로봇 소프트웨어 컴포넌트를 위한 시뮬레이션 기반 인터페이스 테스팅 자동화 도구

SITAT: Simulation-based Interface Testing Automation Tool for Robot Software Component

강 정 석, 최 형 섭, 맹 상 우, 김 시 완, 박 홍 성*
(Jeong-Seok Kang, Hyeong-Seob Choi, Sang-Woo Maeng, Si-Wan Kim, and Hong-Seong Park)

Abstract: Robot software components can be categorized into two types; the pure S/W component and the H/W-related one. Since interface testing of the robot software component is the labour-intensive and complicated work, an effective automated testing tool is necessary. Especially it is difficult to test all types of H/W-related components because it is hard work to prepare all H/W modules related to them. This paper proposes a new simulation-based interface testing automation tool(SITAT) which generates automatically test cases for interface testing of the robot software component and executes the interface test with the generated test cases where the simulator is used for simulation of the activity of a H/W module instead of the real H/W module. This paper verifies the effectiveness of the suggested SITAT with testing of the real H/W-related robot software component.

Keywords: software test, testing automation, robot software component interface test

I. 서론

로봇 소프트웨어 컴포넌트(robot software component)는 순수 S/W로 이루어진 컴포넌트와 H/W와 연동된 컴포넌트로 나눌 수 있고, 이러한 컴포넌트를 통하여 로봇을 제어할 수 있기 때문에 컴포넌트의 신뢰성에 대한 검증이 중요하다. 가장 널리 이용하고 있는 소프트웨어 컴포넌트 검증 방법에는 소프트웨어 테스트 방법이 있으며, 소프트웨어 테스트(software test)는 개발에 소요되는 총 비용의 50% 이상과 총 기간의 50% 정도가 할당될 정도로 중요한 작업 중의 하나이다[1]. 특히 소프트웨어 컴포넌트 테스트는 소스 코드(source code)가 존재하지 않을 경우에 컴포넌트 인터페이스 테스트와 컴포넌트간 연동 테스트로 나눌 수 있다. 본 논문에서는 로봇 소프트웨어 컴포넌트에 대한 인터페이스의 기능 및 동작성 테스트에 초점을 맞춘다.

로봇 소프트웨어 컴포넌트의 인터페이스 테스트(interface test)는 기존 소프트웨어의 단위 테스팅(unit testing)에 비유될 수 있으며 노동 집약적이고 반복적인 복잡한 작업이므로 효율적으로 테스팅을 수행 할 수 있는 테스팅 자동화 도구는 필수적이다. 이러한 테스팅 자동화 도구는 테스트 프로세스(test process)의 지원 범위에 따라 다음과 같은 기능을 제공하여야 한다.

첫째, 로봇 소프트웨어 컴포넌트의 인터페이스에 대한 테스트 케이스를 자동으로 생성할 수 있어야 한다. 컴포넌트의 소스 코드가 존재하지 않을 경우 컴포넌트가 제공하는 인터페이스 정보만을 이용하여 테스트 케이스(test case)를 생성 할 수 있어야 하며 테스트 케이스의 수가 많아질 경우 이를 줄일 수 있는 방법을 제공하여야 한다.

둘째, 테스트 케이스에 따라 테스트를 자동으로 수행하며 테스트 수행 결과를 생성할 수 있어야 한다. 이를 위하여 테스트 수행시 필요한 테스트 드라이버(test driver)와 테스트 스텁 컴포넌트(test stub component)를 생성할 수 있어야 하며, 로봇 소프트웨어 컴포넌트의 특성상 로봇 하드웨어와 연동하여 테스트 대상 컴포넌트가 동작할 경우 물리적인 로봇 하드웨어를 대신하여 동작을 시뮬레이션(simulation) 할 수 있는 기능을 제공하여야 한다.

기존의 소프트웨어 컴포넌트 테스팅 자동화에 대한 연구는 [2-7]이 있다. [2-4]는 소스 코드 정보를 이용하여 테스트 케이스를 생성하고 수행하며, [5,6]은 컴포넌트 자체에 테스트 케이스와 테스트 코드를 삽입하여 테스트를 수행하는 방법을 제안하였다. 그리고 [7]은 테스트 명세와 컴포넌트 구현 사이의 의존 관계를 분리 시켜 테스트의 재 사용성을 증가시키고 테스트를 자동으로 수행하는 방법을 제안하였다. 하지만 앞서 언급한 바와 같은 두 가지의 기능을 모두 만족하는 테스팅 자동화 도구는 없다.

따라서 본 논문은 로봇 소프트웨어 컴포넌트의 인터페이스 테스트에 대해 테스트 케이스를 자동으로 생성하고 로봇 하드웨어를 대신하여 동작을 시뮬레이션 할 수 있는 로봇 H/W 시뮬레이터(robot H/W simulator)와 연동하여 생성된 테스트 케이스에 따라 테스트를 자동으로 수행하는 시뮬레이션 기반 인터페이스 테스팅 자동화 도구(이후 SITAT으로 칭함)를 제안한다. 제안하는 테스팅 자동화 도구는 로봇 소프트웨어 컴포넌트 인터페이스의 입력 파라미터 및 로봇 하드웨어 관련 파라미터 정보를 이용하여 테스트 케이스를 자동으로 생성한다. 또한 생성된 테스트 케이스에 따라 테스트를 자동으로 수행하는 테스트 드라이버 컴포넌

* 책임저자(Corresponding Author)
논문접수: 2010. 2. 1., 수정: 2010. 3. 17., 채택확정: 2010. 4. 20.
강정석, 최형섭, 맹상우, 김시완, 박홍성: 강원대학교 전자통신공학과
(jskang1982@gmail.com/gazam@naver.com/mswok@control.kangwon.ac.kr/malshan@control.kangwon.ac.kr/hspark@kangwon.ac.kr)

트(test driver component), 테스트 대상 컴포넌트의 요구 인터페이스(required interface)에 대한 테스트 스텁 컴포넌트(test stub component), 그리고 테스트 수행시 로봇 h/w 시뮬레이터를 제어하기 위한 시뮬레이션 제어 컴포넌트(simulation control component)의 소스 코드를 생성하여 테스트를 수행한다.

본 논문은 다음과 같이 구성된다. 제 II 장에서는 기존의 소프트웨어 컴포넌트 테스팅 자동화 연구에 대해 기술하며, 제 III 장에서는 제안하는 컴포넌트 테스팅 자동화 도구의 구조 및 기능에 대해 기술한다. 제 IV 장에서는 제안하는 테스팅 자동화 도구를 이용하여 로봇 소프트웨어 컴포넌트를 테스트 하기 위한 실험 과정 및 결과를 기술하며, 마지막으로 V 장에서 결론을 맺도록 한다.

II. 관련 연구

소프트웨어 컴포넌트 테스팅 자동화 도구는 테스트를 위해 필요한 정보에 따라 소스 코드 기반[2-4], BIT (Buillt-in Test) 기반[5,6], 그리고 컴포넌트 사용자 명세 기반[7]으로 나누어 질 수 있으며, 각 정보를 이용하여 테스트 케이스를 자동으로 생성하고 생성된 테스트 케이스에 따라 테스트를 자동으로 수행하는 기능을 제공한다. 특히 테스트 케이스 자동 생성 방법은 크게 블랙박스 테스트(Black-box test)와 화이트박스 테스트(white-box test) 기법으로 나누어진다. 전자는 동치분할[8,9], 경계 값 분석[8,9], 페어와이즈(pair-wise) 기법[12,13] 등이 있으며, 후자는 기호 실행[2,10], 데이터 흐름 분석[2,10], 제어 흐름 분석[10] 등이 있다.

소스 코드 기반 테스팅 자동화 도구[2-4]는 소스 코드가 존재할 때 소스 코드 정보를 이용하여 테스트 케이스를 생성하거나 테스트를 수행하기 위한 테스트 드라이버 코드를 생성하는 도구로서 소스 코드 상의 데이터 흐름 또는 제어 흐름 정보를 분석하고 기호 실행 기법 등을 이용하여 테스트 케이스를 자동으로 생성하며[2,3], 테스트 케이스를 자동으로 실행하기 위해 소스 코드 정보를 이용하여 테스트 드라이버 코드를 생성한다[4]. 하지만 이 방법들은 컴포넌트의 소스 코드가 존재 하지 않을 시 테스트를 수행할 수 없는 단점을 가지고 있다.

BIT 기반 테스팅 자동화 도구[5,6]는 컴포넌트의 테스트 가능성을 높이기 위해 테스트 케이스 및 테스트 드라이버 코드를 컴포넌트 안에 미리 내장하여 테스트를 수행하는 방법이다. BIT 컴포넌트의 구조를 정의하고 정의된 인터페이스를 통해 자동으로 테스트를 수행[5]하거나 컴포넌트 내부의 오류를 찾아내기 위해 BIT 래퍼(wrapper)와 테스트 케이스 및 테스트 드라이버를 자동으로 생성하는 방법[6]을 제공한다. 이 방법들은 컴포넌트의 테스트 가능성을 높일 수 있다는 장점이 있지만 테스트 케이스 및 코드가 컴포넌트 자체에 추가됨에 따라 메모리 사용량나 컴포넌트의 소스 코드 크기가 커질 수 있으며 컴포넌트 개발자가 테스트 정보를 제공하지 않으면 테스트를 수행할 수 없는 단점을 가지고 있다.

컴포넌트 사용자 명세 기반 테스팅[7]은 컴포넌트 개발자에 의해 제공되는 정보가 아닌 사용자 입장에서 컴포넌트를 테스트 하기 위한 방법으로 Spy 클래스 및 XML 어댑터(adapter) 모듈을 이용하여 테스트 명세와 컴포넌트 구현 사이의 의존관계를 분리 시켜 테스트의 재 사용성을 증가시키고 테스트를 자동으로 수행하기 위한 방법을 제공한다. 하지만 테스트 수행에 필요한 테스트 케이스를 자동으로 생성하기 위한 방법은 제공하고 있지 않으며 Java와 같은 언어에만 테스트 수행이 가능하다는 단점이 있다.

그리고 로봇 소프트웨어 컴포넌트는 일반적인 소프트웨어 컴포넌트와 다르게 로봇 하드웨어와 상호 연동하여 동작을 한다. 이러한 로봇 소프트웨어 컴포넌트를 테스트하기 위해 로봇 하드웨어를 직접 소프트웨어 컴포넌트에 연결하여 테스트 하는 경우 로봇 하드웨어를 개발해야 하는 번거로움이 있고, 로봇 하드웨어를 대신하여 테스트 스텁 모듈을 이용하는 경우 대상 하드웨어의 물리적 동작 및 기능을 구현하는데 많은 시간을 소비할 수 있다. 기존 연구에서는 이러한 문제를 해결하기 위한 방법을 제공하고 있지 않다.

본 논문에서는 이러한 문제를 해결하기 위해 로봇 H/W 시뮬레이터를 사용하여 로봇 하드웨어의 동작을 시뮬레이션하고 이 정보를 이용하여 테스트를 자동으로 수행하는 방법 및 테스트 수행시 필요한 테스트 케이스 및 코드를 자동으로 생성하기 위한 방법을 제안한다. 특히 테스트 케이스 자동 생성은 블랙박스 테스트 기법을 활용하였다.

III. SITAT의 구조 및 기능

1. SITAT의 구조 및 동작순서

제안하는 테스팅 자동화 도구는 그림 1에서 보는 바와 같이 다양한 테스트 환경에서 테스트를 수행하고자 웹 기반 테스팅 자동화 엔진 서버(web-based automatic testing engine server), 테스트 자동 빌드 에이전트(automatic test build agent), 그리고 로봇 H/W 시뮬레이터(robot H/W simulator)로 구성되며, 테스트 케이스(test case)와 테스트 애플리케이션(test application)을 자동으로 생성하고 로봇 H/W 시뮬레이터와 상호 연동하여 생성된 테스트 케이스에 따라 테스트를 자동으로 수행한다.

1.1 웹 기반 테스팅 자동화 엔진 서버

웹 기반 테스팅 자동화 엔진 서버는 SITAT의 핵심 요소로서 로봇 소프트웨어 컴포넌트의 인터페이스 테스트에 대한 테스트 케이스를 생성하고, 테스트 수행에 필요한 테스트 드라이버 컴포넌트, 테스트 스텁 컴포넌트, 그리고 시뮬레이션 제어 컴포넌트를 생성하여 서로 연결하는 역할하며 다음과 같이 구성된다.

인터페이스 테스트 케이스 생성기(interface test case generator)는 사용자로부터 테스트 대상 컴포넌트의 인터페이스 표현 정보(IDL 또는 XML 등) 및 테스트 명세 정보를 입력 받아 테스트 케이스를 자동으로 생성하는 모듈이다. 생성된 테스트 케이스는 XML 형식으로 파일에 저장되며 사용자는 웹 인터페이스를 통해 테스트 케이스를 수정할 수 있고 각 테스트 케이스에 대한 예상 결과 값을 입력할 수 있다.

테스트 애플리케이션 생성기(test application generator)는 테스트 케이스와 테스트 대상 컴포넌트 정보를 이용하여

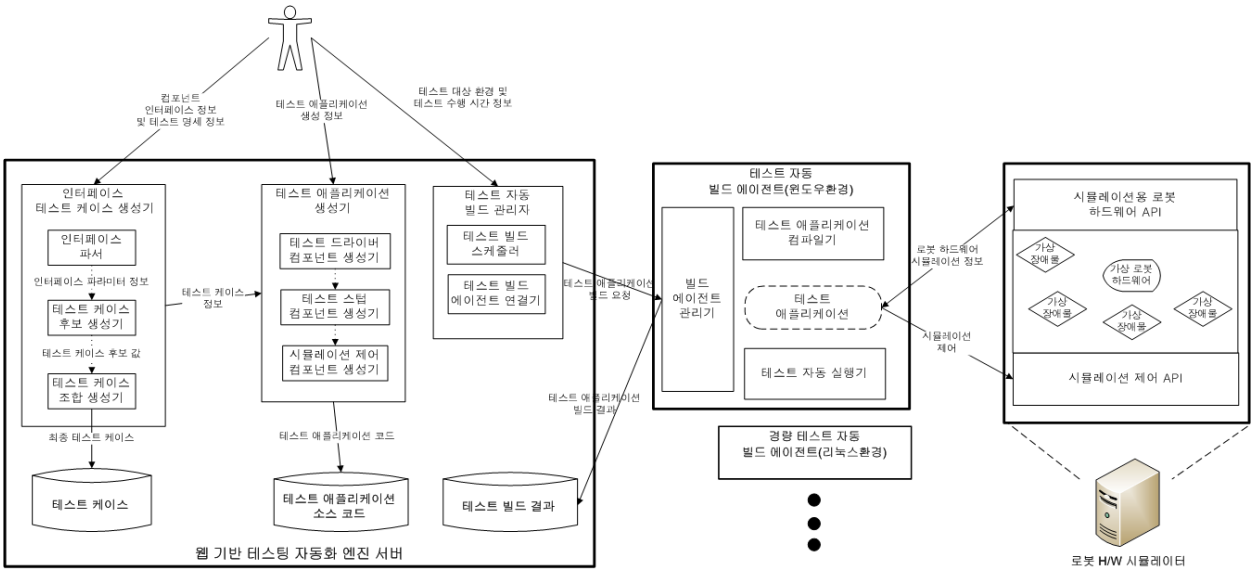


그림 1. SITAT의 상세구조.

Fig. 1. The detailed architecture of the SITAT.

테스트 케이스에 따라 테스트를 수행하는 테스트 드라이버 컴포넌트와 테스트 대상 컴포넌트의 요구 인터페이스(required interface)에 대한 테스트 스텁 컴포넌트를 생성한다. 또한 로봇 하드웨어를 대신하여 시뮬레이션 하는 로봇 H/W 시뮬레이터와 연동하기 위한 시뮬레이션 제어 컴포넌트를 생성하고, 생성된 각 컴포넌트들을 서로 연결하여 자동으로 테스트를 수행할 수 있도록 한다.

테스트 자동 빌드 관리자(automatic test build manager)는 원격의 테스트 대상 환경에 설치된 테스트 자동 빌드 에이전트와 통신하면서 테스트 빌드 요청을 하는 모듈로서 테스트 케이스 및 테스트 애플리케이션 소스 코드를 테스트 대상 환경에 다운로드하여, 소스 코드를 컴파일하거나 테스트를 수행 한 후 결과를 저장한다. 또한 테스트 빌드 요청 시 테스트 빌드 스케줄러를 통해 즉시 빌드, 예약 빌드, 그리고 주기적인 빌드를 수행할 수 있다.

1.2 테스트 자동 빌드 에이전트

테스트 자동 빌드 에이전트는 다양한 테스트 대상 환경에 설치되어 테스트 자동 빌드 관리자와 통신하면서 테스트 애플리케이션 소스 코드를 컴파일하고 테스트를 자동으로 수행하는 역할하며 다음과 같이 빌드 에이전트 관리기, 테스트 애플리케이션 컴파일기, 그리고 테스트 자동 실행기로 구성된다.

빌드 에이전트 관리기(build agent manager)는 테스트 자동 빌드 에이전트를 관리하는 모듈로 테스팅 자동화 엔진 서버의 테스트 자동 빌드 관리자로부터 테스트 빌드 요청을 받아 이를 수행하는 역할을 한다.

테스트 애플리케이션 컴파일기(test application compiler)는 테스트 수행 시 필요한 컴포넌트들을 자동으로 컴파일하고, 컴파일 후 생성되는 컴파일 로그, 실행 파일 또는 동적 라이브러리 파일을 테스팅 자동화 엔진 서버에 업로드 하는 역할을 한다.

테스트 자동 실행기(automatic test executor)는 테스트 애플리케이션을 실행하여 테스트 수행 시 출력되는 로그 및 테스트 결과를 테스팅 자동화 엔진 서버에 업로드 하는 역할을 한다.

1.3 로봇 H/W 시뮬레이터

로봇 H/W 시뮬레이터는 실제 로봇 하드웨어를 대신하여 동작을 시뮬레이션하며 가상의 테스트 환경을 제공하는 역할을 한다. 가상의 테스트 환경은 테스트 대상 컴포넌트의 특성에 맞게 수동으로 구현되며, 시뮬레이션용 로봇 하드웨어 API와 시뮬레이션 제어 API를 제공한다. 시뮬레이션용 로봇 하드웨어 API는 가상의 로봇 하드웨어를 제어하거나 데이터를 수신하기 위한 API로써 테스트 대상 컴포넌트는 이 API를 이용하여 가상의 로봇 하드웨어를 제어하거나 데이터를 수신하여 동작한다. 시뮬레이션 제어 API는 가상의 테스트 환경을 제어하기 위한 API로써 시뮬레이션 제어 컴포넌트는 이 API를 이용하여 각 테스트 케이스에 따라 동적으로 테스트 환경을 변화시켜 테스트를 수행한다.

1.4 SITAT의 각 모듈간 동작 흐름

제안하는 SITAT은 로봇 소프트웨어 컴포넌트의 인터페이스에 대한 테스트를 자동으로 수행하기 위해 그림 2와 같은 순서에 의해 동작하며, 크게 테스트 케이스 생성, 테스트 애플리케이션 생성, 그리고 테스트 수행 순서로 나뉜다.

그림 2(a)에서 보는 바와 같이 인터페이스 테스트 케이스 생성기는 웹 UI를 통해 사용자로부터 테스트 케이스 생성 요청을 수신하면 테스트 대상 인터페이스의 타입을 분석한 후 사용자로부터 테스트 명세 정보를 입력받아 테스트 케이스를 생성하여 XML 형식으로 파일에 저장하고 테스트 케이스 파일 이름을 데이터베이스에 저장한다. 사용자는 웹 UI를 통해 생성된 테스트 케이스를 확인 할 수 있으며, 각 테스트 케이스마다 예상 결과 값을 입력한 후 저장한다.

그런 다음 사용자는 그림 2(b)에서 보는 바와 같이 테스트 애플리케이션 생성기에 테스트 케이스를 수행하기 위한

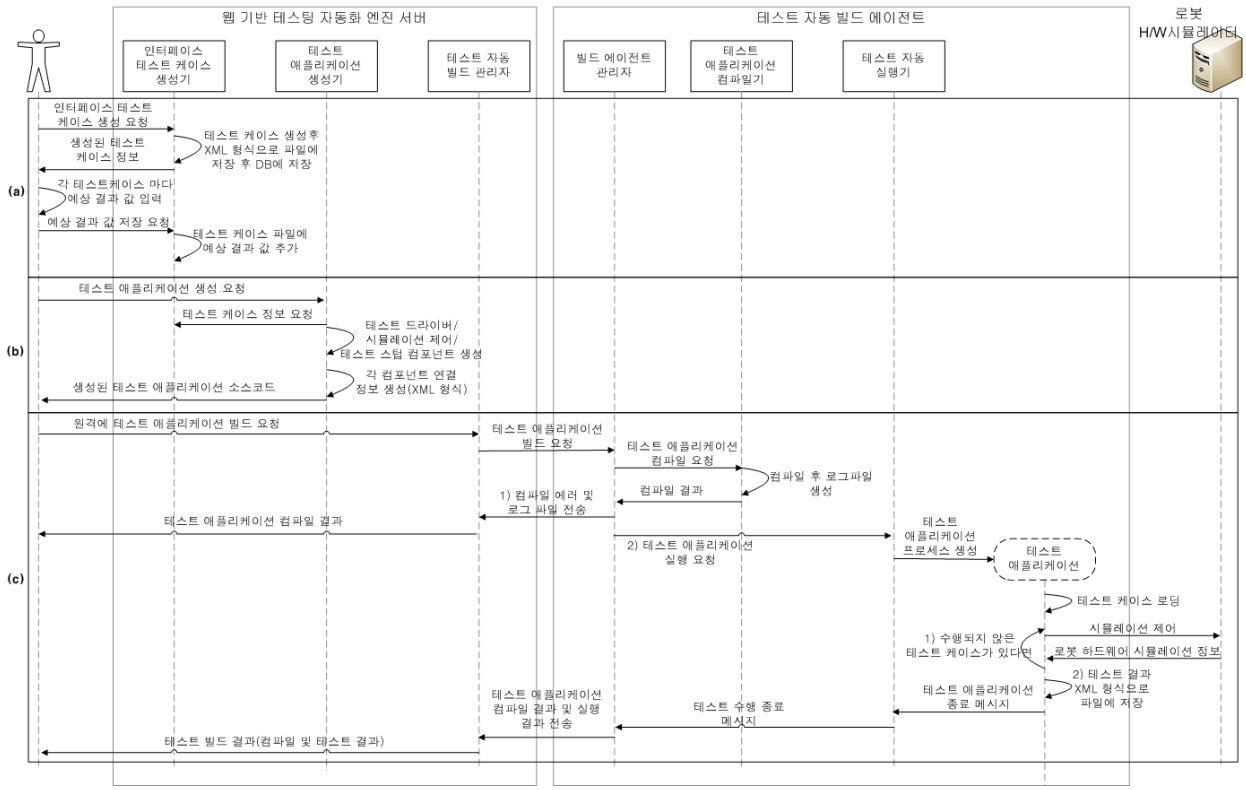


그림 2. SITAT의 테스트 케이스, 테스트 애플리케이션 생성, 그리고 테스트 애플리케이션 빌드에 대한 동작 흐름도.
Fig. 2. A simplified sequence diagram of service (a) test case generation, (b) test application generation, and (c) test application build.

애플리케이션 소스 코드 생성 요청을 한다. 테스트 애플리케이션 생성기는 테스트 케이스 정보를 이용하여 테스트 드라이버 컴포넌트와 시뮬레이션 제어 컴포넌트의 소스 코드를 생성하며, 테스트 대상 컴포넌트가 요구 인터페이스(required interface)를 가지고 있을 경우 동일한 타입의 제공 인터페이스(provided interface)를 가진 테스트 스텁 컴포넌트를 생성한다. 그런 다음 생성된 각 컴포넌트들과 테스트 대상 컴포넌트간 연결을 위한 XML 형식의 연결 정보를 생성하여 파일에 저장한다. 사용자는 웹 UI를 통해 생성된 테스트 애플리케이션의 소스 코드를 확인하거나 수정 가능하다.

테스트 케이스 및 애플리케이션 소스 코드가 생성되면 사용자는 원격의 테스트 자동 빌드 에이전트의 IP 정보를 이용하여 테스트 자동 빌드 관리자에 테스트 애플리케이션 빌드 요청을 하게 된다. 테스트 자동 빌드 관리자는 동일한 IP를 가진 테스트 자동 빌드 에이전트가 접속되어 있는지 확인한 후 접속되어 있지 않다면 사용자에게 에러를 출력하고, 접속되어 있다면 해당 테스트 자동 빌드 에이전트에 테스트 애플리케이션 빌드 요청을 하게 된다.

빌드 에이전트 관리자는 테스팅 자동화 엔진 서버의 테스트 자동 빌드 관리자로부터 빌드 요청이 오면 테스트 케이스와 테스트 애플리케이션 소스 코드를 다운 받아 테스트 애플리케이션 컴파일기를 이용하여 테스트 애플리케이션 소스 코드를 컴파일한다. 만약 컴파일 에러가 발생할 경우 테스트 애플리케이션을 실행할 수 없으므로 컴파일 로그 파일과 함께 결과를 테스팅 자동화 엔진 서버에 업로드한다. 그렇지 않고 컴파일이 성공하였다면 테스트 자동 실행기는 새로운 프로세스를 생성한 후 테스트 애플리케이션을 실행하게 된다. 테스트 애플리케이션은 각 테스트 케이스에 따라 로봇 H/W 시뮬레이터와 상호 연동하여 테스트를 수행한 후 테스트 결과를 XML 형식으로 파일에 저장한다. 테스트가 완료된 후 빌드 에이전트 관리자는 테스트 애플리케이션 컴파일 결과와 테스트 결과를 테스팅 자동화 엔진 서버로 업로드한다. 그 후 사용자는 웹 UI를 통해 테스트 애플리케이션의 컴파일 결과 및 테스트 결과를 분석하여 테스트 대상 인터페이스의 오류 유무를 확인할 수 있다. 그림 2(c)는 테스트 애플리케이션 빌드에 대한 동작 흐름도이다.

2. 인터페이스 테스트 케이스 자동 생성 방법

본 절에서는 실제 로봇 소프트웨어 컴포넌트의 인터페이스를 예로 들어 테스팅 자동화 엔진 서버의 인터페이스 테스트 케이스 생성기를 이용하여 테스트 케이스를 자동으로 생성하는 방법에 대해 상세히 기술한다.

본 논문에서 선정한 로봇 소프트웨어 컴포넌트의 인터페이스는 OPRoS (Open Platform for Robotics Service) 컴포넌트 구조[14]를 따르는 로봇 적외선 센서 컴포넌트의 getDistanceValue 인터페이스이다. getDistanceValue 인터페이스는 물리적인 적외선 센서로부터 장애물과의 거리 값을 측정하여 반환하는 인터페이스로 표 1과 같이 한개의 출력 파라미터와 두개의 입력 파라미터를 가지고 있다.

인터페이스 테스트 케이스 생성기는 소스 코드가 존재하지 않는 로봇 소프트웨어 컴포넌트에 대해 블랙박스 테스트 기법을 이용하여 테스트 케이스를 자동으로 생성하며

표 1. getDistanceValue 인터페이스 타입.
Table 1. Type of "getDistanceValue" interface.

인터페이스	getDistanceValue	
출력 파라미터	타입/이름	설 명
	double	장애물과의 거리 값
입력 파라미터	int / IndexOfSensor	적외선 센서의 Index (0~10 사이의 값)
	int / NumOfSensor	적외선 센서 값의 개수 (1~5 사이의 값)

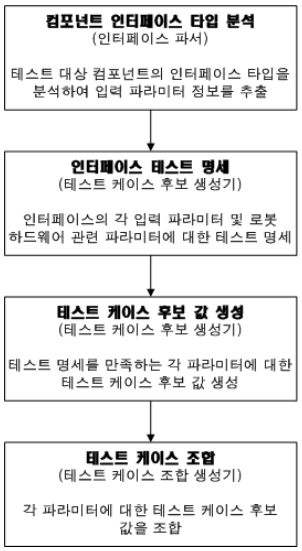


그림 3. 인터페이스 테스트 케이스 자동 생성 절차.
Fig. 3. Automatic generation procedure of test case for an interface.

테스트 케이스 생성 절차는 그림 3과 같다.

컴포넌트 인터페이스 타입 분석 단계는 테스트 대상 인터페이스의 입력 파라미터에 대한 타입을 알아내기 위한 단계이다. 인터페이스 파서는 인터페이스 표현 정보(IDL 또는 XML)를 분석하여 각 입출력 파라미터에 대한 타입 정보를 추출하는 모듈로서 표 1과 같이 getDistanceValue 인터페이스의 입출력 파라미터에 대한 타입을 자동으로 추출한다.

인터페이스 테스트 명세 단계는 추출된 테스트 대상 인터페이스의 입력 파라미터와 시뮬레이션 제어 파라미터에 대한 테스트 명세 정보를 생성하는 단계이다. 사용자는 인터페이스의 입력 파라미터에 대한 범위 값 또는 특정 후보 값을 입력하고, 테스트 대상 컴포넌트가 로봇 하드웨어와 연동하여 동작 할 경우 시뮬레이션 제어 관련 파라미터 정보를 입력한다. 이 단계에서 getDistanceValue 인터페이스의 "IndexOfSensor" 파라미터에 대한 범위 값인 0~10 사이의 값과 "NumOfSensor" 파라미터에 대한 범위 값인 1~5 사이의 값을 입력한다. 또한 이 인터페이스는 적외선 센서로부터 장애물과의 거리 값을 측정하여 반환하므로, 테스트 시뮬레이션 환경 내에 존재하는 가상의 장애물 위치를 제어하기 위한 파라미터인 "#Distance" 파라미터를 추가하고 범위 값인 0~10 사이의 값을 입력한다.

테스트 케이스 후보 값 생성 단계는 각 파라미터에 대한 테스트 명세 정보를 이용하여 테스트 케이스의 후보 값을 생성하는 단계이다. 테스트 케이스 후보 생성기는 각 파라미터에 대한 테스트 명세 정보가 특정 값이 아닌 범위 값일 경우 동치 분할 또는 경계 값 분석 기법을 이용하여 자동으로 테스트 케이스 후보를 생성한다. 동치 분할 기법은 입력 도메인(input domain)을 범위 입력조건, 특수 값의 제한조건, 집합 소속여부 조건, 논리 조건 등에 근거하여 동치 클래스(equivalence class)로 분할하고, 한 클래스 내의 데이터에서 에러가 발생하면 그 동치 클래스 안의 다른 데이터에서도 같은 에러가 발생하게 될 것이라는 가정 하에 각 클래스마다 대표 테스트 케이스 후보를 선정하는 기법이다. 경계 값 분석 기법은 동치 분할 방법의 변형으로써, 입출력 도메인을 동치 클래스로 나누었을 때, 각 범위의 경계 값에서 에러가 발생하는 경우가 많다는 사실에 착안하여 에러 검출 가능성을 높을 수 있는 방법이다. 즉, 동치 클래스 안에서 테스트 케이스를 선정할 때, 임의의 데이터를 이용하는 것 대신에 각 클래스의 경계에 있는 데이터를 이용한다. 이 단계에서 getDistanceValue 인터페이스를 테스트하기 위한 각 파라미터에 대한 후보 값이 표 2와 같이 생성되며, getDistanceValue 인터페이스의 입력 파라미터에 대한 테스트 케이스(Test case for input parameter, 이후 TCIP라 칭함) 타입인 IndexOfSensor, NumOfSensor와 시뮬레이션 제어를 위한 테스트 케이스(Test case for simulation control, 이후 TCSC라 칭함) 타입인 #Distance로 나뉜다.

테스트 케이스 조합 단계는 생성된 테스트 케이스 후보 값들에 대해 페어와이즈(pair-wise) 기법을 이용하여 각 후보 값들을 서로 조합함으로써 테스트 케이스의 수를 줄이는 단계이다. 페어와이즈 기법은 대부분의 결함들이 파라미터들의 상호작용을 통해 발생된다는 관찰에 기반을 둔 효과적인 테스트 케이스 생성기법으로 파라미터의 쌍들이 전체 테스트 케이스에서 최소한으로만 나타나도록 하는 것이 한다.

만약 표 2와 같이 테스트 케이스 후보 생성 단계에서 IndexOfSensor 파라미터의 후보 값이 4개, NumOfSensor 파라미터의 후보 값이 4개, 그리고 #Distance파라미터의 후보 값이 4개라고 한다면 표 3과 같이 모든 가능한 조합은 64가지가 된다.

하지만 getDistanceValue 인터페이스와 같이 두 파라미터의 상호작용에 의해 결함이 발견된다면, 2-way 페어와이즈 기법을 적용하여 I*N, D*I, N*D의 두 파라미터의 조합 쌍에 대해 중복된 테스트 케이스를 제거할 수 있다. 즉, 표 3의 1번째 테스트 케이스에서 각 파라미터의 조합 쌍은 (I*N)1={-1,-1}, (D*I)1={-1.0,-1}, (N*D)1={-1,-1.0}인데 (I*N)1

표 2. getDistanceValue 인터페이스의 테스트 케이스 후보 값.
Table 2. Test case candidates of "getDistanceValue" interface.

파라미터 타입	파라미터 명	테스트 케이스 후보 값
입력 파라미터	IndexOfSensor	-1, 0, 5, 11
	NumOfSensor	-1, 1, 4, 7
시뮬레이션 제어 파라미터	#Distance	-1.0, 0.5, 5.7, 11.5

표 3. getDistanceValue 인터페이스에 대한 테스트 케이스 조합.

Table 3. Combination of test cases for "getDistanceValue" interface.

번호	파라미터			두 파라미터간 중복되는 번호 (I*N, D*I, N*D)
	IndexOf Sensor(I)	NumOfSensor(N)	#Distance(D)	
1	-1	-1	-1.0	2, 13, 17
2	-1	-1	0.5	
3	-1	-1	5.7	2, 7, 35
4	-1	-1	11.5	2, 12, 52
5	-1	1	-1.0	7, 13, 37
6	-1	1	0.5	7, 2, 22
7	-1	1	5.7	
8	-1	1	11.5	7, 12, 24
.....				
12	-1	4	11.5	
13	-1	7	-1.0	
17	0	-1	-1	
22	0	1	0.5	
24	0	1	11.5	
27	0	4	5.7	
30	0	7	0.5	
35	5	-1	5.7	
37	5	1	-1	
42	5	4	0.5	
48	5	7	11.5	
52	11	-1	11.5	
54	11	1	0.5	
57	11	4	-1	
63	11	7	5.7	
64	11	7	11.5	63, 52, 48

값은 2번째 테스트 케이스의 (I*N)2 값과 중복되며 (D*I)1 값은 13번째 테스트 케이스의 (D*I)13 값과 중복된다. 또한 (N*D)1 값은 17번째 테스트 케이스의 (N*D)17 값과 중복된다. 그러므로 1번째 테스트 케이스 대한 두 파라미터간의 조합이 이미 다른 테스트 케이스에 중복되어 존재하므로 제거될 수 있다. 이와 동일한 방법으로 각 테스트 케이스를 제거하게 되면 17개의 테스트 케이스로 줄일 수 있으며, 두 파라미터의 조합 쌍에 대해 최소한으로만 나타나게 된다.

그러나 사용자는 테스트 대상 인터페이스에 따라 두 파라미터의 조합 쌍 또는 세 파라미터의 조합 쌍에 대해 중복된 테스트 케이스를 제거 할 수 있다. 그러므로 테스트 케이스 조합 생성기는 각 파라미터에 대해 2-way 페어와이즈 조합, 3-way 페어와이즈 조합, 그리고 모든 가능한 조합을 생성하는 All-combination 조합 방법을 제공한다.

3. 인터페이스 테스트 애플리케이션 및 테스트 수행 흐름

본 절에서는 앞 절에서 생성된 테스트 케이스에 따라 자동으로 테스트를 수행하는 테스트 애플리케이션과 테스트 애플리케이션을 이용하여 테스트를 자동으로 수행하는 순서에 대해 상세히 기술한다.

테스트 케이스에 따라 자동으로 테스트를 수행하기 위해서는 그림 4에서 보는 바와 같이 테스트 애플리케이션과 로봇 H/W 시뮬레이터가 필요하다.

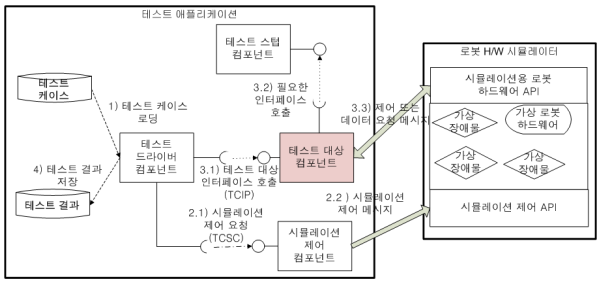


그림 4. 인터페이스 테스트 애플리케이션 구조.

Fig. 4. Architecture of test application for interface test.

3.1 테스트 애플리케이션 구성 요소

테스트 애플리케이션은 테스트 대상 컴포넌트를 테스트하기 위해 필요한 컴포넌트들과 테스트 케이스, 그리고 테스트 결과 파일로 구성되며, 테스트 대상 환경에 설치된 테스트 빌드 에이전트에 의해 자동으로 실행된다.

테스트 드라이버 컴포넌트는 전체 테스트 동작을 제어하는 컴포넌트로서 테스트 케이스 파일을 읽어 테스트 대상 인터페이스를 테스트 하는 역할을 한다. 테스트 수행 시 입력된 테스트 케이스를 TCSC와 TCIP로 나누고 TCSC를 이용하여 시뮬레이션 제어 컴포넌트의 인터페이스를 통해 테스트를 위한 시뮬레이션 환경을 우선 설정하고, TCIP를 이용하여 테스트 대상 인터페이스를 호출하여 테스트를 수행하며 결과를 파일에 저장한다.

시뮬레이션 제어 컴포넌트는 TCSC에 따라 테스트를 위한 시뮬레이션 환경을 설정하는 컴포넌트로서 로봇 H/W 시뮬레이터가 제공하는 시뮬레이션 제어 API를 이용하여 시뮬레이션 환경에 물체 등을 제어할 수 있다. 그리고 테스트 수행 시 테스트 드라이버와 시뮬레이션 제어 부분을 분리함으로써 동일한 인터페이스의 입력 파라미터에 대해 다양하게 로봇 H/W 시뮬레이터를 제어하여 테스트함으로써 테스트 케이스의 재사용성을 높일 수 있다.

테스트 대상 컴포넌트는 테스트 시 실제 로봇 하드웨어 API와 동일한 인터페이스를 가진 시뮬레이션용 로봇 하드웨어 API를 이용하여 로봇 H/W 시뮬레이터로부터 로봇 하드웨어 정보를 입력 받아 기능을 수행한다. 그러나 테스트 대상 컴포넌트가 요구 인터페이스를 가지고 있을 경우 동일한 타입의 제공 인터페이스를 가진 컴포넌트가 존재하지 않으면 기능을 수행할 수 없다. 컴포넌트 기반 소프트웨어 개발에서는 이와 같은 일이 발생할 수 있으므로 가상의 동일한 인터페이스 타입을 가진 테스트 스텁 컴포넌트를 생성하여 테스트를 수행한다. 테스트 스텁 컴포넌트는 실제 로봇 소프트웨어 컴포넌트를 대신하여 테스트 대상 컴포넌트가 기능을 수행할 수 있도록 도와주는 역할을 한다.

3.2 테스트 애플리케이션을 이용한 테스트 자동 수행 흐름

테스트 애플리케이션 및 로봇 H/W 시뮬레이터를 이용하여 테스트를 수행하는 순서는 그림 4에서 보는바와 같이 테스트 드라이버 컴포넌트에 의해 시작된다.

테스트 드라이버 컴포넌트는 테스트 케이스 파일을 로딩

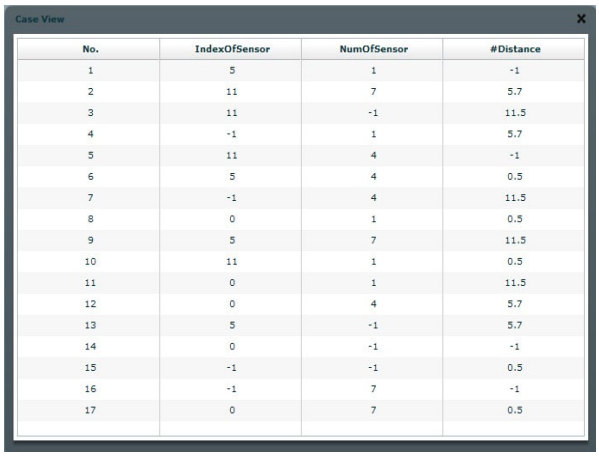
한 후 가상의 테스트 환경을 설정하기 위해 시뮬레이션 제어 컴포넌트의 시뮬레이션 제어 인터페이스를 통해 TCSC를 전송한다. 시뮬레이션 제어 컴포넌트는 TCSC에 따라 로봇 H/W 시뮬레이터에서 제공하는 시뮬레이션 제어 API를 이용하여 가상의 테스트 환경에 존재하는 물체 등의 위치를 변경한다.

가상 테스트 환경 설정이 완료된 후 테스트 드라이버 컴포넌트는 TCIP를 입력 파라미터로 하여 테스트 대상 인터페이스를 호출한다. 테스트 대상 컴포넌트는 테스트 스텁 컴포넌트의 인터페이스를 호출한 후 시뮬레이션용 로봇 하드웨어 API를 이용하여 로봇 하드웨어를 제어하거나 데이터를 수신 받아 동작을 수행한다. 동작이 완료된 후 결과 값을 테스트 드라이버 컴포넌트에게 반환하고 테스트 드라이버 컴포넌트는 실제 반환 값과 예상 결과 값을 비교하여 테스트의 성공 여부를 테스트 결과 파일에 저장하고 테스트를 완료한다.

IV. 실험 및 결과

이 절에서는 III 장에서 제안한 SITAT의 유효성 및 효과성을 검증하기 위하여 getDistanceValue 인터페이스에 대해 테스트를 수행하고 결과를 분석한다. 이를 위해 사용되는 시뮬레이션 엔진은 OPRoS에서 제공하는 것을 사용한다 [15].

getDistanceValue 인터페이스를 테스트하기 위한 테스트 애플리케이션과 테스트 시뮬레이션 환경은 그림 5에서 보는바와 같이 구성하였으며, OPRoS 적외선 센서 컴포넌트의 경우 요구 인터페이스(Required interface)가 존재하지 않으므로 테스트 스텁 컴포넌트를 생성할 필요가 없었다. 우선 SITAT의 인터페이스 테스트 케이스 자동 생성기를 이용하여 그림 6과 같은 테스트 케이스를 자동으로 생성 하였으며, 생성된 테스트 케이스는 표 3과 동일함을 알 수 있다. 그 후 각 테스트 케이스에 대한 예상 결과 값을 수동으로 삽입하고, 테스트 애플리케이션 생성기를 이용하여 테스트 애플리케이션을 생성하였다. 그림 7은 getDistanceValue 인터페이스 테스트 드라이버 컴포넌트에 대한 소스 코드의 일부분을 나타낸다.



Case View

No.	IndexOfSensor	NumOfSensor	#Distance
1	5	1	-1
2	11	7	5.7
3	11	-1	11.5
4	-1	1	5.7
5	11	4	-1
6	5	4	0.5
7	-1	4	11.5
8	0	1	0.5
9	5	7	11.5
10	11	1	0.5
11	0	1	11.5
12	0	4	5.7
13	5	-1	5.7
14	0	-1	-1
15	-1	-1	0.5
16	-1	7	-1
17	0	7	0.5

그림 6. SITAT를 이용하여 생성된 getDistanceValue 인터페이스 테스트 케이스.

Fig. 6. “getDistanceValue” interface test cases by generated SITAT.

getDistanceValue 인터페이스 테스트 드라이버 컴포넌트는 XML 형식의 테스트 케이스 파일을 로딩하여 TCSC 타입인 #Distance 테스트 케이스와 TCIP 타입인 IndexOfSensor, NumOfSensor 테스트 케이스로 나눈다. 그런 다음 장애물 거리 제어 컴포넌트는 로봇 H/W 시뮬레이터에서 제공하는 장애물 거리 제어 API를 이용하여 #Distance 테스트 케이스 값 만큼 적외선 센서로부터 장애물을 이동시킨다. 장애물의 이동이 완료되면 테스트 드라이버 컴포넌트는 IndexOfSensor, NumOfSensor 테스트 케이스 값을 입력 파라미터로 하여 테스트 대상 컴포넌트의 getDistanceValue 인터페이스를 호출한다. OPRoS 적외선 센서 컴포넌트는 로봇 H/W 시뮬레이터에서 제공하는 적외선 센서 시뮬레이션 API를 이용하여 장애물과의 거리 값을 계산하여 테스트 드라이버 컴포넌트에 반환한다. 테스트 드라이버 컴포넌트는

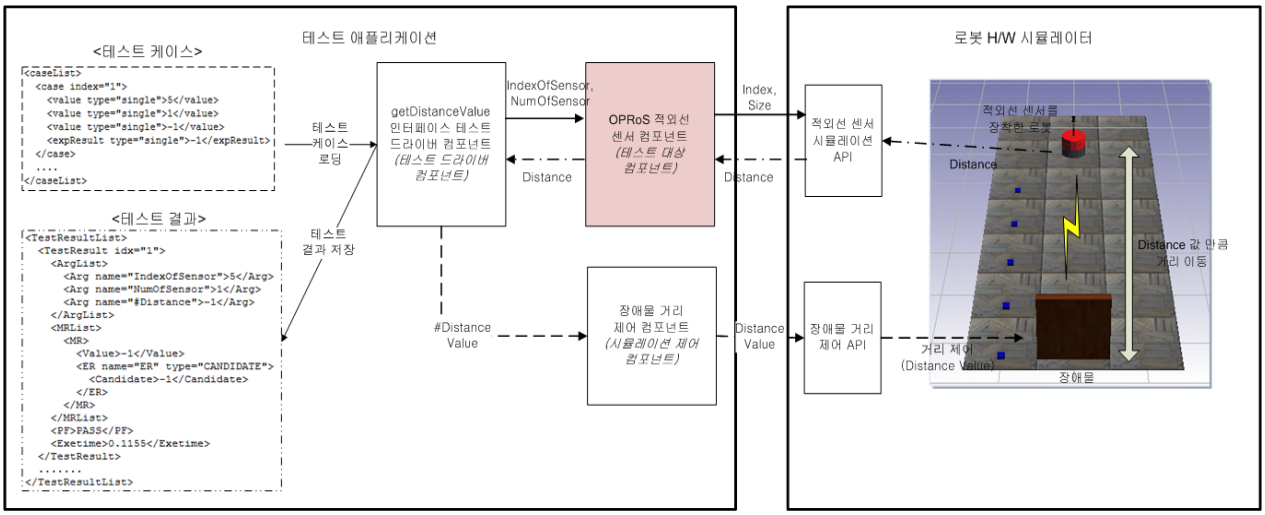


그림 5. getDistanceValue 인터페이스 테스트를 위한 테스트 시스템 구조.

Fig. 5. Test system for testing “getDistanceValue” interface.

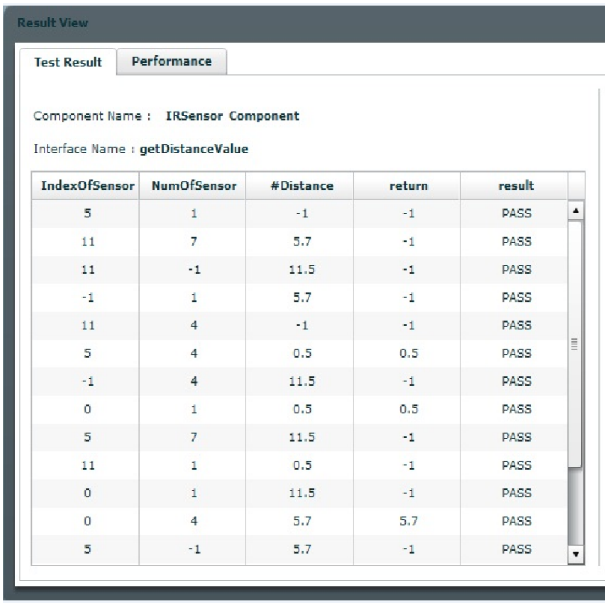
```
void
IRTestDriver::runTest(const std::string tcFile, const std::string trFile)
{
  // Read test case file
  TestSuiteRW *tcr = new
                TestSuiteRW("IRSensor_IRServiceInterface_TC", tcFile);
  tcr->openTS();

  TestTimeUtil::init_timer();   //Init timer

  int numOfTc = tcr->getSizeOfTC();
  for(int cntOfTc=0; cntOfTc< numOfTc; cntOfTc++){
    // Type casting of test case value
    struct testCaseInfo *tc = tcr->getTestCase(cntOfTc);
    ArgInfoList *argList = tc->pArgList;
    string *strIndex = argList->getArgInfo(0)->value;
    string *strNumOfSensor = argList->getArgInfo(1)->value;
    string *strDistance = argList->getArgInfo(2)->value;
    int intIndex = String2Integer(strIndex->c_str());
    int intNumOfSensor = String2Integer(strNumOfSensor->c_str());
    double doubleDistance = String2Double(strDistance->c_str());

    // moving a wall in the robot simuator
    ObstacleControlComp->moveWall(doubleDistance);

    // Invocation target test interface
    double resTime = 0.0;
    TestTimeUtil::start_timer(1);
    double ret = IRSensorCompObj->getDistanceValue(intIndex,
                                                   intNumOfSensor);
    resTime = TestTimeUtil::get_timer(1);

    // Comparing test result
    string strResTime, strRet;
    Double2String(ret, strRet);
    Double2String(resTime, strResTime);
    struct ER_Info *erInfo = tc->pERList->getERInfo(0);
    bool erValue =
        String2Bool(erInfo->ERstrCandidates->getString(0)->c_str());
    string pf = "FAIL";
    if(erValue == ret) pf = "PASS";
```

그림 7. getDistanceValue 인터페이스 테스트 드라이버 컴포넌트 C++ 소스 코드

Fig. 7. C++ code of "getDistanceValue" interface test driver component

실제 반환 값과 예상 결과 값을 비교하여 테스트 성공 여부를 테스트 결과 파일에 저장하고 테스트를 완료하였다. 즉, 장애물의 위치를 변화 시켰을 때 정확한 거리 값을 "getDistanceValue" 인터페이스를 통해 수신할 수 있는지를 테스트 하였다.

그림 8은 getDistanceValue 인터페이스에 대한 테스트 결과를 나타낸다. 그림 8에서 첫번째부터 세번째까지의 열은 테스트 케이스, 네번째 열은 실제 결과 값, 그리고 다섯 번째 열은 성공/실패 여부를 나타낸다. 테스트 케이스 중 #Distance의 의미는 적외선 센서와 장애물과의 거리를 의미하고 -1 값의 의미는 범위를 벗어난 값을 말한다. 그림 8에서 첫번째 테스트 케이스로써 IndexOfSensor, NumOfSensor, #Distance가 5, 1, -1 값이 입력될 경우 #Distance 값이 범위를 벗어난 값 이므로 예상 결과 값은 에러를 의미하는 -1 값인데 실제 결과 값이 -1을 출력하므로 테스트가 성공하였으며, 이에 따라 result 항목이 PASS 값을 가진다. 동일한



IndexOfSensor	NumOfSensor	#Distance	return	result
5	1	-1	-1	PASS
11	7	5.7	-1	PASS
11	-1	11.5	-1	PASS
-1	1	5.7	-1	PASS
11	4	-1	-1	PASS
5	4	0.5	0.5	PASS
-1	4	11.5	-1	PASS
0	1	0.5	0.5	PASS
5	7	11.5	-1	PASS
11	1	0.5	-1	PASS
0	1	11.5	-1	PASS
0	4	5.7	5.7	PASS
5	-1	5.7	-1	PASS

그림 8. getDistanceValue 인터페이스 테스트 결과.

Fig. 8. Test result of "getDistanceValue" interface.

방법으로 여섯 번째 테스트 케이스 값인 IndexOfSensor=5, NumOfSensor=4, #Distance=0.5에 대해 장애물과의 거리가 0.5이므로 예상 결과 값이 0.5 값인데 실제 결과 값이 0.5이므로 이 테스트 케이스에 대해서도 result 항목이 PASS 값을 가진다.

즉, 모든 테스트 케이스에 대해 테스트가 성공하였으므로 OPRoS 적외선 센서 컴포넌트가 정상적으로 동작한다는 것을 검증할 수 있었다. 또한 위의 테스트 결과에서 알 수 있듯이 본 논문에서 제안하는 SITAT는 실제 로봇 소프트웨어 컴포넌트를 테스트 하기 위한 테스트 케이스를 자동으로 생성하고 생성된 테스트 케이스에 따라 로봇 H/W 시뮬레이터와 연동하여 테스트를 자동으로 수행할 수 있음을 보였다.

V. 결론

본 논문에서는 로봇 소프트웨어 컴포넌트의 인터페이스 테스트를 효율적으로 하기 위한 시뮬레이션 기반 로봇 S/W 컴포넌트 인터페이스 테스팅 자동화 도구를 제안하였다.

제안한 SITAT은 블랙박스 테스팅 기법인 동치분할, 경계값 분석, 그리고 페어와이즈 기법을 이용하여 테스트 케이스를 자동으로 생성하고, 생성된 테스트 케이스에 따라 테스트를 자동으로 수행하는 테스트 애플리케이션 소스 코드를 생성하는 기능을 제공한다. 그리고 생성된 테스트 케이스와 테스트 애플리케이션을 이용하여 로봇 H/W 시뮬레이터와 연동함으로써 하드웨어와 연동된 로봇 소프트웨어 컴포넌트 등의 테스트를 자동으로 수행할 수 있는 기능을 제공한다.

제안한 테스팅 자동화 도구의 유효성과 효과성을 검증하기 위해 실제 로봇 소프트웨어 컴포넌트를 대상으로 테스트를 수행 하였으며 테스트 대상 컴포넌트가 정상적으로 동작함을 검증할 수 있었다.

참고문헌

[1] G. J. Myers, *The Art of Software Testing*, John Wiley & Sons, 2004.

[2] U. Buy, C. Ghezzi, A. Orso, M. Pezze, and M. Valsasna, "A framework for testing object-oriented components," *Proc. of the 1st International Workshop on Testing Distributed Component-Based Systems*, 1999.

[3] G. A. Bundell, G. Lee, J. Morris, and K. Parker, "A software component verification tool," *Proc. of the Conf. Software Methods and Tools*, pp. 137-146, 2000.

[4] K. Z. Zamli, N. A. M. Isa, M. F. J. Klaib, and S. N. Azizan, "A tool for automated test data generation(and execution) based on combinatorial approach," *International Journal of Software Engineering and Its Applications*, vol. 1, no. 1, pp. 19-35, July 2007.

[5] M. Momotko and L. Zalewska, "Component+ Built-in Testing: A Technology for Testing Software Components," *Foundations of Computing and Decision Sciences*, pp. 133-148, 2004.

[6] S. H. Edwards, "A framework for practical, automated black-box testing of component-based software," *International Journal of Software Testing, Verification and Reliability*, vol. 11, no. 2, pp. 97-111, June 2001.

[7] A. Polini and A. Bertolino, "A user-oriented framework for component deployment testing," *Testing Commercial-off-the-shelf components and system*, pp. 169-194, 2005.

[8] W. E. Lewis, *Software Testing and Continuous Quality Improvement*, Tayor & Francis, 2000.

[9] P. C. Jorgensen, *Software Testing: A Craftsman's Approach*, 2nd Ed., CRC Press, 2002.

[10] B. Beizer, *Software Testing Techniques*, International Thomson Computer Press, 1990.

[11] K. Beck, *Test-Driven Development by Example*, Addision-Wesley, 2002.

[12] A. W. Williams and R. L. Probert, "A practical strategy for testing pair-wise coverage of network interfaces," *Proc. of the 8th International Symposium on Software Reliability Engineering*, pp. 246-254, 1996.

[13] A. W. Williams, "Determination of test configurations for pair-wise interaction coverage," *Proc. of the 13th Conf. Testing of Communicating Systems*, pp. 59-74, August 2000.

[14] 지능형로봇 표준포럼, "개방형 로봇 소프트웨어 플랫폼: 제 2부-컴포넌트," KOROS 1067-2:2009, 2009.

[15] OPRoS Simulator, http://www.opros.or.kr/

강 정 석

2006년 강원대학교 전자통신공학과 졸업. 2008년 동 대학원 석사. 2009년~현재 동 대학원 박사과정. 관심분야는 소프트웨어 테스팅 자동화, CBSE, 소프트웨어 공학, 통신 미들웨어.

최 형 섭

2008년 강원대학교 IT학부 전자 통신공학과 학사 졸업. 2009년~현재 동 대학원 석사 과정. 관심분야는 컴퓨터 언어, 소프트웨어 테스팅, 컴퓨터 네트워크.

맹 상 우

2008년 강원대학교 전자통신공학과 졸업. 2008년~현재 동 대학원 석사과정. 관심분야는 소프트웨어 테스팅 자동화, 소프트웨어 공학, 영상 처리.

김 시 완

2010년 강원대학교 전자통신공학과 졸업. 2010년~현재 동 대학원 석사과정. 관심분야는 Rich Internet Application, 소프트웨어 테스팅 자동화, 소프트웨어 공학.

박 홍 성

1983년 서울대학교 제어계측공학과 학사 졸업. 1986년 동 대학원 석사. 1992년 동 대학원 박사. 1992년~현재 강원대학교 전자통신공학과 교수. 관심분야는 로봇 S/W 플랫폼, 무선데이터통신, 실시간 통신.