통신을 통해 디바이스를 제어하며, 이 때 디바이스 드라이버는 하드웨어의 레지스터에 명령어를 전달하고, 하드웨어는 그 명령어를 해석하여 특정 작업을 수행하게 된다.[3] 여기에서는 사용되는 레지스터의 지정방식에 따라 디바이스 드라이버의 입출력 방식은 I/O 맵 전달 방식과 메모리 맵 전달 방식 2가지로 분류된다[3].

본 논문에서, 제시하고 있는 디바이스 드라이버 개발 기법은 이러한 메모리 맵 전달방식의 디바이스 통신을 하고 있는 임베디드 시스템을 대상으로 한다.

2.2 모델기반 개발 방식(Model-Driven Development)

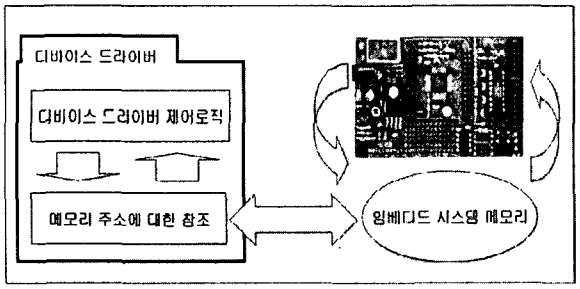
MDD는 MDA 기반의 개발 방식이다[4]. MDA는 국제적인 소프트웨어 기술 표준 단체인 OMG(Object Management Group)에 의해 정의된 소프트웨어 개발을 위한 프레임워크로, 설계환경이나 시스템에 얽매이지 않는 소프트웨어 개발방법론이다. MDA기반 소프트웨어 개발 방식의 핵심은 우선 플랫폼에 독립적인 소프트웨어 모델(PIM)을 정의하고, 이를 플랫폼 종속적인 모델(PSM)로 변환하여, 소프트웨어의 소스코드를 자동 생성하는 것이다. 이러한 MDA는 플랫폼 독립 모델(PIM)과 플랫폼 종속 모

델(PSM)간에 매핑 규칙을 사용하여 일관성 있는 구조를 유지하도록 하며, 이것은, 플랫폼에 독립적인 모델(PIM)을 작성함에 있어서 시스템 구성이나 파라미터와 같은 플랫폼 종속적인 정보들을 모델링 할 수 있게 한다[5][6]. 따라서, 개발 플랫폼(Hot machine)과 실제 구동될 플랫폼(Target Machine)이 다른 임베디드 소프트웨어 개발에 이러한 MDA를 적용하는 것은, 구동 플랫폼(Target Machine)과의 독립적 설계를 통해 소프트웨어의 개발 생산성 향상과 함께, 설계 수준에서의 검증 및 시험을 통한 개발 비용의 축소 및 품질향상을 기대할 수 있다[7].

3. 템플릿을 이용한 디바이스 드라이버 개발

3.1 디바이스 드라이버 템플릿

메모리 맵 전달 방식의 임베디드 시스템에서 디바이스 드라이버는, 임베디드 시스템 메모리에 특정 제어명령의 전달을 위해 동작하는 제어로직과 실제 디바이스와 매핑되는 메모리의 주소에 대한 참조로써 정의할 수 있다. [그림 2]는 디바이스 드라이버와 실제 임베디드 시스템간의 관계를 나타내고 있다



[그림 1] 디바이스 드라이버 모델과 임베디드 시스템

디바이스 드라이버는 임베디드 시스템 내의 메모리 주소에 대한 참조를 통하여 시스템을 모니터링 하고, 제어 메시지를 전달한다. 이때, 메모리 주소에 대한 참조는 디바이스 드라이버가 제어명령을 전달하고자 하는 장치에 대한 시스템상의 논리적인 위치를 제공한다. 따라서, 상이한 임베디드 시스템에서 동작하는 동일한 디바이스에 대하여, 제어로직내에서 사용하고 있는 메모리 주소에 대한 참조와 실제 메모리간의 일관성 있는 매핑 규칙을 정의하여 적용하면, 동일한 디바이스에 대하여 상이한 임베디드 시스템에서 동작하는 디바이스 드라이버를 생성하는 것이 가능하다. 즉, 동일한 디바이스 드라이버를, 목적 임베디드 시스템과는 독립적인 제어로직과 메모리 주소에 대한 참조로 구성된 템플릿 형태로 표현하고, 여기에 다시 메모리 주소에 대한 참조와 목적 시스템의 실제 메모리간의 매핑을 통하여, 실제적인 목적 임베디드 시스템의 디바이스를 제어할 수 있는 디바이스 드라이버를 생성할 수 있다.

본 연구에서는 리눅스 기반 임베디드 시스템을 대상으로 하고 있기 때문에, 디바이스 드라이버 템플릿은, C언어를 기반으로 하여 정의한다. 이러한 템플릿은 함수에 적용될 제어 코드 및 메모리 참조의 정의와 사용으로 구성되며, 디바이스 드라이버 정의를 위한 템플릿 규칙은 다음과 같다.

[표 1]디바이스 드라이버를 위한 템플릿 규칙

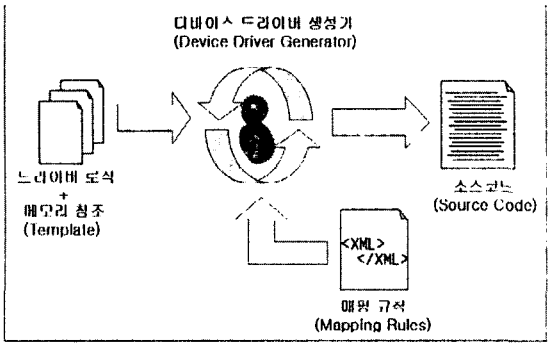
예약어	사용 목적
$[DeviceName]$	제어할 장치의 이름을 기술한다. 대상 디바이스를 명시하기 위해 사용한다.
$[DriverName]$	생성되는 디바이스 드라이버의 이름을 기술한다. .생성되는 소스코드의 파일명 및 구분을 위해 사용한다.
$[HeaderFiles]$	필요한 헤더파일명들을 나열한다. 드라이버 구현에 필요한 헤더파일들을 명시하기 위해 사용한다.
$[Defines]$	상수 선언들을 나열한다. 드라이버 구현에 사용될 상수들을 선언하기 위해 사용한다.
$[ExternalReferences]$	드라이버에 필요한 외부참조들을 정의한다. 드라이버 구현을 위해 필요한 외부 변수나 참조자들을 명시하기 위해 사용한다.
$[MemoryReferences]$	메모리 참조를 위한 참조자들을 정의한다. @ADDRESS_REFERENCE@와 같은 형태로 정의된다. 이후에, XML로 표현된 메모리 매핑 규칙에 의해 임베디드 하드웨어의 메모리 주소로 변환한다.
$[GlobalVariables]$	드라이버 구현에 필요한 전역변수를 정의한다.
$[InitializeModule]$	디바이스 드라이버 초기화시의 작업을 정의한다. 디바이스 드라이버의 커널 적재시 필요한 작업들을 명시하기 위해 사용한다.
$[CleanupModule]$	디바이스 드라이버의 제거시의 작업을 정의한다. 디바이스 드라이버가 커널에서 제거될시 필요한 작업들을 명시하기 위해 사용한다.
$[InterfacesDefine]$	디바이스 드라이버에 구현될 인터페이스 및 함수 매핑을 기술한다. 디바이스 드라이버의 인터페이스 함수 선언과 매핑을 정의하기 위해 사용한다.
$[Func]$[arg][return]	함수를 정의하며, 필요한 매개변수와 반환값이 명시된다. 디바이스 드라이버의 기능을 수행하기 위한 함수들을 정의하기 위해 사용한다. 정의된 함수의 참조를 위해서는 $$[func$$ 형태의 참조지시문을 사용한다.

```
$[DeviceName]$
LED_DEVICE_DRIVER

$[DriverName]$
LED_DRIVER

$[HeaderFiles]$
<linux/kernel.h>
<linux/module.h>
        ...
$[MemoryReferences]$
@LED_DEVICE_ADDRESS@
        ...
$[InitializeModule]$
int error;
if ((major = register_chrdev(0, DEVICE_NAME, &led_fops)) < 0) {
   printk("error : initialize failed");
   return major;
}
printk("initialized : major number = %dWn", major);
        ...
$[InterfaceDefine]$
struct file_operations led_fops = {
    open    : $$led_open$$,
    release : $$led_release$$,
    write   : $$led_write$$,
};
        ...
$[led_memory_mapping]$[][int]
led_ptr  = ioremap((unsigned int)@LED_DEVICE_ADDRESS@,(unsigned int)(1024));
return 0;
```

[그림 2] 디바이스 드라이버 템플릿 적용 예

3.2 디바이스 드라이버의 생성

디바이스 드라이버 템플릿으로 표현된 디바이스 드라이버는 각각의 임베디드 시스템에 맞는 메모리 사용 규칙을 적용하여 실제적인 디바이스 드라이버로 생성된다. [그림 3]은 디바이스 드라이버 제어로직과 메모리 주소에 대한 참조정보를 통한 디바이스 드라이버의 생성과정을 표현하고 있다.



[그림 3] 디바이스 드라이버 생성과정

템플릿 형태로 정의된 디바이스 드라이버는, 디바이스 드라이버 생성기(DDG)에 의해 분석되며, 여기에 메모리 매핑 규칙을 적용하여, 실제적인 디바이스 드라이버의 소스코드를 생성하게 된다. 이 과정에서, 디바이스 드라이버 템플릿내에 정의되어 메모리 주소를 참조하기 위한 참조자들은 메모리 매핑 규칙을 통해 실제 메모리에 매핑된다. 이러한 매핑 규칙은, 디바이스 드라이버 로직에서 사용되는 메모리 참조자의 이름과 그에 대응되는 메모리 주소 그리고 권한들로 명세하여 XML문서 형태로 표현함으로써, 매핑 규칙에 대한 검증을 가능하게 하고, 동시에 확장성을 보장받을 수 있다. 즉, 이것은 임베디드 시스템에 대한 전반적인 정보의 제공을 허용하도록 확장될 수 있다. 메모리 매핑 규칙을 위한 XML 문서의 스키마는 다음과 같이 정의된다.

```
<?xml version="1.0" encoding="euc-kr" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:simpleType name="accessType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="ReadOnly" />
      <xsd:enumeration value="WriteOnly" />
      <xsd:enumeration value="ReadWrite" />
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="referenceType">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[A-Za-z_][A-Za-z0-9_]*" />
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="addressType">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="0x[0-9A-Fa-f]{8}" />
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:group name="MappingGroup">
    <xsd:sequence>
      <xsd:element name="reference" type="referenceType" />
      <xsd:element name="access" type="accessType" />
      <xsd:element name="address" type="addressType" />
    </xsd:sequence>
  </xsd:group>
  <xsd:complexType name="MappingType">
    <xsd:sequence>
      <xsd:element name="devicename" type="xsd:string" minOccurs="1" maxOccurs="1" />
      <xsd:group ref="MappingGroup" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="mapping" type="MappingType" />
</xsd:schema>
```

[그림 4] 디바이스 드라이버의 매핑 규칙에 대한 XML 스키마

XML 스키마에 명시된 태그들은 다음과 같이 사용된다.

태그	설명
<mapping>	루트 태그로 사용한다.
<reference>	템플릿에 정의된 메모리 참조자의 이름을 명시하기 위해 사용한다 메모리 참조자는 C언어의 변수선언 규칙과 동일한 형식으로 정의된다.. <reference>LED_ADDRESS_REF</reference>
<access>	메모리 참조자의 메모리 접근 권한을 명시하기 위해 사용한다. <access> ReadWrite </access>
<address>	템플릿에 정의된 메모리 참조자와 실제 임베디드 시스템에 매핑할 주소를 명시하기 위해 사용한다. 0xFF4E1035와 같은 16진수 형태의 4바이트 표기가 사용된다. <address>0x08300000</address>

4. 결론 및 향후 연구

본 연구에서는, 서로 다른 임베디드 시스템에서의 동일 장치의 제어를 목적으로 하는 디바이스 드라이버를 템플릿을 이용하여 정의하고, 여기에 XML기반의 메모리 매핑 규칙을 적용하여 특정 임베디드 시스템에서 동작하는 디바이스 드라이버를 구현하는 방안을 제시하였다.

템플릿을 이용한 디바이스 드라이버 개발 방안은, 하나의 디바이스에 대한 템플릿을 통해 여러 임베디드 시스템에서 동작가능한 디바이스 드라이버를 자동으로 생성함으로써, 임베디드 소프트웨어의 개발생산성과 유비보수성 향상을 보장한다. 또한, 동일 소프트웨어의 중복개발 방지 및 재사용성 향상을 통해 개발비용의 축소와 개발 기간의 단축을 보장한다. 따라서 이것은, 앞서 언급한 임베디드 소프트웨어 개발 전반에 걸친 생산성 문제와 품질 문제에 대한 하나의 해결책이 될 수 있다.

향후 연구에서는, 본 논문에서 제안한 템플릿을 이용한 디바이스 드라이버 개발 방안에 대한 실제 적용과 개선방안에 대해서 연구가 진행될 것이다.

참고문헌

[1] 김정환, " 전세계 임베디드 S/W 시장 동향," http://kidbs.itfind.or.kr:8888/cgi-bin/WZIN/WebzineRead.cgi?recno=0901013561&db=t_jugidong&menu=12003

[2] The Embedded Software Strategic Market Intelligence Program 2002/2003, Volume II, VDC, 2003.

[3] 한호진, " 디바이스 드라이버의 개념" , http://unix.co.kr/stories.php?story=00/09/21 /9697793

[4] Object Management Group, "Model Driven Architecture," OMG document number ormsc/2001-07-01 2001, http://www.omg.org/mda.

[5] Object Management Group, "MDA Guide Version 1.0.1," OMG document number omg/2003-06-01, 2003. http://www.omg.org/mda.

[6] Jon Siegel and the OMG Staff Strategy Group, "Developing in OMG's Model Driven Architecture," ftp://ftp.omg.org./pub/docs/omg/01-12-01.pdf, 2001.

[7] 이호수, " 유비쿼터스 컴퓨팅의 핵심 기술" , 신기술 경영, 2004.

[8] 최연준, " 템플릿을 이용한 PSM에 독립적인 코드 자동 생성 기법에 관한 연구," 한국정보처리학회 추계학술대회, 2003

[11] Lemon, S., Rossi, K., "On object oriented device driver model," Compcon '95.'Technologies for the Information Superhighway', Digest of Papers, 1995

[12] 이민규, " Understanding MDA," ©Plastic Software, http://www.plasticsoftware.com/download_file/Understanding%20MDA.pdf