논문 2009-3-29

소형 교육용 다관절로봇 RTOS 구현을 위한 디자인 패턴 & 리팩토링 적용*

Applying Design Pattern & Refactoring on Implementing RTOS for the Small Educational Multi-Joint Robot

손현승*, 김우열*, 안홍영**, 김영철**

Hyun-Seung Son, Woo-Yeol Kim, Hong-Young Ahn, Robert Young-Chul Kim

요 약 기존의 교육용 소형 다관절로봇은 펌웨어를 이용하여 개발해왔다. 이런 시스템일 경우 단순동작만 수행할 수 있기 때문에 교육용으로 활용가치가 떨어진다. 그러나 교육용 소형 다관절로봇에 RTOS를 적용하면 다양한 동작의 수행이 가능하다. RTOS를 적용하면 시스템의 효율이 높아지지만 SW 복잡도가 높아져 교육용으로 사용하기 어려운 문제가 있다. 이런 문제를 해결하기 위해서 본 논문에서는 디자인 패턴과 리팩토링을 적용한다. 디자인 패턴과 리팩토링을 적용하여 RTOS를 설계하면 이미 알려진 패턴의 개념이 사용되기 때문에 RTOS의 전문 개발자가 아니어도 이해하기 쉬워진다. 뿐만 아니라 설계가 문서화되기 때문에 기존의 RTOS를 이용하여 새로운 시스템에 알맞은 RTOS로 변경이 용이해 진다. 그래서 본 논문에서는 디자인패턴을 사용하여 RTOS를 설계하고 RTOS 코드에 리팩토링을 적용하였다.

Abstract The traditional small educational multi-joint robots were developed on firmware. In these system's case, we cann't give a chance to educate good practices due on executing just robot's simple movements. But it may be possible for RTOS to control the elaborate movement of the robot with assembling each part on firmware. With this RTOS, we can enhance the efficiency of robot's movements, but too difficult to use the education as increasing the complexity of robot system. To solve the problem, we apply with Design pattern and Refactoring for the Education. Applying robot's design with Design pattern and Refactoring. There may be easily understand what and how to design RTOS for any level ones. We may easily change/upgrade RTOS for new system with this approach. This paper mentions to design RTOS with Design patterns and to apply RTOS's source code with Refactoring.

Key Words : 디자인패턴(Desgin Pattern), 리팩토링(Refactoring), RTOS(Real-Time Operating System), 다관절 로봇(Multi-Joint Robot), 교육용 로봇(Education Robot)

Ⅰ. 서 론

RTOS(Real-Time Operating System)는 하드웨어와 응용프로그램 사이에 위치하여 응용 프로그램이 하드웨어를 쉽게 사용할 수 있도록 한다. 또한 전체적인 시스템의 효율을 극대화시키기 위해 하드웨어 및 소프트웨어 자원(resource)을 관리한다.[1][2] 이렇게 RTOS는 하드웨어 시스템을 제어하고 SW구현이 복잡하기 때문에 많은 개발 시간과 노력이 든다. 뿐만 아니라 시스템 내부의 동작 방법이나 알고리즘을 설명한 자료를 공개하지 않기 때문에 RTOS의 전문 개발자이거나 RTOS 에 정통한 사

*정회원, 홍익대학교 일반대학원
**정회원, 홍익대학교 컴퓨터정보통신
접수일자 2009.05.20, 수정완료.2009.06.10

람이 아니면 큰 어려움이 따른다.

기존의 교육용 소형 다관절로봇은 펌웨어를 이용하여 개발해왔다.[3]-[6] 이런 시스템일 경우 단순동작만 수행할 수 있기 때문에 교육용으로 활용가치가 매우 떨어진다. 그래서 최근에는 교육용 로봇에 RTOS를 적용하여 다양한 동작을 수행할 수 있도록 한다. 하지만 RTOS를 적용하면 시스템의 효율은 높아지지만 복잡도가 높아져 교육용으로 사용하기 어려워지는 문제가 있다.

디자인 패턴은 프로그램 개발에 자주 등장하는 문제를 기술하고 같은 작업을 반복하여 설계하지 않고 여러 번 반복하여 사용할 수 있는 문제에 대한 솔루션을 기술한 것이다.[7][8] 디자인패턴은 소프트웨어를 하나의 건축물로 보는 시각에서 시작되었다. 예를 들어 상세하고 세심한 설계도에 따라서 건물을 짓는다면 튼튼하고 좀 더 쉽게 건물을 완성할 수 있다. 소프트웨어도 이와 같이 내부구조를 상세하게 설계한 후 이를 바탕으로 구현해야 한다는 것이다.

리팩토링은 디자인패턴과는 다르게 소프트웨어를 개발하는데 있어서 구현부를 수정하여 잘 구조화된 소프트웨어를 만드는 것이 목적이다. 즉 밑그림을 그리고 구현을 단계적으로 하면서 끊임없이 개선하여 완성해 가는 것이다. 따라서 프로그램 설계 단계에서 디자인 패턴과 리팩토링을 사용한다는 것은, 그만큼 프로그램의 개발 및 업그레이드가 안정적이다. 디자인패턴과 리팩토링은 좋은 소프트웨어를 개발하기 위한 하나의 도구이지만 보통의 경우 객체지향 설계용도로 기술되어 있다. 이것을 절차지향 프로그램에 적용할 수 있도록 해야한다.

RTOS의 개발에 디자인 패턴과 리팩토링을 사용한다면 이미 알려진 패턴의 개념이 사용되기 때문에 RTOS의 전문 개발자가 아니어도 이해하기 쉬워진다. 뿐만 아니라 설계가 문서화되기 때문에 기존의 RTOS를 이용하여 새로운 시스템에 알맞은 RTOS로 변경이 용이해 질 것이다. 본 논문에서는 디자인패턴과 리팩토링을 RTOS를 설계하고 코드에 적용한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구로서 기존의 디자인패턴과 리팩토링에 대해 언급한다. 3장에서는 디자인패턴과 리팩토링을 RTOS에 적용한 방법에 대하여 설명한다. 4장에서는 적용사례로서 적용한 디자인패턴과 리팩토링을 RTOS구현에 적용한다. 마지막으로 5장에서는 결론 및 향후 연구를 언급한다.

Ⅱ. 관련 연구

1. 디자인 패턴

디자인 패턴[8]은 재사용 가능한 객체지향 설계를 만들기 위해 유용한 공통의 설계 구조를 중요 요소들로 식별하여 이들에게 적당한 이름을 주고 추상화한 것이다. 디자인 패턴은 패턴에 참여하는 클래스와 그들의 인스턴스를 식별하여 역할을 정의하고 그들간의 협력관계를 정의하고 책임을 할당한다. 디자인 패턴은 20개 이상으로 구성되어 있지만 본 논문에서 사용한 브리지 패턴(Bridge pattern), 옵저버 패턴(Observer pattern), 어댑터 패턴(Adapter pattern)에 대하여 설명한다. 브리지 패턴은 서브시스템 사이의 결합도를 줄임으로써 각 서브시스템이 서로에 영향을 미치지 않으면서 변경될 수 있도록 한다. 두 서브시스템 사이에 인터페이스들을 삽입하고, 각 서브시스템이 이들 인터페이스를 사용함으로써 목적을 달성한다. 그림 1은 브리지 패턴의 구조이다.

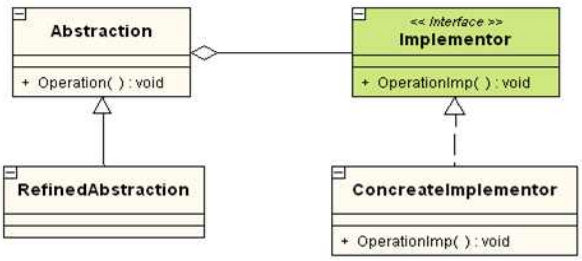


그림 1. 브리지 패턴 구조
Fig. 1. Structure of Bridge pattern

옵저버 패턴은 런타임에 객체가 상태를 변화시킬 때 자신을 등록한 다른 객체들에 통지한다. 통지하는 개체는 이벤트를 자신의 옵저버에 보낸다. 그림 2는 옵저버 패턴의 구조이다.

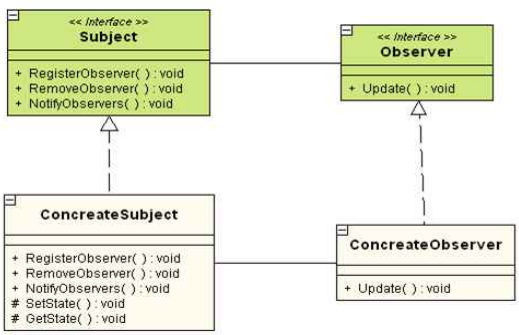


그림 2. 옵저버 패턴 구조
Fig. 2. Structure of Observer pattern

어댑터 패턴은 클래스가 실제로는 지원하지 않는 인터페이스를 지원하는 것처럼 만든다. 이를 통해 리팩토링 없이도 기존의 클래스를 이용해 새로운 클래스를 만들 수 있다. 그림 3은 어댑터 패턴의 구조이다.

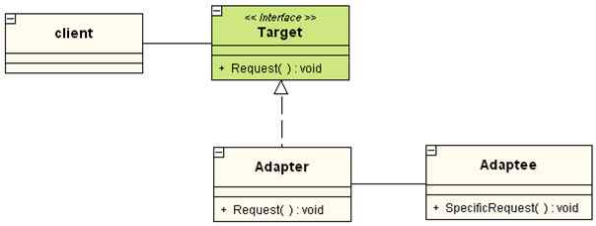


그림 3. 어댑터 패턴 구조
Fig. 3. Structure of Adapter pattern

2. 리팩토링

리팩토링[9]은 디자인패턴과는 다르게 소프트웨어를 개발하는데 있어서 구현부를 수정하여 잘 구조화된 소프트웨어를 만드는 것이 목적이다. 즉 밑그림을 그리고 구현을 단계적으로 하면서 끊임없이 개선하여 완성해 간다는 말이다. 리팩토링은 프로그램을 단계적으로 개선하는 방법들을 모아놓은 것이다. 이를 적용하면 처음부터 과도하게 설계하는 데에 시간을 투자하지 않고 단계적으로 좋은 소스코드로 수정하여 만든다.

Extract Method는 그룹으로 함께 묶을 수 있는 코드 조각이 있으면 목적이 잘 드러나도록 메소드의 이름을 지어 별도의 메소드로 뽑아낸다. 하나의 메소드안에 너무 많은 기능이 들어가지 않도록 적절하게 분리시켜 준다.

Remove Control Flag는 컨트롤 플래그 역할을 하는 변수가 있는 경우 break 또는 return을 대신 사용하는 것이다. 너무 많은 플래그 변수는 코드의 복잡성을 늘리고 오류를 만들어 내기 때문에 제거해야 한다.

Move Method는 메소드가 자신이 정의된 클래스보다 다른 클래스의 기능을 더 많이 사용하고 있다면, 메소드를 가장 많이 사용하고 있는 클래스에 비슷한 몸체를 가진 새로운 메소드를 만든다. 그리고 이전 메소드는 간단한 위임으로 바꾸거나 완전히 제거한다.

Replace Magic Number with Symbolic Constant는 특별한 의미를 가지는 숫자 리터럴이 있으면 상수를 만들고 의미를 잘 나타내도록 이름을 지은 다음 숫자를 상수로 바꾼다. 프로그램에서 사용되는 상수는 주석을 달기 전까지 의미 파악하기 어렵다. 그래서 모든 상수를 문자로 대치한다.

Ⅲ. RTOS에 디자인 패턴 & 리팩토링 적용

1. RTOS에 디자인 패턴 적용

RTOS의 설계를 위해 사용된 패턴은 브리지 패턴(Bridge pattern), 옵저버 패턴(Observer pattern), 어댑터 패턴(Adapter pattern)이다.

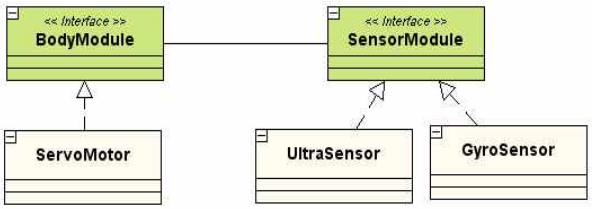


그림 4. RTOS에 적용한 브리지 패턴
Fig. 4. Bridge pattern applied to RTOS

그림 4는 RTOS에 적용한 브리지 패턴의 구조이다. 바디 모듈과 센서모듈의 인터페이스만 맞추어져 있고 바디모듈이 어떻게 수행되는지 센서 모듈은 모르게 되고 센서모듈 또한 바디 모듈의 구현부를 모르게 된다. 둘 간의 연결만 해주면 서보모터는 어떠한 센서들에 관계없이 데이터를 수신 받을 수 있게 된다.

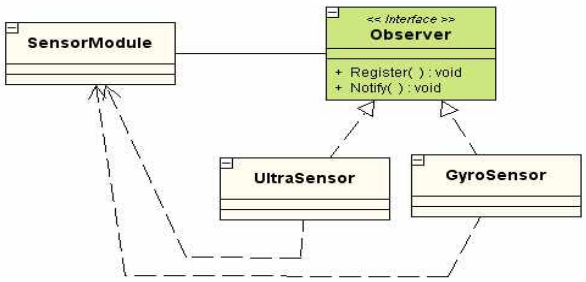


그림 5. RTOS에 적용한 옵저버 패턴
Fig. 5. Observer pattern applied to RTOS

그림 5는 RTOS에 적용한 옵저버 패턴의 구조이다. 센서모듈은 옵저버에게 센서들의 정보들을 받을 수 있도록 등록을 하면 각각의 센서들의 정보가 바뀌었을 때 센서모듈에게 바뀐 정보를 알려주게 된다.

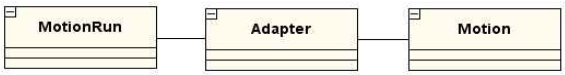


그림 6. RTOS에 적용한 어댑터 패턴
Fig. 6. Adapter pattern applied to RTOS

그림 6은 RTOS에 적용한 어댑터 패턴의 구조이다. 모션 수행을 위해서는 모션데이터가 필요하다. 그러나 기울어진 곳을 움직일 때는 모션데이터의 값이 수정되어야 한다. 그래서 여기에 어댑터 패턴을 이용하여 기존의 모션데이터를 자이로 값을 수신 받아 보정한 후 모션 수행하여 로봇의 구조 변경 없이 동작하게 하였다.

2. RTOS 리팩토링 적용

RTOS의 구현을 위해 사용된 리팩토링 기법은 Extract Method, Remove Control Flag, Move Method, Replace Magic Number with Symbolic Constant 이다.

Extract Method는 보통 이것저것 붙어있는 복잡한 로직일 경우 이를 분리시켜준다. Extract Method를 적용할 대상은 함수의 크기와 관련이 있다. Extract Method를 적용하기 위하서 표 1과 같이 체크리스트를 활용하였다.

표 1. 함수의 라인수 체크 리스트
Table 1. Checlist of line number in function

파일이름	함수이름	라인수
Task.c	TaskCreate	20
	DIOSStart	15
	DIOSInit	11
	DIOS_Scheduler	60

이러한 방법을 통해서 여러 파일 중에서 Extract Mehtod를 수행할 대상을 찾아낸다.

Remove Control Flag는 일련의 BOOL 변수가 컨트롤 플래그 역할을 하는 변수가 있는 경우 break 또는 return으로 대신하는 것이다. 결국 문제가 되는 것은 BOOL형 변수이다. 함수들 중에서 지나치게 많이 사용되는 BOOL형 변수를 찾아서 이를 제거해준다. 이를 분석하기 위해 표 2에 함수내의 플래그 변수 체크 리스트를 사용하여 플래그 변수가 많이 사용되는 함수를 찾아내고 이것을 리팩토링을 적용한다. CheckDIOS함수에서 BOOL형 변수가 하나가 사용되어 이변수를 제거하고 return 문으로 대신하였다.

표 2. 함수내의 플래그 변수 체크 리스트
Table 2. Checklist of flag variable in function

파일이름	함수이름	플래그 변수
DIOS.c	Task1	0
	Task2	0
	Task3	0
	CheckDIOS	1

Move Method는 호출되는 수가 너무 자주 일어날 때 이것을 가장 근접한 함수내로 이동하는 방법이다. 이러한 관계를 분석하기 위해서 표3과 같이 함수 호출 관계 리스트를 사용한다. ∞는 계속적으로 호출됨을 의미한다.

표 3. 함수 호출 관계 리스트
Table 3. Association list of function call

파일이름	함수이름	호출대상	호출 횟수
Timer.c	DIOS_Scheduler	PopRdyQue	∞
	DIOS_Scheduler	PushRdyQue	∞
	DIOS_Scheduler	PopWaitQue	∞

Replace Magic Number with Symbolic Constant는 숫자로 되어있는 의미 없는 값을 문자로 치환한다. 대부분의 하드웨어 제어코드는 비트단위로 처리한다. 그래서 각 비트를 처리하는 문장은 c언어의 쉬프트 명령어를 “<<”를 통해서 문자화로 할 수 있다. 그러므로 “0x07“의 코드를 ”(1 << CS02) | (1 << CS01) | (1 << CS00)“ 이렇게 리팩토링 가능하다.

Ⅳ. 적용사례

1. 하드웨어 구성정보

소형 다관절로봇은 모터제어부에 Atmega128를 센서제어부에 Atmega8535를 사용한 2개의 마이크로컨트롤러를 가지고 있다. 그리고 4측 방향으로 초음파 센서가 장착되어 사방의 물체를 감지 할 수 있도록 되어 있다. 단방향 자이로센서 2개가 장착되어 x축, y축 방향의 기울기를 탐지 할 수 있다. 또한 다리당 3개의 관절이 사용되어 총 18개의 서보모터를 사용한다. 실시간제어를 위해 블루투스가 사용 된다.

2. 브리지 패턴 적용

소형 다관절로봇에는 Sensor 모듈과 Body모듈로 구성되어 있다. SensorModule에서는 센서의 데이터값을 가져와 BodyModule로 데이터를 전송하게 된다. 그림 7은 SensorModule이 Body모듈로 데이터를 전송하는 코드를 구현한 것이다.

```
while(1) {
        PutChar('S');
        _delay_ms(DELAY_UART);
        distance = GetDistance(pluse_end_front);
        PutChar('F');
        _delay_ms(DELAY_UART);
        Put16Dec(distance);
        _delay_ms(DELAY_UART);
        distance = GetDistance(pluse_end_left);
        PutChar('L');
        _delay_ms(DELAY_UART);
        Put16Dec(distance);
        _delay_ms(DELAY_UART);
        distance = GetDistance(pluse_end_right);
        PutChar('R');
        _delay_ms(DELAY_UART);
        Put16Dec(distance);
        _delay_ms(DELAY_UART);
        distance = GetDistance(pluse_end_back);
        PutChar('B');
        _delay_ms(DELAY_UART);
        Put16Dec(distance);
        _delay_ms(DELAY_UART);
}
```

그림 7. SensorModule에서 브리지패턴 구현
Fig. 7. Implementation of the Bridge pattern in SensorModule

SensorModule에서 데이터를 보내면 BodyModule에서는 송신한 데이터를 해석하여 데이터 값을 추출해야한다. 그림 8은 SensorModule에서 보낸 데이터를 해석하는 코드의 일부분이다.

```
while(1) {
        INT8U i = 0, j = 0;
        INT8U fun = 0;
        INT8U sData[4];
        INT8S rdata;
        while(1) {
                rdata = GetCharMCU();
                if(rdata == 'S')
                        break;
        }
        for(i = 0; i < 6; i++) {
                for(j = 0; j < 5; j++) {
                        rdata = GetCharMCU();
                        if(j == 0)
                                fun = rdata;
                        else
                                sData[j-1] = rdata;
                }
                SeonsorSelect(fun,sData);
        }
        OSTimeDly(500);
}
```

그림 8. BodyModule에서 브리지패턴 구현
Fig. 8. Implementation of the Bridge pattern in BodyModule

SensorModule과 BodyModule이 브리지패턴과 같이 독립적으로 구성되어 새로운 하드웨어 추가로부터 자유로워 진다.

3. 옵저버 패턴 적용

Sensor가 수신한 데이터는 BodyModule로 데이터를 보내야한다. 이때 Sensor의 데이터를 수집하기 위해서 옵저버 패턴을 사용한다. 옵저버에 등록된 센서는 자동으로 SensorModule에서 BodyModule로 데이터를 전송할 수 있도록 하고 있다. 그림 9는 센서들의 데이터를 SensorModue에게 알리는 옵저버를 구현한 것이다.

```
while(1) {
        DDRC=0xff;//포트C는 모든 핀을 출력으로 설정
        PORTC=0x00;
        _delay_ms(2);
        DDRC=0xff;//포트C는 모든 핀을 출력으로 설정
        PORTC= BIT4 | BIT3 | BIT2 | BIT1;
        _delay_us(6);
        DDRC=0xff;//포트C는 모든 핀을 출력으로 설정
        PORTC=0x00;
        _delay_us(500);
        DDRC=0x00; //입력 설정

        for(start = 0; start < 1900 ; start++) {
                _getPortC = PINC;
                if((_getPortC & BIT1) == BIT1) {
                        Send(start); }
                if((_getPortC & BIT2) == BIT2) {
                        Send(start); }
                if((_getPortC & BIT3) == BIT3) {
                        Send(start); }
                if((_getPortC & BIT4) == BIT4) {
                        Send(start); }
                        _delay_us(10);
}
```

그림 9. Sensor에서 옵저버 패턴 구현
Fig. 9. Implementation of the Observer pattern in Sensor

4. 어댑터 패턴 적용

로봇에는 일반 평지에서 돌아가는 모션데이터가 저장되어 있다. 이 데이터의 수정 없이 사용하기 위해서 모션데이터를 어댑터 패턴을 사용하여 해결한다. 그림 10은 모션데이터를 실시간으로 수정하는 어댑터를 구현한 것이다.

```
inline void MotorSendAdapter(INT16U data) {
        if(data != g_oldMotorDelay) {
                INT8Ui = 0;
                g_motorCount = 0; //0으로 초기화
                Put16Dec(data);
                for(i = 0; i < 18 ; i++) {
                        g_delay_motor[i] = data - motor_revision[i];
                }
                g_MotorDelay = data;
        }
}
```

그림 10. Motor에서 어댑터 패턴 구현
Fig. 10. Implementation of the Adapter pattern in Motor

5. Extract Method 적용

표 4는 Extract Method를 사용하여 메소드를 추출한 모습이다. 이결과 메소드의 목적이 분명히 들어나 소프트웨어의 복잡성이 줄어든다. 또한 기능의 중복을 줄일 수 있어 최적화가 가능하다.

표 4. 메소드 제거
Table 4. Extract Method

변경전

```
DIOS_API      ERROR TaskCreate(void (*task)(void *pd), void *pdata,
DIOS_STK *ptos,INT16U cycle, INT16U use) {
    DIOS_STK     *psp; INT8U         err, i;
        psp = (DIOS_STK *)DIOS_TaskStkInit(task, pdata, ptos);
        err = DIOS_TCBInit(psp, (DIOS_STK *)0, cycle, use);
        if (err == NO_ERR) {
            ENTER_CRITICAL();
                for(i=0; i < DIOS_TASK_MAX; i++)
                        if(s_DIOS_READY_Q[i] == NULL)
                        {
                                s_DIOS_READY_Q[i] = pData;
                                break;
                        }
                EXIT_CRITICAL(); DIOS_Scheduler();
            }
        }return (err);
}
```

변경후

```
DIOS_API      ERROR TaskCreate(void (*task)(void *pd), void *pdata,
DIOS_STK *ptos,INT16U cycle, INT16U use) {
    DIOS_STK     *psp; INT8U         err;
        psp = (DIOS_STK *)DIOS_TaskStkInit(task, pdata, ptos);
        err = DIOS_TCBInit(psp, (DIOS_STK *)0, cycle, use);
        if (err == NO_ERR) {
            ENTER_CRITICAL();
                PushRdyQue(g_pTCBPoolList->pTCBPrev);
                EXIT_CRITICAL(); DIOS_Scheduler();
            }
        }return (err);
}
```

6. Remove Control Flag 적용

표 5는 Remove Control Flag을 사용하여 코드의 플래그 변수를 삭제하였다. 코드의 복잡성이 줄어들기 때문에 이해도가 높아져 유지보수를 쉽게 할 수 있다.

표 5. 제어 플래그 제거
Table 5. Remove Control Flag

변경전

```
INT8U CheckDIOS()
{
    DIOS_TCB     *pTCB;
        pTCB = g_pTCBPoolHeder;
    INT16U sum = 0;
    INT8U taskCount = 0;
    BOOL  retValue = FALSE;
        while(pTCB != NULL)
        {
         sum += pTCB->nTCBUse/pTCB->nTCBCycle;
                pTCB = pTCB->pTCBNext;
         taskCount++;
        }
    if(taskCount * (pow(2, 1/taskCount) - 1) + sum >= 0)
        retValue = TRUE;
    return retValue;
}
```

변경후

```
INT8U CheckDIOS()
{
    DIOS_TCB     *pTCB;
        pTCB = g_pTCBPoolHeder;
    INT16U sum = 0;
    INT8U taskCount = 0;
        while(pTCB != NULL)
        {
         sum += pTCB->nTCBUse/pTCB->nTCBCycle;
                pTCB = pTCB->pTCBNext;
         taskCount++;
        }
    if(taskCount * (pow(2, 1/taskCount) - 1) + sum >= 0)
        return TRUE;
    return FALSE;
}
```

7. Move Method 적용

표 6은 기존의 Timer.c에 있는 PopRdyQue, PushRdyQue, PopWaitQue, PushWaitQue 메소드를 사용이 빈번한 Task.c로 Move Method 한 것이다. 메소드를 옮겨 코드의 지역성을 높이고 연관성을 줄인다.

표 6. 메서드 이동
Table 6. Move Method

변경전

Timer.c

```
DIOS_TCB *PopRdyQue();
void                PushRdyQue(DIOS_TCB *pData);
DIOS_TCB *PopWaitQue();
void                PushWaitQue(DIOS_TCB *pData);
```

변경후

Task.c

```
DIOS_TCB *PopRdyQue();
void                PushRdyQue(DIOS_TCB *pData);
DIOS_TCB *PopWaitQue();
void                PushWaitQue(DIOS_TCB *pData);
```

8. Replace Magic Number with Symbolic Constant 적용

표 7은 상수로 되어 있는 문자를 Replace Magic Number with Symbolic Constant를 사용하여 문자화 시

킨 것이다.

그 결과 가독성을 높여 유지보수를 쉽게 할 수 있다.

표 7. 매직넘버 심볼릭으로 치환하기
Table 7. Replace Magic Number with Symbolic Constant

변경전	TCCR0 = 0x07;
변경후	TCCR0 = (1 << CS02) | (1 << CS01) | (1 << CS00);

V. 결 론

기존의 교육용 소형 다관절로봇은 펌웨어를 이용하여 개발해왔다. 이런 시스템일 경우 단순동작만 수행할 수 있기 때문에 교육용으로 활용가치가 매우 떨어진다. 그래서 교육용 로봇에도 RTOS를 적용하여 다양한 하드웨어를 사용할 수 있도록 한다. RTOS를 적용하면 시스템의 효율은 높아지지만 복잡도가 높아져 교육용으로 사용하기 어려워지는 문제가 있다. 이런 문제를 해결하기 위해서 본 논문에서는 소형 다관절로봇용 RTOS 설계에 디자인 패턴과 리팩토링을 적용하였다.

하지만 디자인 패턴은 구조적 프로그래밍에 바로적용할 수 없다. 그래서 디자인 패턴을 RTOS에 적용 가능하도록 설계하였다. 또한 적용한 디자인 패턴을 RTOS의 구현 코드에 적용하고 리팩토링으로 코드를 최적화 하였다.

디자인 패턴과 리팩토링을 적용하여 RTOS를 설계한 결과, 이미 알려진 패턴의 개념이 사용되기 때문에 RTOS의 전문 개발자가 아니어도 이해하기 쉬워졌다. 뿐만 아니라 설계가 문서화되기 때문에 기존의 RTOS를 이용하여 새로운 시스템에 알맞은 RTOS로 변경이 용이해졌다.

향후연구과제로 제안한 방법의 클래스 다이어그램과 코드생성의 자동화 도구를 연구 중이다. 또한 RTOS의 재조합을 위해서 모듈화 기능 지원하고 설계와 코드의 정확성을 위해서 둘 간의 추적성도 연구 중이다.

참 고 문 헌

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

[2] Qing Li, 'Real-Time Concepts for Embedded Systems', CMP, 2003.

[3] 김재수, 손현승, 김우열, 김영철, "A Study on Education Softwarefor Controling of Multi-Joint Robot", 한국정보교육학회, Vol. 12, No. 4, 469-476, 2008.12.

[4] 김재수, 손현승, 김우열, 김영철, "A Study on M&S Environment for Designing The Autonomous Reconnaissance Ground Robot", 한국군사과학학회, Vol. 11, No. 6, 127-134, 2008.12.

[5] 손현승, 김우열, 김영철, "Implementation of Technique for Movement Control of Multi-Joint Robot",한국정보처리학회, Vol. 15, No. 2, 593-596, 2008.11.14

[6] 김동우, 손현승, 김우열, 김영철 "Application of M&S(Modeling & Simulation) for The Autonomous Reconnaissance Ground Robot", 국방과학연구소, Vol. 12, 168-171, 2008.10.23.

[7] Bernd Bruegge, Allen H. Dutoit, "Object-Oriented Software Engineering: Using UML, Patterns, and Java Second Edition", Pearson , 2003.

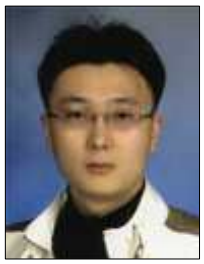
[8] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design patterns: elements of reusable object-oriented software", Addison-Wesley Professional Computing Series, 1995.

[9] M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts, "Refactoring: Improving the Design of Existing Code", Addison-Wesley Object Technology Series, 1999.

※ 본 연구는 지식경제부 및 정보통신연구진흥원의 대학 IT연구센터육성지원사업의 연구결과로 수행되었음" (C1090-0903-0004).
본 연구는 교육과학기술부와 한국산업기술재단의 지역혁신인력양성사업으로 수행된 연구결과임.

저자 소개

손 현 승(정회원)



e-mail : son@selab.hongik.ac.kr

- 2007년 홍익대학교 컴퓨터정보통신 (학사)
- 2009년 홍익대학교 일반대학원 소프트웨어공학전공(석사)
- 2009년 ~ 현재 홍익대학교 일반대학원 박사과정

<주관심분야 : 임베디드 소프트웨어 자동화 도구 개발, 임베디드 RTOS 개발, 임베디드 MDA (Model Driven Architecture) 연구, 모델 검증 기법 연구>

김 우 열(정회원)

e-mail : john@hongik.ac.kr

- 2004년 홍익대학교 컴퓨터정보통신 (학사)
- 2006년 홍익대학교 일반대학원 소프트웨어공학전공(석사)
- 2006년 ~ 현재 홍익대학교 일반대학원 박사과정

<주관심분야 : 상호운용성, 임베디드 소프트웨어 개발 방법론 및 도구 개발, 컴포넌트 시험 및 평가, 리팩토링>

안 홍 영(정회원)

e-mail : hyahn@wow.hongik.ac.kr

- 1991년 University of Florida, Dept. of Electrical & Computer Engineering(공학박사)
- 1975년 ~ 1984 국방과학 연구소, 선임연구원
- 1984년 ~ 1991년 University of Florida, RA
- 1991년 ~ 현재 홍익대학교 컴퓨터정보통신공학과 부교수

<주관심분야 : 데이터통신, 컴퓨터네트워크>

김 영 철(정회원)

e-mail : bob@hongik.ac.kr

- 2000년 : Illinois Institute of Technology(공학박사)
- 2000년 ~ 2001 : LG 산전 중앙연구소 Embedded system 부장
- 2001년 ~ 현재 : 홍익대학교 컴퓨터정보통신 부교수

<주관심분야 : 테스트 성숙도 모델, 임베디드 S/W 개발 방법론 및 도구 개발, 모델 기반 테스팅, CBD, BPM, 사용자 행위 분석 방법론>