DOI: 10.3745/KIPSTD.2009.16-D.1.65

UML 상태기계 다이어그램을 이용한 컴포넌트 인터페이스의 행위 호환성 검증 도구

김 호 준[†] · 이 우 진[††]

요 약

현재 컴포넌트 기반 개발 기법은 재사용성과 생산성 측면에서 효과적인 소프트웨어 개발 방법으로 많은 각광을 받고 있다. 하지만 기존의 UML을 이용한 컴포넌트 기반 개발에서는 컴포넌트의 행위를 배제하고 컴포넌트 인터페이스만 참조하여 컴포넌트를 설계함으로써, 컴포넌트의 구체적인 행위에 대한 파악과 컴포넌트 간 인터페이스 호환성 보장이 불가능하다. 이에 따라 컴포넌트 설계 단계에서 컴포넌트의 행위를 상태기계 다이어그램으로 표현하고, 표현된 상태기계 다이어그램을 통해 컴포넌트의 행위 호환성을 보장할 필요가 있다. 이 연구에서는 상태기계 다이어그램으로 표현된 컴포넌트의 행위를 관찰 일치(observation equivalence)와 호출 일관성(invocation consistency)의 개념을 이용하여 행위 호환성을 검증하는 방법을 제공하고, 동적으로 이를 수행하는 도구를 개발한다.

키워드 : 인터페이스 행위 호환성, 컴포넌트 인터페이스, UML 상태기계

A Behavior Conformance Checker for Component Interfaces using UML State Machine Diagram

Ho Jun Kim[†] · Woo Jin Lee[††]

ABSTRACT

Component based development has increasingly become important in the software industry. However, in the current component based development approach with UML, the absence of behavioral description of components brings about a cost problem which causes semantic errors on the testing phase. Accordingly we cannot grasp the usage pattern of component by its provided interfaces which refer to an abstraction of software component. And we cannot guarantee the behavioral conformance of the provided and required interfaces of components. In order to solve these problems, we describe the behaviors of component interfaces by state machine diagram and guarantee their behavior conformance at the modeling phase. We also propose a method to guarantee the behavior conformance of component interfaces with concept of observation equivalence and invocation consistency. And we provide an analyzing tool which checks interface behavior conformance.

Keywords : Interface Behavior Conformance, Component Interface, UML State Machine

1. 서 론

소프트웨어 개발에 있어서 컴포넌트를 기반으로 소프트웨어의 재사용성을 제공해주는 소프트웨어 개발 방법을 컴포넌트 기반 개발 방법(Component Based Development, CBD)[1]이라 한다. 컴포넌트 기반 개발은 객체지향 개발 방법론으로부터 발전되어 온 방법론으로 소프트웨어 개발을 컴포넌트 단위로 개발하여 재사용성을 극대화할 수 있다. 초기에는 RUP[2], Catalysis[3] 등과 같은 CBD 방법론이 소개되었고, Sun의 J2EE[4]와 Microsoft의 .NET[5]과 같은 CBD 플랫폼이 확산됨에 따라 최근 더욱 주목받고 있다.

컴포넌트 기반 소프트웨어 개발의 가장 큰 특징은 소프트웨어를 컴포넌트 단위로 개발하여 컴포넌트 재사용성이나 대체성을 제공하는 데에 있다. 이를 위해 컴포넌트의 내부 정보를 은닉하고 인터페이스를 통해 공개된 서비스만을 제공한다. 기존의 UML(Unified Modeling Language)[6]을 기반으로 한 CBD에서는 시그너쳐 기반으로 컴포넌트 인터페이스를 정의한다. 하지만 시그너쳐 기반의 컴포넌트 인터페이스 정의만으로는 사용자 입장에서 컴포넌트의 사용 패턴을 파악할 수 없는 단점이 있다. 따라서, 행위 기반의 컴포넌트 인터페이스 정의가 행해져야만 사용자 입장에서 컴포

※ 본 연구는 BK 21 사업과 방위사업청과 국방과학연구소의 지원으로 수행되었습니다(2009-SW-32-DJ-01).
† 정 회 원 : 삼성전자 근무
†† 정 회 원 : 경북대학교 전자전기컴퓨터학부 부교수
논문접수 : 2008년 7월 28일
수 정 일 : 1차 2008년 11월 3일
심사완료 : 2008년 11월 6일

넌트 인터페이스를 통한 컴포넌트의 사용 패턴 파악이 가능하고, 컴포넌트 조합 시에 의미적으로도 정확한 조합이 가능하다. 이를 위해서는 컴포넌트와 인터페이스의 행위 표현이 필요하며 이를 이용한 인터페이스 행위 호환성 검사 방법이 필요하다.

컴포넌트 인터페이스의 정형적 표현에 대한 대표적인 연구로는 정형적 컴포넌트 명세 언어인 Wright[7], 컴포넌트의 행위 프로토콜[8]의 연구를 들 수 있다. Wright에서는 프로세스 대수의 하나인 CSP(Communicating Sequential Processes)[9]를 이용하여 컴포넌트의 연결성을 표현하고 있고 행위 프로토콜에서도 프로세스 대수와 유사한 방법으로 컴포넌트 인터페이스를 표현하고 있다. 이러한 연구들은 프로세스 대수라는 정형적 기법을 사용하고 있어 일반 사용자들이 이해하기에는 조금 어려운 점이 있으며 또한 컴포넌트의 연결성 관점에서만 행위를 표현하며 내부 컴포넌트의 행위에 대한 모델은 제공하지 않고 있다. 행위 모델링 관점에서 컴포넌트간의 연결성뿐만 아니라 컴포넌트 내부 행위를 표현하여야 하며 또한 컴포넌트 인터페이스와 컴포넌트 내부 행위간의 연계성을 나타낼 수 있어야 한다.

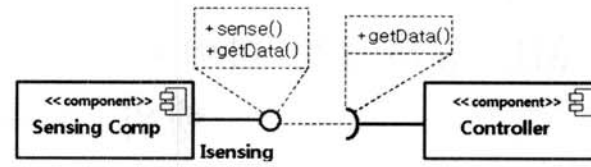
본 논문에서는 기존 UML기반 CBD의 문제점을 해결하고 컴포넌트 인터페이스의 행위 호환성을 보장하기 위한 검증 방법에 대해 기술한다. 컴포넌트 인터페이스의 행위 호환성은 컴포넌트의 인터페이스 명세에 대한 만족성과 컴포넌트 인터페이스 연결성의 두 가지 측면이 있다. 컴포넌트의 인터페이스 명세에 대한 검증은 인터페이스에 기술된 행위에 대한 컴포넌트 내부 행위의 검증이며, 컴포넌트 간 인터페이스 연결성은 요구 인터페이스 행위에 대한 제공 인터페이스 행위 검증이다. 본 논문에서는 컴포넌트의 행위를 상태기계(state machine)로 표현하고 이를 비교적 간단한 레이블 전이 시스템(Labeled Transition System)으로 변형하여 관찰 일치(observation equivalence)와 호출 일관성(invocation consistency)[10]을 통하여 검증한다. 또한 이를 컴포넌트 설계 단계에서 동적으로 수행하는 도구를 제안한다.

본 논문의 구성은 다음과 같다. 제 2 절에서는 연구배경으로 UML 컴포넌트 다이어그램과 상태기계 다이어그램을 소개하고 이 논문에서 분석 시에 활용되는 LTS 모델을 소개한다. 제 3 절에서는 컴포넌트 인터페이스 행위 호환성 검증 기법에 대해서 상세히 다루고 제 4 절에서는 이를 지원하는 검증 도구의 구조를 설명한다. 제 5 절에서는 도구 구현환경과 수행 결과를 기술하고 제 6 절에서 결론을 맺는다.

2. 연구 배경

2.1 UML 컴포넌트 다이어그램

컴포넌트란 소프트웨어 시스템에서 독립적인 업무 또는 독립적인 기능을 수행하는 모듈로서 인터페이스를 통해서만 접근 가능하며 다른 컴포넌트와 조합이 가능해야 한다[1]. 이 논문에서는 UML 2.1 명세[6]의 컴포넌트 표기와 인터페이스의 개념을 사용한다. 컴포넌트 인터페이스는 컴포넌트



(그림 1) 오퍼레이션 간의 순서 제약이 있는 인터페이스

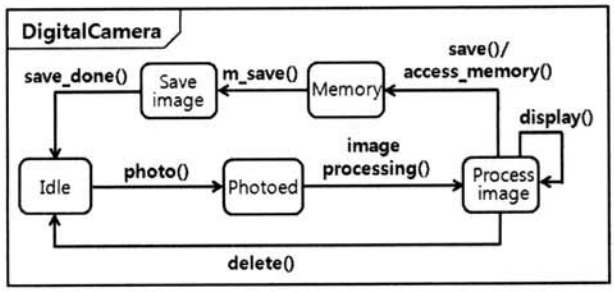
간의 연결 정보를 나타내는 것으로 특정 컴포넌트가 제공하는 또는 필요로 하는 오퍼레이션 집합을 나타낸다. 컴포넌트가 다른 컴포넌트에게 제공하는 서비스를 제공 인터페이스(provided interface)라 하고, 자신의 내부 행위를 수행을 위해 필요로 하는 다른 컴포넌트의 서비스를 요청 인터페이스(required interface)라 한다. 인터페이스는 일종의 규약이며 인터페이스를 제공하는 컴포넌트는 인터페이스에 표현된 규약을 만족해야 한다.

UML 2.1 명세에는 이러한 컴포넌트와 컴포넌트 간의 관계를 컴포넌트 다이어그램으로 표현한다. 컴포넌트 간의 연관 관계는 제공 인터페이스와 요구 인터페이스를 연결하여 표현한다. 컴포넌트는 외부에 제공하는 서비스를 제공 인터페이스를 통해 공표하며, 인터페이스를 통해 호환성을 제공한다. 따라서, 기존 컴포넌트는 호환 가능한 인터페이스를 가지는 다른 컴포넌트와 대체 가능하며, 이러한 호환성은 제공 인터페이스와 요구 인터페이스의 관계로 확인할 수 있다.

기존의 인터페이스 호환성은 인터페이스에 포함된 오퍼레이션 집합의 포함관계만으로 검사된다. 요청 인터페이스의 오퍼레이션이 모두 제공 인터페이스에 있으면 아무런 문제가 없지만 그렇지 않은 경우는 문제가 발생한다. 하지만 이러한 단순한 호환성 검사는 인터페이스 내의 오퍼레이션들 간의 수행 순서를 고려하지 않으므로 좀더 정확한 호환성 검사가 불가능하다. 예를 들어, (그림 1)과 같이 sense()와 getData() 오퍼레이션을 제공 인터페이스로 가지는 수동적 센싱 컴포넌트가 있다고 가정해보자. 그리고 센싱 컴포넌트는 수동적 기능을 가지므로 반드시 sense()를 먼저 수행하고 getData()를 수행하여야만 현재 센싱 값을 리턴한다는 제약조건이 있다. 하지만 컴포넌트를 사용하는 관점에서는 이러한 제약조건이 제공 인터페이스에 나타나 있지 않으므로 자율형 센싱 컴포넌트인 것으로 간주하여 getData()만을 호출하게 되면 현재의 센싱 값을 얻을 수 없다. 기존 인터페이스 호환성 검사로는 이러한 오퍼레이션 간의 호출 순서 제약사항에 관련된 호환성을 검사할 수 없다.

2.2 UML 상태기계 다이어그램

UML에서는 시스템의 행위 표현 다이어그램으로 상태기계 다이어그램을 정의하고 있다. 상태기계를 객체의 생명주기 동안 이벤트들에 대응하면서 겪게 되는 일련의 상태들에 대한 명세라고 정의하고 있으며, 시스템의 부분적인 행위에 대해 상태기계로 표현이 가능하다. 따라서 하나의 컴포넌트 행위를 하나의 상태기계로 표현 가능하다. UML 2.1의 상태기계 다이어그램에서 상태는 몇몇 불변 조건(invariant condition)들이 유지되는 동안의 상황을 모델링하는 것이라



(그림 2) 컴포넌트 내부행위를 나타내는 상태기계

고 정의하고 있다. 불변 조건이란 행위가 수행되기 위해 만족되어야 하는 객체 속성 값과 링크에 대한 제약조건을 말한다. 전이는 한 상태가 이벤트 발생으로 다른 상태로 이동하는 것을 말한다. 또한, 상태기계나 다른 상태를 포함하는 복합 상태는 하나 이상의 영역(region)을 가지며, 여러 영역은 상태나 상태기계 상에서 병행성을 표현하기 위해 사용된다.

상태기계에는 표현하는 대상에 따라 행위 상태기계와 프로토콜 상태기계로 나뉜다. 일반적인 시스템 내부의 상태 변화나 객체 내부의 상태 변화는 행위 상태기계로 표현되며 프로토콜 상태기계는 객체 또는 컴포넌트 등의 엔터티의 사용 규약을 표현하기 위해 사용된다. 컴포넌트는 컴포넌트 인터페이스를 실현해야 하기 때문에 컴포넌트 인터페이스의 상태기계 명세에 따라야 하며, 컴포넌트 인터페이스의 상태기계를 컴포넌트 행위 상태기계가 지켜야 하는 규약이라는 의미에서 프로토콜 상태기계라 한다. 컴포넌트 인터페이스가 가지는 프로토콜 상태기계는 컴포넌트 내의 어떠한 오퍼레이션이 어떠한 상태와 조건 하에서 호출될 수 있는지를 표현하며 인터페이스가 가지는 오퍼레이션의 호출 순서를 나타낸다. UML에서 프로토콜 상태기계는 행위 상태기계와 매우 유사하며, 키워드{protocol}을 상태기계의 이름 옆에 두어 시각적으로 행위 상태기계와 구분한다. (그림 2)는 디지털 카메라 컴포넌트의 내부 행위를 상태기계로 표현한 것이다.

2.3 레이블 전이 시스템(Labeled Transition System)

상태기계는 내부적으로 복합상태, 병행상태 등을 계층적으로 표현할 수 있으므로 분석하기에는 어려움이 많다. 분석을 효율적으로 수행하기 위해 이 논문에서는 상태기계를 비교적 간단한 전이 시스템인 레이블 전이 시스템(Labeled Transition System; LTS)[10]로 변환하여 행위를 분석하고자 한다. LTS는 상태 전이 시스템의 일종으로 다음과 같이 정의한다.

[정의 3.1] *레이블 전이시스템(LTS)*

LTS (S, Act, { $\rightarrow_S$: s ∈ Act }, S_0) 모델은 다음과 같이 4개의 항목으로 정의된다.

- S : 상태들의 집합
- Act : LTS내에서 나타나는 행위집합
- $\rightarrow_S$: 모든 행위들의 시퀀스로부터 발생 가능한 전이 관계
- S_0 : 초기상태

위의 정의에서 Act는 모든 행위를 뜻하는데, Act에는 관찰 가능한 행위집합 L과 관찰 가능하지 않은 행위 "τ"가 포함된다. 즉, Act = L ∪ {τ}이다. LTS는 상태의 구조적, 병행적 표현이 불가능하므로 하나의 상태기계를 LTS로 표현하기 위해서는 상태기계의 구조적, 병행적 표현을 효과적으로 포함해야 한다. 이 연구에서는 구조적, 병행적 표현이 없는 간단한 LTS로 변환된 상태기계를 통해 컴포넌트 행위를 분석함으로써 컴포넌트 인터페이스의 행위 호환성을 검증하고자 한다.

3. 컴포넌트 인터페이스의 행위 호환성 검증

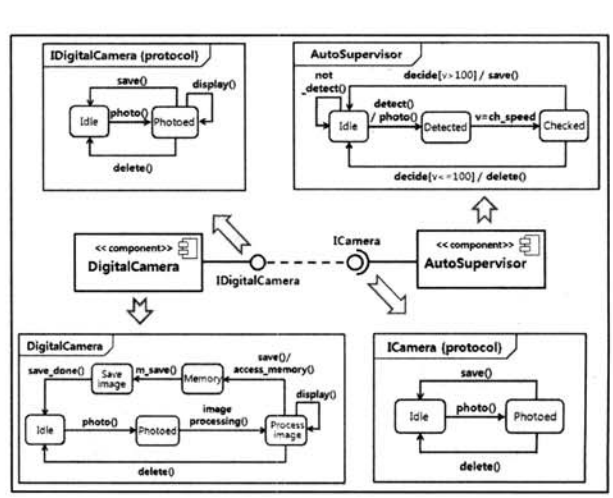
컴포넌트 인터페이스의 행위 호환성을 검사하기 위해서는 먼저 인터페이스에서 정의된 오퍼레이션들에 대한 타입 검사가 우선적으로 수행되어야 한다. 이 연구에서는 이러한 타입 검사가 미리 선행되었다고 가정하며 동일한 이름은 오퍼레이션들은 동일한 타입으로 정의된 것이라 가정한다.

3.1 컴포넌트 인터페이스의 사용패턴 표현

컴포넌트나 인터페이스의 행위란 오퍼레이션 목록이나 수행순서, 매개변수의 타입 등을 뜻하며, 이를 전이 시스템을 통해 표현할 수 있다. UML에서는 컴포넌트 행위를 상태기계로 표현하고 있는데, 본 논문에서는, 컴포넌트 설계 시에는 다양한 표현이 가능한 상태기계를 이용하고 컴포넌트 인터페이스 행위 호환성을 검증 시에는 상태기계를 비교적 간단한 LTS로 변환하여 행위를 분석하고자 한다. UML에서는 다양한 모델 요소의 행위를 표현하기 위한 행위 상태기계와, 사용 규약을 표현하기 위한 프로토콜 상태기계를 정의하고 있다. 프로토콜 상태기계는 컴포넌트 인터페이스에 명세하며, 컴포넌트는 컴포넌트 인터페이스를 실현해야 하기 때문에 컴포넌트의 행위 상태기계는 컴포넌트 인터페이스의 프로토콜 상태기계 명세에 따라야 한다. (그림 3)은 무인 과속단속카메라 시스템에 나타난 컴포넌트의 행위 모델과 컴포넌트 인터페이스의 사용 패턴을 상태기계로 나타낸 것이다.

3.2 컴포넌트의 인터페이스 명세에 대한 검증

사용자 입장에서는 인터페이스에 표현된 프로토콜 상태기



(그림 3) 상태기계를 이용한 컴포넌트와 인터페이스의 행위 표현

계를 통해 컴포넌트가 제공하는 행위의 구체적인 사용 패턴에 대한 파악이 가능한데, 이를 위해서는 컴포넌트가 컴포넌트 인터페이스에 기술된 행위를 올바로 실현하고 있는지에 대한 보장이 우선되어야 하며, 이를 컴포넌트의 인터페이스 명세에 대한 검증이라 한다.

행위 상태기계와 프로토콜 상태기계는 구현과 명세의 관계로 추상화 정도에 차이가 있을 수 있다. 따라서 상태기계를 통한 컴포넌트의 인터페이스 명세에 관한 검증에서는 상태기계 사이의 추상화 정도를 고려하여, 추상화 정도가 동일한 관찰 가능 행위의 일치검사가 이루어져야 한다. 프로토콜 상태기계에 표현된 행위가 행위 상태기계에서 만족되어야 하며, 프로토콜 상태기계에 표현되지 않은 행위는 행위 상태기계에서 만족되지 않아야 한다. 이는 관찰 일치(observation equivalence)를 뜻하며, 이를 약한 바이시뮬레이션(weak bisimulation) 기법[10]을 통하여 수행한다. 약한 바이시뮬레이션에 대한 정의는 다음과 같다.

[정의 2] 약한 바이시뮬레이션(*Weak Bisimulation*)

두 LTS 모델 P, Q의 이차원 관계 $S \subseteq P \times Q$에서 모든 $a \in Act$ 에 대해 다음을 만족하면 약한 바이시뮬레이션이다. 여기에서, $t = a_1 \cdots a_n \in Act^*$ 일 때, $E(\rightarrow_\tau)^* \rightarrow_{a_1} (\rightarrow_\tau)^* \cdots (\rightarrow_\tau)^* \rightarrow_{a_n} (\rightarrow_\tau)^* E'$ 이면 $E \Rightarrow_t E'$ 이고, $t^\wedge \in L^*$ 는 $t \in Act^*$ 에서 모든 τ를 제거한 시퀀스이다.

i) 모든 $P \rightarrow_a P'$ 에 대하여 $Q \Rightarrow_{a^\wedge} Q'$ 이고 $(P', Q') \in S$인 Q'가 존재한다.

ii) 모든 $Q \rightarrow_a Q'$ 에 대하여 $P \Rightarrow_{a^\wedge} P'$ 이고 $(P', Q') \in S$인 P'가 존재한다.

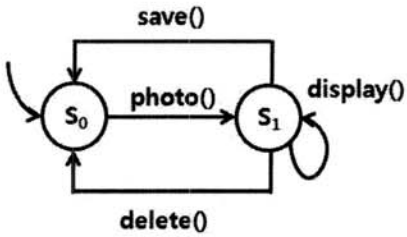
두 LTS 모델에서, 추상화 정도가 높은 행위에서 보이지 않는 전이, 즉, 추상화 정도가 낮은 행위에서만 나타나는 전이를 τ로 규정하고 두 행위의 약한 바이시뮬레이션 여부를 검증하게 된다. (그림 3)의 무인 과속 단속카메라 시스템의 예를 통해 컴포넌트의 인터페이스 명세에 대한 검증을 해보자.

DigitalCamera 컴포넌트를 설계하며, 이 컴포넌트는 오퍼레이션 photo(), display(), save(), delete()를 갖는다. DigitalCamera의 컴포넌트와 컴포넌트 인터페이스 행위를 각각 행위 상태기계와 프로토콜 상태기계로 표현한 후, 이를 (그림 4)와 (그림 5)와 같이 LTS 모델로 변환한다.

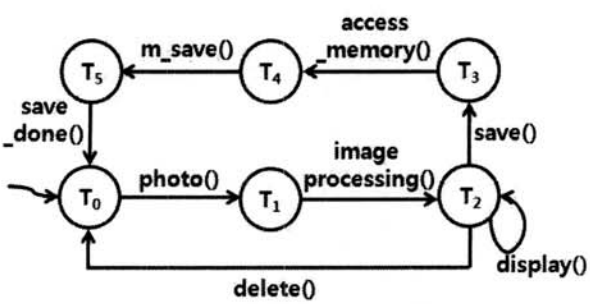
컴포넌트의 인터페이스 명세에 대한 검증을 수행하기 위해서는 먼저, 행위 LTS 모델의 내부 전이들인 image_processing(), access_memory(), m_save(), save_done()을 τ로 생각하고 두 LTS 모델을 합성해야 한다. (그림 6)은 합성 LTS 모델을 보여준다. 합성 LTS 모델의 합성상태 S_0T_3에서의 약한 바이시뮬레이션 여부를 검증한다.

먼저, 합성상태 S_0T_3에서 $S_0 \rightarrow_{photo} S_1$에 대해, $T_3 \Rightarrow_{photo^\wedge} T'$ 가 존재하고, S_1T'가 합성 LTS 모델에 포함되어야 한다. T_3에서 $T_3 \rightarrow_\tau T_4 \rightarrow_\tau T_5 \rightarrow_\tau T_0 \rightarrow_{photo} T_1$가 존재하므로, $T_3 \Rightarrow_{photo^\wedge} T_1$이고, S_1T_1가 합성 LTS 모델에 포함되어 있다. 마찬가지로, 합성상태 S_0T_3에서 $T_3 \rightarrow_\tau T_5$에 대해 $S_0 \Rightarrow_{\tau^\wedge} S'$가 존재하고 $S'T_3$가 합성 LTS 모델에 포함되어야 한다. $\tau^\wedge = \varepsilon$이고 S'는 S_0 자신이므로, $S_0 \Rightarrow_{\tau^\wedge} S_0$가 존재하고 S_0T_4이 합성 LTS 모델에 포함되어 있으므로 합성상태 S_0T_3상에서 약한 바이시뮬레이션임을 보일 수 있다. 나머지 합성상태에서의 약한 바이시뮬레이션 검증도 동일한 방법을 통해 수행할 수 있으며, 이처럼 모든 합성상태에 대해 약한 바이시뮬레이션을 보일 수 있으므로 두 LTS 모델은 약한 바이시뮬레이션 관계를 만족한다.

이와 같은 약한 바이시뮬레이션 검증을 통해 프로토콜 상태기계와 행위 상태기계의 관찰 일치성을 검증할 수 있으며, 이를 통해 컴포넌트 인터페이스의 행위에 대한 컴포넌트 내부 행위를 보장할 수 있다.



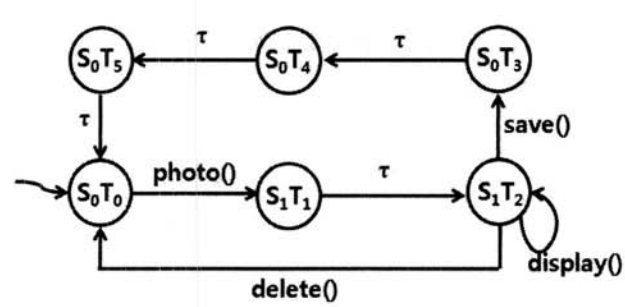
(그림 4) 인터페이스 IDigitalCamera의 프로토콜 LTS 모델



(그림 5) 컴포넌트 Digital Camera의 행위 LTS 모델

3.3 컴포넌트 인터페이스의 연결성 검증

두 컴포넌트의 제공 인터페이스와 요구 인터페이스를 행위적으로 비교하기 위해 두 컴포넌트의 제공 및 요구 인터페이스에 기술된 프로토콜 상태기계를 비교하여 요구 인터페이스가 요구하는 행위를 제공 인터페이스가 올바로 기술되어 있는지에 대한 보장이 필요하다. 이에 대한 검증을 컴포넌트 간 인터페이스 연결성에 대한 검증이라고 한다. 제공 인터페이스와 요구 인터페이스의 두 프로토콜 상태기계



(그림 6) 컴포넌트 DigitalCamera의 합성LTS

는 추상화 정도가 동일하며 요구 인터페이스의 행위가 제공 인터페이스에 포함되는지 여부의 검증이다. 이러한 검증은 상위 타입의 행위가 하위 타입의 행위에 부합하는지를 검사하는 호출 일관성(invocation consistency)[10] 검증이라 말하며 다음과 같이 정의된다.

[정의 3] 호출 일관성*(invocation consistency)*

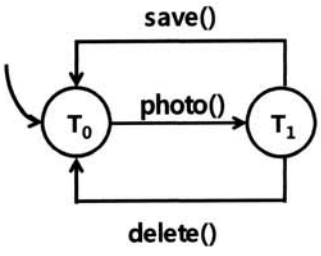
두 LTS 모델 P, Q의 이차원 관계 $S \subseteq P \times Q$ 에서 모든 $a \in Act$ 에 대해 다음을 만족하면 P에 대해 Q가 호출 일관성을 갖는다.

- 모든 $P \rightarrow_a P'$ 에 대하여 $Q \rightarrow_a Q'$ 이고 $(P', Q') \in S$ 인 Q'가 존재한다.

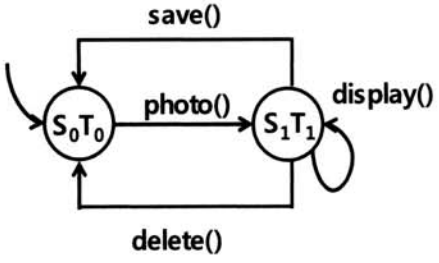
컴포넌트 인터페이스의 연결성에 대한 검증에서는 요구 인터페이스의 프로토콜 상태기계를 LTS 모델 P로, 제공 인터페이스의 프로토콜 상태기계를 LTS 모델 Q로 두고 이에 대한 호출 일관성을 보이는 방법으로 컴포넌트 인터페이스의 연결성에 대한 검증을 수행할 수 있다. 이러한 방법을 통해 컴포넌트의 요구 인터페이스에서 요구하는 행위를 또 다른 컴포넌트의 제공 인터페이스가 올바로 제공하고 있다는 보장이 가능하다. 앞에서와 마찬가지로 무인 과속 단속 카메라 시스템의 예를 통해 컴포넌트 간 인터페이스 연결성에 대해 검증한다.

하나의 요구 인터페이스 ICamera를 가지는 컴포넌트 AutoSupervisor를 설계한다. 제공 인터페이스 IDigitalCamera의 행위와 요구 인터페이스 ICamera의 행위는 각각 프로토콜 상태기계로 표현한 후, 이들을 각각 (그림 4)와 (그림 7)과 같이 LTS 모델로 변환하여 제공 인터페이스에서 요구 인터페이스의 호출 일관성을 검증한다.

이들 두 LTS 모델을 합성한 (그림 8)의 합성 LTS 모델에서 요구 인터페이스 LTS에 대한 제공 인터페이스 LTS의



(그림 7) 요구 인터페이스 IDigitalCamera의 LTS



(그림 8) (그림 4)와 (그림 7)의 합성 LTS모델

호출 일관성을 검증한다. 합성상태 S_0T_0에서 $S_0 \rightarrow_{photo} S_1$에 대해서 $T_0 \rightarrow_{photo} T'$가 존재하고, S_1T'가 합성 LTS모델에 포함되어야 하는데, $T_0 \rightarrow_{photo} T_1$가 존재하고, S_1T_1가 합성 LTS 모델에 포함되어 있음을 알 수 있다. 합성상태 S_1T_1에서도 위와 같은 방법으로 호출 일관성을 간단하게 보일 수 있으므로 이는 생략하도록 한다.

이와 같은 호출 일관성 검증을 통해 요구 인터페이스 요구 행위에 대한 제공 인터페이스의 제공 행위의 만족을 보장할 수 있으며, 이를 통해 컴포넌트 인터페이스의 연결성을 보장할 수 있다.

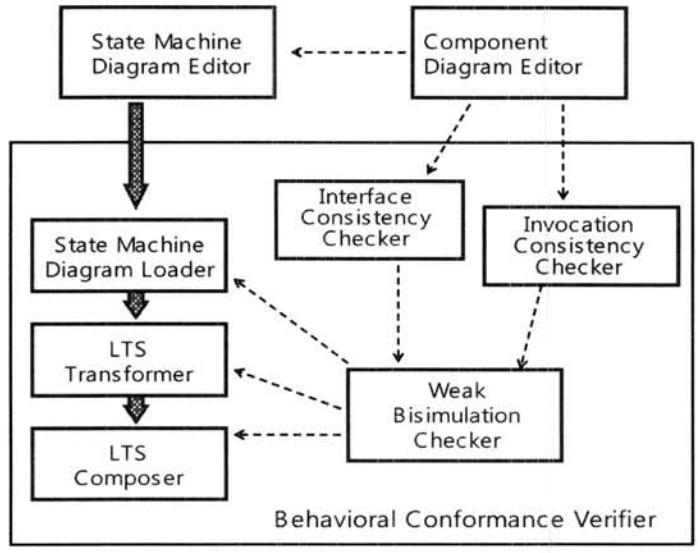
4. 컴포넌트 인터페이스의 행위 호환성 검증 도구

컴포넌트 인터페이스 행위 호환성 검증을 수행하기 위해서 먼저 컴포넌트를 설계할 수 있는 컴포넌트 다이어그램 편집기와 컴포넌트의 행위를 기술할 수 있는 상태기계 다이어그램 편집기가 필요하다. 이들 두 다이어그램 편집기는 Graphical Modeling Framework(GMF)[11]기반으로 이클립스 플러그인의 형태로 구현된다. GMF는 Eclipse Modeling Framework(EMF)와 Graphical Edition Framework(GEF)를 기반으로 하며, 다이어그램 편집기 개발을 위한 생성 도구와 실행 환경에 대한 기반 구조를 지원하는 프레임워크를 말한다. 이 논문에서는 GMF와 UML2 메타모델 명세를 이클립스 기반 EMF 기반으로 구현한 UML2 프로젝트의 메타모델 플러그인 UML2를 이용하여 UML 다이어그램 편집기를 개발한다.

또한 상태기계 다이어그램 편집기와 연계되어 상태기계를 간단한 LTS 모델로 변형시켜 주는 LTS 변환기가 필요하다. 상태기계는 영역을 가지며, 하나의 상태기계는 가지고 있는 영역의 수만큼 LTS 모델로 변형된다. 이를 하나의 LTS 모델로 표현하기 위해 여러 LTS 모델을 합성하여 하나의 합성된 LTS 모델로 만들어주는 LTS 합성기가 필요하다. 따라서, LTS 변환기와 LTS 합성기를 통해 하나의 상태기계가 하나의 LTS로 변환될 수 있다. 또한 하나의 LTS 모델로 표현된 각 상태기계를 이용하여 궁극적으로 컴포넌트 인터페이스 행위 호환성을 수행하는 합성 모델 분석기를 제안한다. 이들 도구들을 바탕으로 컴포넌트 다이어그램 편집기는 합성 모델 분석기와 연계하여 동적으로 행위 호환성을 검증을 수행한다. (그림 9)는 인터페이스 행위 호환성 검증기의 구성도를 나타낸다.

행위 호환성 검증기는 컴포넌트와 상태기계 다이어그램 편집기와 연계하여 동작하며, 이클립스 플러그인의 형태로 구현된다.

- 인터페이스 일관성 검사기
  제공 인터페이스와 컴포넌트 내부 상태기계와의 행위 일관성을 검사하는 부분으로 컴포넌트 다이어그램에 의해 사용된다. 행위 일관성은 양방향 일치 관계를 보여야 함으로 약한 바이시뮬레이션 검사기를 두 번 호출한다.



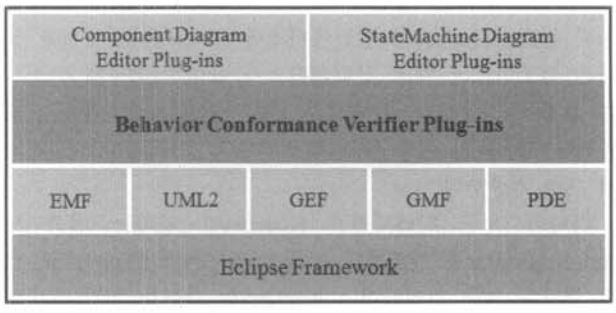
(그림 9) 컴포넌트 행위 호환성 검증기의 구성도

- 호출 일관성 검사기
  제공 인터페이스와 요구 인터페이스를 연결할 때, 동적으로 인터페이스간의 호출 일관성을 검사하는 부분으로 약한 바이시뮬레이션 검사기의 기능을 사용한다.
- 약한 바이시뮬레이션 검사기
  인터페이스 일관성 검사기와 호출 일관성 검사기에서 공통으로 필요로 하는 기능을 구현한 것으로 두 모델 간의 행위 포함 관계를 검사한다. 두 모델의 추상화 수준을 맞추기 위해 내부 전이(τ)를 설정하고 너비 우선 탐색을 통해 stateQueue라는 큐를 사용하여 합성모델을 생성해가며 검증을 수행한다. 이를 통해 행위 호환성 검증이 만족하지 않을 시에 합성모델 생성에 대한 시간적, 공간적 낭비를 막을 수 있다.
- LTS 변환기
  상태기계 다이어그램 편집기를 통해 설계된 상태기계 정보를 파싱하여 LTS 모델로 변환시켜 주는 컴포넌트로, 상태기계 상에서 하나의 영역을 하나의 LTS 모델로 변환시킨다.
- LTS 합성기
  하나의 상태기계로부터 LTS 변환기를 통해 변환된 여러 LTS 모델을 하나의 LTS 모델로 합성하여 축약하는 역할을 하는 컴포넌트이다.

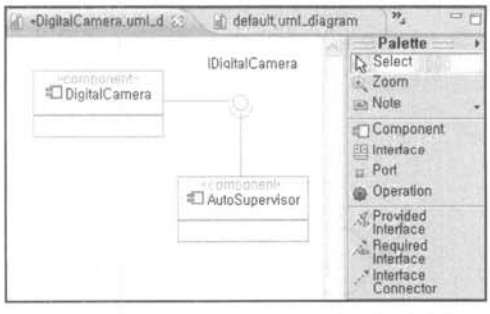
5. 구현 환경 및 수행 결과

컴포넌트 인터페이스 행위 호환성을 검증하기 위한 도구의 구조는 (그림 10)과 같다. 이클립스 환경에서 컴포넌트 다이어그램 편집기와 상태기계 다이어그램 편집기, 행위 호환성 검증기를 플러그인 형태로 구현한다.

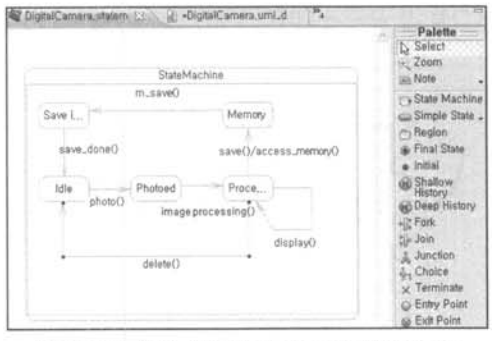
구현된 UML 다이어그램 편집기를 통해서 무인 과속 단속카메라 시스템의 컴포넌트와 각 컴포넌트 및 인터페이스의 행위를 나타내는 상태기계를 설계한다. (그림 11)과 (그



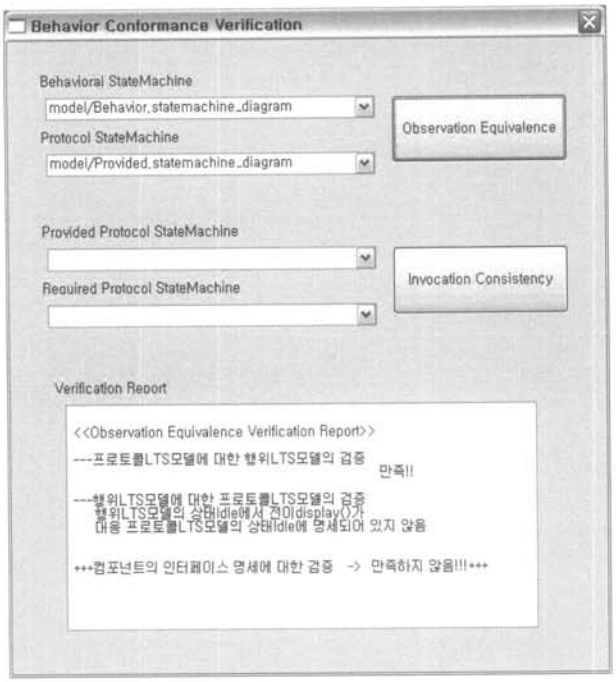
(그림 10) 구현 환경



(그림 11) UML 컴포넌트 다이어그램 편집기



(그림 12) UML 상태기계 다이어그램 편집기



(그림 13) 행위 호환성 검증기

림 12)는 구현된 UML 컴포넌트 다이어그램 편집기와 UML 상태기계 다이어그램 편집기를 보여준다.

이를 바탕으로 컴포넌트 인터페이스 행위 호환성 검증을 수행하는 행위 호환성 검증기를 구현하였다. 컴포넌트 다이어그램 편집기 상에서 원하는 검증 버튼을 누르면, 검증 결과를 보고 받는다. (그림 13)에서 설계된 예제시스템은 두 가지 컴포넌트 인터페이스 호환성 검증을 모두 만족하는 경우이므로, 두 검증 모두 만족된다는 결과를 출력한다.

위와 같이 두 가지 다이어그램 편집기를 통해 컴포넌트와 컴포넌트의 행위를 설계하고, 그것들을 바탕으로 컴포넌트 인터페이스의 행위 호환성 검증을 수행할 수 있다. 또한 수행 결과를 통해 호환성을 만족하지 못하는 상태와 전이를 알 수 있고 이를 통해 효율적인 시스템 설계 및 호환성 검증이 이루어질 수 있다.

6. 결 론

본 논문에서는 UML을 통한 컴포넌트 기반 소프트웨어 개발에서 컴포넌트 설계 단계 시에 컴포넌트의 행위를 표현하고, 이를 통한 컴포넌트 인터페이스 행위 호환성 검증 방법과 이를 동적으로 수행하는 도구를 제안하였다. 기존의 UML을 통한 컴포넌트 기반 소프트웨어 개발에서는 컴포넌트와 인터페이스의 행위 표현을 배제함으로써 제공 인터페이스를 통한 컴포넌트 행위 파악이 불가능하고, 요구 인터페이스에서 기술된 행위를 제공 인터페이스가 행위적으로 올바르게 제공하는지를 보장할 수 없었다.

따라서, 본 논문에서 제공하는 도구를 이용하여 컴포넌트와 인터페이스의 행위를 상태기계 다이어그램으로 설계하여 컴포넌트 인터페이스 행위 호환성을 검증함으로써, 컴포넌트가 제공 인터페이스의 행위 명세에 따른 올바른 행위를 제공하고 있고 컴포넌트 간 연결 시에도 행위적으로 호환됨을 보장할 수 있다. 또한 이를 컴포넌트 설계 단계에서 동적으로 수행할 수 있는 도구를 제안함으로써, 테스팅 단계에서 발생하는 의미 오류를 방지하여 개발 비용을 절감할 수 있다. 이를 통해 컴포넌트 설계 단계에서 의미적으로 정확한 설계가 가능해짐으로써, 컴포넌트 재사용성 향상을 기대할 수 있다.

참 고 문 헌

[1] Keith Short, *Component-Based Development and Object Modeling*, Sterling Software, 1997.
[2] Rational Software Corporation, "Rational Unified Process," A Rational Software Corporation White Paper, 1998.
[3] Desmond F. D'Souza and Alan Cameron Willis, *Objects, Components, and Frameworks with UML : The Catalysis Approach*, Addison Wesley, 1998.
[4] Java2 Platform, Enterprise Edition, http://java.sun.com/j2ee.
[5] Microsoft .NET framework, http://www.microsoft.com/net/.
[6] OMG, *Unified Modeling Language : Superstructure*, version 2.1.1, 2007.
[7] R. Allen and D. Garlan, "A Formal Basis for Architectural Connection," *ACM Transactions on Software Engineering and Methodology*, July, 1997.
[8] Frantisek Plasil, and Stanislav Visnovsky, "Behavior Protocols for Software Components," *IEEE Transactions on Software Engineering*, Vol.28, No.11, Nov., 2002.
[9] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985.
[10] Robin Milner, *Communication and Concurrency*, Prentice Hall, 1989.
[11] Graphical Modeling Framework (GMF) version 2.0.2, Available at http://download.eclipse.org/modeling/gmf/downloads/index.php, 2008.

김 호 준
e-mail : sisqo1227@hotmail.com
2006년 경북대학교 전자전기컴퓨터학부 (학사)
2008년 경북대학교 전자전기컴퓨터학부 (공학석사)
2008년~현 재 삼성전자 근무
관심분야 : 컴포넌트 기반 개발 방법론, 소프트웨어공학, Eclipse GMF 등

이 우 진

e-mail : woojin@knu.ac.kr

1992년 경북대학교 컴퓨터과학과(학사)

1994년 한국과학기술원 전산학과 (공학석사)

1999년 한국과학기술원 전산학과 (공학박사)

1999년~2002년 한국전자통신연구원 S/W공학연구부 선임연구원

2002년~현 재 경북대학교 전자전기컴퓨터학부 부교수

관심분야 : 임베디드 실시간 시스템 모델링 및 분석, Requirements Engineering, Petri nets, 웹 서비스 기술 등