실시간 운영체제의 모듈화를 위한 그래픽 기반 AOP 프레임워크

박지용° 김세화 홍성수
서울대학교 전기· 컴퓨터공학부
{parkjy°, ksaehwa, sshong}@redwood.snu.ac.kr

Graphical Aspect-Oriented Programming Framework for Modularizing Real-Time Operating Systems

Jiyong Park°, Saehwa Kim, Seongsoo Hong
School of Electrical Engineering & Computer Science
Seoul National University

요 약

본 논문에서는 실시간 운영체제(RTOS)를 모듈화 하기 위하여 그래픽 기반의 Aspect-Oriented Programming (AOP) 프레임워크를 제시한다. 기존의 컴포넌트, 객체 지향 방법론, 그리고 최근의 AOP들은 RTOS와 같이 많은 기능들이 복잡하게 연관된 소프트웨어를 모듈화 하는 데는 적합하지 않았다. 본 논문의 새로운 AOP 프레임워크는 다음과 같은 특징을 가지고 있다. 첫째, 클래스나 메소드의 단위를 넘어서서 구현되는 기능들이 어떻게 aspect로 모듈화 되는지를 시각적으로 보여준다. 또한 기존의 AOP와 같이 여러 aspect들을 아리저리 옮겨 다닐 필요 없이, 한 곳에서 코드가 어떤 순서로 수행될 지 알 수 있도록 해 준다. 둘째, 코드를 삽입할 위치를 지정하는 단위를 메소드 단위보다 더 세밀하게 하여 메소드의 수행 중간에 특정 aspect를 위한 코드를 삽입할 수 있도록 하였다. 그래서 하나의 메소드에 여러 aspect가 복잡하게 관여하는 경우가 많은 RTOS를 디자인 할 때 특히 유리하다.

1. 서 론

특정 응용에 최적화 된 내장형 시스템을 구성하기 위해서는, 내장형 시스템을 구성하는 요소 중 하나인 실시간 운영체제(RTOS)의 기능들을 다양하게 선택, 제거, 설정 할 수 있어야 한다. 이를 위해 기존에는 컴포넌트 기반 혹은 객체 지향 프로그래밍 기술을 사용하여 RTOS의 각 기능, 예를 들면 파일 시스템, 디바이스 드라이버, 네트워크 프로토콜 들을 모듈화 하였다. 그러나 이 기술들은 RTOS의 모든 기능들을 모듈화 하는 것이 불가능하다는 한계를 지니고 있다. RTOS의 기능들 중 스케줄링, 동기화, 보안 과 같은 기능들은 RTOS 전반의 여러 다른 기능들에 걸쳐져서 (crosscut) 구현이 되어야 하기 때문에 한 파일이나 한 클래스 등으로 모듈화 될 수 없으며, 이 기능들을 구현한 코드들은 흩어져서 존재할 수 밖에 없다. 더군다나 이런 기능들은 RTOS로서의 성능과 기능에 중요한 부분이므로 이런 기능들이 제대로 모듈화 될 수 없다는 것은 기존 컴포넌트 혹은 객체 지향 프로그래밍 기술을 이용한 방법의 큰 약점이 된다.

컴포넌트 혹은 객체 지향 프로그래밍의 이 같은 단점을 보완하고자 고안된 것이 Aspect-Oriented Programming (AOP) [1] 이다. 객체 지향 프로그래밍에서는 클래스, 혹은 메소드라는 단 하나의 모듈화 방법만을 제시하여 여러 클래스나 메소드에 걸쳐서 구현되어야 하는 기능들의 모듈화에는 실패하였다. 그런 반면, AOP에서는 그런 기능들을 aspect라는 또 다른 독립적인 개체로 모듈화 할 수 있도록 하고 있다. 보통, aspect는 특정 기능을 구현하고 있는 코드들과 그 코드들이 삽입되어야 할 위치들로 기술되어 있다.

그러나 기존의 AOP 기술도 RTOS의 기능들을 모듈화 하는데 이용되기에는 다음과 같은 부족한 점들이 있다. 첫째, 기존의 AOP 기술에서는 텍스트 기반의 언어[2, 3]를 사용하기 때문에 aspect들 사이의 관계 또는 aspect와 클래스 사이의 관계를 명확하게 보일 수 없다. 특히 RTOS를 개발하는 중에는 aspect로 모듈화 되기 이전의, 실제로 수행될 코드를 보아야 할 경우가 있는데, 텍스트 기반의 AOP 언어에서는 이것이 불가능하다. 어떤 클래스에 대하여 그 클래스를 대상으로 새로운 기능들을 정의하고 있는 aspect들을 알 수가 없기 때문이다. 또한 그것을 알고 있다고 하더라도 코드가 어떻게 수행될 지 알기 위해서는 그 부분의 수행에 관련된 모든 aspect들을 이리 저리 옮겨가면서 살펴 보아야 하는데 이는 매우 불편하다.

둘째, 기존의 대부분의 AOP 언어에서는 코드가 삽입될 위치를 기술하는 단위가 메소드 단위인데 이는 충분히 세밀하지 않다. RTOS에서는 보통 메소드(혹은 함수)가 매우 길며 여러 가지 기능들을 복합적으로 수행하고 있다. 예를 들어, 프로세스를 새로 생성하는 메소드에서는 스레드, 가상 메모리 관리, 동기화, 파일 시스템, 시그널, 보안등의 여러 기능들에 대한 코드가 혼재한다. 코드가 삽입될 위치를 기술하는 단위가 메소드 단위일 경우 메소드의 시작 전이나 후에만 코드를 삽입할 수 있다. 이런 방식으로는 한 메소드 안에 복잡하게 얽힌 코드들을 따로 떼어 내어서 aspect로 모듈화 할 수 없다. 실제로 RTOS의 메소드의 대부분의 경우 메소드의 앞, 뒤가 아니라 수행 중간에 특정 기능들을 위한 코드가 삽입되어야 하는 경우가 대부분이다. 따라서 RTOS의 기능들을 모듈화 하기 위해서는 메소드 단위보다 더 세밀한 단위로 코드의 삽입 위치를 기술할 수 있어야 한다.

본 논문에서는 기존 AOP 기술의 이러한 한계점을 극복하고자 그래픽 기반의 AOP 프레임워크를 제시한다. 이 프레임워크에서 한 aspect는 선택, 제거, 설정 될 수 있는 하나의 기능을 나타내며, 전체 RTOS는 aspect들의 모임으로 구성되어 있다. 한 aspect안에서는 여러 클래스들을 정의할 수 있으며, 반대로 한 클래스는 여러 aspect에 의해서 정의될 수 있다. 한 클래스의 최종적인 정의는 여러 aspect에 의해서 정의된 내용들의 합집합이

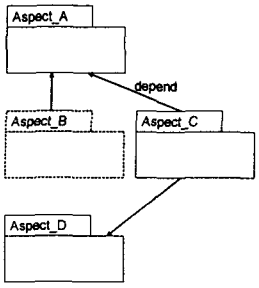

그림1: Aspect 다이어그램

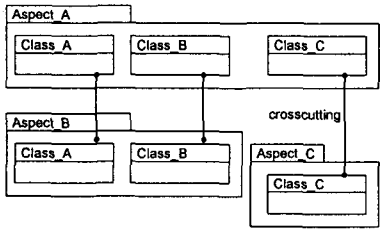

그림2: Aspect-클래스 다이어그램

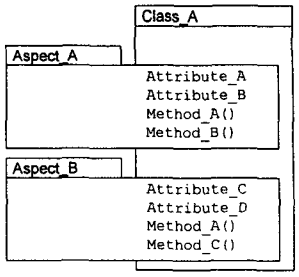

그림 3: 클래스 구조 다이어그램

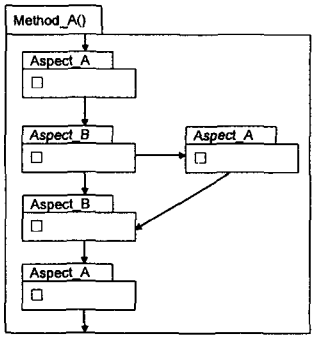

그림 4: 메소드 구조 다이어그램

된다. 이런 개념은 오픈 클래스[4] 에서 빌어왔는데, 오픈 클래스는 클래스를 직접 수정하지 않고, 클래스의 밖에서 클래스에 새로운 메소드를 추가할 수 있도록 한 것이다. 메소드 보다 더 세밀한 단위로 코드의 삽입 위치를 기술할 수 있도록 하기 위하여 메소드의 내부를 기본 블록 (basic block) 들이 모인 것으로 표현한다. 이들 기본 블록들은 각각 서로 다른 aspect에 속한 코드들이다. 본 논문의 AOP 프레임워크는 위와 같은 관계를 시각화하여 aspect들이 서로 어떻게 짜여서 RTOS를 구성하는지를 명확히 보여준다.

2. 그래픽 기반 AOP 프레임워크

AOP 프레임워크는 네 개의 다이어그램으로 이루어져 있다: (1) aspect 다이어그램, (2) aspect-클래스 다이어그램, (3) 클래스 구조 다이어그램, (4) 메소드 구조 다이어그램. 첫 번째 다이어그램이 가장 상위 레벨의 다이어그램이며, 마지막 다이어그램이 가장 하위 레벨의 다이어그램이다. 프로그래머가 새로운 aspect를 추가하고자 할 때에는 위의 순서대로 다이어그램들을 거치게 된다.

2.1. Aspect 다이어그램

가장 상위 레벨의 다이어그램인 Aspect 다이어그램에서는 전체 시스템(RTOS)을 aspect의 모임으로 생각한다. Aspect는 특정 응용에 최적화된 RTOS를 구성하기 위해서 선택되거나, 제거될 수 있는 모듈화된 기능을 뜻한다. 예를 들면, aspect는 UDP 프로토콜, 파일 시스템 할당량 제한 정책 지원, FIFO 스케줄링이 될 수 있다. 또한 aspect 다이어그램에서는 aspect사이의 의존관계를 나타낼 수도 있다. 어떤 aspect는 자신이 의존하고 있는 모든 aspect가 선택된 경우에만 선택될 수 있다. 그림 1은 aspect 다이어그램의 예를 보인 것이다. 이 예에서, 시스템에는 Aspect_A 부터 Aspect_D 까지 총 4개의 aspect가 있다. 점선으로 표시된 Aspect_B는 선택되지 않은 aspect이다. Aspect사이의 화살표는 의존 관계를 나타내는데, 예를 들어, Aspect_C는 Aspect_A와 Aspect_D 모두에게 의존적이다. 즉 이 두 aspect가 선택되어야만 Aspect_C도 선택 가능하다. 그렇지 않다면 Aspect_C는 자동적으로 점선으로 표시되어 선택 불가능한 상태로 고정된다.

2.2. Aspect-클래스 다이어그램

Aspect 다이어그램이 시스템 전체를 대상으로 했다면 Aspect-클래스 다이어그램은 특정 aspect와 직접적으로 관련된 aspect들만을 대상으로 한다. 직접적으로 관련되었다는 것은 두 aspect가 공통된 클래스를 정의하고 있다는 뜻이다. 즉 이 다이어그램은 특정 aspect가 정의하고 있는 클래스들에 대해서 그 클래스를 정의하고 있는 다른 aspect들을 보여줌으로써 aspect들이 어떻게 클래스의 경계를 넘어서서 코드를 모듈화 하는지 나타낸다. 그림 2는 aspect-클래스 다이어그램의 예이다. 이 다이어그램은 Aspect_A를 기준으로 나타낸 것인데, 이 aspect에서는 Class_A, Class_B, Class_C의 3개의 클래스를 정의하고 있다. 이 클래스들을 정의하고 있는 다른 aspect들인 Aspect_B와 Aspect_C가 다이어그램에 나타난다. 그리고 여러 aspect에 걸쳐서 정의된 클래스들은 같은 종류의 것들끼리 crosscutting이라는 관계를 가진다. 즉 crosscutting관계를 가지는 클래스 정의들이 모여 이들의 합집합이 특정 클래스의 완전한 정의가 된다.

2.3. 클래스 구조 다이어그램

클래스 구조 다이어그램에서는 더 세밀하게 들어가서 특정 클래스가 어떤 어트리뷰트와 메소드로 이루어져 있으며 이들이 각각 어떤 aspect에 의해서 정의된 것인지를 보여준다. 그림 3에서

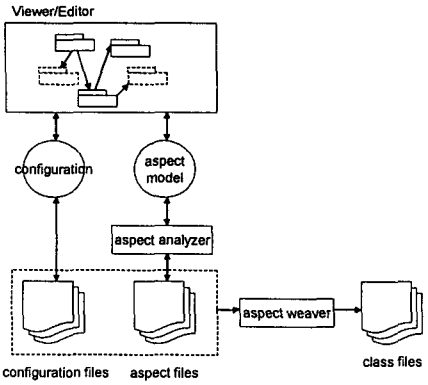

그림 5: AOP 프레임워크의 아키텍처

클래스 Class_A는 Aspect_A와 Aspect_B의 두 aspect에 의해서 정의되었다. 각 aspect에서는 어트리뷰트들과 메소드를 정의하고 있는데, Class_A의 최종 정의는 이 두 aspect에 의해서 정의된 내용들을 모두 합한 것이 된다. 어트리뷰트의 경우 서로 다른 aspect에 의해서 동일한 클래스가 정의될 경우 서로 중복되어서 정의를 할 수 없다. 그러나 메소드의 경우는 Method_A()처럼 중복되어서 정의할 수 있다. 이 경우에 대한 설명은 다음의 메소드 구조 다이어그램에서 한다.

2.4. 메소드 구조 다이어그램

본 논문의 AOP 프레임워크에서는, 한 클래스가 여러 aspect에 의해서 정의될 수 있듯이, 한 메소드 또한 여러 aspect에 의해서 정의될 수 있다. 즉 기존의 AOP는 메소드의 앞 뒤에 새로운 코드를 붙여 메소드를 확장할 수 밖에 없었던 것에 비하여 본 논문에서 제시하는 프레임워크에서는 메소드의 내부 구조를 aspect를 이용하여 자유롭게 확장하는 것이 가능하다. 메소드는 서로 연결된 기본 블록(basic block)으로 이루어져 있다. 기본 블록은 특정 aspect에 속하는 연속된 코드들을 하나로 모은 것이다. 그림 4는 기본 블록으로 이루어진 메소드의 내부를 나타내는 메소드 구조 다이어그램이다. 각 기본 블록은 자신이 속한 aspect의 이름으로 태그가 붙여져 있으며, 그림에는 보이지 않지만, 그 기본 블록에 속한 코드가 어떤 일을 하는 코드인지에 대한 간략한 설명이 있다. 프로그래머는 기본 블록을 클릭하여 그 내부에 있는 실제 코드를 보고 그 자리에서 코드를 수정할 수도 있다. 새로운 코드를 삽입하기 위해서는 이미 존재하는 기본 블록들 사이의 연결선을 끊고 그 사이에 기본 블록을 삽입할 수도 있으며, 한 개의 기본 블록을 두 개로 쪼갠 다음 그 사이에 삽입할 수도 있다. Aspect 다이어그램에서 선택되지 않은 aspect에 해당하는 기본 블록들은 메소드 구조 다이어그램에서는 설정하기에 따라 표시되지 않거나, 회색으로 표시되어 선택 되지 않았음을 나타낸다. 이런 식으로 기존의 AOP에서 보다 코드의 가독성이 훨씬 좋아졌으며, aspect들이 서로 어떤 관계를 가지고 있는지 잘 볼 수 있다.

2.5. 아키텍처

그림 5는 본 논문에서 제시하는 AOP 프레임워크의 아키텍처이다. 이 프레임워크에서는 위에서 설명한 aspect, 클래스, 메소드, 기본 블록 들에 대한 정의와 관계를 모델화 한 aspect 모델을 가지고 있다. 또한 특정 응용에 적합하도록 aspect들을 선택한 설정에 대한 정보를 가지고 있다. 이 aspect 모델과, 현재 설정에 대한 정보를 바탕으로 Viewer/Editor 툴은 위에서 소개된 다양한 다이어그램들 보여준다. 프로그래머는 이를 이용하여 aspect를 선택하거나, 삭제하고, 기본 블록에 코드를 추가하여 RTOS를 특정 응용에 적합하도록 구성할 수 있다.

현재의 설정과 aspect 모델은 각각 설정 파일과 소스 파일의 형태로 저장할 수 있다. 우리는 한 aspect가 한 파일에 저장되도록 계획하고 있으나, 그 파일의 구체적인 포맷은 아직 정하지 않았다. 가장 간단하게는 C언어 에서의 `#ifdef` 를 이용할 수도 있고, AspectJ[3]와 같은 aspect 전용 언어를 이용할 수도 있다. 어느 경우에던지 aspect 모델은 변하지 않는다. Aspect analyzer가 사용되는 언어와, aspect를 기술하기 위한 기술에 따른 차이점을 흡수하는 역할을 한다.

선택된 aspect들에서 공통된 클래스를 찾아내어 하나로 합쳐서 클래스 단위로 기술된 파일을 생성한다. 이 파일들은 순수하게 C/C++, 또는 Java와 같은 언어로 이루어져 있어서 실행 파일로 쉽게 컴파일될 수 있다.

5. 결 론

본 논문에서는 RTOS를 모듈화 하기 위하여 그래픽 기반의 AOP 프레임워크를 제시하였다. 이 프레임워크는 기존의 AOP 기술로는 불가능하였던 다음과 같은 특징을 가진다. 첫째, aspect가 여러 클래스와 메소드를 걸치는 기능들을 어떻게 모듈화 하는지 시각적으로 보여준다. 또한, 여러 aspect들을 이리저리 옮겨 다닐 필요 없이 한 곳에서 코드가 어떤 순서로 수행될 지 알 수 있도록 해 준다. 둘째, 코드를 삽입할 위치를 지정하는 단위를 메소드 단위보다 더 세밀하게 하여 메소드의 수행 중간에 특정 aspect를 위한 코드를 삽입할 수 있도록 하였다. 이러한 특징들은 RTOS의 모든 기능을 모듈화 하고 RTOS를 특정한 응용에 맞도록 구성하기 위하여 반드시 필요한 것들이다.

우리는 현재 이 논문에 제시된 내용을 기반으로 하여 AOP 프레임워크를 구현하고 있다. 그리고, 리눅스나 FreeBSD 같은 실제 운영체제에서 기능들을 자동으로 추출하여 aspect로 모듈화하는 방법에 대하여서도 연구 중이다.

참고문헌

[1] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. Loingtier and J. Irwin, " Aspect-Oriented Programming" , In Proceedings of European Conference on Object-Oriented Programming, 1997.

[2] Y. Coady, G. Kiczales, M. Feeley and G. Smolyn, " Using ApectC to Improve the Modularity of Path-Specific Customization in Operating System Code" , In Proceedings of European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2001.

[3] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, " An Overview of AspectJ" , In Proceedings of the European Conference on Object-Oriented Programming, 2001.

[4] C. Clifton, G. T. Leavens, C, Chambers, and T. Millstein, " MultiJava: Modular open classes and symmetric multiple dispatch for java." . In Procesdings of OOPSLA, 2000.