객체 지향 공정 제어 루프 프레임워크의 설계 및 구현

(Design and Implementation of an Object-Oriented Process Control Loop Framework)

노 성 환 † 전 태 웅 †† 이 승 룡 †††

(Sunghwan Roh) (Taewoong Jeon) (Sungyoung Lee)

요 약 제어 루프는 입력 값들 만으로써는 정확한 출력 값들을 계산하기 어렵거나 불가능한 물리적 공정들을 제어하는 공정 제어 시스템에 핵심적인 구성 요소이다. 본 논문에서는 실시간 공정 제어 응용 시스템의 효율적인 개발을 지원하기 위하여 공정 제어 루프 소프트웨어를 재사용성이 높은 객체 지향 프레임웍으로 설계, 구현한 사례를 기술한다. 본 논문의 제어 루프 프레임웍은 포인트 클래스를 기본 단위로 하여 제어 루프의 공정 변수들과 제어 알고리즘을 캡슐화 함으로써 다양한 구조와 행위를 갖는 공정 제어 응용 시스템으로 쉽게 개조, 확장이 가능하도록 설계되었다. 공정 변수들에 대한 연속적인 재계산을 수행하는 포인트 객체들의 상호 작용을 통하여 요구된 공정의 감시 제어 기능을 유연하게(flexible) 구현할 수 있는 사건/시간 구동적인(event/time-triggered) 포인트 클래스의 설계 패턴이 본 논문의 핵심이다. 본 제어 루프 프레임웍의 설계에는 Observer, Composite, Strategy, Proxy 등과 같은 객체 지향 패턴들이 사용되었다.

Abstract Control loop is an essential part of the process control system that must control physical processes in which it is difficult or impossible to compute correct output value with input values alone. In this paper, we describe the design and implementation of a highly reusable object-oriented control loop framework to support the efficient development of real time process control applications. The basic building block in our control loop framework is the Point class. The Point class encapsulates process variables of a control loop together with control algorithms so that it can be easily adapted and extended to process control applications that have various structures and behaviors. The core of this paper is the design pattern of event/time-triggered Point class that can be used for flexible implementation of monitor and control functions required of target processes through the interaction of point objects performing continuous recomputation. Object oriented patterns such as Observer, Composite, Strategy, Proxy were used to design the control loop framework.

1. 서 론

공정 제어 시스템은 끊임없이 변동하는 외부 환경에 적시에 반응하여 공정의 입력 자원으로부터 원하는 형태의 공정 산출물을 만들어내거나, 공정의 처리 대상들 사이에 요구된 상호 관계를 유지시키기 위하여 공정(들)을 연속적으로 감시, 제어하는 시스템이다.

공정 제어 시스템에서는 입력 값들만으로 정확한 출력 값들을 계산하기 어렵거나 불가능하고 값의 계산에 시간 제약이 따르게 된다. 제어 루프는 이러한 공정 제어 시스템의 핵심 구성 요소이다. [1]

객체 지향 프레임웍은 특정 영역이나 기능 군에 속한 응용 소프트웨어의 개발에 공통적으로 사용 가능한 아키텍쳐를 개조와 확장이 용이한 클래스들로 제공한다. 따라서 프레임웍을 사용하여 응용 소프트웨어를 개발할

· 본 논문은 한국과학재단의 특정기초연구과제(과제번호 : 1999-1-30300-004-3)의 지원에 의해서 작성되었습니다

† 학생회원 : 고려대학교 전산학과
roh@selab.korea.ac.kr
†† 종신회원 : 고려대학교 전산학과 교수
jeon@tiger.korea.ac.kr
††† 종신회원 : 경희대학교 전자계산공학과 교수
sylee@mns.kyunghee.ac.kr

논문접수 : 1999년 9월 2일
심사완료 : 2000년 9월 20일

경우에 개발자는 프레임웍의 일부 클래스들을 수정하고 프레임웍에 포함되지 않은 기능들만을 추가한 후, 프레임웍의 빌딩 블록들과 연결함으로써 완전한 응용 시스템을 만들 수 있게 된다.[2]

근래에 들어, 객체 지향 방법론을 이용한 실시간 시스템의 개발에 대한 연구가 많이 진행되어 왔다. [3]에서는 객체지향과 그래픽 표현(graphic representation), 그리고 정형화한 접근 방법을 통합함으로써 실시간 소프트웨어 개발에서의 요구 사항 명세를 수학적으로 보다 엄격하게 나타내고 여러 구성원들이 쉽게 이해할 수 있도록 하였다. [4]에서는 실시간 영역의 시스템 아키텍처를 객체 지향 기법을 통하여 그래픽으로 나타냄으로써 정확하고 간략한 시스템 모델을 제공한다. 또한 추상화 수준의 모든 단계에서 실행 가능한 모델을 제공함으로 개발 초기에 요구 사항이나 설계의 결함을 찾아낼 수 있다.

[3]이나 [4]의 일반적인 실시간 시스템 개발에 비하여 본 논문에서는 개발 영역을 공정 제어 시스템에 한정하여 재사용성이 더욱 향상된 공정 제어 프레임웍을 개발하고자 한다. 즉, 공정 제어 응용 소프트웨어의 공통 부분들을 높은 재사용성과 유연성을 갖는 객체 지향 프레임웍으로 설계하는 방법을 연구함으로써 공정 제어 시스템의 효율적인 개발이 가능하도록 하는 것이다. 따라서 어플리케이션의 실행 중에도 동적으로 제어루프의 구성이 변경 가능하며 여러 개의 공정 변수들을 갖는 단일 제어 루프 또는 여러 개의 단일 제어 루프들로 구성된 복잡한 구조의 제어 루프 어플리케이션으로 쉽게 확장할 수 있게 된다.

본 논문의 핵심은 제어 루프의 공정 변수들에 대한 연속적인 재계산을 유연하게 지원하는 포인트 클래스의 설계 패턴이다. 본 논문에서 설계된 제어 루프 프레임웍은 가변 부위(hot-spot)가 고정 부위(frozen-spot)로부터 분리된 구조로 구성되어 있다. 고정 부위는 여러 제어 루프들의 공통적인 기능으로 구성되며 어플리케이션의 개발 시에 변경 없이 사용될 수 있는 부분이다. 제어 루프 어플리케이션은 프레임웍의 가변 부위를 시스템의 요구사항에 맞게 개조, 확장하여 프레임웍의 고정 부위에 합성함으로써 완성할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 개발하고자 하는 제어 루프 시스템 프레임웍의 도매인을 분석한다. 제어루프 모델에 대해 설명한 후 재사용성과 유연성이 높은 프레임웍을 개발하기위해 고려되어야 할 요구사항을 살펴본다. 3장에서는 2장의 요구사항을 충족하도록 다양한 구조와 행위를 갖는 공정 제어 응용 시스템으로 쉽게 개조, 확장이 가능한 제어 루프 시스템 프레임웍을 설계한다. 4장에서는 본 논문에서 개발된 제어루프 프레임웍을 사용하여 제어 루프 어플리케이션을 구현한 예를 설명한다. 5 장에서는 공정 제어 소프트웨어를 객체 지향 프레임웍으로 설계하기 위한 기존의 연구들을 본 논문의 연구와 관련하여 설명한다. 끝으로 6장에서는 결론과 앞으로 해결해야 할 연구 방향에 대해 기술한다.

2. 제어루프 시스템의 도매인 분석

2.1 제어 루프 모델

공정 제어 시스템은 제어의 대상인 공정과 공정을 제어하는 제어기로 구성된다. 제어기가 감시 제어하는 공정의 현재 상태는 공정 변수들로 추상화하여 표현할 수 있다. 공정 변수들은 제어 변수(controlled variable), 입력 변수(input variable), 조작 변수(manipulated variable) 및 설정변수(reference variable) 등으로 구분된다. 제어 변수는 공정의 제어 대상에 대한 실제 측정값을 나타내는 공정 변수이다. 입력변수는 공정의 직접적인 제어 대상은 아니지만 제어에 필요한 입력 값들을 나타내는 공정변수 이다. 제어 변수와 입력 변수는 제어기에 의해 감시 되는 공정 변수, 즉 공정의 감시 변수(monitored variable)로서 사용된다. 조작 변수는 제어기에 의하여 직접적으로 조작, 변경이 가능한 공정 변수이다. 설정변수는 제어변수의 목표 치로 설정된 값(setpoint)을 갖는 변수이다.[1]

공정 제어 시스템의 외부 환경은 시스템과 관계없이 연속적인 물리량으로 스스로, 끊임없이 변동하는 비결정적이고 불안정한 성격을 갖고 있으므로 시스템 입력 값과 시스템 내부 상태 값들 만으로써는 정확한 출력 값을 결정하기 어렵거나 불가능하다. 따라서 설정변수에 의하여 명시된 공정의 산출물들의 요구상태를 만족, 유지시키기 위해 제어기는 설정 변수와 감시 변수 값들을 토대로 조작변수의 값을 계산하여 공정으로 내보내고, 공정은 제어기로부터 입력 받은 조작변수의 값에 따라서 조작변수를 조정한다.[1] 이와 같이 공정 제어 시스템은 제어 루프 모델로 볼 수 있다 (그림1). 이러한 제

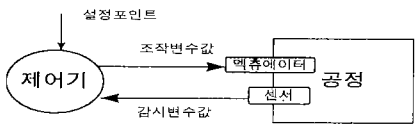

그림 1 제어 루프 모델

어 루프 모델은 제어기와 공정으로 구분된 두개의 컴포넌트들과 상호 비대칭적이고 순환적인 데이타 흐름의 경로를 제공하는 두개의 서로 다른 커넥터로 구성된다.

2.2 제어 루프 프레임웍의 요구사항

본 논문에서 제어 루프 프레임웍의 재사용성과 유연성을 높이기 위하여 고려된 사항들은 아래와 같다.

- 제어 루프 프레임웍의 가변 부분을 개조, 확장함으로써 특정 시스템에 요구된 공정 변수들의 상호작용과 제어 알고리즘을 구현하는 제어 루프 어플리케이션을 쉽게 구성할 수 있어야 한다. 또한, 어플리케이션의 실행 중에도 동적으로 제어루프의 구성을 변경할 수 있도록 한다.
- 하나의 제어 루프가 아닌 여러 제어 루프를 포함하는 복잡한 형태의 공정 제어 어플리케이션으로 쉽게 확장할 수 있도록 한다.
- 제어 루프 어플리케이션이 시간 구동 방식과 사건 구동 방식을 병용하여 실행될 수 있어야 한다.
- 프레임웍으로 만들어진 제어 루프 어플리케이션의 시뮬레이션이 가능하고 시뮬레이션이 완료된 어플리케이션을 실제 공정 제어 환경으로 쉽게 대체될 수 있어야 한다.
- 프레임웍 사용자가 경보 감시 등의 기능을 원할 경우 제어 루프에 쉽게 추가될 수 있어야 한다.

3. 제어루프 프레임웍의 설계

3.1 포인트 클래스

제어 루프는 공정의 상태가 설정치를 유지하도록 액츄에이터를 통하여 공정을 제어하고 센서를 통하여 공정의 제어 상태를 감시한다. 이를 위하여 제어 루프 시스템은 센서로부터 입력된 감시 변수 값의 변화에 적시에 반응하여 조작 변수의 값을 재계산하고 그 결과를 액츄에이터에게 전달함으로써 공정의 제어 변수 값을 변화 시킨다. 본 논문의 제어루프 프레임웍은 제어 루프의 공정 변수들 사이에 필요한 상호 관계를 유연하게 구성할 수 있도록 공정 변수에 대한 제어 및 계산 메커니즘을 공정 변수와 함께 캡슐화 한 포인트 클래스를 기본 구성 단위로 하여 설계되었다.

본 프레임웍의 포인트는 포인트 값이 결정되는 방식에 따라 입력 포인트(input point)와 계산 포인트(computed point)의 두 가지 유형으로 나누어진다. 입력 포인트는 외부의 데이타 소스로부터 입력된 값을 갖는 포인트이다. 계산 포인트는 시스템 내부의 다른 포인트들로부터 계산에 의해 구해진 값을 갖는 포인트이다. 즉, 계산 포인트는 자신의 값을 구하는데 입력 포인트 또는 다른 계산 포인트에 의존하게 되며, 그렇게 의존하는 포인트에 자신을 종속 포인트(dependent point)로 등록(register)하게 된다. 한 포인트의 값이 변하게 되면 그 포인트는 자신에게 등록된 모든 포인트들에게 값의 변화를 알린다. 한 포인트의 변화 사실이 종속 포인트들(dependents 참조변수)에게 알려지게 되면, 종속 포인트는 자신의 값을 구하는데 필요한 모든 포인트들(myPoints 참조변수)의 값을 읽어서 자신의 계산방식(Formula 클래스)으로 포인트 값을 재계산하게 된다. 이러한 방식으로 포인트 값의 변화가 의존 관계로 연결된 모든 포인트들에게 연쇄적으로 전파되면서 재계산이 연속적으로 이루어진다. 그림 2는 특정 포인트 객체 P와 참조변수가 가르치는 객체들 간의 관계의 개념을 나타낸다.

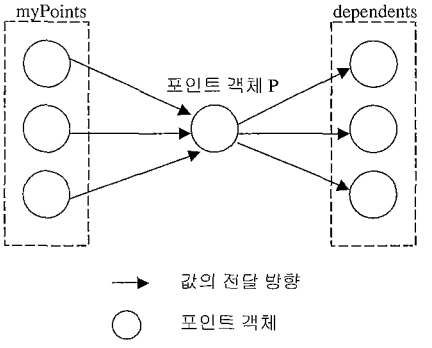

그림 2 포인트 객체와 참조 변수 객체 간의 관계의 개념도

공정 제어 프레임웍을 다양한 구조와 행위를 갖는 공정 제어 응용 시스템으로 쉽게 개조, 확장이 가능하기 위해서는 객체들 사이의 결합도가 낮아야 한다. Observer 패턴[5]과 Composite 패턴[5]을 제어 루프 프레임웍의 핵심 개념인 포인트의 설계에 적용하여 포인트들간의 제어 관계를 나타냄으로써 포인트 객체들 사이에 낮은 결합도를 갖게 된다.

제어 루프 시스템에서 조작 변수, 감시 변수, 액츄에이터 등은 모두 계산 포인트로 볼 수 있다. 이러한 계산 포인트들은 서로 동일한 구조를 갖고 다만 계산방식 함수의 행위만이 서로 다를 뿐이다. 따라서 Strategy 패턴[5]을 적용하여 계산 포인트를 설계한다

그림 3은 위와 같은 행위를 갖는 포인트 클래스 설계 패턴으로써 포인트 클래스와 포인트의 서브 클래스인 입력 포인트와 계산 포인트들간의 상호 관계를 보여준

다. 그림 3에서 계산 포인트는 자신을 종속 포인트로 갖는 포인트 클래스의 공정 변수 값의 변화에 반응하며 Observer패턴을 형성한다. 계산 포인트 클래스는 myPoints 참조 변수를 통하여 다른 포인트 객체들을 구성요소로 갖는 Composite 패턴을 형성한다.

이러한 계산 포인트의 계산 알고리즘은 Strategy 패턴에 따라 동적으로 결합되는 계산방식 클래스(Formula Class)객체에 의해 결정된다. 계산 포인트 클래스는 계산방식 클래스 객체에 대한 참조인 myFormula을 갖는다. 계산방식 클래스는 계산 포인트가 가질 수 있는 여러 알고리즘의 공통적인 인터페이스인 Compute멤버 함수를 제공한다. 계산 포인트는 이 Compute함수를 이용하여 계산방식 클래스의 하위 클래스에 정의된 각 알고리즘을 호출하게 된다. 계산 포인트 객체가 PID 제어 알고리즘[6]을 사용하는 조작변수를 나타내는 경우에 그 계산 포인트 객체는 PIDManipulVarFormula 클래스 객체를 참조하게 된다. 계산 포인트 객체가 여러 센서들의 평균값을 갖는 감시변수를 나타내게 되면 그 계산 포인트 객체는 AverageFormula 클래스 객체를 참조한다. 마찬가지로 계산 포인트 객체가 조작변수의 값을 받아 작동하는 액츄에이터를 나타내면 계산 포인트 객체는 TransferFormula 클래스 객체를 참조한다. PIDManipulVarFormula 클래스의 Compute멤버 함수는 내부에 PID 제어 알고리즘을 가지고 있으며, SetPara 멤버 함수는 프로그램 실행 시작 시에 PID 제어 알고리즘에 필요한 패러미터 값을 읽어 들이게 된다. 이렇게 패러미터의 재구성이 가능하도록 함으로써 프로그램을 다시 작성할 필요 없이 시스템을 쉽게 변경할 수 있다.

그림 3에서 입력 포인트는 중첩된(overload) = 연산자에 의해서 외부로부터 값을 직접 입력 받는다. 그러나 계산 포인트의 값은 자신에게 영향을 주는 포인트들의 값을 사용한 계산에 의해 간접적으로 구해진다. 따라서 그림 2의 myPoints 참조변수는 입력 포인트 객체와 계산 포인트 객체 모두 될 수 있으나 dependents참조변수는 계산 포인트 객체만이 될 수 있다. 입력 포인트에 값이 지정되면 자신의 Notify함수가 호출되고 Notify함수는 해당 입력 포인트의 모든 종속 포인트들의 Update 함수를 호출하게 된다. 호출된 Update 함수는 포인트 값을 재계산한 후 다시 자신의 종속 포인트들의 Update 함수를 호출한다(그림 4). 그림 5는 계산 포인트가 자신의 값을 계산방식 클래스를 통하여 재계산하는 과정이다. 센서로부터 입력된 감시 변수 값들은 이러한 포인트 재계산과 Update 호출의 연쇄적인 실행을 거쳐서 액츄

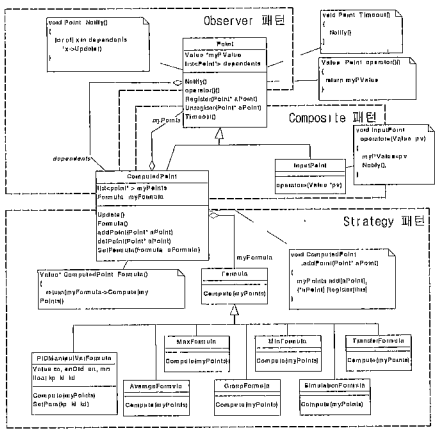

그림 3 포인트 클래스의 설계

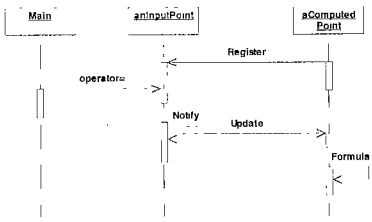

그림 4 포인트 객체들의 인터액션 다이어그램

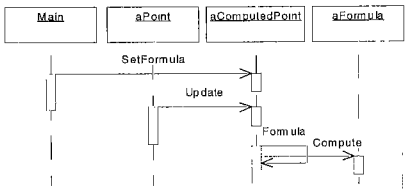

그림 5 계산 포인트의 인터액션 다이어그램

에이터의 구동으로 귀결됨으로써 기대되는 제어 효과를 얻게 된다.

3.2 포인트 밸류(value) 클래스

본 제어 루프 프레임웍에서 포인트의 값은 포인트가 나타내는 공정 변수의 성격에 따라 실수형, 정수형, 또는 부울(Boolean)형 등 여러 가지 타입을 가질 수 있으므로 좀더 유연한 설계를 위해서 공정변수의 제어와 값을 분리하여 포인트 클래스 객체는 제어를 담당하게 하며 밸류 클래스 객체는 값을 나타내게 한다.

또한 포인트 값의 실제 타입에 상관 없이 포인트의 연산이 균일하게 이루어지도록 그림 6에서와 같이 포인

트의 값은 추상 클래스인 밸류(value) 클래스의 인스턴스 변수(myPValue)를 통하여 참조된다. 밸류 객체는 실수, 정수, 또는 부울 값 등이 될 수 있으며 각 하위 클래스들이 연산자를 재정의하여 다형성을 갖게 된다. 그림 6에서 포인트와 밸류 클래스의 관계는 Abstract Class 패턴[7]에 해당한다.

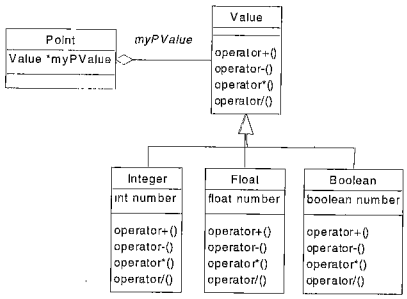

그림 6 밸류 클래스의 설계

3.3 필요 시에만 계산하는 포인트 (Lazy Point) 클래스 [8]

계산 포인트의 값이 사용되는 빈도가 낮을 경우, 그러한 계산 포인트가 의존하는 포인트의 값이 변할 때마다 항상 자신의 값을 재계산하기보다는 계산 포인트의 값이 요청되는 시점에서 재계산하는 것이 효율적이다. 이러한 경우를 감안하여 본 제어 루프 프레임웍의 계산 포인트는 재계산 시점에 따라 Lazy 포인트와 Eager 포인트의 두 가지 서브 클래스로 구분되어 있다. Lazy 포인트는 다른 포인터에 의해 사용될 때만 자신의 값을 재계산하게 된다. 이에 반대되는 개념인 Eager 포인트는 의존하는 포인트(myPoints 참조변수)의 값의 변화가 발생할 때마다 재계산을 하게 된다. 그림7의 클래스 다이어그램은 계산 포인트를 Lazy 포인트와 Eager 포인트로 나누어 설계한 것이다.

Eager 포인트는 Update 함수가 호출될 때마다 자신의 값을 재계산하여 등록된 포인트들에게 알리게 되나, Lazy 포인트는 그림 8에서와 같이 변경 사실만을 기록하고 외부에서 값을 요구할 때까지 재계산을 보류한다. 외부에서 값을 요구할 때는 변경 사실이 있으면 포인트 값을 재계산하고 그렇지 않으면 저장된 포인트 값을 그대로 전달하게 된다. 따라서 불필요한 포인트 계산에 드는 시간의 낭비를 줄일 수 있다.

3.4 제어 루프 프레임웍의 객체 구조와 동적 행위

본 논문의 제어 루프 프레임웍을 사용하면 그림 1의

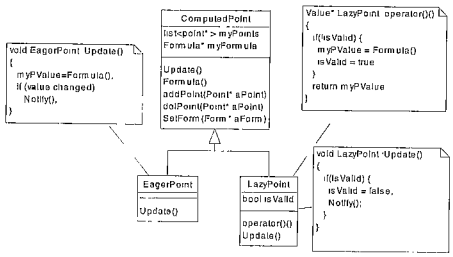

그림 7 Lazy 포인트와 Eager 포인트의 설계

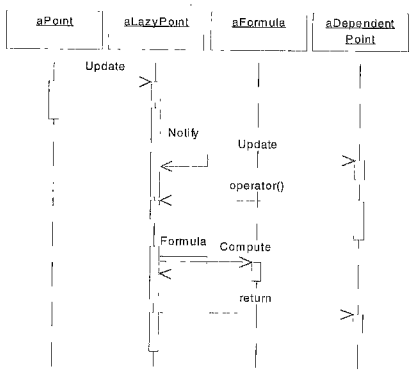

그림 8 Lazy 포인트의 인터액션 다이어그램

제어 루프 모델을 기반으로 한 제어 루프 소프트웨어 컴포넌트의 유연한 설계가 가능하다. 프레임웍을 사용하는 제어 루프 시스템 개발자는 자신의 어플리케이션에 필요한 클래스들을 프레임웍이 제공하는 포인트 클래스 또는 포인트의 서브 클래스들로부터 상속 받아 확장함으로써 원하는 제어 루프 어플리케이션을 구현할 수 있다. 그림 3의 제어 루프 프레임웍으로부터 그림 1의 제어 루프 모델을 구현하는 방식은 다음과 같다. 제어기의 설정 포인트와 공정의 센서는 프레임웍의 입력 포인트의 서브 클래스의 인스턴스로 생성된다. 제어기의 공정 변수들인 입력 변수, 제어 변수, 조작 변수와 공정의 액츄에이터는 프레임웍의 계산 포인트의 서브 클래스의 인스턴스로 생성된다. 제어 알고리즘은 계산방식 클래스의 인스턴스로 캡슐화 되어 구현되서 조작 변수 포인트 객체에 동적으로 결합된다.

그림 9는 본 논문의 제어 루프 프레임웍으로부터 생성되는 전형적인 단일 제어 루프 시스템의 포인트 객체 구조를 나타낸다. 그림 9에서 화살표 선은 포인트 값의 변경이 트리거 되는 방향과 이에 따른 포인트 값의 흐름 방향을 의미한다. 그림9의 설정 변수(setpoint)와 센서(sensor)는 입력 포인트로서, 외부로부터 값이 직접

입력된다. 이와 반면 조작 변수(manipulated variable), 입력 변수(input variable), 액츄에이터는 계산 포인트로서, 의존하는 포인트 값들(myPoints 참조변수)을 사용한 계산에 의해 값이 구해진다.

그림9에서 감시 변수 포인트인 입력 변수(Input Variable)와 제어 변수(Controlled Variable)는 센서 값의 변화에 반응하여 자신의 값을 재계산한 후 이를 조작 변수 포인트에게 알리고 조작변수 포인트는 다시 이에 반응하여 참조 변수(설정) 포인트 값과 감시 변수 값을 사용하여 자신의 값을 재계산한 후 이를 액츄에이터에게 알리는 연쇄 반응이 일어난다.

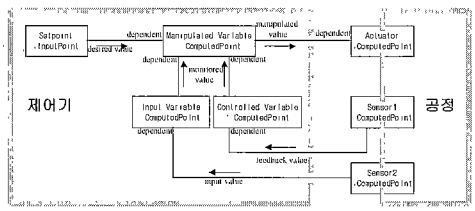

그림 9 전형적인 제어 루프 시스템의 구조

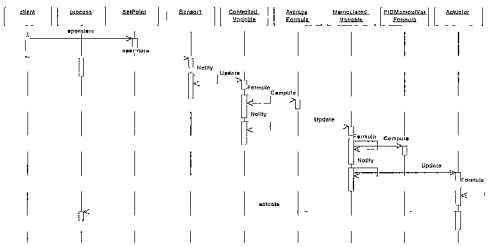

그림 10 포인트 객체들의 인터액션 다이어그램

그림 10은 그림 9의 제어 루프에서 설정 변수에 값이 입력되어 있을 때 제어기가 센서로부터 공정의 상태 값을 감지하여 액츄에이터를 구동하는 과정에서 일어나는 포인트 객체들의 상호 작용을 보여준다.

3.5 시간 구동 포인트 클래스의 설계

제어 루프 시스템은 공정의 상태 변화에 반응하는 사건 구동 방식 외에도 공정의 상태 변화가 없더라도 일정한 시간 간격으로 공정의 제어 상태를 지속적으로 감시하여 공정의 설정치를 요구된 시간 안에 만족시키거나 계속 유지되도록 시간 구동 방식이 함께 지원되어야 한다. 이를 위하여 본 제어 루프 프레임웍에서는 클락과 타이머 클래스가 그림 11에서와 같이 포인트 클래스의 서브 클래스로 정의되어 있다.

시간 구동 센서 객체나 경보 감시 기능을 갖춘 조작 변수 객체처럼 타이머가 필요한 포인트 객체는 자신이 필요로 하는 시간 간격을 갖는 타이머 객체를 생성하여 생성된 타이머 객체에게 자신을 등록시키게 되면 타이머 객체는 지정된 시간 간격마다 등록된 포인트 객체의 Timeout 멤버 함수를 호출하여 등록한 시간이 경과되었음을 알린다.

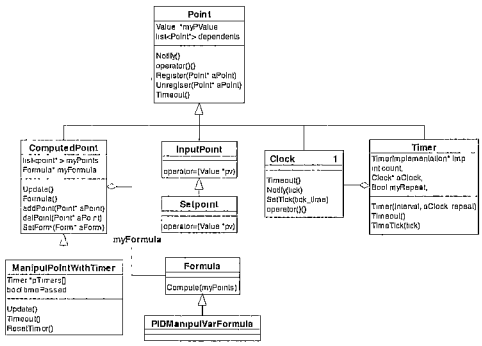

그림 11 클락 및 타이머 클래스와 조작 변수 클래스의 설계

타이머 객체가 작동되기 위해서는 타임 틱(tick)을 발생시키는 클락 객체가 필요하다. 클락 클래스는 포인트 클래스로부터 상속을 받아 만들어진다. 클락 클래스는 제어루프 시스템에서 하나의 인스턴스만이 존재하도록 Singleton 패턴[5]으로 설계되어 있다. 클락 객체는 자신에게 등록된 타이머 객체들에게 순서대로 틱을 계속해서 보내주게 된다. 타이머 객체는 틱을 받아 자신의 시간 간격(interval)이 경과되었는가를 계산하게 된다.

제어 루프의 조작 변수 포인트는 설정 변수(setpoint) 값과 감시 변수(monitored variable) 값을 토대로 자신의 조작 변수 값을 계산한 후 이를 액츄에이터에게 전달한다. 만약 감시 변수 값이 일정 시간이 지난 후에도 설정치에 도달하지 못하면 경보 상태를 알려야 할 필요가 있을 수 있다. 본 제어 루프 프레임웍에서는 이러한 성격의 조작변수를 나타내기 위해 타이머를 갖는 조작변수 클래스(ManipulPointWithTimer)를 계산 포인트의 서브 클래스로 제공한다.

실행 시에 입력 포인트인 설정 변수 객체에 설정치가 입력되면, 설정 변수 객체는 조작변수 객체의 Reset-Timer함수를 호출한다. ResetTimer 함수가 호출되면 조작 변수 포인트는 새로운 타이머를 동적으로 생성하고 생성된 타이머에 자신을 등록한다. 일정시간이

경과하면 타이머는 조작변수 객체의 Timeout함수를 호출하여 조작변수에게 시간이 경과되었음을 알린다. 이 시간 이후부터는 감시 변수가 조작변수 객체의 Update 함수를 호출할 때마다 조작 변수는 감시변수 값이 설정치에 도달했는가를 계산하여 도달하지 못한 경우엔 경보 상태를 알리게 된다 (그림 12).

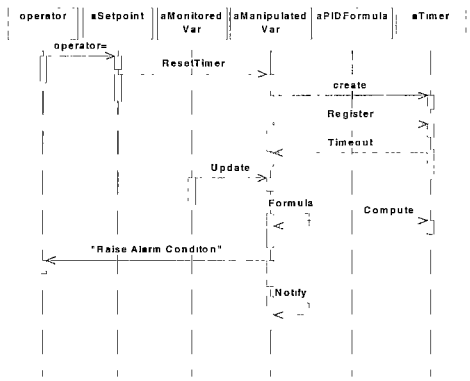

그림 12 시간 구동 조작 변수의 인터액션 다이어그램

제어 결과에 대한 경보가 필요하지 않은 조작 변수 객체는 단순히 계산 포인트 클래스의 인스턴스로 생성된다.

3.6 시뮬레이션 공정의 설계

제어 시스템에서는 시뮬레이션이 매우 중요하다. 예를 들어 공장의 실제 제어기나 기계를 사용해서 공정제어 어플리케이션을 테스트, 디버그 할 수 없으므로 제어 시스템이 실제 운용 환경에서 사용되기 전에 시스템이 정확한가를 검사할 수 있어야 한다.

시뮬레이션이 가치를 발휘하기 위해서는, 시뮬레이션 환경으로 구성된 시스템이 실제 시스템으로 변경될 때 생기는 변화가 되도록 적어야 한다. 너무 많은 변화가 생기게 되면 시뮬레이션으로 테스팅할 때 얻어지는 결과를 믿을 수 없게 된다. 또한 변화가 발생하더라도 일부 영역에서만 발생해야 하며 시뮬레이트 되지 않은 부분은 영향을 받지 않아야 한다.

본 논문에서는 시뮬레이션 환경을 시뮬레이션 포인트(SimulationPoint)로 추상화하여 액츄에이터와 센서의 관계를 시뮬레이션 한다. 시뮬레이션 포인트는 가상 공정의 상태를 나타낸다. 액츄에이터가 공정에 가하는 효과는 시뮬레이션 클래스 객체에 전달되며 시뮬레이션 클래스 객체는 그 효과를 바탕으로 시뮬레이션 된 값을 일정한 피드백 지연 시간 후 센서에 전달하게 된다(그림 13). 따라서 프레임웍으로 만들어진 어플리케이션을 실제 환경에서 실행하게 될 때 시뮬레이션 포인트만 프레임웍의 실제 공정으로 대치하면 된다. 그림 14의 클래스 다이어그램의 시뮬레이션 포인트 클래스에서 myPoint 객체들은 액츄에이터 객체들이고, 종속 포인트 객체들(dependents)은 센서 객체들이다. 그림 15는 시뮬레이션 포인트의 인터액션 다이어그램을 나타낸다. 액츄에이터 객체가 시뮬레이션 포인트 객체의 Update()를 호출하게 되면 시뮬레이션 객체는 자신이 참조하고 있는 시뮬레이션 계산방식(SimulationFormula) 객체를 통해 시뮬레이션 값을 계산하게 된다. 시뮬레이션 포인트 객체에서 동적으로 생성된 타이머 객체가 피드백 지연 시간 경과 뒤 시뮬레이션 포인트의 Timeout함수를 호출하게 되면 종속 포인트인 센서 객체들에게 시뮬레이션 된 값이 전달된다.

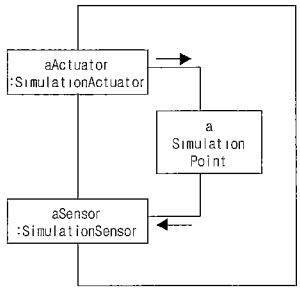

그림 13 시뮬레이션 포인트의 개념

제어 루프의 센서 클래스 객체는 불안정한 외부 환경의 연속적인 물리량을 입력 받아 제어 루프의 감시변수에 전달하는 역할을 한다. 본 제어 루프 프레임웍은 시뮬레이션 환경과 실제 공정 환경에서 모두 사용될 수 있도록 센서가 시뮬레이션 환경에서 작동되는 시뮬레이션 센서와 실제 외부 환경에서 작동되는 실제 센서로 나누어 설계되어 있다.

제어 루프 프레임웍에서 만들어진 실제 어플리케이션은 시뮬레이션 환경과 실제 환경에서 모두 사용될 수 있으므로 그림 14와 같이 Proxy 패턴[5]을 이용하여 센서 클래스를 설계하여 클라이언트 입장에서 시뮬레이션 센서와 실제 센서를 동일하게 사용할 수 있도록 한다.

프락시 센서(ProxySensor) 객체는 제어루프 시스템이 시뮬레이션 환경인가를 판단해서 시뮬레이션 환경인 경우에는 시뮬레이션 센서 객체를, 실제 환경인 경우에는 실제 센서 객체를 구동 시킨다. 시뮬레이션 센서(SimulationSensor)는 시간 구동 센서와 사건 구동 센

서의 두 가지 서브 클래스를 갖는다. 시간 구동 센서는 타이머에 등록되어 일정시간 단위마다 타이머가 센서의 Notify함수를 호출하여 센서의 종속 포인트들에게 현재의 센서 값을 전달하게 된다. 이에 반해 사건 구동 센서는 센서 값이 변경될 경우에 자신이 Notify함수를 호출하여 값을 전달한다.

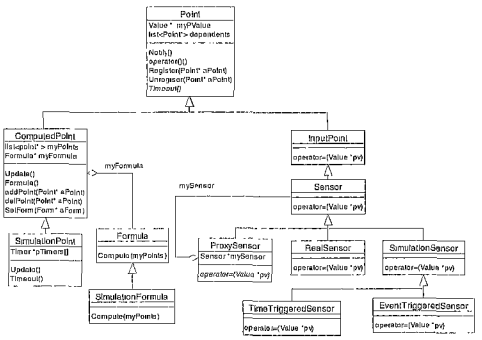

그림 14 시뮬레이션 포인트와 센서의 설계

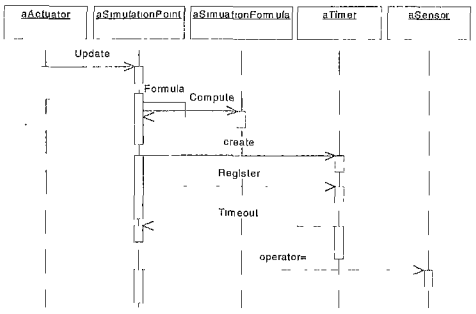

그림 15 시뮬레이션 포인트의 인터액션 다이어그램

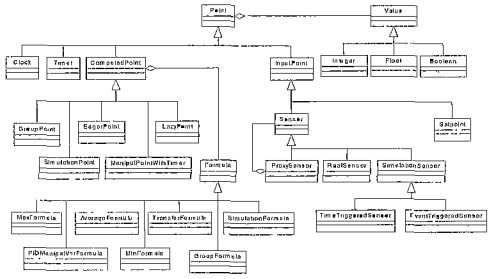

그림 16 제어 루프 프레임웍의 전체 설계

3.7 제어 루프 프레임웍의 전체 클래스 다이어그램

그림 16은 지금까지 설명한 제어 루프 프레임웍 클래스들 사이의 전체적인 상속과 합성 관계를 요약하여 보여준다. 본 논문의 프레임웍 설계에서의 기본 개념은 포인트 값의 연속적인 재계산이므로 제어 루프 프레임웍을 구성하는 대부분의 중요한 클래스들은 포인트 클래스로부터 상속을 받아 만들어지는 것을 그림 16에서 볼 수 있다.

4. 제어 루프 프레임웍을 사용한 샘플 어플리케이션 구현 예

본 장에서는 본 논문에서 개발된 제어루프 프레임웍을 사용하여 MS도스 상에서 실행하는 제어 루프 어플리케이션을 구현한 예를 설명한다. 구현된 샘플 어플리케이션에서 클라 클래스와 타이머 클래스는 도스 운영체제가 제공하는 시스템 서비스 콜을 사용하였다.

그림 17은 방안 온도를 제어하는 난방 제어 루프 모델이다. 그림 17에서 제어기는 설정 변수로 설정된 방안 온도를 유지하기 위한 난방 제어기이고, 공정은 제어기에 의해 감시, 제어되는 난방기이다. 제어 변수는 센서에 의해 측정된 본 난방 제어루프가 제어하고 있는 방안의 현재 기온을 의미한다. 측정된 기온 값은 제어기에게 피드백 되어 액츄에이터에 대한 조작 변수 값을 계산하는데 사용된다.

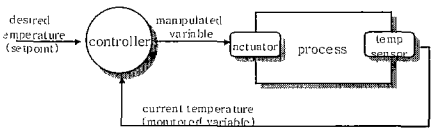

그림 17 방안 온도 제어 루프

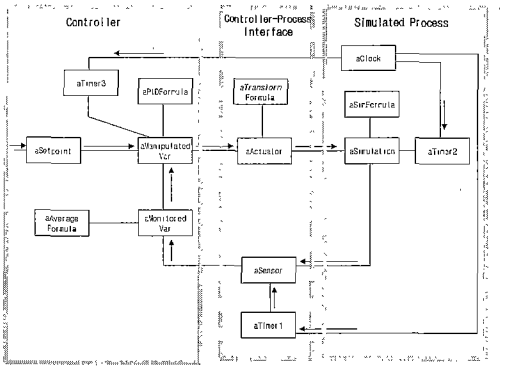

그림 18 방안 온도 제어 샘플 어플리케이션의 구조

그림 18은 그림 17의 방안 온도 제어 루프를 본 논문의 제어 루프 프레임웍의 포인트 클래스들을 사용하여

구성한 것이다. 그림 18에서 화살표는 서로 연결된 포인트 객체들 사이에 값의 변화가 알려지는 방향(trigger direction)을 의미한다.

그림 18에 표시되어 있듯이 난방 제어 루프를 구성하고 있는 포인트 객체들은 그림 17의 제어 루프 모델의 제어기와 공정의 두개의 컴포넌트들을 각각 구현하는 구성 요소들로 구분된다.

그림 18의 설정 변수(aSetpoint), 센서(aSensor) 객체는 난방 제어 루프의 입력 포인트이다. 그 밖의 조작변수(aManipulatedVar), 감시변수(aMonitoredVar), 액츄에이터(aActuator) 및 시뮬레이션(aSimulation) 객체는 모두 계산 포인트이며 자신의 계산 방법을 나타내는 계산방식 객체를 참조하고 있다.

그림18의 난방 제어 루프 시스템의 제어 메커니즘은 다음과 같다. 센서객체는 타이머1 객체에 의해 정해진 시간마다 트리거 되어 감시 변수 객체에게 측정된 기온 값을 전달한다. 조작 변수 객체는 감시 변수 객체로부터 감시변수 값을 입력 받아 자신이 참조하고 있는 PID 계산방식(aPIDFormula) 객체의 PID 제어 알고리즘을 이용하여 조작변수 값을 계산한다. 설정 변수 객체에 설정치가 입력되면 조작 변수 객체는 그 때부터 타이머3 객체를 생성하여 정해진 시간 후에 감시 변수 객체의 값과 설정 변수 객체의 값을 비교하여 제어 상태의 이상 여부를 판단한다. 액츄에이터 객체가 시뮬레이션 객체에게 값의 변화를 알릴 때마다 시뮬레이션 객체는 타이머2 객체를 생성한다. 타이머2 객체가 시뮬레이션 객체에게 피드백 지연 시간이 경과 되었음을 알리면 시뮬레이션 객체는 센서 객체에게 자신의 값을 전달하게 된다. 조작 변수 객체와 시뮬레이션 객체가 생성하는 타이머 객체들은 어플리케이션 실행 시에 동적으로 생성되어 제어 루프 시스템에 연결된 후 지정된 시간이 경과되면 제어 루프 시스템에서 분리되어 소멸하게 된다. 모든 타이머 객체들은 클락(aClock) 객체로부터 틱을 받아 자신의 시간 간격(interval)을 계산한다.

그림 19는 위 제어루프 샘플 어플리케이션을 시험하였을 때 나타나는 포인트 객체들의 상호 작용 과정의 일부를 보여준다.

그림 20은 본 제어루프 샘플 어플리케이션의 제어 효과에 대한 시험 결과를 나타내는 제어 변수의 트랜드(trend) 그래프로써 시간의 흐름에 따른 설정치와 감시 변수 값의 변화의 추이를 보여준다. 그림 20에서 20도로 설정된 목표 기온 값이 15도로 변경되면 감시된 제어 변수 값도 설정치에 접근하고 있어서 제어 루프가 정상적으로 작동하고 있음을 알 수 있다.

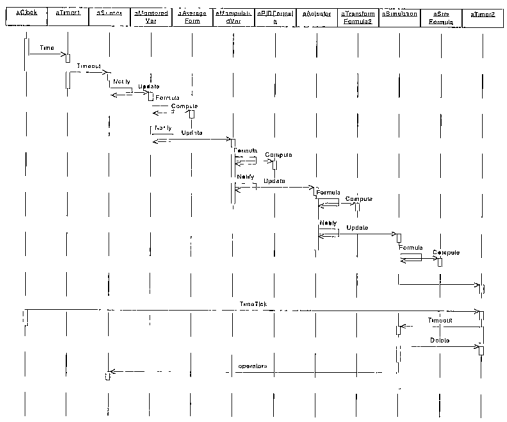

그림 19 방안 온도제어 루프 시스템의 인터액션 다이어그램

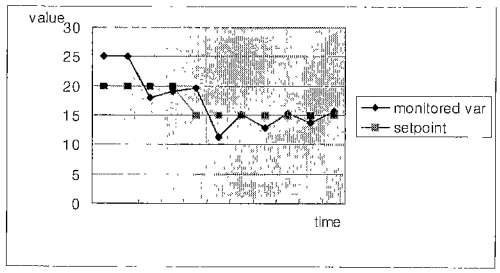

그림 20 방안 온도 제어 루프 시스템의 제어 Trend 그래프

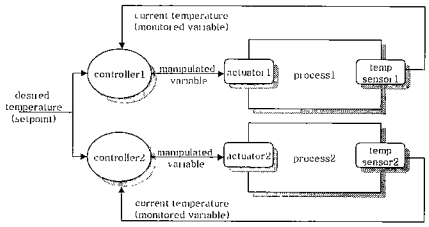

그림 21 복수의 공정 변수들을 갖는 제어 루프

지금까지 설명한 난방 제어 루프 시스템은 액츄에이터, 센서, 제어 변수 및 조작 변수 포인트 객체들이 각각 한 개씩인 단일 제어 루프의 예이다. 그밖에도 여러 개의 공정 변수들을 갖는 단일 제어 루프 또는 여러 개의 단일 제어 루프들로 구성된 복잡한 구조의 제어 루프 어플리케이션들도 본 논문의 제어 루프 프레임웍을 사용하면 쉽게 구현할 수 있다. 그림 21은 본 제어 루

프 프레임웍으로부터 생성될 수 있는 복수의 공정 변수들을 갖는 제어 루프의 예를 보여준다. 그림 22는 그림 21의 제어 루프를 본 논문의 제어 루프 프레임웍의 포인트 클래스들을 사용하여 구성한 것이다.

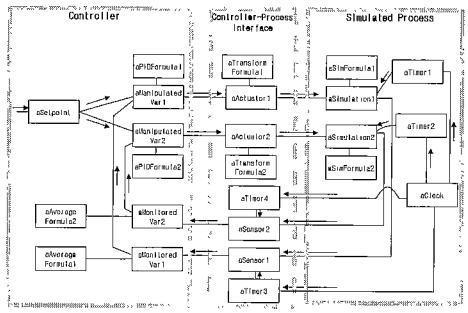

그림 22 복수의 공정 변수들을 갖는 제어 루프 어플리케이션의 객체 구조

5. 관련 연구

본 장에서는 공정 제어 소프트웨어를 객체 지향 프레임웍으로 설계하기 위한 기존의 연구들을 본 논문의 연구와 관련하여 설명한다.

[8]에서는 공정의 감시 제어를 입력 값의 변화에 따른 여러 출력 값들의 연속적인 재계산으로 보았다. [8]에서 value는 공정 변수 값을 추상화한 개념으로써 input value와 computed value로 구분하였다. Value 클래스 객체는 값이 변하게 되면 자신에게 등록된 모든 computed value들에게 값이 변화되었음을 알린다. 값의 변화 사실을 전달 받은 computed value는 이에 따라 자신의 값을 재계산하게 된다.

[9]에서는 소규모의 사무실에서부터 대규모의 분산환경의 공장에 걸치는 여러 화재 경보 시스템에 대한 프레임웍을 설계하였다. [9]에서는 센서나 액츄에이터 같은 여러 실제 입출력 장치들의 인터페이스에서의 논리적인 행위를 표준화하여 포인트로 정의하였다. 이러한 여러 장치들의 추상화 단위인 [9]의 포인트 개념을 공정 변수의 추상화에 적용하여 제어 관계를 담당하는 본 논문의 포인트 클래스로 설계하였다.

[10]에서는 제조 공정에서 생산된 제품의 품질을 측정하여 허용된 품질 수준을 벗어나는 제품에 대해 적절한 조치를 취하는 측정 시스템의 객체 지향 프레임웍을 설계하였다. 측정 시스템 프레임웍 설계에는 주로 Strategy 패턴과 Composite 패턴을 이용하였다. 측정된 값을 제조 공정을 제어하는데 직접적으로 사용하지 않는다는 점에서 본 논문의 제어 루프와 차이가 있다. 본 논문에서는 측정된 값, 즉 감시변수가 제어 변수로 사용되어 공정을 제어하는데 사용된다.

본 논문에서는 제어 루프에서 제어기가 감시 제어하는 공정 변수들과 제어기와 공정 사이의 인터페이스 장치인 센서와 액츄에이터들을 모두 연속적인 재계산을 수행하는 포인트 클래스의 서브 클래스들로 정의하였다.

[8]에서는 공정 변수의 데이타 타입, 계산 알고리즘 및 계산의 제어가 모두 밸류 클래스에 캡슐화 되어 있는데 반하여 본 논문에서는 이들을 별도의 클래스들로 분리하였다. 즉, 공정 변수의 데이타 타입을 공정 변수의 제어와 분리하여 밸류 클래스로 정의함으로써 공정 변수의 연산이 다형적으로 처리되도록 하여 제어 루프 프레임웍의 포인트 값이 여러 형을 쉽게 가질 수 있게 되었다. 또한, 계산 포인트의 재계산에 필요한 계산 알고리즘을 Strategy 패턴에 의거한 계산방식 클래스로 정의하여 계산의 제어와 분리함으로써, 계산 포인트 객체의 알고리즘을 쉽게 교환할 수 있으며 동적으로도 바꾸어 사용할 수 있게 되었다.

위와 같이 본 논문의 포인트 클래스 설계에서는 밸류 클래스와 계산방식 클래스를 별도의 구성 요소로 정의하였다. 제어 관계를 나타내는 포인트 클래스가 밸류 클래스 객체와 계산방식 클래스 객체를 각각 참조함으로써 공정 변수들 사이의 제어 관계와 연산이 분리되었다. 따라서 제어 루프의 유연한 구성이 가능해져 더욱 재사용성이 높은 공정 제어 시스템 프레임웍으로 설계되었다.

6. 결론 및 향후 연구 방향

본 논문에서는 객체 지향의 여러 개념들과 디자인 패턴을 이용하여 재사용성이 높도록 공정 제어 시스템에 대한 프레임웍을 설계하였다. 제어 루프 프레임웍의 핵심 개념은 공정의 상태 값을 제어 메커니즘과 함께 캡슐화 한 포인트 클래스 설계 패턴이다.

본 제어 루프 프레임웍의 포인트 클래스들은 Observer 패턴과 Composite 패턴을 사용하여 포인트 객체들 사이의 결합도가 낮게 설계되어(loose coupling) 있어서 여러 포인트 객체들을 쉽게 재구성할 수 있다. 또한 어플리케이션 실행 중에도 객체의 연결과 분리가 쉽게 이루어진다. 따라서 제어 루프 어플리케이션을 개발하는 프레임웍 사용자는 프레임웍의 포인트 클래스들을 용도에 맞게 개조, 확장하여 프레임웍에 연결함으로써 제어 루프 어플리케이션을 효율적이고 유연하게 완

성할 수 있다.

대부분의 공정 제어 시스템은 병행 및 실시간 처리를 필요로 한다. 본 논문의 제어 루프 프레임웍은 병행 처리와 실시간 스케줄링을 직접 하지는 않으며 어플리케이션으로 확장 시 이를 지원하는 실시간 커널의 사용을 전제로 하고 있다. 즉, 어플리케이션으로 확장 시 클락 클래스, 타이머 클래스를 포함한 프레임웍의 일부 클래스를 사용할 커널에 맞게 개조, 확장하여 프레임웍에 합성함으로써 어플리케이션이 구현된다. 그렇지만 이렇게 병행 스레드, 동기화, 통신 등에 공통적인 메커니즘들이 프레임웍에서 제외됨으로 인하여 본 프레임웍을 특정 어플리케이션으로 확장할 때 제조 노력이 많이 들어가게 되는 문제가 있다.

따라서 향후 후속 연구 과제로써 포인트 클래스들이 병행 스레드와 실시간 반응을 직접적으로 지원할 수 있도록 실시간 병행 자원 관리 메커니즘에 공통적인 부분을 포함함으로써 특정 실시간 OS환경으로 쉽게 확장 가능한 제어 루프 프레임웍으로 개선, 확장하는 연구가 필요하다.

참 고 문 헌

[1] Mary Shaw, "Beyond Objects : A Software Design Paradigm Based on Process Control," ACM Software Engineering Notes, Vol 20, No 1, Jan, 1995

[2] Gregory F. Rogers, "Framework-Based Software Development in C++," Prentice Hall PTR, 1997[3] S.Faulk, J. Brackett, P.Ward, and J.Kirby Jr, "The Core Method for Real-Time Requirements," IEEE Software, pp 22-23, Sept. 1992.

[4] B.Selic, G.Gullekson, and, P.T. Ward, "Real-Time Object-Oriented Modeling," John Wiley and Sons, 1994.

[5] Erich Gamma and et al, "Design Patterns : Elements of Reusable Object-Oriented Software," Addison-Wesley Publishing Company, 1995.

[6] Stuart Bennett, "Real-time Computer Control : An Introduction," 2nd Edition, Prentice Hall International(UK) Limited, 1994.

[7] Bobby Woolf, "The Abstract Class Pattern"

[8] Per Dagermo, Jonas Knutsson, "Development of an Object-Oriented Framework for Vessel Control Systems," Technical Report, Dover Consortium 1996.

[9] P.Molin and L. Ohlsson, "Points & Deviations - A Pattern Language for Fire Alarm Systems," Pattern Languages of Program Design 3, Addison-Wesley Publishing Company, 1995

[10] Jan Bosch, "Design of Object-Oriented Framework for Measurement Systems"

노 성 환

1997년 고려대학교 전산학과 학사. 1999년 고려대학교 전산학과 석사. 2000년 3월 ~ 현재 고려대학교 전산학과 박사과정. 관심분야는 객체지향 프레임웍, 디자인 패턴, 소프트웨어 테스팅

전 태 웅

1981년 서울대학교 계산통계학과 학사. 1983년 서울대학교 계산통계학과 석사. 1992년 Illionis Institute of Technology 전산과학 박사. 1983년 ~ 1987년 금성통신연구소 주임연구원. 1992년 ~ 1995년 LG산전 연구소 책임연구원. 1995년 ~ 현재 고려대학교 자연과학부(전산학 전공) 부교수. 관심분야는 소프트웨어 테스팅, 소프트웨어 아키텍쳐, 객체지향 프레임워크, 실시간 소프트웨어 공학

이 승 룡

1978년 고려대학교 재료공학과 학사. 1987년 12월 Illinois Institute of Technology 전산학 석사. 1991년 12월 Illinois Institute of Technology 전산학 박사. 1992년 ~ 1993년 Governors State University 조교수. 1993년 ~ 현재 경희대학교 전자정보학부(전자계산공학 전공) 부교수, 관심분야는 실시간 컴퓨팅, 멀티미디어 시스템