제33회 한국정보처리학회 춘계학술발표대회 논문집 제17권 제1호 (2010. 4)

SPARK Ada 기반 안전필수 내장형 시스템 개발

오준석*, 김진현, 최진영
고려대학교 컴퓨터・전파통신 공학과
e-mail : {jsoh, jhkim, choi}@formal.korea.ac.kr

Developing Safety-critical Embedded System using SPARK Ada

Joon-Seok Oh*, Jin-Hyun Kim, Jin-Young Choi
Dept of Computer and Radio Communications Engineering, Korea University

요 약

소프트웨어가 대형화되고 복잡해짐에 따라 발생하는 오류가 증가되고 있다. 안전성이 특히 중요시되는 안전필수(safety-critical) 내장형 시스템에서 오류가 발생하면 인명상의 피해 또는 재산상의 피해를 야기한다. 개발 후, 테스팅을 통해 이런 오류를 찾는 비용은 매우 크고, 모든 오류를 찾는 것은 불가능하다고 인식되고 있다. 따라서 소프트웨어 개발단계에서 이런 오류를 탐지하고 제거하려는 노력이 증대되고 있다. 본 논문에서는 SPARK Ada를 사용하여 안전필수 내장형 시스템을 개발할 때, 오류를 제거할 수 있는 흐름분석(flow analysis) 기법을 사용하여 특정한 타입의 오류를 제거할 수 있음을 보인다. 또한 이를 적용하여 안전필수 시스템을 개발한다.

1. 서론

소프트웨어는 점점 대형화되고 복잡해지고 있으며, 내장형 시스템에 탑재되고 있다. 이런 대형화와 복잡성이 증가됨에 따라 발생하는 소프트웨어 오류가 증가한다. 이런 오류는 개발 후 테스팅을 통해 탐지하는 비용이 매우 크며, 테스팅을 통해 모든 오류를 탐지하는 것은 불가능하다고 인식되어 있다. 따라서 이런 오류를 개발단계에서 탐지하여 예방해야 한다. 안전성이 특히 강조되는 안전필수(safety-critical) 시스템에서 소프트웨어 오류가 발생하면 인명상의 피해 혹은 큰 재산상의 피해를 입는다. 이런 피해로는 최근에 많은 전문가들이 말하는 도요타 자동차 사고가 있다[1]. 또 다른 예로 1996년 6월에 발생했던 Ariane 5 로켓발사 사고가 있다. 개발단계에서 발견하지 못한 오류를 개발 후에 찾는 것은 매우 힘들고 막대한 비용이 든다는 것을 이런 사건들을 통해 알 수 있다.

복잡하고 대형화된 안전필수 내장형 시스템을 개발할 때, 안전한 운행을 위해 소프트웨어 오류를 개발단계에서 탐지하고 제거해야 한다. 개발단계에서 오류를 제거하기 위한 기법으로 정형기법(formal method)이 있다. 정형기법은 정형명세와 정형검증으로 구성되며, 이를 통해 안전성에 대한 속성을 명세하고 검증한다. 만약 이 속성에 오류가 있는 경우, 이를 탐지하여 제거할 수 있다. 이와 관련된 기법으로 계약에 의한 설계(Design by Contract)가 있다. 이 기법은 정형기법에 기반하고 있다. 이 기법에서 사용되는 계약은 인터페이스에 작성되며, 설계자와 코드개발자 사이의 계약이다. 이 계약은 설계자에 의해 명세가 되고, 이후에 코드개발자에 의해 코드가 완성되면 정형검증을 한다. 이를 통해 코드가 명세를 만족함을 보장할 수 있다. 따라서 코드가 명세를 만족하지 않는 경우를 탐지하며, 이는 코드개발자에 의해 제거되어야 한다. 따라서 소프트웨어 오류를 개발단계에서 제거할 수 있다.

SPARK Ada는 안전 및 보안 필수 소프트웨어를 개발하는데 적합한 언어이다[2]. 이는 계약에 의한 설계 기법을 지원한다. 따라서 코드가 명세를 만족하지 않는 경우, 오류를 탐지하고 이를 제거할 수 있다. SPARK Ada의 주요 특징 중 하나는 어노테이션 기법이다. 이를 통해 코드의 흐름분석(flow analysis) 및 증명분석(proof analysis)을 한다. SPARK Ada에서 흐름분석은 필수적이고, 증명분석은 필요에 따라 추가적으로 사용된다. 따라서 본 논문에서는 SPARK Ada에서 꼭 수행해야 하는 흐름분석 기법을 통해 특정 타입의 오류를 제거하는데 초점을 맞춘다.

현재까지 국내의 안전필수 내장형 시스템에 SPARK Ada를 적용하여 개발한 사례가 없기 때문에 본 논문에서는 SPARK Ada를 사용하여 국내의 안전필수 내장형 시스템을 개발하고자 한다. 또한 흐름분석 기법을 통해 소프트웨어의 오동작을 야기할 수 있는 특정 타입의 오류들을 개발단계에서 탐지하고 제거하여, 이와 같은 오류가 코드에 없음을 보장한다.

본 논문의 구성은 다음과 같다. 2장에서는 SPARK Ada의 어노테이션 기법을 자바언어에 대해 사용하는 JML, 그리고 이와 연관하여 계약에 의한 설계 기법을 간단하게 소개를 하고, 3장에서는 SPARK Ada의 간단한 소개와 흐름분석 기법을 통해 어떤 오류들을 초기에 탐지할 수 있는지를 소개하고, 4장에서는 이를 바탕으로 국내의

안전필수 내장형 시스템을 개발한다. 마지막으로 5장에서는 결론 및 향후연구로 맺는다.

2. 관련연구

JML[3]은 SPARK Ada에 있는 어노테이션 기법을 자바언어에 대해 사용하며, 자바 프로그램에 대한 명세언어이다. JML은 SPARK Ada와 마찬가지로 계약에 의한 설계 기법을 지원하기 때문에 코드가 명세를 만족하지 않는 코드부분을 탐지할 수 있다. 본 논문에서 SPARK Ada를 사용한 이유는 SPARK Ada가 안전필수(safety-critical) 시스템 및 기능필수(mission-critical) 시스템을 개발하는데 적합한 언어이며, 이런 시스템들을 개발하는데 사용되었기 때문이다.

계약에 의한 설계[4]는 정형기법[5]에 근원을 두고 있는 기법이다. 다양한 프로그래밍 언어들이 이 기법을 지원하고 있으며[6], SPARK Ada는 그 중에 하나이다. SPARK Ada에서 지원되는 이 기법의 목적은 코드가 명세를 만족함을 보장하기 위함이다. 이를 위해 정형기법의 명세와 검증과정을 한다. 설계자는 SPARK Ada의 어노테이션을 이용해 SPARK 인터페이스에 명세를 한다. 이후에 코드개발자에 의해 SPARK 인터페이스의 구현이 완성되면, 코드가 명세를 만족하는지를 검증한다. 이 과정을 통해 개발단계에서 명세를 만족하지 않는 코드를 탐지하고 제거할 수 있다.

3. SPARK Ada

SPARK Ada는 고무결성(High integrity) 어플리케이션을 개발하기 위해 고안된 프로그래밍 언어이다. 고무결성은 안전성(safety)과 보안성(security)을 포함하는 의미이다. 따라서 안전필수 내장형 시스템을 개발하는데 적합한 언어이다. 해외에서 안전필수 내장형 시스템을 개발하는데 SPARK Ada를 사용한 프로젝트들은 SHOLIS, MULTOS CA, 그리고 LOCKHEED C130J 등 많이 있다[7].

SPARK Ada는 패키지 단위로 프로그램을 작성한다. 패키지에는 서브프로그램을 선언하는 인터페이스와 이를 실제 구현하는 패키지 몸체(package body)로 구성되어 있다. SPARK Ada에서 사용되는 함수(function)와 프로시저(procedure)를 서브프로그램이라고 하고, 인터페이스를 SPARK 명세(specification), 그리고 패키지 몸체를 SPARK 몸체(body)라고 한다.

SPARK Ada의 중요한 특징 중 하나는 어노테이션(annotation) 기법이다[8]. 이 어노테이션을 이용하여 계약에 의한 설계 기법을 실행할 수 있다. 따라서 코드가 명세를 만족하지 않는 코드 부분을 탐지할 수 있고, 탐지된 부분은 코드개발자에 의해 제거될 수 있다.

3.1 흐름분석(flow analysis)기법

SPARK Ada에 있는 흐름분석기법을 사용하면 특정한 타입의 오류들을 탐지하여 제거할 수 있다. 이런 특정한 타입의 오류들에는 서브프로그램에 사용된 전역변수 및 파라메타가 변수 모드를 위배하고 사용된 경우이다. SPARK 서브프로그램에 사용된 변수들은 모드를 가지며, 이 모드에는 in, in out, 그리고 out이 있다. in 모드는 초기화된 변수가 서브프로그램에 사용되었음을 의미하고, in out 모드는 초기화된 변수가 서브프로그램에 사용되어 업데이트가 된다는 것을 의미한다. 그리고 out 모드는 초기화되지 않은 변수가 서브프로그램에 의해 업데이트된다는 것을 의미한다. 예를 들어, 서브프로그램에 in 모드로 선언된 변수 var가 있다고 할 때, 탐지할 수 있는 오류는 서브프로그램이 초기화되지 않은 변수 var사용, 서브프로그램이 초기화된 변수 var를 사용하여 업데이트 한 경우, 그리고 서브프로그램에 변수 var가 사용되지 않은 경우이다. 이 과정은 SPARK 도구에 의해 진행된다.

흐름분석기법은 SPARK Ada의 서브프로그램에 사용되는 주 어노테이션(core annotation)을 사용하여 작성된다. 주 어노테이션은 서브프로그램에 사용된 변수들과 관련하여 SPARK 명세를 위해 사용된다. SPARK 몸체가 완성되면, SPARK 도구에 의해 흐름분석이 수행된다. 서브프로그램에 사용되는 주 어노테이션에는 서브프로그램에 사용된 전역변수를 나타내는 “--# global”과 서브프로그램에 사용된 변수들의 관계를 나타내는 “--# derives .. from ..” 이 있다.

흐름분석 기법은 데이터흐름분석과 정보흐름분석으로 구성되어 있다. 데이터흐름분석은 프로시저에 선언된 ‘--# global’ 어노테이션에 관련되며, 특정한 모드를 가지고 프로시저에 선언된 전역변수 및 파라메타가 모드를 위배하기 않고 실제 구현에 사용되었는지를 분석한다. 정보흐름분석은 프로시저에 선언된 “--# derives .. from ..” 어노테이션과 데이터흐름분석에 사용된 변수들의 특정한 모드에 기초한다. 이를 통해 실제 구현에서 사용된 변수들의 관계가 올바르게 구현되었는지를 분석한다.

4. 사례연구 : DCM 모듈 개발

DCM(Distance Control Module)은 안전필수 내장형 시스템으로써 철도시스템에서 열차간격을 제어하는 시스템 모듈이다. DCM의 기능에는 열차간격제어, 열차위치확인, 열차이동/방향감시, 임시속도제한명령처리, 블록개방/폐쇄가 있다. 본 논문에서는 열차블록에 대한 블록개방/폐쇄에 관련된 모듈을 다룬다. SPARK Ada를 사용하여 이 모듈의 기능 및 검증되어야하는 속성들을 구현한다. 또한 흐름분석 기법을 통해 개발단계에서 특정 타입의 소프트웨어 오류를 제거할 수 있기 때문에, 시스템에 이런 오류가 없다는 것을 보장한다.

4.1 블록개방/폐쇄 모듈 : 블록관리자

블록관리자는 블록개방/폐쇄에 관련하여 블록을 관리하는 모듈이다. 블록은 블록 ID와 허용이동공간(PMA)에

대한 정보를 담고 있다. 허용이동공간(PMA)은 적색, 황색, 그리고 녹색의 값을 가질 수 있으며, 열차간격제어 모듈에서 열차의 이동을 제어하기 위해 사용된다.

블록개방/폐쇄 모듈은 전체 블록에 대한 개방 및 폐쇄가 가능하다. 블록폐쇄 전 허용이동공간(PMA)이 녹색으로 세팅되어 있으면, 블록폐쇄 후에는 적색으로 세팅이 된다.

블록관리자의 주요 역할은 다음 <표1> 과 같다.

<표 1> 블록관리자의 역할

블록개방/폐쇄	블록개방 명령을 받은 경우, 해당 블록개방이 개방 가능한 상태를 확인하여 블록을 개방한다. 비상 경우와 같이 블록폐쇄 명령을 받은 경우, 해당 블록을 폐쇄한다.
전체 블록개방	전체 블록개방 명령을 받은 경우, 전체 블록을 개방한다.
전체 블록폐쇄	전체 블록폐쇄 명령을 받은 경우, 전체 블록을 폐쇄한다.

4.2 SPARK 명세(SPARK specification)

4.1절의 블록관리자의 역할에 근거하면, 블럭관리자는 총 3개의 서브프로그램들로 구성된다. 다음 <표 2>는 블록관리자의 서브프로그램들에 대한 간략한 설명이다.

<표 2> 블록관리자의 서브프로그램

프로시저 이름	역할
BlockSetting	단일 블록에 대하여 개방 및 폐쇄를 수행한다. 프로그램 인수로 블록 ID와 PMA의 값을 받으며, 해당 블록 ID에 PMA 값을 세팅한다. 블록을 개방한 경우, 해당 블록의 PMA 값은 녹색으로 세팅된다. 블록을 폐쇄한 경우, 해당 블록의 PMA 값이 적색으로 세팅된다.
AllBlockOpen	모든 블록에 대하여 개방을 하는 프로시저이다. 모든 블록에 해당하는 PMA 값을 녹색으로 세팅한다.
AllBlockClose	모든 블록에 대하여 폐쇄를 하는 프로시저이다. 모든 블록에 해당하는 PMA 값을 적색으로 세팅한다.

다음 (그림 1)은 SPARK 어노테이션을 사용해 <표 2>에 있는 블록관리자의 서브프로그램들을 SPARK 명세로 나타낸 것이다. “--#” 로 시작하는 부분이 어노테이션이다. (그림 1)에서 사용된 “--# global” 및 “--# derives .. from ..” 어노테이션은 서브프로그램을 위한 주 어노테이션이고 흐름분석을 하는데 사용된다.

```
with DCMType;
--# inherit DCMType;
package BlockManager
--# own state1, state2;
is
   procedure BlockSetting(Bid : in DCMType.BlockIdType;  Pv : in
DCMType.PMA);
   --# global in out state1; out state2;
   --# derives state1 from state1, Bid, Pv &
   --#          state2 from ;

...

end BlockManager;
```

(그림 1) 블록관리자 SPARK 명세

이 어노테이션들에 근거하여 프로시저에 어떤 변수가 어떤 모드로 사용되었는지, 그리고 변수들의 관계가 어떻게 되는지를 알 수 있다. package BolckManager 바로 밑에 있는 “--# own”은 블록관리자가 2개의 전역변수(state1, state2)를 사용한다는 것을 나타낸다. SPARK 명세는 사용자에게 보이는 부분이기 때문에 위와 같은 추상화방법을 사용한다. 이 추상화된 부분(--# own state1, state2)은 SPARK 몸체를 구현할 때, refinement가 되어 구체적인 변수로 나타낸다. 위의 (그림 1)에서 프로시저 BlockSetting에 있는 “--# global” 어노테이션을 통해 다음의 정보를 알 수 있다.

1. 프로시저 BlockSetting은 전역변수 state1을 in out 모드로 사용한다.
2. 프로시저 BlockSetting은 전역변수 state2를 out 모드로 사용한다.

다음으로 “--# derives .. from ..” 어노테이션을 통해 알 수 있는 정보는 다음과 같다.

1. 프로시저 BlockSetting에서 전역변수 state1은 state1, Bid, 그리고 Pv에 의해 업데이트가 된다.
2. 프로시저 BlockSetting에서 전역변수 state2는 다른 변수들에 영향을 받지 않고 업데이트가 된다.

4.3 SPARK 몸체(SPARK body)

다음 (그림 2)는 4.2절에 있는 블록관리자 명세를 실제로 구현한 블록관리자 몸체이다.

```
package body BlockManager
--# own state1 is trainBlocks  &
--#     state2 is blockManagerMsg;
is
   trainBlocks : DCMType.Blocks;
   blockManagerMsg : DCMType.Result_Msg;

   procedure BlockSetting(Bid : in DCMType.BlockIdType;  Pv : in
DCMType.PMA)
   --# global in out trainBlocks; out blockManagerMsg;
   --# derives trainBlocks from trainBlocks, Bid, Pv &
   --#          blockManagerMsg from ;
   is
   begin
      trainBlocks(Bid) := Pv;
      blockManagerMsg := DCMType.CompleteBlockSetting;
   end BlockSetting;

...

end BlockManager;
```

(그림 2) 블록관리자 SPARK 몸체

블록관리자 몸체가 시작하는 부분을 보면, 블록관리자 명세에서 추상화된 전역변수들(--# own state1, state2)을 refinement하였다. 실제 구현에 사용되는 변수를 명시해 주어야 하기 때문이다. 따라서 SPARK 몸체에 사용된 어노테이션들은 refinement된 전역변수들이 삽입된다.

SPARK 도구는 흐름분석에 관련된 특정한 타입의 오류들을 제거하기 위해 어노테이션들과 실제 구현에 기초하여 검사를 수행한다. 즉, SPARK 몸체가 SPARK 명세를 만족함을 보장하기 위해 검증하며, 만족하지 않을 경우, 오류를 탐지한다.

먼저, 데이터흐름분석에 관련하여 전역변수 및 파라메타의 모드가 올바른지 검사한다. 예를 들어, (그림 2)에 있는 프로시저 BlockSetting에 있는 "--# global in out trainBlocks"에 관련하여 BlockSetting의 구현을 만족하지 않는 경우 탐지할 수 있는 오류는 다음과 같다.

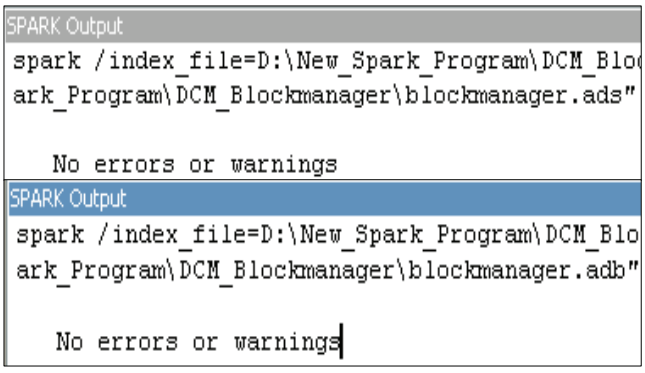
1. trainBlocks가 초기화되지 않고 업데이트만 된 경우.
2. trainBlocks가 초기화는 되었지만 업데이트가 되지 않은 경우.
3. trainBlocks 초기화되지 않고 업데이트도 되지 않은 경우.
4. trainBlocks이 서브프로그램 BlockSetting에 사용되지 않은 경우.

다음으로 변수들의 관계가 올바르게 구현되었는지를 검사한다. 예를 들면, (그림 2)에 프로시저 BlockSetting에 있는 "**--# derives** trainBlock **from** trainBlock, Bid, Pv"는 trainBlock가 이전의 trainBlock와 Bid, 그리고 Pv에 의해 새로운 값으로 업데이트 된다는 것을 나타낸다. 이 어노테이션의 정보에 근거하여 실제 구현이 이를 만족하는지 검사하고, 올바르지 않은 경우 오류를 출력한다.

이와 같이 "--# global"과 "--# derives .. from .."인 어노테이션을 사용하여 흐름분석 기법을 사용할 수 있으며, 이를 통해 특정 타입에 관련된 오류들을 개발단계에 탐지하고 제거할 수 있다.

4.4 블록관리자 흐름분석 결과

흐름분석에 관련된 소프트웨어 오류를 개발단계에서 탐지하고 수정하기 위해 SPARK 도구를 사용하여 4.2절과 4.3절에서 구현한 블록관리자 모듈을 분석한다. 다음 (그림 3)은 흐름분석을 한 결과이다.



(그림 3) 블록관리자의 흐름분석 결과

(그림 3)에서 blockmanager.ads는 블록관리자의 SPARK 명세를 검사한 것이고, blockmanager.adb는 블록관리자의 SPARK 몸체를 검사한 것이다. 위의 결과에서 볼 수 있듯이 블록관리자 모듈의 흐름분석의 결과가 오류 및 경고가 없다는 것을 확인 할 수 있다. 따라서 구현이 설계를 만족함과 이를 통해 특정 타입의 오류가 없음을 보장할 수 있다.

5. 결론 및 향후연구

본 논문에서는 SPARK Ada를 이용하여 현재까지 국내의 안전필수 내장형 시스템을 개발한 사례가 없는 국내 안전필수 내장형 시스템을 SPARK Ada를 이용하여 개발하였다. 계약에 의한 설계 기법에 따라 SPARK 어노테이션으로 명세하고, 구현된 코드가 명세를 만족함을 보장하기 위해 SPARK 도구를 통해 검사하였다. 그러므로 개발된 시스템에서 코드는 명세를 만족함을 보증할 수 있다.

시스템 개발과정에서 SPARK Ada의 흐름분석 기법을 적용하여, 개발 후 발생할 수 있는 소프트웨어 오류들을 개발단계에서 탐지하고 제거하여, 이런 특정 타입의 오류들이 발생하지 않다는 것을 보장할 수 있음을 보였다.

향후 연구로는 시스템의 안전성뿐만 아니라 보안성까지 고려하여, 보안속성에 관련하여 발생할 수 있는 특정 타입의 오류를 개발단계에서 탐지하고 제거하여, 이런 오류가 없음을 보장하기 위해 이를 적용하고자 한다.

참고문헌

[1] R.Wilson, "Toyota Prius and Camry, drive-by-wire, and out failure to learn from experience". In EDN News, 4 February 2010. "http://www.edn.com/blog/1690000169/post/630052463.html?nid=3351&rid=2872056"

[2] A.Ireland, B.J.Ellis, A.Cook, R.Chapman, J.Barnes. "An Integrated Approach to High Integrity Software Verification". Journal of Automated Reasoning : Special Issue on Empirically Successful Automated Reasoning, 29 September 2006

[3] Groslambert, J., Julliand, J., and Kouchnarenko, O. 2006. "JML-based verification of liveness properties on a class in isolation". In Proceedings of the 2006 Conference on Specification and Verification of Component-Based Systems (Portland, Oregon, November 10 - 11, 2006)

[4] Meyer, B., "Applying "design by contract"". IEEE Computer, vol. 25, no 10, pp 40-51, Octal 1992

[5] Formal Methods, "http://formal.korea.ac.kr/wiki/Public/Research"

[6] Wikipedia, Design by Contract, "http://en.wikipedia.org/wiki/Design_by_contract"

[7] Chapman, R., "Industrial Experience with SPARK", Ada Letters, vol. XX, no 4, pp 64-68, December 2000

[8] John Barnes, "High Integrity Software : The SPARK Approach to Safety and Security", Addison Wesley, 2002