임베디드 시스템 테스트 — GuT 준비

Vincent Encontre

Rational Software 백서

TP 317, 11/01

Rational®
the software development company

목차

소개

이 문서는 임베디드 시스템 테스트에 대한 전반적인 내용을 소개하고 임베디드 시스템의 문제가 테스트 프로세스 및 기술에 미치는 영향과 Rational Test RealTime 이 이러한 문제에 대한 솔루션을 제공하는 방식에 대해 설명합니다.

공통 개념 정의

먼저 공통 개념에 대한 정의가 필요합니다.

테스트의 정의. 테스트란 응용프로그램 및 해당 컴포넌트의 동작, 성능 및 견고성이 예상 기준과 일치하는지 여부를 확인하는 정해진 프로세스입니다. 기본 표준 중 하나는 응용프로그램에 가능한 결함이 없어야 한다는 것입니다(일반적으로 명시적으로 나타나지는 않음). 따라서 예상 동작, 성능 및 견고성을 공식적으로 명시해야 하며 동시에 측정 가능한 것이어야 합니다. 디버깅은 문자 그대로 결함(버그)을 제거하는 것으로서, 테스트 프로세스의 일부로만 고려되어야 합니다.

"임베디드 시스템"의 정의. 임베디드 시스템은 정확하게 정의하기가 어렵고 논쟁의 여지가 많으므로 몇 가지 예제로 설명합니다. 임베디드 시스템은 주머니 속의 휴대 전화와 모든 관련 무선 하부 구조, 책상 위의 PDA, 전자 우편을 전달하는 인터넷 라우터, 대형 화면의 홈 시어터 시스템, 항공 교통 관제 시스템 및 이 시스템으로 모니터하는 지연 항공기와 같이 일상 생활과 깊은 관련이 있는 모든 "지능형" 장치에 사용됩니다. 이러한 장치 가치의 90%는 소프트웨어로 제공합니다.

그림 1 임베디드 소프트웨어로 움직이는 세계

모두 그런 것은 아니지만 대부분의 임베디드 시스템은 "실시간"으로 실행됩니다. 따라서 "실시간"이라는 용어와 "임베디드"라는 용어는 종종 같은 뜻으로 사용되기도 합니다. 실시간 시스템은 계산의 정확성이 로컬 시스템의 정확성뿐만 아니라 결과가 생성되는 시간에도 영향을 받는 시스템입니다. 즉, 시스템의 시간 제한조건이 충족되지 않으면 시스템 장애가 발생할 수 있습니다. 안전성이 중요한 일부 시스템의 경우 이러한 시스템 장애가 발생해서는 안됩니다. 따라서 ***임베디드 시스템에서는 시간 제한조건에 대한 테스트가 기능 동작에 대한 테스트만큼 중요합니다***.

임베디드 시스템의 테스트 프로세스가 여러 가지 다른 유형의 응용프로그램에 사용되는 테스트 프로세스와 유사하긴 하지만 특히 임베디드 시스템과 관련된 몇 가지 중요한 문제는 다음과 같습니다.

- 응용프로그램 개발 플랫폼과 실행 플랫폼의 구분
- 실행 프로그램과 상호 개발 환경의 다양성
- 다양한 배치 아키텍처
- 다양한 구현 패러다임의 공존
- 실행 플랫폼에 대한 한정된 자원 및 시간 제한조건
- 명확한 디자인 모델의 부족
- 새로운 품질 및 인증 표준

이 문제에 대한 자세한 내용은 이 백서 끝 부분에서 설명합니다.

이 문제는 임베디드 시스템의 테스트 및 측정에 많은 영향을 미칩니다. 이 문서는 또한 임베디드 시스템을 테스트하기 어려운 이유와 그에 따라 현재 개발 사례에서 테스트 부분이 가장 취약한 부분일 수 밖에 없는 이유에 대해 설명합니다. 따라서 최근 연구([R1] 참조)에 따르면 임베디드 시스템 개발 프로젝트의 50% 이상이 최초 일정보다 몇 달씩 지연되고 있으며, 전체 개발 노력의 50% 이상을 테스트에 매진한다고 하더라도 44%의 디자인만 예상 기능 및 성능의 20%를 달성할 수 있습니다.

이 백서의 나머지 부분에서 다루는 내용은 다음과 같습니다.

- 원하는 테스트 기술의 최소 세트를 도출하는 일반 테스트 통합 수행
- 실제 임베디드 환경과 같이 복잡한 시스템 테스트를 처리하기 위한 반복 인스턴스화 및 이를 고려하여 이상적인 테스트 기술에 성능 추가
- 임베디드 시스템의 개발 및 테스트가 어려운 이유 점검
- 이러한 문제를 테스트하는 데 사용되는 기술로 이행된 기능 목록에 해당 문제를 추가하는 방법 평가

Rational Test RealTime 은 이러한 이상적인 기술의 상당 부분을 구현하는 제품의 한 예로 사용됩니다.

일반 테스트 반복

GuT(Granule Under Test) 준비

모든 테스트에 있어 첫 번째 필수 단계는 테스트할 단위(granule)를 식별하는 것입니다. "단위(granule)"라는 단어는 컴포넌트, 단위(unit) 또는 시스템과 같이 해당 의미의 정의가 일반적이지 않은 다른 단어를 사용하지 않도록 하기 위해 사용됩니다. 예를 들어, UML 의 경우 컴포넌트는 고유한 제어 스레드(즉, 타스크 또는 UNIX 프로세스)를 갖는 응용프로그램입니다. 또한 "단위(granule)"라는 단어를 단일 코드 행에서 대규모 분산 시스템까지 테스트할 수 있는 넓은 범위의 요소에 대해 변환되는 세분성의 루트로 사용하려고 합니다.

테스트할 단위(granule)를 식별한 후 테스트 가능한 단위(granule) 또는 GuT(Granule under Test)로 변환합니다. 이 단계는 *스텁* 또는 경우에 따라 *어댑터*를 이용하여 단위(granule)를 독립적으로 만들어 환경과 분리시키는 작업으로 구성됩니다. 스텁은 단위(granule)와 나머지 응용프로그램 간의 양방향 액세스를 시뮬레이션하는 코드입니다.

그런 다음 테스트 드라이버를 빌드합니다. 이 드라이버는 적절한 입력으로 GuT 를 자극한 다음 출력 정보를 측정하여 예상 응답과 비교합니다. 어댑터는 테스트 드라이버로 GuT 를 자극할 수 있도록 하는 데 사용됩니다. 자극 및 측정 단계는 GuT 의 게이트를 통한 특정 경로를 따릅니다. 이 경로를 PCO(Points of Control and Observation)라고 하며 이 용어는 전기 통신 산업에서 실제로 사용되는 용어입니다. PCO 는 GuT 의 경계 또는 그 내부에 위치할 수 있습니다.

C 함수 단위(granule)의 PCO 예제는 다음과 같습니다.

- 단위(granule) 내부의 관찰 위치(PO): 함수 내 특정 코드 행의 범위
- 단위(granule) 경계의 관찰 위치(PO): 함수에서 리턴되는 매개변수값
- 단위(granule) 내부의 제어 위치(PO): 로컬 변수 변경
- 단위(granule) 경계의 제어 위치(PO): 실제 매개변수를 사용하는 함수 호출

스텁은 물론 테스트 드라이버도 응용프로그램의 기타 파트가 될 수 있으므로(사용 가능한 경우) C 함수 또는 C++ 클래스를 테스트하기 위해 반드시 개발되지 않아도 됩니다. GuT 는 응용프로그램의 다른 파트를 통해 액세스하거나 자극할 수 있으며 해당 파트가 스텁 또는 테스트 드라이버의 역할을 수행합니다. 스텁과 테스트 드라이버는 GuT 의 *테스트 하니스* 환경을 구성합니다.

테스트 케이스 설명

테스트 케이스를 설명하는 것은 다음 사항을 규정짓는 것과 같습니다.

- 적절한 PCO — 수행할 테스트 유형(예: 기능, 구조, 로드 등)에 따라 다릅니다.
- PCO 이용 방법 — PCO 를 통해 주고 받을 정보(있는 경우)와 해당 순서

테스트 유형과 송신 또는 예상 정보는 GuT 에 대한 요구사항 세트를 기반으로 합니다. 안전이 중요한 시스템의 경우 정확한 정규 요구사항이 개발 프로세스의 필수 요소입니다. 정규 요구사항은 테스트에 있어 중요한 동기부여 소스이지만 중요한 시스템 결점을 발견할 테스트를 명확하게 식별하지 못하는 경우도 있습니다. 정규 요구사항을 사용하는 경우와 특정 요구사항이 부족하고 그다지 중요하지 않은 응용프로그램의 경우, 테스터는 GuT 테스트를 위한 일부 요구사항을 도출할 적절한 테스트 세트("테스트 아이디어"라고도 함) 수행을 고려해야 합니다. 이러한 요구사항은 사용 가능한 PCO 를 이용하는 정규 테스트 케이스로 변환되어야 합니다. 여기서 "정규"라는 표현은 *실행 가능*한 상태를 나타납니다.

일반적으로 요구사항 자체는 정규사항이 아니므로 자연적으로 정규 테스트 케이스로 변환되지 않습니다. 이 변환 프로세스를 통해 테스트 케이스가 요구사항을 정확히 반영하지 않는 오류를 발견하는 경우가 종종 있습니다. UML 의 도입으로 스펙 언어는 보다 정규성을 나타내게 되었으며 이제 정규 요구사항 기반 테스트 케이스를 표현하여 예상치 못한 변환 위험을 막을 수 있게 되었습니다. Rational QualityArchitect 및 일부 Rational Test RealTime 은 이러한 모델 기반 테스트 기법을 사용하는 바람직한 예제를 제공합니다.

그러나 특히 임베디드 환경의 경우 UML 을 사용하여 모든 요구사항을 설명할 수 있는 것은 아닙니다. 테스트 케이스에 대해 가장 일반적인 정규 설명 기법은 C 또는 C++과 같은 프로그래밍 언어를 사용하는 것입니다. C 와 C++은 널리 알려져 있으므로 쉽게 학습할 수 있지만 PCO 정의 또는 예상 GuT 동작과 같은 테스트 케이스 요구를 고려하는 데 있어서는 적합하지 않습니다. 따라서 쉽게 이해할 수 있는 테스트 케이스를 작성할 수 없습니다. 이 문제점은 데이터 집약 또는 트랜잭션 기반 테스트와 같은 특정 테스트 도메인에 적합한 특정 상위 레벨 테스트 언어 디자인으로 해결되었습니다. Rational Test RealTime 은 기본 3GL 및 전용 상위 레벨 스크립팅 언어를 혼합함으로써 두 언어의 장점을 활용합니다. 즉, 학습 기간을 단축시키고 테스트 케이스를 효율적으로 작성할 수 있습니다.

테스트 케이스를 작성하기 위한 효과적이고 생산적인 또 다른 방법은 세션 레코더를 사용하는 것입니다. 수동으로 또는 향후 환경에서 GuT 를 촉진하면 특정 PO(Points of Observation)에서 GuT 내외부에 정보를 기록합니다. 해당 PO 는 나중에 재생될 적절한 테스트 케이스로 자동 변환됩니다. 이러한 테스트 레코더의 한 예는 Rational Rose RealTime 에서 찾을 수 있습니다. Rational Rose RealTime 에서는 모델 실행이 추적 실행을 반영하는 UML 시퀀스 다이어그램 생성으로 연결되며 이후 Rational QualityArchitect RealTime 에서 테스트 케이스 모델로 사용됩니다.

각 테스트 케이스에서는 GuT 가 테스트를 실행할 수 있는 특정 시작 상태로 변환되어야 합니다. 이러한 테스트 케이스 파트를 프리앰블이라고 합니다. 효과적인 테스트의 종료 시점에는 결과에 관계 없이 테스트 케이스 스크립트로 GuT 상태를 다음 테스트 케이스를 실행할 수 있는 안정적인 최종 상태로 변환시켜야 합니다. 이 테스트 케이스 파트를 포스트앰블이라고 합니다.

배치 및 실행 테스트

테스트 케이스를 설명한 후에는 테스트 드라이버 및 스텁의 정보 파트(작동 파트와 대비)로 변환 및 통합됩니다. 스텁 설명은 테스트 케이스의 핵심 파트입니다. 테스트 케이스는 테스트 하니스 실행 시 함께 실행됩니다.

테스트 결과 관찰

테스트 실행 결과는 PO(Points of Observation)를 통해 모니터됩니다.

단위(granule) 경계에서의 일반적인 PO(Points of Observation) 유형은 다음과 같습니다.

- *함수에서 리턴된 매개변수* 또는 수신 메시지
- *글로벌 변수 값*
- *정보의 순서 및 시간 지정*

단위(granule) 내부에서의 일반적인 PO(Points of Observation) 유형은 다음과 같습니다.

- *소스 코드 범위* – GuT 에서 포함된 소프트웨어 파트에 대한 세부사항 제공
- *제어 그래프* – 실행된 다양한 논리 분기 표시
- *정보 플로우* – 상이한 GuT 파트 간 시간에 대한 정보 교환을 시각적으로 표시. 일반적으로 이러한 유형의 플로우는 Rational Test RealTime 과 같이 UML 시퀀스 다이어그램으로 표시됩니다.
- *자원 사용법* – 다양한 GuT 파트에서 소요된 시간, 메모리 풀 관리 또는 이벤트 처리 성능과 같은 비기능적 정보 표시

이러한 모든 관찰 내용은 단일 테스트 케이스의 경우 수집되거나 테스트 케이스 세트의 경우 집계될 수 있습니다.

다음 단계 결정

모든 테스트 데이터를 수집하여 집계한 후의 결과는 하나 이상의 테스트 케이스가 실패하거나 모든 테스트가 패스되는 것입니다.

테스트 케이스의 실패 이유는 다음과 같습니다.

- *요구사항 비준수*(암묵적인 실패율 0% 요구사항 포함) — 구현 또는 디자인 단계로 돌아가 GuT 문제점을 수정해야 합니다.
- *테스트 케이스 오류* — 생각보다 자주 발생하는 문제입니다. 테스트는 소프트웨어와 마찬가지로 처음 사용할 때는 예상대로 실행되지 않는 경우도 있으므로 문제점을 해결하려면 테스트 케이스를 수정해야 합니다.
- *테스트 케이스 실행 불가능* — 역시 소프트웨어와 마찬가지로 오류 없이 완벽한 것처럼 보이지만 실제로는 테스트 하니스를 배치 또는 시작할 수 없거나 GuT 에 연결할 수 없습니다.

모든 테스트에 패스하면 다음과 같은 일련의 조치를 고려할 수 있습니다.

- *테스트 재평가* — 테스트 프로세스의 초기 단계에서는 테스트의 가치와 목적에 대한 의문을 제기할 수 있습니다. 특히 개발 노력의 초기 단계에서나 문제가 있는 경우 해당 문제점을 찾기 위한 테스트가 수행됩니다.…
- *테스트 케이스 수 증가* — GuT 의 신뢰성을 향상시킬 수 있어야 합니다. 신뢰성은 요구사항에 포함되므로 정확해야 한다는 반대 의견도 가능합니다. 그러나 신뢰성 레벨은 일반적으로 전체 테스트 케이스 세트에서 GuT 의 적용 범위 레벨과 직접 상관됩니다.

- *코드 적용 범위는 가장 널리 사용되는 적용 범위 유형으로서* —Rational Test RealTime 에서 구현되는 범위와 같습니다. 코드 적용 범위를 기반으로 하는 테스트는 적용 범위 레벨을 요구사항에서 합의된 레벨까지 향상시키는 추가 테스트 케이스를 정의하는 데 도움이 됩니다. 이 테스트 파트는 구조적 테스트라고도 합니다. —테스트 케이스는 GuT 의 컨텐츠를 기반으로 하며 해당 요구사항을 직접 기반으로 하지는 않습니다.
- *단위(granule) 집계를 통한 테스트 범위 증가* —이 일반 프로세스는 다음 단락의 설명대로 보다 많은 시스템 부분에 적용됩니다.

테스트를 중지해야 하는 경우

이 문제는 소프트웨어 종사자(practitioner)에게 끊임 없이 제기되는 문제이지만 이 문서의 목적과는 맞지 않습니다. 그러나 테스트할 시스템의 안전 심각도를 고려하는 한 가지 발견적 방법을 사용할 수 있습니다. 즉, 시스템의 안전성이 중요한지 여부를 고려하는 것입니다. 안전성이 중요하지 않은 시스템의 경우 출시 시간, 예산 및 "충분"한 상태와 같은 비교적 객관적인 기준에 따라 테스트를 중지할 수 있습니다. 그러나 인전성이 중요한 시스템의 경우 절대로 장애가 발생해서는 안되므로 같은 기준으로 테스트 중지 결정을 내릴 수 없습니다. 이 문서 끝 부분에 있는 *새로운 품질 및 인증 표준* 섹션에서는 이 문제를 처리하기 위한 몇 가지 권장 사항을 제공합니다.

일반 테스트 기술 요구사항

앞에서 설명한 일반 테스트 반복을 통해, 도구 테스트로 이행해야 하는 최소 기능 세트를 추론할 수 있습니다. 해당 기능은 다음과 같습니다.

- GuT 정의 및 분리 지원
- GuT 로 송신하거나 GuT 에서 예상되는 정보와 시작/종료에 대한 테스트 케이스 표기법 제공(PCO 에 대한 3GL, 비주얼 또는 상위 레벨 스크립팅, 지원 정의)
- 요구사항 또는 테스트 아이디어에서 정확한 테스트 케이스 도출 지원
- 세션 레코더를 사용하여 테스트 케이스를 구현하는 대체 방법 제공
- 테스트 케이스 배치 및 실행 지원
- 관찰 내용 보고
- 성공 또는 실패 평가

Rational Test RealTime 은 이러한 기능을 지원하며, 이러한 요구사항을 넘어 임베디드 시스템 분야에서 사용하는 복잡한 시스템의 테스트를 처리합니다.

복잡한 시스템 테스트를 위한 점진적인 6 단계

복잡한 시스템의 일반 아키텍처 및 구현

임베디드 시스템은 소형 8 비트 마이크로컨트롤러에서 멀티프로세서 플랫폼으로 구성되는 대규모 분산 시스템에 이르기까지 다양한 아키텍처로 구성될 수 있는 복잡한 시스템입니다. 그러나 이러한 시스템의 2/3 는 상용으로 구입하거나 내부에서 사용할 수 있는 실시간 운영 체제(RTOS)에서 실행되며 RTOS 타스크 또는 프로세스로 확장되는 스레드 개념을 구현합니다. 스레드는 독립적인 제어 플로우를 갖는 단위(granule)입니다. UML 에서는 이 개념을 컴포넌트라고 하며 노드는 RTOS 에서 관리하는 타스크 세트를 실행하는 독립 처리 단위를 나타냅니다. 노드 간 통신은 일반적으로 TCP/IP 와 같은 메시지 전달 프로토콜을 사용하여 수행됩니다.

임베디드 시스템의 개발자 중 상당수는 C, C++, Ada 또는 Java 를 프로그래밍 언어로 사용합니다. 즉, [R1]에 표시된 대로 2002 년 기준 70%는 C 를 사용하고 60%는 C++, 20%는 Java, 5%는 Ada 를 사용하게 됩니다. 일반적으로 하나의 임베디드 시스템에서 여러 언어를 사용할 수 있으며 특히 C 와 C++, C 와 Java 를 함께

사용하는 경우가 많습니다. C 는 보다 효율적이며 플랫폼의 세부사항과 유사한 것으로 알려져 있으며 Java 또는 C++는 객체 지향 개념에 의해 보다 생산적인 언어로 평가됩니다. 그러나 임베디드 시스템의 프로그래머는 오브젝트만 지향하지는 않습니다.

임베디드 시스템 관점에서의 단위(granule)는 다음과 같습니다. 이 목록의 정렬 기준은 점진적 복잡도입니다.

- C 함수 또는 Ada 프로시저
- C++ 또는 Java 클래스
- C 또는 Ada 모듈(세트)
- 클래스의 C++ 또는 Java 클러스터
- RTOS 타스크
- 노드
- 전체 시스템

가장 단순한 임베디드 시스템의 경우 전체 시스템은 C 모듈 세트로만 구성되며 RTOS 관련 코드는 통합되지 않습니다. 반대로 가장 복잡한 임베디드 시스템(분산 시스템)의 경우 네트워크 프로토콜로 인해 더 복잡해집니다.

다음 섹션은 이러한 일반 아키텍처가 다양한 테스트 단계에 미치는 영향에 대해 설명합니다.

점진적인 6 단계 테스트

단위(granule) 유형과 업계에서의 *일반적인* 용례(이 섹션 후반부에서 설명)에 따라, 응용프로그램의 동작, 성능 및 견고성이 예상 기준과 일치하는지 확인하기 위한 6 단계 테스트가 필요합니다. 해당 단계는 다음과 같습니다.

- 소프트웨어 유닛 테스트
- 소프트웨어 통합 테스트
- 소프트웨어 유효성 검증 테스트
- 시스템 유닛 테스트
- 시스템 통합 테스트
- 시스템 유효성 검증 테스트

소프트웨어 유닛 테스트

GuT 는 분리된 C 함수 또는 C++ 클래스입니다. 테스트 케이스는 GuT 의 목적에 따라 다음 요소로 구성될 수 있습니다.

- *데이터 집약 테스트* — 함수 매개변수값에 대한 다양한 데이터 변경 적용
- *시나리오 기반 테스트* — 요구사항의 가능한 모든 유스 케이스를 수행하기 위한 다양한 C++ 메소드 호출 시퀀스 연습

PO(Points of Observation)에는 값 매개변수, 오브젝트 특성 평가 및 소스 코드 적용 범위가 리턴됩니다. 화이트 박스 테스트는 유닛 테스트에 사용되므로 테스터는 GuT 컨텐츠에 대해 잘 알고 있어야 합니다. 유닛 테스트는 개발자의 책임입니다.

복잡한 임베디드 시스템의 경우 사소한 오류를 추적하기가 쉽지 않으므로 유닛 테스트 레벨에서 해당 오류를 찾아 제거하기 위한 모든 노력을 기울여야 합니다.

소프트웨어 통합 테스트

GuT 는 함수 세트 또는 클래스 클러스터입니다. 인터페이스의 유효성 검증은 통합 테스트에서 가장 중요한 단계입니다. 유닛 테스트(데이터 집약 기본 함수 호출 또는 메소드 호출 시퀀스)와 동일한 유형의 PC(Points of

Control)가 적용되는 반면 PO(Points of Observation)는 정보 플로우 다이어그램을 사용하는 하위 레벨 단위(granule) 간 상호작용에 초점을 맞춥니다.

GuT 가 의미를 갖게 되면, 즉 전체 테스트 시나리오를 GuT 에 적용할 수 있게 되면 아키텍처의 유효성을 잘 나타내야 하는 성능 테스트를 실행할 수 있어야 합니다. 기능 테스트는 빠를수록 좋습니다. 앞으로의 각 단계에는 성능 테스트가 포함됩니다. 화이트 박스 테스트 또한 이 단계에서 사용되는 방법입니다. 소프트웨어 통합 테스트는 개발자의 책임입니다.

소프트웨어 유효성 검증 테스트

GuT 는 컴포넌트 내부의 모든 사용자 코드입니다. 이 단계는 소프트웨어 통합의 최종 단계로 고려될 수 있습니다. 이제 유스 케이스는 구현 세부사항이 아닌 일반 사용자 시나리오에 근접합니다. GuT 는 전체 시스템의 중요 파트이므로 PO(Points of Observation)에는 자원 사용법 평가가 포함됩니다. 최종적으로 이 단계 역시 화이트 박스 테스트로 간주합니다. 소프트웨어 유효성 검증 테스트 또한 개발자의 책임입니다.

시스템 유닛 테스트

이제 GuT 는 전체 시스템 컴포넌트에 포함됩니다(예: 소프트웨어 유효성 검증 테스트 단계에서 테스트되는 사용자 코드, 모든 RTOS 및 플랫폼 관련 코드, 타스크 메커니즘, 통신, 인터럽트 등). 제어 위치(PO) 프로토콜은 더 이상 함수 호출 또는 메소드 호출이 아닌, 예를 들어, RTOS 메시지 대기열을 사용하여 주고 받은 메시지입니다.

메시지 전달 패러다임에서의 대칭은 이 단계에서 테스트 드라이버와 스텁 간의 구별을 관계없는 것으로 간주할 수 있음을 나타냅니다. 이를 *가상 테스터*라고 합니다. 각 테스터는 GuT 와 관련이 있는 다른 시스템 컴포넌트를 바꾸고 해당 기능을 수행할 수 있기 때문입니다. "시뮬레이터" 및 "테스터"는 "가상 테스터"의 동의어입니다. 가상 테스터 기술은 여러 RTOS 및 네트워크 프로토콜에 맞게 수정될 수 있어야 합니다. 이제부터 테스트 스크립트는 일반적으로 GuT 를 원하는 초기 상태로 변환시킨 후 메시지 샘플의 지정된 시퀀스를 생성하며, 메시지 컨텐츠를 예상 메시지와 비교하고 수신 날짜와 시간 제한조건을 비교하여 수신한 메시지의 유효성을 검증합니다. 테스트 스크립트는 다양한 가상 테스터에 대해 분산되고 배치됩니다. 시스템 자원은 임베디드 시스템 실행을 유지할 수 있는 시스템 기능을 평가하기 위해 모니터됩니다. 이 단계부터는 그레이 박스 테스트가 효과적인 테스트 방법입니다. GuT 인터페이스에 대한 정보만 필요합니다. 시스템 유닛 테스트는 조직에 따라 개발자 또는 담당 시스템 통합 팀의 책임입니다.

시스템 통합 테스트

GuT 는 단일 노드 내 컴포넌트 세트에서 모든 시스템 노드 및 분산 노드 세트까지 적용됩니다. PCO 는 RTOS 이벤트 및 네트워크 메시지와 같은 RTOS 및 네트워크 관련 통신 프로토콜이 혼합된 형태입니다. 가상 테스터는 컴포넌트 이외에도 노드 역할을 수행할 수 있습니다. 소프트웨어 통합의 중점사항은 다양한 인터페이스의 유효성을 검증하는 데 있습니다. 여기서는 그레이 박스 테스트가 효과적인 테스트 방법입니다. 시스템 통합 테스트는 시스템 통합 팀의 책임입니다.

시스템 유효성 검증 테스트

GuT 의 마지막 단계는 완벽한 임베디드 시스템 전체입니다. 이 최종 단계의 목표는 다음과 같습니다.

- *일반 사용자의 기능적 요구사항을 충족시킵니다*. 일반 사용자는 무선 네트워크의 장치(해당 임베디드 시스템이 인터넷 라우터인 경우) 또는 사용자(시스템이 이용자 장치인 경우)이거나 두 가지 모두(일반 사용자가 관리할 수 있는 인터넷 라우터)일 수 있습니다.
- *로드 및 견고성 테스트와 같은 비기능적 최종 테스트를 수행합니다.* 가상 테스터는 복사하여 로드를 시뮬레이션하고 시스템 장애를 일으키도록 프로그래밍할 수 있습니다.
- *다른 연결 장비와의 상호운용성을 확인합니다*. 해당 상호 연결 표준에 대한 준수 여부를 확인합니다.

이 목표에 대한 자세한 논의는 이 문서의 범위에 포함되지 않습니다. 이 단계에 적합한 테스트 방법은 블랙 박스 테스트 방법으로서, 테스터는 자주 사용하면서도 위험한 유스 케이스에 집중합니다.

단계 시퀀스 지정 방법 결정

점진적인 6 단계를 사용할 수 있으며 필요한 단계를 결정하려면 다음과 같은 다양한 기준이 적용됩니다.

- 모든 단계가 시스템에 적용되는지 여부
- 시스템의 모든 파트 또는 일부 파트에 대해서만 선택한 단계를 모두 수행해야 하는지 여부
- 선택한 단계를 선택한 파트에 적용해야 하는 순서

단계 결정은 이러한 기준을 기반으로 하며 현재 개발 중인 임베디드 시스템의 유형(안전성의 중요성 여부, 출시 시간, 배치 규모 등)에 따라 다릅니다. 이 선택 프로세스에 활용할 수 있는 가이드라인을 제공하는 문서는 향후 작성될 예정입니다. 문서가 작성되면 Rational 개발자 네트워크를 통해 액세스할 수 있습니다.

이 테스트 단계를 수행하는 다른 방법은 모든 단계를 단일 단계로 통합하는 것입니다. 유효성 검증 테스트는 대규모 GuT 에 대한 유닛 테스트로 간주할 수 있습니다. 유효성 검증 테스트 단계에서는 통합 여부도 확인할 수 있습니다. 문제는 GuT 에 액세스하는 방법, 삽입할 수 있는 PCO 유형과 해당 위치 및 테스트 케이스에서 원격으로 GuT 의 특정 부분을 대상으로 지정하는 방법입니다. 그러나 이 문서에서는 이 내용에 대해 다루지 않습니다.…

복잡한 시스템 테스트 기술에 대한 추가 요구사항

복잡한 임베디드 시스템 테스트 문제를 처리하려면 다음 기능에 테스트 기술을 추가해야 합니다.

- *다양한 PC(Points of Control) 유형 관리* — GuT 를 촉진하려면 함수 호출, 메소드 호출 및 메시지 전달 또는 RPC(Remote Procedure Call)를 사용하거나 Ada, C, C++ 또는 Java 와 같은 다른 언어 바인딩을 통해야 합니다.
- *다양한 PO(Points of Observation) 제공*(예: 매개변수 및 글로벌 변수 검수, 주장, 정보 및 제어 경로 추적, 코드 적용 범위 기록 또는 자원 사용법 모니터링). 이러한 PO(Points of Observation)는 각각 예상 평가 기능과 실제 평가 기능을 제공해야 합니다.

임베디드 시스템 문제가 테스트 프로세스 및 기술에 미치는 영향

이 섹션에서는 임베디드 시스템의 특정 문제에 대해 집중 설명하며, 해당 문제가 Rational Test RealTime 을 테스트 도구로 사용하여 테스트하는 데 사용되는 기술에 미치는 영향에 대해 평가합니다.

응용프로그램 개발 플랫폼과 실행 플랫폼의 분리

임베디드 시스템에 대한 여러 정의 중 하나는 다음과 같습니다.

> *임베디드 시스템은 응용프로그램을 배치하고 대상으로 지정하는 플랫폼과 다른 플랫폼에 디자인해야 하는 소프트웨어 시스템입니다.*

개발 측면에서 볼 때 플랫폼은 일반적으로 Windows, Solaris 또는 HP-UX 와 같은 운영 체제를 의미합니다. 임베디드 시스템 분야에 있어 UNIX 및 Linux 사용자의 백분율이 다른 사용자와 비교할 때 더 높으며([R1]의 경우 40%) IT 시스템 분야에서는 더 높습니다. 대상 플랫폼의 경우 앞에서 언급한 모든 장치가 포함됩니다.

제한조건이 필요한 이유. 대상 플랫폼은 일반 사용자(실제 사용자 또는 다른 장치 세트)에 대해서만 최적화되고 사용자 조정되므로, 개발 시 수행해야 할 필수 컴포넌트(예: 키보드, 네트워크, 디스크 등)는 포함하지 않습니다.

이러한 이중 플랫폼 문제를 해결하려면 테스트 도구가 가장 투명하면서도 효율적인 방식으로 개발 플랫폼에서 실행 플랫폼에 액세스할 수 있는 권한을 제공해야 합니다. 실제로 이러한 액세스 방법의 복잡성은 사용자에게 공개되지 않아야 합니다. 액세스 권한에는 테스트 케이스 정보 다운로드, 테스트 실행 원격 모니터링(시작,

동기화, 중지) 및 테스트 결과와 관찰 결과 업로드가 포함됩니다. Rational Test RealTime 에서는 대상 배치 기술로 모든 대상 플랫폼 액세스가 제어됩니다.

또한 Rational Test RealTime 은 Windows, Solaris, HP 및 Linux 에서 사용할 수 있습니다. 이러한 플랫폼은 장치, 임베디드 시스템 및 하부 구조 업계의 선두 기업이 사용하는 개발 플랫폼입니다.

실행 플랫폼 및 상호 개발 환경의 다양성과 확대

실행 플랫폼의 범위는 소형 8 비트 마이크로컨트롤러 기반 보드에서 대규모 분산 네트워크 시스템까지 방대합니다. 다양한 칩 및 시스템 벤더가 존재하므로 모든 플랫폼에는 응용프로그램 개발을 위해 다른 도구가 필요합니다. 또한 동일한 임베디드 시스템에서 여러 플랫폼을 사용하는 경우도 증가하고 있습니다.

일반적으로 이러한 유형의 개발 환경을 상호 개발 환경이라고 합니다. 실행 플랫폼이 다양하다는 것은 컴파일러, 링커, 로더 및 디버거와 같은 개발 도구 또한 그만큼 다양함을 의미합니다.

이러한 경우 우선 PO(Points of Observation) 기술은 소스 코드만을 기반으로 할 수 있습니다. Rational PurifyPlus 제품군에서 사용하며 몇몇 기본 컴파일러 세트에만 사용할 수 있는 오브젝트 코드 삽입 기술과 달리 Rational Test RealTime 은 소스 코드 인스트루먼테이션을 사용하여 숫자 인수를 처리합니다. 이 기술을 10 년간 적용해본 결과 매우 효율적인 코드 인스트루먼테이션으로 밝혀졌습니다.

이러한 다양성에 따른 또 다른 직접적인 결과는 상호 개발 도구의 벤더가 복잡성을 숨기고 개발자의 편의를 도울 수 있는 통합 개발 환경(IDE)을 제공하는 경향을 나타낸다는 것입니다. 모든 추가 도구는 해당 IDE 에 밀접하게 통합되어야 합니다. 예를 들어, Rational Test RealTime 은 WindRiver 의 Tornado 또는 GreenHills 의 Multi IDE 에 통합됩니다.

또 다른 특징은 동적 컴퓨터 칩 산업에서 두드러진 현상으로서, 새 실행 플랫폼 및 연관 개발 도구가 매우 자주 릴리스된다는 것입니다. 이러한 경우 테스트 기술은 레코드 시간에 이 새 아키텍처에 적용될 수 있는 고도의 유연성을 갖추어야 합니다. 일반적으로 새 대상 플랫폼에 대한 Rational Test RealTime 대상 배치가 수행되며 해당 간격은 보통 2 일 이내에서 일주일 미만입니다.

실행 플랫폼에 대한 한정된 자원 및 시간 제한조건

임베디드 시스템은 그 정의에서 알 수 있듯이 실행 대상 응용프로그램에 대한 자원이 한정적입니다. 이는 특히 사용 가능한 RAM 이 1KB 미만인 소형 플랫폼에서 두드러집니다. 이러한 경우 개발 환경에 대한 연결은 JTAG 프로브, 에뮬레이터 또는 직렬 링크를 사용하는 경우 또는 마이크로프로세서의 속도가 작업을 올바르게 처리할 수 있는 경우에만 설정할 수 있습니다. 테스트 도구는 어려운 절충안을 선택해야 합니다. 즉, 테스트 데이터를 개발 환경에 두고 성능에 큰 영향을 미치더라도 연결 링크를 통해 해당 데이터를 송신할지 또는 대상 플랫폼의 테스트 드라이버로 테스트 데이터를 해석할지 여부를 결정해야 합니다.

Rational Test RealTime 에서 사용하는 기술은 테스트 하니스를 대상 시스템에 임베드하는 것입니다. 이 기술은 이전에 변환된 테스트 데이터를 대상 하니스의 응용프로그램 프로그래밍 언어(C, C++ 또는 Ada)로 컴파일하고 사용 가능한 상호 컴파일러를 사용하여 이 테스트 하니스 오브젝트 파일을 나머지 응용프로그램에 링크합니다. 이 빌드 체인은 make 파일에서 Rational Test RealTime 명령행 인터페이스를 사용하여 사용자가 볼 수 있습니다. 최적화된 코드 생성, 스마트 링크 맵 할당 및 낮은 메모리 풋프린트 기능이 함께 대상 플랫폼의 테스트 하니스에서 필요한 자원을 최소화하며 다음과 같은 이점을 제공합니다.

- 시간이 정확해지고 대상 내 위치로 성능에 대한 실시간 영향이 줄어듭니다.
- 테스트 케이스 실행 중에 연결 링크로 데이터 정보를 순환시키지 않아 호스트와 대상 간의 통신이 최소화됩니다. 필요한 경우, 관찰 데이터는 RAM 에 저장되며, 테스트 케이스가 정확히 완료될 때 디버거 또는 에뮬레이터를 이용하여 업로드됩니다.

상호 개발 환경이 점점 더 사용하기에 편리해지고 있기는 하지만 현재 제공되는 임베디드 시스템의 상당 부분은 개발 및 테스트 작업이 어려워 작업자에게 어려움을 주고 있습니다. Rational Test RealTime 의 목적은 이처럼 임베디드 시스템 개발자의 어려운 작업 과정을 단순화시키는 데 있습니다.

명확한 비주얼 모델 디자인의 부족

임베디드 시스템 개발자의 공통적인 특징은 코드를 즐겨 사용한다는 것입니다. 그러나 일반 개발자의 경우 비주얼 모델링에 익숙하지 않아 코드 작업을 "실제 작업"이라고 생각하는 경향이 있습니다. 비주얼 모델링 및 UML 과 같은 언어 또한 임베디드 정보를 디자인으로 통합하는 데 많은 발전을 이루어왔지만 다수의 임베디드 시스템 프로그래머는 여전히 이전의 일반 프로그래밍 언어를 사용하여 대부분의 작업을 수행하고 있습니다. 또한 많은 사용자는 Java 를 디자인에 사용하지 않고 있습니다.

Rational Test RealTime 은 개발자가 자신이 원하는 방식으로 작업을 수행할 수 있도록 하기 위해 테스트 템플리트 생성기 및 API 랩퍼와 같은 마법사를 제공함으로써 응용프로그램의 소스 코드를 기반으로 테스트 케이스를 디자인할 수 있도록 지원하는 데 중점을 두고 있습니다. 테스트가 응용프로그램에만 적용될 수 있도록 하는 것은 이러한 코드 기반 테스트 빌딩 프로세스의 주요 이점입니다. 단점은 테스트 케이스가 확인해야 하는 해당 요구사항을 반영하는 것으로 보장할 수 없다는 것입니다. 확실한 것은 비주얼 모델링을 광범위하게 채택함으로써 요구사항과 테스트 케이스 간 차이를 줄일 수 있다는 것입니다. Rational Test RealTime 은 또한 테스트 케이스 컴파일러에 Rational Rose RealTime UML 시퀀스 다이어그램을 제공함으로써 이 기술을 활용합니다.

새로운 품질 및 인증 표준

특정 임베디드 시스템 카테고리의 경우 안전성이 중요한 시스템에는 절대로 장애가 발생해서는 안됩니다. 이러한 시스템은 핵, 의료 및 항공 전자 공학 산업에서 그 예를 찾을 수 있습니다. 항공 산업 및 정부 기관(예: 미 연방 항공국)에서는 오래 전 실패율 0%의 목표를 달성하기 위해 *공수 시스템 및 장비 인증을 위한 소프트웨어 고려사항*(RTCA DO-178B 표준으로 참조)을 규정하는 데 합의했습니다([R2 참조]). 이 표준은 전세계 항공 전자 공학 산업에서 안전성이 중요한 소프트웨어 개발의 주요 표준입니다. 이 표준은 가장 엄격한 소프트웨어 개발 표준의 하나로서 자동차, 의료(FDA 에서는 이미 DO-178B 와 유사한 표준 릴리스), 방위 또는 교통 산업 분야와 같이 인간의 생명이 중요한 시스템을 제조하는 다른 분야에서도 부분적으로 채택하고 있습니다.

DO-178B 에서는 시스템 기능을 방해하는 비정상적인 소프트웨어 동작과 관련된 심각도 레벨로 소프트웨어를 분류합니다. 가장 높은 심각도는 레벨-A 장비로서 장애 발생 시 전체 시스템에 치명적인 영향을 미치게 됩니다. DO-178B 에는 특히 테스트 영역에서 레벨-A 장비의 안전성을 보장하는 정밀한 단계가 포함됩니다. Rational Test RealTime 은 레벨-A 장비를 포함하여 모든 레벨에 대한 필수 DO-178B 테스트 환경을 모두 충족시킵니다.

요약

이 문서에서는 임베디드 시스템을 테스트하는 데 사용되는 점진적인 6 단계 프로세스의 개요에 대해 설명합니다. 이 프로세스(매우 작은 규모에서 대규모의 시스템에 이르는 전체 범위를 고려) 및 임베디드 시스템의 특정 특성과 제한조건을 고려해서 임베디드 시스템의 테스트를 위한 이상적인 기술에 필요한 요구사항 세트를 추론했습니다. 임베디드 시스템 분야에 대한 새로운 Rational 제품인 Rational Test RealTime 은 이 이상적 기술의 상당 부분을 나타냅니다.

이 문서는 임베디드 시스템 테스트에 대한 일반 개요 정보를 제공했으며 앞으로 이 문서에서 논의한 여러 주제에 대한 개별 문서가 추가 작성될 예정입니다. ☒

용어

작성자: Paul Szymkowiak

소프트웨어 엔지니어링 분야에서 사용되는 많은 용어에 대한 정의가 통일되어 있지 않은 것이 현실입니다. 이 문서에서 사용한 용어는 실시간 임베디드 시스템 분야에 종사하는 사람에게는 친숙한 용어입니다. 그러나 소프트웨어 테스트에서는 다른 유사 용어와 관련 용어가 사용되므로 두 분야의 용어를 맵핑시켜 다음 표에 정리했습니다.

이 문서에서 사용되는 용어	유사/관련 용어
GuT(Granule under Test): 테스트 목적으로 해당 환경에서 분리된 시스템 요소	*관련 용어*: 테스트 항목, 대상, 테스트 대상
PCO(Point of Control and Observation): 테스트 환경에서 관찰한 내용을 기록하거나 테스트의 제어 흐름과 관련된 결정을 내리는 특정 테스트 시점	가장 유사한 용어: **검증 포인트**
종료: 시스템을 다음 테스트를 실행하는 데 필요한 안정된 특정 종료 상태로 만들기 위해 수행되는 조치입니다.	사후 조건, 재설정
시작: 시스템을 테스트를 실행하는 데 필요한 안정된 특정 시작 상태로 만들기 위해 수행되는 조치입니다.	전제 조건, 설정
테스트 하니스: 일련의 관련 테스트를 실행하기 위해 결합된 하나 이상의 테스트 드라이버, 테스트 특정 스텁 및 인스트루먼테이션의 배열	*관련 용어*: 테스트 스위트 또는 테스트 드라이버, 테스트 스텁
가상 테스터: (1) 테스트 시 단위(granule)와 상호작용하는 GuT 외부 요소 (2) 일반적으로 사용자의 조치를 나타내는 실행 테스트 인스턴스	*관련 용어*: 테스트 스크립트 또는 테스트 드라이버, 테스트 스텁, 시뮬레이터, 테스터

참조

[R1] *"Critical Issues Confronting Embedded Solutions Vendors", Electronics Market Forecast*, http://www.electronic-forecast.com/, April 2001.

[R2] DO-178B, *Software Considerations in Airborne Systems and Equipment Certification*, Radio Technical Commission for Aeronautics — RTCA, http://www.rtca.org/, January 1992.

작성자 소개

Vincent Encontre 는 프랑스 Toulouse 에 있는 Rational 의 새 엔지니어링 센터에 본부를 둔 Embedded and RealTime, Automated Testing Business Unit 의 책임자입니다. Vincent 는 출장, 회의 참석 또는 수많은 전자 우편에 답하는 시간 이외에는 Rational 제품이자 차세대 Rational 제품에 재사용가능한 기술로서의 Rational Test RealTime 가능성에 대해 연구하고 있습니다. Vincent 는 또한 임베디드 모델링 및 테스트 기술과 우수 사례에 대한 풍부한 경험을 갖고 있습니다. Rational 및 ATTOL 이전에 Vincent 는 Philips 와 Verilog 에서 13 년간 근무하며 소프트웨어 엔지니어링 도구의 지원 ,디자인 및 마케팅 경험을 쌓았고 이들 도구가 소프트웨어를 보다 신속하게 빌드하는 데 도움이 된다고 믿고 있습니다.

여가 시간에는 주로 세 명의 아들 및 주위 사람들과 축구를 하며 프랑스 남부 지방에서 생활의 멋을 즐기며 생활하고 있습니다.

Rational®
the software development company

본사 안내:

Rational Software
18880 Homestead Road
Cupertino, CA 95014
전화번호: (408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
전화번호: (781) 676-2400

수신자 부담 전화번호: (800) 728-1212
전자 우편: info@rational.com
웹: www.rational.com
전세계 지사 안내: www.rational.com/worldwide