디자인 패턴을 이용한 안전필수 시스템 설계 [1]

이혁*, 이진호*, 황종규**, 최진영*

*고려대학교 컴퓨터학과
{hlee, jhlee, choi}@formal.korea.ac.kr

**한국철도기술연구원
jghwang@krri.re.kr

Design of safety-critical system using Design Pattern

Hyuk Lee*, Jeanho Lee*, Jong-Gyu Hwang**, Jin-Young Choi *
*Dept. of Computer Science, Korea University
**Korea Railroad Research Institute (KRRI)

요 약

시스템의 올바른 동작이 반드시 보장되어야 하는 안전필수 시스템의 안전성과 신뢰성을 확보하기 위한 방법으로 정형기법이 있다. 정형기법은 수학과 논리를 기초로 하는 방법으로, 초심자에게는 다소 난해하고 어려운 부분이 있다. 이와 같은 어려움을 조금이나마 덜 수 있는 방법으로 본 논문에서는 정형기법을 적용한 디자인 패턴을 제안한다.

1. 서론

안전필수 시스템은 높은 안전성과 신뢰성이 요구되는데, 전세계적으로 고 등급 안전필수 시스템을 명세할 때는 수학이나 논리에 기반한 정형명세 언어를 사용하여 명확하게 명세하고 필요한 속성들을 검증하는 것이 권고되고 있다[1]. 하지만 정형기법의 사용에 있어서 가장 부담이 되는 부분인 수학과 논리 기반이라는 점 때문에 활용하기가 어렵다[2]. 모델링 할 때 자주 반복되는 문제에 대해서 패턴화 시키고, 이렇게 패턴화 된 가이드를 사용하여 정형기법 기반 명세 작성을 쉽게 할 수 있다.

본 논문에서는 자연어로 기술된 철도제어시스템의 간격제어모듈 요구 명세를 제안한 디자인 패턴에 적용하여 정형명세언어인 상태차트[3,4]를 용이하게 명세할 수 있도록 가이드 하는 것을 최종 목표로 하고 있다.

본 논문은 구성은 다음과 같다. 다음 장에서는 정형 명세언어인 상태차트에 대해 설명 하고, 3 장에서는 일반적인 디자인 패턴의 개념에 대해 설명을 하고, 4 장에서는 상태차트 도출을 위한 디자인 패턴에 대해 설명하고, 5 장에서는 디자인 패턴의 적용사례에 대해서 설명한다. 마지막으로 6 장에서는 결론 및 향후 연구에 대해서 설명하고 마무리 한다.

2. 상태차트

상태차트(Statechart)는 하렐(Harel)에 의해 제안된 시각적인 정형명세 언어로 매우 직관적이며 우수한 가독성을 특징으로 가지며 동시성 및 계층성을 표현하기에 용이하여 시스템의 행위적인 부분을 효과적으로 명세 하기에 좋다. 상태차트는 시스템을 상태들과 전이들로 표현하며, 상태들 간의 전이는 E[C]/A 형태로 표기된다. E 는 전이를 유발시키는 이벤트이며, C 는 상태 변화의 조건, A 는 Action 으로 조건이 만족되고 해당 전이가 일어나면 명시된 명령을 수행한다. 상태차트가 갖는 특징은 다음과 같이 표현될 수 있다.

Statechart = State-diagrams + Depth + Orthogonality + Broadcast-communication

3. 디자인 패턴

디자인 패턴은 특정 분야에서 디자인 문제에 대한 해결책을 상세히 기록하기 위한 형식적인 방법이다. 건축가 Christopher Alexander 가 처음 소개한 개념으로 건축분야 이외에도 여러 분야에 걸쳐 디자인 문제에 대해 사용되고 있다[5]. Christopher Alexander 가 말하는 디자인 패턴은 기존 환경 내에서 반복적으로 일어나는 문제들을 설명한 후, 그 문제들에 대한 해결책의 핵심을 설명하는 것으로, 이 후에도 재사용할 수 있도록 하는 것이 핵심이다. 컴퓨터과학 분야 중 소프트웨어공학에서는 본래 디자인 패턴의 의미를 반영하여 소프트웨어 디자인에서 자주 발생하는 문제점들에 대한 일반적이고 재사용이 가능한 해결책의 개념으로

[1] 감사의 글: 본 논문은 국토해양부가 출연하고 한국건설교통평가원에서 위탁 시행한 철도종합안전기술개발사업의 결과입니다.

사용하고 있다[6].

디자인 패턴을 구성하는 요소들은 크게 4 가지로 패턴이름, 문제, 해법, 결과로 구성된다. 각 부분에 서술되는 내용으로는 서술하고자 하는 디자인 문제와 해법에 대한 패턴의 이름을 정의하고, 해결할 디자인 문제와 해당 디자인 문제가 발생하게 된 배경을 제시한다. 그리고, 디자인을 구성하는 요소들과 그 요소들 사이의 관계를 서술하며, 디자인 패턴을 적용해서 얻는 결과와 장단점을 포함한다[7].

4 가지 구성요소들은 다음의 13 개의 항목으로 나뉘어 서술된다.

1) 패턴 이름과 분류 (Pattern Name and Classification)
2) 의도 (Intent)
3) 다른 이름 (Also Known As)
4) 동기 (Motivation)
5) 활용성 (Applicability)
6) 구조 (Structure)
7) 참여자 (Participants)
8) 협력방법 (Collaboration)
9) 결과 (Consequences)
10) 구현 (Implementation)
11) 예제 코드 (Sample Code)
12) 잘 알려진 사용 예 (Known Uses)
13) 관련 패턴 (Related Patterns)

4. 제안한 정형명세 디자인 패턴

앞서 언급된 소프트웨어의 디자인 패턴에 대한 분석결과, 이 디자인 패턴은 객체 지향 소프트웨어의 디자인에 많은 초점이 맞춰져 있으며 이러한 디자인 패턴을 객체 지향이 아닌 디자인에 적용 가능한 객체 지향의 성격을 지니고 있지 않은 항목들을 추출하였다. 소프트웨어 디자인을 위한 디자인 패턴 13 개의 항목 중, 6 개의 적용 가능한 항목을 선택하였고, 각 항목들에서 상세히 기술될 내용들은 다음과 같다.

1) 패턴의 이름
   특정 패턴의 명칭
2) 의도
   해결하고자 하는 문제와 이슈, 또는 목적
3) 활용성
   적용 가능한 상황에 대한 기술
4) 구조
   추상화된 패턴의 시각적 표현
5) 결과
   패턴을 사용한 결과가 갖는 장점과 단점
6) 예제
   패턴을 가지고 실제 차트를 작성할 때, 작성 단계를 보여주는 사례

객체 지향에 기반을 하고 있는 기존의 항목들 중에서, 참여자 (Participants)와 협력방법 (Collaboration)은 객체 지향에서 다루고 있는 클래스(Classes)들과 객체(Objects)들이 사용하는 패턴에서 누가, 어떻게 상호작용을 하는지에 관해서 서술하는 부분이다. 정형기법이 적용되는 많은 시스템이 객체 지향적이지 않으며, 본 논문에서 명세 언어로 선정된 상태차트는 시스템의 행위를 기반으로 하는 모델을 명세하기에 적합하여 클래스와 객체를 다루는 항목은 제외하였다.

또한, 구현 (Implementation)과 예제 코드 (Sample Code) 항목에서는 패턴을 사용하여 나온 구현물에 대한 설명과 패턴이 프로그래밍 언어에서 어떻게 사용될 수 있는 지에 대한 설명을 포함하는 항목이다. 본 논문에서는 설계에 앞서 진행되어야 하는 요구사항 분석, 명세단계에 초점을 맞추고 있기 때문에 구현과 구현물에 대한 항목인 구현과 예제 코드를 제외하였다.

반복적으로 발생하는 디자인 문제에 대한 해결책을 위와 같이 패턴화 하여 가이드를 제안함으로써 이를 사용하여 정형기법의 사용에 있어 어려운 점을 어느 정도 줄일 수 있을 것으로 생각된다.

5. 디자인 패턴의 적용 사례

5.1 패턴의 적용 대상

철도제어시스템의 핵심모듈 중 열차의 안전한 간격을 유지토록 하여 열차의 안전한 운행을 위한 간격제어모듈(Distance Control Module: DCM)의 일부 기능의 명세에 대해 디자인 패턴을 적용한 사례이다.

철도제어시스템은 전체 시스템을 총괄하는 ATS (Automatic Train Supervision), 열차방호를 담당하는 CRD (Control Route Distance), 열차시뮬레이터인 ATP (Automatic Train Protection)와 신호설비 시뮬레이터로 구성되어 있다. 이들 중에서, 열차방호를 담당하는 CRD 는 열차간격제어를 위한 DCM 과 진로명령처리를 위한 ICM (Interlocking Control Module)으로 구성된다. 열차간격제어를 위하여 DCM 의 다음과 같은 기능들을 포함하고 있다.

- 블록 개방/폐쇄
- 임시속도제한명령처리
- 열차위치확인
- 열차이동방향감시
- 열차간격제어

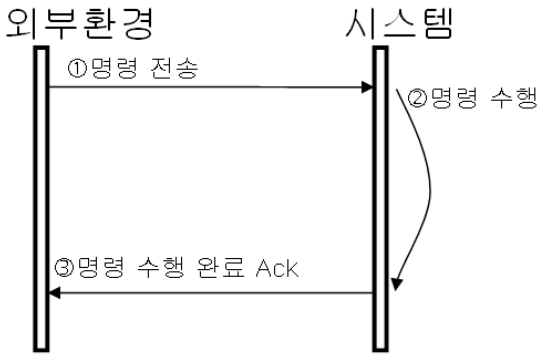
5.2 패턴 적용 예제

디자인 패턴을 적용한 블록 개방/폐쇄 기능은 다음과 같다.

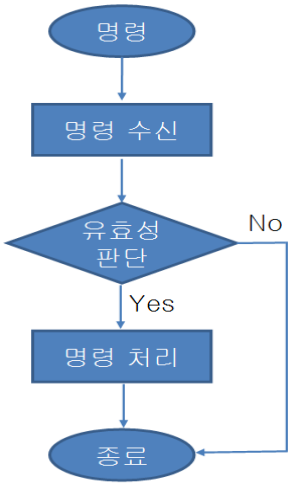
1) 패턴의 이름 (Name)
   외부명령에 대한 수행과 Acknowledge 전송 패턴

2) 의도 (Intent)
시스템이 외부환경으로부터 명령을 받아, 내부적으로 수행을 하고, 수행완료 사실을 외부환경에게 알려줌으로써, 명령에 대한 기능을 전담시킬 수 있다.

3) 활용성 (Applicability)
외부환경으로부터 시스템으로 명령이 전달되고, 시스템으로 하여금 명령을 수행하도록 하고, 작업 완료를 외부환경에게 통보하는 상황에 적합하다.



4) 구조 (Structure)



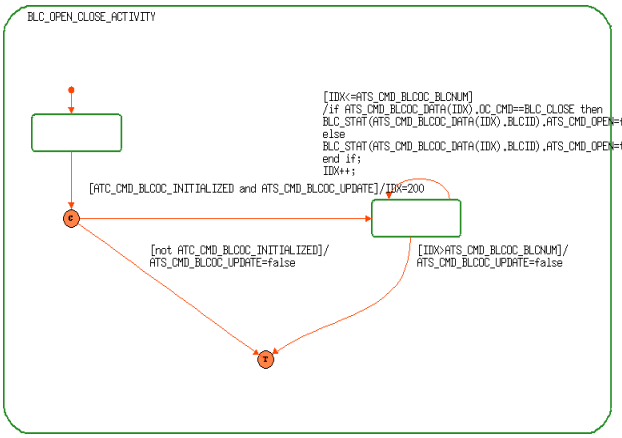
- 수행 명령 처리 이전에, (명령 이행/처리 가능)의 유효성을 확인하고, 유효성 확인 결과에 따라, 명령 수행 여부가 결정되는 상황에 적용될 수 있다.
- 유효할 경우에 명령을 실행한 후에 종료하고, 유효하지 않은 경우에는 실행하지 않고 종료한다.

5) 결과 (Consequence)
- 장점
  명령에 대한 결정과 수행을 각각 외부환경과 시스템으로 분리함으로써, 시스템의 계산 부하가 줄어든다.
- 단점
  시스템은 외부환경으로부터의 명령을 수신하여 명령을 수행하기 때문에, 외부환경으로부터 명령을 수신하지 못하는 경우에, 문제를 일으킬 수 있다.

6) 예제 (Example)

ATS(Automatic Train Supervision: 열차 시스템 총괄)는 궤도 유지보수와 같은 작업을 위하여 선로의 일정 구간에 대하여 작업 구간 보호 구역 및 제한된 진로 보호를 위해 블록 폐쇄명령이 DCM 으로 전달되면 DCM 은 해당블록을 폐쇄하고 인접한 블록도 이에 준하여 이동권한을 설정한 후 해당정보를 ATS 로 전송한다.

블록 개방은 반드시 ATS 에서 전송된 명령에 따라 이루어지며, 이러한 블록개방 명령이 DCM 으로 전송되면, DCM 은 해당 블록 상태를 확인한 후 블록을 개방하고 인접한 블록도 이에 준하여 이동권한을 설정한 후 해당정보를 ATS 로 전송한다.



블록 개방 및 폐쇄명령이 들어오면, 가상블록의 초기화 상태와 업데이트 가능여부를 판단한다.

- 가상블록이 초기화 되어있지 않으면, 블록 업데이트 가능여부를 'false'로 설정하고 종료한다.
- 가상블록이 초기화 되어있고 블록 업데이트가 가능하면, 인덱스 값을 200 으로 (200=첫 블록) 설정한다.
- 가상 블록에 설정된 블록의 개방 및 폐쇄상태에 따라 모든 실제블록에 대해 동일하게 설정한다.

설정이 완료되면, 블록 업데이트 가능여부를 'false'로 설정하고 종료한다.

6. 결론 및 향후 연구

본 논문에서는 소프트웨어 공학에서 반복적으로 발생하는 디자인 문제의 해결책으로 사용되는 디자인 패턴에 대해서 정형기법의 사용에 대해 적용할 수 있는 디자인 패턴을 제안하였다. 또한, 자연어로 기술된 철도제어시스템의 간격제어모듈 요구 명세에 대해 제안한 디자인 패턴을 적용한 예를 적용사례에서 설명하였다. 본 논문에서 제안한 디자인 패턴은 정형명세 언어를 사용하여 시스템을 명세할 때, 명세자가 명세 디자인 패턴을 참고하여 정형기법의 사용에 좀더 쉽게 다가갈 수 있을 것으로 생각한다.

참고문헌

[1] Jame F. Peters, Witold Pedrycz, “Software Engineering – An Engineering Approach}, Wiley, 2000
[2] C. Michael Holloway. Why Engineers Should Consider Formal Methods. 16th Digital Avionics Systems Conference, Oct. 1997.
[3] David Harel, “Statecharts: A Visual formalism for Complex Systems”, Science of Computer Programming. Vol. 8, issue 3, pp. 231-274, 1987
[4] David Harel and Ammon Naamad, “The STATEMATE Semantics of Statecharts”, ACM Trans. Soft. Eng. Method, Oct. 1996.
[5] http://en.wikipedia.org/wiki/Design_pattern (*Accessed on 2010.03.15*)
[6] http://en.wikipedia.org/wiki/Design_pattern_(computer_science) (*Accessed on 2010.03.15*)
[7] Gamma, Erich; Richard Helm, Ralph Johnson, and John Vlissides (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. ISBN 0-201-63361-2.