열차제어시스템 바이탈 소프트웨어를 위한 정형기법 적용 방안 분석
The Analysis of Formal Methods for Applying to Vital S/W in Train Control Systems

조현정*, 황종규**, 윤용기**
Jo, Hyun-Jeong Hwang, Jong-Gyu Yoon, Yong-Ki

ABSTRACT

Recently, many critical control systems are developed using formal methods. When software applied to such systems is developed, the employment of formal methods in the software requirements specification and verification will provide increased assurance for such applications. Earlier error of overlooked requirement specification can be detected using formal specification method. Also the testing and full verification to examine all reachable states using model checking to undertake formal verification are able to be completed. In the comparison of other formal specification methods, we choose the Z formal language for applying to the train control system. Using Z is able to realize higher correctness in the requirement specification, and we propose the Statemate of the best solution in formal verification tools for the system modeling and verification. The Statemate makes it possible to prove thoroughly the system execution from the simple graphical modeling of the complicated train control system. Then we can expect that the model-based formal method combining Z with Statemate will be utilized widely for the railway systems due to various strong points.

1. 서론

최근 우주, 항공, 원자력과 같은 중대한 시스템에서 사용하는 바이탈 소프트웨어를 개발할 때, 안전성과 신뢰성을 증가시키기 위해 정형기법을 적용하고 있다. 정형기법은 정형명세(Formal Specification)와 정형검증(Formal Verification)으로 나눌 수 있으며, 각각을 사용하여 요구사양의 모호함을 없애고 철저한 검증으로 인해 해당 시스템이 완전성을 획득할 수 있게 해준다. 이와 마찬가지로 대규모의 인명 피해를 발생시킬 수 있는 열차제어시스템에서도 고신뢰성이 요구되므로, 열차제어시스템에 서로 다른 여러 가지 정형기법을 적용하여 안전성을 높이려는 연구들이 이미 국외에서 진행 중에 있다. 본 논문에서는 국내 열차제어시스템 바이탈 소프트웨어를 위한 정형기법 적용 방안을 제시하기 위해 기존의 정형명세와 정형검증을 위한 방법들의 장단점을 비교 분석하였다. 비교 분석 결과를 바탕으로 여러 방법 중에 요구사양 명세의 높은 정확성을 기대할 수 있는 정형명세 언어인 Z(Zed)와 Statechart 기반 설계 검증 툴인 Statemate MAGNUM을 절충하여 사용하는 방안을 제시하고자 한다.

* 한국철도기술연구원 열차제어연구팀
E-mail : hjjo@krri.re.kr
TEL : (031)460-5458 FAX : (031)460-5449
** 한국철도기술연구원 열차제어연구팀

2. 정형명세와 정형검증

정형기법이란 소프트웨어 공학의 일종으로 오류가 없는 시스템을 설계하여 시스템의 신뢰성을 높이려는데 목적이 있다[1]. 이러한 정형기법은 수학적 기호를 사용함으로 시스템의 명세를 작성하는데 있어서 자연어가 일으킬 수 있는 애매모호함이나 불확실성을 최소한으로 줄일 수 있고, 설계된 사용자의 요구조건과 동일한지 수학적인 성질을 이용하여 증명할 수 있는데, 크게 정형명세와 정형검증으로 나눌 수 있다. 정형명세는 시스템이 달성해야 하는 요구 사항과 그러한 요구 사항을 만족시키는 설계를 묘사하는데 그 목적이 있고, 정형 검증은 그 설계가 요구사항을 만족하는지를 검사하는 일련의 과정을 의미한다.

2.1 정형명세

정형 명세는 수학적 논리 기호식이나 의미가 명확한 다이어그램 등을 그래픽 표현을 사용해서 명세를 작성하는 기법을 말한다. 명세는 시스템이 무엇을 만족해야 하는가를 정의하고, 시스템이 어떻게 이루어져 있는가를 나타냄으로서 시스템의 동작 환경, 요구사항, 설계 등을 기술하는 것이다.

정형명세 언어로서는 크게 State-diagram과 같은 도식적인 명세 언어와 논리식이나 프로세스 대수(Process Algebra)와 같은 수학적 기호와 문자를 사용하는 언어가 사용된다. 이런 정형명세 기법으로 널리 쓰이는 방법으로 Statecharts, SCR(Software Cost Reduced), Esterel, Z 등을 들 수 있다. 다음 장에서 각각의 방법에 대한 분석과 비교를 하도록 한다.

2.2 정형검증

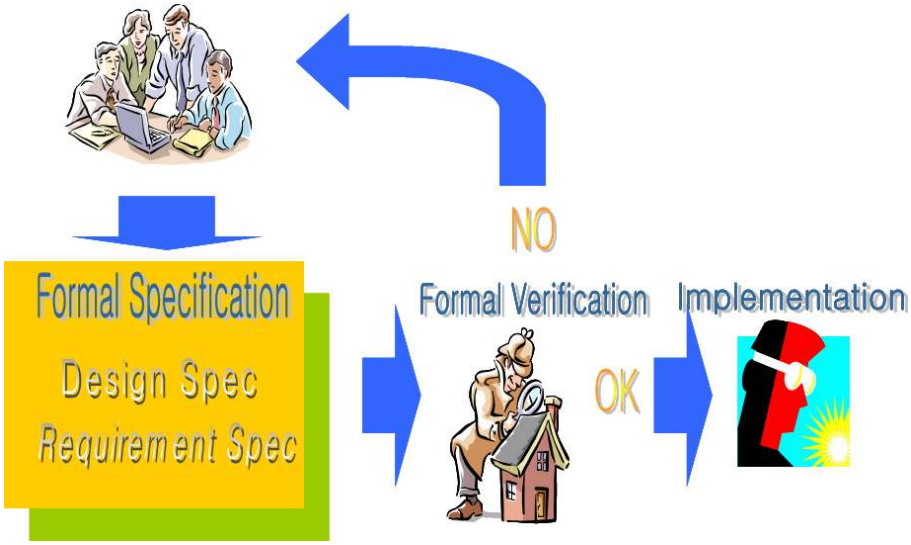

그림 1. 정형기법 절차

정형검증은 그림 1과 같이 정형명세 기법을 사용하여 작성된 시스템 설계 및 요구사항의 만족여부를 수학적, 논리적 증명방법을 사용하여 검사하는 것을 말한다. 정형검증 기법에는 크게 두 가지가 있는데, 하나는 모델체킹 방법이며 다른 하나는 정리증명법(Theorem Proving)이다. 모델체킹은 유한 상태의 병렬 시스템을 자동으로 검증해 주고 항상 원하는 결과가 나옴을 수학적으로 증명해 주는 기법으로 SMV(Symbolic Model Verifier), VIS(Verification Interacting with Synthesis), SPIN(Simple Promela INterpreter) 등의 방법이 여기에 속한다. 또한, 정리증명법은 논리수학식을 사용해서 수학적으로 증명하는 기법으로 PVS(Prototype Verification System) 방법을 들 수 있다. 이러한 다양한 정형기법 방법들의 분석을 통해 바이탈 열차제어시스템 소프트웨어에 적절한 정형기법과 적용 방안에 대해 알아보고자 한다.

3. 정형기법 방법들의 비교 및 분석

각각의 장단점을 지닌 다양한 정형기법 중에서 열차제어시스템의 바이탈 소프트웨어에 가장 적합한 방법을 찾기 위해 여러 정형기법들을 비교, 분석하려고 한다. 먼저 정형명세를 SRS(Software Requirement Specification)와 SDD(Software Design Specification)의 명세 작성에 이용하고 각각을 정형검증 기법으로 확인하는 절차에 있어서 정형기법들을 다음과 같은 기준으로 분석한다.

- 명세 작성 단계 : 명세 언어 및 명세 방식, 명세 언어 이해의 난이도
- 검증 단계 : 명세의 완전성 및 정확성, 검증의 난이도와 완전성

3.1 Statechart

Statechart는 상태기반의 시각적 명세언어로 도식적으로 표현되기 때문에 이해하기가 편하고, Reaction system의 행위적인 면을 표현하는데 장점을 가진다[2]. Statechart 최초의 semantics는 1986년 전부터 개발되어 Reactive 시스템의 설계 및 시뮬레이션에 응용되었으며, 현재에는 내장형 시스템의 설계 및 구현에 널리 이용되고 있다.

본 논문에서는 현재 Statechart 도구로 가장 널리 알려지고 상용화된 툴인 Statemate MAGNUM에서 사용되는 Semantics를 바탕으로 설명하겠다. 우선 Statechart는 시스템의 모든 디자인을 도식적으로 표현한다. 시스템의 입출력은 시그널과 데이터의 입출력으로 하고, 시스템의 행위는 상태도의 전이로 표현하게 된다. 이와 같이 Semantics가 정형적으로 정의되어 있음으로 정확한 이해만 있다면, 애매모호함 없이 설계하고 이해할 수 있다고 본다.

Statemate가 적용되는 전체적인 절차는 그림 2와 같으며, 다음과 같은 activity chart와 statechart 그래픽을 사용하여 시스템 사양을 모델링하고 검증하고 있다. Activity chart는 그림 3과 같이 시스템의 상태를 표현하는 기능을 갖고 있어, 여러 계층으로 연결되어 있는 데이터의 흐름을 볼 수 있다. 또한 flow line을 이용해서 인터페이스 사이의 작동을 확인할 수 있다. Statechart의 state는 작동하고 있는 운영모드를 확인할 수 있게 해주며, connection을 통해 다양한 상태로 전개될 수 있는 여러 조건과 진행 내역을 알 수 있다. 임의의 상태에서 다른 임의의 상태로 움직이기 위한 기준을 확인한 후, 이벤트(trigger)가 발생하면 다음 발생 동작을 정의한다.

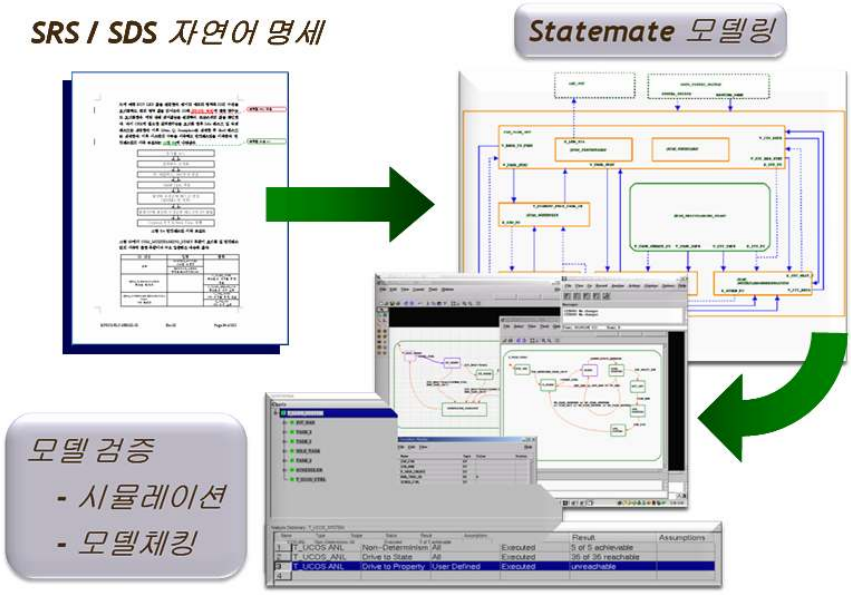

그림 2. Statemate의 적용 절차

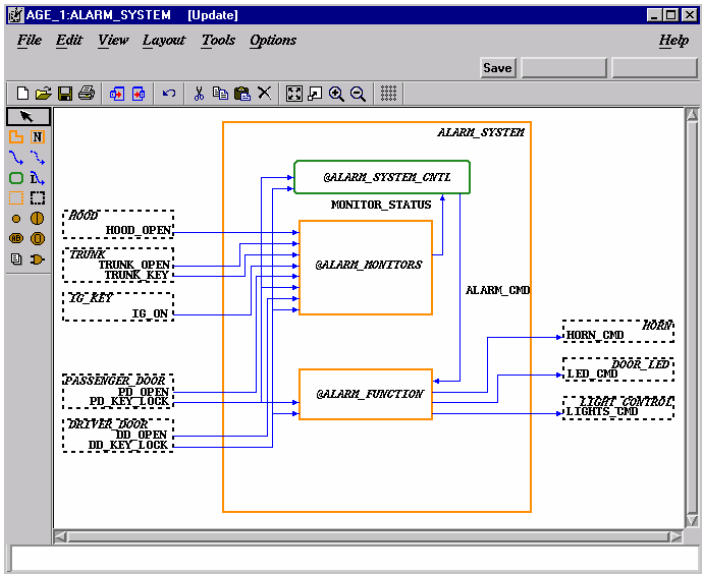

그림 3. Statemate의 activity chart의 예시

이처럼 Statemate는 그래픽 기호를 사용하여 실행 가능한 명세서를 그래픽 모델로 쉽게 만들 수 있으며, 명세서는 시스템의 동작을 시뮬레이션 하기 위해 사용되고 시스템의 모든 명세 단계에서 선택적으로 검증하고, 디버그하고 시스템이 어떻게 동작 하는지를 가시적으로 볼 수 있게 해준다. Statemate의 가장 큰 장점은 비주얼 모델링, 시뮬레이션 코드생성, 도큐먼트 생성, 테스트 계획 등을 하나로 통합함으로써 복잡한 시스템의 명세 작성, 디자인, 검증까지 모두 가능하게 해주는 강력한 도구라는 점이다. 나아가 다른 정형명세 언어에 대해 알아보겠지만, Statemate처럼 정형명세와 검증을 모두 지원하는 도구는 찾아보기 힘들며 이는 그림 4에서 볼 수 있는 ModelCertifier와 ModelChecker의 모델체킹 역할이 크다고 볼 수 있다.

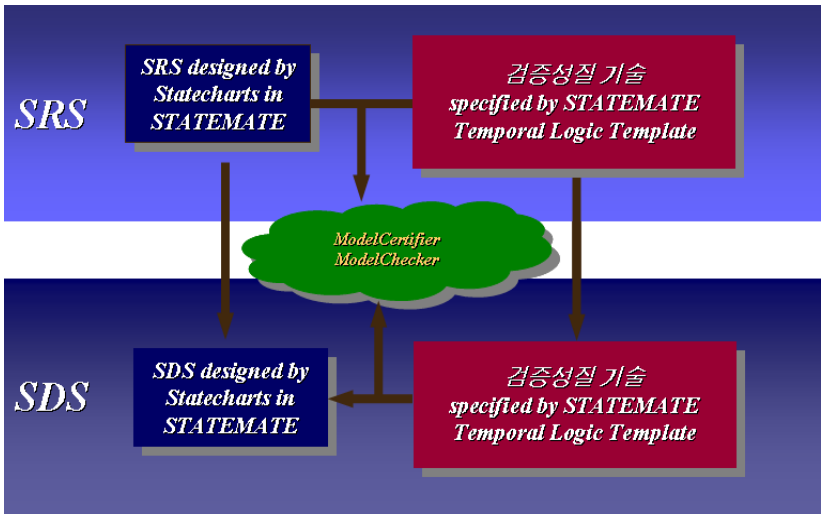

그림 4. Statemate의 검증 과정

3.2 SCR(Software Cost Reduction)

SCR은 안전성이 중요한 시스템의 소프트웨어 요구사항을 기술하기 위해 테이블에 기반한 정형명세 기법이다[3]. SCR에서 하드웨어 인터페이스는 입력과 출력 데이터 아이템과 소프트웨어 기능으로 기술되며, 소프트웨어 기능은 조건 테이블, 이벤트 테이블, 모드 전이 테이블의 세 가지 테이블(수학적 함수를 정의)의 집합으로 이루어져 있다. SCR이 시스템이나 소프트웨어를 표현할 때 유한상태 기계(finite

state machine)를 사용한다. 이를 mode class라고 한다. 이 mode class는 mode들의 집합과 transition들의 집합으로 구성된다. mode들의 집합은 각 시스템과 소프트웨어의 환경의 상태를 나누어 놓은 것이고, transition들의 집합은 source modes, destination modes, events로 이루어진다.

이러한 SCR 요구명세는 어떻게 시스템이나 소프트웨어가 물리적 양의 값을 결정하고 영향을 미치는지 상관하지 않고 정확하게 그 물리적 양만을 기술한다. 그래서 SCR 명세는 두 환경 변수를 정의하는데 monitored variable과 controlled variable이 바로 그것이다. controlled variables는 시스템을 제어하거나 시스템에 영향을 주는 물리적 양을 나타내는 환경 변수이고, environment variables는 단지 시스템을 측정하고 감시하는 환경 변수이다. 작성된 SCR 명세는 변수들의 값을 가지고 시뮬레이션도 할 수 있고, SCR tool을 통해 검증을 할 수도 있다. 그러나 Statemate처럼 모델체킹의 정형적 검증을 이루기 위해서는 SPIN이나 SMV의 변환과정을 거쳐야만 한다.

3.3 Esterel

Esterel은 Reactive 시스템의 동기적 프로그램을 작성하는데 사용되는 동기적 언어이며, 이러한 동기적 프로그램 언어는 완벽한 동시성 모델에 근거하여 동시적 프로세스가 0시간 내에 계산과 정보의 교환을 일으킨다[4]. 즉, Esterel은 동기적 가설(synchronous hypothesis)을 기반으로 하기 때문에 입력 set에 대한 모든 반응은 시간을 소모하지 않고 즉각적인 것으로 간주된다. 이러한 동기적 가설을 충족시키도록 통신과 선점은 결정성을 유지하면서, 시그널의 출력과 입력은 시간적 소모가 없는 것으로 여기게 된다.

Esterel과 관계하여 각종 언어로 바꾸어 주고 언어의 문법적 오류와 의미적 오류를 찾아내어 수정하게 하는 컴파일러인 Esterel toolset이 있으며, 이에 대한 시뮬레이션 도구로 XES가 있다. 또한 이를 정형검증하는 도구인 XEVE가 있다. 또한 바이시뮬레이션(bisimulation)으로 시스템을 검증하게 하는 fc2symbmin이 있으며 또한 이를 모델체킹으로 검증하게 하는 VIS라는 도구 역시 이를 검증하고 시뮬레이션 하는 도구로 알려져 있다.

Esterel도 SCR과 같이 정형검증을 위해서는 또 다른 도구를 활용해야 한다는 점과 C와 유사한 함수형 언어로 시스템을 묘사하기 때문에 Esterel을 잘 아는 사람이 짠 요구 명세만이 정확한 시스템 분석이 가능하다는 단점이 있다. 그러나 ESTEREL 내의 동시성은 프로그램의 편리를 위해 구조적 도구이며 런타임 오버헤드를 일으키지 않게 해준다는 장점을 가져올 수 있으며, 표현력이 풍부하고 프로그래머가 입력 이벤트에 대한 요구된 반응을 정확하고 모호하지 않게 명세할 수 있다는 장점으로 여전히 많은 연구가 진행 중이다.

3.4 Z(Zed)

Z 언어는 일차 논리(First-Order Logic)와 집합론과 같은 수학적인 기반을 가지고 있고, 이로 인해서 명세에 많은 이득을 가지고 있다[5]. 예를 들면, 이러한 수학적인 표현은 간결하고, 애매모호함이 없기 때문에 매우 정확한 명세를 할 수 있다는 것이다. 이미 Z 언어는 산업 환경에서 커다란 소프트웨어 시스템을 명세 하는 실질적인 결과를 내면서 성장하였으며, 앞으로도 활용 분야가 넓어질 것으로 보인다.

Z를 이용하여 명세를 하는 절차는 그림 5와 같으며, 방법은 다음과 같다. 먼저, 명세에 대한 주어진 집합과 전역상수를 그 의미에 대한 비정형 설명과 함께 기술한 후, 추상적 상태를 묘사하는 스키마를 만들고, 시스템의 초기상태를 표현하는 스키마를 표현한다. 에러와 관계없이 추상적 동작을 스키마로 기술하고, 동작 스키마를 명확하게 해주는 부분적 동작의 선조건을 기술한다. 에러조건을 명시하는 스키마를 기술하고 나면, 동작을 종합하여 부분적 동작과 에러조건을 기술한 스키마를 기반으로 하여 기술한다.

Z는 수학적 논리와 집합을 이용하므로 시스템 정의와 같은 요구사항의 명세 작성에는 매우 높은 완전성을 이룰 수 있지만, 집합론과 논리에 대한 정확한 지식과 경험이 요구된다는 점과 모델링과 시뮬레이션과 같은 작업에 있어서는 데이터 흐름을 시간적인 개념을 포함하여 나타내기 힘들다는 단점을 지니고 있다. 따라서 단독으로 사용해서는 정형검증까지 개발시간을 단축하여 이루어 내기가 어려우므로, 다른 정형기법과 절충하여 사용하는 방법이 효과적이다.

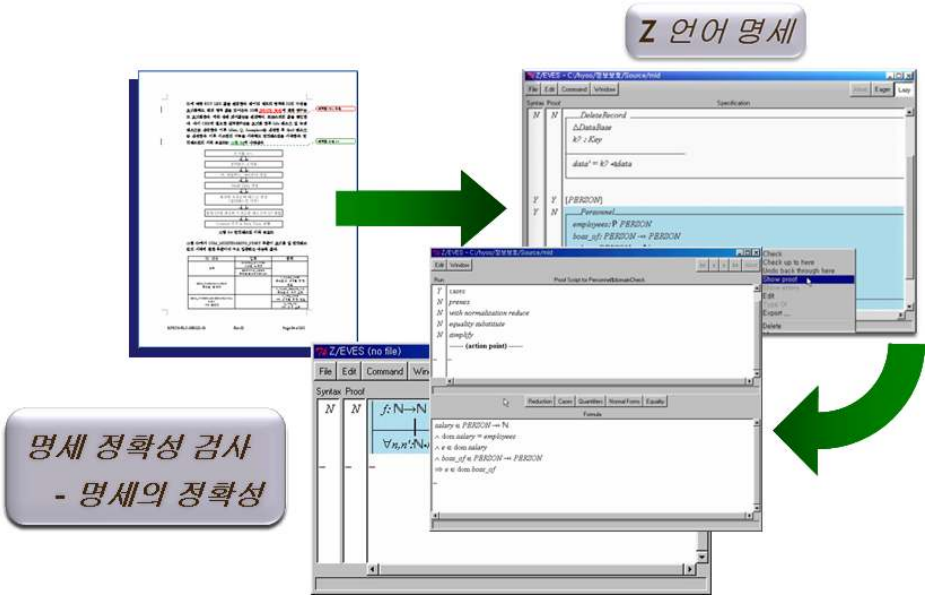

그림 5. Z를 이용한 명세화 절차

3.5 SMV(Symbolic Model Verifier)

SMV 시스템은 유한 상태 시스템이 CTL(Computational Tree Logic)로 표현된 요구 명세를 만족하는지를 자동으로 검증하는 정형 검증 도구이다[6]. SMV의 입력 언어는 유한 상태 시스템을 명세하기 위해 만들어 졌으며, 이 입력언어를 이용해서 시스템을 동기적인 밀리 머신(Mealy machine)이나 비동기적인 네트워크로 손쉽게 명세할 수 있다. SMV 언어가 유한 상태 시스템을 위해 만들어진 것이기 때문에 언어가 제공하는 자료 구조도 유한한 형태(Boolean, scalar, fixed array,...)만을 제공한다. 시스템은 SMV입력 언어로 나타내어진 모델이 CTL로 표현된 요구 명세를 만족하는지의 여부를 효율적으로 검사하기 위해서, 심볼릭 모델체킹 알고리즘을 사용한다.

이러한 SMV는 SRS를 만족시키는지 검증하기 위해 CTL 모델체킹을 수행한다는 것을 알 수 있으며, 행위적인 측면에서 SRS가 요구하는 특성들을 만족시키는지를 검증하는 것은 가능하므로 행위적인 묘사는 완전하게 명세될 수 있다고 볼 수 있다. 그러나 요구명세를 CTL로 명세해야만 하므로 표현력이 제한될 수 있다는 단점이 존재한다.

3.6 VIS(Verification Interacting with Synthesis)

VIS는 검증, 시뮬레이션, 유한 상태 하드웨어 시스템의 합성을 종합적으로 수행하는 도구이다[7]. 입력 언어를 Verilog를 이용하고 fair CTL 모델체킹, Language emptiness 검사, Combinational and sequential equivalence checking, Cycle-based 시뮬레이션과 계층적 합성(Hierarchical synthesis)을 지원한다. VIS가 사용하는 입력언어는 Verilog로서 일반적인 Verilog와 다른 점은 비결정성과 기호적 변수를 사용한다는 것이다.

VIS를 이용하여 검증하기 위해서는 먼저 간단하게 검증할 수 있는 형태로 바꾸어야 한다. 상태와 상태 상이의 전이는 유한 상태 기계로 구성되고 완전한 시스템은 각 구성 요소와 연관된 유한 상태 기계를 구성함으로써 얻어지는 유한 상태 기계이다. 그러므로 검증할 때, 첫번째 단계는 시스템을 유한 상태 기계로 나타내는 것이다. 유한 상태 기계로 나타낸 시스템을 VIS의 입력 언어인 Verilog 로 설계한 후, VIS를 이용하여 현재 자동 정형 검증에서 가장 많이 쓰이는 두 가지 방법인 Language containment와 SMV와 같이 CTL을 이용한 모델체킹 기법으로 검증한다. 따라서 VIS 방법도 SMV와 같이 CTL을 사용하므로 표현력이 풍부하지 않아 자연어를 그대로 묘사하기에는 수월하지 않다는 단점을 가지고 있다.

3.7 SPIN(Simple Promela INterpreter)

SPIN은 LTL(Linear Temporal Logic) model checker로서, 기존의 model checking 방법에서 문제점으로 제시되어 온 memory explosion을 개선하기 위해 고안되었다[8]. 이는 전체 상태들을 모두 구현하는 것이 아니라 각 상태에서 필요한 것만을 만들어내기 때문에 메모리의 효율적인 사용을 가능하도록 하였다. SPIN은 비동기 프로세스 시스템의 설계와 검증을 지원한다. SPIN은 또한 분산 소프트웨어와 통신 프로토콜의 검증에 아주 유용하게 사용되고 있다. SPIN 검증 모델은 프로세스 상호작용의 정확성에 초점을 둔다.

SPIN은 PROMELA(PROcess MEta LAnguage)라는 검증 언어를 사용하여 설계를 하고 PROMELA는 LTL의 문법으로 명세화 하여 정확도를 증명한다. SPIN은 LTL의 형태로 받아들여진 어떤 특성을 부치 오토마타(Buchi automata)의 형태로 변환시킨 후 이를 다시 PROMELA의 형태로 변환시켜 명세 된 시스템과 병렬로 수행을 시킨다. 이때 검증기를 생성하여 그 결과를 분석하면 명세 된 시스템이 요구되는 특성을 만족하는지를 분석할 수가 있다. 이처럼 SPIN은 다소 복잡한 과정을 통해 정형검증이 이루어진다는 단점과, CTL보다도 제한적으로 단 하나의 시간 흐름에 대해 고려하는 LTL을 사용하므로 표현력이 매우 제한적이라는 단점을 지니게 된다.

3.8 PVS(Prototype Verification System)

PVS는 대표적인 정리증명법의 정형검증 방법이다[9]. 정리증명법은 공리와 시스템의 정확성을 증명하기위한 증명 규칙들의 사용을 가리킨다. 이것은 시간을 많이 소모하고 전문가들을 요구하기 때문에, 모델체킹 방법들을 사용하여 검증하기 어려운 infinite state system에 대한 추론에 일부 사용된다. 일반적으로 모델체킹 방법이 자동적인 검증 능력으로 인해 정리증명법보다 더 수월하기 때문에 더욱 선호되어지는 것이다.

PVS를 사용하면 논리의 증명에 대한 교육과 다양한 증명에 익숙해져야하므로 일반적인 사용자가 특별한 교육을 받지 않는다면 PVS의 검증방법을 이용하기 힘들다. 하지만 정리증명법은 모델체킹과 같은 검증 방법이 매우 복잡한 시스템 분석을 할 때와 같이 달리 시스템 분석을 위한 memory explosion을 일으키지 않는다.

4. 국내 열차제어시스템 바이탈 소프트웨어를 위한 정형기법 적용 방안 제시

위와 같이 다양한 정형기법에 대한 분석을 바탕으로 열차제어시스템 바이탈 소프트웨어 적용을 위한 가장 적합한 정형기법 방법으로 Z를 선택하고자 한다. 다른 명세 언어보다 요구 사양의 타입 정의를 명확하고 간결하게 할 수 있으며, 이러한 높은 정확성으로 인해 시스템 개발 초기단계부터 대상 시스템의 동작 요구사항이 명확히 만족되는지 판단이 가능하기 때문이다. 그러나 앞서 언급하였듯이 Z는 도식적인 모델링과 시뮬레이션을 거치고 정형검증의 일반적인 방법인 모델체킹을 통한 확인 작업에서 한계를 갖고 있기 때문에, 이 부분에서 강점을 지니고 있는 Statemate 툴을 Z와 절충하여 사용하려고 한다.

이와 같이 Z와 Statemate를 절충한 열차제어시스템에 활용 절차 방안은 다음 순서와 같다. 먼저 그림 6과 같이 SRS와 SDS를 작성한 후, Z를 사용하여 완전하게 요구사양의 오류를 점검한다. 그 후, Z에 의해서 검증된 시스템요구사양에 대해서 Statemate 툴을 이용하여 모델링한다. Statemate에 의해 완성된 그래픽 모델의 시뮬레이션을 통해 대상 시스템의 동작을 검증하고 여러 가지 테스트를 시험한다. 이 때, 자동으로 시스템에 내장될 프로그램 코드와 결과 문서 작성이 통합적으로 이루어진다. 이처럼 Z를 통한 개발 초반의 정밀한 명세 오류 검지와 Statemate를 통한 비주얼 모델링, 시뮬레이션, 코드 작성의 통합 자동화로 인해 결과적으로 다른 방법에 비해 30% 이상의 개발 기간 단축 효과를 거둘 수 있다고 본다.

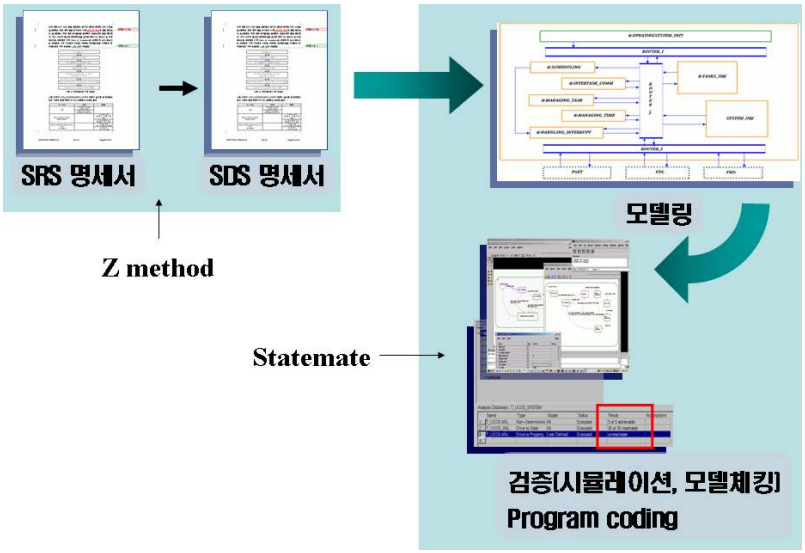

그림 6. 열차제어시스템 바이탈 S/W를 위한 정형기법 활용 방안

5. 결론

중대한 시스템에 사용되는 바이탈 소프트웨어를 개발할 때, 정형기법에 해당되는 정형명세와 정형검증을 사용하면 개발된 제어 시스템들의 안전성이 효율적으로 증가된다. 본 논문에서는 열차제어시스템 바이탈 소프트웨어에 적합한 정형기법을 찾기 위해 여러 가지 다양한 기법들을 서로 비교 분석하였으며, 그 결과를 바탕으로 Z와 Statemate 툴을 절충하여 활용할 것을 제안하였다. 다른 정형명세 언어에 비하여 Z를 사용하여 요구사양 명세의 높은 정확성을 기대할 수 있으며, Statemate를 이용하여 복잡한 대상 시스템을 단순화된 그래픽으로 모델링하여 시스템의 동작을 시뮬레이션하고 철저한 검증까지 통합적으로 이룰 수 있다. 이와 같이 열차제어시스템을 위한 모델 기반 정형기법의 적용에 있어서 다른 방법을 선택하는 것보다 많은 강점을 지니는 Z와 Statemate를 절충하여 사용한다면, 안전성이 중시되는 철도분야에서의 활용 가능성은 매우 크다고 내다볼 수 있다.

참고문헌

[1] Kotonya, G., and Sommerville, I., “Requirements Engineering: Process and Techniques”, Wiley, 1998.

[2] David Harel and Ammon Naamad, “The STATEMATE Semantics of Statecharts”, ACM Trans. Soft. Eng. Method, Oct. 1996.

[3] C. Heitmeyer, J. Kirby, B. Labaw, and R. Bharadwaj, “SCR: A Toolset for Specifying and Analyzing Software Requirements”, Proc. of CAV'98, Vancouver Canada, 1998.

[4] Frédéric Boussinot, and Robert De Simone, “The Esterel Language Another Look at Real Time Programming”, Proc. of the IEEE, vol. 79, pp 1293-1304, 1991.

[5] Jonathan Jacky, “The Way of Z”, Cambridge, 1997

[6] K. L. McMillan, “Symbolic Model Checking”, Kluwer Academic Publishers, 1993.

[7] http://www-cad.eecs.berkeley.edu/Respep/Research/vis/

[8] Moshe Y. Vardi, and Pierre Wolper, “An automata-theoretic approach to automatic program verification”, Proc. First IEEE Symp. on Logic in Computer Science, pp. 322-331, 1986.

[9] S. Owre, J. Rushby, and N. Shankar, “Analyzing tabular and state transition requirements specification in PVS”, SRI international, 1995.