디자인 패턴의 점진적 통합을 이용한 패턴지향 소프트웨어 개발 방법

김 운 용[†] · 최 영 근[††]

요 약

디자인 패턴은 소프트웨어 생산성 향상을 위해 사용되는 효율적인 기술로써 인식되고 있다. 이러한 디자인 패턴은 소프트웨어 설계시 자주 발생되는 특정 상황에 대한 문제를 효과적으로 해결할 수 있는 방법을 제공한다. 현재까지 다양한 분야에 필요한 디자인 패턴들이 발견되고 이들의 활용성을 증명하는 연구가 진행되고 있다. 그러나 소프트웨어 개발에 이들 디자인 패턴을 효과적으로 적용시키기 위한 체계적인 접근방법에 대한 연구가 부족하다. 본 논문에서는 점진적 디자인 패턴 통합을 통한 패턴지향 소프트웨어 개발 방법을 제시한다. 이를 위해 먼저 디자인 패턴을 활용하는 개발 프로세스를 정의하고 이 프로세스에서 요구되는 점진적 디자인 패턴 통합기법 및 디자인 패턴 기반의 소프트웨어 설계 관점을 보인다. 또한 소프트웨어 시스템에 존재하는 디자인 패턴의 효율적인 추적 및 관리방법을 제시한다. 이러한 과정은 피드백 프레임워크 시스템 설계를 통해 구체화된다. 소프트웨어 개발시 디자인 패턴을 이용한 체계적인 접근과 활용은 초기 개발 단계부터 디자인 경험과 기법들을 효과적으로 활용할 수 있게 함으로써 시스템 개발에 효율성을 증대시킨다. 그 결과보다 안정되고 재사용 가능한 시스템을 이끌어내고 개발 시간과 비용을 단축하는 효과를 제공할 것이다.

Pattern-Oriented Software Development Process using Incremental Composition for Design Patterns

Woon-Yong Kim[†] · Young-Keun Choi[††]

ABSTRACT

Design patterns are known widely by the techniques to improve software quality. The design patterns are efficient solutions for problems occurring frequently in software development. Recently there are wide researches for design patterns to find them and to verify usability for them. But there are very few researches to define systematic development approaches about constructing application using design patterns. In this paper, we propose an approach for the pattern-oriented software development process using incremental composition for design patterns. For this proposal, first we define a development process using design patterns, propose technique for incremental composition for design patterns and view ports for software in the process. Also we deal with the problem of efficient traceability and maintenance to design patterns in the software system. And we use a feedback framework system as an illustrative example to show how the process can be used to develop the pattern-oriented software. In the development of software, the systematic development approach and usage increase efficiency to develop the system by using design experience and technique early in the development lifecycle. Therefore the system will assure the high stability and reusability and offer the low cost and time for development.

키워드 : 디자인 패턴(Design Patterns), 패턴지향 소프트웨어 개발(Pattern-Oriented Software Development Process), 패턴 통합(Pattern Composition), 피드백 시스템(Feedback System)

1. 서 론

소프트웨어 개발에서 디자인 패턴의 활용은 생산성을 향상시키고 재사용과 신뢰성을 가지는 시스템을 구축할 수 있게 한다. 현재까지 다양한 분야에 필요한 디자인 패턴들이 발견되어지고 그 활용을 증명하는 연구들이 진행되고 있다[9, 15]. 그러나 이러한 디자인 패턴을 소프트웨어 시스템 개발과정에서 체계적으로 접근하기위한 연구가 부족하다. 본 논문에서는 점진적인 디자인 패턴 통합과정을 통한 소프트웨어 시스템 개발 기법을 제시한다.

소프트웨어 시스템 개발에서 디자인 패턴의 활용은 몇 가지 고려해야할 문제를 가진다. 첫 번째 소프트웨어 시스템은 다수의 디자인 패턴들로 구성될 수 있고 그들 간의 협력관계를 가진다. 그러므로 사용된 디자인 패턴들은 시스템 관점에서 효율적으로 통합될 수 있는 방법이 요구된다. 일

† 준 회 원 : 광운대학교 정보통신연구원
†† 정 회 원 : 광운대학교 컴퓨터과학과 교수
논문접수 : 2003년 5월 20일, 심사완료 : 2003년 8월 18일

반적으로 패턴의 통합방법은 두 가지 형태를 가진다. 하나는 낮은 통합(Shallow Composition)으로 디자인 패턴별로 존재하는 역할들을 독립적인 클래스로 구성하고 이 클래스를 이용하여 소프트웨어 시스템을 구성하는 방법이다. 이것은 소프트웨어 클래스를 쉽게 생성할 수 있고 디자인 패턴 단위의 분석이 용이하지만 작은 크기의 많은 클래스를 이끌어낸다. 그 결과 시스템 관점의 분석과 관리가 어려워진다. 다른 방법은 높은 통합(Deep Composition)이다. 이 방법은 시스템 클래스에 다수의 디자인 패턴역할들을 포함하는 형태이다. 이 경우 클래스의 응집력은 증가되고 낮은 통합 방법에 비해 적은 클래스를 만들어냄으로써 시스템 관리가 용이해진다. 그러나 이러한 방법은 여러 패턴의 역할들이 하나의 클래스에 포함되는 관계를 가짐으로 적용된 패턴에 대한 분석이 어려워진다. 그 결과 적용된 디자인패턴에 대한 효율적인 분석방법이 요구된다. 본 논문에서는 이 두 가지 통합 방법이 가지는 디자인 패턴의 효율적인 분석능력과 높은 응집력을 가진 시스템을 이끌어내기 위한 효율적인 통합 방법을 제시한다. 이 개발 방법은 톱다운 설계 방식을 이용해 시스템의 구조를 형성하고 설계 과정에서 디자인 패턴을 활용한다. 제시된 소프트웨어 개발과정은 자동화 시스템에서 널리 이용되는 피드백 프레임워크 설계를 통해 기술된다.

본 논문의 구성은 다음과 같다. 먼저 2장의 관련 연구에서 디자인 패턴과 일반적 패턴 활용 기법을 다루고 3장에서 점진적 디자인 패턴 통합을 통한 패턴지향 소프트웨어 개발 방법을 제시한다. 이곳에서 개발 프로세스를 정의하고 이 프로세스에 요구되는 디자인 패턴 통합 및 최적화 문제 그리고 적용된 디자인 패턴에 대한 추적문제를 다룬다. 4장에서는 피드백 프레임워크 시스템 예를 통해 소프트웨어 시스템 개발 과정을 기술하고 5장에서 제시된 방법에 대한 비교 분석을 한다. 그리고 6장에서 결론을 내린다.

2. 관련 연구

이장에서는 디자인 패턴의 특징을 소개하고 소프트웨어 설계에서 디자인 패턴의 일반적 활용방법을 기술한다.

2.1 디자인 패턴

디자인 패턴은 특정상황에서 자주 발생되는 문제에 대한 효율적인 해결방법을 제시하고 있다[4, 5]. 그 결과 특정상황에서 디자인 패턴의 적용은 소프트웨어 품질의 향상과 보다 안정되고 견고한 소프트웨어를 구축할 수 있다. 이러한 이점 때문에 다양한 분야에서 디자인 패턴에 대한 연구가 이루어지고 있다. 일반적인 연구 분야는 디자인 패턴의 식별과 정의이다. 지금까지도 다양한 패턴들이 패턴학회를 통해 발표되고 있다[4, 5, 12]. 또 다른 분야는 디자인 패턴의 응용 분야로 디자인 패턴을 이용한 컴포넌트들간의 통합문제[16, 18] 그리고 소프트웨어 개발에서 디자인 패턴을 효율적인 활용과 생성문제에 대한 연구[2, 3]와 적용된 디자인 패턴의 검증문제[13]등이 존재한다.

이 디자인 패턴들 중 객체지향 설계를 위해 가장 대중적인 설계패턴은 GoF[4] 패턴으로 알려져 있다[4]. 이곳에는 23가지의 디자인 패턴들을 정의하고 있으며 소프트웨어 시스템에서 발생되는 문제에 대한 해결방법을 제시하기위해 구성요소(역할)간의 관계를 통해 설명한다.

2.2 소프트웨어 시스템 개발에서 디자인 패턴 활용 방법

일반적으로 소프트웨어 시스템 개발에서 디자인 패턴 활용방법은 소프트웨어 시스템의 재구성 관점에서 이루어지고 있다[18]. 이 방법은 시스템의 일부 구조를 좀더 효율적으로 구성하기 위해 특정 문제 단위로 디자인 패턴을 적용한다. 그러나 이러한 접근은 소프트웨어 개발 과정에서 디자인 패턴을 활용하기 위한 체계적인 접근방법을 제시하지는 않는다. 또 다른 형태는 패턴 언어를 이용하는 방법으로 이 방법은 특정 도메인에 필요한 디자인 패턴들을 정의하고 이들을 활용하는 프로세스를 정의한다. 이 접근방법은 특정 도메인의 문제 해결방법에 대한 프로세스를 정의하지만 디자인 패턴에 대한 통합 및 추적방법을 제시하지 못한다. 그 결과 소프트웨어 개발과정에서 적절한 디자인 패턴 활용이 이루어지지 않고 있다. 본 논문에서는 개발 초기 단계부터 보다 체계적이고 효율적인 디자인 패턴을 활용하기 위한 개발 프로세스를 정의하고 디자인 패턴에 대한 추적 및 최적화 기법을 제시함으로써 보다 안정되고 신뢰성있는 시스템을 구축하기위한 방법을 제시한다.

3. 점진적 디자인 패턴 통합을 이용한 디자인 패턴지향 소프트웨어 개발 프로세스

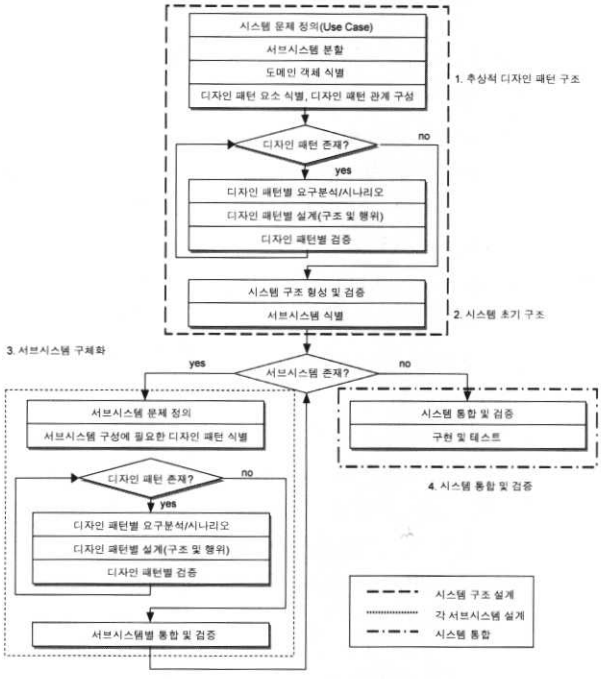
이 장에서는 디자인 패턴을 적용시키기 위한 소프트웨어 개발 프로세스를 정의하고 이 과정에서 요구되는 다양한 관점 및 디자인 패턴의 통합과 추적문제를 다룬다.

3.1 점진적 디자인 패턴지향 소프트웨어 개발 프로세스

소프트웨어 시스템의 거대화는 보다 체계적인 접근 방법과 기술이 요구된다. 이러한 적용기술로써 디자인 패턴과 프레임워크 기반의 설계는 좋은 대안으로 지적되고 있다[1, 4, 8]. 디자인 패턴을 활용한 소프트웨어 설계는 개발초기부터 디자인 패턴요소를 식별하는 과정을 통해 이루어진다. 이러한 접근 방법은 톱다운 설계 방식을 통해 점진적으로 진행되고 개발 초기 단계에서 보다 구조적이고 추상적인 기법을 제공한다. (그림 1)에서 제시된 개발 프로세스는 크게 4단계로 구성된다.

첫 단계는 디자인 패턴 구조 형성 단계이다. 이 단계는 요구사항 분석을 통한 추상적인 관점의 소프트웨어 설계 단계이다. 이 단계를 통해 디자인 패턴 관계 모델이 형성된다. 이 모델은 시스템 구조 관점에서 적용되어지는 디자인 패턴들과 서브 시스템들간의 관계를 보여준다. 두 번째 단

계는 시스템 초기 구조 형성 단계이다. 이 단계는 첫 번째 단계에서 정의된 디자인 패턴요소들을 구체화 시키는 단계이다. 시스템 구조를 형성하기위해 요구되는 디자인 패턴의 구체화는 개발 시스템의 전체 구조를 형성한다. 세 번째 단계는 서브 시스템 구체화 단계이다. 이 단계에서 각각의 서브 시스템들은 이전단계를 통해 구축된 시스템 초기 구조를 기반으로 각각 구체화되어진다. 또한 실제적인 행위 모델들은 이 서브 시스템 단위로 완성되어진다. 이렇게 구성된 서브 시스템의 통합을 통해 시스템 통합구조를 이끌어낸다. 마지막단계로 시스템의 제약사항 및 일관성검증을 거친 후 구현과 테스팅 단계를 거친다.



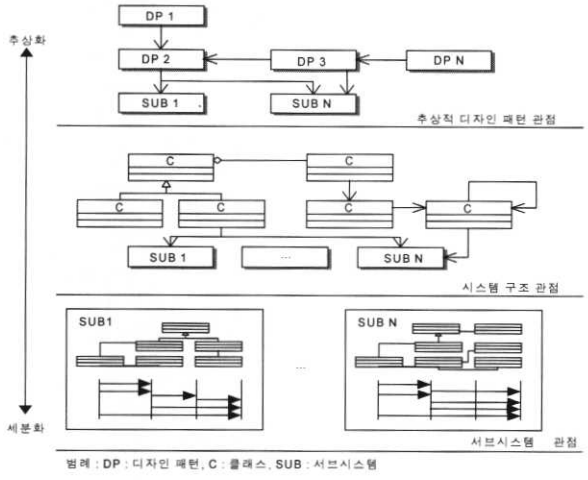
(그림 1) 패턴 지향 소프트웨어 개발 프로세스

제시된 디자인 패턴 지향 소프트웨어 개발 과정에서 고려할 첫 번째 문제는 각 접근단계에 대한 소프트웨어 설계 관점의 제시이다. 이 관점에 대해서는 3.2절에서 설명된다.

3.2 디자인 패턴지향 소프트웨어 설계 관점

디자인 패턴지향 소프트웨어 설계 관점은 크게 3 가지 형태로 구성된다. (그림 2)는 이들 관점을 보여준다.

첫 번째는 디자인 패턴 관계 모델을 통한 추상적인 디자인 패턴 관점이다. 이것은 시스템의 구조를 형성하기 위해 요구되는 디자인 패턴과 서브 시스템들간의 관계를 보여준다. 이 관점은 초기 설계 단계에서 시스템의 구조를 위한 디자인 패턴요소와 서브 시스템 관계를 보여준다. 이러한 추상적인 다자인 패턴 관점은 시스템 구조로 확장된다. 시스템 구조 관점은 요구되는 디자인 패턴을 구체화함으로써 개발 시스템에 적합한 초기 구조를 보여준다. 마지막으로 서브 시스템 관점은 각 서브 시스템들의 구체적인 행위와 구조관점을 제공한다. 각각의 서브 시스템은 시스템 구조관점에서 제시된 구조를 기반으로 서브 시스템의 행위를 수행하기위해 요구되는 기능단위로 확장된다. 이러한 설계 관점들은 톱다운 설계 관점을 제공함으로써 시스템의 전체구조로부터 세부구조를 이끌어낸다.



(그림 2) 소프트웨어 시스템 설계 관점

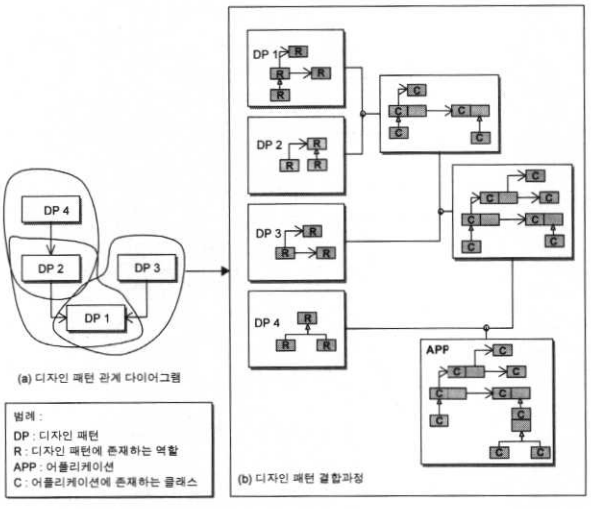
이러한 추상화 관점에서 세분화 과정은 디자인 패턴들의 통합문제와 추적문제를 이끌어낸다. 3.3절에서는 이러한 통합 방법을 다루고 3.4절에서는 추적문제를 다룬다.

3.3 점진적 디자인 패턴 통합

디자인 패턴은 특정 상황의 문제를 해결하기 위한 효율적인 방법을 제공하고 이를 설명하기 위해 필요한 역할의 정의와 이들의 활용방법을 기술한다[4]. 이러한 역할은 시스템 클래스에 적용됨으로써 구체화된다. 즉 시스템의 클래스들은 디자인 패턴의 역할들을 담당함으로써 해당 디자인 패턴을 구체화 시킨다. 이러한 역할 할당 과정에서 디자인 패턴들의 통합이 요구된다. 이에 본 논문에서는 깊은 통합 방법의 클래스간의 높은 결합력을 통한 작은 클래스를 생성하면서 낮은 통합에서 가지는 패턴에 대한 효율적인 분석 능력을 제공하는 기법을 제시한다. 본 논문에서는 점진적 디자인 패턴 통합 방법을 제시한다. 이 방법의 (그림 3)와 같은 과정을 가진다.

디자인 패턴 관계 다이어그램은 시스템을 구성하기 위해 서로간의 의존관계를 가진다. 그 결과 통합과정은 이들 의존관계를 고려하여 점진적으로 이루어질 수 있다. 먼저 각각의 디자인 패턴은 단위 문제해결을 위해 독립적으로 설계 되어지고 구성되어진다. 그런 다음 서로간의 의존성을 고려하여 통합되어진다. (그림 3)(a)는 디자인 패턴 관계 다이어그램을 보여준다. 이 그림에서 디자인 패턴들간의 의존관계가 존재한다. 의존관계를 가진 디자인 패턴은 서로 통합될 수 있는 의미를 가진다. 최초의 통합 과정은 의존관계를 가지지 않는 디자인 패턴과 이들 의존관계를 가지는 디

자인 패턴들을 통해 수행된다. 이 그림에서 초기 DP1은 의존관계를 가지지 않는 디자인 패턴이므로 이 패턴과 이 패턴에 의존하는 DP2와 통합을 수행한다. 이렇게 통합된 결과는 다음 디자인 패턴과 다시 통합되어지는 점진적인 과정을 거쳐 통합된다. 이 점진적 통합방법은 통합된 결과를 다음 패턴의 통합에 적용함으로써 디자인 패턴들의 합병에서 발생될 수 있는 일관성문제를 해결한다.



(그림 3) 디자인 패턴의 점진적 통합 과정

디자인 패턴결합과정의 일반식은 다음과 같이 표현된다.

● DP : 디자인 패턴

$$Composition(DP_n) = (\sum_{i=0}^{n-1} DP_i) \vee DP_n$$

이 식에서 $Composition(DP_n)$은 n 번째 디자인 패턴의 결합을 나타내며 이것은 n-1 번째까지에 의해 형성된 클래스 구조에 n 번째의 디자인 패턴의 결합으로 표현된다. 점진적인 통합을 위한 후보 클래스 식별은 통합과정에서 발생되는 오류를 줄이는 효과를 가져 올 수 있다. 후보 객체의 식별 식은 다음과 같다.

● 후보 클래스 식별
① 디자인 패턴 의존관계 그래프 : DPDG
② 디자인 패턴 합병 클래스 구조 : STRUCT
③ 합병 대상 디자인 패턴 : CDP
④ CDP에 의존관계를 가지는 디자인 패턴 : SDP
⑤ 합병 대상 후보 클래스 : CC

점진적인 합병과정에서 i 번째 후보 클래스 식별은 다음과 같이 정의한다.

$$CC_i = CandidateClass(STRUCT_{i-1}, CDP_i)$$

이 정의에서 $STRUCT_{i-1}$은 i 번째 이전 단계까지 구성된 클래스 구조를 나타내고 CDP_i는 i 번째에 합병대상 디자인 패턴이다. 후보 클래스 생성을 통해 i 번째에 합병 가능한 후보클래스 집합을 CC_i로 나타내고 다음 조건을 만족한다.

$$SDP = x \mid x \in DPDG,\ x = RELATION(CDP_i)$$

합병대상 디자인 패턴 CDP_i에 의존관계를 가지는 SDP는 디자인 패턴 의존관계 다이어그램의 요소이고 합병대상 디자인 패턴의 의존된 디자인 패턴의 집합이 된다. 이렇게 만들어진 SDP을 통해 후보 클래스 CC를 선택할 수 있다. 이 조건은 다음과 같다.

$$CC_i = x \mid x \in STRUCT_{i-1},\ x = INCLUDE(SDP)$$

후보 클래스는 이전단계에서 완성된 $STRUCT_{i-1}$를 구성하는 클래스 집합에서 합병할 디자인 패턴에 의존관계를 가지고 디자인패턴의 역할을 포함하는 집합으로 표현된다.

점진적인 통합에서 디자인 패턴을 시스템 클래스에 통합할 때 두 가지 측면을 고려해야한다. 첫 번째는 기능적인 측면이다. 이 관점은 개발자의 의도에 의존한다. 즉 시스템의 어떤 클래스에 어떤 역할들을 할당할 것인가에 대한 고려는 요구분석과 시스템 특성의 고려를 통해 이루어진다. 두 번째 측면은 구조적인 관점이다. 시스템에서 어떤 특정 목적들을 달성하기 위해 동일한 패턴이 여러 번 사용될 수 있다. 이때 동일한 디자인 패턴에 의해 만들어진 클래스는 동일한 구조를 이끌어낸다. 즉 유사한 클래스들이 한 시스템에 존재할 수 있다. 이러한 클래스들은 시스템의 최적화를 위해 통합되어야 한다. 이러한 통합은 클래스의 유사성 측정을 통해 이루어진다. 이 유사성 측정 식은 다음과 같다.

● 클래스 유사성 측정

클래스간의 유사성(SIM)은 다음 조건을 만족해야 한다.

$SIM(C_{ij})$는 임의의 클래스들간의 유사성을 측정하는 척도로 임의의 클래스 C_i와 C_j사이의 유사성 측정함수 $SIM(C_{ij}) = f(C_i, C_j)$로 정의하고 다음과 같은 특징을 가진다.

① $0 \le SIM(C_{ij}) \le 1$
② $SIM(C_{ij}) = SIM(C_{ji})$
③ $SIM(C_{ij}) = 1,\ C_i = C_j$
④ $SIM_{mv}(A, B) = \frac{(mv(A \wedge B))^2}{mv(A) \times mv(B)}$: 클래스 멤버 변수 유사성 식
⑤ $SIM_{mf}(A, B) = \frac{(mf(A \wedge B))^2}{mf(A) \times mf(B)}$: 클래스 멤버 함수 유사성 식
⑥ mv(Class Name) : 해당 클래스에 속하는 멤버변수의 수이다.
⑦ mf(Class Name) : 해당 클래스에 속하는 멤버함수의 수이다.

두 클래스의 함수와 변수의 유사성 비교를 통해 클래스들 간의 유사성을 판단할 수 있다.

클래스 유사성 표현 식

$$SIM(A,B) = P \times SIM_{mv}(A,B) + (1-P) \times SIM_{mf}(A,B),$$

$$P = \frac{mv(A) + mv(B)}{mv(A) + mv(B) + mf(A) + mf(B)}$$

위의 유사성 표현 식을 통해 클래스구조에서 유사한 클래스를 식별함으로써 클래스구조를 최적화 한다.

이러한 기능적 그리고 구조적 관점을 통한 디자인 패턴의 통합은 보다 안정되고 효율적인 디자인 패턴지향 설계를 이끌어낼 수 있을 것이다. 3장에서 제시된 프로세스와 이를 위해 필요한 접근 기법들은 4장의 적용 예를 통해 더 세부적으로 기술된다.

4. 적용 예 : 피드백 프레임워크

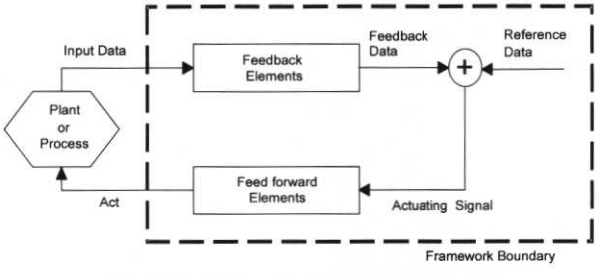
이장에서는 자동화 시스템에 자주 이용되는 피드백 프레임워크 시스템 개발을 제시된 접근방법을 통해 기술한다. 이 접근방법은 개발 초기부터 디자인 패턴을 식별하고 통합하는 과정을 통해 시스템 구조를 형성하고 이 구조를 기반으로 세부 서브 시스템을 완성하는 과정으로 이루어진다. 크게 추상적 디자인 패턴 구조, 시스템 초기 구조, 서브 시스템 구체화, 시스템 통합 및 검증단계를 통해 이루어진다.

4.1 추상적 디자인 패턴 구조

이 단계는 시스템 문제 정의로부터 시스템 구성에 요구되는 서브 시스템 분할 및 도메인 객체 식별 그리고 사용가능한 디자인 패턴 요소 식별 단계등으로 이루어진다.

4.1.1 시스템 문제 정의

피드백 시스템의 일반적인 문제 정의는 (그림 4)와 같은 블록다이어그램으로 표현 된다. 이 피드백 시스템은 입력정보를 통해 행위가 결정되는 형태의 시스템이다. 이 다이어그램과 Use Case와 같은 요구 사항을 통해 요구분석이 이루어진다.



(그림 4) 피드백 시스템 블록 다이어그램

4.1.2 서브 시스템 분할

피드백 시스템의 분석을 통해 시스템 구성에 요구되는 서브 시스템을 식별하는 단계이다. 이 단계에서 식별된 서브 시스템 구조는 크게 3가지 서브 시스템으로 분할 가능하다.

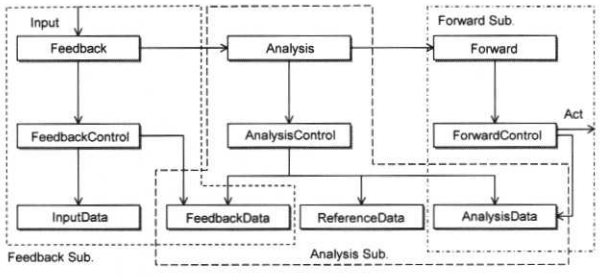
① Feedback System
입력정보을 얻어 분석가능한 정보로 변환하는 시스템

② Analysis System
입력된 정보와 참조정보의 비교를 통해 행위를 결정하는 시스템

③ Forward System
결정된 행위정보를 통해 적절한 응답을 제공하는 시스템

4.1.3 도메인 객체 식별

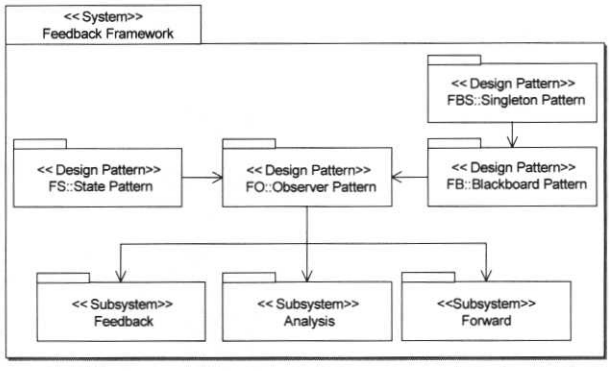
이 단계에서는 시스템을 구성하는 서브 시스템과 요구 분석내용을 통해 시스템 구조를 형성하기 위해 요구되는 도메인 객체를 식별하는 단계이다. 이것은 일반적인 객체 식별 과정을 통해 얻어낼 수 있다. (그림 5)는 이들 도메인 객체와 그들간의 관계를 보여준다.



(그림 5) 도메인 객체와 그들간의 관계

4.1.4 디자인 패턴 요소식별 및 디자인 패턴 관계 구성

다음 단계는 위의 단계를 통해 얻은 서브 시스템과 객체 그리고 그들 간의 요구분석을 통해 시스템에 요구되는 디자인 패턴 요소를 식별하는 단계이다. 분석된 정보를 통해 필요한 디자인 패턴 요소를 식별하고 이들 디자인 패턴요소를 통해 시스템의 추상적인 구조를 만드는 단계이다. 이 과정을 통해 디자인 패턴 요소들과 서브 시스템들로 구성된 추상적은 관계가 (그림 6)과 같이 표현한다.



(그림 6) 식별된 디자인 패턴과 관계

소프트웨어 시스템 구조에서 효율적인 디자인 패턴의 식별은 그 시스템에 대한 이해와 분석능력을 향상시킨다[11]. 본 논문에서는 시스템에 적용된 디자인 패턴의 효율적인 적용을 위해 디자인 패턴 명명자 표기법을 제시한다. 소프트웨어 시스템에서 디자인 패턴은 크게 2 가지 형태로 나타난다. 첫 번째는 추상적 관점에서의 디자인 패턴으로 (그림 6)과 같이 디자인 패턴에 대한 의존관계도를 통해 보여준다. 이들은 <식별자 :: 자인 패턴이름>형식으로 구성된다. 두 번째는 각 디자인 패턴이 구체화되는 형태에서 나타난다. 이 경우 <식별자 : 디자인 패턴역할>로 구성된다. 그 결과 구체화된 디자인 패턴들의 역할들이 어떻게 확장되어지는지를 식별자를 통해 쉽게 분석가능하게 한다. 예로 (그림 6)에서 FS :: State Pattern은 State Pattern을 나타내고 이 패턴의 고유 식별자는 FS이다. 이러한 식별자는 패턴들의 통합 시 각 패턴의 의미와 분석을 위해 필요하다. 프레임워크를 구조를 설계하기 위한 디자인 패턴요소와 서브 시스템의 관계를 설정한 후 다음단계는 각각의 디자인 패턴을 구체화시키는 단계이다.

4.2 시스템 초기 구조 형성 단계

시스템 초기 구조 형성 단계는 위의 추상적 디자인 패턴 구조를 확장함으로써 시스템의 초기 구조를 형성하는 단계이다. 이러한 초기 구조를 형성하기 위해서 각각의 디자인 패턴 요소는 도메인 객체와의 조합을 통해 구체화된다.

이 단계에서는 먼저 각각의 다자인 패턴별 설계를 거치고 (그림 6)과 같은 관계를 기반으로 각 디자인 패턴은 점진적으로 통합이 이루어진다. 이 과정은 크게 디자인 패턴별 설계 단계와 시스템 구조 형성 단계로 나누어진다. 이 시스템 구조 형성 단계는 점진적 통합을 통해 구성된다.

4.2.1 디자인 패턴 설계 단계

이 단계는 식별된 각각의 디자인 패턴을 도메인 객체와 요구분석 내용을 기반으로 구체화 하는 단계이다. 디자인 패턴 설계 단계에서 각 디자인 패턴의 구체화과정을 표현하기위해 <패턴 정의> <문제 정의> <패턴 구조> <활용 구조>의 형태로 기술한다.

● State Pattern 구체화

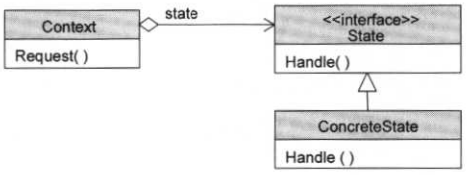
① 패턴 정의

객체가 내부적인 상태의 변화시 객체의 행위를 변경하는 것을 가능하게 한다. 즉 상태에 따라 행위를 다르게 가져가는 방법을 제공한다[4].

② 문제 정의

피드백 프레임워크를 구성하는 각 서브 시스템은 어느 특정 상태에서 동작한다. 이러한 특정 상태에 따라 어느 서브 시스템이 수행해야 하는지를 결정하기위해 State 패턴을 적용한다. 이 피드백 프레임워크는 초기상태, 피드백 상태, 분석상태, 포워드 상태인 4가지의 상태를 가진다.

③ 패턴 구조



(그림 7) State 패턴 구조

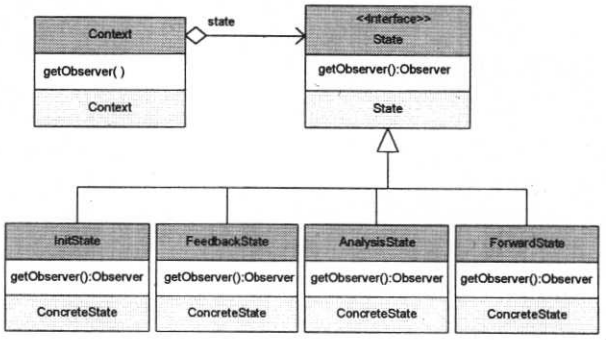
State 패턴은 내부적인 상태에 의해 처리방법을 다르게 하기 위해 3가지 역할을 가진다.

- Context : 이 역할은 현재의 상태정보를 관리하여 현재의 상태에 필요한 행위를 요구한다.
- State : 상태 처리를 다루기 위한 인터페이스
- ConcreteState : State의 구체화된 클래스로 각각의 구체화된 클래스는 그 상태에 적합한 행위를 가진다.

④ 활용 구조

패턴 식별자 : FS(Framework State)

피드백 프레임워크에서 상태는 네 가지로 구성되어 있으며 상태에 의해 처리할 작업은 특정 상태에서 수행해야할 서브 시스템을 선택하는 것이다. 이러한 구조를 만족하는 형태는 (그림 8)와 같다.



(그림 8) 피드백 프레임워크 구조에 필요한 State 패턴 활용 구조

● Observer Pattern 구체화

① 패턴 정의

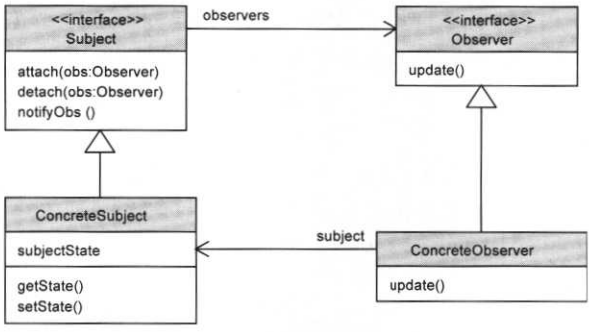
객체들 사이의 일대 다의 의존관계를 정의하고, 객체의 상태 변화시 관련된 객체가 이 변화를 인식하고 그와 관련된 작업을 수행한다. 이 패턴은 일대 다의 의존관계를 통해 객체의 의존성이 증가될 수 있는 상황에서 객체의 의존성을 줄이는 구조를 제공한다[4].

② 문제 정의

프레임워크를 구성하는 각각의 서브 시스템들은 외부 정보의 상태에 연관되어지고 이들 정보의 상태를 인식하고 적절한 작업을 수행해야 한다. 또한 각 서브 시스템은 다른 서브 시스템에 정보를 제공하는 기능을 수행해야 한다. 이

러한 목적을 달성하기 위해 Observer 패턴을 활용한다.

③ 패턴 구조



(그림 9) Observer 패턴 구조

이 Observer 패턴은 네 개의 역할로 구성되어 있다.

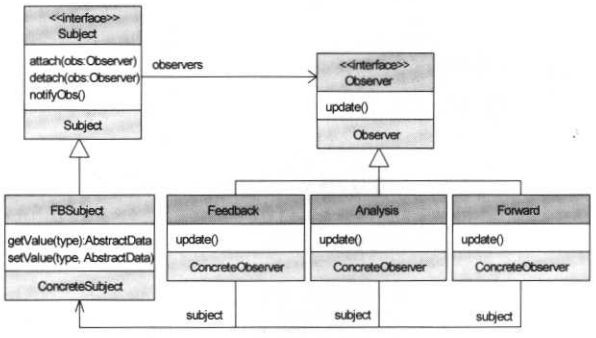
- Subject : 상태 정보를 관리하는 인터페이스를 정의
- Observer : 상태 정보에 따라 행위를 수행하는 인터페이스를 정의
- ConcreteSubject : Subject를 구체화한 클래스로 상태 정보를 관리하고 상태 정보의 변경시 관련 객체에게 update()을 호출한다.
- ConcreteObserver : Observer를 구체화한 클래스로 상태 정보의 변화에 따른 행위를 수행하고, 상태 정보를 얻어오거나 변경할 수 있다.

④ 활용 구조

패턴 식별자 : FO(Framework Observer)

위의 패턴 이름은 프레임워크에 적용된 디자인 패턴의 이름을 나타낸다.

이제 해당 디자인 패턴 구조를 이용하여 피드백 프레임워크 구조를 형성한다. 이러한 과정은 먼저 분석된 도메인 객체에 해당 패턴의 역할을 부여하고, 존재하지 않는 역할은 새롭게 클래스를 생성함으로써 구성된다. 이 피드백 프레임워크에서 상태 정보를 감시하고 정보를 변경하는 객체로 이 패턴에 적용될 객체는 Feedback, Analysis, Forward이다. 이들 객체를 이 패턴에 적용시키면 (그림 10)와 같다.



(그림 10) 피드백 프레임워크에서 Observer 패턴 활용 구조

(그림 10)에서 각 클래스의 맨 하단에는 적용된 패턴의 역할을 기술하고 있다. 이것은 해당 클래스가 적용된 패턴의 어떤 역할을 담당하는지를 확인할 수 있게 한다. 이 역할의 이름에 패턴의 식별자는 생략된다. 이는 단지 하나의 패턴을 묘사하고 있기 때문이다. 디자인 패턴 관계도에서 나타난 나머지 패턴들의 구성역시 비슷한 과정으로 이루어진다.

4.2.2 시스템 구조 형성 단계

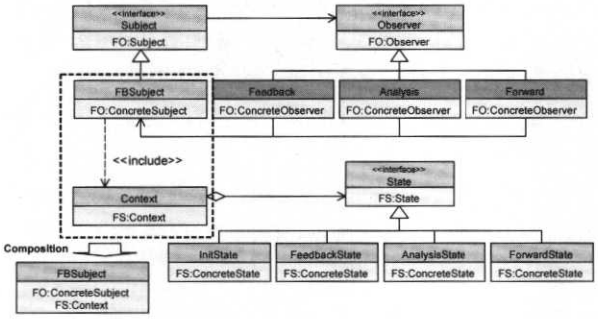
각각의 디자인 패턴 요소의 구체화는 전체 시스템 구조를 이끌어내기 위해 합병되어야한다. 3.2절에서 제시된 진적 합병방법을 이용해 각각의 디자인 패턴은 시스템 초기 구조를 형성한다. 구체화된 디자인 패턴 합병과정은 추상적 디자인 패턴 구조 단계에서 만들어진 디자인 패턴 관계의 의존성을 고려하여 패턴 합병이 요구되는 후보객체를 식별하고 이들을 점진적으로 합병하는 과정을 통해 이루어진다. 점진적 합병을 위한 후보객체 식별 알고리즘은 (그림 11)과 같다.

```
STRUCT : Classes structure, CDP : Candidate Design Pattern,
CC : Candidate Classes for Output
DPDG : Design Pattern Dependency Diagram
SDP : selected design patterns with dependency to CDP
DP : Design Pattern in SDP, kind_of_step : the number of step
ROLE : A role in a design pattern,
SC : A selected class in STRUCT
Initialization :
  DPDG ← get_design_pattern_dependency_diagram ( )
     // get design patterns and dependency of it
  kind_of_step ← kind_of_step + 1 :   // increment step
  STRUCT ← get_structure (kind_of_step - 1) ;
     // get structure for composition
  CDP ← get_candidate() ;   // get candidate design pattern for
                               composition
SDP ← dependency_DP (CDP, DPDG) ;
     // get design patterns to depend on CDP from DPDG
Let h_min and h_max designate the lowest and highest SDP
        magnitude values respectively.
For h ← h_min to h_max {
 DP ← SDP.get_design_pattern ( h ) ;
     // get a candidate design pattern for composition from
        selected design patterns
  Let s_min and s_max designate the lowest and highest role
        magnitude values in DP
  For s ← s_min to s_max {
    ROLE ← DP.getRole ( s ) :   // get a role in a design pattern
    SC ← retrieve_class_with_role ( ROLE, STRUCT ) ;
      // look up a class with the role in the classes structure
      CC.addElement (SC) ;
      // Insert selected classes into the candidate class list
  }
 }
 return CC ;
}
```

(그림 11) 후보 클래스 식별을 위한 의사코드

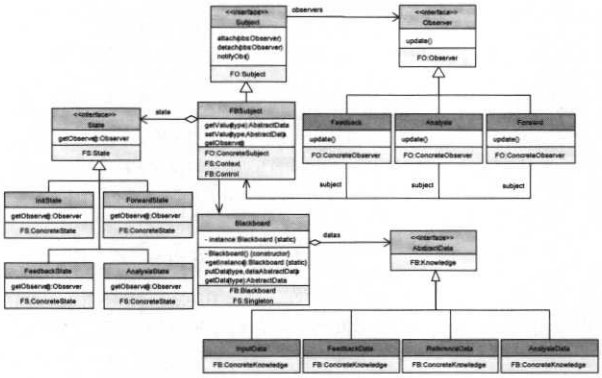
(그림 12)는 Observer 패턴과 State 패턴의 합병을 보여준다. 이 두 패턴은 시스템을 구성하기 위해 서로 의존관계

를 가지고 있으며 이들 두 패턴의 구체화된 구조는 합병을 위한 후보객체 식별을 통해 서로 통합되어진다. State 패턴의 합병은 디자인 패턴 의존 관계 그래프를 통해 의존성을 가지는 Observer 패턴을 식별하고 이 패턴의 역할을 포함하는 클래스의 요소와 합병을 이룬다. 기능적으로 State 패턴의 Context 역할은 Observer 패턴의 ConcreteSubject 역할에 포함되어 상태 관리를 이루는 구조로 FBSubject 클래스로 합병될 수 있다. 이렇게 합병된 구조는 다음 패턴과 점진적인 과정을 거쳐 합병된다.



(그림 12) State 패턴의 합병

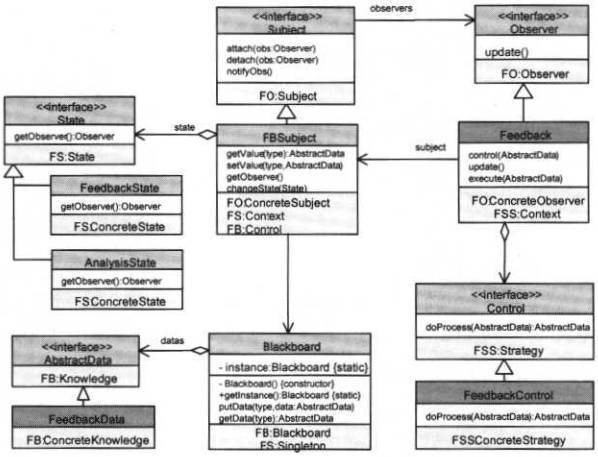
이들 합병된 클래스는 디자인패턴 명명자를 포함하는 클래스 구조를 가진다. 이 구조를 통해 시스템 클래스는 디자인 패턴들의 관계와 역할들을 쉽게 파악할 수 있게 한다. 각각의 패턴들을 합병한 시스템 초기 구조는 (그림 13)에서 보여준다. 이 구조를 통해 완성된 시스템 구조는 다음 전개될 각각의 서브 시스템의 설계 기준을 제공함으로써 서브 시스템들의 개발과정의 일관성을 제공한다.



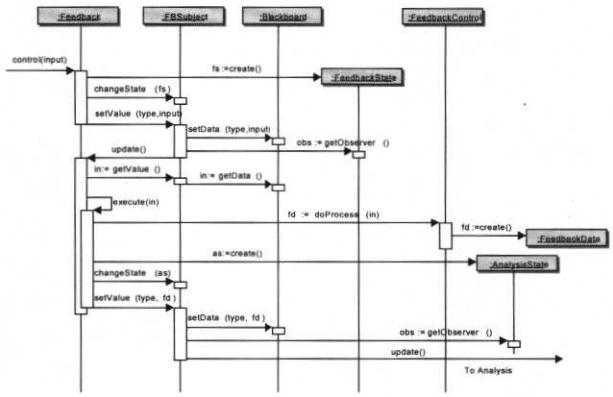
(그림 13) 디자인 패턴을 결합한 피드백 프레임워크 초기 구조

4.3 서브 시스템 구체화 단계

이 단계는 이전 단계에서 완성된 초기 구조 모델을 기반으로 서브 시스템을 확장해 나감으로써 각 서브 시스템의 세부적인 구체화 단계가 이루어진다. 이는 각 서브 시스템별로 문제 정의와 서브 시스템에 요구되는 디자인 패턴 식별 그리고 위에서 보였던 디자인 패턴별 구체화 단계를 통해 이루어진다. 또한 서브 시스템 설계 단계는 행위 모델의 관점에서 이루어진다. 이 단계에서 적용된 디자인 패턴 기능 이외에 추가적으로 요구되는 속성과 기능들이 추가된다. 서브 시스템의 구성은 상위 단계에서 구성된 초기모델을 기준으로 서브 시스템의 행위 모델을 포함시킴으로써 구체화 된다. (그림 14)와 (그림 15)은 피드백 서브 시스템의 구조와 행위 모델을 보여준다. 이 그림에서 보듯이 FSS : Strategy라는 Strategy 패턴이 서브 시스템을 구성하기 위해 추가적으로 발견되어 포함되어지고 있으며, 행위 모델을 통해 서브 시스템 구조를 구성한다.



(그림 14) 피드백 서브 시스템 구조 다이어그램

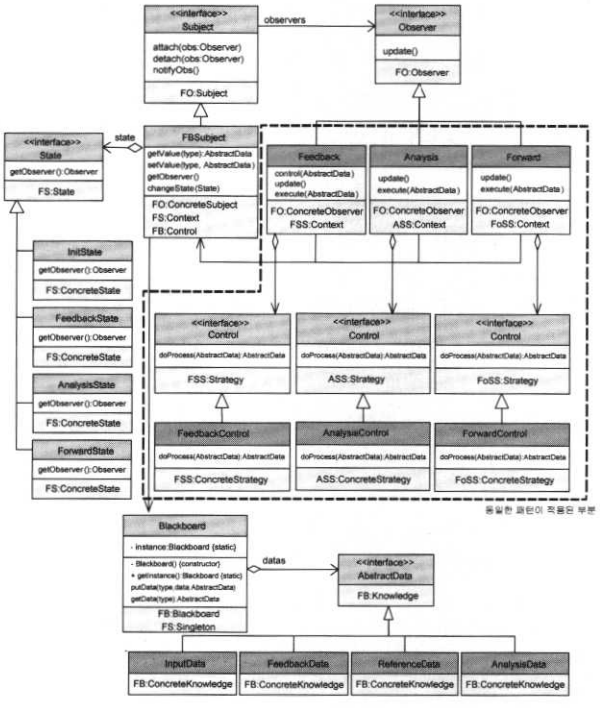


(그림 15) 피드백 서브 시스템 행위 다이어그램

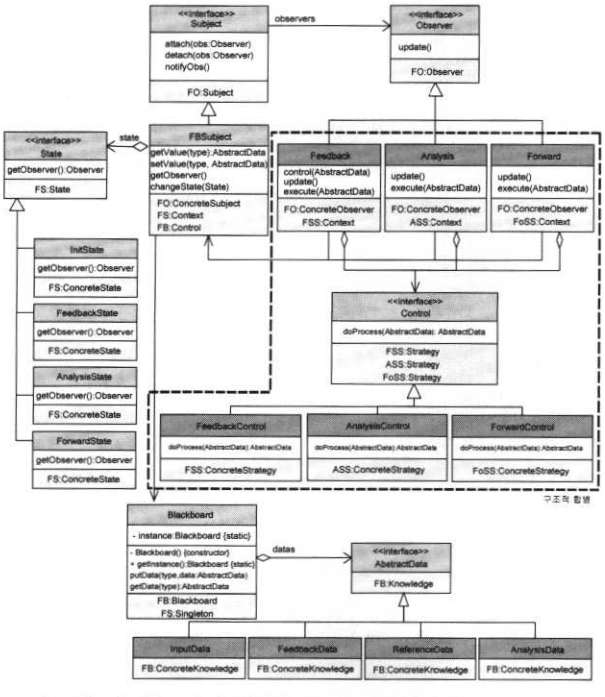
4.4 시스템 통합 및 검증 단계

이 단계는 각각의 구성된 서브 시스템들을 통합함으로써 전체 시스템을 완성하는 단계로 시스템 최적화 과정을 포함한다. 시스템에 동일한 패턴들의 적용은 유사한 클래스를 발생시킬 수 있다. 그러므로 이들 클래스는 구조적인 입장에서 통합되어야한다. 이 과정은 3.3절에서 제시한 클래스 유사성 측정을 통해 유사한 클래스를 식별하고 이들을 하나로 통합하는 과정을 거쳐 완성된다. (그림 16)은 이러한 유사한 클래스가 발생되는 상황을 보여준다. 각각의 서브

시스템은 자신의 알고리즘을 수행하기 위해 Strategy 패턴을 적용하고 있다. 그 결과 Strategy 패턴 구조를 형성하기 위해 유사한 구조를 가진 Control 클래스를 만들어낸다. 이들은 시스템 최적화를 위해 합병되어진다.



(그림 16) 동일한 패턴 적용 후 유사클래스가 존재하는 클래스 구조



(그림 17) 구조적인 합병을 통해 최적화된 시스템 구조

(그림 17)은 유사성 측정을 통해 식별된 클래스가 통합된 형태의 구조를 보여준다. 또한 최종적으로 완성된 피드백 프레임워크 시스템 설계 구조를 보여준다.

전체적으로 점진적인 디자인 패턴 통합을 통한 시스템 설계 과정은 초기 시스템 설계 과정부터 디자인 패턴 요소를 식별하고 이들을 구체화는 과정을 통해 톱다운 방식의 설계 방식을 따른다. 각각의 구조의 분할과 정복방식을 통한 톱다운 설계 방식은 보다 명확한 추적성과 효율성을 제공할 수 있다. 또한 디자인 패턴을 설계 과정에 적극적으로 이용함으로써 초기 시스템부터 더욱 신뢰성 있는 시스템을 구축할 수 있을 것이다.

5. 제안된 기법에 대한 비교 분석

디자인 패턴을 이용한 일반적인 설계방법은 미리 구성된 설계 구조에서 특정 문제에 대한 해결 방법으로 기존의 구조를 재구성하는 수준으로 이루어진다[9, 15]. 이러한 방식은 초기 구조를 구성하는데 걸리는 비용과 이 구조를 변경하기 위해 걸리는 추가적인 비용이 요구된다. 또한 특정문제에 대한 일부분의 해결책만을 제시하기 때문에 전체 시스템에 디자인 패턴의 적극적인 활용에 비해 활용성이 극히 떨어진다. 또 다른 디자인 패턴 활용 접근 방법은 설계 구조에 디자인 패턴들을 구성하고 관리하는 문제를 다룬다. 디자인 패턴의 설계 구조에 구성하는 방법은 크게 기능적인 방법과 구조적인 방법으로 나누어진다. 기능적인 방법은 다양한 디자인 패턴의 구성관계를 기능적인 관점을 통해 해결하는 방법에 대한 연구[8, 10]이고 구조적 방법은 정적인 패턴 구조를 이용하여 구성하는 연구이다[6, 9]. 이러한 두 가지 디자인 패턴 구성 방법은 설계시 발생한 디자인 패턴의 합병 문제를 다루지만 디자인 패턴 적용에 필요한 체계적인 프로세스와 추적 방법을 제시하지 못하고 있다. 본 논문에서는 디자인 패턴을 소프트웨어 시스템 개발 초기 단계부터 체계적으로 접근하는 방법과 요구되는 기술을 제시함으로써 소프트웨어 시스템을 구조적이고 최적화하여 구성하게 하고 디자인 패턴에 대한 추적성을 제공함으로써 분석능력을 향상시킬 수 있게 한다.

6. 결 론

디자인 패턴의 활용은 소프트웨어 시스템의 생산성을 높이는데 효과적인 방법으로 인식되고 있다. 그러나 이들 디자인 패턴에 대한 체계적인 접근 방법이 부족하다. 이에 본 논문에서는 점진적 디자인 패턴 통합 방법을 이용한 소프트웨어 개발 프로세스를 정의하고 이 과정에서 요구되는 기술들을 제시하였다. 제시된 개발 프로세스는 톱다운 접근 방식을 통해 시스템 구조를 형성하고 그들을 구체화시키는 과정으로 이루어졌다. 이 과정에서 디자인 패턴은 개발 초기 과정부터

적극적으로 활용함으로써 기존의 설계방법에서 제시하지 않고 있는 디자인 경험에 대한 효율적인 접근 방법을 제공한다. 또한 점진적인 패턴 통합을 통해 시스템 개발과정에서 발생할 수 있는 오류 및 재구성 비용을 감소시킬 수 있게 하고 디자인 패턴의 효율적인 식별을 통해 시스템 분석과 유지보수 효과를 증대시킨다. 결국 체계적인 디자인 패턴의 활용은 소프트웨어 개발시간의 단축과 재사용성을 높이고 보다 견고하고 안정된 시스템을 이끌어낼 수 있을 것이다.

참 고 문 헌

[1] A. Silva, Development and Extension of Frameworks, In S. Zamir, editor, Handbook of Object Technology, CRC Press, 1998.

[2] B. Schulz, T. Genssler, B. Mohr and W. Zimmer, On the computer aided introduction of design patterns into object-oriented systems, Technology of Object-Oriented Languages, 1998, TOOLS 27, Proceedings, 1998.

[3] C. Marcos, M. Compos and A. Pirotte, Reifying Design Patterns as Metalevel Constructs, Electronic Journal of Sadio, 2(1), pp.17-19, 1999.

[4] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design Patterns : Elements of Object-Oriented Software, Addison Wesley, 1995.

[5] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad and M. Stal, Pattern-Oriented Software Architecture-A Pattern System, Addison-Wesley, 1996.

[6] G. Larsen, Designing Component-Based Frameworks using Patterns in the UML, Communications of the ACM, 1999.

[7] G. Rogers, Framework-Based Software Development in C++, Prentice Hall, 1997.

[8] J. Bosch. Specifying Frameworks and Design Patterns as Architecture Fragments, Proceedings of Technology of Object-Oriented Language and systems, 1998.

[9] J. Garlow, C. Holmes and T. Mowbary, Applying Design Pattern in UML Rose Architect, 1999.

[10] J. Jezequel, M. Train and C. Mingins, Design Patterns and Contracts, Addison Wesley, 2000.

[11] L. Prechelt, B. Unger-Lamprecht, M. Philippsen and W. Tichy, Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance, Software Engineering, IEEE Transactions on, Vol.28, Issue.6, June, 2002.

[12] M. Grand, Pattern in Java, Wiley Computer Publishing, Vol.1, 1998.

[13] M. Sefika, A. Sane and R. Campbell, Monitoring Compliance of a Software System with its high-Level Design Models, Proceedings of the 18th International Conference of Software Engineering, ICSE '96, Berlin, Germany, March, 1996.

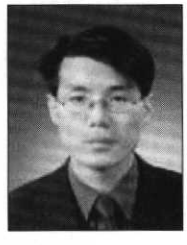
[14] S. Ambler, The Unified Process-Elaboration Phase, Lawrence, KA, R&D Books, 2000.

[15] S. Sirinivasan and J. Vergo, Object-Oriented Reuse : Experience in Developing a Framework for Speech Recognition Applications, Proceedings of 20 International Conference on Software Engineering, ICSE '98, 1998.

[16] S. Stephen, S. Yau and Ning Dong, Integration in component-based software development using design patterns, Computer Software and Applications Conference, 2000, COMPSAC 2000, The 24th Annual International, 2000.

[17] S. Yacoub, H. Ammar, An object-oriented framework for feedback control applications Application-Specific Software Engineering Technology, ASSET-98 Proceedings. 1998.

[18] S. Yacoub, H. Xue and H. Ammar, Automating the development of pattern-oriented designs for application specific software systems, Application-Specific Systems and Software Engineering Technology, 2000, Proceedings, 3rd IEEE Symposium on, pp.163-170, 2000.



김 운 용

e-mail : wykim@cs.kwangwoon.ac.kr

1996년 독학사 전자계산학과(이학사)

1999년 광운대학교 정보통신대학원(이학석사)

2003년 광운대학교 컴퓨터과학과(공학박사)

2003년~현재 광운대학교 정보통신연구원

관심분야 : 객체지향 프로그래밍, 객체 모델링, 디자인패턴, 컴포넌트 개발방법, 분산컴퓨팅기술, 유비쿼터스

최 영 근

e-mail : ygchoi@cs.kwangwoon.ac.kr

1980년 서울대학교 수학교육과(이학사)

1982년 서울대학교 계산통계학과(이학석사)

1989년 서울대학교 계산통계학과(이학박사)

1992~2000년 광운대학교 전산정보원 원장

2001~2002 광운대학교 정보통신연구원장

1992~현재 광운대학교 컴퓨터과학과 교수

2002~현재 광운대학교 교무연구처장

관심분야 : 프로그래밍 언어, 병렬 프로그래밍언어, 객체지향 설계 및 분석, 분산 컴퓨팅기술, Mobile agent