임베디드 소프트웨어의 소스 코드 품질 향상을 위한 Practice Patterns의 적용

홍 장 의†

요 약

임베디드 소프트웨어는 하드웨어 플랫폼에 탑재하기 전, 소스 코드에 대한 품질을 검증하는 작업이 매우 중요하다. 임베디드 소프트웨어의 코드 품질을 향상시키기 위해서는 분석 및 설계 단계의 모델에 대한 품질과 생성된 코드에 대한 품질이 관리되어야 한다. 본 연구에서는 임베디드 소프트웨어의 소스 코드 품질을 향상시키기 위한 방법으로 Practice Pattern을 제안한다. 이는 모델링 과정이나 코딩 과정에서 개발자를 가이드 하는 절차 패턴으로써, 모델의 품질과 소스 코드 품질을 향상시키는 방법으로 사용될 수 있다. 제시하는 패턴의 적용은 기능의 정확성 뿐만 아니라 성능, 모듈화, 재사용성 및 이식성 등과 같은 품질 요소들을 향상시킬 수 있을 것으로 보인다.

키워드 : 임베디드 소프트웨어, 소스코드 생성, 소프트웨어 품질, 절차 패턴

Applying Practice Patterns to Improve Source Code Quality of Embedded Software

Jang-Eui Hong†

ABSTRACT

Source code quality is very important that software embedded into product is difficult to change. In order to improve source code quality, it should be considered the quality of analysis and design models as well as the quality of source code. In this paper, we suggest "Practice Pattern" as one of practical techniques to improve embedded software source code quality. Practice pattern is a procedural pattern to guide modeling and coding activities in software development phases. We believe that applying our pattern provides the improvement of optimum performance, modularization, and portability for embedded software source code.

Key Words : Embedded Software, Source Code Generation, Software Quality, Practice Pattern

1. 서 론

임베디드 소프트웨어 개발에 있어서 소스코드를 생성하기 위한 과정은 요구사항으로부터 개발된 소프트웨어 모델을 입력하는 것으로부터 시작된다. 초기 사용자로부터 시스템에 대한 요구사항이 제시되면, 이를 기반으로 요구사항을 분석하고, 소프트웨어를 설계하게 된다. 소프트웨어 설계 과정을 통해 작성된 모델을 기반으로 소스 코드가 생성되며, 생성되는 소스 코드에 대한 품질을 향상시키는 것은 임베디드 소프트웨어 운용 환경에서 매우 중요한 이슈로 부각되고 있다[1, 2, 10].

임베디드 소프트웨어의 소스 코드에 대한 품질이라 함은 소스 코드가 정확한 기능적 요구사항의 제공과 함께 비기능적 요구사항인 메모리 요구사항, 성능 요구사항, 전력소비 요구사항, 그리고 다양한 플랫폼으로의 이식성 및 유지보수성 등을 충족해야 함을 의미한다[3].

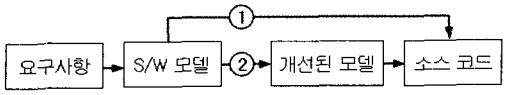
본 연구에서는 소스 코드의 품질을 향상시키기 위한 방법으로 Practice Pattern을 제시한다. 이는 임베디드 시스템에 대한 모델링과 코딩 과정에서 적용될 수 있는 절차 패턴으로, 엔지니어에게 작업 절차에 대한 가이드라인을 제시함으로써 모델의 품질을 향상시킴과 동시에 코드의 품질을 향상시키기 위한 기법이라 할 수 있다.

본 논문에서는 먼저 소스코드 품질 향상을 위한 기법에 관한 기존의 연구를 조사 분석하고, 3장에서는 제시하는 방법을, 4장에서는 적용 절차 및 적용 결과를, 5장에서는 결론을 기술한다.

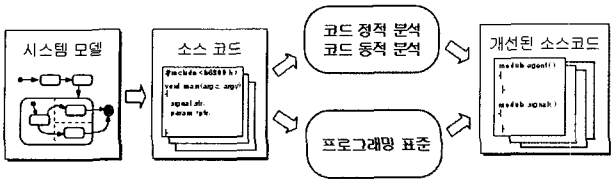
2. 관련 연구

자동 생성되는 소스 코드의 품질을 향상시키기 위한 방법

※ 본 연구는 정보통신부 정보통신연구진흥원이 지원하는 선도기반기술 개발 과제의 지원을 받았음.
† 정 회 원 : 충북대학교 전기전자컴퓨터공학부 조교수
논문접수 : 2005년 9월 14일, 심사완료 : 2005년 12월 12일



(그림 1) 소스 코드의 품질 향상 방안



(그림 2) 코드 수준에서의 품질 향상 방안

은 (그림 1)과 같이 크게 두 가지 접근방법으로 제시될 수 있다. 첫 번째는 코드 수준에서 품질을 향상시키는 방법이다. 즉 생성된 소스 코드 자체에 대한 최적화 과정을 통해 코드 품질을 더욱 향상 시키는 방법이다.

두 번째는 코드를 생성하기 이전에 코드 생성의 기반이 되는 모델의 품질을 향상 시키는 방법이다. 모델의 품질을 향상시킴으로써 그에 따라 생성 되는 소스 코드의 품질을 향상시킬 수 있다. 이와 같이 두 가지 방법을 함께 이용함으로써 소스 코드의 품질 향상을 보다 효과적으로 기대할 수 있다[11].

2.1 코드 수준에서의 품질 향상 기법

임베디드 소프트웨어 소스 코드 품질을 향상시키기 위한 첫 번째 방법은 자동 생성된 소스코드를 최종적으로 타겟에 탑재하기 전에 소스 코드 자체에 대한 최적화 과정을 통해 코드 품질을 더욱 향상 시키는 방법이다.

코드 수준에서 품질을 향상 시키는 방안은 생성된 코드 자체의 정적 및 동적 분석을 수행함으로써, 계산의 병렬성, 메모리의 최적성, 변수 참조의 신속성 등을 향상시키게 된다. 코드의 품질을 향상시키는 또 하나의 방법은 (그림 2)에서와 같이 표준 C 언어 문법 및 프로그래밍 표준을 적용하는 방법이 제시될 수 있다.

소스 코드에 대한 정적 및 동적 분석 기법에 대한 연구의 대표적인 방법은 UC Irvine의 Code Motion 기법이다. 이 방법의 기본 개념은 개발 시스템에 대한 성능을 극대화할 목적으로 제어 흐름을 주관하는 조건절의 전, 후 또는 내부로 일련의 연산을 이동시키는 코드 이동 규칙을 사용하는 것이다[4]. 코드 이동(code motion)에 대한 이동 변환 규칙들은 병렬성 식별을 통한 코드 이동, 조건 분기 내부로의 코드 이동, 조건 변수의 조기 실행 등이며, 이를 통해 연산 속도의 극대화 및 스케쥴 가용성의 확보를 도모한다. UC Irvine 연구에서는 이러한 코드이동 프레임워크를 이용하여 MPEG-1 알고리즘에 적용한 결과 제어부의 복잡도를 나타내는 상태 수와 가장 긴 실행 패스에 대하여 35%에서 50%의 감소 효과를 보여 주었다.

소스 코드 품질 향상을 위한 정적 분석 방법에 대한 또 다른 연구는 Victoria 대학의 포인터 분석에 의한 Cycle Detection 기법이다. 이 연구는 C 언어로 작성된 프로그램 코드로부터 정의된 포인터 연산에 대한 실행 시간을 줄이기 위한 시도로 이루어졌다[5]. 프로그램 코드 상에 포인터가 선언되고 사용될 때, 포인터가 궁극적으로 지시하는 데이터가 무엇이며, 이것을 접근하기 위한 효과적인 방법은 없는 가에 대하여 연구하였다. 이를 위해 참조 경로에 대한 길이를 감소시키는 추론 체계를 이용하였다.

코드 수준에서의 품질 향상을 위한 또 하나의 방법은 코드 작성에 대한 프로그래밍 표준을 적용하는 것이다. 코딩 표준의 적용은 이미 오래전부터 임베디드 소프트웨어 개발에 적용되어 왔으며, 특히 유럽에서는 자동차 제어용 소프트웨어 개발하는 과정에서 표준화된 C 언어 가이드라인-예를 들면, MISRA(Motor Industry Software Reliability Association)-을 사용하도록 함으로써, 하드웨어 플랫폼의 영향을 최소화하고, 부품에 대한 호환성을 보장하는 방법을 취하고 있다[6]. 따라서 임베디드 소프트웨어에 대한 소스 코드 자동 생성 과정에서 생성된 소스 코드가 표준화된 코드 작성 규칙을 준수하는지, 이로 인하여 특정 플랫폼에 의존적인 코드는 사용하고 있지 않은가 등을 자동적으로 점검할 수 있다.

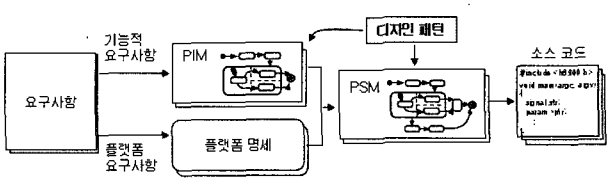
2.2 모델 수준에서의 품질 향상 기법

임베디드 소프트웨어에 대한 소스 코드 자동 생성시 소스 코드 품질을 향상시키기 위한 두 번째 방안은 모델 수준에서 소스 코드의 품질을 향상시키는 방법이다. 즉 모델 수준에서 품질을 향상시킴으로써 그에 따라 자동 생성되는 코드의 품질을 향상 시킬 수 있다.

모델 수준에서 품질을 향상시키는 대표적인 방법은 (그림 3)과 같이 플랫폼에 종속적인 부분과 독립적인 부분을 상호 구분하여 코드 생성의 효율성 및 이식성을 높이는 방법과 임베디드 소프트웨어의 디자인 패턴을 적용하는 방법이 있다.

University of pennsylvania의 CHARON 프로젝트에서는 전처리부(front-end)와 후처리부(back-end)로 나누어 임베디드 소프트웨어에 대한 코드를 생성한다[7]. 전처리부에서는 아키텍처를 나타내는 에이전트(agent)와 에이전트의 제어흐름을 나타내는 모드(mode)를 통해 시스템을 모델링하고, 후처리부에서는 이들 모델을 이용하여 타겟 코드를 생성한다.

모델 수준에서의 품질 향상을 기하고자 하는 또 다른 연구는 임베디드 소프트웨어 디자인 패턴을 적용하는 방법이다. 디자인 패턴은 시스템 설계 도중에 되풀이 되어 일어나는 특정 문제에 대해서 잘 해결된 방안을 정의한 것으로써, 시스템 설계에 디자인 패턴의 적용을 통해 성능 향상, 유연성 등 장점을 얻을 수 있다. 결과적으로 생성된 코드의 품



(그림 3) 모델 수준에서의 품질 향상 방안

질 향상을 가져올 수 있으며, 또한 시스템 설계를 디자인 패턴의 단위로 이해함으로써 시스템 이해에 대한 추상화 수준을 높일 수 있다.

패턴에 대한 연구는 1996년부터 진행된 PTTES(Patterns of Time-Triggered Embedded System) Collection이 대표적인데, 이는 임베디드 시스템 도메인에서 자주 사용되는 디자인 패턴을 수집하고, 몇 번의 수정을 거쳐 정의되었다[8]. 현재 70개 이상의 임베디드 시스템에 대한 모델 수준에서의 디자인 패턴을 정리하고 있다.

이와 같이 제공되는 임베디드 시스템에 대한 설계 패턴은 모델러(modeller)가 시스템을 디자인하는 과정에서 정의된 컴포넌트에 대하여 해당 패턴을 선택하고 시스템 모델에 적용함으로써, 이미 검증된 형태의 모델 구축을 가능하도록 한다. 또한 해당 부분에 대한 별도의 처리 없이 생성할 수 있다. 디자인 패턴의 정의는 패턴에 대한 의미와 패턴이 해결하고자 하는 문제, 패턴과 연관된 지식, 그리고 모델에 사용될 솔루션(solution)을 포함한다. 솔루션에는 모델에 대한 세부 사항과 함께 생성되는 소스 코드를 포함하게 된다[8, 9].

3. Practice Pattern

본 연구를 통해 제시하고자 하는 소스코드 품질향상 방법은 모델의 품질 향상을 위한 Practice Pattern에 대한 것이다. Practice Pattern이라 함은 임베디드 시스템을 개발하는 모델러나 프로그래머가 모델링 또는 코딩 과정에서 수행하는 작업의 절차 패턴이라고 정의할 수 있다. 즉, 다이어그램의 구성 요소를 식별하는 방법이나 모델링 전개 방법, 또는 코딩 준수사항 등을 패턴으로 정의하고, 이들을 모델링 및 코딩 과정에 적용하도록 하는 것이다. 기존에 제시된 디자인 패턴은 모델링 결과 및 산출물 중심의 패턴인 반면 본 연구에서는 행위 및 절차를 정의하는 패턴이라고 할 수 있다.

본 연구에서 제시하는 Practice Pattern은 두 가지 형태로 제시한다. 첫 번째는 모델링 패턴(modeling pattern)이고, 두 번째는 코딩(coding pattern)이다. 이 두 가지 패턴의 목적은 (1) 모델러가 시스템 분석 및 설계 과정에서 적용하는 모델링 휴리스틱(heuristic)에 대하여 체계적으로 가이드 함으로써, 생성된 모델의 품질을 향상시키는 것이며, (2) 소스 코드 생성시, 코딩 패턴의 적용을 통해 다양한 하드웨어 플랫폼에 대한 이식성을 증대시키기 위함이다.

3.1 모델링 패턴(Modeling Pattern)

〈표 1〉 제시하는 모델링 패턴들

패턴명	적용 과정
분할 패턴	시스템 구성 컴포넌트 식별 및 분할 과정에서 적용되는 패턴
프로세스 타입 패턴	시스템 구성 프로세스에 대한 타입을 결정하기 위한 패턴
이벤트 정의 패턴	입력 이벤트에 대한 정의 및 모델링을 가이드하기 위한 패턴
패키징 패턴	모델의 구성요소를 그룹핑 하거나 패키징(packaging)하기 위한 패턴

모델링 패턴은 임베디드 소프트웨어에 대한 분석 및 설계 과정에서 적용되는 패턴이다. 개발 방법론이 구조적 방법을 선택하든, 객체지향 방법을 선택하든지와 무관하게 적용될 수 있는 기본적인 몇 가지 패턴을 제시한다. 제시된 패턴을 요약하면 <표 1>과 같다.

3.1.1 분할(decomposition) 패턴

분할 패턴은 시스템의 구성 요소를 상위 수준에서 하위수준으로 상세화 하는 과정에 적용된다. 구조적 분석 및 설계 과정에서는 자료흐름도에 대한 분할 과정에서 적용될 수 있으며, 객체지향 방법에서는 상태 분할을 통한 시스템 전이도의 작성 과정에서 적용될 수 있다. 분할 패턴에 대한 형식화된 정의는 다음과 같다.

[적용의도] 시스템에 대한 모델링은 일반적으로 계층 구조를 갖는 점진적 상세화 방법을 채택한다. 이 과정에서 분할 패턴은 모델의 구성 요소에 대한 모듈화를 증진시킨다.

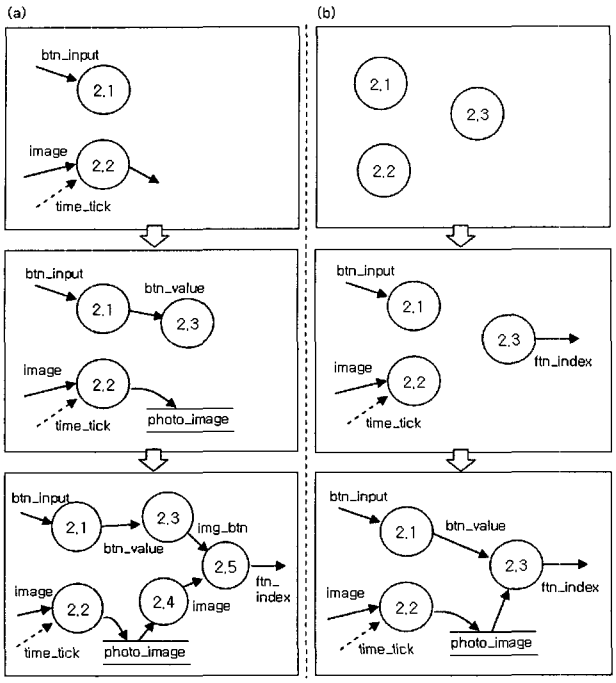
[문제] 모델러는 초기 시스템 분할 과정에서 최상위 시스템을 두 개로 하위 시스템으로 분할할 것인지, 세 개로 분할할 것인지 결정하기를 어려워한다. 특히 시스템에 대한 동작 절차를 잘 알고 있는 모델러는 분할도를 제어흐름도(flowchart)로 작성해버린다.

[솔루션] 분할 패턴을 적용하기 위한 모델링 패턴은 다음과 같다.

① 최상위 시스템의 분할은 하드웨어 컴포넌트 중심이 아닌 모니터링과 액션(action)의 두 부분으로 구성한다.
② 하위 요소로의 분할은 정보의 입출력 흐름이 아닌 단위 기능 중심으로 식별한다.
③ 식별된 기능의 수준(Granularity)을 검사하고 필요시 추가 분할한다.
④ 한 기능에서 입력과 출력이 모두 2개 이상인 경우, 분할과정을 반복 수행한다(단, 타이머 입력과 같은 묵시적인 경우는 제외).
⑤ 기능간의 관계를 정의한다.
⑥ 관계에 해당되는 (입출력)데이터를 정의한다.

[적용예제] 구조적 분석 및 설계과정에서 작성되는 자료흐름도의 경우 (그림 4)와 같이 (a)와 (b)의 두 가지 과정으로 모델을 작성할 수 있다. (a)는 제어 흐름(처리과정)을 중심으로 표현한 것이며, (b)는 시스템을 구성하는 구성요소 중심으로 모델링한 경우이다.

(그림 4)의 모델은 단말기의 입력 이벤트들을 인식하기 위한 요구사항을 모델링 한 것이다. (a)는 모델에 나타난 제어 흐름만을 시스템이 제공하는 것으로 표현하고 있으나, (b)의 경우는 일반적인 처리 흐름을 모두 포함하여 모델링 되었다. 특히 시간의 개념을 모델에 표현하는 경우 올바른 분할 방법이 될 수 없다. 처리 순서를 고려한 모델에서는 프로세스 버블에 대한 기능의 독립성이 약화된다는 단점을 갖는다.



(그림 4) 분할패턴 적용예제

이상과 같이 분할 패턴에 대하여 간략히 설명하였으며, 다른 패턴에 대해서는 그 의의와 문제점, 그리고 적용 가이드라인을 간략히 설명하기로 한다.

3.1.2 프로세스 타입(Process Type) 정의 패턴

프로세스 타입 정의 패턴이라 함은 시스템을 구성하는 프로세스들의 타입을 결정하는 과정에 적용되는 패턴이다. 특히 임베디드 소프트웨어의 경우는 프로세스 타입을 기준으로 태스크 타입이 정의되기 때문에 이들에 대한 결정이 중요하다.

프로세스 타입은 일반적으로 주기적/비주기적(periodic/aperiodic), 동기적/비동기적(sync/asyn), 입력/출력/처리(input/output/function), 그리고 제어(control)의 네 가지 형태에 대한 조합으로 결정된다. 예를 들면, 키보드의 입력을 처리하는 프로세스를 인터럽트 타입의 비동기적 입력으로 볼 것인지, 아니면 폴링(polling) 방식에 의한 주기적인 입력으로 고려할 것인지를 결정해야 한다. 이러한 프로세스 타입의 결정은 태스크 구현에 영향을 주며, 특히 시스템 구현의 복잡성과 성능에 깊은 관련성이 있다.

[적용의도] 임베디드 소프트웨어를 구성하는 프로세스의 타입을 적합하게 정의하는 것은 소프트웨어 동작의 정확성을 보장하고, 태스크들의 자원 공유도를 높이며, 성능을 향상시킨다.

[문제] 모델러는 분석 및 설계 과정에서 도출된 단위 프로세스에 대한 처리 타입을 결정하기 어려워한다. 즉, 시스템의 특정 동작이 구현 수준에서 주기적 또는 비주기적으로 동기적 또는 비동기적으로 모두 가능할 수 있기 때문이다.

[솔루션] 프로세스 타입 정의 패턴을 적용하기 위한 모델링 패턴은 다음과 같다.

① 타이머 입력이 있는 경우, 주기적 프로세스 타입으로 정의한다.

② 입력 이벤트를 발생시키는 장치의 특성을 고려한 프로세스 타입을 결정한다.

③ 사람에 의한 버튼 입력인 경우, 주기적이든 비주기적이든 문제가 없으나, 해당 프로세스가 입력처리만을 담당하는 프로세스인 경우 주기적 프로세스 타입으로 정의한다.

④ 입력되는 이벤트가 어떤 용도로 프로세스에서 사용되는지에 따라 타입을 결정한다.

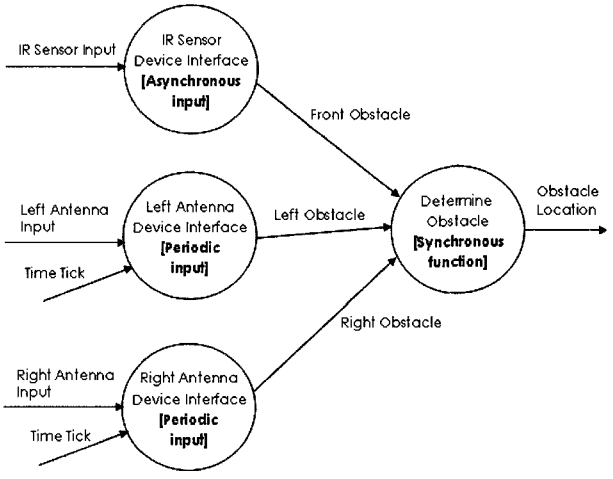
⑤ 입력 이벤트가 출력 이벤트에 반영되지 않고, 프로세스가 단순히 조건에 의한 행위의 분기만을 수행하는 경우 제어 타입으로 정의한다.

⑥ 이전 프로세스로부터의 출력 이벤트가 메모리에 저장되지 않고 직접 입력되는 경우, 동기 함수(synchronous function) 타입으로 정의한다.

⑦ 해당 프로세스가 칩(chip)의 내부에 존재하고, 모델러가 칩의 내부에서 이벤트를 모니터링 한다는 관점에서 타입을 결정한다.

[적용예제] (그림 5)는 임베디드 소프트웨어의 모델에 정의된 프로세스들의 타입을 결정하기 위한 솔루션 적용 예제이다.

(그림 5)는 센서와 안테나로부터 이벤트를 입력받아 인식된 물체(Obstacle)의 위치를 결정하는 기능의 모델이다. 여기서 "IR Sensor Device Interface" 프로세스는 디바이스의 특성에 따라 비동기 입력으로 정의되었으며, 안테나의 경우("Left Antenna Device Interface"와 "Right Antenna Device Interface")는 일정 시간 간격으로 모니터링하는 주기적인 프로세스로 정의되었다.

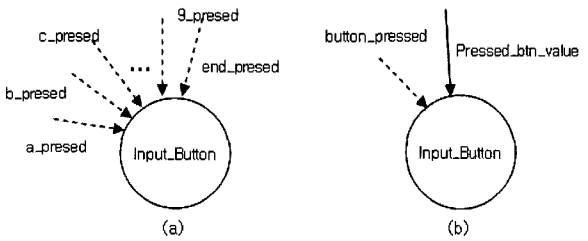


(그림 5) 프로세스 타입 패턴의 적용 예제

3.1.3 이벤트 정의(Event Definition) 패턴

이벤트 정의 패턴은 시스템에 입력되는 이벤트를 어떻게 표현할 것인가에 대한 가이드라인을 제시한다. 이벤트에 대한 정의는 추후 설계 및 시스템 구현 복잡도에 영향을 주기 때문에 합리적인 표현이 중요하다. 특히 다수의 입력 장치를 각각 별도로 모델링하는 경우 매우 복잡한 시스템 모델이 생성되게 된다. 따라서 ① 한 프로세스로 입력되는 이벤트들을 식별하고, ② 이벤트들에 대한 입력 값(value)을 정의하고, ③ 이벤트 처리 타입으로 분류하여, ④ 분류된 이벤트들을 추상화하여 정의한다.

예를 들어 (그림 6)의 (a)의 경우는 다수의 입력 버튼을 갖는 단말기 모델링에서 모든 버튼들이 인터럽트로 정의되고, 각 인터럽트에 대하여 처리 루틴이 작성되는 반면, (b)의 경우는 먼저 인터럽트 신호를 전달하고, 연이어 데이터 입력을 제공함으로써, 입력 장치를 구분할 수 있도록 효과적으로 모델링 되었다.



(그림 6) 이벤트 정의 패턴의 적용 예제

3.1.4 패키징(Packaging) 패턴

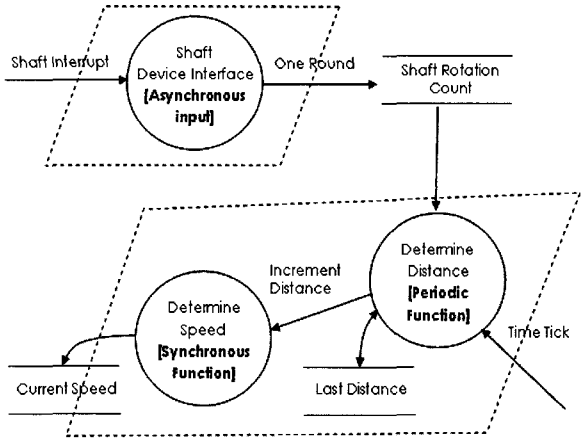
패키징 패턴은 상세 모델의 구성요소들을 플랫폼에 배치하거나, 상위의 소프트웨어 요소로 그룹핑하고자 할 때, 적용하는 패턴이다. 구조적 설계 방법의 경우 식별된 세부 프로세스들을 묶어서 태스크로 정의할 수 있으며, 객체지향 방법의 경우 클래스들을 묶어서 패키지로 정의할 수 있다.

[적용의도] 상세하게 정의된 세부 프로세스를 묶어 태스크 단위(또는 서브 시스템 단위)로 정의하고자 할 때, 어떤 프로세스를 함께 패키징 할 것인지는 태스크간의 인터액션과 태스크 스케쥴링, 자원 공유 등에 영향을 준다.

[문제] 상세화 단계에서 도출된 세부 프로세스에 대한 패키징 과정에서 모델러는 어떤 프로세스를 묶어서 하나의 태스크로 정의할 것인지 쉽게 알지 못한다. 단순히 이웃하는 프로세스들을 묶어서 태스크들로 정의하는 경우, 이들 간의 인터액션이 급격히 증가하는 상황을 초래할 수 있다.

[솔루션] 패키징 패턴을 적용하기 위한 모델링 패턴은 다음과 같다.

① 자율성을 갖는 프로세스는 독립 태스크로 정의한다. 자율성을 갖는 프로세스는 일반적으로 제어 프로세스, 타이머에 의한 주기적 프로세스, 그리고 외부 액터에 의존적인 비동기적 프로세스가 해당된다.



(그림 7) 패키징을 통한 프로세스 그룹핑

② 이전 프로세스의 출력을 입력받는 동기 함수 프로세스는 단일 입력의 경우, 이전 프로세스와 그룹핑한다.

③ 두 프로세스간의 인터액션에 정보 저장소를 사용하는 경우 서로 다른 태스크로 분리한다.

④ 특정 입력 이벤트에 의해 동기적으로 활성화되어야 하는 프로세스는 그룹핑한다.

⑤ 동일한 이벤트에 의해 연속적으로 수행되어야 하는 두 프로세스는 그룹핑한다.

⑥ 데이터 및 제어 측면에서 상호 배타적으로 수행되는 두 프로세스는 그룹핑한다.

⑦ 두 프로세스의 그룹핑은 상호 인터액션을 줄이고, 메모리를 효율적으로 사용하는 방향으로 그룹핑한다.

[적용예제] 상세화 단계에서 도출된 단위 프로세스를 묶어서 태스크로 정의하는 패키징 패턴의 적용 예제이다(그림 7).

(그림 7)은 차량의 바퀴 회전을 통해 속도를 알아내기 위한 기능의 모델이다. 여기서 "Shaft Device Interface"와 "Determine Distance"는 패턴의 적용으로 서로 분리되었으며, "Determine distance"와 "Determine Speed" 프로세스는 그룹핑되어 하나의 태스크로 할당되었다.

이상에서와 같이 간략히 모델링 패턴에 대하여 정의하였다. 이러한 패턴은 모델에 대한 품질을 향상시키기 위한 목적으로 적용될 수 있다.

3.2 코딩 패턴(Coding Pattern)

Practice Pattern 중에서 코딩 패턴은 프로그래머가 소스 코드 작성할 때 고려하기 위한 패턴으로 크게 <표 2>에서와 같이 두 가지를 패턴을 제안한다.

〈표 2〉 제시하는 코딩 패턴들

패턴명	적용 과정
선언 패턴	플랫폼에 대한 이식성을 보장하기 위해 변수 타입의 선언 시 적용하기 위한 패턴
펼침 패턴	시스템에 대한 비기능적 요구사항을 만족시키기 위한 코드 중복성의 허용하는 Unfolding 패턴

3.2.1 선언(Declaration) 패턴

선언 패턴이라 함은 프로그램 코딩시에 프로그래머가 준수해야 하는 코딩에 대한 일종의 규칙으로써 제안한 것이다. 특히 본 연구에서 제시하고자 하는 것은 변수 명명법, 주석 다는 법 등과 같은 일반적인 사항이 아니라 다종의 임베디드 플랫폼에 포팅을 용이하게 할 코딩 가이드라인을 제시하는 것이다.

임베디드 시스템의 플랫폼은 제품의 종류에 따라 8비트, 16비트, 32비트, 또는 64비트 프로세서를 사용하기도 하며, 처리하는 워드(word)의 길이나 변수타입의 메모리 사이즈가 서로 다를 수 있다. 따라서 일반적인 변수 타입의 정의는 플랫폼에 따라 프로그램 코드에 대한 변경을 초래한다. 따라서 <표 3>과 같은 코딩 선언 패턴을 적용해야 한다.

〈표 3〉 소스 코드에서의 선언 패턴 대상들

적용대상	적용 가이드라인
문자 셋	사용하는 문자 코드 값은 ISO 표준(10646-1)에서 정의된 ASCII 값을 기준으로 한다.
변수 타입	변수 타입에서 char, int, short, float, long 등의 기본 타입을 사용하지 않는다.
타입 캐스트	묵시적인 타입 캐스트는 고려하지 않는다. 단순한 형 변환도 명시적으로 캐스팅한다.
헤더파일	헤더파일로 stdio.h, signal.h, time.h 등을 포함하도록 선언하지 않는다.

3.2.2 펼침(Unfolding) 패턴

코드에서의 성능 패턴은 작성된 코드의 형식에 따라 실행 성능이 달라지기 때문에 고려하는 패턴이다. 특히 임베디드 소프트웨어의 경우는 실시간 요구사항을 중요시하기 때문에 코드의 기술 방식에 따라서 성능상의 문제가 유발될 수 있다. 펼침 패턴에 대한 정의는 다음과 같다.

[적용의도] 실시간 특성이 강한 임베디드 소프트웨어의 경우 코딩 스타일에 따라 반응시간의 차이가 발생한다. 따라서 주기적 프로세스(periodic process)들로 구성된 실시간 시스템의 경우, 모듈화나 함수화를 통한 코드 조직화는 피한다.

[문제점] 소스 코드에 대한 이해의 용이성 및 재사용성을 향상시키기 위해 반복적으로 사용되는 코드는 함수로 정의하여 코드를 작성한다. 그러나 이 경우, 실행 시간의 증가를 초래하여 반응시간을 충복시키지 못하는 경우가 발생한다.

[솔루션] 대부분 주기적 프로세스들로 구성된 실시간 시스템의 경우에는 다음과 같은 코딩 패턴을 적용한다.

① 큰 사이즈의 배열 사용을 피한다.
② 이중 루프의 경우 가능한 단일 루프로 변경한다.
③ 단일 호출 함수의 경우 호출 부분에 직접 인라인(Inline) 코딩한다.

```
    :
void motor_stepping( )
{
  SI_16 src_strp, tgt_step;
    :
}

main( )
{
    :
  switch(motor_flag) {
  case "1" : motor_stepping();
             motor_flag ⟨⟨== 2;
             break;
  case "2" : motor_stepping()
             motor_flag ⟨⟨= 3;
             break;
  case "3" :
    :
    :
  case "9" : motor_stepping()
             motor_flag ⟨⟨= 1;
             break;
otherwise : motor_flag = '\0';
}
}
```

(그림 8) 펼침 패턴 적용대상 코드의 예

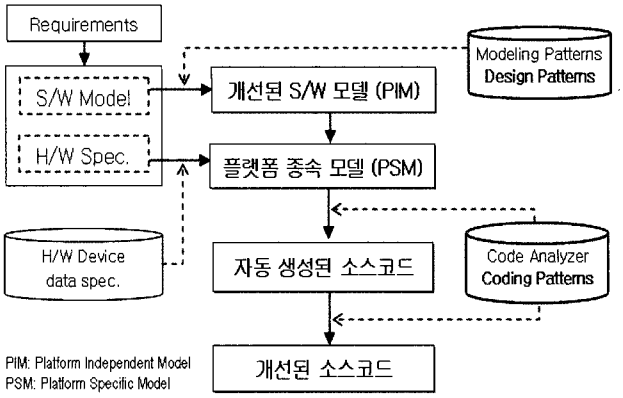
④ Switch Case의 경우 내부에서 함수 호출을 피한다.
⑤ 재귀 함수는 사용하지 않는다.

[적용예제] (그림 8)에 기술한 코드는 switch case 문에 나타난 공통의 동작을 motor_stepping() 함수로 정의하여 호출도록 하였다. 이 경우 모듈화로 인하여 코드에 대한 가독성이 증대되었지만, 반응시간을 만족시키지 못하였다. 따라서 case문 내부에서 motor_stepping 함수를 호출하는 대신 함수 내부의 코드를 모든 case 문에 펼쳐서 중복적으로 작성한다.

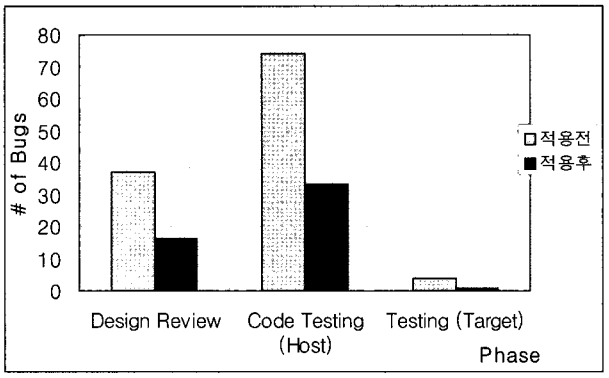
이상에서와 같이 본 연구에서는 임베디드 소프트웨어에 대한 소스 코드의 품질을 향상시키기 위한 방법으로 모델 중심의 개선 방법과 코드 수준에서의 향상 방법으로 Practice Pattern으로 제시하였다. 이러한 패턴은 보다 다양한 소프트웨어 공학적 활동에서 도출될 수 있을 것이다.

4. 향상 기법 적용 절차 및 결과

본 절에서는 제시된 모델의 품질 및 코드 품질의 향상 기법을 임베디드 소프트웨어의 개발 및 소스코드 자동생성 과정에 적용하기 위한 절차를 설명한다. (그림 9)는 패턴의 적용 단계를 보여주고 있다. 제시한 모델링 패턴의 사용은 PIM 모델을 생성하는 과정에 적용될 수 있으며, 코딩 패턴은 소스코드 생성 과정에 적용될 수 있다.



(그림 9) Practice Pattern의 적용 단계



(그림 10) Practice Pattern 적용 결과

임베디드 소프트웨어 분석 및 설계 과정에 모델링 패턴을 적용함으로써, 하드웨어 플랫폼에 독립적인 PIM 모델의 품질을 향상시킬 수 있으며, 최종 소프트웨어 모델로부터 소스 코드를 생성하는 과정에서 코딩 패턴을 적용할 수 있다. 코딩 패턴의 적용은 자동화된 도구를 통해 펼침 패턴의 솔루션들을 코드로부터 자동 식별함으로써, 보다 효과적으로 적용할 수 있다.

이와 같은 모델링 패턴과 코딩 패턴의 적용으로부터 기대할 수 있는 효과는 임베디드 소프트웨어의 정확성, 변경 용이성, 재사용성, 이식성 등에 대한 품질 향상을 도모하는 것이다. 모델링 패턴과 코딩 패턴의 적용으로부터 얻은 실험적 데이터는 (그림 10)과 같다. 본 결과는 핸드 셋에 내장되는 응용 소프트웨어 개발자를 대상으로 적용한 결과이다.

패턴의 적용 결과 적용하기 전보다 50% 이상의 결함 감소 효과를 얻을 수 있었다. 이는 특정 부분에 대한 올바른 모델링은 다른 부분의 모델 품질도 함께 개선시킬 수 있다는 것을 보여주는 것이다. 버그의 수는 패턴의 적용에 양향을 받는 부분만을 고려한 것이 아니라 전체 모델 및 코드에서 발견되는 결함의 수로 나타내었다.

5. 결 론

본 연구를 통해 제시하고자 하는 패턴의 적용이나 관련 연구에서 제시하고 있는 코드 품질 향상 기법은 궁극적으로 빠른 실행시간, 메모리 사용의 최적화, 플랫폼 이식성 향상, 소프트웨어 컴포넌트 재사용성 증대, 모듈화를 통한 변경 용이성 등을 목표로 한다. 이러한 품질 향상 인자들은 임베디드 시스템에 대한 상위 수준에서의 시스템 모델뿐만 아니라, 하위의 코드 수준에서도 품질이 향상될 수 있도록 적용되어야 할 것이다.

임베디드 소프트웨어에 대한 소스 코드의 품질을 향상시키기 위하여 코드 정적분석이나 가상 시뮬레이션과 같은 연구들이 진행되어 왔었다. 그러나 최근에는 모델 수준에서의 향상이 소스 코드의 향상을 가져올 수 있다는 관점으로 전환되고 있다. 따라서 본 연구에서도 이러한 측면에서 모델 수준과 코드 수준에서 적용할 수 있는 품질 향상 패턴을 제시하였다. 특히 임베디드 소프트웨어에 대한 다양한 패턴의 발굴과 패턴 기반의 소프트웨어 개발을 지원하는 자동화 도구의 개발이 향후 필요할 것으로 본다.

참 고 문 헌

[1] David E. Simon, Embedded Software Primer, Addison-Wesley, 1999.

[2] M.P.E. Heimdahl and D.J. Keenan, "Generating code from hierarchical state-based requirements," In Proceedings of the IEEE International Symposium on Requirements Engineering (RE'97), pp.210-219, 1997.

[3] P. Koopman, "Embedded System Design Issues(the Rest of Story), Proceedings of the International Conference on Computer Design (ICCD 96), October, 1996, Austin

[4] S. Gupta, et al, Using Global Code Motion to Improve the Quality of Results for High-Level Synthesis, Technical Report #02-29, Dept. of Info. and Com. Sci., University of California, Irvine, October, 2002.

[5] D.J. Pearce, P. Kelly and C. Hankin, "Online Cycle Detection and Difference Propagation: Application to Pointer Analysis," Software Quality Journal, Vol.12, pp.311-337, 2004.

[6] MISRA, Guidelines for the Use of the C Language in Vehicle Based Software, April, 1998.

[7] R. Alur, J. Kim, I, Lee and O. Sokolsky, "Generating Embedded Software form Hierarchical Hybrid Models," Languages, Compilers, and Tools for Embedded System (LCTES'03), June, 2003, California.

[8] M.J. Pont and M.P. Banner, "Designing embedded systems using patterns: A case study," Journal of Systems and Software, Vol.71, pp.201-213, 2003.

[9] S.M. Yacoub and H.H. Ammar, Pattern-Oriented Analysis and Design, Addison-Wesley, 2004.

[10] 백윤흥, "임베디드 시스템을 위한 ASIP 설계 방법론", 정보처리학회지, 제9권 제1호, 2002년, 1월.

[11] 한국전자통신연구원 연구보고서, 임베디드 S/W 소스코드 자동생성에 대한 기술 및 시장동향 분석에 관한 연구, pp.100-132, 2004년 12월.

홍 장 의

e-mail : jehong@chungbuk.ac.kr

1988년 충북대학교 전산학(학사)

1990년 중앙대학교 컴퓨터공학(공학석사)

2001년 한국과학기술원 전산학(공학박사)

2002년 국방과학연구소 선임연구원

2003년 국가기술지도(NTRM) 및 국제협력제도 작성위원, 과기부

2002년~2004년 (주) 솔루션링크 기술연구소장

2004년~현재 충북대학교 컴퓨터공학 조교수

관심분야 : 소프트웨어공학, 임베디드 소프트웨어, 소프트웨어 품질공학, 소프트웨어 프로세스