ISSN 2383-630X(Print) / ISSN 2383-6296(Online)
Journal of KIISE, Vol. 46, No. 8, pp. 814-822, 2019. 8
https://doi.org/10.5626/JOK.2019.46.8.814

FMProjector: 표준 인터페이스를 준수하는 운영체제를 위한 정형 검증 프레임워크

(FMProjector: A Formal Verification Framework for an Operating System Complying with a Standard Interface)

이 동 아 †　　유 준 범 ††
(Dong-Ah Lee)　　(Junbeom Yoo)

요 약 정형 검증은 소프트웨어의 정확성을 검증할 수 있는 유용한 기법이지만, 운영체제와 같이 규모가 큰 소프트웨어 전체를 대상으로 한번에 적용하는 것에는 상태 폭발같은 문제로 인해 어려움이 따른다. IEC 61508이나 DO-178과 같은 안전 필수 분야의 표준에서는 정형 검증을 통한 소프트웨어의 정확성 확인을 소프트웨어의 안전 수준에 따라 일부 권장하고 있다. 본 논문에서는 표준 인터페이스를 준수하는 운영체제를 위한 정형 검증 프레임워크, FMProjector를 소개한다. 운영체제를 검증하기 위하여 소프트웨어를 수평적·수직적으로 분석하기 위한 접근 방법을 제안하며, 표준 인터페이스로부터 운영체제의 모든 개발 산출물에 대한 추적성을 기반으로 검증 영역을 식별할 수 있는 체계적인 방법을 제시한다. FMProjector를 항공용 RTOS의 인터페이스 표준인 ARINC-653을 준수하도록 개발한 운영체제, Qplus-AIR를 대상으로 적용한 사례연구를 소개한다.

키워드: 정형 검증, 운영체제, 추적성분석, 표준 인터페이스

Abstract Formal verification techniques facilitate the verification of functional correctness of software. The verification, however, is rarely applicable to large-scale software, such as operating systems, because of the state explosion problem. International standards or certifications, such as IEC-61508 or DO-178, highly recommend formal verification of such software according to the level of safety. The paper introduces a formal verification framework, FMProjector, for operating systems complying with a standard interface. The framework includes horizontal and vertical approaches for systematic analysis of the software based on traceability from the standard interface to the source code. The paper also introduces a case study for the application of FMProjector to Qplus-AIR complying with ARINC-653 which is a standard interface for avionics real-time operating system.

Keywords: formal verification, operating system, traceability analysis, standard interface

· 본 연구는 2016년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터(B0101-16-0663,오류 없는 시스템 통합을 위한 안전우선 분산 모듈형 SW 플랫폼)와 2017년도 정부(과학기술정보통신부)의 재원으로 한국연구재단-차세대정보컴퓨팅기술개발사업(NRF-2017M3C4A7066479)의 지원을 받아 수행하였습니다.

† 학생회원 : 건국대학교 컴퓨터공학과
ldalove@konkuk.ac.kr
†† 정 회 원 : 건국대학교 컴퓨터공학과 교수(Konkuk Univ.)
jbyoo@konkuk.ac.kr
(Corresponding author임)

논문접수 : 2019년 4월 5일
(Received 5 April 2019)
논문수정 : 2019년 5월 5일
(Revised 5 May 2019)
심사완료 : 2019년 5월 20일
(Accepted 20 May 2019)

정보과학회논문지 제46권 제8호(2019. 8)

1. 서 론

항공, 자동차, 원자력 등과 같은 안전필수시스템(safety-critical system)에서 사용하는 소프트웨어는 기능의 정확성을 철저히 검증할 필요가 있다. IEC 61508-3[1]이나 DO-178 시리즈[2]와 같은 표준 및 인증에서는 소프트웨어를 요구되는 안전 무결성 수준(safety integrity level)에 따라 분류하고, 수준에 따라 기능의 정확성을 확인하는 정도를 제시하고 있다. 소프트웨어에서 가장 높은 수준의 안전이 보장되어야 하는 부분에 대하여 정형검증과 같은 기법을 활용할 것은 강하게 권장(highly recommend)한다.

정형검증 기법 중 하나인 모델체킹[3]은 모델의 모든 상태를 탐색하며 검증 속성의 만족여부를 확인하는 기법으로 다양한 분야에 적용되어 왔다. 모델체킹은 정형 모델에 대한 철저한 수준의 검증이 가능한 반면, 모델의 크기에 따라 고려해야 할 상태의 수가 기하급수적으로 늘어나는 상태 폭발 문제(state explosion problem)[4]를 반드시 고려해야 한다. 따라서 운영체제(OS)와 같이 규모가 큰 소프트웨어를 한번에 모델링 하여 철저히 검증하는 것은 대단히 제한적이다.

본 논문에서는 표준 인터페이스를 준수하는 운영체제를 모델체킹 기법을 활용하여 검증을 수행하기 위한 검증 프레임워크, FMProjector(Formal Model Projector)를 소개한다. FMProjector는 운영체제 전체에서 검증 목적과 관련된 부분을 모델로 도출하는 방법을 제안한다. 이를 위하여 운영체제를 수평적 관점으로 바라보고, 대상 전체를 넓고 얕게 모델링한다. 수평적 관점의 검증 결과를 토대로 시스템의 특정 기능에 대한 상세한 모델을 소스코드로부터 생성하여 수직적 관점의 검증을 수행한다. 특히, 수직적 관점의 모델을 생성하기 위하여 표준 인터페이스로부터 요구사항, 설계 및 소스코드에 이르는 추적성을 이용한다.

본 논문에서 제안한 FMProjector를 운영체제 검증에 활용하기 위하여 한국전자통신연구원(ETRI)에서 개발한 Qplus-AIR[5]를 대상으로 수행한 사례연구를 소개한다. Qplus-AIR는 통합된 형태의 항공기 시스템인 IMA (Integrated Modular Avionics) 구조에 적합하도록 개발한 항공용 운영체제로서 ARINC-653[6]을 준수하는 실시간 운영체제(real-time operating system)이다. 항공기의 운영체제는 항공기 시스템을 운영하기 위한 가장 근본적인 소프트웨어로서, 기능의 정확성이 반드시 검증 되어야 한다.

본 논문의 구조는 다음과 같다. 2장에서는 정형검증과 논문의 검증 대상인 Qplus-AIR에 대한 배경지식을 간략히 설명한다. 3장에서는 운영체제와 같이 규모가 큰 시스템을 정형검증하기 위한 기존의 연구들을 소개하고, 4장에서는 본 논문에서 제안하는 검증 프레임워크를 상세히 설명한다. 5장에서는 제안한 검증 프레임워크에 따라 Qplus-AIR를 검증한 결과를 보인다. 마지막으로 6장에서는 본 연구의 결론과 향후 연구에 대하여 논한다.

2. 배경지식

정형검증(Formal Verification)은 정형언어로 명세된 대상을 논리적 혹은 수학적인 방법을 통해 증명하는 기법으로, 대표적인 정형검증 기법으로는 모델체킹(Model Checking)과 Theorem Proving 등이 있다. 이 중 모델체킹은 모델이 속성을 만족하는지 상태 탐색을 통해 증명하는 기법으로, 테스팅이나 시뮬레이션과는 달리 모델의 모든 상태를 빠짐없이 검사하기 때문에 보다 높은 수준의 검증이 가능하다. 이러한 모델체킹은 적용 대상이 한정되고 비용이 많이 소요되지만, 항공, 원자력, 자동차와 같은 안전최우선(Safety-Critical) 시스템과 같이 시스템의 안전성(Safety) 및 정확성(Correctness)이 철저히 요구되는 시스템 또는 소프트웨어의 검증에 제한적으로 적용되고 있다.

무인항공기(UAV: Unmanned Aerial Vehicle)에 사용되는 운영체제(O/S: Operating System)는 항공기 전체 시스템을 관리하고 운영하는 소프트웨어로서, 항공기 운행에 가장 핵심이 되는 소프트웨어이다. Qplus-AIR[3,7,8]는 ETRI에서 개발한 무인항공용 실시간 운영체제로서, 운영체제와 응용 프로그램 간의 인터페이스를 규정한 ARINC-653표준을 준수한다. 특히 핵심 모듈인 Qplus-653은 국제 항공전자 소프트웨어 인증인 DO-178B의 가장 높은 등급인 Level A를 획득한 소프트웨어이다. DO-178B 표준에서 정형검증을 소프트웨어의 일부에 적용하기를 권장하지만 필수 요구사항으로 정의하지는 않는다. Qplus-AIR는 품질을 철저히 관리하며 개발했으나, 보다 높은 수준의 안전성 및 정확성 확보하기 위해서는 정형검증과 같이 높은 수준의 검증을 적용할 필요가 있다.

3. 관련 연구

운영체제를 검증하기 위하여 정형검증을 적용한 연구는 지속적으로 이어져 왔다. NICTA와 UNSW의 공동연구 결과인 seL4 운영체제에 대한 정형검증 사례가 대표적이다[9,10]. 정형검증 기법 중 Theorem Proving을 사용하여 운영체제 전체를 성공적으로 검증한 사례로 소개된다. 이외에도 Theorem Proving을 사용해 운영체제를 검증한 사례를 MASK 프로젝트[11], Verisoft 프로젝트[12], VFiasco 프로젝트[13] 등에서 찾아볼 수 있다. 이러한 연구 중 일부는 검증을 위하여 여러 Theorem Prover를 복합적으로 활용하였다. 차량용 임베디드 운

영체제의 표준인 OSEK/VDX를 준수하는 다양한 운영체제의 검증 사례가 존재한다. 모델체커 SPIN을 이용해 Trampoline OS를 점증적으로 검증하는 방안을 제시하였다[14]. 그밖에 EMERALDS-OSEK[15], ORIENTAIS[16] 등 다양한 운영체제 검증 사례가 존재한다.

정형검증을 사용하여 운영체제를 검증하기 위해 Theorem proving기법이 주로 사용되고 있으며, 모델체킹을 적용한 경우는 검증 대상에 적합한 새로운 방법을 제안해 사용한다. 특히 ARINC 653을 준수하는 운영체제는 앞서 소개했던 운영체제보다 복잡하고 많은 기능을 지원하여 일반적인 방법으로는 모델체킹을 적용하는 것이 어렵다. 제안하는 FMProjector는 시스템을 체계적으로 나눠 모델체킹을 적용하기 위한 접근 방법을 제시하여 이와 같은 문제를 일부 해결하고자 한다.

FMProjector를 활용한 연구의 일환으로 Qplus-AIR의 Schedulability를 정형검증 도구인 Times를 사용하여 검증한 사례도 [17]에서 확인할 수 있다. Times는 Timed automata를 사용해 모델링을 하며, 다수의 task들이 주어진 스케줄링 정책에 부합하게 스케줄되는 지를 확인하기에 적합한 도구이다. [17]의 모델을 생성하기 위하여 본 논문에서 제시한 수평적 검증 결과를 토대로 핵심 모듈을 선별할 수 있었다. 이외에도 CBMC를 사용해 소스코드를 대상으로 모델체킹을 직접 수행한 연구도 진행하여 논문으로 소개될 예정이다.

4. 정형검증 프레임워크

4.1 FMProjector

본 논문에서 제안하는 프레임워크는 표준 인터페이스의 요구사항을 준수하는 운영체제를 대상으로 한다. 표준 인터페이스에 따라 개발된 운영체제의 추적성을 기반으로 크게 두 가지 관점으로 소프트웨어를 바라보고 분석한다. 그림 1은 표준인터페이스를 준수하는 운영체제를 분석하기 위한 두 가지 관점을 나타낸다. 첫 번째 관점인 수평적 관점은 시스템 전체의 기능을 포괄적으로 모델링하지만 각 기능이 구현된 상세한 내용을 모델에 포함시키지 않는다. 대상의 구조를 유지하며 기능은 제한적으로 모델링하여 모델의 복잡도를 낮춘다. 포괄적인 모델링을 통해 시스템 전체에 대한 검증 및 시뮬레이션을 수행하고, 궁극적으로 수직적 관점에서 모델링할 영역을 도출을 위한 참조모델로 사용한다. 두 번째 관점은 검증 속성에 따라 시스템의 특정한 부분을 모델링 대상으로 삼으며, 검증 대상의 추상화 정도를 최소화하여 구현된 상태 그대로를 반영하도록 모델링한다. 이 때 시스템의 특정 부분을 체계적으로 선별하기 위해서는 개발 산출물의 추적성이 확보되어야 한다.

그림 2는 FMProjector의 구성요소를 나타낸 그림이다.

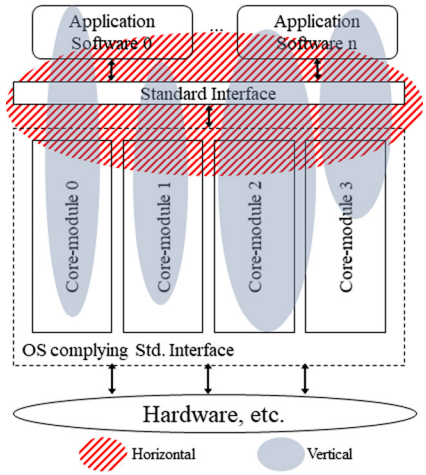

그림 1 표준인터페이스를 준수하는 운영체제를 분석하기 위한 수직·수평적 관점

Fig. 1 Horizontal and vertical views for the analysis of an operating system complying with a standard interface

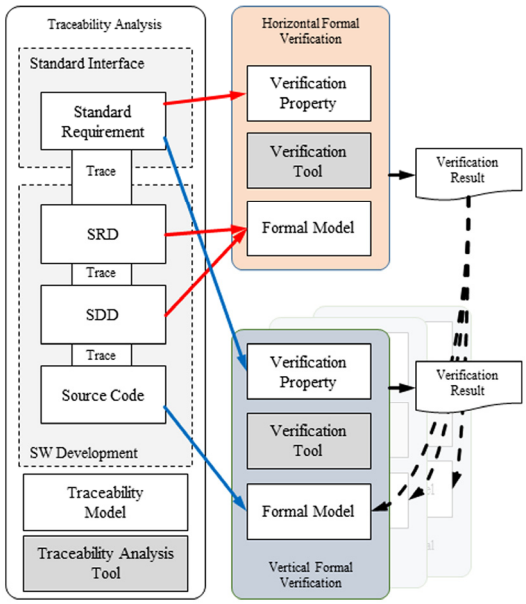

그림 2 FMProjector: 표준 인터페이스를 준수하는 운영체제의 정형검증을 위한 프레임워크

Fig. 2 FMProjector: A formal verification framework for formal verification of an operating system complying with a standard interface

FMProjector에서 제안하는 정형검증을 수행하기 위해서는 소프트웨어 개발 산출물(SW Requirement Document, SW Design Document, Source Code) 간의 추적성(Traceability)을 확보해야 한다. 표준 인터페이스에

서 시작하여 소스코드에 이르는 모든 산출물에 대한 추적성 뿐만 아니라, 정형검증 과정에서 개발하는 모델과 산출물 간의 추적성도 지속적으로 관리해야한다. 만일 개발된 소프트웨어의 산출물 간 추적성이 확보되지 않는다면, 역공학 또는 재공학과 같은 접근 방법을 통해 확보할 수 있다. 추적성을 체계적으로 확보 또는 분석하기 위해서는 추적성 모델과 지원 도구를 사용하는 것이 도움이 된다.

수평·수직적 관점 모두 검증을 위한 속성은 표준 인터페이스의 요구사항으로부터 도출한다. 하지만 수평적 관점은 운영체제의 요구사항 및 설계에서 모델을 도출하고, 수직적 관점은 소스코드로부터 모델을 도출한다. 수평적 모델을 생성하여 수직적 검증을 수행할 영역을 도출하는 과정을 소프트웨어 설계 단계, 즉 소프트웨어가 구현되기 전에 수행하는 것이 가능하다. 각 관점에 따라 모델링을 수행할 범위와 심도를 결정하는 것이 중요한 만큼, 이러한 특징을 반영할 수 있는 검증 도구를 선택하는 것 역시 중요하다. 이어지는 절에서 관점별 모델링 방법 및 검증 방법을 상세히 설명한다.

4.2 수평적 관점

수평적 관점의 모델링은 시스템의 최상위 수준에서 이루어진다. 시스템의 최상위 수준이란 그림 1의 응용 소프트웨어(Application Software)가 있는 수준을 의미하며, 최상위 수준의 모델링을 위해서는 응용층에 존재하는 응용 소프트웨어와 운영체제를 모델링해야 한다. 이와 같이 넓은 범위의 소프트웨어를 모델링하기 위해서는 응용 소프트웨어의 복잡도를 최소화해야 할 뿐만 아니라, 요구되는 운영체제의 기능의 복잡도도 최소화할 수 있도록 추상화해야한다. 수평적 모델의 복잡도를 줄이기 위하여 표 1과 같은 사항을 고려해야 한다.

표 1 수평적 관점의 모델링에서 모델의 복잡도를 줄이기 위한 고려사항

Table 1 Factors to reduce the complexity of model for the horizontal view

Model	Considerations
Application	- Number of processes - Complexity of operations - Features of communication
OS	- Scheduling policy - Management methods of processes - Assigned resource size for processes - HW-specific features

그림 3은 수평적 관점으로 표준 인터페이스를 준수하는 운영체제를 검증하기 위한 절차를 나타낸 그림이다. 대상을 수평적으로 모델링(M_H)하기 위해서는 인터페이스를 구현한 구조를 그대로 유지하지만 각 모듈의 역할을 최대한 추상화한다. 도출한 M_H를 검증하기 위해서 표준 인터페이스의 요구사항에서 검증 속성(P_n)을 도출한다.

인터페이스를 구현한 소프트웨어의 특정 기능을 실행하는 것은 응용 소프트웨어의 동작에 달렸다. 예를 들어 운영체제를 부팅하는 과정에 해당하는 기능들은 응용 소프트웨어와 무관하게 실행되지만, 응용 소프트웨어 관리 또는 통신과 같은 기능은 응용 소프트웨어가 구동되지 않으면 동작하지 않는다. 따라서 검증 목적에 따라 응용 소프트웨어를 모델링 할 필요가 있으며, 검증 목적에 해당하는 검증 속성에 기반하여 응용 소프트웨어의 모델(M_{An})을 생성해야 한다. 이렇게 도출한 모델과 속성을 대상으로 검증 도구(model checker 등)를 이용해 검증을 수행한다.

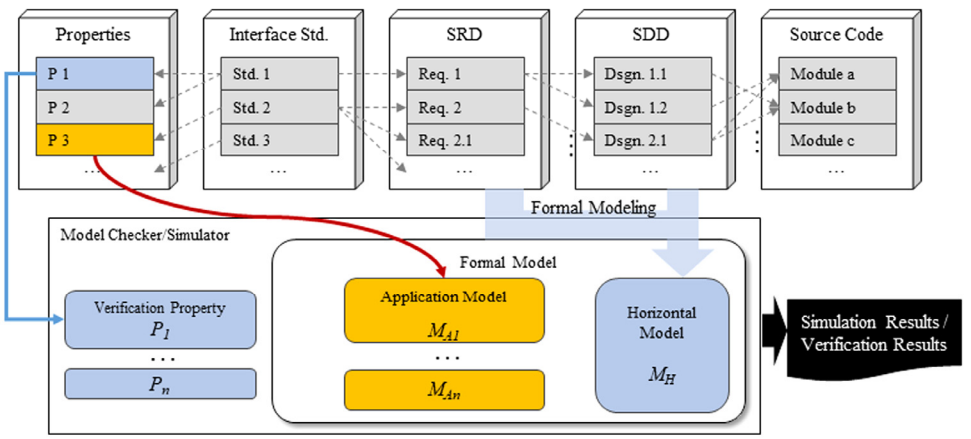

그림 3 표준 인터페이스를 준수하는 운영체제에 대한 수평적 관점에서의 검증 절차

Fig. 3 Verification process (horizontal view) for an operating system complying with a standard interface

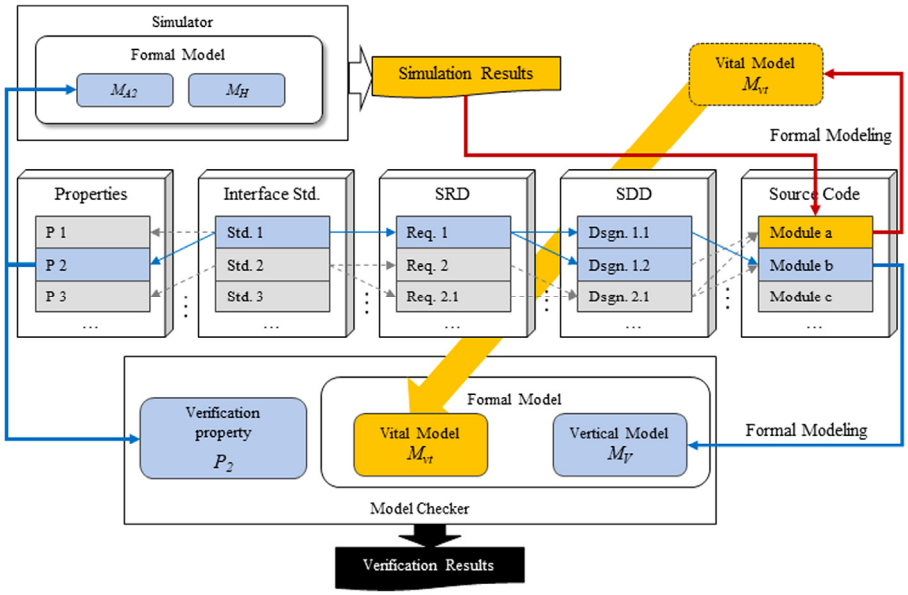

그림 4 표준 인터페이스를 준수하는 소프트웨어에 대한 수직적 관점에서의 검증
Fig. 4 Verification process (vertical view) for an operating system complying with a standard interface

수평적 관점을 적용하여 정형검증을 수행하기 위해서는 M_H, M_{An}, P_n을 모두 반영할 수 있는 검증도구를 확보해야 한다. Statemate[18]는 시스템 수준의 설계, 모델링, 시뮬레이션이 가능한 도구로써, Activity-chart, State-chart, Flow-chart 등의 다양한 모델링 방식을 제공한다. 또한 연동되는 Model Checker를 사용해 개발한 모델의 정형검증이 가능한 점도 제안하는 수평적 접근 방법의 모델링 및 검증도구로 적합한 이유이다.

수평적 관점의 모델링 및 검증 결과는 수직적 관점의 모델링 시 참조해야 하는 정보이므로 모델 및 속성에 대한 추적성도 확보해야 한다. 수직적 관점에서 수평적 관점의 결과를 참조하는 방식은 다음 절에서 상세히 설명한다.

4.3 수직적 관점

표준 인터페이스를 준수하는 운영체제를 대상으로 수직적으로 검증하는 과정을 그림 4에 나타내었으며, 검증 절차를 그림의 속성 P_2를 예시로 들어 나열하면 다음과 같다.

1. 검증을 위한 속성(P_2) 선정
2. P_2을 검증하기 위한 검증도구(model checker) 선정
3. P_2에 해당하는 소스코드의 모듈 추적(Module b) 및 모델링(M_V)
4. 수평적 모델의 시뮬레이션 결과를 참조하여 M_{A2}실행에 필수적인 모듈(Module a)을 도출하고 모델링(M_{vt})
5. M_V와 M_{vt}를 포함한 모델이 P_2를 만족하는지 검증

표 2 정형검증을 지원하는 도구 및 특징
Table 2 Tools and features for formal verification

Tools	Features
UPPAAL [19]	- Verification of real-time systems - GUI-based timed automata modeling
TIMES [20]	- Verification of schedulability for real-time systems - Source code generation from model
SPIN [21]	- Verification of multi-thread software - Communication algorithms and protocol
CBMC [22]	- Verification of embedded software - Direct verification of ANSI-C programs

정형검증을 수행할 수 있는 다양한 도구 중, 단계 1에서 선정한 검증 속성에 적합한 도구를 선정하는 것이 수직적 관점의 핵심이다. 표 2는 검증도구 종류와 그 특징을 간단히 보여준다. 검증 대상의 실시간성에 대한 검증을 수행하기 위해서는 시간을 모델링할 수 있는 UPPAAL이나 TIMES와 같은 도구를 사용해야 한다. 만일 표준 인터페이스가 여러 프로세스의 통신에 대한 서비스를 제공하며, 이에 대한 검증이 필요하다면 SPIN과 같은 도구를 사용하는 것이 유리하다. 대상 소프트웨어를 C/C++로 개발한 경우에는 새로운 모델링 언어로 모델링하는 과정을 줄일 수 있는 CBMC와 같은 도구를 사용하는 것도 고려해볼 수 있다. 나열한 도구 외에도 다양한 도구들이 존재하며 목적에 맞는 검증도구를 선택하여 사용할 수 있다.

검증 속성에 해당하는 모듈을 소스코드에서 찾아 모델(M_V)로 만들기 위해서는 해당 소스코드를 정확하며 체계적으로 찾을 수 있는 방법이 존재해야 한다. 이를 위해 개발 산출물의 추적성을 기반으로 속성에 해당하는 소프트웨어 모듈을 찾아낼 수 있다. 인터페이스 표준으로부터 소프트웨어의 요구사항(SRD), 설계(SDD), 소스코드에 이르는 경로를 파악하여 대상을 추적한다.

추적을 통해 속성에 해당하는 특정 모듈을 지정하고 모델링하면서, 해당 모듈이 실행되는 환경에 대한 모듈 및 모델(M_{vt})도 함께 도출해야 한다. 모델 M_{vt}가 갖춰지지 않을 경우 전체 모델을 실행할 수 없거나 부정확한 검증 결과가 도출될 수 있다. 예를 들어 운영체제가 지원하는 프로세스 간 통신에 대한 검증을 수행하는 경우, 통신 기능을 사용하는 응용 소프트웨어와 응용 소프트웨어를 스케줄링하기 위한 기능이 추가로 모델링 되어야 한다. 만일 환경이 갖춰지지 않는다면 통신 모듈에 대한 모델을 실행할 수 없거나, 운영체제가 지원하지 않는 스케줄링 방식으로 소프트웨어가 동작해서 원치 않는 결과를 도출하게 될 수도 있다. 다만 위와 같은 상황에서도 검증 대상에 대한 모델은 구체적인 수준으로 모델링해야 하지만, 환경에 대한 모델은 반드시 필요한 범위만을 포함하여 전체 모델의 복잡도를 낮춰야 한다.

주변 환경에 대한 모델은 수평적 접근 방법의 검증 또는 시뮬레이션 결과를 활용하여 찾을 수 있다. 단계 1에서 선정한 검증 속성에 대한 수평적 접근 방법의 결과를 분석하여, 동일한 속성에 대한 응용 소프트웨어를 실행할 경우에 함께 작동하거나 영향을 받는 모듈을 식별할 수 있다. 특히 Statemate와 같이 GUI방식의 시뮬레이션이 가능한 도구를 사용할 경우에는 전체 시스템의 실행 결과를 단계별로 파악할 수 있기 때문에 보다 쉽게 M_{vt}에 해당하는 영역을 식별할 수 있다.

5. 사례 연구

본 논문에서 제시한 검증 프레임워크는 ARINC-653을 준수하며 개발 산출물간의 추적성이 확보된 운영체제에 적용하는 것이 가능하다. ETRI에서 개발한 Qplus-AIR를 대상으로 프레임워크를 통한 검증의 유용성을 확인한다. 표 3은 수평적 검증을 위한 속성으로 ARINC-653에서 도출한 속성을 비롯하여 기본적으로 검사해야 할 속성을 정리하여 보여준다.

5.1 Qplus-AIR의 수평적 검증

Qplus-AIR의 모델링은 Statemate에서 지원하는 3가지 그래픽 언어(Activity-chart, Statechart, Flowchart)를 사용하여 수행하였다. Activity-chart는 시스템의 특정 기능을 의미하는 단위로 사용되며, 해당 기능은 Statechart와 Flowchart를 사용해 명세를 한다. Statechart는 유한

표 3 수평적 검증을 위한 속성

Table 3 Verification properties for the horizontal verification

ID	Description	ARINC-653
$P_{0.1}$	No random restarts	-
$P_{0.2}$	No deadlock	-
$P_{1.1}$	Partition runs iif time is assigned.	2.3.1.3
$P_{1.2}$	Process scheduling is allowed iif the mode of the partition is Normal	2.3.1.4.1
$P_{1.3}$	Process runs iif time is assigned to the partition of the process.	2.3.2
$P_{2.1}$	Mode transitions of partition are allowed iif it is predefined.	2.3.1.4

상태머신(Finite State Machine)과 유사한 형태로 시스템의 기능을 나타내고, Flowchart는 특정 기능을 흐름 순서로 정의한 것이다. Statechart는 시간의 흐름과 상태전이 조건에 따라 시스템의 행위를 결정짓는 반면, Flowchart는 시간의 흐름 없이 정해진 특정 기능을 수행한다는 차이점이 있다. 그림 5는 Statemate를 사용하여 구현한 Qplus-AIR의 모델을 나타낸다. 모델링을 통해 생성되는 모든 Chart는 계층 구조에 따라서 트리 형식으로 관리되며 모델의 상세한 내용은 지면의 제한으로 생략한다.

Qplus-AIR 분석에서 시작하여 모델링 및 검증에 2명, 3.5개월이 소요되었으며, 검증 속성을 위반하는 상태에 도달하는지 확인하기 위하여 최대 10000 Engine-Depth까지 상태 탐색 깊이를 넓혀가며 검증을 수행하였다. 속성 검증 결과는 표 4에 나타나 있다. ARINC-653에 정의되지 않은 Partition 모드전이 중 일부가 발생할 수 있음을 확인하였다($P_{2.1}$). 해당 모드전이는 실제 시스템에 영향을 주지 않거나, 추가적인 전제조건을 바탕으로 발생할 수 없다는 결론을 내릴 수 있다. 하지만 정의되지 않은 다른 전이에 대한 처리방식과 일관되도록 수정할 필요가 있음을 통보하였다.

수평적 검증 모델을 통해 수행한 검증 결과를 토대로 보다 상세한 수준의 검증을 수행하기 위해 수직적 접근 방법을 사용해 검증을 수행한다. 수평적 모델에서 확인한 파티션의 모드전이를 비롯하여 프로세스의 상태 전이를 수직적 접근 방법을 사용해 추가로 검증하며, 임계구역 접근과 관련된 파티션 내의 통신에 대한 상세한 검증도 추가로 수행하였다.

5.2 Qplus-AIR의 수직적 검증

수평적 검증의 결과 중, 상세한 검증이 필요한 부분과 해당 검증에서 확인할 수 없었던 속성에 대하여 검증하기 위해 수직적 검증을 수행한다. 수직적 검증은 속성에 따라 다양한 검증도구를 활용해야 한다. 예를 들어 스케줄링에 관한 검증을 위해서는 TIMES를 활용할 수 있으며, 검증 속성을 실제 코드를 활용해 구체적으로 재확

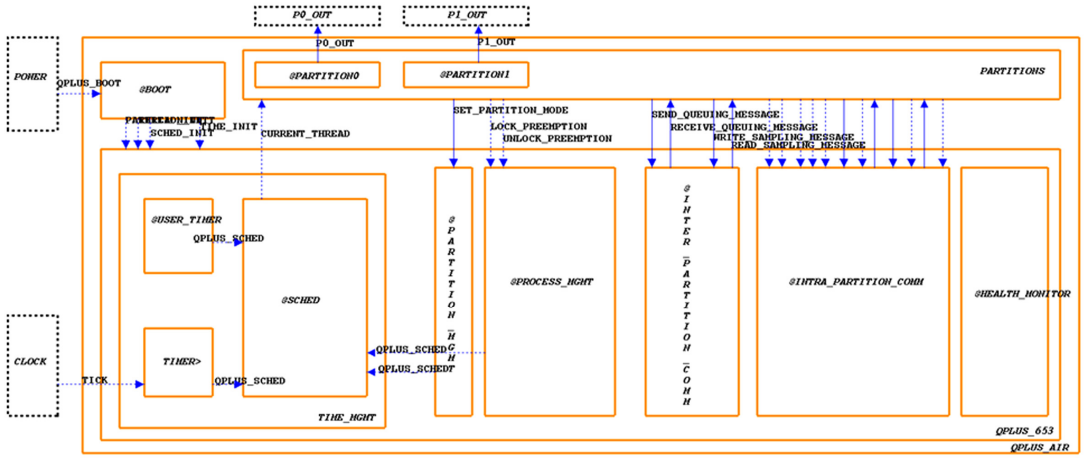

그림 5 Statemate로 모델링한 Qplus-AIR의 수평적 모델
Fig. 5 The horizontal model of Qplus-AIR using Statemate

표 4 Statemate를 활용한 수평적 검증 결과
Table 4 Results of the horizontal verification using Statemate

ID	Results	Engine-Depth
$P_{0.1}$	Satisfied	10000 (Not Reachable)
$P_{0.2}$	Satisfied	10000 (Not Reachable)
$P_{1.1}$	Satisfied	10000 (Not Reachable)
$P_{1.2}$	Satisfied	10000 (Not Reachable)
$P_{1.3}$	Satisfied	10000 (Not Reachable)
$P_{2.1}$	*Not satisfied*	14 (Reachable)

인하기 위해서는 CBMC를 사용할 수 있다. 본 논문에서는 제안하는 프레임워크를 사용한 모델 도출 과정을 명확하게 보여줄 수 있는 대표적인 수직적 검증 결과 중 일부만을 보인다.

수평적 접근으로 확인할 수 없는 속성 중 대표적인 것은 프로세스 간 통신에 관한 부분이다. 이는 수평적 접근 방법의 특성상 OS 커널에 해당하는 모듈에 대한 구체적인 모델링을 수행하지 않기 때문이며, 이와 더불어 Statemate를 활용한 검증에 있어서 프로세스의 통신에 대한 속성을 철저히 검증하는 것에 한계가 있기 때문이다.

ARINC-653은 파티션 내부의 프로세스 사이의 세마포어를 활용한 임계구역 접근 제어를 지원한다. 해당 표준은 세마포어 서비스를 활용한 접근 제어 시, 설정된 값보다 적은 수의 프로세스의 접근을 허용해야 하는 것을 강제한다. 표 5는 ARINC-653에서 요구하는 세마포어에 대한 제약사항(P_6)을 나타낸 것이다. P_6과 같은 속성을 검증하기 위하여 통신에 대한 검증에 특화된 모델 체커 SPIN을 사용한다.

P_6에 해당하는 소스코드를 알아내기 위하여, 속성에서부터 소스코드 모듈에 이르는 전 산출물 간의 추적성

표 5 수직적 검증을 위한 검증속성
Table 5 Verification properties for the vertical verification

ID	Description	ARINC-653
P_6	The number of processes is always a semaphore's initial value or less.	2.3.6.2.1

을 이용한다. 추적성 분석을 이용해 소스코드의 12개의 함수를 찾아냈다. P_6을 추적하여 생성한 모델(M_{T6})과 함께 실제 통신이 발생하며 검증 속성을 확인할 수 있도록 환경에 대한 모델(M_{V6})을 추가해야 한다. M_{V6}를 모델링하기 위하여 다음과 같인 응용 소프트웨어 모델을 Statemate모델에 추가하였다.

파티션 내부의 통신(Intra-partition communication) 및 외부(Inter-partition communication) 통신에 대한 속성을 반영하는 응용 모델 M_A를 개발하였다. 2개의 Partition이 각각 2개의 Process를 실행하도록 설계된 모델이다. 각 Process는 파티션 내부의 다른 Process와 통신하거나 파티션 외부의 다른 Process와 통신 하도록 응용 소프트웨어에 해당하는 모델을 생성하였다. M_A는 파티션 내부 통신에 해당하는 속성인 P_6과 함께 파티션 외부 통신에 대한 응용 모델도 포함한다. P_6과 관련된 모델 M_{V6}를 도출하기 위해서 내부 통신에 해당하는 시뮬레이션 결과만을 활용한다.

그림 6은 M_A를 포함한 Qplus-AIR의 전체 모델을 Statemate를 사용해 시뮬레이션하는 화면이다. 본 시뮬레이션을 통해 Qplus-AIR 모델의 통신 서비스를 이용할 때, 해당 부분을 제외한 부팅과 스케줄링 관련 모델이 실행됨을 확인하였다. Statemate 모델과 소스코드 사이의 추적성을 통해 총 6개의 함수가 영향을 받는 것을 확인하였으며, 해당 함수를 M_{V6}로 구현하고 M_{T6}와

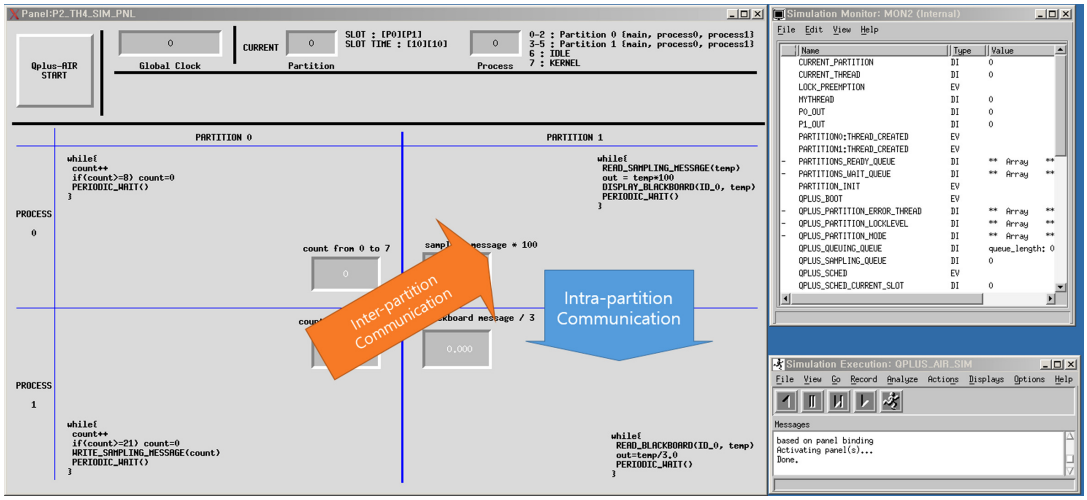

그림 6 M_A를 포함한 수평적 모델의 시뮬레이션 화면

Fig. 6 A screen dump of simulation of the horizontal model including M_A

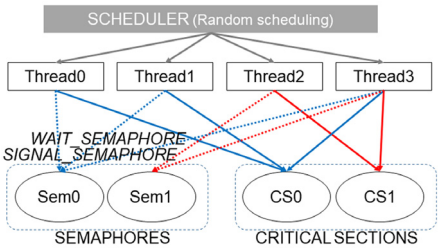

그림 7 세마포어 서비스의 정형검증을 위한 수직적 모델

Fig. 7 The vertical model for the verification of the semaphore service

함께 SPIN을 이용해 검증을 수행하였다. 그림 7은 Qplus-AIR의 파티션 내부통신 서비스 중 하나인 세마포어를 검증하기 위한 M_{V6}와 M_{T6}를 포함한 전체 모델을 그림으로 나타낸 것이다. SPIN을 사용하여 검증하기 위해서는 SPIN의 모델링 언어인 PROMELA로 모델링해야 한다.

SPIN을 사용하여 Qplus-AIR의 세마포어 서비스에 대한 정형검증 수행을 통하여 현재까지 발생한 이력은 없지만 세마포어 서비스에 잠재적 결함이 있음을 발견하였다. SPIN이 생성한 반례(Counter example)를 효과적으로 확인하기 위하여 상태 탐색의 깊이를 줄여가며 검증을 수행하였다. 찾아낸 최소한의 탐색 영역은 약 2만개의 State, 320 Depth이며, 반례를 분석한 결과 세마포어 값보다 많은 수의 프로세스가 세마포어를 요청했을 경우에 발생하는 것으로, 세마포어를 처리하는 방법 및 프로세스 스케줄링이 특수한 순서로 이루어진 경우에 나타날 수 있음을 확인하였으며, 해결할 수 있는 방안을 제시하였다.

6. 결론 및 향후 연구

정형검증 기법을 사용한 검증은 대상과 대상의 범위에 대한 제약으로 인해서 운영체제처럼 규모가 크고 기능이 많은 시스템을 대상으로 수행할 때 한계점을 가지고 있다. 이러한 한계점을 보완하기 위해서는 큰 규모의 대상을 합리적은 수단으로 나눠 검증하는 것이 하나의 방법이 될 수 있다. 본 논문에서는 표준 인터페이스를 준수하는 운영체제를 추적성 분석에 기반한 수평·수직적 접근 방법을 통해 정형검증 할 수 있는 프레임워크인 FMProjector를 소개하였다. Qplus-AIR를 대상으로 수행한 사례연구를 통해 속성에 맞는 정형 모델을 도출하고 검증을 수행하는 절차를 상세히 보였으며, 유의미한 검증 결과를 도출할 수 있음을 확인하였다.

현재 본 논문의 프레임워크를 사용하기 위해서는 수평적 접근방법에 Statemate라는 정해진 도구를 사용해야 하며, 해당 도구의 시뮬레이션 기능을 이용해 응용 모델에 영향을 받는 영역을 찾을 수 있는 능력을 보유해야만 한다. 이러한 제약사항을 개선하기 위하여 수평적 모델을 정형화하고, 속성에 따라 영향을 받는 영역을 모델을 논리적으로 도출하기 위한 투영(Projection) 기법을 연구 중에 있다.

References

[1] IEC, *61508: Functional Safety of Electrical/ Electronic/ Programmable Electronic Safety-related Systems*, 2010.

[2] Radio Technical Commission for Aeronautics (RTCA), *DO-178B: Software Considerations in Airborne*

Systems and Equipment Certification, 1992.
[3] E. M. Clarke and J. M. Wing, "Formal methods: State of the art and future directions," *ACM Computing Surveys (CSUR)*, Vol. 28, No. 4, pp. 626-643, 1996.
[4] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, "Model checking and the state explosion problem," *LASER Summer School on Software Engineering*, Springer, Berlin, Heidelberg, 2011.
[5] D. Choi, et al., "A Study on Development of UAV Flight Control Software using Qplus-AIR," *The Korean Society For Aeronautical And Space Sciences Spring Conference 2012*, pp. 1176-1180, 2012.
[6] Airlines Electronic Engineering Committee, *Avionics Application Software Standard Interface-ARINC Specification 653-Part 1 - Required Service*, Aeronautical Radio Inc., 2006.
[7] T. Kim, et al., "Qplus/Esto-AIR: DO-178B Level A Certified RTOS and IDE for Supporting ARINC 653," *Communications of the Korean Institute of Information Scientists and Engineers*, Vol. 30, No. 9, pp. 65-70, 2012.
[8] T. Kim, et al., "A Certification Experience of Qplus-AIR by DO-178B," *Korea Information Science Society*, Vol. 31, No. 5, pp. 32-39, 2013.
[9] Andronick, June, et al., "Large-scale formal verification in practice: A process perspective," *Proc. of the 34th International Conference on Software Engineering*, pp. 1002-1011. IEEE Press, 2012.
[10] Gerwin Klein, et al., "Comprehensive formal verification of an OS microkernel," *ACM Trans. Comput. Syst.*, Vol. 32, No. 1, Article 2 (Feb. 2014).
[11] Martin, W.B., White, P.D., and Taylor, F.S., "Creating high confidence in a separation kernel," *Automated Software Engineering*, Vol. 9, No. 3, pp. 263-284, 2002.
[12] Alkassar, E., Paul, W. J., Starostin, A., and Tsyban, A., "Pervasive verification of an OS microkernel," *International Conference on Verified Software: Theories, Tools, and Experiments*, Springer, pp. 71-85, 2010.
[13] Hohmuth, M., and Tews, H., "The VFiasco approach for a verified operating system," *Proc. 2nd ECOOP Workshop on Programm Languages and Operating Systems*, 2005.
[14] Choi, Yunja, "Model checking trampoline OS: a case study on safety analysis for automotive software," *Software Testing, Verification and Reliability*, Vol. 24, No. 1, pp. 38-60, 2014.
[15] Y. Huang, Y. Zhao, L. Zhu, Q. Li, H. Zhu and J. Shi, "Modeling and Verifying the Code-Level OSEK/VDX Operating System with CSP," *2011 Fifth International Conference on Theoretical Aspects of Software Engineering*, pp. 142-149, 2011.
[16] J. Shi, J. He, H. Zhu, H. Fang, Y. Huang and X. Zhang, "ORIENTAIS: Formal Verified OSEK/VDX Real-Time Operating System," *2012 IEEE 17th International Conference on Engineering of Complex Computer Systems*, Paris, pp. 293-301, 2012.
[17] Yoon, Sanghyun, Dong-Ah Lee, Eunji Pak, Taeho Kim, and Junbeom Yoo, "Timed Model-Based Formal Analysis of a Scheduler of Qplus-AIR, an ARINC-653 Compliance RTOS," *IEICE TRANSACTIONS on Information and Systems Vol.E100-D*, no. 10, 2644-2647, 2017.
[18] Harel, David, and Michal Politi. *Modeling reactive systems with statecharts: the STATEMATE approach*, McGraw-Hill, Inc., 1998.
[19] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi, "Uppaal - A Tool Suite for Symbolic and Compositional Verification of Real-Time Systems," *Proc. of the 1st Workshop on Tools and Algorithms for the Construction and Analysis of Systems, LNCS 1019*, Springer, 1995.
[20] Amnell, Tobias, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi., "TIMES: A tool for modelling and implementation of embedded systems," *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 460-464. Springer, Berlin, Heidelberg, 2002.
[21] Holzmann, G. J., "The model checker SPIN," *IEEE Transactions on software engineering*, Vol. 23, No. 5, pp. 279-295, 1997.
[22] Clarke, Edmund, Daniel Kroening, and Flavio Lerda, "A tool for checking ANSI-C programs," *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, Berlin, Heidelberg, 2004.

이 동 아

2010년 건국대학교 컴퓨터공학과 졸업(학사). 2012년 건국대학교 컴퓨터공학과 졸업(석사). 2018년~2019년 한국정보통신기술협회 선임연구원. 2012년~현재 건국관대학교 컴퓨터·정보통신공학과 박사과정. 관심분야는 정형검증 및 위험분석

유 준 범

2005년 KAIST 전자전산학과 전산학 전공 졸업(박사). 2008년 삼성 전자주식회사 통신연구소 책임연구원. 2008년~현재 건국대학교 컴퓨터공학부 부교수. 관심분야는 소프트웨어 공학, 안전성 분석, 정형기법