인공위성 소프트웨어 무결점 검증

이욱세 @ 한양대학교

2009.07.09

제 2회 소프트웨어무결점연구센터 워크샵

인공위성 소프트웨어 검증팀 (한양대-서울대)

- 책임교수: 한양대학교 이욱세 교수
- 한양대 연구팀: 정승철, 박경수
- 서울대 연구팀: 오학주, 최원태
- KAIST 인공위성연구센터와 협력

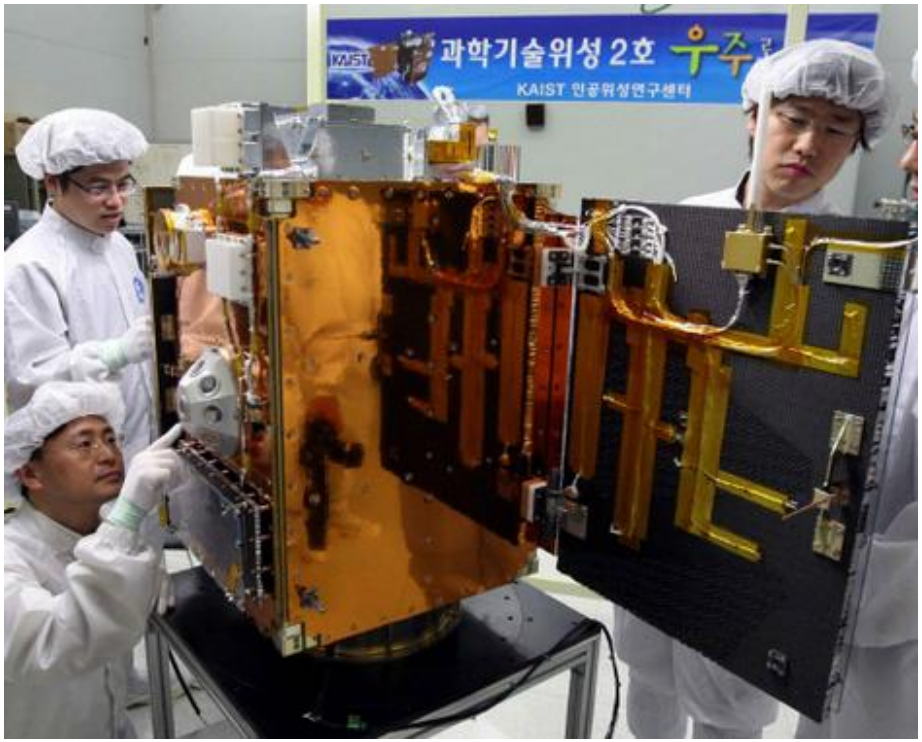
- 2008년 12월에 결성

KAIST 인공위성연구센터

- 1989.10 인공위성 연구센터 설립
- 1990.03 한국과학재단 **ERC** 선정
- 1992.08 최초 국적위성 우리별1호 발사
- 1993.09 최초 국내제작위성 우리별2호 발사
- 1999.05 최초 독자위성 **우리별3호** 발사
- 2003.09 최초 천문관측위성 **과학기술위성1호** 발사
- 2007.02 최초 국내발사위성 **과학기술위성2호** 개발완료
- 2007.05~ **과학기술위성3호** 개발중

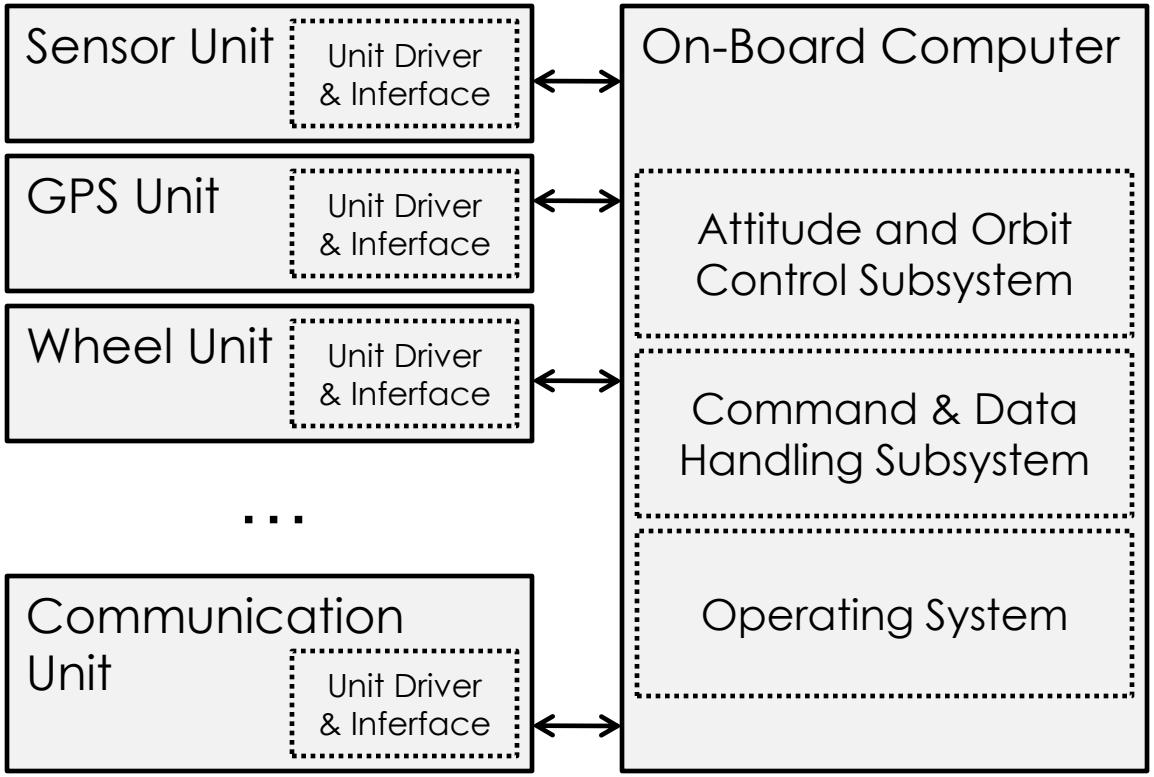
- http://satrec.kaist.ac.kr

검증 대상: 과학기술위성 2호/3호 탑재 SW



- 과학기술위성 2호 (사진)
  - 2007년 2월 개발완료
  - 2009년 7월 30일 발사예정
- 과학기술위성3호
  - 2007년 5월부터 개발

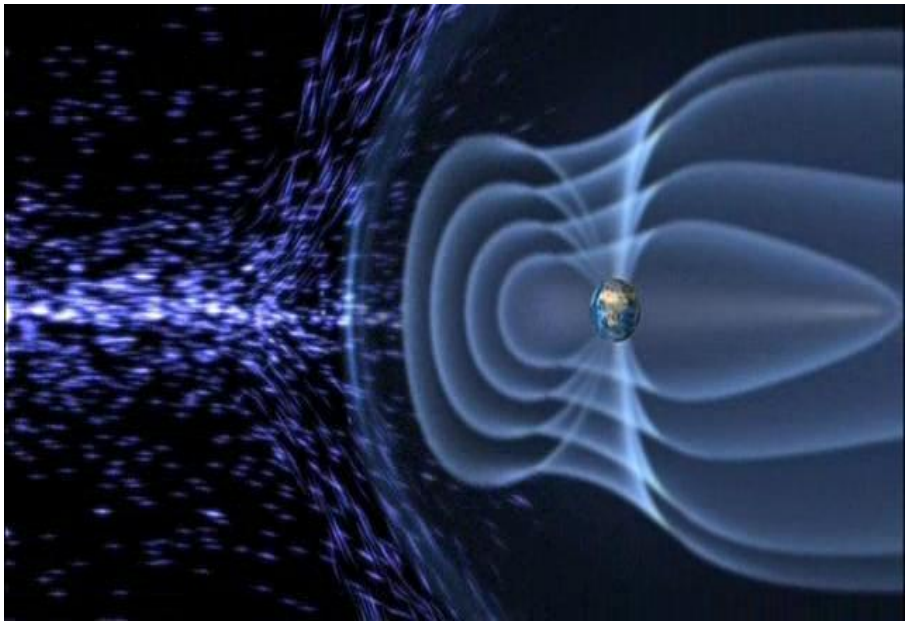
대략적인 인공위성 구조 (SW 관점에서)



인공위성 소프트웨어 특징 (과학기술위성2호)

- 프로그래밍 언어: C
- 프로그램 소스
  - 운영체제: 상용제품 구입하여 사용
  - 명령 및 데이터처리, 자세제어 모듈 자체 개발
- 크기
  - 명령 및 데이터처리 모듈의 경우 70,000 lines
  - 자세제어 모듈의 경우도 유사한 것으로 추정
  - 운영체제의 경우 10 배 정도로 추정
- 일반적인 작은 내장형 시스템 프로그램과 유사

차이점: 우주 방사선 (Cosmic Radiation)



- 인공위성 기체에 방사능 피폭을 당하면 기체 손상 가능
- 뿐만 아니라 "메모리(RAM)"의 값 변경 가능
- 고가의 복원 가능 하드웨어를 사용하여 1bit 변경 정도는 복구 가능

- 즉, 하드웨어의 지원이 있지만 완벽하지 않음
- 개발자 입장에서는 기체 손상 및 변수의 값 변경을 가정하고 프로그래밍

인공위성 SW 개발자들의 관심사

- 성능!
  - 제한된 환경에서 동작 가능한 SW
  - 작업량 조정 튜닝
- 오류 대처 능력
  - 하드웨어 오류, 사용자 실수, 방사능 피폭 등 다양한 오류에 대처
  - 예, EEPROM에서 값을 읽어 왔을 때 범위 검사
  - 예, 0, 1을 가질 수 있는 변수에 대해 0, 1, 그외의 값 모두 가정
- 의미 오류(semantic error)는 다소 무관심
  - 코드의 크기가 크지 않음
  - 코드의 복잡도 낮음: malloc/free 사용 금지, 포인터 제한 사용
  - 테스트

현재까지 유도한 인공위성 소프트웨어 검증 문제

- 일반적인 의미 오류
  - 배열 범위 초과 등
- 종료 검증
  - 루프가 항상 종료하는가?
- 작업량 조정 문제
  - 작업량이 시스템이 견딜 수 있도록 조정되어 있는가?

방사능 피폭과 종료 문제

- 임의의 루프에 대해 방사능 피폭을 고려하면 "항상" 종료한다고 보장 할 수 없음

- 다음 루프가 종료하는가?

```
for (i=0; i<10; i++) {
    i를 변경하지 않는 코드
    /* 피폭에 의해 i를 항상 0으로 변경 */
}
```

- 방사능 피폭에 대한 적절한 가정이 필요

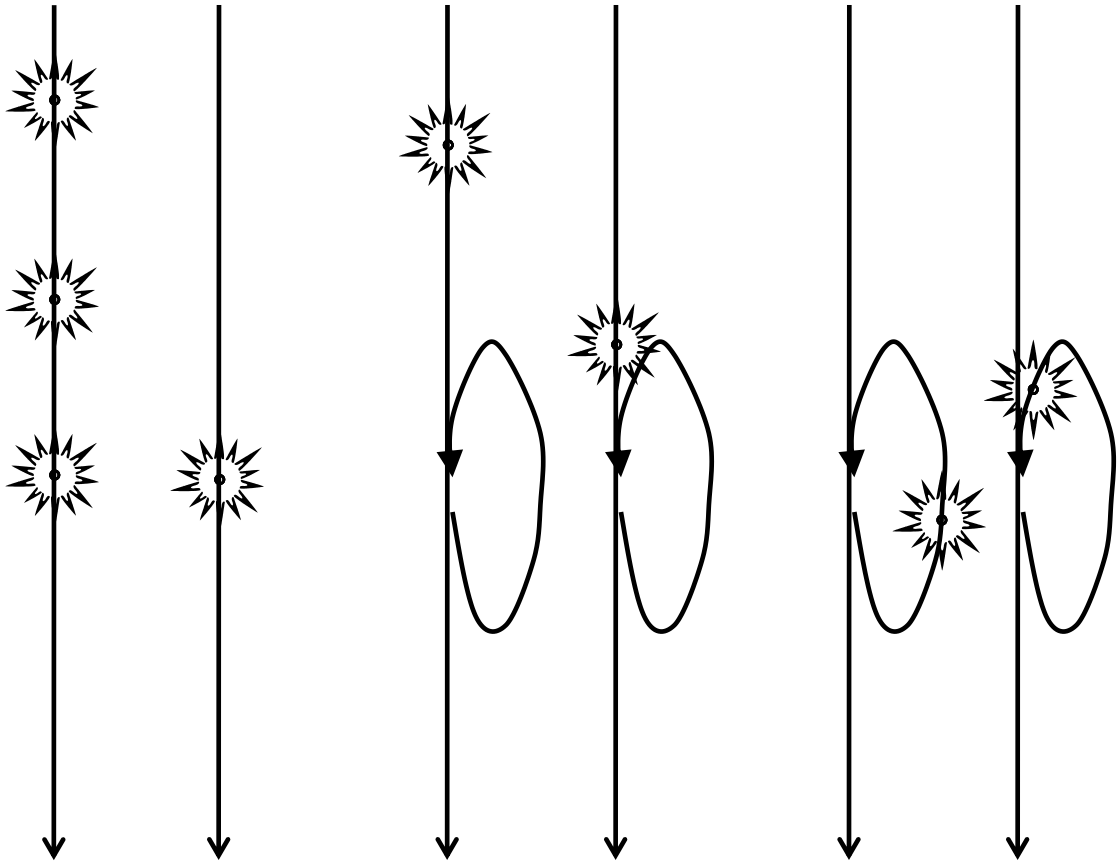
힌트: 좋은 루프와 나쁜 루프

```
for (i=0; i<10; i++) {
  …
}
```

```
for (i=0; i!=10; i++) {
  …
}
```

- 두 루프는 일반적으로 동일, 하지만 방사능 피폭을 고려하면 왼쪽은 좋은 루프, 오른쪽은 나쁜 루프이다.
  - 루프 내에서 피폭을 받아 i=100이 되었다고 가정해 보라.
- 프로그램의 임의의 지점에서 피폭을 받았을 때 남은 코드가 종료하는지를 검증하는 문제로 축소
  - 유한 루프가 유한 번 피폭 받았을 때 종료하는지 검증

피폭 위치 경우의 수 줄이기



- 피폭 = 임의의 상태로 변경
- 모든 루프에 대해 탈출 조건 검사하기 직전에 임의의 상태로 가정하고 이후에 루프가 종료하는지 검증하는 것으로 충분

인공위성 소프트웨어에서의 종료 검증

- 일반적인 종료 검증 분석을 적용
  - 단순한 for 루프는 패턴 분석으로 가능
  - 복잡한 루프에 대해, 최근 다수의 연구가 있었고 성숙한 단계
  - 각 루프에 대해 종료하는지 검증
  - 중첩 루프의 경우 내부 루프가 종료한다는 가정에서 검증

- 단, 루프의 탈출 조건 비교 지점 직전에 임의의 상태 가능하다고 가정하고 분석
  - 지역적 분석 가능

일반적인 종료 분석 [Cook *et al.* 2007, POPL]

- 기초가 되는 관계 분석 (relational analysis)이 있다고 가정
  - 예, octagon analysis
- 루프의 시작 부분 상태에 대해
  - 모든 변수 x에 대해 x' 추가
  - x=x'이라는 관계를 추가
- 추가된 상태에 대해 루프 몸체를 분석
- 분석된 상태가 종료를 보장하는지 검사

for(x=0; x<10; x++) skip;

시작 상태:

true

추가된 상태

x=x'

루프 몸체 분석 후 상태

x=x'+1

종료 보장?

yes ➔ termination!

작업량 조정 문제

- 인공위성 소프트웨어 개발자들이 많은 시간을 소요하는 부분이 작업량 조정을 통한 튜닝
- 시스템의 성능이 다소 낮고 예측하기 어렵기 때문에 현재 실제 테스트만을 통해서 튜닝
- 작업량 조정이 부적절할 때 발생하는 문제
  - 특정 작업이 다른 작업의 실행을 방해
  - 버퍼 넘침
  - 시스템 리셋

예: 타이머 인터럽트 처리 루틴

- 주기적으로 호출되어 실행되고, 너무 많은 작업은 하면 안됨
- 시스템이 주기적으로 실행해야 하는 주요 알고리즘 배치
  - 예, 방사능 피폭으로 인한 메모리 오염을 복원하기 위해 전체 메모리를 refresh 해 주는 루틴
- 한 번에 얼마나 많은 양을 해야 할까?
  - refresh는 자주 해 주면 좋음, 자주 안 해 주면 오염된 값으로 프로그램이 진행됨
  - 한 번에 너무 많이 해주면 작업량 증가
  - 테스트를 통해 결정

문제 정의 난해

- 정적 스케줄러(static scheduler)?
  - 운영체제의 스케줄링을 컴파일 타임에
  - No, 어떤 주기로 한 번에 얼만큼 해야할지 사용자가 결정
  - No, 작업량을 분배해야 하는 곳을 발견하는 것도 문제
- 작업량이 너무 많은 곳을 지적(hot spot finder)?
- 작업량 측정?
  - 시스템에 관한 정보가 풍부해야 정확한 측정
  - 복잡하지 않은 코드에서는 동적 측정이 더 정확

인공위성 소프트웨어 무결성 검증

- 종료 분석
  - 방사선 피폭에 대한 모델링 완료
  - 분석 방법론 도출
  - 프로토타입 구현률 90% 이상
  - 소스코드 조사 결과 소수이지만 복잡한 루프 발견

- 향후 과제
  - 추가 검증 문제 도출
  - 작업량 조정에 대한 실마리 도출