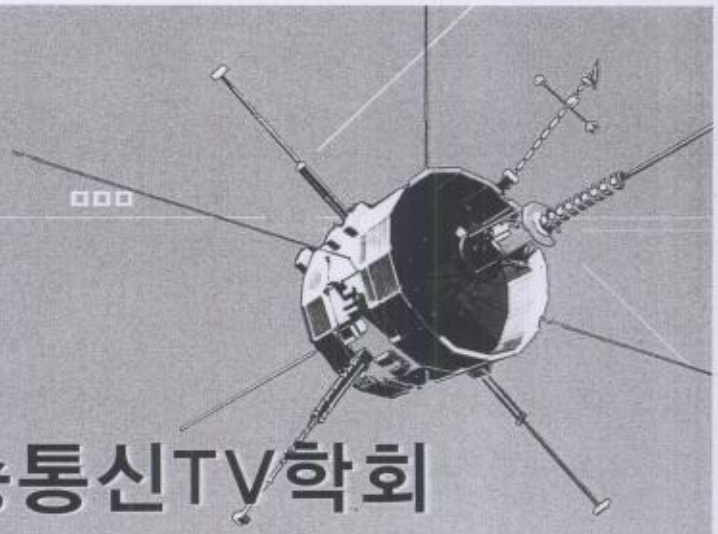

제7권1호

2009년도
한국인터넷방송통신TV학회
춘계학술대회 논문집

일시 : 2009년 5월 29일(금)
장소 : 한국과학기술회관(강남역)
홈페이지 : http://www.iwit.or.kr
주관 및 주최 : (사)한국인터넷방송통신TV학회
(사)인터넷방송통신기술원
동아방송예술대학 성장동력사업단

IWIT (사)한국인터넷방송통신TV학회

소형 다관절로봇 RTOS 구현을 위한 디자인 패턴 적용

Design Pattern on Implementing RTOS for the Small Multi-Joint Robot

손현승, 김우열, 김영철

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

홍익대학교 컴퓨터정보통신공학과

{son, john, bob}@selab.hongik.ac.kr

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 패턴을 사용한다. 디자인 패턴을 이용하여 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 디자인패턴의 브리지, 옵저버, 어댑터 패턴을 사용하여 RTOS를 설계하고 RTOS 코드에 적용하였다.

키워드: 디자인패턴(Desgin Pattern), RTOS(Real-Time Operating System), 다관절 로봇(Multi-Joint 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 SW 구현이 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 사람이 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. 따라서 프로그램 설계 단계에서 디자인 패턴을 사용한다는 것은, 그만큼 프로그램의 개발 및 업그레이드가 안정적이다. 디자인패턴은 좋은 소프트웨어를 개발하기 위한 하나의 도구이지만 보통의 경우 객체지향 설계용도로 기술되어 있다.

RTOS의 개발에 디자인 패턴을 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 디자인패턴을 RTOS를 설계하고 코드에 적용한다.

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

II. 관련연구

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.

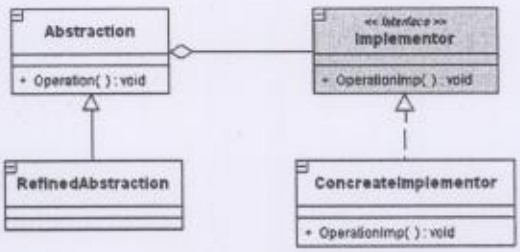

그림 1. 브리지 패턴 구조

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

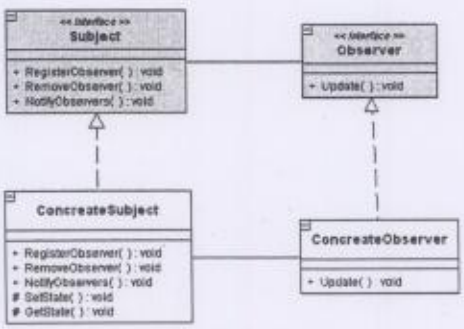

그림 2. 옵저버 패턴 구조

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.

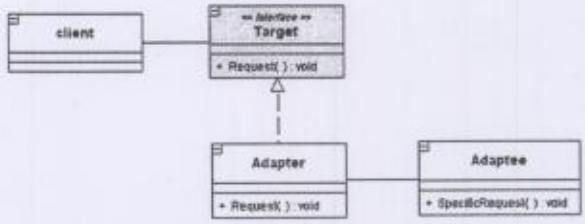

그림 3. 어댑터 패턴 구조

III. 디자인패턴 적용

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

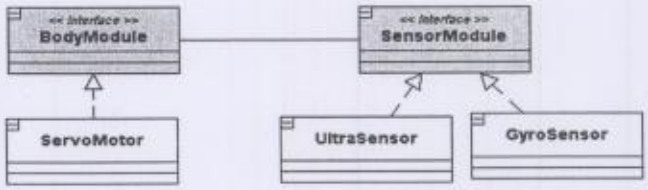

그림 4. RTOS에 적용한 브리지 패턴

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.

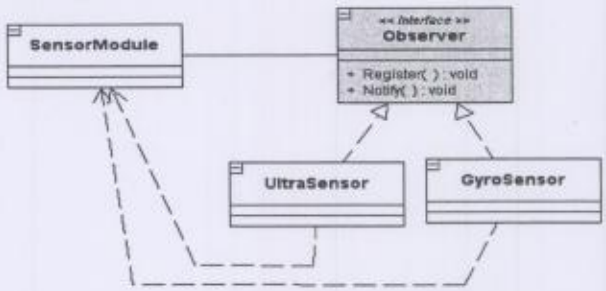

그림 5. RTOS에 적용한 옵저버 패턴

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.

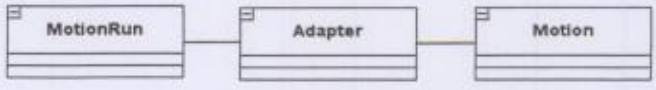

그림 6. RTOS에 적용한 어댑터 패턴

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.

IV. 적용사례

4.1 하드웨어 구성정보

소형 다관절로봇은 모터제어부에 Atmega128를 센서 제어부에 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.

4.2 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. SensorModule 브리지패턴 구현

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. BodyModule의 브리지패턴 구현

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

4.3 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 9은 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) {
         pluse_end_front = start; }
      if((_getPortC & BIT2) == BIT2) {
         pluse_end_right = start; }
      if((_getPortC & BIT3) == BIT3) {
         pluse_end_back = start; }
      if((_getPortC & BIT4) == BIT4) {
         pluse_end_left = start; }
         _delay_us(10);
}
```

그림 9. Sensor의 옵저버 패턴 구현

4.4 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) {
    INT8U  i = 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. Motor의 어댑터 패턴 구현

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 패턴을 적용하였다.

하지만 디자인 패턴은 구조적 프로그래밍에 바로적용 할 수 없다. 그래서 디자인 패턴을 RTOS에 적용 가능하도록 설계하였다. 또한 적용한 디자인 패턴을 RTOS의 구현 코드에 적용하여 보았다.

디자인 패턴을 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 코드 생성의 자동화 도구를 연구 중이다. 또한 RTOS의 재조합을 위해서 모듈화 기능 지원하고 설계와 코드의 정확성을 위해서 둘 간의 추적성도 연구 중이다.

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

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.